COMPARISON OF SEARCH-BASED AND KERNEL-BASED METHODS FOR GRAPH-BASED RELATIONAL LEARNING

by

CHRIS MANUEL GONSALVES

Presented to the Faculty of the Graduate School of

The University of Texas at Arlington in Partial Fulfillment

of the Requirements

for the Degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE AND ENGINEERING

THE UNIVERSITY OF TEXAS AT ARLINGTON

December 2005

ACKNOWLEDGEMENTS

I would like to thank Dr. L. Holder, my advisor and mentor. His expertise, guidance, understanding and support gave me the vision to successfully complete my thesis and also added considerably to my graduate experience. I would also like to thank Dr. D. Cook and Dr. L. Peterson for their participation and valuable advice on the defense committee.

I would also like to thank my family. Their encouragement and love always kept my spirits up. Very special thanks to all my friends without whom this experience would not have been the same. Lastly, I want to thank God, who gave me the spiritual motivation to achieve what I have.

August 9, 2005

ABSTRACT

COMPARISON OF SEARCH-BASED AND KERNEL-BASED METHODS FOR GRAPH-BASED RELATIONAL LEARNING

Publication No. _____

Chris M Gonsalves, M.S.

The University of Texas at Arlington, 2005

Supervising Professor: Dr. Lawrence B. Holder

Graph-based relational learning has been the focus of relational learning for quite some time. As most of the real-world data is structured, and hence cannot be represented in a single table, various logic-based and graph-based techniques have been proposed for dealing with structured data. Our goal is to perform an in-depth analysis of two such graph-based learning systems. We have selected Subdue to represent the search-based approach and support vector machine (SVM) with graph kernels to represent the kernel-based approach. We perform a comparison between search-based and kernel-based approaches and evaluate their performance in various domains.

A search-based approach to learning typically involves a search through a larger hypotheses space. The main concern of a search-based learning system is to search

through the hypothesis space efficiently. Kernel-based approaches on the other hand do not involve generation and search of a hypotheses space. Instead, a kernel-based system maps the given input space to a higher-dimensional space to perform linear classification.

Experiments are performed on the mutagenesis dataset which is one of the benchmark datasets for structured, relational learning and artificial domains. Our aim is to evaluate these two systems based on their classification performance on the mutagenesis dataset with and without background knowledge. Besides mutagenesis data, various synthetic data including concepts such as ring, tree, clique problem and simple geometric shapes were used to study the ability of the systems to learn the required concepts. Subdue uses an inexact graph match to compute the cost involved in transforming one graph to another. We have implemented this inexact graph match as a potential graph kernel with Support Vector Machine and study its performance on various data domains.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS..... ii

ABSTRACT iii

LIST OF ILLUSTRATIONS..... viii

LIST OF TABLES..... xi

Chapter

1. INTRODUCTION……. 1

1.1 Problem Description ……... 1

2. BACKGROUND AND RELATED WORK……..... 3

2.1 Overview……..... 3

2.2 Graph…… 3

2.2.1 Basics…….. 3

2.2.2 Graph Isomorphism 4

2.3 Relational Learning: Graph-Based approach …..…… 5

2.3.1 Introduction – Relational Learning..... 5

2.3.2 Graph-based Relational Learning 7

3. GRAPH-BASED RELATIONAL LEARNING SYSTEMS…….. 9

3.1 Subdue…… 9

3.2 Support Vector Machine……..... 13

3.3 SVM – Kernel Trick…….……. 13

3.4 Kernel Properties…… 21

3.5 Graph Kernel …… 23

3.6 Algorithms…… 26

3.6.1 Subdue…… 26

3.6.2 SVM…… 27

3.6.3 Random Tree Graph Kernel …… 28

4. EXPERIMENTAL SETUP 34

4.1 Comparison Factors…. 34

4.2 k-Fold Cross Validation…. 37

4.3 Mutagenesis Data..…. 37

4.3.1 Graph-based Representation of Mutagenesis Data…. 40

4.4 System Implementation 43

4.4.1 Subdue…….... 43

4.4.2 SVM – Random Tree Graph Kernel…….. 47

4.4.3 SVM – Inexact Graph Match Kernel…….……..... 50

5. EXPERIMENTAL RESULTS 51

5.1 Artificial Data Domain – Ring Domain and Tree Domain 51

5.1.1 Structurally Large Concepts……..... 52

5.1.2 Structurally Complex Concepts..…….……..... 71

5.2 Mutagenesis Data Domain..... 74

5.3 Artificial Domain Experiments 92

5.4. Graph Kernel – Inexact Graph Match Kernel…… 103

5.5 Summary…… 106

6. CONCLUSION AND FUTURE WORK 108

Appendix

A. GRAPH REPRESENTATION OF MUTAGENESIS DATA USED BY SUBDUE AND SUPPORT VECTOR MACHINE 113

B. CONCEPTS LEARNED BY SUBDUE 125

REFERENCES 130

BIOGRAPHICAL INFORMATION 135

LIST OF ILLUSTRATIONS

Figure Page

2.1 Connected Graph 4

2.2 Simple Concept 5

3.1 Subdue – Inexact Graph Match 10

3.2 Support Vector Machine. (a) Input Space,
(b) Mapped Space and (c) Application of Linear Machine. 14

3.3 Support Vector 15

3.4 SVM Components 16

3.5 Hyperplane 17

3.6 Kernel Trick 20

3.7 Random Walk Graph Kernel 25

3.8 Random Tree Graph Kernel – Problem Setup 32

3.9 Random Tree Graph Kernel – Generated Trees 33

4.1 Possible Hyperplanes 35

4.2 Mutagenesis Compounds. 39

4.3 Mutagenesis Data with Atoms and Bonds 40

4.4 Graph Representation of Mutagenesis Data 41

4.5 Bond between Carbon and Hydrogen Atom 42

5.1 Simple Ring Domain Experiment 53

5.2 Chart – Simple Ring Domain 54

5.3 Ring Domain Experiment with Noise in Dataset 55

5.4 Chart – Ring Domain Experiment with Noise in Dataset 56

5.5 Chart – Subdue's Performance in Ring Domain
with Varying Complexity 58

5.6 Chart – SVM's Performance in Ring domain
with Varying Complexity 59

5.7 Ring Domain Experiment with Noise in Concept. (a) pure ring
concept, (b) noise as unwanted edge, (c) noise as unwanted
vertex label and (d) noise as self-loop 60

5.8 Simple Tree Domain 63

5.9 Chart – Simple Tree Domain 64

5.10 Tree Domain Experiment with Noise in Dataset 65

5.11 Chart – Tree Domain Experiment with Noise in Dataset 66

5.12 Chart – Subdue's Performance in Ring domain
with Varying Complexity 67

5.13 Chart – SVM's Performance in Ring domain
with Varying Complexity 68

5.14 Tree Domain Experiment with Noise in Concept. (a) pure tree
concept, (b) noise as unwanted edge, (c) noise as unwanted
vertex label and (d) noise as double edge 69

5.15 Structurally Complex Structures Domain 72

5.16 Chart – Structurally Complex Structures Domain 73

5.17 Mutagenesis Data – Without Partial Charges 77

5.18 Mutagenesis Data – Without Partial Charges,
Atom Type and Bond Type 79

5.19 Mutagenesis Data- Atom and Elements 81

5.20 ‘Hub’ Node as Background Knowledge 83

5.21 ‘Hub’ Node as Background Knowledge with ‘Ia’ and ‘I1’ Nodes 85

5.22 ‘Concept’ Node as Background Knowledge..... 87

5.23 ‘Benzene’ as Embedded ‘concept’..... 88

5.24 Molecular Structures Learned..... 90

5.25 Disconnected Concepts Domain 93

5.26 Disconnected Concepts Domain with ‘HUB’ Node as Background Knowledge..... 94

5.27 Concept learned with ‘HUB’ Node as Background Knowledge 95

5.28 Concept learned without ‘HUB’ Node as Background Knowledge 96

5.29 Dataset with Negated Concepts 97

5.30 Concepts Learned..... 98

5.31 Dataset with ANDed Concepts 100

5.32 Concepts Learned..... 101

5.33 Clique Problem 102

5.34 Substructures Learned..... 102

LIST OF TABLES

Table Page

5.1 Artificial Domain – Simple Concept 61

5.2 Tree Domain Experiment with Noise in Concept 70

5.3 Mutagenesis Data with Atom, Bond, Atom Type, Bond Type, Element and Charge 75

5.4 Mutagenesis Data – Without Partial Charges 78

5.5 Mutagenesis Data without Partial Charges, Atom Type and Bond Type 80

5.6 Mutagenesis Data with Atoms, Elements and Bonds 82

5.7 'Hub' Node as Background Knowledge 84

5.8 'Hub' Node as Background Knowledge with 'Ia' and 'I1' Nodes 86

5.9 'Concept' Node as Background Knowledge 89

5.10 Mutagenesis Data with Atoms and Bonds 90

5.11 Disconnected Concept Domain with 'HUB' Node 95

5.12 Dataset with Negated Concepts 98

5.13 Positive Examples : Triangle Concepts & Negative Examples: Triangle Concept AND Square Concept 99

5.14 Positive Examples : Square Concepts & Negative Examples: Triangle Concept AND Square Concept 99

5.15 Dataset with ANDed Concepts 101

5.16 Clique Problem 102

5.17 Graph Kernel results for simple Ring Domain..... 104

5.18 Graph Kernel results for Mutagenesis Domain..... 105

CHAPTER 1

INTRODUCTION

1.1 Problem Description

Graph-based learning approaches have gained focus for relational learning in recent years. The main idea behind graph-based relational learning is the ease of representing structured relational data in graph form. Various graph-based techniques have been the focus of research in recent years. Our aim is to perform an empirical analysis on two such graph-based relational systems – Subdue and Support Vector Machine. In this course of research we represent Subdue as a candidate for search-based learning approach and Support Vector Machine as a candidate for kernel-based approach. Our aim is to evaluate these systems to analyze the performance based on classification accuracy and time complexity on various artificial data domains and real-world data domains. As a conclusion, we will highlight the various aspects for which one system is better than the other.

In chapter 2 we will give a brief introduction to relational data and graph-based approaches to representing relational data. In chapter 3 we will introduce graph-based learning systems: Subdue and Support Vector Machine (SVM) – Graph Kernel. We will do an in-depth analysis of both the systems and discuss their characteristics and functioning. We will also discuss the kernel trick and its application with SVM. In this

chapter we will give a detailed approach to the implemented Random Tree Graph Kernel. In chapter 4 we will describe the relational dataset used in our experiments and give an introduction to the Mutagenesis dataset. The Mutagenesis dataset is a standard benchmark dataset used widely in relational learning. We will also comment on the various comparison factors taken under consideration. In this chapter, we will discuss the implementation of Subdue's inexact graph match as an potential graph kernel. In chapter 5 we will show experimental results, depicting the performance of the two systems. In chapter 6 we will discuss the conclusion of the research.

CHAPTER 2

BACKGROUND AND RELATED WORK

2.1 Overview

In this chapter we will give a basic introduction to graphs and graph isomorphism. We will then introduce relational learning and see the various representations possible. Finally we will discuss the representation of structured relational data using graph.

2.2 Graph

2.2.1 Basics

A graph G can be represented as

$G = (V, E)$.

Where V represent a set of vertices/nodes in the graph

E represent a set of edges in the graph.

$E \in \{(u, v) \mid u, v \in V\}$.

There are various ways to represent a graph. For our purpose we will consider a labeled graph G [23] as:

Let Σ_V, Σ_E be the sets of vertex labels and edge labels respectively. Let X be a finite nonempty set of vertices. Let v be a function such that v: $X \rightarrow \Sigma_V$.

Let L be a set of ordered pairs of vertices called edges and e be a function such that e: $L \rightarrow \Sigma_E$.

Then G = (X, v, Σ_V, L, e Σ_E) is a labeled graph with directed edges.

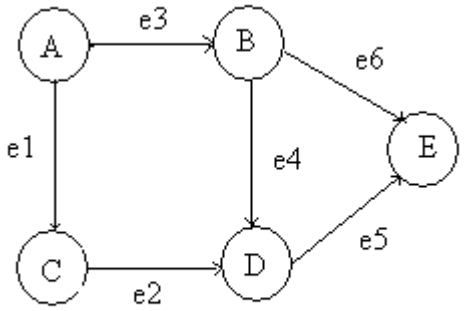


Figure 2.1 Connected Graph

Σ_V = {A, B, C, D, E}

Σ_E = {e_1, e_2, e_3, e_4, e_5, e_6}

2.2.2 Graph Isomorphism

As we will discuss later, graph isomorphism and subgraph isomorphism are few of the many techniques used in graph-based approaches. Isomorphism can be used as tools to calculate the similarity level between given two graphs. Consider two graphs G_1 = (V_1, E_1) and G_2 = (V_2, E_2). G_1 and G_2 are said to be isomorphic if and only if there exists a function.

F: $V_1 \rightarrow V_2$ such that for all u, v ϵ V_1,

(u, v) ϵ $E_1 \leftrightarrow$ (f (u), f (v)) ϵ E_2.

The function f is said to be an isomorphism between G_1 and G_2.

Subgraph isomorphism generalization of the graph isomorphism problem.

A subgraph can be defined as a graph whose vertices and edges are a subset of the other graph. Subgraph isomorphism for given two graphs G_1 and G_2 decides if there is a subgraph of one graph (G_1 or G_2) which is isomorphic to another graph.

In the later chapters we will observe how graph isomorphism and subgraph isomorphism are both implemented in graph-based learning algorithms. As subgraph isomorphism is NP-Complete, most of the algorithms implementing this technique face the drawback of time complexity.

2.3 Relational Learning: Graph-Based approach

2.3.1 Introduction - Relational Learning

Relational learning involves learning a structured concept from structured data. By structured data we mean data that is generated by combining simpler components into more complex objects. Problems involving natural language understanding, complex planning problems or understanding molecular biology involve understanding the entities involved and relations among them. Such relational representations use a first-order model to describe the problem domain; wherein examples are given to the learner in terms of attribute and attribute relations rather than by simple propositions. ILP is the study of learning methods for data and rules that are represented in first-order predicate logic [41]. Recently many approaches have been practiced to represent and learn concepts relationally, as a propositional representation leads to loss of relational information and generates larger representations of data. An alternative representation

of relational data besides first-order logic is graphs. The main advantage of a graph-based representation is the ease of representing the data and the ease of understanding the sample space, the hypothesis space and the concepts learned.

Graphs help in representing more complex domains as compared to traditional attribute-value approaches, i.e., to represent spaces that provide a more natural means for expressing real-world data. Most work in machine learning has been based on attribute-value learning. However, as the size of the data increases, the attributes associated with them increase. Often real-world data has attribute associations, which are usually lost in an attribute-value representation. Consider an example concept to be learned – 'Water molecule: H_2O'

This concept can be represented in attribute format as

1. <atom, atom, atom> : <H, H, O>
2. <atom, no_atoms, atom> : <H, 2, O>
3. <atom, no_atoms, bonding, atom> : <H, 2, single_bond, O>

We however observe that the attribute-based approach is incapable of representing the actual concept. Also the semantics associated with the concept becomes more complex as more information needs to be represented.

A graph-based representation of this concept can be given as:

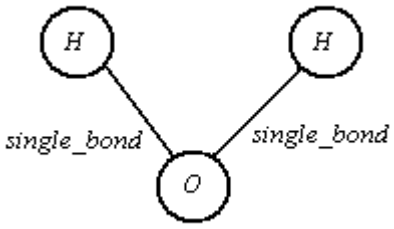


Figure 2.2 Simple Concept

For real-world relational data, the best representation is either first-order logic or graphs. We will concentrate on graph-based approaches for representing structured relational data.

2.3.2 Graph-based Relational Learning

The field of relational learning has only recently focused on graph-based approaches. In a graph-based approach, the relational data is represented as a graph. Even though the graph-based approach has been applied in unsupervised learning, with the advancement of graph-based search techniques with potential heuristics, recently the graph-based approach has been introduced in supervised learning.

Most of the graph-based learning approaches make use of graph search techniques and other characteristics of graph to learn concepts and to search the hypothesis space. Some of the most common graph-based search techniques used are heuristic searches, random walk, tree, graph isomorphism and subgraph isomorphism.

An efficient and approximate method of extracting typical patterns from graph structured data by Graph-Based Induction (GBI) and its improvement (B-GBI, Cl-GBI) is introduced by Motoda [45, 46, 47].

In a graph-based learning system, the basic issue to be considered is that of searching for a graph pattern, also called a subgraph common to a given set of graphs. A graph G_1 is said to be a subgraph of G_2 if and only if the set of vertices and set of edges of G_1 are a subset of the set of vertices and set of edges of G_2 respectively. A subgraph can represent a concept to be learned. The second issue is an extension of the first, wherein one has to comment on the existence of a particular subgraph in a given list of graphs. The efficiency of a graph-based learning system depends on the efficiency of searching for regular graph patterns in a given set of datasets. Some approaches such as QSAR based on SVM [31] and random walk kernel by Kashima, Tsuda and Inokuchi [10], use random walk on graphs to search for subgraphs. Other approaches based on a tree approach are discussed by Gartner [8]. Few approaches involved in the search of such graph patterns include subgraph isomorphism. A system that implements both graph isomorphism and greedy search is Subdue [2].

CHAPTER 3

GRAPH-BASED RELATIONAL LEARNING SYSTEMS

In this chapter we will introduce two graph-based learning systems: Subdue and Support Vector Machine (SVM). We will present the basic differences in these systems on the basis of learning from the hypothesis space and performing classification. We will introduce the kernel trick applied by SVM and discuss in depth the graph kernel. Finally we will introduce the random tree graph kernel implemented in the course of this research.

3.1 Subdue

Subdue [2] is a graph-based learning system capable of performing both unsupervised and supervised learning. Subdue represents data as labeled graphs where the vertices of these graphs represent entities and the edges represent entity attribute and relationships. As an unsupervised learner, Subdue does knowledge discovery by finding common and frequent subgraphs in the given set of input graphs using a compression-based heuristic. It then compresses the graphs with the substructures found. This process continues until all potential substructures have been discovered and the graph can be represented by a single vertex.

Subdue implements an inexact graph match method to search for potential substructures. For an error free dataset, an exact graph match can suffice. However,

inexact graph match is applied when the input data contains noise. Due to the presence of noise there can occur small differences in the relational concept to be learned in the given data. The inexact graph match approach handles this situation by calculating the cost required to transform one graph into another. Thus for given graphs G_1 and G_2,inexact graph match calculates the minimum cost required to make G_1 and G_2 isomorphic to each other. The working of the inexact graph match is shown in figure 3.1.

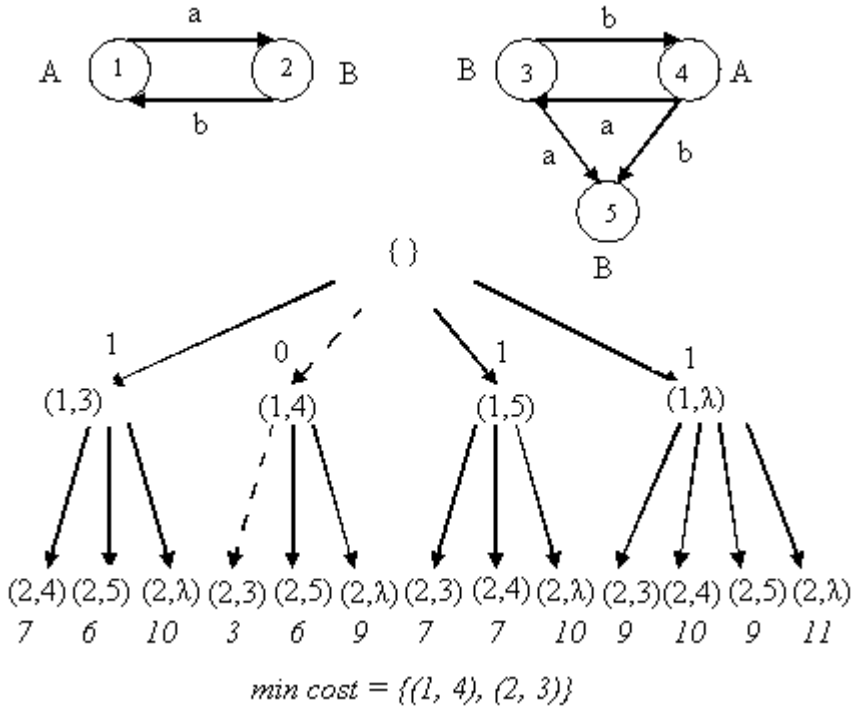


Figure 3.1 Subdue - Inexact Graph Match

In the inexact graph match approach [42], each distortion is assigned a cost. A distortion can be a transformation such as deletion, insertion and substitution of vertices and edges. Given two graphs G_1 and G_2, we assume G_2 to be the distorted version of G_1. By definition inexact graph match is a mapping

$$f : V_1 \rightarrow V_2 \cup \{\lambda\}$$

where V_1 and V_2 are the set of vertices of G_1 and G_2, respectively.

A vertex $v \in V_1$ that is mapped to λ (i.e. $f(v) = \lambda$) is deleted. The cost involved in an inexact graph match *cost(f)*, is the sum of the cost of individual transformations resulting from *f*. We also define *matchcost(G_1, G_2)* as the least-cost function that is sufficient to map graph G_1 onto graph G_2.

Given two graphs G_1 and G_2, and a set of distortion costs, the actual computation of *matchcost(G_1, G_2)* is determined using a tree search approach as shown in figure 3.1. A state in the search tree corresponds to a partial match that maps a subset of the vertices of G_1 to a subset of the vertices in G_2. Initially we start with an empty mapping ({}) at the root of the tree. Expanding a state corresponds to adding a pair of vertices, one from G_1 and one from G_2. A final state in the search tree is a match that maps all vertices of G_1 to G_2 or to λ. In figure 3.1 we assume unit distortion cost. The numbers in circles are the costs computed at the corresponding states. The goal state to be found by our tree search procedure is the final state with minimum cost among all final states. From figure 3.1 we observe that the minimum cost is given by the mapping $f(1) = 4$ and $f(2) = 3$. Given graphs G_1 with n vertices and G_2 with m vertices, $m \geq n$, the complexity of the full inexact graph match is $O(n^{m+1})$. The inexact graph match is widely used throughout the discovery and evaluation process in Subdue and can thus degrade the performance significantly.

Subdue performs a search – "beam search", where the generated substructures are stored for further expansion on a limited-length queue. The process of generating a

substructure begins with a single vertex. Subdue then expands one adjacent vertex and the corresponding edge at a time. If the newly generated substructure is better than the parent substructure, the parent substructure is replaced with the new substructure. As an unsupervised learner, Subdue uses these generated substructures to compress the input graphs. The process of finding the substructures and compressing the graph continues until no further compression is possible and the whole graph can be represented by a single vertex.

As a supervised learner, Subdue removes all the positive examples covered by the generated substructures. Thus the process of generating better substructures and removing all the positive examples continues until all positive examples are covered. The beam search implemented represents a general to specific search guided by a Minimum Description Length (MDL) heuristic [2]. The MDL is based on the principle that the best way to describe a given set of data is by minimizing the description length of the entire dataset. Description length can be best described as the number of bits necessary to describe a given graph. This description length can then be used to calculate the MDL of each substructure considered by the beam search.

For classification, Subdue uses subgraph isomorphism. A graph S = (V', E') is a subgraph of another graph G = (V, E) if and only if

$V' \subseteq V$, and $E' \subseteq E$ and $((v_1, v_2) \in E' \rightarrow (v_1, v_2) \in E')$.

From an application point of view, Subdue has been applied as a knowledge discovery system to learn from structured data in various domains. Some of the domains of application are

- Link Discovery for Counter Terrorism.
- Anomaly Detection.
- CAD Circuit Analysis.
- Web Search.
- Protein Structure Analysis.
- DNA gene transcription sites.
- Analyzing carcinogenic chemical compounds.
- Analyzing compounds for chemical toxicity.
- Analyzing aviation incident reports.
- Analyzing seismic events.
- Analyzing program source code.
- Analyzing telecommunications data.

3.2 Support Vector Machine

The Support Vector Machine (SVM) originated from the statistical learning theory proposed by Vladimir Vapnik [31] and developed by Boser, Guyon and Vapnik [32]. SVM is a method for generating functions from a given set of labeled training examples. SVM as a learning system can be implemented for both classification as well as regression. The simplest model of SVM is the Maximal Margin Classifier (MMC). MMC works only for data which are linearly separable in the hypothesis space and hence cannot be used in many real-world problems.

For classification, SVM takes in a set of training examples and maps the input space into a feature space. The feature space is a high dimensional space and the mapping is given by Φ. SVM then tries to create a hyperplane in the generated feature space. This hyperplane will partition the given space into positive examples space and negative examples space. Figure 3.2 shows the basic operation of SVM. The given input-space is a non-linear space as shown in figure 3.2.a. The input-space is mapped to feature-space via the mapping Φ as shown in figure 3.2.b. Figure 3.2.c shows the linear hyperplane learned by SVM.

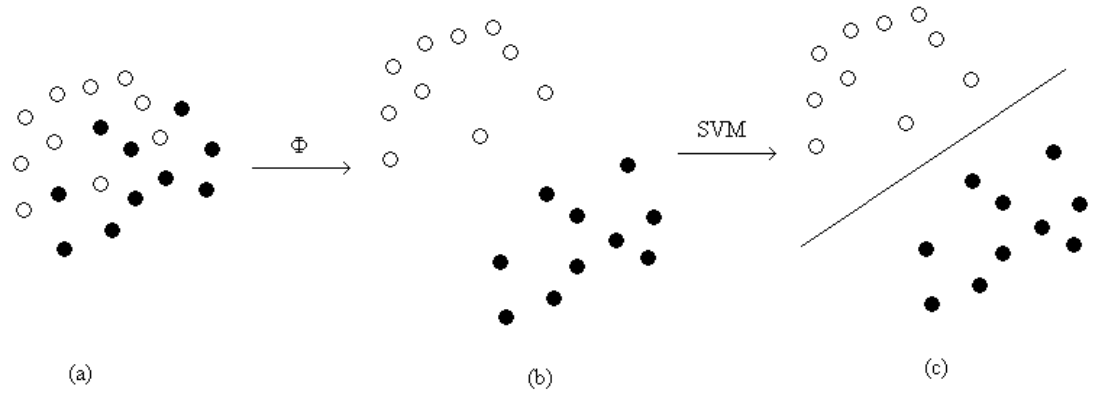


Figure 3.2 Support Vector Machine. (a) Input Space, (b) Mapped Space and (c) Application of Linear Machine.

