José Miguel Faustino Carneiro

Bachelor of Science

Classification and Scoring of Protein Complexes

Dissertation submitted in partial fulfillment
of the requirements for the degree of

Master of Science in
Computer Science and Informatics Engineering

Adviser: Ludwig Krippahl, Assistant Professor,
NOVA University of Lisbon

Examination Committee

Chairperson: Susana Maria dos Santos Nascimento Martins de Almeida
Raporteur: João Montargil Aires de Sousa

FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

March, 2019

Classification and Scoring of Protein Complexes

This document was created using the (pdf)LaTeX processor, based in the "novathesis" template[1], developed at the Dep. Informática of FCT-NOVA [2].
[1] https://github.com/joaomlourenco/novathesis [2] http://www.di.fct.unl.pt

To all my friends and family

Acknowledgements

First and foremost, I would like to give thanks to all the *Departamento de Informática* and *Faculdade de Ciências e Tecnologia* of the *Universidade NOVA de Lisboa* for providing the infrastructure and the body of teachers that taught me most of what I know about computers.

I am also grateful to my advisor, Ludwig Krippahl, for introducing me to the two fascinating fields of bioinformatics and machine learning, for his patience and his supervision. To the folks at Pyrosetta, for providing me with the license to use their software that enabled the development of the scoring solution. To Pedro Almeida, for helping me with the phrasing, structure and proof-reading of the document.

Last, but most certainly not least, to all the incredible minds who contributed and made available all the tools used throughout the project, without which the software would likely never see the light of day.

Abstract

Proteins interactions mediate all biological systems in a cell; understanding their interactions means understanding the processes responsible for human life. Their structure can be obtained experimentally, but such processes frequently fail at determining structures of protein complexes. To address the issue, computational methods have been developed that attempt to predict the structure of a protein complex, using information of its constituents. These methods, known as docking, generate thousands of possible poses for each complex, and require effective and reliable ways to quickly discriminate the correct pose among the set of incorrect ones. In this thesis, a new scoring function was developed that uses machine learning techniques and features extracted from the structure of the interacting proteins, to correctly classify and rank the putative poses. The developed function has shown to be competitive with current state-of-the-art solutions.

Keywords: Machine Learning, Bioinformatics, Protein-Protein Interactions, Docking

Resumo

Interações de proteínas regulam todos os processos biológicos de uma célula; compreender estas interações implica compreender os processos responsáveis pela vida humana. A estrutura de uma proteína pode ser obtida experimentalmente, mas estes processos são muitas vezes incapazes de obter a estrutura de complexos proteicos. Tentativas de resolver o problema passam pelo desenvolvimento de métodos computacionais que tentam prever a estrutura de um complexo proteico, usando informação dos seus constituintes. Estes metódos, conhecidos como *docking*, geram milhares de poses possíveis para cada complexo dos quais a maioria é incorrecta, e necessitam de métodos fiáveis e eficazes de distinguir a pose correcta entre o conjunto de poses incorrectas. Nesta dissertação, um método foi desenvolvido usando técnicas de aprendizagem automática e atributos extraídos das estruturas das proteínas em interacção, de modo a classificar e ordenar as possíveis poses. O método desenvolvido revelou-se competitivo com soluções existentes.

Palavras-chave: Aprendizagem Automática, Bioinformática, Interacções the Proteínas, Docking

Contents

List of Figures

List of Tables

Glossary

backbone	Series of covalently bonded atoms that together compose the protein.
complex	A protein formed from the assembly of two or more protein chains.
condensation	Chemical reaction in which two or more molecules or parts of one combine, releasing a water molecule.
force field	A set of energy functions and parameters used to calculate the potential energy of a system based on the coordinates of its atoms.
ligand	A molecule that is able to bind to one or more specific sites of another molecule. In computational docking the smaller of the two proteins is the ligand.
native	The state of a protein as it exists in the interior of a cell, properly folded and capable of performing some biological function.
receptor	In computational docking, the molecule that stays still, usually the larger of the two proteins.
residue	Name given to the amino acid's remaining elements after combining to form the backbone chain.
rotamer	A set of possible conformations originating from a restricted rotation around a single bond.
sterical	Relating to the spatial arrangement of the atoms.

Acronyms

AMBER	Assisted Model Building with Energy Refinement.
AUC	Area Under the Curve.
CA	Carbon Alpha.
CAPRI	Critical Assessment of PRediction of Interactions.
CHARMM	Chemistry at HARvard Molecular Mechanics.
FFT	Fast Fourier Transform.
HADDOCK	High Ambiguity Driven protein-protein DOCKing.
MC	Monte Carlo.
ML	Machine Learning.
NMR	Nuclear Magnetic Resonance.
PDB	Protein Data Bank.
PPI	protein-protein interactions.
RCSB	Research Collaboratory for Structural Bioinformatics.
RMSD	Root Mean Square Deviation.
ROC	Receiver Operating Characteristic.
SASA	Solvent Accessible Surface Area.
SVM	Support Vector Machine.
vdW	van der Waals.

Chapter 1

Introduction

Proteins and their interactions with other molecules form the basis for all life. Virtually every biological process within cells is carried out by proteins or complexes of proteins. New protein-protein interactions, integral to all aspects of life on Earth are steadily being found, and knowledge regarding their structure is of extreme value for the understanding of their function. However, current methods of experimental protein structure determination have been unable to keep up.

X-Ray crystallography, the method by which most protein structures have been determined thus far (accounts for ~89% structures in the Protein Data Bank) requires the formation of high quality crystals, a notably difficult and lengthy process, further exacerbated by incredible size and complexity of proteins, and the transient interaction profile many of them display.

This higher difficulty of determining structural information of protein assemblies, or complexes, lead to a knowledge gap between structures of individual proteins and of structures of protein complexes. Efforts to close this gap lead researchers to turn to computational methods that could overcome the limitations of experimental processes and make use of the already known information pertaining to protein structure.

Computational methods that attempt to predict structures of protein complexes using knowledge of their constituents are known as protein docking.

Docking processes typically simulate contact of the two interacting proteins and generate thousands of putative poses, of which only a select few correspond to poses close to the real complexes' native state. There is a pressing need to then filter out these false positives and rank poses in such a way that poses closer to the real complex are ranked higher than the others. These mechanisms are known as scoring functions, and they present the main issue tackled in this thesis.

Solving the problem of complexes' structure determination can be immense for the development of new treatments for life threatening conditions such as cancer[60, 77] and Alzheimer's disease[59].

1.1 Proposed Solution

The main contribution described in this thesis, is a protein-protein docking scoring function, capable of competing with other state-of-the-art methods, that makes use of machine learning techniques and features extracted from the three dimentional structures of proteins to discriminate the native orientation of a protein complex, amidst a multitude of incorrect ones.

1.2 Document Structure

In order to familiarize the reader with the general outline of the document, this section aims to provide some basic knowledge of the organization of each chapter: chapter 2 summarily introduces relevant biological concepts important for the understanding of the work, namely amino acids, protein structure, homology and protein-protein interactions (PPI); Chapter 3 presents the ideas behind protein docking and scoring, as well as a representative overview of various state of the art algorithms in these fields; In chapter 4 a select number of machine learning algorithms used throughout the work are described, together with their principal advantages and disadvantages; The process of data acquisition, the Protein Data Bank and the main file format used throughout the work will be detailed in chapter 5; Chapter 6 describes the process of development of the scoring solution, which comprises the practical component of the work presented; In chapter 7 a comparative study between the the developed scoring function and other state of the art scoring functions; Finally, chapter 8 concludes the document providing an overview of the work achieved, as well as possible directions for future improvements.

Chapter 2

Protein Structure

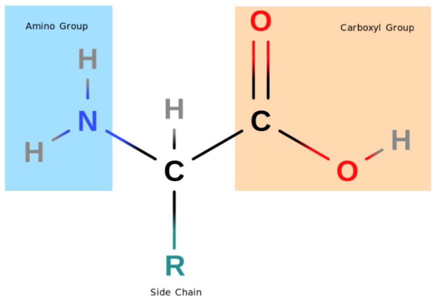
Proteins are a fundamental component of all living things. Every single cell in our body makes use of proteins, in one way or another, to exert some biological function. These incredible molecular devices are responsible for a plethora of critical roles in all of life's processes, for example: they provide structural support, allowing cells to maintain their shape; act as catalysts, facilitating biochemical reactions; assist in the formation of new molecules; are an active part of immune system; help regulating hormones and protein transcription; serve as biological reserves of metal ions and amino acids; and can transport ions and other molecules within an organism[7, 12, 54].

Their structure is intrinsically tied to their activity and is as diverse as the functions they provide. Thus, the study of their structure is paramount if we are to understand how proteins are formed, how they interact and, ultimately, how they function. Insight obtained from protein structure could be precious in understanding and preventing diseases, or developing new and improved treatments, drugs and other products that can be put to use for the betterment of countless aspects of human life.

Proteins' sizes can vary wildly, from simple ones containing only a few dozen atoms to gigantic chains of several thousands. For example, Titin, the protein responsible for the passive elasticity of muscles can reach upwards of 33 000 residues, and 244 domains[51]. Describing such structures atom by atom would easily become intractable. This difficulty in describing protein structures lead to the formulation of several layers of complexity on which to analyze it. In the following sections, the four levels of protein structure will be briefly explained, together with a few other important concepts to keep in mind throughout the work. This description will aim to provide only the basic background for the understanding of the work, for a more exhaustive detailing the author would refer to other books such as *Introduction to Protein Structure*[7].

2.1 Amino Acids and Primary Structure

Amino acids are simple organic compounds. There are twenty canonical amino acids specified by the genetic code, each sharing a common core structure of a central carbon $C\alpha$ bonded with an amino group (NH_2), a carboxyl group ($COOH$) and a so called side-chain, or R group, which differs for every amino acid.



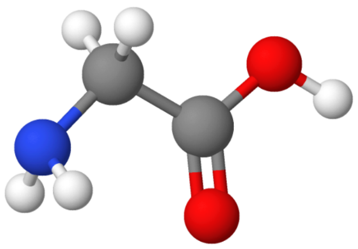


Figure 2.1: Illustration of the atomic composition of an amino acid (Left). The simplest amino acid, Glycine (Right).

These side-chains make the distinction of amino-acids between each other, giving them unique chemical and sterical characteristics. These properties impact and ultimately define the structure, the function and the interactions of proteins.

As a means to simplify description of these properties, amino acids are often grouped into categories. One of the most simple and common classifications puts amino acids into one of the following: hydrophobic, polar-uncharged, polar charged negatively and polar charged positively[7, 12]. A table with the residues' common classification can be seen in table 2.1.

Name	3-Letter	Residue Type
Alanine	ALA	Hydrophobic
Arginine	ARG	Positive
Asparagine	ASN	Polar
Aspartic Acid	ASP	Negative
Cysteine	CYS	Polar
Glutamic	GLU	Negative
Glutamine	GLN	Polar
Glycine	GLY	Hydrophobic
Histidine	HIS	Positive
Isoleucine	ILE	Hydrophobic
Leucine	LEU	Hydrophobic
Lysine	LYS	Positive
Methionine	MET	Hydrophobic
Phenylalanine	PHE	Hydrophobic
Proline	PRO	Hydrophobic
Serine	SER	Polar
Threonine	THR	Polar
Tryptophan	TRP	Hydrophobic
Tyrosine	TYR	Polar
Valine	VAL	Hydrophobic

Table 2.1: Table of residue types as classified by *Structural bioinformatics: an algorithmic approach*[12], this residue grouping is used later for feature extraction.

During protein synthesis the amino acids are joined together by a condensation process in which the amino group of one amino acid forms a peptide bond with the carboxyl of another. This process happens multiple times resulting in a chain of these amino acids (now also known as residues), to which we call polypeptide chains. These chains display a repeating sequence of *Nitrogen-Carbon-Carbon* atoms, known as the backbone.

Some amino acids possess other special properties, for example: Proline forms cycles back to the polypeptide chains, which imposes limitations on the flexibility of the backbone; Glycine on the other side of the spectrum, due to its size (side-chain is a single hydrogen atom, see fig 2.1) allows more flexibility than any other amino acid[12].

The description of the linear sequence of these amino acids by order of appearance on a protein's backbone chain (ordered from amino group to carboxyl group) is known as a protein's primary structure.

2.2 Secondary Structure

Amino acids from a polypeptide chain fold upon themselves, forming fairly regular and energetically favourable structure configurations. These structures occur and are held together by hydrogen bonding between atoms in the backbone chain.

A common classification for these structures puts them into one of three categories: *α-helices*, *β-sheets* and loops. An example of a protein with all three can be seen in figure 2.2.

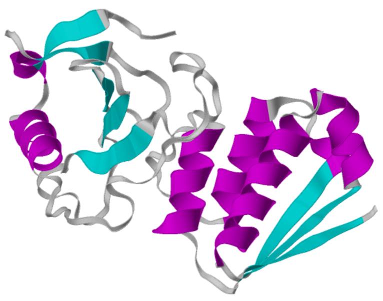


Figure 2.2: Complex of PDB code 1AY7 colored by secondary structure. *α-helices* can be seen in purple, *β-sheets* in cyan, and loops in white.

Given the interaction being almost entirely independent from specific amino acid residues of the chain, these structures are present in a vast majority of proteins. In most, *β-sheets* and *α-helices* will form an hydrophobic core, connected by loops[12].

The characterization of these regular structures is known as a protein's secondary structure.

2.3 Tertiary Structure

Soon after proteins are formed, they undergo a folding process in which all the different residues interact between themselves and the solvent according to their chemical and sterical preferences. These interactions result in innumerous temporary spatial arrangements of atoms. Eventually however, one particular folding pattern will be more energetically favourable than all the others, leading to the formation of a semi-stable three dimensional conformation capable of performing some biological function; the native state. The full description of the arrangement of all the atoms and their positions is known as tertiary structure.

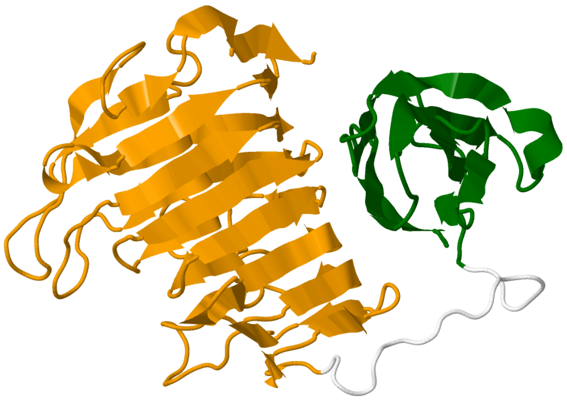


Figure 2.3: Example of two domains, in orange and green, from Pectate Lyase (PDB code 4U4B)

Often, proteins form separate cores of residues known as domains. A formal definition of a domain is not exactly agreed upon, but one good definition can be found in Burkowski's book, in which a domain is described as: "[...] a region of a protein that has more interactions within itself than with the rest of the chain". Typically, these regions exhibit a globular, compact core of hydrophobic residues organized in *αhelices* and *β-strands* linked by loops around the surface[7]. Domains are capable of evolving and interacting independently, and the same domain structures may appear in different proteins, making them the basic unit of function and structure of proteins[12].

2.4 Quaternary Structure

Quaternary structure is the only non compulsory level of structure. A protein possesses quaternary structure when it is an assembly of multiple polypeptide chains. Such structures are known as complexes.

A vast majority of proteins belong to this category, as they incur in interactions with other proteins in order to perform their function.

These interactions occur at patches of the protein's surface known as interfaces. Interactions between proteins can broadly be classified into two types: obligatory (permanent) or transient. Obligatory contacts mean the proteins are required to be in a complexed state in order to maintain their structure and function, that is to say, they cannot be found *in vivo* uncomplexed. Transient contacts on the other hand can interact with several different molecules throughout their life, thus existing in both complexed and uncomplexed states[2, 24].

