Machine Learning for Enzyme Promiscuity

Parisa Mapar

School of Electrical Engineering

Thesis submitted for examination for the degree of Master of Science in Technology.

Espoo 23.5.2018

Thesis supervisor:

Prof. Juho Rousu

Thesis advisors:

Ph.D. Markus Heinonen

Ph.D. Sandor Szedmak

Aalto University
School of Electrical
Engineering

AALTO UNIVERSITY
SCHOOL OF ELECTRICAL ENGINEERING

ABSTRACT OF THE
MASTER'S THESIS

Author: Parisa Mapar

Title: Machine Learning for Enzyme Promiscuity

Date: 23.5.2018 Language: English Number of pages: 6+71

Department of Computer Science

Professorship: Computer Science

Supervisor: Prof. Juho Rousu

Advisors: Ph.D. Markus Heinonen, Ph.D. Sandor Szedmak

With the discovery of an increasing number of catalytically promiscuous enzymes, which are capable of catalyzing multiple reactions, the traditional view of enzymes as highly specific proteins has been brought into question. The significant implications of protein promiscuity for the theory of enzyme evolution suggest that this inherent feature can be utilized as the seed for engineering new functions in biotechnology and synthetic biology as well as in drug design. Therefore, understanding protein promiscuity is becoming even more important as it provides new insights into the evolutionary process that has led to such vast functional diversity. While there have been numerous efforts devoted to recognizing the determinants of promiscuity, till date, this pertinent question regarding the distinctions between specialized enzymes and promiscuous enzymes has remained unanswered.

As an *in silico* approach, in this thesis, we attempt to find a predictive model which can accurately classify unseen proteins into catalytically promiscuous and non-promiscuous. To this end, we exploit different representations and properties of proteins, and adopt different computational approaches accordingly. The role of proteins sequences as indicators of promiscuity is investigated by means of the BLAST algorithm as well as string kernels. Additionally, to validate the interplay between proteins' three-dimensional structures and their promiscuous behaviors, we employ a novel method which is modeling the topological details of proteins as graphs. Graph kernel functions are then applied to measure the structural similarities between the 3D structures of proteins. The classification is performed using SVM as a kernel-based method. The results indicate that proteins' sequences have limited bearings on promiscuity. Conversely, proteins' 3D structures can reliably predict whether a protein has promiscuous activities with an accuracy of 96%. Our best results are achieved using the Weisfeiler-Lehman subtree graph kernel and the secondary structure information of proteins.

Keywords: enzyme promiscuity, proteins, machine learning, classification, BLAST, kernel methods, SVM, string kernels, graph kernels

Preface

Foremost, I would like to express my profound gratitude to my supervisor, Prof. Juho Rousu, for offering me the opportunity to conduct this research under his supervision at the Department of Computer Science, for his patience, endless support, and immense knowledge. My sincere thanks also goes to my advisors, Dr. Markus Heinonen and Dr. Sandor Szedmak, for steering me in the right direction with their insightful and valuable comments.

I would also like to thank the current members and alumni of Prof. Rousu's research group for all their help and friendship, and creating a vibrant environment. My thanks also goes to my friend, Clemens Westrup, for his constructive comments on my thesis and supporting me throughout the process of writing.

Also, I thank all my friends for their unwavering support and in particular, I want to give special thanks to Nanna Koivula for her true friendship and unfailing support. My parents deserve a particular note of thanks as I am forever indebted to them for their unconditional love and continuous encouragement throughout my years of study. Lastly, I would like to express my deepest thanks to my sister, Farimah Mapar, who has been not only my best friend, but also the best guide and inspiration in every step of my life.

Helsinki, 23.5.2018

Parisa Mapar

Contents

Symbols and abbreviations

Symbols

Å Ångstrom

Abbreviations

3D	Three-Dimensional
AA	Amino Acid
BLAST	Basic Alignment Search Tool
BLOSUM	BLOck SUbstitution Matrix
Cα	Alpha-carbon
CCA	Canonical Correlation Analysis
CV	Cross-Validation
DSSP	dictionary of protein secondary structures
EC number	Enzyme Commission number
FN	False Negative
FP	False Positive
GP	Gaussian Processes
GS kernel	Generic String kernel
GWSK	Gap-Weighted Subsequence Kernel
IBL	Instance-Based Learning
MDS	MultiDimensional Scaling
OHE	One Hot Encoding
PCA	Principal Component Analysis
PDB	Protein Data Bank
SSE	Secondary Structure Elements
SVM	Support Vector Machine
TN	True Negative
TP	True Positive
WASK	Weighted All-Substrings Kernel
WL	Weisfeiler-Lehman

1 Introduction

The genetic information of a cell is carried by its cellular DNA which consists of thousands of genes. Each gene is a segment of the DNA molecule and contains instructions that, when decoded, serve as a recipe on how to build unique, large and complex molecules, called proteins. Therefore, the function of each cell is determined by its decoded genes, which in turn, lead to a collection of formed proteins that perform specialized functions depending on their unique compositions. Proteins are assembled with different amino acids joined together to form long chains, which are then folded into a variety of three-dimensional structures held together by different bonds. The folded shape, or conformation of a protein is dictated directly by its linear sequence of amino acids. As workhorses of the cell, protein macromolecules have an enormous array of indispensable functions within organisms, ranging from providing structure and support for cells, DNA replication, and acting as antibodies to transporting molecules from one location to another. However, the best-known role of proteins in the cell is catalyzing biochemical reactions. These type of proteins, which are called enzymes, act as catalysts by accelerating the chemical reactions inside living cells without being consumed or permanently altered.

In a chemical reaction, one or more chemical substances, known as reagents, reactants, or substrates, are transformed into other types of substances, called products. Enzymes facilitate the occurrence of almost all metabolic processes in the cell, which are impossible under ordinary conditions. These substrate molecules are acted upon and subsequently converted into products by binding to a region on the surface of enzymes, called the active site [Alberts et al., 2002].

The classical definition of enzymes refers to them as remarkably specific catalysts. It suggests that enzymes are usually highly specific as to what substrate they bind to as well as the chemical reaction they catalyze, meaning that they selectivity accommodate certain substrates, referred to as substrate specificity and catalyze specific reactions, known as reaction specificity. These specificities are achieved by binding sites which have complementary shapes and physicochemical characteristics to the substrates. While the notion of "one enzyme—one substrate—one reaction", which attributes enzyme catalysis to precise optimization of an enzyme for one substrate and reaction, is dominant, there has been a growing appreciation in recent years that this picture is oversimplified [Copley, 2003; Khersonsky and Tawfik, 2010]. Many, if not most, enzymes exhibit capabilities of transforming multiple substrates or catalyzing various reactions, in addition to the ones for which they evolved. An increasing number of enzymes have been reported to enjoy such inherent property referred to as 'promiscuity'. According to previous reviews, promiscuity can be described based on a wide range of fundamentally different phenomena. Some studies [Patrick et al., 2007; Khersonsky and Tawfik, 2010] define promiscuity as the ability of an enzyme to catalyze secondary adventitious reactions that are not part of the organism's physiology, which is referred to as 'catalytic promiscuity' [O'Brien and Herschlag, 1999]. Hult and Berglund, on the other hand, classify enzymatic promiscuity into three major types that can be combined: substrate promiscuity, which is shown by enzymes with relaxed or broad substrate specificity, catalytic

promiscuity, and condition promiscuity which applies to the enzymes with catalytic activity in a variety of temperatures, pH, etc. [Hult and Berglund, 2007] (see Figure 1). Pyruvate decarboxylase [Hult and Berglund, 2007], carbonic anhydrase [Pocker and Stone, 1967], pepsin [Reid and Fahrney, 1967], chymotrypsin [Nakagawa and Bender, 1969], and L-asparaginase [Jackson and Handschumacher, 1970] are among early examples of catalytically promiscuous enzymes [Khersonsky and Tawfik, 2010]. Promiscuity, however, is distinguished from moonlighting [Jeffery, 1999], which is the utilization of protein parts outside its active site scaffold for additional functions that are mostly regulatory and structural [Copley, 2003].

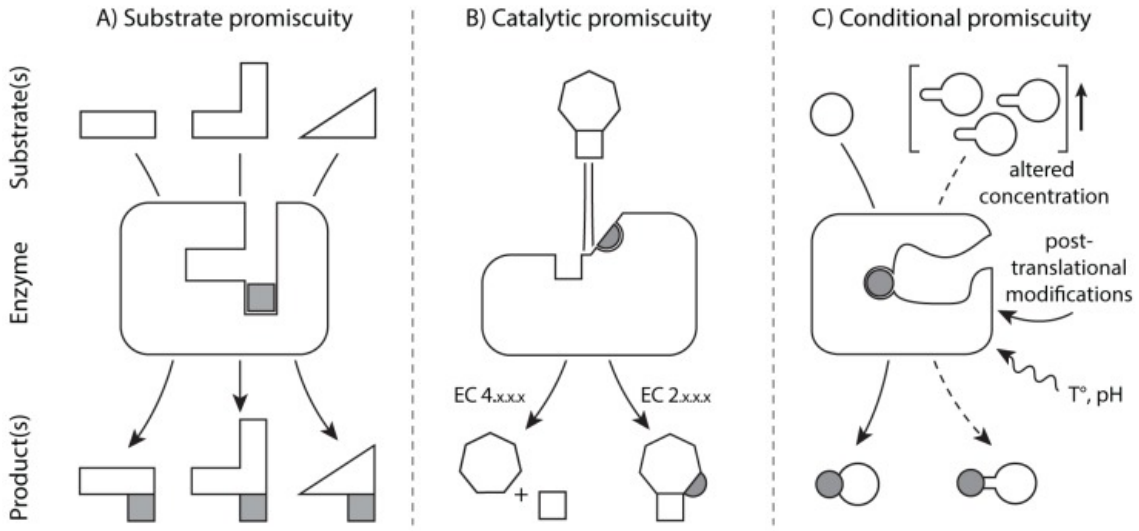

Figure 1: Mechanistic classification of enzyme promiscuity into three types: (A) Substrate promiscuity or multispecificity; (B) Catalytic promiscuity; (C) Conditional promiscuity [Piedrafita et al., 2015].

During the past two decades, enzyme promiscuity has received considerable attention for its practical applications and has begun to be recognized as a valuable source of information in enzyme evolution [O'Brien and Herschlag, 1999; Copley, 2003]. As early as 1976, Jensen proposed that promiscuity shaped the evolution of protein evolution by suggesting that primitive ancient enzymes which presumably had minimal gene content possessed very broad specificities in contrast to modern enzymes that tend to specialize in one substrate and reaction [Jensen, 1976]. He conceptualized the evolvability of proteins as a process whereby the catalytic versatility enabled a relatively limited arsenal of rudimentary enzymes to afford a wider range of functions that were necessary to maintain ancestral organisms [Khersonsky et al., 2006; Khersonsky and Tawfik, 2010]. Concurrently, in 1977, Jacob postulated in his classical note "Evolution and Tinkering" [Jacob, 1977] that new functions were not produced from scratch, but rather from preexisting suboptimal functions [Khersonsky and Tawfik, 2010]. It is widely accepted that new protein structures and functions were generated from their ancestors whose genes were modified, or 'tinkered with'

[Khersonsky et al., 2006]. This view, which is supported by many elegant studies, suggests that divergence of enzymes started with ancestral promiscuous enzymes that were changed due to the environmental selection pressure, gene duplication, and mutation resulting in higher metabolic efficiencies [Khersonsky and Tawfik, 2010; Baier et al., 2016]. This process has led to the creation of enzyme families and superfamilies [Gerlt and Babbitt, 2001] whose members share similar scaffold and active site architectures (see Figure 2). For instance, organophosphate hydrolase and atrazine chlorohydrolase are xenobiotic degrading enzymes which evolved from precursor promiscuous enzymes that possessed those functions as their latent activities [Seffernick et al., 2001; Afriat-Jurnou et al., 2012; Baier et al., 2016]. Conservation of structural and catalytic features in the α/β-hydrolase folds in the enolase superfamily is another example suggesting that within each superfamily, enzymes with different substrate and reaction specificities arose from a common ancestor via divergent evolution [Babbitt and Gerlt, 1997; Gerlt and Babbitt, 1998; O'Brien and Herschlag, 1999].

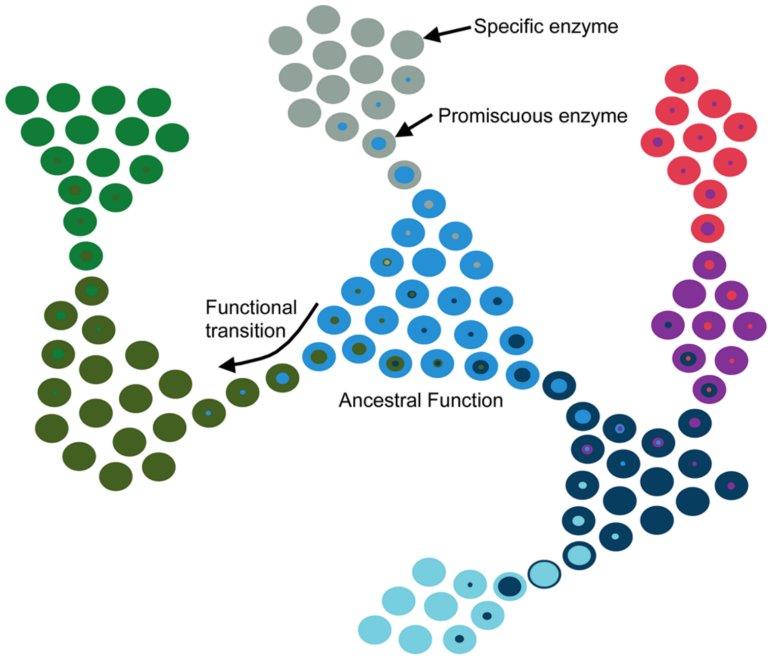

Figure 2: Schematic representation of enzyme evolution within a theoretical enzyme superfamily. Enzymes and their native physiological functions are represented by circles and colors, respectively [Baier et al., 2016].

The importance of promiscuity from an enzymological point of view and its potential mechanistic and evolutionary implications were first highlighted by O'Brien and Herschlag [O'Brien and Herschlag, 1999] and later by Copley [Copley, 2003]. According to them, protein promiscuity is an advantageous feature which can be leveraged to obtain starting points for the evolution of novel enzyme activities. This implies that existing catalysts can be improved by exploiting enzyme promiscuity which ultimately leads to novel metabolic pathways that are currently unavailable

[Hult and Berglund, 2007]. Therefore, relaxed substrate and reaction specificity is a new frontier for biocatalysis and direct evolution. Today, protein promiscuity has fueled much of the growth in biotechnology and synthetic biology by facilitating a better understanding of the evolution of new functions. Indeed, protein engineers have succeeded in tailoring innovative proteins by mimicking this natural process in the laboratory [Kazlauskas, 2005; Peisajovich and Tawfik, 2007; Turner, 2009]. Besides protein engineering, enhanced understanding of promiscuity finds application in drug design for both biomedical or industrial applications [Nobeli et al., 2009]. However, it is worth mentioning that some studies consider promiscuity also as a hurdle jeopardizing the performance of synthetic systems with unwanted side effects [Nobeli et al., 2009].

1.1 Problem Statement

The potential applications of enzyme promiscuity have motivated the search of viable starting places with measurable activities for the development and engineering of novel biocatalysts. However, predicting and rationalizing the existence of promiscuous activities from a given sequence, structure, and native function are challenging tasks with daunting complexity [Babtie et al., 2010]. While some enzyme families have been extensively characterized, our current knowledge of potential factors that influence promiscuity is still limited and biased towards these few examples [Babtie et al., 2010]. For instance, some studies associate promiscuity with conserved regions in the active site [Anandarajah et al., 2000]. Nevertheless, it is not yet resolved to what extent we can generalize these observations to other enzymes. Therefore, in order to uncover determinants that can help us predict the promiscuity, we need to test a large repertoire of enzymes for potential characteristics. Quantitative, mechanistic or structural characterization of promiscuous enzymes can aid in revealing undeveloped or unrealized promiscuous activities in enzymes that currently appear to be highly specific. Furthermore, understanding the mechanistic and structural basis for catalytic promiscuity can provide opportunities for annotating previously uncharacterized enzymes.

1.2 Related Work

In spite of the intense efforts being devoted, till date there is no clear pattern linked to promiscuity which can be generalized to all enzyme families and superfamilies. In this context, various computational tools have attempted to predict and quantify promiscuity. In one study, a quantitative index for assessing the degree of substrate promiscuity based on the catalytic efficiencies has been proposed [Nath and Atkins, 2008]. This index, which computes the degree of variability between different substrates, lacks scalability as it targets only a predefined set of substrates. Moreover, assuming the same chemical transformation for all substrates makes this approach applicable only to substrate promiscuity. On a more general level, another study has aimed to predict both catalytic and substrate promiscuity using a sequence-based kernel with protein sequences as the input [Carbonell and Faulon, 2010]. For an en-

zyme with known catalytic activity, this method also tries to investigate promiscuous activities by evaluating similarities between its reactions via graph-based representations of them. While this is an effective method, it does not take into account the protein structures which have proved to carry significant information regarding their biological functions and evolution. Chakraborty and Rao, on the other hand, exploit the 3D structures of proteins to compute their relative promiscuity via an index derived from the spatial and electrostatic properties of the catalytic residues, modeled as signatures [Chakraborty and Rao, 2012]. For a protein with known active site residues and 3D structure, this index ranks its promiscuity based on the number and quality of different active site signatures that have congruent matches in the vicinity of its native catalytic site, as well as differences in enzymatic activities. However, investigation of the role of residues beyond the active site in promiscuity might add new dimensions to our understanding of the interplay between protein structure and its function. Another attempt at leveraging the 3D structures of proteins to predict promiscuity has been discussed in [Steinkellner et al., 2014], where they mine structural databases using active site constellations to identify promiscuous ene-reductase activity. They have shown that typical Old Yellow Enzyme substrates and ligands have equivalent binding modes in their high-resolution crystal structures despite completely different amino acid sequences, overall structures and protein folds. A genome-wide method which aims to predict promiscuous functions of genes in a systematic and unsupervised manner has been proposed as well in a recent study [Oberhardt et al., 2016]. This approach utilizes an unsupervised PSI-BLAST based method to predict promiscuous 'replacer' functions that may compensate for primary 'target' functions of genes in *E. coli* if they are altered or lost.

1.3 Research Scope and Objectives

Despite all the previous *in silico* analyses which have endeavored to pinpoint determinants of promiscuity, it remains unclear whether there are general and fundamental structural and mechanistic differences between promiscuous proteins and those which are highly specialized. In this thesis, the objective is to establish a novel computational framework to systematically predict catalytic promiscuity of enzymes by exploiting their sequences as well as their 3D structures. To this end, we aim to design a predictive model which can successfully classify proteins into promiscuous and non-promiscuous. First, targeting a diverse set of enzyme sequences, by means of the BLAST algorithm [Altschul et al., 1990; Altschul et al., 1997] and string kernels, we probe whether sequences have any direct bearings on catalytic promiscuity. Secondly, this thesis fills a gap by investigating the role of protein structure as a whole in promiscuity. In order to unravel whether there are any generic topological structures promoting promiscuity, we attempt, for the first time, to model protein structures as graphs to be later compared via graph kernels. Furthermore, we enrich our model by incorporating a variety of physicochemical, primary and secondary structure properties of the enzymes into their graph representations. We then, employ Support Vector Machines [Cortes and Vapnik, 1995] to evaluate the performance of our string and graph kernels.

In order to facilitate a better comprehension of the succeeding chapters, we provide detailed descriptions of proteins and their structures in the following.

1.4 Protein Structure

Proteins are macromolecules which consist of one or more long chains of amino acid residues called polypeptides. Polypeptides are built from series of up to 20 different amino acids, each of which having a unique side chain with different substituents and physicochemical properties. Proteins have four different levels of structure, namely primary, secondary, tertiary, and quaternary.

- *Primary structure*: the primary structure of a protein is the sequence of amino acids in a polypeptide chain, which ultimately confers a unique three-dimensional structure to the protein.

- *Secondary structure*: secondary structure refers to local folded structures that form within a polypeptide chain due to interactions between atoms of the chain. The α-helix and β-pleated sheet are the most common elements of the secondary structure.

- *Tertiary structure*: The overall three-dimensional shape of a single polypeptide chain, defined by the atomic coordinates, constitutes the tertiary structure, which is an ensemble of formations and folds. The tertiary structure is stabilized by bonding interactions between the side-chain groups of the amino acids.

- *Quaternary structure*: while all proteins contain primary, secondary and tertiary structures, quaternary structures are reserved for proteins which have two or more polypeptide chains, also known as subunits in this context. In proteins with quaternary structures, each polypeptide folds separately and adopts a tertiary structure. Tertiary structures then assemble with each other via intermolecular interactions to form the quaternary structure, also called an oligomer. Depending on the number of subunits, proteins adopt different names. Dimers, trimers, and tetramers are, for instance, proteins composed of two, three and four subunits, respectively. In this sense, a homo-oligomer would be formed by few identical subunits and by contrast, a hetero-oligomer would be made of more than one, different, subunits (see Figure 3).

As enzymes are a special type of proteins, throughout this thesis, we use ‘protein’ and ‘enzyme’ interchangeably. The remainder of the thesis is organized as follows. In Chapter 2, we describe the data and demonstrate how we have preprocessed and curated it to generate our datasets. Chapter 3 lays out the adopted computational approaches that we have investigated to find a fitting model. Chapter 4 deals with a series of experimental setting concerning the employed computational methods. In Chapter 5, we report the results, and discuss our findings in Chapter 6.

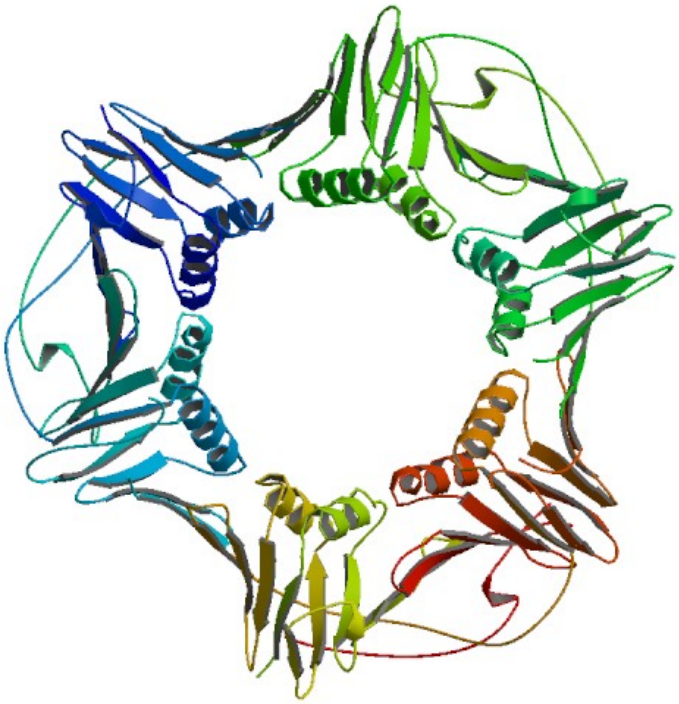

Figure 3: An example of a protein quaternary structure consisting of 3 polypeptide chains. PDB ID: 1AXC (www.rcsb.org) [Berman et al., 2000; Gulbis et al., 1996].