SVM uses a real-valued function $h(x) = sign(\mathrm{w}.\mathrm{x} + \mathrm{b})$, such that for any input x_i Є X, x is a positive class if $h(\mathrm{x}) \geq 0$ and a negative class if $h(\mathrm{x}) < 0$. Thus SVM when used for a simple linear binary classification, creates a hyperplane that divides the given data examples into two binary classes. This is done by setting a maximum margin such that the distance from the closest examples i.e. the margin to the hyperplane is maximized. The use of maximum-margin hyperplane was motivated by Vapnik

Chervonenkis's theory, which states that the error involved is minimized when the margins are maximized.

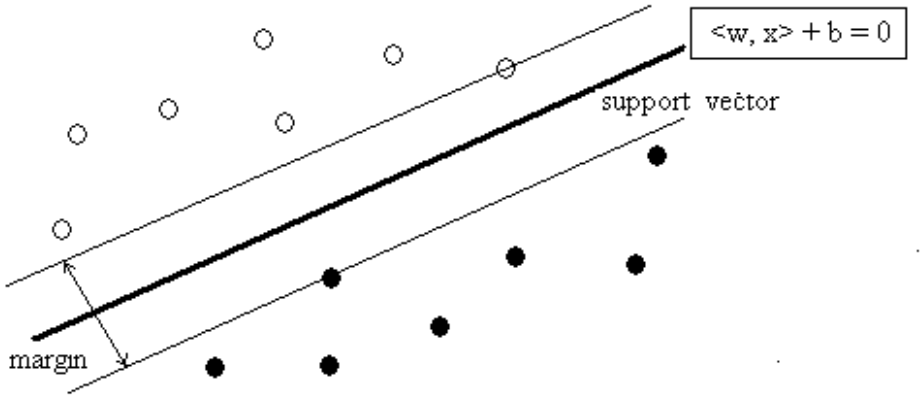


Figure 3.3 Support Vector

Figure 3.3 shows a hyperplane and the associated support vectors. The hyperplane is defined as <w, x> + b = 0 [1]. A hyperplane is an affine subspace of dimension (n-1) which divides the space into two spaces which corresponds to the inputs of the two distinct classes. We will discuss more about hyperplanes in a later section. In the maximum margin method, the supporting planes (margins) are pushed apart until they bump into the support vectors as shown in figure 3.3. The solution hence only depends on these support vectors [43]. The parameters involved in defining the maximum-margin hyperplane are derived by solving a quadratic programming optimization (QP) problem. Various approaches are presented for solving the QP problem. The most common method implemented for solving a QP problem is Platt's Sequential Minimal Optimization (SMO) [36]. SMO breaks the given QP problem into various smaller possible QP problems. These subset QP problems are then solved

analytically to avoid the time complexity involved in the original numerical QP optimization problem. This is also a major reason why the memory requirement involved in SVM with SMO is linear. This also assists the SMO to handle larger training sets.

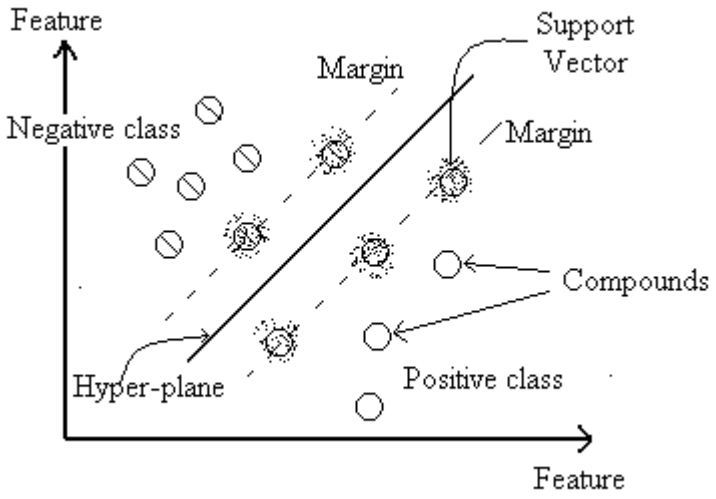


Figure 3.4 SVM Components.

Figure 3.4 shows the various components that contribute to the SVM. The figure shows a hyperplane drawn in linear space. The hyperplane is defined as

$$f(x) = \langle w, x \rangle + b$$

The linear separation of the feature space is then given by

$$f(x) = \langle w, x \rangle + b$$

$$h(x) = sign(f(x))$$

The solution to the linear combination of training points is then given by

$$W = \Sigma \alpha_i y_i x_i$$

$$\alpha_i \geq 0$$

where α_i are the parameters that define the margin and y_i is the corresponding class of x_i.

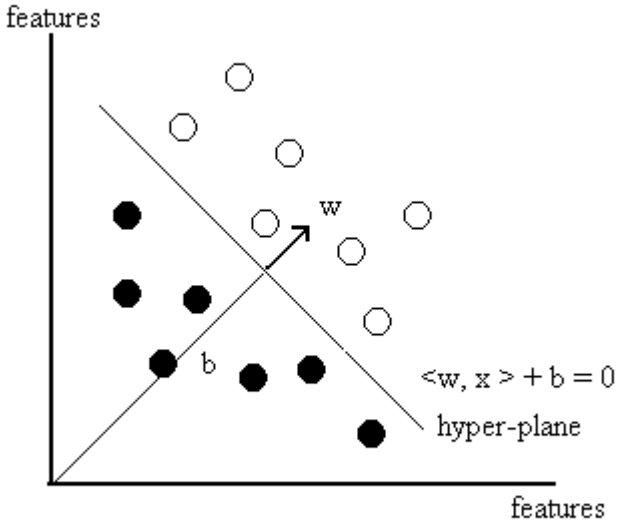


Figure 3.5 Hyperplane

A geometric interpretation of the hypothesis defined by the hyperplane that splits the input space X into two parts is defined by <w, x> + b = 0 as shown in figure 3.5. The positive examples are above the hyperplane and the negative examples below the hyperplane as shown in figure 3.5. The vector w defines a direction perpendicular to the hyperplane, while varying the parameter b moves the hyperplane parallel to itself.

The parameter w is often referred to as the weight while the parameter b is referred to as bias or the threshold.

The decision function *f(x)* can then be re-written as

$$f(x) = \Sigma \alpha_i y_i x_i + b$$

where b is the offset of the hyperplane and is given as [43]

$$b = y_p - \sum_{j=1}^{m} \alpha_j y_j K(x_j, x_p)$$

b is determined by the support vectors for which $\alpha_i \geq 0$. y_i is the class corresponding to x_i and m is the number of data points in the input space. K (x_i, x_j) is the kernel implemented to map the input space into the feature space.

H(x) is the classifying function. Thus if $h(x) \geq 0$, then x is classified positive else negative. The output of SVM, usually a real value is hard to interpret. An algorithm to map the SVM outputs into posterior probabilities has been introduced by Platt [3].

3.3 SVM – Kernel Trick

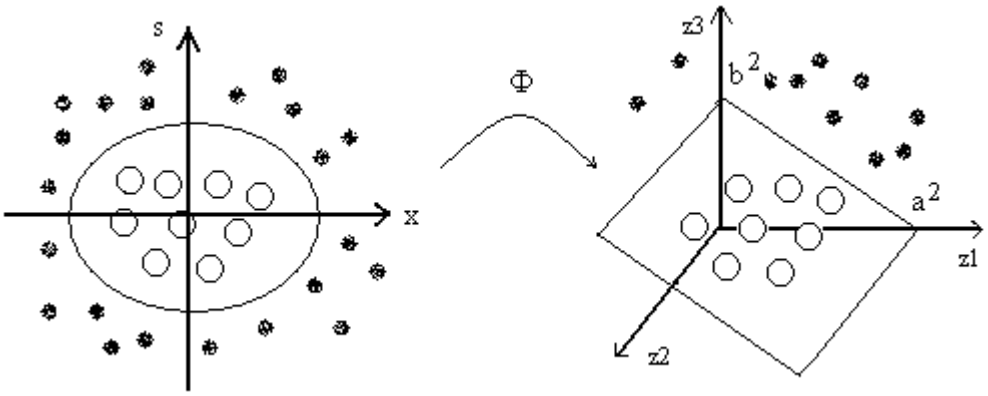
The original optimal hyperplane algorithm proposed by Vladimir Vapnik [31] is a linear classifier. As most data is not linear, Bernhard Boser, Isabelle Guyon and Vapnik [32] suggested a way to create non-linear classifiers by applying the kernel trick to maximum-margin hyperplanes. The kernel trick was originally proposed by Aizerman [33]. By using the kernel trick, one can map the given nonlinear space to a

higher-dimensional space. The need to map the non-linear space to a higher-dimensional space is to allow the classifier to draw a linear hyperplane in the mapped space. Thus the new approach with the kernel trick which generates a linear hyperplane in the higher dimensional space is actually a non-linear plane in the original input space.

The kernel K can be represented as $K(x, s) = < \Phi(x). \Phi (s) >$; where x and s are the examples from the input space $X \subset R_d$, Φ is a non-linear mapping from the input space X to feature space F and <,.,> is the inner dot product. Thus a kernel K(x, s) can be described as the inner dot product of not x and y but $\Phi(x)$ and $\Phi(s)$ in higher dimension. Thus the mapping $\Phi: X \rightarrow H$ is called a feature map from the input data space X to a higher-dimension feature space H. The feature space H is assumed to be a Hilbert space of real-valued functions defined on X. A Hilbert space is a vector space H with an inner product of <x, s>. An example of a Hilbert space includes a real number R^n which is the dot product of <x, s>.

Let us consider a mapping of data from R2 to R3 using a feature map

$$K (x, s) = <\Phi(x) .\Phi (s)> = (x^2, \sqrt{2}xs, s^2)$$



$$\Phi : (x, s) \rightarrow (x^2, \sqrt{2}xs, s^2)$$

$$\left(\frac{x}{a}\right)^2 + \left(\frac{s}{b}\right)^2 = 1 \rightarrow \left(\frac{z_1}{a^2}\right) + \left(\frac{z_3}{b^2}\right) = 1$$

Figure 3.6 Kernel Trick

There are various approaches for mapping data elements from R^2 space to R^3 space. Figure 3.6 shows such a mapping from R^2 space to R^3 space. The mapping function Φ maps the given input space R^2 to R^3 space The mapping discussed above has a useful property of allowing the computation of the inner products of feature vectors Φ(x) and Φ(w) in R^3 by just squaring the inner product of the data vectors x and s in R^2. Thus we have:

$$K(x, s) = < \phi(x), \phi(s) >$$

$$= x_1^2 w_1^2 + 2\, x_1 x_2 s_1 s_2 + x_2^2 s_2^2$$

$$= (x_1 s_1 + x_2 s_2)^2$$

$$= (< x, s >)^2$$

Thus, for SVM with kernel trick, the hyperplane is defined as

$$f(x) = \Sigma\, \alpha_i\, y_i < \Phi(x_i),\, \Phi(x) > + b$$

Hence we have,

$$f(x) = \Sigma\, \alpha_i\, y_i\, K(x_i, x) + b$$

In the above equations we observe that one need not compute the dot product between x_1 and x_2. The kernel $K(x_1, x_2)$ is used to implicitly compute the dot product. Thus the kernel K can be used to map the non-linear space into a linear higher-dimensional space, enabling linear separation by a hyperplane.

3.4 Kernel Properties

To prove the existence of feature spaces, Mercer [1, 30] showed that the necessary and sufficient condition for a symmetric kernel K(x, s) to be a kernel, is that it needs to be positive definite [44]. A function *h* is positive definite, if for given inputs $x_1 \ldots x_n \in R^n$ and complex numbers $\alpha_1 \ldots \alpha_n$ we have

$$\Sigma\, h(x_j - x_k)\, \alpha_j\, \alpha_k \geq 0.$$

$$j, k = 1 \ldots n.$$

Hence for any given set of input examples $x_1, x_2 \ldots x_n$ and for a set of some real-valued numbers $\lambda_1 \ldots \lambda_n$

$$\sum_{i=1}^{l} \sum_{j=1}^{l} \lambda i\, \lambda j\, K(xi, xj) \geq 0$$

Some commonly-used kernels include:

Polynomial Kernel:

$$K(x, x') = (x \cdot x')^{d}$$

Radial Basis Kernel:

$$K(x, x') = exp\left(-\frac{\|x - x'\|}{2\sigma^{2}}\right)$$

Sigmoid Kernel:

$$K(x, x') = \tanh(kx \cdot x' + c)$$

Exponential Kernel:

$$K(x, x') = exp\left(a + b K_{G}(x, x')\right)$$

The basic properties of a kernel are:

1. A kernel is continuous.
2. A kernel is symmetric: K(x, s) = K(s, x).

It is also easy to create a new hybrid kernel [29] from existing kernels due to some basic properties of being positive definite.

1. Given two kernels K_1 and K_2, let α_1, α_2 be two positive numbers.

$$K(x, s) = \alpha_1 K_1(x, s) + \alpha_2 K_2(s, x).$$

2. Given two kernels K_1 and K_2, the multiplication yields a kernel.

$$K(x, s) = K_1(x, s)\, K_2(s, x)$$

3. Also $K(x, s) = \exp(K_1(x, s))$, is a kernel by taking the limit of the series expansion of the exponential function.

4. If ψ is an $\mathbb{R}p$ – valued function on X and K_3 is a kernel on $\mathbb{R}p \times \mathbb{R}p$, then: $K(x, z) = K_3(\psi(x), \psi(z))$.

5. If A is a positive definite matrix of size $d \times d$, then: $K(x, z) = x^T Az$.

Our aim is to develop a graph kernel, which satisfies the basic properties of kernels discussed above. The graph kernel we developed is both continuous and symmetric.

3.5 Graph Kernel

In this section we will give a brief introduction to graph kernels and their application in SVM. As discussed in chapter 2, a labeled graph G can be defined by its set of vertices V, set of edges E, set of vertex labels Σ_V and set of edges labels Σ_E. Our task is to construct a kernel function $K(G_1, G_2)$ between graphs G_1 and G_2. In the case of SVM one tries to map the given data space to a higher-dimensional space. With graph kernels, we try to compute product graphs $G_1 x G_2$.

We define the product of two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ as $G_1 \times G_2$. The vertex and edge set of the product graphs are thus defined as

$$V(G_1 \times G_2) = \{(v_1, v_2) \in V_1 \times V_2 : (\text{label}(v_1) = \text{label}(v_2))\}$$

$$E(G_1 \times G_2) = \{((u_1, u_2), (v_1, v_2)\} \in V^2(G_1 \times G_2):$$

$$(u_1, v_1) \in E_1 \wedge (u_2, v_2) \in E_2 \wedge (\text{label}(u_1, v_1) = \text{label}(u_2, v_2))\}$$

Various approaches have been proposed for a complete graph kernel [20]. The most common approach is the measure of the common subgraph of the two graphs.

Thus a basic graph kernel can be given as,

$$K(G_1, G_2) = < \Phi(G_1), \Phi(G_2) >.$$

Where, $\Phi(G_1)$ represents the mapping of G_1 into a higher-dimensional space and $\Phi(G_2)$ represents the mapping of G_2 into a higher-dimensional space.

Thus if G_1 and G_2 are isomorphic to each other then we have the mapping Φ as

$$K(G_1, G_1) - 2K(G_1, G_2) + K(G_2, G_2) = < \Phi(G_1) - \Phi(G_2), \Phi(G_1) - \Phi(G_2) > = 0$$

If G_1 and G_2 are not isomorphic to each other then

$$K(G_1, G_1) - 2K(G_1, G_2) + K(G_2, G_2) \neq 0.$$

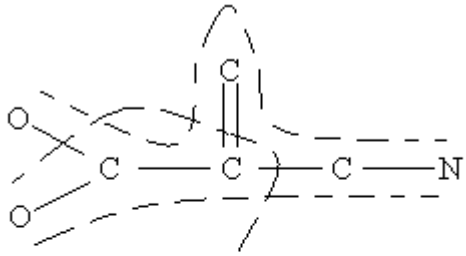
Hence to decide if given graphs G_1 and G_2 are isomorphic it is sufficient to evaluate the expressions $K(G_1, G_1)$, $K(G_1, G_2)$ and $K(G_2, G_2)$.

As general isomorphism problems are computationally expensive, various approaches have been developed to generate an efficient graph kernel. One such approach for a graph kernel between labeled graphs has been proposed by Kashima [10, 11, 12]. In this approach, graph comparison is based on a comparison of the path distributions under two independent random walks on the graphs to be compared.

For graphs G_1 and G_2 the kernel is given as

$$K(G_1, G_2) = \sum_{(h1, h2) \in H(G1) \times H(G2)} p_{G1}(h_1)\, p_{G2}(h_2)\, K_L(h_1, h_2)$$

Where, H (G) is the set of paths of the graph G and p_G the corresponding random walk model. K_L is a label kernel, i.e. a function assessing the paths similarity.



h1 = (O-C-C-C=O) - p(h1)

h2 = (O-C-C) - p(h2)

h3 = (N-C-C=C=C-C-O) - p(h3)

Figure 3.7 Random Walk Graph Kernel

This kernel thus averages the label kernel with respect to the set of paths found in the graphs. The mapping involves implicit projection of the graphs G_1 and G_2 into a higher-dimensional feature space where each dimension corresponds to a particular path in the graph. For more details on the functioning of the graph kernel refer to [10].

3.6 Algorithms

In this section we will present the algorithms for both Subdue and Support Vector Machine. We will also introduce the Random Tree Graph kernel implemented in this research.

3.6.1 Subdue

1. Training: Given – Training examples (x_1, y_1), $(x_2, y_2)\ldots$ $(x_n\ y_n)$ and $y_i \in \{+1/-1\}$. Where, y_i is the class corresponding to x_i.
    A. Search for best substructure
        1. Initially consider single vertex. This is the current substructure. Let best-list be the list of best substructures found.
        2. Let parent-list be the list of all parent substructures found.

           Let child-list be the list of substructures been generated.

           While the queue is not empty and the number of substructures found ≤ limit, do
            - Extend the current substructures on the parent-list in all possible ways. This gives the new child substructure list.
            - Evaluate the current substructures found and replace the parent-list with the child-list.
            - Replace the substructures in the best substructure list with new substructures based on the value.
    B. Store the best substructures.

C. Remove all the positive examples covered by the best substructures.

D. Repeat step A-C until all positive examples are covered.

2. Testing: Given – Testing examples (x_1, y_1), $(x_2, y_2)\ldots$ $(x_n\ y_n)$.

   A. Consider each substructure concept learned and perform a subgraph isomorphism on the given testing examples.

   B. If the concept is found in the test example xi, then xi is classified as (+1) else (-1).

3.6.2 SVM

1. Training: Given – Training examples (x_1, y_1), $(x_2, y_2)\ldots$ $(x_n\ y_n)$ and $y_i \in \{+1/-1\}$. Where, y_i is the class corresponding to x_i.

   A. Map the given input data into a higher-dimension by applying the mapping Φ. This mapping will generate the feature space. The feature space is generated by applying the kernel trick as discussed in section 3.3.

   B. Create a hyperplane in the Feature Space. SVM creates a hyperplane by implementing SMO. SMO helps solve the QP problem involved thus generating the parameters necessary to define the hyperplane. The hyperplane has maximum margin in the feature space and minimal error.

   C. The results of the training are the parameters of the margins of the Support Vector (SV) i.e. α_1, $\alpha_2\ldots$ α_n. These parameters α_i define the margins and the hyperplane.

2. Testing: Given – Testing examples (x_1, y_1), $(x_2, y_2)\ldots (x_n, y_n)$. The hypothesis space H is the feature space defined by Kernel K (g_1, g_2).

   A. For a test example x_i compute

$$h(xi) = \operatorname{sign}\Big(\sum_{xi \in sv} \alpha_1\ yi\ K(g_1, g_2) + b\Big)$$

Where, α_i is the margin parameter.

Y_i is the class corresponding to x_i.

K (g_1, g_2) is the graph Kernel.

B is the offset of the plane from the origin.

To determine b, we consider an example, say p, which is unbound i.e. α_i is non-zero.

$$b = y_p - \sum_{j=1}^{m} \alpha_j y_j K(x_j, x_p)$$

If h $(x_i) \geq 0$, x_i is classified as class +1 and if h $(x_i) < 0$, x_i is classified as class -1.

3.6.3 Random Tree Graph Kernel

As compared to various graph kernel approaches mentioned above, we have concentrated on the information flow in the graph. We calculated information flow by calculating entropy at the nodes. Each node can be assumed to have information flowing in and out from the neighboring nodes. Each adjacent edge-vertex pair can define the information flow depending on the weight associated with each adjacent

edge. We have considered unit weight for all edges. A very general definition of information entropy can be given as the amount of information carried in a signal at a given time. For more detailed information on information entropy refer to Shannon's discussion [40].

In our approach we are concentrating on the amount of information carried at the nodes of the graph. As discussed above, various kernels can be applied with SVM. However, the e*xponential* kernel has shown better performance as compared to other kernels on many datasets. More about exponential families and kernels methods are discussed by Canu and Smola [39]. We will use the *exponential* kernel as a part of the graph kernel to calculate K (G_1, G_2).

Algorithm:

1. Calculate Information Entropy at all nodes of the graphs G_1 and G_2. In case of directed graphs, each node consists of incoming edges and outgoing edges. Let *Wi* be the weight of node *Ni*. Let *p* and *q* be the number of incoming edges and outgoing edges. Thus each outgoing edge from node *Ni* will transfer (*Wi* / *q*) units of information. To simplify the calculation we consider unit weight of both edges and nodes. Hence in this case, the total information at any node is equal to the summation of the incoming nodes and the outgoing nodes. Thus at all time, the information content of any node is the number of its adjacent edges.
2. Sort the nodes in ascending order. This generates $v_i \in V$, where V is the set of all vertices of graph G and the corresponding $e_i \in E$, where E is the list of edges

in the graph G, for graph G_1 and G_2. Depending on the required accuracy level and resource allocation, we consider $v_i \in V$ as potential roots for random tree generation. Here we consider a tradeoff between accuracy and computation time. Ideally all nodes in the graph should be considered as potential roots. But this proves computationally expensive. We hence consider a set of nodes as potential roots for tree expansion. It may happen that, skipping a few nodes as potential roots may led to a fall in accuracy. In our course of experimentation we have considered minimum five nodes as potential roots. The number of potential roots to be considered is a user defined value and can be varied depending on the time constraints. More number of roots involves more computation.

3. For each $u_i \in V_1$.

   Set Cost to zero.

   For each $v_i \in V_2$

      If label(u_i) = label(v_i) then

         Expand tree with u_i as root, one edge and one vertex at a time. Let eu_i and p_i be the expanded edge and vertex pair for u_i. Similarly, let ev_i and q_i be the expanded edge and vertex pair for v_i.

      If label(eu_i) = label(ev_i) and label(q_i) = label(p_i) then

         Add eu_i and p_i to the expanded graph.

            Calculate cost = cost + weight (eu_i) * weight (p_i).

            Cost (u_i, v_i) = max_cost (Cost(ui,vi), cost).

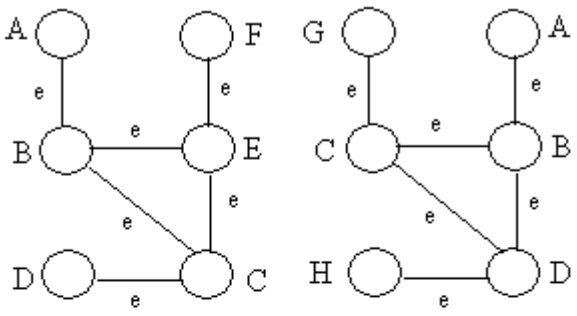
4. Final_Cost = max(Cost(ui, v_i))
5. Normalization factor: norm_factor = (number of vertices in G_1 + number of vertices in G_2 + number of edges in G_1 + number of edges in G_2)
6. Thus,

$$K(G1, G2) = \exp\left(1.0 + \frac{2.0 * \text{Final\_Cost}}{\text{Norm\_factor}}\right)$$

An example implementation of the random tree graph kernel is shown in figure 3.8 and figure 3.9. Our aim is to calculate K (G_1, G_2) for two graphs G_1 and G_2. As shown, the potential roots considered for G_1 are {B, E, C} and G_2 are {C, B, D}. In the case of G_1, node B, E and C each have three adjoining edges. Likewise, nodes A, F and D each have one adjoining edge. Thus arranging the nodes in ascending order we get {B, E, C, A, F, D}. We can consider all nodes as potential roots for tree expansion. However in figure 3.8 we have considered three to speed up the process. Thus the potential roots considered are {B, E, C}. Similarly, in case of G_2, we obtain the potential roots as {C, B, D}. The max cost calculated is 7 as shown in figure 3.9.

Hence K (G_1, G_2) = exp (1.0 + 2.0 * 7.0 / 24.0) = 4.87.

Similarly, we can evaluate the various kernels required by the SVM.



G_1	| B (3) | E (3) | C (3) | | A (1) | F (1) | D (1) |	G_2	| C (3) | B (3) | D (3) | | G (1) | A(1) | H (1) |
G_1 Potential Roots = {B, E, C}		G_2 Potential Roots = {C, B, D}	

Random Trees Generated :

$\overset{1}{\{B}, \overset{2}{C\}} = \Phi$ cost = 0

$\overset{1}{\{C}, \overset{2}{B\}} = \Phi$ cost = 0

$\overset{1}{\{E}, \overset{2}{C\}} = \Phi$ cost = 0

$\overset{1}{\{C}, \overset{2}{D\}} = \Phi$ cost = 0

$\overset{1}{\{E}, \overset{2}{B\}} = \Phi$ cost = 0

$\overset{1}{\{B}, \overset{2}{D\}} = \Phi$ cost = 0

$\overset{1}{\{E}, \overset{2}{D\}} = \Phi$ cost = 0

Figure 3.8 Random Tree Graph Kernel – Problem Setup

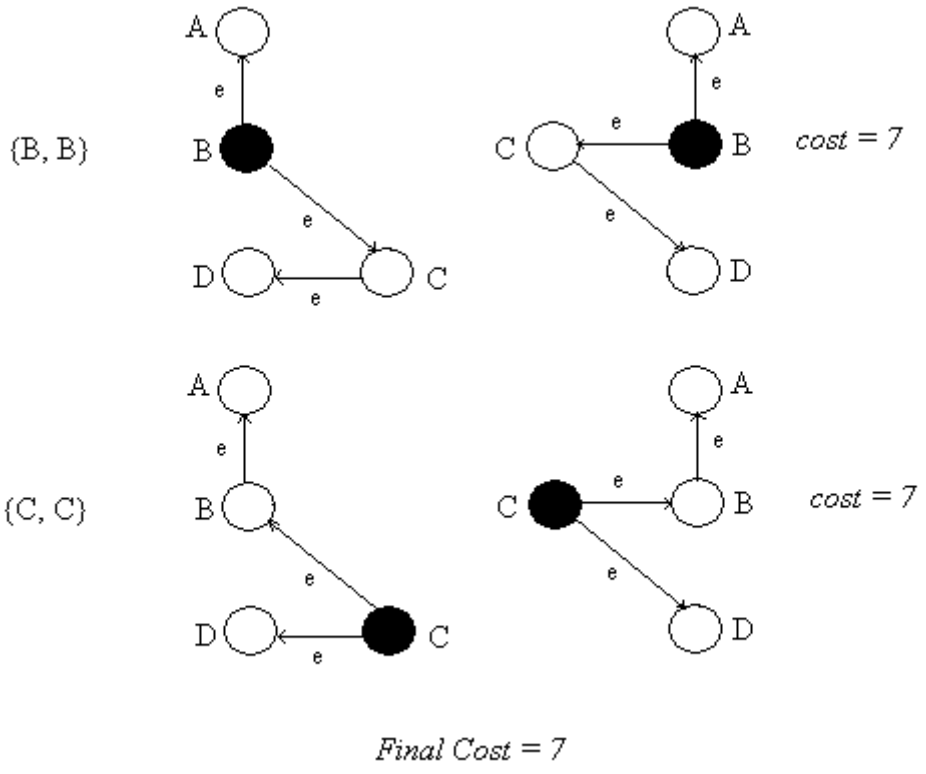


Figure 3.9 Random Tree Graph Kernel – Generated Trees

CHAPTER 4

EXPERIMENTAL SETUP

Comparison of two learning systems involves analysis of the ability to learn a particular concept and classification efficiency. In this chapter we will consider the comparison factors involved. We will also discuss the Mutagenesis dataset and the graph representation of Mutagenesis Data. We also present the two systems under consideration and analyze their functionality.

