

University of Southampton Research Repository ePrints Soton

Copyright © and Moral Rights for this thesis are retained by the author and/or other copyright owners. A copy can be downloaded for personal non-commercial research or study, without prior permission or charge. This thesis cannot be reproduced or quoted extensively from without first obtaining permission in writing from the copyright holder/s. The content must not be changed in any way or sold commercially in any format or medium without the formal permission of the copyright holders.

When referring to this work, full bibliographic details including the author, title, awarding institution and date of the thesis must be given e.g.

AUTHOR (year of submission) "Full thesis title", University of Southampton, name of the University School or Department, PhD Thesis, pagination

UNIVERSITY OF SOUTHAMPTON

**Graph Kernel Extensions and
Experiments with Application to
Molecule Classification, Lead Hopping
and Multiple Targets**

by

Anthony A. Demco

A thesis submitted in partial fulfillment for the
degree of Doctor of Philosophy

in the
Faculty of Engineering, Science and Mathematics
School of Electronics and Computer Science

February, 2009

UNIVERSITY OF SOUTHAMPTON

ABSTRACT

FACULTY OF ENGINEERING, SCIENCE AND MATHEMATICS
SCHOOL OF ELECTRONICS AND COMPUTER SCIENCE

Doctor of Philosophy

by Anthony A. Demco

The discovery of drugs that can effectively treat disease and alleviate pain is one of the core challenges facing modern medicine. The tools and techniques of machine learning have perhaps the greatest potential to provide a fast and efficient route toward the fabrication of novel and effective drugs. In particular, modern structured kernel methods have been successfully applied to range of problem domains and have been recently adapted for graph structures making them directly applicable to pharmaceutical drug discovery. Specifically graph structures have a natural fit with molecular data, in that a graph consists of a set of nodes that represent atoms that are connected by bonds. In this thesis we use graph kernels that utilize three different graph representations: molecular, topological pharmacophore and reduced graphs. We introduce a set of novel graph kernels which are based on a measure of the number of finite walks within a graph. To calculate this measure we employ a dynamic programming framework which allows us to extend graph kernels so they can deal with non-tottering, soft-matching and allows the inclusion of gaps. In addition we review several graph colouring methods and subsequently incorporate colour into our graph kernels models. These kernels are designed for molecule classification in general, although we show how they can be adapted to other areas in drug discovery. We conduct three sets of experiments and discuss how our augmented graph kernels are designed and adapted for these areas. First, we classify molecules based on their activity in comparison to a biological target. Second, we explore the related problem of *lead hopping*. Here one set of chemicals is used to predict another that is structurally dissimilar. We discuss the problems that arise due to the fact that some patterns are filtered from the dataset. By analyzing lead hopping we are able to go beyond the typical cross-validation approach and construct a dataset that more accurately reflect real-world tasks. Lastly, we explore methods of integrating information from multiple targets. We test our models as a multi-response problem and later introduce a new approach that employs Kernel Canonical Correlation Analysis (KCCA) to predict the best molecules for an unseen target. Overall, we show that graph kernels achieve good results in classification, lead hopping and multiple target experiments.

Contents

Nomenclature	ix
Acknowledgements	x
1 Introduction	1
1.1 Drug Discovery	3
1.1.1 Selecting a Biological Target	3
1.1.2 Drug Discovery Process	4
1.2 High-Throughput Methods	5
1.2.1 Virtual Screening	6
1.2.2 Quantitative Structure-Activity Relationship Analysis	7
1.2.3 SVMs for QSAR Analysis	8
1.3 Support Vector Machines and Kernel methods	9
1.3.1 Support Vector Machines	9
1.3.2 Kernel Methods	12
1.4 Research Questions	13
1.5 Contributions of Thesis	14
1.5.1 Finite-Length Graph kernels	14
1.5.2 Graph Kernels using Reduced and TP Graphs	14
1.5.3 Lead Hopping	15
1.5.4 Multiple Targets	15
1.5.5 SVM ^{Light} for graphs	16
1.6 Summary and Overview of Thesis	16
2 Molecular Representations	17
2.1 Descriptors	18
2.1.1 1D Descriptors	18
2.1.2 2D Descriptors	19
2.1.3 3D Descriptors	21
2.2 Graph Representations	22
2.2.1 Molecular Graph	22
2.2.2 Reduced Graph	22
2.2.3 Topological Pharmacophore Graph	25
2.3 Open Issues in Representing Molecules as Graphs	26
2.3.1 Colouring	26
2.3.2 Explicit Hydrogen Atoms	27
2.3.3 Aromatic Bonds	28

2.4	Summary	28
3	A Review of Graph Kernels	29
3.1	Labeled, Directed Graphs	29
3.2	Complexity of Graph Kernels	31
3.3	Approaches to Calculating Graph Kernels	31
3.3.1	A Trivial Symbolic Kernel	32
3.3.2	Walks	32
3.3.3	Extensions to Walk Kernels	37
3.3.4	Label Distances	38
3.3.5	Trees	39
3.3.6	Cycles	39
3.4	Graph Kernels for Molecules	40
3.5	Summary	42
4	Finite-Length Graph Kernels and Extensions	43
4.1	Product Graphs	43
4.2	Finite-Length Graph Kernels	43
4.2.1	FC Graph Kernel	44
4.2.2	FS Graph Kernel	44
4.3	Infinite-Length Graph Kernels	45
4.3.1	IM Graph Kernel	45
4.3.2	IG Graph Kernel	46
4.4	Non-Tottering	47
4.4.1	Non-Tottering using DP	47
4.4.2	Non-Tottering in the Original Graphs	49
4.5	Soft-Matching	49
4.5.1	Soft-Matching using DP	51
4.6	Gaps	52
4.6.1	Singly-Gapped Walks	52
4.6.2	Allowing Multiple Gaps	53
4.7	Algorithms for Combined Kernels	54
4.7.1	FC and FS Graph Kernels	55
4.7.2	IM and IG Graph Kernels	55
4.8	Summary	56
5	Experiments in Classification of Molecules	58
5.1	Experimental Design Decisions for Graph Kernel Models	59
5.2	Measuring Success	60
5.3	Visualizing the Kernel Space	61
5.4	Dataset 1	63
5.4.1	Models with Implicit Hydrogen Atoms	64
5.4.2	Colouring Choice for Graph Kernels	65
5.4.3	Choosing Soft-Matching and Single Gap Parameters	67
5.4.4	Comparing Graph Representations	68
5.5	Dataset 2	70
5.5.1	Experimental Setup	71

5.5.2	Parameter Tuning Dataset	72
5.5.3	Full Results	75
5.6	Experiment: Scaling Number of Inactive Molecules	77
5.7	Summary	79
6	Experiments in Lead Hopping	80
6.1	Lead Hopping	80
6.2	Constructing Datasets That Exhibit Lead Hopping	82
6.3	Measuring Success	84
6.4	Actives Near Cluster Centers	85
6.4.1	Calculating the Tanimoto Cluster Center	85
6.4.2	Cluster Centers of Dataset 1 - NCI-HIV Data	86
6.4.3	Cluster Centers of Dataset 2 - Pyruvate Kinase Data	88
6.5	Similarity of Actives Between Splits	90
6.5.1	Dataset 1	90
6.5.2	Dataset 2	92
6.6	Kernels for Fingerprints	94
6.7	Experimental Design Decisions for Graph Kernel Models	95
6.8	Experimental Methodology	96
6.9	Dataset 1 - NCI-HIV Data	98
6.9.1	Split 1 vs. Split 2	98
6.9.2	Split 2 vs. Split 1	101
6.9.3	Discussion for Dataset 1	104
6.10	Dataset 2 - Pyruvate Kinase Data	105
6.10.1	Split 1 vs. Split 2	105
6.10.2	Split 2 vs. Split 1	108
6.10.3	Discussion for Dataset 2	111
6.11	Summary	111
7	Experiments with Multiple Targets	113
7.1	Previous Research with Multiple Targets	114
7.2	Experiments in Predicting Multiple Targets	114
7.2.1	Dataset	115
7.2.2	Parameter Tuning	115
7.2.3	Results for SVM Leave One Target Out	118
7.2.4	Results for KPLS	120
7.3	Experiments in Predicting an Unseen Target	121
7.3.1	KCCA for Drug Discovery	122
7.3.2	Dataset	123
7.3.3	Experimental Methodology	124
7.3.4	Parameter Tuning	125
7.3.4.1	KCCA Parameters	125
7.3.4.2	Parameter Tuning Graph Kernels	125
7.3.5	Results	126
7.3.6	Discussion	128
7.4	Summary	129

8	Conclusions	130
8.1	Conclusions	130
8.2	Future Work	132
	Bibliography	133

List of Figures

1.1	Ligand and Protein	4
1.2	Hyperplane	10
1.3	Kernel Space	12
2.1	Molecular Representations of Aspirin	18
2.2	Fingerprint	20
2.3	Aspirin Molecular Graph	23
2.4	Aspirin Reduced Graph	25
2.5	Aspirin Topological Pharmacophore Graph	26
2.6	Ligand and Protein	27
3.1	From Molecular Graph to Labeled, Directed Molecular Graph	30
3.2	Product Graph	35
3.3	Geometric and Exponential Relative Importance	36
3.4	Bi-Connected Components and Bridges	40
4.1	Convergence of Infinite Length Walks	46
4.2	Tottering Walks	49
4.3	Soft Matching Walks	51
4.4	Gap Walk Features	53
4.5	Multiple Gaps in Walks	54
5.1	Visualization of MuTag and NCI-HIV Kernel Space	62
5.2	MuTag Results with Graph Kernels	66
5.3	Parameter Tuning FC and FS Kernels	73
5.4	Parameter Tuning IM and IG Kernels	74
5.5	Parameter Tuning SM and 1G Kernels	75
6.1	Constructing Lead Hopping Datasets	83
6.2	Closest Molecules to Cluster Center of Split 1 of Dataset 1 - NCI-HIV	87
6.3	Closest Molecules to Cluster Center of Split 2 of Dataset 1	88
6.4	Closest Molecules to Cluster Center of Split 1 of Dataset 2	89
6.5	Closest Molecules to Cluster Center of Split 2 of Dataset 2	90
6.6	Unique Actives from Dataset 1 Split 1vs2	100
6.7	Highest Ranked Actives in 1vs2 of Dataset 1	101
6.8	Unique Actives from Dataset 1 Split 2vs1	103
6.9	Highest Ranked Actives in 2vs1 of Dataset 1	104
6.10	Unique Actives from Dataset 2 Split 1vs2	107
6.11	Highest Ranked Actives in 1vs2 of Dataset 2	108

6.12	Unique Actives from Dataset 2 Split 2vs1	110
6.13	Highest Ranked Actives in 2vs1 of Dataset 2	111
7.1	Parameter Tuning FC and FS Kernels	116
7.2	Parameter Tuning IM and IG Kernels	117
7.3	Parameter Tuning SM, TPSM and 1G Kernels	118
7.4	Targets per Molecule	123
7.5	Molecules per Target	124
7.6	KCCA Results per target	127
7.7	KCCA Results per target	128

List of Tables

2.1	Topological Pharmacophore Notation	24
5.1	Graph Kernel Notation	60
5.2	Molecular Graph Notation	60
5.3	Results for MuTag With and Without Hydrogen Atoms	64
5.4	MuTag Results From Literature	65
5.5	MuTag Results for SM, TPSM and 1G Extensions	68
5.6	Parameter Tuning MuTag for MG, RG and TP Representations	69
5.7	MuTag Results for MG, RG and TP Representations	69
5.8	Storage Required When Varying Size of Kernel Matrix	71
5.9	Parameter Tuning NCI-HIV for Graph Colouring	73
5.10	Results for NCI-HIV with MG Representation	76
5.11	Varying Amount of Inactive Examples When Training	78
5.12	Varying Training Size Times	79
6.1	Closest Actives in Split 1 to Split 2, Dataset 1	91
6.2	Closest Actives in Split 2 to Split 1, Dataset 1	92
6.3	Closest Actives in Split 1 to Split 2, Dataset 1	93
6.4	Closest Actives in Split 2 to Split 1, Dataset 1	94
6.5	Lead Hopping Parameter Tuning for Dataset 1: Training Split 1, Testing Split 2	98
6.6	Lead Hopping Results for Dataset 1: Training Split 1, Testing Split 2	99
6.7	Lead Hopping Parameter Tuning for Dataset 1: Training Split 2, Testing Split 1	101
6.8	Lead Hopping Results for Dataset 1: Training Split 2, Testing Split 1	102
6.9	Lead Hopping Parameter Tuning for Dataset 2: Training Split 1, Testing Split 2	105
6.10	Lead Hopping Results for Dataset 2: Training Split 1, Testing Split 2	106
6.11	Lead Hopping Parameter Tuning for Dataset 2: Training Split 2, Testing Split 1	108
6.12	Lead Hopping Results for Dataset 2: Training Split 2, Testing Split 1	109
7.1	Results for Graph Models	119
7.2	Results for Graph+Target Models	120
7.3	KPLS Results	121
7.4	Parameter Tuning Graph Kernels for KCCA	126
7.5	KCCA Results	126

Nomenclature

S	set of training examples
ℓ	number of training examples
\mathbf{x}	example
y	class
\mathbb{R}	real numbers
κ	Kernel function
p	Length of walk
ℓ	Number of examples in training set
G	A labeled graph
\mathcal{V}	Vertex set of a graph
\mathcal{E}	Edge set of a graph
\mathcal{L}	Alphabet of graph labels
label	Function that assigns labels to vertices or edges
MG	Molecular graph
RG	Reduced graph
TP	Topological pharmacophore graph

Acknowledgements

I would like to thank all my friends, family and colleagues who have supported me throughout the Ph.D. process. In particular, I would like to thank my supervisor, Dr. Craig Saunders for his continual support and guidance which has made this Ph.D possible. I would also like to thank Dr. Stephen Pickett and Dr. Gavin Harper of GlaxoSmithKline (GSK) for their enthusiasm and advice. Several visits to their Stevenage research centre and their guidance greatly benefited this work. Finally I would like to acknowledge the financial support for this Ph.D. provided by GlaxoSmithKline and the University of Southampton.

To my family...

Chapter 1

Introduction

Drug discovery is a complex process which greatly benefits humanity by discovering drugs capable of relieving symptoms of disease and prolonging life. An important problem in early phase drug discovery is the identification of the initial starting point or lead molecule. One method for achieving this is to use machine learning approaches applied to a set of known ligands. This molecule is found using the activity of a molecule with a target. A ligand with a strong binding is labeled ‘active’ while others are labeled ‘not active’ to form a binary classification problem. Using sets of classified molecules, learning algorithms are trained to produce models which can predict the activity of unseen molecules. The unseen molecules classified as drugs are taken for further testing and analysis. A drug must meet other requirements for their physicochemical and pharmaceutical properties to reach the site of action and to avoid unwanted side effects. The best candidate from this analysis may be chosen for further testing and development for use as a commercial drug.

In our research, we use Support Vector Machines (SVM)s and other *kernel methods* in conjunction with graph kernels to build our classification models. Kernel methods include a set of learning algorithms that use a positive-definite kernel function from the data. A kernel function for graphs, or *graph kernel*, allows algorithms to learn directly from the structure in a graph. This seems a sensible approach as graphs are the most natural way to represent a molecule. The most well known representation is the *molecular graph* consisting of a set of atoms (or vertices) which are connected by bonds (or edges).

A graph kernel is calculated by finding matching features between two graphs. One must decide which features give the best kernel between any two molecular graphs. In our work, we use graph kernels that count the number of matching walks in each graph. We describe how counting walks can be modified to include soft-matching and gappy features. Furthermore, we describe how soft-matching can use molecular descriptors such as topological pharmacophores in their computation. We analyze and explore these

graph kernel choices using two publicly-available datasets. This analysis will describe how graph kernel parameters including walk length, soft-matching and gappy weighting parameters affect binary classification performance on molecular datasets.

In addition to this analysis, experiments are conducted to explore graph kernel efficacy in classifying specific subsets of examples in the dataset. Besides overall classification accuracy, we hope to understand whether graph kernels perform well in lead-hopping experiments; a crucial property that allows the identification of leads from different chemical families or chemotypes. We model the lead-hopping problem by splitting the data into two very structurally different sets. The difference is measured using Tanimoto similarity. By constructing a model with data from only one set, predicting the other becomes a very difficult task. We evaluate graph kernel performance for this problem and consider whether certain features can be used to overcome this challenge in order to predict molecules from a separate *lead series* or structurally similar set of ligands.

Finally, we use graph kernels in experiments with multiple targets. Here, a set of molecules have been evaluated for the interaction with several different targets. Each example molecule is given real value numbers that quantify the interaction with each target. These experiments are motivated by the fact that potential drugs must often meet several constraints. The best drug candidate is one that interacts correctly with several different targets. A drug must have a strong binding with some targets associated with a disease, although a drug must also have no binding with other targets that may cause undesirable or possibly deadly side effects. We first test our models as a multiple response problem and later introduce a new method to predict the best molecules for an unseen target using Kernel Canonical Correlation Analysis (KCCA).

Currently, a very popular descriptor used for learning models is the *fingerprint* descriptor. A fingerprint is a binary feature vector or table of values consisting of 1 or 0 entries that each describe the existence of certain sub-structural features. We provide a direct comparison between fingerprint and graph kernel models for each of our problems. Graph kernels appear to be a promising alternative to fingerprints, as no features have to be manually defined. A graph kernel is calculated using any two graphs. Furthermore, graph kernels can use features, such as gaps and soft-matching that are very hard to represent with just a fingerprint, or table of values. As a result, graph kernels provide an interesting alternative view of the data, potentially allowing new promising drugs to be discovered.

This Chapter is outlined as follows: we begin by describing the stages involved in drug discovery in Section 1.1. After, we describe how this is automated with computers using virtual screening in Section 1.2. Then we will describe kernel methods and other algorithms in Section 1.3. Finally, we will lay out our research question in Section 1.4 and the contributions and overview of this thesis in Section 1.5.

1.1 Drug Discovery

Drug discovery is a process that involves several steps. We will review this process in the following Sections. The first step is to identify the biological target that is involved in a disease (Section 1.1.1). Next, the drug discovery cycle begins (Section 1.1.2) where a number of drugs are screened through several phases against this target. Finally, experiments are carried out to clinically validate its efficiency and safety in animals and humans.

1.1.1 Selecting a Biological Target

The goal of drug discovery is to find a small molecule that interferes with the metabolic or signaling pathways, which are series of reactions, that correspond with a disease. This interference occurs when the small molecule binds to a protein (enzyme, receptor, ion channel, etc) and is a part of these pathways. The largest majority of these targets (around 45%) are G-protein coupled cellular receptors (GPCRs) which carry the signal from outside the cell membrane by conforming to a different shape when a *ligand*, or small molecule, binds causing a metabolic product of the G-protein to be released and used within the cell (Drews (2000)). The second largest class comprising around 28% of targets for drugs are enzymes. A ligand may bind with an enzyme whereby the activation causes an alteration of gene expression (Drews (2000)). When a ligand binds to either of these targets the associated metabolic or signaling pathway associated with the disease is disrupted.

A target is usually chosen rationally by a pharmaceutical programme team. In practice, a selection of pathways may be used when finding the target for a disease. These are recorded in several well-known databases for signalling pathways¹ and metabolic pathways². One of the largest problem of finding a drug is that a designed molecule may bind in other places producing usually undesired yet possibly beneficial results (Apic et al. (2005)).

There are currently less than 500 molecular targets identified and a considerably smaller amount, around 40, are used by pharmaceutical companies to target diseases, although it is suggested that there are feasibly 10 times more (between 5,000 and 10,000) potential targets that are not being used (Drews (2000)).

Drugs are tested in a number of ways including *in vivo*, *in vitro*, or *in silico*. *In vivo* and *in vitro* describe experiments which are performed inside or outside a living organism respectively. *In vivo* experiments are conducted in a living organism, such as genetically modified animal (termed *transgenic animal models*). *In vitro* is used to measure a

¹<http://cgap.nci.nih.gov/Pathways/>

²<http://http://www.genome.jp/keg/kegg2.html>

response directly against the target of interest in a test tube, or via some secondary response in a cell based assay. A third method, *in silico*, uses computers to test the activity of the drugs, we delay this introduction to section 1.2.1.

1.1.2 Drug Discovery Process

Once the target has been identified, the next step is to find a small molecule, called a *ligand*, that is able to bind to the target to alter its normal functioning. This is achieved by testing a set of molecules for their activity with a certain target. This test is known as a *target* or *biological assay* and quantifies the effect of the ligand for the desired activity with the target. The *potency* of a drug is the measure of this activity in a biological system. An example of a ligand binding to a protein is shown in figure 1.1. This image was originally generated by the CCP4 program, given as an example in their documentation on their website³.

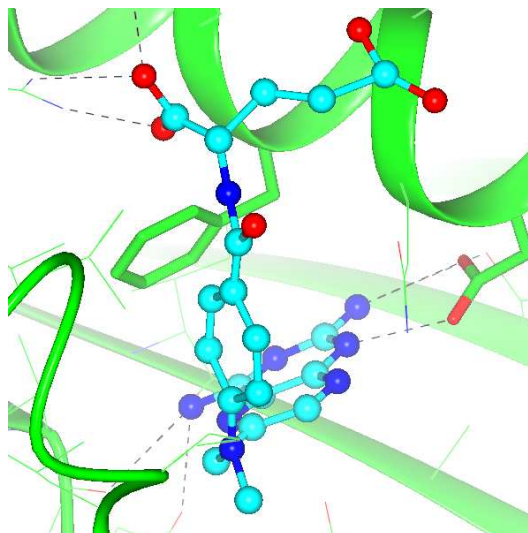


FIGURE 1.1: This figure show a ligand docking to a protein. The ligand is represented by the blue and red atoms and bonds in the center, while the protein is shown as long strands in green. Key interactions between ligands and protein atoms are shown by the dotted lines.

Following target identification, the drug discovery cycle begins which involves iterations of selecting certain chemicals, *synthesizing* or physically creating the selected molecules and after screening for the desired activity. A set of *hits* are identified in early stages of this cycle. A hit is a molecule which achieves an adequate activity on a target assay along with a low molecular weight (Drews (2000); Bleicher et al. (2003)). From these hits a set of *leads* are chosen that meets other requirements.

Some of these requirements restrict properties of the hits to ones that are more drug-like. A drug must be able to reach the target *in vivo*, which involves passing through

³<http://www.ysbl.york.ac.uk/~ccp4mg/>

physiological barriers including the blood-brain barrier and cell membranes. The *blood-brain barrier* is a cellular structure that restricts the passages of chemicals from the bloodstream into the tissue. Furthermore, a drug must remain in a body for a certain amount of time for the drug to take effect and then later it must be removed or excreted. Also, the drug's toxicity levels must be low enough to not cause damage (Leach and Gillet (2003)). These properties are described by the term ADME (or ADMET) which stands for absorption, distribution, metabolism, excretion and toxicity (Xu and Hagler (2002); Boobis et al. (2002)). If the molecule has poor ADME properties, it will be precluded from further processing.

These stages allows for a *lead series* to be selected. A lead series is a group of molecules with similar molecular properties and activity with a target (Bleicher et al. (2003)). The lead optimization phase is a further step usually carried out using a combination of in vitro and in vivo experiments. This differs from earlier hit stages that use in vitro experiments. The final step towards a commercial drug is clinical valuation. Usually, several different *lead series*, or set of molecules with similar molecular structure, are selected for clinical valuation as unforeseen side effects only appear in later clinical validation stages. This reduces the risk of starting the entire discovery process again, as an alternative lead series may not exhibit this unforeseen side effect.

The final phase of drug discovery involves *clinical trials*, where the drug will be dosed in human subjects. These trials usually take place over four phases. The first three occur before large-scale production of the drug, and the fourth occurs after the drug goes to market (Krogsgaard-Larsen et al. (2002)).

1.2 High-Throughput Methods

With the advent of high-throughput screening (HTS) technology, and ultra-HTS, hundreds of thousands of molecules can be processed per day, producing volumes of data for screens against a biological target. HTS utilizes robots that are capable of performing all tasks including setting up and screening molecules. Similarly, combinatorial chemistry allows for a large number of molecules to be *synthesized*, or physically created, at a low cost and used in these screening experiments. These two technologies allow a far greater number of molecules to be synthesized and processed which has greatly increased the number of hits for a given target. Despite a large number of hits, the number of lead series did not increase (Manly et al. (2001)). As a result, other approaches must be considered to find drug-like molecules.

By using the data from HTS, computers continue the drug discovery process and help reduce the number of false-positive hits. Using further computer screening, molecules can be identified which not only have good target interaction, but also have other desirable properties including ADME properties. This area of research has emerged into a

new field called *chemoinformatics*. Chemoinformatics encompasses the representation, storage, retrieval and analysis of chemical information. We refer the reader to the introductory books on chemoinformatics including Leach and Gillet (2003) and Gasteiger and Engel (2003).

1.2.1 Virtual Screening

Virtual screening or *in silico screening* is the process of using computers to analyze the activity of a molecule for a specific target. This is used as a selection process prior to wet-lab experiments with several advantages including the ability to analyze far greater numbers of molecules, including virtual molecules, which have not yet been synthesized (Gasteiger and Engel (2003)). To perform virtual screening, either the structure of the target must be known, or the activity must be known for a set of molecules with the target. Although the exact structure of the target would give the most information, it is the most time-consuming and sometimes impossible to obtain. With targets, such as G-Protein Coupled Receptors (GPCR)s, one would need to solve the *protein-folding problem* which obtains a 3D orientation of the protein target. Research into protein-folding is currently on-going, and it is known to be a difficult problem to solve. In Fraenkel (1993), it is described that 3D conformation of a protein into it's lowest energy state is an NP-hard problem. As a result, most chemoinformatic research focuses on the ligand and it's structural information, along with activity information which quantifies the interaction with the target.

Despite the complexity involved in using the target alone, methods do exist. Methods which use the structure of the target are termed structure-based approaches. This type of approach can only be used if the structure of the target is known. The most common approach is *docking*, which computes an activity score based on the optimal position that a ligand can be positioned at a target (Jorgensen (2004)). The docking approach uses both a search algorithm to identify the optimal alignment of the ligand and target, along with an evaluation function which quantifies the resulting binding of intramolecular forces. The search algorithm must take into account the flexibility of a molecule. Often in a 3D space, a molecule can exhibit different structures (or *conformations*) as rotatable bonds allow different orientations of the molecular structure. The docking approach to virtual screening will attempt to dock all molecules from a library of molecules and will select the top scored molecules as the most promising candidates. A second type of virtual screening that uses the structure of the target to choose potential drugs is *de novo design*. This approach builds a potential ligand at the site of the target by iteratively adding drug fragments that fit well with the target (Jorgensen (2004)). Typically, graphical modeling tools are used, although other approaches are possible. One recent example of de novo design in Schneider et al. (2000) used evolutionary algorithms to search a set of fragments. The fitness function

was measured using a pharmacophore distance between a known molecule that was a good thrombin inhibitor and the derived molecules.

Virtual screening can also be accomplished using approaches that do not use the information from the structure of the target. A *ligand-based* approach only uses information from the ligand along with a number to quantify the interaction with a target. The most well-known ligand-based approach is *similarity searching*. Using a set of known actives, a set of potential drugs are chosen by ranking these based on similarity between molecular structures. The highest ranked are most structurally similar to known actives and are chosen for further processing (Jorgensen (2004)). Similarity can also be measured using *pharmacophores* that define the function of different areas within a molecule in a 3D space. Not only can similarity searching be used to find the best candidate molecules, but the lowest ranked molecules can be *filtered* or removed to produce a smaller more useful set for further testing (Jorgensen (2004)).

Models which find patterns between the structure of the ligand and the class (either ‘active’ or ‘not active’) are categorized as Quantitative Structure-Activity Relationship (QSAR) analysis and is discussed in the following Section. We note that similarity searching does not fit into the typical SAR or machine learning framework. In QSAR analysis and machine learning research, data is split into training and testing phases. In the training phase, a model will learn patterns between a feature vector describing a molecule and an associated class ‘active’ or ‘not active’. Next, predictions are made in a testing phase, and the accuracy of the classification model is recorded. Although some machine learning models only train using one class, such as Nearest Neighbour algorithms and One-Class SVMs, these models are able to predict both the ‘active’ and ‘not active’ class. Similarity searching is only used to locate other similar ‘active’ models, so it can not be compared with other machine learning algorithms.

1.2.2 Quantitative Structure-Activity Relationship Analysis

Quantitative Structure-Activity Relationship (QSAR) analysis is a form of virtual screening that is used to find patterns between the structure of a molecule and its biological activity. A related problem is to find patterns between molecular structure and pharmacokinetic properties such as ADME properties; this is termed Quantitative Structure-Property Relationship (QSPR) analysis. The origins of QSAR/QSPR are found in the pioneering work of Hansch (Hansch and Fujita (1964)), who found equations that describe a relationship between biological activity and molecular structure.

QSAR analysis follows the approach used often in machine learning. The data is split into two sets, one for training known as the *training set* and a second *test set*. A model is trained using data from the training set and later this model is used to predict the activity of molecules in the test set. The *accuracy*, or number of correct predictions, is

recorded and can be used to compare to other models of the same data. QSAR analysis typically uses both classes to train (class ‘active’ and class ‘not active’) which forms a binary classification problem.

In order to derive this binary classification some decisions must be made. The activity is a real number quantifying the interaction between ligand and target. Some datasets have ranges defined for the activity, where the values fall into a highly active, moderately active or inactive class. A binary classification problem can then be modeled using two of these classes, for example highly active vs. inactive. Although we focus on classification, the activity is a real number, so QSAR can also be modeled using regression techniques.

A wide range of machine learning algorithms are available and can be performed for QSAR analysis, including multiple linear regression, partial least square and artificial neural networks. More recently Support Vector Machines (SVM)s have been applied. A good introduction to machine learning which describes many of the learning algorithms is given in the book Duda and Hart (2001).

1.2.3 SVMs for QSAR Analysis

Support Vector Machines (SVMs) have been applied to a wide variety of problems that find patterns between the structure and activity (QSAR) as well as other properties (QSPR). In this section we review recent literature that discusses several different applications of SVMs in drug discovery. This research is grouped into applications of SVMs in QSAR, applications in QSPR, and other applications.

QSAR is an important task in drug discovery where SVMs have been applied successfully. In many papers, SVMs were compared to neural networks and other classifiers. Byvatov et al. (2003) found that SVMs outperformed neural networks using a number of different molecular descriptors, and irrespective of training set size. Similarly, results in Zernov et al. (2003) showed that SVMs could more accurately identify drug-like molecule from non-drugs. Interestingly, when Glick et al. (2006a) added noise to the datasets SVMs still outperform conventional classifiers. Learning from the structure of a molecule, SVMs have been used to predict a number of different targets. These include COX-2 inhibitors (Yao et al. (2004)), P450 3A4, 2D6 and 2C9 inhibitors (Yap and Chen (2005)) AP-1 and NF-kB mediated gene expression inhibitors (Liu et al. (2003)) and the binding affinities to human serum Albumin (Xue et al. (2004c)). Using 21 targets related to depression, SVM models were analyzed for their ability to discover new lead series in Glick et al. (2006b). SVMs have also been used in other QSAR research problems. Instead of using the ranking formulation of an SVM, Jorissen and Gilson (2005) showed that actives could be enriched so models will maximize the number of actives in the top 10% of the results. In Warmuth et al. (2003), an active learning methodology is used to show how SVMs can be used to choose the minimum batch of lab-tested molecules by increasing

the training set size in subsequent iterations. Finally, Byvatov and Schneider (2004) showed that SVMs could be used for feature selection by considering examples chosen as support vectors. By only training with the most relevant features, similar accuracy was achieved with models using all examples.

Research has shown that SVMs can also be used for Quantitative Structure-Property Relationship (QSPR) analysis. Research has shown that a number of different properties can be predicted with SVMs. These include the toxicity of phenols (Yao et al. (2004)), the capacity factor (Liu et al. (2004a)), the isoelectric point (Liu et al. (2004b)), the O-H bond dissociation energy (Xue et al. (2004d)), the aqueous solubility (Lind and Maltseva (2003)) and finally the heat capacity (Xue et al. (2004a)).

SVMs have been applied to learn patterns between molecular structures and other relationships. A model of Quantitative Structure-Mobility Relationship (QSMR) was introduced in Xue et al. (2004b) where the electrophoretic mobilities of 58 aliphatic and aromatic carboxylic acids were predicted. A Quantitative Structure-Retention Relationship (QSRR) model was used in Song et al. (2002) to predict the retention behaviour of 24 proteins.

Some work has been done using graph kernels and SVMs to predict QSAR, although we delay this discussion to Chapter 3.

1.3 Support Vector Machines and Kernel methods

In this Section, we will introduce Support Vector Machines and Kernel Methods. We will use SVMs with graph kernels to perform virtual screening of molecules.

1.3.1 Support Vector Machines

Support Vector Machines (SVM) are a linear binary classification algorithm. A detailed introduction describing how SVMs are derived is given in the book Cristianini and Shawe-Taylor (2000). SVMs are a type of learning algorithm that fall into a group of algorithms known as kernel methods. We refer the reader to the books on kernel methods including Shawe-Taylor and Cristianini (2004) and Scholkopf and Smola (2002), although they were first introduced in Boser et al. (1992).

We begin by describing how an SVM can be used for classification when the data is linearly separable. A binary classification problem is one where the set of ℓ examples has $\mathcal{S} = \{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_\ell, y_\ell)\}$ where $\mathbf{x} \in \mathbb{R}^n$ and $y \in \{-1, 1\}$. When the training data are *linearly separable*, then there exists a hyperplane that separates all of the positive examples from the negative examples. In this case, a hard-margin SVM can be used, which selects a hyperplane with the largest distance between the closest data point. See

Figure 1.2 for the case of a hard-margin SVM. If an example is positive then it will have $\langle \mathbf{w}, \mathbf{x} \rangle + b \geq 0$, and negative will have $\langle \mathbf{w}, \mathbf{x} \rangle + b \leq 0$. Selecting the largest margin will hopefully produce a model with good *generalization* abilities, or ability to predict unseen data.

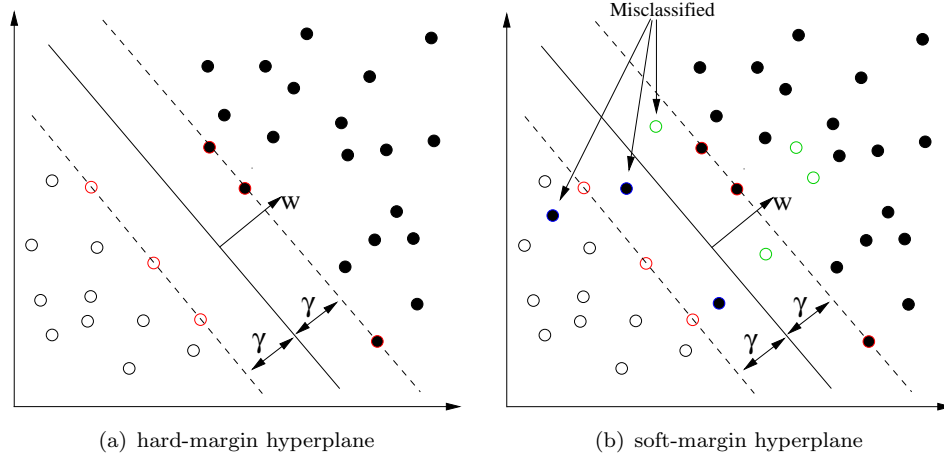


FIGURE 1.2: A hyperplane, is shown that separates two classes of data. \mathbf{w} and b are two parameters which define the slope and distance from the origin respectively. The distance between the closest point(s) and the hyperplane is given by γ , the margin. The left image shows a hard-margin SVM, where the data is linearly separable. The right image shows a soft-margin SVM that allows examples to violate the margin; this is essential as the data are not linearly separable.

With most real-world data, the data are not linearly separable and therefore slack variables are introduced which allow examples to cross the hyperplane. Now, the optimization problem will involve both finding a hyperplane that has the smallest errors from examples which violate the margin. This is termed the *soft-margin SVM* and is defined by the following optimization problem.

$$(\mathbf{w}^*, b^*) = \operatorname{argmin}_{\mathbf{w}, b} \quad \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_i \xi_i \quad (1.1)$$

$$s.t. \quad y_i[\langle \mathbf{w}, \mathbf{x}_i \rangle + b] \geq 1 - \xi_i \quad (1.2)$$

$$\xi_i \geq 0, 1 \leq i \leq \ell \quad (1.3)$$

This results in the primal lagrangian:

$$L(\mathbf{w}, b, \boldsymbol{\alpha}, \boldsymbol{\xi}) = \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_i \xi_i - \sum_i \alpha_i (y_i[\langle \mathbf{w}, \mathbf{x}_i \rangle + b] - 1 + \xi_i) - \sum_i r_i \xi_i \quad (1.4)$$

where $\alpha_i \geq 0$ and $r_i \geq 0$ for $1 \leq i \leq \ell$. Taking the partial derivatives with respect to w , b and ξ gives:

$$\frac{\partial L}{\partial w} = w - \sum_i \alpha_i y_i x_i = 0 \quad (1.5)$$

$$\therefore w = \sum_i \alpha_i y_i x_i \quad (1.6)$$

$$\frac{\partial L}{\partial b} = \sum_i \alpha_i y_i = 0 \quad (1.7)$$

$$\frac{\partial L}{\partial \xi_i} = C - \alpha_i - r_i = 0 \quad (1.8)$$

$$\therefore C = r_i + \alpha_i, 1 \leq i \leq \ell \quad (1.9)$$

Inserting these partial derivatives back into the primal gives the dual formulation:

$$(\alpha^*) = \operatorname{argmax}_{\alpha} \quad \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j \langle x_i, x_j \rangle \quad (1.10)$$

$$s.t. \quad C \geq \alpha_i \geq 0, \quad (1.11)$$

$$\sum_i \alpha_i y_i = 0 \quad (1.12)$$

From Karush-Kuhn Tucker (KKT) conditions, only examples with a functional margin of 1 have non-zero α_i^* , and are called the support vectors (SV). If the data are linearly separable, all the SVs will lie on the margin, hence making the algorithm and decision function efficient by only focusing on these examples. This efficiency will also exist in the soft margin case, although examples which violate the margin are included. Predictions are made only using the support vectors:

$$f(x) = \operatorname{sgn} \left(\sum_{SV_s} \alpha_i^* y_i \langle x_i, x \rangle + b^* \right) \quad (1.13)$$

Here, b^* can be obtained from any support vector, x_i :

$$b^* = \frac{y_i}{\sum_{j=1}^{\ell} \alpha_j^* \langle x_j, x_i \rangle} \quad (1.14)$$

1.3.2 Kernel Methods

Kernel methods include a number of algorithm such as SVMs, Kernel Canonical Correlation Analysis (KCCA), Kernel Partial Least Squares (KPLS) and others. In order for an algorithm to use a kernel function, a dual form must be calculated where only inner products between examples are required.

An SVM can be transformed into a kernel SVM by replacing all inner products with a positive definite kernel function, κ . This transformation is termed the *kernel trick*. This is equivalent to transforming all input vectors $\mathbf{x}_1, \dots, \mathbf{x}_\ell$ to $\phi(\mathbf{x}_1), \dots, \phi(\mathbf{x}_\ell) \in \mathcal{H}$, where ϕ is a mapping from the input space \mathcal{X} to the feature space \mathcal{H} ($\phi: \mathcal{X} \rightarrow \mathcal{H}$). Please see Figure 1.3 for an example.

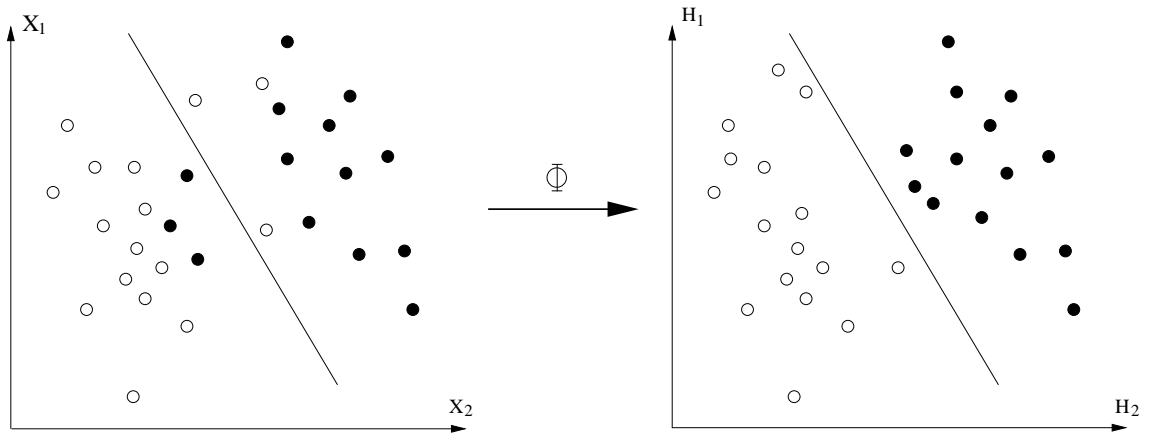


FIGURE 1.3: Data located in an input space \mathcal{X} (left) can be passed through a kernel function, where they are linearly separable in the feature space \mathcal{H} (right).

In the higher dimensional feature space, the SVM can be applied and solve non-linear patterns (in the input space). With the inner product replaced by a kernel function, the decision function for the SVM in (1.13) becomes:

$$f(\mathbf{x}_j) = \text{sgn} \left(\sum_{SV_s} \alpha_i^* \kappa(\mathbf{x}_i, \mathbf{x}_j) + b^* \right) \quad (1.15)$$

One of the key properties of a kernel function is that it is positive definite.

Definition 1.1 (Positive Definite Kernel). κ is said to be a positive definite kernel if and only if, for all positive integers ℓ and for all $\mathbf{x}_1, \dots, \mathbf{x}_\ell \in \mathcal{X}$, the square $\ell \times \ell$ matrix $K = (\kappa(\mathbf{x}_i, \mathbf{x}_j))_{1 \leq i, j \leq \ell}$ is positive semi-definite which means that all eigenvalues are nonnegative.

If a kernel function can be defined between two examples, then it can be used with any kernel method. An example of a kernel that maps data into a higher dimensional space is the *polynomial kernel*:

$$\kappa(\mathbf{x}_i, \mathbf{x}_j) = (\langle \mathbf{x}_i, \mathbf{x}_j \rangle + R)^d \quad (1.16)$$

where $R \geq 0$ and d are integers.

Often, the kernel is normalized using the following:

$$\hat{\kappa}(\mathbf{x}_i, \mathbf{x}_j) = \frac{\kappa(\mathbf{x}_i, \mathbf{x}_j)}{\sqrt{\kappa(\mathbf{x}_i, \mathbf{x}_i)\kappa(\mathbf{x}_j, \mathbf{x}_j)}} \quad (1.17)$$

This normalization ensure that the data lies on the unit hypersphere. We will use this with graph kernels as it removes the bias from larger graphs.

One of the most remarkable properties of kernel methods is their ability to handle structured data. Structured data exists in two types: either each example contains an inherent structure and cannot readily be described by explicit feature vectors (i.e. documents, images, graphs) or the input space is structured (i.e. dependencies exist between examples). A method to calculate structured kernels by using successive comparisons of smaller parts of the structure was proposed in Haussler (1999). Independently, string kernels for biological sequences were proposed in Watkins (2000). More recently, rational kernels, a general framework that allows most kernels to be calculated from a weighted state transducer, is given in Cortes et al. (2004).

One type of structured kernel is the string kernel. Strings are sequences of characters from a given alphabet. String kernels were first proposed using a dynamic programming approach to allow insertions and deletions in strings as given in Watkins (2000). A similar kernel, termed the *n-gram kernel*, allows gaps in the strings and was used for text categorization (Lodhi et al. (2001)). Many efficient implementations of string kernels have been proposed including using suffix arrays as discussed in Leslie and Kuang (2004). The computational performance of this suffix array implementation was later improved in other work (Teo and Vishwanathan (2006)).

A tree kernel is another type of structured kernel. A tree is a hierarchical data structure composed of root nodes and branches to lower edge nodes. Tree kernels were first proposed in Collins and Duffy (2002), using a recursive calculation, that compares each subtree at a given leaf. A fast implementation was proposed in Vishwanathan and Smola (2003) by calculating the kernel using a suffix tree.

More recently, a third type of structured kernels for graphs have been proposed. We postpone a review of graph kernels to Chapter 3, as we give a detailed description of each kernel.

1.4 Research Questions

In this thesis, we will be analyzing several issues that incorporate both chemoinformatics and machine learning topics. We will use kernel methods for graphs in order to analyze molecular data. Specifically, we will address the following research questions.

1. What is best graph representation for a molecule? What methods should be used to colour vertices of this graph?
2. What is the best graph kernel to accurately classify molecular data?
3. What is the best graph kernel to classify molecular data for lead hopping?
4. What is the best graph kernel for experiments that use multiple targets?