The distinction of the two types of contacts is relevant, as interfaces of both types display several differences such as area, structure and residue composition. Obligatory interfaces for example, have typically been characterized as being larger and more hydrophobic than transient ones[24], which seem to possess higher distribution of charged and polar residues[2]. Hydrophobicity is the main factor in the stability of protein structure[20], this can help explain these differences in residue composition, as transient interfaces need to associate and dissociate quickly[2].

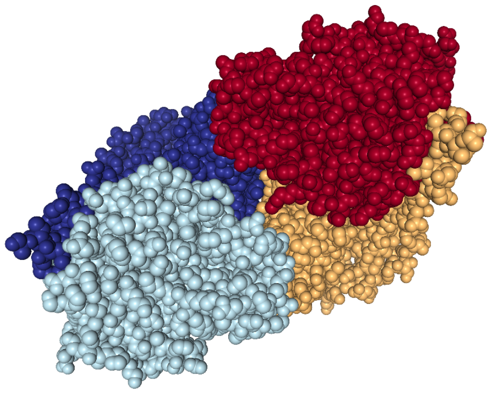


Figure 2.4: Structure of 4NIF heteromonomer, each monomer is shown in different colors.

Residues at interfacial patches have specific pairing preferences. Hydrophobic residues tend to pair with other hydrophobic residues, while charged residues prefer residues with complementary charge[2, 30, 58].

Knowledge of protein interfaces and composition can impact both stages of molecular docking. Docking algorithms can use this knowledge to limit the number of conformations sampled thus making the search process faster[5, 32]. Furthermore, interfacial information such as residue composition and pairing preferences or statistics can also be helpful in order to identify non-native solutions[2, 30].

2.5 Protein Homology

Proteins, much like every living thing, are subjected to selective pressure to evolve in an advantageous manner. For proteins specifically, this means that throughout evolution, mutations should minimally disrupt their ability to interact with a partner molecule[58].

Homologous proteins descend from a common ancestor by divergent evolution and they are usually identified by sequence similarity[7]. The interest in homology derives from the observation that homologous proteins exhibit similar backbone structure, even in proteins with sequence similarity as low as 30%[12, 58].

This property has been used to enhance prediction of protein structure based on knowledge of the structure of the protein's homologues (homology modeling). Everyday, new structural data becomes available, making homology an even more appealing prediction method.

Besides structure prediction, analysis of residue conservation in homologous proteins can also aid in identifying protein interfaces[5].

Chapter 3

Protein Docking and Scoring

Protein interactions are the basis for nearly every cellular process. The vast majority of proteins form assemblies of proteins named complexes, with specific three dimensional structure that allow the complex the execution of some biological function.

Determination of the structure of such complexes is exceptionally difficult to perform experimentally. They can reach incredible sizes, and many posses a rapid rate of association and dissociation that make assembly of high quality crystals necessary for X-Ray Crystallography nearly impossible. On the other hand, determination of structures of individual proteins is comparatively simpler; this lead to a gap between the number of known structures of individual proteins and that of protein complexes.

In order to bridge this gap and harness the power of existing structural knowledge, other methods of studying and determining structures of complexes have been explored, more relevant of which, computational methods.

These methods, in which the structure of the complexes' constituents is used to predict the structure of the complex itself, is called protein docking. Interest in the field of protein docking has been around since the early 80's; however, technical and computational constraints put a limit to what was achievable at the time. Since then, a tremendous increase in computational power, data quality and availability, together with better knowledge of protein interactions, made the approach more feasible and widespread[7, 8]. Despite the progress, docking remains an unsolved and challenging problem.

Current docking procedures can be conceptually split into two sequential stages: an initial stage in which thousands of possible poses for a given protein complex are generated; and a second stage, known as refinement or scoring stage, in which algorithms try to effectively discriminate and rank which of the generated putative poses correspond to near-native structures, and which do not[32, 36].

In the following sections, a general overview of both initial stage docking and scoring stages will be provided together with some concrete examples of frameworks for both. Given the focus of this thesis being scoring, the docking section is presented mainly for context, although they can influence some viable scoring approaches.

3.1 Search Stage

The search stage, also known as initial stage docking, is the first step in computational docking frameworks. The goal at this stage is to iterate through possible arrangements of the two molecules and generate a subset of those that might correspond to native conformations. Efficiency and speed are the focus of this stage, as well as attempting to reduce the number of different poses in the generated set without discarding those approaching the correct structure.

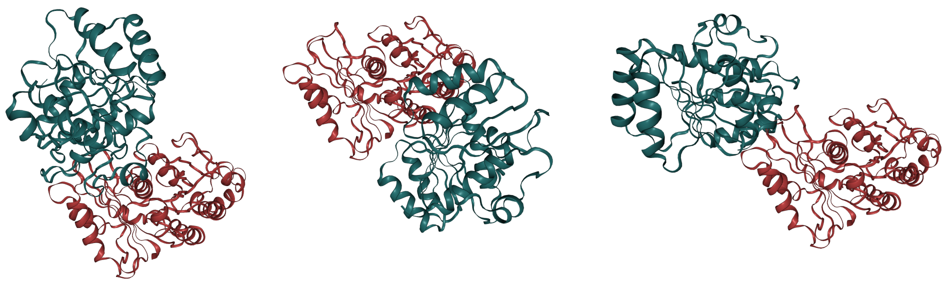


Figure 3.1: Example of several different conformations generated by GRAMM-X for complex with PDB code 1ofu. Receptor is shown in red and putative poses for the ligand in green.

To this end, algorithms make use of simplified scoring functions, typically involving some measurement of shape complementarity and/or handling proteins as rigid bodies. This requires workarounds to handle molecular flexibility, usually by allowing a soft margin of penetration between the interacting proteins[43] or by using geometric descriptions of lower resolution[88] to implicitly handle side-chain flexibility.

Searching algorithms on this stage can be broadly classified into either global or local search algorithms. Global search procedures such as Fast Fourier Transform (FFT) correlation method[43] exhaustively search all the possible arrangements of the proteins, which includes many incorrect configurations. Despite this, it remains a popular searching algorithm with PIPER[47], DOT[61] and ZDOCK[18] being some examples of FFT based docking frameworks. Often, one of the interacting molecules (typically the receptor) is held in place for simplification, with all poses being different three dimensional arrangements of the other.

By contrast, local based searches typically sample the search space, either randomly or by being provided a starting conformation, together with an optimization algorithm to gradually improve the quality of the conformation. Given the reduced search space, these methods are more easily capable of incorporating the means to deal with molecular flexibility. Examples include RosettaDock[57] and HADDOCK[25].

Additional information such as knowledge of interfacial residues or homologous proteins can be used to further restrict the search space, which can significantly improve docking performance[25, 27, 78].

3.2 Scoring/Refinement Stage

The refinement stage, also known simply as scoring, takes the generated set output from the previous stage and applies any number of methods and transformations with the end goal of discriminating and ranking conformations that correspond to near-natives and those that do not. Ideally, a scoring function would rank the best putative pose the highest, with successively worse poses being ranked accordingly.

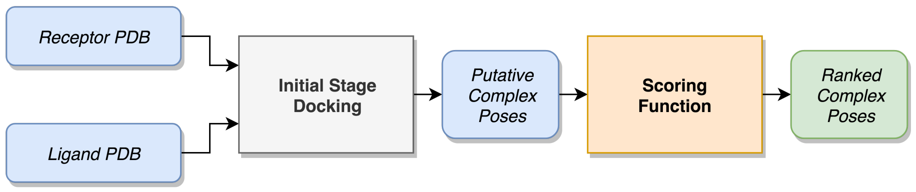


Figure 3.2: Illustration of the usual two stage docking approach, displaying where the scoring stage fits in a typical docking pipeline.

Given the often large number of generated poses and the high complexity and computational cost, scoring is often preceded by some technique to reduce the number of putative poses, typically by using simplified scoring functions as a filter, clustering, or a combination of both. Clustering can be of particular interest as it may be a good indicator of so called energy-basins, which often correspond to near-native arrangements[68].

Side-chain searches can also be performed at this stage, usually with the aid of rotamer libraries, which help save time and resources by providing pre-determined side-chain conformations of low energy.

Molecular force fields such as Assisted Model Building with Energy Refinement (AMBER)[71] and Chemistry at HARvard Molecular Mechanics (CHARMM)[10] are common tools of molecular dynamics simulations used in energy minimization steps. Energy minimization being procedures in which a given structure is "relaxed" into a more energetically favored state by, for example, altering bond-lengths, angles, etc.

Present in almost all scoring functions are values for several energetic terms, modeling properties such as shape complementarity, hydrophobicity, solvation and electrostatics[15, 73]. Statistical and Knowledge based terms derived from residue contacts on databases of determined protein structures have also been known to perform well in discriminating near-native conformations[31, 65].

It is precisely at this stage that this work takes place, applying machine learning techniques to develop an effective scoring solution capable of competing with state of the art solutions.

3.3 Docking Frameworks

In the following subsections, some of the most popular and influential full docking frameworks will be briefly described. The selection of the algorithms to present here was made so as to provide a representative set. All the presented frameworks were tested and validated in Critical Assessment of PRediction of Interactions (CAPRI)[41], a communitywide experiment to assess the ability of protein docking procedures in predicting protein-protein interactions.

3.3.1 RosettaDock

RosettaDock[57] is a Monte Carlo (MC) based docking framework that approximates the theoretical behavior of a protein encounter and subsequent binding with another protein[13]. The algorithm starts by obtaining a good random starting orientation for both proteins by means of score filtering and clustering. Alternatively, a starting position may be provided based off available information of the complex.

A low-resolution representation of this starting structure, in which each side-chain is replaced by a representative fake atom is then used for a 500-step rigid body MC search. Each step adjusts the translation and rotation of one molecule with step size being fine tuned to maintain a 50% acceptance rate. For this stage a simplified scoring function is used that includes contact and bump scores as well as environment and residue pair potentials.

The structure with the lowest energy is then passed onto a refinement stage in which the full description of the side-chain is added back to backbone of the proteins, replacing the fake atom representation.

A 50-step MC search then ensues where each step randomly perturbs the position of the structure followed by a side-chain packing using a rotamer library[82] and energy minimization. After each MC move, a score is calculated that includes terms such as van der Waals (vdW) attractive and repulsive energies, residue-residue pair interactions, electrostatics, hydrogen bonding energies, solvation and a side-chain probability term.

The Metropolis acceptance criterion is then employed to decide if the position should be updated. The entire search procedure is repeated a finite number of times(typically 10^5) and the 200 best scoring structures are clustered by RMSD(2.5Å) with the lowest scoring pose of each cluster being its representative. The final predictions are the cluster representatives ranked in descending order of cluster size.

3.3.2 ClusPro

ClusPro[21, 46, 48] was the first fully automated computational docking framework as well as one of the most successful. It takes the output poses of either ZDOCK[18], DOT[61] or PIPER (only in version 2.0)[47] and keeps the best *n* poses (1000~2000) for further processing. Considering interfacial residues to be those from the ligand within 10Å of any residue of the receptor, the interfacial Root Mean Square Deviation (RMSD) of the Cα of each pose is calculated with respect to every other model and stored in a N x N matrix. An iterative clustering stage immediately follows where the ligand with the highest number of neighbors withing a certain threshold (9Å by default) is considered the cluster's representative, and all elements from that cluster are removed from the matrix. This process is repeated until at least 30 clusters are formed. Finally, the cluster's representatives are subjected to vdW energy minimization using CHARMM potential and the algorithm outputs the cluster representatives in descending order of cluster size.

3.3.3 HADDOCK

High Ambiguity Driven protein-protein DOCKing (HADDOCK)[25] is a docking framework that prides itself on being data-driven. It takes a list of active (known to interact) and passive (might interact) residues from each protein, derived from information obtained from mutagenesis, mass spectrometry and/or NMR, to create distance restrictions between any residue of the active list of the ligand and atoms of both active and passive lists of the receptor. If no experimental data is available, interface prediction algorithms can be used instead.

These constraints are then incorporated into a scoring function, that includes energetic terms such as vdW, electrostatic and desolvation, in order to bias the search toward conformations that satisfy them.

HADDOCK then undergoes a rigid body energy minimization after which top 1000 structures are retained for further processing. These structures are then refined with a simulated annealing algorithm in which limited side-chain and backbone flexibility is introduced to segments of the protein that encompass the active and passive residues. These segments are calculated automatically, but they can also be specified manually.

Lastly, the leftover structures are once more subjected to energy minimization and molecular dynamics simulations in explicit solvent, with the results being clustered and ranked according to the average energy of the best four conformations in each cluster.

3.4 Scoring Functions

Several representative scoring functions will be detailed in the next subsections. Once again, the choice of function was biased towards popular functions with distinct approaches, and the descriptions will be brief, focusing mostly on the intuition behind the algorithms. FastContact[14], dDFIRE[85] and SPIDER[44] are of particular interest, as they were used in the work for comparative performance assessment (chapter 7).

3.4.1 ZRANK

ZRank[73] is a scoring framework that takes putative poses from ZDOCK[18] and uses a linear weighted sum function with seven energetic terms to rank them:

$$\begin{gathered} score = \sum_{x} w_{(x)} \times E_{(x)} \quad , \\ x = \{vdW_a,\ vdW_r,\ elec\_sr_a,\ elec\_lr_a,\ elec\_sr_r,\ elec\_lr_r,\ solv\} \end{gathered} \tag{3.1}$$

With E being a feature vector whose terms are van der Waals, electrostatic and solvation energies, with both van der Waals and electrostatics being segmented into attractive and repulsive components, and the latter further segmented into short range and long range (criteria for separation was 5Å). W being the corresponding weight vector for each feature. In order to obtain their weights for each term of the function a 7D downhill simplex minimization algorithm was used on a subset of the DBMK 1.0[17]. The scoring function was then tested on DBMK 2.0[63] in which it was shown to improve the results of ZDOCK.

3.4.2 PioDock

PioDock[27] is a scoring framework that makes use of homology information and interfaces to score the structure of a protein complex. It follows a streamlined process that starts by classifying the query proteins (QP), that is to say, proteins that form the complex, into three classes: trivial, homologous and unknown.

If the proteins have interacting homologues present in the PDB, the query proteins are superimposed on that complex, consequently obtaining a prediction for the interface.

If the proteins have homologues, but none of them interact, an interface is predicted using WePIP[26], which uses the best 30 homologues of each protein to predict its interface. Lastly, PredUs[87] is used to predict the interface of any proteins with no known homologues.

Finally, once the set of potential solutions obtained from some docking algorithm and the predicted interfaces are known, a similarity score is calculated by averaging the the overlaps of both query proteins:

$$complexOverlap_{A-B} = \frac{overlap_A - overlap_B}{2} \tag{3.2}$$

In which A and B represent the overlapping score value of each query protein obtained from:

$$overlap = \frac{|interface_{docking} \cap interface_{predicted}|}{\sqrt{|interface_{docking}| \times |interface_{predicted}|}} \tag{3.3}$$

The final prediction is then obtained by sorting the potential solutions according to this score.

3.4.3 FastContact

FastContact[14] is a scoring function, also available online as a server, that estimates the direct electrostatic and desolvation interaction free energies between two proteins. The choices of using only these two comes from the observation that they are the main contributors to the stability of protein complexes[11].

The electrostatic component is calculated using Coulomb electrostatics with a distance-dependent dielectric constant. A minimum threshold between the atom-pairs was enforced to avoid overlaps. The desolvation contribution is calculated by means of an empirical contact potential calculated using a database of crystal monomers (single chain proteins) from the PDB. The maximum number of residues accepted by the function is 1500.

By splitting the energetic terms, FastContact also allows understanding of some properties regarding the scored complexes, for example, a negative desolvation energy indicates an hydrophobic pocket at the interface, with a positive desolvation energy suggesting a mainly polar interface. FastContact is also one of the few freely available scoring functions validated at CAPRI.