4.1 Comparison Factors

The main factors to be considered while comparing graph-based approaches such as Subdue and SVM-Graph Kernel is the ability of the system to learn a particular complex concept efficiently. In case of both SUBDUE and SVM-Graph Kernel, the systems search the hypothesis space for a possible concept that best classifies the given datasets. However, the main difference lies in fact that, Subdue actually searches for a concept in the hypothesis space. On the other hand, SVM tries to generate a hyperplane in the mapped higher-dimensional space. Theoretically, there are an infinite number of possibilities for generating a hyperplane. Figure 4.1 shows examples of various possible hyperplanes.

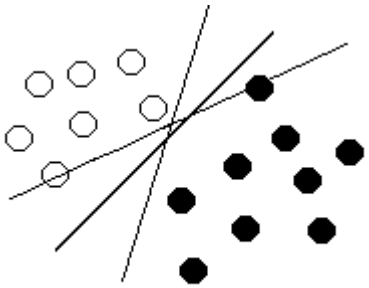


Figure 4.1 Possible Hyperplanes.

These hyperplanes form the Hypothesis space for SVM. However, SVM does not actually generate all the possible hyperplanes. A hyperplane is generated by calculating the parameters αi, required as discussed in chapter 3. These parameters are the result of the implemented kernel. Hence, the better the kernel the better is the hyperplane generated and the better is the classification. Thus we are interested in evaluating Subdue on the basis of the hypothesis space generated and the search for a potential hypothesis. For SVM, we will try to evaluate the performance of the system by evaluating the hyperplane generated based on the implemented kernel.

For any learner, the concept to be learned depends on the representation of that particular concept. By representation we mean the ability to represent the knowledge involved i.e. the syntax and the semantics associated with the meaning of the concept. The concept learned by Subdue is a subgraph while SVM outputs a vector that maps the input data into the feature space. Due to the graph format of the concept learned it is usually straightforward to understand the concept. However, the output of SVM is a higher-dimensional space vector and hence difficult to interpret.

Various techniques have been implemented in improving the performance of supervised learners. Some of the techniques are discussed below:

1. Semantics:

   Improving the representation of the input space reduces semantic complexity. By reducing semantic complexity, the hypothesis space generated is simple and easy to search. As the representation of a particular dataset becomes complex, the time and resources involved in generating the hypothesis space and searching for the best concept increases. Hence resolving the best representation of the data before being fed to the learning system can definitely improve the performance of the learning system.

2. Background Knowledge:

   In most cases of supervised learning, we usually have some knowledge about the concept we want to learn. This knowledge can be passed on to the learning system as background knowledge. By providing the background knowledge, we make sure that the system is learning in the right direction. This can also be called biased supervised learning. There are basically two ways of providing background knowledge. Background knowledge can be implicitly provided to the learner by including it in the input datasets by adding the necessary additional information to the data examples. The other method is to explicitly provide the background knowledge. In this case the background knowledge is provided explicitly to the learner during the learning process.

The techniques described above help in reduction of the hypothesis space learned thus aiding the learner to learn more accurate concepts. This helps in improving the performance of the system. We have addressed the performance of the systems under consideration on the basis of the accuracy of classifying data and the time complexity involved.

4.2 k-Fold Cross Validation

In this research we have concentrated on 10-fold cross validation. In a k-fold cross-validation, the given sample set is divided into k subsets (folds) each of size m/k. For each fold, the classifier is allowed to train using the remaining subsets and the current subset is used for testing. Thus we randomly divide the sample into k disjoint subsets which are fed to the classification algorithm. The left-out subset of the training set is used to evaluate classification accuracy.

4.3 Mutagenesis Data

The Mutagenesis dataset is a standard benchmark dataset for most Inductive Logic Programming [38] related data mining. The dataset consists of 230 compounds, from which 138 are the positive examples while the remaining 92 are the negative examples. The data is obtained from ILP experiments conducted with Progol [37]. The ILP experiments used the generic description of compounds consisting of atoms and their bond connectivity. Mutagenicity is measured using the Ames test using S.

typhimurium TA98. Nitroaromatic and Hydrocyclic compounds are of most concern because of their carcinogenicity. More about this test and results can be found in [4].

The prediction of mutagenesis is important for the understanding and prediction of carcinogenesis as mutagenic chemicals have often been found to be carcinogenic. Various structures of chemical Mutagenesis are discussed by Dr. Josephy [28]. Some common compounds used in mutagenesis study are shown in figure 4.2.

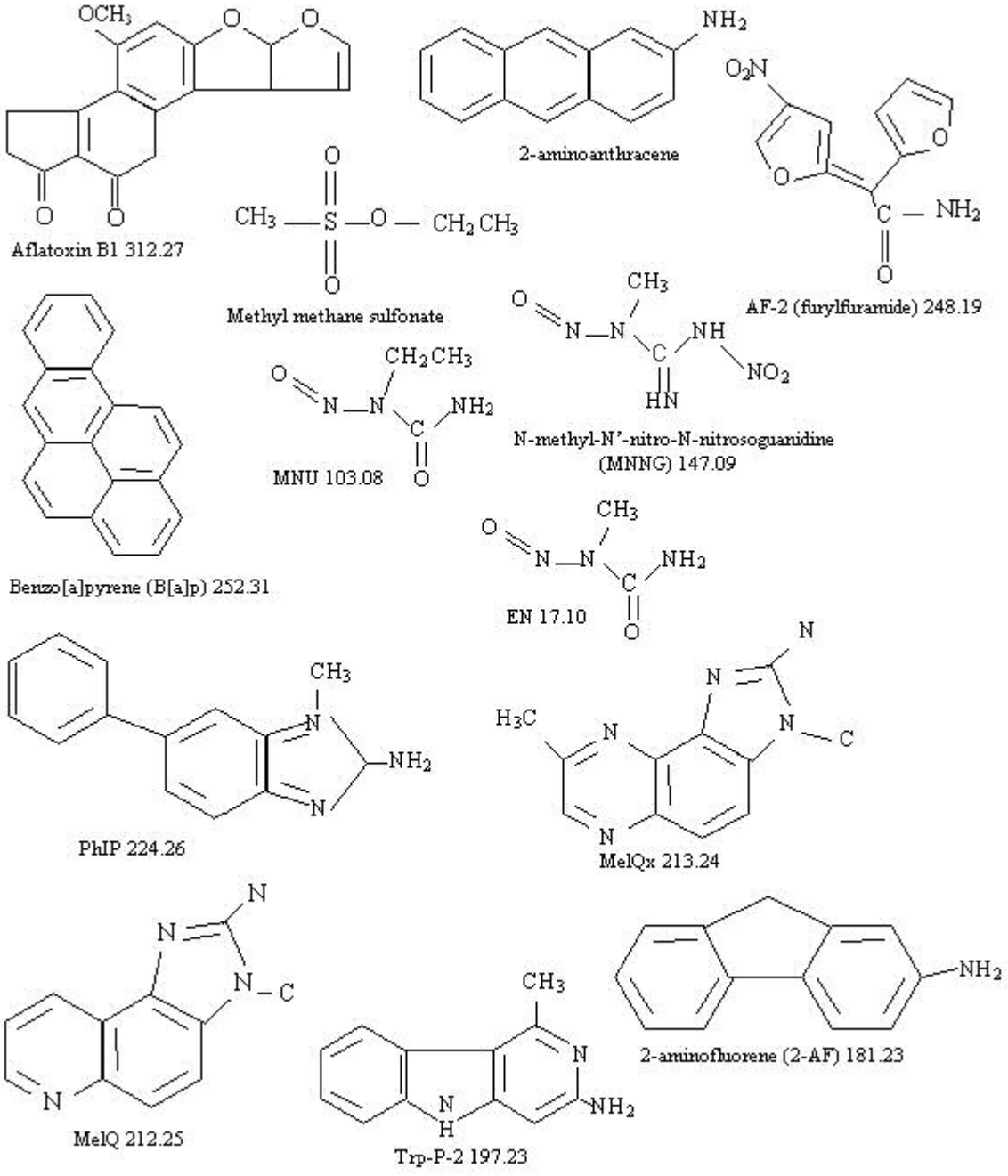


Figure 4.2 Mutagenesis Compounds*.

* Redrawn with permission from Dr. Josephy.

4.3.1 Graph-based Representation of Mutagenesis Data

A basic graph-based representation of mutagenesis data was obtained from Akihiro Inokuchi, Tokyo Research Laboratory IBM Japan. The dataset consisted of plain atoms and bonds. Figure 4.3 shows partial Mutagenesis data as provided by Akihiro Inokuchi.

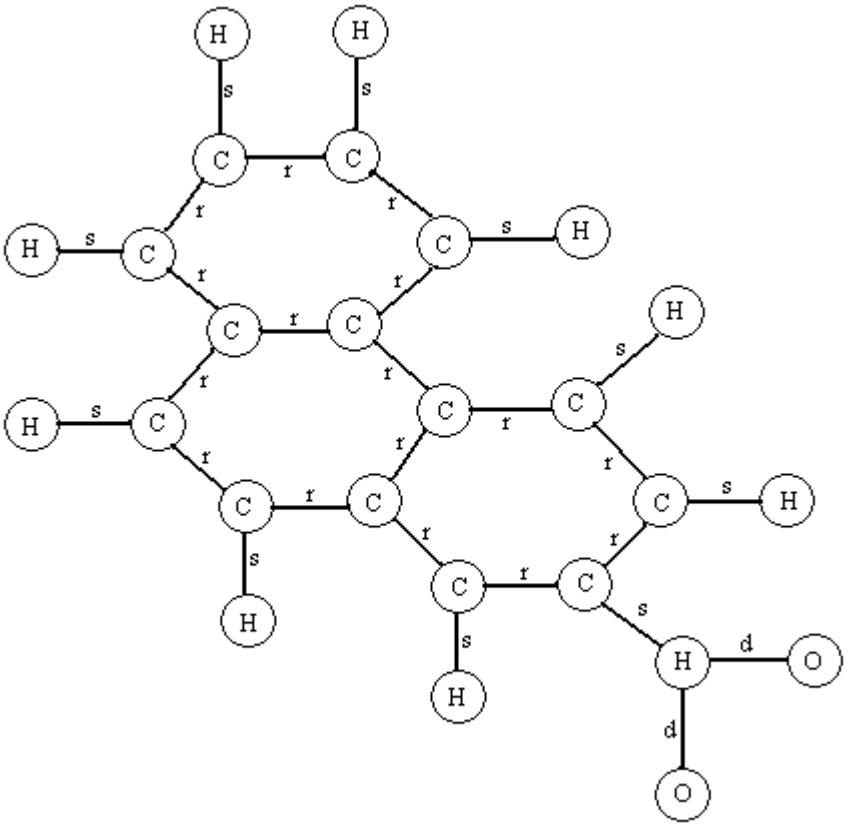


Figure 4.3 Mutagenesis Data with Atoms and Bonds.

To learn more complex structures consisting of more information than just atoms and bonds, we concentrated on the representations described in [27]. This

mutagenesis data contains information about the atoms, bonds, elements, atom type, partial charge and bond type.

In the graph-based representation of mutagenesis data, we represent entities as nodes while the entity-to-entity relationships are represented as edges. Figure 4.4 shows a graph-based template representation of a bond in the mutagenesis dataset.

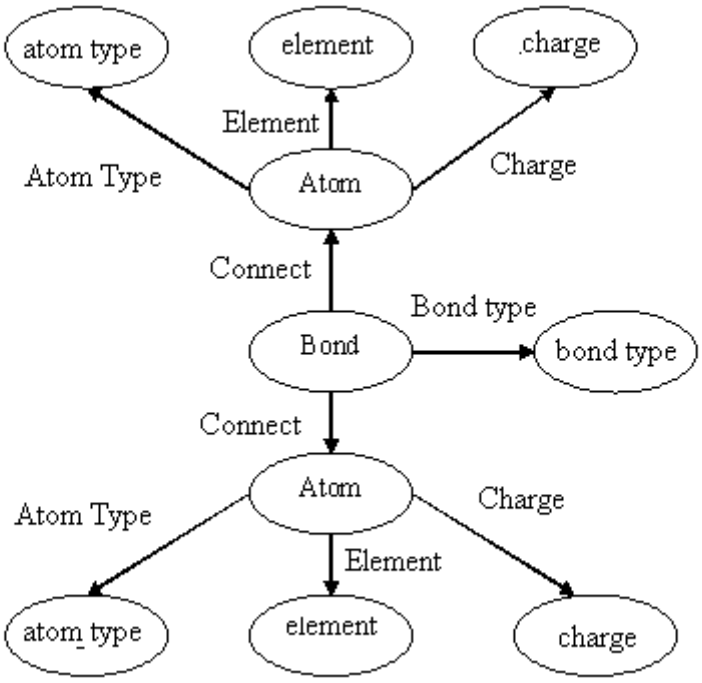


Figure 4.4 Graph-Representations of Mutagenesis Data.

This dataset describes a basic template that forms the entire mutagenesis compound. In this structure, we consider atoms having three basic properties – the atom type, the element itself and the charge of the atom. One atom connects to another through as bond. The bond specifies the bonding type between any two atoms. The various elements under consideration are Carbon (C), Hydrogen (H), Nitrogen (N) and

Oxygen (O). These elements are the major elements constituting a mutagenesis compound. The various charges that are possibly available on these atoms are -0.117, -0.087, 0.013, 0.142, 0.143 and -0.388. Figure 4.5 shows an example bond between a Carbon and Hydrogen, i.e., (C – H).

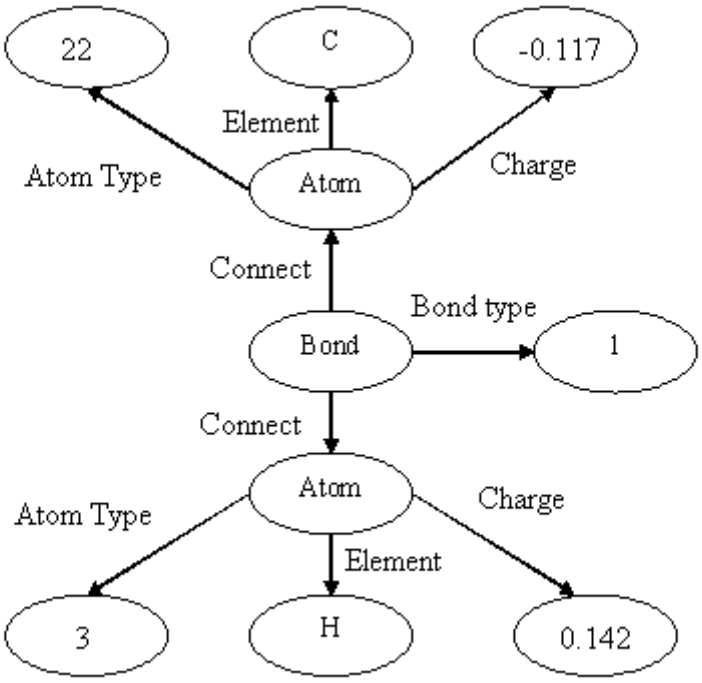


Figure 4.5 Bond between Carbon and Hydrogen Atom.

In our course of research, we will show empirical results of experimentation on the various forms of mutagenesis data. We will also discuss about the role of background knowledge in these mutagenesis datasets.

4.4. System Implementation

4.4.1 Subdue:

A. Algorithm Execution: The source code can be found at http://ailab.uta.edu/subdue/.

B. *Training:* The command to train SUBDUE is –

subdue [parameters] <train graph file>

C. The basic parameters used are –

a. *-eval [1, 2, 3]*

i. *MDL (Minimum Description length)*: The value of substructure S in a graph G is

$$\text{Value}(S, G) = \frac{DL(G)}{(DL(S) + DL(G \mid S)}$$

Where DL is the description length in bits, and (G|S) is G compressed with S.
If Negative graph Gn is present then

$$\text{Value}(S, Gp, Gn) = \frac{[DL(Gp) + DL(Gn)}{[DL(S) + DL(Gp \mid S) + DL(Gn) - DL(Gn \mid S)]}$$

ii. *Size:* The value of substructure S in a graph G is

$$\text{Value}(S, G) = \frac{\text{Size}(G)}{\text{Size}(G) + \text{Size}(G \mid S)}$$

Where,

$$\text{Size}(G) = [\#\text{Vertices}(G) + \#\text{Edges}(G)]$$

(G | S) is G compressed with S.

If Negative Graph Gn is present then,

$$\text{Value}(S, Gp, Gn) = \frac{[\text{Size}(Gp) + \text{Size}(Gn)]}{[\text{Size}(S) + \text{Size}(Gp \mid S) + \text{Size}(Gn) - \text{Size}(Gn \mid S)]}$$

iii. *Set Cover:* The value of a substructure S is computed as the number of positive examples containing S, plus the number of negative examples not containing S, all divided by the total number of examples. At the end of the each iteration, the compression done is replaced by removing all positive examples containing S. In our course of experiments we have used 'Set Cover' evaluation.

b. *- iterations #:*

This specifies the number of iterations made over the input graph in which the best substructure from the previous substructure is used to compress the graph and use in the next iteration. The default value is 1.

c. *- limit #:*

This specifies the number of substructures to be considered in different iterations. The default value is (Vertices + Edges) / 2.

d. *– out<file name>*

This option outputs the best substructure found during the training session in the specified file.

D. *Testing:* The command to test the input graphs is –

test <sub graph file> <test graph file>

The parameters include a sub graph file; one generated using the –out option in Subdue command. The *test* utility computes the following for a given set of substructures and a given set of test graphs:

FP – false positives

TP – true positives

FN – false negatives

TN – true negatives

These parameters can then be used to compute the test set accuracy.

In our experiments we have mostly used 10 fold cross validation. The command to perform m-fold cross validation is –

cvtest [parameters] –nfold m <train graph file>

The parameters are the same as discussed in the training stage. The parameter –nfold specifies the number of cross validation test to be performed. The cvtest command internally calls the test command discussed above.

E. *Input and Output Data Type:*

Subdue takes in a graph file (*.g) as the input file. A Subdue graph is a sequence of vertices and edges. A positive graph is addressed as XP whereas a negative graph is addressed XN.

A vertex is defined as:

v <#> <label>

Where, <#> is a unique vertex ID and <label> is any string or real number.

An edge is defined as follows:

e <vertex 1 #> <vertex 2 #> <label>

d <vertex 1 #> <vertex 2 #> <label>

u <vertex 1 #> <vertex 2 #> <label>

Where, <vertex 1 #> & <vertex 2 #> are the vertex ID's of vertices between which the edge lies. <label> is any string or real number.

The edge ‘e’ is assumed to be directed, unless ‘-undirected’ option is explicitly specified. An edge‘d’ is always directed while an edge ‘u’ is always undirected.

In our course of research, we have concentrated on the evaluation technique the set-cover with limit set to 100. Rest of the parameters were set to their default values.

4.4.2 SVM – Random Tree Graph Kernel:

A. Graph Kernel Algorithm: We have applied a Random Tree Graph Kernel in our approach. The Random Tree Graph Kernel is discussed in details in chapter 3.

B. Algorithm Execution: The executable and source code can be found at http://svmlight.joachims.org/.

C. *Training:* The command to train SVM is –

svm_learn [parameters] <train data file> <output model file>

D. The basic parameters used are –

a. *– t [0,1, 2 ,3 ,4]:*

i. *0: linear (default)*

ii. *1: polynomial (s a * b + c)* ^{d}

iii. *2: radial basis function exp(-gamma ||a - b||* ^{2}*)*

iv. *3: sigmoid tanh(s a * b + c)*

v. *4: user defined kernel (Graph Kernel)*

b. *– q #*

Maximum size of QP– sub problem. The value of # should be more than 2 and by default the value is 10.

c. *– m #*

Define the size of cache for kernel evaluations, in MB (default 40).

d. – u [1, 2, 3]:

i. 1: Vertex Only (default). Used for datasets wherein all the information is concentrated at the vertices. Faster than the other two options as computationally simple.

ii. 2: Vertex – Edge pair. Used for datasets wherein the information is concentrated at both the vertices and edges.

iii. 3: Vertex-Edge-Vertex. A general approach combining the above two options.

This option is implemented to provide background knowledge to SVM while dealing with different datasets.

E. *Testing:* The command to test the input test examples is –

svm_classify [parameters] <test file> <model file> <output file>

F. Input and Output Data Types:

SVM takes in a data file (*.dat) as the input file. The syntax of a 'dat' is as follows:

<line>=<target><feature>:<value> feature>:<value>...
<feature>:<value>#<info>

<target>=+1|-1|0|<float>

<feature>=<integer>|<#>

<value>=<float>

<info>=<string>

At each <line>, the <target> specifies the target class (+1/-1). The <feature> : <value> pair specifies the data feature and the associated value. The string `<info>` can be used to pass additional information to the kernel.

Example -

```
-1 1:0.43 3:0.12 9284:0.2 # abcdef
+1 2:1.00 5:0.93 1000:0.5 # pqrstu
```

The string passed after '#' is the user defined representation of the graphs and is as follows:

|#c| [1 v1] [2 v2] … [n vn] <#e v_i v_j e1> ... <#e v_p v_q en>

Where, |#c| - class of the graph i.e. +1 or -1.

[vertex_no vertex_label] is the vertex information

<edge_type start_vertex end_vertex edge_lable>. Where, 'edge_type' can be 'd' for directed edges and 'u' for undirected edges. 'start_vertex' and 'end_vertex' indicated the vertices between whom the edge exist and 'edge_lable' corresponds to the edge label.

4.4.3 SVM – Inexact Graph Match Kernel:

A. Graph Kernel Algorithm: We have applied Subdue's Inexact Graph match as the graph kernel. The Inexact Graph match is discussed in more detail in section 3.1.

The Inexact Graph Match calculates a cost factor involved in transforming graph G1 to graph G2.

The kernel K (G_1, G_2) is calculated as:

$$K(G1, G2) = \exp\left(1.0 + \frac{\text{Final Cost}}{T}\right)$$

Where, T = number of vertices in G1 + number of edges in G1 + number of vertices in G2 + number of edges in G2.

B. Algorithm Execution:

The training and testing is the same as for SVM – Graph Kernel discussed above.

C. Input and Output Data Type:

The input and output data types are the same as for SVM – Graph Kernel discussed above.

In this chapter we have introduced the Mutagenesis Data domain and its graph-based representation. We have also described in details the experiment setup for both systems and the parameters involved. We also introduced Subdue's inexact graph match as a graph kernel to function with SVM.

CHAPTER 5

EXPERIMENTAL RESULTS

In this chapter, we will evaluate the systems using both an artificial domain and a real-world domain – the Mutagenesis domain. Our main aim is to analyze the performance of the systems on the basis of classification accuracy and computation time. For artificial datasets, we will concentrate on geometric shapes such as rings of varying size and completely-connected graphs. As the graph kernel implemented uses a tree-based approach, we will analyze the performance of both the systems on trees with varying depth. We will then perform experiments on a variety of mutagenesis datasets and analyze the results. Finally, we have introduced the concept of using Subdue's inexact graph match technique as a potential graph kernel.

5.1 Artificial Data Domain – Ring Domain and Tree Domain

We have generated a variety of artificial datasets to test both Subdue and SVM – Graph Kernel. The artificial datasets were generated using 'subgen' an artificial graph-based dataset generator developed for the Subdue project. Subgen takes in input of vertex labels ΣV, edge labels ΣE, size of the graphs which specifies the number of vertices and edges to be included in the graph and the substructure under consideration plus some additional information regarding the connectivity and coverage. Subgen outputs either instances or graphs. Subgen first generates N instances. It then computes

the number of common structures needed to be overlapped. The overlapping limit is provided to Subgen as an input parameter. Subgen tries to merge the instances such that the preferred overlapping is achieved. The final graph is generated from the instances. To achieve the desired graph size and connectivity, Subgen adds random vertices and edges. For source code and information visit: http://ailab.uta.edu/subdue/.

5.1.1 Structurally Large Concepts

Initially we performed experiments with artificial domains such as the ring domain and the tree domain. Our aim was to measure the time constraint involved in learning a concept of increasing structural size from a given set of training examples while maintaining a 100 % training set accuracy. By increasing the size of the ring under consideration, it is possible to analyze the performance of both the systems.

The training set consisted of 10 positive and 10 negative examples from the ring domain. The positive example is the ring of size N (N nodes), whereas the negative example is a line of N nodes. Figure 5.1 shows the list of positive and negative examples for ring with varying size. All the vertices are labeled 'V' and all the edges are labeled 'E'. The positive and negative examples are such that, the basic concept learned by the system is the ring concept, i.e. the only structure that can distinguish the given positive and negative examples is the ring structure.

The parameters set for Subdue are as follows:

1. – eval = 3 (Set cover.)
2. – limit (default).

3. – iterations = 2 (Sufficient to cover all the concepts necessary to be learned.)