1.5 Contributions of Thesis

In the course of this thesis, our research has been presented in the following publications:

1. A presentation at 6th IARP-TC-15 Workshop on Graph-based Representations in Pattern Recognition in Alicante, Spain (Demco and Saunders (2007b)).
2. A poster at UK-QSAR and Chemoinformatics Group, Spring meeting at AstraZeneca (Alderley Park, UK) (Demco and Saunders (2007a)).
3. A book chapter in *Perspectives of Neural-Symbolic Integration* that reviews tree and graph kernels (Saunders and Demco (2007)).
4. A technical report that introduces finite-length graph kernels and extensions Demco and Saunders (2009), under preparation.

We review the contributions of the thesis in the following sections.

1.5.1 Finite-Length Graph kernels

We present Finite-length graph kernels in Chapter 4. These kernels extend the work of Gärtner et al. (2003), although it is calculated efficiently via dynamic programming. Our framework allows the inclusion of expert knowledge. We describe how one can include two new extensions: soft-matching and gaps. Finally, we show how non-tottering walks, previously introduced in Mahé et al. (2004) are included.

1.5.2 Graph Kernels using Reduced and TP Graphs

The classic representation of a molecule is a molecular graph where the vertices are atoms connected by edges (bonds). In this thesis, we will use this standard representation along with two other graph representations: topological pharmacophore and reduced graphs. The first is the topological pharmacophore (TP) graph. The TP graph has the same structure as the molecular graph although each vertex label is replaced with

a label representing the nearest topological pharmacophore descriptor to each atom. The TP descriptor defines functional areas of the molecule related to binding with a target. A second representation we use is reduced graphs. A reduced graph is a compact representation of a molecular graph. Such reductions as collapsing carbon rings to a single node are used to create a reduced graph from a molecular graph. We will show results that includes walks which soft-match between atom labels and TP labels, along with reduced graphs in Chapter 5.

At the time of writing this thesis, only limited work with reduced graphs has been analyzed with optimal assignment graph kernels (Fröhlich et al. (2005)). No research using TP graphs has previously been presented.

1.5.3 Lead Hopping

Lead hopping is the term used to describe learning from one lead series of molecules (group of molecules with similar molecular structure) in order to find actives in a different set of lead molecules. Most other research that has used graph kernels has measured results with a cross-validation using all examples. In practice, drug discovery ideally wants to learn a model that returns a diverse set of molecules for further testing. By finding molecules that are from other chemical series, allows more flexibility in later stages if one lead has undesired side-effects. Lead Hopping is a relatively hard problem as typically the features used for learning are 2D features, so using a model built from one group which is dissimilar in a 2D sense is not trivial. We will describe how we model this problem in Chapter 6 and present new results using the graph kernels along with several graph representations. To our knowledge, no other research has been presented using graph kernels to analyze the lead hopping problem.

1.5.4 Multiple Targets

In practice, a drug candidate must often meet several requirements. The best candidate may be one which has a strong binding with certain targets associated with a disease and no binding with targets that cause undesired side-effects. We performed two sets of experiments with multiple targets in Chapter 7. We first used a publicly-available dataset which has over seven thousand compounds along with activity data for 36 targets. We test several graph kernels and also several graph-based representations. In our first test, we tried a binary-classification of 1 target that uses information from both the remaining 35 targets and from the structure of the molecule with a graph kernel. Next, we used kernel PLS to predict all 36 targets. Our second set of experiments used Kernel Canonical Correlation Analysis (KCCA). We used string kernels to represent the targets and graph kernels to represent molecules. Using this approach, we can predict molecules that may bind with a target using a model that did not train on any examples that bind

with the target. This could be useful in drug discovery if a new target is generated although no training data is available.

1.5.5 SVM^{Light} for graphs

The software used to compute these kernels presented in this thesis has been implemented using a significantly modified version of SVM^{Light}⁴ (Joachims (1999)). This software allows any graph to be used as input, and any combination of our Finite-length graph kernel to be computed. We plan to release this software publicly and will offer it as a separate package, similar to another package, SVM^{Light} for trees by Alessandro Moschitti⁵. We chose to develop our kernels in SVM^{Light} as it has built in features to work with large datasets by using approximations when solving the SVM optimization problem. SVM^{Light} was initially developed to work with large datasets of strings, another class of structured data. Internally, the kernel matrix is not pre-computed which allows large datasets to be computed, which would otherwise require a large amount of memory for storage of the kernel matrix.

1.6 Summary and Overview of Thesis

In this Chapter we have described one of the primary motivations for designing kernel methods for graphs: drug discovery. This led to a review of virtual screening and QSAR analysis. We describe a key tool in this thesis: kernel methods and support vector machines. Lastly, we discussed our research questions and the contributions of the thesis.

This thesis will be organized in the following manner. In Chapter 2, we will describe different descriptors and graphs that can be used to represent molecules. We also discuss other issues involved in representing molecules as graphs. In Chapter 3, we will review kernel methods for graph structures. Following this, we will introduce new Finite-Length graph kernels and extensions in Chapter 4. In Chapter 5, results are given for the Finite-Length graph kernel on two publicly available datasets. We give results for a number of graph kernel parameters and test three graph-based representations. In Chapter 6, we describe a way of modeling the lead-hopping problem using two publicly available datasets. We give results for a number of Finite-Length graph kernels and parameters along with three graph representations. In Chapter 7, we give results that use our graph kernels with multiple targets. We use kernel PLS and SVMs to understand how this extra target information benefits our models. Also, we use KCCA to predict the activity of molecules for an unseen target. Finally, in Chapter 8, we conclude and give future directions.

⁴<http://svmlight.joachims.org/>

⁵<http://ai-nlp.info.uniroma2.it/moschitti/TK1.2-software/Tree-Kernel.htm>

Chapter 2

Molecular Representations

Molecules are complex real-world objects. In order to build models from this data, a consistent representation must be chosen. There are many features that can be used from a molecule, ranging from its physicochemical properties, to structural features in two and three dimensions. A QSAR and machine learning model can only provide patterns as good as the information that is used to describe the molecule. Ideally, we want to choose a representation that will give the model enough information to find patterns that distinguish between drugs and non-drugs.

Molecular *descriptors* are numerical values that characterize properties of molecules. Descriptors can be considered features in a machine learning sense. In QSAR research, descriptors are categorized into three types: 1D, 2D or 3D descriptors. Although this categorization is consistent throughout QSAR literature (such as reviews given in Leach and Gillet (2003) and Gasteiger and Engel (2003)), the naming convention for these categories may at first be confusing to the reader as 1D refers to the dimension of the descriptor and 2D, 3D refer to the dimension from which the descriptor is derived. A 1D descriptor, is a single real value which describes overall properties of the molecule. A 2D descriptor, is a feature vector (table of values) that is derived from the molecular graph (a two dimensional structure). A 3D descriptor, is a feature vector that describes properties derived only from a 3D view of the molecule (such as distances between molecules). An example of several types of structural representations of an Aspirin molecule is illustrated in Figure 2.1.

In our research, we use graph kernels which can learn directly from the structure of a graph. Graphs are given as input to the learning algorithm. As a result, all of the structural information originally in the graph is available. Graph kernels do not discard any structural information in early data processing steps. Only once the graph kernel is chosen are certain features selected and implicitly given a feature vector representation. This approach allows different graph kernels and parameters to be used that choose different features without re-processing the original input feature vectors.

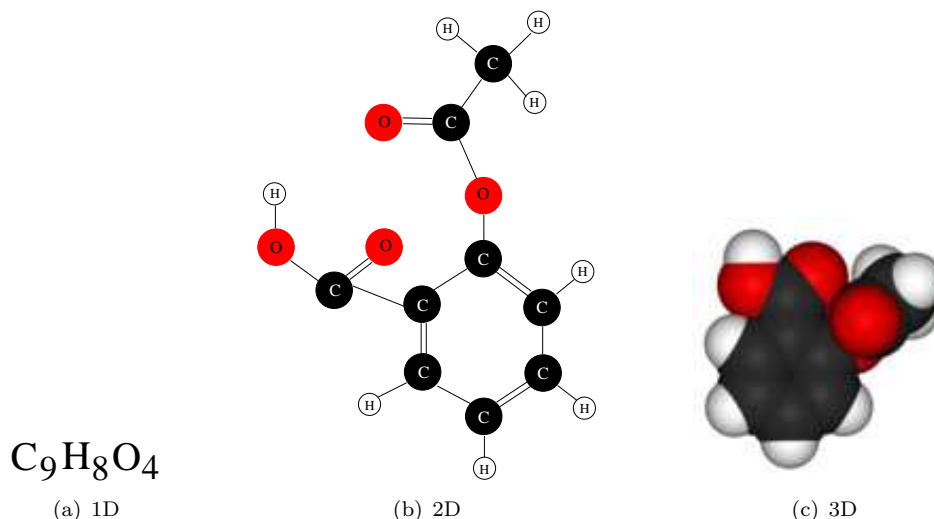


FIGURE 2.1: These images show several different molecular representations of aspirin. The first image is a 1D representation. In the first image, we give the chemical name for Aspirin as a 1D descriptors are single values describing global chemical and structural properties of the molecule. 2D descriptors are derived from structural features in the molecular graph. 3D descriptors are derived from the molecules shape and representation in a 3D space, including its 3D coordinates. The atoms in 2.1(b) have been coloured in the same manner as 2.1(c) to show the relationship between these two views.

2.1 Descriptors

Descriptors are numerical values that describe properties of molecules. Using 1D descriptors we are able to differentiate between most molecules, although descriptors from greater dimensions, 2D and 3D, provide more information about its shape and function. Ultimately, we hope to encode information about the molecule in order to differentiate between classes such as ‘active’, ‘not active’ or ‘toxic’ and ‘not toxic’. Potentially a specific part of a molecule, only described by 2D or 3D features, may be key to the interaction with a target associated with a disease. Introductory books on chemoinformatics, including Leach and Gillet (2003) and Gasteiger and Engel (2003), give detailed reviews of many different descriptors. In this section, we will describe some of the most prevalent ones.

2.1.1 1D Descriptors

Descriptors from 1D contain the least amount of information. A 1D descriptor is a single value which describes physicochemical properties and counts of features of a molecule. This can include counts of atom types, bond types, aliphatic and aromatic rings. Other types of 1D descriptors include physicochemical properties of a molecule. For example, a useful 1D descriptor is the hydrophobicity which measures the ability to pass through a cell membrane (Gasteiger and Engel (2003)). Finally, values that summarize properties

of a molecular graph are considered 1D descriptors. Several examples of these are described in Gasteiger and Engel (2003) including the *Wiener index* (Kier and Hall (1986)) which is a sum of the distances (in bonds) between any two atoms in a molecule. Another type is the *Molar Refractivity* which is a proportion of the molecular weight to density.

2.1.2 2D Descriptors

2D descriptors give a much more expressive representation of molecules. This category of descriptor gives a feature vector that describes structural elements from the molecular graph. It is called a 2D descriptor as the molecular graph is a 2D structure. The molecular graph is one of the most natural representations of a molecule.

Before describing how 2D descriptors are extracted from a molecular graph, we mention how molecular graphs are physically stored on a computer. The standard representation for molecular graphs is the MOL (or SDF) file format. In this format the graph is described using a *connection table* which consists of a list of atoms followed by a list of bonds (or connections) between each of the atoms. So for example, a single line of a connection table may contain atom 1 is connected to atoms 3 and 5. The next line may contain atom 2 is connected to atom 4 and so on. In our research, we use an *adjacency matrix* representation, used often in graph theory. The same information is contained in the connection table and adjacency matrix, although it is expressed in a different format. For a molecular graph consisting of ℓ atoms, an adjacency matrix A is an ℓ by ℓ matrix where entry $A_{i,j}$ is 0 if no bond exists and $A_{i,j} > 0$ if a bond exists. Converting from the connection table in the MOL format to an adjacency matrix is straightforward. A matrix of size ℓ is filled with zeros, iterating through each bond connection the corresponding adjacency entry is updated. Both Leach and Gillet (2003) and Gasteiger and Engel (2003) give a more detailed description of the MOL file format.

After constructing a molecular graph, one must choose a 2D descriptor which defines specific subgraphs and features which represent the graph in a fixed table of values. As graphs are of different sizes, we cannot directly use the adjacency matrix as a descriptor. Even if every graph had the same number of atoms, and therefore had the same size adjacency matrix. To build an adjacency matrix, an ordering of the vertices must be given, so a certain atom in a molecule corresponds to the first row and column of the adjacency matrix, and so on. If the same molecule was given, although a different order was used, a different adjacency matrix would be created. As a result, directly using the adjacency matrix is not a good descriptor. We must choose counts of sub-graphs and features in order to create a consistent descriptor for each molecule.

A *fingerprint* is a binary feature vector, where each entry indicates the presence of a *fragment* (or small sub-structure). For example, a small fragment, $C - O$, is a Carbon atom singly bonded to an Oxygen atom.

There are two types of fingerprints. The first type uses a set of predefined structural keys which define each element in the fingerprint. Each entry corresponds to a specific fragment. An example of this type is the MDL MACCS keys (McGregor and Pallai (1997)) which use 166 predefined fragments. A second type of fingerprint uses open-hashing allowing each entry to correspond to more than one fragment. Starting with a single atom, consecutively larger fragments are built in each iteration of an algorithm up to a certain depth. Each of these fragments is hashed into the fingerprint. The proprietary *Daylight*¹ fingerprints use up to eight atoms in a fragment. The length of each fingerprint is usually restricted to a size of 1,024 entries. Please see Figure 2.2 for an example of the fingerprinting process. As an entry in the second type of fingerprints may define the presence of more than one fragment (or collisions), one cannot easily map from a fingerprint to a molecule.

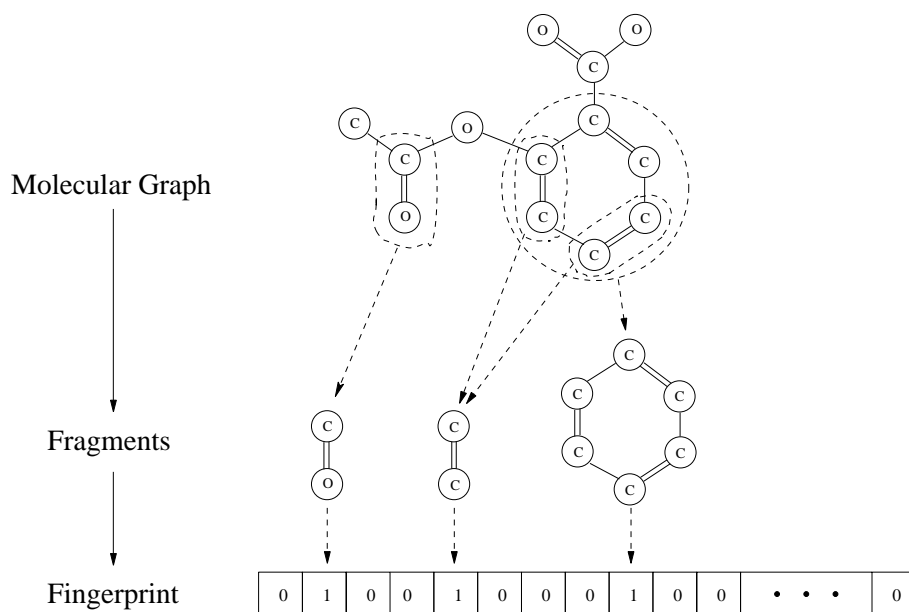


FIGURE 2.2: This figure shows the process of how a fingerprint is generated. Fragments, or sub-graphs, from a molecular graph are selected and represented in a binary-value fingerprint. On the left of the diagram the stages of the fingerprinting process are given. On the right, the corresponding stages are shown for an Aspirin molecule. Examples of a few possible fragments are given below the original molecule. Below these fragments, an example of a fingerprint of these fragments is shown. A 1 indicates the presence of the fragment and 0 otherwise.

The *Tanimoto coefficient* measures the similarity between two molecules (Willet et al. (1986)). For example, if we have a fingerprint \mathbf{F}_1 and \mathbf{F}_2 , it is calculated as follows:

¹<http://www.daylight.com>

$$T(\mathbf{F}_1, \mathbf{F}_2) = \frac{\langle \mathbf{F}_1, \mathbf{F}_2 \rangle}{\langle \mathbf{F}_1, \mathbf{F}_1 \rangle + \langle \mathbf{F}_2, \mathbf{F}_2 \rangle - \langle \mathbf{F}_1, \mathbf{F}_2 \rangle} \quad (2.1)$$

The Tanimoto score is a value between 0 and 1, where 0 indicates no similarity and 1 indicates a high 2D similarity. As all features used in the fingerprints are pulled from the 2D molecular graph, this Tanimoto value defines a 2D similarity measure for two molecules.

Topological Pharmacophores (TP) are functional areas of the molecule. TPs are 2D, based on the bond count between features (Schneider et al. (1999a)) which is the IUPAC definition of a pharmacophore (Böhm and Schneider (2003)). We will describe how reduced graphs and TP graphs can be constructed using Topological Pharmacophores in Section 2.2.3.

2.1.3 3D Descriptors

3D descriptors have the most information about the structure of a molecule. Basic 3D descriptors are calculated using three dimensional properties of molecules such as surface, shape and volume properties. Brown and Martin (1997) compared several basic 3D descriptors to both types of 2D fingerprints: non-hashed fingerprints (MACCS keys) and hashed fingerprints. They found that 2D descriptors, specifically non-hashed MACCS keys, outperformed both basic 3D descriptors such as distance between atoms in pairs and triplets as well as hashed 2D fingerprints. Despite these results, many other types of 3D descriptors have been studied.

One type of 3D descriptor is calculated using *molecular alignment*, which compares two molecules to find a common 3D orientation. The best alignment is used and a scoring function returns the degree of similarity of two molecules. A review of molecular alignment techniques is given in Lemmen and Lengauer (2000).

One of the most recent approaches to building 3D descriptors uses TPs. A TP fingerprint can be generated using the spatial locations of the TPs of a molecule (Pickett et al. (1996)). An extensive review of 3D pharmacophores in Mason et al. (2001) describes other approaches to building 3D descriptors including ones generated from both the ligand and target sites. Mason et al. (2001) also reviews 3D pharmacophores that account for flexibility (or conformations) in the molecule as well as methods of obtaining TP fingerprints.

2.2 Graph Representations

In this Section, we discuss several ways to represent a molecule as a graph. In our research, we use several graph representations with graph kernels. Using this approach, features are extracted from the graphs when the graph kernel model is computed, instead of during earlier data-processing stages. We will discuss three representations including molecular, reduced and topological pharmacophore graphs.

2.2.1 Molecular Graph

A molecular graph is a natural way to represent a molecule. In a molecular graph each vertex represents an atom, and the edges represent bond types. A molecular graph, G , is a type of labeled, directed graph consisting of a set of vertices (atom types), \mathcal{V} , and edges (bond types), \mathcal{E} , of size $|\mathcal{V}|$ and $|\mathcal{E}|$. We describe molecules as directed rather than undirected graphs as this is required for graph kernels that count walks in the graph. This adjustment is made by inserting two opposing, directed edges for each edge (bond). The adjacency matrix, A , of graph G is a $|\mathcal{V}|$ by $|\mathcal{V}|$ matrix. An entry in $A_{i,j} > 0$ (and consequently $A_{j,i} > 0$) if there is an edge from vertex v_i to v_j and 0 otherwise. The type of bond (or edge label) is stored in $A_{i,j}$, by setting $A_{i,j} = 1$ for single bonds, 2 for double bonds and 3 for aromatic bonds. All of the information needed to describe a molecular graph is given in the adjacency matrix augmented with edge labels, along with a list of vertex labels.

In Figure 2.3, we show two molecular graphs of an Aspirin molecule: one without hydrogen and one with hydrogen. The Simplified Molecular Input Line Entry System (SMILES) representation is a string representation of a molecular graph (Leach and Gillet (2003)). The SMILES representation and MOL files often do not include hydrogen atoms as they are implicitly represented. We show examples of molecular graphs with and without hydrogen atoms, as we consider both types in our research. By adding Hydrogen molecules to the molecular graph the size has increased. The pharmacophore features discussed in the previous section capture the role of hydrogens in the molecule.

2.2.2 Reduced Graph

Reduced graphs are another representation of molecules as a graph. A reduced graph presents a functional view of a molecular graph, by collapsing groups of atoms into single nodes that represent binding groups within the molecule. Reduced graphs were originally developed for structure and substructure searching of generic chemical structures (Gillet et al. (1991)). Later, reduced graphs have been used in similarity searching (Takahashi et al. (1992); Gillet et al. (2003)) and QSAR (Birchall et al. (2008b,a)).

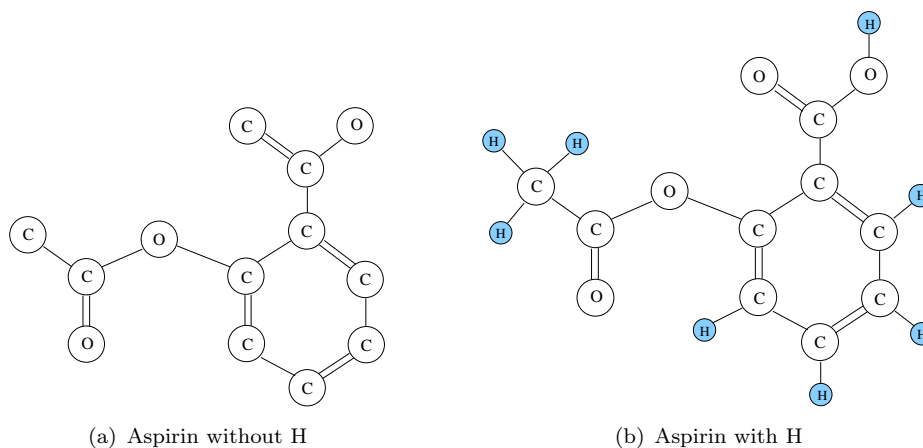


FIGURE 2.3: This figure shows the molecular graph of an aspirin molecule without hydrogen atoms on the left, and with hydrogen atoms on the right. It is drawn as a labeled, directed graph. Vertex labels consists of atom types C (Carbon), O (Oxygen) and H (Hydrogen). The hydrogen atoms are coloured blue to highlight the difference between the two graphs.

We first introduce some chemical terms used to describe Topological Pharmacophores. An *Aromatic ring* is a planar ring of six Carbon atoms known as Benzene. It consists of alternating single and double bonds. Other types of ring are *Aliphatic rings*. Finally, another type is a 5 atom ring which contains a *heteroatom* (which is not a Carbon or Hydrogen atom) such as Furan which contains a single Oxygen and 4 Carbon atoms. *positively ionizable* and *negatively ionizable* refer to atoms that have a positive or negative charge, which means each atom has an extra electron or is missing an electron. A *donor* or *acceptor* feature is an area of the molecule which has an extra electron or requires an electron that is used in a hydrogen bond.

There are several different variations of reduced graphs. We use the reduced graph definition of Harper et al. (2004) that is based on the Ar/F(4) definition in Gillet et al. (2003). The Ar/F(4) definition gives the most specificity among other reduced graph definitions given in Gillet et al. (2003). Both aromatic and aliphatic rings in a molecular graphs are collapsed into a single aromatic or aliphatic vertex type. Also, other functional areas of the molecule (composed of several vertices) are collapsed into a single vertex. Further features are built into each reduced graph vertex by defining whether it is hydrogen bond donor, hydrogen bond acceptor, donor and acceptor, or neither. Harper et al. (2004) extend this definition by including positively and negatively ionizable features, allowing each vertex to be further distinguished based on this feature. The full range of possible combinations is given in Table 2.1. A further possible vertex type is the linker node (named Zn in Table 2.1). It is an area of the molecule that does not match any of these features but connects them and so it is useful in defining some structural information in the reduced graph. Each of the possible combinations is assigned a unique label. The naming convention in Harper et al. (2004) uses atom names to describe combinations of

features. Using atom names for vertex labels allows these reduced graphs to be used in chemical graphing software such as ACD/3D Viewer² and ISIS/Draw³. Furthermore, we use this naming convention as the reduced graphs were generated from molecular graph SMILES strings, using GSK’s proprietary software built with the Daylight⁴ toolkit.

Pos.	Neg.	Aromatic	Aliphatic	Donor	Acceptor	Name
		•				Sc
		•		•		Ti
		•			•	V
		•		•	•	Cr
•		•				Mn
	•	•				Fe
			•			Hf
			•	•		Ta
			•		•	W
			•	•	•	Re
•			•			Y
	•		•			Zr
				•		Co
					•	Ni
				•	•	Cu
•						Nb
	•					Mo
						Zn

TABLE 2.1: Definitions for names used to represent features in topological pharmacophore and reduced graphs consistent with Harper et al. (2004).

As the reduced graph collapses several vertices in a single vertex, the size of reduced graph is much smaller than the molecular graph. An illustration of an Aspirin molecule as both a molecular and reduced graph is given in Figure 2.4. An Aspirin molecular graph contains 13 vertices (atoms), while the reduced graph only contains 3. Clearly, the reduced graph is a fraction of the size of the molecular graph, when counting both the number of vertices and edges. Despite removing a large amount of structural information, the reduced graph is designed to retain the most information about functional areas involved in binding with the target. As we will discuss in Chapter 4, the computation of the graph kernel is quite expensive. As the reduced graph is much smaller, there is a large computational benefit, especially when using large datasets.

As described in Section 2.2.1, we use an adjacency matrix and vector label to store the structure and labels of molecular graph. We use this same structure to store reduced graph. We note that the set of vertices and edges is entirely different from the molecular graph. The label types for the reduced graph are given in Table 2.1.

²<http://www.acdlabs.com>

³<http://www.mdli.com>

⁴<http://www.daylight.com>

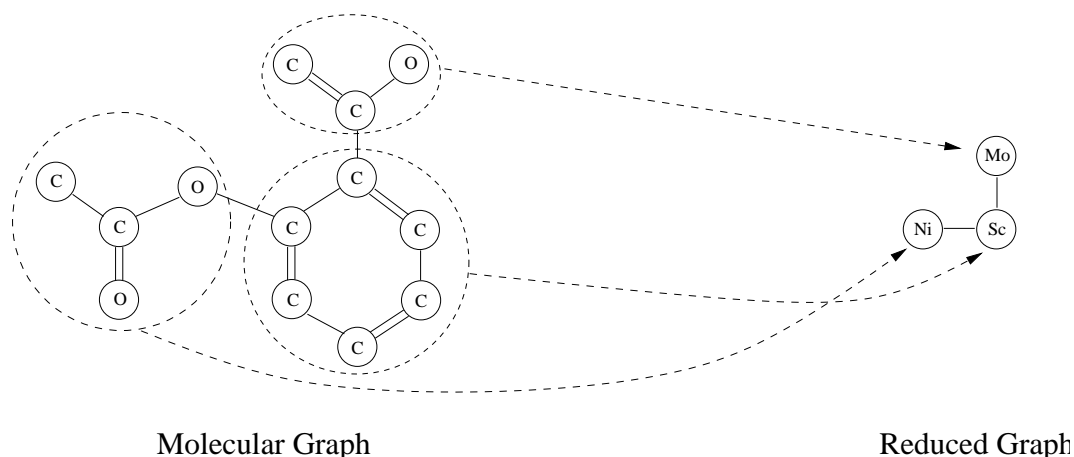


FIGURE 2.4: This figure shows a molecular graph (on the left) and a reduced graph (on the right) for an Aspirin molecule. The areas in dashed regions of the molecular graph show an area which is compacted into a single node in the reduced graph. A set of unique vertex label are used in the reduced graph to describe which combination of features are used. The meaning of the vertex labels is given in Table 2.1. Edge labels in reduced graphs are set to 1, except for fused rings such as Naphalene (containing two fused 6-atom Carbon rings), where edge labels are set to 2 for a double bond (Sc-Sc).

2.2.3 Topological Pharmacophore Graph

An alternative representation for a molecule as a graph is given by the Topological Pharmacophore (TP) graph. A Topological Pharmacophore is an area of a molecule which describe important functional aspects that may be involved in binding with a desired target. We use the same TPs that are used for the reduced graphs given in Table 2.1. The features used are: positively ionizable group, negatively ionizable group, aromatic ring, aliphatic ring, hydrogen bond donor, hydrogen bond acceptor.

The TP graph has the same structure of a molecular graph, although the vertex and edge labels are different. We note that although we described molecular graphs that include hydrogen atoms in Section 2.2.1, the implicit hydrogen atoms are not used in the TP graph though used in the feature definition. The standard view, in the SMILES format for a molecule does not include hydrogen atoms, so no modification is needed to the data.

Given a molecular graph, one can obtain a TP graph by iterating through each vertex and changing its label based on the set of features given in Table 2.1. While reduced graphs collapse rings and groups of atoms into a single node that describe a TP feature, the TP graph retains the original structure. For example, each atom/vertex within an aromatic ring would be relabeled as an aromatic atom. The edge labels are not used so they are all set to 1 in practice. The TP graphs were generated from molecular graph

SMILES representation using GSK's proprietary software and the Daylight⁵ toolkit. An example of a topological pharmacophore graph is given in Figure 2.5.

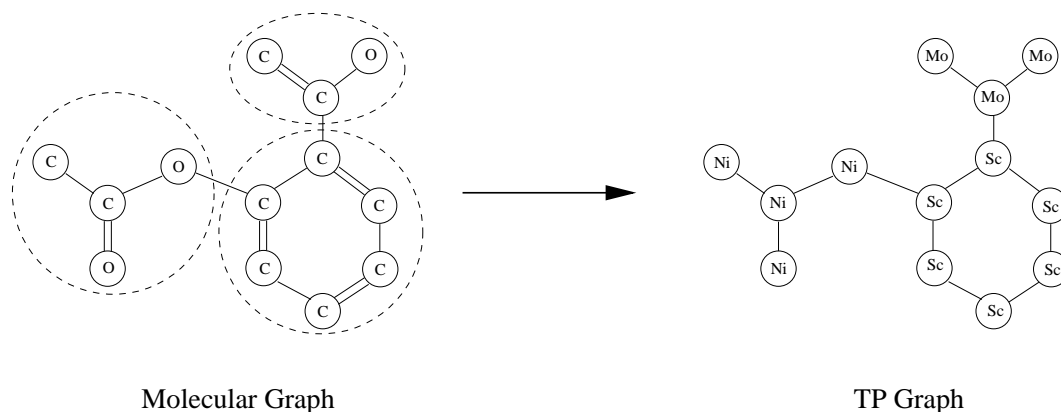


FIGURE 2.5: This figure shows the molecular graph (on the left) and TP graph (on the right) of an Aspirin molecule. A TP graph has the same structure as the molecular graph, although it contains different vertex and edge labels. Vertex labels in a TP graph use labels described in Table 2.1.

2.3 Open Issues in Representing Molecules as Graphs

In this Section, we discuss several issues involved in choosing a graph representation for a molecule. These considerations must be made in early data processing stage and must be addressed when building a dataset. We will discuss some issues including vertex colouring, usage of hydrogen atoms and aromatic bond types.

2.3.1 Colouring

Graph colouring is a technique that adds more structural information into vertex labels. As there are relatively few atom types (or vertex labels), we can increase the distinction between graphs by including some local structural information around each vertex in the labels. Graph colouring must be performed in an initial data processing step which modifies the original graphs. Graph colouring replaces the original vertex labels with colour labels containing more structural information. In recent graph kernel literature, two approaches have been suggested to colour graph vertex labels. The first method notated by *N-colour* is described in Gärtner (2005a). This method defines a coloured vertex label as the set of each neighbouring vertex labels as well as its own label. A second method is presented in Mahé et al. (2004) and uses an iterative algorithm called the Morgan index. At the first iteration when calculating the Morgan index (M0), each vertex is labeled 1. Subsequent iterations define each vertex as the sum of its label and

⁵<http://www.daylight.com>

each surrounding label. We will only use the first three iterations (denoted by M1, M2 and M3). Please see Figure 2.6 for an illustration of how Morgan and N-Colour are implemented.

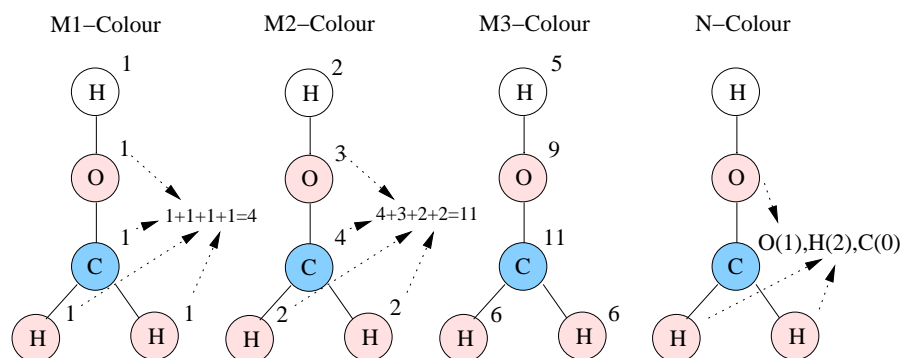


FIGURE 2.6: A small molecule example of how the N-Colour and Morgan colour algorithms work. This figure shows how a coloured label using each colouring algorithm is calculated for the Carbon atom (in blue). The Carbon atom is neighboured by 3 other atoms (in red). For M1-Colour, the coloured label is C,1, M2-Colour is C,4, M3-Colour is C,11 and N-Colour is C,0(1),H(2),C(0)

After determining the coloured labels, each vertex in the graph is re-labeled with a new unique identifier given by both the coloured and No-Colour label. This allows information from the atom label and local structural information to be incorporated in the new vertex label.

By generating a coloured label, both N-Colour and Morgan colour, incorporate local structural information in the vertex labels. The coloured label provides greater discrimination between graphs and more unique vertex labels. This modification has good computational benefits as there are less matches between graphs.

We have only considered colouring of vertex labels, although we note that another possibility would be to add colouring to edge labels in a similar manner. We will only experiment with vertex colouring in our work although structural information could also be added to edge labels. Furthermore, other properties such as whether an edge is a rotatable bond could be added into a coloured edge label. Rotatable bonds are very useful information in a molecular graph as they allow a molecule to change shape and assume different conformations.

2.3.2 Explicit Hydrogen Atoms

In Section 2.2, we show how a molecular graph can be drawn with and without Hydrogen atoms. The SMILES format describes molecules without Hydrogen atoms, although different publicly available datasets sometimes include Hydrogen atoms. Although it is possible to modify the data from one format to the other, the two publicly available datasets used in Chapter 5 differ in their use of Hydrogen atoms. In the MuTag dataset,

the data gave molecules with Hydrogen atoms in the MOL file, while in the NCI-HIV dataset they were not included. Despite this discrepancy, the most common format, SMILES, does not include Hydrogen atoms as they are implicitly represented in the structure. Using the Daylight software toolkit it is possible to rebuild the graphs to include hydrogen atoms. This procedure was used to make both types of representations so that this difference could be analyzed. We note that Hydrogen atoms typically comprise a large proportion of the graph (about a third) and by including them in our graphs the kernel calculation becomes less efficient. Furthermore, Hydrogen atoms are implicitly described in the molecular graph.

2.3.3 Aromatic Bonds

A final consideration is the notation used when labeling aromatic bonds. Although an Aromatic ring can be drawn with alternating single and double bonds, the placement of these bonds is arbitrary and depends on how the 2D molecular graph is drawn. We use the approach of creating a new bond type for aromatic bonds to resolve this issue.

2.4 Summary

In this Chapter we have reviewed several types of 1D, 2D and 3D descriptors which are the most common method of representing a molecule. Afterwards, we introduced three representations of a molecule as a graph: molecular (MG), reduced (RG) and topological pharmacophore (TP) graph. We will use these representations in conjunction with graph kernels in order to compare molecules for use in QSAR analysis. We concluded by mentioning several features that must be considered when choosing a graph representation. Although some of these features may be beneficial when classifying drugs, one must also consider the impact on space and time when calculating the graph kernel.

Chapter 3

A Review of Graph Kernels

Graphs are a very general data structure that are useful in modeling many types of data. A graph consists of a set of vertices and relationships between these vertices modeled by edges. Although some research in kernel methods use graphs to describe the input space (where each vertex in a graph represents an example (Kondor and Lafferty (2002))), we use graphs to model the structure within a single example or molecule. We reviewed three types of graph-based representations of molecules in Chapter 2: the molecular graph (MG), reduced graph (RG) and topological pharmacophore (TP) graph. The most well known of these is the molecular graph which has vertices for atoms and edges for bonds.

In this Chapter, we describe how a positive-definite kernel function, or *graph kernel* can be calculated directly using the structure of the graph. We begin by reviewing notation for labeled, directed graphs in Section 3.1. Computational issues involved in calculating graph kernels are presented in 3.2. A review of several different graph kernels is organized based on their use of graph features used to formulate the kernel in Section 3.3. Finally, in Section 3.4 we review a number of recent advances towards designing graph kernels for molecules.

3.1 Labeled, Directed Graphs

In order to describe graph kernels, we first introduce some notation that is used throughout this Chapter and the rest of the thesis. A graph G is composed of a set of vertices, \mathcal{V} and edges, \mathcal{E} , of size $|\mathcal{V}|$, $|\mathcal{E}|$ respectively. When differentiating between graphs and its elements we subscript each so a graph, G_i , has elements \mathcal{V}_i and \mathcal{E}_i . The vertex set, \mathcal{V} , is composed of elements $\{v_1, v_2, \dots, v_{|\mathcal{V}|}\}$, while an edge from vertex v_i to v_j is described as $(v_i, v_j) \in \mathcal{E}$ or alternatively the set of by a set of edges: $\{e_1, e_2, \dots, e_{|\mathcal{E}|}\} \in \mathcal{E}$.

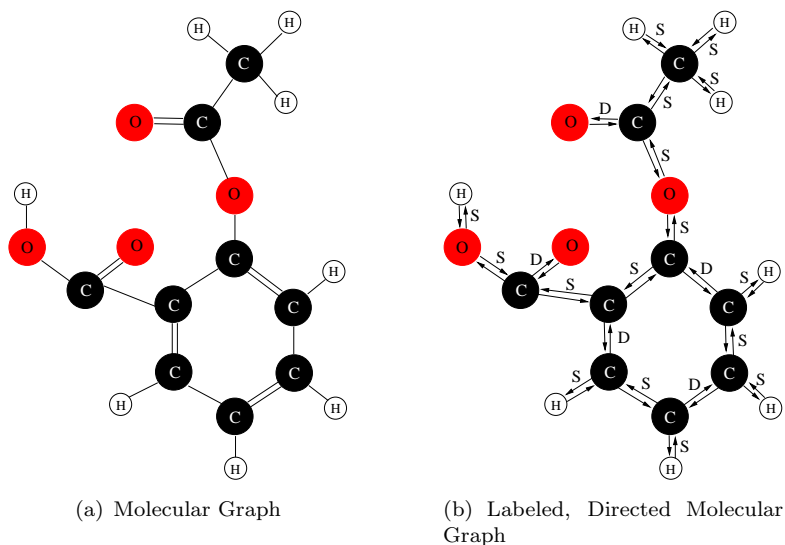


FIGURE 3.1: This figure shows the molecular graph of an Aspirin molecule in 3.1(a) and the same graph with labeled, directed edges in 3.1(b). In this example, vertex labels include atom types: Carbon (C), Oxygen (O) and Sulphur(S). The edge labels are S for single bonds and D double bonds.

A *labeled, directed graph* has a set of labels, \mathcal{L} , along with a function, `label`, that assigns labels to each vertex and edge. Graphs without labels can be considered a special case where the same label is assigned to every edge. We will consider graphs which contain *directed* edges where an edge contains additional information about its starting and ending vertex, as graph kernels need this description to obtain certain features from the graph. This directional information does not occur naturally in molecular graphs although it is easy to convert from an undirected to directed graph by replacing each undirected edge with two opposing directed edges. Please see Figure 3.1 for an example.

Some important graph theory concepts are used to calculate graph kernels include *walks*, *paths* and *cycles*. A *walk* is a sequence of vertices, v_1, v_2, \dots, v_n , where $v_i \in \mathcal{V}$ and $(v_i, v_{i+1}) \in \mathcal{E}$. Here, the length of the walk is the number of edges, given by $n - 1$ in the preceding example. A *path* is a walk with no repeated vertices, i.e. $v_i \neq v_j$ except where $i = j$. A *cycle* is a path where the last vertex connects with the first or $(v_1, v_n) \in \mathcal{E}$.

Finally, we mention two other special-cases of labeled, directed graphs, that can be used to calculate graph kernels: *trees* and *strings*. A rooted *tree* is a type of hierarchical structure. The vertices of a tree are described as either *root* nodes or *leaf* nodes. A tree contains only one *root* node at the highest level in the hierarchy. Subsequent lower-level leaf nodes are connected by *branches* (or edges). A *child* node is one level below its *parent* node and two levels below its grandparent node. Another special-case of graphs are *strings*. A string is a sequence of characters from some alphabet. Although strings contain little structure, they can be expressed as a graph by considering each character

in a string as a vertex and edges connect other characters immediately before and after in the string.

3.2 Complexity of Graph Kernels

Before addressing methods of calculating a graph kernel, we discuss the complexity of graph kernels. One of the first papers on graph kernels included a discussion on complexity associated with calculating a *complete* graph kernel. A complete graph kernel is one which uses all possible sub-graphs as features (Gärtner et al. (2003)). This kernel would follow the convolution kernel approach (Haussler (1999)) to evaluate the similarity between all sub-features of the graph. If we consider a feature vector $\phi(G)$ that has a feature for every possible sub-graph. An inner product between these features can be used to identify whether a graph has a *hamiltonian path* (Gärtner et al. (2003)). A hamiltonian path is a path that visits each vertex in a graph exactly once. This problem is known to be NP-hard on even small graphs (Skiena (1997)) and therefore a graph kernel using sub-graphs as features is not tractable. As a result, one must take a different approach that uses features other than every possible sub-graph.

3.3 Approaches to Calculating Graph Kernels

In this Section, we review several ways of calculating a graph kernel. A positive-definite function between two labeled graphs, G_1 and G_2 , is known as a *graph kernel*. We use κ to denote a kernel function, so a graph kernel is given by $\kappa(G_1, G_2)$.

As described in Shawe-Taylor and Cristianini (2004), the design of kernel function is an important step in using a kernel learning algorithm. A kernel function defines the similarity between two examples, and is the learning algorithm's only view of the data. One can choose a kernel function that weights different features accordingly, and as a result, one can decide what similarity is important for comparing the data. When normalized, a kernel function returns 0 if two graphs show no similarity. A value of 1 indicates high similarity and would imply that these graphs are implicitly mapped to the same point in the kernel feature space.

In order to build a graph kernel by following the convolution kernel approach, one must enumerate all 'sub parts' of two graphs and return the cardinality of the intersection of these two sets. As described in the previous section, using the sub-graphs is not the best choice, therefore one must use other features to define the 'sub parts'. Several graph kernels have been developed using different features including walks, label distances, trees, and cycles. We will begin our discussion by describing a trivial symbolic graph

kernel. Later, we will introduce several classes of graph kernels that are grouped based on their use of features.

3.3.1 A Trivial Symbolic Kernel

One of the simplest graph kernels, introduced in Kashima and Inokuchi (2002), is a trivial symbolic graph kernel that uses counts of vertex labels as features. This kernel differs from structure kernels described in later sections as we use a feature vector to describe the graph. A feature vector of a graph $\phi(G)$ is composed of elements which each describe counts of unique vertex labels (l_i). One of these elements is calculated using:

$$\phi_i(G) = \sum_{v_j \in \mathcal{V}} 1_{\text{label}(v_j)=l_i} \quad (3.1)$$

The indicator function 1 , returns 1 if the unique label \mathcal{L}_i matches a vertex of the graph, v_j . The kernel is the inner product between the feature vector for two graphs, G_1 and G_2 :

$$\kappa(G_1, G_2) = \langle \phi(G_1), \phi(G_2) \rangle \quad (3.2)$$

This kernel counts the number of labels in common between two graphs. A similar function could count edge labels or other simple counts. However, much more expressive kernel functions can be calculated using the structural information in the graph.

3.3.2 Walks

One of the most popular approaches to calculating a graph kernel uses the number of matching random walks between two graphs. This type of graph kernel has been termed a contiguous label sequence graph kernel in Gärtner et al. (2003) as a walk in a graph does not skip any vertices when walking from one vertex to the next in a graph. We will use the term *walk-based graph kernel* to refer to this type of kernel. Two approaches to calculating a walk-based graph kernel were introduced independently: one that uses a marginalized kernel and one that uses a product graph approach. We will describe each of these and discuss how they are equivalent.