3.4.4 dDFIRE

Dipole Distance-scaled, Finite Ideal gas REference (dDFIRE), is an all atom knowledge based scoring function that extends DFIRE[86] by introducing orientation dependence of polar atom interactions. In this scoring function, each polar atom (Nitrogen, Oxygen and Sulfur) is treated as a dipole with a direction, with dipoles essentially being regions of molecules with separated partial charges, positive and negative. The orientation of these dipoles are defined by the connected bonds each polar atom has with other heavy atoms.

The derivation of the equations and methods of their obtention are beyond the scope of this description. Nevertheless, the formula for obtaining the atom-atom potential between atom types i and j separated by distance r for the original DFIRE is calculated as follows:

$$\bar{u}(i,j,r) = \begin{cases} -\eta RT ln\left(\frac{N_{obs}(i,j,r)}{\left(\frac{r}{r_{cut}}\right)^{\alpha}\left(\frac{\Delta r}{\Delta r_{cut}}\right)N_{obs}(i,j,r_{cut})}\right) & ,r < r_{cut} \\ 0 & ,r \geq r_{cut} \end{cases} \tag{3.4}$$

Where $\eta = 0.0157$, R is the gas constant, $T = 300K$, $\alpha = 1.61$, $r_{cut} = 14.5Å$, $N_{obs}(i,j,r)$ the number of (i,j) pairs within the distance shell r observed in a given structure database and $\triangle r$ the bin width for a particular r.

dDFIRE extends on this function by incorporating three additional variables $(\Theta_i, \Theta_j, \Theta_i j)$ defining the orientation angles of the polar atoms' dipole moments, replacing in the above equation $N_{obs}(i,j,r)$ for $N_{obs}(i,j,\Theta_i,\Theta_j,\Theta_{ij},r)$. dDFIRE was the best performer of the external scoring functions used to validate the final solution in chapter 7.

3.4.5 SPIDER

Scoring Protein Interaction Decoys using Exposed Residues, or SPIDER[44] for short, is a protein scoring function that makes use a coarse-gained representation (with side-chains being represented by centroids) and a geometry based approach to transform the three dimensional coordinates of a complex into a residue contact network.

The first step in this scoring function is to obtain all the interfacial residues. As customary, interfacial residues are those with at least one residue from the opposite protein within 10Å threshold distance. Next, a modified version of Delaunay-Tesselation[3] is used to analyze the geometry of the points, and convert these interfaces into a labeled graph, in which nodes are residue side-chain centroids and edges connecting them a contact, thus forming an interaction residue network.

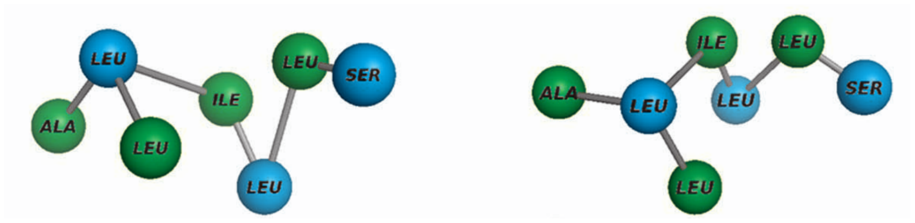


Figure 3.3: Examples of extracted subgraphs derived from the mining process. The same pattern may appear in several different geometries and different complexes. Image extracted from the original paper[44].

The Fast Frequent Subgraph Mining (FFSM)[37] technique then follows, which finds frequent subgraphs occuring in at least a certain fraction(5%) of the interfaces of a subset of 241 native complexes obtained from Dockground (training set). Subgraphs that belong to a larger subgraph are eliminated thus avoiding overlapping subgraphs and preventing undue influences these overlaps might have caused. These "native" subgraphs (or patterns), were then stored and are used for calculating new protein's scores.

From here, the process of converting these subgraphs and subgraph frequencies into a scoring function assumes the following: higher subgraph frequencies should equate to higher scores; and bigger patterns are more valuable than smaller ones. With these assumptions in mind, the final score is calculated using the following formula:

$$score = \|\sum_i^N \sum_j^M \frac{|P_i|}{RMSD_{ij}^{Pattern}}\| + \|X1\| + \|X2\| + \|X3\| + \|X4\| \tag{3.5}$$

Where N is the total number of frequent patterns found at the interface, M is the frequency of pattern i in the training set, $|P_i|$ is the size of the pattern P_i and $RMSD_{ij}^{Pattern}$ is the RMSD calculated between pattern i and the best matching pattern from the training set. $\|X1\|$, $\|X2\|$, $\|X3\|$ and $\|X4\|$ are the number of interfacial residues that match the patterns, the fration of interfacial residues that match the patterns, the number of patterns at the interface and the average number of patterns matched per one interfacial residue respectively.

CHAPTER 4

MACHINE LEARNING

Machine Learning (ML) is the name given to the subfield of Artificial Intelligence dedicated to the engineering of systems capable of learning from and improving with data. Alternatively, Tom Mitchell famously defines machine learning as: "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"[64].

Immediately, this definition highlights the central points of any machine learning problem. These systems take data and use it to approximate a function that turns these data inputs into knowledge. This ability of turning experience into knowledge is exactly the reason behind the advent of machine learning. Everyday, from social media to the Large Hadron Collider, incredible quantities of data are generated[69, 80]; data that would be virtually useless without the help of computers and algorithms to digest it. Machine learning fits today's data-centric society like a glove.

Furthermore, machine learning is a tried and tested field, having been applied to a vast array of different fields with great success, some of which include retail, finance, healthcare, transportation, education and sports, and, more in line with this thesis, bioinformatics and computational biology.

The benefits of machine learning do not end here either. These algorithms also possesses the extremely advantageous ability to adapt to changing conditions automatically, as data inputs themselves change and, perhaps even better than that, allows development of solutions for problems which would be impractical or even impossible to formally define. A notorious exemple is the problem of digital handwriting recognition, which is especially difficult to define, but for which machine learning has been successfully applied, achieving an astoundishing error rate of only 0.21% on the well known MNIST dataset[53, 75].

Machine learning is a vast field, capable of tackling innumerous problems, often requiring a multitude of distinct approaches. The problem might require a descriptive approach, providing deeper understanding of intrinsic properties of the data (unsupervised learning) or a predictive approach, used to estimate some labels or values of previously unseen data (supervised learning). Other approaches such as reinforcement learning and semi-supervised learning also exist, but will not be covered as they have no bearing for the understanding of this thesis. Unsupervised learning, while often used in docking frameworks, also took a backseat in this work, given the choice of dataset used; nevertheless, the intuition behind unsupervised learning and some use cases will still be provided.

This leaves out supervised learning, which is the most commonly used branch of ML in practice, and also the most relevant approach for this work. Supervised learning, along with some of the most iconic and performant algorithms will be detailed in the next sections. Once again, these sections aim to provide only a basic understanding, and do not serve as a replacement for a more thorough book such as *Machine learning: a probabilistic perspective*[66].

4.1 Unsupervised Learning

Unsupervised learning is a type of machine learning in which the input data is a collection of unlabeled examples. Intuitively, this absense of a ground truth means unsupervised methods do not possess a reliable reference point by which to evaluate a model. The goal of these methods is often to simply transform the set of input data, into a new, different representation that can be used for some other practical application, commonly supervised learning. This ability of uncovering underlying structure between data is particularly valuable when taking into account how a vast majority of existing data is unlabeled, and how costly it often is to label it.

The three main use cases of unsupervised learning are dimentionality reduction, outlier detection and clustering. In dimensionality reduction, methods such as Principal Component Analysis(PCA) or Autoencoders are used to reduce the total number of input features into a smaller, representative set, which may provide many benefits such as increasing speed of a supervised model, or simply enabling visual representation for human interpretability.

Outlier detection, as the name implies, aims to detect examples in the dataset that are significantly atypical from the other instances, a common application being credit card fraud detection for example.

Clustering is perhaps the most popular use case of unsupervised learning. In clustering, the goal shifts to attempting to find natural groups of data (clusters) that share high similarity between instances of the same group and high dissimilarity between instances of other groups. Clustering is often called unsupervised classification and can be used, for example, to discover partitions of customers for use in targeted publicity or product placement, to recommend media or products of any kind to a user based on the user's preferences, to organize proteins into families of common ancestry, etc[66, 89].

4.2 Supervised Learning

Contrasting to unsupervised learning, supervised approaches are called so because they deal with collections of labeled data. The existence of these labels allow the use of metrics based on ground truth, so evaluation of quality of a supervised model is often a lot more straightforward than in unsupervised approaches. Intuitively, these algorithms are also applied to significantly different problems. These advantages come at the higher difficulty of obtaining data, as the labeling process usually requires manual labor from domain experts.

The main goal of supervised algorithms is to use a set of labeled data as input and attempt to approximate a function that best internalizes an assumed relationship between the inputs and the labels (outputs). As an example, linear models use a simple linear combination of weighted input features:

$$y = \theta_0 + \sum_{i=1} \theta_i x_i \tag{4.1}$$

With x_i being the i^{th} feature, θ_0 being the bias unit and θ_i being the weights associated to each feature. The weights in this scenario are the parameters we need to determine, and the process by which we determine these parameters is known as training. However, in order to be able to use data to obtain an optimal set of parameters, we first have to define the concept of optimality, usually by means of an optimization function together with a regularization term. This regularization term allows control of the complexity of the model in order to avoid overfitting, that is to say, memorizing the training data instead of learning from it.

Ideally, if the data is good and the model appropriate, the learned function should be able to generalize and correctly predict labels for previously unseen data inputs.

When the predicted outputs are among a set of discrete values, this task is known as classification. If the outputs are instead a continuous value, the task is known as regression. An alternative interpretation of such values can also allow for ranking, using the regression value or, for classification, the probabilities of a given instance belonging to a class.

There are a multitude of models and derivations thereof that can be applied to a supervised learning problem. They usually differ in their complexity, train time, adaptability, interpretability, etc. The choice of which model better suits a particular problem is non-trivial (No Free Lunch[84]) and the best approach is often to sample the most common methods and try to assess which ones warrant further examination.

The next subsections will highlight the most important aspects of the most relevant supervised learning algorithms

4.2.1 Logistic Regression

Logistic regression is, despite its name, a classification algorithm. It is named after the core function behind it, the logistic function (also known as sigmoid). This function is an S shaped curve with the ability to map any real value into a value within 0 and 1 (not including both). Given its linear nature, the outputs are simply the result of a linear weighted combination of the input features.

$$y = \frac{1}{1 + e^{-\theta_0 + \sum_{i=1} \theta_i \times x_i}} \tag{4.2}$$

The outputs of a logistic regression classifier are thus interpreted as the probability that a given instance belongs to a certain class.

It is a simple, well understood algorithm that despite it's simplicity, has been shown to perform well for several practical applications. Its main advantages lay precisely on its simplicity, as it requires little parameterization and given its linear nature is less likely to overfit.

Despite its simplicity, it was among the best performing algorithms, managing to outperform several more complex algorithms such as SVMs and Random Forests.

4.2.2 Decision Trees

Decision trees are machine learning algorithms that recursively split the feature space according to a conditional statement so that homogeneity is maximized within those new regions. As one may expect, a complete decision tree will exhibit a binary tree shape, with each node being a decision node, and each leaf the predicted value.

An important aspect in building a decision tree is thus to find the best feature-value pair by which to split the data. This is done by means of so-called impurity measures, which assess how pure a split is, by how much class uniformity is present in each outgoing branch of the decision node. A pure split of a binary classification problem would generate two nodes in which each node would only have instances of one of the classes. While many metrics exist, Gini-index(4.3) is the most commonly used impurity measure, and studies have shown there is little reason to use alternatives[74].

$$GiniIndex = 1 - \sum_{i=1}^{C} (p_i)^2 \tag{4.3}$$

With C being the total amount of classes and p_i the probability of each class in a given branch. Learning an optimal decision tree is known to be a NP-Complete problem[52], so in practice, these algorithms are based on greedy heuristics that make locally optimal decisions.

Once a decision tree is learnt, new data simply transverses the tree according to the decision nodes until the leaf with the prediction is reached.

Decision trees have a multitude of valuable properties: they are very easy to interpret, take feature interactions into account, are fast to train, require little to no data preprocessing and can handle both classification and regression problems. Their most significant downsides are their instability, meaning they are highly sensitive to changes in input data, and their tendency to easily overfit, that it to say, generalizing poorly to new, unobserved data. Decision trees are rarely used by themselves, but instead are core components of more advanced machine learning mechanisms like RandomForests (4.2.5.1) and Boosted Trees(4.2.5.2).

4.2.3 Support Vector Machines

Support Vector Machine (SVM)[6] are a popular machine learning algorithm that rose in popularity given its impressive classification performance on small to medium-sized datasets. SVM are maximum margin classifiers, this means SVMs explicitly try to find the decision boundary (separating hyperplane) that best separates two classes, with best being defined as the one that maximizes the distance between the hyperplane and the closest points on either side of the plane (support vectors).

Intuitively, data points that comprise the support vectors are the points more likely to be misclassified.

SVMs are also capable of handling non-linearly separable data by using the so called kernel trick to expand the feature set into higher dimensions so that the classes can be separated linearly. Since SVMs only require information regarding the support vectors, they are fairly memory efficient. Furthermore, they maintain effectiveness even in high dimentional spaces, are resistant to outliers and make fast predictions once trained. However, SVMs require a good selection of kernel functions and precise tuning of parameters and do not scale very well for large datasets.

Support Vector Machines were used in the work initially, but soon became apparent that their hefty limitations when dealing with large datasets, coupled with their already large training time and the fact they require additional computations if one wants to extract probabilities, made SVMs more cumbersome than the alternatives thus were dropped during the development process.

4.2.4 Neural Networks

An Artificial Neural Network (ANN) is a computational approach to learning roughly inspired by the workings of our own brain. Essentially, ANN try to simulate the behavior of a neuron using a set of weighted inputs together with an activation function. These weighted inputs are multiplied and added together, and if their sum is higher than a certain threshold then the neuron will "fire". Subsequently data can be used to train the neuron, that is to say, update the weights in such a way as to only fire when that is the correct answer (using an error function).

A single neuron has very limited predictive power, only being able to correctly classify linearly separable classes, that is to say, classes that can be separated by a single hyperplane. However, incorporating multiple neurons into a layered network structure allows the approximation of any continuous function and to any desired precision, as stated by the Universal approximation theorem[22].

A neural network will have an input layer (which takes the features of our data), an output layer(the predicted label/value) and layers in the middle, called the hidden layers. Neurons from a layer typically fully connect to neurons in the next layer.

Neural networks learn in a similar fashion to that of a single neuron. A data point is passed as input, and each neuron will fire or not according to the input and its set of weights. Once the output layer is reached, the weights of each individual neuron will be updated by means of the backpropagation algorithm.

Typically, neural networks require many iterations of the same data to learn. With stoppage criterion examples including the number of iterations or when the training error reaches a certain lower bound.

Neural networks have grown immensely popular in the last few decades, owing to the great variety of problems they can be applied to, their ability to be updated in real time and the low memory requirements once fully trained. Despite the advantages, training a neural network can be immensely time consuming and often requires large datasets. Furthermore, choosing the right network structure can be difficult. During the work, sklearn's Multi Layer Perceptron was the only neural network algorithm attempted, but its performance was not satisfactory enough and ended up not being used in the final solution.

4.2.5 Ensemble Methods

Ensemble methods are learning algorithms that, instead of focusing on a single very accurate model, train several, low-accuracy models and aggregate their predictions hoping to obtain an accurate meta-model.

