論文 / 著書情報
Article / Book Information

題目(和文)	
Title(English)	Cluster Graph Classi cation Using the Generalized Shortest Path Kernel
著者(和文)	HermanssonLinusHakan
Author(English)	Linushakan Hermansson
出典(和文)	学位:博士(学術), 学位授与機関:東京工業大学, 報告番号:甲第10382号, 授与年月日:2016年12月31日, 学位の種別:課程博士, 審査員:渡辺 治,増原 英彦,鹿島 亮,鈴木 大慈,脇田 建
Citation(English)	Degree:Doctor (Academic), Conferring organization: Tokyo Institute of Technology, Report number:甲第10382号, Conferred date:2016/12/31, Degree Type:Course doctor, Examiner:,,,,
学位種別(和文)	博士論文
Type(English)	Doctoral Thesis

Cluster Graph Classification Using the Generalized Shortest Path Kernel

Doctor of Philosophy
Department of Mathematical and Computing Science

LINUS HERMANSSON

Tokyo Institute of Technology
Department of Mathematical and Computing Science
Tokyo, Japan. August 2016.

Cluster Graph Classification Using the Generalized Shortest Path Kernel

LINUS HERMANSSON

Examiner: Osamu Watanabe

Department of Mathematical and Computing Science
Tokyo Institute of Technology
Meguro-ku Ookayama, Tokyo 152-8552
Japan

Abstract

Classifying graphs into different classes based on their structure is a problem which has become increasingly popular in recent years and that has many useful applications. Using various techniques to represent real world data as mathematical graphs and then classifying these graphs using different machine learning approaches; it has become possible to detect certain types of cancers, classify proteins by their enzyme class, which is very useful in bioinformatics, predicting weather, and much more.

Classifying graphs can be done in many different ways. The approach to classifying graphs taken in this thesis is one based on graph kernels and support vector machines, where we consider the supervised machine learning setting. We use different graph kernels to represent graphs as mathematical vectors and then train a support vector machine to find a separating hyperplane which accurately classifies the vectors. Using this separating hyperplane, we are then able to predict the class of future graphs by transforming them into vectors using a graph kernel, and then testing which side of the separating hyperplane they lie. We mainly investigate two different graph kernels, the well known and popular shortest path kernel and the recently published generalized shortest path kernel. We compare the two graph kernels in the task of classifying graphs by the number of clusters they contain.

The result of our work is an analysis of, in which situations, and why, the generalized shortest path kernel outperforms the shortest path kernel. We also provide a lot experimental results, that confirms our analysis. We provide several random models for generating random cluster graphs, which we use in our experiments in order to test the performance of the graph kernels.

Acknowledgments

I would like to extend my deepest gratitude to my supervisor, Osamu Watanabe. Professor Watanabe has helped my greatly with my PhD, in that he was frequently available for discussions and provided guidance which focused my research.

I also would like to thank the secretaries of Watanabe-lab, Kazuyo Kawaguchi and Tamami Watanabe. Who always supported me and all other students in the laboratory.

Sincere thanks to all my friends at Tokyo Institute of Technology, who made finishing this PhD much more enjoyable. Among these friends are Tommi Kerola, Tatsuya Imai, Johan Rohdin and Ashida Ryo.

Finally I thank my family–Håkan, Helena, Viola and Rasmus Hermansson–for their love and support from Sweden.

This work is supported in part by KAKENHI No. 24106008.

I dedicate this thesis to my parents, Håkan and Helena Hermansson.