The parameters set for SVM are as follows:

1. – t = 4 (Graph kernel.)
2. – u = 1 (default)

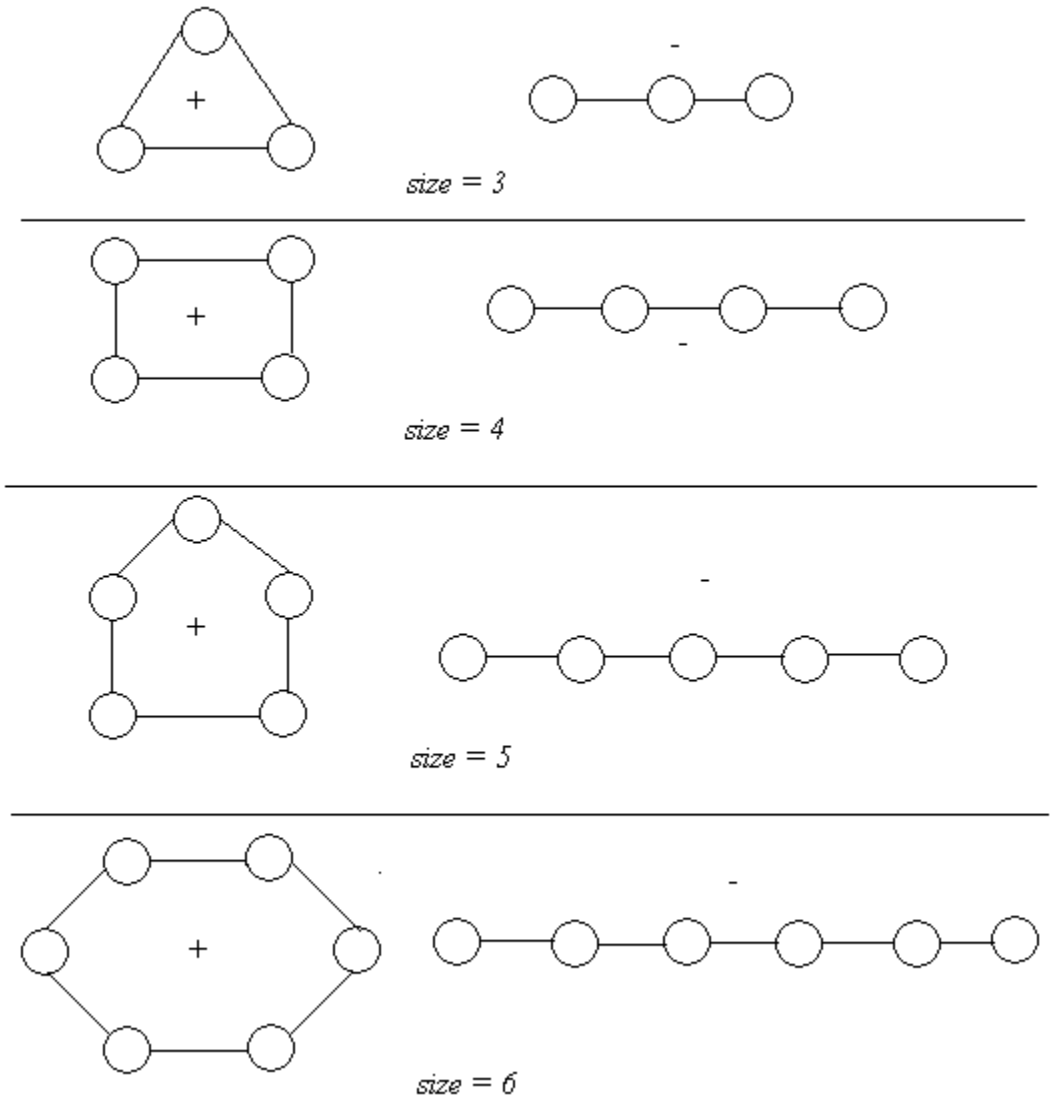


Figure 5.1 Simple Ring Domain Experiment.

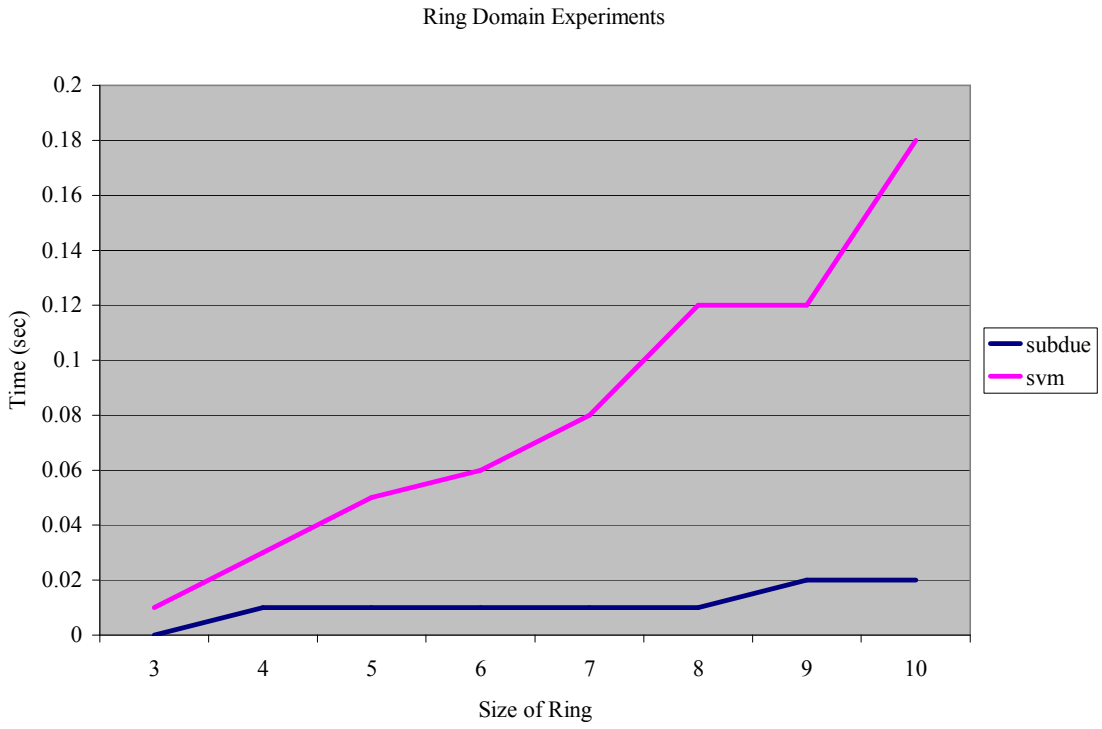


Figure 5.2 Chart - Simple Ring Domain.

Figure 5.2 shows the performance of the systems in learning a simple ring concept. A point to be noted is that the performance of both the systems in terms of training set classification-accuracy has been 100%. We observe that Subdue has performed faster than SVM in learning the simple ring concept.

We will next try to observe the effect of noise in the process of learning a concept. Noise can be introduced in a variety of ways. In our experiments, we will introduce noise as incorrect vertex or edge labels in the concept to be learned or as introduction of unwanted edges in the concept or by varying the number of adjacent nodes.

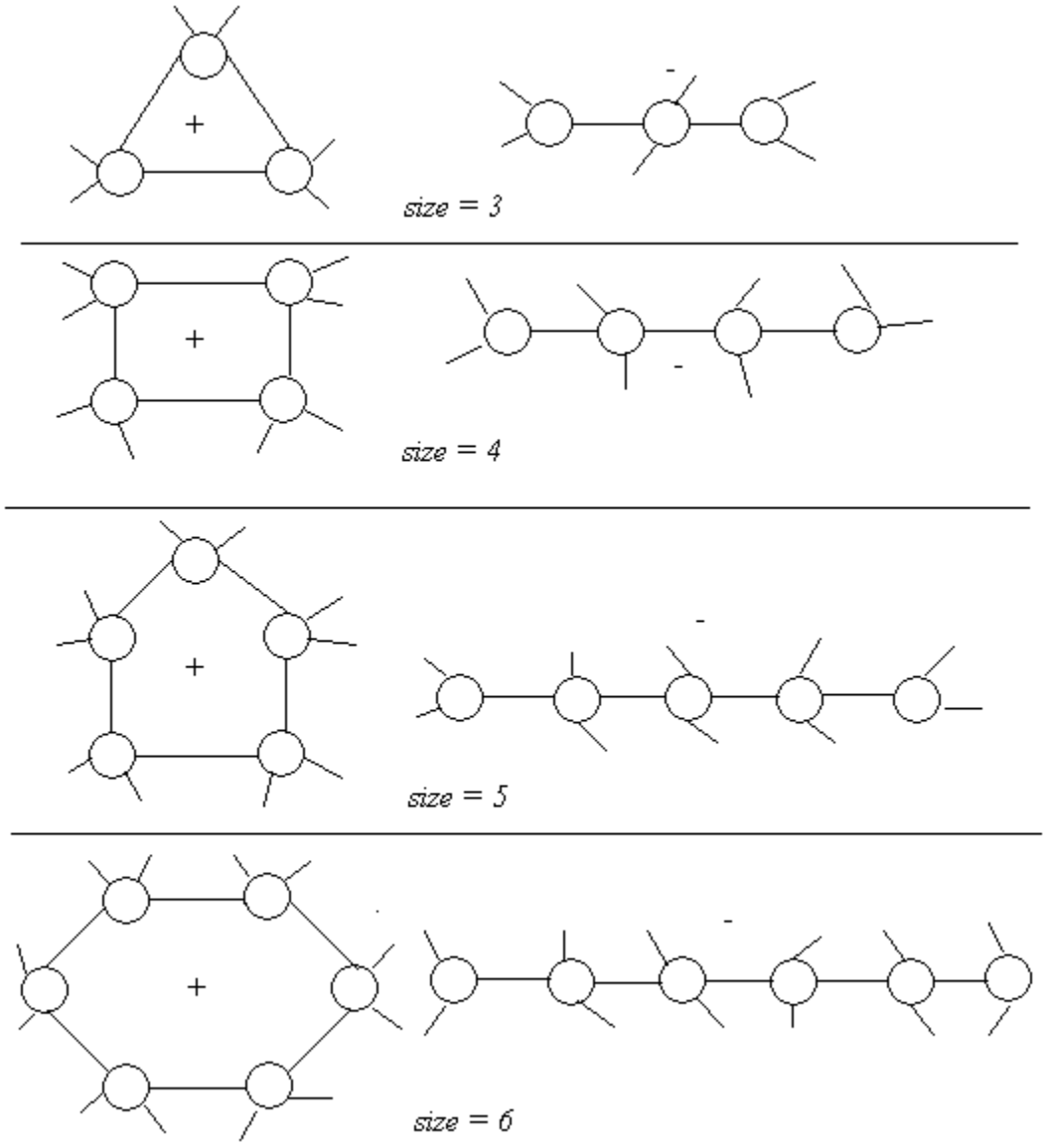


Figure 5.3 Ring Domain Experiment with Noise in Dataset.

Figure 5.3 shows noise being introduced as adjacent irrelevant nodes. We will consider two adjoining nodes, per node of the concept as shown in figure 5.3. The presence of such irrelevant nodes causes the search-based approach to explore the hypothesis space to larger extent.

The parameters set for Subdue are as follows:

1. – eval = 3 (Set cover.)
2. – limit (default)
3. – iterations = 2 (Sufficient to cover all the concepts necessary to be learned.)

The parameters set for SVM are as follows:

1. – t = 4 (Graph kernel.)
2. – u = 1 (default)

Figure 5.4 shows the performance of both search-based and kernel-based approaches in terms of time, maintaining 100% classification accuracy on training set.

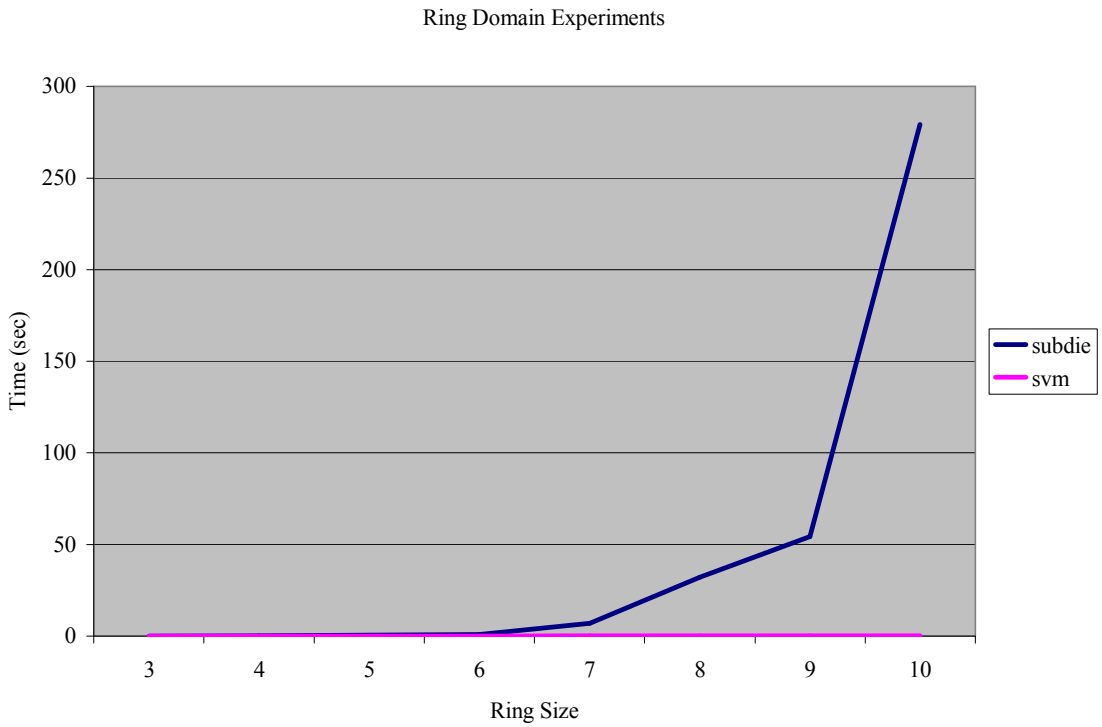


Figure 5.4 Chart - Ring Domain Experiment with Noise in Dataset.

The introduction of noise in the dataset has shown that Subdue spends more time analyzing the irrelevant nodes along with the actual concept in order to learn the concept. SVM on the other hand gives a much better performance. The main reason for the increase in Subdue's time complexity is due to the fact that Subdue expands one vertex and an edge or edge in all directions at a time. As discussed in chapter 3, Subdue expands one vertex and one edge or an edge in all directions of the substructure at a time, till all the positive graphs are covered by the substructure found. On the other hand, SVM with graph-kernel does a simple graph traversal and match. Even in case of the complex ring structure discussed above, the graph kernel has to do a simple tree traversal, which in this case is along nodes of the concept. We can also comment that the number of kernels evaluated by SVM in this case is less, and with fewer kernel evaluations SVM is able to optimize the problem faster.

In the next experiment, we will increase the number of irrelevant adjoining nodes to 3. Figure 5.5 shows the performance of Subdue in case of a simple ring concept, a ring with two irrelevant nodes and a ring with three irrelevant nodes.

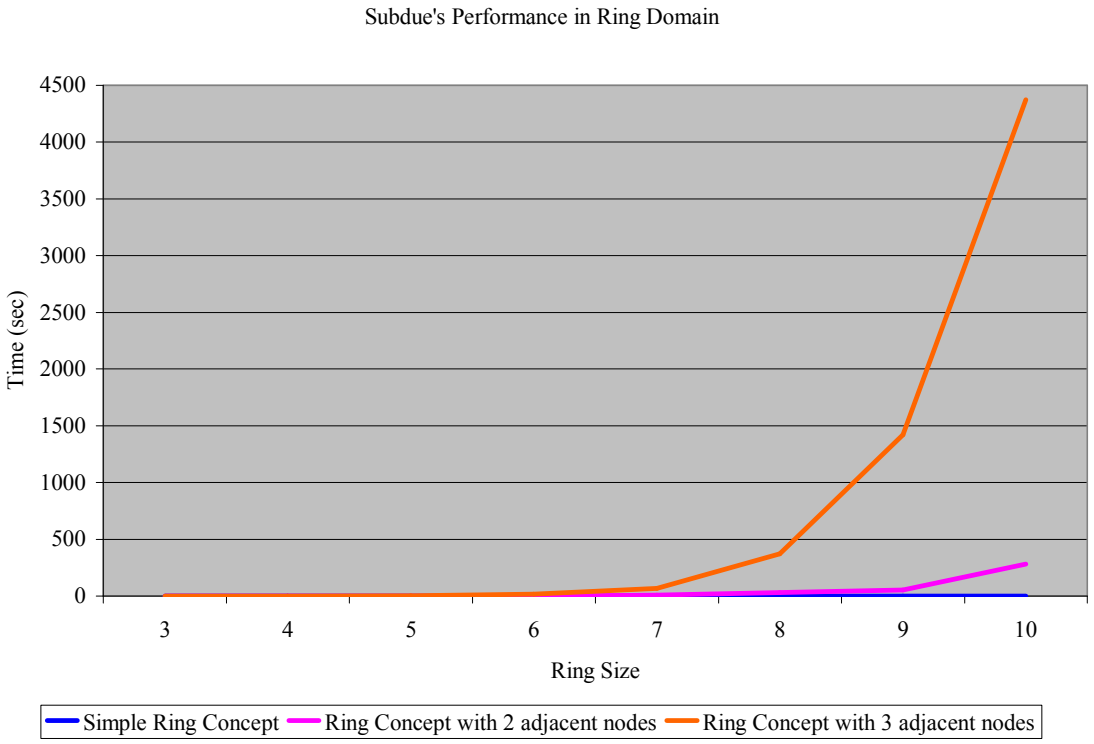


Figure 5.5 Chart - Subdue's Performance in Ring Domain with Varying Complexity.

Figure 5.6 shows the performance of SVM in case of a simple ring concept, ring with two irrelevant nodes and ring with three irrelevant nodes.

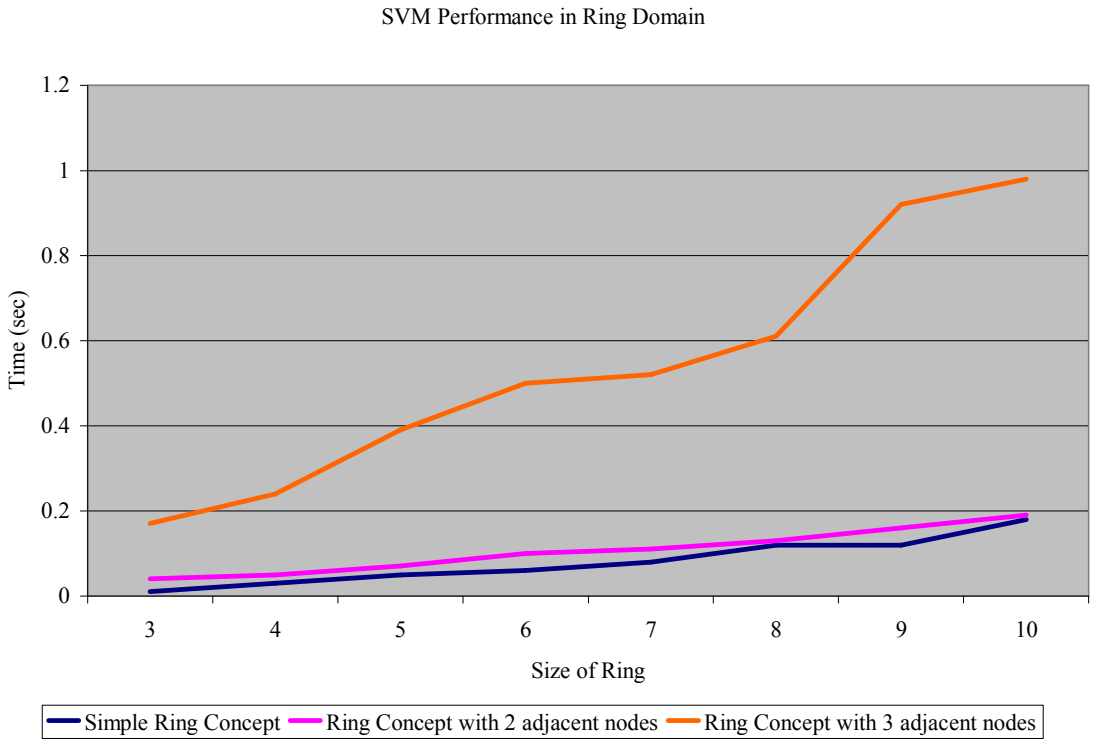


Figure 5.6 Chart - SVM's Performance in Ring Domain with Varying Complexity.

From the above results (figure 5.5 and figure 5.6), we observe that Subdue's time complexity is exponential as the dataset becomes more complex and large, though the concept needed to be learned is the same. SVM however has shown almost linear time complexity in the ring domain. The linearity observed is due to the implementation of the SMO algorithm in the SVM, which solves the QP problem in a more efficient and linear fashion. Refer to chapter 3 for more details on the functioning of SVM and SMO.

In the next set of experiments we introduced noise in the actual concept to be learned. Noise was introduced by changing the vertex/edge labels and by adding/removing edges of the concept. We have considered a dataset consisting of 10 positive and 10 negative examples. We have considered a ring of size N, with five

positive examples having the pure concept while noise is introduced into the remaining 5 positive concepts. Figure 5.7 shows an example setup of such a dataset for a ring of size 5.

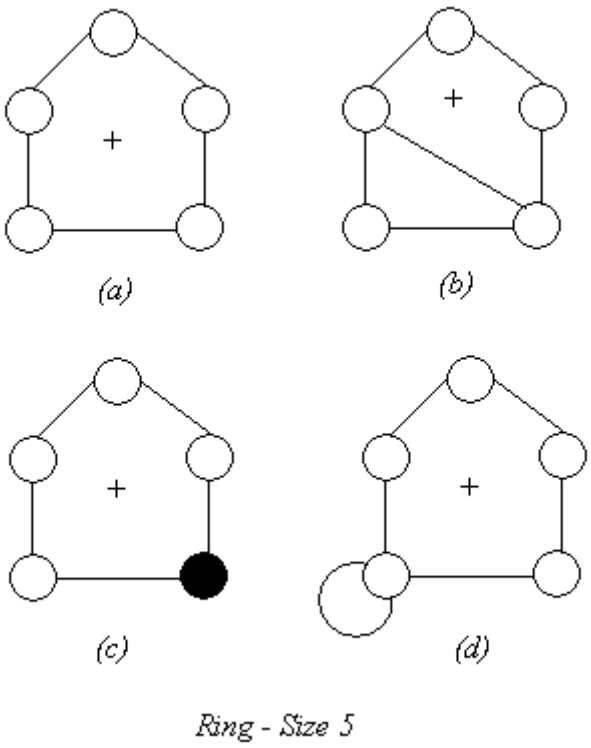


Figure 5.7 Ring Domain Experiment with Noise in Concept. (a) pure ring concept, (b) noise as unwanted edge, (c) noise as unwanted vertex label and (d) noise as self-loop.

Figure 5.7 shows various ways of introducing noise in the concept to be learned. Figure (5.7.a) is the actual concept to be learned. In figure (5.7.b) noise has been introduced as an additional edge in the concept. In figure (5.7.c) noise has been introduced as an incorrect vertex-label. Similarly, in figure (5.7.d), noise has been introduced as a self loop. The dataset was initially generated with 10 positive and 10 negative examples. Noise was then manually introduced as described above. The final

training set consisted of 5 positive examples with pure ring concepts, 5 positive examples with noise in them and remaining 10 negative examples. The introduction of such noise makes it difficult and sometimes impossible for any system to learn the desired concept efficiently.

The parameters set for Subdue are as follows:

1. – eval = 3 (Set cover.)
2. – limit (default)
3. – iterations = 2 (Sufficient to cover all the concepts necessary to be learned.)

The parameters set for SVM are as follows:

1. – t = 4 (Graph kernel.)
2. – u = 1 (default)

Table 5.1 Ring Domain Experiment with Noise in Concept.

size of ring	subdue accuracy (training set)	svm accuracy (training set)
3	55 %	90%
4	55%	95%
5	60%	100%
6	60%	95%
7	55%	95%

From table 5.1, we observe that the performance of Subdue for learning a simple ring concept in the presence of noise has reduced. SVM- graph kernel on the other hand shows comparatively better performance. This is due to the fact that the concepts learned by Subdue from the ring domain with noise, are not the complete ring structure as desired. Hence, the presence of noise in the concept causes Subdue to learn

an incomplete concept reducing its classification efficiency. We experimented with the same dataset with increasing beam-width for Subdue. We observed that in this case Subdue gave the same performance, even though we provided it larger limits to search the hypothesis space. We can thus comment that this is a potential drawback of Subdues representation wherein it fails to learn the desired concept in the presence of noise in the concept.

In the next set of experiments, we have concentrated on the ability of both search-based and kernel-based systems to learn a tree-like concept. As we have implemented a random tree graph kernel in the kernel-based approach, we can predict a better performance of SVM with a random tree graph kernel as compared to Subdue. Figure 5.8 shows a 3-level and a 4-level tree with both positive and negative example. The positive graph is a complete binary tree of level-n. The negative graph is a similar tree with one fewer vertex. Due to this, the only differentiating structure between a positive and negative graph is the tree structure of level-n. Hence the concept to be learned is by minimum a complete binary tree. All the vertices of the tree have label 'V' while the edges are undirected and labeled 'E'.

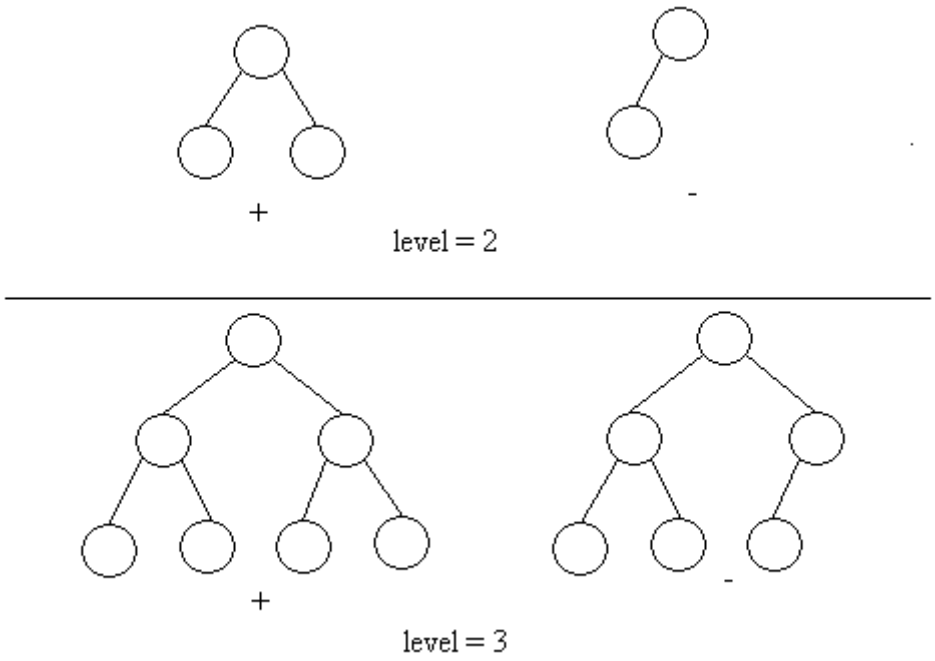


Figure 5.8 Simple Tree Domain.