The two most common approaches to ensemble methods are bootstrap aggregation (bagging) and boosting. In bagging, a learning algorithm is run several times using different, randomly sampled (with repetition) subsets of the data. The other approach, boosting, instead of training several models independently, instead trains each model sequentially with new models being trained with additional emphasis on correctly predicting the instances the previous model predicted wrong. They also vary in the way they make their final predictions, with bagging algorithms usually predicting the final estimation by majority voting, while boosting algorithms perform similarly, using weighted voting instead.

While any machine learning algorithm can be used for ensembles, decision trees are particularly well suited for the task, as their high variance all but assures each individual model will make different predictions, which is one of the most important properties for ensemble methods to perform well. The most popular methods of both approaches, Random Forests and Boosted Trees are described further in the following subsections.

4.2.5.1 Random Forests

Random forests[9] are a decision tree based learning algorithm that builds upon the concepts of bagging by adding yet another layer of randomness. Besides subsampling the input data, random forests also subsample the features themselves (known as random subspace). This extension stems from the fact that if all the predictive power is concentrated on only a few subset of the features, a great many individual trees may select these same features for splitting, making each model highly correlated.

Random forests address many of the issues around decision trees. The application of both baggging and random subspace methods make the algorithm robust to overfitting, and even turns the otherwise harmful variance traditional to decision trees into a boon, since instability is a pre-requisite for effective bagging[9].

Unfortunately, these improvements come at the expense of interpretability, as the number of decision trees increases, it becomes very difficult to understand the decision making behind the predictions. This issue has been somewhat addressed by random forests' ability to estimate feature importance by averaging the inpurity reductions of each feature over all the nodes in which a feature appears.

Overall random forests are incredibly potent machine learning models, capable of handling seamlessly both categorical and continuous data, estimate feature importances, handle non-linear relationships, are robust to overfitting, and all of this with very little parameterization, which makes it hard to make a bad random forest classifier/regressor.

Random forests, while not being the fastest algorithm to train and make predictions, are also efficient enough for a vast majority of applications, as long as the number of estimators is kept to a reasonable amount.

The algorithm was tested initially, but the superior performance of the boosted approach meant it was left out in the end.

4.2.5.2 Stochastic Gradient Boosting

Stochastic Gradient Boosting is a particular instance of the boosting approach. Traditional boosting algorithms can be interpreted as being an attempt to improve bagging algorithms and they share many similarities. In boosting, the system starts, like bagging, by collecting a random subsample with repetition from the training set. To each data instance, a weight is given, initially equal for all instances and adding up to one. A model is then trained on the selected data subset, and the training set is used to test the model. Depending on the model's performance, the tree will have associated a certain amount weight:

$$TreeWeight = \frac{1}{2}log(\frac{1-error}{error}) \tag{4.4}$$

If the tree performs accurately, its weight will be very high, and the opposite if such is not the case. Also adjusted are the weights of all the data instances, with more weight being given to the instances the current model failed to predict, and less to the ones predicted correctly. Afterwards, a new random subset is obtained, this time, the distribution of the subset will be biased according to the weight of the instances, so wrong instances will be more likely to be sampled, essentially training the new model with higher emphasis on the data instances the previous got wrong. The process will continue iterating a defined number of times, and the final prediction is made according to each tree weight.

Gradient boosting performs similarly, but deviates slightly from traditional boosting by, instead of assigning estimators different weights at every iteration, it trains the new models to the residuals of the previous model. XGBoost[19], the implementation used in the work, additionally extends this by incorporating a regularization term. XGBoost was one of the better performers, only competing with logistic regression.

4.3 BorutaPy

Boruta[50] is a feature selection algorithm developed with the express intent of finding the subset of *all-relevant* features from a dataset, contrasting with wrapper methods' problem of obtaining the *minimum-optimal* subset. Boruta is centered around calculating feature importances using randomized forests (boosted trees are also compatible) (see 4.2.5.1), which allow handling of feature correlation and non-linearity. The feature selection method starts by expanding the original set of features with so called "shadow" features, which are essentially copies of the real attributes shuffled within each column so as to eliminate correlation with the target variable (fig. 4.1).

These shadow features can be considered noise, and their calculated feature importance, an attempt at approximating the importance of a completely irrelevant feature.

At each iteration the shadow features are reshuffled and the feature importance of the real feature is compared with the feature importance of the best scoring shadow feature. Those that consistently perform worse than the best shadow feature across multiple iterations are deemed irrelevant and removed from the dataset. Conversely, a feature that consistently performs above the best scoring shadow feature is marked as relevant. The algorithm stops when all the remaining features have been marked as relevant or after a user defined number of iterations.

BorutaPy is a python open-source implementation of the boruta algorithm[23], that is several orders of magnitude more efficient than its original R counterpart. This performance disparity is owed to the powerful implementation of random forest algorithms available in python, as well as some adjustments made to this version, such as the ability to automatically adjust the number of estimators from the random forest according to the number of active features. Moreover, it provides a finer control in deciding which features are accepted or rejected by defining a percentile threshold to use instead of the shadow feature's maximum.

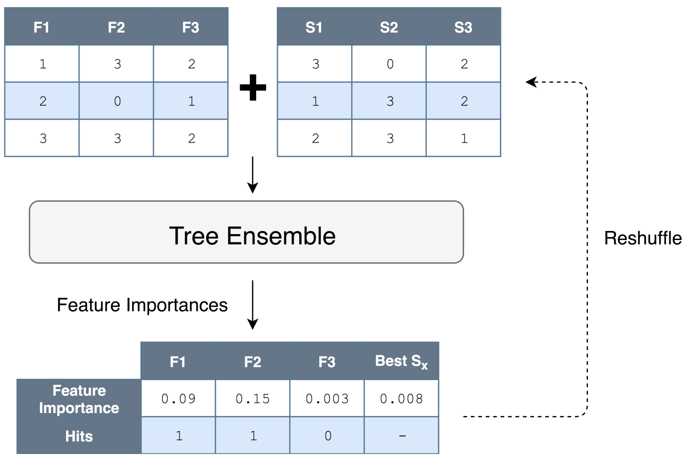


Figure 4.1: Diagram displaying boruta's feature selection procedure.

Since BorutaPy's implementation does not expose some intermediary results (namely the feature importances at each iteration), and the code is open source, a local modified version of this package was used throughout the work, that allows inspection of these values.

Chapter 5

Data Collection

Availability of high quality data is, arguably, the most important prerequisite for the development of any machine learning solution. More often than not, a sub-par algorithm with good data will yield better results than the best conceivable model with low quality, insufficient or corrupted data.

Unfortunately, there are often many setbacks in data acquisition, doubly so in fields such as molecular biology in which the means of collecting data are limited by available technology and domain knowledge, coupled with expensive and time consuming processes (ex: X-Ray Crystallography, NMR).

Development of a good scoring function requires access to datasets of several putative poses (known as decoys) where near-natives are paired with incorrect predictions, which, in turn, need to be generated by some docking framework using structural information of two interacting proteins in their unbound state and of the complex they form.

Conveniently, several datasets have been created for the specific purpose of creating and benchmarking protein-proteins scoring functions. Two of these will be described in more detail in section 5.3.

Exploration of possible data sources and benchmarks pertaining to protein interactions was thus a critical step for the development of a good solution. In the remainder of the chapter an introduction to the Protein Data Bank (PDB), followed by a description of the file format detailing molecular structure used throughout the project and ending with the choice and reasoning behind the dataset selected.

5.1 Protein Data Bank

In 1971, in efforts to facilitate the research of molecular biology, a worldwide repository of biological structures was established, the Protein Data Bank (PDB)[29]. It is a large and public domain repository managed by the Research Collaboratory for Structural Bioinformatics (RCSB)[45].

At the time of its inception, only a few structures were available. However, the PDB currently houses ∼149 424 (as of February 2019) biological structures, of which approximately 92% belong to proteins. The vast majority of these structures were determined by X-ray crystallography (89%) with NMR and Electron Microscopy being responsible for ∼8% and ∼2% respectively.

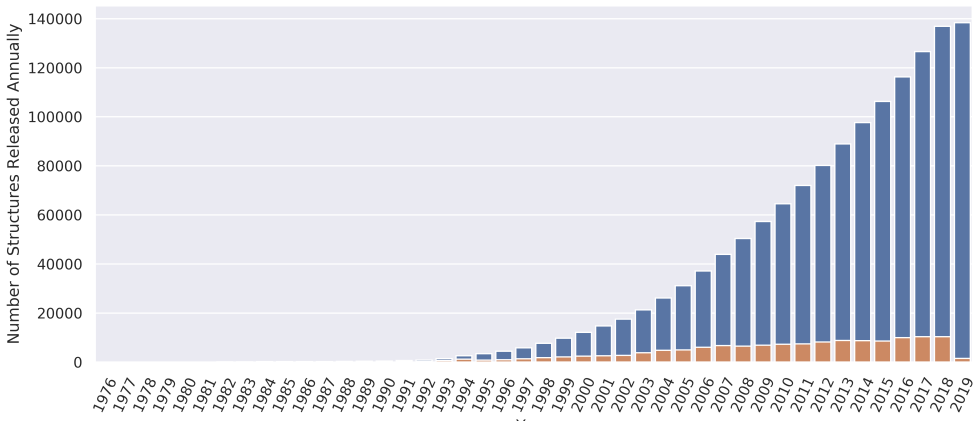


Figure 5.1: Total amount of protein structures deposited in PDB annually (in blue) and new protein structure per year (in orange)[70].

Each protein available in this database is given a unique 4-letter code, the PDB ID, which uniquely identifies a protein complex. Despite the seemingly large amounts of structural data available, it is important to note that it is but a fraction, and a small one at that, of all the proteins of known sequence. Furthermore, despite considerable effort to maintain a high quality standard, some of the structures available at PDB contain inconsistent, informal or erroneous data, with missing atoms or different levels of resolution. When building a dataset for scoring, one must also account for the substantial amount of redundancy present in the PDB.

Lastly, while the PDB provides structures of the unbound proteins and the complexes they form, they do not provide a decoy dataset, so one would have to be generated using some docking framework.

5.2 PDB Format

The vast majority of information regarding protein structure, including the one provided in the PDB is stored in pdb files. Pdb files are essentially essentially a text file containing multiple lines (records) containing some standardized keyword specifying what kind of information is provided at each row, coupled with the actual contents of said row.

```
ATOM      1  N   ASP A   1      11.869  13.685  12.658  1.00  1.00
ATOM      2  CA  ASP A   1      11.860  12.876  13.882  1.00  1.00
ATOM      3  C   ASP A   1      11.822  13.749  15.172  1.00  1.00
ATOM      4  O   ASP A   1      12.227  14.916  15.208  1.00  1.00
ATOM      5  CB  ASP A   1      13.067  11.934  14.008  1.00  1.00
ATOM      6  CG  ASP A   1      13.381  11.045  12.811  1.00  1.00
ATOM      7  OD1 ASP A   1      12.540  10.892  11.890  1.00  1.00
ATOM      8  OD2 ASP A   1      14.513  10.499  12.824  1.00  1.00
ATOM      9  N   VAL A   2      11.343  13.037  16.198  1.00  1.00
ATOM     10  CA  VAL A   2      11.261  13.696  17.536  1.00  1.00
ATOM     11  C   VAL A   2      12.530  13.201  18.259  1.00  1.00
ATOM     12  O   VAL A   2      12.904  12.022  18.149  1.00  1.00
ATOM     13  CB  VAL A   2       9.914  13.369  18.192  1.00  1.00
ATOM     14  CG1 VAL A   2       9.914  13.751  19.670  1.00  1.00
ATOM     15  CG2 VAL A   2       8.737  13.914  17.403  1.00  1.00
```

Figure 5.2: Excerpt of a PDB file from the receptor molecule of 1ay7 complex.

Protein structures are often reported in publications, and for these, their respective pdb files detail the author's names, date and method of discovery, literature references, biological source, etc. More importantly for this project, the files also contain a listing of every known atom of the molecule together with the following information: the sequential number of the atom in the protein; the chemical element; the residue it is part of; the chain it belongs to; the residue number; the 3D coordinates, in Angstroms (Å); and the occupancy and temperature factors.

For a more thorough detailing of the pdb file format and all the types of records it provides, the full description can be found at `http://www.wwpdb.org/documentation/file-format`.

5.3 Dockground Dataset

The Dockground[49, 56] resource is a collection of five integrated datasets created to provide all necessary information for all types of molecular structure research. For the purposes of this work however, only the decoys dataset is relevant (version 2.0)[49].

This decoy dataset was curated specifically to create and benchmark protein-protein scoring functions. The data was generated using GRAMM-X[79] docking framework using 396 different unbound-unbound complexes. For each of these complexes, 300 000 low resolution putative poses were generated, without ranking or scoring outside of shape complementarity. For 43 of these complexes, no near-natives were found, so they were excluded from the dataset.

A single near-native structure was chosen, with 99 accompanying incorrect matches selected in such a way as to maximally spread spatial distribution (see 5.3). The choice of using a near-native as opposed to the real known native pose is due to the fact that finding this position would set unrealistic expectations for scoring functions.

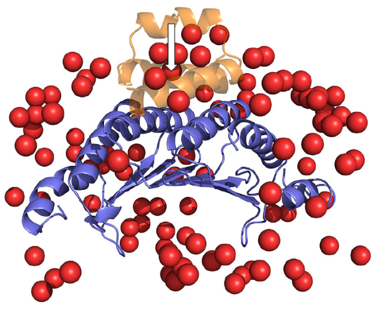


Figure 5.3: Example decoys from 1f93 complex generated by Dockground. False positives are shown by ligand center of mass (red) with the true positive identifyed by the arrow. Image taken from original paper[49].

All this information is packed into two pdb files per complex: *xxxx_u1.pdb* and *xxxx_u2_decoys.pdb*, with *xxxx* replaced by the pdb code. Each file contains the full atomic description of the unbound molecules, with the first file pertaining to the receptor, and the second one detailing the same information for the selected 100 poses, in which the first is the near-native.

Other possible datasets, such as the ZDOCK's dataset (available at: `https://zlab.umassmed.edu/zdock/decoys.shtml`) were explored, however, the dataset contains less protein diversity, 174 complexes from DBMK 4.0[40] as opposed to the 351 from Dockground, and no care is taken regarding redundancy (54 000 putative poses per complex), which, given time constraints and the available computational resources made Dockground a more attractive choice. The largest advantage to using the ZDOCK's dataset is that it comes bundled with the scores of the ZRANK (or IRAD) scoring functions, which would be beneficial for performance comparisons. Despite opting for the Dockground dataset, scripts to generate and compute scores for the ZDOCK dataset were created, but ultimately not used.

Chapter 6

Prototype Development

As displayed in chapter 3, there are several viable approaches to protein-protein scoring. From a machine learning perspective alone, this problem could be interpreted as classification, regression, clustering, ranking, or a combination of these.

However, the choice of dataset (see chapter 5) turns some of these approaches, namely clustering, less viable. This results from the reduced number of putative poses per complex, which were selected in a manner similar to clustering already. Given the data and the goal of the thesis, the problem was tackled as a classification problem, with estimated probabilities later used for the purposes of ranking the possible poses.

In the next section, a catalogue of the most important tools used during development is provided. The process of convertion of the input pdb files into meaningful features then follows, with analysis of the properties of each feature being detailed straight after. The reason and means of feature selection is given in section 6.3. Section 6.4 describes the evaluation metrics, as well as why these metrics are effective ways of assessing performance of the scoring function. The process of model building, experiences and hyperparameter tuning is explained in section 6.5 ending with definition of the final model.