Contents

List of Figures

List of Tables

1

Introduction

In recent years, the amount of data which is freely available online for anyone to access has increased greatly. With this increase of available data and information, the need for efficient and accurate ways of classifying this data has become a popular topic of research. When we are dealing with data which is so large that it cannot possibly be classified manually by humans, it is incredibly useful to find automatic ways of having computers classify the data, even if the computers might not achieve 100% accuracy in their classification. There are many different types of data which need to be classified, such as social networks, weather patterns, x-ray images, different types of enzymes, the list goes on and on. In order for a computer to classify data into different classes, the data needs to be in some standardized format so that it is possible to design a well thought out algorithm, by which the computer is able to classify the data into different classes. One such standard model of data is mathematical graphs, of vertices and edges. Although the data which needs to be classified may not initially be stored in the form of graphs, it is often possible to transform the data into graphs and then classify the graphs themselves. This has been done, for instance, when classifying proteins as different types of enzymes [5]. The initial data available is the spatial data of how the proteins are made up. Proteins can belong to different enzyme classes and consists of helices, sheets and loops, which exist at points in space. Using this information, it is possible to create a graph of each protein, based on this protein structure and then classify the graph in order to predict the enzyme class of the protein. Another example is to classify human tissue as containing cancer or not. The data which needs to be classified is a picture of human tissue, this image

can then be translated into a graph, which can then be classified as containing cancer or not [2]. Thus it is possible to classify a wide range of different types of data, as long as we are able to classify standard mathematical graphs. With this in mind, our goal in this thesis is to classify different types of graphs, and analyze for which types of graphs we may expect different methods to work.

1.1 Method

We consider the problem of classifying graphs into different classes using a supervised machine learning approach. In the supervised machine learning setting we are provided with a training set, of correctly classified data, for each problem. This training set we may use in order to make the computer "learn" how to classify the data, the computer should then be able to classify new data correctly if the learning process worked correctly. One example of this would be classifying images of human tissue as either containing cancer or not. Say that we are given 100 images of tissue without cancer and 100 with cancer, where we are given the information of which images contain cancer and which do not. In the supervised machine learning setting of classifying graphs which we consider, we would then transform all the images to graphs, using an appropriate process. We may then train our SVM so that it is able to classify the 200 training example graphs with 100% or close to 100% accuracy. When anyone later gives us a new image of human tissue, if the SVM training worked correctly, we would be able to give this new image, after transforming it to a graph, to our trained SVM and the SVM will tell us if the new image represents tissue with or without cancer.

Classifying graphs into classes based on their structure has been studied for a long time and has been shown to have many useful applications [2, 5, 21, 22]. By classifying graphs researchers have been able to solve important problems such as to accurately predict the toxicity of chemical compounds [22], classify if human tissue contains cancer or not [2], predict if a particular protein is an enzyme or not [5], and many more.

The supervised machine learning approach has proven very effective in classifying several different datasets [2, 5, 16, 18, 21, 22] and thus has become very popular as a tool for classifying data. In this thesis we consider the supervised machine learning approach, for classifying graphs by the number of clusters they contain. We use graph kernels in order to compare graphs and then classify them using a *support vector machine* (SVM). Throughout this thesis we consider binary classification on

graphs, meaning that each graph belongs to one of two classes and it is our goal to find which class any particular graph belongs to. Another convinient way of expressing this is to say that each graph has a label, which is either $+1$ or -1 and it is our task to find this label for each graph. One example of a problem handled in this thesis is to, for instance, classify if a graph contains 2 or 3 clusters. For such a problem we could say that the 2 cluster graphs have the label $+1$ while the 3 cluster graphs have the label -1. We also always consider supervised machine learning, which means that for each dataset, we are provided with a training set, which we can use to train our SVM on. When the SVM has been trained for any particular dataset we may use it in order to predict the label of any other graphs.

1.1.1 Support Vector Machines

This section contains a brief summary of what an SVM is, for a detailed definition see Sect. 3.1. An SVM is a tool used in supervised machine learning in order to classify data into different classes. The SVM uses training examples, where each example consists of a vector $\mathbf{x}^{(i)} \in \mathbb{R}^d$ and an associated label $y_i \in \{+1, -1\}$. I.e. a training example is a pair $(\mathbf{x}^{(i)}, y_i)$. The basic SVM finds a maximum margin hyperplane that separates the training examples of different labels. New examples which we want to find the label of can then be classified by checking which side of the hyperplane they are on.