The first walk-based approach to calculating a graph kernel we consider was introduced in Kashima and Inokuchi (2002) and Kashima et al. (2003). This approach counts all random walks that match between two graphs up to infinite length by down-weighting longer walks using a marginalized kernel. A walk is described using several probabilities

that determine its starting vertex, transitions and length. The starting probability, P_s , is defined with equal probability for all vertices, allowing the random walk to start from any vertex. A walk transitions from one vertex to another with probability P_t . This is defined with equal probability for all adjacent vertices. Finally, the length is described by a stopping probability, P_q . The probability of a walk, \mathbf{w} , of length p is given by:

$$P(\mathbf{w}|G) = P_s(w_1) \prod_{i=2}^p P_t(w_i|w_{i-1}) P_q(w_p) \quad (3.3)$$

We must also define a kernel κ_w between two walks, which is the joint probability in the marginalized kernel. We use the notation \mathbf{w}, \mathbf{w}' for walks from G_1, G_2 respectively. We first define an indicator function for vertex labels, $\kappa_{\mathcal{L}}(w_i, w'_i)$, which returns 1 if w_i and w'_i have the same vertex label, and 0 otherwise. Similarly, we define an indicator function for edge labels, $\kappa_{\mathcal{L}}((w_i, w_{i+1}), (w'_i, w'_{i+1}))$, which returns 1 if given the same edge labels and 0 otherwise. We can now define a kernel between two walks:

$$\kappa_w(\mathbf{w}, \mathbf{w}') = \begin{cases} 0 & (\mathbf{p} \neq \mathbf{p}') \\ \kappa_{\mathcal{L}}(w_1, w'_1) \prod_{i=2}^p \kappa_{\mathcal{L}}((w_{i-1}, w_i), (w'_{i-1}, w'_i)) \kappa_{\mathcal{L}}(w_i, w'_i) & (\mathbf{p} = \mathbf{p}') \end{cases} \quad (3.4)$$

Finally, the marginalized walk-based kernel is given for two graphs which uses the definitions given in eq (3.3) and (3.4). This kernel calculates all walks, \mathbf{w}_1 and \mathbf{w}_2 , that match from graphs G_1 and G_2 respectively.

$$\kappa(G_1, G_2) = \sum_{p=0}^{\infty} \sum_{\mathbf{w}_1} \sum_{\mathbf{w}_2} \kappa_w(\mathbf{w}_1, \mathbf{w}_2) P(\mathbf{w}_1|G_1) P(\mathbf{w}_2|G_2) \quad (3.5)$$

Some additional constraints must be set to ensure that P_s and P_t of eq (3.3) gives a probability distribution. This is done by defining a parameter, $0 < P_q < 1$, and setting $P_s = \frac{P_q}{|V|}$. Similarly, we define P_a by setting $\sum_{u \in \mathcal{V}} P_a(u|v) = 1$ for all neighbouring edges. We set the following constraints for P_t to ensure a probability distribution: $P_t(u|v) = \frac{1-P_q}{P_q} P_a(u|v) P_q$.

Since the path lengths considered span from 1 to ∞ , one cannot calculate this kernel directly in this formulation. In Mahé et al. (2004), a method is given to calculate this kernel efficiently with product graphs and matrix inversions which is similar to another walk-based kernel (Gärtner et al. (2003)) that we describe later. Given $G_1 = (\mathcal{V}_1, \mathcal{E}_1)$ and $G_2 = (\mathcal{V}_2, \mathcal{E}_2)$, we construct a product graph, $G_1 \times G_2$. Vertices for G_1, G_2 are subscripted by 1 and 2 respectively. An edge in the product graph connects vertices (u_1, u_2) and (v_1, v_2) if the edge label $(u_1, v_1) = (u_2, v_2)$. The starting probability of a

vertex, (u_1, u_2) , in the product graph is defined by:

$$\pi_s(u_1, u_2) = P_s(u_1)P_s(u_2) \quad (3.6)$$

where $P_s(u)$ is a uniform probability for all vertices in G_1 and G_2 . By substituting the constraints to make P_s a probability distribution described above, π_s can be described in terms of P_q .

$$\pi_s = \frac{P_q^2}{|\mathcal{V}_1||\mathcal{V}_2|} \quad (3.7)$$

The transition probability of a walk in the product graph is defined by:

$$\pi_t((v_1, v_2)|(u_1, u_2)) = P_t(v_1|u_1)P_t(v_2|u_2) \quad (3.8)$$

where $\mathcal{V}_1, \mathcal{V}_2$ are from G_1, G_2 respectively. The walk transitions, from one product vertex v_i to vertex v_j , are recorded in a transition matrix, Π_t . Similarly, using the constraints given earlier for P_t , this transition probability over product vertices is described in terms of P_q alone.

$$\Pi_t(v_i|v_j) = \frac{(1 - P_q)^2}{\text{degree}_{G_1}(v_i)\text{degree}_{G_2}(v_i)} \quad (3.9)$$

where $\text{degree}_{G_1}(v)$ refers to the degree of a vertex in G_1 that is from the product vertex v . The kernel is defined using a functional describing every possible walk, $H(G_\times)$, of length $n = 1$ to ∞ :

$$\kappa(G_1, G_2) = \sum_{n=1}^{\infty} \left(\sum_{h \in H(G), |h|=n} \pi_s^\top \Pi_t^n \mathbf{1} \right) \quad (3.10)$$

$$= \pi_s^\top (I - \Pi_t)^{-1} \mathbf{1} \quad (3.11)$$

This method of taking the inverse of the product graph was proposed independently as a method of calculating walk-based graph kernels in Gärtner et al. (2003). A kernel between G_1 and G_2 is calculated by constructing a product graph and then obtaining the exponential or geometric convergence of the adjacency matrix of this product graph when multiplied by a certain weights.

The product graph, $G_1 \times G_2$, is constructed using:

$$\begin{aligned}
\mathcal{V}_\times(G_1 \times G_2) &= \{(v_1, v_2) \in \mathcal{V}_1 \times \mathcal{V}_2 : (\text{label}(v_1) = \text{label}(v_2))\} \\
\mathcal{E}_\times(G_1 \times G_2) &= \{((u_1, u_2), (v_1, v_2)) : \\
&\quad (u_1, v_1) \in \mathcal{E}_1 \wedge (u_2, v_2) \in \mathcal{E}_2 \wedge (\text{label}(u_1, v_1) = \text{label}(u_2, v_2))\}
\end{aligned}$$

As described in previous equations, the vertex set of the product graph, \mathcal{V}_\times , creates a new product vertex for each vertex that has matching labels in G_1 and G_2 . An edge in the product graph, \mathcal{E}_\times , between two product vertices is created if the corresponding vertices in G_1 and G_2 have an edge and if they have the same label. Please see Figure 3.2 for an example of how a product graph is constructed.

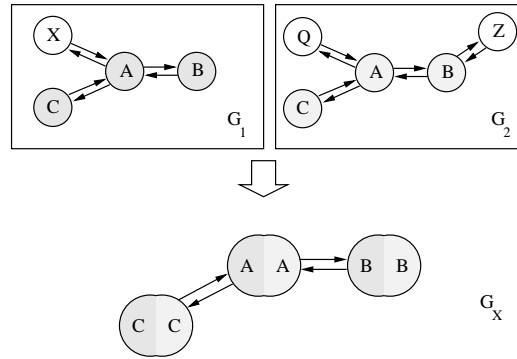


FIGURE 3.2: **Construction of a Product Graph** - A product graph, G_\times , is constructed from two graphs, G_1 and G_2 . Each vertex in G_1 is matched with vertices in G_2 . Edges are added between product vertices if they exist in the original graphs. The shading and labels in the product graph shows which product vertices correspond to the vertices in G_1 .

The adjacency matrix of the product graph is notated by E_\times . The i, j -th element of E_\times is 1 if product vertex v_i is connected product vertex v_j and is zero otherwise. In order to calculate all walks up to infinite length, the following graph-theoretic property is used. If E_\times is taken to the power n , E_\times^n , each i, j th element now lists the number of walks of length n starting at vertex v_i and ending at v_j . By defining a sequence of weights, λ ($\lambda_0, \lambda_1, \dots$), we are now able to calculate a walk-based graph kernel:

$$\kappa_\times(G_1, G_2) = \sum_{i,j=1}^{|\mathcal{V}_\times|} \left[\sum_{n=0}^{\infty} \lambda_n E_\times^n \right]_{ij} \quad (3.12)$$

if the limit exists. The outer sum of eq (3.12) returns a single value, summing all values of the converged matrix. Choices of λ allow the inner sum in eq (3.12) to be expressed as either an exponential or geometric series expansion. To reach a convergence with a geometric series, we must choose a parameter γ , which must satisfy

$\gamma < 1/\min(\text{degree}(G_1), \text{degree}(G_2))$. The function **degree**, gives the maximum number of connected edges to any single vertex (calculated as a column or row sum of the adjacency). The limit is computed by inverting the matrix: $(\mathbf{I} - \gamma E_{\times})^{-1}$. This calculation takes $O(|V_{\times}|^3)$. In practice, for molecular graphs, this is very expensive as there are relatively few atom types, so a very large product graph is constructed.

When deciding whether to use the geometric or exponential convergence to find all matching walks summed to infinity, one should first consider the graphs in Figure 3.3. Here, a graph is given that compares the relative importance of walk lengths for several graph kernel parameters. For exponential series, we evaluated walk lengths $1, \dots, 20$ along with exponential parameters 3, 5 and 7, and used the formula presented in Gärtner et al. (2003) for exponential series convergence $\frac{\alpha^i}{i!}$. For geometric series, we again used walk lengths $1, \dots, 20$, and used geometric parameters $\frac{1}{3}, \frac{1}{5}, \frac{1}{7}$, and used the geometric term for convergence α^i . All values are normalized between 0 and 1 allowing for a 1:1 comparison, which gives the relative importance between parameters and individual walk lengths. As shown in Figure 3.3, more information is taken from longer walks with an exponential weight than geometric weights.

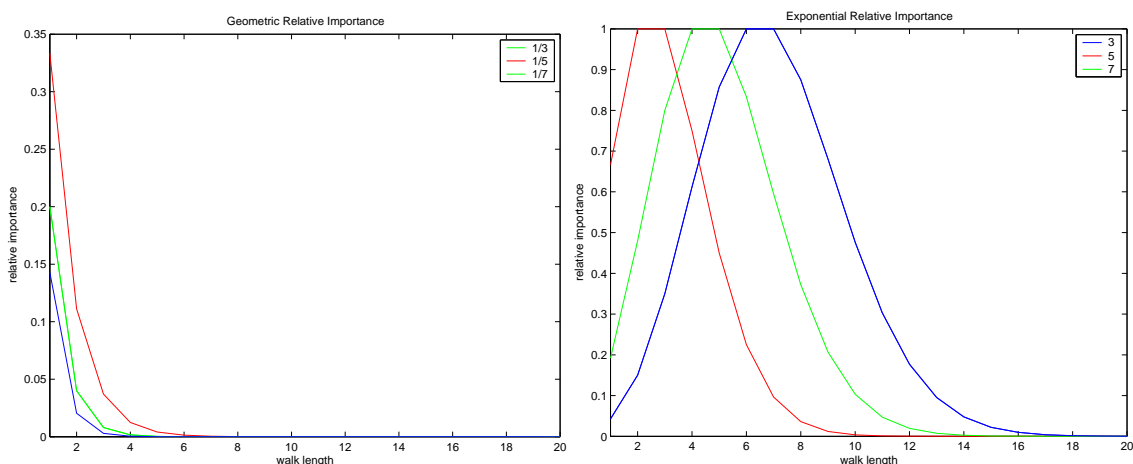


FIGURE 3.3: These figures show the relationship between the user defined parameter γ (used to define the convergence of the walk kernel) and the relevance given to different walk lengths. Different weightings may occur for the exponential and geometric series, although walks longer than 10 account for little to the kernel as they are down-weighted so greatly.

In recent work (Gärtner (2005b)), the exponential parameter has often been used, additionally the relative importance of walks when calculating the convergence of the marginalized kernel is similar to the Geometric Relative Importance in Figure 3.3.

It is interesting to note that although each kernel in Kashima et al. (2003) and Gärtner et al. (2003) are formulated differently, they can be computed with the same method. Graph kernels are calculated using the number of matching walks up to infinite length by down-weighting longer paths. The only difference between these is the down-weights

used in order to find a convergence. A discussion comparing these methods is given in Tsuda and Vert (2004).

3.3.3 Extensions to Walk Kernels

Two extensions to walk-based kernels have been proposed: one counts non-contiguous walks and the other non-tottering walks.

The first extension we will consider is *non-contiguous* walks. A non-contiguous walk considers the similarity when walks contain a vertex label that does not match. For example, let's consider two walks of length 3: *CCC* and *CSC*. Using a contiguous walk kernel, this feature would not be matched. Despite not being a perfect match, they are only different by one character. A non-contiguous kernel will match these walks, although a penalty is applied to the second character which has different labels, sometimes referred to as a *soft match*. Clearly, parallels can be drawn with Lodhi et al. (2002) which considers similar features in string kernels. A non-contiguous walk kernel was first introduced in Gärtner et al. (2003) and is calculated in a similar way to the contiguous walk kernel.

To define this kernel we now introduce an alternative product graph, G_o , that includes vertices with non-matching labels. The set of vertices and edges of G_o is given by:

$$\begin{aligned}\mathcal{V}_o(G_1 \times G_2) &= \{(v_1, v_2) \in \mathcal{V}_1 \times \mathcal{V}_2\} \\ \mathcal{E}_o(G_1 \times G_2) &= \{((u_1, u_2), (v_1, v_2)) : \\ &\quad (u_1, v_1) \in \mathcal{E}_1 \wedge (u_2, v_2) \in \mathcal{E}_2 \wedge (\text{label}(u_1, v_1) = \text{label}(u_2, v_2))\}\end{aligned}$$

To define this kernel, we must also introduce a parameter, α (where $0 \leq \alpha \leq 1$), to penalize soft matches. α allows the kernel to choose the amount of relevance taken from the contiguous walks in E_\times and non-contiguous in E_o . We can construct E_\times to have the same size as E_o by adding corresponding empty rows and columns where there is no corresponding vertex in E_\times . This modification is necessary to calculate the kernel as these two matrices will be added. The non-contiguous label sequence kernel is defined as:

$$\kappa_o(G_1, G_2) = \sum_{i,j=1}^{|\mathcal{V}_\times|} \left[\sum_{n=0}^{\infty} \lambda_n((1-\alpha)E_\times + \alpha E_o)^n \right]_{ij} \quad (3.13)$$

if the limit exists. As shown by the outer sum of eq (3.13), the kernel only counts *anchored* walks, where a walk begins and ends with matching product vertices. A

matching product vertex is one which will have the same label in G_1 and G_2 , therefore it will exclusively be in E_\times and not one which was created with mismatched vertices that may exist in E_o . Although this kernel gives a sum to ∞ , it is calculated using the same method as the walk kernel discussed in the previous section. It can be calculated using either an exponential or geometric convergence. With a geometric convergence the kernel is calculated as: $[(\mathbf{I} - \gamma)((1 - \alpha)E_\times + \alpha E_o)]^{-1}$.

A second extension to walk-based graph kernels was presented in Mahé et al. (2004) that removed *tottering* walks. A tottering walk is one of the form $\mathbf{w} = w_1, \dots, w_n$ with $w_i = w_{i+2}$, so a walk will move to a new vertex and immediately return to the previous vertex. Intuitively, it seems that this type of walk will contain less information. For example, if we consider a very large graph, a walk which moves back and forth between only two vertices, will only describe a small part of the graph. In Mahé et al. (2004) showed that totters could be removed from a walk-based graph kernel.

3.3.4 Label Distances

A second class of graph kernels uses distances between labels as features. We present the method given in Gärtner et al. (2003). We note that a similar kernel is defined in Borgwardt and Kriegel (2005), although it is faster to compute using Gärtner et al. (2003) so we only review this method.

A *label pair* is the distance (in edges) between any two vertices in a graph. In order to calculate a label distance based graph kernel, we define the following: a *label matrix*, L . It is of size $l \times |V|$ matrix. l is the number of unique vertex labels and $|V|$ is the number of vertices in the graph. An element in this label matrix, $L_{i,j}$, is 1 if the label of vertex v_i is the type of unique label given by column j . For a given row, every column will be 0, except for the correct column which is the type of vertex label for the vertex represented by the row.

By multiplying the adjacency matrix, E , by the label matrix, L , and L^T , the resulting matrix $[LE^n L^T]_{ij}$, has an element i, j that correspond to the number of walks of length n between vertices labeled l_i and vertices labeled l_j . The kernel is defined as:

$$\kappa(G_1, G_2) = \left\langle L_1 \left(\sum_{i=0}^{\infty} \lambda_i E_1^i \right) L_1^T, L_2 \left(\sum_{j=0}^{\infty} \lambda_j E_2^j \right) L_2^T \right\rangle_F \quad (3.14)$$

where F indicates the Frobenius norm. The Frobenius norm is a pairwise inner-product between elements in two matrices (A and B):

$$\langle A, B \rangle_F = \sum_i \sum_j a_{ij} b_{ij} \quad (3.15)$$

The matrix $[LE^\infty L^T]_{ij}$, is a matrix order $l \times l$. The kernel in eq (3.14), includes a sum to ∞ , although the convergence can be found using an exponential or geometric series. As the features for two graphs can be calculated independently, the features can be pre-computed before running the learning algorithm for a fast computation.

3.3.5 Trees

Another class of kernels uses tree patterns as features. This was first presented in Ramon and Gärtner (2003). A tree pattern is one that is constructed by selecting a vertex in a graph as the root node and then recursively adding leaves to the tree according to each neighbouring vertex in the graph. We use the notation given in Mahé and Vert (2006) to describe tree kernels. A function, ψ_t , counts the number of times a tree-pattern of depth t occurs in a graph. The kernel function counts tree patterns between two graphs, G_1 and G_2 as defined by:

$$\kappa(G_1, G_2) = \sum_{t \in \tau} \lambda(t) \psi_t(G_1) \psi_t(G_2) \quad (3.16)$$

where τ is a set of trees from all possible graphs of depths from 1 to ∞ . The λ function is a parameter that sets trees of greater depth to have smaller values, by setting $\lambda(t) = \alpha^t$ where α is between 0 and 1. In practice, the set of tree fragments, τ , can be restricted to a finite depth as in Mahé and Vert (2006). As the labeled, directed molecular graph contains many cycles (two vertices contain opposing directed edges which form cycles), the largest-depth tree pattern will be infinite so this restriction must be made.

3.3.6 Cycles

A final class of graph kernels uses cycles as features. A graph kernel that uses features from both simple cycles and trees in Horvath et al. (2004). A *simple cycle* is a cycle that contains no repeated vertices. The first step to calculating this kernel involves decomposing the graph into bi-connected components and bridges using a depth-first search algorithm described in Tarjan (1972). A *bi-connected* component is a sub-graph which cannot be disconnected (split into two sub-graphs) by removing any vertex from this sub-graph. A *bridge* is an edge in a graph which when removed increases the number of connected components. We give an example of a graph that shows all bi-connected components and bridges in Figure 3.4.

After generating the bi-connected components and bridges of a graph, the simple cycle and tree patterns can be obtained. The set of simple cycles, $S(G)$, is enumerated for a graph, G , from the bi-connected components. The set of tree patterns, $T(G)$, is calculated by generating trees rooted at each bridge. The kernel uses both the number

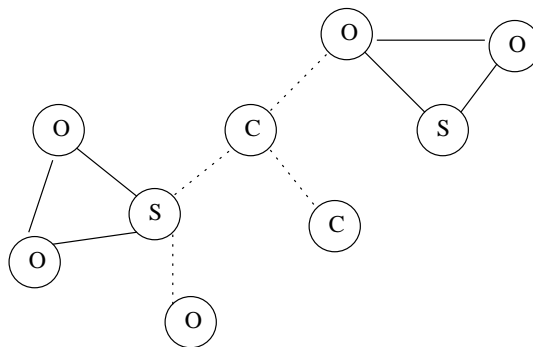


FIGURE 3.4: A graph is shown along with its associated bi-connected components and bridges. The bridges are the edges given by dashed lines, while the bi-connected components are given by solid lines. These are two cycles (containing vertex labels O,O,S) and three single vertices O,C and C.

of simple cycle, $S(G)$, and tree patterns, $T(G)$. This approach differs slightly from other graph kernel research by describing the kernel using an *intersection kernel*, which is a set kernel that operates on structured data. The kernel is calculated by the intersection or number of matching simple cycles $S(G)$ and tree patterns $T(G)$ between two graphs as given by:

$$\kappa(G_1, G_2) = |S(G_1) \cap S(G_2)| + |T(G_1) \cap T(G_2)| \quad (3.17)$$

3.4 Graph Kernels for Molecules

Several recent papers have used graph kernels to classify molecules. Each of the types of graph kernels discussed in Section 3.3 have been tested with molecular datasets. In this Section, we review the kernels in this Chapter that have been applied to molecular data. We also describe some other kernels as well as related approaches that do not fit into a structured kernel framework.

The walk-based kernels formulated as marginalized kernels in Kashima and Inokuchi (2002) and Kashima et al. (2003), were tested with and without the non-tottering extensions in Mahé et al. (2004) as well as Mahé et al. (2005). These kernels were tested on two very small, publicly-available datasets: MuTag and PTC. The results showed that kernels with only non-tottering features performed better than graph kernels that included tottering features.

The walk-based kernels developed in Gärtner et al. (2003) were later tested on the much larger, publicly available NCI-HIV dataset in Gärtner (2005a). These walk kernel outperformed cyclic pattern kernels from Horvath et al. (2004) as well as a frequent sub-graph approach. The frequent sub-graph approach was developed in Deshpande et al.

(2002) and Deshpande et al. (2003) and creates feature vectors that indicate the presence of frequently occurring sub-graphs. Despite the complexity involved in enumerating all possible sub-graphs, when only enumerating up to a certain threshold this approach is tractable. Results in Horvath et al. (2004) also confirm those in Gärtner (2005a), suggesting that the cyclic pattern kernel outperforms a frequent sub-graph approach on the NCI-HIV dataset.

The label-distance kernel defined in Gärtner et al. (2003) remains untested on molecular datasets, as well as the similar kernel defined in Borgwardt and Kriegel (2005). Results were given in Borgwardt and Kriegel (2005), for a bioinformatics problem where their kernels outperformed a walk-based kernel. The tree kernels originally presented in Ramon and Gärtner (2003) considered tree patterns from a graph that could be calculated up to infinite depths. This work was furthered by Mahé and Vert (2006) which examined tree patterns up to a finite depth and tree patterns where totters were removed. Using the MuTag dataset and a second larger, publicly-available dataset, results showed improvements compared to walk-based kernels.

The optimal assignment graph kernel, introduced in Fröhlich et al. (2005), does not use the convolution kernel approach, as is common with others in this Chapter. This kernel uses a bipartite graph matching algorithm to calculate the kernel. By defining two graphs as a *bipartite graph*, where the vertices of G_1 and G_2 are two disjoint sets with edges only between vertices in G_1 and G_2 . The edges between G_1 and G_2 with the highest similarity are given the largest weight for the edge label. This similarity is based on matching the atom labels for two vertices, as well as surrounding vertices. Using the Hungarian method, solvable in polynomial time, the most similar areas of G_1 are matched with those in G_2 . Although, intuitively, this seems like a good approach to match two molecules, it is not positive definite. We refer the reader to a proof given in Gärtner (2005b) that shows that the min and max function are not positive definite. This kernel uses a max function to return the optimal match between two graphs and so this is not positive definite.

A weighted decomposition kernel between two graphs was presented in Ceroni et al. (2007), which has many parallels to the optimal assignment kernel. This kernel compares each vertex in graph G_1 to vertices in G_2 , and scores how similar each atom is based on atom type, atom charge and bond type, as well as information from neighbouring vertices. Also in parallel to the optimal assignment kernel, there is a min operation when comparing vertices so this type of kernel may not be positive definite in general as well. Results on a publicly-available NCI cancer dataset suggest that this kernel performs better than frequent sub-graph and cyclic pattern kernels.

Other related research, that does not directly use a structured kernel approach has recently been presented. Research in Ralaivola et al. (2005) generated fingerprints for each molecule, and then used a Tanimoto normalization to create a *Tanimoto graph*

kernel. Results are given for the MuTag, PTC and NCI-HIV datasets which performed better than walk-based kernels on some tasks. Some other work (Azencott et al. (2007)) has also used fingerprints taken from 1D-4D space for a molecule with promising results.

3.5 Summary

In this Chapter we have reviewed a number of graph kernels. We first described notation for labeled, directed graphs. After, we described why a complete graph kernel which uses all sub-graphs as features is NP-complete and so other features must be chosen. We then reviewed a number of graph kernels that used different features including walks, trees, cycles and distances between labels. Finally, we reviewed several new graph kernels for molecules.

Chapter 4

Finite-Length Graph Kernels and Extensions

In this Chapter we introduce a new approach to calculating graph kernels. We show that graph kernels computed with dynamic programming are very efficient allowing walks of several hundred vertices in length to be counted. Furthermore, our approach allows more domain knowledge to be used. We show how soft-matching and gappy walks can be included in our framework. This allows prior knowledge to be incorporated so that one can tailor graph kernels to molecular data. We introduce two new finite-length graph kernels in Section 4.2. These kernels allow one to specify long walks which may define a high degree of similarity. Next, we show how this approach can be used to calculate two infinite-length kernels in Section, 4.3. Following this, we describe extensions that can be added to these four basic kernels. We show that non-tottering walks can be removed in Section 4.4. Walks that have miss-matched labels can be matched using soft-matching as described in Section 4.5. Finally, we show how walks can be matched where one or more gaps are inserted (Section 4.6).

4.1 Product Graphs

Recall, from Chapter 3 that a product graph can be constructed from two graphs in order to count the number of walks in common. To begin calculating this kernel, one first needs to calculate the product graph given two graphs.

4.2 Finite-Length Graph Kernels

In this Section we describe an alternative method to calculate walk-based kernels that allows more flexibility in feature-weighting so that different walk-weighting schemes can

be introduced. We introduce two new kernels that use finite-length walks as features. The first uses a single walk length (termed *FC graph kernel*) and the second uses all walk lengths up to a certain length (termed the *FS graph kernel*).

4.2.1 FC Graph Kernel

Definition 4.1 (FC Graph Kernel). Assume we have the direct product graph $G_{\times} = (\mathcal{V}_{\times}, \mathcal{E}_{\times})$ of two graphs G_1 and G_2 . The finite-length contiguous label sequence or FC graph kernel counts the number of matching p -length walks in two graphs using the following dynamic programming equations:

$$D_0(v_i) = 1 \quad (4.1)$$

$$D_n(v_i) = \sum_{v_j: (v_j, v_i) \in \mathcal{E}_{\times}} D_{n-1}(v_j) \quad (4.2)$$

for each vertex $v_i \in \mathcal{V}_{\times}$ and where the notation (v_j, v_i) describes a directed edge which starts at vertex v_j and ends at v_i . The FC graph kernel is then:

$$\kappa(G_1, G_2) = \sum_{v_i \in \mathcal{V}_{\times}} D_p(v_i) \quad (4.3)$$

From eq (4.3) it is clear that the kernel counts all walks of length p in the direct product graph G_{\times} . For each vertex, v_i , D_0 records the existence of each vertex in the graph (walk of length 0). At each iteration $D_n(v_i)$ sums up the counts of $p - 1$ walks that end in vertex v_i . The kernel is the sum of these walks for each vertex in the graph.

This method is much more attractive computationally than other graph kernels. The worst case time complexity of this method is $O(p|\mathcal{V}_{\times}|d)$, where d is the maximum degree of G_{\times} . Big-Oh notation ($O(\cdot)$) is used to describe an upper-bound on the worst-case time complexity of the algorithm (Skiena (1997)). We compare this to the graph kernel in Gärtner et al. (2003), which computes the kernel by inverting the adjacency matrix, E_{\times} . This gives a worst-case time complexity of $O(|\mathcal{V}_{\times}|^3)$.

4.2.2 FS Graph Kernel

A slight variation of the previous kernel is the *FS graph kernel*. While the FC graph kernel returns only the number of all walks of length p , the FS kernel returns the number of all walks up to length p (i.e. walks of length $1 + 2 + \dots + p$).

Definition 4.2 (FS Graph Kernel). Assume we have the direct product graph $G_{\times} = (\mathcal{V}_{\times}, \mathcal{E}_{\times})$ of two graphs G_1 and G_2 . The FS graph kernel, counts the number of all walks

up to length p , given by:

$$\kappa(G_1, G_2) = \sum_{\hat{p}=0}^p \sum_{v_i \in \mathcal{V}_\times} D_{\hat{p}}(v_i) \quad (4.4)$$

We note that the FS kernel is quick to compute. Although it counts walks for each length, $1, \dots, p$, it only needs to use the DP equations (eq (4.3)) for the last length p and sum the values at each successive application of the DP update. The worst case time complexity of the FS kernel is $O(2p|\mathcal{V}_\times|d)$ as one must apply eq (4.3) at each iteration of p . This is strictly the same as the FC kernel as the constant 2 is not counted with Big-Oh notation.

4.3 Infinite-Length Graph Kernels

In this Section, we describe how our framework can be used to approximate infinite walk-length kernels. We will show how two variants of infinite-length kernels fit into this framework: the kernel in Mahé et al. (2004) (we term *IM graph kernel*) and the kernel in Gärtner et al. (2003) (we term *IG graph kernel*). The original formulation suggests the kernel could be derived using an inverse of the product graph adjacency. Please refer to Chapter 3 for a more discussion of these kernels.

4.3.1 IM Graph Kernel

We first describe how the infinite walk-length kernel given in Mahé et al. (2004) can be computed with our DP approach. The kernel is calculated by first generating the product graph for G_1 and G_2 . A walk in the product graph can be defined by a starting probability, π_s , and transition probability matrix, Π_t . Also, recall from Chapter 3 that the IM kernel is parameterized by P_q . P_q allows one to put more emphasis on short paths when it is close to 0 and long paths when P_q is close to 1.

Definition 4.3 (IM Graph Kernel). Assume we have the direct product graph $G_\times = (\mathcal{V}_\times, \mathcal{E}_\times)$ of two graphs G_1 and G_2 . The IM graph kernel is defined by one parameter, P_q . Recall from Chapter 3 that walks in the product graph are defined by P_q using two probabilities. The starting probability (uniform for all vertices), $\pi_s = \frac{P_q^2}{|\mathcal{V}_1||\mathcal{V}_2|}$, where $\mathcal{V}_1, \mathcal{V}_2$ are from G_1, G_2 respectively. The walk transitions, from one product vertex v_i to vertex v_j , are recorded in a transition matrix, Π_t . The elements of this matrix are composed of $\Pi_t(v_j|v_i) = \frac{(1-P_q)^2}{\text{degree}(v_i(G_1))\text{degree}(v_i(G_2))}$, where $v_i(G_1)$ refers to the original vertex in G_1 which was used to create the product vertex v_i .

The DP equations to approximate the IM graph kernel are therefore:

$$D_0(v_i) = \pi_s \quad (4.5)$$

$$D_n(v_i) = \sum_{v_j: (v_j, v_i) \in \mathcal{E}_\times} \Pi_t(v_j|v_i) D_{n-1}(v_j) \quad (4.6)$$

$$(4.7)$$

The kernel is obtained by choosing the number of iterations, p , to estimate the kernel. It is given by:

$$\kappa(G_1, G_2) = \sum_{v_i \in \mathcal{V}_\times} D_p(v_i) \quad (4.8)$$

In order to determine how many iterations are necessary for a good estimation, for a range of stopping probabilities we constructed finite-length approximation testing iterations 1 to 8. We then measured the Frobenius norm of the difference between the finite-length approximation and the infinite-length walk kernel calculated using an inverse of the transition matrix suggested in the original kernel. One can see from Figure 4.1 that walks of only a few vertices in length are needed for a good approximation.

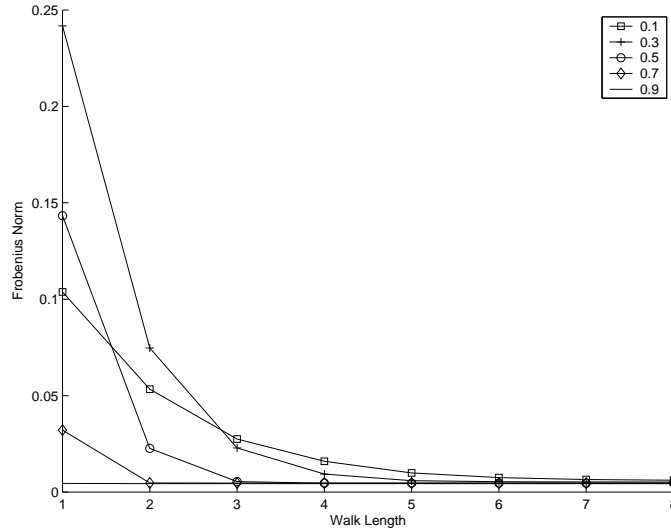


FIGURE 4.1: Graph showing $\|K - \hat{K}\|_F^2$ for an infinite length kernel, K , and the finite length approximation, \hat{K} . Increasing iterations (Walk Lengths) are tested, as well as several stopping probabilities, P_q , (0.1, 0.3, 0.5, 0.7, 0.9). Very few iterations were needed to find a good approximation.

4.3.2 IG Graph Kernel

A second infinite walk-length kernel in Gärtner et al. (2003) can be approximated with our DP approach. Recall from Chapter 3, that after constructing a product graph for

G_1 and G_2 , all walks of the product graph can be described as a geometric series. The convergence of this series is given by one parameter γ .

Definition 4.4 (IG Graph Kernel). Assume we have the direct product graph $G_\times = (\mathcal{V}_\times, \mathcal{E}_\times)$ of two graphs G_1 and G_2 . The IG graph kernel is defined by one parameter γ .

The DP equations to approximate the IG graph kernel are therefore:

$$D_0(v_i) = 1 \quad (4.9)$$

$$D_n(v_i) = \sum_{v_j: (v_j, v_i) \in \mathcal{E}_\times} \gamma D_{n-1}(v_j) \quad (4.10)$$

$$(4.11)$$

The kernel is obtained by choosing the number of iterations, p , to estimate the kernel. It is given by:

$$\kappa(G_1, G_2) = \sum_{v_i \in \mathcal{V}_\times} D_p(v_i) \quad (4.12)$$

Clearly, the IG kernel down-weights walk transitions evenly, while the down-weight for walks in IM is based on the density of vertex.

4.4 Non-Tottering

Mahé et al. (2004) introduced the idea of *non-tottering walks*. A tottering walk is one which simply turns around and traverses the previous edge in the opposite direction. A tottering walk is less informative as it will contain repeated vertices and therefore the tottering walks will explore less structure of a graph. A walk which only totters will move back-and-forth between two vertices, without exploring any other area of the graph. In Mahé et al. (2004), they showed that by modifying the original graphs, totters could be removed. In this Section, we will describe a new method of removing totters that is implemented using product graphs and dynamic programming. This approach will allow us combine it with the four kernels given in Sections 4.2 and 4.3. Next, we will describe how walks may totter in the original graphs but not in the product graph. We update the equations to ensure that we are not tottering in the original graphs as well.

4.4.1 Non-Tottering using DP

Given a product graph, $G_\times = (\mathcal{V}_\times, \mathcal{E}_\times)$, we can count the number of matching walks between two graphs using the DP equations given in Section 4.2. Non-tottering can be added to these DP equations with the following. For each round n in the computation

and for each vertex of the product graph, one simply keeps an array of how each previous vertex, j , contributed to the sum. For $n > 1$, let:

$$A_{nj}(v_i) = \begin{cases} D_{n-2}(v_j) & \text{if } (v_j, v_i) \in \mathcal{E}_\times \\ 0 & \text{otherwise} \end{cases} \quad (4.13)$$

then the DP update equation can be written as:

$$D_n(v_i) = \sum_{v_j: (v_j, v_i) \in \mathcal{E}_\times} \sum_{t: (v_t, v_j) \in \mathcal{E}_\times \wedge v_t \neq v_i} A_{nt}(v_j). \quad (4.14)$$

For the finite-length kernels, one uses the original DP equations for $n = 0$ (eq (4.1)) and $n = 1$ (eq (4.2)). Although, for $n > 1$ we use eq (4.14).

Similarly, for the infinite-length kernels, one uses original DP equations for $n = 0$ and $n = 1$. For the IM graph kernel, the new DP update for $n > 1$ is multiplied by $\Pi_t(v_j|v_i)$ (described in Section 4.3.1) giving:

$$D_n(v_i) = \sum_{v_j: (v_j, v_i) \in \mathcal{E}_\times} \sum_{t: (v_t, v_j) \in \mathcal{E}_\times \wedge v_t \neq v_i} \Pi_t(v_j|v_i) A_{nt}(v_j). \quad (4.15)$$

For the IG graph kernel, the new update for $n > 1$ is multiplied by γ (described in Section 4.3.2) giving:

$$D_n(v_i) = \sum_{v_j: (v_j, v_i) \in \mathcal{E}_\times} \sum_{t: (v_t, v_j) \in \mathcal{E}_\times \wedge v_t \neq v_i} \gamma A_{nt}(v_j). \quad (4.16)$$

This modification to the DP update, requires storing A which requires a maximum memory penalty of $O(2|\mathcal{V}_\times|d)$, where d is the maximum degree of \mathcal{V}_\times . Although $d = |\mathcal{V}_\times|$ for a fully connected graph, it is typically much smaller for a molecular graph. We require 2 times $|\mathcal{V}_\times|d$ as storage for the current and previous iteration in the DP algorithm.

In order to remove totters, the computational cost rises to $O(p|\mathcal{V}_\times|d^2)$ from the original formulation without totters which was $O(p|\mathcal{V}_\times|d)$. It now requires d^2 as for a given vertex, the algorithm must loop over each connected edge as the values in A which shows the contribution from two back. Similarly, one could remove walks that return after k iterations (a cycle of length k) although the complexity rises to $O(p|\mathcal{V}_\times|^k)$.

We will not include non-tottering features in the DP equations of Section 4.7 as we will not test this in further experimental chapters. Despite its reported success on a small dataset (Mahé et al. (2004)), we found that removing totters significantly reduced the accuracy of our models.

4.4.2 Non-Tottering in the Original Graphs

In this Section we discuss how a walk in the original graph may contain totters even though the product graph does not totter. In Figure 4.2 we show an example of two graphs, G_1 and G_2 , along with the full direct product graph, G_\times . We have uniquely subscripted each of the vertices in G_1 and G_2 from 1 to 5, although note that we are only matching based on the labels A or B . A walk in G_\times , ABA , has product vertices (A_2, A_3) , (B_1, B_4) , (A_2, A_5) . This corresponds to a tottering walk in G_1 , given by the first component (shaded), A_2, B_1, A_2 , and a non-tottering walk in the second graph G_2 , from the second component, A_3, B_4, A_5 .

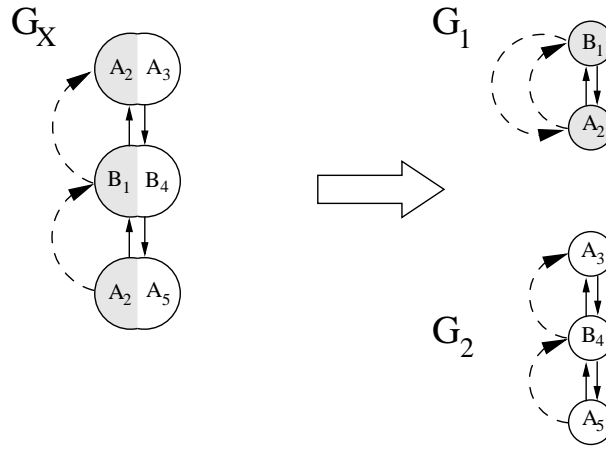


FIGURE 4.2: On the left is the product graph, G_\times for two graphs on the right: G_1 and G_2 . The dashed lines show a walk on each of the graphs. There is no tottering in the walk on G_\times , although tottering occurs in the original graph G_1 .

The dynamic programming equations that remove tottering in Section 4.4.1 can be updated to remove totters in the original graph as well. We note that each vertex v_p has a label of the form (a, b) where $a \in \mathcal{V}_1$ and $b \in \mathcal{V}_2$. Totters can be removed from the original graphs, by modifying the inner summation from eq (4.16). For a given vertex v_i , we check that the vertex two back, v_t is not the same vertex, given by the following constraint $t : (v_t = (a, b), v_i = (c, d)) \in \mathcal{E}_\times \wedge (a \neq c) \wedge (b \neq d)$. Removing totters may be useful in more complex graph matching, where we allow soft matching of labels and gaps, as it will remove features that may be undesirable.

4.5 Soft-Matching

In this Section we introduce new extension for graph kernels. When matching vertex labels between two graphs, we may want to consider *soft-matching* which allows inexact matches. A soft-match potentially allows a more appropriate weighting scheme,

rather than a binary comparison: match or no match. Even though two labels may not match exactly, expert knowledge may allow us to define a degree of similarity between mismatched labels that provides a better model of the data.

In our research, we are motivated to use soft-matching as molecules are complex structures with other features that can be matched besides the atom label (i.e. Carbon, Oxygen). Soft-matching allows much more flexibility in the graph kernel so that one can incorporate more prior knowledge when designing the kernel. As we are interested in finding walks which correspond with parts of the molecule that binds with the target, we will soft-match mismatched atom labels if the atom shares a common functional property associated with binding to the target. The functional property is given by the topological pharmacophore (TP) label that is used in the TP graph (please see Chapter 2). Each vertex in the molecular graph has a corresponding TP label which we will use for soft-matching.

Soft-matching was introduced with infinite-length graph kernels in Gärtner et al. (2003), where it was termed a non-contiguous label sequence kernel. Here, we prefer the term non-matching, as later on we will introduce a method for matching walks with gaps, and we wish to make clear the distinction between non-contiguous (with gaps) and non-matching (no gaps, but walk symbols may not be aligned). Assume we have the full direct product graph (i.e. ignoring labels) $G_o = (\mathcal{V}_o, \mathcal{E}_o)$. Recall from Chapter 3, the soft-matching infinite-length graph kernel defined in Gärtner et al. (2003) is:

$$\kappa(G_1, G_2) = \sum_{i,j=1}^{|\mathcal{V}_\times|} \left[\sum_{n=0}^{\infty} \lambda_n ((1-\alpha)E_\times + \alpha E_o)^n \right]_{ij}, \quad (4.17)$$

if the limit exists. There are two important points to note about this kernel that should be considered when defining the DP equations for soft-matching. First, the outer sum is over $i, j \in \mathcal{V}_\times$, so only walks that are *anchored* (or start and end with matching labels) are counted. Secondly, each block of non-matching symbols of length n has a penalty factor of α^n .

In Figure 4.3, we show an example of a product graph, G_\times , that allows soft-matching. If no soft-matching were allowed this example would not produce any walks with length greater than 0 (i.e. counts of matching product vertices). The vertices of the full, direct product graph will allow a match between every vertex so for this example $|\mathcal{V}_\times| = 16$. We only show the product vertices that have edges to emphasize where the soft-matching occurs. The 12 remaining product vertices will not be counted as we only start and end walks at matching vertices (anchored walks).

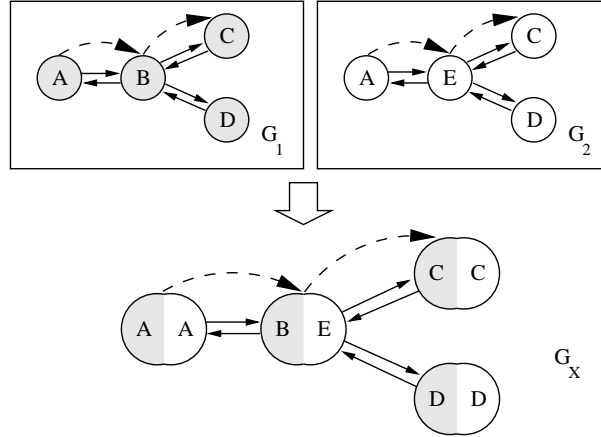


FIGURE 4.3: Two graphs, G_1 and G_2 are shown with their product graph G_\times . Note that walk ABC in G_1 is soft-matched with walk AEC in G_2 . This is shown with the corresponding walk in G_\times , $(A, A), (B, E), (C, C)$. The first component of a product vertex (shaded) is from G_1 , while the second component is from G_2 .

4.5.1 Soft-Matching using DP

In this Section we describe how soft-matching is added to the dynamic programming equations. Assume we have a product vertex, $v_i = (a, b)$. We first introduce a substitution matrix S where $S_{v_i} = [0, 1]$ indicates the strength of a match for a product vertex. If a product vertex has matching labels (i.e. $\text{label}(a) = \text{label}(b)$) then $S_{v_i} = 1$. In order to describe the DP equations, we need to introduce notation to differentiate between product vertices with matching labels and non-matching labels. Let \mathcal{V}_N be the set of vertices from \mathcal{V}_o that do not have matching labels, i.e. $\mathcal{V}_N = \mathcal{V}_o \setminus \mathcal{V}_\times$. For specific vertices, we will indicate this by superscript, that is v_i^\times , v_i^N will denote vertices from \mathcal{V}_\times , \mathcal{V}_N respectively. The dynamic programming equations are as follows:

$$\begin{aligned} D_0(v_i^\times) &= 1 \\ D_0(v_i^N) &= 0 \\ D_n(v_i) &= S_{v_i} \sum_{v_j: (v_j, v_i) \in \mathcal{E}_o} D_{n-1}(v_j) \end{aligned}$$

with the kernel computation the same as eq (4.3):

$$\kappa(G_1, G_2) = \sum_{v_i \in \mathcal{V}_\times} D_p(v_i) \quad (4.18)$$

Setting $S_{v_i} = 1$ for $v_i \in \mathcal{V}_\times$ and $S_{v_i} = \alpha$ for all $v_i \in \mathcal{V}_N$ gives the kernel defined in eq (4.17) from Gärtner et al. (2003). We note that our definition gives a much more general soft-matching kernel where one can choose specific weights for each label allowing one to incorporate more prior knowledge. With our approach to soft-matching we can specify different weights for certain mismatched labels, which allows one to incorporate expert knowledge. The walks are anchored to matching vertices by initializing the matching vertices to 1 and non-matching to 0 in the DP equations. The last vertex in the walk is anchored to a matching vertex as well, since the kernel in eq (4.18) only counts walks that finish in \mathcal{V}_\times . The time complexity of this kernel is $O(p|\mathcal{V}_o|d)$, where d is the maximum degree of G_o . Although we have only shown the soft-match DP equations for finite-length kernels, we will show the DP equations for infinite-length kernels in Section 4.7.