6.1 Tools

Python is a general purpose programming language that has grown in popularity in the scientific, machine learning and data science community. It provides an extensive collection of powerful libraries with state-of-the-art performance allied with an easy to use syntax and interactive developing environment. For these reasons as well as familiarity with the language, Python was the chosen programming language for the development of this work. Other important tools used throughout the work were:

SciPy/Numpy[42, 81]: powerful library that adds multi-dimensional arrays and matrices along with many useful linear algebra and other high level mathematical functions;

Pandas[62]: useful library to manage and pre-process data, with tools to read and filter data, handle missing values and more;

Nglview/JMol[35, 67]: software to create interactive visualizations of protein structures;

PyRosetta[16]: A python based interface for the Rosetta molecular modeling suite. Includes functions to allow the extraction of some energetic features, as well as implementations of local search algorithms. Requires a license, free for academic or non-commercial software. Pyrosetta was the primary means of feature extraction used to convert the pdb structures into meaningful data suited for machine learning models;

Matplotlib/Seaborn[38, 83]: plotting libraries that allow the creation and visualization of several plots such as histograms, bar charts, scatterplots, etc;

SciKit-Learn[72]: a library that provides state-of-the-art implementations of several machine learning algorithms encompassing classification, regression, clustering, dimensionality reduction, validation, model selection, feature extraction, etc. All the models developed with the exception of the implementation of boosted trees had their basis on this library;

XGBoost[19]: optimized gradient boosting library. Provides parallel tree boosting and has been recently gaining traction and popularity given its performance at solving several machine learning problems.

BorutaPy[50]: Feature selection tool used in the work, section 4.3 provides a description of the process.

Scikit-Optimize[34]: Library that provides several optimization algorithms, used in the work for hyperparameter tuning using Bayesian search.

6.2 Feature Extraction

The chosen dataset provides the protein structures in two pdb files; one containing the tertiary structure of the receptor, and another containing the structure of one hundred decoys, of which one (the first) is a near-native. However, before applying any machine learning algorithm, it is necessary to first convert this information into an useful and labeled feature vector.

The feature extraction pipeline starts by using pyrosetta's parser to load the data into a suitable format for manipulation. Afterwards, the receptor and a random ligand are fed to

Pyrosetta Features	Description
fa_atr	Lennard-Jones attractive energy between atoms on different residues
fa_rep	Lennard-Jones repulsive energy between atoms on different residues
fa_dun	Internal energy of side-chain rotamers derived from Dunbrack's statistics
fa_elec	Coulombic electrostatic energy of interaction between non-bonded charged atoms
fa_pair	
fa_sol	Gaussian exclusion implicit solvation energy between atoms on different residues
p_aa_pp	Probability of amino acid identity given backbone angles
fa_sasa	Solvent accessible surface area
interface_dd_pair	
pack_stat	Score of how well a complex is packed
hbond_bb_sc	Energy of backbone-sidechain hydrogen bonds
hbond_lr_bb	Energy of long-range backbone-backbone hydrogen bonds
hbond_sr_bb	Energy of short-range backbone-backbone hydrogen bonds
hbond_sc	Energy of sidechain-sidechain hydrogen bonds
Other Features	**Description**
inter_size	Total number of residues considered part of the interface (10Å)
interface_ratio	Ratio of the total number of residues of the complex that comprises the interface
inter_hydro_ratio	Ratio of the hydrophobic residues of the interface
{atr, dun, elec, rep, sasa, pair, sol}	fa_{atr, dun, elec, rep, sasa, pair, sol} divided by the total number of residues in the complex
dd_pair	interface_dd_pair divided by the interface size
{hyd-hyd, neg-pos, ... , hyd-pos}	Counts for each residue-residue type contacts between the receptor and the ligand
n_contacts	Total number of contacts using 6Åcutoff
{elec/sol/sasa}_dif	Energy difference between the sum of fa_{elec/sol/sasa} energies of the molecules

Table 6.1: Features extracted for the project. Pyrosetta was used directly to calculate some of the features (top half) with the remaining ones (bottom half) having been posteriorly calculated using combinations of the other features, or obtained with custom scripts.

a simplified function that makes use of pyrosetta's exposed methods to extract solvation, Solvent Accessible Surface Area (SASA) and electrostatic energies for each molecule in isolation. These energies are then summed and stored for later use.

The procedure continues by iterating over every decoy, and generating a complex by merging each decoy with a copy of the receptor. The generated complex is then given to a second, more detailed function, and several physico-chemical features are extracted (table 6.1).

Immediately after, the interface of each complex is calculated, with a molecule's residue being considered part of the interface if its Cα is within 10Å of a residue on the other molecule. Knowledge of these residues is then used to calculate several interfacial statistics, namely: total interface size, interface ratio and ratio of hydrophobic residues. A more stringent criterion of 6Å is then applied, and any residue of a molecule within that threshold of a residue of the other is considered a contact. Contacts are then classified according to their residue-pair types2.1 (hydrophobic-hydrophobic, negative-positive, etc). The choice of thresholds was done with basis on molecular biology literature[1, 55].

Lastly, the difference between the energies of the isolated molecules and of each complex is calculated, ending the feature extraction process. A diagram illustrating the entire extraction process is shown in 6.1.

Of the 351 different protein complexes available on the Dockground dataset, features were extracted for 325 of them, with the remaining having had errors during calculation

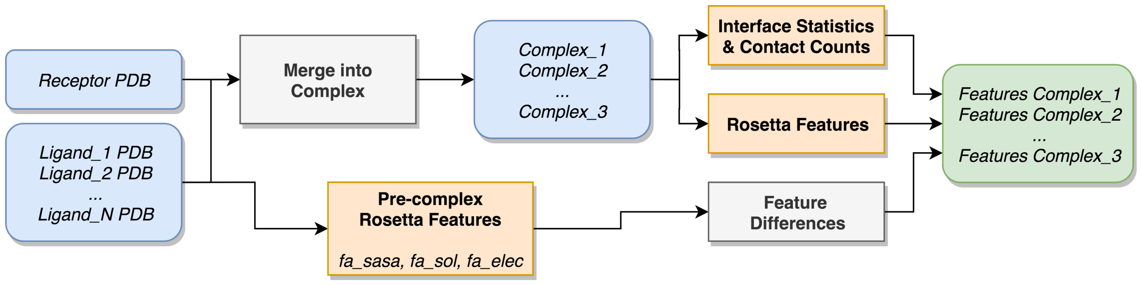


Figure 6.1: Diagram highlighting the principal components of the feature extraction pipeline.

with pyrosetta's methods.

6.3 Data Preparation and Feature Selection

There are many considerations one should address when developing a machine learning model. Several features were obtained from the intial pdb files, however, there is no guarantee that all of them will be useful for the developed models. This process of determining which features represent important aspects for the algorithms is known as feature selection. Feature selection is often an important step in developing a good model, reasons being that for some models, providing undescriptive features may actively harm performance; furthermore, by removing unecessary variables, one makes training easier and less likely to overfit. Features in the dataset also have different measurement scales, from residue counts to ratios to continuous values of arbitrary size. Making these features conmesurate is a critical step in guaranteeing that the higher order features do not have undue influence in the models. Finally, one must consider which metrics will be used to assess the performance of the models.

Before starting on attending to any of these issues, several additional features were calculated and combined to generate a new expanded feature set. The new calculated features were created by exploring expected relationships between the variables, for example calculating the total number of contacts, the total number of residues and the interface ratio ($\frac{interface_{nres}}{total_{nres}}$). For the full-atom features, a new set was calculated by dividing these values by the total number of residues. A similar procedure was applied to the **interface_dd_pair** feature by dividing it by the total number of interfacial residues. This resulted in an expanded set of 32 features.

Having finalized the feature set, the data was split the into two distinct subsets: a training set and a test set. This split was done by randomly sampling 20% of the data into the test set (6 500 instances), with the remaining belonging to the training set (260 000), with care being taken so all complexes from a certain pdb code belong to the same split. For all the subsequent methods, only the training data was accounted.

With the data now prepared, feature selection was the first issue to be addressed; there are many methodologies for tackling this problem, usually segmented into three distinct groups:

Filter methods typically encompass performing statistical tests and removing features above or below certain pseudo-arbitrary thresholds. Common examples include comparing feature correlation, either amongst features themselves (and removing those with extremely high correlation), or with the target variable (and removing those with very little). These methods, while often exceedingly fast and easy to apply to just about any dataset, fail to accurately determine a feature's true worth, and are unable to ascertain non-linear contributions.

Wrapper methods on the other hand, attempt to capture a feature's true importance by training several subsets of features on a given ML model, and compare its performance according to some metric. This method allows capturing more subtly relevant features and is tailor-suited to a particular model, but this comes at the expense of a much larger computational cost as well as dependency on that particular model.

Embedded method are conceptually similar to wrapper methods, but the feature importances are estimated during model building, without requiring additional external metrics. Common models capable of performing this type of feature selection are decision tree ensembles and logistic regression with L1-regularization.

Several of these approaches were tried, starting by plotting and inspecting histograms and boxplots for all variables, grouped by their labels (near-native or not). Immediately, several features stood out as features of interest, on account of their non-overlapping distributions against the target variable (fig.6.3).

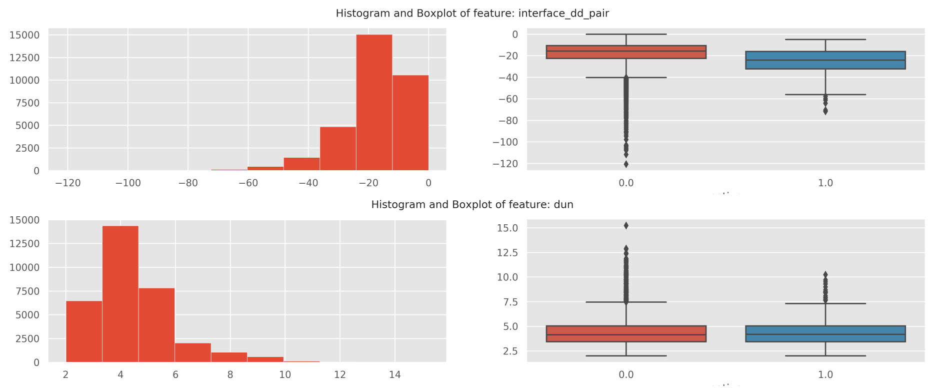


Figure 6.2: Examples of histograms(left) and boxplots(right) for features **interface_dd_pair** and **dun**. The top image shows a distinct difference in distributions when grouped by label, as opposed to the bottom one.

Unsurprisingly, most contact attributes, as well as the features calculated from the difference between the sum of the isolated molecules and the complex itself, displayed these distribution inequalities. Another notable fact is the presence of several outliers (shown as diamond symbols in the boxplots) in a majority of the features, and the skewed nature of many of their distributions.

The next step was to investigate correlations between the features, both between each other and between the target variable. The heatmap can be seen in figure 6.3.

There are several reasonably high correlated features, most of which can easily be explained. For example, **inter_size** being highly correlated to **sasa_dif** and **ncontacts** makes sense, considering a higher number of residues at the interface, blocks those residues from being accessible to the solvent. Somewhat surprisingly, is the apparent independence of the interface ratio and interface size, which indicate that the percentage of residues at the interface seems unafected by the total size of the complex.

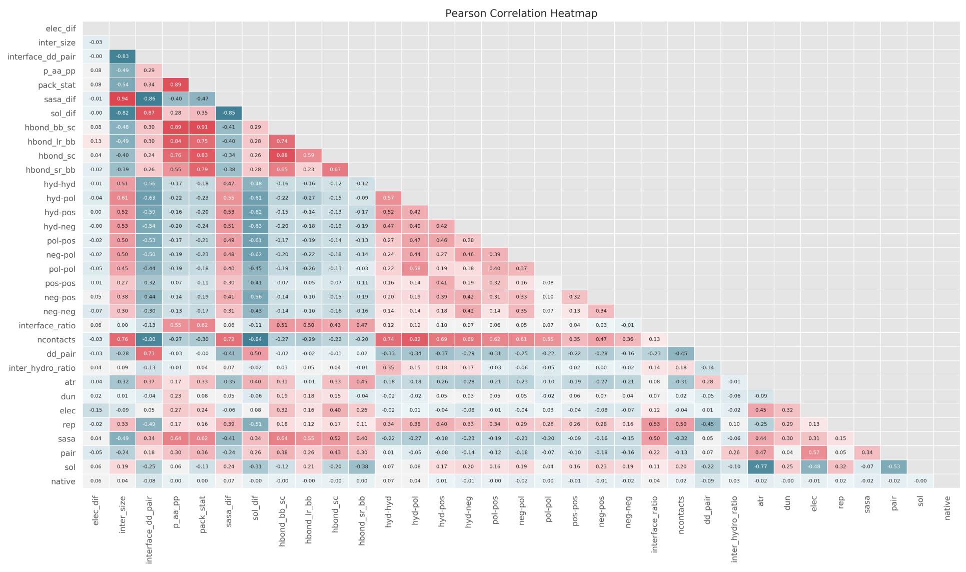


Figure 6.3: Bottom image displays the correlation heatmap between the features.

The correlation between the variables and the target variable are, expectedly so, small, with **elec_dif**, **sasa_dif**, **dd_pair**, **interface_dd_pair** and hydrophobic contacts displaying the highest absolute values between 0.06 and 0.09.

Nevertheless, correlation alone is not conclusive enough (unless the values are extreme) to deem a variable useless. A more definitive method should be employed. As such, a local version of the BorutaPy package was edited and used to obtain all the relevant features for the prediction. A description of boruta was provided in 4.3.

This procedure takes a decision tree ensemble estimator to obtain feature importances. The random forest classifier from sklearn was used for this, with class weights adjusted according to the ratio of instances of the positive class and of the negative class. Max tree depth was set to 3, as per recommended by the authors of the software and a percentile of 95 was used instead of the vanilla 100. Number of estimators is defined by boruta itself, so adjustment of that parameter was not required. The results can be seen in figure 6.4. Of the original 32 features, only 16 were kept by this procedure.

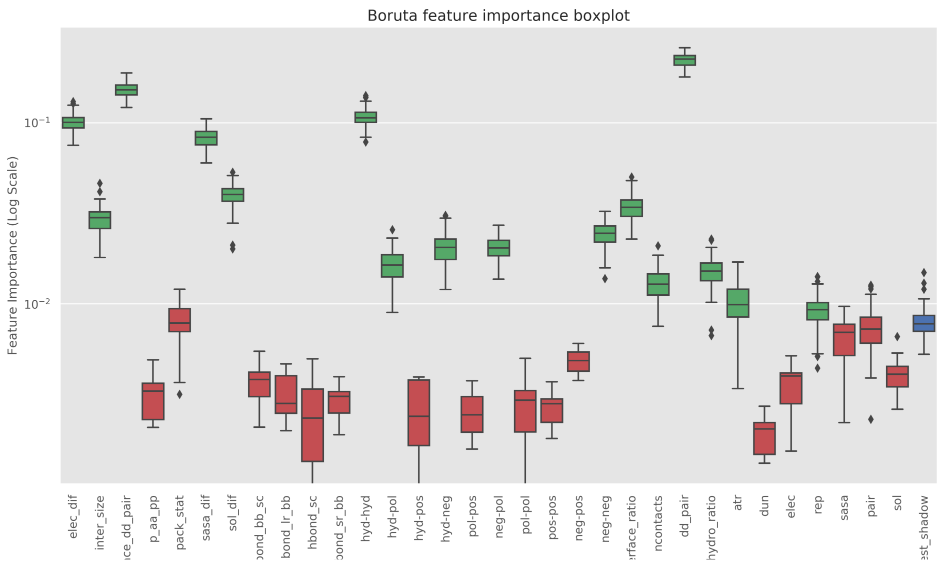


Figure 6.4: Boruta result plot. The blue (rightmost) boxplot corresponds to the best scoring shadow attribute. Green and red boxplots represent scores of accepted and rejected attributes.

For most machine learning algorithms, it is important to constrain the relative scaling of any particular feature to a comparable range. The two most common methods of doing so are range normalization and standardization or z-score normalization.

$$range(x_{if}) = \frac{x_{if} - min(f)}{max(f) - min(f)} \qquad (6.1) \qquad\qquad z-score(x_{if}) = \frac{x_{if} - \bar{x}_f}{std(f)} \qquad (6.2)$$

With x_{if} being the value for the i^{th} instance and f^{th} feature and $min(f)$, $max(f)$, $\bar{x}_f$ and $std(f)$ being the minimum, maximum, mean and standard-deviation for feature f.