SVMs have proven to work very well together with graph kernels in order to classify graphs. In this thesis we mainly consider using two different graph kernels, the *shortest path* (SP) kernel and the *generalized shortest path* (GSP) kernel, together with an SVM in order to classify graphs. Sometimes for simplicity we write "the accuracy of the SP/GSP kernel", when we mean the accuracy of an SVM which uses the SP/GSP kernel.

1.1.2 Graph Kernels

This section contains a brief summary of what a graph kernel is, for a detailed definition see Sect. 3.2. For two graphs, G_1 and G_2, a graph kernel is a function $k(G_1,G_2)$ on pairs of graphs, which can be represented as an inner product $k(G_1,G_2) = \langle \phi(G_1), \phi(G_2) \rangle_{\mathcal{H}}$ for some mapping $\phi(G)$ to a Hilbert space $\mathcal{H}$, of possibly infinite dimension. It is convenient to think of graph kernels as similarity

functions on graphs. Graph kernels have been used as tools for SVM classifiers for several graph classification problems [4, 5, 16].

One of the most popular graph kernels is the *shortest path* (SP) kernel. The SP kernel compares graphs based on the number of vertex pairs with shortest paths of certain lengths. Intuitively, the SP kernel gives a high value if vertex pairs from G_1 tend to have similar shortest path lengths as the vertex pairs in G_2. The SP kernel has been shown to be able to accurately classify several types of graphs into their relevant classes [4, 16, 18]. In this thesis we analyze the features of the SP kernel for the relevant datasets that we investigate.

A very recent graph kernel, called the *generalized shortest path* (GSP) kernel, which was published by this thesis's author [15], has been proven to outperform the SP kernel on certain datasets [15], even though the GSP kernel takes the same amount of time to compute as the SP kernel when using standard algorithms. The generalized shortest path kernel compares graphs by the number of vertex pairs, that have shortest paths of certain lengths, and the number of such shortest paths. Intuitively, the GSP kernel gives a high value if vertex pairs from G_1 tend to have similar shortest path lengths and similar number of such shortest paths as the vertex pairs from G_2.

1.2 Scope

In this thesis we analyze and compare experimentally the SP kernel with the GSP kernel. Our goal is to analyze when the GSP kernel outperforms the SP kernel and vice versa and what the reason for this is.

The approach taken in this thesis is one based on graph kernels and SVMs, in the supervised machine learning setting. Thus we are not concerned with other, although they may be useful and interesting, approaches such as unsupervised learning, neural networks and such. We also only consider binary classification, i.e. each graph in the dataset belongs to one of two classes.

In this thesis we limit ourselves to two main graph kernels, the SP kernel and the GSP kernel. Meaning that we do not consider other popular graph kernels such as the graphlet kernel [28], random walk kernel [13] etc. We also only consider datasets of cluster graphs. The cluster graphs which we consider are generated using random models see Sect. 4.1 and 5.1. The reason for this is so that we have well defined random models which generate our random graphs. Based on these random models we are then able to perform accurate analyses, see Sect. 4.3 and 5.2.

1.3 The Importance of Classifying Cluster Graphs

By a cluster we mean a subgraph of a graph where the edge density is higher inside each cluster than between the clusters. Graphs with a cluster structure appear quite frequently in real world datasets and can represent a wide variety of things. A cluster could, for instance, represent a group of friends in a graph of a social network. For such graphs, determining the number of clusters would mean finding the number of groups of friends in the social network. It is possible to represent the different web pages, or a subset of them, of the world wide web as a graph. By finding clusters of web pages it is possible to give more relevant search results when a user queries a search engine. For instance, if we have identified that web pages related to booking airplane tickets are in the same cluster as web pages containing information about tourist guides (because they are frequently accessed together). Then when a user searches for a way to book air plane tickets, they could also receive information about tourist guides. It is also possible to imagine a graph of different businesses. Then, for a company who is already dealing a lot with a company A, it could be very useful to know about other companies who are very similar, I.e. are in the same cluster as company A. Identifying clusters in graphs have many other useful applications as well, such as for instance predicting weather [17], dividing images into different regions or object recognition in images [1], and many more.