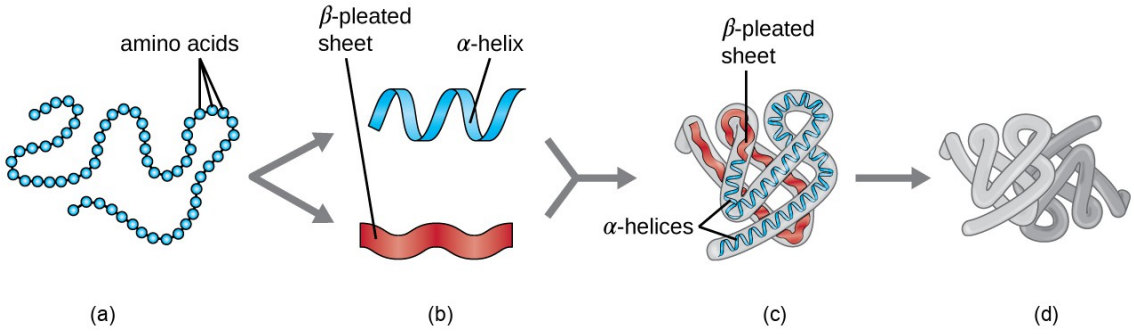

Figure 4: An illustration of the 4 level of protein structures. (a) Primary structure: sequence of a chain of amino acids. (b) Secondary structure: local folding of the polypeptide chain into helices or sheets. (c) Tertiary structure: 3D folding pattern of a protein due to side chain interactions. (d) Quaternary structure: protein consisting of more than one amino acid chain. [OpenStax CNX,]

2 Data

The initial data was extracted from the UniProt Knowledgebase (UniProtKB) database [The UniProt Consortium, 2017], a protein database partially curated by experts, by querying a list of all entries with at least one cross-reference to the Protein Data Bank (PDB) database [Berman et al., 2000]. The retrieved list consists of 43,394 unique UniProt IDs, containing both reviewed (manually annotated) and unreviewed (automatically annotated) entries. Thus, the initial dataset was gathered using the information provided for the entries (UniProt IDs) such as proteins' amino acid sequences, catalytic activities (the chemical reactions proteins catalyze) and pointers to PDB. The dataset was then filtered by selecting those proteins with annotated catalytic activities, or in other terms, Enzyme Commission (EC) numbers [Bairoch, 2000], totaling 14,100 proteins.

2.1 Promiscuity

In order to define promiscuity in a more concrete manner, we followed Khersonsky and Tawfik's approach, which assesses the degree of promiscuity by comparing differences in the EC numbers [Khersonsky and Tawfik, 2010]. The Enzyme Commission (EC) number systematically classifies and names the enzymes based on the chemical names of the substances they modify (substrates) and the chemical reactions catalyzed by them. Each EC code consists of four numbers separated by periods representing a progressively finer classification of the enzyme as we move from the leftmost number towards the rightmost one. According to Khersonsky and Tawfik, in case of catalytic promiscuity, a protein is annotated with at least two EC codes which differ in the third, second, or even first numbers (which indicates a different reaction category) ignoring the fourth number, whereas multispecificity, or substrate promiscuity, is concerned with differences only in the fourth number.

Focusing solely on catalytic promiscuity and considering Khersonsky and Tawfik's approach, we split the list into promiscuous (positive) and non-promiscuous (negative) sets containing 688 and 13,412 proteins, respectively.

2.2 Data Preprocessing

Each UniProt ID might have more than one cross-reference to PDB meaning that a number of different structures might be available for each protein. These structures differ for a number of reasons, one of which is having different resolutions. Another reason is that some structures apply to apoproteins (proteins that require a cofactor but do not have one bound) while some others apply to proteins complexed with various ligands. Moreover, different structures can belong to various mutants of the same protein. Similar to UniProt entries, PDB structures can point to different identifiers and databases as well. This information can be found in the DBREF record of PDB files.

Here, we restricted our dataset to having only one cross-reference to PDB per each protein by introducing the following conditions:

1. For each protein, we considered only those PDB structures which have cross-references to its UniProt ID in the DBREF record of their PDB files.

2. We discarded all PDB structures whose resolution is greater than 5Å or lack a reported resolution.

3. In order to select the best PDB structure for each protein, we took into account the completeness of the structures which is examining the number of residues for which 3-dimensional coordinates are provided in the PDB files.

Applying these conditions, we excluded the proteins whose all PDB structures failed to meet the aforementioned conditions, leaving us with 590 and 11,294 promiscuous and non-promiscuous proteins, respectively.

Furthermore, redundancy was removed within each set by clustering the protein sequences at a threshold of 50% of sequence similarity using the CD-HIT program [Huang et al., 2010], and then selecting only one protein per each cluster. This similarity threshold corresponds to the alignment of 3-mers, which are peptides containing three amino-acid residues, called tripeptides. As an additional filter, redundancy in terms of uniqueness of cross-references to PDB has was as well. Also, in an attempt to eliminate the possible noise inflicted mainly by misannotations, overlaps between the promiscuous and non-promiscuous sets was removed by leaving out the proteins which have identical protein sequences, or are annotated with identical sets of EC numbers. This resulted in 406 promiscuous and 7,916 non-promiscuous proteins as the our base set.

2.3 Datasets

Based on the initial data extracted from UniProt along with the pointers to PDB, we formed two individual sets of datasets according to proteins' different representations (see Figure 4).

2.3.1 The Sequence Dataset

The sequence dataset contains proteins' linear amino acid sequences, or in other terms, their primary structures, coupled with their physicochemical properties. We study the sequences associated with the UniProt IDs in our base set along with their amino acids' descriptors, namely residue types, hydropathy indices and weights as well as side chain class, polarity, and charge (see Table 1). Moreover, BLOSUM50 and BLOSUM62 substitution matrices are examined as additional amino acid descriptors.

Similar to our base set, we have 406 promiscuous and 7,916 non-promiscuous sequences in our sequence dataset.

AA[1]	Side Chain Class	Side Chain Polarity	Side Chain Charge	Hydropathy Index	Weight
Ala	aliphatic	nonpolar	neutral	1.8	89.0940
Arg	basic	basic polar	positive	-4.5	174.2030
Asn	amide	polar	neutral	-3.5	132.1190
Asp	acid	acidic polar	negative	-3.5	133.1040
Cys	sulfur-containing	nonpolar	neutral	2.5	121.1540
Glu	acid	acidic polar	negative	-3.5	147.1310
Gln	amide	polar	neutral	-3.5	146.1460
Gly	aliphatic	nonpolar	neutral	-0.4	75.0670
His	basic aromatic	basic polar	neutral	-3.2	155.1560
Ile	aliphatic	nonpolar	neutral	4.5	131.1750
Leu	aliphatic	nonpolar	neutral	3.8	131.1750
Lys	basic	basic polar	positive	-3.9	146.1890
Met	sulfur-containing	nonpolar	neutral	1.9	149.2080
Phe	aromatic	nonpolar	neutral	2.8	165.1920
Pro	cyclic	nonpolar	neutral	-1.6	115.1320
Ser	hydroxyl-containing	polar	neutral	-0.8	105.0930
Thr	hydroxyl-containing	polar	neutral	-0.7	119.1190
Trp	aromatic	nonpolar	neutral	-0.9	204.2280
Tyr	aromatic	polar	neutral	-1.3	181.1910
Val	aliphatic	nonpolar	neutral	4.2	117.1480

Table 1: Physicochemical properties of amino acids

[1]AA stands for Amino Acid type

2.3.2 The 3D Structure Datasets

The 3D structure datasets was gathered using the cross-references to PDB. PDB files often contain information on proteins' quaternary structures in case of oligomers, or tertiary structures in case of proteins with only one single polypeptide chain. However, sometimes coordinates of only a portion of a protein macromolecule is available in its PDB file. We have constructed 3 different datasets using the 3D coordinates in the PDB files.

- *The UniProt-unique-chains dataset*: as previously mentioned, a protein can be an oligomer containing more than one chain. However, in the DBREF record of PDB files, not all chains necessarily point to the same ID or even database. For a specific protein with a unique UniProt ID, this dataset takes into account only those chains corresponding to the same UniProt ID. Furthermore, it includes the coordinates of only one copy of each distinctive chain if several identical chains are available.

- *The UniProt-all-chains dataset*: for each protein, this dataset comprises the coordinates of all the chains associated with its UniProt ID.

- *The all-coordinates dataset*: the coordinates of all the chains available in the PDB file irrespective of their cross-referenced IDs or databases are included in this dataset.

In order to avoid computational complications, in all the 3D structure datasets, the coordinates of each are represented by the coordinates of its alpha-carbon (Cα) atoms. If the α-carbon coordinates are missing, the coordinates of either the amine group's nitrogen or the carboxyl group's carbon atoms are used.

Residue's secondary structure attributes, which are α-helix and β sheet, and primary structure descriptors, namely residue type, side chain class, polarity, and charge are used as data features.

Similar to our base set, we have 406 promiscuous and 7,916 non-promiscuous proteins in each of the 3D structure datasets. In the following chapter, we will discuss the methods we have employed to analyze these datasets.

3 Methods

We treat the task of predicting the promiscuity status of a protein as a supervised binary classification problem where promiscuous and non-promiscuous proteins form the positive and negative classes, respectively. In a supervised binary classification, there are a set of training instances with known binary classes, and the objective is to find a predictive model which is capable of predicting the classes of unseen data accurately.

In this chapter, we will endeavour to illustrate the computational approaches that we have investigated to find a fitting predictive model. We begin the chapter with discussing a baseline approach. We will then continue with elaborating on kernel methods preceded by a brief description on the type of their learning algorithms. The kernel methods themselves are divided into two groups based on the type of the dataset, namely, the sequence-based or 3D structure-based.

3.1 BLAST

In order to have a reference point for predicting the enzyme promiscuity, we have attempted to classify the unseen data by means of the BLAST algorithm [Altschul et al., 1990; Altschul et al., 1997] as a baseline. BLAST stands for Basic Alignment Search Tool, which is arguably the most heavily used algorithm for annotating and comparing biological sequences such as nucleotides of DNA or proteins' amino acid sequences quickly and accurately. As there are many possible ways a sequence might align with sequences in a database, searching a large database of sequences can be exhaustive. Blast speeds up this process by conducting local alignments, that is, it looks for small regions of perfect match between the queried sequence and target sequences irrespective of where they are in the sequence. It then investigates the sequence that adjoins these short regions to see whether there is a longer stretch that matches perfectly.

A blast search starts with sets of three-letter words, also called 3-mers, which represent three nucleotides or amino acids in a specific order. It then looks for all common three-letter words between the query sequence and the hit sequences from the database and counts the number of times these words appear. Closely related words, called neighbors, which differ in only one or two letters are examined as well. These neighbors, however, should satisfy a similarity threshold of at least T according to a scoring matrix. Matches are then found by comparing the assembled words and their neighbors to the sequences in the database.

BLAST is a family of different programs which vary depending on the type of input, the database being searched, and what is being compared [Camacho et al., 2009]. Since we are interested in proteins, we use protein-protein BLAST (blastp) which is a program that for a protein query, returns the most similar protein sequences from a protein database specified by the user. In our classification problem, a protein sequence with unknown label is scanned against a local database formed by the training sequences. We then assign the class label of the hit sequence with the highest similarity score to the new protein.

3.2 Instance-Based Learning

In machine learning, instance-based learning (IBL), sometimes called memory-based learning, refers to a family of classifiers whose main distinctive characteristic is to use the instances themselves as class representations instead of constructing explicit generalizations and abstractions such as decision trees [Quinlan, 1986] or rules. In order to classify unseen instances, IBL algorithms compare new problem instances with instances seen in training, which have been stored in memory. The classification, thus, relies on the similarity between the new observation to be classified and the previously seen instances as the hypothesis is constructed directly from the training instances.

Two examples of instance-based learning classifiers are the k-nearest neighbor algorithm (k-NN) [Cover and Hart, 1967], which we will explain in greater detail later in this chapter (Section 3.3.1.3), and kernel methods [Shawe-Taylor and Cristianini, 2004; Vapnik, 1995]. When predicting a value/class for a new instance, these algorithms compute distances or similarities between this instance and the training instances to make an inference.

3.3 Kernel Methods

Kernel methods are a class of algorithms for pattern analysis, in which the general task is to find and study general types of relations in datasets. In many classification algorithms, the objective is to learn a nonlinear function or decision boundary. To this end, the raw data, which is often non-vectorial, needs to be explicitly transformed into feature vector representations via a user-specified feature map. Strings, graphs and trees are examples of non-vectorial, structured data. Kernel methods, however, offer an alternative which is a single similarity function over pairs of data points in their raw representation without the need for explicit computation of the coordinates in the feature space. This approach , called the "kernel trick", is often computationally cheaper than the explicit computation of the coordinates.

Kernel Trick

The explicit mapping that is needed to get linear learning algorithms to learn a nonlinear function or decision boundary can be avoided by employing the kernel trick. If $k\colon \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is a kernel function, for all $\mathbf{x}$ and $\mathbf{x}'$ in the input space $\mathcal{X}$, $k(\mathbf{x}, \mathbf{x}')$ can be expressed as an inner product in another space $\mathcal{V}$. In other terms, if $\varphi\colon \mathcal{X} \to \mathcal{V}$ is a feature map, then $k(\mathbf{x}, \mathbf{x}') = \langle \varphi(\mathbf{x}), \varphi(\mathbf{x}') \rangle_{\mathcal{V}}$. Hence, any linear model has the potential to be converted into a non-linear model by applying the kernel trick which replaces the features by a kernel function.

Kernel methods can be seen as instance-based learners in that instead of learning fixed parameters corresponding to the features of their inputs, they make predictions for the labels of the unseen data, which are not in the training, according to the kernel values computed between the unlabeled input $\mathbf{x}'$ and each of the training inputs $\mathbf{x}_i \in \mathcal{X}$. A kernel matrix for an input space $\mathcal{X}$ with n examples is obtained by evaluating the kernel function over all pairs of examples $(\mathbf{x}_i, \mathbf{x}_j), i, j = \{1, \ldots, n\}$.

In other words, the kernel matrix $\mathbf{K}_n \in \mathbb{R}^{n \times n}$ is a symmetric $n \times n$ square matrix with entries

$$[\mathbf{K}_n]_{ij} = k(\mathbf{x}_i, \mathbf{x}_j). \tag{1}$$

There are numerous algorithms which are capable of operating with kernels such as kernel perceptron, support vector machine (SVM) [Cortes and Vapnik, 1995], Gaussian processes (GP) [Williams and Rasmussen, 1996; Neal, 1996], principal components analysis (PCA), canonical correlation analysis (CCA), and ridge regression, to name a few. In this study, as the binary classification tool, we employ SVM, which is an effective model used for the same purpose.

Support Vector Machine

In a binary classification, the key idea of the Support Vector Machine is to construct a hyperplane which could successfully split the high-dimensional data points which are linearly separable into two classes. As there might be a large set of different hyperplanes separating the two classes, intuitively, a good separation is achieved by the hyperplane which has the maximum distance to the closest data points of either class. These data points are referred to as support vectors and the distance from the hyperplane to them is called margin (see Figure 5).

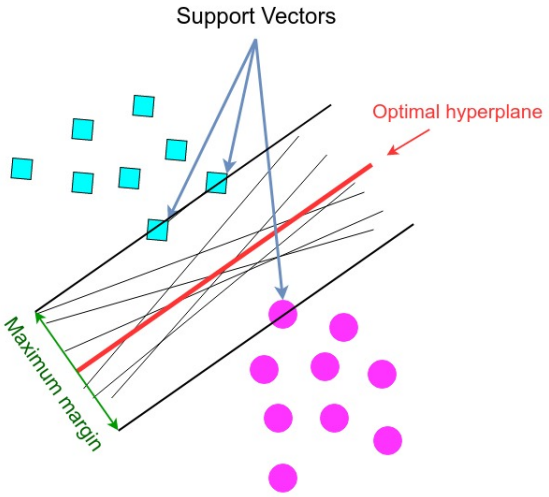

Figure 5: An intuition for the main idea behind Support Vector Machine.

For a specific decision hyperplane, the linear discriminant function can be formulated as $f(\mathbf{x}) = \mathbf{w}^T\mathbf{x}+b$, where $\mathbf{x}$ is a data point, $\mathbf{w}$ is the decision hyperplane normal vector which is perpendicular to the hyperplane, and b is an intercept term. The class assignment is then given by $y = sign[f(\mathbf{x})]$, where $y \in \{-1, 1\}$. Accordingly, the functional margin of a data point $\mathbf{x}_i$ with respect to the hyperplane is defined as the quantity $\gamma(\mathbf{x}) = y_i f(\mathbf{x})$, where y_i tells in which side of the hyperplane the data point lies.

SVM insists on a large margin around the decision boundary. However, one can increase the margin without limit by scaling up $\mathbf{w}$. In order to circumvent this problem, we can impose a scaling constraint which is fixing the margin to $\gamma = 1$ and seeking for the shortest weight vector which fulfills this margin. This leads to the following constrained optimization problem:

$$\begin{aligned} \min_{\mathbf{w},b} \quad & \frac{1}{2}\|\mathbf{w}\|^2 \\ \text{s.t.} \quad & y_i(\mathbf{w}^T\mathbf{x} + b) \geq 1. \end{aligned} \tag{2}$$

This is called the hard margin SVM where constraints ensure that there are no missclassified examples, which is possible since we assumed that the data is linearly separable. However, in practice, data is seldom linearly separable; and even if it is, a greater margin can be achievable by allowing the margin to make some mistakes. To allow errors, we modify the inequality constraints in Equation 2 and reformulate it as

$$\begin{aligned} \min_{\mathbf{w},b,\xi} \quad & \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_i \xi_i \\ \text{s.t.} \quad & y_i(\mathbf{w}^T\mathbf{x} + b) \geq 1 - \xi_i, \\ & \xi_i \geq 0, \end{aligned} \tag{3}$$

where $\xi_i \geq 0$ are slack variables that allow an example to have a smaller margin than $\gamma = 1$, and even to be misclassified, subject to a penalty depending on how far it is from meeting the margin. However, we would like to minimize the sum of the slacks in order to have as few misclassifcations as possible. $C > 0$ is a constant which controls the trade off between maximizing the margin and the amount of slack needed. The formulation is called the soft-margin SVM which was introduced by Cortes and Vapnik [Cortes and Vapnik, 1995]. Using the method of Lagrange multipliers, we can formulate the Equation 3 as a dual optimization problem expressed in terms of variables α_i:

$$\begin{aligned} \max_{\alpha_i} \quad & \sum_i \alpha_i - \frac{1}{2}\sum_i\sum_j \alpha_i\alpha_j y_i y_j \mathbf{x}_i^T\mathbf{x}_j \\ \text{s.t.} \quad & \sum_i y_i\alpha_i = 0, \\ & 0 \leq \alpha_i \leq C. \end{aligned} \tag{4}$$

The dual formulation, which gives the same solution to the soft-margin SVM, leads to an expansion of the weight vector in terms of the input examples:

$$\mathbf{w} = \sum_i y_i\alpha_i\mathbf{x}_i. \tag{5}$$

If the classes cannot be separated linearly, we need to transform the data via some transformation $\varphi\colon \mathbf{x} \to \varphi(\mathbf{x})$ on to a higher dimensional space where we can use a linear classifier. Following this mapping, Equation 6 can be given by

$$\mathbf{w} = \sum_i y_i\alpha_i\varphi(\mathbf{x}_i), \tag{6}$$

and consequently, substituting it in Equation 4 yields:

$$\begin{aligned} \max_{\alpha_i} \quad & \sum_i \alpha_i - \frac{1}{2} \sum_i \sum_j \alpha_i \alpha_j y_i y_j \varphi(\mathbf{x}_i)^T \varphi(\mathbf{x}_j) \\ \text{s.t.} \quad & \sum_i y_i \alpha_i = 0, \\ & 0 \leq \alpha_i \leq C. \end{aligned} \tag{7}$$

According to Section 3.3, by means of the kernel trick, we can replace the inner product $\varphi(\mathbf{x}_i)^T \varphi(\mathbf{x}_j)$ with a kernel function $k(\mathbf{x}_i, \mathbf{x}_j)$, which makes the computations considerably simpler. SVM is an example of a convex optimization problem which can be solved with the help of existing off-the-shelf efficient algorithms.

This will set the scene for the subsequent sections where we describe the kernels we have employed in this study based on the type of dataset.

3.3.1 Sequence-Based Kernels

Traditional approaches for determination of similarity between two protein sequences begin with strings of letters (amino acids) that represent the sequences. Therefore, in this study as for the proteins sequence dataset discussed in Section 2.3.1, we have adopted string kernels [Watkins, 1999; Haussler, 1999], a family of kernel functions designed for sequences. String kernels have applications in text mining and gene analysis where sequence data are to be clustered or classified.

String Kernels

The basic idea of string kernels is to compute the similarity between two strings, which are finite sequences of symbols and can be of different length, without explicitly extracting the features. SVM allows string kernels to work with strings, without having to translate these to fixed-length, real-valued feature vectors. The more subsequences with similar features two strings have in common, the more similar they are considered and thus, the higher value of the kernel function. There are numerous types of string kernels depending how the subsequences are defined. Subsequences can be contiguous or non-contiguous, they can have bounded or unbounded length, and gaps and mismatches can be taken into account subject to different ways of penalization. k-spectrum kernel [Leslie et al., 2002], gap-weighted subsequence kernel (GWSK) [Lodhi et al., 2002], weighted all-substrings kernel (WASK) [V. N. Vishwanathan and Smola, 2003], and generic string (GS) kernel [Giguère et al., 2013] are among examples of string kernels. In the following, after a brief clarification of some of the notations related to string kernels, we will elaborate upon the k-spectrum and generic string kernels. All the kernels described in this chapter are intended to be evaluated by SVM, however, we will further assess the generic string kernel by utilizing the k-NN method.

String Kernel Notations

$\Sigma = \{\sigma_1, \ldots, \sigma_n\}, n \in \mathbb{N}$ denotes a finite alphabet of n characters over which the strings are defined. The kleene star of Σ is denoted by Σ^* which is the set of all finite strings formed by characters in Σ. Therefore, a string over Σ can be written as a sequence $x = x_1 x_2 \ldots x_{|x|}$ which is a concatenation of characters from the alphabet: $x_i \in \Sigma$ for $i = 1, \ldots, |x|$, where $|x|$ denotes the length of string x. A substring of string x is a string $\hat{x} = x_{i+1} \ldots x_{i+l}$ where $0 \leq i$ and $i + l \leq |x|$. We denote all substrings of length k by Σ^{k}.

3.3.1.1 The k-Spectrum Kernel

Given two sequences, x and x', the k-spectrum kernel, presented in [Leslie et al., 2002], counts the number of substrings of length k in two sequences. The feature vector for each string x is $\phi^k(x)$ which is indexed by substrings of length k:

$$\phi^k_u(x) = |\{(v_1, v_2) : x = v_1 u v_2\}|, u \in \Sigma^{\mathrm{k}}, \tag{8}$$

where each feature $\phi^k_u(x)$ counts for occurrences of a substring.
Therefore, the kernel is defined as

$$K_k(x, x') = \langle \phi^k(x), \phi^k(x') \rangle = \sum_{u \in \Sigma^{\mathrm{k}}} \phi^k_u(x) \phi^k_u(x'). \tag{9}$$

3.3.1.2 The Generic String Kernel

In this study, special attention has been given to the generic string kernel presented in [Giguère et al., 2013], an effective kernel on biosequences. The elegance of the generic string kernel lies in the fact that it builds on several ideas:

- Bounded-length subsequence: it computes kernels over different-length subsequences up to a length L.
- Position dependency: discrepancies in substring positions are penalized by a Gaussian kernel on the starting indices of the compared substrings.
- Factorized representations: using an encoding function $\psi \colon \Sigma \to \mathbb{R}^d$, each symbol a in the alphabet is represented as a feature vector $\psi(a) = (\psi_1(a), \psi_2(a), \ldots, \psi_d(a))$, also called a descriptor, where each $\psi_i(a)$ encodes one of the d properties of a. Subsequently, the encoding function $\psi^l \colon \Sigma^l \to \mathbb{R}^{dl}$ generates the feature vector of a substring of length l as $\psi^l(a_1, a_2, \ldots, a_l) = (\psi(a_1), \psi(a_2), \ldots, \psi(a_l))$, which is a concatenation of the feature vectors of l symbols, each of d components.
- Soft matching: a Gaussian kernel is computed over the squared Euclidean distance between the feature vectors of two subsequences.