4.6 Gaps

In this Section we consider matching walks which are allowed to contain gaps which corresponds to insertion or deleting a vertex in a walk. This is analogous to the string kernel approach. See Lodhi et al. (2002) for details. It may seem surprising that this is at all possible without incurring a prohibitive cost, as many graph matching problems are NP-complete, although we will show that gap features can be included in our DP framework.

We are motivated to use graphs in our research as they may be useful for reduced graphs. As described in Chapter 2, reduced graphs are very small, often only a few vertices, and allowing gaps in these graphs will give flexibility to match features that may be useful in binding.

4.6.1 Singly-Gapped Walks

We first consider the case where we only allow gaps of length 1, or *singly-gapped* walks. Therefore the walk ABC would match AB-C or A-B-C, but not A--BC. For now, we will not consider non-matching vertices from G_o . We will introduce gaps by only describing matching product vertices, i.e. G_\times . We will later show how both non-matching vertices (or soft-matching) can be combined in Section 4.7. In order to add gaps we modify the original graphs, G_1 and G_2 , by adding wildcard vertices where an edge exists. Let W_1 be a set of wildcard symbols w_i such that $|W_1| = |\mathcal{E}_1|$ and hence the index i uniquely identifies an edge in G_1 . For each edge $e_i = (u, v) \in \mathcal{E}_1$ we create an additional node labeled with the appropriate wildcard symbol, w_i . We therefore add a node labeled w_i to the graph along with two additional edges $e_{i1} = (u, w_i)$ and $e_{i2} = (w_i, v)$ ¹. The

¹Note that if the edge e_i was labeled, then e_{i1} and e_{i2} are assigned with this label

graph $G_1 = (\mathcal{V}_1, \mathcal{E}_1)$ is transformed into $G_{s1} = (\mathcal{V}_{s1}, \mathcal{E}_{s1})$ where $|\mathcal{V}_{s1}| = |\mathcal{V}_1| + |\mathcal{E}_1|$ and $|\mathcal{E}_{s1}| = 3|\mathcal{E}_1|$. A similar transformation is applied to G_2 which we denote $G_{s2} = (\mathcal{V}_{s2}, \mathcal{E}_{s2})$. Please see Figure 4.4 for an illustration of this process.

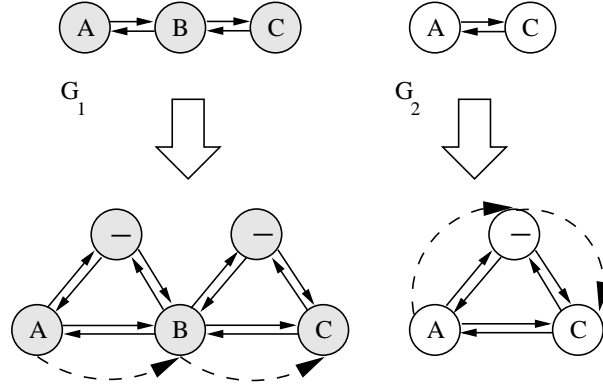


FIGURE 4.4: Two graphs, G_1 and G_2 are shown before (above) and after (below) the transformation. Note that walk ABC in G_1 matches with walk A-C in G_2 after the transformation. A '-' symbol represents a wildcard.

The direct product graph, $G_{s\times} = (\mathcal{V}_{s\times}, \mathcal{E}_{s\times})$, can now be constructed from the original graphs with added wildcard vertices, $G_{s1} = (\mathcal{V}_{s1}, \mathcal{E}_{s1})$ and $G_{s2} = (\mathcal{V}_{s2}, \mathcal{E}_{s2})$. We do not allow wildcard symbols to match. Therefore, there is no vertex in $\mathcal{V}_{s\times}$ composed of two wildcard symbols. More formally the vertex and edge sets are composed of:

$$\mathcal{V}_{s\times} = \{(a, b) \in \mathcal{V}_{s1} \times \mathcal{V}_{s2} : a \notin W_1 \vee b \notin W_2\} \quad (4.19)$$

$$\wedge \text{label}(a) = \text{label}(b) \quad (4.20)$$

$$\wedge \text{label}(a) \in W_1 \vee \text{label}(b) \in W_2\} \quad (4.21)$$

$$\mathcal{E}_{s\times} = \{((a, b), (c, d)) \in \mathcal{V}_{s\times} \times \mathcal{V}_{s\times} : ((a, c) \in \mathcal{E}_{s1} \wedge (b, d) \in \mathcal{E}_{s2})\} \quad (4.22)$$

We will introduce a parameter in Section 4.7 that down-weights symbols which match with a wildcard. This kernel will be more expensive computationally as we have increased the number of vertices in the direct product graph. Without wildcards, the maximum size of \mathcal{V}_{\times} is $|\mathcal{V}_1| \times |\mathcal{V}_2|$ when all labels match. Since wildcard labels cannot match each other, the maximum size of the product vertices, $\mathcal{V}_{s\times}$, is $(|\mathcal{V}_1| \times |\mathcal{V}_2|) + (|\mathcal{V}_1| \times |\mathcal{E}_2|) + (|\mathcal{V}_2| \times |\mathcal{E}_1|)$. Although the second and third terms always match, the product graph must obey the structure of the original graphs, so in practice there are not as many matches.

4.6.2 Allowing Multiple Gaps

In order to allow more than a single-spaced gap, one must add additional edges between wildcard symbols in each graph. See Figure 4.5 for an example of the resulting graph.

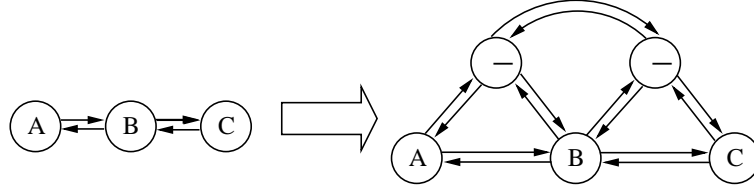


FIGURE 4.5: A graph is shown after the transformation. Note that an additional link is added between wildcard nodes (labeled '-').

As discussed in the last section, the maximum number of vertices in the product graph that includes wildcards is \mathcal{V}_\times , is $(|\mathcal{V}_1| \times |\mathcal{V}_2|) + (|\mathcal{V}_1| \times |\mathcal{E}_2|) + (|\mathcal{V}_2| \times |\mathcal{E}_1|)$. If we allow gaps we allow edge labels to match so this now becomes $(|\mathcal{V}_1| \times |\mathcal{V}_2|) + (|\mathcal{V}_1| \times |\mathcal{E}_2|) + (|\mathcal{V}_2| \times |\mathcal{E}_1|) + (|\mathcal{E}_1| \times |\mathcal{E}_2|)$. The second and third terms will increase as it does not need to obey the structure of the graph, unlike with single gaps. Furthermore, the fourth term adds further product vertices. We will not include multiple gaps in the DP equations of Section 4.7 because of the increased complexity of this feature.

4.7 Algorithms for Combined Kernels

In this Section we give the full dynamic programming equations for computing the kernels described in this Chapter. As the contribution from each of these kernels is broken up in the sum when using dynamic programming, one can choose which of these feature to include.

Before giving the DP equations, we will repeat some definitions and provide some additional terms. We define the single-gap, non-matching direct product graph to be $G_{sn} = (\mathcal{V}_{sn}, \mathcal{E}_{sn})$. We do not remove tottering features, although it is possible to add this to the combined DP equations as well. Let \mathcal{V}_N be the set of vertices from \mathcal{V}_o that do not have matching labels, i.e. $\mathcal{V}_N = \mathcal{V}_o \setminus \mathcal{V}_{sn}$. Also, let \mathcal{V}_W be the set of vertices which contain a wildcard symbol, $\mathcal{V}_W = \mathcal{V}_{s\times} \setminus \mathcal{V}_{sn}$. The remaining vertices are matching vertices, \mathcal{V}_\times . We will use the notation v_j^\times , v_j^N , v_j^W to denote a node j from \mathcal{V}_\times , \mathcal{V}_N and \mathcal{V}_W respectively. As in Section 4.5, we assume we are given a substitution matrix S which gives the cost of mismatching two labels. We also introduce two new parameters that allow one to choose the amount of influence from soft-match and gappy extensions. $\sigma_{sm} = [0, 1]$ down-weights soft-match features and $\sigma_{1g} = [0, 1]$ down-weights single gap features. We note that σ_{sm} could be incorporated into S , although we choose to create a separate parameter as in practice one will often define specific values to compare labels in S and tune the model using σ_{sm} .

4.7.1 FC and FS Graph Kernels

The dynamic programming equations for computing the FC and FS kernel are as follows:

$$D_0(v_i^\times) = 1 \quad (4.23)$$

$$D_0(v_i^N) = 0 \quad (4.24)$$

$$D_0(v_i^W) = 0 \quad (4.25)$$

$$D_n(v_i^\times) = \sum_{v_j: (v_j, v_i) \in \mathcal{E}_{sn}} D_{n-1}(v_j) \quad (4.26)$$

$$D_n(v_i^N) = \sigma_{sm} S_{v_i} \sum_{v_j: (v_j, v_i) \in \mathcal{E}_{sn}} D_{n-1}(v_j) \quad (4.27)$$

$$D_n(v_i^W) = \sigma_{1g} \sum_{v_j: (v_j^\times, v_i) \in \mathcal{E}_{sn}} D_{n-1}(v_j^\times) \quad (4.28)$$

The FC graph kernel is defined as:

$$\kappa(G_1, G_2) = \sum_{v_i \in \mathcal{V}_\times} D_p(v_i) \quad (4.29)$$

As described in Section 4.2, the FS kernel is almost identical to the FC kernel except at each update the sum D_n is summed. This modification allows the kernel to count all walks up to length p .

4.7.2 IM and IG Graph Kernels

As described in Section 4.3.1 and 4.3.2, the IM and IG graph kernels are calculated by multiplying the update equations by a down-weight. As the combined DP equations are very similar, we will describe them in terms of variables A and B and then later define these for each kernel. The dynamic programming equations for computing the IM and IG graph kernels are:

$$D_0(v_i^\times) = A \quad (4.30)$$

$$D_0(v_i^N) = 0 \quad (4.31)$$

$$D_0(v_i^W) = 0 \quad (4.32)$$

$$D_n(v_i^\times) = \sum_{v_j: (v_j, v_i) \in \mathcal{E}_{sn}} BD_{n-1}(v_j) \quad (4.33)$$

$$D_n(v_i^N) = \sigma_{sm} S_{v_i} \sum_{v_j: (v_j, v_i) \in \mathcal{E}_{sn}} BD_{n-1}(v_j) \quad (4.34)$$

$$D_n(v_i^W) = \sigma_{1g} \sum_{v_j: (v_j^\times, v_i) \in \mathcal{E}_{sn}} BD_{n-1}(v_j^\times) \quad (4.35)$$

The kernel is calculated with:

$$\kappa(G_1, G_2) = \sum_{v_i \in \mathcal{V}_\times} D_p(v_i) \quad (4.36)$$

For the IM kernel, $A = \pi_s$, the initial starting probability of a walk. $B = \Pi_t(v_j|v_i)$ from the transition matrix, Π_t . Both of these are parameterized by one variable, P_q (described in Section 4.3.1). We use the notation σ_{IM} (for this parameter P_q) in our experiments.

For the IG kernel, $A = 1$ and $B = \gamma$. As described in Section 4.3.2, $\gamma = [0, 1]$ and evenly down-weights walk transitions (unlike the IM kernel). We use the notation σ_{IG} to describe the parameter γ for this kernel.

Both the IM and IG kernels are parameterized by one variable, which we describe as σ_{IM} and σ_{IG} . Both of these parameters are defined in $[0, 1]$, where a value close to zero down-weights so that short walks are emphasized in the kernel. Conversely, a value close to 1 emphasizes longer walks.

4.8 Summary

In this Chapter we presented a new approach to calculate graph kernels using a framework where we can incorporate expert knowledge. We first introduced two new finite walk-length kernels, FC and FS graph kernels. Next, we describe how two infinite walk-length kernels, IM and IG, are approximated with the DP framework. Following this, we showed how several other features could be introduced in this DP framework, including non-tottering, soft-matching and gaps. Soft-matching allows information from alternative vertex labels to be incorporated such as topological pharmacophore labels.

Gaps will allow small walks that have slight structural differences to be matched which should prove useful with reduced graphs as they are very small.

Chapter 5

Experiments in Classification of Molecules

Graph kernels can be used to classify molecules into relevant classes such as ‘active’ and ‘not active’. There are many parameters that must be chosen when using graph kernels for classification: there are parameters for graph kernels which emphasize different features, parameters for the SVM model, and a choice of several graph-based representations. This Chapter introduces a number of new experiments that are tested in one consistent experimental design. We hope to understand patterns between these variables and which combinations produce the best model. Firstly, our finite-length graph kernels are tested and compared with two known infinite-length graph kernels. Secondly, we test our new soft-matching and gap extensions with finite and infinite length graph kernels. Also, we try two graph colouring algorithms (with several parameters) and compare these with experiments that do not include this colouring. Furthermore, we run experiments comparing molecular graphs (MG) and two previously untested graph-based representations including reduced graphs (RG) and topological pharmacophore graphs (TP).

This Chapter is organized as follows. Our experimental setup and design choices are described in Section 5.1. Next, we give metrics to measure experimental success in Section 5.2 and visualize the kernel space in Section 5.3. Experiments on a small dataset are analyzed in Section 5.4. Later in Section 5.5, we give experiments with a much large dataset that contains over forty thousand molecules. A further experiment that uses progressively larger numbers of negative examples for training is given in Section 5.6. Finally, we summarize in Section 5.7.

5.1 Experimental Design Decisions for Graph Kernel Models

There are many parameters and representations possible to experiment with graph-kernels for drug discovery. Crucially, we must choose a graph kernel, colouring and graph representation when running an experiment. In this Section, we will describe these choices and layout which kernels and parameters we will test in our Classification experiments of this Chapter.

Choosing a graph-kernel is one fundamental first step, although there is no obvious choice. As discussed in Chapter 3, there are several ways to calculate a graph kernel. A graph kernel can be calculated using several different features including walks, cycles, trees, and shortest distances between vertices. In this work, we will use our dynamic programming framework which counts finite walks. We will calculate two finite walk-length kernels including the FC kernel which uses a single walk-length for the kernel. We will also use the FS kernel which sums all walk-lengths up to a given length. Both kernels are tested with 1 to 15 iterations (walks). We focus on smaller walk-lengths as previous research has shown that these are the most informative (Mahé et al. (2004)). We also will test the infinite walk-length kernels which down-weight longer walks. These kernels are also computed using the same framework as the finite-length kernel, although longer paths are down-weighted. As discussed in Chapter 3, only a few iterations of the algorithm are needed, although we use 15 iterations in all experiments to achieve a good approximation. The two infinite-length kernels are the down-weight scheme from Mahé et al. (2004) (IM) and the down-weighting scheme from Gärtner et al. (2003) (IG). Both of these kernels are parameterized by one parameter which chooses the amount of down-weight added to longer paths. We term this parameter σ_{IM} and σ_{IG} for the IM and IG kernel respectively. We experiment with the following weights (0.01, 0.05, 0.1, 0.2, ..., 0.9) for all experiments. Finally, we also will experiment with the soft-matching (SM) and single gap (1G) kernel extensions. We will experiment adding these two extensions to all of the four basic graph kernels (FC, FS, IM, IG). The amount of soft-matching and single-gaps added to these kernels is parameterized by two weights σ_{sm} and σ_{1g} , which we will experiment with values using (0.01, 0.05, 0.1, 0.2, ..., 0.5).

In Table 5.1, we summarize the notation used to describe graph kernel and extensions in our experiments.

Notation	Description
FC	Finite walk-length single
FS	Finite walk-length all paths summed
IM	All walks down-weighted Mahé
IG	All walks down-weighted Gartner
SM	Soft-matching
1G	Single Gaps

TABLE 5.1: Notation for graph kernels

Another decision is the type of colouring used to modify the graphs. This is a pre-processing step performed to modify the original graphs and is computed off-line. For a molecular graph the original atom labels can be replaced with colour labels which contain more structural information. We will test both approaches introduced in graph kernel literature for experiments. The first method used in Gärtner (2005a) (notated as N-Colour) creates new vertex labels by combing each vertex label and each neighbouring vertex label. We will also use the morgan colour method introduced in Mahé et al. (2004) with 1,2 and 3 iterations (notated by M1-Colour, M2-Colour and M3-Colour). Finally, we will also experiment with the original graph (notated by No-Colour). A discussion of edge colouring is given in Chapter 2, although we do not test this modification.

A third decision necessary when deciding a graph-kernel model is the graph-based representation of the data. In our experiments we use three types of graph-based representations. Those are the classic molecular graphs (MG). Second, the reduced graphs (RG). And third, the topological pharmacophore graphs (TP). A discussion of these three types of representations is given in Chapter 2. In general, an RG is much smaller than an MG, although more structural information is given in MGs. A TP graph contains the same structure as an MG, although each atom is re-labeled with the closest topological pharmacophore.

In Table 5.2, we review the notation used to describe graph-based representations.

Notation	Description
MG	Molecular graph
RG	Reduced graph
TP	Topological pharmacophore graph

TABLE 5.2: Notation for three graph-based representations of molecules

5.2 Measuring Success

In order to interpret the success and compare with previous results we must define a measure to compare classifiers. With the first dataset we will record the accuracy to

compare results. Accuracy is suitable for small datasets, furthermore the first dataset is much more balanced with 68% from the active class and 32% inactive. We consider this relatively balanced as typically active examples comprise less than 1% of the dataset.

Unfortunately, this measure is not as useful with the second datasets as it is highly skewed with only 1% from the active class. This class distribution is fairly common in chemoinformatic datasets as there are often only a few examples that are active against a target; millions of other molecules do not share this property. As a result, overall classification accuracy would lead to misleading results. If a classifier was built that always guessed negative, it would in fact do very well, obtaining over 99% correct, unfortunately it would also guess that all of the drugs are inactive, which would defeat the purpose of this classifier - to discover new drugs. This problem of drug discovery is in some senses like finding a needle in a haystack. As a result, we need to find a measure that considers the importance of the active class and weighs these examples proportionally to the negative class.

This balance is achieved with Receiver Operator Characteristic (ROC) analysis. We define data that are ‘active’ as members of the positive class while all other molecules are from the negative class. If the example is positive, and if the classifier correctly predicts that it is positive, then it is a *true positive*. However, if it is classified as negative then it is a *false positive*. If an example is from the negative class (‘not active’), and the classifier predicts it as negative, then it is a *true negative*, if instead it were predicted as positive, then it is a *false negative*. The *true positive rate* is the number of true positives divided by the total positives. The false positive rate is the number of false positives divided by the total negatives. The ROC curve graphs the progression of the true positive rate versus the false positive rate (Fawcett (2003)). Since these two rates are scaled by the total positives and negatives, they are not sensitive to class imbalances. The Area Under the ROC curve (AUC), is a one number statistic that describes the performance of a classifier, where an ideal classifier would achieve an AUC of 1 and a random classifier 0.5. A value less than 0.5 is worse than random; if we reverse all the predictions we would have a classifier that has found a pattern. We will report the AUC for our classifiers, and we will use this to measure the performance.

5.3 Visualizing the Kernel Space

Data visualization techniques are an excellent method to gain an initial understanding of the input space where the data lies. Visualizing the data enables one to see any obvious patterns such as data clusters and outliers. As the kernel space is often high-dimensional and as graph kernels exist in a high or infinite-dimensional space, a technique such as dimensionality-reduction must be used to represent the data in two or three dimensions (2D,3D) where it can be visualized.

An overview of kernel-based data visualization techniques along with Matlab code for a simple visualization algorithm is given in Shawe-Taylor and Cristianini (2004). For a 2D view, the data is projected onto the first two principal components using a principal components analysis (PCA) algorithm. A visualization for our two datasets used in this Chapter is computed using this algorithm. The positive class, ‘actives’ are coloured red, while the ‘not actives’ are coloured blue to examine the differences between these classes. All 188 molecules from the smaller MuTag dataset are used. For our second dataset, only 1,000 inactives are used along with all 411 active molecules. The FC graph kernel is used for all visualizations. The one parameter for this kernel, walk-length, is fixed at length 5. Therefore, this kernel only uses walk-lengths of length 5 that match between each graph to calculate the kernel. Six plots are given in Figure 5.1: both datasets are visualized using each of the three representations (MG,RG,TP).

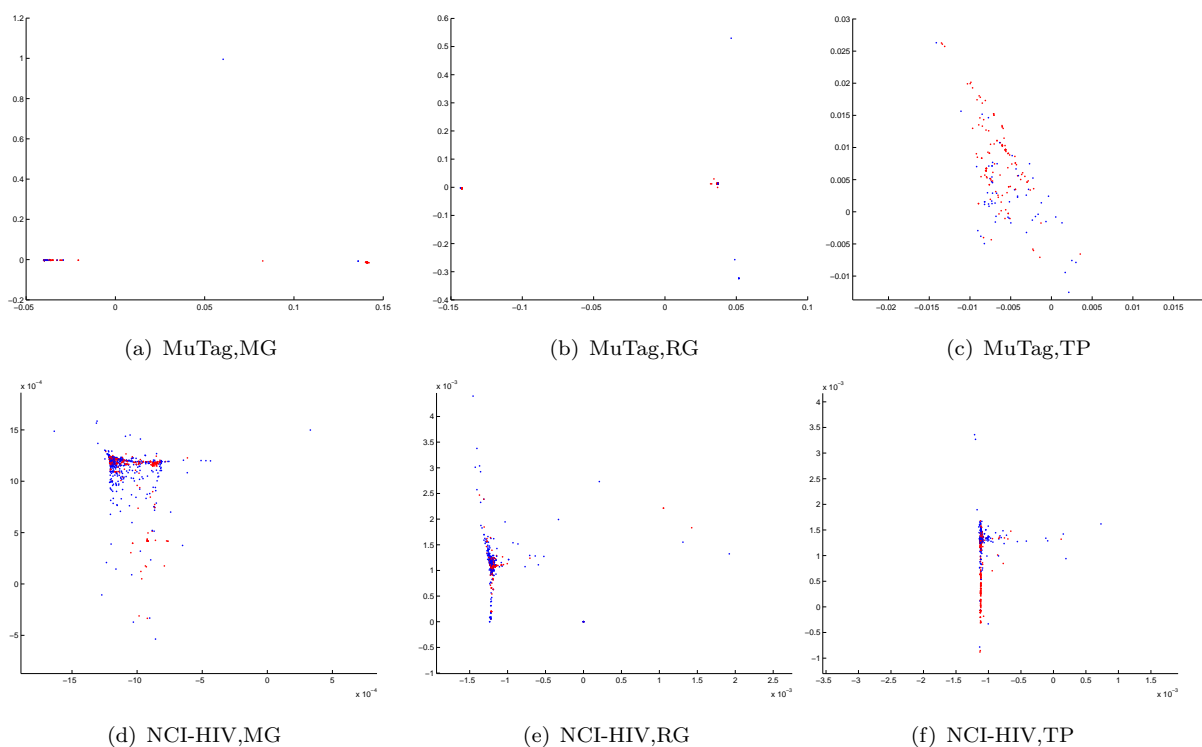


FIGURE 5.1: Visualizations of dataset 1 (MuTag) and dataset 2 (NCI-HIV) using MG, RG and TP representations. ‘Active’ examples are coloured red, while ‘not active’ examples are coloured blue. Plots 5.1(a) and 5.1(b) show actives in two clusters. In the remaining graphs (5.1(c), 5.1(d), 5.1(e) and 5.1(f)) actives appeared in only one cluster, so we have increased the focus on this area of the plot in order to show greater detail near the class boundaries. As a result, a few inactive outliers (around 10) are not shown in 5.1(c)-5.1(f)

In this 2D space, there appears to be clustering of actives and inactive molecules across

all 6 plots. Clearly, the data are not linearly separable based on only the first two dimensions, although patterns exist as both actives and inactives appear in separate clusters.

5.4 Dataset 1

The MuTag dataset (Debnath et al. (1991)) consists of 230 molecules tested for Mutagenicity in *Salmonella Typhimurium*. Only 188 of these are considered useful for classification of Mutagenicity (Debnath et al. (1991)). Of these, the data is relatively balanced with 125 active and 63 inactive forming this two-class problem.

For building SVM models, we used the publicly available¹ Gist-SVM in conjunction with our `SVMLight for graphs` to build the kernel matrix. Leave-one-out cross-validation is performed using Gist-SVM's built-in functionality.

In order to compare our kernels directly with Mahé et al. (2004) and Mahé (2006), we used Gist-SVM's built-in radial basis kernel function (RBF) heuristic, that automatically determines the RBF width. This heuristic estimates the RBF width using the average distance from each positive to each closest negative example. After Gist-SVM loads a kernel matrix, it applies this transformation before learning the SVM model. This has been suggested to be useful in practice, although no mathematical justification is given (Noble and Demco (2007)). We note that this heuristic is estimated using all examples from the kernel matrix. As the leave-one-out error is computed using the entire kernel matrix, some information from the test set is used while training and hence this may be considered cheating. On the other hand this RBF width is a single value which is averaged over all examples, so little test information is used for each LOO run. In light of this, we have decided to include this RBF parameter in order to directly compare with the highest published accuracies on the MuTag dataset (Mahé (2006)). We do not use this RBF parameter for experiments on the second, larger dataset or in further experiments.

As this dataset is very small, only containing 188 examples, there may be no measurable significance between classifiers. Moreover, as a grid-search is performed when searching for the best accuracy, one must question whether the results have been over-trained to this small dataset. Despite these issues, its small size makes it a good initial test to compare a number of computationally expensive graph kernel parameters. Furthermore, it has become the de-facto benchmark for most papers discussing graph kernels. In Section 5.5, analysis of a much larger dataset is given where significance between classifiers is compared.

¹<http://www.bioinformatics.ubc.ca/gist/>

5.4.1 Models with Implicit Hydrogen Atoms

After designing new graph kernels, we first wanted to check that our implementation matched the current state-of-the-art (Mahé et al. (2004)). Using the same Gist-SVM implementation, RBF parameter, IM graph kernel, no graph vertex colouring and using a LOO experiment, we were able to match almost all of their experimental parameters. We used our own implementation of the IM graph kernel, although we note that their implementation has been recently publicly released². This experiment allows us not only to compare our implementation of the IM kernel, but also to compare results for models that include Hydrogen atoms (Explicit H) and models that do not include Hydrogen atoms (Implicit H). We will only use the molecular graph (MG) representation for these experiments.

pq-Mahé	With H		No H	
	Accuracy (%)	AUC	Accuracy (%)	AUC
0.01	88.8	0.9468	84.6	0.8900
0.1	89.9	0.9435	86.7	0.8900
0.2	90.4	0.9425	86.2	0.8890
0.3	90.4	0.9392	85.1	0.8840
0.4	90.4	0.9357	83.0	0.8820
0.5	90.4	0.9360	82.4	0.8800
0.6	90.4	0.9337	83.0	0.8740
0.7	87.8	0.9280	82.4	0.8720
0.8	87.2	0.9142	82.4	0.8570
0.9	81.4	0.8930	77.7	0.8350

TABLE 5.3: With the MuTag data, we ran a set of parameters for the IM kernel, and compared representations that uses explicit hydrogen atoms to a model with implicit hydrogen atoms. Improved accuracy and AUC are achieved using models with explicit hydrogen or ‘With H’.

In Table 5.3, results are given comparing implicit and explicit Hydrogen models. The best accuracy is achieved with a model that includes Hydrogen with a value of 90.4%. This is achieved by setting the graph kernel IM parameter σ_{IM} to 0.2. The best implicit Hydrogen model only achieves an accuracy of 86.7% when the kernel parameter is set to 0.1.

Our best model achieves 90.4%, which is higher than accuracies for several other classifiers shown in Table 5.4, which were originally reported in King et al. (1996).

²<http://chemcpp.sourceforge.net>

Classifier	Accuracy
Linear Regression	89.3%
Neural Network	89.4%
Decision Trees	88.3%
Inductive Logic Programming	87.8%

TABLE 5.4: MuTag Dataset accuracy from several machine learning classifiers

Furthermore, much higher AUC scores were achieved when including Hydrogen (0.9468 vs 0.8900). These results match those given in Mahé et al. (2004) and Mahé (2006), suggesting that our implementation of the IM kernel is correct. In order to narrow the grid-search of parameters, and because of this result, all further experiments on the MuTag dataset are only built with models that include H in the representation. Furthermore, we are primarily interested in finding the positive class, so the AUC provides a much more sensible weighting. In the remaining tests on the MuTag we will only report the AUC score for comparing classifiers.

5.4.2 Colouring Choice for Graph Kernels

Using the same experimental procedure as the previous section, we next tried to find the best type of colouring for our four basic graph kernels. These kernels include the FC, FS, IG and IM kernels. Again, we only considered the MG representation. The finite-length kernels, including FC and FS, are parameterized by the walk-length parameter. The infinite-length kernels, IG and IM, are parameterized by the weight given on longer walks. We iterated to walk-length 15 in order to estimate the converged value. All four kernels were estimated over a range of parameters in order to find the optimal combination and to understand this parameter space. We try the five colouring types including M1, M2, M3 and N-Colouring as well as ones with No-Colouring. In these initial tests we do not include the SM or 1G extensions.

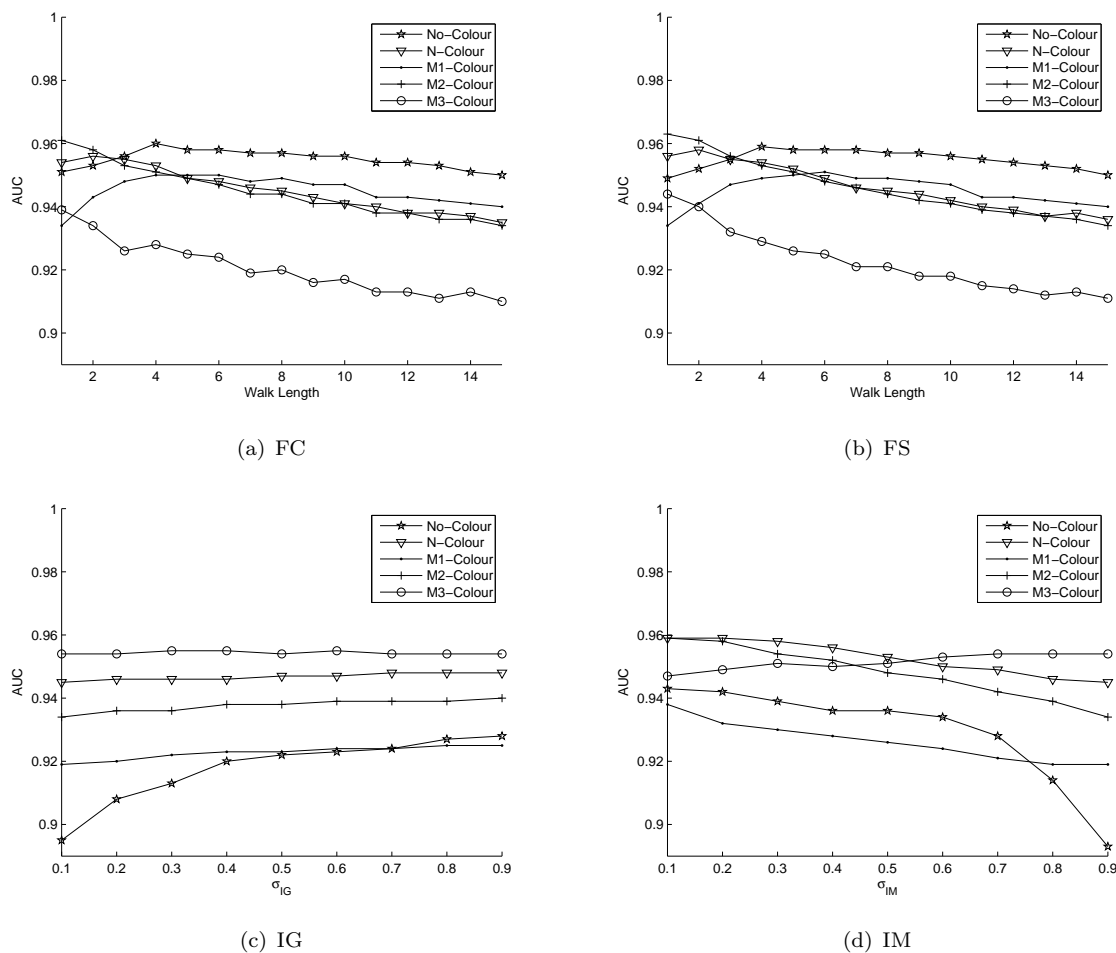


FIGURE 5.2: Results that compare the four basic graph kernels over a range of parameter values. Each line represents a set graph colouring along with a set of these parameters.

We first tested the finite walk-length kernels in combination with a number of graph colouring. In Figure 5.2(a), results are given for the FC kernel. Results for the FS kernel are given in Figure 5.2(b). Results for both of these kernels are very similar as the FS kernel simply sums up all walks walk lengths, while the FC kernel uses only a single walk length. Despite this they are similar especially since the largest walk has the greatest impact on the kernel. For short walks (length 1 to 3), the colouring methods N-Colour and M2 give the best results for both FS and FC kernels. This patterns suggests that incorporating more structural information using these colouring methods is beneficial for these shorter paths with the FC and FS kernel as these walks cannot reach this information on their own. With longer walks greater than 3, models with no colouring performs the best. This suggests that incorporating structural information in longer walks (by colouring with M1,M2,M3 or N-Colour) is not useful as these longer

walks can obtain this information directly. Also, the specificity of each vertex label increases, allowing less graphs to match for longer walks. Overall, the best AUC for the FC and FS kernel was found using a very short walk (length 1) with M2-Colour.

Next, we consider models that use infinite walk-lengths. In Figure 5.2(c) and 5.2(d), results are given for the IG and IM kernels along with different graph colourings. The best results were achieved using N-Colour and M3-Colour, which seems sensible as only the first few walks contribute to the kernel (longer walks are down-weighted). Including more information into each vertex allows these shorter length walks to use this information. In general, the IG results show that better results are found using longer walks (larger Exponential Weight parameters), while the IM kernel achieve better results with shorter walks (IM parameter). This inconsistency is a result of the weighting scheme, the IG kernel uses a single parameter to down-weight all values, while the IM kernel has different down-weights based on transition probabilities (this is described in Chapter 3).

Among all kernels, including finite and infinite walk-length, the highest AUC was found with the FS kernel by setting the walk-length to 1 and using M2-Colour. This achieves an AUC of 0.9630. The second highest AUC was found with the FS kernel, setting the walk-length to 1 and using M2-Colour. This achieves an AUC of 0.9610. Two models had the third highest AUC of 0.9600. Those were the IM kernel, setting σ_{IM} to 0.001, and M2-Colour, or by setting σ_{IM} to 0.001 and using N-Colour.

We note that among all of these models, the highest classification rate was found with the IM kernel by setting $\sigma_{IM} = 0.01$ and M2-Colour, resulting in a classification rate of 92%. This is higher than any previously published results on MuTag.

5.4.3 Choosing Soft-Matching and Single Gap Parameters

We next experimented with the soft-matching kernel extensions, SM and TPSM, and single gap (1G) parameters. We continued with the same experimental setup as before and only use molecular graphs.

We chose the FS kernel and added all 3 kernel extensions and compared to the basic kernel without these extensions. In Table 5.5, results are given for these experiments. We note that the first column refers to an alpha value of 0. By setting the SM, TPSM and 1G parameter to 0, the kernel does not use any information from these types of patterns. The kernels that use SM and TPSM are parameterized by σ_{sm} while the single gap extension is parameterized by σ_{1G} . Refer to Chapter 4 for more details.

As shown in the previous section, the best model with the FC kernel used No-Colour so we used this for testing the SM, TPSM and 1G extensions. We fixed the walk-length parameter for the FC kernel to 4 for all experiments, as this produced the best model in the previous section.

	0	0.01	0.05	0.1	0.3	0.5
FS+SM	0.9596	0.9597	0.9599	0.9605	0.9614	0.9615
FS+TPSM	0.9596	0.9590	0.9592	0.9595	0.9601	0.9599
FS+1G	0.9596	0.9591	0.9587	0.9580	0.9561	0.9549

TABLE 5.5: Results for SM, TPSM and 1G extensions added to FC kernel for MuTag dataset

Overall, the best SM model was found by using the largest value for σ_{sm} setting, $\sigma = 0.5$. This model achieved an AUC of 0.9615. The best TPSM model was found by setting the σ_{SM} to 0.3 to get an AUC of 0.9601. For both SM and TPSM, all parameter values achieved higher results than models which did not include this extension (the first column in Table 5.5).

Including the 1G extension, did not help the FS kernel. The FS kernel with no 1G extension achieved a higher AUC than for each of the parameter settings of 0.1 to 0.5. The larger the σ_{1G} parameter, and consequently the more influence on the kernel, the worse the performance. This extension may not be beneficial for molecular graphs, although it may be beneficial with reduced graphs, which are very small, and may benefit from greater flexibility when matching two graphs with different structure.

5.4.4 Comparing Graph Representations

Our final tests on the MuTag dataset, included experiments comparing the MG, RG and TP graph-based representations. This was the only change made to the experimental set-up and otherwise all other aspects were similar. For all experiments, we only record the highest AUC found from a grid search performed over all kernel parameter values and colouring types.

We summarize the optimal colour and kernel parameter choices in the following Table 5.6 for the three representations: MG, RG and TP. We also include the choices for SM, TPSM and 1G extensions added only to the FC kernel. The TPSM is only possible with the MG representation.

Kernel	MG		RG		TP	
	Colour	Parameter	Colour	Parameter	Colour	Parameter
FC	M2	1	No	13	N	1
FS	M2	1	M1	1	N	1
IG	M3	0.3	M1	0.5	M3	0.3
IM	N	0.001	M1	0.8	M3	0.9
FC+SM	M2	0.5	M1	0.9	N	0.9
FC+TPSM	M2	0.9	-	-	-	-
FC+1G	M2	0.01	M1	0.9	N	0.9

TABLE 5.6: Parameter choices for MG, RG and TP and graph kernels using MuTag

The patterns between colour and kernel parameter choices vary greatly. Of the models which select colour, the finite kernels (FC and FS) prefer very short paths choosing length 1. This seems reasonable as the structural information is now brought into each vertex label so longer walks are not required to find the desired pattern. Similarly, the one experiment for FS that chose No-Colouring found a very long walk-length to be the most informative: possibly because this structural information was not in the vertex labels. A less clear trend was evident with the IG and IM kernels. All experiments selected colouring to be used with the models, although the kernel parameter varied between the smallest possible value (0.001) and the largest (0.9), favoring short and long paths respectively.

Using these optimal parameters, the corresponding AUC scores are calculated for all three representations in Table 5.7.

	MG	RG	TP
FC	0.9629	0.9502	0.9415
FS	0.9601	0.9471	0.9319
IM	0.9604	0.9503	0.9500
IG	0.9548	0.9501	0.9501
FC,WL4	0.9596	0.9430	0.9222
FC,WL4+SM	0.9615	0.9514	0.9268
FC,WL4+TPSM	0.9606	-	-
FC,WL4+1G	0.9591	0.9465	0.9275

TABLE 5.7: Results for MG, RG and TP and graph kernels using MuTag

The MG representation performed well for all four basic kernels. Adding SM and TPSM for the molecular graphs slightly improved performance from 0.9601 to 0.9615 and 0.9606 respectively. The 1G extension resulted in a slightly lower AUC score. It is surprising that adding these extensions had such a small effect on the AUC score as a high weight was applied to SM models at 0.5 and TPSM at 0.9. The weight given to

the 1G extension was very small at 0.01, although even this slight addition to the kernel resulted in a smaller AUC.

With the RG representation, the highest SM extension AUC score outperformed the models with no SM. The 1G extension slightly decreased the top AUC score from 0.9471 to 0.9465. Both the SM and 1G extension weights were very high at 0.9 each, although this only resulted in very small change in the overall AUC score.

The TP representation achieved some of the lowest scores. Both the SM and 1G extensions reduced performance. These extensions were both given high weighting in the kernel at 0.9, although with the SM extension this resulted a decrease from 0.9319 to 0.9268. With the 1G extension a slightly smaller drop followed from 0.9319 to 0.9276

Overall, these results suggest that molecular graphs (MG) can be used to obtain better models for classification compared to RG and TP graphs. For all kernels, and extensions we tested the molecular graphs achieved the highest AUC for each experiment. The RG did worse than MG, although they achieved better results than TP for each kernel and extension. The best AUC scores were achieved without including soft-matching or gappy extensions.

5.5 Dataset 2

We used the NCI-HIV data for our second dataset. It is a publicly available dataset³, containing over 40 thousand molecules along with their ability to protect human cells from HIV-1 infection. Each compound is assigned one of three categories: highly active (CA), moderately active (CM) or inactive (CI) comprising 1.0%, 2.5% and 96.4% of the data respectively. We will focus on the binary classification experiment CA vs CI, as this is of the highest interest in real world drug discovery. This data was first used with graph kernels in Horvath et al. (2004), and more recently in Gärtner (2005b).

To model large datasets that consists of tens of thousands of examples, the cost of memory for a kernel matrix of this size becomes prohibitively expensive. In Table 5.8, we calculate the approximate memory requirements for a kernel matrix with different numbers of examples. With 40,000 examples, we would require roughly 11.9 Gigabytes of memory. Although this is possible with some high performance computers, the time required to build this entire kernel matrix must be considered. The computation of a graph kernel is quite complex, in our implementation it takes approximately 0.001 seconds on a 3Ghz Processor. For a kernel matrix of size 40 thousand, this would take roughly 222 hours. As a result, we decided to add our kernels to the popular `SVMLight` software, and call it `SVMLight for graphs`. `SVMLight` is one of the only SVM implementations that does not require the entire kernel matrix to be built. It takes

³<http://cactus.nci.nih.gov/ncidb/download.html>

advantage of the sparsity in kernel matrices and takes samples of the data during the calculation of the SVM, instead of computing all kernel values before hand.

Number of Examples	Space Required
100	0.07 Megs
1,000	8 Megs
5,000	186 Megs
10,000	745 Megs
20,000	2.98 Gigs
40,000	11.9 Gigs

TABLE 5.8: Storage required when varying size of kernel matrix

In our `SVMLight for graphs` software we added a requirement which set an upper limit for the largest size of molecule and product graph. The max number of vertices per example is set to 150, and the max number of product graph vertices per example was set at 20,000. Any example that was larger than this was excluded during training. Only around 30 examples exceeded this requirement and were not included in any experiments. This requirement affected our first dataset as well, although all 188 examples were within this threshold. We note that experts in the field suggest that very large molecules will most likely fail at later stages of the drug discovery process. In general a ‘drug’-like compound has less than 50 atoms. Salt compounds were removed from each molecule resulting in many duplicates. Each of these duplicates were filtered resulting in a smaller dataset. This step reduced the size of the dataset from the original 42,689 to 40,543 examples. After this we also removed a few examples who’s molecular graph could not produce a reduced graph. Only 31 molecules had this issue. After removal, this decreased the size from 40,543 to 40,512 molecules. This final desalted dataset, composed examples that were available in all three graph-based representations we use for this work: MG, RG and TP.

5.5.1 Experimental Setup

As described in Section 5.1, there are many design decisions involved in these experiments. In general, we will follow the experimental process used in Gärtner (2005b). The data is split into 10% for parameter tuning and 90% for testing. Both of these parameter tuning and testing splits are then further divided into 5 folds. The average AUC from a 5 fold cross-validation (CV) is used.

As the parameter tuning data only uses 10% of the data or around 4 thousand molecules, we built the kernel matrix for these experiments. After building the kernel matrix, our

`SVMLight for graphs` software chooses the one parameter for the SVM model, the SVM-C value. Experiments are run using a grid search and the highest AUC is selected by setting the SVM-C parameter to $10^3, 10^2, 10^1, 10^{-1}, 10^{-2}$ and 10^{-3} .

Using the parameter tuning data, we first chose the colouring type for each of the four basic kernels. We fix the kernel values for all four kernels and then test with each of the five colouring choices (N, M1, M2, M3 and No-Colour). We fix the FC and FS values to walk-length 5, and the IM and IG kernels to 0.1. The type of colouring is then selected for each kernel using the highest AUC score from these tests.

After the colour has been selected we only use this for each kernel for further experiments. Using this colouring model, we then choose the kernel value for each of the four kernels. We choose the walk-length for the FC and FS kernel by testing walk-lengths 1 to 15 and then selecting the highest AUC. Similarly, the IM and IG kernels are selected from a range of values including 0.01, 0.1, 0.2, \dots , 0.9. Additionally, for each of those selected, we record the SVM-C value for use when testing with the full 90% of the data.