1.4 Our Results

In this thesis we consider two main problems (Chap. 4 and 5), each with two different models for generating random graphs with cluster structures. Using these models we are able to generate datasets and investigate the performances of the SP and GSP kernels. We provide a heuristic analysis of the performances of both the SP and the GSP kernels. Our analysis provides us with formulas for predicting the accuracy of the SP and GSP kernels, which depends on certain parameters used when generating the random graphs for the classification problems. We also show experimental results that provide evidence that our heuristic analysis is in fact accurate. We also provide a large number of experimental results where we show when the GSP kernel outperforms the SP kernel. Below follows a summary of all our results, for the full explanations, see all chapters after the introduction.

1.4.1 Preliminaries of Results

Our experimental setup is as follows. Given a dataset of graphs, where each graph has a corresponding label, either $+1$ or -1. We transform each graph into a vector using one of two different graph kernels. The vectors, together with their labels, are then given to an SVM, in order to train the SVM so that it is able to predict the label of future graphs. When testing the accuracy of our graph kernels we used 10-fold cross validation.

The two different graph kernels we investigated were, an already existing graph kernel, called the SPI kernel; and our own new graph kernel, the GSPI kernel. Let $D(G)$ denote the multi set of shortest distances between all vertex pairs in the graph G. By $ND(G)$ we denote the multi set of numbers of shortest paths between all vertex pairs of G. The SPI kernel is defined as follows

$$K_{\text{SPI}}(G_1,G_2) = \sum_{d_1 \in D(G_1)} \sum_{d_2 \in D(G_2)} \mathbb{1}\left[d_1 = d_2\right].$$

The GSPI kernel is defined as

$$K_{\text{GSPI}}(G_1, G_2) = \sum_{d_1 \in D(G_1)} \sum_{d_2 \in D(G_2)} \sum_{t_1 \in ND(G_1)} \sum_{t_2 \in ND(G_2)} \mathbb{1}\left[d_1 = d_2\right] \mathbb{1}\left[t_1 = t_2\right].$$

A good property of our new GSPI kernel is that we can calculate the feature vectors of the GSPI kernel in the same running time as we can calculate the feature vectors of the SPI kernel.

1.4.2 One-Cluster Graphs and Two-Cluster Graphs

In this problem we consider classifying graphs as containing either 1 or 2 clusters. The one-cluster graphs where generated using the Erdős-Rényi model. Which means that each graph consists of n vertices and each vertex pair is connected with probability p_1. The two-cluster graphs also contain n number of vertices but are generated using a different random model, called the planted partition model [20]. In this model, for our two-cluster graphs, the graph consists of two clusters, with half of the vertices in each cluster. Vertex pairs that are from the same cluster are connected with a probability p_2 and vertices that are from different clusters are connected with the probability q_2. Where we define that

$$p_2 = (1+\alpha)p_1, \quad \text{and} \quad q_2 = 2p_1 - p_2 - 2(p_1 - p_2)/n.$$

Where α is a parameter which we vary in our experiments. q_2 is defined so that the expected number of edges is the same for the one-cluster graphs and the two-cluster graphs. In our experiments each dataset consisted of 100 graphs of each type (200 graphs in total). To evaluate the two different graph kernels we used an SVM with 10-fold cross validation. Our experiments showed that our new GSPI kernel performed better, in terms of accuracy, than the SPI kernel for nearly all datasets that were tested. These results can be seen in Tbl. 1.1.

We also performed an analysis of the feature vectors for both the SPI kernel and the GSPI kernel, in order to better understand when the two different kernels are able to classify the graphs with high accuracy. For our analysis we used the following method of approximation. For any functions a and b depending on n, we write $a \approx_{\text{rel}} b$ by which we mean

$$b\left(1-\frac{c}{n}\right) < a < b\left(1+\frac{c}{n}\right)$$

holds for some constant $c > 0$ and sufficiently large n.