The parameters set for Subdue are as follows:

1. – eval = 3 (Set cover.)
2. – limit = 100 (Specially for trees with level 4 and above)
3. – iterations = 10 (Sufficient to cover all the concepts necessary to be learned.)

The parameters set for SVM are as follows:

1. – t = 4 (Graph kernel.)
2. – u = 1 (default)

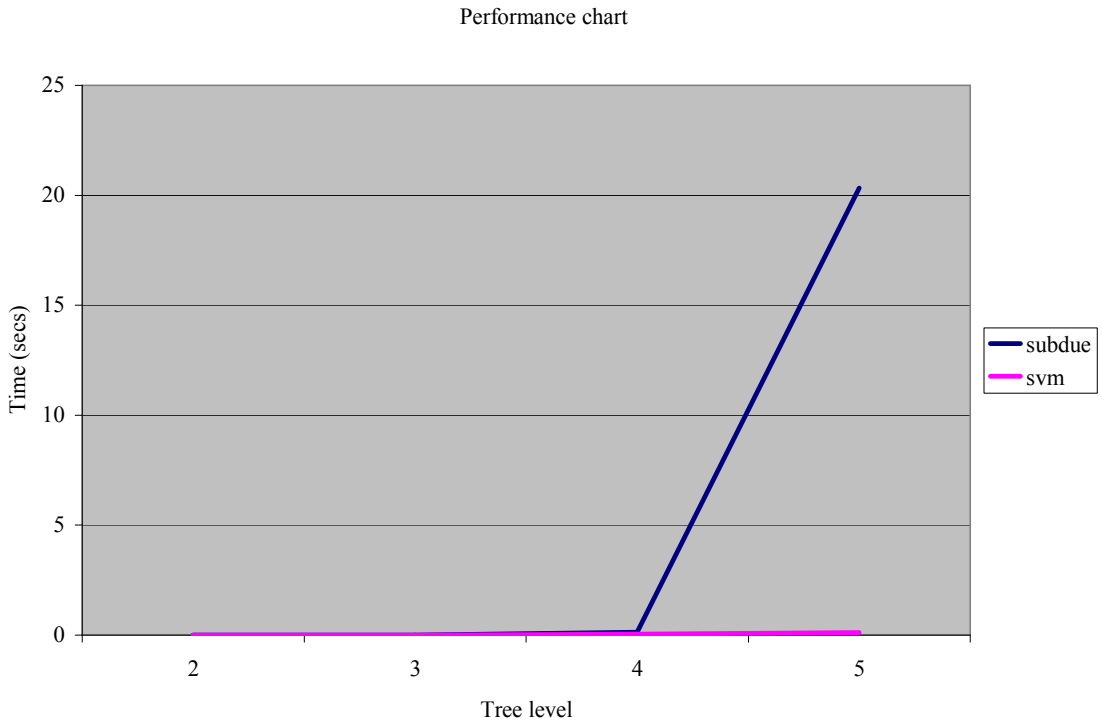


Figure 5.9 Chart - Simple Tree Domain.

Figure 5.9 shows the performance (time complexity) of the systems in learning a simple tree concept. A point to be noted is that the performance of both the systems in terms of training set classification accuracy has been 100%. As we had predicted, SVM outperforms Subdue in terms of speed in learning a tree-like concept. We also observe that Subdue is incapable of learning larger concepts in a comparable time frame with SVM. The larger the concept, the more exploration of the hypothesis space is needed to learn the concept. This is due to the fact that Subdue generates its hypothesis space from the given training examples. As discussed in Subdue's algorithm, the candidate hypotheses are generated by expanding the subgraph by a vertex and edge or an edge in

all possible directions. Thus as the concepts get larger, more and more of the hypothesis space need to be explored to learn the particular concept.

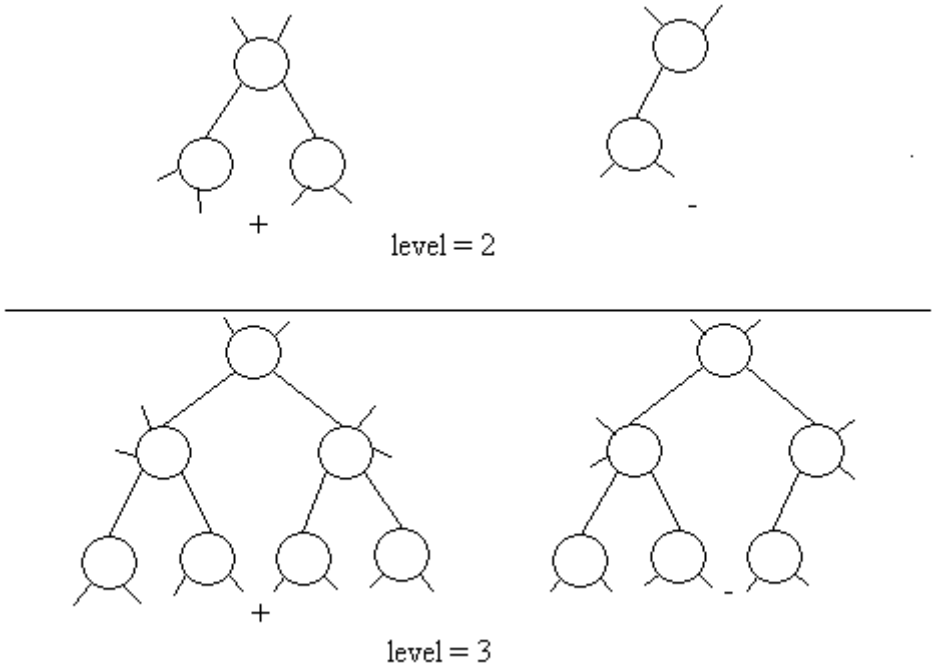


Figure 5.10 Tree Domain Experiment with Noise in Dataset.

Figure 5.10 shows noise being introduced in the tree domain dataset as adjacent irrelevant nodes. We will consider two adjoining nodes, per node of the desired concept, as shown in figure 5.10. The presence of such irrelevant nodes causes the search-based approach to explore the hypothesis space to a larger extent. Figure 5.11 shows the performance of both search-based and kernel-based approach in terms of time, maintaining 100% classification accuracy on the training set.

The parameters set for Subdue are as follows:

1. – eval = 3 (Set cover.)
2. – limit = 100 (Specially for trees with level 4 and above)

3. – iterations = 10 (Sufficient to cover all the concepts necessary to be learned.)

The parameters set for SVM are as follows:

1. – t = 4 (Graph kernel.)
2. – u = 1 (default).

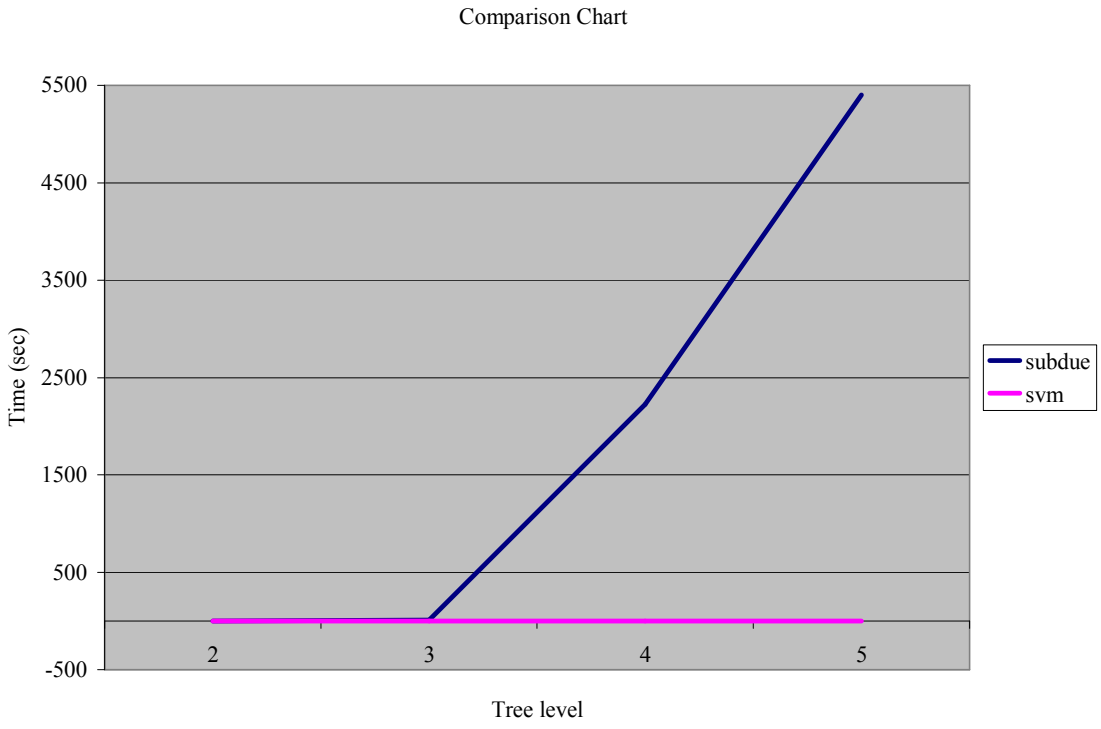


Figure 5.11 Chart - Tree Domain Experiment with Noise in Dataset.

The introduction of noise in the dataset results in Subdue spending more time analyzing the irrelevant nodes along with the actual concept in order to learn the concept. SVM, on the other hand, gives a much better performance. The main reason for such an increase in Subdue's time complexity is due to the fact that Subdue expands one vertex at a time. As discussed in chapter 3, Subdue expands one vertex or one edge

of the substructure at a time, until a substructure is found that covers many positive, but fewer negative example graphs. On the other hand, SVM with graph-kernel does a simple graph traversal and match. Due to the tree nature of the concept to be learned, SVM with the implemented random tree graph kernel is able to traverse and match the given graphs in a much more efficient way than Subdue.

The figure 5.12 shows the performance of Subdue in the case of a simple tree concept, a tree concept with one irrelevant node and a tree concept with two irrelevant nodes.

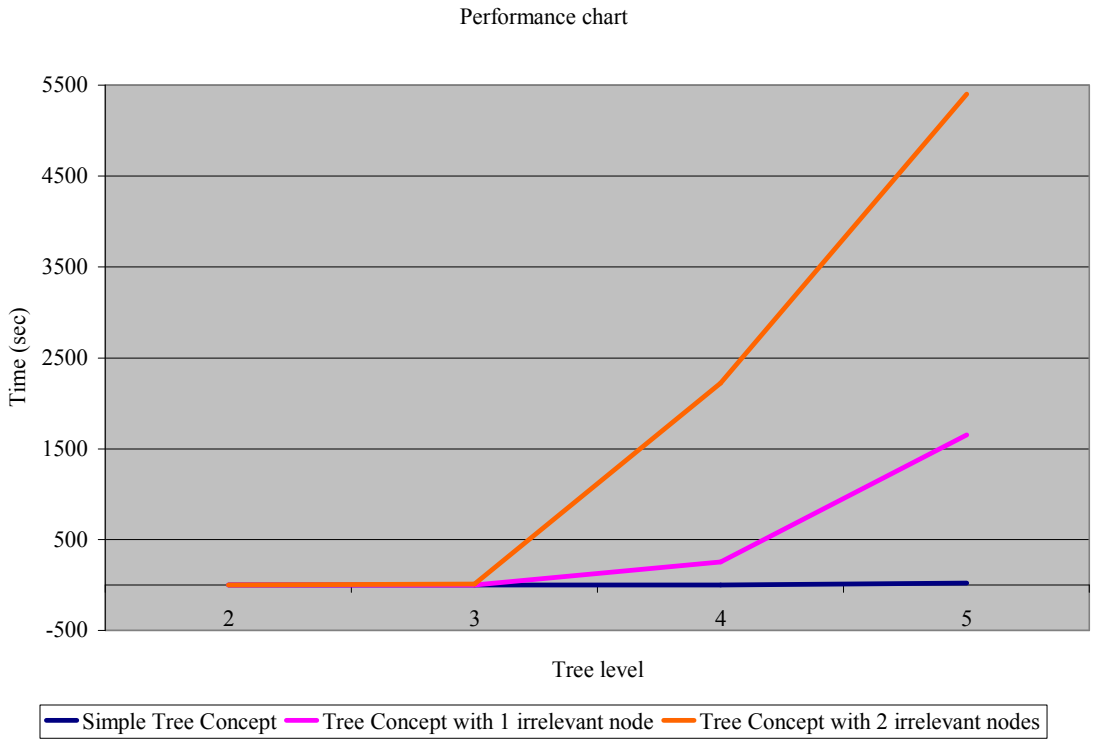


Figure 5.12 Chart - Subdue's Performance in Ring domain with Varying Complexity.

Figure 5.13 shows the performance of SVM in case of a simple tree concept, a tree concept with one irrelevant node and a tree concept with two irrelevant nodes.

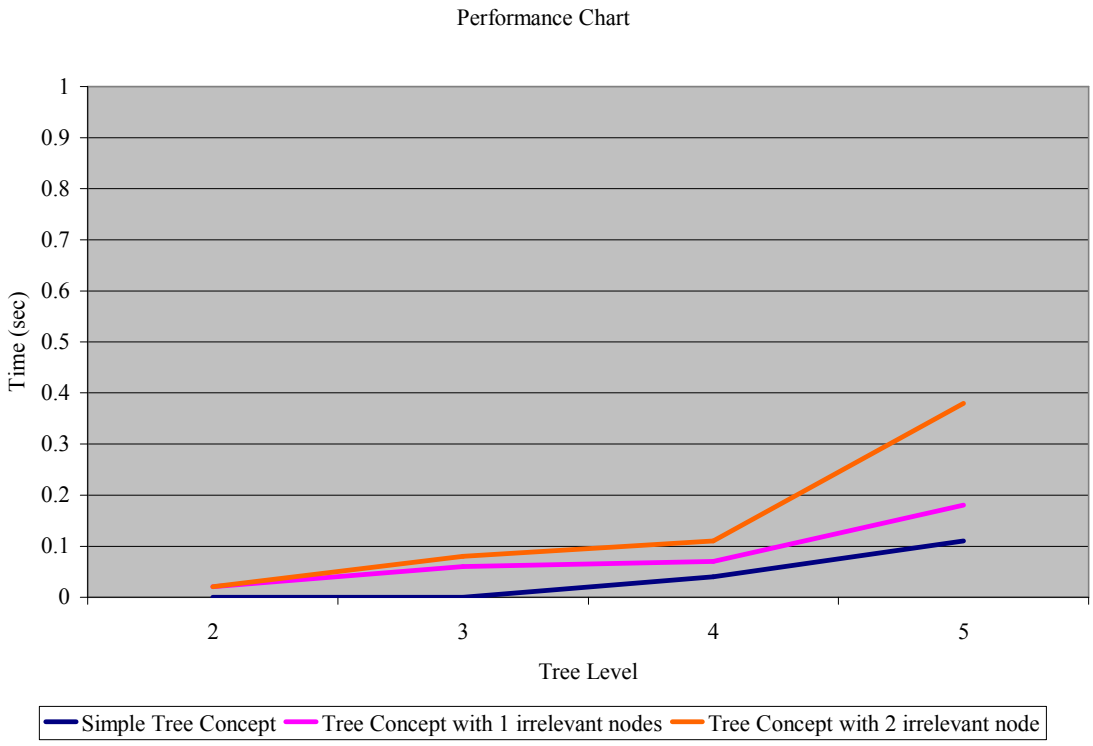


Figure 5.13 Chart - SVM's Performance in Tree domain with Varying Complexity.

From the above results (figure 5.12 and figure 5.13), we observe that Subdue's time complexity increases at a faster rate compared to SVM as the dataset becomes more and more complex, though the concept needed to be learned is the same. The improved complexity observed is due to the implementation of the SMO algorithm in the SVM, which solves the QP problem in a more efficient manner. Refer to Chapter 3 for more details on the functioning of SVM. More information about the working of SMO can be found at [36].

In the next set of experiments we introduced noise in the actual concept to be learned. Noise was introduced by changing the vertex/edge labels and by adding/removing edges of the concept. We have considered a dataset consisting of 10 positive and 10 negative examples. We have considered a tree of level N, with five positive examples having the pure concept while noise is introduced into the remaining 5 positive concepts. The dataset was generated in a similar way as in the case of the ring domain with noise in the concept. Figure 5.14 shows an example setup of such a dataset for a tree of level 3.

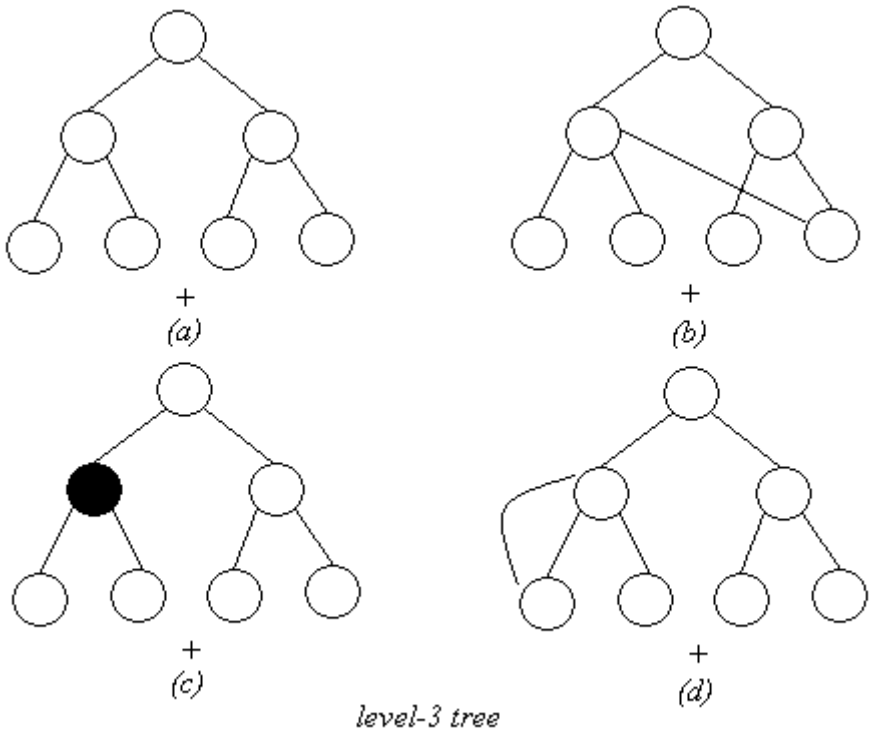


Figure 5.14 Tree Domain Experiment with Noise in Concept. (a) pure tree concept, (b) noise as unwanted edge, (c) noise as unwanted vertex label and (d) noise as double edge.

Figure 5.14 shows various ways of introducing noise in the concept to be learned. Figure 5.14.a is the actual concept to be learned. In figure 5.14.b noise has been introduced as an additional edge in the concept. In figure 5.14.c noise has been introduced as an incorrect vertex-label. Similarly, in figure 5.14.d, noise has been introduced as a double edge. The introduction of such noise makes it difficult and sometimes impossible for the systems to learn the desired concept efficiently.

The parameters set for Subdue are as follows:

1. – eval = 3 (Set cover.)
2. – limit = 100 (Specially for trees with level 4 and above)
3. – iterations = 10

The parameters set for SVM are as follows:

1. – t = 4 (Graph kernel.)
2. – u = 1(default).

Table 5.2 Tree Domain Experiment with Noise in Concept.

Level	subdue accuracy (training set)	svm accuracy (training set)
2	95.00%	100.00%
3	60.00%	95.00%
4	55.00%	95.00%
5	55.00%	50.00%
6	65.00%	100.00%

From table 5.2, we observe that the performance of Subdue for learning a simple tree concept in the presence of noise has reduced. SVM- graph kernel on the other hand shows comparatively better performance. This is due to the fact that the

concepts learned by Subdue from the tree domain with noise is not the complete tree structure as desired. Hence, the presence of noise in the concept causes Subdue to learn an incomplete concept reducing its classification efficiency.

From the above sets of results, we can conclude that Subdue is not capable of learning structurally large concepts in a comparable time frame with SVM. The time complexity of Subdue increases in case of datasets with irrelevant information. Subdue also falls short of learning the actual concepts in the presence of the noise in both datasets and the concept. Noise is introduced in the dataset as the number of irrelevant nodes present around the actual concept to be learned. Due to this, a search-based approach explores a larger hypothesis space taking such noise into consideration to learn the concept. Noise can also be embedded into the concept itself. We have discussed a few ways of embedding noise in the concept in the experiments above. We have also observed that the kernel-based approach is able to give better performance in terms of both time and accuracy as compared to a search-based approach in case of structurally large concepts and datasets involving noise.

5.1.2 Structurally Complex Concepts

We will now try to analyze the performance of both systems in learning structurally complex concepts. We considered a series of completely connected graphs for the same. Our aim here is to see how a search-based approach and kernel-based approach learn a complex concept. We have considered 10 positive and 10 negative examples. The positive example is a complex concept with varying degree of

complexity. The negative examples are subsets of the positive example under consideration, such that the only differentiating factor between a positive and negative example is the complex structure. Figure 5.15 shows a 4 vertex completely connected graph with positive and negative examples.

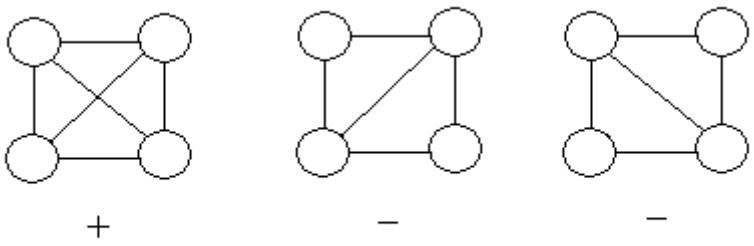


Figure 5.15 Structurally Complex Structures Domain.

The parameters set for Subdue are as follows:

1. – eval = 3 (Set cover.)
2. – limit (default)
3. – iterations = 2 (Sufficient to cover all the concepts necessary to be learned.)

The parameters set for SVM are as follows:

3. – t = 4 (Graph kernel.)
4. – u = 1 (default).

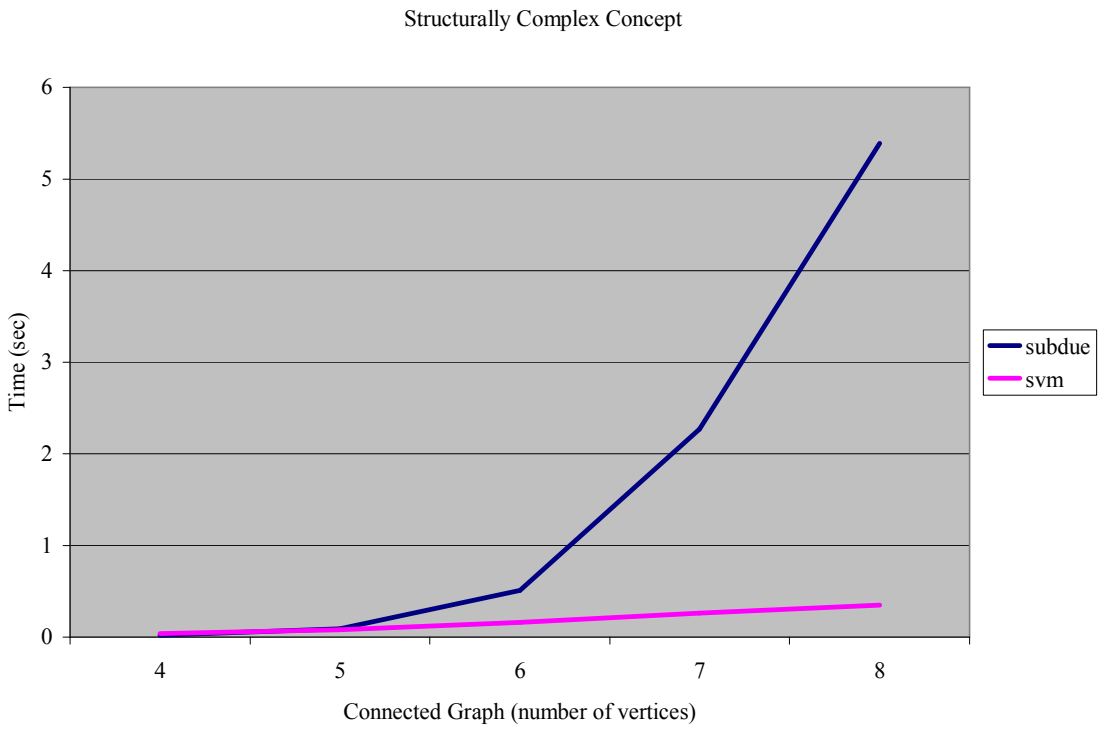


Figure 5.16 Chart - Structurally Complex Structures Domain.

Figure 5.16 shows the performance of the systems in learning structurally-complex concepts. One point to be noted is that the training set accuracy in case of both systems was 100%. We can observe that the search-based approach requires more exploration of the hypothesis space when learning structurally complex concepts. The kernel-based approach on the other hand has shown good performance because SVM with the implemented random tree graph kernel is able to reach all the vertices and edges of the concept in a faster way as compared to Subdue, which expands one vertex or one edge at a time to compute the potential substructure.

From the above results, we can observe that with the increase of structural complexity of a concept to be learned, the exploration of the hypothesis space results in

an increase in the time complexity. We also observe that in the case of Subdue, the time involved in learning a particular concept also depends on the amount of irregularities (noise) in and around the concept. Thus from the above set of experiments on a variety of artificial domains we have observed that Subdue though able to learn larger concepts, is not capable of learning larger concepts in a comparable time frame as SVM. The main reason behind the time complexity of Subdue is the basic Subdue algorithm. As we discussed in chapter 3, Subdue implements a inexact graph match algorithm. As we have already seen, the time complexity of inexact graph match is $O(n^{m+1})$. Thus as the umber of nodes in the graphs increase, the time complexity involved with Subdue increases. SVM on the other hand shows much better time complexity due to the SMO and the kernel implementation in its basic algorithm. As discussed in chapter 3, the SMO approach is used to solve the QP problem in a more efficient and linear fashion. The random tree graph kernel implemented has time complexity $O(n^2)$. These factors result in a faster working of the SVM algorithm.