The kernel is formulated as

$$K_{\mathrm{GS}}\left(x, x', L, \sigma_p, \sigma_c\right) = \sum_{l=1}^{L} \sum_{i=0}^{|x|-l} \sum_{j=0}^{|x'|-l} e^{\left(\frac{-(i-j)^2}{2\sigma_p^2}\right)} e^{\left(\frac{-\|\psi^l(x_{i+1},..,x_{i+l})-\psi^l(x'_{j+1},..,x'_{j+l})\|^2}{2\sigma_c^2}\right)}, \quad (10)$$

where $L \leq$ is the maximum length for substring comparison, σ_p is a parameter which controls the penalty for position differences, and σ_p is a parameter controlling the amount of penalty incurred when the squared Euclidean distance between the vectors $\psi^l(x_{i+1}, .., x_{i+l})$ and $\psi^l(x'_{j+1}, .., x'_{j+l})$ differs.

3.3.1.3 k-NN

k-NN [Cover and Hart, 1967], a type of instance-based learning discussed in Section 3.2, is one of the simplest classification and regression algorithms. It is a lazy non-parametric method that does not make any assumptions on the underlying data distribution. In a classification task, where the data points are separated into several classes, the k-NN algorithm classifies a new point by a majority vote of its k nearest neighbors, that is, the class most common among them. 1-NN is the simplest variation of the k-NN algorithm where $k = 1$. The 1-NN algorithm simply assigns the new point to the class of its closest neighbor. There are different approaches how to define a distance metric for measuring the closeness between two data points. Euclidean distance is the most commonly used distance measure which is defined as

$$D(\mathbf{x}, \mathbf{x}') = \|\varphi(\mathbf{x}) - \varphi(\mathbf{x}')\| = \sqrt{\langle\varphi(\mathbf{x}), \varphi(\mathbf{x})\rangle + \langle\varphi(\mathbf{x}'), \varphi(\mathbf{x}')\rangle - 2\langle\varphi(\mathbf{x}), \varphi(\mathbf{x}')\rangle}. \quad (11)$$

By exploiting the kernel trick we can reformulate the Equation 11 as

$$D(\mathbf{x}, \mathbf{x}') = \sqrt{k(\mathbf{x}, \mathbf{x}) + k(\mathbf{x}', \mathbf{x}') - 2k(\mathbf{x}, \mathbf{x}')}, \quad (12)$$

where k can be any kernel. In this study, we use the 1-NN algorithm and consider the generic string kernel discussed in Section 3.3.1.2 for computing the distances in Equation 12. Therefore, for each new instance $\mathbf{x}'$ to be classified, we construct the generic string kernel $k_{gs}(\mathbf{x}, \mathbf{x}')$ between the new instance and all the instances $\mathbf{x}$ in the input data space $\mathcal{X}$ whose labels are known. The class label of the new data point $\mathbf{x}'$ is then determined by the class of the instance whose distance to $\mathbf{x}'$ is the smallest.

The remainder of this chapter is devoted to detailing the kernels which are adopted for the 3D structure datasets.

3.3.2 3D Structure-Based Kernels

One of the present challenges is to sort out similarities in protein structures in order to gain insight into their functions and characteristics. Previous methods to predict protein characteristics mainly relied on identifying similarity in sequence or structure between a protein of unknown attribute and one or more well-understood proteins [C Whisstock and M Lesk, 2003]. These methods require explicit transformation of protein structures into feature vectors, called topological descriptors, which often suffer from not preserving the rich topological information embedded in structures. Moreover, computation of these explicit topological descriptors can be computationally expensive. Therefore, to avoid this loss of information as well as the problem of explicit feature vector transformation, protein structures can be modeled as graphs and subsequently, graph kernel functions can be employed to measure the structural similarities between them. Graph kernels are efficient to compute as they compare substructures of graphs that are computable in polynomial time. Additionally, they allow us to use any kernel-based machine learning method on them. Besides bioinformatics and chemoinformatics, graph kernels find applications further afield in social network analysis, computer vision, and natural language processing.

At its most basic, a graph can be defined as a set of 2D or 3D entities and relationships that function in some interrelated way. Therefore, the 3D nature of proteins' tertiary or quaternary structures and the complex relationships in their polymers enable them to be represented as graphs by means of different mapping rules.

After introducing some of the most important concepts of graph theory, we will continue the chapter with detailed explanations how to model a protein as a graph as well as describing different and state-of-the-art graph kernels.

Graph Theory Concepts

A graph $G = \langle V, E \rangle$ consists of a set of vertices $V = \{v_1, v_2, \ldots, v_n\}$ and edges E, where $E \subset V \times V$. An edge $e \in E$ connects a pair of vertices $u, v \in V$, and is denoted as $e = (u, v)$. Vertices $u, v \in V$ are said to be neighbors or adjacent if they are connected by an edge, that is, if $e = (u, v) \in E$. Accordingly, we define the adjacency matrix of a graph $G = \langle V, E \rangle$ as $A_{n \times n} = [a_{ij}]$ where n is the number of nodes in the graph G and $a_{ij} = 1$ if (v_i, v_j) is an edge of G, and 0 otherwise.

We call a graph attributed or labeled when there are labels on nodes, edges, or both. Labels can be scalars or vectors with either discrete or continues values. A graph is called directed when the edges in E are ordered pairs of vertices, and undirected when they do not have a particular order or direction, i.e. the edge (u, v) is identical to the edge (v, u) (see Figure 6). A graph is simple and self-loop free if there is no edge connecting a vertex to itself and there are no multiple edges connecting the same pair of vertices.

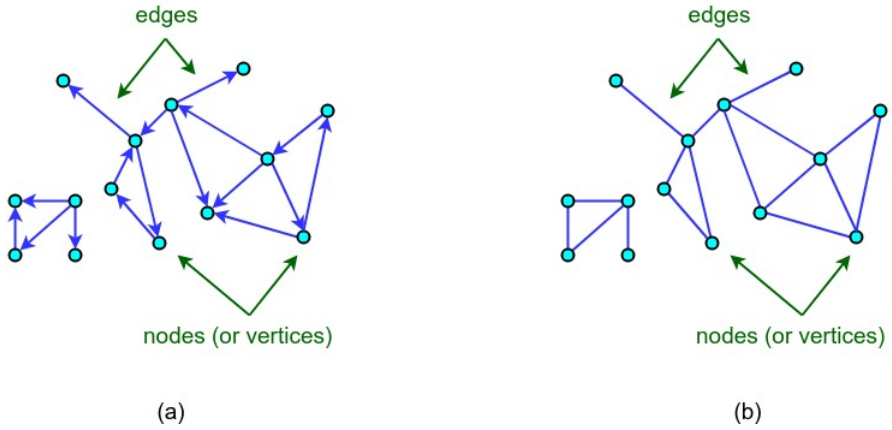

Figure 6: Examples of (a) a directed graph and (b) an undirected graph.

A walk of length k in graph G is a nonempty sequence of vertices $v_1, v_2, \ldots, v_k$ connected by edges $e_1, e_2, \ldots, e_{k-1}$ such that $e_i = (v_i, v_{i+1})$ for all $1 \leq i < k$. w is called a path p in G if the vertices in the sequence are all distinct (see Figure 7). We can also say that a graph G is connected if there is a path between every pair of distinct vertices in G, and disconnected otherwise. Examples in Figure 6 and Figure 7 are disconnected and connected graphs, respectively.

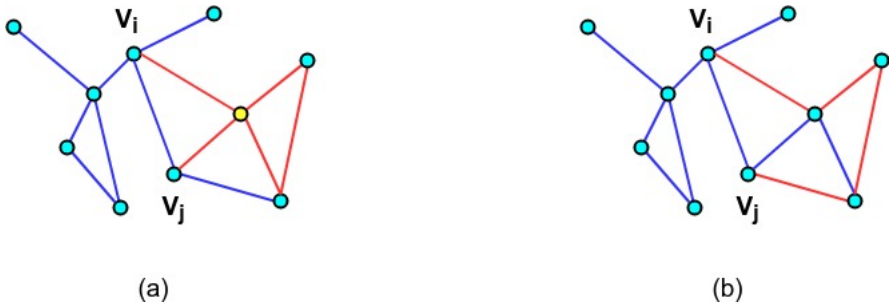

Figure 7: Examples of (a) a walk and (b) a path between vertices v_i and v_j.

$H = \langle V_H, E_H \rangle$ is a subgraph of G (or H has an embedding in G) with $V(H) \subseteq V(G)$ and $E(H) \subseteq E(G)$, denoted by $H \sqsubseteq G$. A graph is called a tree when any two vertices are connected by exactly one edge, resulting in a connected structure with no cycles. Therefore, a (rooted) subtree is an acyclic subgraph of a graph which has a designated root. The height of a subtree is then defined as the longest path between the root and any other node in the subtree. Subtree patterns allow repetitions of nodes. However, in order to avoid cycles, they treat the repetitions of the same node as distinct nodes (see Figure 8).

For two graphs $G = \langle V, E \rangle$ and $G' = \langle V', E' \rangle$, isomorphism is defined as a bijective mapping $f : V \to V'$ such that $(v_i, v_j \in E)$ if and only if $(f(v_i), f(v_j)) \in V'$. If there exist such an isomorphism f, G and G' called isomorphic.

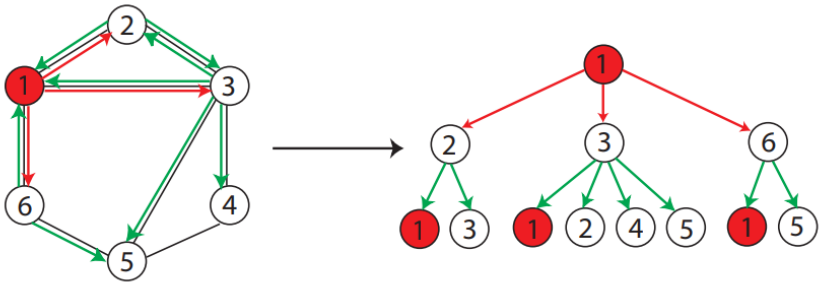

Figure 8: An example of a a subtree pattern of height 2 rooted at the node 1 where repetitions of nodes are allowed. [Shervashidze et al., 2011]

Proteins as Graphs

The structure of a protein is mainly governed by its residual interactions, peptide bonding, covalent interactions, and hydrophobic packing. Depending on the type of interaction, proteins can be mapped to topologically different graphs. When modeling a protein as a graph, one can define the vertex set based on different aspects of protein structures. Examples include residues [Samudrala and Moult, 1998], side chains [Canutescu et al., 2003], Cα atoms [Huan et al., 2005], SSE (secondary structure elements) [Borgwardt et al., 2005], and DSSP (the dictionary of protein secondary structures) [Peng and Tsay, 2010]. As for the edge set, usually the distance of two vertices with some labels, e.g., chemical properties, is translated into an edge between them. A pictorial overview of protein graph remodeling is depicted in Figure 9.

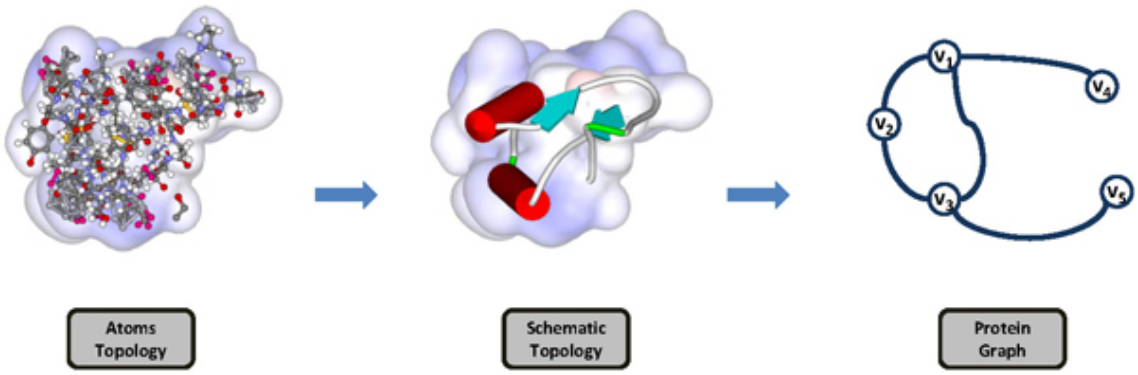

Figure 9: An overview of protein graph remodeling. [Peng and Tsay, 2014]

In this study, we model the proteins such that they contain information on their structures, sequences and chemical properties. To this end, we design our models as simple attributed and undirected graphs having no self-loops. Each graph corresponds to exactly one protein. Among different elements previously discussed to construct the vertex set, we opt for Cα atoms and consider edges between them if they are within a sphere of 5Å. In other terms, for each pair of Cα atoms, we measure the Euclidean distance between their 3D coordinates and assign an edge between them

only if the distance is smaller than 5Å. We do not consider any labels for the edges. However, vertices bear single-valued labels. We try different node attributes, namely secondary structure elements, i.e. helices, sheets or unknown, amino acid types, side chain classes, polarity statuses, and charge attributes. Details of these categorical features can be found in Appendix A. We follow the same approach for all the three 3D structure datasets discussed in Section 2.3.2.

Graph Kernels

The basic idea of graph kernels is to count common substructures in two graphs. Given two graphs G and G' from the space of graphs $\mathcal{G}$, the problem of graph comparison is defined as finding a mapping $s : \mathcal{G} \times \mathcal{G} \to \mathbb{R}$ such that the similarity of G and G' can be quantified as $s(G, G')$. A mapping satisfying the conditions of symmetry and positive definiteness is called a graph kernel. Many of the existing graph kernels are instances of the family of so-called R-convolution kernels [Haussler, 1999] which is a generic framework for defining kernels on discrete compound objects decomposed into smaller components. All pairs of decompositions, or subgraphs in other words, are then compared. In consequence, different types of decomposition relations R result in different graph kernels. The main purpose of convolution kernels is to make the comparison less difficult by means of defining and computing a simpler similarity measure over the smaller, less complex subgraphs. However, comparing all the decomposed subgraphs is at least as hard to compute as deciding if two graphs are isomorphic [Gärtner et al., 2003] which leads to restricting graph kernels to compare only specific types of subgraphs that are computable in polynomial time. The family of R-convolution kernels can be grouped into the following categories, namely graph kernels based on comparing all pairs of decomposed random walks [Gärtner et al., 2003; Kashima et al., 2003; Mahé et al., 2004; Vishwanathan et al., 2006], shortest paths [Borgwardt and Kriegel, 2005], cycles [Horváth et al., 2004], limited-size subgraphs (graphlets) [Borgwardt et al., 2007; Shervashidze et al., 2009], and subtree patterns [Ramon and Gärtner, 2003; Mahé and Vert, 2009; Shervashidze and Borgwardt, 2009; Shervashidze et al., 2011; Neumann et al., 2012; Feragen et al., 2013]. Most of these kernels suffer from poor scalability to large, labeled graphs containing more than 100 nodes. In this study, we have investigated two graph kernels which are scalable to large graphs: the graphlet kernel proposed by [Shervashidze et al., 2009] and the Weisfeiler-Lehman graph kernel, a state-of-the-art subtree kernel presented in [Shervashidze et al., 2011].

3.3.2.1 The Graphlet Kernel

The key idea of the graphlet kernel [Shervashidze et al., 2009] is similar to that of the k-spectrum kernel discussed in Section 3.3.1.1 with a difference that in the graphlet kernel, the value of k is bounded. The graphlet kernel compares the frequencies of all types of subgraphs of size $k \in \{3, 4, 5\}$ in two unlabeled graphs. These subgraphs are referred to as graphlets [Pržulj, 2007]. If G is a graph with n nodes, let $\mathcal{G} = \{graphlet(1), \ldots, graphlet(N_k)\}$ be the set of size-k graphlets in G. We denote

the number of occurrences of $subgraph(i)$ in G by $\#(graphlet(i) \sqsubseteq G)$. The vector f_G of length N_k, called the k-spectrum vector, can be defined as

$$f_{Gi} = \#(graphlet(i) \sqsubseteq G). \tag{13}$$

The graphlet kernel k_g over two graphs G and G' then takes the form:

$$k_g(G, G') = {f_G}^T f_{G'}. \tag{14}$$

Since the differences in the sizes of the graphs can greatly skew the frequency counts f_G, counts are normalized to probability vectors:

$$D_G = \frac{1}{\#all\ graphlets\ in\ G} f_G, \tag{15}$$

and therefore, the graphlet kernel can be reformulated as

$$k_g(G, G') = {D_G}^T D_{G'}. \tag{16}$$

In order to reduce the expensive runtime of computing all the graphlet distributions, two speedup schemes are employed which are based on graphlet sampling and the limitation of bounded degree graphs. According to the sampling scheme, if the number of samples achieves a given confidence with a small probability of error, then the empirical distribution is close to the actual distribution of the graphlets. The scheme based on the exploitation of the low maximum degree of graphs states that if d denotes the maximum degree of a graph, then the exact number of all graphlets of size $k \in \{3, 4, 5\}$ in a bounded graph G can be enumerated in $O(nd^{k-1})$ time, where n is the number of nodes in G.

In this project, we only consider graphlets of size 3 (see Figure 10).

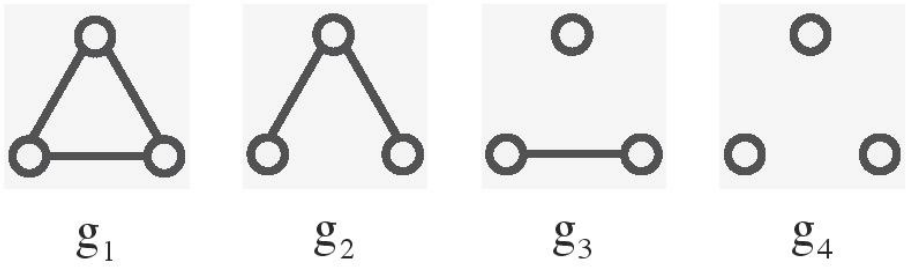

Figure 10: Graphlets of size 3 with different number of edges.

3.3.2.2 The Weisfeiler-Lehman Subtree Kernel

The Weisfeiler-Lehman (WL) subtree kernel [Shervashidze et al., 2011] is an efficient kernel which can scale to large graphs with thousands of both unlabeled and discretely labeled nodes. It encompasses many of the previously known graph kernels and in graph classification tasks, is competitive with or outperforms other state-of-the-art graph kernels. The general framework for the WL graph kernels is constructed according to the Weisfeiler-Lehman test of isomorphism which is used to determine whether two graphs are isomorphic.

The Weisfeiler-Lehman Test of Isomorphism

For two graphs G and G', the Weisfeiler-Lehman test algorithm proceeds in iterations, each of which consisting of a sequence of steps. In every iteration, the label of each node is augmented by the sorted set of node labels of its neighbouring nodes, called a multiset. These augmented labels are then compressed into new, short labels. Figure 11 depicts an illustration of these steps in the first iteration of the algorithm.

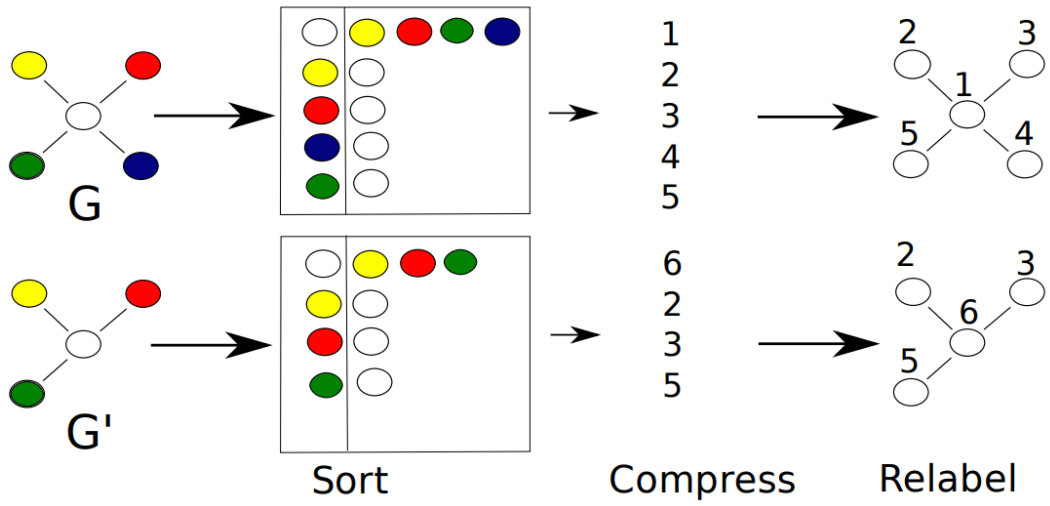

Figure 11: An illustration of the first iteration of the Weisfeiler-Lehman test of isomorphism [Borgwardt and Stegle, 2010].

Relabeling of the nodes with compressed, new labels in the last step is concordant in G and G', meaning that they will get identical new labels only if they have identical multiset labels. The Weisfeiler-Lehman algorithm terminates after this step if the number of iterations reaches n, or if the node label sets of G and G' start to diverge, that is, if the sets of newly created labels are not identical in G and G'. After n iterations, if the sets are identical, two cases are possible:

- either G and G' are isomorphic in fact, or
- the algorithm has failed to determine that they are not isomorphic.

The Weisfeiler-Lehman Kernel Framework

In this context, a graph G is defined as a triplet (V, E, l) where $l\colon V \to \Sigma$ is a mapping function which assigns labels from an alphabet Σ to the graph nodes. According to the WL algorithm, each node v gets a new labeling $l_i(v)$ in each iteration i. Therefore, one iteration of the WL algorithm can be defined as a relabeling function $r((V, E, l_i)) = (V, E, l_{i+1})$ which transforms all the graphs in a set of graphs $\mathcal{G}$ in the same manner. The compressed labels $l_i(v)$ in iteration i can be considered a subtree pattern of height i rooted at v (see Figure 8 for an illustration of subtree patterns).

If $G_i = (V, E, l_i)$ is the Weisfeiler-Lehman graph at height i of the graph $G = (V, E, l) = (V, E, l_0)$, the sequence of Weisfeiler-Lehman graphs up to height h of G is then given by

$$\{G_0, G_1, \ldots, G_h\} = \{(V, E, l_0), (V, E, l_1), \ldots (V, E, l_h)\}, \tag{17}$$

where $G_0 = G$, $l_0 = l$ and $G_i = r(G_{i-1})$.

Having defined the sequence of Weisfeiler-Lehman graphs, we can formulate the general Weisfeiler-Lehman kernel on two graphs G and G' with h iterations as

$$k_{WL}^{(h)}(G, G') = k(G_0, G'_0) + k(G_1, G'_1) + \ldots, k(G_h, G'_h), \tag{18}$$

where $k(.,.)$ is a base kernel and $\{G_0, \ldots, G_h\}$ and $\{G'_0, \ldots, G'_h\}$ are the Weisfeiler-Lehman sequences of G and G' up to height h, respectively.