Given the presence of several outliers, another useful form of normalization was tested, Interquantile range scaler(IQR):

$$IQRscale = \frac{x_{if} - \tilde{x}_f}{Q3 - Q1} \qquad (6.3)$$

With x_{if} retaining the previous meaning, $\tilde{x}_f$ being the median and $Q3$ and $Q1$ being the third and first quartile respectively. However, the data was not scaled immediately, instead being applied during model building, due to the usage of cross validation.

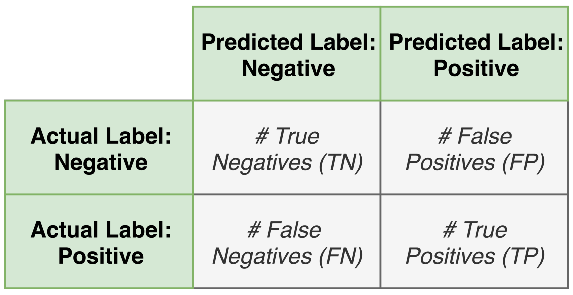
6.4 Performance Metrics

Development of good machine learning solutions can only be done effectively if the performance metrics used to evaluate them is adequate. The method and metrics for calculations is particularly important to get right when handling data with extreme class imbalances, which is the case in this work. Class imbalance is, as the name implies, when a given dataset has a large skew in the distributions of a class. This skewness can generally cause algorithms to perform inadequately on the minority class, which is often the class of actual interest.

There are many possible ways to tackle the issue of class imbalance, the two most common being sampling methods, either oversampling the minority class or undersampling the majority class, or to adjust the weights the algorithms give to the minority class to balance the discrepancy. Given the expected class distributions are, in fact, skewed, the latter approach was taken throughtout the project.

Unfortunately, the problems in dealing with imbalanced sets do not end there. Appropriate metrics to evaluate the algorithms should also be carefully picked, as several of the most popular means of assessing an algorithm's performance do not handle this issue well, and can provide misleading results. The most striking example is that of **accuracy** which can obtain obtain high values simply by predicting the majority class for every data instance.

Given the classification approach to the problem, the metrics discussed will be constrained to those used in this approach. Particularly, the problem will be framed as a binary classification. In binary classification, labels are usually considered positive or negative, with positive in this case being a native decoy pose and false one that is not. All possible classification scenarios can then be represented as a matrix like so:



	Predicted Label: Negative	**Predicted Label: Positive**
Actual Label: Negative	*# True Negatives (TN)*	*# False Positives (FP)*
Actual Label: Positive	*# False Negatives (FN)*	*# True Positives (TP)*

Figure 6.5: Example of a confusion matrix.

These measures make the essential components of a vast array of performance metrics. Given that some, as stated before, do not handle data imbalance adequately, these will be ommited. Instead, some useful metrics for this problem are:

$$precision = \frac{\#true\ positives}{\#true\ positives + \#false\ positives} \tag{6.4}$$

$$recall = \frac{\#true\ positives}{\#true\ positives + \#false\ negatives} \tag{6.5}$$

$$F_\beta = (1+\beta^2) \times \frac{precision \times recall}{\beta^2 \times precision + recall} \tag{6.6}$$

$$average\ precision = \sum_n (recall_n - recall_{n-1}) \times precision_n \tag{6.7}$$

With β being a value between 0 and infinity, that essentially controls how much importance one should give to either precision or recall. Typical values for β are 0.5 and 2. However, while these metrics may work in a classification approach, they fail make to make the distinction in regards to the position (per pdb complex) in which a particular decoy pose is ranked. As such, a different metric was used to ensure the quality of the top ranked poses, the Normalized Discounted Cummulative Gain, or *nDCG*. The formulas used to calculate the normalized DCG are:

$$DCG_p = \sum_{i=1}^{p} \frac{2^{rel_i} - 1}{log_2(i+1)} \tag{6.8}$$

$$nDCG_p = \frac{DCG_p}{IDCG_p}, \tag{6.9}$$

where p is the rank position, rel_i the graded relevance of the result at position i (for this case simply 1 for native and 0 for non-native), and $IDCG_p$ the ideal discounted cummulative gain.

Of course, one is not restricted exclusively to numerical metrics, with some common visual inspections being precision-recall curves and Receiver Operating Characteristic (ROC). Precision-recall curves in particular, are well suited to deal with the issues inherent to imbalanced datasets[28, 76].

Evaluating the ranking abilities of the algorithms are also expected, first so one can compare with external scoring functions, and also because ranking a native position first is more important than ranking it tenth and so on. The following additional metrics were established to assess and compare performance at the ranking task:

- The mean, median and standard deviation of the rank of near-native hits;
- The percentage of hits in the first 1, 10 and 50 putative poses;

These metrics should provide sufficient evidence of the performance of a scoring function for the Dockground dataset. Both the choice of best model and the comparative study of chapter 7 make use of these metrics for validation. Similar metrics were used in the papers of ZRANK and SPIDER[44, 73].

6.5 Model Building & Hyperparameter Tuning

It is difficult to know, *a priori*, exactly which ML method will perform better for a given dataset[84]. Machine learning algorithms can vary wildly in the decision boundaries they can learn, in how they handle missing data or types of features, parameterization required, etc.

As such, one of the first steps attempted was to simply try out several machine learning algorithms, using only basic, intuitive parameterization. This established a baseline for comparing the performance after optimization as well as a good intuition of which models to pursue further. For these models alone, the features were standardized before running them, as not doing so would lead to complications in using the support vector classifier.

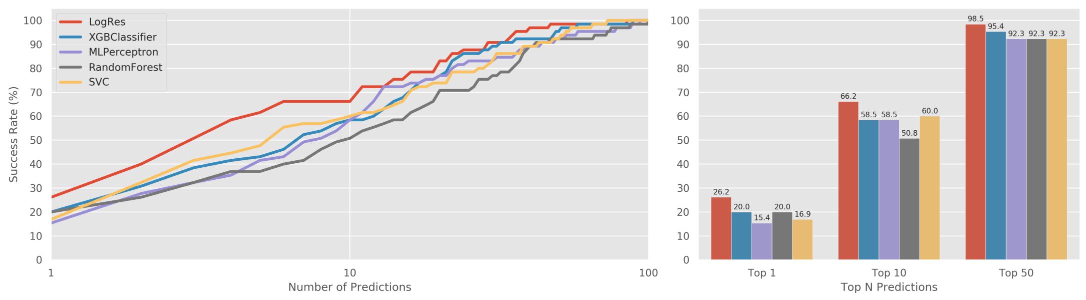


Figure 6.6: Success rate of the models at identifying near-natives among the false positives, with default parameterization.

Name	Train Time (s)	Prediction Time (s)
Support Vector Classifier	70.260455	1.623152
Random Forest	4.407558	0.309603
XGBoost Classifier	8.057078	0.051299
Logistic Regression	0.219064	0.003400
MLPerceptron Classifier	58.965497	0.013142

Table 6.2: Training and prediction times for the models used.

These results allowed for some overview of the expected performance of several models. Given the observed performance and the train an test times, only the Logistic Regression classifier and the XGBoost Classifier were retained for optimization. Optimization of the SVC would be very computationally expensive, given the amount of parameters to adjust.

Hyperparameter optimization is the name given to the process of choosing a set of optimal hyperparameters for a given machine learning algorithm. The parameters essentially control how a certain algorithm will learn the data.

Running models with arbitrarly chosen parameters is very rarely appropriate, so finding the optimal set of hyperparameters usually involves some searching procedure over possible arrangements of values to find the parameters that produce the best model, according to some evaluation criteria.

The three most common search procedures to finding the best set of hyperparameters are Grid search, Random search and Bayesian search. GridSearch is, apart from arbitrarily defined parameters, the most basic method of hyperparameter tuning. When given a grid of manually defined parameters, this algorithm exhaustively searches over all possible combinations of parameters, and trains a model using each combination to assess each combination's performance. Naturally, when training models with a vast number of hyperparameters to adjust, the complexity increases exponentially, making it ill suited for such algorithms. Despite these flaws, it remains a popular method of performing optimization for simple models.

Similarly simple, but with a much higher potential[4], is the widely used random search. As the name indicates, in random search, the parameter grid is defined as a set of distributions from which to sample instead of exhaustively defining each value. At each iteration, the algorithm randomly samples a combination of possible parameters and trains the model. Despite the stochastic nature of this approach, the fact that the choice of parameters is not dependent on any previous iteration makes this method trivially parallelized, as well as providing a finer granularity of optimization for numerical hyperparameters.

More recently, particularly for the optimization of neural networks, a new approach has been considered, Bayesian search. Studies have shown that Bayesian optimization often achieves better results in fewer iterations, when compared to Grid search or Random search[39]. Yet again, the name gives away the intuition behind the algorithm. Much like random search, a set or distribution for each hyperparameter is provided, and sampled a defined number of times in such a way as to cover as much of the parameter field as possible. From there, bayesian probabilities are calculated, and the algorithm starts to give preferencial treatment to vectors of hyperparameters around the values that previously resulted in better models. Nevertheless, the algorithm still seeks to explore other combinations of parameters, expected to be near to the optimum. This dependency on previous iterations however, makes this approach much harder if at all possible to parallelize, so despite it's better performance when compared to random search, the lack of paralellization makes it so there is a valid argument for the usage of random searches instead.

Given hardware limitations and the expected better performance for low-end CPUs, Bayesian search was the chosen procedure during the model development, with the implementation provided in the scikit-optimize model.

Hyperparameter optimization is almost always used in tandem with a procedure known as cross validation. Cross-validation is a method which attempts to make the most out of available data, by splitting the training dataset into n different partitions known as folds in which one of them becomes a validation set, i.e., a pretend test set. The remaining folds are fed to a model, that can then assess its performance on the validation fold. The process is repeated n times until all the different folds have been used once as the validation set. By averaging out the error or performance metrics over these folds, once can obtain a better estimate of the true error. A diagram depicting the cross-validation procedure can be found in 6.7

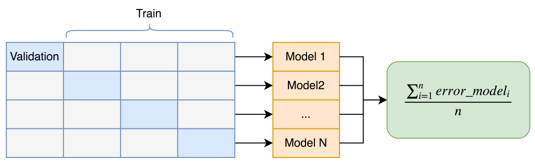


Figure 6.7: Illustration of the cross-validation method (4-Fold in this diagram).

For the two chosen machine learning models, meaning logistic regression and boosted trees, the pipeline of attempting to improve the model by tuning their hyperparameters went as such: 5-folds were extracted from the training data, guaranteeing that each fold contains the entirety of some randomly chosen putative poses from a single pdb file. Then, adequate hyperparameter grids were carefully selected for each model, and given to the sklearn optimizer.

More specifically, Logistic regression takes two important parameters, the regularization strength (C) and the regularization type (L1 or L2). The C parameter essentially controls the complexity of a given model, with small values of C increasing the regularization strength, thus forcing the model to be simpler; and large values decreasing it, thus leading to a more complex model.

	XGBoost Parameters
Max Depth	$(3, 12)$
Gamma	$(1^{-4}, 1^{4})$
Reg Alpha	$(1^{-4}, 1^{4})$
Reg Lambda	$(1^{-4}, 1^{4})$
Subsample	$(0.5, 1)$
Column Sample	$(0.3, 1)$
Learning Rate	$(0.0, 1.0)$
Number Estimators	$[100, 500, 1000]$

Table 6.3: Gradient boosted tested parameterization, parenthesis mean a uniform sampling between the two values, brackets indicate discrete options.

XGBoost however, provides many possible parameters to optimize, despite usually providing good performance out of the box. The most important to keep in mind being: the max depth of a tree, with higher numbers leading to more complex models and thus more likely to overfit; subsample ratio, which as the name indicates is the ratio of training instances to subsample from the training data; colsample_by_tree, which is similar to the subsample ratio, but for features instead; the learning rate, which controls how much we adjust the weights with respect to the gradient; the gamma, reg_alpha(L1) and reg_lambda(L2) which all act regularization parameter; and the number of estimators. Both for XGDBoost and for the logistic regression, class weights were adjusted so as to be balanced; meaning that for this dataset, each positive instance is weighted 99 times as much as a negative one (the ratio of negatives to positives in this dataset). Without this adjustment, the algorithms could resort simply to vote for the majority class and trivially minimizing error. The parameter grid for XGBoost can be seen in table 6.3.

As the number of hyperparameters and their options increases, so should the number of iterations in the bayesian search, as such, for logistic regression, a smaller number of 50 was chosen, with boosted trees being set for 150 iterations. At every iteration, in a sample parameter vector is extracted, and five models are trained according to the IQR normalized cross-validation splits. As previously discussed(see 6.3), IQR is the range comprising the middle 50% of the data, and this normalization is important so as to convert the features into a similar range while remaining unaffected by outliers. The metric selected for evaluating which iteration provided the best results was the normalized Discounted Cummulative Gain (nDCG). This presented an issue as sklearn is not well prepared to handle ranking methodologies. As such, a customized scoring function was created, that would group the ranking predictions per pdb complex, calculate the nDCG for each complex set, then average out the results and return the value as the metric to optimize. The choice of rank threshold for the nDCG was chosen to be 10, as it seemed a reasonable compromise given the goal and the number of putative decoys per group.

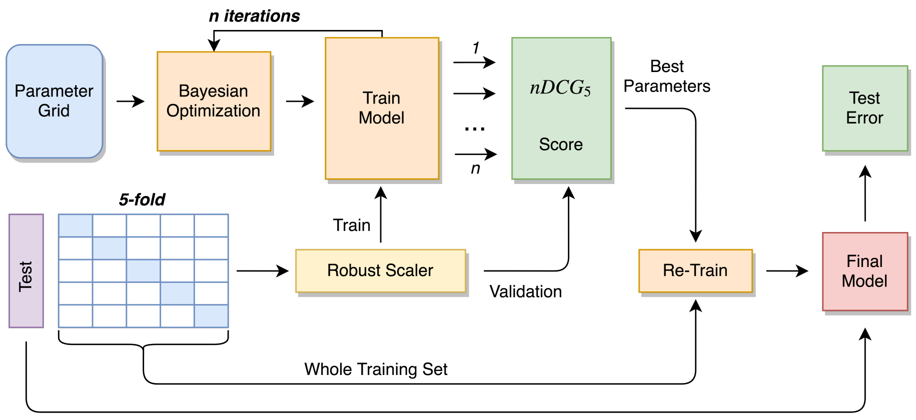


Figure 6.8: Illustration of the entire model-building pipeline. The parameter grid contains possible parameters for a given model. A combination of these parameters is selected by Bayesian optimization algorithm, which then feeds them to a model that is trained five times with the same parameters on different cross-validation splits that were IQR scaled using RobustScaler. This occurs for *n* iterations, after which the best model's parameters according to the specified criterion obtained by testing the trained model on the validation splits is fed to another model that is trained on the full training set. This generated the finalized model, which is then used on the test set to assess an estimate of the true error.

Once the best set of parameters is found, the whole training set is used to train a finalized model on those parameters. These models were then compared, with the final results being, in somewhat surprising fashion, better for the logistic regression model (see fig.6.9 and table 6.4). Of particular note is the fact that for neither of them the optimization process generated models amazingly better than the vanilla ones. While

	Logistic Regression	XGBoost Classifier
Mean	10.40	13.74
Median	3.00	5.00
Std. Deviation	15.13	19.63

Table 6.4: Mean, Median and Standard Deviation of the near-native hit ranks for both Logistic Regression and XGBoosted trees finalized model.

this does not seem particularly surprising, as the nature of the problem is difficult, it may also indicate that the features themselves might be bottlenecking the scoring.