We decided to only test the extensions, SM, TPSM and 1G with the finite kernels, as these extensions seem most relevant to longer walk lengths. Using infinite walk-length kernels, walks greater than two are so heavily down-weighted that they have only a small influence in the kernel. After the kernel parameters are tested, we choose among the FC and FS kernels based on highest AUC. After selecting FC or FS, we choose the best parameter for each of these extensions. The SM and TPSM extension are parameterized by σ_{sm} and 1G by σ_{1g} . We select the highest AUC from a range of parameters including 0.01, 0.1, 0.2, \dots , 0.5. We record the SVM-C value for testing with the full 90% of data.

After choosing the colouring and kernel parameters we then run experiments with these on the remaining 90% of the data (around 7 thousand examples per fold and around 29 thousand examples using 4-folds for each of the 5-fold CV run). Using a 5-fold CV we record the average AUC for these experiments.

5.5.2 Parameter Tuning Dataset

Fixing the kernel parameters, we tried five colouring types including N, M1, M2, M3 and No-Colour. The colour chosen for each kernel and representation is given in the Table 5.9. Most of the kernels and representations chose models with colouring, although the finite-length graph kernels, FC and FS did not. As the reduced graphs are very small and encode the local structure in the vertex label, encoding too much of the structural information into the labels may not work well. Therefore, the original labels may be best for classification.

	MG	RG	TP
FC	M1	No	M2
FS	M1	No	M1
IM	N	M2	M3
IG	N	N	M3

TABLE 5.9: Graph colouring types from the NCI-HIV parameter tuning set.

We first optimized the two finite-length kernels, FC and FS. In the following table, we list the average AUC for a 5-fold CV on 10% of the data. Results are listed for a range of walk-lengths and using the FC and FS kernel with each of the MG, RG and TP representations in Figure 5.3. It is interesting that most kernels choose fairly long paths, four of six choose length five or greater. The FC and FS produce different results for walk-length 1, as the FS kernel includes walk-length 0 which simply counts the product vertices.

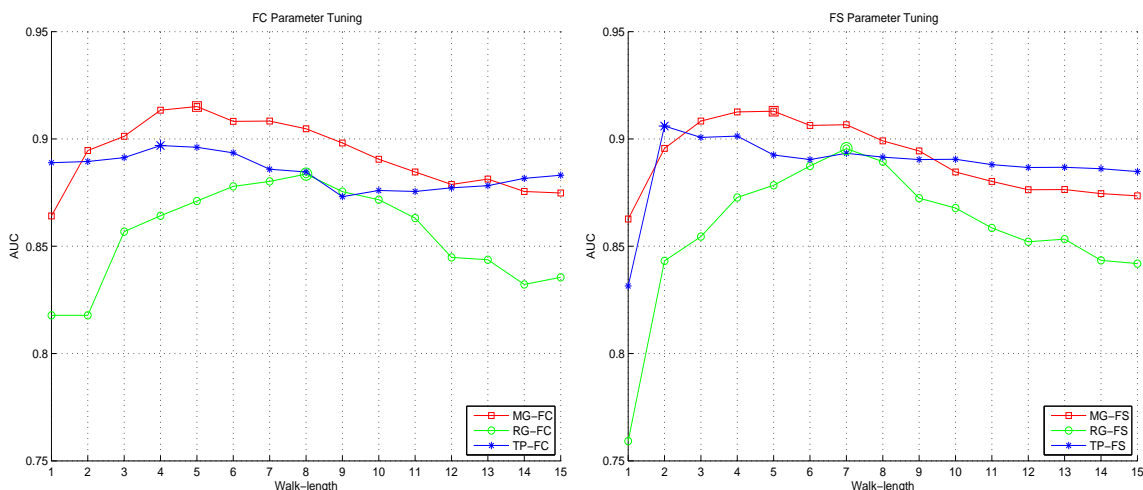


FIGURE 5.3: Parameter tuning experiments for the FC kernel are given in 5.3(a) and the FS kernel in 5.3(b). The best parameter will have an AUC score close to 1. We have set the Y-scale between 0.75 and 0.95, although an AUC score may have a value between 0 and 1.

In Figure 5.4, we list results for experiments testing a range of kernel parameters for the infinite-length kernels, IM and IG. Average AUC values from a 5-fold CV on 10% of the data are listed. A range of kernel parameters are tested with the IM and IG kernel for each of the MG, RG and TP representation. There is quite a large variety among infinite-walk kernel parameter choices. The IM kernel with MG chose 0.05, adding more emphasis to very short paths, while the IG with MG chose the largest weight adding more emphasis to longer paths. Although the IM and IG both count infinite-walks, they are calculated differently and this may be the cause of this variation.

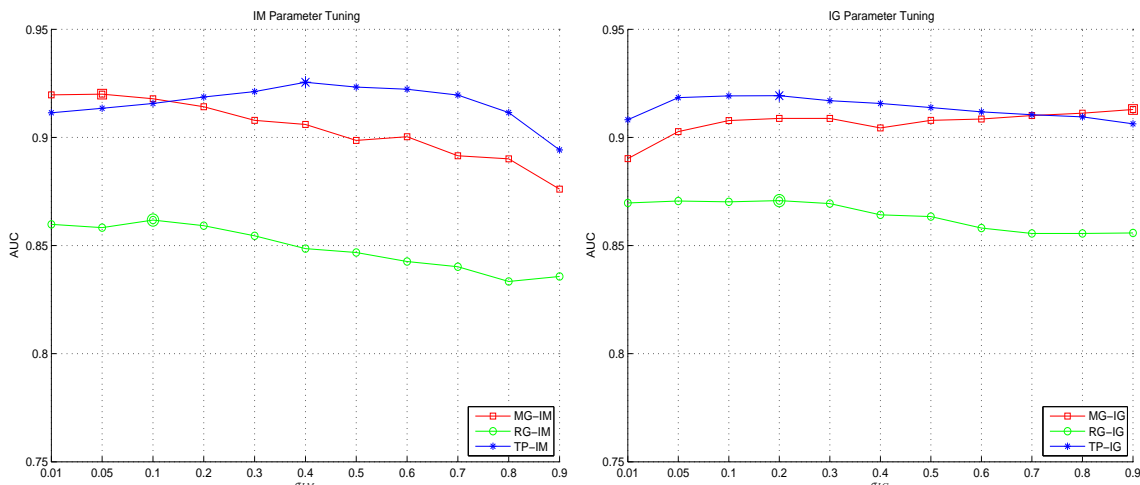


FIGURE 5.4: Parameter tuning experiments for the IM kernel are given in 5.4(a) and the IG kernel in 5.4(b). The best parameter will have an AUC score close to 1. We have set the Y-scale between 0.75 and 0.95, although an AUC score may have a value between 0 and 1.

Finally, results are given for parameter tuning the SM, TPSM and 1G extensions in Figure 5.5. The average AUC from a 5-fold CV on 10% of the data is given. We follow the same procedure as before and test a range of weighting values for the extensions along with the FC or FS kernel and each of MG, RG and TP representations. For each representation, we chose the best among the finite-length kernels, FC and FS and chose the best walk-length as a base to add these extensions. Again, we are using the colouring for each kernel and representation chosen in the initial stage. For molecular graphs, we used the FC kernel with walk-length 5. For reduced graphs we used the FS kernel and fixed the walk-length at 7. For TP graphs we used the FC kernel with walk-length 4. It is apparent that smaller weights between 0.01 and 0.1 produce the best models with highest AUC. After a certain threshold (around 0.1), the extensions appear to add too much noise to the kernel, and the performance of kernels with higher weights progressively decreases.

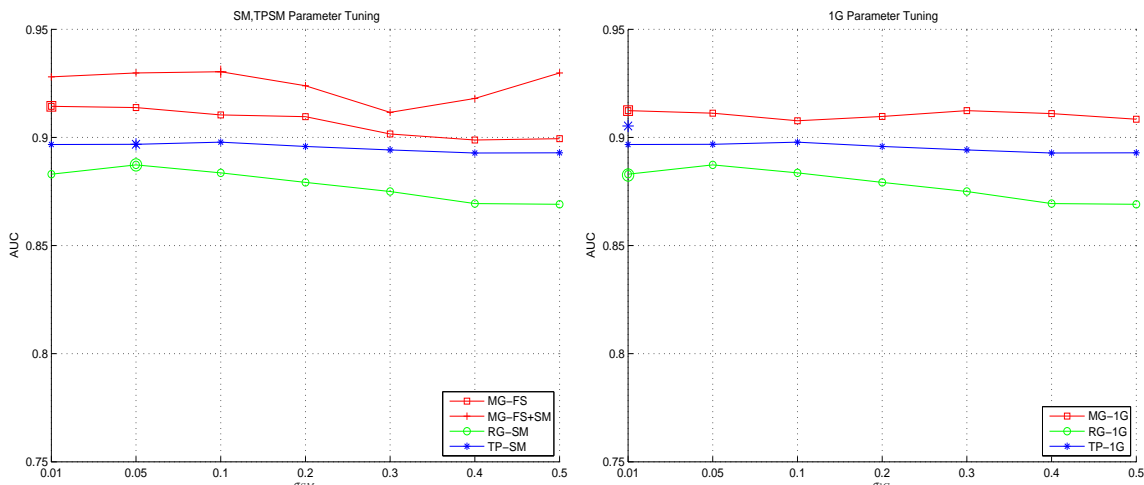


FIGURE 5.5: Parameter tuning for SM and TPSM is given in 5.5(a) and 1G is given in 5.5(b). The best parameter will have an AUC score close to 1. We have set the Y-scale between 0.75 and 0.95, although an AUC score may have a value between 0 and 1.

5.5.3 Full Results

After choosing the optimal parameter choices in the previous section, including colouring and kernel parameters for each kernel and representation, we next ran 5-fold CV experiments for each kernel and representation on the remaining 90% of the data.

For these experiments, each of the 5-fold CV was trained using our `SVMLight for graphs` software. Each of the five cross-validation (CV) experiments required four training folds which included around 29 thousand examples. Our software does not require the entire kernel to be pre-computed so these experiments could be run on a typical cluster node which had 1 Gigabyte of memory.

As we have set up these experiments to allow a 1:1 correspondence between training sets, for each representation, we can not only compare the performance between each kernel, but how well kernels perform across each representation: MG, RG and TP.

In Table 5.10, we list results for the MG, RG and TP representations. The average AUC and standard deviation is given for each kernel, along with results for each of the five folds. For the MG representation, the highest average AUC was obtained using the IM kernel, followed by IG, FC and FS. We added SM, TPSM and 1G extensions to the FC kernel. By adding the SM and 1G extensions, the FC kernel increased from 0.9192 to 0.9315 and 0.9327 respectively. Adding the TPSM to the FC kernel decreased from 0.9192 to 0.8690. The highest AUC for the RG representation was obtained using the IM kernel followed by the FS, IG and FC kernel. We added the SM and 1G extensions to the FC kernel. Both the SM and 1G extensions increased performance slightly from 0.9101 to 0.9112 and 0.9113 respectively. The highest AUC for the TP representation

was achieved with the IM kernel followed by the IG, FC and FS kernel. We added the SM and 1G extensions to the FC kernel, although lower results were achieved with these models. With no extensions the FC kernel achieved 0.9138 while with SM the performance reduced slightly to 0.9107 and with 1G it reduced to 0.9025. It is interesting to note that among all three representations, the IG kernel obtained the highest average AUC score among all 4 graph kernels.

Given a set of AUC scores from a 5-fold CV model, $\mathbf{A} = \{A_1, A_2, A_3, A_4, A_5\}$, we can use a t distribution to determine if two models are significantly different. We begin by calculating the sample mean:

$$\hat{\mathbf{A}} = \frac{\sum_{i=1}^5 A_i}{5} \quad (5.1)$$

We calculate the sample variance, $\text{var}(\mathbf{A})$, using:

$$\text{var}(\mathbf{A}) = \frac{1}{5} \sum_{i=1}^5 (A_i - \hat{\mathbf{A}})^2 \quad (5.2)$$

Given AUC scores from two models, \mathbf{A}_1 and \mathbf{A}_2 , we can calculate the test statistic, \hat{t} , with the following:

$$\hat{t} = \frac{\hat{\mathbf{A}}_1 - \hat{\mathbf{A}}_2}{\sqrt{\frac{\text{var}(\mathbf{A}_1) + \text{var}(\mathbf{A}_2)}{2}} \cdot \sqrt{\frac{2}{5}}} \quad (5.3)$$

Since the degrees of freedom is $2 \cdot 5 - 2$, we compare \hat{t} with $t_8(\alpha)$ and determine whether they are significantly different at a 5% level. According to Dekking et al. (2005), $t_8(\alpha)$ at a 5% level is 0.1860. If $\hat{t} > 1.860$ then we reject the hypothesis that they are the same, so they are therefore significantly different.

	MG	RG	TP
FC	0.9335(± 0.0055)	0.9101(± 0.0313)	0.8998(± 0.0077) \circ
FS	0.9192(± 0.0044)	0.9057(± 0.0255)	0.9138(± 0.0101)
IM	0.9355(± 0.0176)	0.9200(± 0.0114)	0.9266(± 0.0250)
IG	0.9335(± 0.0158)	0.9059(± 0.0085)	0.9245(± 0.0219)
F+SM	0.9315(± 0.0094)	0.9112(± 0.0332) \circ	0.9107(± 0.0113) \circ
F+TPSM	0.8690(± 0.0148)	-	-
F+1G	0.9327(± 0.0075)	0.9113(± 0.0301)	0.9025(± 0.0497)

TABLE 5.10: Results for NCI-HIV. We give the average AUC from 5-fold cross-validation along with the standard deviation. We use the notation \circ to identify if a significant loss (at 5%) over the same kernel and MG representation.

Overall, the molecular graphs performed better than the reduced or TP graphs for all four kernels. There was some variation for the second-best representation among the four kernels. RG had better results for the FC kernel, although TP had better results for the FC, IM and IG kernels. Although in general, each kernel for every representation

performed at a very similar level as shown by the significance levels. For the basic kernel, only the MG+TP combination resulted in a significant loss, otherwise a given kernel that uses either MG, RG or TP representations will give similar results.

We note that with reduced graphs, it was relatively fast to run all models, including those which added soft-matching and gaps. The most time-intensive were the MG and TP models with soft-matching and gaps. These took a long a very long time to build. The MG and 1G models took over 30 days for each of the 5 folds that had to be trained. This is most likely not a good method in practice. This result motivates the following section where we determine a faster approach to learning from large datasets.

5.6 Experiment: Scaling Number of Inactive Molecules

Experiments on the large NCI-HIV dataset, the second dataset in this Chapter, helped emphasize the time-intensive nature of computing graph kernels. Although one could use all available data, we question whether it is possible to do as well with less information. Specifically, we wondered whether all the ‘not active’ examples are required for training.

We argue that by removing some of the ‘not active’ examples we still retain the same patterns. In effect, what we are truly interested in is the positive examples. The goal of drug discovery is to find the active examples. We can reduce the number of negative examples to the point where we are left with a one-class classification problem (also termed outlier detection or distribution support estimation depending on the domain). One-class algorithms, for example the 1-SVM (Schölkopf et al. (1999)), find a region which contain all the points of this class corresponding to the class of interest. Any data point which lies outside this region is identified as an outlier, or ‘not active’. Inherently, one-class algorithms should do worse than binary classification, as you are giving the model less information. Therefore, we will continue to use binary classification as this class information is available.

By removing some of the negative examples, we can remove a major bottleneck in the training phase. In general, an increase in the number of training examples results in much greater training times. In this Section, we address the question of how many inactive examples are needed to achieve good performance on the data with AUC. We will keep all active examples and use only some of the negative examples.

To model this problem, we will use the NCI-HIV data, the second dataset that contains over 40,000 examples. We will re-run the four basic kernel experiments using the MG representation, and using the four basic kernels that don’t include any of the extensions, these are FC, FS, IM and IG. Also, we will use the colouring methods chosen in the previous experiments on dataset 2. The FC and FS kernels are coloured with N-Colour and IM and IG with M1-Colour. We will use the same parameter tuning from for these

kernels and just re-run the 90% examples, training and testing. Only the training sets will change for the 5 folds. The 5 test sets will remain the same with all negative examples used.

Since we use the same parameter tuning for the kernels (10% of the NCI-HIV data), the remaining 90% portion is used for testing. We create 4 new datasets (5-folds each) which include 1, 10 100 and 500 inactive examples in the following manner. We first selected all of the positive examples from the original dataset. We then created progressively larger folds by randomly sampling inactive examples from all ‘not active’ example for use. Each subsequently larger fold includes the inactive found in the previous iteration.

Results are given in Table 5.11 with varying amounts of inactive molecules. For comparison, we note that the number of inactive examples in the full 90% (notated by ‘Full’) has the following number of examples in folds 1 to 5 contain 25,892, 25,887, 25,864, 25,870 and 25,873 respectively. Each of the smaller datasets for example ‘10’ will have 10 inactive examples paired with all active examples for all 5 folds. We calculate the significance at 5% using the t distribution as discussed in the previous section.

	FC	FS	IM	IG
1	0.6391(± 0.0254) \circ	0.6714(± 0.0246) \circ	0.5702(± 0.0746) \circ	0.5729(± 0.0721) \circ
10	0.7220(± 0.0521) \circ	0.7528(± 0.0333) \circ	0.7056(± 0.0519) \circ	0.7355(± 0.0524) \circ
100	0.8688(± 0.0126) \circ	0.8985(± 0.0103) \circ	0.8583(± 0.0192) \circ	0.8683(± 0.0169) \circ
500	0.8483(± 0.0318) \circ	0.9291(± 0.0044) \circ	0.9077(± 0.0099) \circ	0.9033(± 0.0134) \circ
Full	0.9335(± 0.0055)	0.9192(± 0.0044)	0.9355(± 0.0176)	0.9335(± 0.0158)

TABLE 5.11: Results are given for subsets of randomly selected inactives. We report the average AUC from a 5-fold cross-validation as well as the standard deviation. We use the notation \circ to identify if a significant loss (at 5%) over the same kernel that uses all inactive examples (Full).

As shown in Table 5.12, both the training and testing phases of using this model were greatly sped up by sampling less numbers of examples. Another important point is that when using less examples for training, less possible support vectors can be given by the model. In terms of efficiency for the testing phase, this is very efficient as when classifying an example, this classification is a linear combination of the support vectors, as described in Chapter 1. Furthermore, all models that used less examples (in rows 1, 10, 100 and 500 of Table 5.12), had significantly lower AUC scores.

Significance was measured using the technique described above. It was expected that a lower AUC score would be achieved as the model is given less information, although it is surprising that when using a model with only one positive example, results can be achieved better than random classifiers. We note that although these scores were lower, they may be good enough when speed is more important than obtaining the optimal model.

	Avg. Train Time	Avg. Classify Time
1	5s	33s
10	18s	1m34s
100	1m25s	4m44s
500	6m7s	10m21s
Full	12h	6h

TABLE 5.12: The average time to train an SVM for all 5 folds (Avg. Train Time) and similarly to perform classification (Avg. Classify Time) is compared for varying training data sizes.

Even with just 100 negative examples, we are achieving an AUC close to 0.9 for all representations. This success rate may be sufficient for some domains, and therefore only a few negative examples are needed. It is shown that using 500 negative examples is able to achieve AUC scores that are within 1.5% of the full dataset for FC, IM and IG kernels. With the FS kernel, a higher AUC is achieved using just 500 examples as opposed to using all 25 thousand, all of the information.

These results suggest that training with a sample of the negative examples is a good alternative. In larger systems where millions of molecules are available, training with less examples would make it still feasible to create models and run experiments using this data.

5.7 Summary

In this Chapter we have tested two new graph kernels the FC and FS kernel, along with known infinite-length graph kernels (IM and IG) on two datasets. We have also tested three extensions to simple graph kernels including soft-matching, TP soft-matching and single gaps. For both datasets, we experimented with three graph-based representations including MG, RG and TP graphs. Finally, we have experimented with models that include fewer ‘not active’ examples in the training phase.

Chapter 6

Experiments in Lead Hopping

Lead hopping is the ability to identify molecules that belong to a different chemical series while still being active with a target. The Chapter will analyze combinations of graph kernels, graph colouring and molecular representations to determine which is best suited for lead hopping. We begin by introducing the lead hopping problem and reviewing recent literature in Section 6.1. Next, in Section 6.2, we will describe how two sets of data are built in order to model lead hopping. Although we compare our models using AUC scores (as described in Section 6.3), we also give the active molecules found near each cluster center (Section 6.4) as well as the closest actives between splits (Section 6.5) to aid in our analysis. A kernel between fingerprints is introduced in Section 6.6 and is compared with graph kernel models. We will describe the parameters of graph kernel models as well as colourings and representations in Section 6.8. After, we give an experimental methodology to test for lead hopping in Section 6.8. We give results for the first dataset in Section 6.9 and the second dataset in Section 6.10. Finally, we summarize in Section 6.11.

6.1 Lead Hopping

In this Section, we describe the lead hopping problem and review some recent literature in this area. *Lead hopping* (also known as scaffold hopping) is the ability to identify molecules that belong to a different chemical series while still being active with a target. Although chemists, pharmacologists and patent attorneys may have different views of what defines a unique chemical series (Schneider et al. (2006, 1999b)), in a QSAR sense it is a group of molecules which have the same core or central structure. As such, molecules within this group will only vary by a few atoms added or removed to the outer edge of the molecular graph. Lead hopping involves learning patterns from training data and predicting a separate test set that has very different molecular graphs. Learning algorithms rely on a similarity measure between molecules to make predictions, so to

make learning possible, one must define descriptors for learning that are as different as possible from the calculation of molecular similarity used to define lead hopping.

In drug discovery, the motivation to find models that lead hop is clear. Often when developing a drug, issues arise with a specific lead series because of its biological activity or inability to meet certain ADMET properties (absorption, distribution, metabolism, excretion, and toxicity) (Renner and Schneider (2006)). As a result, this lead series cannot be used and one must have a back-up lead series (Zhao (2007)) or else the drug discovery cycle would have to restart. Furthermore, there are commercial motivations to find new leads. After identifying a drug candidate, one may determine that a patent exists for this series of drug. Therefore, it may not be commercially viable to proceed because of this intellectual property. From another perspective, at an early stage of the drug discovery process, one may be given a commercially successful, patented drug, and asked to identify an alternative which can be used to emulate this drug.

The key question in lead hopping research is which features of a molecule are independent enough from the similarity calculated using the structure of the molecular graphs to allow a model to lead hop. In general, there are four approaches to lead hop which include shape matching the entire molecule, pharmacophore searching, fragment replacement and similarity searching (Böhm et al. (2004)). From these, the use of *topological pharmacophores* (TP) has prevailed. TPs are new labels for atoms or groups of atoms that perform certain functions in the molecule which may be involved in binding the ligand to the target. Using these TP labels, one can then relabel each of the atoms in the molecular graph to form a *TP graph*. Alternatively, the groups of atoms involved in one functional area can be collapsed into a single node to produce a *reduced graph*. We use the reduced graphs from Harper et al. (2004), although the weights can also be optimized in a more systematic fashion (Birchall et al. (2006)). Please refer to Chapter 2 for more discussion on these representations.

In recent literature, it has been demonstrated that fingerprints (used as feature vectors) derived from topological pharmacophores and reduced graphs are useful for lead hopping. Using fingerprints generated from TPs and an SVM model, Saeh et al. (2005) removed one cluster of actives (corresponding to a held-out lead series) and were able to recall 75% of the held-out actives and correctly classify almost all of the inactives. In Barker et al. (2006), using reduced graph representation and a similarity searching method it was found that reduced graphs find topologically different molecules. In Stiefl et al. (2006), a different type of reduced graphs is analyzed with lead hopping. They use *extended reduced graphs* which augments reduced graphs with specific differences within rings, substitution patterns and fused rings and found these can be used to find structurally diverse molecules.

6.2 Constructing Datasets That Exhibit Lead Hopping

To model lead hopping, we used two datasets that are derived from two publicly available datasets. The first is derived from the NCI-HIV¹, while the second is derived from Pyruvate Kinase data from PubChem² Assay ID 361 (containing 602 actives out of 51415 tested). In this Section, we will describe the methodology used by GlaxoSmithKline to generate these datasets for our research.

Each of the datasets were constructed using the following steps. First, the structures were preprocessed to remove salts and normalize the structures. After, the dataset was split into its constituent clusters. This was performed using GSK's proprietary software and Spotfire³ data visualization software. A complete link clustering algorithm (described in Martin et al. (2002) and Brown and Martin (1997)) uses daylight fingerprints and the Tanimoto similarity index to cluster molecules. This algorithm begins by assigning each molecule to its own cluster. It then successively merges the most similar clusters. This similarity is calculated by the minimum Tanimoto similarity between two molecules in each cluster. A single parameter is used to terminate the algorithm which occurs when no two clusters are more similar than this cut-off parameter. The maximum similarity between two molecules in dataset 1 is 0.4, while the maximum for dataset 2 is 0.5. These are very tough criterion which will create two very structurally dissimilar clusters.

The two largest clusters were selected for experimentation, while the remaining data was discarded. As described below, a final step filters molecules that are moderately active to form a binary classification problem: active vs. inactive. Please see Figure 6.1, for a visualization of this procedure.

¹<http://cactus.nci.nih.gov/ncidb/download.html>

²<http://pubchem.ncbi.nlm.nih.gov/>

³<http://spotfire.tibco.com/>

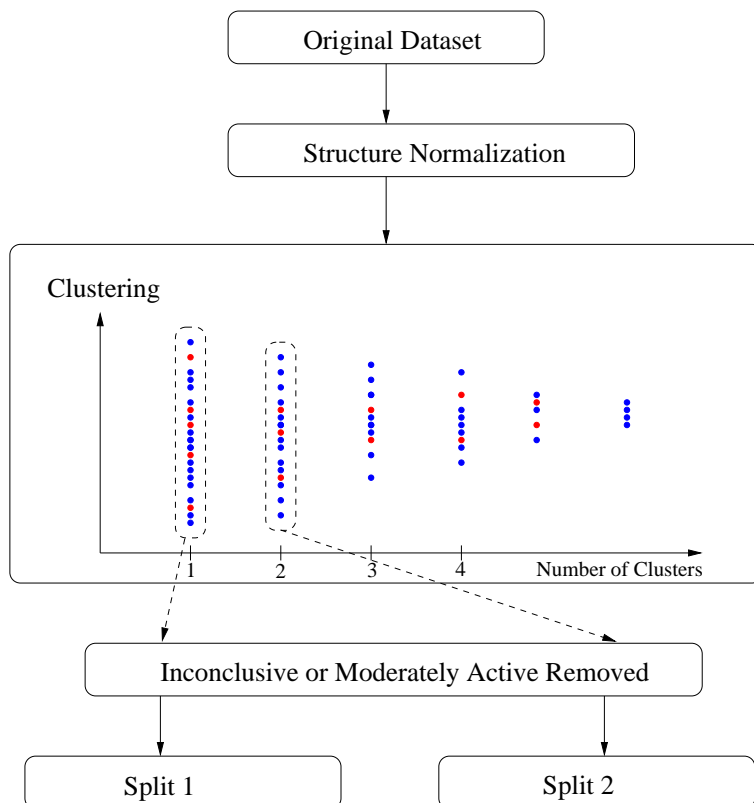


FIGURE 6.1: This diagram outlines the process to generate two splits of data which together can be used to model lead hopping. Given a dataset, this is done by picking out the two largest clusters. Each of these clusters (splits) now represent one lead or chemical series. Finally, we remove inconclusive molecules to form a binary dataset. This process was used to create splits for both the NCI-HIV and the Pyruvate Kinase data.

The similarity measure used for clustering was the *Tanimoto similarity*. We note that this similarity is also a type of kernel (Ralaivola et al. (2005)). Recall from Chapter 2, given two fingerprints, \mathbf{F}_1 and \mathbf{F}_2 , the Tanimoto similarity is calculated as:

$$T(\mathbf{F}_1, \mathbf{F}_2) = \frac{\langle \mathbf{F}_1, \mathbf{F}_2 \rangle}{\langle \mathbf{F}_1, \mathbf{F}_1 \rangle + \langle \mathbf{F}_2, \mathbf{F}_2 \rangle - \langle \mathbf{F}_1, \mathbf{F}_2 \rangle} \quad (6.1)$$

It would have been equally valid to split the data into clusters based on the normalization used with graph kernels. This would correspond to:

$$T(\mathbf{F}_1, \mathbf{F}_2) = \frac{\langle \mathbf{F}_1, \mathbf{F}_2 \rangle}{\sqrt{\langle \mathbf{F}_1, \mathbf{F}_1 \rangle \langle \mathbf{F}_2, \mathbf{F}_2 \rangle}} \quad (6.2)$$

Although clustering using the normalization from eq (6.2) may produce a slightly different split of the data from the two largest cluster selected with eq (6.1), this will not affect our analysis as either method presents a split that shows data which are structurally different based on the molecular graphs in some 2D sense. We will use the notation Split 1 and Split 2 for data from the first and second clusters respectively.

After the data has been clustered, we had to further reduce dataset 1 to create a binary classification problem. From dataset 1, split 1 had a total of 9,199 molecules while split 2 contained 9,972 molecules. Split 1 comprised 1.3% active (CA), 2.7% moderately active (CM) and 96.0% inactive (CI). Split 2 comprised 1.0% active (CA), 2.5% moderately active (CM) and 96.5% inactive (CI). In order to create a binary classification task we removed all of the moderately active (CM) molecules which left us with the problem active vs. inactive (CA vs CI). After this reduction, split 1 now contains 120 active and 8,831 inactive comprising 1.3% and 98.7% of the data. Split 2 contains 100 active and 9,623 inactive comprising 1.0% and 99.0% of the data.

We performed the same steps on dataset 2 (Pyruvate Kinase data) in order to create a binary classification problem. Split 1 had a total of 13,446 molecules, while split 2 contained 7,131 molecules. Split 1 comprised 1.5% active, 1.7% inconclusive (labeled moderately active in dataset 1) and 96.7% inactive. Split 2 comprised 0.9% active, 1.5% inconclusive and 97.6% inactive. In order to create a binary classification task we then removed all of the inconclusive molecules which left us with the problem active vs. inactive. After this reduction, split 1 now contains 204 active and 13,004 inactive comprising 1.5% and 98.5% of the data. Split 2 contains 64 active and 6,963 inactive comprising 0.9% and 99.1% of the data.

6.3 Measuring Success

In our lead hopping experiments, we will use the area under the ROC curve (AUC) to compare classifiers. The AUC is a single value which gives equal emphasis to the correct positive and negative examples. This is important as the proportion of the active and inactive classes are highly skewed with our data. Only 1.3% of the molecules in Dataset 1 are active and 1.5% in Dataset 2. If considering classification accuracy alone, a model which always predicted inactive would achieve a 98% success rate. The AUC of an ideal classifier is 1, while a random classifier has an AUC around 0.5 (Fawcett (2003)).

In addition to reporting AUC, we will also analyze the top active molecules retrieved by each model. As discussed in Section 6.1, the aim of lead hopping is to learn patterns from one set of molecules to predict a separate set with a different core structure of their molecular graph. Although this problem has been incorporated into our training and testing sets, we can confirm our results with a visual analysis. Potentially, we may find that some models perform well although their success is due to a weakness in dividing the data with Tanimoto clustering. We will visualize active molecules from the cluster centers of both splits and datasets in Section 6.4. After we will visualize the active molecules which are closest to active molecules in the other split in Section 6.5. Again, we give this analysis for both splits and datasets. Note that in each molecule image, we give the molecule number corresponding to the numbering used in each dataset.

6.4 Actives Near Cluster Centers

In order to confirm that our models lead hop, we provide a visual analysis of each cluster. In this Section, we will draw molecules which are closest to the cluster center. We begin by describing a method to calculate the center of a cluster where distance between examples is calculated with the Tanimoto similarity in Section 6.4.1. After, we graph 10 active molecules which are closest to the cluster center. Analysis of clusters (splits) from dataset 1 are given in Section 6.4.2 and clusters from dataset 2 are given in Section 6.4.3.

6.4.1 Calculating the Tanimoto Cluster Center

Using a basic clustering algorithm, such as K-means, the cluster center would be given by the average feature vector for the examples in a cluster. More formally, given a set of N fingerprints, $\mathbf{F}_1, \mathbf{F}_2, \dots, \mathbf{F}_N$, the cluster center is calculated as:

$$\mathbf{F}_{center} = \frac{\mathbf{F}_1 + \mathbf{F}_2 + \dots + \mathbf{F}_N}{N} \quad (6.3)$$

Although using a complete link clustering algorithm, as described in the previous Section, a different similarity can be defined between examples such as the Tanimoto similarity (eq (6.1)). As a result, the cluster center is not the same as eq (6.3). We define the center of the cluster using a point, $\hat{\mathbf{F}}$, that maximizes the Tanimoto similarity between $\hat{\mathbf{F}}$ and each fingerprint in the cluster.

$$\mathbf{F}_{center} = \max_{\hat{\mathbf{F}}} \sum_{i=1}^N T(\mathbf{F}_i, \hat{\mathbf{F}}) \quad (6.4)$$

$$= \max_{\hat{\mathbf{F}}} \sum_{i=1}^N \frac{\langle \mathbf{F}_i, \hat{\mathbf{F}} \rangle}{\langle \mathbf{F}_i, \mathbf{F}_i \rangle + \langle \hat{\mathbf{F}}, \hat{\mathbf{F}} \rangle - \langle \mathbf{F}_i, \hat{\mathbf{F}} \rangle} \quad (6.5)$$

This can be solved by taking the gradient of eq (6.5) and then iteratively estimating closer solutions.

$$\mathbf{F}_{t+1} = \mathbf{F}_t + \epsilon \nabla_{\mathbf{F}} \sum_{i=1}^N T(\mathbf{F}_i, \mathbf{F}_t) \quad (6.6)$$

Setting an appropriate step size, ϵ , and iterating until $|\mathbf{F}_{t+1} - \mathbf{F}_t| < \text{tol}$, we can find the cluster center. A fingerprint is composed of components $\mathbf{F} = (f_1, f_2, f_n)$ so $\nabla_{\mathbf{F}}$ is:

$$\nabla_{\mathbf{F}} = \left(\frac{\partial \mathbf{F}}{\partial f_1}, \frac{\partial \mathbf{F}}{\partial f_2}, \dots, \frac{\partial \mathbf{F}}{\partial f_n} \right) \quad (6.7)$$

We will use an initial guess for \mathbf{F}_0 using eq (6.3). The gradient is given by:

$$\nabla_{\mathbf{F}} \sum_{i=1}^N T(\mathbf{F}_i, \mathbf{F}_t) = \frac{\mathbf{F}_i (\langle \mathbf{F}_i, \mathbf{F}_i \rangle + \langle \mathbf{F}_t, \mathbf{F}_t \rangle - \langle \mathbf{F}_t, \mathbf{F}_i \rangle) - \langle \mathbf{F}_t, \mathbf{F}_i \rangle (2\mathbf{F}_t - \mathbf{F}_i)}{(\langle \mathbf{F}_i, \mathbf{F}_i \rangle + \langle \mathbf{F}_t, \mathbf{F}_t \rangle + \langle \mathbf{F}_i, \mathbf{F}_t \rangle)^2} \quad (6.8)$$

$$= \frac{\mathbf{F}_i \langle \mathbf{F}_i, \mathbf{F}_i \rangle + \mathbf{F}_i \langle \mathbf{F}_t, \mathbf{F}_t \rangle - 2\mathbf{F}_t \langle \mathbf{F}_i, \mathbf{F}_t \rangle}{(\langle \mathbf{F}_i, \mathbf{F}_i \rangle + \langle \mathbf{F}_t, \mathbf{F}_t \rangle + \langle \mathbf{F}_i, \mathbf{F}_t \rangle)^2} \quad (6.9)$$

6.4.2 Cluster Centers of Dataset 1 - NCI-HIV Data

In this Section, we describe the 10 closest active molecules to the cluster center of splits in dataset 1. The data was originally separated into two clusters using fingerprints of the molecular graph and the Tanimoto similarity (described in Section 6.2). We calculate the center of each cluster using an iterative approach described in Section 6.4.1. The center of both cluster in dataset 1 could be found in around 50 iterations by setting $\epsilon = 0.01$ and $\text{tol} = 0.001$.

The closest 10 active molecules in split 1 to the cluster center are given in Figure 6.2. A common pattern among these actives appears to be the presence of two aromatic rings separated by a group of 1 Sulphur and 2 Oxygen atoms (seen in molecules 633001, 624231, 629267, 633011 and 667950). Also, there is a group of atoms (1 Nitrogen double bonded to 2 Oxygen atoms) attached as side groups to aromatic rings in molecules 629273, 633001, 624231, 629267 and 667950. Molecule 7229, 638478, 641295 do not share these properties, although as they are large molecules with many rings, they will have many overlapping features with other molecules according to their Tanimoto fingerprint similarity. Overall, all of these molecules are similar according to the clustering algorithm and molecular fingerprints.

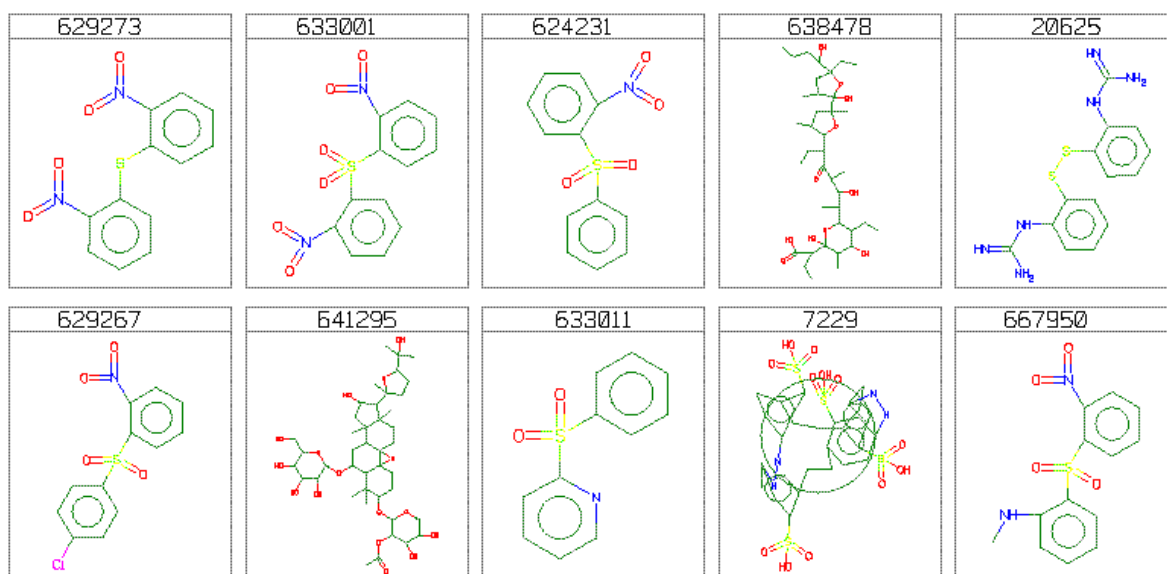


FIGURE 6.2: The figure gives the 10 closest molecules to the cluster center of split 1, dataset 1 (NCI-HIV data). Molecule names are given above each molecular graph. They are sorted from the closest in the top-left (molecule 629273) to the 10-th closest in the bottom-right (molecule 667950).

The closest 10 active molecules in split 2 to the cluster center are given in Figure 6.3. There appears to be two groups of similar molecules. The first set includes molecules 647648, 637646 and 646444, sharing a similar central structure of two rings. They are only different by a few groups of atoms attached at the side. The second set includes molecules 50848 and 50850. Again, these are almost identical except for a few atoms attached at one end of the molecule. The remaining five (624151, 695836, 128701, 684881 and 318534) do not appear to be similar visually, although they are grouped together in the same cluster according to their Tanimoto similarity from molecular fingerprints

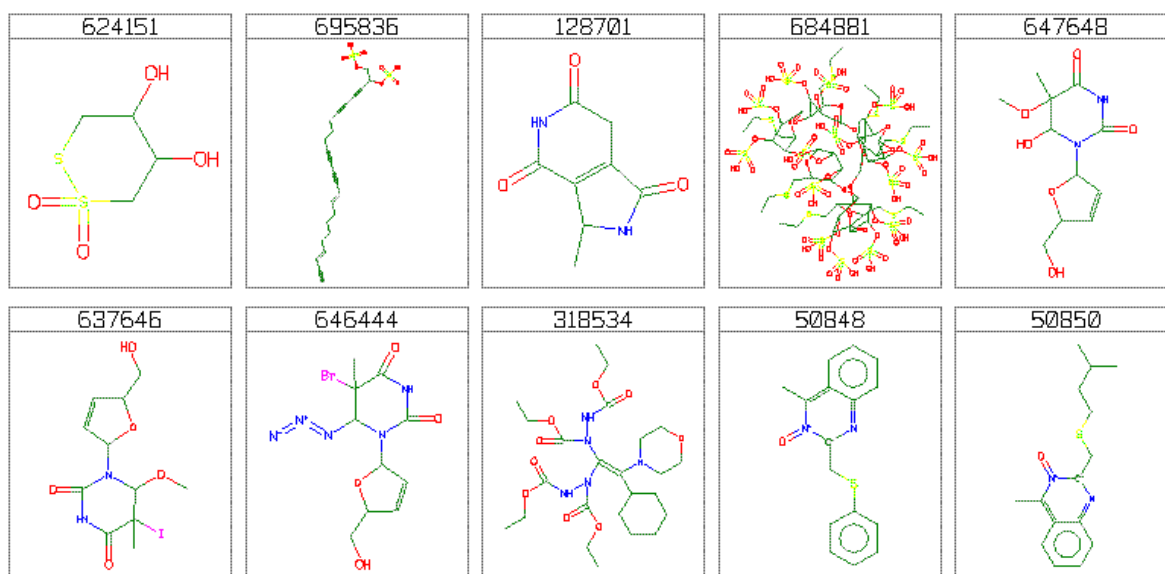


FIGURE 6.3: The figure gives the 10 closest molecules to the cluster center of split 2, dataset 1 (NCI-HIV data). Molecule names are given above each molecular graph. They are sorted from the closest in the top-left (molecule 624151) to the 10-th closest in the bottom-right (molecule 50850).

6.4.3 Cluster Centers of Dataset 2 - Pyruvate Kinase Data

In this Section, we will show the 10 closest active molecules to the cluster center of dataset 2. Using an iterative approach described in Section 6.4.1, the center of both clusters in dataset 2 could be found in around 50 iterations by setting $\epsilon = 0.01$ and $\text{tol} = 0.001$.

The closest 10 active molecules in split 1 to the cluster center are given in Figure 6.4. Two pairs of molecules (5356192,5771437 and 445154,3238641) appear to match visually, although the remaining molecules do not. Despite this, they are grouped together in the same cluster according to the Tanimoto similarity from their molecular fingerprints.

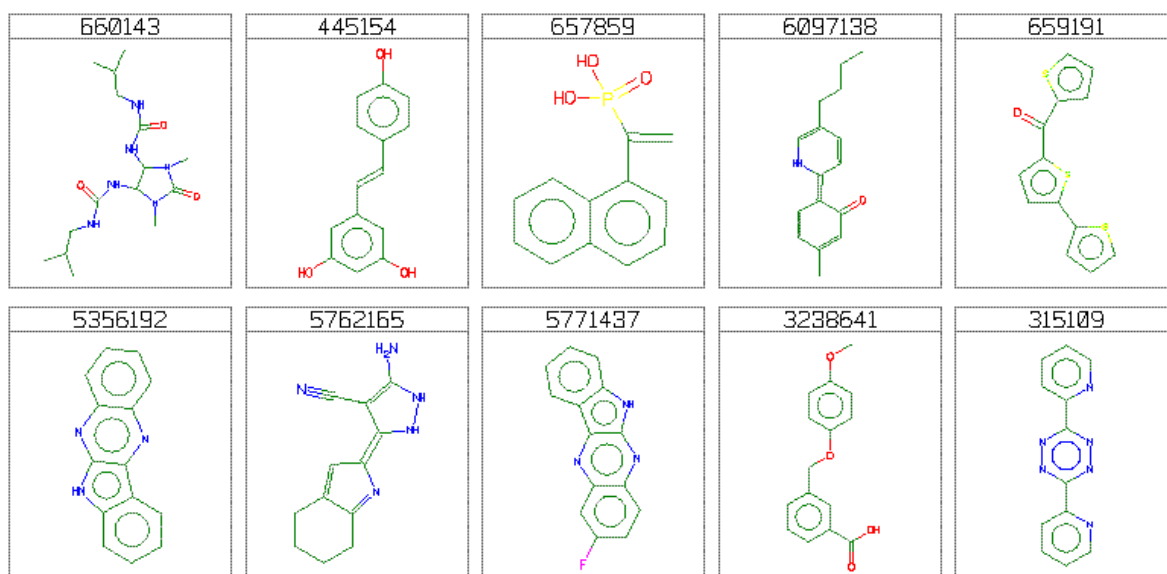


FIGURE 6.4: The figure gives the 10 closest molecules to the cluster center of split 1, dataset 2 (Pyruvate Kinase data). Molecule names are given above each molecular graph. They are sorted from the closest in the top-left (molecule 660143) to the 10-th closest in the bottom-right (molecule 315109).