The Weisfeiler-Lehman subtree kernel is an instance of the general WL kernel (Equation 20) that compares subtrees of height h in two graphs. The base kernel k of the WL subtree kernel is a function which counts pairs of matching node labels in two graphs G and G' given by

$$k(G, G') = \sum_{v \in V} \sum_{v' \in V'} \delta(l(v), l(v')), \tag{19}$$

where δ is the Dirac kernel, which is 1 when the labels $l(v)$ and $l(v')$ are equal, and 0 otherwise. In other terms, the Weisfeiler-Lehman subtree kernel adds up the common labels in two graphs counted in each iteration $i \in \{0, 1, \ldots, h\}$. Thus, it can be also expressed as

$$k_{WLsubtree}^{(h)}(G, G') = \langle \phi_{WLsubtree}^{(h)}(G), \phi_{WLsubtree}^{(h)}(G') \rangle, \tag{20}$$

where $\phi_{WLsubtree}^{(h)}$ is a feature vector whose components correspond to the frequency of occurrences of common original and compressed labels in two graphs G and G' up to a height h (see Figure 12 for an illustration).

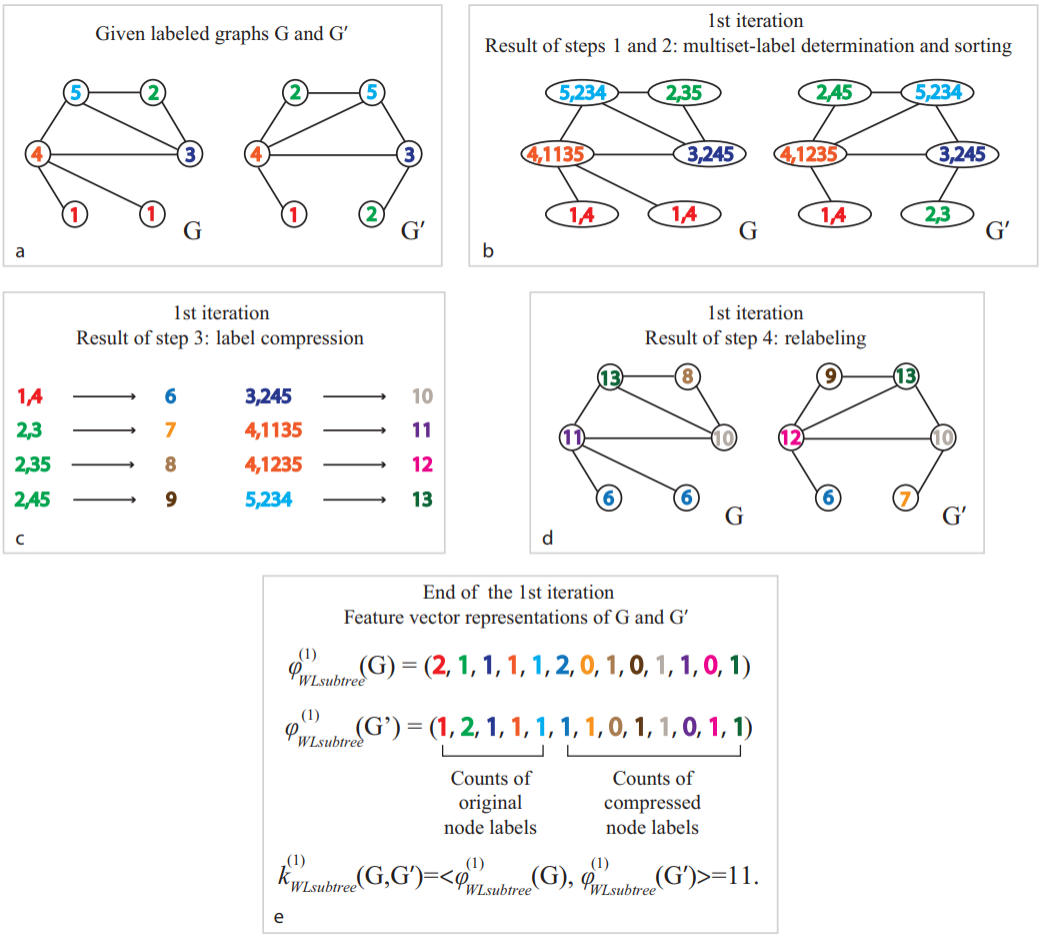

Figure 12: Illustration of the computation of the Weisfeiler-Lehman subtree kernel for height $h = 1$ [Shervashidze et al., 2011].

4 Experimental setting

This chapter is concerned with a series of experiments and their setups conducted to evaluate the performance of the methods discussed in Chapter 3, which address the problem of protein promiscuity classification.

4.1 Performance Measures

In a binary classification, there are 4 types of predictions:

- True Positive (TP): instances correctly classified as positive
- False Positive (FP): instances incorrectly classified as positive
- True Negative (TN): instances correctly classified as negative
- False Negative (FN): instances incorrectly classified as negative.

Confusion matrix, which is constructed according to the above-mentioned definitions, is a clean and unambiguous way to present and summarize the prediction results of a classifier (See Figure 13).

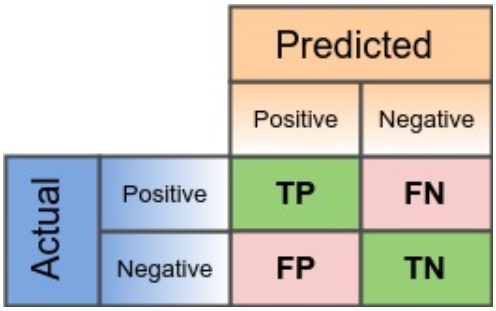

Figure 13: Confusion matrix.

In machine learning, there are different metrics for measuring the performance of a classifier. In order to assess and compare the different methods, we have utilized the following measures defined based on the confusion matrix:

Accuracy: the most intuitive and natural measure to evaluate a classifier on a set of test data is accuracy defined as the number of correct predictions the classifier has achieved divided by the total number of observations in the test set.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} * 100 \tag{21}$$

Precision: the ratio of correctly predicted positive observations to the total predicted positive observations is called precision.

$$Precision = \frac{TP}{TP + FP} \tag{22}$$

According to Equation 22, if all the test instances have been predicted as negative, the denominator becomes zero. We have reported these precisions as N/A in the next chapter.

Recall: the ratio of correctly predicted positive observations to the all observations in the positive class is called precision.

$$Recall = \frac{TP}{TP + FN} \tag{23}$$

F1 Score: in case of imbalanced datasets, where the samples from one of the classes outnumbers the other, the accuracy determined using Equation 21 may not be an adequate measure of performance. Moreover, accuracy is not capable of capturing the effectiveness of a classifier when the performance of one of the classes in particular is of interest. Compared to the above 3 metrics, the F1 score provides a more balanced view. It is defined as the weighted average of precision and recall.

$$F1\ Score = \frac{2 * (Precision * Recall)}{Precision + Recall} \tag{24}$$

Acquiring a high precision, but low recall indicates that the predictor is quite reliable and sensitive not to make mistakes in labeling the negative examples incorrectly as positive. This means that the positive portion of the predictions can be safely assumed as actually positive. However, the classifier has not been successful in detecting most of the positive examples.

The opposite scenario, when the precision is low, but a high recall is achieved suggests that the classifier is capable of identifying most of the positive examples, while it is not reliable whether a positive-labeled example is actually positive.

Low precision and low recall, on the other hand, means that the classifier's performance in identifying actual positive instances has been quite poor. This results in a low F1 score as well.

High values for both precision and recall are achieved when the classifier has been able to accurately identify both of the negative and positive classes accurately. This scenario leads to a high F1 score.

4.2 Balanced Datasets

The most commonly used classification algorithms such as SVM [Cortes and Vapnik, 1995], neural networks and decision trees [Quinlan, 1986] are designed primarily for balanced datasets, where there are equal or almost equal number of observations for all the classes. The objective functions of these algorithms are usually optimized such that the overall accuracy is maximized. When these methods are applied on very imbalanced datasets, they often tend to produce majority classifiers as the data is skewed in favour of one of the classes. This culminates in overpredicting the presence of the majority class. One way to tackle this problem is to sample the dominating class such that the classes are represented equally. While this can improve the classification performance, we might loose information about frequency of appearances of certain examples.

According to Chapter 2, our final list consist of 406 promiscuous and 7,916 non-promiscuous proteins. The ratio of the negative class (non-promiscuous) to the positive class (promiscuous) instances is thus 19.5:1, resulting in a highly imbalanced dataset. In order to circumvent the aforementioned problem caused by sampling, and to draw fair inferences, we have aimed to assess the methods over 5 different balanced subsets sampled from the datasets described in Chapter 2. However, due to the small number of positive instances, for each subset, we have included all the 406 promiscuous proteins, and have sampled only the negative class which has 7,916 non-promiscuous proteins. This led to the construction of 5 balanced subsets, each of which consisting of 406 non-promiscuous proteins sampled uniformly at random from the negative class, and all the 406 promiscuous proteins. The sampling was done with replacement, yet there is little overlap between the negative instances of the 5 subsets. All the results presented in Chapter 5 based on the metrics described in Section 4.1 are reported as averages over the 5 subsets.

4.3 Cross-Validation

In order to limit problems like overfitting, in each subset, all the statistics reported as performance measures described in Section 4.1 are evaluated by performing a 5-fold cross-validation (CV). In case of parametric methods, where optimizing the hyperparameters is required, a nested cross-validation setting with 5 outer and 3 inner folds is employed. In a nested cross-validation scheme, fitting of the hyperparameters takes place in the inner folds according to the achieved accuracy, while the outer folds estimate the performance of the model. The overall performance of the model for each of the 5 subsets is then computed as the average over the 5 outer test folds.

4.4 SVM

In this study, all the generated kernels are first centered and then normalized before being evaluated by SVM. For any kernel matrix $\mathbf{K} \in \mathbb{R}^{n \times n}$ with n instances, the centered kernel matrix $\mathbf{K}_c$ can be expressed as follows

$$\mathbf{K}_c = \left[\mathbf{I} - \frac{\mathbf{1}\mathbf{1}^\top}{n}\right]\mathbf{K}\left[\mathbf{I} - \frac{\mathbf{1}\mathbf{1}^\top}{n}\right]. \tag{25}$$

For a pair of instances $(\mathbf{x}_i, \mathbf{x}_j)$, we normalize their corresponding kernel $k(\mathbf{x}_i, \mathbf{x}_j)$ as

$$\hat{k}(\mathbf{x}_i, \mathbf{x}_j) = \frac{k(\mathbf{x}_i, \mathbf{x}_j)}{\sqrt{k(\mathbf{x}_i, \mathbf{x}_i)k(\mathbf{x}_j, \mathbf{x}_j)}}. \tag{26}$$

Centering the kernels, we set the parameter C of SVM equal to 1 (See Equation 3). SVM then trains and cross-validates the kernels, and for each of the outer test folds, outputs the predicted labels as well as the accuracies achieved by the methods.

5 Results

The objective of this study was to find a predictive model to successfully classify the unseen proteins into promiscuous and non-promiscuous. Our approach was to leverage the physicochemical properties of proteins' amino acid sequences as well as their structural information, and employ appropriate models accordingly.

Having investigated the previously discussed methods, in this chapter, we will report our observed results. The chapter falls into two main sections. The first presents the results from the analysis of protein's primary structures, which correspond to the sequence-based dataset. In the latter, we provide the results acquired from studying the 3D-structured datasets which carry information on proteins' tertiary or quaternary structures. The chapter then continues with detailing the most important observations accompanied by their comprehensive visualizations.

5.1 Sequence-Based Models

This section presents evaluations of the sequence-based dataset by means of different methods.

5.1.1 BLAST

Analyzing proteins' sequences is often the first and most standard approach for perceiving proteins' functionality. We investigated the significance of protein's sequences by employing the BLAST algorithm, which is one of the most established methods for biological sequence analysis, and hence an apt baseline.

We evaluated the BLAST algorithm by means of BLAST+, which is a suite of command-line tools [Camacho et al., 2009]. In order to perform a 5-fold cross-validation, we randomly divided each of the 5 subsets discussed in Section 4.2 into 5 predefined folds. Then using the `makeblastdb` command, for each test fold of the cross-validation, a BLAST protein database was built from the training data in the remaining 4 folds. Next, by running the `blastp` command, we scanned each of the sequences in the test set against the database to find their most similar sequences. The default setting of blastp returned the hits according to the BLOSUM62 scoring matrix, and for a similarity threshold of $T = 11$ to add a word to the BLAST lookup table. Since we considered only one value for the similarity threshold T, performing a nested cross-validation was not required and an ordinary 5-fold cross-validation sufficed. Table 2 demonstrates the statistical results obtained for the BLAST algorithm.

BLAST			
Accuracy	**F1 Score**	**Precision**	**Recall**
61.5%	0.67	0.59	0.77

Table 2: Performance of the BLAST algorithm evaluated in terms of accuracy, and the F1 score, precision, and recall of the positive class.

According to Table 2, the low values of the accuracy and F1 score indicate the poor performance of the BLAST algorithm for this application. However, a higher value of the recall compared to the precision means that the model has been, to some extent successful, in identifying most of the promiscuous proteins, while failing to detect a pattern in the non-promiscuous class to make them distinguishable. In other words, the promiscuous determinants have been incorrectly generalized to the non-promiscuous proteins as well.

5.1.2 The k-Spectrum Kernel

Among string kernels, k-spectrum kernel is one of the simplest and most straightforward approaches for analyzing biological sequences. For this kernel discussed in Section 3.3.1.1, we tried k-mers of different length $k \in \{3, 4, 5\}$. The optimal value for k is then determined by the inner folds of the cross-validation. The results are shown in Table 3. We also took a closer look at the CV to investigate for each of the 5 outer folds, which length k has been selected as the most optimal during the inner CV. Figure 14 shows a histogram of the number of times each length k has been selected as the best over the 5 subsets, each of which 5 outer folds.

k-Spectrum Kernel			
Accuracy	**F1 Score**	**Precision**	**Recall**
62.7%	0.58	0.67	0.51

Table 3: Performance of the k-spectrum kernel evaluated with a nested cross-validation scheme for $k \in \{3, 4, 5\}$.

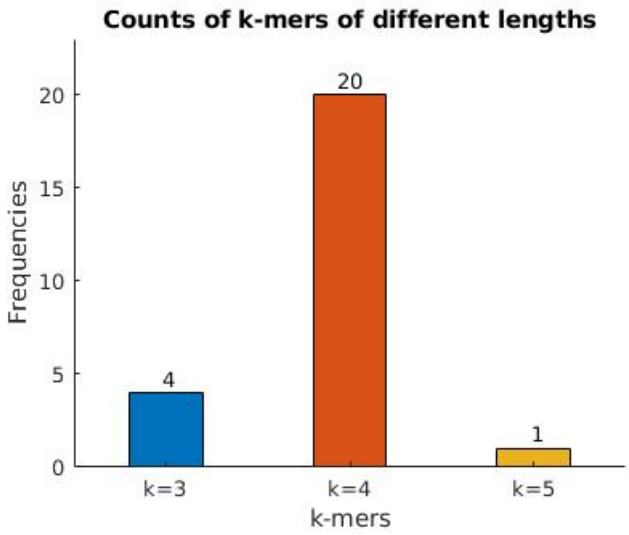

Figure 14: Counts of optimal k-mers, $k \in \{3, 4, 5\}$ selected during the inner cross-validations for the k-spectrum kernel.

Table 3 demonstrates that the k-spectrum kernel has not succeeded in classifying the promiscuous and non-promiscuous proteins in terms of any of the evaluation metrics. However, as it can been seen, the precision is slightly higher than the recall. This could be due to the model distinguishing certain patterns in some of the promiscuous proteins, while not being able to generalize them to the other promiscuous proteins. Figure 14 displays that k-mers of length 4 have been the most informative compared to lengths 3 and 5.

Furthermore, we assessed the performance of the k-spectrum kernel for each k-mer, $k \in \{3, 4, 5\}$, separately with a regular 5-fold cross-validation and computed the averages over the 5 subsets. The statistical results are shown in Table 4.

k-Spectrum Kernel				
k-mer	**Accuracy**	**F1 Score**	**Precision**	**Recall**
$k = 3$	61.1%	**0.58**	0.64	**0.54**
$k = 4$	**63.2%**	**0.58**	0.68	0.52
$k = 5$	62.1%	0.48	**0.78**	0.36

Table 4: Performance of the k-spectrum kernel for different k-mers, $k \in \{3, 4, 5\}$. Best results for each metric are highlighted in pale yellow.

Table 4 is consistent with the findings of Figure 14 in that k-mers of length 4 have led to a better performance in terms of accuracy, while k-mers of length 5 have been preferred the least. However, k-mers of length 5 and 3 offer better precision and recall, respectively. k-mers of length 3 and 4 have performed equivalently based on the to F1 score.

5.1.3 The Generic String Kernel

According to Equation 10 in Section 3.3.1.2, the generated kernel K_{GS} varies depending on the choice of the encoding function ψ. Investigating different factorized representations $\psi(a)$ for amino acid symbols a, we constructed different kernels. BLOSUM50 and BLOSUM62 substitution matrices have been tried as the most common descriptor choices. Furthermore, as for the physicochemical properties of amino acids, we used their weight and hydropathy index real numbers as single-valued descriptors (see Table 1). To deal with other properties which are categorical variables, we employed the one hot encoding (OHE) technique to convert them into vectorial representations. By means of the OHE method, each categorical property can be represented as a sparse binary vector where each column corresponds to one possible category of that property. Then, for each category variable, all columns are equal to zero except for its category column which is equal to one. The categorical physicochemical properties examined in this study are amino acids' residue types, side chain class, polarity, and charge. Elaborate vectorial representations of these features as well as the BLOSUM50 and BLOSUM62 substitution matrices can be found in Appendix B.

According to Equation 10, for each kernel corresponding to a descriptor, 3 hyperparameters needed to be optimized, namely L, σ_p, and σ_c, which are the substring length, variance in position, and variance in content, respectively. The optimal hyperparameters were selected during the inner cross-validation by grid search using the following ranges: $L \in [1\,..\,9]$, $\sigma_p \in [1\,..\,16]$, and $\sigma_c \in (0, 5]$. This was carried out for each of the subsets individually. Table 5 presents the results on the performance of the GS kernel for different amino acid descriptors computed as averages over the 5 subsets.

Generic String Kernel				
AA Descriptors	**Accuracy**	**F1 Score**	**Precision**	**Recall**
BLOSUM50	63.3%	**0.60**	**0.66**	0.55
BLOSUM62	63%	0.59	**0.66**	0.55
Hydropathy Index	**63.4%**	**0.60**	**0.66**	0.54
Weight	62.3%	0.59	0.65	0.54
AA Type	62.9%	**0.60**	0.65	**0.56**
Side Chain Class	63.3%	0.59	**0.66**	0.54
Side Chain Polarity	61.9%	0.58	0.65	0.52
Side Chain Charge	61.8%	0.57	0.65	0.51

Table 5: Performance of the generic string kernel evaluated for different amino acid descriptors. Best results for each metric are highlighted in pale yellow.

According to Table 5, performance of the generic string kernel is statistically disadvantageous. However, the kernel performs marginally better in terms of accuracy when hydropathy indices of amino acids are employed. Similarly, F1 score, precision, and recall are consistent as for almost all of the amino acid descriptors, except for an insignificant increase of 0.1 for some of them, which are highlighted in pale yellow. Furthermore, slight differences are observed in the values of precision and recall, where the former is higher by 0.10.

Similar to the k-spectrum kernel, for each amino acid descriptor, we have scrutinized the frequencies of optimal hyperparameters selected during the inner cross-validation. The histogram counts of these hyperparameters for different amino acid descriptors are illustrated in Figures 15, 16, and 17.

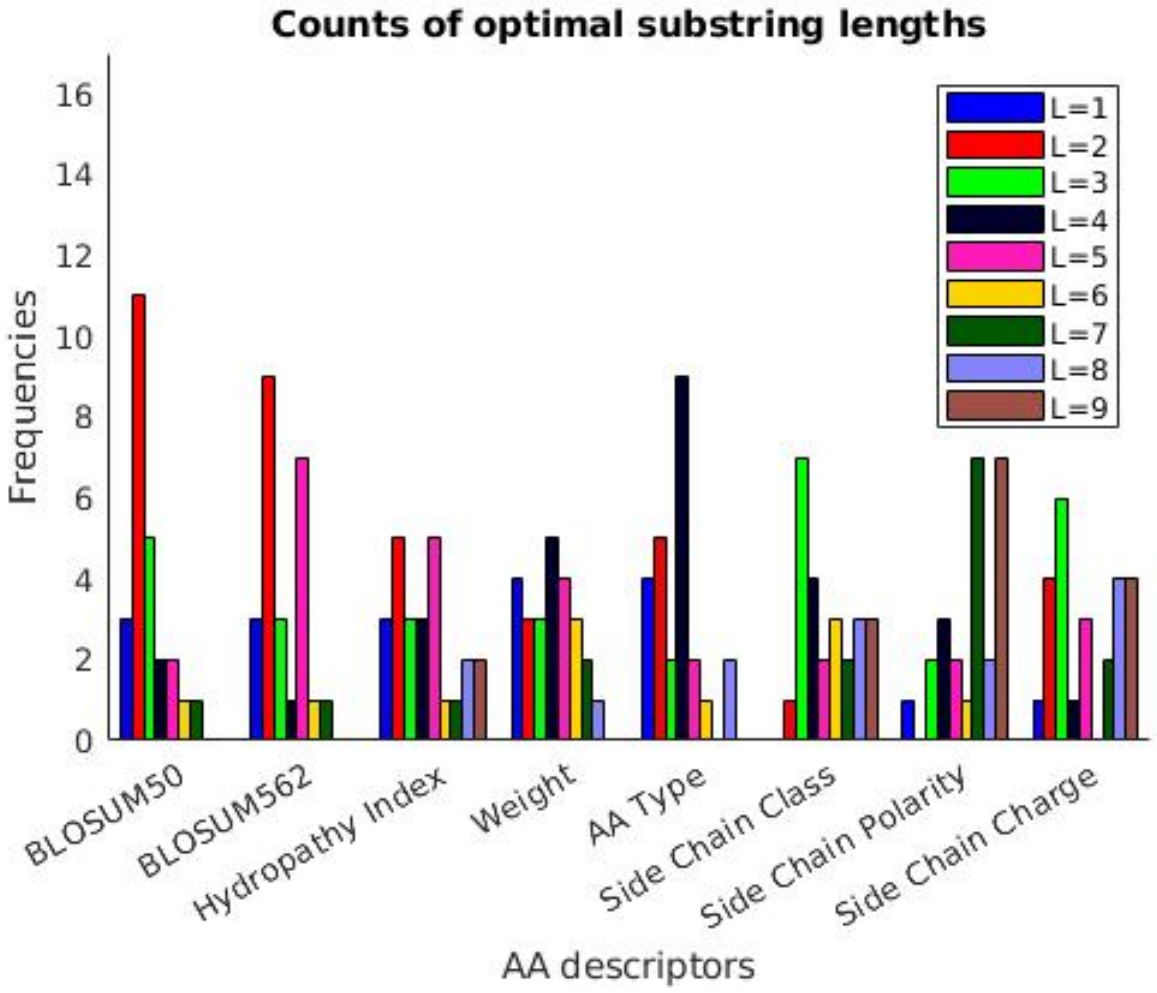

Figure 15: Counts of optimal substring lengths selected during the inner cross-validations for different AA descriptors of the generic string kernel.