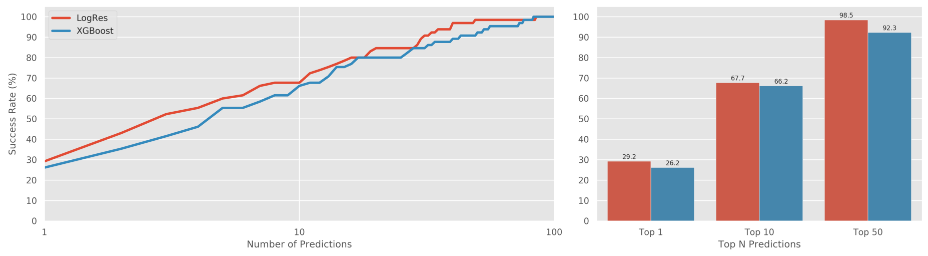


Figure 6.9: Comparative performance of the finalized models after being trained on the whole train dataset with the optimal set of parameters.

Chapter 7

Comparative Study

In this chapter, a performance assessment study is described. Three different state of the art scoring functions were used for validation, namely SPIDER, FastContact and dDFIRE.

The choice of these scoring functions is tied to the diversity in approach they display, with FastContact being a force-field based scoring function, SPIDER being geometry and statistics based and dDFIRE being a statistical potential. Moreover, dDFIRE uses a full atom description, much like the scoring solution developed, while FastContact uses a simplified, coarse atom representation.

The pool of potential choices were however, fairly limited as several other scoring functions were not available for local use, or could not be found decoupled from their docking procedure, or required special dependencies of format files making their usage difficult. Nevertheless, additional work went into attempting to find and use additional functions.

For all the tested functions, the files were prepared as specified in the provided documentation, and the default parameterization was used. Given the format of the dockground dataset, and the different means of output of each scoring function, scripts were created to handle all the file manipulation necessary to run each function, and then to parse the output files into appropriate python data structures for manipulation.

For dDFIRE, this meant merging the dockground decoys with copies of the receptor and saving the generated structures as pdb files. The software is then run on these files and the generated output file parsed with regex and converted into a pandas dataframe, saving only the final score.

FastContact requires similar file manipulation, but this time with the receptor and putative decoys split instead. As such, the dockground files are read and the ligands split across several files. A txt file containing the location for all of them is then generated, as required for FastContact. The output file is once again parsed into a dataframe, but as FastContact does not provide a single score at the end, but instead the two terms, electrostatics and desolvation, both were saved. Given that, the sum of these values and the values in isolation were used to assess the scoring performance, the best performer in regards to ranking was the desolvation term alone, so the results provided for FastContact refer to that.

Lastly, SPIDER also requires the decoy files split, so code got reused to do exactly that. SPIDER output file would not contain scores for certain poses with the default parameterization, so a threshold cutoff was relaxed so every pose had a score. Unlike the previous two, SPIDER output file already provided the ranks themselves, so these were used for evalutation.

The test set used for evaluating the scoring functions contained 65 complexes (100 putative poses each, for a total of 6500 instances) from the Dockground dataset, that were not used in training. It is also relevant to state that not all these functions successfully scored all the instances of the test dataset. FastContact failed to calculate for the *'1is7'* complex, and SPIDER failed for pdbs *'2a1t'*, *'1is7'*, *'4fqr'*, *'4gxu'*, *'1w1i'*, *'3l5n'*.

All of the provided scoring functions have been validated in CAPRI, and several assessment studies have already made regarding these scoring functions[33], highlighting their relevancy.

In the next sections the models developed throughout the thesis will be compared, first according to their ranking ability, and then in their temporal component.

7.1 Ranking Performance

The goal of any protein-protein scoring function is to identify the near-native poses amongst the set of all computationally generated ones for each pdb complex. Furthermore, an ideal scoring function would rank poses according to their structural similarity with the native complex. Assessing the performance of the scoring function required comparing how well it performed, when put up against other state-of-the-art scoring functions using the same input data.

For comparing these external functions, the following criteria were used: the cumulative percentage of a scoring function ability at successfully identifying the near-native within the first 1, 10 and 50 ranks; the metrics pertaining to the predicted near-native ranks of the scoring functions, namely the mean, median and standard-deviation;

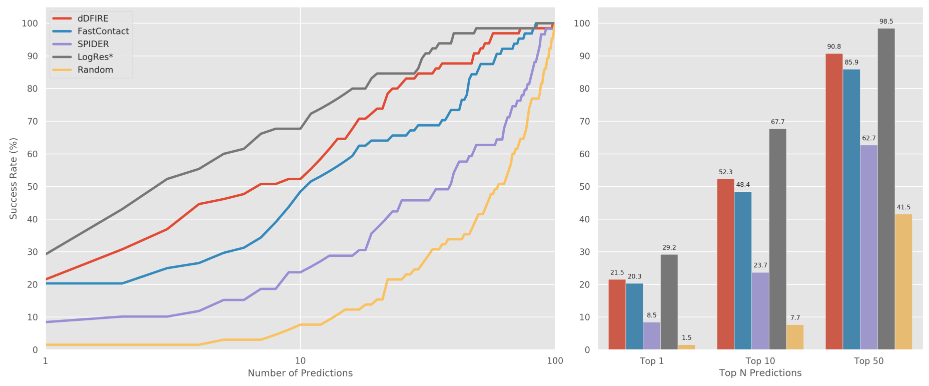


Figure 7.1: Success rate of the several scoring approaches at identifying the near native in the Top 1, 10 and 50 positions for the 65 dockground complexes.

The results show that SPIDER seems to perform the worst out of the evaluated scoring functions by a fair margin, finding the near-native in the first 50 poses in 62.7% of the cases. FastContact and dDFIRE perform similarly, with slight advantage to dDFIRE. The developed solution seems to perform the best out of the tested scoring functions for this dataset.

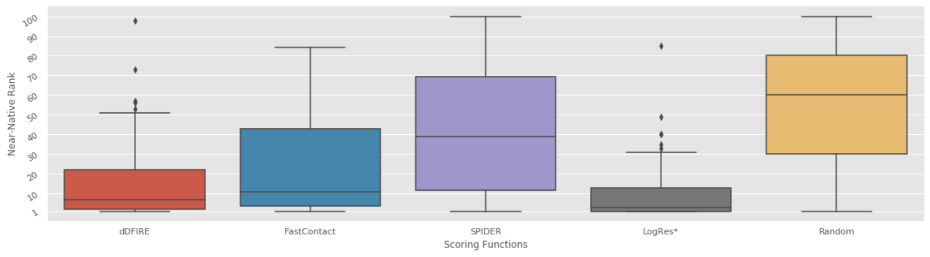


Figure 7.2: Boxplot of the hit stats of every tested scoring function.

With the exception of a few outliers, in the worst scenario the developed function still ranks the near-native in the top 30, with most being near the top 15.

7.2 Temporal Performance

Despite not being the most important factor, nor the end goal of the project, it still remains relevant to assess the temporal performance of the developed scoring model. Figure 7.3 details the running times of all the tested scoring functions.

Immediately, it is noticeable that the higher predictive power of the developed function came at the cost of temporal complexity. FastContact lives up to its name, displaying incredible times per complex; this makes sense given that FastContact is comprised of only two terms. Expectedly, the size of the protein complexes has an incredible negative

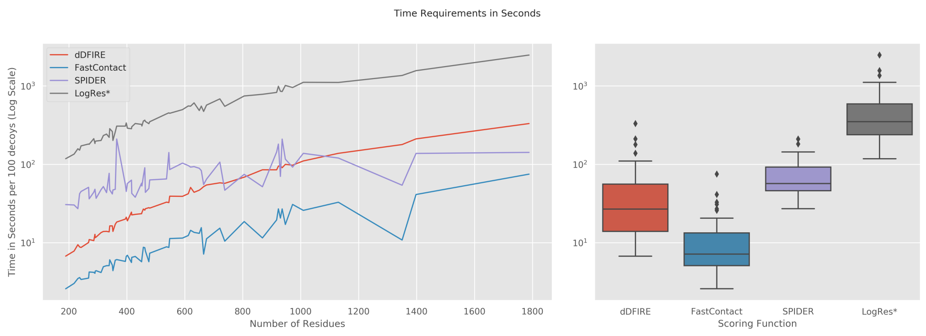


Figure 7.3: Scoring times, in seconds, per set of one hundred complexes from Dockground. Left image displays how time changes with the number of residues of the complex. Right image shows the boxplot of times per scoring function.

effect on the calculation times, increasing it by orders of magnitude. Notably, SPIDER seems to increase at a lower rate than all the other functions with total number of residues.

Nevertheless, despite the higher time complexity displayed for the developed function, one should bear in mind that the feature extraction process takes up most of this computational time; and that this process can easily be parallelized, which should boost the time considerably. Finally, even assuming these values were final and unimproveable, the better results are easily worth the exchange given the context of protein-protein scoring. It is far more important to reliabilty obtain the correct models than it is to do so very fast.

Chapter 8

Conclusion and Future Work

Throughout the work, several conclusions and insights were reached regarding the problem of protein docking.

The first is that it is clearly a very difficult problem, despite continuous effort of many researchers, a solution for this problem is still illusive. Nevertheless, the sheer amount of distinct possible ways to tackle the issue, from classification, ranking, clustering, etc. also means there may be hope of combining the best of these approaches or discovering new ones that may help in discovering new and improved solutions.

Machine learning has been growing in popularity and been applied to a plethora of different fields, and in this work was once again shown that machine learning approaches can compete with current state-of-the-art solutions, regardless of the field.

Many difficulties and setbacks were met throughout the development process, often due to my lack of knowledge, but I can say with certainty to have a newfound admiration for biology and the intricate ways in which life transpires.

There are several possible improvements to be made as future work. The developed solution, while competitive, can be improved on its efficiency by parallelizing the feature extraction pipeline. Moreover, future research should likely focus on the development of better, more meaningful features, as they seem to be a clear bottleneck in current methodologies, homology or secondary structure information for example, might help boosting performance. Most certainly this process will be arduous, not only due to the inherent difficulty of the task itself, but also because it will likely require collaboration and expertise in both biology and computation.

Integration of the scoring function in an actual docking framework might also prove beneficial, as one can likely find opportunities to improve the scoring function with a more unified approach that complements the strengths and weaknesses of the docking process.

With the description of the work, I believe the main objectives of this thesis were acomplished. From start to finish, a protein scoring function was developed, that was shown to perform competitively with other state-of-the-art scoring functions. Its creation involved solving many issues along the way: understanding the biological concepts fundamental to the understanding of the work; sampling the field of protein docking and scoring algorithms, and the plethora of different viable approaches to do so; looking for and obtaining good quality data for developing the function; learning about which potential features could be useful; finding the tools and creating the scripts and pipelines to extract said features; learning how to create descriptive visualizations of both graphs and protein structures; learning of all the possible machine learning algorithms used in the field, and the many metrics one can use; learning how to manipulate the data and optimize a given learning algorithm to generate a solution for the task; and finally, learning how to write a thesis.

Bibliography

[1] B. Adhikari and J. Cheng. “Protein residue contacts and prediction methods.” In: *Data Mining Techniques for the Life Sciences*. Springer, 2016, pp. 463–476.

[2] S. Ansari and V. Helms. “Statistical analysis of predominantly transient protein–protein interfaces.” In: *Proteins: Structure, Function, and Bioinformatics* 61.2 (2005), pp. 344–355.

[3] D. Bandyopadhyay and J. Snoeyink. “Almost-Delaunay simplices: Robust neighbor relations for imprecise 3D points using CGAL.” In: *Computational Geometry* 38.1-2 (2007), pp. 4–15.

[4] J. Bergstra and Y. Bengio. “Random search for hyper-parameter optimization.” In: *Journal of Machine Learning Research* 13.Feb (2012), pp. 281–305.

[5] A. J. Bordner and R. Abagyan. “Statistical analysis and prediction of protein–protein interfaces.” In: *Proteins: Structure, Function, and Bioinformatics* 60.3 (2005), pp. 353–366.

[6] B. E. Boser, I. M. Guyon, and V. N. Vapnik. “A training algorithm for optimal margin classifiers.” In: *Proceedings of the fifth annual workshop on Computational learning theory*. ACM. 1992, pp. 144–152.

[7] C.-I. Brändén and J. Tooze. *Introduction to protein structure*. Taylor & Francis, 1999.

[8] A. Breda, N. F. Valadares, O. N. de Souza, and R. C. Garratt. “Protein structure, modelling and applications.” In: *Bioinformatics in Tropical Disease Research: A Practical and Case-Study Approach [Internet]*. National Center for Biotechnology Information (US), 2007.

[9] L. Breiman. “Random forests.” In: *Machine learning* 45.1 (2001), pp. 5–32.

[10] B. R. Brooks, R. E. Bruccoleri, B. D. Olafson, D. J. States, S. a. Swaminathan, and M. Karplus. “CHARMM: a program for macromolecular energy, minimization, and dynamics calculations.” In: *Journal of computational chemistry* 4.2 (1983), pp. 187–217.

[11] M. Bueno, C. J. Camacho, and J. Sancho. “SIMPLE estimate of the free energy change due to aliphatic mutations: superior predictions based on first principles.” In: *Proteins: Structure, Function, and Bioinformatics* 68.4 (2007), pp. 850–862.

[12] F. J. Burkowski. *Structural bioinformatics: an algorithmic approach*. Chapman and Hall/CRC, 2008.

[13] C. J. Camacho and S. Vajda. “Protein–protein association kinetics and protein docking.” In: *Current opinion in structural biology* 12.1 (2002), pp. 36–40.

[14] C. J. Camacho and C. Zhang. “FastContact: rapid estimate of contact and binding free energies.” In: *Bioinformatics* 21.10 (2005), pp. 2534–2536.

[15] C. J. Camacho, D. W. Gatchell, S. R. Kimura, and S. Vajda. “Scoring docked conformations generated by rigid-body protein-protein docking.” In: *Proteins: Structure, Function, and Bioinformatics* 40.3 (2000), pp. 525–537.

[16] S. Chaudhury, S. Lyskov, and J. J. Gray. “PyRosetta: a script-based interface for implementing molecular modeling algorithms using Rosetta.” In: *Bioinformatics* 26.5 (2010), pp. 689–691.

[17] R. Chen, J. Mintseris, J. Janin, and Z. Weng. “A protein–protein docking benchmark.” In: *Proteins: Structure, Function, and Bioinformatics* 52.1 (2003), pp. 88–91.

[18] R. Chen, L. Li, and Z. Weng. “ZDOCK: an initial-stage protein-docking algorithm.” In: *Proteins: Structure, Function, and Bioinformatics* 52.1 (2003), pp. 80–87.

[19] T. Chen and C. Guestrin. “XGBoost: A Scalable Tree Boosting System.” In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD ’16. San Francisco, California, USA: ACM, 2016, pp. 785–794. ISBN: 978-1-4503-4232-2. DOI: 10.1145/2939672.2939785. URL: http://doi.acm.org/10.1145/2939672.2939785.

[20] C. Chothia and J. Janin. “Principles of protein–protein recognition.” In: *Nature* 256.5520 (1975), p. 705.

[21] S. R. Comeau, D. W. Gatchell, S. Vajda, and C. J. Camacho. “ClusPro: an automated docking and discrimination method for the prediction of protein complexes.” In: *Bioinformatics* 20.1 (2004), pp. 45–50.

[22] G. Cybenko. “Approximation by superpositions of a sigmoidal function.” In: *Mathematics of control, signals and systems* 2.4 (1989), pp. 303–314.

[23] H. Daniel. *Python implementation of the Boruta all-relevant feature selection method.* 2015. URL: https://github.com/scikit-learn-contrib/boruta_py (visited on 12/2018).