The closest 10 active molecules in split 2 to the cluster center are given in Figure 6.5. Besides one pair (2195987,2193330), the remaining molecules do not appear to match visually. Despite this, they are grouped together in the same cluster according to the Tanimoto similarity from their molecular fingerprints.

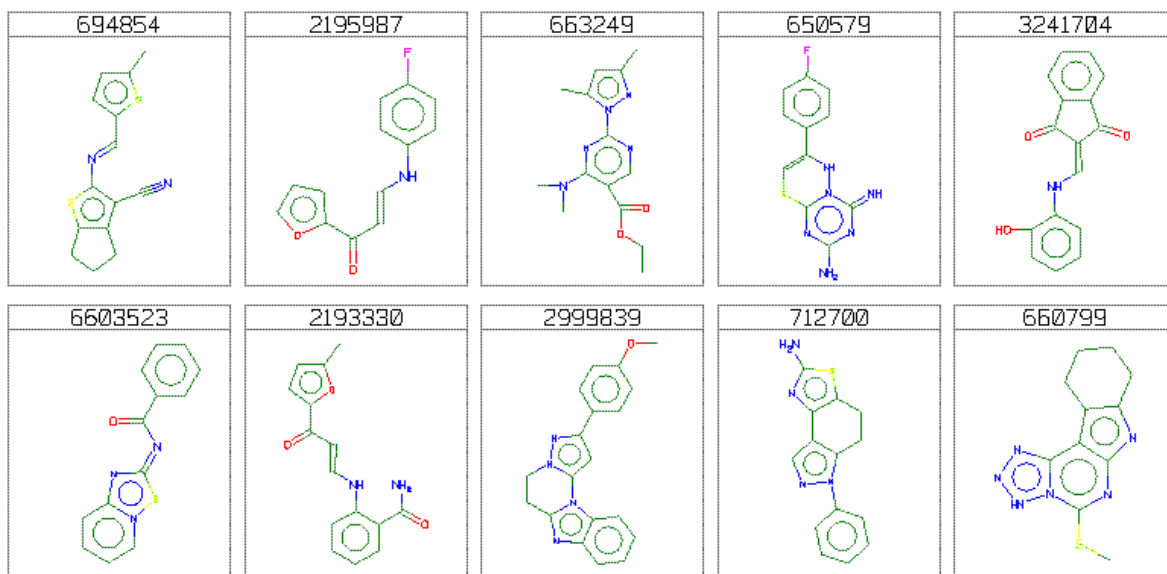


FIGURE 6.5: The figure gives the 10 closest molecules to the cluster center of split 2, dataset 2 (Pyruvate Kinase data). Molecule names are given above each molecular graph. They are sorted from the closest in the top-left (molecule 694854) to the 10-th closest in the bottom-right (molecule 660799).

6.5 Similarity of Actives Between Splits

In this Section, we show the active molecules in one split that are closest to the actives in the other split. The similarity is calculated using the Tanimoto similarity for molecular fingerprint since we used this to separate the clusters (as described in Section 6.2). We are interested in viewing the actives that are close to the other split, as these molecules should be the easiest to find by a model using similar molecular descriptors. For each of the 10 closest actives, we also give the corresponding closest active from the other split so that we can visually determine the differences.

6.5.1 Dataset 1

The closest molecules in Split 1 to molecules in Split 2 are given in Table 6.1. There appears to be no similarity visually between each active and the closest active in the opposite split.

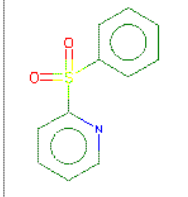
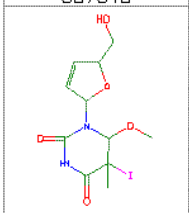
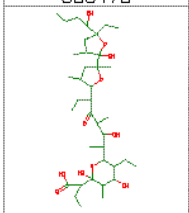
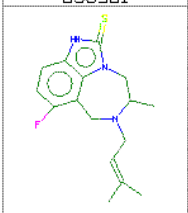
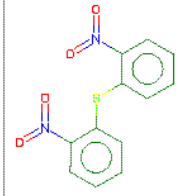
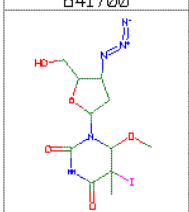
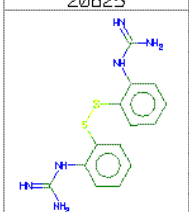
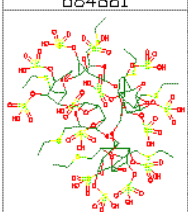
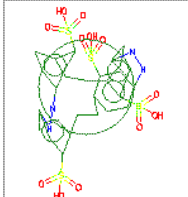
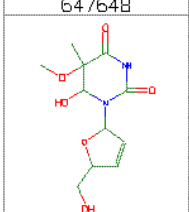
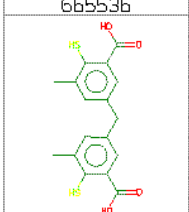
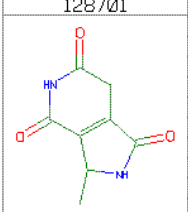
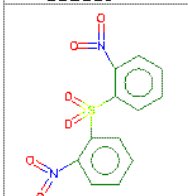
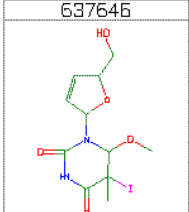
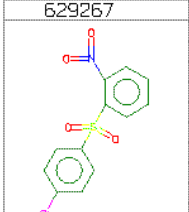
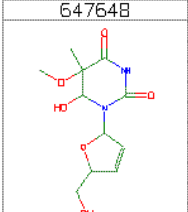
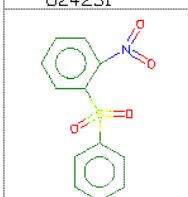
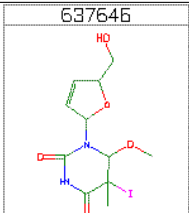
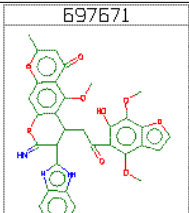
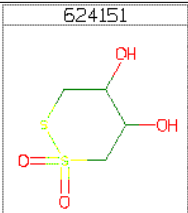
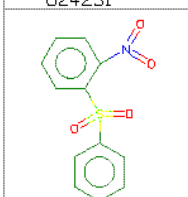
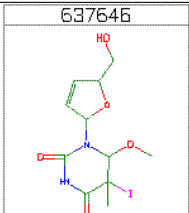
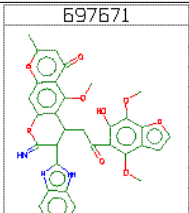
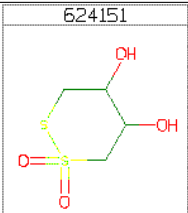
Rank	Split 1	Closest Split 2	Rank	Split 1	Closest Split 2
1	633011 	637646 	6	638478 	638581 
	629273 	641700 		20625 	684881 
2	7229 	647648 	8	665536 	128701 
	633001 	637646 		629267 	647648 
3	624231 	637646 	10	697671 	624151 
	624231 	637646 		697671 	624151 

TABLE 6.1: The 10 closest actives in split 1 to actives in split 2, dataset 1 (NCI-HIV data), are listed. We draw each of these molecules in column ‘Split 1’ and the corresponding closest molecules in ‘Closest Split 2’. They are ranked from the closest (molecule 633011) to the 10-th closest (molecule 697671).

The closest molecules in Split 2 to molecules in Split 1 are given in Table 6.2. There appears to be no similarity visually between each active and the closest active in the opposite split. These 10 actives are closest to only two molecules from split 1 (633011 and 629273). Among these 10 actives, 8 of them have a small group of 3 Nitrogen (N) atoms attached to the side of the atom (known as an azide group), which may be involved in binding. This group does not appear in the closest active molecule from the other split.

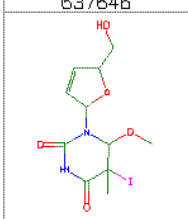
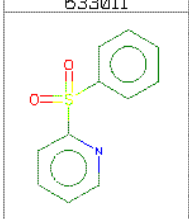
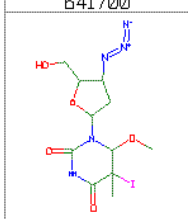
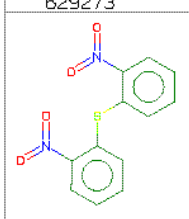
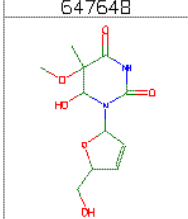
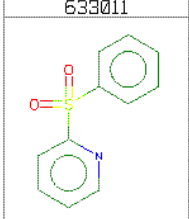
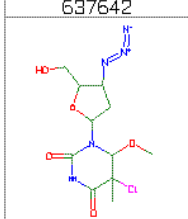
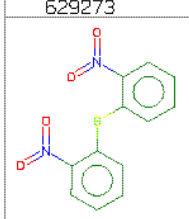
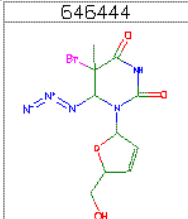
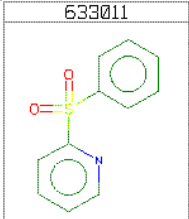
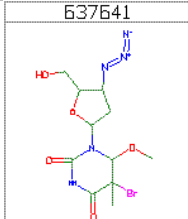
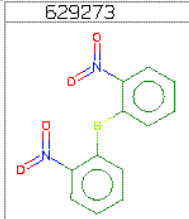
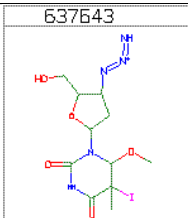
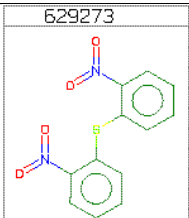
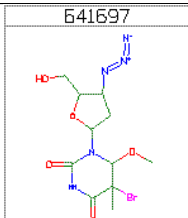
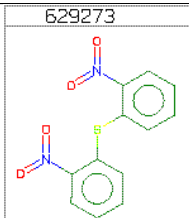
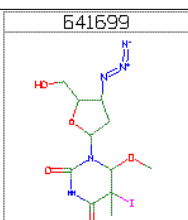
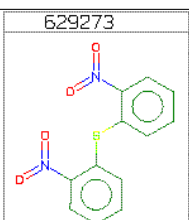
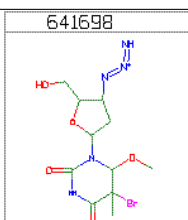
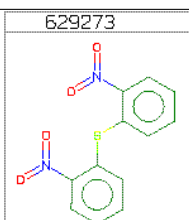
Rank	Split 2	Closest Split 1	Rank	Split 2	Closest Split 1
1	637646 	633011 	6	641700 	629273 
2	647648 	633011 	7	637642 	629273 
3	646444 	633011 	8	637641 	629273 
4	637643 	629273 	9	641697 	629273 
5	641699 	629273 	10	641698 	629273 

TABLE 6.2: The 10 closest actives in split 2 to actives in split 1, dataset 1 (NCI-HIV data), are listed. We draw each of these molecules in column ‘Split 2’ and the corresponding closest molecules in ‘Closest Split 1’. They are ranked from the closest (molecule 637646) to the 10-th closest (molecule 641698).

6.5.2 Dataset 2

The closest molecules in Split 1 to molecules in Split 2 are given in Table 6.3. There appears to be no similarity visually between each active and the closest active in the opposite split.

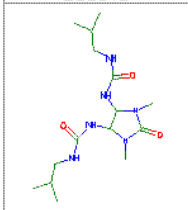
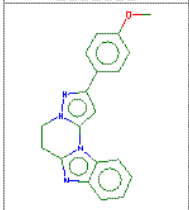
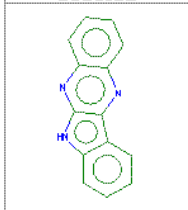
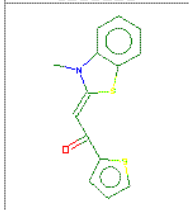
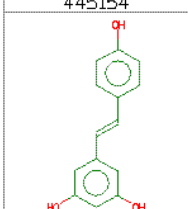
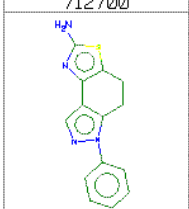
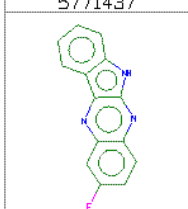
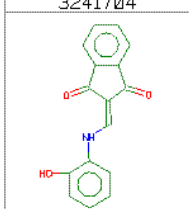
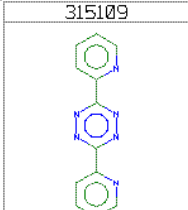
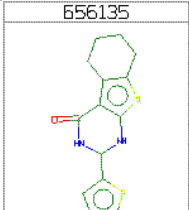
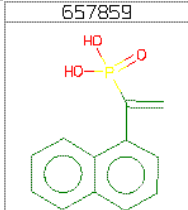
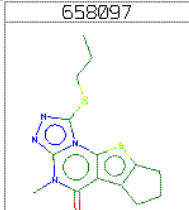
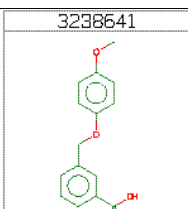
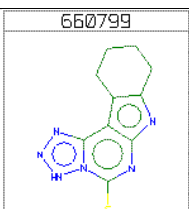
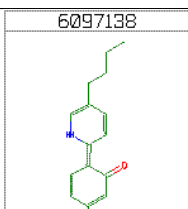
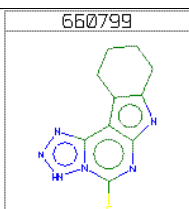
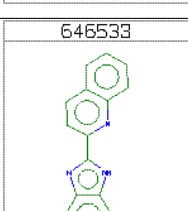
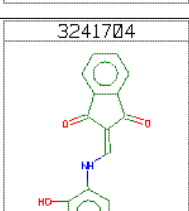
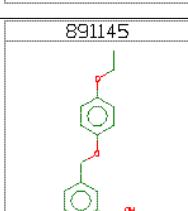
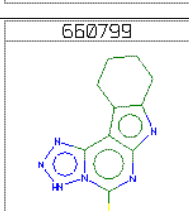
Rank	Split 1	Closest Split 2	Rank	Split 1	Closest Split 2
1	660143 	2999839 	6	5356192 	1633229 
2	445154 	712700 	7	5771437 	3241704 
3	315109 	656135 	8	657859 	658097 
4	3238641 	660799 	9	6097138 	660799 
5	646533 	3241704 	10	891145 	660799 

TABLE 6.3: The 10 closest actives in split 1 to actives in split 2, dataset 2 (Pyruvate Kinase data), are listed. We draw each of these molecules in column ‘Split 1’ and the corresponding closest molecules in ‘Closest Split 2’. They are ranked from the closest (molecule 660143) to the 10-th closest (molecule 891145).

The closest molecules in Split 2 to molecules in Split 1 are given in Table 6.4. There appears to be no similarity visually between each active and the closest active in the opposite split

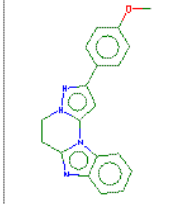
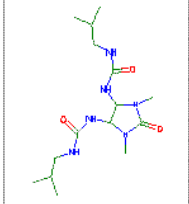
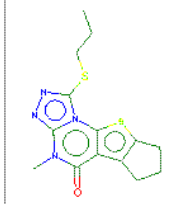
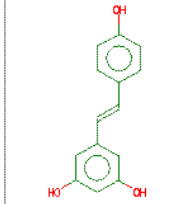
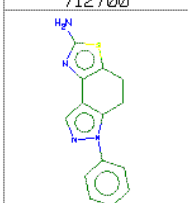
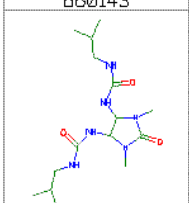
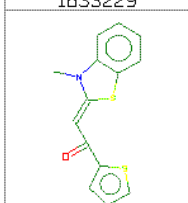
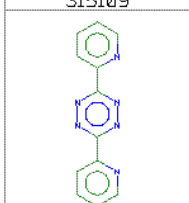
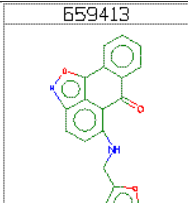
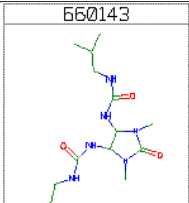
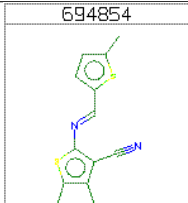
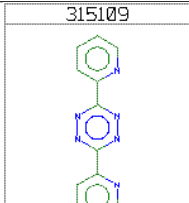
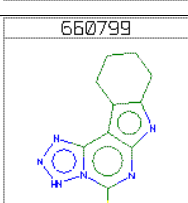
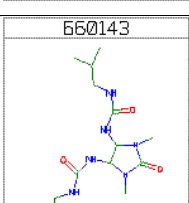
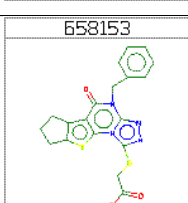
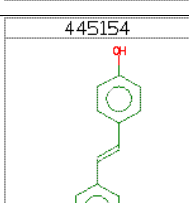
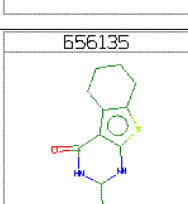
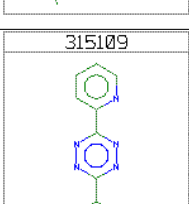
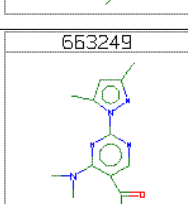
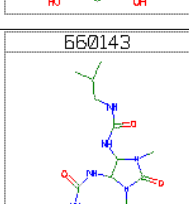
Rank	Split 2	Closest Split 1	Rank	Split 2	Closest Split 1
1	2999839 	660143 	6	658097 	445154 
2	712700 	660143 	7	1633229 	315109 
3	659413 	660143 	8	694854 	315109 
4	660799 	660143 	9	658153 	445154 
5	666135 	315109 	10	663249 	660143 

TABLE 6.4: The 10 closest actives in split 2 to actives in split 1, dataset 2 (Pyruvate Kinase data), are listed. We draw each of these molecules in column ‘Split 1’ and the corresponding closest molecules in ‘Closest Split 2’. They are ranked from the closest (molecule 2999839) to the 10-th closest (molecule 663249).

6.6 Kernels for Fingerprints

In this Chapter we will also introduce a new kernel that uses fingerprints derived from MG, RG and TP. We test models that use a kernel between fingerprints for each representation. A similar kernel was also used in Ralaivola et al. (2005). The MG kernel

is calculated using the Daylight⁴ proprietary software. Each MG fingerprint contains binary values and is length 1024. Similarly, the RG and TP fingerprints contains binary value and has a length 1024. The RG and TP fingerprints are calculated using internal GSK software. We use the notation FPRINT for experiments with fingerprints. The kernel is a simple inner product between each of these feature vectors.

$$\kappa(\mathbf{F}_1, \mathbf{F}_2) = \langle \mathbf{F}_1, \mathbf{F}_2 \rangle \quad (6.10)$$

As shown in eq (6.10), this kernel does not use any normalization. Note that a similar measure was used for separating the splits, the Tanimoto similarity, in eq (6.1). Using this normalization for the kernel, we would obtain the Tanimoto kernel introduced in Ralaivola et al. (2005). It would have also been possible to use the normalization from our graph kernels for the kernel as in eq (6.2).

6.7 Experimental Design Decisions for Graph Kernel Models

In this Section, we briefly review some design decisions for graph kernels introduced in Chapter 5. We will discuss what graph kernels, graph colourings and representations are used to test for Lead Hopping.

Among the many graph kernels introduced in Chapter 3, we show that walk-based graph kernels can be calculated in a dynamic programming framework in Chapter 4. In our research we will use four kernels calculated using the dynamic approach, two new finite-length kernels and two known infinite-length kernels. We use the FC kernel that uses single walks of a certain length as features as well as the FS kernel that uses all walks up to a specified length. We will test walks from length 1 to 15 in our research. We also use the IM and IG graph kernel that down-weight longer walks effectively counting all walks up to infinite length. We will use 15 iterations of our dynamic programming equations for all kernels and test parameters for each using weights (0.01, 0.05, 0.1, 0.2, ..., 0.9).

We then add extensions soft-matching (SM), TP soft-matching (TPSM), and single gaps (1G). We add this to the finite-length kernels by weighting these features using the σ_{sm} and σ_{1g} parameters with values (0.01, 0.05, 0.1, 0.2, ..., 0.5). The 1G extension evenly down-weights all product vertices that contain a wild-card by σ_{1g} . The SM extension evenly down-weights all non-matching vertices by σ_{sm} . The TPSM extension is only added to the MG representation as we only test soft-matching from atom labels to TP labels. For each atom in the molecular graph, there are 6 possible TP labels (discussed in Chapter 2): positively ionizable, negatively ionizable, aromatic ring, aliphatic ring,

⁴<http://www.daylight.com/>

hydrogen bond donor and hydrogen bond acceptor. If two vertices in the product graph do not have matching atom labels, they can still be matched up to 100% if all of the TP labels match. If only a few TP labels match, the amount is summed for each matching TP label according to: $\frac{1}{4}, \frac{1}{4}, \frac{1}{16}, \frac{1}{16}, \frac{1}{16}, \frac{1}{16}$. We give more importance to the first two (positively ionizable and negatively ionizable) as experts believe that these are most important among TP labels. In addition to this weighting scheme, the matching is multiplied by σ_{sm} to choose the amount of weight given to the TPSM feature. For each of these models, the SVM-C value used for testing is the one chosen in the loop over 7 choices for C when selecting parameters.

Another consideration that must be made with the graph kernel approach is choosing a graph colouring method. In our experiments, we modify the original graph by colouring vertex labels and test two different colouring methods. The first method used in Gärtner (2005a) (notated as N-Colour) creates new vertex labels by combining each vertex label and each neighbouring vertex label. We will also use the Morgan colour method introduced in Mahé et al. (2004) with 1, 2 and 3 iterations (notated by M1-Colour, M2-Colour and M3-Colour). Finally, we will also experiment with the original graph (notated by No-Colour).

A third consideration is choosing a graph representation of the molecule. We will test three representations including the original molecular graph (MG), reduced graphs (RG) and topological pharmacophore graphs (TP). Please refer to Chapter 2 for discussion of these types of graphs.

6.8 Experimental Methodology

In this Section we describe our methodology for testing lead hopping on dataset 1 and 2. As described in Section 6.2, given a dataset, the first step is to generate two splits of the data.

Using these two splits, we will run experiments where we train on split 1 and test on split 2 (split 1 vs. split 2) and also train on split 2 and test on split 1 (split 2 vs. split 1). We further divide each of the splits into a 10% portion for parameter tuning and 90% for training. For split 1 vs. split 2, we will parameter tune on 10% of split 1, then using the best parameters train a single SVM model on the remaining 90%. We will then test this model on 100% of the examples in split 2. The same steps are taken for split 2 vs. split 1.

In order to choose a parameter (colouring and graph kernel parameter) from the 10% parameter tuning set, we used a 5-fold cross-validation (CV). This would return the average AUC from 5 iterations of training on 4 folds and testing on the held-out fold. For each of these 5 folds, we would choose the best SVM-C value from $10^3, 10^2, 10^1, 10^0$,

10^{-1} , 10^{-2} and 10^{-3} . Therefore, if we want to test a certain colour and graph kernel parameter we have to run the SVM a total of 35 times (5 folds by 7 values for SVM-C). To make this more efficient, for each parameter we pre-built the kernel matrix on the 10% of data to run these 35 iterations of the SVM.

In the parameter tuning phase, we first choose graph colouring and graph kernel parameters for the four basic kernels. This is done by fixing the value for the graph kernels and testing each colouring method. The finite-length kernels (FC and FS) used a fixed setting with the walk-length parameter to 5. The infinite-length kernels (IM and IG) use a fixed setting with the down-weight parameter set to 0.1. We choose the best colouring method among No-Colour, N-Colour, M1-Colour, M2-Colour and M3-Colour. In the next phase, the graph kernel parameters for each kernel and representation (MG, RG and TP) are selected. We use the colouring type chosen in the previous phase and test a range of values for each of the four basic graph kernels. For the FC and FS kernel, we test walk-lengths from 1 to 15 and select the walk corresponding to the highest AUC. For the IM and IG kernel, we test the down-weight parameter (σ_{SM} , σ_{1G}) from 0.01, 0.05, 0.1, 0.2, ..., 0.9 and select the weight corresponding to the highest AUC. For each of these models, the SVM-C value used for testing is the one chosen in the loop over 7 choices for C when selecting parameters.

The best kernel among the finite-length kernels (FC and FS) are selected and tested with the SM, TPSM and 1G extensions. We have decided to only test these extensions with the finite-length kernels as these extensions are more relevant to longer walks. Infinite-length walks down-weight longer walks and so this feature will have little impact. To add this extension to the finite-length kernel we must choose the weight parameter for each extension. Given the optimal colouring and graph kernel parameter from the previous stage, we then test a range of values 0.01, 0.05, 0.1, 0.2, ..., 0.5 for σ_{sm} and σ_{1g} and select the best parameter to weight the influence from the extensions. The TPSM extension is only added to the MG representation as we only test soft-matching from atom labels to TP labels. For each of these models, the SVM-C value used for testing is the one chosen in the loop over 7 choices for C when selecting parameters.

Using the same folds and parameter tuning sets as above, we also need to parameter tune our fingerprint models. Only the SVM-C parameter needs to be chosen for this model. Using the same folds of the parameter tuning 10% of data, we choose the highest AUC from SVM-C values 10^3 , 10^2 , 10^1 , 10^0 , 10^{-1} , 10^{-2} and 10^{-3} with a 5-fold cross-validation. As mentioned earlier, using these optimized parameters we can then build a single SVM model on the remaining 90% of the training data and test it on 100% of the data from the other split.

6.9 Dataset 1 - NCI-HIV Data

In this Section, we give results for the NCI-HIV dataset. We give results for training on the first split and testing on the second cluster (split 1 vs split 2) in Section 6.9.1. After, we give results for training on the second and testing on the first cluster (split 2 vs split 1) in Section 6.9.2.

6.9.1 Split 1 vs. Split 2

Training on split 1 and classifying split 2, we performed experiments for MG, RG and TP graph representations. We parameter tune graph kernels on 10% of the data. The optimal parameters are given in Table 6.5. We remove the ending ‘-Colour’ when describing colouring types for brevity. The column Param gives the parameters for each of the kernels. FC and FS are parameterized by a walk-length. IM and IG are parameterized by a down-weight parameter. SM, TPSM are parameterized by a down-weight σ_{sm} and 1G extensions are parameterized by a down-weight σ_{1g} . The FC or FS kernel parameter used with the extensions are found in the corresponding rows above. The column C gives the SVM-C parameter.

	MG			RG			TP		
	Colour	Param	C	Colour	Param	C	Colour	Param	C
FC	N	2	10^{-1}	M1	4	10^{-3}	N	7	10^{-3}
FS	No	3	10^2	M1	5	10^{-3}	M3	1	10^{-3}
IM	No	0.4	10^3	No	0.01	10^2	M2	0.8	10^{-3}
IG	No	0.6	10^{-1}	M1	0.05	10^{-1}	M2	0.01	10^{-3}
F+SM	N	0.05	10^{-2}	M1	0.3	10^{-3}	N	0.3	10^3
F+TPSM	N	0.01	10^{-2}	-	-	-	-	-	-
F+1G	N	0.01	10^0	M1	0.05	10^{-3}	N	0.01	10^{-2}
FPRINT	-	-	0.1	-	-	1000	-	-	0.1

TABLE 6.5: Parameter tuning split 1 vs. split 2 for dataset 1

The results for split 1 vs. split 2 are given in Table 6.6. The best model without extensions used MG and an FC kernel. The FC and FS kernel for the RG representation performed very poorly as it must have overtrained to the parameter tuning set. These models chose a relatively long walk-length of 4 and 5 considering that reduced graphs usually only contain a few vertices and the specificity of each label was increased with M1 colouring. The soft-matching (SM) extension improved all three representations (MG, RG and TP). Only a slight increase in AUC was seen when adding SM to the MG and FC model as this extension used the largest down-weight (0.01). RG and TP chose much larger down-weights (0.3) for soft-matching. Adding the 1G extension to

MG, RG and TP models gave a consistent improvement according to the AUC. It is somewhat surprising to see that MG and FPRINT kernel had an AUC greater than 0.5, although the SVM must find non-linear patterns that were not addressed when separating the data in the clustering phase. When analyzing the results, we note that this lead hopping is a difficult problem. A score above 0.5 allows us to potentially find molecules from another lead which could be extremely beneficial. Therefore, even a small hit-rate can be considered a success.

	MG	RG	TP
FC	0.7178	0.3872	0.5440
FS	0.6281	0.4403	0.6609
IM	0.6399	0.6507	0.6087
IG	0.4694	0.5300	0.6563
F	0.7178	0.4403	0.5440
F+SM	0.7198	0.4557	0.6015
F+TPSM	0.7155	-	-
F+1G	0.7156	0.5300	0.6550
FPRINT	0.6652	0.6179	0.5772

TABLE 6.6: Results for dataset 1, split 1 vs. split 2. We report the single AUC score from training on 90% of split 1 and testing on 100% of split 2.

In Figure 6.6, we show the number of actives found in the top 300 molecules ranked by each model. We also give the number of unique actives for each model compared to those returned by MG and FPRINT kernel. We compare to this model as it is nearly identical to the measure used to separate the clusters except for a normalization factor in the Tanimoto similarity. It is interesting to note that almost all actives returned by the other models are unique to those from the MG and FPRINT kernel.

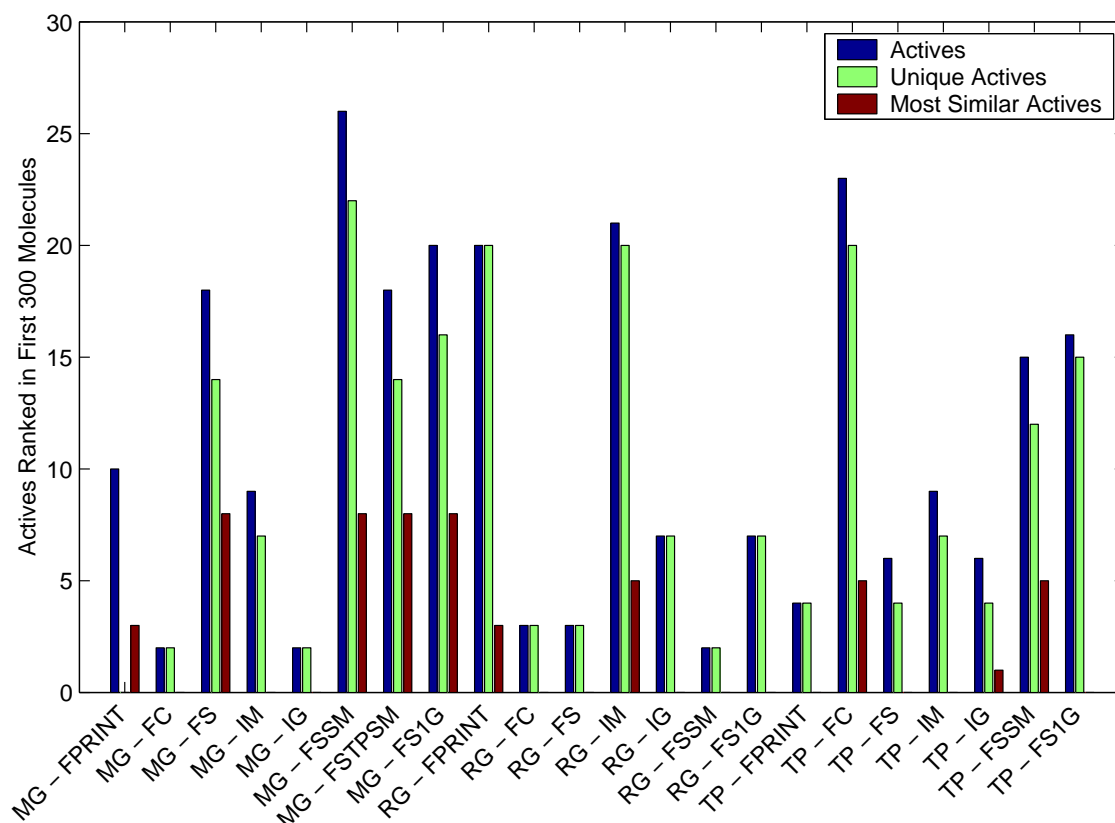


FIGURE 6.6: This graph shows the number of unique actives found in the top 300 molecules ranked by the SVM (blue). We also give the number of unique actives found compared to the MG-FPRINT model (green). Finally, we show the number of these actives which are from the top 10 most similar to the other split's actives described in Section 6.5 (red).

In Figure 6.7, we give the top 10 actives ranked by the SVM output for the best performing graph kernel model. As shown in Table 6.6, the model with MG graphs and FS kernel with SM gave an AUC of 0.7198. 8 of these actives are listed as the closest to actives in the opposite split (described in Section 6.5). These actives all have a similar structure and have a group of 3 Nitrogen atoms (an Azide group).

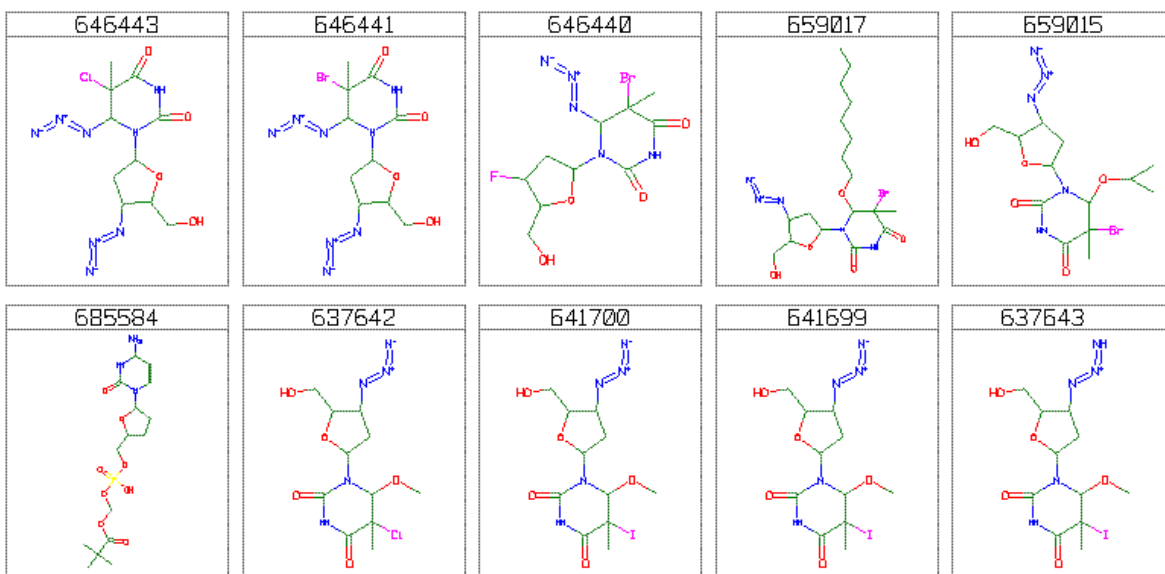


FIGURE 6.7: Highest ranked actives in MG - FS+SM model for split 1 vs. split 2 of dataset 1. They are in order from top left to bottom right.

6.9.2 Split 2 vs. Split 1

We follow the same set-up as in the previous section and train on split 2 and classify split 1 of dataset 1.

	MG			RG			TP		
	Colour	Param	C	Colour	Param	C	Colour	Param	C
FC	No	4	10^{-2}	M1	4	10^{-3}	No	3	10^{-2}
FS	No	1	10^0	N	4	10^{-3}	No	6	10^{-2}
IM	M2	0.6	10^{-1}	M3	0.9	10^3	N	0.05	10^3
IG	M1	0.2	10^{-2}	M3	0.01	10^3	M1	0.01	10^{-2}
F+SM	No	0.3	10^2	N	0.2	10^{-2}	No	0.2	10^{-3}
F+TPSM	No	0.3	10^{-2}	-	-	-	-	-	-
F+1G	No	0.4	10^3	N	0.05	10^{-2}	No	0.5	10^{-2}
FPRINT	-	-	10^{-1}	-	-	10^2	-	-	10^0

TABLE 6.7: Parameter tuning split 2 vs. split 1 for dataset 1

The results for split 2 vs. split 1 are given in Table 6.8. A single SVM is trained on 90% of split 1 using the parameters from Table 6.7 and then 100% of the data from split 2 is tested. We give the AUC score to measure performance of each model. Without including extensions, the best model was found with MG and the IM kernel. Both IM and IG kernels with the RG representation chose M3-Colouring and performed

the worst. M3-Colour adds the most structural information into a vertex label, so this representation must have been too specific not allowing for generalization to other examples. The SM extension gave a huge improvement to the models from MG, RG and TP. The largest improvement was from adding SM to the FC kernel with the MG representation where the AUC nearly double from 0.4800 to 0.8179, although adding SM to the RG and TP models gave large increases as well. The TPSM also gave a large increase to the FC and MG model. Adding 1G to the FC and MG model gave a lower AUC, possibly because one of the largest weights was chosen (0.4). Adding 1G to RG and TP gave much better models than without. The FPRINT kernels all found models better than random (0.5) with the highest from MG.

	MG	RG	TP
FC	0.4800	0.7858	0.6827
FS	0.7046	0.7440	0.5459
IM	0.8305	0.4474	0.7047
IG	0.7469	0.5157	0.8139
F	0.4800	0.7440	0.5459
F+SM	0.8179	0.8221	0.6527
F+TPSM	0.7915	-	-
F+1G	0.4364	0.8123	0.7242
FPRINT	0.7050	0.7418	0.3961

TABLE 6.8: Results for dataset 1, split 2 vs. split 1. We report the AUC score from a single SVM model trained on 90% of split 2 and testing on 100% of split 1.

In Figure 6.8, we show the number of actives found in the top 300 molecules ranked by each model. There was very little overlap in the top ranked actives as shown in this figure. Two of the RG models with FC+SM and FC+1G had almost twice as many actives as other models. While the RG and FC kernel had an AUC of 0.7440, the SM and 1G had AUCs of 0.8221 and 0.8123. The number of actives greatly increased from 14 to 79 and 76.

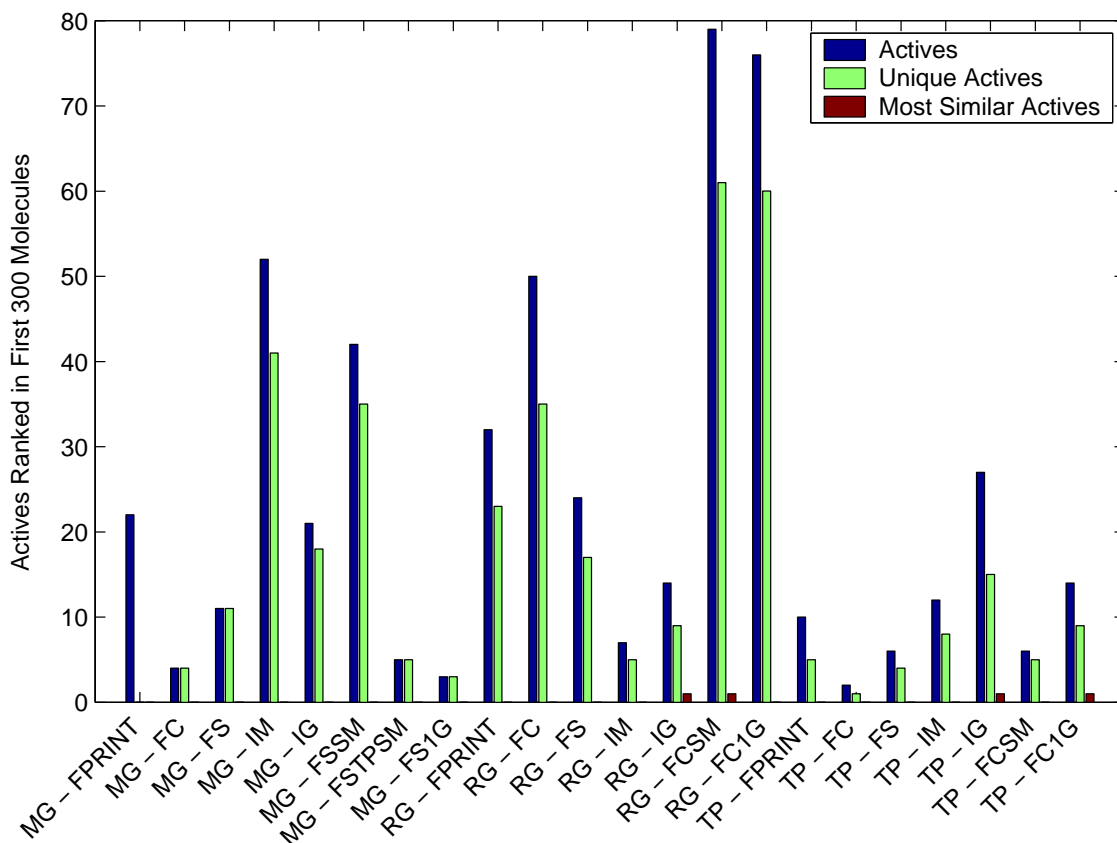


FIGURE 6.8: This graph shows the number of unique actives found in the top 300 molecules ranked by the SVM (blue). We also give the number of unique actives found compared to the MG-FPRINT model (green). Finally, we show the number of these actives which are from the top 10 most similar to the other split’s actives described in Section 6.5 (red).

In Figure 6.9, we give the top 10 actives ranked by the SVM output for the best performing graph kernel model. As shown in Table 6.8, the model with MG graphs and IM kernel gave an AUC of 0.8305. 9 of these actives look very similar visually to the top actives returned from the split 2vs1 experiment (Figure 6.7). These actives a similar core structure of two rings with a group of 3 Nitrogen atoms attached at the side. This structure appears to be useful for the HIV target that is not found from the Tanimoto similarity.

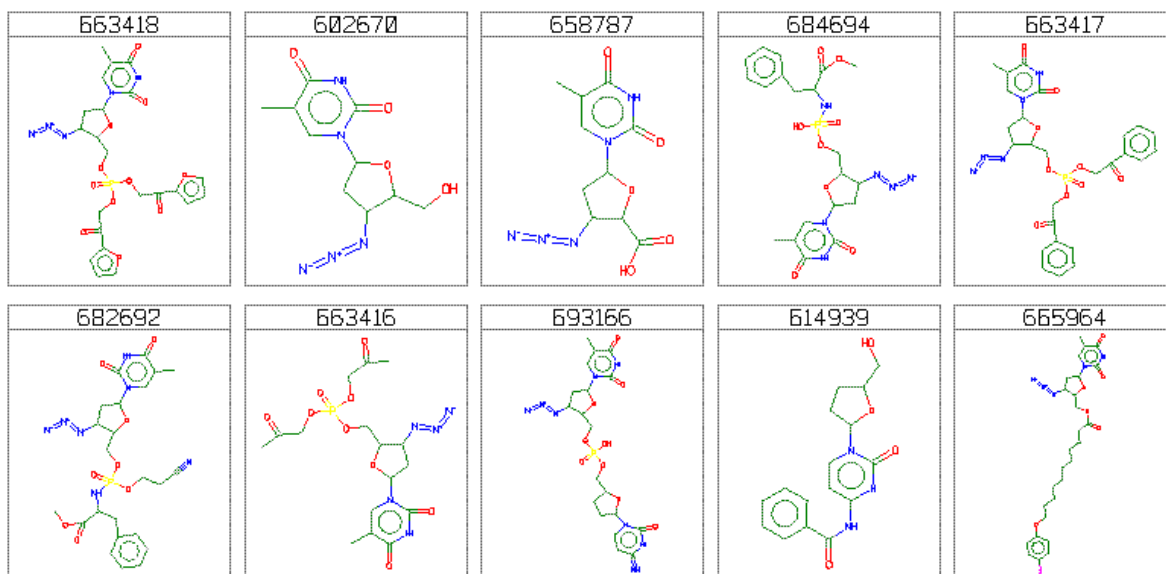


FIGURE 6.9: Highest ranked actives from MG - IM model in split 2 vs. split 1 of dataset 1. They are in order from top left to bottom right.

6.9.3 Discussion for Dataset 1

There was quite a bit of variation in the results for both Split 1 vs 2 and Split 2 vs 1. Some models found no patterns and achieved an AUC less than 0.5, although this was expected as lead hopping is a very difficult problem where much of the correlation between the training and test data is removed. One of the first things one must consider when reviewing the results is the variable chosen in the parameter tuning phase. We choose a vertex colouring type in this stage so two kernels in the full results may have different vertex labels. Although one might expect two similar kernels to achieve similar results, one must consider the parameters that each model has selected using cross-validation. For example, in Table 6.6, the MG-FC model achieved a much higher AUC than the other three kernels with MG, although this difference might be caused from MG-FC using N-Colouring while the other three used No-Colouring.

One of the most consistent patterns emerged from the soft-matching extensions. In experiments from both Split 1 vs 2 (Table 6.6) and Split 2 vs 1 (Table 6.8), we found that adding soft-matching to the FC or FS kernel increased the AUC score. Surprisingly, adding the TP soft-matching (TPSM) also resulted in a higher AUC, although it did worse than the simple SM extension. The 1G extension also increased the AUC, although the results were less consistent than the soft-matching extensions.