The large variance of the substring lengths in Figure 15 indicates that the generic string kernel has not achieved a consensus whether a certain substring length of amino acids is more informative than the others. This can be due to the diverse set of sequences being examined which are of different lengths. The same can be concluded from Figure 16. Position differences are penalized in an inconsistent way displaying no specific patterns.

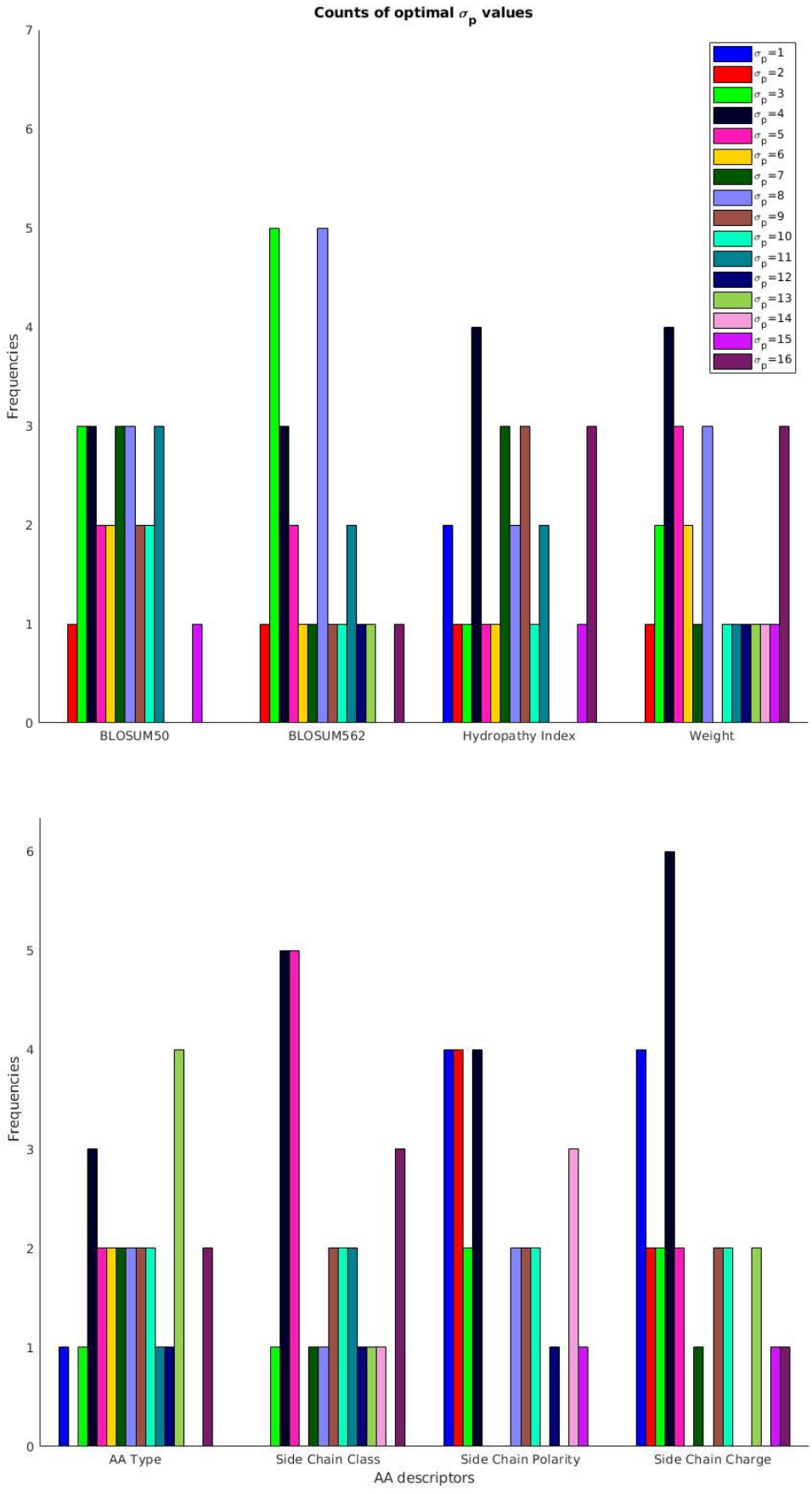

Figure 16: Counts of optimal σ_p values selected during the inner cross-validations for different AA descriptors of the generic string kernel.

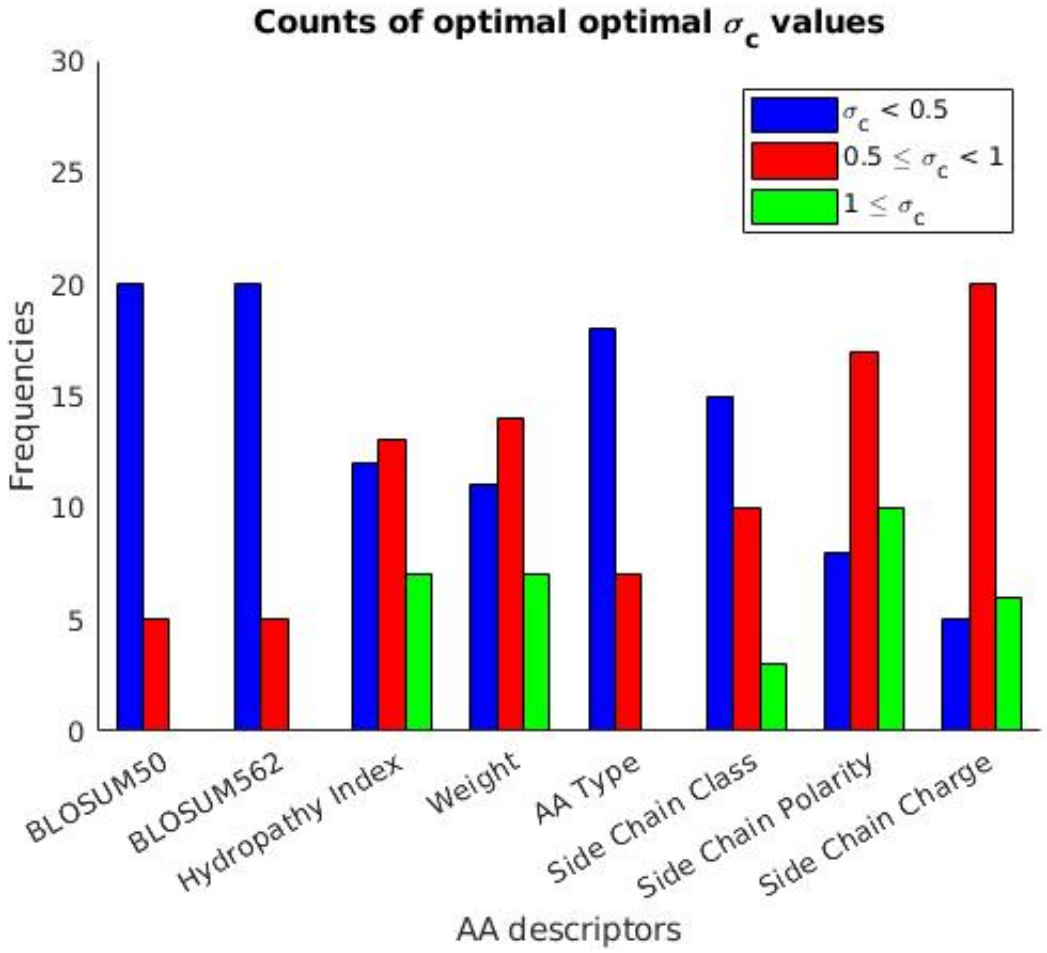

Figure 17: Counts of optimal σ_c values selected during the inner cross-validations for different AA descriptors of the generic string kernel.

Unlike the L and σ_p hyperparameters, σ_c exhibits a preference for smaller values as shown in Figure 17. The histogram frequencies display that all of the amino acid descriptors are in favor of values smaller than 1 in spite of the large range which was tested ($(0, 5]$). According to Equation 10, the low values of σ_c indicate that the differences in the vectors of descriptors are penalized heavily, meaning that they carry relevant information up to some extent.

5.1.3.1 k-NN

To further evaluate the performance of the GS kernel constructed by the BLOSUM50 scoring matrix, we employed the 1-NN algorithm according to the Euclidean distance metric defined in Equation Figure 12. Similar to Section 5.1.3, a nested cross-validation is performed to optimize the hyperparameters of the centered and normalized generic string kernel over the same range. Table 6 shows the results by means of statistical metrics.

k-NN			
Accuracy	**F1 Score**	**Precision**	**Recall**
61.7%	0.63	0.61	0.65

Table 6: Performance of the generic string kernel evaluated by the 1-NN algorithm.

According to Table 6, the results obtained from assessing the performance the generic string kernel via the 1-NN algorithm demonstrate no significant improvement compared to the previous methods. This suggests that the generic string kernel is not capable of distinguishing the determinants of promiscuity accurately.

5.2 3D Structure-Based Models

Next, we will present the results from analyzing the 3D structure-based datasets using graph kernels. As discussed in Section 2.3.2, we have constructed 3 different 3D structure datasets by considering different criteria. For the sake of brevity, we rename them as $3D_1$, $3D_2$, and $3D_3$ which correspond to the UniProt-unique-chains dataset, the UniProt-all-chains dataset, and the all-coordinates dataset, respectively.

5.2.1 The Graphlet Kernel

The original graphlet kernel proposed by [Shervashidze et al., 2009] was designed for unlabeled graphs. However, an extension of the algorithm applicable to labeled graphs has been provided as well[2]. In order to investigate the importance of the attributes, we studied both of the unlabeled and labeled graphlet kernels. For the labeled graphlet kernel, we considered different node attributes, namely secondary structure elements, amino acid types, side chain classes, polarities, and charges. A comprehensive list of each attribute and its corresponding categories can be found in Appendix A. Since the graphlet kernel is not a parametric method, an ordinary cross-validation setting was used. Table 7 presents the results on the performance of the unlabeled graphlet kernel as well as evaluations of the labeled kernel for different node attributes. The aforementioned results are reported for all the 3D structure datasets.

[2] https://github.com/BorgwardtLab/graph-kernels.git

Graphlet Kernel					
Node Attributes	**Datasets**	**Accuracy**	**F1 Score**	**Precision**	**Recall**
Unlabeled	$3D_1$	49.6%	0.43	0.40	0.48
	$3D_2$	50%	**0.55**	**0.48**	0.65
	$3D_3$	**50.4%**	0.54	0.47	**0.66**
SSE	$3D_1$	**55.3%**	**0.50**	**0.57**	**0.44**
	$3D_2$	51.3%	0.40	N/A	0.35
	$3D_3$	49.4%	0.43	0.51	**0.44**
AA Type	$3D_1$	**55.4%**	**0.50**	**0.57**	**0.45**
	$3D_2$	50.5%	0.42	0.50	0.39
	$3D_3$	48%	0.31	0.37	0.33
Side Chain Class	$3D_1$	**55.3%**	**0.50**	**0.57**	**0.44**
	$3D_2$	51.4%	0.40	N/A	0.35
	$3D_3$	50%	0.41	N/A	0.40
Side Chain Polarity	$3D_1$	**55.4%**	**0.50**	**0.57**	0.44
	$3D_2$	50.5%	0.42	N/A	0.41
	$3D_3$	48%	0.40	N/A	**0.45**
Side Chain Charge	$3D_1$	**55.4%**	**0.50**	**0.57**	**0.44**
	$3D_2$	51.5%	0.41	N/A	0.36
	$3D_3$	49.4%	0.38	N/A	0.38

Table 7: Performance of the graphlet kernel evaluated for different 3D structure datasets and different node attributes. Best results for each attribute are highlighted in bold. The statistics highlighted in pale yellow correspond to the best results among all the datasets and attributes.

In Table 7, the first apparent observation is an overall improvement in the accuracy by 5% when node labels are considered compared to the unlabeled graph. The low performance of the unlabeled graph for all the $3D_1$, $3D_2$ and $3D_3$ datasets can indicate that the graphlet kernel cannot identify any special patterns in the 3D shapes of the promiscuous proteins when the nodes are unlabeled. As for the node attributes, the values of all performance metrics are quite similar: firstly, for all of the attributes, the accuracy of the $3D_1$ dataset outperforms the $3D_2$ and $3D_3$ datasets by almost 4% and 5%, respectively. Secondly, there is an improvement of roughly 0.10 in the F1 score when dataset $3D_1$ is employed compared to datasets $3D_2$ and $3D_3$. However, the F1 score achieved by the unlabeled graphs is slightly better than all the graphs with node attributes. As for the precisions, most of the attributed graphs in datasets $3D_2$ and $3D_3$ have failed to predict any promiscuous proteins leading to an N/A value, while the $3D_1$ dataset has acquired better precisions, which are also higher than the recalls by 0.13. The recall values of the graphlet kernel are relatively low for all the datasets, yet the unlabeled graph has gained a higher recall. As an overall inference, the graphlet kernel shows incompetency in classifying the promiscuous and non-promiscuous proteins.

As an additional experiment, in all the 3D structure datasets, we discarded the nodes whose secondary structure labels were 'unknown' (see Appendix A), and reevaluated the graphlet kernel with all the node attributes as well as the unlabeled graphlet kernel. The results are illustrated in Table 8.

All the observations in Table 7 are applicable to Table 8 as well. However, in Table 7, slightly superior results are seen compared to Table 8 where the nodes with SSE labels as 'unknown' are discarded. These differences are too insignificant to bear any implications.

Graphlet Kernel					
Node Attributes	**Datasets**	**Accuracy**	**F1 Score**	**Precision**	**Recall**
Unlabeled	$3D_1$	49.2%	0.50	0.36	0.61
	$3D_2$	48.9%	**0.52**	**0.39**	**0.66**
	$3D_3$	**49.4%**	0.51	0.10	**0.66**
SSE	$3D_1$	**54.7%**	**0.49**	**0.56**	**0.44**
	$3D_2$	51.3%	0.42	N/A	0.38
	$3D_3$	49%	0.35	N/A	0.36
AA Type	$3D_1$	**54.9%**	**0.49**	**0.56**	**0.44**
	$3D_2$	51.4%	0.36	0.46	0.29
	$3D_3$	48%	0.38	0.44	0.40
Side Chain Class	$3D_1$	**54.9%**	**0.49**	**0.56**	0.44
	$3D_2$	51.5%	0.36	N/A	0.30
	$3D_3$	49.1%	0.42	N/A	**0.45**
Side Chain Polarity	$3D_1$	**54.8%**	**0.49**	**0.56**	**0.44**
	$3D_2$	50.3%	0.41	N/A	0.38
	$3D_3$	49.6%	0.37	N/A	0.35
Side Chain Charge	$3D_1$	**54.9%**	**0.49**	**0.56**	0.44
	$3D_2$	50.9%	0.42	N/A	0.41
	$3D_3$	49.9%	0.45	0.52	**0.48**

Table 8: Performance of the graphlet kernel evaluated for different 3D structure datasets and different node attributes without considering the nodes with the 'unknown' label in the SSE category. Best results for each attribute are highlighted in bold. The statistics highlighted in pale yellow correspond to the best results among all the datasets and attributes.

5.2.2 The Weisfeiler-Lehman Subtree Kernel

The WL subtree kernel elucidated in Section 3.3.2.2 has been evaluated for different kernels generated with the same node attributes as the graphlet kernel (See Section 5.2.1). For each kernel, subtrees up to height $h = 20$ have been considered. The optimal height h is then selected during the inner 3-fold cross-validation. We have repeated the same procedure for all the 3D structure datasets. The statistical evaluations are shown in Table 9.

Similar to the graphlet kernel, we conducted an experiment to assess the performance of the WL kernel when the nodes with the secondary structure labels as 'unknown' are excluded. The evaluations are illustrated in Table 10.

WL Kernel					
Node Attributes	**Datasets**	**Accuracy**	**F1 Score**	**Precision**	**Recall**
SSE	$3D_1$	**93.2%**	**0.93**	**0.96**	**0.91**
	$3D_2$	69.6%	0.60	0.86	0.47
	$3D_3$	65.4%	0.55	0.79	0.43
AA Type	$3D_1$	**65%**	**0.52**	**0.85**	0.38
	$3D_2$	59.8%	0.49	0.73	**0.41**
	$3D_3$	56.7%	0.46	0.66	0.37
Side Chain Class	$3D_1$	**69.4%**	**0.60**	**0.87**	**0.47**
	$3D_2$	63.1%	0.55	0.72	0.46
	$3D_3$	60%	0.50	0.69	0.41
Side Chain Polarity	$3D_1$	**70.6%**	**0.60**	**0.96**	**0.44**
	$3D_2$	60.4%	0.49	0.71	0.40
	$3D_3$	59.1%	0.45	0.71	0.35
Side Chain Charge	$3D_1$	**76.1%**	**0.70**	**0.95**	**0.55**
	$3D_2$	63.9%	0.52	0.80	0.40
	$3D_3$	61.1%	0.49	0.74	0.38

Table 9: Performance of the WL kernel evaluated for different 3D structure datasets and different node attributes. Best results for each attribute are highlighted in bold. The statistics highlighted in pale yellow correspond to the best results among all the datasets and attributes.

WL Kernel					
Node Attributes	**Datasets**	**Accuracy**	**F1 Score**	**Precision**	**Recall**
SSE	$3D_1$	**96.3%**	**0.96**	**0.94**	**0.99**
	$3D_2$	95.8%	**0.96**	0.93	**0.99**
	$3D_3$	88.4%	0.88	0.91	0.85
AA Type	$3D_1$	**75.7%**	**0.69**	**0.94**	**0.55**
	$3D_2$	61.9%	0.49	0.76	0.38
	$3D_3$	56.9%	0.45	0.69	0.37
Side Chain Class	$3D_1$	**91.8%**	**0.91**	**0.98**	**0.85**
	$3D_2$	63.9%	0.53	0.78	0.42
	$3D_3$	61.5%	0.50	0.74	0.39
Side Chain Polarity	$3D_1$	**91.7%**	**0.91**	**0.98**	**0.85**
	$3D_2$	63.5%	0.51	0.79	0.39
	$3D_3$	60.7%	0.46	0.78	0.35
Side Chain Charge	$3D_1$	**85.5%**	**0.85**	**0.90**	**0.80**
	$3D_2$	66.4%	0.56	0.83	0.44
	$3D_3$	61.9%	0.48	0.79	0.37

Table 10: Performance of the WL kernel evaluated for different 3D structure datasets and different node attributes without considering the nodes with the 'unknown' label in the SSE category. Best results for each attribute are highlighted in bold. The statistics highlighted in pale yellow correspond to the best results among all the datasets and attributes.

The results of the Weisfeiler-Lehman subtree kernel displayed in Table 9 illustrate a substantial improvement over the graphlet kernel. As the first finding, we can confirm that the statistics of the $3D_1$ dataset significantly outperform those of the $3D_2$ and $3D_3$ datasets for most of the attributes. This enhancement is remarkable specially when the secondary structure elements are evaluated as attributes, which have also achieved the best performance among all the attributes. The $3D_3$ dataset, on the other hand, demonstrates the lowest evaluation metric values. The second inference is concerning the side chain properties of the amino acids which show similar performances. Furthermore, amino acid types as attributes of the $3D_1$ dataset result in the lowest accuracy, F1 score, precision and recall compared to the other attributes of this dataset. However, their ability to find a pattern in some promiscuous proteins, which are not generalized to others, explained by the high precision and low recall is worth noting. This phenomenon is the case for the side chain, polarity, and charge attributes as well. As an overall inference, these evaluations indicate that the WL kernel with the secondary structure elements can successfully identify the relevant patterns in both of the promiscuous and non-promiscuous proteins. While the performance of the other attributes are not as promising as the secondary structure elements, by their high precision values they suggest there are certain promiscuous proteins for which a specific pattern is detectable.

Inferences from Table 9 are also applicable to Table 10, where the nodes with 'unknown' SSE labels are discarded. However, as can be seen clearly, the accuracy of the WL kernel with SSE attributes in the $3D_1$ dataset has been improved by 3% compared to Table 9. This can imply that the discarded nodes were adding extra information which was deleterious to the model. The same statement can be made as for the other attributes as well considering the significant improvements in their performance measures. One of the most interesting observations is the huge shift in the performance of the SSE attributes in the $3D_2$ dataset as well as a slightly less significant enhancement in the $3D_3$ dataset. Moreover, the recall values of 0.99 in both $3D_1$ and $3D_2$ datasets when SSE attributes are evaluated indicate that the WL graph kernel has succeeded in accurately predicting almost all of the promiscuous proteins correctly. The high F1 score supports the fact that it has been as successful in the non-promiscuous class as well. The other intriguing fact is the significant enhancements of the side chain class and polarity attributes. Since excluding the nodes whose SSE labels are 'unknown' has affected all of the attributes, we can infer that there is an interconnection between all of them as determinants of promiscuity.

All the results in Tables 9 and 10 have been evaluated using a nested cross-validation to optimize the subtree height h of the WL subtree kernel. In order to probe the possibility of a certain height being favored over the others, in Figures 18-20, we have depicted the frequency counts of each subtree height selected during the inner cross-validations. These frequency counts are reported for each of the 3D structure datasets based on the kernels of Table 10.

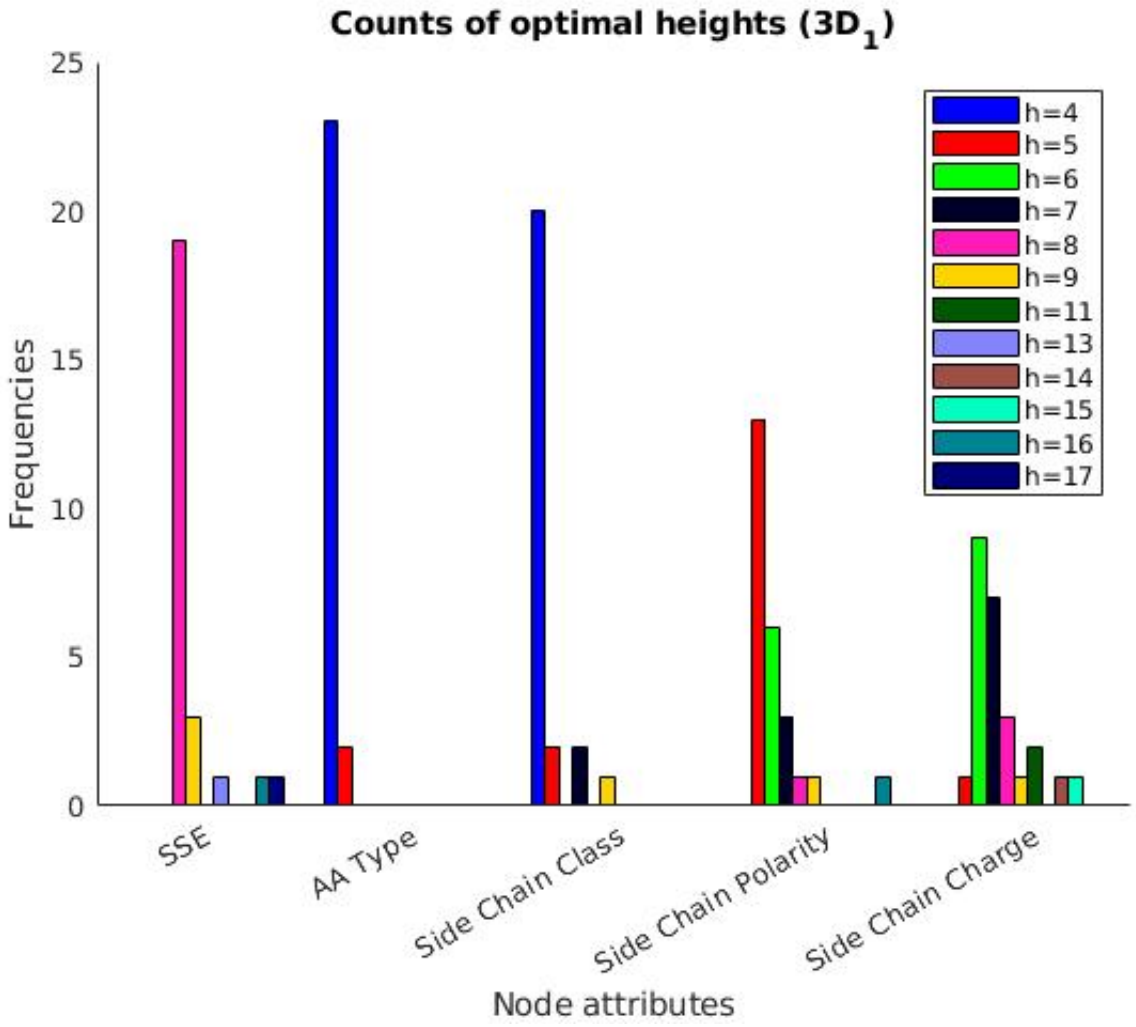

Figure 18: Counts of optimal heights h selected during the inner cross-validations for different node descriptors of the WL kernel.
Dataset: UniProt-unique-chains ($3D_1$).