[24] S. De, O Krishnadev, N Srinivasan, and N Rekha. “Interaction preferences across protein-protein interfaces of obligatory and non-obligatory components are different.” In: *BMC Structural Biology* 5.1 (2005), p. 15.

[25] C. Dominguez, R. Boelens, and A. M. Bonvin. “HADDOCK: a protein- protein docking approach based on biochemical or biophysical information.” In: *Journal of the American Chemical Society* 125.7 (2003), pp. 1731–1737.

[26] R. Esmaielbeiki and J.-C. Nebel. "Unbiased protein interface prediction based on ligand diversity quantification." In: (2012).

[27] R. Esmaielbeiki and J.-C. Nebel. "Scoring docking conformations using predicted protein interfaces." In: *BMC bioinformatics* 15.1 (2014), p. 171.

[28] P. Flach and M. Kull. "Precision-recall-gain curves: PR analysis done right." In: *Advances in Neural Information Processing Systems*. 2015, pp. 838–846.

[29] G. Gilliland, H. M. Berman, H. Weissig, I. N. Shindyalov, J. Westbrook, P. E. Bourne, T. N. Bhat, and Z. Feng. "The Protein Data Bank." In: *Nucleic Acids Research* 28.1 (Jan. 2000), pp. 235–242. ISSN: 0305-1048. DOI: 10.1093/nar/28.1.235. eprint: http://oup.prod.sis.lan/nar/article-pdf/28/1/235/9895144/280235.pdf. URL: https://dx.doi.org/10.1093/nar/28.1.235.

[30] F. Glaser, D. M. Steinberg, I. A. Vakser, and N. Ben-Tal. "Residue frequencies and pairing preferences at protein–protein interfaces." In: *Proteins: Structure, Function, and Bioinformatics* 43.2 (2001), pp. 89–102.

[31] M. M. Gromiha, K Yugandhar, and S. Jemimah. "Protein–protein interactions: scoring schemes and binding affinity." In: *Current opinion in structural biology* 44 (2017), pp. 31–38.

[32] I. Halperin, B. Ma, H. Wolfson, and R. Nussinov. "Principles of docking: An overview of search algorithms and a guide to scoring functions." In: *Proteins: Structure, Function, and Bioinformatics* 47.4 (2002), pp. 409–443.

[33] L. Han, Q. Yang, Z. Liu, Y. Li, and R. Wang. "Development of a new benchmark for assessing the scoring functions applicable to protein–protein interactions." In: *Future medicinal chemistry* 10.13 (2018), pp. 1555–1574.

[34] T. Head, MechCoder, G. Louppe, I. Shcherbatyi, fcharras, Z. Vinícius, cmmalone, C. Schröder, nel215, N. Campos, T. Young, S. Cereda, T. Fan, rene rex, K. K. Shi, J. Schwabedal, carlosdanielcsantos, Hvass-Labs, M. Pak, SoManyUsernamesTaken, F. Callaway, L. Estève, L. Besson, M. Cherti, K. Pfannschmidt, F. Linzberger, C. Cauet, A. Gut, A. Mueller, and A. Fabisch. *scikit-optimize/scikit-optimize: v0.5.2*. Mar. 2018. DOI: 10.5281/zenodo.1207017. URL: https://doi.org/10.5281/zenodo.1207017.

[35] A. Herraez. "Biomolecules in the computer: Jmol to the rescue." In: *Biochemistry and Molecular Biology Education* 34.4 (2006), pp. 255–261.

[36] D. Hoffmann, B. Kramer, T. Washio, T. Steinmetzer, M. Rarey, and T. Lengauer. "Two-stage method for protein- ligand docking." In: *Journal of medicinal chemistry* 42.21 (1999), pp. 4422–4433.

[37] J. Huan, W. Wang, and J. Prins. "Efficient mining of frequent subgraphs in the presence of isomorphism." In: *Third IEEE International Conference on Data Mining*. IEEE. 2003, pp. 549–552.

[38] J. D. Hunter. "Matplotlib: A 2D graphics environment." In: *Computing in science & engineering* 9.3 (2007), p. 90.

[39] F. Hutter, H. H. Hoos, and K. Leyton-Brown. "Sequential model-based optimization for general algorithm configuration." In: *International Conference on Learning and Intelligent Optimization*. Springer. 2011, pp. 507–523.

[40] H. Hwang, T. Vreven, J. Janin, and Z. Weng. "Protein–protein docking benchmark version 4.0." In: *Proteins: Structure, Function, and Bioinformatics* 78.15 (2010), pp. 3111–3114.

[41] J. Janin, K. Henrick, J. Moult, L. Ten Eyck, M. J. Sternberg, S. Vajda, I. Vakser, and S. J. Wodak. "CAPRI: a critical assessment of predicted interactions." In: *Proteins: Structure, Function, and Bioinformatics* 52.1 (2003), pp. 2–9.

[42] E. Jones, T. Oliphant, and P. Peterson. "{SciPy}: Open source scientific tools for {Python}." In: (2014).

[43] E. Katchalski-Katzir, I. Shariv, M. Eisenstein, A. A. Friesem, C. Aflalo, and I. A. Vakser. "Molecular surface recognition: determination of geometric fit between proteins and their ligands by correlation techniques." In: *Proceedings of the National Academy of Sciences* 89.6 (1992), pp. 2195–2199.

[44] R. Khashan, W. Zheng, and A. Tropsha. "Scoring protein interaction decoys using exposed residues (SPIDER): a novel multibody interaction scoring function based on frequent geometric patterns of interfacial residues." In: *Proteins: Structure, Function, and Bioinformatics* 80.9 (2012), pp. 2207–2217.

[45] A. Kouranov, L. Xie, J. de la Cruz, L. Chen, J. Westbrook, P. E. Bourne, and H. M. Berman. "The RCSB PDB information portal for structural genomics." In: *Nucleic acids research* 34.suppl_1 (2006), pp. D302–D305.

[46] D. Kozakov, K. H. Clodfelter, S. Vajda, and C. J. Camacho. "Optimal clustering for detecting near-native conformations in protein docking." In: *Biophysical journal* 89.2 (2005), pp. 867–875.

[47] D. Kozakov, R. Brenke, S. R. Comeau, and S. Vajda. "PIPER: an FFT-based protein docking program with pairwise potentials." In: *Proteins: Structure, Function, and Bioinformatics* 65.2 (2006), pp. 392–406.

[48] D. Kozakov, D. Beglov, T. Bohnuud, S. E. Mottarella, B. Xia, D. R. Hall, and S. Vajda. "How good is automated protein docking?" In: *Proteins: Structure, Function, and Bioinformatics* 81.12 (2013), pp. 2159–2166.

[49] P. J. Kundrotas, I. Anishchenko, T. Dauzhenka, I. Kotthoff, D. Mnevets, M. M. Copeland, and I. A. Vakser. "Dockground: A comprehensive data resource for modeling of protein complexes." In: *Protein Science* 27.1 (2018), pp. 172–181.

[50] M. B. Kursa, W. R. Rudnicki, et al. "Feature selection with the Boruta package." In: *J Stat Softw* 36.11 (2010), pp. 1–13.

[51] S. Labeit and B. Kolmerer. "Titins: giant proteins in charge of muscle ultrastructure and elasticity." In: *Science* 270.5234 (1995), pp. 293–296.

[52] H. Laurent and R. L. Rivest. "Constructing optimal binary decision trees is NP-complete." In: *Information processing letters* 5.1 (1976), pp. 15–17.

[53] Y. LeCun and C. Cortes. "MNIST handwritten digit database." In: (2010). URL: http://yann.lecun.com/exdb/mnist/.

[54] A. M. Lesk. *Introduction to protein architecture: the structural biology of proteins*. Oxford University Press Oxford, 2001.

[55] C. Levinthal, S. J. Wodak, P. Kahn, and A. K. Dadivanian. "Hemoglobin interaction in sickle cell fibers. I: Theoretical approaches to the molecular contacts." In: *Proceedings of the National Academy of Sciences* 72.4 (1975), pp. 1330–1334.

[56] S. Liu, Y. Gao, and I. A. Vakser. "Dockground protein–protein docking decoy set." In: *Bioinformatics* 24.22 (2008), pp. 2634–2635.

[57] S. Lyskov and J. J. Gray. "The RosettaDock server for local protein–protein docking." In: *Nucleic acids research* 36.suppl_2 (2008), W233–W238.

[58] H. Madaoui and R. Guerois. "Coevolution at protein complex interfaces can be detected by the complementarity trace with important impact for predictive docking." In: *Proceedings of the National Academy of Sciences* 105.22 (2008), pp. 7708–7713.

[59] A. Malhotra, E. Younesi, S. Sahadevan, J. Zimmermann, and M. Hofmann-Apitius. "Exploring novel mechanistic insights in Alzheimer's disease by assessing reliability of protein interactions." In: *Scientific reports* 5 (2015), p. 13634.

[60] D. Malkin, F. P. Li, L. C. Strong, J. F. Fraumeni, C. E. Nelson, D. H. Kim, J. Kassel, M. A. Gryka, F. Z. Bischoff, M. A. Tainsky, et al. "Germ line p53 mutations in a familial syndrome of breast cancer, sarcomas, and other neoplasms." In: *Science* 250.4985 (1990), pp. 1233–1238.

[61] J. G. Mandell, V. A. Roberts, M. E. Pique, V. Kotlovyi, J. C. Mitchell, E. Nelson, I. Tsigelny, and L. F. Ten Eyck. "Protein docking using continuum electrostatics and geometric fit." In: *Protein engineering* 14.2 (2001), pp. 105–113.

[62] W. McKinney et al. "Data structures for statistical computing in python." In: *Proceedings of the 9th Python in Science Conference*. Vol. 445. Austin, TX. 2010, pp. 51–56.

[63] J. Mintseris, K. Wiehe, B. Pierce, R. Anderson, R. Chen, J. Janin, and Z. Weng. "Protein–protein docking benchmark 2.0: an update." In: *Proteins: Structure, Function, and Bioinformatics* 60.2 (2005), pp. 214–216.

[64] T. M. Mitchell and M. Learning. "Mcgraw-hill science." In: *Engineering/Math* 1 (1997), p. 27.

[65] I. H. Moal, M. Torchala, P. A. Bates, and J. Fernández-Recio. "The scoring of poses in protein-protein docking: current capabilities and future directions." In: *BMC bioinformatics* 14.1 (2013), p. 286.

[66] K. P. Murphy. *Machine learning: a probabilistic perspective*. MIT press, 2012.

[67] H. Nguyen, D. A. Case, and A. S. Rose. "NGLview–interactive molecular graphics for Jupyter notebooks." In: *Bioinformatics* 34.7 (2017), pp. 1241–1242.

[68] N. O'Toole and I. A. Vakser. "Large-scale characteristics of the energy landscape in protein–protein interactions." In: *Proteins: Structure, Function, and Bioinformatics* 71.1 (2008), pp. 144–152.

[69] E. Ozcesmeci. *LHC: pushing computing to the limits*. 2019. URL: https://home.cern/news/news/computing/lhc-pushing-computing-limits (visited on 03/14/2019).

[70] *PDB Statistics: Protein-only Structures Released Per Year*. 2019. URL: https://www.rcsb.org/stats/growth/protein (visited on 02/2019).

[71] D. A. Pearlman, D. A. Case, J. W. Caldwell, W. S. Ross, T. E. Cheatham III, S. DeBolt, D. Ferguson, G. Seibel, and P. Kollman. "AMBER, a package of computer programs for applying molecular mechanics, normal mode analysis, molecular dynamics and free energy calculations to simulate the structural and energetic properties of molecules." In: *Computer Physics Communications* 91.1-3 (1995), pp. 1–41.

[72] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, et al. "Scikit-learn: Machine learning in Python." In: *Journal of machine learning research* 12.Oct (2011), pp. 2825–2830.

[73] B. Pierce and Z. Weng. "ZRANK: reranking protein docking predictions with an optimized energy function." In: *Proteins: Structure, Function, and Bioinformatics* 67.4 (2007), pp. 1078–1086.

[74] L. E. Raileanu and K. Stoffel. "Theoretical comparison between the gini index and information gain criteria." In: *Annals of Mathematics and Artificial Intelligence* 41.1 (2004), pp. 77–93.

[75] V. Romanuke. *Parallel Computing Center (Khmelnitskiy, Ukraine) represents an ensemble of 5 convolutional neural networks which performs on MNIST at 0.21 percent error rate*. 2016.

[76] T. Saito and M. Rehmsmeier. "The precision-recall plot is more informative than the ROC plot when evaluating binary classifiers on imbalanced datasets." In: *PloS one* 10.3 (2015), e0118432.

[77] S. Srivastava, Z. Zou, K. Pirollo, W. Blattner, and E. H. Chang. "Germ-line transmission of a mutated p53 gene in a cancer-prone family with Li–Fraumeni syndrome." In: *Nature* 348.6303 (1990), p. 747.

[78] M. J. Sternberg, H. A. Gabb, and R. M. Jackson. “Predictive docking of protein—protein and protein—DNA complexes.” In: *Current Opinion in Structural Biology* 8.2 (1998), pp. 250–256.

[79] A. Tovchigrechko and I. A. Vakser. “GRAMM-X public web server for protein–protein docking.” In: *Nucleic acids research* 34.suppl_2 (2006), W310–W314.

[80] P. Vagata and K. Wilfong. *Scaling the Facebook data warehouse to 300 PB*. 2014. URL: https://code.fb.com/core-data/scaling-the-facebook-data-warehouse-to-300-pb/ (visited on 03/14/2019).

[81] S. Van Der Walt, S. C. Colbert, and G. Varoquaux. “The NumPy array: a structure for efficient numerical computation.” In: *Computing in Science & Engineering* 13.2 (2011), p. 22.

[82] C. Wang, O. Schueler-Furman, and D. Baker. “Improved side-chain modeling for protein–protein docking.” In: *Protein Science* 14.5 (2005), pp. 1328–1339.

[83] M. Waskom, O. Botvinnik, D. O’Kane, P. Hobson, J. Ostblom, S. Lukauskas, D. C. Gemperline, T. Augspurger, Y. Halchenko, J. B. Cole, J. Warmenhoven, J. de Ruiter, C. Pye, S. Hoyer, J. Vanderplas, S. Villalba, G. Kunter, E. Quintero, P. Bachant, M. Martin, K. Meyer, A. Miles, Y. Ram, T. Brunner, T. Yarkoni, M. L. Williams, C. Evans, C. Fitzgerald, Brian, and A. Qalieh. *mwaskom/seaborn: v0.9.0 (July 2018)*. July 2018. DOI: 10.5281/zenodo.1313201. URL: https://doi.org/10.5281/zenodo.1313201.

[84] D. H. Wolpert, W. G. Macready, et al. “No free lunch theorems for optimization.” In: *IEEE transactions on evolutionary computation* 1.1 (1997), pp. 67–82.

[85] Y. Yang and Y. Zhou. “Specific interactions for ab initio folding of protein terminal regions with secondary structures.” In: *Proteins: Structure, Function, and Bioinformatics* 72.2 (2008), pp. 793–803.

[86] C. Zhang, S. Liu, and Y. Zhou. “Accurate and efficient loop selections by the DFIRE-based all-atom statistical potential.” In: *Protein science* 13.2 (2004), pp. 391–399.

[87] Q. C. Zhang, L. Deng, M. Fisher, J. Guan, B. Honig, and D. Petrey. “PredUs: a web server for predicting protein interfaces using structural neighbors.” In: *Nucleic acids research* 39.suppl_2 (2011), W283–W287.

[88] Q. Zhang, M. Sanner, and A. J. Olson. “Shape complementarity of protein–protein complexes at multiple resolutions.” In: *Proteins: Structure, Function, and Bioinformatics* 75.2 (2009), pp. 453–467.

[89] Y. Zhao and G. Karypis. “Data clustering in life sciences.” In: *Molecular biotechnology* 31.1 (2005), pp. 55–80.