6.10 Dataset 2 - Pyruvate Kinase Data

In this Section, we give results for the Pyruvate Kinase dataset. We give results for training on the first split and testing on the second cluster (split 1 vs split 2) in Section 6.10.1. After, we give results for training on the second and testing on the first cluster (split 2 vs split 1) in Section 6.10.2.

6.10.1 Split 1 vs. Split 2

We followed a similar set-up to dataset 1. We first train on split 1 and classify split 2 of dataset 2.

	MG			RG			TP		
	Colour	Param	C	Colour	Param	C	Colour	Param	C
FC	N	5	10^{-3}	N	2	10^{-1}	No	12	10^{-2}
FS	M2	15	10^{-2}	N	10	10^3	No	10	10^{-2}
IM	M1	0.5	10^{-3}	N	0.9	10^{-3}	M2	0.2	10^{-1}
IG	M1	0.1	10^{-3}	M2	0.01	10^2	M2	0.3	10^{-3}
F+SM	M2	0.01	10^{-2}	N	0.01	10^{-1}	No	0.4	10^{-3}
F+TPSM	M2	0.01	10^{-2}	-	-	-	-	-	-
F+1G	M2	0.1	10^{-2}	N	0.01	10^0	No	0.3	10^3
FPRINT	-	-	10^{-2}	-	-	10^{-1}	-	-	10^{-2}

TABLE 6.9: Parameter tuning split 1 vs. split 2 for dataset 2

The results for split 1 vs. split 2 are given in Table 6.10. The best model that did not use extensions was found with the IM kernel and TP representation. Adding the SM extension improved models from all three representations (MG, RG and TP). Adding the 1G extension slightly decrease the AUC for MG, RG and TP. Some of the best models were found using the FPRINT kernel. The best model from both graph kernel, extensions and fingerprint kernels was found with the RG and FPRINT kernel with an AUC of 0.7587. This was slightly higher than the best graph kernel model with an AUC of 0.7501.

	MG	RG	TP
FC	0.6679	0.6130	0.5695
FS	0.6664	0.5251	0.6138
IM	0.6008	0.5962	0.7501
IG	0.5617	0.6626	0.6527
F	0.6664	0.6130	0.5695
F+SM	0.6783	0.6154	0.6101
F+TPSM	0.6307	-	-
F+1G	0.6627	0.6121	0.5623
FPRINT	0.6959	0.7587	0.7029

TABLE 6.10: Results for dataset 2, split 1 vs split 2. We report the AUC score from a single SVM model trained on 90% of split 1 and testing on 100% of split 2.

In Figure 6.10, we show the number of actives found in the top 300 molecules ranked by each model. We also give the number of unique actives for each model compared to those returned by MG and FPRINT kernel. We compare to this model as it is nearly identical to the measure used to separate the clusters except for a normalization factor in the Tanimoto similarity. It is interesting to note that almost all actives returned by the other models are unique to those from the MG and FPRINT kernel.

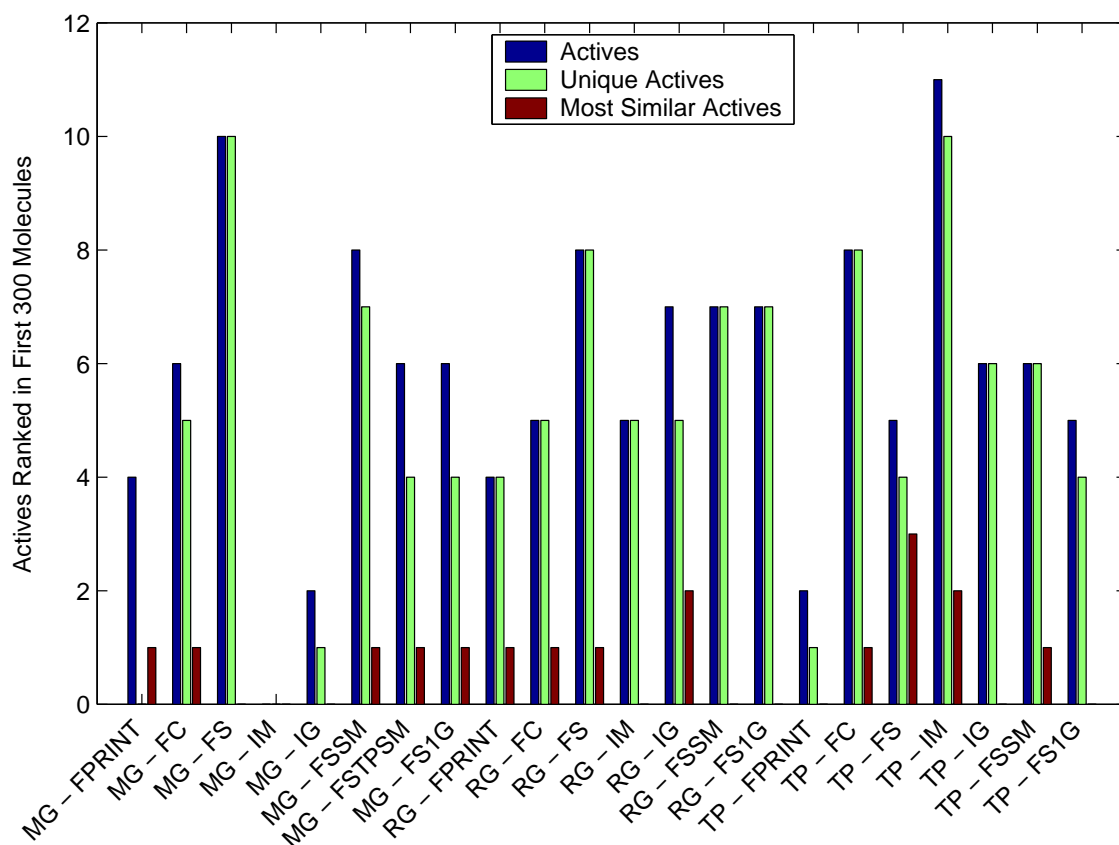


FIGURE 6.10: This graph shows the number of unique actives found in the top 300 molecules ranked by the SVM (blue). We also give the number of unique actives found compared to the MG-FPRINT model (green). Finally, we show the number of these actives which are from the top 10 most similar to the other split's actives described in Section 6.5 (red).

In Figure 6.11, we give the top 10 actives ranked by the SVM output for the best performing graph kernel model. As shown in Table 6.10, the model with TP graphs and IM kernel gave an AUC of 0.7501.

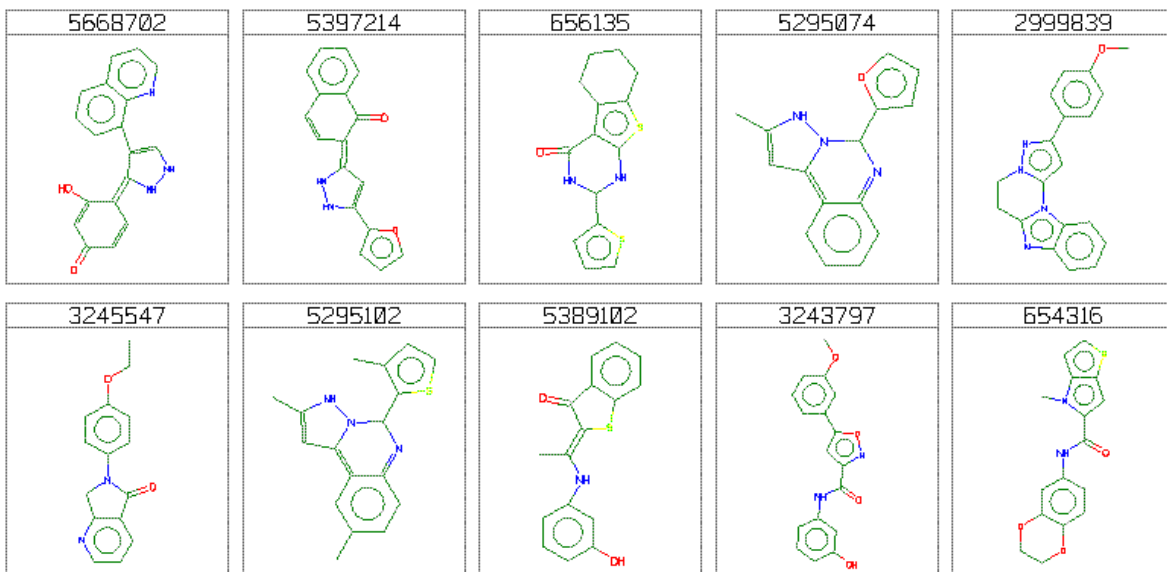


FIGURE 6.11: Highest ranked actives in 1vs2 of dataset 2. They are in order from top left to bottom right.

6.10.2 Split 2 vs. Split 1

We followed a similar set-up to dataset 1. We train on split 2 and classify split 1 of dataset 2.

	MG			RG			TP		
	Colour	Param	C	Colour	Param	C	Colour	Param	C
FC	N	12	10^3	No	1	10^2	M3	7	10^{-2}
FS	N	15	10^{-1}	No	1	10^3	M1	2	10^{-3}
IM	No	0.7	10^{-2}	No	0.01	10^{-2}	No	0.1	10^{-2}
IG	M1	0.1	10^{-2}	No	0.2	10^{-2}	M1	0.3	10^{-2}
F+SM	N	0.01	10^0	No	0.01	10^2	M1	0.01	10^{-3}
F+TPSM	N	0.01	10^{-1}	-	-	-	-	-	-
F+1G	N	0.01	10^0	No	0.01	10^2	M1	0.1	10^3
FPRINT	-	-	10^{-1}	-	-	10^0	-	-	10^{-1}

TABLE 6.11: Parameter tuning split 2 vs. split 1 for dataset 2

The results for split 2 vs. split 1 are given in Table 6.12. The best model among graph kernel models with no extensions was the FS kernel and TP graph. Only one model performed poorly, IM kernel with MG graphs. This difference is probably from the colouring choice, the other kernels with MG (FC, FS and IG) all used graphs that were coloured and this might have been necessary to find patterns that can lead hop. Adding

SM to the models varied among representation. With MG there was a slight decrease and TP a slight increase. It seems sensible that only modest changes in the AUC occurred as the largest down-weight (0.01) was chosen for both models. A large change increase Adding the 1G extension increased the AUC for both MG and TP graphs. No change occurred in AUC when adding SM and 1G extensions to the RG graphs as a walk-length of 1 was chosen, so no soft-matching could be used. The best FPRINT kernel was found using TP graphs, followed by RG and MG graphs.

	MG	RG	TP
FC	0.6538	0.6018	0.6001
FS	0.5560	0.5844	0.7434
IM	0.4475	0.6033	0.6800
IG	0.6009	0.6227	0.5380
F	0.5560	0.6018	0.7434
F+SM	0.5650	0.6018	0.7444
F+TPSM	0.6286	-	-
F+1G	0.5648	0.6018	0.7449
FPRINT	0.5727	0.6173	0.7209

TABLE 6.12: Results for dataset 2, split 2 vs split 1. We report the AUC score from a single SVM model trained on 90% of split 1 and testing on 100% of split 2.

In Figure 6.12, we show the number of actives found in the top 300 molecules ranked by each model. We also give the number of unique actives for each model compared to those returned by MG and FPRINT kernel. We compare to this model as it is nearly identical to the measure used to separate the clusters except for a normalization factor in the Tanimoto similarity. It is interesting to note that almost all actives returned by the other models are unique to those from the MG and FPRINT kernel.

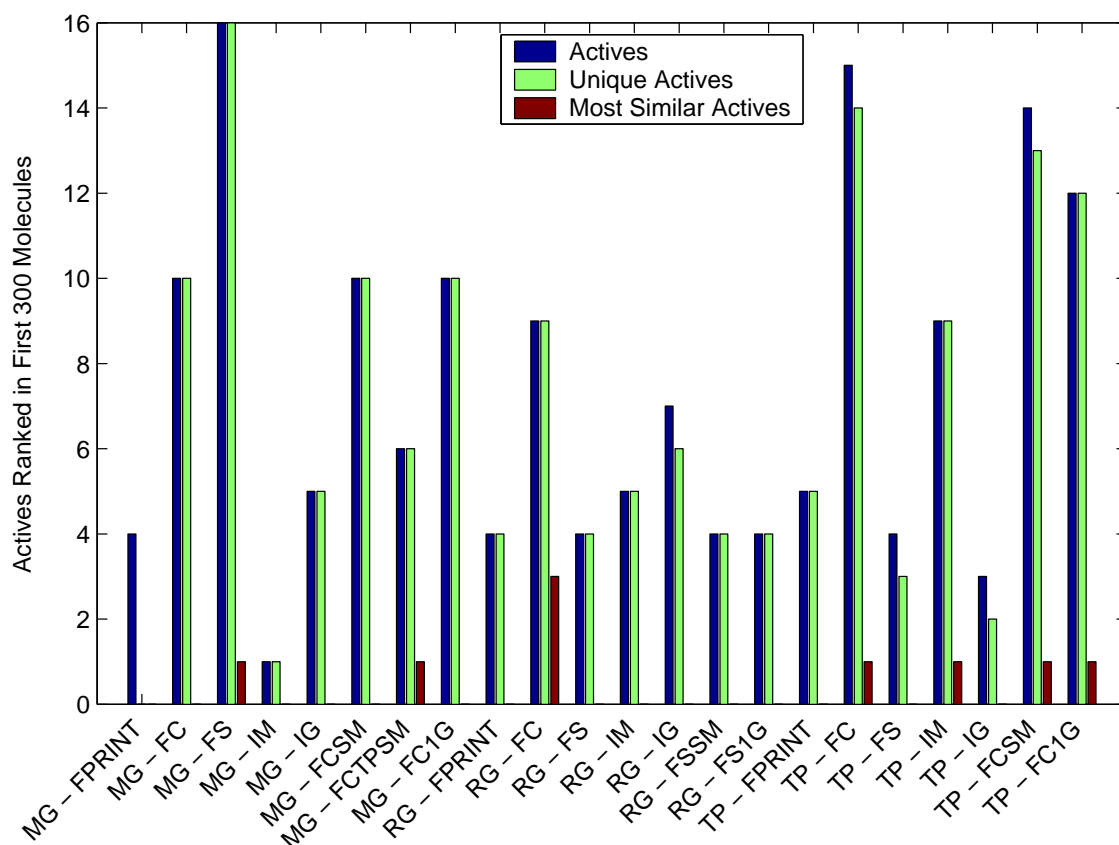


FIGURE 6.12: This graph shows the number of unique actives found in the top 300 molecules ranked by the SVM (blue). We also give the number of unique actives found compared to the MG-FPRINT model (green). Finally, we show the number of these actives which are from the top 10 most similar to the other split's actives described in Section 6.5 (red).

In Figure 6.13, we give the top 10 actives ranked by the SVM output for the best performing graph kernel model. As shown in Table 6.12, the model with TP graphs and FS kernel gave an AUC of 0.7434.

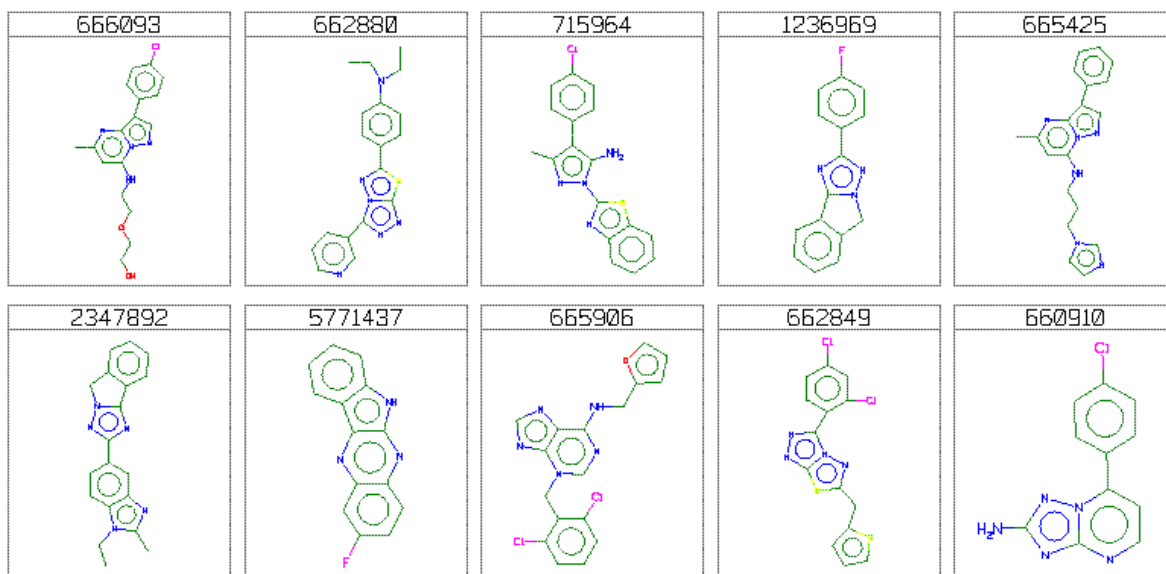


FIGURE 6.13: Highest ranked actives in TP graph with FS kernel for split 2 vs. split 1 of dataset 2. They are in order from top left to bottom right.

6.10.3 Discussion for Dataset 2

Overall, dataset 2 also contained inconsistencies in the results, although in general the results were much easier to interpret. Only a few models achieved an AUC less than 0.5, while on average most models obtained an AUC close to 0.6 which indicates that some learning was taking place.

As shown in the results for Split 1 vs 2 (Table 6.10) and Split 2 vs 1 (Table 6.12), the models that used TP graphs achieved some of the highest results. In Split 1 vs 2, the IM kernel with TP graphs achieved the highest AUC. In Split 2 vs 1, the FS kernel achieved the highest and IM kernel achieved the second highest using TP graphs.

Similar to results for Dataset 1, we found that adding SM to the FC or FS kernel consistently increased the AUC score. The 1G extension increased in some cases although the results were less consistent.

6.11 Summary

In this Chapter we analyzed a dataset that was modeled to exhibit the lead hopping which involves learning from one set of data and predicting another which has low 2D Tanimoto similarity calculated from molecular graph fingerprints. Adding soft-matching (SM) and single gap (1G) extensions increased the performance of the FC and FS kernel

in almost all of our experiments. Furthermore, the number of actives returned in the top 300 ranked by the SVM increased dramatically (dataset 2, split 2vs1) when adding soft-matching and gaps. We tested a number of models that used different graph kernels, extensions, colouring and graph-based representations. Overall, we found that it was possible to predict across splits although very different relative performance occurred based on the dataset. Most graph kernel models achieved an AUC greater than 0.5 and therefore outperforms a random classifier.

Chapter 7

Experiments with Multiple Targets

A recent approach in drug discovery uses the activity from multiple targets to find small molecules (or *ligands*). This approach, sometimes referred to as *polypharmacology*, deviates from the traditional approach of analyzing drugs with only a single target (as performed in experiments from Chapters 5 and 6). With a single target, binary classification uses a pre-determined threshold of the activity to label each example as ‘active’ or ‘not active’. In machine learning research, an example with multiple labels is referred to as a multi-class classification problem and several kernel learning machines exist to solve this type of problem. With multiple targets, we use the underlying binding activity for each target. This activity is a real value number giving the strength of interaction between a ligand and target. Although in our research we use these real values directly, it is also possible to convert these to binary (‘active’, ‘non active’) values by setting a threshold as performed with a single target.

In this Chapter, we give results using our graph kernels for two types of experiments with multiple targets. We begin, in Section 7.2, by reviewing research that uses multiple targets and give results for the first set of experiments. Using the publicly available NCI Cancer data that has activity information for 36 targets with a set of molecules, we use an SVM to predict each target individually and compare this with a kernel PLS algorithm that predicts all targets simultaneously. Next, in Section 7.3, we give results for the second set of experiments. In these experiments, we use views from both the target and ligand to perform Kernel Canonical Correlation Analysis (KCCA) which finds the best mapping from each view onto a common projection. We used a subset of the publicly available Glida database that has a set of molecules with activity information for different protein targets. Using this method, we are able to predict the closest molecules in the training data for an unseen target. Both types of experiments are a

novel approach to model ligand-target activity as it is the first research that uses graph kernels with multiple targets.

7.1 Previous Research with Multiple Targets

In this Section, we highlight some recent research that approaches drug discovery using multiple targets. A review of *polypharmacology*, where one is interested in molecules that bind with more than one target, is given in Hopkins et al. (2006). By clustering molecules from the BioPrint database¹, it was shown that similar molecules have similar binding properties for 70 pharmacological assays. They also show that there is a high correlation between the most *promiscuous* molecules, that bind with many targets, to a low molecular weight. It is noted however that designing drugs using multiple targets has certain problems. For example, two drugs that are safe on their own may be unsafe in combination (Hopkins et al. (2006)). As a result, the expensive procedure of finding an optimal dosage becomes more complex. Using the BioPrint database which contains 997 compounds and 316 targets, Paolini et al. (2006) showed that a Bayesian classification approach with molecular fingerprints could be used to predict a number of targets. In Schuffenhauer et al. (2007), clustering and descriptor-based classification methods from both reduced and molecular graphs were analyzed for a range of targets in the NCI cancer dataset. No superior method was found, although the best clustering used only a few targets. Finally, in Glick et al. (2006b), SVMs were used to predict the activity of molecules with 21 targets from the MDDR dataset. Although each target was predicted individually, it was shown that patterns exist between active molecules and targets.

7.2 Experiments in Predicting Multiple Targets

In this Section, we describe experiments which use graph kernels to predict multiple targets. We hope to understand how different graph kernels, colourings and representations perform for several tasks with multiple targets. First, in Section 7.2.1, we describe the dataset used in our experiments. After, we describe experiments used to tune parameters in Section 7.2.2. In Section 7.2.3, we predict each target individually using an SVM and graph kernels. We also test graph kernels in combination with a kernel between the remaining 35 targets (all except the held-out target). Finally, in Section 7.2.4, we use Kernel Partial Least Squares (KPLS) to predict all targets simultaneously.

¹<http://www.cerep.fr/cerep/users/pages/collaborations/bioprint.asp>

7.2.1 Dataset

In our experiments, we use a set of molecules derived from the NCI cancer dataset². This subset of examples was used for experiments in Schuffenhauer et al. (2007) and includes 7,787 molecules with activity information for all 36 targets. The original NCI cancer dataset contains 32,557 molecules, although the remaining molecules do not have activity information for every target so they cannot be used in our experiments. The activity is given by the value pGI_{50} , which is the negative log of the concentration required for 50% of a target (cancer cell) to exhibit growth inhibition. If a molecule is ‘not active’, this value is set to 4 otherwise it is greater than 4.

Using 7,787 molecules, a GSK proprietary structure normalization and duplicate removal program was used. This left 6,229 molecules for our experiments. We used 10% of these for parameter tuning (626 molecules) and the remaining 90% (5,603 molecules) for testing. We further divide each of these into 5 sets so that we can perform 5-fold cross-validation for both parameter tuning and during experimental testing.

7.2.2 Parameter Tuning

We choose parameters for our graph kernel models using only the first target. Although we run experiments with the remaining 35 targets in Section 7.2.3, we only parameter tune our models with the first target. This may produce suboptimal results, although it should give a reasonable setting.

To measure success, we compare our results using the average loss. The average loss is calculated by adding the absolute differences from each prediction and output of a model. For each model, we give the average loss from 35 targets and a 5-fold cross-validation on the parameter tuning set.

A similar experimental methodology to Chapters 4 and 6 is used in our experiments. The procedure to choose parameters for graph kernels, colourings and representations is briefly summarized here. For the MG representation, we first fix the parameters of the four basic kernels (FC, FS, IM and IG) and then choose a colouring method (N, M1, M2, M3 and No-Colour). We only test No-Colour for the RG and TP representations. Next using this colouring method, we test the FC and FS kernel with walks from length 1 to 15. We test the IM and IG graph kernel using 15 iterations of our dynamic programming equations using weights 0.01, 0.05, 0.1, 0.2, ..., 0.9. We then add soft-matching (SM) extensions, TP soft-matching (TPSM), and single gaps (1G). We add these extensions to the finite-length kernels by choosing a weight with the σ_{sm} and σ_{1g} parameters. We choose these weights from values 0.01, 0.05, 0.1, 0.2, ..., 0.5. The TPSM extension is only added to the MG representation as we only test soft-matching from atom labels

²<http://cactus.nci.nih.gov/ncidb2/download.html>

to the 6 TP labels. If the atom labels do not match, the TP labels are used and weighted according to the following: $\frac{1}{4}, \frac{1}{4}, \frac{1}{16}, \frac{1}{16}, \frac{1}{16}, \frac{1}{16}$. In addition to this weighting scheme, the σ_{sm} parameter is used to choose the amount of weight given to the TP-SM feature. For each parameter setting above we also chose the best SVM-C value from $10^3, 10^2, 10^1, 10^{-1}, 10^{-2}$ and 10^{-3} .

Colouring methods were chosen for each kernel and representation. For the MG representation, We fixed the kernel parameter for FC, FS, IM and IG kernels and then chose the colour among N, M1, M2, M3 and No-Colouring. FC and FS kernels obtained the best result with N-Colouring using this fixed setting. IM kernel found the best result with M3-Colour and IG with N-Colour. For the RG and TP representations, we only test our models with No-Colour as these seem like good choices from experiments in previous chapters.

Next, using these colouring choices, we optimized parameters for the two finite-length kernels, FC and FS. In Figure 7.1, we give the results for parameter tuning these kernels. Unlike Chapters 4 and 6 which use an AUC score, we are giving the average absolute loss from a 5-fold cross-validation, so the optimal parameter is the smallest instead of largest value.

A general trend appears for both kernels, the MG and RG kernels find an optimal parameter setting at 4 or 5, while TP chooses the longest walk. This may occur as RG are smaller graphs, and MG has structural information built into each vertex. For the FC kernel, the best parameters are 5, 5 and 15 for the MG, RG and TP representations respectively. For the FS kernel, the best parameters are 3, 4 and 15 for the MG, RG and TP representations respectively.

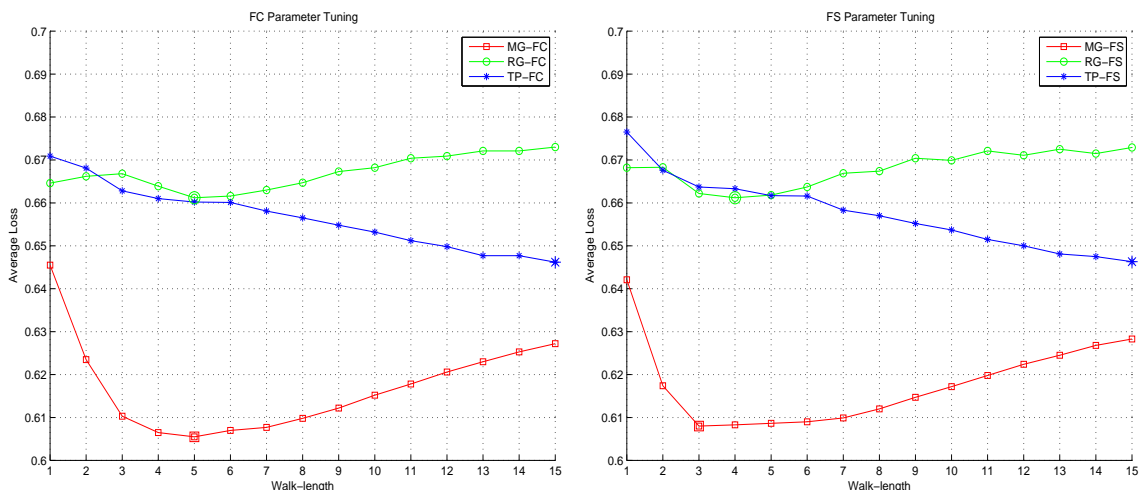


FIGURE 7.1: Parameter tuning experiments for the FC kernel are given in 7.1(a) and the FS kernel in 7.1(b). As the average loss values for each model quite similar, the scale is fixed between 0.6 and 0.7.

In Figure 7.2 we give results for optimizing parameters for the infinite-length kernels, IM and IG. In this Figure we have fixed the scale of the average absolute loss between 0.6 and 0.7. We give these graphs to show how these parameters perform over a certain range, although only the smallest of these parameters are chosen for full testing. The best parameter for the IM kernel was 0.01, the smallest parameter, for all three representations. The IG kernel similarly chose a small value, 0.05, for the RG representation which seems sensible as the reduced graphs are quite small. The MG and TP representations chose the longest value, 0.9, with the IG kernel, providing more emphasis to longer walks.

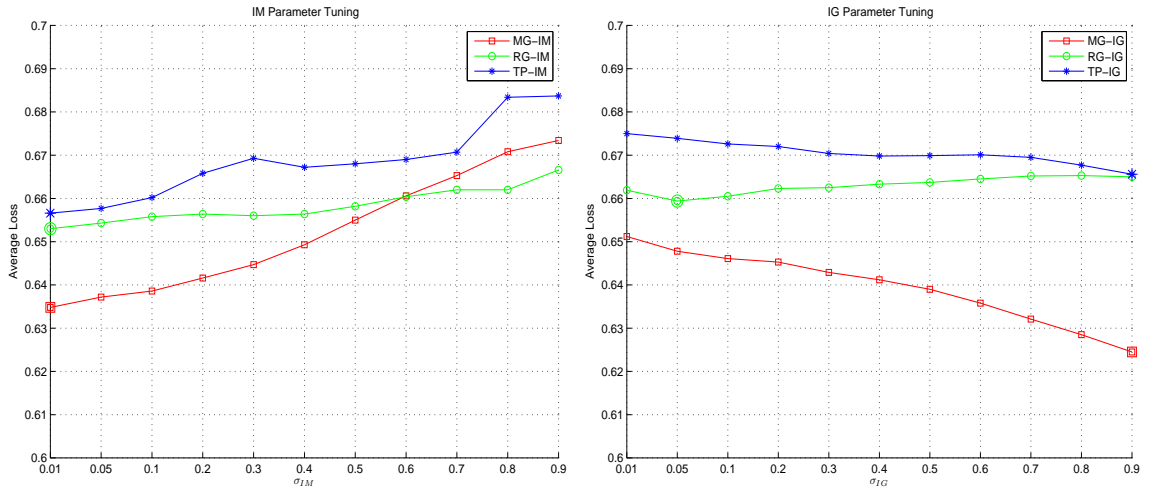


FIGURE 7.2: Parameter tuning experiments for the IM kernel are given in 7.2(a) and the IG kernel in 7.2(b). As the average loss values for each model quite similar, the scale is fixed between 0.6 and 0.7.

Finally, we give results for optimizing SM, TPSM and 1G extension parameters in Figure 7.3. We test these with the best FS kernel and graph colouring chosen in the initial parameter tuning experiments. We test a range of parameter values that weight each extension. For the MG representation, the best parameter was found with a small weight (0.01) with SM and 1G, although a large weight (0.5) was chosen with TPSM. The RG representations chose the largest parameters (0.5) for both SM and 1G, suggesting that more flexibility in these smaller graph is beneficial. The TP graphs chose the smallest parameter (0.01) to weight SM and 1G, suggesting that these are not very beneficial for these models.

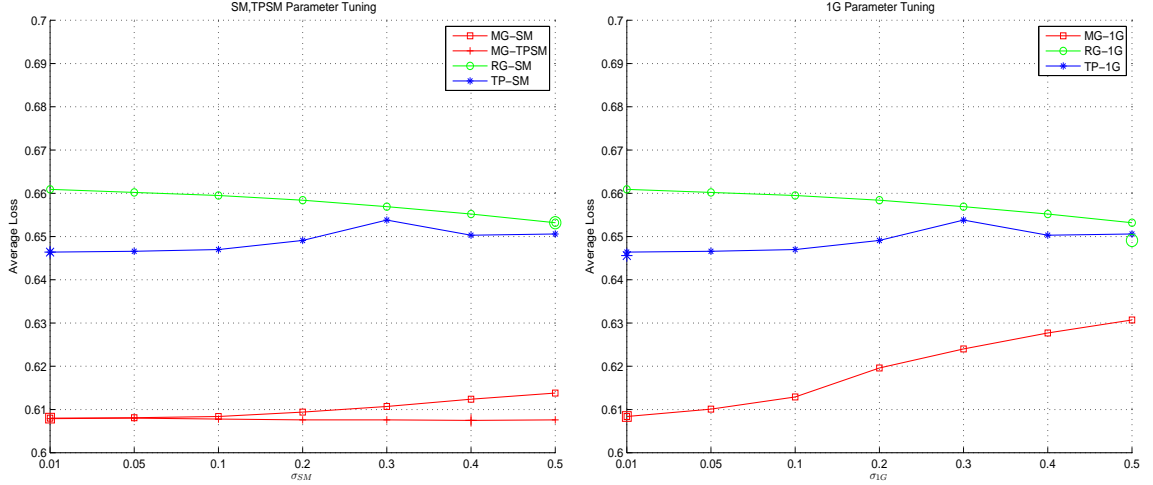


FIGURE 7.3: Parameter tuning for SM and TPSM is given in 7.3(a) and 1G is given in 7.3(b). As the average loss values for each model quite similar, the scale is fixed between 0.6 and 0.7.

7.2.3 Results for SVM Leave One Target Out

Using the parameters selected in Section 7.2.2, we ran experiments with the remaining 90% testing portion of the data. In our first experiment, we predict each of the 36 targets individually using an SVM. To measure success, we use the average loss calculated by averaging the differences between each prediction and label of a model. We give the average loss from a 5-fold cross-validation on the full test set.

Given a set of absolute loss values from a 5-fold CV model, $\mathbf{L} = \{L_1, L_2, L_3, L_4, L_5\}$, we can use a t distribution to determine if two models are significantly different. We begin by calculating the sample mean:

$$\hat{\mathbf{L}} = \frac{\sum_{i=1}^5 L_i}{5} \quad (7.1)$$

We calculate the sample variance, $\text{var}(\mathbf{L})$, using:

$$\text{var}(\mathbf{L}) = \frac{1}{5} \sum_{i=1}^5 (L_i - \hat{\mathbf{L}})^2 \quad (7.2)$$

Given average loss values from two models, \mathbf{L}_1 and \mathbf{L}_2 , we can calculate the test statistic, \hat{t} , with the following:

$$\hat{t} = \frac{\hat{\mathbf{L}}_1 - \hat{\mathbf{L}}_2}{\sqrt{\frac{\text{var}(\mathbf{L}_1) + \text{var}(\mathbf{L}_2)}{2}} \cdot \sqrt{\frac{2}{5}}} \quad (7.3)$$

Since the degrees of freedom is $2 \cdot 5 - 2$, we compare \hat{t} with $t_8(\alpha)$ and determine whether they are significantly different at a 5% level. According to Dekking et al. (2005), $t_8(\alpha)$ at a 5% level is 0.1860. If $\hat{t} > 1.860$ then we reject the hypothesis that they are the same, so they are therefore significantly different.

We will test three types of models. We refer to the first as κ_{graph} , where we only use information from graph kernels with KPLS. For this type of model we give the average loss from a 5-fold cross-validation. The second model additionally uses information from the remaining 35 targets as well as the graph kernel and we refer to it as $\kappa_{graph+target}$. For this type of model we give the average loss from 35 leave-one-target-out over a 5-fold cross-validation. Finally, the third type of model will only use target information and we refer to it as κ_{target} . We use the same measure as $\kappa_{graph+target}$ and report the average value from 35 leave-one-target, 5-fold CVs.

Results for MG, RG and TP models for the κ_{graph} models are given in Table 7.1. Almost all RG and TP models performed significantly worse than the MG kernels for each kernel, except the RG and IM kernel. Overall, the best model (lowest absolute error) was found with the MG and FS kernel. The worst model (highest absolute error) was found with the RG and FC+1G kernel. Adding SM, TPSM and 1G extensions to the FS kernel did not improve performance for any representation.

	MG	RG	TP
FC	0.5577(± 0.0178)	0.5900(± 0.0164) \circ	0.6462(± 0.0194) \circ
FS	0.5422(± 0.0168)	0.6021(± 0.0172) \circ	0.6462(± 0.0194) \circ
IM	0.6989(± 0.0178)	0.6125(± 0.0151) \bullet	0.6950(± 0.0215)
IG	0.5948(± 0.0177)	0.6886(± 0.0129) \circ	0.7019(± 0.0203) \circ
FC+SM	0.5579(± 0.0178)	0.6030(± 0.0142) \circ	0.6465(± 0.0194) \circ
FC+TPSM	0.6645(± 0.0222)	-	-
FC+1G	0.5593(± 0.0178)	1.8009(± 0.0498) \circ	0.6488(± 0.0192) \circ

TABLE 7.1: Results for κ_{graph} models. We report the average absolute loss so the best model has the lowest value. The standard deviation reported for the κ_{graph} model is from the 5-fold cross-validation. We use the notation \circ to identify if a significant loss (at 5%) over the same kernel and MG representation. Similarly, we identify a significant gain (at 5%) by \bullet .

The results for the $\kappa_{graph+target}$ and κ_{target} models are given in Table 7.2. It appears that using the target information consistently performs better than without (κ_{target}). Every RG and TP model performs significantly worse than the same kernel with MG representation. Overall, the best model (lowest absolute error) was found with MG and FS kernel. The MG and FS kernel also was the best model found using the κ_{graph} in the previous experiments. The worst model was found adding gaps to the RG and FC+1G kernel. This model used the largest value to weight the gap feature, so it may have overtrained to the parameter tuning set. Adding SM, TPSM and 1G extensions to the FS kernel did not improve performance for any representation.

	MG	RG	TP
FC	0.5087(± 0.0063)	0.5285(± 0.0061) \circ	0.5606(± 0.0066) \circ
FS	0.4983(± 0.0065)	0.5320(± 0.0060) \circ	0.5606(± 0.0064) \circ
IM	0.5294(± 0.0064)	0.5428(± 0.0062) \circ	0.5800(± 0.0072) \circ
IG	0.5303(± 0.0098)	0.5787(± 0.0068) \circ	0.5833(± 0.0073) \circ
FC+SM	0.5088(± 0.0061)	0.5357(± 0.0057) \circ	0.5608(± 0.0065) \circ
FC+TPSM	0.5235(± 0.0066)	-	-
FC+1G	0.5097(± 0.0063)	1.0199(± 0.0027) \circ	0.5618(± 0.0065) \circ
κ_{target}	0.5422(± 0.0325)		

TABLE 7.2: Results for $\kappa_{graph+target}$ and κ_{target} models. We report the average absolute loss so the best model has the lowest value. The standard deviation reported for the $\kappa_{graph+target}$ model is obtained from the average of the 36 leave-one-target-out tests. We use the notation \circ to identify if a significant loss (at 5%) over the same kernel and MG representation. Similarly, we identify a significant gain (at 5%) by \bullet .

Surprisingly, using κ_{target} alone with no information from the structure performed as well as the best κ_{graph} model (MG-FS). Although overall, combining these two sources of information with $\kappa_{graph+target}$ gave the best model as shown by the lowest absolute loss (0.4983).

7.2.4 Results for KPLS

Using the parameters selected in Section 7.2.2, we ran experiments with the remaining 90% testing portion of the data for another experiment to predict multiple targets. In this experiment, we used Kernel Partial Least Squares (KPLS) (Rosipal and Trejo (2001)). KPLS is an iterative algorithm that constructs a linear PLS algorithm in a kernel space. We used 5 components of the KPLS in our tests. We list results for the MG, RG and TP graphs in Table 7.3. We calculate the significance at 5% using the t distribution as discussed in the previous section.

	MG	RG	TP
FC	0.1765(± 0.0495)	0.1571(± 0.0311)	0.1318(± 0.0436)
FS	0.1826(± 0.0480)	0.2807(± 0.0195) \circ	0.1335(± 0.0429)
IM	0.0766(± 0.0359)	0.1607(± 0.0319) \circ	0.0275(± 0.0192) \bullet
IG	0.0801(± 0.0381)	0.0875(± 0.0358)	0.0339(± 0.0242) \bullet
F	0.1765(± 0.0495)	0.2807(± 0.0195) \circ	0.1335(± 0.0429)
FS+SM	0.1748(± 0.0492)	0.1008(± 0.0260) \bullet	0.1118(± 0.0382) \bullet
FS+TPSM	0.1066(± 0.0400)	-	-
FS+1G	0.1711(± 0.0485)	0.1327(± 0.0315)	0.1115(± 0.0384) \bullet

TABLE 7.3: KPLS results for MG, RG and TP and several kernels. We report the average loss as well as the standard deviation from a 5-fold CV. The best model has the lowest value as we are considering the average absolute loss. We use the notation \circ to identify if a significant loss (at 5%) over the same kernel and MG representation.

Similarly, we identify a significant gain (at 5%) by \bullet .

The best result was found with the TP and IM kernel, followed by the TP and IG kernel. The SM extension helped improve results for each representation. It is interesting to note that the FS and TPSM extension dramatically improved the MG representation, decreasing the error from 0.1765 to 0.1066. The SM and 1G extensions also improved the model although to a much smaller degree. Every TP model except the FC and FS kernels achieved significantly better results than the same kernel with the MG representation.

Comparing the results from KPLS (Table 7.3), to those that used SVMs (Tables 7.1 and 7.2) it is clear that the KPLS provides a much better model. The KPLS models had an average loss around 0.1 while the SVM models had average loss much higher around 0.5. In fact, the best KPLS model that used IM and TP achieved a very low average loss of 0.0275. Clearly, it is much more beneficial to use KPLS than an SVM for this data.

7.3 Experiments in Predicting an Unseen Target

In this Section, we introduce a new approach to drug discovery that allows one to predict the activity of molecules for a target that was unseen in the training set. We utilize Kernel Canonical Correlation (KCCA) which develops a model that finds the best correlation between two views of data. KCCA is a natural model for drug discovery where a molecule-target binding can be viewed from features that describe either the molecule or target. KCCA also requires a kernel function for each view and specifically tailored kernel functions are available for each view. We use graph kernels to represent the view of the molecule and a string kernel for the view of a protein target. By mapping an unseen target onto a KCCA mapping, we try to find the best molecules in the training set that bind with this target.

This Section is outlined as follows. We begin by reviewing KCCA and introducing a method to apply it to drug discovery in Section 7.3.1. We describe our dataset in Section 7.3.2 and experimental methodology in Section 7.3.3. Using a subset of the data, we discuss parameters for KCCA models and choose parameters for graph kernels using a small parameter tuning set in Section 7.3.4. Finally, we give results and analysis in Section 7.3.5.

7.3.1 KCCA for Drug Discovery

If a dataset has two representations (or views) for each example, we can use Kernel Canonical Correlation Analysis (KCCA) to find the correlation between these views. KCCA has been applied to several problems including classification experiments where a document has both image and text representations (Hardoon et al. (2003)) and in cross-language experiments where an example has text from two different languages (Vinokourov et al. (2002); Li and Shawe-Taylor (2005)). Several reviews of KCCA are available such as Hardoon et al. (2003) and as a part of books on kernel methods (Shawe-Taylor and Cristianini (2004)).

Given a dataset with views a and b , we define the corresponding kernel matrices, K_a and K_b . We define ϵ , as a parameter to allow control over the regularization in KCCA. The correlation coefficient, ρ , between two kernel matrices is calculated with KCCA using:

$$\rho = \max_{\alpha, \beta} \frac{\alpha' K_a K_b \beta}{\sqrt{(\alpha' K_a^2 \alpha + \epsilon \alpha' K_a^2 \alpha) \cdot (\beta' K_b^2 \beta + \epsilon \beta' K_b^2 \beta)}} \quad (7.4)$$

The corresponding optimization problem is:

$$\max_{\alpha, \beta} \alpha' K_a K_b \beta \quad (7.5)$$

$$\text{subject to } (\alpha' K_a^2 \alpha + \epsilon \alpha' K_a^2 \alpha) = 1 \quad (7.6)$$

$$(\beta' K_b^2 \beta + \epsilon \beta' K_b^2 \beta) = 1 \quad (7.7)$$

In drug discovery, two views exist of the binding event between the ligand and target. Kernel functions have been developed for each of these views. Graph kernels have been specifically designed for molecules in Chapter 4. String kernels have previously been tested with protein targets (Leslie et al. (2003)). We will only test the simplest string kernel known as the n-gram kernel which do not allow mismatches (Leslie et al. (2003)) or gaps (Leslie and Kuang (2004)). A string kernel is used to model the protein target as it is composed as a sequence of characters. The string kernel is computed in an efficient manner using suffix arrays (Teo and Vishwanathan (2006)). Although each

view has been tested independently, we propose combining these two views to model the ligand-target binding using KCCA.

To model this relationship, it is important to note that we are modeling the binding relationship which is composed of one molecule and one target. Each molecule may bind with more than one target and conversely each target may bind with more than one molecule. Therefore, each molecule or target may be represented more than once when building a dataset to represent every binding relationship.

7.3.2 Dataset

We used a division of the Glida database³ from Jacob et al. (2008), which removed molecules with properties not suitable for drugs as well as any duplicates. From the original database consisting of 22,964 ligands for 3,738 GPCR targets, this left 2,446 molecules and 80 targets. These correspond to 4,051 binding interactions between molecules and targets.