For the SPI kernel we were able to show the following theorem,
Theorem. *For any constant d, we have $\mathrm{E}[n_d^{(1)}] \in \mathrm{E}[n_d^{(2)}](1 \pm \frac{2}{np_1-1})$, holds within our $\approx_{\text{rel}}$ approximation when $np_1 \geq 2+\sqrt{3}$.*

See Chap. 4 for the proof of the theorem. For this analysis we consider one fixed vertex s, in a graph. $\mathrm{E}[n_d^{(1)}]$, is the expected number of vertices that are at distance d from s in a one-cluster graph. $\mathrm{E}[n_d^{(2)}]$ is the expected number of vertices at distance d from s in a two-cluster graph. The expected values $\mathrm{E}[n_d^{(1)}]$ and $\mathrm{E}[n_d^{(2)}]$, when considering vertex pairs instead of one fixed vertex s, are in fact all the expected values inside the SPI feature vectors for the one-cluster and two-cluster graphs, when we consider $d \geq 1$ up to the longest shortest path in the graph. Thus, this theorem shows that the expected values of the SPI feature vectors for the two types of graphs are relatively close, in fact they are within a factor $1 \pm 2/(np_1 - 1)$ of each other. Note that this difference vanishes when np_1 grows. We believe this is one reason why the SPI kernel is not able to distinguish the two classes of graphs so well, since if the feature vectors of the two different classes are very similar, the SVM will not be able to distinguish them with high accuracy.

For the GSPI kernel we analyzed a part of the feature vectors, namely when $d = 2$. We write this part of the feature vectors as $[\mathrm{E}[n_{2,x}^{(z)}]]_{x\geq 1}$ for $z \in \{1,2\}$. In this analysis, we once again consider feature vectors relative to one fixed vertex s. By $\mathrm{E}[n_{2,x}^{(z)}]$ we mean the number of vertices that are at distance 2 from s and have exactly x number of shortest paths to s, in a z cluster graph. Using a heuristic analysis, we

Table 1.1: The accuracy of the SPI kernel and the GSPI kernel using 10-fold cross validation. The datasets where $p_2 = 1.2p_1$ are the hardest and the datasets where $p_2 = 1.5p_1$ are the easiest. Very big increases in accuracy are marked in bold.

Kernel	n	p_2	Accuracy
SPI	200	$\{1.2p_1, 1.3p_1, 1.4p_1, 1.5p_1\}$	$\{52.5\%, 55.5\%, \mathbf{54.5}\%\mathbf{56.5}\%\}$
GSPI	200	$\{1.2p_1, 1.3p_1, 1.4p_1, 1.5p_1\}$	$\{52.5\%, 64.0\%, \mathbf{99.0}\%, \mathbf{100.0}\%\}$
SPI	400	$\{1.2p_1, 1.3p_1, 1.4p_1, 1.5p_1\}$	$\{55.5\%, 63.5\%, \mathbf{75.5}\%, 95.5\%\}$
GSPI	400	$\{1.2p_1, 1.3p_1, 1.4p_1, 1.5p_1\}$	$\{54.0\%, 62.0\%, \mathbf{96.5}\%, 100.0\%\}$
SPI	600	$\{1.2p_1, 1.3p_1, 1.4p_1, 1.5p_1\}$	$\{58.0\%, 60.5\%, \mathbf{75.5}\%, 93.5\%\}$
GSPI	600	$\{1.2p_1, 1.3p_1, 1.4p_1, 1.5p_1\}$	$\{58.0\%, 67.0\%, \mathbf{94.0}\%, 100.0\%\}$
SPI	800	$\{1.2p_1, 1.3p_1, 1.4p_1, 1.5p_1\}$	$\{57.5\%, 59.0\%, 72.0\%, 98.0\%\}$
GSPI	800	$\{1.2p_1, 1.3p_1, 1.4p_1, 1.5p_1\}$	$\{57.5\%, 58.0\%, 82.0\%, 100.0\%\}$
SPI	1000	$\{1.2p_1, 1.3p_1, 1.4p_1, 1.5p_1\}$	$\{53.5\%, 55.0\%, \mathbf{66.0}\%, 98.5\%\}$
GSPI	1000	$\{1.2p_1, 1.3p_1, 1.4p_1, 1.5p_1\}$	$\{55.0\%, 62.0\%, \mathbf{87.5}\%, 100.0\%\}$