5.2 Mutagenesis Data Domain

We have discussed the Mutagenesis data in chapter 4, section 4.2. We have considered the graph-based representation of Mutagenesis data as presented by [27]. Initially we consider a dataset consisting of all the related information such as atom type, bond type, element and charge. We will try to explore the idea of how a system is capable of learning a concept given all the information associated with the concept. The inclusion of this additional information makes the dataset structurally large. Figure 4.5

shows a template example of a bond between a carbon and hydrogen atom. Refer to appendix A.1 and appendix A.2 for more details on the graph representation of this dataset used by both Subdue and SVM.

The parameters set for Subdue are as follows:

1. – eval = 3 (Set cover.)
2. – limit = 100
3. – iterations = 25
4. – nfolds = 10 (10-fold cross validation)

The parameters set for SVM are as follows:

1. – t = 4 (Graph kernel.)
2. – u = 1(default).
3. – q = 2

Table 5.3 Mutagenesis Data with Atom, Bond, Atom Type, Bond Type, Element and Charge.

System	Training Set	10-fold cross validation	time (sec)
Subdue	73.36%	61.30%	49380
SVM-Graph Kernel	90.24%	87.16%	1050

From the results in table 5.3 we can observe that SVM has performed considerably better than Subdue. We can also observe that the time complexity in case of Subdue is much larger than SVM. One reason can be the presence of a large amount of irrelevant information around the concept to be learned which causes Subdue to explore a much larger hypothesis space. Another reason can be the fact that the concept to be learned is structurally large or complex. From the substructures learned, we can

see that Subdue performs classification mostly based on the charges involved rather than on molecular structure. Some of the concepts learned by Subdue are listed in appendix B.1.

As we have observed in the case of artificial domains, Subdue has shown a larger exploration of the hypothesis space in search of a particular concept when the dataset included irrelevant information. In the next set of experiments, we will try to reduce some of the information associated with the atoms and bonds to analyze the performance of Subdue and SVM.

In the above experiment we observed that the classification was done based on the partial charges involved with the atoms. In our next experimentation, we will concentrate on learning concepts without the charges involved to see if Subdue is able to learn a better concept for classification. This data representation is similar to the one shown in figure 4.4. The basic difference is the absence of the partial charge associated with each atom.

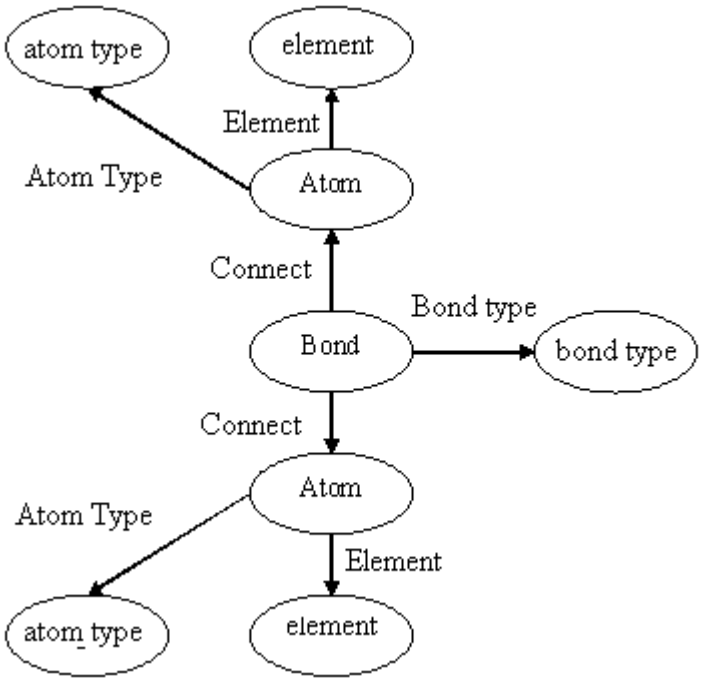


Figure 5.17 Mutagenesis Data – Without Partial Charges.

The parameters set for Subdue are as follows:

1. – eval = 3 (Set cover.)
2. – limit = 100
3. – iterations = 25
4. – nfolds = 10 (10-fold cross validation)

The parameters set for SVM are as follows:

1. – t = 4 (Graph kernel.)
2. – u = 1 (default).
3. – q = 2

Table 5.4 Mutagenesis Data – Without Partial Charges.

System	Training Set	10-fold cross validation	time (sec)
Subdue	63.02%	62. 80%	300
SVM-Graph Kernel	81.47%	78.45%	690

From table 5.4, we once again observe that SVM outperforms Subdue in case of classification accuracy. Subdue, however, has show some improvement in the time complexity and accuracy as compared to the results shown in table 5.3. However from the above result we can see that removal of information, such as the partial charges has not greatly improved Subdues classification accuracy. For SVM we observe that as the amount of information in the dataset is reduced, the time involved for the random tree graph kernel to traverse through the graphs decreases thus reducing the time complexity involved. This supports our results from figure 5.6 and figure 5.13.

In the next dataset under consideration, we will further reduce the amount of information associated with the dataset. This is done by removing all the 'Atom Type' and 'Bond Type' information associated with the atoms and bonds. A basic template representing the dataset is shown in figure 5.18.

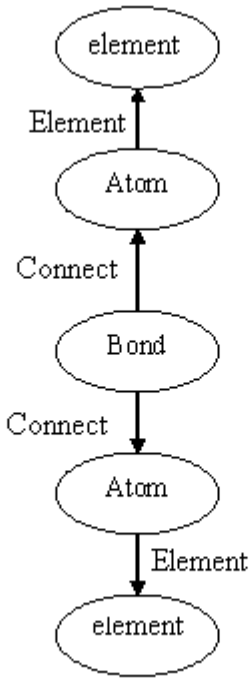


Figure 5.18 Mutagenesis Data – Without Partial Charges, Atom Type and Bond Type.

The parameters set for Subdue are as follows:

1. – eval = 3 (Set cover.)
2. – limit = 100
3. – iterations = 25
4. – nfolds = 10 (10-fold cross validation)

The parameters set for SVM are as follows:

1. – t = 4 (Graph kernel.)
2. – u = 1(default).
3. – q = 2

Table 5.5 Mutagenesis Data without Partial Charges, Atom Type and Bond Type.

System	Training Set	10-fold cross validation	time (sec)
Subdue	63.45%	60.00%	53000
SVM-Graph Kernel	79.75%	76.25%	310

From table 5.5, we can observe that the removal of additional information such as the 'Atom Type' and 'Bond Type' has not made any significant improvement in the performance of both systems. The only change observed in case of Subdue is the increase in the time complexity. This is due to the fact that Subdue in the case of the given dataset has to explore the hypothesis space to a much larger extent in order to find an optimal concept. The larger the search through the hypothesis space the higher is the time complexity. SVM on the other hand has shown a reduction in the time complexity. This supports our observations from figure 5.6 and figure 5.13. The fewer the number of features involved, the less computation involved and hence less time complexity. For this dataset, we observe no significant difference in the classification accuracy for both the systems.

In the next dataset under consideration, we further reduced the amount of information associated with the atoms and bonds in the dataset. A basic template of the dataset is shown in figure 5.19.

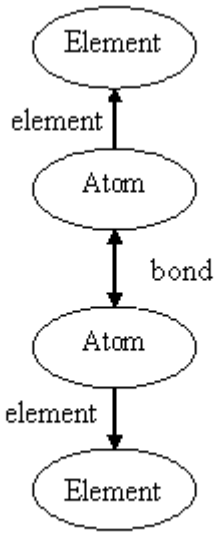


Figure 5.19 Mutagenesis Data- Atom and Elements.

This data representation is similar to the one shown in figure 5.18. The basic difference is the declaration of each element as an atom and introducing a bond as an edge between two atoms.

The parameters set for Subdue are as follows:

1. – eval = 3 (Set cover.)
2. – limit = 100
3. – iterations = 25
4. – nfolds = 10 (10-fold cross validation)

The parameters set for SVM are as follows:

1. – t = 4 (Graph kernel.)
2. – u = 1(default).
3. – q = 2

Table 5.6 Mutagenesis Data with Atoms, Elements and Bonds.

System	Training Set	10-fold cross validation	time (sec)
Subdue	60.85%	60.00%	1350
SVM-Graph Kernel	84.74%	76.52%	140

From table 5.6 we once again observe that neither Subdue or SVM have shown any significant improvement in classification accuracy. However a significant difference is observed in the time complexity of both Subdue and SVM. As we have discussed in the artificial datasets, as the number of features involved in the dataset reduces, the time complexity associated with SVM reduces. In case of Subdue we can comment that Subdue is able to find substructures that cover most of the positive examples and fewer negative examples in much lesser time as compared to the previous dataset.

For the above Mutagenesis datasets we observe in table 5.1 to table 5.6 that SVM has outperformed Subdue in terms of both time and accuracy. We have also observed that Subdue is not able to learn an optimal concept for classification even as we reduced the amount of information in the dataset and hence the classification accuracy is low compared to SVM.

To enhance the performance of the search-based approach, we will try to guide the search operation by explicitly introducing background knowledge. In the next set of experiments, we will try to analyze the effect of background knowledge on the performance of both Subdue and SVM.

Figure 5.20 shows a template that represents the introduction of a 'HUB' node to help learning systems learn disconnected concepts. It might happen that the complexity of the concept and its layout in the dataset is such that, it becomes computational costly for a learning system to learn the complete concept. By introducing the 'HUB' node we will try to guide the learning systems in reaching far away nodes with ease.

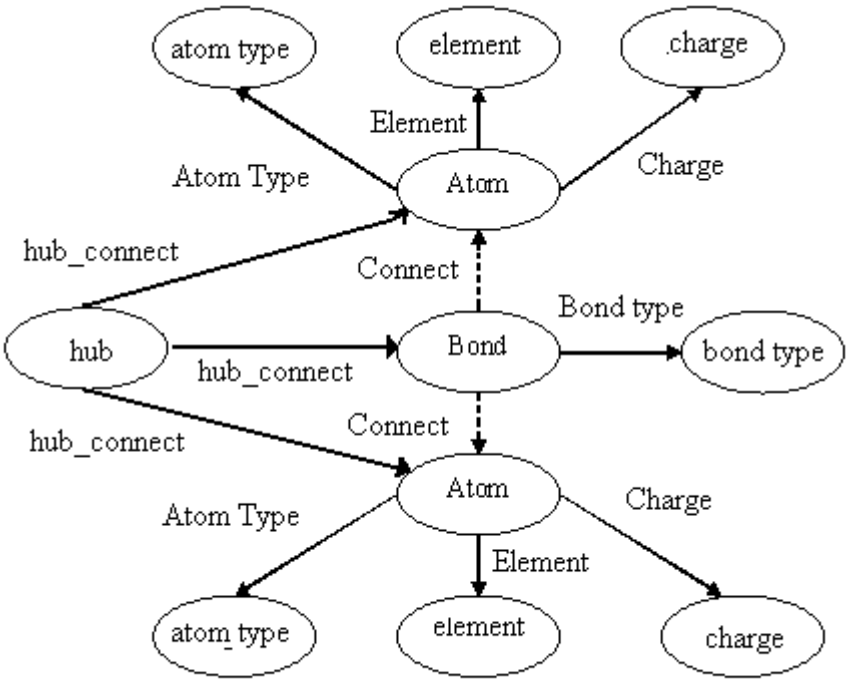


Figure 5.20 'Hub' Node as Background Knowledge.

The parameters set for Subdue are as follows:

1. – eval = 3 (Set cover.)
2. – limit = 100
3. – iterations = 25
4. – nfolds = 10 (10-fold cross validation)

The parameters set for SVM are as follows:

1. – t = 4 (Graph kernel.)
2. – u = 1(default).
3. – q = 2

Table 5.7 'Hub' Node as Background Knowledge.

System	Training Set	10-fold cross validation	time (sec)
Subdue	75.56%	63.48%	127
SVM-Graph Kernel	88.31%	77.83%	850

From table 5.7 we observe that the introduction of the 'HUB' node has helped in reducing the time complexity associated with Subdue as compared to the above results. SVM on the other hand has shown no such improvement in its performance. However, in this case SVM still outperforms Subdue in classification accuracy.

The next Mutagenesis dataset is a variation of the above dataset (figure 5.20), wherein background knowledge is introduced in the form a 'HUB' node along with 'Ia' and 'I1' nodes. These 'Ia' and 'I1' nodes are indicator variables [27]. They indicate the presence of features which are important in distinguishing mutagenesis compounds from non-mutagenesis compounds. The 'Ia' node indicates the presence of

acenthrylenes while ‘I1’ node indicates the presence of three or more fused rings. Figure 5.21 shows a schematic representation of the dataset.

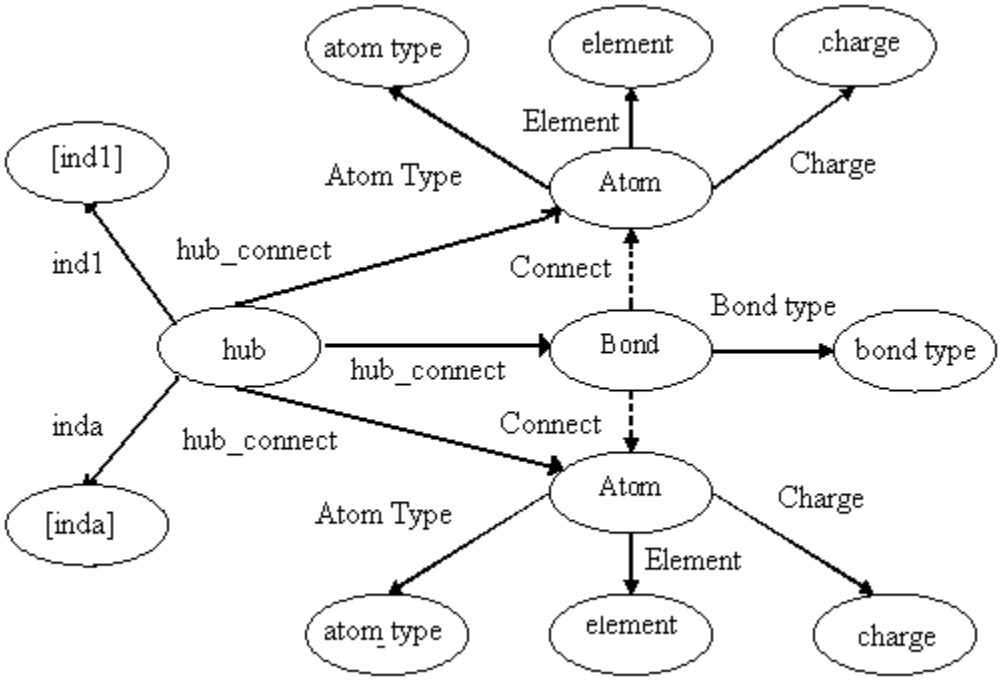


Figure 5.21 ‘Hub’ Node as Background Knowledge with ‘Ia’ and ‘I1’ Nodes.

The parameters set for Subdue are as follows:

1. – eval = 3 (Set cover.)
2. – limit = 100
3. – iterations = 25
4. – nfolds = 10 (10-fold cross validation)

The parameters set for SVM are as follows:

1. – t = 4 (Graph kernel.)
2. – u = 1(default).
3. – q = 2

Table 5.8 ‘Hub’ node as Background Knowledge with ‘Ia’ and ‘I1’ Nodes.

System	Training Set	10-fold cross validation	time (sec)
Subdue	79.00%	78.69%	1100
SVM-Graph Kernel	80.00%	77.34%	1005

From the table 5.8, we observe that Subdue has obtained a performance boost in terms of both time and classification accuracy as compared to previous results. Thus we can comment on the fact that, the introduction of the ‘hub’ node as background knowledge with ‘Ia’ and ‘I1’ nodes have resulted in learning of a more accurate concept for classification.

With the success of the ‘hub’ node as introduced background knowledge, let us now try to introduce background knowledge in the form of complete chemical compounds. Background knowledge in this sense will guide the learning systems to understand certain chemical compounds in a better fashion. Figure 5.22 shows the schematic representation of such background knowledge in the dataset. We will use this representation with ‘benzene’ as the concept. Hence we introduce node ‘benzene’ that connects to all the atoms that form the benzene compound. The main functionality of ‘compound’ is to guide the systems to learn a particular concept directly. Our aim is to

see if the systems taken into consideration are able to make use of the embedded background knowledge to improve classification performance.

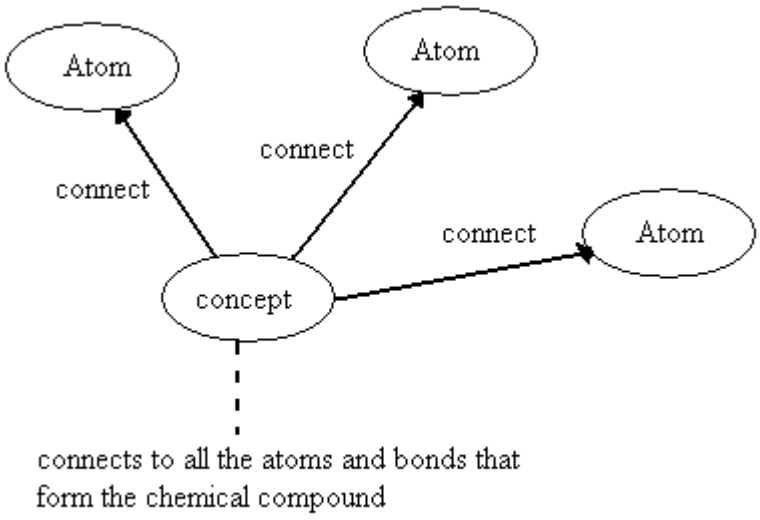


Figure 5.22 'Concept' Node as Background Knowledge.

The compound in this case is 'benzene', which connect to the surrounding six carbon atoms that form the benzene ring as shown in figure 5.23.

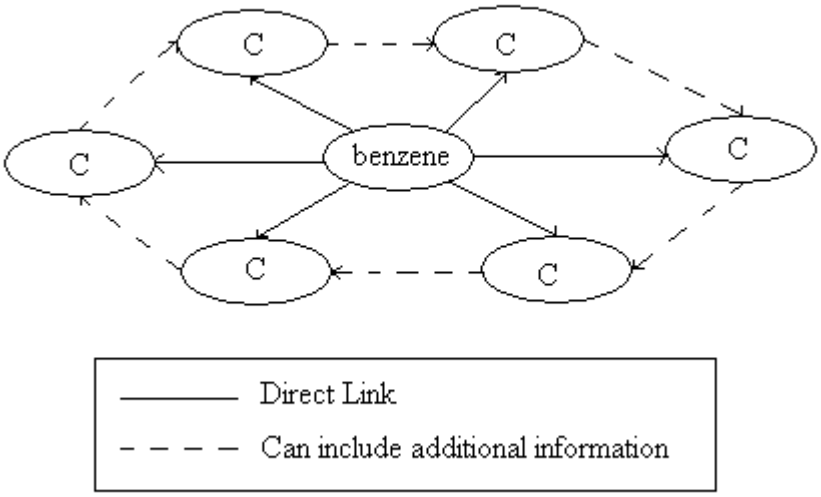


Figure 5.23 ‘Benzene’ as Embedded ‘concept’.

The parameters set for Subdue are as follows:

1. – eval = 3 (Set cover.)
2. – limit = 100
3. – iterations = 25
4. – nfolds = 10 (10-fold cross validation)

The parameters set for SVM are as follows:

1. – t = 4 (Graph kernel.)
2. – u = 1(default).
3. – q = 2

Table 5.9 'Concept' Node as Background Knowledge.

System	Training Set	10-fold cross validation	time (sec)
Subdue	62.10%	60.88%	1480
SVM-Graph Kernel	81.45%	75.65%	270

From table 5.9, we observe that by introducing the background knowledge in terms of a compound concept has not improved the performance of Subdue. SVM has maintained a similar performance as before.

From the above experiments we observe that Subdue is unable to learn molecular structure in close proximity to those discussed in figure 4.2. In the next dataset, we will concentrate on the capability of the systems to learn a well-defined molecular structure as in figure 4.2. We will hence consider a Mutagenesis domain that consists of just atoms and bonds. This dataset was provided by Akihiro Inokuchi, Tokyo Research Laboratory IBM Japan. The graph representation of this dataset is shown in figure 4.2. The Mutagenesis data under consideration is a basic dataset with atoms and bonds. Each atom is a carbon C, hydrogen H, nitrogen N, chlorine CL, or oxygen O.

The parameters set for Subdue are as follows:

1. – eval = 3 (Set cover.)
2. – limit = 100
3. – iterations = 25
4. – nfolds = 10 (10-fold cross validation)

The parameters set for SVM are as follows:

1. – t = 4 (Graph kernel.)
2. – u = 1(default).
3. – q = 2

Table 5.10 Mutagenesis Data with Atoms and Bonds.

System	Training Set	10-fold cross validation	time (sec)
Subdue	69.30%	68.95%	1579
SVM-Graph Kernel	93.13%	81.40%	17

From table 5.10, we observe that SVM outperforms Subdue in terms of both time and accuracy. We will now try to analyze some concepts learned by Subdue which are shown in figure 5.24.

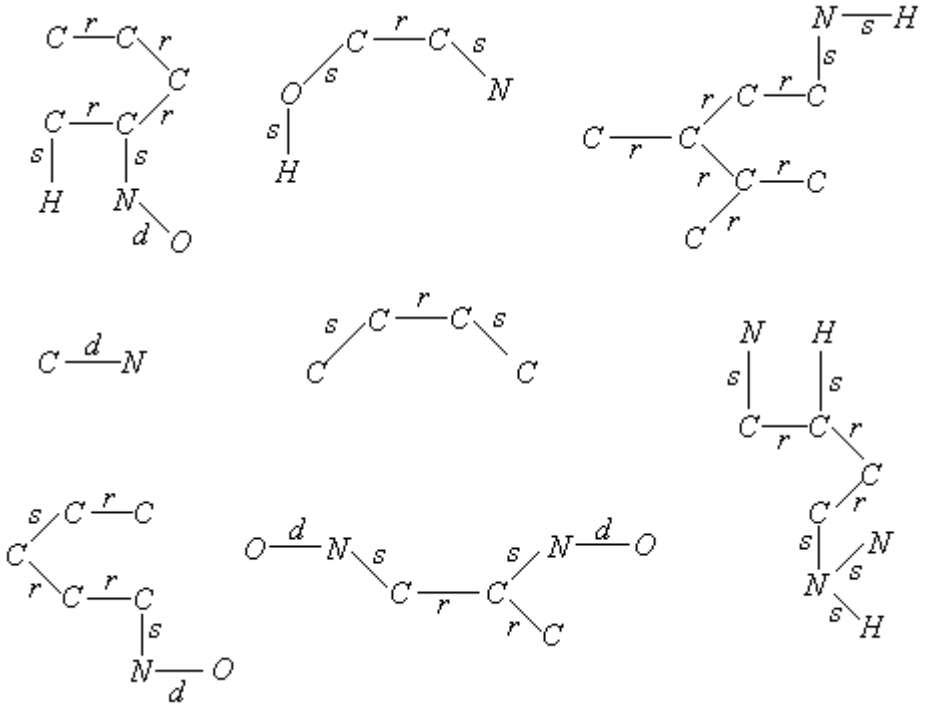


Figure 5.24 Molecular Structures Learned.

From figure 5.24, we notice that Subdue is not able to learn any of the molecular structures as described in figure 4.2. However, the concepts learned are parts of the various molecular structures shown in figure 4.2. The performance of SVM indicates that the random tree graph kernel is successful in traversing the given datasets and calculating cost optimal enough to generate a good hyperplane. As the number of nodes (features) in the dataset under consideration is less, the dimension of the feature space will be less, thus reducing the time complexity involved. A fewer number of features indicates less computation involved.

We experimented with the Mutagenesis data from various perspectives. We initially tried to learn a concept from a dataset consisting of all the information associated with the atoms and bonds. In this case, we observed that Subdue failed to learn an accurate concept as the classification done was purely based on the charges associated with the atoms. SVM on the other hand showed comparatively better performance in terms of time and accuracy. We then considered experimentation with reduction in the information associated with the atoms and bonds, trying to learn a more accurate concept that falls in a close proximity to the various mutagenesis compounds as shown in figure 4.2. In these experiments we observe that Subdue did not show much improvement in its performance as compared to SVM. A few reasons that support these results can be obtained from the artificial domain experiments.

In the artificial domain experiments, we were able to show that Subdue was incapable of learning larger concepts in a comparable time frame as SVM. Though

Subdue was able to learn larger concepts, its time complexity was dependent on the amount of irrelevant information associated with the concept to be learned. More the number of irrelevant information in the dataset, the more was the exploration of the hypothesis space. We also observed that Subdue's performance degraded as compared to SVM, while learning concepts and datasets involving noise.

In our experimentation with datasets with background knowledge for the mutagenesis data, we observed that Subdue was incapable of making the optimal use of background knowledge in various forms. Subdue was not able to improve its performance when background knowledge was introduced in the form of learning a chemical compound directly. However, background knowledge in term of a 'Hub' node, which connected to all the atoms and bonds of the data, provided a significant performance improvement for Subdue. This can also mean that background knowledge in form of connecting 'HUB' node can help learn disconnected concepts.

To summarize this section, we have seen that SVM has outperformed Subdue in terms of time and accuracy in most of the cases.

5.3 Artificial Domain Experiments

Our next aim is to look more deeply into artificial datasets, trying to find various scenarios in which each of the systems will be able to outperform each other. To understand how a 'HUB' node was able to help Subdue improve its performance, we will take into consideration one scenario wherein the concept to be learned is disconnected. Figure 5.25 shows an example of such a dataset taken into consideration

without background knowledge. All the nodes except the concept nodes are labeled ‘V’ and all the edges are labeled ‘E’.

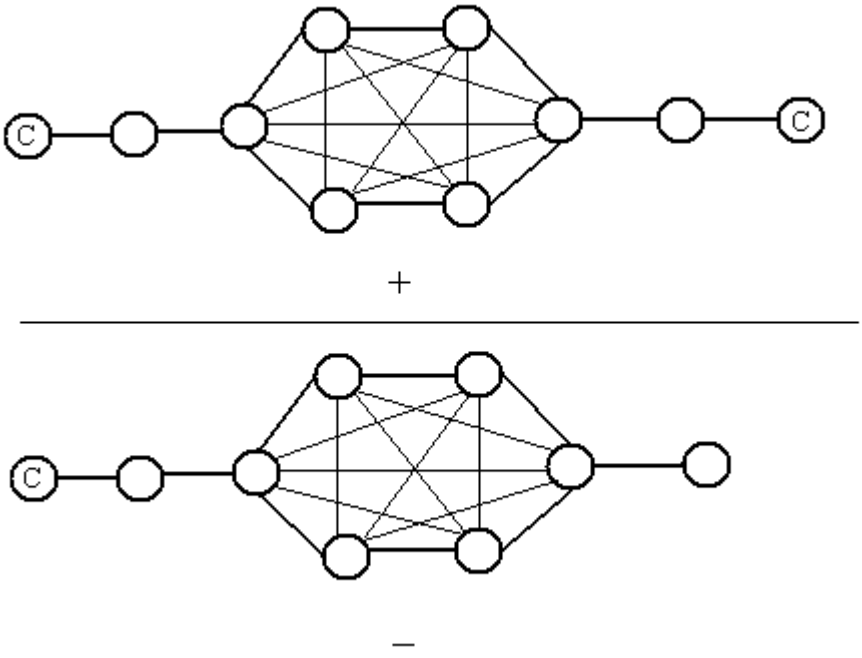


Figure 5.25 Disconnected Concepts Domain.