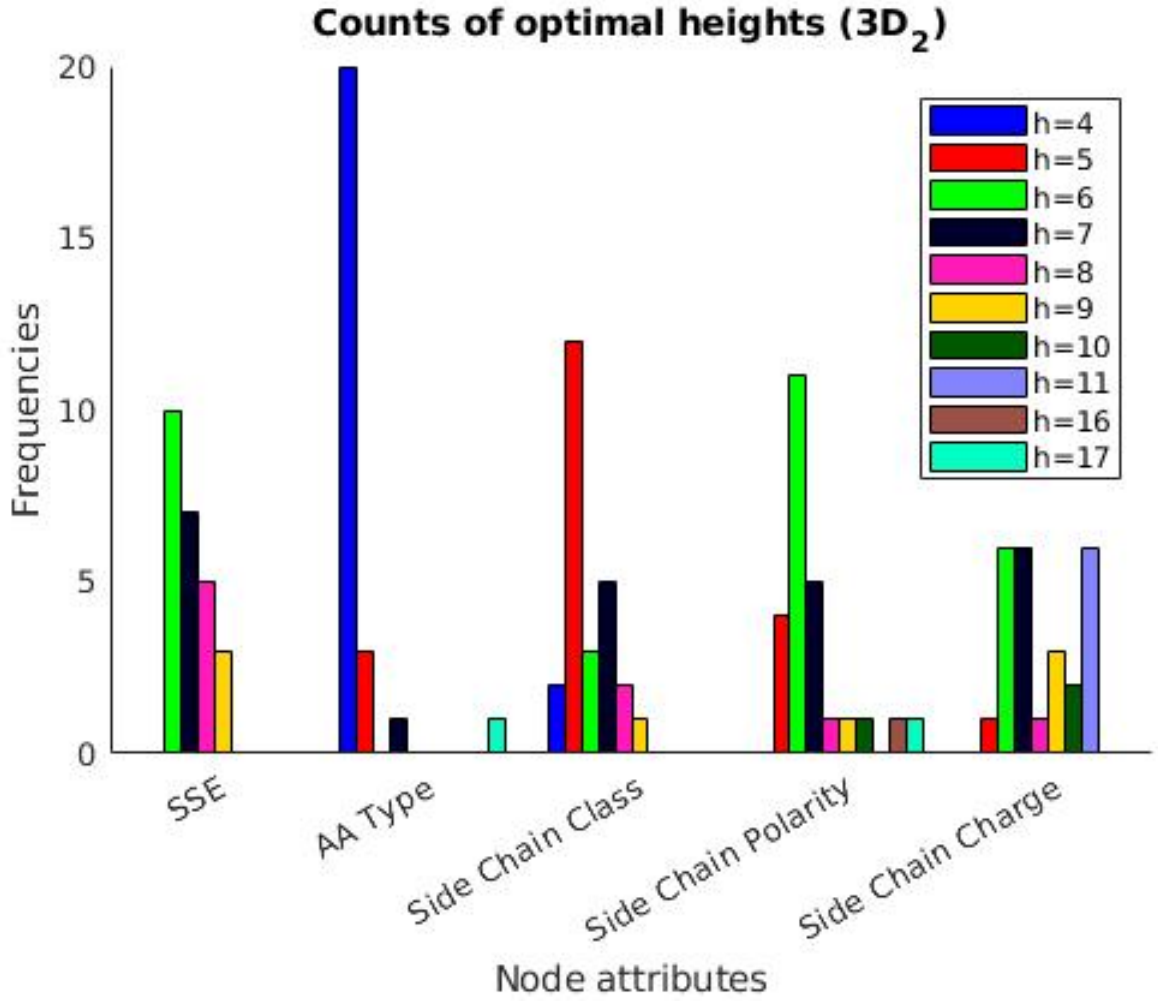

Figure 19: Counts of optimal heights h selected during the inner cross-validations for different node descriptors of the WL kernel.
Dataset: UniProt-all-chains ($3D_2$).

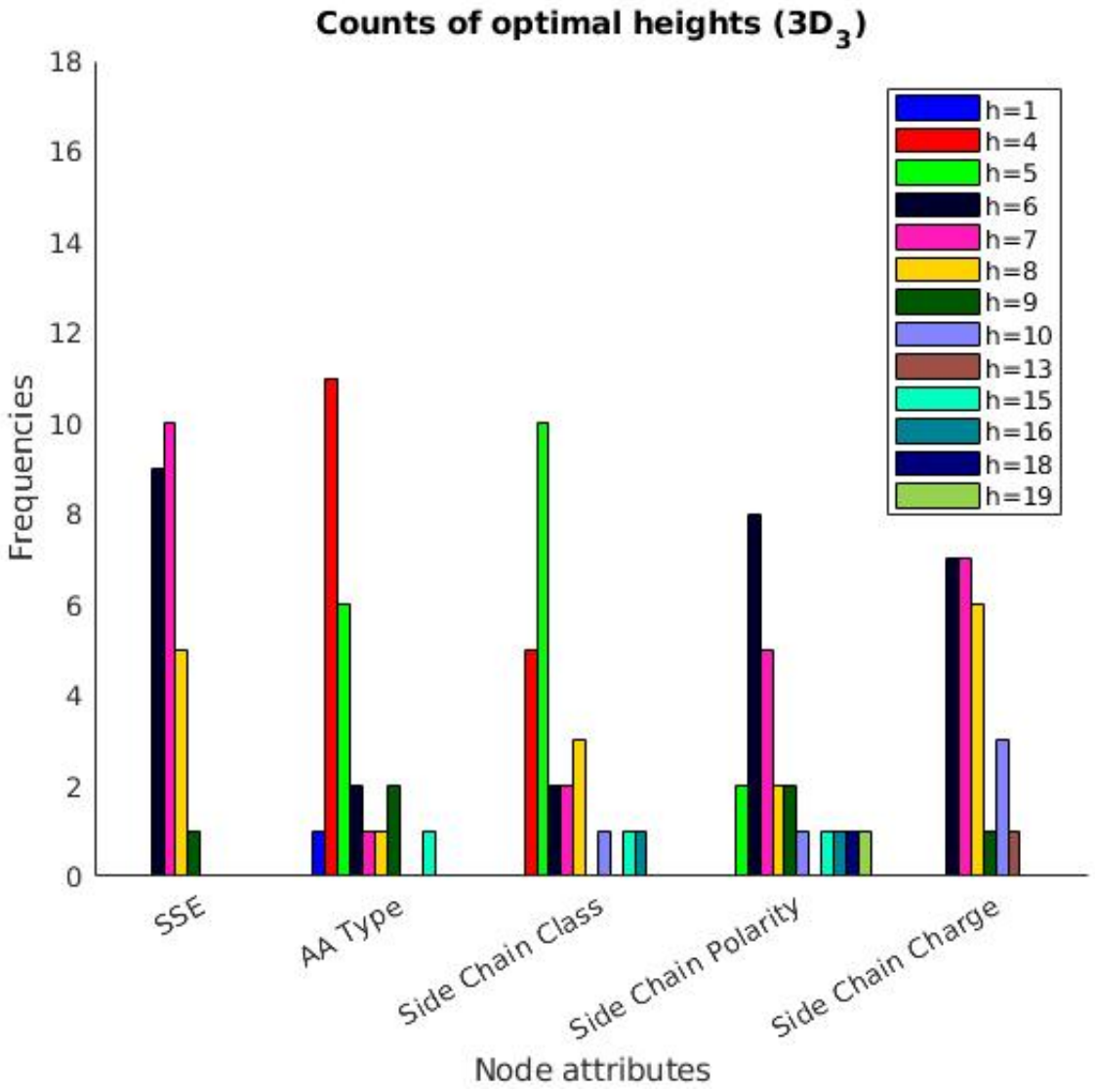

Figure 20: Counts of optimal heights h selected during the inner cross-validations for different node descriptors of the WL kernel.
Dataset: all-coordinates ($3D_3$).

Table 18 clearly shows that the WL kernel over the dataset $3D_1$ has a strong preference for heights h which are smaller than 9 and greater than 3. Graphs with secondary structure elements, which have achieved the best results, find height $h = 8$ to be the most informative, while graphs with other attributes have a tenancy to pick heights $h = \{4, 5, 6\}$.

In Table 19, we can observe that height selection preferences of the attributes are consistent with that of the $3D_1$ dataset, illustrated in Table 18. However, the SSE attributes in dataset $3D_2$, which have achieved a performance as high as the dataset $3D_1$, favor a height $h = 6$.

According to Table 20, dataset $3D_3$, which has led to the poorest performances compared to the datasets $3D_1$ and $3D_2$, similarly favors heights in the range 4 to 8. SSE attributes select the height $h = 7$, while AA type attributes consistently vote for $h = 4$. The observation in the side chain attributes is in line with that of dataset $3D_2$.

All the datasets exhibit consistent preferences as to what height h to select when different attributes are examined. The best height for SSE attributes is always in the range $h = \{6, 7, 8\}$. Amino acid type attributes always opt for a height $h = 4$. This finding has been observed also in the results of the k-spectrum kernel in Figure 14, where the k-mers of length 4 perform better, as well as in the results of the generic string kernel in Figure 15, where the amino acid type descriptors prefer a substring length of $L = 4$. This can imply that while the information from the types of the amino acids in the sequences might not help the classification of promiscuity much, substrings of length 4 seem to be carrying some information which cannot be directly inferred from other substring lengths. Moreover, side chain attributes' preferences for heights are always in the range $h = \{4, 5, 6, 7\}$ in all of the datasets.

5.3 Data Visualizations

In order to examine how well the abovementioned kernels have succeeded in recognizing similar patterns in each of the positive and negative classes, we attempted to visualize the separation of the classes in space using the multidimensional scaling (MDS) technique. MDS provides a visual representation of distances or dissimilarities between sets of objects as well as serving as a dimension reduction technique for high-dimensional data. The objective of MDS is to place each data point in an N-dimensional space such that actual distances between each pair of objects are preserved as well as possible. In the new N-dimensional space, data points which have shorter distances (or are more similar) are closer than those with less similarities. The MDS algorithm takes an $n \times n$ distance matrix of pairwise dissimilarities between n objects as input, and reconstructs a map that preserves the distances. Since the kernels can be interpreted as pairwise similarities between objects, we need to translate the kernel values into distances. To this end, for each pair of data points $(\mathbf{x}, \mathbf{x}')$, we have defined the distance $D(\mathbf{x}, \mathbf{x}')$ between them according to Equation 12. Inputting the D matrix into the MDS algorithm, we visualized 3-dimensional embeddings of the data for each of the previously discussed kernels with their optimal setting.However, since the graphlet kernel has only one non-zero

eigenvalue, the visualization is 1-dimensional. Furthermore, to detect meaningful underlying dimensions, we plotted the eigenvalues of each kernel in ascending order which can be found in Appendix C.

5.3.1 k-Spectrum Kernel Visualization

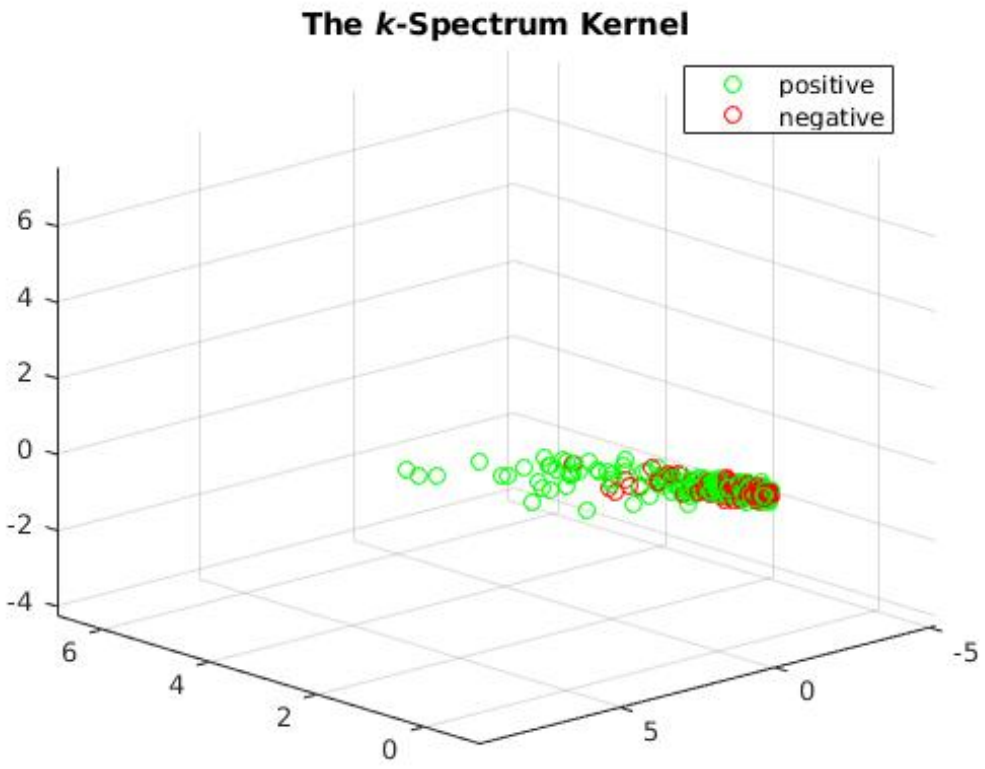

Figure 21: 3D visualization of the k-spectrum kernel for $k = 4$.

The 3D visualization of the k-spectrum kernel in Figure 21 shows that the classifier has not succeeded in separating the two classes of promiscuous and non-promiscuous proteins. However, the green tail of promiscuous proteins on the left-hand side of the figure validates the results obtained from Table 3 regarding a higher precision compared to the recall. This implies that the k-spectrum kernel can detect certain patterns in some of the promiscuous proteins, while not being able to generalize them to the other promiscuous proteins to make them distinguishable from non-promiscuous ones. This fact also accounts for an accuracy of 63.2% reported in Table 3.

5.3.2 Generic String Kernel Visualization

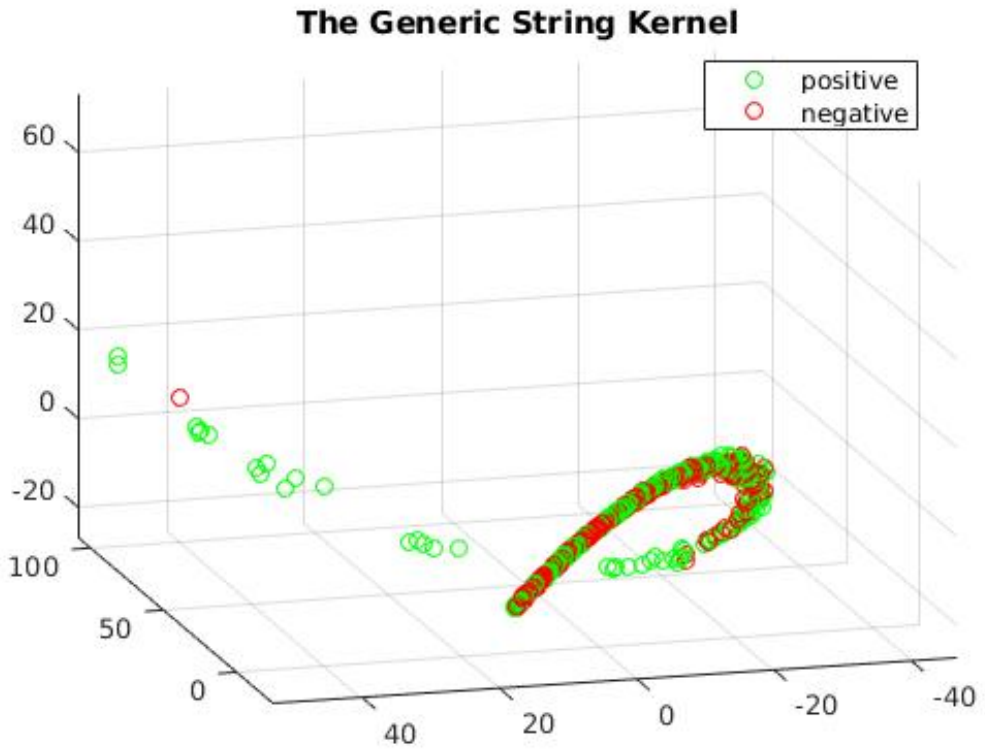

Figure 22: 3D visualization of the generic string kernel. AA descriptor = BLOSUM50, $L = 3$, $\sigma_p = 10$, $\sigma_c = 0.5$.

The findings from Figure 21 can be extended to the 3D visualization of the generic string kernel as well, depicted in Figure 22. Similarly, we can observe a green tail on the left side of the figure, which is in line with the higher value of the precision compared to the recall in Table 5 when BLOSUM50 substitution matrix is exploited. The accuracy of 63.4% in Table 5 correlates to this fact.

5.3.3 Graphlet Kernel Visualization

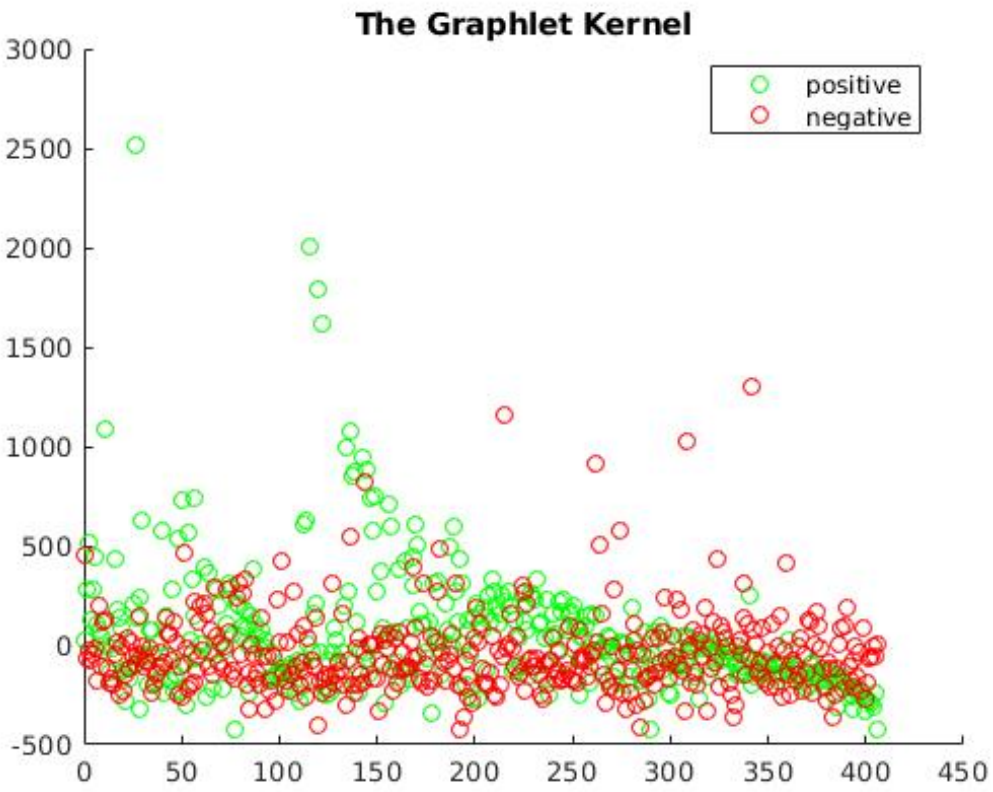

Figure 23: 1D visualization of the graphlet kernel. Dataset: $3D_1$, Node attribute: AA type.

The incompetency of the graphlet kernel reported in Table 7 can be reinforced by the visual illustration in Figure 23, where the kernel with AA type attributes has failed to separate the two classes. However, some promiscuous instances, shown in green, can be observed in the upper side of the figure, which can correspond to the higher value of the precision over the recall in Table 7. Similar to the kernel visualizations, this implies certain patterns found in some of the promiscuous enzymes, not being observed in others. The same phenomenon can be seen in the non-promiscuous instances, shown in red, as well. However, we can not verify this as we have not investigated the classification statistics of the negative class.

5.3.4 Weisfeiler-Lehman Subtree Kernel Visualization

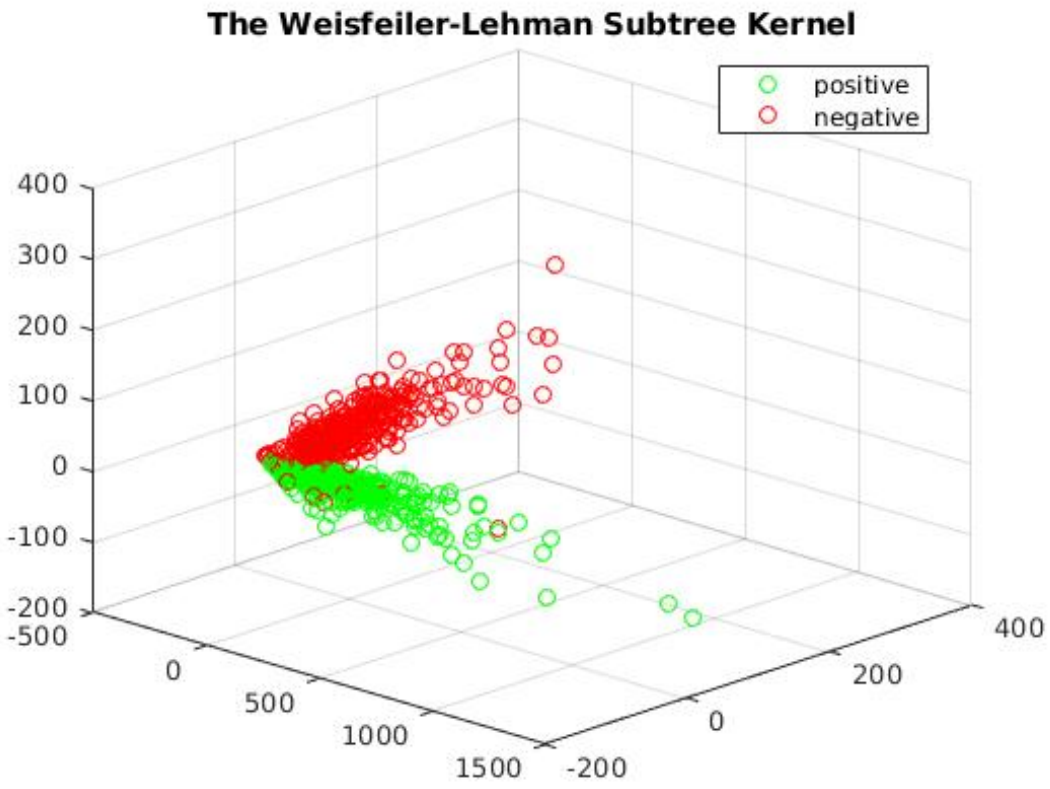

Figure 24: 3D visualization of the WL subtree kernel. Dataset: $3D_1$, Node attribute: SSE without the 'unknown' category, $h = 8$.