were able to determine that these values are proportional, for a one-cluster graph, approximately to the following normal distribution $\mathrm{N}(np_1^2, np_1^2(1-p_1))$. Which has one peak at np_1^2. For the two-cluster graphs however, the corresponding values, approximately, are proportional to the mixture distribution of the two distributions $\mathrm{N}(n(p_2^2+q_2^2)/2, \sigma_1^2+\sigma_2^2)$ and $\mathrm{N}(np_2q_2, \sigma_3^2+\sigma_4^2)$. This mixture distribution, which is observed for the two-cluster graphs, is quite different from the simple normal distribution observed for the one-cluster graphs. The mixture distribution for the two-cluster graphs has two peaks, located at $n(p_2^2+q_2^2)/2$ and np_2q_2. In our analysis, we managed to show that these two peaks have non-negligible relative difference. More precisely we showed that the difference between the peaks is approximately $2n\alpha^2p_1^2$. Which is significant relative to the position of the peaks themselves. These values, $\mathrm{E}[n_{2,x}^{(z)}]$, are part of the GSPI feature vectors. This difference between the values of the feature vectors, can be seen as a reason why the GSPI kernel is able to accurately classify the two types of graphs.

1.4.3 k-Cluster Graphs and $k+1$-Cluster Graphs

In this problem we consider classifying graphs as containing either k or $k+1$ clusters, where $k \geq 2$. All graphs consists of n number of vertices, where the k-cluster graphs are generated according to the following model. Each cluster contains n/k number of vertices and the presence of edges is determined randomly. Two vertices that are from the same cluster are connected with probability p, while vertices that are from different clusters are connected with probability q_1. q_1 is defined as

$$q_1 = p(1 - \beta).$$

Where β is a parameter which we vary in our experiments in order to make the classification task easier or more difficult. The $k+1$-cluster graphs are generated in a very similar way. For the $k+1$-cluster graphs, each cluster contains $n/(k+1)$ number of vertices. Two vertices that are from the same cluster are connected with probability p, which is the same as for the k-cluster graphs. Two vertices that are from different clusters are connected with probability q_2, where we define q_2 in such a way that the expected number of edges is the same for the k-cluster graphs and the $k+1$-cluster graphs. This means that q_2 is defined as

$$q_2 = q_1 + \frac{1}{k^2}(p - q_1) = q_1 + p\frac{\beta}{k^2}.$$

For the experiments, each dataset consisted of 200 graphs of each type (400 in total). For this classification task, where we determine if a graph contains k or $k+1$ clusters, we evaluated the performance of the two graph kernels using an SVM with 10-fold cross validation. We tested the performances of the two graph kernels on a large number of datasets, here we present two tables of results which are representative of how the graph kernels normally performed relative to each other. These results can be seen in Tbl. 1.2 and 1.3. As can be seen the GSPI kernel performed significantly better than the SPI kernel for the shown datasets, this was in fact observed for virtually all of the tested datasets (see Chap. 5 for details). We also note that the GSPI kernel has a bigger advantage, in terms of accuracy, over the SPI kernel when p is larger. This phenomenon is also discussed in more detail in Chap. 5.