Our aim is to learn the disconnected concept C. Note that the node C represents a part of the actual disconnected concept to be learned. Our aim is to observe how a ‘HUB’ node can be used to guide the system in reaching the disconnected concept efficiently. We considered 10 positive and 10 negative training examples. Figure 5.26 shows an example of the dataset with ‘HUB’ as background knowledge. All the nodes except the concept nodes are labeled ‘V’ and all the edges are labeled ‘E’. The nodes colored black indicate the knowledge about the concept to be learned. Let us assume

that the vertices v1 and v2 are part of the disconnected concept to be learned. They imply the background knowledge we have about the concept to be learned. They can also be a part of the remaining graph, as shown by vertices v3 and v4. We will introduce the 'HUB' node over these vertices to guide the systems in learning the disconnected concept.

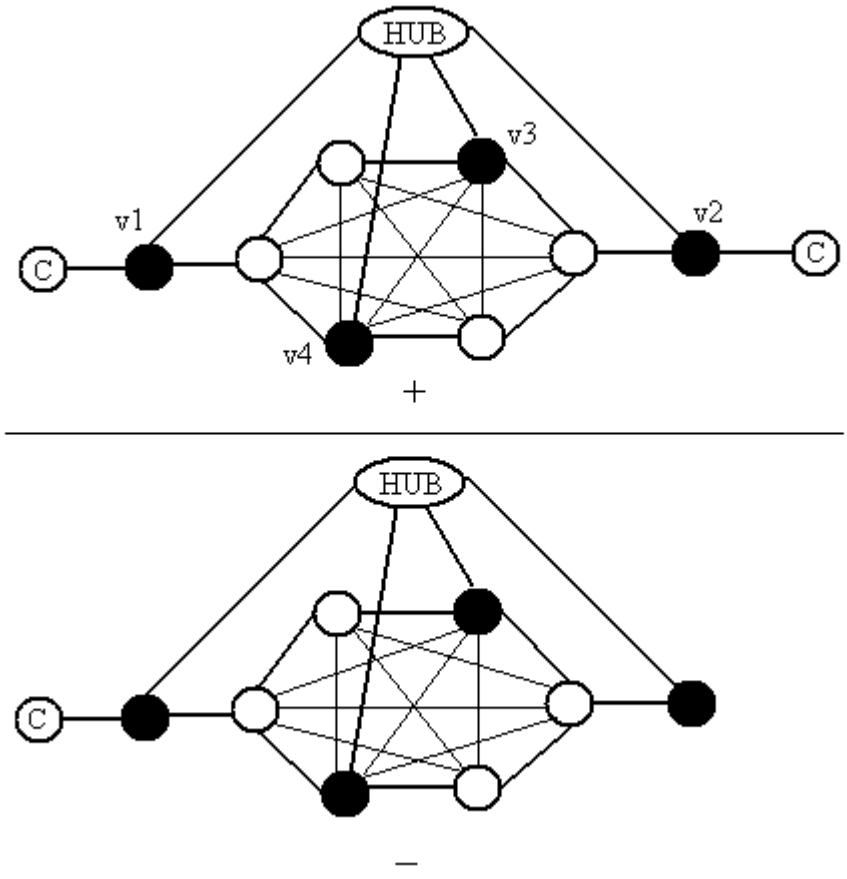


Figure 5.26 Disconnected Concepts Domain with 'HUB' Node as Background Knowledge.

The parameters set for Subdue are as follows:

1. – eval = 3 (Set cover.)
2. – limit (default)
3. – iterations = 2.

The parameters set for SVM are as follows:

1. – t = 4 (Graph kernel.)
2. – u = 1(default).

Table 5.11 Disconnected Concept Domain with 'HUB' Node.

Background knowledge (HUB Node)	Subdue accuracy (training set)	subdue time* (sec)	SVM accuracy (training set)	svm time* (sec)
NO	100.00%	300.2	100.00%	0.37
YES	100.00%	63.84	100.00%	0.43

(* - Training time + Testing time)

Figure 5.27 and figure 5.28 show the concepts learned by Subdue in the case of above datasets with and without the 'HUB' node as background knowledge respectively.

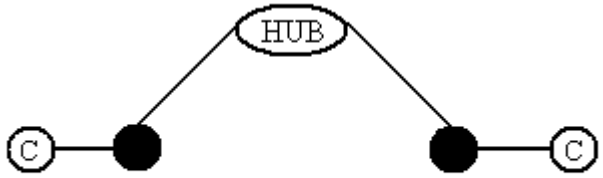


Figure 5.27 Concept learned with 'HUB' Node as Background Knowledge.

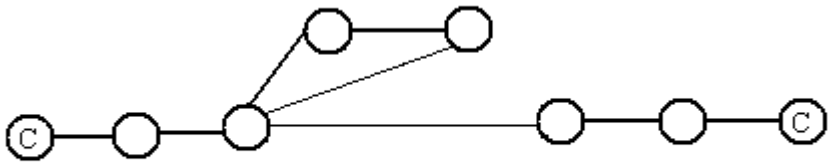


Figure 5.28: Concept learned without 'HUB' Node as Background Knowledge.

From figure 5.27 and figure 5.28 we observe that the concept learned by Subdue is large in the case of the dataset with the disconnected concept and no background knowledge as compared to dataset with disconnected concept and 'HUB' node as background knowledge. From table 5.11, we observe that the training and testing time involved in the case of the dataset without the 'HUB' node is more than in the case of dataset with the 'HUB' node. This implies that a larger hypothesis space is explored in order to learn a larger concept as shown in figure 5.28. The introduction of the 'HUB' node helps Subdue in searching for the optimal concept with less exploration of the hypothesis space. Thus we can conclude that, in the case of learning a disconnected concept, background knowledge in the form of a *guiding* 'HUB' node can improve Subdue's time efficiency. One overhead of introducing background knowledge in this form is the preprocessing of the input dataset. An explicit knowledge of the concept to be learned is required before introducing the 'HUB' node.

We will now analyze the performance of both systems in domains containing negations of concepts. To begin with, let us consider the dataset wherein the positive examples contain concepts that are a negation of the concepts in the negative examples

as shown in figure 5.29. In this dataset we are basically trying to learn ¬(square concept AND triangle concept) which is equivalent in learning ¬(square concept) OR ¬(triangle concept). We know that Subdue is incapable of learning concepts from such a scenario, as the concepts to be learned are in both positive and negative examples. We will try to observe how SVM performs in a situation.

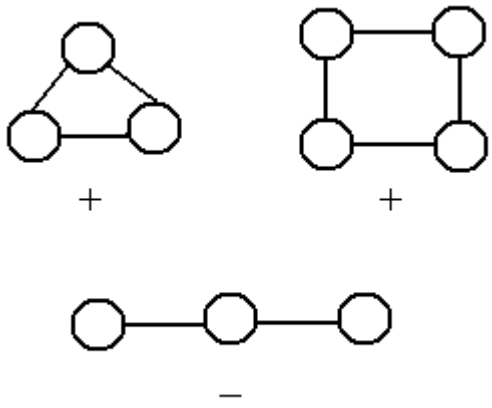


Figure 5.29 Dataset with Negated Concepts.

We have considered 10 positive and 10 negative examples in this dataset. From the 10 positive examples, 5 contained the concept triangle while the remaining 5 contained the square concept. We will try to analyze the performance of both search-based and kernel-based approach in such a scenario.

The parameters set for Subdue are as follows:

1. – eval = 3 (Set cover.)
2. – limit (default)
3. – iterations = 2.

The parameters set for SVM are as follows:

1. – t = 4 (Graph kernel.)
2. – u = 1(default).

Table 5.12 Dataset with Negated Concepts.

systems	training set accuracy	tIME (SEC)
Subdue	50.00%	0.01
SVM	100.00%	0.05

Figure 5.30 Concepts Learned.

From figure 5.30 we can observe that as predicted Subdue is incapable of learning the triangle concept or the square concept. Figure 5.30 indicates that Subdue is capable of distinguishing the positive and negative examples only by just one vertex. This supports our results in table 5.12, where we observe that Subdue is not able to do a good classification. SVM on the other hand gives a 100% training set classification accuracy.

We further modify the testing dataset by removing all the square concepts. The new testing dataset contains 10 positive examples with the triangle concept and 10 negative examples with both triangle and square concept as shown in figure 5.29.

Table 5.13 Positive Examples – Triangle Concepts & Negative Examples – Triangle Concept AND Square Concept.

systems	training set accuracy	tIME (SEC)
Subdue	50.00%	0.01
SVM	100.00%	0.05

Similarly, we modify the testing dataset, by removing all the triangle concepts. The new testing dataset contains 10 positive examples with the square concept and 10 negative examples with both triangle and square concept as shown in figure 5.29.

Table 5.14 Positive Examples – Square Concepts & Negative Examples – Triangle Concept AND Square Concept.

systems	training set accuracy	tIME (SEC)
Subdue	50.00%	0.01
SVM	100.00%	0.05

From table 5.13 and table 5.14, we observe that SVM is able to maintain a 100% accuracy as compared to Subdue which is incapable of learning in a scenario wherein the concepts to be learned are negations. However, the 100% classification accuracy by SVM indicates that it is unaffected by the presence of such negated concepts. This supports our claim that SVM does not learn any specific concept. All that matters is the match between the given graphs which can result in a mapping.

We will now try to look into a dataset opposite to the one used in the previous experiment. We consider 10 positive example consisting of 'the square concept and the pentagon concept' and the negative examples consisting of 5 examples with the square concept and 5 examples with the pentagon concept. In this dataset we are basically

trying to learn (square concept AND pentagon concept). Figure 5.31 shows the setup of the dataset.

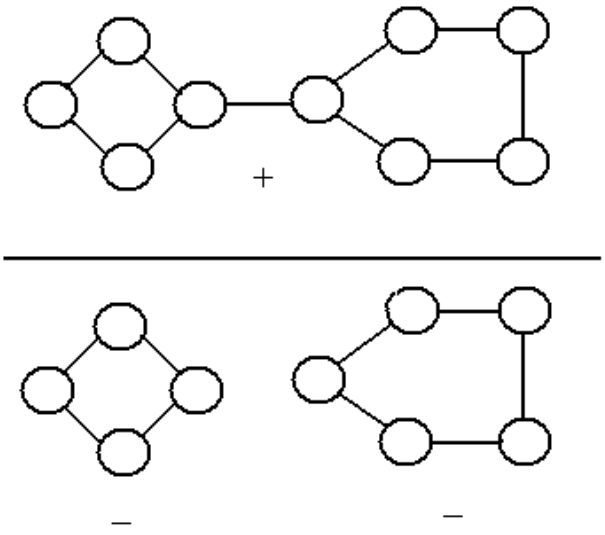


Figure 5.31 Dataset with ANDed Concepts.

The parameters set for Subdue are as follows:

1. – eval = 3 (Set cover.)
2. – limit (default)
3. – iterations = 2.

The parameters set for SVM are as follows:

1. – t = 4 (Graph kernel.)
2. – u = 1(default).

Table 5.15 Dataset with ANDed Concepts.

systems	training set accuracy	tIME (SEC)
Subdue	100.00%	0.05
SVM	100.00%	0.07

From table 5.15, we can observe that both systems are capable giving good classification. The substructures learned by Subdue are shown in figure 5.32.

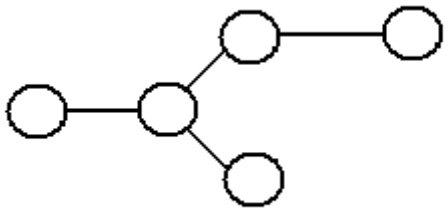


Figure 5.32 Concepts Learned.

Both Subdue and SVM are able to give good performance in classifying the given dataset. Subdue is able to learn the optimal concept efficiently thus giving good classification performance. In this section we can thus conclude that, Subdue has shown poor performance in case of datasets consisting of learning negated concepts where SVM have proved its efficiency.

In our next experiment, we will look into the clique problem. The dataset consists of 10 positive examples with a graph with 6 vertices and containing a maximum clique of size 4 and negative examples consisting of a partial clique. Figure 5.33 shows the setup of the dataset.

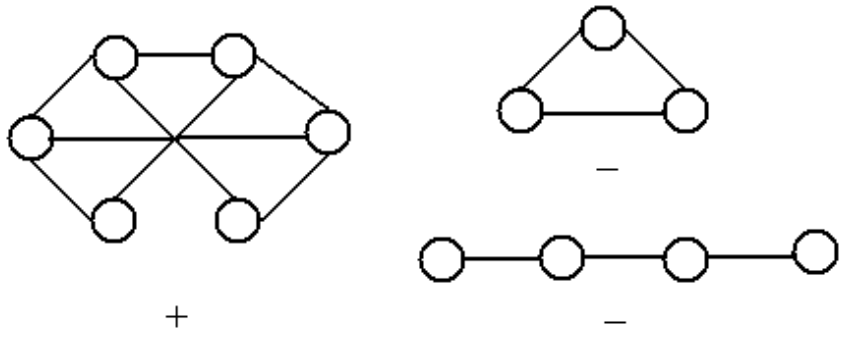


Figure 5.33 Clique Problem.

Table 5.16 Clique Problem

systems	training set accuracy	tIME (SEC)
Subdue	100.00%	0.02
SVM	100.00%	0.07

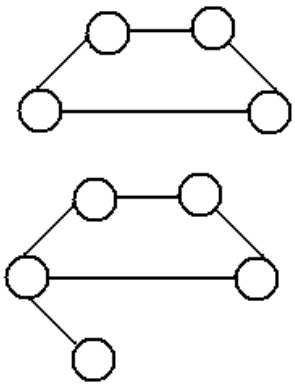


Figure 5.34 Substructures Learned.

The concepts learned by Subdue are shown in figure 5.34. From table 5.16 and figure 5.34 we observe that Subdue is able to learn and classify the clique problem correctly. SVM too has been able to show good performance in the clique problem domain.

5.4 Graph Kernel – Inexact Graph Match Kernel

In the follow experiment we will consider the possibility of using Subdue's inexact graph matcher as a kernel function. As discussed in chapter 3, Inexact Graph Match tries to compute the cost involved in transforming a given graph G_1 into graph G_2. Thus the Inexact Graph Match computes a cost based on the similarity of the input graphs G_1 and G_2. The cost is less if G_1 is more similar to G_2. If G_1 and G_2 are the same, then the cost is zero. Similarly, in the worst case, if G_1 and G_2 are completely different, then the maximum cost involved can be given as –

$$\text{Cost} = |V_1| + |E_1| + |V_2| + |E_2|$$

Where, $|V_1|$ = number of vertices in G_1.

$|E_1|$ = number of edges in G_1.

$|V_2|$ = number of vertices in G_2.

$|E_2|$ = number of edges in G_2.

This property of Inexact Graph Match makes it a potential candidate for a graph kernel. As discussed in chapter 3, the inexact graph match is continuous in the range $[0, |V_1| + |E_1| + |V_2| + |E_2|]$. Also for a given pair of graphs G_1 and G_2, $K(G_1, G_2) = K(G_2, G_1)$. The inexact graph match satisfies the property of being positive definite. We ran Support Vector Machine with Inexact Graph Match as a graph kernel on both artificial datasets and the Mutagenesis datasets.

In this set of experiment, as the base system used is the same, i.e., SVM, our main aim is to study the kernels evaluated by SVM with both Random Tree Graph Kernel and Inexact Graph Match Kernel. For artificial data, we have considered the

same dataset as discussed in the simple ring domain experiment (figure 5.1). We have considered a dataset of 10 positive and 10 negative examples.

The parameters set for SVM with inexact graph match and random tree as graph kernels, are as follows:

1. – t = 4 (Graph kernel.)
2. – u = 1(default).

Table 5.17 Graph Kernels results for simple Ring Domain

Graph Kernels	kernels evaluated	support vectors	training set accuracy	time (SEC)
Random Tree	512	10	100.00%	0.59
Inexact Graph Match	554	15	100.00%	1.63

We then applied SVM with both random graph kernel and inexact graph match kernel to the mutagenesis data. The mutagenesis data under consideration consisted of simple atoms and bonds as shown in figure 4.3. In this case too, we concentrate on evaluating the kernel performance with both random tree graph kernel and inexact graph match graph kernel.

The parameters set for SVM with inexact graph match and random tree as graph kernels, are as follows:

1. – t = 4 (Graph kernel.)
2. – u = 1(default).

Table 5.18 Graph Kernels results for Mutagenesis Domain

Graph Kernels	kernels evaluated	support vectors	training set accuracy	time (SEC)
Random Tree	671	18	87.50%	3.6
Inexact Graph Match	944	20	85.00%	6.38

In table 5.17 we observe that both the kernels have shown 100% classification in the ring domain experiment. The evaluating factors here are the number of kernels evaluated and the support vectors. We observe that the number of kernels and support vectors evaluated in the case of the random tree graph kernel are less compared to inexact graph match kernel. A similar observation is seen in table 5.18, where we observe that the accuracy of the inexact graph match kernel is less than the random tree graph kernel. The time complexity of the inexact graph match kernel is due to the root algorithm, which implements a graph isomorphism approach. For more information on inexact graph match algorithm and time complexity refer chapter 3. More number of kernels evaluated implies more optimization required by SVM to achieve the desired default threshold. This default threshold is set in the root algorithm of SVM source code and can be changed manually at runtime. In our set of experiments we have maintained the default threshold. More about the threshold definition and characteristics can be found at http://svmlight.joachims.org/. However, due to its ability to compute the cost involved in transforming a given graph G_1 into graph G_2 and satisfying the basic properties of a kernel as discussed in section 3.4, inexact graph match stands as a candidate graph kernel. From the above experiment we observe that the implementation

of inexact graph match as a graph kernel increases the time complexity of SVM as compared to random tree graph kernel. From this we can conclude that one of the major reasons for the large time complexity of Subdue as compared to SVM is the use of the inexact graph match. The random tree graph kernel approach is a more efficient way of calculating cost as compared to inexact graph match.

5.5 Summary

In this chapter we have looked into various artificial domain problems and evaluated the performance of the systems. Initially we considered the ring domain and tree domain. We evaluated the systems on these domains with of noise introduced in the dataset as irrelevant information and as concepts irregularities. We evaluated the systems on structurally large and structurally complex concepts.

We experimented with systems on real-world data – the Mutagenesis domain. We observed the systems performance on variation of the Mutagenesis data. We observed the effect of removing the additional information involved in the dataset on the classification accuracy and time. We analyzed the performance of the systems with introduction of background knowledge. We introduce background knowledge as a 'HUB' node connecting all the atoms and bonds. This approach was introduced to try to learn disconnected concepts if any.

We then performed a series of experiments on various artificial datasets involving various domains. We experimented with some complex domains such as the

clique problems and disconnected concepts. We also studied the effect of background knowledge as used in the Mutagenesis domain.

Finally we introduced Subdue's inexact graph match as a potential graph kernel. We performed experiments in both artificial and real-world domain understanding the functioning of both random tree graph kernel and inexact graph match as graph kernel.

We analyzed the kernels based on the support vectors and kernels evaluated and the accuracy achieved. In case of both artificial domains and real-world domains the random tree graph kernel performed better than the inexact graph match graph kernel in terms of classification accuracy and time.

From the complete experimentations we performed, we observed that SVM outperformed Subdue in almost all domain taken into consideration. Subdue has shown strong potential in case of domains with background knowledge. An in-depth study in this direction can help Subdue achieve better performance.

CHAPTER 6

CONCLUSIONS AND FUTURE WORK

In this research we have evaluated and analyzed the functioning of search-based and kernel-based systems fro supervised learning from graphs. Our candidate for a search-based system is Subdue and for kernel-based is Support Vector Machine with graph kernel. We have implemented the Random Tree Graph Kernel as a graph kernel to operate with SVM. The goal is to evaluate the performance of these systems based on performance metrics such as classification accuracy and time complexity. From a theoretical perspective, a search-based approach explores through the hypothesis space searching for candidate hypotheses for classification. Subdue as a search-based system generates candidate subgraphs. Subgraphs are generated, expanded and evaluated as more examples are explored. The hypothesis space consists of these substructures. Subdue generates candidate hypotheses from the examples it is learning to classify. The main reason behind Subdue's time complexity is the application of graph isomorphism approach in its core algorithm. The inexact graph match implemented by Subdue includes additional complexity thus increasing the computational time involved. From the set of artificial domain and real-world domain experiments we have observed that, though Subdue is capable of learning larger concepts, it is incapable of learning large concepts in a comparable time frame with SVM. This gives SVM an advantage over Subdue for implementation in real-world situation where the data is dynamic and changing. Subdue has also failed to achieve good performance in case of datasets

involving noise. Noise was added in two ways. In first case we added noise in the dataset as irrelevant nodes. In second method, noise was added as irregularities in the concept to be learned. In both the cases we observed that Subdue failed to learn the desired concept and hence gave a comparatively poor performance. In artificial and real-world domains we experimented with various situations analyzing the performance of the systems. We observed that in most of the cases SVM with graph kernel outperformed Subdue in terms of both accuracy and time.

SVM's better performance is due to the fact that SVM does not learn any particular concept to classify the data. On the contrary, the graph kernel just tries to find the most common factor between any two given graphs. In our case, this factor is provided by the random tree graph kernel. The main idea behind the random tree graph kernel is to traverse the given two graphs in such a way that the maximum depth in both graphs can be achieved. The calculated cost factor is then used to map the two graphs into a higher-dimensional feature space. Due to this nature of the graph kernel, SVM is not affected much by the noise in the dataset or the number of features involved. Thus for a good classification, the better the kernel involved, the better is the match cost computed and the better is the mapping done. The resulting higher-dimensional feature space is hence a well-spaced feature space. A good mapping results in a simple linear hyperplane. In most of the datasets we considered, we observe that the random tree graph kernel is able to map the given graphs into a well-spaced feature space, thus achieving a good classification. The time complexity in case of SVM is due to the SMO algorithm included in the basic SVM algorithm. The main aim while generating a good

hyperplane is to solve a Quadratic Problem. The SMO as discussed in [36], gives an efficient way of solving such QPs in linear time frame. Also, as the number of features involved with each input example reduces, the complexity of the feature space reduces, thus reducing the computational time involved.

From out experiment involving use of inexact graph match as a potential graph kernel, we observer that the introduction of inexact graph match as a substitute for random tree graph kernel increases the time complexity of SVM by a larger margin. Hence we can conclude that though inexact graph match is an efficient way of calculating the graph match cost, is not efficient in a comparable time frame with random tree graph kernel.

From an experimental perspective we observe that SVM has outperformed Subdue in almost all considered artificial and real-world domains. Even though Subdue's inexact graph match proves a good candidate for a graph kernel the main drawback is the time complexity involved which prevents it from being applied in practical use. We also observed that Subdue showed strong potential in domains with particular background knowledge. An in-depth look into this category can help understand the various aspects that Subdue can utilize to enhance performance. There are various other ways background knowledge can be embedded in datasets. In our experimentation we have taken into consideration two approaches. Broader looks into the implementation of background knowledge to guide Subdue to understand and learn better concepts are a key to enhancing Subdues performance. To improve the time complexity of Subdue the root algorithm need to introduce some additional heuristics.

Subdue is a general purpose knowledge discovery and learning system. A heuristic that would help Subdue work more efficiently with certain kinds of data can reduce the learning time involved with Subdue as a classifier. Subdue implements a greedy search technique to search the hypothesis space for potential concepts. One heuristic can be implementing a proper use of background knowledge. Background knowledge should be provided either implicitly to the system at runtime or explicitly through the dataset. Another kind of heuristic can be implemented in the case of domains wherein we have some idea of the concepts to be learned. In such a case weight can be assigned to nodes and edges giving them more importance than the others. This can guide Subdue internally to learn the desired concept in a much simpler way.

As additional future work, we would like to investigate why SVM with random tree graph kernel performs better than Subdue algorithmically. We would like to investigate the way a feature space is generated by SVM and how classification is done in more detail. We would like to investigate how SVM learns a particular concept and the role of the kernel function in the mapping of the input space to the feature space. We will also experiment with more datasets analyzing the performance of the systems. We will also compare various other graph kernels with the implemented random tree graph kernel and evaluate their performance. A good future approach will be implementation of random tree graph kernel as a substitute for the inexact graph match function in Subdue. From our experimentation with both inexact graph match and random tree walk as graph kernels with SVM, we observed that the main drawback with inexact graph match is its time complexity. Besides this, random tree graph kernel has proved to be a

considerable graph matching approach. We would hence like to implement random tree graph kernel or other such graph kernels with lower matching accuracy as a substitute for the inexact graph match function and analyze the performance of Subdue.