The 3D visualization of the Weisfeiler-Lehman subtree kernel in Figure 24 depicts a nearly perfect separation of the two classes, which verifies the remarkable performances achieved by the kernel in Table 10 when the SSE attributes are utilized.

6 Discussion

The main objective of this thesis was to design a rigorous binary classifier which could accurately predict the promiscuity status of an unseen protein as promiscuous or non-promiscuous. In order to investigate the determinants of promiscuity, we utilized two distinctive approaches: we leveraged the primary structures of proteins as well as their 3D structures. According to the nature of each type of protein structure, we fitted different models to examine their decisive powers in promiscuity classification. Exploiting the information extracted from the primary structure of proteins, or their sequences in other terms, we attempted to classify the proteins by means of the BLAST algorithm as a baseline. As other methods built based on the proteins' primary structures, string kernels were considered. We tried the k-spectrum kernel in addition to the generic string kernel which was evaluated for different amino acid descriptors. As our second approach, we modeled the 3D structure of proteins as graphs in which proteins' physicochemical and structural information were embedded as node labels. We constructed three different graph datasets according to the different ways we preprocessed the data. Two different graphlet kernels, namely the graphlet kernel and the Weisfeiler-Lehman subtree kernel were then employed for measuring the structural similarities of proteins. The classification inferences were made by SVM evaluations of the constructed string and graph kernels.

In the following, we present our findings and inferences for each of the approaches we investigated followed by future directions which can possibly benefit from these findings.

6.1 Sequence-Based Models

The evidence from examining different sequence-based methods suggests that, in general, we are not capable of making promising inferences in promiscuity classification exclusively by means of protein sequences. While the performances of all of the sequence-based methods are quite similar, the generic string kernel delivers slightly superior results compared to the BLAST algorithm and the k-spectrum kernel. In terms of the reliability of the classifier, which is quantitatively defined as precision, the k-spectrum and generic string kernels have achieved higher values compared to BLAST. The low accuracies and F1 scores in all of the sequence-based methods, however, indicate the poor performances of these classifiers.

The best performance statistics achieved by evaluating the protein sequences can be summarized as an accuracy of 63.4%, and F1 score of 0.67. In conclusion, it appears that inferring protein promiscuity with relying solely on proteins' sequences is hard.

6.2 3D Structure-Based Models

Assessing the proteins' 3D structures by means of the graphlet kernel unveils an overall incompetency of this kernel in making decisions about the proteins' labels. While enriching the graphlet kernel with node attributes manifests a slightly better accuracy, the enhancement is relatively insignificant. The results of the graphlet kernel are inferior to the sequence-based methods: 55.4% as the best accuracy, and 0.55 as the highest F1 score.

On the contrary, considerable progress has been made employing the Weisfeiler-Lehman subtree kernel. The WL kernel has outperformed all the previously-discussed methods by a large margin giving an account of its power in promiscuity classification. Observing a higher accuracy with the secondary structure elements provides further evidence that these structures can be the fundamental determinants of the promiscuity. This finding can be validated also by the improvement in the performance when those amino acids with unknown secondary structure attributes are excluded. Furthermore, this phenomenon shows that all of the attributes are interconnected up to some extent, meaning that structural patterns of promiscuity most probably have similar physicochemical properties, side chain classes and polarities in particular, as well.

As the main contribution of this thesis, we can refer to our data preprocessing whereby significantly better results are delivered. Acquiring the best performance via the $3D_1$, or the UniProt-unique-chains, dataset hints that first of all, useful information about promiscuity probably does not lie in the intersection of the chains. Moreover, in case of several identical chains, investigating the topological properties of only one copy of each distinctive chain suffices. This further suggests that full quaternary structures are not necessarily needed for inferring promiscuity.

In case of the $3D_2$, or the UniProt-all-chains, dataset similar results with the $3D_1$ dataset are achieved when secondary structure elements excluding the unknown category are used. This means that the whole quaternary structures can be as informative if the irrelevant parts are removed from the structures. However, this applies only to the secondary structure elements and not to the other attributes.

The poor performance of the $3D_3$, or the all-coordinates, dataset can be manifold. One possible justification is the fact that the crystal structures of proteins, provided by the PDB database, can also include the ligands or cofactors molecules which have been complexed with the proteins at the time of crystallography. The topological descriptors of these molecules can detrimentally affect the performance of the model in finding the actual patterns of the promiscuity. As another scenario, a protein macromolecule can include multiple active sites which can correspond to different UniProt IDs. However, one active site exhibiting promiscuous behaviors does not imply that all the other active sites have such property as well. Therefore, examining the whole structure of these macromolecules might inject so much unnecessary information to the model so that it fails to capture the actual determinants.

The best performance statistics achieved by the Weisfeiler-Lehman subtree kernel exploiting proteins' 3D structures are an accuracy of 96.3% and F1 score of 0.96. According to the superiority of these results, proteins' topological information manifests signs of encompassing promiscuity determinants.

6.3 Future Directions

Our results can cast a new light on the evolution and divergence of protein functions. Moreover, in our view, these results can constitute an excellent initial step towards designing *de novo* functions by employing patterns of promiscuity. However, the complete and correct set of such determinants has to be established by further investigation.

This research also has the potential to give rise to many other questions. We can probe the role of proteins' 3D structures in determining the catalytic functions of them as a relevant question. Moreover, it can be investigated whether this information finds applications also in studying non-enzymatic proteins such as hormones. And lastly, we can further study how effectively these findings can help in designing new drugs and enhancing the translational research.

References

[Afriat-Jurnou et al., 2012] Afriat-Jurnou, L., Jackson, C. J., and Tawfik, D. S. (2012). Reconstructing a missing link in the evolution of a recently diverged phosphotriesterase by active-site loop remodeling. *Biochemistry*, 51(31):6047–6055. PMID: 22809311.

[Alberts et al., 2002] Alberts, B., Bray, D., Lewis, J., Raff, M., Roberts, K., and Watson, J. (2002). *Molecular Biology of the Cell*. Garland, 4th edition.

[Altschul et al., 1990] Altschul, S. F., Gish, W., Miller, W., Myers, E. W., and Lipman, D. J. (1990). Basic local alignment search tool. *Journal of Molecular Biology*, 215(3):403 – 410.

[Altschul et al., 1997] Altschul, S. F., Madden, T. L., Schäffer, A. A., Zhang, J., Zhang, Z., Miller, W., and Lipman, D. J. (1997). Gapped blast and psi-blast: a new generation of protein database search programs. *Nucleic Acids Research*, 25(17):3389–3402.

[Anandarajah et al., 2000] Anandarajah, K., Kiefer, P. M., Donohoe, B. S., and Copley, S. D. (2000). Recruitment of a double bond isomerase to serve as a reductive dehalogenase during biodegradation of pentachlorophenol. *Biochemistry*, 39(18):5303–5311. PMID: 10820000.

[Babbitt and Gerlt, 1997] Babbitt, P. C. and Gerlt, J. A. (1997). Understanding enzyme superfamilies chemistry as the fundamental determinant in the evolution of new catalytic activities. *Journal of Biological Chemistry*, 272(49):30591–30594.

[Babtie et al., 2010] Babtie, A., Tokuriki, N., and Hollfelder, F. (2010). What makes an enzyme promiscuous? *Current Opinion in Chemical Biology*, 14(2):200 – 207. Biocatalysis and Biotransformation/Bioinorganic Chemistry.

[Baier et al., 2016] Baier, F., Copp, J. N., and Tokuriki, N. (2016). Evolution of enzyme superfamilies: Comprehensive exploration of sequence–function relationships. *Biochemistry*, 55(46):6375–6388. PMID: 27802036.

[Bairoch, 2000] Bairoch, A. (2000). The enzyme database in 2000. *Nucleic Acids Research*, 28(1):304–305.

[Berman et al., 2000] Berman, H. M., Westbrook, J., Feng, Z., Gilliland, G., Bhat, T. N., Weissig, H., Shindyalov, I. N., and Bourne, P. E. (2000). The protein data bank. *Nucleic Acids Research*, 28(1):235–242.

[Borgwardt and Stegle, 2010] Borgwardt, K. and Stegle, O. (2010). Computational approaches for analyzing complex biological systems (2010).

[Borgwardt and Kriegel, 2005] Borgwardt, K. M. and Kriegel, H. P. (2005). Shortest-path kernels on graphs. In *Fifth IEEE International Conference on Data Mining (ICDM'05)*, pages 8 pp.–.

[Borgwardt et al., 2005] Borgwardt, K. M., Ong, C. S., Schönauer, S., Vishwanathan, S. V. N., Smola, A. J., and Kriegel, H.-P. (2005). Protein function prediction via graph kernels. *Bioinformatics*, 21(suppl$_1$) : $i47 - -i56$.

[Borgwardt et al., 2007] Borgwardt, K. M., Petri, T., Vishwanathan, S. V. N., and Kriegel, H.-P. (2007). An efficient sampling scheme for comparison of large graphs. In *MLG*.

[C Whisstock and M Lesk, 2003] C Whisstock, J. and M Lesk, A. (2003). Prediction of protein function from protein sequence and structure. 36:307–40.

[Camacho et al., 2009] Camacho, C., Coulouris, G., Avagyan, V., Ma, N., and Papadopoulos, J. (2009). Blast+: architecture and applications. 10:1–9.

[Canutescu et al., 2003] Canutescu, A. A., Shelenkov, A. A., and Dunbrack, R. L. (2003). A graph-theory algorithm for rapid protein side-chain prediction. *Protein Science*, 12(9):2001–2014.

[Carbonell and Faulon, 2010] Carbonell, P. and Faulon, J.-L. (2010). Molecular signatures-based prediction of enzyme promiscuity. *Bioinformatics*, 26(16):2012–2019.

[Chakraborty and Rao, 2012] Chakraborty, S. and Rao, B. J. (2012). A measure of the promiscuity of proteins and characteristics of residues in the vicinity of the catalytic site that regulate promiscuity. *PLOS ONE*, 7(2):1–10.

[Copley, 2003] Copley, S. (2003). Copley, s. d. enzymes with extra talents: moonlighting functions and catalytic promiscuity. curr. opin. chem. biol. 7, 265-272. 7:265–72.

[Cortes and Vapnik, 1995] Cortes, C. and Vapnik, V. (1995). Support-vector networks. *Machine Learning*, 20(3):273–297.

[Cover and Hart, 1967] Cover, T. and Hart, P. (1967). Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13(1):21–27.

[Feragen et al., 2013] Feragen, A., Kasenburg, N., Petersen, J., de Bruijne, M., and Borgwardt, K. (2013). Scalable kernels for graphs with continuous attributes. In Burges, C. J. C., Bottou, L., Welling, M., Ghahramani, Z., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems 26*, pages 216–224. Curran Associates, Inc.

[Gerlt and Babbitt, 1998] Gerlt, J. A. and Babbitt, P. C. (1998). Mechanistically diverse enzyme superfamilies: the importance of chemistry in the evolution of catalysis. *Current Opinion in Chemical Biology*, 2(5):607 – 612.

[Gerlt and Babbitt, 2001] Gerlt, J. A. and Babbitt, P. C. (2001). Divergent evolution of enzymatic function: Mechanistically diverse superfamilies and functionally distinct suprafamilies. *Annual Review of Biochemistry*, 70(1):209–246. PMID: 11395407.

[Giguère et al., 2013] Giguère, S., Marchand, M., Laviolette, F., Drouin, A., and Corbeil, J. (2013). Learning a peptide-protein binding affinity predictor with kernel ridge regression. *BMC Bioinformatics*, 14(1):82.

[Gulbis et al., 1996] Gulbis, J. M., Kelman, Z., Hurwitz, J., O'Donnell, M., and Kuriyan, J. (1996). Structure of the c-terminal region of p21waf1/cip1 complexed with human pcna. *Cell*, 87(2):297 – 306.

[Gärtner et al., 2003] Gärtner, T., Flach, P., and Wrobel, S. (2003). On graph kernels: Hardness results and efficient alternatives. In *IN: CONFERENCE ON LEARNING THEORY*, pages 129–143.

[Haussler, 1999] Haussler, D. (1999). Convolution kernels on discrete structures. Technical Report UCS-CRL-99-10, University of California at Santa Cruz, Santa Cruz, CA, USA.

[Horváth et al., 2004] Horváth, T., Gärtner, T., and Wrobel, S. (2004). Cyclic pattern kernels for predictive graph mining. In *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '04, pages 158–167, New York, NY, USA. ACM.

[Huan et al., 2005] Huan, J., Bandyopadhyay, D., Wang, W., Snoeyink, J., Prins, J., and Tropsha, A. (2005). Comparing graph representations of protein structure for mining family-specific residue-based packing motifs. *Journal of Computational Biology*, 12(6):657–671. PMID: 16108709.

[Huang et al., 2010] Huang, Y., Niu, B., Gao, Y., Fu, L., and Li, W. (2010). Cd-hit suite: a web server for clustering and comparing biological sequences. *Bioinformatics*, 26(5):680–682.

[Hult and Berglund, 2007] Hult, K. and Berglund, P. (2007). Enzyme promiscuity: mechanism and applications. *Trends in Biotechnology*, 25(5):231 – 238.

[Jackson and Handschumacher, 1970] Jackson, R. C. and Handschumacher, R. E. (1970). Escherichia coli l-asparaginase. catalytic activity and subunit nature. *Biochemistry*, 9(18):3585–3590. PMID: 4928351.

[Jacob, 1977] Jacob, F. (1977). Evolution and tinkering. *Science*, 196(4295):1161–1166.

[Jeffery, 1999] Jeffery, C. J. (1999). Moonlighting proteins. *Trends in Biochemical Sciences*, 24(1):8 – 11.

[Jensen, 1976] Jensen, R. A. (1976). Enzyme recruitment in evolution of new function. *Annual Review of Microbiology*, 30(1):409–425. PMID: 791073.

[Kashima et al., 2003] Kashima, H., Tsuda, K., and Inokuchi, A. (2003). Marginalized kernels between labeled graphs. In *Proceedings of the Twentieth International Conference on International Conference on Machine Learning*, ICML'03, pages 321–328. AAAI Press.

[Kazlauskas, 2005] Kazlauskas, R. J. (2005). Enhancing catalytic promiscuity for biocatalysis. *Current Opinion in Chemical Biology*, 9(2):195 – 201. Bioiorganic chemistry / Biocatalysis and biotransformation.

[Khersonsky et al., 2006] Khersonsky, O., Roodveldt, C., and Tawfik, D. S. (2006). Enzyme promiscuity: evolutionary and mechanistic aspects. *Current Opinion in Chemical Biology*, 10(5):498 – 508. Analytical techniques / Mechanisms.

[Khersonsky and Tawfik, 2010] Khersonsky, O. and Tawfik, D. S. (2010). Enzyme promiscuity: A mechanistic and evolutionary perspective. *Annual Review of Biochemistry*, 79(1):471–505. PMID: 20235827.

[Leslie et al., 2002] Leslie, C., Eskin, E., and Stafford Noble, W. (2002). The spectrum kernel: A string kernel for svm protein classification. 7:564–75.

[Lodhi et al., 2002] Lodhi, H., Saunders, C., Shawe-Taylor, J., Cristianini, N., and Watkins, C. (2002). Text classification using string kernels. *J. Mach. Learn. Res.*, 2:419–444.

[Mahé et al., 2004] Mahé, P., Ueda, N., Akutsu, T., Perret, J.-L., and Vert, J.-P. (2004). Extensions of marginalized graph kernels. In *Proceedings of the Twenty-first International Conference on Machine Learning*, ICML '04, pages 70–, New York, NY, USA. ACM.

[Mahé and Vert, 2009] Mahé, P. and Vert, J.-P. (2009). Graph kernels based on tree patterns for molecules. *Machine Learning*, 75(1):3–35.

[Nakagawa and Bender, 1969] Nakagawa, Y. and Bender, M. L. (1969). Modification of .alpha.-chymotrypsin by methyl p-nitrobenzenesulfonate. *Journal of the American Chemical Society*, 91(6):1566–1567.

[Nath and Atkins, 2008] Nath, A. and Atkins, W. M. (2008). A quantitative index of substrate promiscuity. *Biochemistry*, 47(1):157–166. PMID: 18081310.

[Neal, 1996] Neal, R. M. (1996). *Bayesian Learning for Neural Networks*. Springer-Verlag New York, Inc., Secaucus, NJ, USA.

[Neumann et al., 2012] Neumann, M., Patricia, N., Garnett, R., and Kersting, K. (2012). Efficient graph kernels by randomization. In Flach, P. A., De Bie, T., and Cristianini, N., editors, *Machine Learning and Knowledge Discovery in Databases*, pages 378–393, Berlin, Heidelberg. Springer Berlin Heidelberg.

[Nobeli et al., 2009] Nobeli, I., D Favia, A., and Thornton, J. (2009). Protein promiscuity and its implications for biotechnology. 27:157–67.

[Oberhardt et al., 2016] Oberhardt, M. A., Zarecki, R., Reshef, L., Xia, F., Duran-Frigola, M., Schreiber, R., Henry, C. S., Ben-Tal, N., Dwyer, D. J., Gophna, U., and Ruppin, E. (2016). Systems-wide prediction of enzyme promiscuity reveals a new underground alternative route for pyridoxal 5'-phosphate production in e. coli. *PLOS Computational Biology*, 12(1):1–19.

[O'Brien and Herschlag, 1999] O'Brien, P. J. and Herschlag, D. (1999). Catalytic promiscuity and the evolution of new enzymatic activities. *Chemistry Biology*, 6(4):R91 – R105.

[OpenStax CNX,] OpenStax CNX. OpenStax Microbiology, Microbiology. https://github.com/BorgwardtLab/graph-kernels.git. Online; accessed 22-April-2018.

[Patrick et al., 2007] Patrick, W. M., Quandt, E. M., Swartzlander, D. B., and Matsumura, I. (2007). Multicopy suppression underpins metabolic evolvability. *Molecular Biology and Evolution*, 24(12):2716–2722.

[Peisajovich and Tawfik, 2007] Peisajovich, S. and Tawfik, D. S. (2007). Protein engineers turned evolutionists. *Nature Methods*, 4:991–994.

[Peng and Tsay, 2010] Peng, S.-L. and Tsay, Y.-W. (2010). Measuring protein structural similarity by maximum common edge subgraphs. In *Proceedings of the Advanced Intelligent Computing Theories and Applications, and 6th International Conference on Intelligent Computing*, ICIC'10, pages 100–107, Berlin, Heidelberg. Springer-Verlag.

[Peng and Tsay, 2014] Peng, S.-L. and Tsay, Y.-W. (2014). Adjusting protein graphs based on graph entropy. *BMC Bioinformatics*, 15(15):S6.

[Piedrafita et al., 2015] Piedrafita, G., Keller, M. A., and Ralser, M. (2015). The impact of non-enzymatic reactions and enzyme promiscuity on cellular metabolism during (oxidative) stress conditions. *Biomolecules*, 5(3):2101–2122.

[Pocker and Stone, 1967] Pocker, Y. and Stone, J. T. (1967). The catalytic versatility of erythrocyte carbonic anhydrase. iii. kinetic studies of the enzyme-catalyzed hydrolysis of p-nitrophenyl acetate*. *Biochemistry*, 6(3):668–678. PMID: 4960944.

[Pržulj, 2007] Pržulj, N. (2007). Biological network comparison using graphlet degree distribution. *Bioinformatics*, 23(2):e177–e183.

[Quinlan, 1986] Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning*, 1(1):81–106.

[Ramon and Gärtner, 2003] Ramon, J. and Gärtner, T. (2003). Expressivity versus efficiency of graph kernels. In *Proceedings of the First International Workshop on Mining Graphs, Trees and Sequences*, pages 65–74.

[Reid and Fahrney, 1967] Reid, T. W. and Fahrney, D. (1967). The pepsin-catalyzed hydrolysis of sulfite esters. *Journal of the American Chemical Society*, 89(15):3941–3943. PMID: 5343112.

[Samudrala and Moult, 1998] Samudrala, R. and Moult, J. (1998). A graph-theoretic algorithm for comparative modeling of protein structure11edited by f. cohen. *Journal of Molecular Biology*, 279(1):287 – 302.

[Seffernick et al., 2001] Seffernick, J., de Souza, M., Sadowsky, M., and Wackett, L. (2001). Melamine deaminase and atrazine chlorohydrolase: 98 percent identical but functionally different. *Journal of bacteriology*, 183(8):2405—2410.

[Shawe-Taylor and Cristianini, 2004] Shawe-Taylor, J. and Cristianini, N. (2004). *Kernel Methods for Pattern Analysis*. Cambridge University Press, New York, NY, USA.

[Shervashidze and Borgwardt, 2009] Shervashidze, N. and Borgwardt, K. M. (2009). Fast subtree kernels on graphs. In *Proceedings of the 22Nd International Conference on Neural Information Processing Systems*, NIPS'09, pages 1660–1668, USA. Curran Associates Inc.

[Shervashidze et al., 2011] Shervashidze, N., Schweitzer, P., van Leeuwen, E. J., Mehlhorn, K., and Borgwardt, K. M. (2011). Weisfeiler-lehman graph kernels. *J. Mach. Learn. Res.*, 12:2539–2561.

[Shervashidze et al., 2009] Shervashidze, N., Vishwanathan, S., Petri, T., Mehlhorn, K., and Borgwardt, K. (2009). Efficient graphlet kernels for large graph comparison. In van Dyk, D. and Welling, M., editors, *Proceedings of the Twelth International Conference on Artificial Intelligence and Statistics*, volume 5 of *Proceedings of Machine Learning Research*, pages 488–495, Hilton Clearwater Beach Resort, Clearwater Beach, Florida USA. PMLR.

[Steinkellner et al., 2014] Steinkellner, G., Gruber, C., Pavkov-Keller, T., Binter, A., Steiner, K., Winkler, C., Lyskowski, A., Schwamberger, O., Oberer, M., Schwab, H., Faber, K., Macheroux, P., and Gruber, K. (2014). Identification of promiscuous ene-reductase activity by mining structural databases using active site constellations. *Nature Communications*, 5(4150):1–9.

[The UniProt Consortium, 2017] The UniProt Consortium (2017). Uniprot: the universal protein knowledgebase. *Nucleic Acids Research*, 45(D1):D158–D169.

[Turner, 2009] Turner, N. (2009). Directed evolution drives the next generation of biocatalysts. 5:567–73.

[V. N. Vishwanathan and Smola, 2003] V. N. Vishwanathan, S. and Smola, A. (2003). Fast kernels for string and tree matching. 15.

[Vapnik, 1995] Vapnik, V. N. (1995). *The Nature of Statistical Learning Theory*. Springer-Verlag New York, Inc., New York, NY, USA.

[Vishwanathan et al., 2006] Vishwanathan, S. V. N., Borgwardt, K. M., and Schraudolph, N. N. (2006). Fast computation of graph kernels. In *In NIPS 19*.

[Watkins, 1999] Watkins, C. (1999). Dynamic alignment kernels. In *Advances in Large Margin Classifiers*, pages 39–50. MIT Press.