In Figure 7.4, we show in decreasing order the number of targets that bind with each molecule. Out of 2,446 molecules, only 723 bind with more than 2 targets.

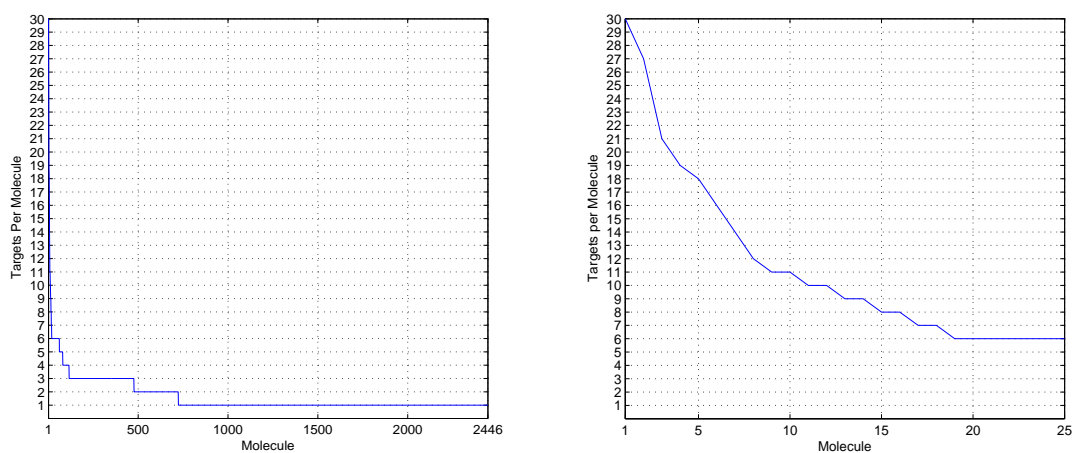


FIGURE 7.4: The number of targets per molecule are shown in 7.4(a). In 7.4(b), we show the first 25 molecules in greater detail.

In Figure 7.5, we show the number of molecules that bind with each target. There are 66 targets that bind with at least two molecules. The first target, ACM1, binds with over 526 molecules although this number decreases with the remaining targets.

³<http://pharminfo.pharm.kyoto-u.ac.jp/services/glida/>

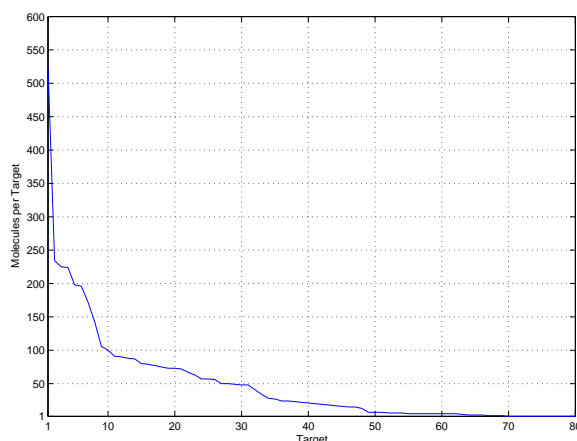


FIGURE 7.5: The number of molecules that bind with each target in our dataset.

7.3.3 Experimental Methodology

To experiment with KCCA, we will remove the bindings associated with one target from the data. Next, we will map this target on to the KCCA projection and determine if the closest molecules in the remaining data contain molecules that bind with the target. In order to measure success, we require that every molecule in our data binds with at least two targets. Therefore, when we remove one target (and its binding molecules) there will be ‘correct’ responses which are molecules that bind with this target in the remaining data. We can rank the molecules in the training data that are closest to this held-out target. The best model will list molecules that bind with this target first in this list. These ‘correct’ molecules are found in the bindings from the held-out target. We can calculate an AUC score from this ranked list, and we use this to compare models. The results are measured with the average AUC from a set of leave-one-target-out experiments.

To model this problem, we first sort the molecules by counting the number of targets that bind with each molecule. We select the top 100 molecules that bind with the most targets to use for the ‘full’ set. For a parameter tuning set, we select the next 10 molecules (molecules 100 to 110) that have the most target bindings. The first 110 molecules selected have bindings with at least two targets so they meet the requirement mentioned earlier to model this problem. This results in a parameter tuning set that consists of 10 unique molecules and 28 targets to form 40 molecule-target bindings. The full set consists of 100 unique molecules that bind with 48 targets to form 680 bindings. We then build kernel matrices for each view of the binding.

The string kernel matrices are calculated using the publicly-available software⁴ based on Teo and Vishwanathan (2006). Similar to experiments in previous chapters, we use

⁴<http://users.rsise.anu.edu.au/~chteo/SASK.html>

our graph kernels as well as several graph colouring and representations. We review this methodology in Section 7.3.4, where we run experiments to select parameters for our graph kernel models.

We use the publicly-available KCCA matlab software ⁵ to model KCCA. It is straightforward to modify this code so that it returns a list of the closest examples in view a given an example in view b .

7.3.4 Parameter Tuning

In this Section, we review the parameters of KCCA models. Next, using a small parameter tuning set we experimentally select parameters for the graph kernel models.

7.3.4.1 KCCA Parameters

There are three parameters that define a KCCA model. The ϵ parameter controls the regularization in the KCCA. We choose this parameter for each experiment using the method suggested in Hardoon et al. (2003). This is calculated experimentally by choosing a range of ϵ values. The best value is selected by choosing the one which has the largest difference in eigenvalues returned by the KCCA algorithm and those returned when computed with randomized kernel matrices. By choosing the maximum difference, this gives a model with correlations furthest from the randomized version. We use $\eta = 0.5$ for the Gram-Schmidt Decomposition (GSD) precision. This is a heuristic value suggested in Hardoon et al. (2003). Finally, we must decide how many eigenvectors used for the GSD algorithm which we refer as *nfactors*. Using the parameter tuning set we found that setting *nfactors* to 5 gave the best results which agrees with Hardoon et al. (2003).

7.3.4.2 Parameter Tuning Graph Kernels

Although we only use one string kernel for the view of the target, we use a number of different graph kernels, colourings and representations for the ligand view. The best setting using the parameter tuning set and selecting the parameter with the highest average AUC from a series of leave-one-target-out trials.

A similar experimental methodology to Chapters 5 and 6 is used in our experiments. We used the same method to parameter tune graph kernels and colours as described in earlier experiments (Section 7.2), although we chose to test colouring methods with RG and TP representations to allow more flexibility. We give the colour and graph kernel parameter settings chosen in Table 7.4.

⁵<http://www.davidroihardoon.com/>

	MG		RG		TP	
	Colour	Param	Colour	Param	Colour	Param
FC	M1	10	M1	5	M1	1
FS	M1	4	N	11	M1	5
IM	N	0.1	N	0.2	M3	0.1
IG	M2	0.1	N	0.01	No	0.1
F+SM	M1	0.01	M1	0.05	M1	0.3
F+TPSM	M1	0.05	-	-	-	-
F+1G	M1	0.01	M1	0.01	M1	0.01

TABLE 7.4: Parameter tuning graph kernel models for KCCA experiments

7.3.5 Results

Using the ‘full’ set that includes 100 unique molecules and 48 targets to give 680 bindings, we performed experiments using the parameters chosen in Section 7.3.4. The results are given in Table 7.5. The value reported is an average AUC score from series of 48 leave-one-target-out experiments. An AUC score greater than 0.5 shows that patterns are discovered from the data on average for all targets. The FS, IM and IG kernel have values greater than 0.5 for MG. The FC kernel had an average AUC less than 0.5 on average. This kernel used a very long walk-length (length 10) that may be too specific. None of the RG models appear to learn patterns from the data. Only the IM kernel for the TP representation did not achieve an AUC greater than 0.5. This kernel used M3-Colour which incorporates the greatest amount of structural information into each node. The other kernels FC, FS and IG for the TP representation used M1 or No-Colour which incorporate much less information into each node. This suggests that the original labeling of the graph may produce the best results. Only the TP representation was improved by adding SM and 1G extensions. The SM extension increased the AUC from 0.5159 to 0.5604.

	MG	RG	TP
FC	0.4280	0.4958	0.5192
FS	0.5632	0.4844	0.5159
IM	0.5524	0.4955	0.4705
IG	0.5883	0.4848	0.5208
F	0.5632	0.4958	0.5159
F+SM	0.5365	0.4820	0.5604
F+TPSM	0.4999	-	-
F+1G	0.5006	0.4512	0.5416

TABLE 7.5: KCCA Results. These values are AUC scores averaged from a 5-fold CV for the kernel and representation. The three results in bold will be analyzed further.

Although we have given an average AUC for leave-one-target-out experiments, we give analysis to understand how each of the 48 targets perform individually. Using the top three models highlighted in Table 7.5, we analyze the results for each target. The top three models highlighted are MG with the IG kernel, followed by MG with FS kernel and TP with FS kernel and SM extension.

In Figure 7.6, we sort the targets based on the number of molecules that bind with each target. We report the number of hits found in the first 100 molecules in the ranked list of molecules closest in the training data to the target. In general, it appears that most targets are able to retrieve a good proportion of the actives in the first 100 molecules. As shown for the last 10 targets, almost all of the targets are found in the first 100.

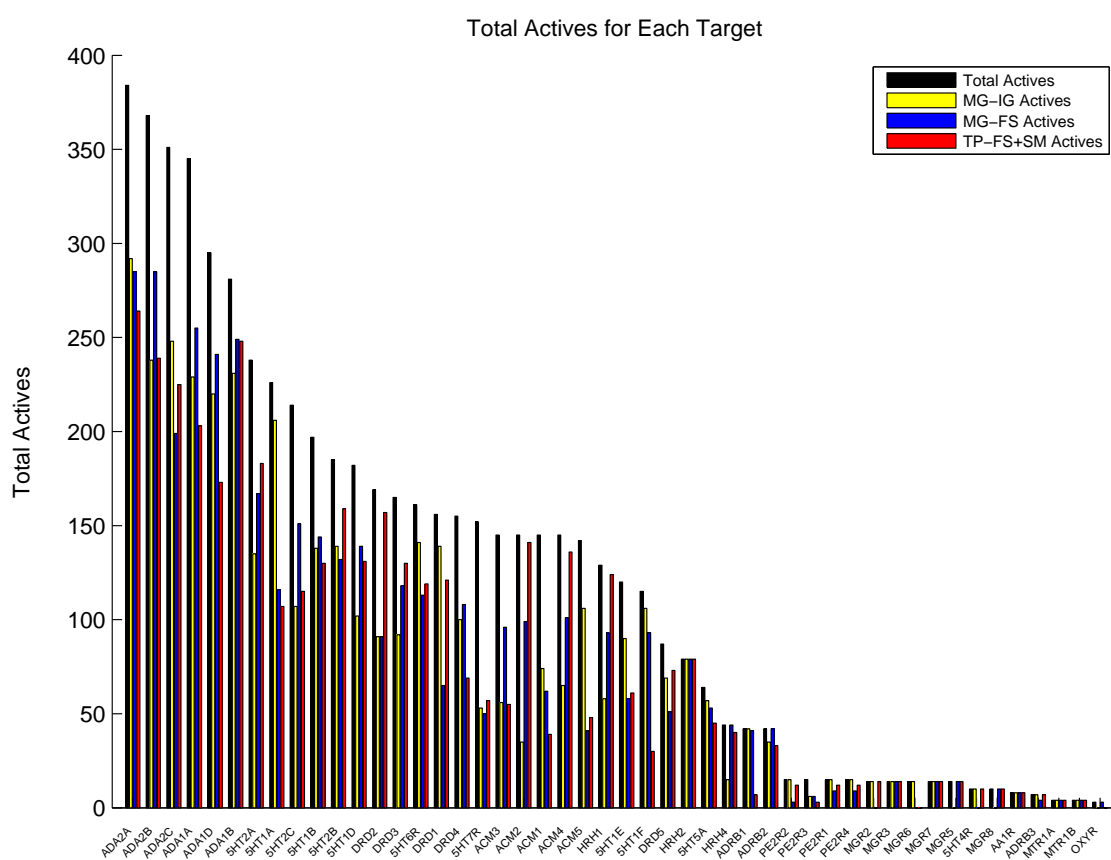


FIGURE 7.6: The number of actives retrieved for the top 100 molecules ranked by output of three KCCA models with highest average AUC

In Figure 7.7, we give the AUC found for each target. The targets are ordered in the same manner as Figure 7.6 so that these figures can be compared. In the last 10-15 targets, it appears that many models achieve a high AUC close to 0.9. The MG-IG model (coloured yellow) has many targets with AUC close to 1.

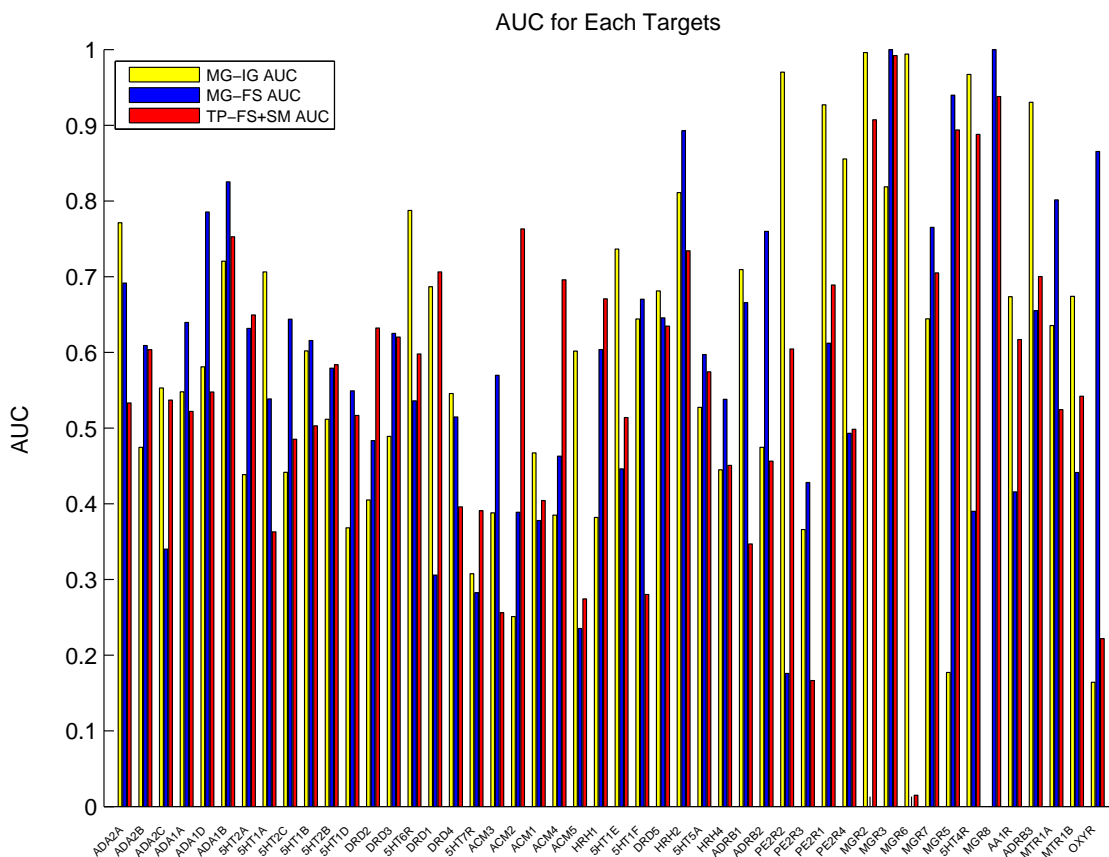


FIGURE 7.7: The AUC achieved by each individual held-out target for three KCCA models with highest average AUC

7.3.6 Discussion

A new approach to virtual screening was presented in this Section. This is the first known method that combines information from both ligand and target to predict a molecule's activity with a target. Given an unseen target, we can find the best molecules in the training data that may interact with this target. We found that experimentally our models could learn from this data and achieve an AUC greater than 0.5. This is quite surprising as it is a very difficult problem; the relationship between the structure of the target and the structure of the ligand is very complex. Overall, using MG we achieved the highest results with the IG kernel obtaining an average AUC of 0.5883.

This approach may be very useful in the drug discovery cycle when training data is not available for a certain target. One very recent paper (Holden et al. (2008)), gave the structure of an APOBEC3G target that is believed to be useful in restricting the replication of the immunodeficiency virus (HIV). Although we use NCI-HIV data in earlier Chapters, only 13 inactive molecules overlapped with the data used in this Section. Therefore, although we are able to predict the most suitable molecules for this target in our training data, we cannot validate these because of a lack of data.

7.4 Summary

In this Chapter we have given new results that use graph kernels with multiple targets in two sets of experiments. In the first set of experiments, we predict each target individually and then compared this with kernel PLS that predicted all targets simultaneously. We found trends among each graph kernel and representation. Our finite length kernels performed better than infinite-length kernels for a leave-one-target SVM experiment. Also, using the remaining target information (κ_{graph}) consistently achieved better models than ones that did not use this information. In the second set of experiments we used KCCA to predict molecules for a target that was not used in the training data. This method is a new approach to drug discovery, and we found that several settings of graph kernels are able to accurately predict the activity of molecules for an unseen target confirmed by analysis of the top 100 ranked molecules and the AUC score.

Chapter 8

Conclusions

8.1 Conclusions

This thesis suggests that graph kernels are a powerful tool for virtual screening of molecules. Through the course of this research we have identified three decisions that one must address to use graph kernel models. Firstly, one must choose a graph-based representation of a molecule. Although previous research has only tested molecular graph, in this thesis we introduce two new graphs for testing with graph kernels: the reduced graph and topological pharmacophore (TP) graph. These are described in detail in Chapter 3. As every molecule in our dataset can be described as any of the three graphs, we test all three graphs and make direct comparisons when used to classify their activity and perform other experiments. A second decision is whether to modify the vertex labels of the graphs by enriching each label with structural information. We describe how this graph colouring is achieved with either N-Colour or Morgan colour methods in Chapter 2. A third decision involves choosing the type of graph kernel. As described in Chapter 3, there are a number of different graph kernels available.

We addressed some of the problems with previous kernels by introducing a new flexible framework to calculate graph kernels efficiently, in Chapter 4. As a part of this framework, we show that two infinite-length graph kernels can be approximated. Also, we introduce two new finite-length graph kernels, that use counts of walk-lengths of a set length as features. Finally, we introduce extensions to graph kernels including soft-matching and gaps. The primary motivation for soft-matching is the availability of alternative TP labels for molecular graphs. Using our kernels we can include mismatched walks that have matching TP labels. We are motivated to allow gap features in graphs primarily for use with reduced graphs. Reduced graphs are very small, typically containing only 5 nodes. Important features may be separated by internal nodes, although no matching would occur. By allowing gaps we can include features and match these reduced graphs which seems appropriate. This framework allows much more flexibility

so that one can incorporate expert knowledge and weight certain features to design the best kernel. All three of these considerations were addressed in three different Chapters of experiments.

Our first experiments, in Chapter 5, used graphs kernels for ‘active’, ‘not active’ binary classification of two publicly-available datasets: MuTag and NCI-HIV datasets. Using an AUC score, we compared several graph kernels and extensions using MG, RG and TP representations. In both datasets, we found that MG graphs achieved slightly higher AUC scores. We consider the second, larger dataset, a more reliable source to draw conclusions as it is much larger. In this second dataset, we found that the IM graph kernel achieved the highest AUC for every representation. Adding the soft-matching and gap extensions to the finite-length graph kernels improved the AUC score. By examining the statistical significance of the results for each kernel, we found that a given kernel performed similarly using either MG, RG or TP. Finally, as the NCI-HIV dataset is highly skewed containing over 96% examples from the ‘not active’ class which is typical of most chemoinformatic datasets, we tried experiments that varied the number of ‘not active’ examples used for training. We found that good models could be built with a fraction of the total ‘not actives’ and thus not all of these are needed to build a good classification model with SVMs.

While previous research has used cross-validation techniques using the entire dataset, in Chapter 6 we modify our data to exhibit lead hopping so that we can understand how well these models generalize to other classes of drugs. We used two datasets derived from publicly-available dataset including the NCI-HIV data and the Pyruvate Kinase data. In these experiments, we went beyond just the AUC score and performed a visual analysis of the top actives from each model to determine if lead hopping occurred. We did this as the definition of lead hopping is fairly subjective and although in general two molecules must have different core structure to be considered lead hopping. Many of the kernels and representations were able to achieve an AUC greater than 0.5 suggesting lead hopping. We found that with certain graph kernels and colourings chosen in the parameter tuning phase, could result in very poor performance on the full dataset. Overall, no one kernel or representation emerged as the best choice. One consistent trend that occurred for both datasets while predicting either way on the splits was that adding the SM extension to either the FC or FS kernel increased the models performance. The 1G extension also increased performance, although this was much less consistent than the SM extension. Surprisingly, we found that models could learn with MG fingerprints even though most of the information was removed by the nature of the data split.

Finally, in Chapter 7, we introduce two sets of experiments for learning with multiple targets. This type of problem, sometimes referred to as *polypharmacology*, differs from experiments in earlier chapters where we predicted the activity of a single target. To predict multiple targets, we tried two approaches. First, we predicted each target individually with an SVM and used the remaining activity information with a

linear kernel. Next, we predicted all targets simultaneously with kernel PLS. We found that using KPLS we could achieve much better results than using a leave-one-target-out SVM. Models that used an SVM performed best when using a kernel that took information from both the target and molecular graph. Finally, we performed a second set of experiments which introduced a new method to integrate information from both ligand and target to predict the activity of a target. Using Kernel Canonical Correlation Analysis (KCCA), we found that some kernels and representations could learn from the data achieving an average AUC greater than 0.5 from a set of leave-one-target-out experiments.

8.2 Future Work

We see several areas where our work could be extended. In Chapter 7, we introduced a new framework using KCCA to perform drug discovery. In our experiments we only used one type of string kernel for analysis. One future direction would be to run experiments with string kernels that allow mismatches and gaps (sometimes referred to as the n-gram kernel). Another alternative would be to use the structural information in the protein class hierarchy, which was used to define a kernel between targets in Jacob et al. (2008). While in Chapter 7 we optimized our model for the best view of the molecule, we could use one of these alternative kernel for targets to optimize the view of the target as well.

Another area for future work is motivated by an interesting result from Chapter 5. We found that using only a fraction of the ‘not active’ examples, we could achieve nearly as good AUC scores as models that used all the examples. This result is especially important for large chemical databases that include millions of compounds and only hundreds of hits or active compounds. As the active class information is so critical, we would like to test one-class classifiers such as the 1-SVM (Schölkopf et al. (1999)) with our framework. Using one-class this is a type of novelty detection and it appears that drug discovery fits nicely with this interpretation, as we are only interested in the hits and not the inactive examples.

Finally, the dynamic programming framework to calculate graph kernels introduced in Chapter 4, could be applied to other problems beyond drug discovery. One of the key aspects of our kernels is the ability to incorporate prior knowledge using soft-matching and gappy features. We could test our new graph kernels on a bioinformatics tasks such as the classification of proteins where previous graph kernel research has been effective (Borgwardt and Kriegel (2005)). Graph kernels may be useful for other domains where the data can be modeled using a graph. Other possible areas where graph kernels may prove useful include spam detection and social network analysis. Our kernels could be tailored to address each of these areas.

Bibliography

- G. Apic, T. Ignjatovic, S. Boyer, and R. B. Russell. Illuminating drug discovery with biological pathways. *FEBS Letters*, 579(8):1872–1877, 2005.
- C.-A. Azencott, A. Ksikes, S. J. Swamidass, J. H. Chen, L. Ralaivola, and P. Baldi. One- to four-dimensional kernels for virtual screening and the prediction of physical, chemical and biological properties. *J. Chem. Inf. Model.*, 47(3):965–974, 2007.
- E. J. Barker, D. Buttar, D. A. Cosgrove, E. J. Gardiner, P. Kitts, P. Willett, and V. J. Gillet. Scaffold hopping using clique detection applied to reduced graphs. *J. Chem. Inf. Model*, 46:503–511, 2006.
- K. Birchall, V. J. Gillet, G. Harper, and S. D. Pickett. Training similarity measures for specific activities: Applications to reduced graphs. *J. Chem. Inf. Model.*, 46(2):577–586, 2006.
- K. Birchall, V. J. Gillet, G. Harper, and S. D. Pickett. Evolving interpretable structure-activity relationship models. 2. using multiobjective optimization to derive multiple models. *J. Chem. Inf. Model.*, 48(8):1558–1570, 2008a.
- K. Birchall, V. J. Gillet, G. Harper, and S. D. Pickett. Evolving interpretable structure-activity relationships. 1. reduced graph queries. *J. Chem. Inf. Model.*, 48(8):1543–1557, 2008b.
- K. H. Bleicher, H.-J. Böhm, K. Müller, and A. I. Alanine. Hit and lead generation: beyond high-throughput screening. *Nat Rev Drug Discov*, 2(5):369–378, May 2003.
- H. J. Böhm, A. Flohr, and M. Stahl. Scaffold hopping. *Drug Discovery Today: Technologies*, 1(3):217–223, 2004.
- H. J. Böhm and G. Schneider. *The biophore concept*, chapter 4, pages 73–100. John Wiley and Sons, 2003.
- A. Boobis, U. Gundert-Remy, P. Macheras, and O. Pelkonen. In silico prediction of adme and pharmacokinetics. *Eur J Pharm Sci*, 17(4-5):183–193, 2002.
- K. M. Borgwardt and H. Kriegel. Shortest-path kernels on graphs. In *International Conference on Data Mining*, 2005.

- B. E. Boser, I. M. Guyon, and V. N. Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the 5th Annual ACM Workshop on Computational Learning Theory*, pages 144–152. ACM Press, 1992.
- R. D. Brown and Y. C. Martin. The information content of 2d and 3d structural descriptors relevant to ligand-receptor binding. *J. Chem. Inf. Comput. Sci.*, 37(1): 1–9, 1997.
- E. Byvatov, U. Fechner, J. Sadowski, and G. Schneider. Comparison of support vector machine and artificial neural network systems for drug/nondrug classification. *J. Chem. Inf. Comput. Sci.*, 43(6):1882–1889, 2003.
- E. Byvatov and G. Schneider. Svm-based feature selection for characterization of focused compound collections. *J. Chem. Inf. Comput. Sci.*, 44:993–999, 2004.
- A. Ceroni, F. Costa, and P. Frasconi. Classification of small molecules by two- and three-dimensional decomposition kernels. *Bioinformatics*, 23(16):2038–2045, 2007.
- M. Collins and N. Duffy. Convolution kernels for natural language. *Advances in Neural Information Processing Systems*, volume 14, 2002.
- Corinna Cortes, Patrick Haffner, and Mehryar Mohri. Rational kernels: Theory and algorithms. *Journal of Machine Learning Research*, 5:1035–1062, 2004. ISSN 1533-7928.
- N. Cristianini and J. Shawe-Taylor. *An introduction to Support Vector Machines*. Cambridge University Press, 2000.
- A. K. Debnath, R. L. L. Debnath, G. Schusterman, and C. Hansch. Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro molecules, correlation with molecular orbital energies and hydrophobicity. *Journal of Medicinal Chemistry*, pages 786–797, 1991.
- F. M. Dekking, C. Kraaikamp, and H. P. Lopuhaä. *A Modern Introduction to Probability and Statistics*. Springer, 2005.
- A. Demco and C. Saunders. Graph kernels for molecular and reduced graphs. In *UK-QSAR and Chemoinformatics Group, Spring meeting*, 2007a.
- A. Demco and C. Saunders. Molecular graph kernels for drug discovery. In *6th IARP-TC-15 Workshop on Graph-based Representations in Pattern Recognition*, 2007b.
- A. Demco and C. Saunders. Finite-length graph kernels and extensions. Technical report, University of Southampton, 2009.
- M. Deshpande, M. Kuramochi, and G. Karypis. Automated approaches for classifying structures. In *2nd ACM SIGKDD Workshop on Data Mining in Bioinformatics*, 2002.

- M. Deshpande, M. Kuramochi, and G. Karypis. Frequent sub-structure based approaches for classifying chemical compounds. *IEEE International Conference on Data Mining*, 2003.
- J. Drews. Drug discovery: A historical perspective. *Science*, 287:1960–1964, 2000.
- R. O. Duda and P. E. Hart. *Pattern Classification*. Wiley-Interscience, 2001.
- T. Fawcett. Roc graphs: notes and practical considerations for data mining researchers. Technical report, HP Laboratories, Palo Alto, CA, USA, 2003.
- A. S. Fraenkel. Complexity of protein folding. *Bulletin of Mathematical Biology*, 55(6): 1199–1210, 1993.
- H. Fröhlich, J. K. Wegner, F. Sieker, and A. Zell. Optimal assignment kernels for attributed molecular graphs. In *Int. Conf. on Machine Learning (ICML)*, pages 225 – 232, 2005.
- T. Gärtner. Predictive graph mining with kernel methods. *Advanced Methods for Knowledge Discovery from Complex Data*, 2005a.
- T. Gärtner. Predictive graph mining with kernel methods. In *Advanced Methods for Knowledge Discovery from Complex Data*, 2005b.
- T. Gärtner, P. A. Flach, and S. Wrobel. On graph kernels: Hardness results and efficient alternatives. In *Learning Theory and Kernel Machines, 16th Annual Conferences on Learning Theory and 7th Kernel Workshop, Proceedings*, volume 2843, pages 129–143. Springer Verlag, 2003.
- J. Gasteiger and T. Engel, editors. *Chemoinformatics: A Textbook*. Wiley, 2003.
- V. J. Gillet, G. M. Downs, J. D. Holliday, and M. F. Lynch. Computer storage retrieval of generic chemical structures in patents. 13. reduced graph generation. *J. Chem. Inf. Comput. Sci.*, 31:260–270, 1991.
- V. J. Gillet, P. Willett, and J. Bradshaw. Similarity searching using reduced graphs. *J. Chem. Inf. Model.*, 43(2):338–345, 2003.
- M. Glick, J. L. Jenkins, J. H. Nettles, H. Hitchings, and J. W. Davies. Enrichment of high-throughput screening data with increasing levels of noise using support vector machines, recursive partitioning, and laplacian-modified naive bayesian classifiers. *J. Chem. Inf. Model.*, 46(1):193–200, 2006a.
- M. Glick, J. L. Jenkins, J. H. Nettles, H. Hitchings, and J. W. Davies. Screening for new antidepressant leads of multiple activities by support vector machines. *J. Chem. Inf. Model.*, 46(1):158–167, 2006b.
- C. Hansch and T. Fujita. A method for the correlation of biological activity and chemical structure. *J. Am. Chem. Soc.*, 86:1616–1626, 1964.

- D. R. Hardoon, S. Szedmak, and J. Shawe-Taylor. Canonical correlation analysis: An overview with applicaiton to learning methods. Technical Report CSD-TR-03-02, Royal Holloway, University of London, 2003.
- G. Harper, G. S. Bravi, S. D. Pickett, J. Hussain, and D. V. S. Green. The reduced graph descriptor in virtual screening and data-driven clustering of high-throughput screening data. *J. Chem. Inf. Comput. Sci.*, 44(6):2145–2156, 2004.
- D. Haussler. Convolution kernels on discrete structures. Technical report, Department of Computer Science, University of Santa Cruz, 1999.
- L. G. Holden, C. Prochnow, Y. P. Chang, R. Brandsteitter, L. Chelico, U. Sen, R. C. Stevens, M. F. Goodman, and X. S. Chen. Crystal structure of the anit-viral apobec3g catalytic domain and functional implications. *Nature*, 2008.
- A. L. Hopkins, J. S. Mason, and J. P. Overington. Can we rationally design promiscuous drugs? *Current Opinion in Structural Biology*, 16(1):127–136, 2006.
- T. Horvath, T. Gärtner, and S. Wrobel. Cyclic pattern kernels for predictive graph mining. In *Proceedings of the International Conference on Knowledge Discovery and Data Mining*, 2004.
- L. Jacob, B. Hoffman, V. Stoven, and J.-P. Vert. Virtual screening of gpcrs: An in silico chemogenomics approach. *BMC Bioinformatics*, 9(363), 2008.
- T. Joachims. Making large-scale SVM learning practical. In B. Schölkopf, C. Burges, and A. Smola, editors, *Advances in Kernel Methods - Support Vector Learning*, chapter 11, pages 169–184. MIT Press, Cambridge, MA, 1999.
- W. L. Jorgensen. The many roles of computation in drug discovery. *Science*, 303(5665):1813–1818, Mar 2004.
- R. N. Jorissen and M. K. Gilson. Virtual screening of molecular databases using a support vector machine. *J. Chem. Inf. Model.*, 45(3):549–561, 2005.
- H. Kashima and A. Inokuchi. Kernels for graph classification. In *1st ICDM Workshop on Active Mining (AM-2002)*, Maebashi, Japan, 2002.
- H. Kashima, K. Tsuda, and A. Inokuchi. Marginalized kernels between labeled graphs. In *Proceedings of the Twentieth International Conference on Machine Learning (ICML-2003)*, Washington DC, USA, 2003.
- L. B. Kier and L. H. Hall. *Molecular Connectivity in Structure-Activity Analysis*. Wiley, New York, 1986.
- R. D. King, S. H. Muggleton, A. Srinivasan, and M. J. Sternberg. Structure-activity relationships derived by machine learning: the use of atoms and their bond connectivities to predict mutagenicity by inductive logic programming. *Proc Natl Acad Sci U S A*, 93(1):438–442, Jan 1996.

- R. I. Kondor and J. Lafferty. Diffusion kernels on graphs and other discrete structures. In *Proceedings of the International Conference on Machine Learning*, 2002.
- P. Krosgaard-Larsen, T. Liljefors, and U. Madsen, editors. *Textbook of Drug Design and Discovery*. Taylor and Francis, 2002.
- A. R. Leach and V. J. Gillet. *An Introduction to Chemoinformatics*. Kluwer Academic Publishers, 2003.
- C. Lemmen and T. Lengauer. Computational methods for the structural alignment of molecules. *J. Comput. Aided. Mol. Des.*, 14(3):215–232, Mar 2000.
- C. Leslie, E. Eskin, J. Weston, and W. S. Noble. Mismatch string kernels for svm protein classification. *Advances in Neural Information Processing Systems*, 15, 2003.
- C. Leslie and R. Kuang. Fast string kernels using inexact matching for protein sequences. *Journal of Machine Learning Research*, 5:1435–1455, 2004. ISSN 1533-7928.
- Y. Li and J. Shawe-Taylor. Using kcca for japanese-english cross-language information retrieval and document classification. *Journal of Intelligent Systems*, 2005.
- P. Lind and T. Maltseva. Support vector machines for the estimation of aqueous solubility. *J. Chem. Inf. Comput. Sci.*, 43(6):1855–1589, 2003.
- H. X. Liu, C. X. Xue, R. S. Zhang, X. J. Yao, M. C. Liu, Z. D. Hu, and B. T. Fan. Quantitative prediction of logk of peptides in high-performance liquid chromatography based on molecular descriptors by using the heuristic method and support vector machines. *J. Chem. Inf. Model.*, 44(6):1979–1986, 2004a.
- H. X. Liu, R. S. Zhang, X. J. Yao, M. C. Liu, Z. D. Hu, and B. T. Fan. Qsar study of ethyl 2-[(3-methyl-2,5-dioxo(3-pyrrolinyl))amino]-4-(trifluoromethyl) pyrimidine-5-carboxylate: An inhibitor of ap-1 and nf-kb mediated gene expression based on support vector machines. *J. Chem. Inf. Comput. Sci.*, 43(4):1288–1296, 2003.
- H. X. Liu, R. S. Zhang, X. J. Yao, M. C. Liu, Z. D. Hu, and B. T. Fan. Prediction of the isoelectric point of an amino acid based on ga-pls and svms. *J. Chem. Inf. Model.*, 44(1):161–167, 2004b.
- H. Lodhi, C. Saunders, J. Shawe-Taylor, N. Cristianini, and C. Watkins. Text classification using string kernels. *Journal of Machine Learning Research*, 2:419–444, 2002.
- H. Lodhi, J. Shawe-Taylor, N. Cristianini, and C. Watkins. Text classification using string kernels. *Advances in Neural Information Processing Systems*, volume 13, 2001.
- P. Mahé. *Kernel design for virtual screening of molecules using support vector machines*. PhD thesis, Ecolé des Mines de Paris, 2006.

- P. Mahé, N. Ueda, T. Akutsu, J.-L. Perret, and J.-P. Vert. Graph kernels for molecular structure-activity relationship analysis with support vector machines. *J. Chem. Inf. Model.*, 45(4):939–51, 2005.
- P. Mahé, N. Ueda, T. Akutsu, and J. Vert. Extensions of marginalized graph kernels. In *Proceedings of the 21st International Conference on Machine Learning (ICML-2004)*, Banff, Alberta, Canada, 2004.
- P. Mahé and J.-P. Vert. Graph kernels based on tree patterns for molecules. Technical report, Technical report HAL:ccsd-000095488, Sep 2006.
- C. Manly, S. Louise-May, and J. Hammer. The impact of informatics and computational chemistry on synthesis and screening. *Drug Discov. Today*, 6(21):1101–1110, Nov 2001.
- Y. C. Martin, J. L. Kofron, and L. M. Traphagen. Do structurally similar molecules have similar biological activity? *Journal of Medicinal Chemistry*, 45(19):4350–4358, 2002.
- J. S. Mason, A. C. Good, and E. J. Martin. 3-d pharmacophores in drug discovery. *Current Pharmaceutical Design*, 7(7):567–597, 2001.
- M. J. McGregor and V. Pallai. Clustering of large databases of molecules: Using the mdl "keys" as structural descriptors. *J. Chem. Inf. Comput. Sci.*, 37:443–448, 1997.
- W. Noble and A. Demco. Personal communication, 2007.
- G. V. Paolini, R. H. B. Shapland, W. P. van Hoorn, and J. S. Mason. Global mapping of pharmacological space. *Nature Biotechnology*, 24(7):805–815, July 2006.
- S. D. Pickett, J. S. Mason, and I. M. McLay. Diversity profiling and design using 3d pharmacophores: Pharmacophore-derived queries (pqd). *J. Chem. Inf. Comput. Sci.*, 36:1214–1223, 1996.
- L. Ralaivola, S. J. Swamidass, H. Saigo, and P. Baldi. Graph kernels for chemical informatics. *Neural Networks*, 18(8):1093–1110, 2005.
- J. Ramon and T. Gärtner. Expressivity versus efficiency of graph kernels. In *First International Workshop on Mining Graphs, Trees and Sequences (held with ECM-L/PKDD03)*, 2003.
- S. Renner and G. Schneider. Scaffold-hopping potential of ligand-based similarity concepts. *ChemMedChem*, 1(2):181–185, 2006.
- R. Rosipal and L. J. Trejo. Kernel parital least squares regression in reproducing kernel hilbert spaces. *Journal of Mahcine Learning Research*, 2:97–123, December 2001.
- J. C. Saeh, P. D. Lyne, B. K. Takasaki, and D. A. Cosgrove. Lead hopping using svm and 3d pharmacophore fingerprints. *J. Chem. Inf. Model.*, 4(45):1122–1133, Jul 2005.

- C. Saunders and A. Demco. *Perspectives of Neural-Symbolic Integration*, chapter Kernel for Strings and Graphs. Springer Berlin / Heidelberg, 2007.
- G. Schneider, M. Lee, M. Stahl, and P. Schneider. De novo design of molecular architectures by evolutionary assembly of drug-derived building blocks. *Journal of Computer-Aided Molecular Design*, 2000.
- G. Schneider, W. Neidhart, T. Giller, and G. Schmid. ‘scaffold-hopping’ by topological pharmacophore search: A contribution to virtual screening. *Angewandte Chemie International Edition*, 38(19):2894–2896, 1999a.
- G. Schneider, W. Neidhart, T. Giller, and G. Schmid. ”scaffold-hopping” by topological pharmacophore search: A contribution to virtual screening. *Angew. Chem. Int. Ed.*, 38(19):2894–2896, 1999b.
- G. Schneider, P. Schneider, and S. Renner. Scaffold-hopping: How far can you jump? *QSAR & Combinatorial Science*, 25(12):1162–1171, 2006.
- B. Scholkopf and A. J. Smola. *Learning with Kernels*. MIT Press, 2002.
- B. Schölkopf, R. Williamson, A. Smola, and J. Shawe-Taylor. Sv estimation of a distribution’s support. *NIPS*, 1999.
- A. Schuffenhauer, N. Brown, P. Ertl, J. L. Jenkins, P. Selzer, and J. Hamon. Clustering and rule-based classifications of chemical structures evaluated in the biological activity space. *J. Chem. Inf. Model.*, 47(2):325–336, 2007.
- J. Shawe-Taylor and N. Cristianini. *Kernel Methods for Pattern Analysis*. Cambridge, 2004.
- S. Skiena. *The Algorithm Design Manual*. Springer-Verlag, 1997.
- M. Song, C. M. Breneman, J. Bi, N. Sukumar, K. P. Bennett, S. Cramer, and N. Tugcu. Prediction of protein retention times in anion-exchange chromatography systems using support vector regression. *J. Chem. Inf. Comput. Sci.*, 42(6):1347–1357, 2002.
- N. Stiefl, I. A. Watson, K. Baumann, and A. Zaliani. Erg: 2d pharmacophore descriptions for scaffold hopping. *J. Chem. Inf. Model.*, 46(1):208–220, 2006.
- Y. Takahashi, M. Sukekawa, and S.-I. Sasaki. Automatic identification of molecular similarity using reduced-graph representation of chemical structure. *J. Chem. Inf. Comput. Sci.*, 32:639–643, 1992.
- R. Tarjan. Depth-first search and linear graph algorithms. *SIAM Journal of Computing*, 1(2):146–160, 1972.
- Choon-Hui Teo and S.V.N. Vishwanathan. Fast and space efficient string kernels using suffix arrays. In *In Proceedings of the Twenty Third International Conference on Machine Learning*, 2006.

- K. Tsuda and J. Vert. *Kernel Methods in Computational Biology*. MIT Press, 2004.
- A. Vinokourov, J. Shawe-Taylor, and N. Cristianini. Inferring a semantic representation of text via cross-language analysis. *Advances of neural information processing systems*, 15, 2002.
- S.V.N. Vishwanathan and A.J. Smola. Fast kernels for string and tree matching. *Advances in Neural Information Processing Systems*, volume 15, 2003.
- M. K. Warmuth, J. Liao, G. Rätsch, M. Mathieson, S. Putta, and C. Lemmen. Active learning with support vector machines in the drug discovery process. *J. Chem. Inf. Comput. Sci.*, 43(2):667–673, 2003.
- C. Watkins. *Advances in Large Margin Classifiers*, chapter Dynamic Alignment Kernels, pages 39–50. MIT Press, 2000.
- P. Willet, V. Winterman, and D. Bawden. Implementation of nearest neighbour searching in an online chemical structure search system. *J. Chem. Inf. Comput. Sci.*, 26(1):36–41, 1986.
- J. Xu and A. Hagler. Chemoinformatics and drug discovery. *Molecules*, 7:566–600, 2002.
- C. X. Xue, R. S. Zhang, H. X. Liu, M. C. Liu, Z. D. Hu, and B. T. Fan. Support vector machines-based quantitative structure-property relationship for the prediction of head capacity. *J. Chem. Inf. Model.*, 44(4):1267–1274, 2004a.
- C. X. Xue, R. S. Zhang, M. C. Liu, Z. D. Hu, and B. T. Fan. Study of the quantitative structure-mobility relationship of carboxylic acids in capillary electrophoresis based on support vector machines. *J. Chem. Inf. Model.*, 44(3):950–957, 2004b.
- C. X. Xue, R. S. Zhang, M. C. Liu, X. J. Yao, Z. D. Hu, and B. T. Fan. Qsar models for the prediction of binding affinities to human serum albumin using the heuristic method and a support vector machine. *J. Chem. Inf. Model.*, 44(5):1693–1700, 2004c.
- C. X. Xue, R. S. Zhang, M. C. Liu, X. J. Yao, H. X. Liu, Z. D. Hu, and B. T. Fan. An accurate qspr study of o-h bond dissociation energy in substituted phenols based on support vector machines. *J. Chem. Inf. Model.*, 44(2):669–677, 2004d.
- X. Yao, A. Panaye, J. P. Doucet, R. S. Zhang, H. F. Chen, M. C. Liu, Z. D. Hu, and B. T. Fan. Comparative study of qsar/qspr correlations using support vector machines, radial basis function neural networks, and multiple linear regression. *J. Chem. Inf. Model.*, 44(4):1257–1266, 2004.
- C. W. Yap and Y. Z. Chen. Prediction of the cytochrome p450 3a4, 2d6, and 2c9 inhibitors and substrates by using support vector machines. *J. Chem. Inf. Model.*, 44(4):982–992, 2005.

- V. V. Zernov, K. V. Balakin, A. A. Ivaschenko, N. .P. Savchuk, and I. V. Pletnev. Drug discovery using support vector machines. the case studies of drug-likeness, agrochemical-likeness, and enzyme inhibition predictions. *J. Chem. Inf. Comput. Sci.*, 43(6):2048–2056, 2003.
- H. Zhao. Scaffold selection and scaffold hopping in lead generation: a medicinal chemistry perspective. *Drug Discovery Today*, 12(3/4):149–155, 2007.