APPENDIX A

GRAPH REPRESENTATION OF MUTAGENESIS DATA USED BY SUBDUE AND SUPPORT VECTOR MACHINE

A.1: SUBDUE REPRESENTATION: MUTAGENESIS DATA - ATOMS, BONDS, ATOM TYPE, BOND TYPE, ELEMENTS AND CHARGES

```
XP
v 1 atom
v 2 atom
v 3 atom
v 4 atom
v 5 atom
v 6 atom
v 7 atom
v 8 atom
v 9 atom
v 10 atom
v 11 atom
v 12 atom
v 13 atom
v 14 atom
v 15 atom
v 16 atom
v 17 atom
v 18 atom
v 19 atom
v 20 atom
v 21 atom
v 22 atom
v 23 atom
v 24 atom
v 25 atom
v 26 atom
v 27 atom
v 28 atom
v 29 atom
v 30 atom
v 31 c
v 32 c
v 33 c
v 34 c
v 35 c
v 36 c
v 37 h
v 38 h
v 39 c
```

v 40 c
v 41 c
v 42 c
v 43 h
v 44 h
v 45 c
v 46 c
v 47 c
v 48 h
v 49 h
v 50 c
v 51 c
v 52 c
v 53 h
v 54 h
v 55 n
v 56 o
v 57 o
v 58 n
v 59 h
v 60 h
v 61 22
v 62 22
v 63 27
v 64 27
v 65 27
v 66 22
v 67 3
v 68 3
v 69 27
v 70 27
v 71 22
v 72 22
v 73 3
v 74 3
v 75 22
v 76 22
v 77 27
v 78 3
v 79 3
v 80 22
v 81 22
v 82 22
v 83 3

v 84 3
v 85 38
v 86 40
v 87 40
v 88 32
v 89 1
v 90 1
v 91 -0.111
v 92 -0.111
v 93 0.019
v 94 -0.081
v 95 0.019
v 96 -0.111
v 97 0.149
v 98 0.15
v 99 -0.081
v 100 0.019
v 101 -0.111
v 102 -0.111
v 103 0.15
v 104 0.15
v 105 -0.111
v 106 -0.111
v 107 0.019
v 108 0.149
v 109 0.149
v 110 -0.111
v 111 -0.111
v 112 -0.111
v 113 0.15
v 114 0.149
v 115 0.82
v 116 -0.38
v 117 -0.38
v 118 -0.76
v 119 0.35
v 120 0.35
v 121 bond
v 122 bond
v 123 bond
v 124 bond
v 125 bond
v 126 bond
v 127 bond

v 128 bond
v 129 bond
v 130 bond
v 131 bond
v 132 bond
v 133 bond
v 134 bond
v 135 bond
v 136 bond
v 137 bond
v 138 bond
v 139 bond
v 140 bond
v 141 bond
v 142 bond
v 143 bond
v 144 bond
v 145 bond
v 146 bond
v 147 bond
v 148 bond
v 149 bond
v 150 bond
v 151 bond
v 152 bond
v 153 bond
v 154 7
v 155 7
v 156 7
v 157 7
v 158 7
v 159 7
v 160 1
v 161 1
v 162 7
v 163 7
v 164 7
v 165 7
v 166 7
v 167 1
v 168 1
v 169 7
v 170 7
v 171 7

v 172 7
v 173 1
v 174 1
v 175 7
v 176 7
v 177 7
v 178 7
v 179 1
v 180 1
v 181 1
v 182 2
v 183 2
v 184 1
v 185 1
v 186 1
d 1 31 element
d 1 61 atom_type
d 1 91 charge
d 2 32 element
d 2 62 atom_type
d 2 92 charge
d 3 33 element
d 3 63 atom_type
d 3 93 charge
d 4 34 element
d 4 64 atom_type
d 4 94 charge
d 5 35 element
d 5 65 atom_type
d 5 95 charge
d 6 36 element
d 6 66 atom_type
d 6 96 charge
d 7 37 element
d 7 67 atom_type
d 7 97 charge
d 8 38 element
d 8 68 atom_type
d 8 98 charge
d 9 39 element
d 9 69 atom_type
d 9 99 charge
d 10 40 element
d 10 70 atom_type

d 10 100 charge
d 11 41 element
d 11 71 atom_type
d 11 101 charge
d 12 42 element
d 12 72 atom_type
d 12 102 charge
d 13 43 element
d 13 73 atom_type
d 13 103 charge
d 14 44 element
d 14 74 atom_type
d 14 104 charge
d 15 45 element
d 15 75 atom_type
d 15 105 charge
d 16 46 element
d 16 76 atom_type
d 16 106 charge
d 17 47 element
d 17 77 atom_type
d 17 107 charge
d 18 48 element
d 18 78 atom_type
d 18 108 charge
d 19 49 element
d 19 79 atom_type
d 19 109 charge
d 20 50 element
d 20 80 atom_type
d 20 110 charge
d 21 51 element
d 21 81 atom_type
d 21 111 charge
d 22 52 element
d 22 82 atom_type
d 22 112 charge
d 23 53 element
d 23 83 atom_type
d 23 113 charge
d 24 54 element
d 24 84 atom_type
d 24 114 charge
d 25 55 element

d 25 85 atom_type
d 25 115 charge
d 26 56 element
d 26 86 atom_type
d 26 116 charge
d 27 57 element
d 27 87 atom_type
d 27 117 charge
d 28 58 element
d 28 88 atom_type
d 28 118 charge
d 29 59 element
d 29 89 atom_type
d 29 119 charge
d 30 60 element
d 30 90 atom_type
d 30 120 charge
d 121 154 bond_type
d 122 155 bond_type
d 123 156 bond_type
d 124 157 bond_type
d 125 158 bond_type
d 126 159 bond_type
d 127 160 bond_type
d 128 161 bond_type
d 129 162 bond_type
d 130 163 bond_type
d 131 164 bond_type
d 132 165 bond_type
d 133 166 bond_type
d 134 167 bond_type
d 135 168 bond_type
d 136 169 bond_type
d 137 170 bond_type
d 138 171 bond_type
d 139 172 bond_type
d 140 173 bond_type
d 141 174 bond_type
d 142 175 bond_type
d 143 176 bond_type
d 144 177 bond_type
d 145 178 bond_type
d 146 179 bond_type
d 147 180 bond_type

d 148 181 bond_type
d 149 182 bond_type
d 150 183 bond_type
d 151 184 bond_type
d 152 185 bond_type
d 153 186 bond_type
d 1 121 connect
d 121 2 connect
d 2 122 connect
d 122 3 connect
d 3 123 connect
d 123 4 connect
d 4 124 connect
d 124 5 connect
d 5 125 connect
d 125 6 connect
d 6 126 connect
d 126 1 connect
d 1 127 connect
d 127 7 connect
d 2 128 connect
d 128 8 connect
d 4 129 connect
d 129 9 connect
d 9 130 connect
d 130 10 connect
d 10 131 connect
d 131 11 connect
d 11 132 connect
d 132 12 connect
d 12 133 connect
d 133 5 connect
d 11 134 connect
d 134 13 connect
d 12 135 connect
d 135 14 connect
d 3 136 connect
d 136 15 connect
d 15 137 connect
d 137 16 connect
d 16 138 connect
d 138 17 connect
d 17 139 connect
d 139 9 connect

d 15 140 connect
d 140 18 connect
d 16 141 connect
d 141 19 connect
d 17 142 connect
d 142 20 connect
d 20 143 connect
d 143 21 connect
d 21 144 connect
d 144 22 connect
d 22 145 connect
d 145 10 connect
d 20 146 connect
d 146 23 connect
d 21 147 connect
d 147 24 connect
d 25 148 connect
d 148 6 connect
d 26 149 connect
d 149 25 connect
d 27 150 connect
d 150 25 connect
d 22 151 connect
d 151 28 connect
d 28 152 connect
d 152 29 connect
d 28 153 connect
d 153 30 connect

A.2: SVM REPRESENTATION: MUTAGENESIS DATA - ATOMS, BONDS,

ATOM TYPE, BOND TYPE, ELEMENTS AND CHARGES

1 1:1 2:1 3:1 4:1 5:1 6:1 7:1 8:1 9:1 10:1 11:1 12:1 13:1 14:1 15:1 16:1 17:1 18:1 19:1
20:1 21:1 22:1 23:1 24:1 25:1 26:1 27:1 28:1 29:1 30:1 31:2 32:2 33:2 34:2 35:2 36:2
37:3 38:3 39:2 40:2 41:2 42:2 43:3 44:3 45:2 46:2 47:2 48:3 49:3 50:2 51:2 52:2 53:3
54:3 55:4 56:5 57:5 58:4 59:3 60:3 61:6 62:6 63:7 64:7 65:7 66:6 67:8 68:8 69:7 70:7
71:6 72:6 73:8 74:8 75:6 76:6 77:7 78:8 79:8 80:6 81:6 82:6 83:8 84:8 85:9 86:10 87:10
88:11 89:12 90:12 91:13 92:13 93:14 94:15 95:14 96:13 97:16 98:17 99:15 100:14
101:13 102:13 103:17 104:17 105:13 106:13 107:14 108:16 109:16 110:13 111:13
112:13 113:17 114:16 115:18 116:19 117:19 118:20 119:21 120:21 121:22 122:22
123:22 124:22 125:22 126:22 127:22 128:22 129:22 130:22 131:22 132:22 133:22
134:22 135:22 136:22 137:22 138:22 139:22 140:22 141:22 142:22 143:22 144:22

145:22 146:22 147:22 148:22 149:22 150:22 151:22 152:22 153:22 154:23 155:23 156:23 157:23 158:23 159:23 160:12 161:12 162:23 163:23 164:23 165:23 166:23 167:12 168:12 169:23 170:23 171:23 172:23 173:12 174:12 175:23 176:23 177:23 178:23 179:12 180:12 181:12 182:24 183:24 184:12 185:12 186:12 # |1| (2113.601318) [1 atom] [2 atom] [3 atom] [4 atom] [5 atom] [6 atom] [7 atom] [8 atom] [9 atom] [10 atom] [11 atom] [12 atom] [13 atom] [14 atom] [15 atom] [16 atom] [17 atom] [18 atom] [19 atom] [20 atom] [21 atom] [22 atom] [23 atom] [24 atom] [25 atom] [26 atom] [27 atom] [28 atom] [29 atom] [30 atom] [31 c] [32 c] [33 c] [34 c] [35 c] [36 c] [37 h] [38 h] [39 c] [40 c] [41 c] [42 c] [43 h] [44 h] [45 c] [46 c] [47 c] [48 h] [49 h] [50 c] [51 c] [52 c] [53 h] [54 h] [55 n] [56 o] [57 o] [58 n] [59 h] [60 h] [61 22] [62 22] [63 27] [64 27] [65 27] [66 22] [67 3] [68 3] [69 27] [70 27] [71 22] [72 22] [73 3] [74 3] [75 22] [76 22] [77 27] [78 3] [79 3] [80 22] [81 22] [82 22] [83 3] [84 3] [85 38] [86 40] [87 40] [88 32] [89 1] [90 1] [91 -0.111] [92 -0.111] [93 0.019] [94 -0.081] [95 0.019] [96 -0.111] [97 0.149] [98 0.15] [99 -0.081] [100 0.019] [101 -0.111] [102 -0.111] [103 0.15] [104 0.15] [105 -0.111] [106 -0.111] [107 0.019] [108 0.149] [109 0.149] [110 -0.111] [111 -0.111] [112 -0.111] [113 0.15] [114 0.149] [115 0.82] [116 -0.38] [117 -0.38] [118 -0.76] [119 0.35] [120 0.35] [121 bond] [122 bond] [123 bond] [124 bond] [125 bond] [126 bond] [127 bond] [128 bond] [129 bond] [130 bond] [131 bond] [132 bond] [133 bond] [134 bond] [135 bond] [136 bond] [137 bond] [138 bond] [139 bond] [140 bond] [141 bond] [142 bond] [143 bond] [144 bond] [145 bond] [146 bond] [147 bond] [148 bond] [149 bond] [150 bond] [151 bond] [152 bond] [153 bond] [154 7] [155 7] [156 7] [157 7] [158 7] [159 7] [160 1] [161 1] [162 7] [163 7] [164 7] [165 7] [166 7] [167 1] [168 1] [169 7] [170 7] [171 7] [172 7] [173 1] [174 1] [175 7] [176 7] [177 7] [178 7] [179 1] [180 1] [181 1] [182 2] [183 2] [184 1] [185 1] [186 1] < d 1 31 element > < d 1 61 atom_type > < d 1 91 charge > < d 2 32 element > < d 2 62 atom_type > < d 2 92 charge > < d 3 33 element > < d 3 63 atom_type > < d 3 93 charge > < d 4 34 element > < d 4 64 atom_type > < d 4 94 charge > < d 5 35 element > < d 5 65 atom_type > < d 5 95 charge > < d 6 36 element > < d 6 66 atom_type > < d 6 96 charge > < d 7 37 element > < d 7 67 atom_type > < d 7 97 charge > < d 8 38 element > < d 8 68 atom_type > < d 8 98 charge > < d 9 39 element > < d 9 69 atom_type > < d 9 99 charge > < d 10 40 element > < d 10 70 atom_type > < d 10 100 charge > < d 11 41 element > < d 11 71 atom_type > < d 11 101 charge > < d 12 42 element > < d 12 72 atom_type > < d 12 102 charge > < d 13 43 element > < d 13 73 atom_type > < d 13 103 charge > < d 14 44 element > < d 14 74 atom_type > < d 14 104 charge > < d 15 45 element > < d 15 75 atom_type > < d 15 105 charge > < d 16 46 element > < d 16 76 atom_type > < d 16 106 charge > < d 17 47 element > < d 17 77 atom_type > < d 17 107 charge > < d 18 48 element > < d 18 78 atom_type > < d 18 108 charge > < d 19 49 element > < d 19 79 atom_type > < d 19 109 charge > < d 20 50 element > < d 20 80 atom_type > < d 20 110 charge > < d 21 51 element > < d 21 81 atom_type > < d 21 111 charge > < d 22 52 element > < d 22 82 atom_type > < d 22

112 charge > < d 23 53 element > < d 23 83 atom_type > < d 23 113 charge > < d 24 54 element > < d 24 84 atom_type > < d 24 114 charge > < d 25 55 element > < d 25 85 atom_type > < d 25 115 charge > < d 26 56 element > < d 26 86 atom_type > < d 26 116 charge > < d 27 57 element > < d 27 87 atom_type > < d 27 117 charge > < d 28 58 element > < d 28 88 atom_type > < d 28 118 charge > < d 29 59 element > < d 29 89 atom_type > < d 29 119 charge > < d 30 60 element > < d 30 90 atom_type > < d 30 120 charge > < d 121 154 bond_type > < d 122 155 bond_type > < d 123 156 bond_type > < d 124 157 bond_type > < d 125 158 bond_type > < d 126 159 bond_type > < d 127 160 bond_type > < d 128 161 bond_type > < d 129 162 bond_type > < d 130 163 bond_type > < d 131 164 bond_type > < d 132 165 bond_type > < d 133 166 bond_type > < d 134 167 bond_type > < d 135 168 bond_type > < d 136 169 bond_type > < d 137 170 bond_type > < d 138 171 bond_type > < d 139 172 bond_type > < d 140 173 bond_type > < d 141 174 bond_type > < d 142 175 bond_type > < d 143 176 bond_type > < d 144 177 bond_type > < d 145 178 bond_type > < d 146 179 bond_type > < d 147 180 bond_type > < d 148 181 bond_type > < d 149 182 bond_type > < d 150 183 bond_type > < d 151 184 bond_type > < d 152 185 bond_type > < d 153 186 bond_type > < d 1 121 connect > < d 121 2 connect > < d 2 122 connect > < d 122 3 connect > < d 3 123 connect > < d 123 4 connect > < d 4 124 connect > < d 124 5 connect > < d 5 125 connect > < d 125 6 connect > < d 6 126 connect > < d 126 1 connect > < d 1 127 connect > < d 127 7 connect > < d 2 128 connect > < d 128 8 connect > < d 4 129 connect > < d 129 9 connect > < d 9 130 connect > < d 130 10 connect > < d 10 131 connect > < d 131 11 connect > < d 11 132 connect > < d 132 12 connect > < d 12 133 connect > < d 133 5 connect > < d 11 134 connect > < d 134 13 connect > < d 12 135 connect > < d 135 14 connect > < d 3 136 connect > < d 136 15 connect > < d 15 137 connect > < d 137 16 connect > < d 16 138 connect > < d 138 17 connect > < d 17 139 connect > < d 139 9 connect > < d 15 140 connect > < d 140 18 connect > < d 16 141 connect > < d 141 19 connect > < d 17 142 connect > < d 142 20 connect > < d 20 143 connect > < d 143 21 connect > < d 21 144 connect > < d 144 22 connect > < d 22 145 connect > < d 145 10 connect > < d 20 146 connect > < d 146 23 connect > < d 21 147 connect > < d 147 24 connect > < d 25 148 connect > < d 148 6 connect > < d 26 149 connect > < d 149 25 connect > < d 27 150 connect > < d 150 25 connect > < d 22 151 connect > < d 151 28 connect > < d 28 152 connect > < d 152 29 connect > < d 28 153 connect > < d 153 30 connect >

APPENDIX B

CONCEPTS LEARNED BY SUBDUE

B.1 CONCEPTS LEARNED FROM DATASET CONSISTING OF PARTIAL CHARGE, ATOMS, BONDS, ATOM TYPE, BOND TYPE AND ELEMENT

S
v 1 27

S
v 1 29

S
v 1 52

S
v 1 27

S
v 1 29

S
v 1 52

S
v 1 -0.152

S
v 1 -0.093

S
v 1 -0.142

S
v 1 -0.148

S
v 1 -0.191

S
v 1 0.315

S
v 1 -0.132

S

v 1 -0.096

S
v 1 -0.208

S
v 1 0.146

S
v 1 0.315

S
v 1 -0.152

S
v 1 0.011

S
v 1 -0.114

S
v 1 -0.108

S
v 1 atom
v 2 29
v 3 bond
v 4 1
u 1 2 atom_type
u 3 4 bond_type
u 3 1 connect

S
v 1 atom
v 2 atom
v 3 atom
v 4 c
v 5 c
v 6 h
v 7 29
v 8 22
v 9 bond
v 10 bond
v 11 bond

v 12 bond
v 13 7
v 14 7
u 1 4 element
u 1 7 atom_type
u 2 5 element
u 2 8 atom_type
u 3 6 element
u 9 13 bond_type
u 10 14 bond_type
u 9 1 connect
u 1 10 connect
u 10 2 connect
u 2 11 connect
u 11 3 connect
u 1 12 connect

S
v 1 atom
v 2 atom
v 3 atom
v 4 atom
v 5 atom
v 6 atom
v 7 c
v 8 c
v 9 c
v 10 n
v 11 o
v 12 22
v 13 22
v 14 22
v 15 38
v 16 40
v 17 -0.114
v 18 -0.114
v 19 0.817
v 20 -0.384
v 21 bond
v 22 bond
v 23 bond
v 24 bond
v 25 bond
v 26 bond

v 27 bond
v 28 bond
v 29 bond
v 30 7
v 31 7
v 32 7
v 33 1
u 2 7 element
u 2 12 atom_type
u 2 17 charge
u 3 8 element
u 3 13 atom_type
u 3 18 charge
u 4 9 element
u 4 14 atom_type
u 5 10 element
u 5 15 atom_type
u 5 19 charge
u 6 11 element
u 6 16 atom_type
u 6 20 charge
u 22 30 bond_type
u 23 31 bond_type
u 24 32 bond_type
u 27 33 bond_type
u 21 2 connect
u 2 22 connect
u 22 3 connect
u 3 23 connect
u 23 4 connect
u 4 24 connect
u 24 1 connect
u 2 25 connect
u 4 26 connect
u 5 27 connect
u 27 3 connect
u 28 5 connect
u 5 29 connect
u 29 6 connect

REFERENCES

1. N. Cristianini and J. Shawe Taylor. *An Introduction to Support Vector Machines (and Other Kernel-Based Learning Methods)*. Cambridge University Press, 2000.
2. Cook D. J. and Holder, L. B. 2000. *Graph Based Data Mining*. IEEE. Intelligent Systems.
3. J. Platt, *Probabilistic Outputs for Support Vector Machines and Comparisons to Regularized Likelihood Methods*, *Advances in Large Margin Classifiers*, A. Smola, P. Bartlett, B. Scholkopf, D. Schuurmans, eds., MIT Press, (1999).
4. Lopez de Compadre, A.K., Debnath, R.L., Debnath, G., Shusterman, A.J., and Hansch, C. (1991). *Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. Correlation with molecular orbital energies and hydrophobicity.*
5. Gartner, T. *Exponential and geometric kernels for graphs*. In NIPS Workshop on Unreal Data: Principles of Modeling Non vectorial Data, 2002.
6. Gartner, T. *A survey of kernels for structured data*. SIGKDD Explorations, 2003.
7. Gartner, T., Flach, P. A., and Wrobel, S. *On graph kernels: Hardness results and efficient alternatives*. In Proceedings of the 16th Annual Conference on Computational Learning Theory and the 7th Kernel Workshop, 2003b.

8. Gartner, T., Flach, P. A., and Wrobel, S. *Kernels for structured data*. Machine Learning, 2004.
9. Karchin, R., Karplus, K., and Haussler, D. *Classifying g-protein coupled receptors with support vector machines*. Bioinformatics, 18(1):147–159, 2002.
10. Kashima, H. and Inokuchi, A.. *Kernels for graph classification*. In ICDM Workshop on Active Mining, 2002.
11. Kashima, H. and Koyanagi, T. *Kernels for semi-structured data*. Proceedings of the 19th International Conference on Machine Learning. Morgan Kaufmann, 2002.
12. Kashima, H., Tsuda, K., and Inokuchi, A. *Marginalized kernels between labeled graphs.* In Proceedings of the 20th International Conference on Machine Learning, 2003.
13. Kondor, R. I. and Lafferty, J. *Diffusion kernels on graphs and other discrete input spaces.* Proceedings of the 19th International Conference on Machine Learning, pages 315–322. Morgan Kaufmann, 2002.
14. Muller, K. R., Mika, S., Ratsch, G., Tsuda, K., and Scholkopf, B. *An introduction to kernel-based learning algorithms*. IEEE Transactions on Neural Networks, 2(2), 2001.
15. Scholkopf, B. and Smola, A. J. *Learning with Kernels*. MIT Press, 2002.
16. Vishwanathan, S.V.N. and Smola, A.J. *Fast kernels for string and tree matching.* Advances in Neural Information Processing Systems, volume 15. MIT Press, 2003.

17. Yeung, K. Y., and Bumgarner, R. E. *Multiclass classification of microarray data with repeated measurements: application to cancer*. Department of Microbiology, University of Washington, USA.
18. Joachims T., *Support Vector and Kernel Methods*. SIGIR 2003
19. Joachims, T., *Learning to Align Sequences: A Maximum-Margin Approach,* Technical Report, August, 2003.
20. Joachims, T., *Estimating the Generalization Performance of a SVM Efficiently*. Proceedings of the International Conference on Machine Learning (ICML), Morgan Kaufman, 2000.
21. Joachims, T., *Making Large-Scale SVM Learning Practical*. In: Advances in Kernel Methods - Support Vector Learning, B. Scholkopf, C. Burges, and A. Smola (ed.), MIT Press, 1999.
22. Ramon, J. and Gartner, T. *Expressivity versus Efficiency of Graph Kernels*. In Proceedings of the First International Workshop on Mining Graphs, Trees and Sequences (MGTS-2003).
23. Meyer, A and Nagpal, R. *Graphs.* Massachusetts Institute of Technology, 2002.
24. Scholkopf B., Christopher J.C., Burges Alexander J. Smola. Kernel Methods and Support Vector Learning. The MIT Press Cambridge, Massachusetts London, England .1998.
25. Rodríguez, Carlos C. *The Kernel Trick.*
26. R. M. Gray, *Entropy and Information Theory*.
27. Ketkar, N. *Graph-Based Representation of the Mutagenesis Dataset.*
28. Josephy D., *Structures of Chemical Mutagens*. University of Guelph.

29. Genton, M. G., *Classes of Kernels for Machine Learning: A Statistics Perspective.* Journal of Machine Learning Research 2 (2001) 299-312.

30. Haussler, D., *Convolution kernels on discrete structures*, tech. rep., UCSC-CRL-99-10, 1999.

31. Vapnik, V., *Statistical Learning Theory*, Wiley-Interscience, (1998).

32. Boser, B. E., Guyon, I. M., and Vapnik, V. N. *A training algorithm for optimal margin classifiers.* In D. Haussler, editor, 5th Annual ACM Workshop on COLT, pages 144-152, Pittsburgh, PA, 1992. ACM Press.

33. Aizerman, M., Braverman, E. and Rozonoer, L., *Theoretical foundations of the potential function method in pattern recognition learning.* Automation and Remote Control.

34. Mahe, P., Ueda, N., Akutsu, T., Perret J. L. and Vert J. P. *QSAR analysis with Support Vector Machine and Graph kernels.* 3rd Joint Sheffield Conference on Chemo-informatics. 2004.

35. Stone, M. H. *Linear Transformations in Hilbert Space and Their Applications Analysis*. Providence, RI: Amer. Math. Soc., 1932.

36. Platt, J., *Sequential Minimal Optimization: A Fast Algorithm for Training Support Vector Machines*, Microsoft Research Technical Report MSR-TR-98-14, (1998).

37. Srinivasan, A., Muggleton, S., King, R.D., and Sternberg, M.J.E. (1994) *Mutagenesis: ILP experiments in a non-determinate biological domain.* Proceedings of the Fourth Inductive Logic Programming Workshop.

38. Muggleton, S. and Raedt, L. D. *Inductive Logic Programming: Theory and Methods.* J. Log. Program.1994

39. Canu, S. and Smola, A. J., *Kernel methods and the exponential family.*

40. Shannon, C. E., *A Mathematical Theory of Communication*, the Bell System Technical Journal, 1948.

41. Mooney, R.J., Melville, P., Tang L. R., Shavlik J., Dutra I., Page D., *Relational Data Mining with Inductive Logic Programming for Link Discovery*, Kargupta, H., Joshi, A., Sivakumar K., and Yesha, Y. (Eds.), Data Mining: Next Generation Challenges and Future Directions , pp. 239--254, AAAI Press, Menlo Park, CA, 2004.

42. Cook D. J. and Holder, L. B. 2000. *Substructure Discovery Using Minimum Description Length and Background Knowledge*. IEEE. Journal of Artificial Intelligence Research 1 (1994) 231-255.

43. Bennett, K.P. and Campbell, C., *Support Vector Machines: Hype or Hallelujah ?* SIGKDD Explorations, 3:1, pp. 1-13 (2001).

44. Weisstein, Eric W., *Positive Definite Function*. From MathWorld--A Wolfram Web Resource.

45. Nguyen,P. C., Ohara,K., Motoda, H. and Washio, T., *"Cl-GBI: A Novel Strategy to Extract Typical Patterns from Graph Data"*, Joint Workshop of Vietnamese Society of AI, SIGKBS-JSAI, ICS-IPSJ and IEICE-SIGAI on Active Mining,pp.105-110 (2004)

46. Inokuchi A., Washio T., Motoda H. 2000 *An Apriori-based Algorithm for Mining Frequent Substructures from Graph Data.* Principles of Data Mining and Knowledge Discovery 2000.

47. Washio T., Motoda H. 2003. *State of the Art of Graph-based data Mining*, SIGKDD Explorations, 2003.

BIOGRAPHICAL INFORMATION

Chris M. Gonsalves received his Bachelor of Engineering in Information Technology from Mumbai University, Mumbai, India, in July 2003.

He was a research assistant for Dr. Merchant, Indian Institute of Technology, Powai, Mumbai, India. He worked under Dr. Merchant in the fields of Data Security via Digital Watermarking.

He received this Masters in Computer Science and Engineering from the University of Texas at Arlington, Texas, in August 2005. His research interests include data security, machine learning and data mining.