[Williams and Rasmussen, 1996] Williams, C. K. I. and Rasmussen, C. E. (1996). Gaussian processes for regression. In Touretzky, D. S., Mozer, M. C., and Hasselmo, M. E., editors, *Advances in Neural Information Processing Systems 8*, pages 514–520. MIT Press.

Appendices

A Categorical Features

AA Type = { **Ala**, **Arg**, **Asn**, **Asp**, **Cys**, **Gln**, **Glu**, **Gly**, **His**, **Ile**, **Leu**, **Lys**, **Met**, **Phe**, **Pro**, **Ser**, **Thr**, **Trp**, **Tyr**, **Val** }

Side chain class = { **acid**, **aliphatic**, **amide**, **aromatic**, **basic**, **basic aromatic**, **cyclic**, **hydroxyl-containing**, **sulfur-containing** }

Side chain polarity = { **acidic polar**, **basic polar**, **nonpolar**, **polar** }

Side chain charge = { **negative**, **neutral**, **positive** }

Secondary structure elements = { **helix**, **sheet**, **unknown** }

B Vectorial Representations of Features

	Ala	Arg	Asn	Asp	Cys	Gln	Glu	Gly	His	Ile	Ieu	Lys	Met	Phe	Pro	Ser	Thr	Trp	Tyr	Val	Asx	Glx	Xaa	END
Ala	5	-2	-1	-2	-1	-1	-1	0	-2	-1	-2	-1	-1	-3	-1	1	0	-3	-2	0	-2	-1	-1	-5
Arg	-2	7	-1	-2	-4	1	0	-3	0	-4	-3	3	-2	-3	-3	-1	-1	-3	-1	-3	-1	0	-1	-5
Asn	-1	-1	7	2	-2	0	0	0	1	-3	-4	0	-2	-4	-2	1	0	-4	-2	-3	4	0	-1	-5
Asp	-2	-2	2	8	-4	0	2	-1	-1	-4	-4	-1	-4	-5	-1	0	-1	-5	-3	-4	5	1	-1	-5
Cys	-1	-4	-2	-4	13	-3	-3	-3	-3	-2	-2	-3	-2	-2	-4	-1	-1	-5	-3	-1	-3	-3	-2	-5
Gln	-1	1	0	0	-3	7	2	-2	1	-3	-2	2	0	-4	-1	0	-1	-1	-1	-3	0	4	-1	-5
Glu	-1	0	0	2	-3	2	6	-3	0	-4	-3	1	-2	-3	-1	-1	-1	-3	-2	-3	1	5	-1	-5
Gly	0	-3	0	-1	-3	-2	-3	8	-2	-4	-4	-2	-3	-4	-2	0	-2	-3	-3	-4	-1	-2	-2	-5
His	-2	0	1	-1	-3	1	0	-2	10	-4	-3	0	-1	-1	-2	-1	-2	-3	2	-4	0	0	-1	-5
Ile	-1	-4	-3	-4	-2	-3	-4	-4	-4	5	2	-3	2	0	-3	-3	-1	-3	-1	4	-4	-3	-1	-5
Leu	-2	-3	-4	-4	-2	-2	-3	-4	-3	2	5	-3	3	1	-4	-3	-1	-2	-1	1	-4	-3	-1	-5
Lys	-1	3	0	-1	-3	2	1	-2	0	-3	-3	6	-2	-4	-1	0	-1	-3	-2	-3	0	1	-1	-5
Met	-1	-2	-2	-4	-2	0	-2	-3	-1	2	3	-2	7	0	-3	-2	-1	-1	0	1	-3	-1	-1	-5
Phe	-3	-3	-4	-5	-2	-4	-3	-4	-1	0	1	-4	0	8	-4	-3	-2	1	4	-1	-4	-4	-2	-5
Pro	-1	-3	-2	-1	-4	-1	-1	-2	-2	-3	-4	-1	-3	-4	10	-1	-1	-4	-3	-3	-2	-1	-2	-5
Ser	1	-1	1	0	-1	0	-1	0	-1	-3	-3	0	-2	-3	-1	5	2	-4	-2	-2	0	0	-1	-5
Thr	0	-1	0	-1	-1	-1	-1	-2	-2	-1	-1	-1	-1	-2	-1	2	5	-3	-2	0	0	-1	0	-5
Trp	-3	-3	-4	-5	-5	-1	-3	-3	-3	-3	-2	-3	-1	1	-4	-4	-3	15	2	-3	-5	-2	-3	-5
Tyr	-2	-1	-2	-3	-3	-1	-2	-3	2	-1	-1	-2	0	4	-3	-2	-2	2	8	-1	-3	-2	-1	-5
Val	0	-3	-3	-4	-1	-3	-3	-4	-4	4	1	-3	1	-1	-3	-2	0	-3	-1	5	-4	-3	-1	-5
Asx	-2	-1	4	5	-3	0	1	-1	0	-4	-4	0	-3	-4	-2	0	0	-5	-3	-4	5	2	-1	-5
Glx	-1	0	0	1	-3	4	5	-2	0	-3	-3	1	-1	-4	-1	0	-1	-2	-2	-3	2	5	-1	-5
Xaa	-1	-1	-1	-1	-2	-1	-1	-2	-1	-1	-1	-1	-1	-2	-2	-1	0	-3	-1	-1	-1	-1	-1	-5
END	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	1

Table 11: BLOSUM50 substitution matrix

	Ala	Arg	Asn	Asp	Cys	Gln	Glu	Gly	His	Ile	Ieu	Lys	Met	Phe	Pro	Ser	Thr	Trp	Tyr	Val	Asx	Glx	Xaa	END
Ala	4	-1	-2	-2	0	-1	-1	0	-2	-1	-1	-1	-1	-2	-1	1	0	-3	-2	0	-2	-1	0	-4
Arg	-1	5	0	-2	-3	1	0	-2	0	-3	-2	2	-1	-3	-2	-1	-1	-3	-2	-3	-1	0	-1	-4
Asn	-2	0	6	1	-3	0	0	0	1	-3	-3	0	-2	-3	-2	1	0	-4	-2	-3	3	0	-1	-4
Asp	-2	-2	1	6	-3	0	2	-1	-1	-3	-4	-1	-3	-3	-1	0	-1	-4	-3	-3	4	1	-1	-4
Cys	0	-3	-3	-3	9	-3	-4	-3	-3	-1	-1	-3	-1	-2	-3	-1	-1	-2	-2	-1	-3	-3	-2	-4
Gln	-1	1	0	0	-3	5	2	-2	0	-3	-2	1	0	-3	-1	0	-1	-2	-1	-2	0	3	-1	-4
Glu	-1	0	0	2	-4	2	5	-2	0	-3	-3	1	-2	-3	-1	0	-1	-3	-2	-2	1	4	-1	-4
Gly	0	-2	0	-1	-3	-2	-2	6	-2	-4	-4	-2	-3	-3	-2	0	-2	-2	-3	-3	-1	-2	-1	-4
His	-2	0	1	-1	-3	0	0	-2	8	-3	-3	-1	-2	-1	-2	-1	-2	-2	2	-3	0	0	-1	-4
Ile	-1	-3	-3	-3	-1	-3	-3	-4	-3	4	2	-3	1	0	-3	-2	-1	-3	-1	3	-3	-3	-1	-4
Leu	-1	-2	-3	-4	-1	-2	-3	-4	-3	2	4	-2	2	0	-3	-2	-1	-2	-1	1	-4	-3	-1	-4
Lys	-1	2	0	-1	-3	1	1	-2	-1	-3	-2	5	-1	-3	-1	0	-1	-3	-2	-2	0	1	-1	-4
Met	-1	-1	-2	-3	-1	0	-2	-3	-2	1	2	-1	5	0	-2	-1	-1	-1	-1	1	-3	-1	-1	-4
Phe	-2	-3	-3	-3	-2	-3	-3	-3	-1	0	0	-3	0	6	-4	-2	-2	1	3	-1	-3	-3	-1	-4
Pro	-1	-2	-2	-1	-3	-1	-1	-2	-2	-3	-3	-1	-2	-4	7	-1	-1	-4	-3	-2	-2	-1	-2	-4
Ser	1	-1	1	0	-1	0	0	0	-1	-2	-2	0	-1	-2	-1	4	1	-3	-2	-2	0	0	0	-4
Thr	0	-1	0	-1	-1	-1	-1	-2	-2	-1	-1	-1	-1	-2	-1	1	5	-2	-2	0	-1	-1	0	-4
Trp	-3	-3	-4	-4	-2	-2	-3	-2	-2	-3	-2	-3	-1	1	-4	-3	-2	11	2	-3	-4	-3	-2	-4
Tyr	-2	-2	-2	-3	-2	-1	-2	-3	2	-1	-1	-2	-1	3	-3	-2	-2	2	7	-1	-3	-2	-1	-4
Val	0	-3	-3	-3	-1	-2	-2	-3	-3	3	1	-2	1	-1	-2	-2	0	-3	-1	4	-3	-2	-1	-4
Asx	-2	-1	3	4	-3	0	1	-1	0	-3	-4	0	-3	-3	-2	0	-1	-4	-3	-3	4	1	-1	-4
Glx	-1	0	0	1	-3	3	4	-2	0	-3	-3	1	-1	-3	-1	0	-1	-3	-2	-2	1	4	-1	-4
Xaa	0	-1	-1	-1	-2	-1	-1	-1	-1	-1	-1	-1	-1	-1	-2	0	0	-2	-1	-1	-1	-1	-1	-4
END	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	1

Table 12: BLOSUM62 substitution matrix

	Ala	Arg	Asn	Asp	Cys	Gln	Glu	Gly	His	Ile	Ieu	Lys	Met	Phe	Pro	Ser	Thr	Trp	Tyr	Val	Asx	Glx	Xaa	END
Ala	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Arg	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Asn	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Asp	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Cys	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Gln	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Glu	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Gly	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
His	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Ile	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Leu	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
Lys	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
Met	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
Phe	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
Pro	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
Ser	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
Thr	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
Trp	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
Tyr	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
Val	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
Asx	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
Glx	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
Xaa	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
END	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Table 13: One hot encoding of AA types

	acid	aliphatic	amide	aromatic	basic	basic aromatic	cyclic	hydroxyl-containing	sulfur-containing
Ala	0	1	0	0	0	0	0	0	0
Arg	0	0	0	0	1	0	0	0	0
Asn	0	0	1	0	0	0	0	0	0
Asp	1	0	0	0	0	0	0	0	0
Cys	0	0	0	0	0	0	0	0	1
Gln	0	0	1	0	0	0	0	0	0
Glu	1	0	0	0	0	0	0	0	0
Gly	0	1	0	0	0	0	0	0	0
His	0	0	0	0	0	1	0	0	0
Ile	0	1	0	0	0	0	0	0	0
Leu	0	1	0	0	0	0	0	0	0
Lys	0	0	0	0	1	0	0	0	0
Met	0	0	0	0	0	0	0	0	1
Phe	0	0	0	1	0	0	0	0	0
Pro	0	0	0	0	0	0	1	0	0
Ser	0	0	0	0	0	0	0	1	0
Thr	0	0	0	0	0	0	0	1	0
Trp	0	0	0	1	0	0	0	0	0
Tyr	0	0	0	1	0	0	0	0	0
Val	0	1	0	0	0	0	0	0	0

Table 14: One hot encoding of side chain classes

	acidic polar	basic polar	nonpolar	polar
Ala	0	0	1	0
Arg	0	1	0	0
Asn	0	0	0	1
Asp	1	0	0	0
Cys	0	0	1	0
Gln	0	0	0	1
Glu	1	0	0	0
Gly	0	0	1	0
His	0	1	0	0
Ile	0	0	1	0
Leu	0	0	1	0
Lys	0	1	0	0
Met	0	0	1	0
Phe	0	0	1	0
Pro	0	0	1	0
Ser	0	0	0	1
Thr	0	0	0	1
Trp	0	0	1	0
Tyr	0	0	0	1
Val	0	0	1	0

Table 15: One hot encoding of side chain polarities

	negative	neutral	positive
Ala	0	1	0
Arg	0	0	1
Asn	0	1	0
Asp	1	0	0
Cys	0	1	0
Gln	0	1	0
Glu	1	0	0
Gly	0	1	0
His	0	1	0
Ile	0	1	0
Leu	0	1	0
Lys	0	0	1
Met	0	1	0
Phe	0	1	0
Pro	0	1	0
Ser	0	1	0
Thr	0	1	0
Trp	0	1	0
Tyr	0	1	0
Val	0	1	0

Table 16: One hot encoding of side chain charges

C Plots of Eigenvalues

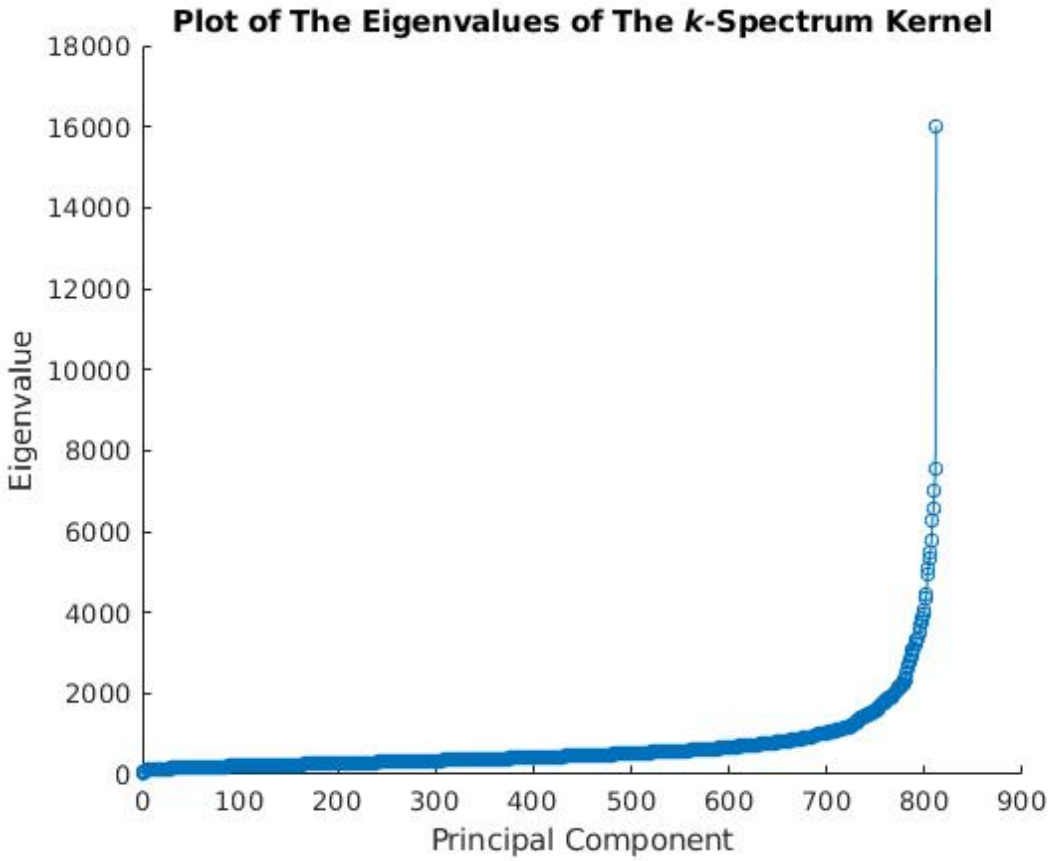

Figure 25: Plot of the eigenvalues of the k-spectrum kernel for $k = 4$.

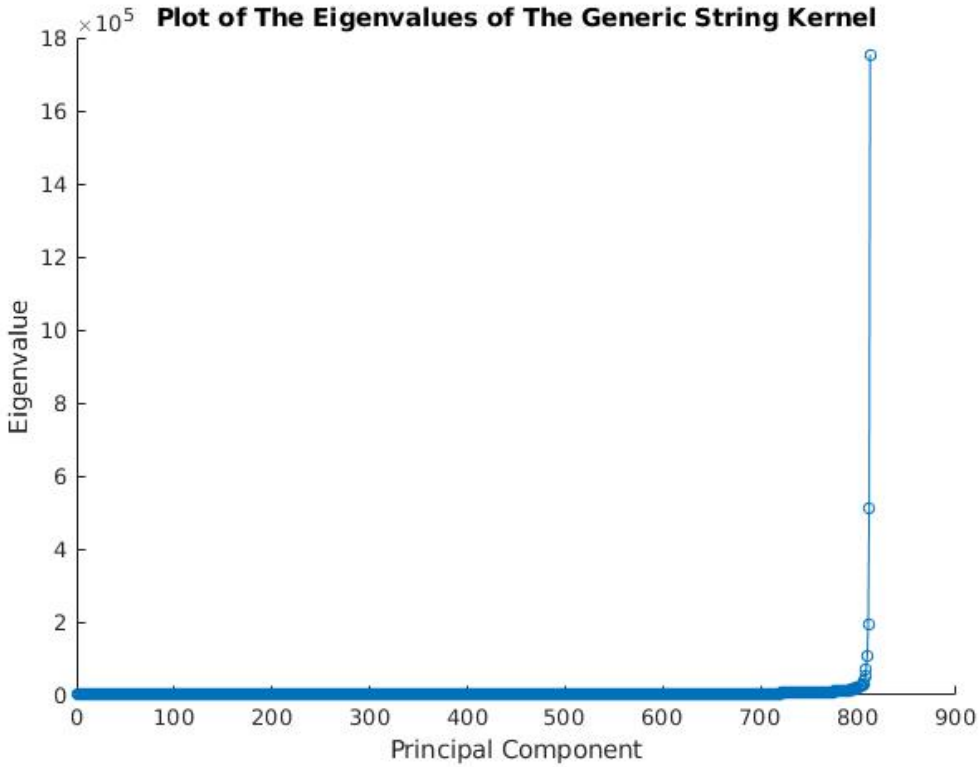

Figure 26: Plot of the eigenvalues of the generic string kernel. AA descriptor = BLOSUM50, $L = 3$, $\sigma_p = 10$, $\sigma_c = 0.5$.

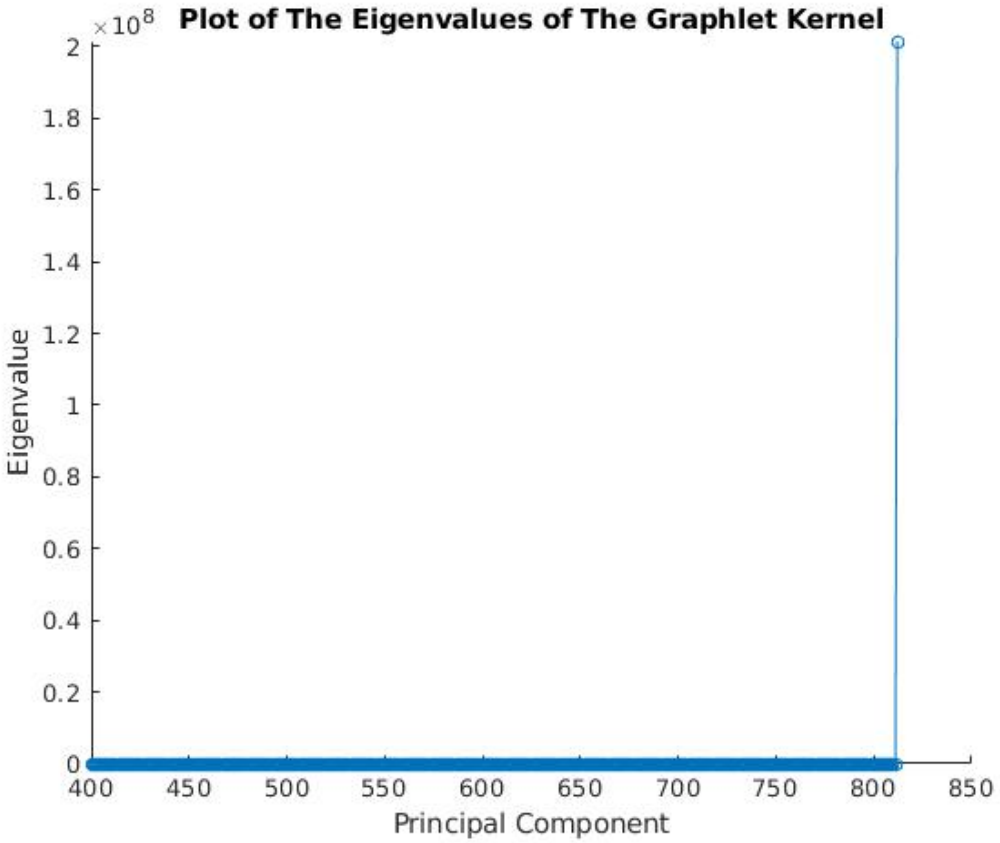

Figure 27: Plot of the eigenvalues of the graphlet kernel. Dataset: $3D_1$, Node attribute: AA type

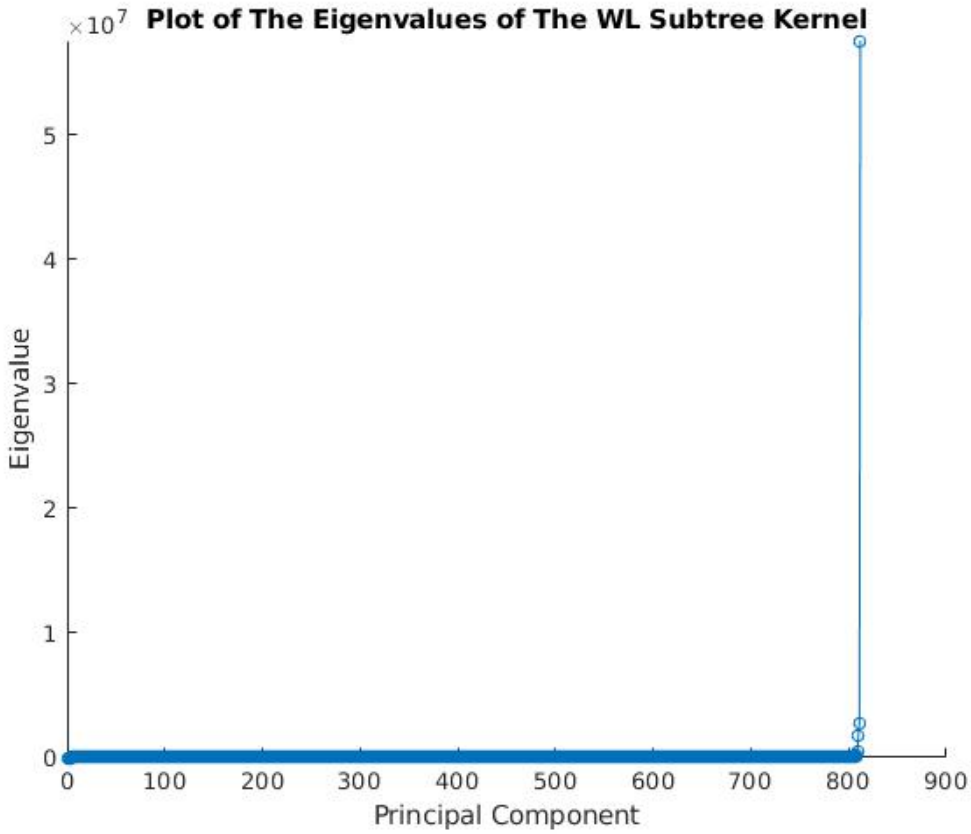

Figure 28: Plot of the eigenvalues of the WL subtree kernel. Dataset: $3D_1$, Node attribute: SSE without the 'unknown' category, $h = 8$.