We also performed a heuristic analysis of the accuracy of the GSPI kernel, which can be summarized as follows. We consider a fixed vertex s which is in one of the random graphs (either a k-cluster or a $k+1$-cluster graph). The number of shortest paths between s and another vertex t, that is at distance 2 from s, is then approximately

Table 1.2: Comparison of accuracy for the SPI kernel and the GSPI kernel. $p = 0.15$, $k = 2$.

β	SPI accuracy	GSPI accuracy
0.2	56%	63.6%
0.22	56%	70.2%
0.24	56%	78.2%
0.26	56%	85.3%
0.28	56%	92.0%
0.3	56%	95.2%
0.32	56%	98.4%

Table 1.3: Comparison of accuracy for the SPI kernel and the GSPI kernel. $p = 0.09$, $k = 2$.

β	SPI accuracy	GSPI accuracy
0.26	60.3%	71.3%
0.28	61.7%	74.7%
0.3	60.3%	82.3%
0.32	62.2%	84.1%
0.34	68.1%	91.9%
0.36	74.5%	92.8%
0.38	78.9%	98.5%

distributed according to the following mixture distribution

$$w_1^{(k)}(\text{Bin}(\frac{np}{k},p) + (k-1)\text{Bin}(\frac{nq}{k},q)) +$$
$$w_2^{(k)}(\text{Bin}(\frac{np}{k},q) + \text{Bin}(\frac{nq}{k},p) + (k-2)\text{Bin}(\frac{nq}{k},q)).$$

Were q should be replaced by q_1 when considering k-cluster graphs. When considering $k+1$-cluster graphs q should be replaced by q_2 and k by $k+1$. $w_1^{(k)}$ and $w_2^{(k)}$ are weights of the distributions and

$$w_1^{(k)} + w_2^{(k)} = 1$$

The fact that the number of shortest paths between s and t are distributed in this way, means that the GSPI feature vectors will get a specific shape. Since the values are distributed according to a mixture distribution, the distribution has two peaks, where the first peak is

$$x_{\text{peak}}^{(1,k)} = \frac{n}{k}(p^2 + (k-1)q^2),$$

and the second peak is

$$x_{\text{peak}}^{(2,k)} = \frac{n}{k}(2pq + (k-2)q^2),$$

When considering k-cluster graphs.

Our experiments indicated that, if the two peaks are very close together (low β), the feature vectors tend to look the same for the k-cluster graphs and the $k+1$-cluster graphs. Thus, for such datasets, the accuracy of the SVM which uses the GSPI kernel will not be able to achieve a high accuracy. While if the two peaks are far apart (high β), the feature vectors tend to look different for the two types of graphs. For these types of datasets the accuracy of the SVM which uses the GSPI kernel is high. Since the distance between the peaks is important for the accuracy of the GSPI kernel, we want to analyze when the peaks are significantly far apart. It is well known that the difficulty of distinguishing between a simple one peak distribution and a mixture distribution with two peaks, is related to the ratio of the distance of the two peaks and the standard deviation of the distributions [25]. This ratio, for our distribution, can be written approximately as

$$R_{n,p,k,\beta} \approx \sqrt{n}\, p\, \beta^2/(k(1-\beta)).$$

We conjecture that this ratio is a major factor for determining the accuracy of the GSPI kernel. In our experiments we only consider $n = 1000$, and thus we consider n to be constant in our analysis as well.

When the accuracy of the GSPI kernel is plotted with β as the x-axis, the accuracy seems to increase linearly once it has started to increase above 60%. See for instance Fig. 1.1. This means that the accuracy can be written, at least approximately, as $a\beta + b$. But note that

$$\sqrt{R_{n,p,k,\beta}} \propto \left(\sqrt{p/k}\right) \beta/\sqrt{1-\beta},$$

and that $\beta/\sqrt{1-\beta}$ is close to linear for the values of β that we are interested in. Thus, if $R_{n,p,k,\beta}$ is the major factor for determining the accuracy of the GSPI kernel, the accuracy can be written as

$$\sqrt{R_{n,p,k,\beta}} + b \approx c\left(\sqrt{p/k}\right)\beta + b.$$

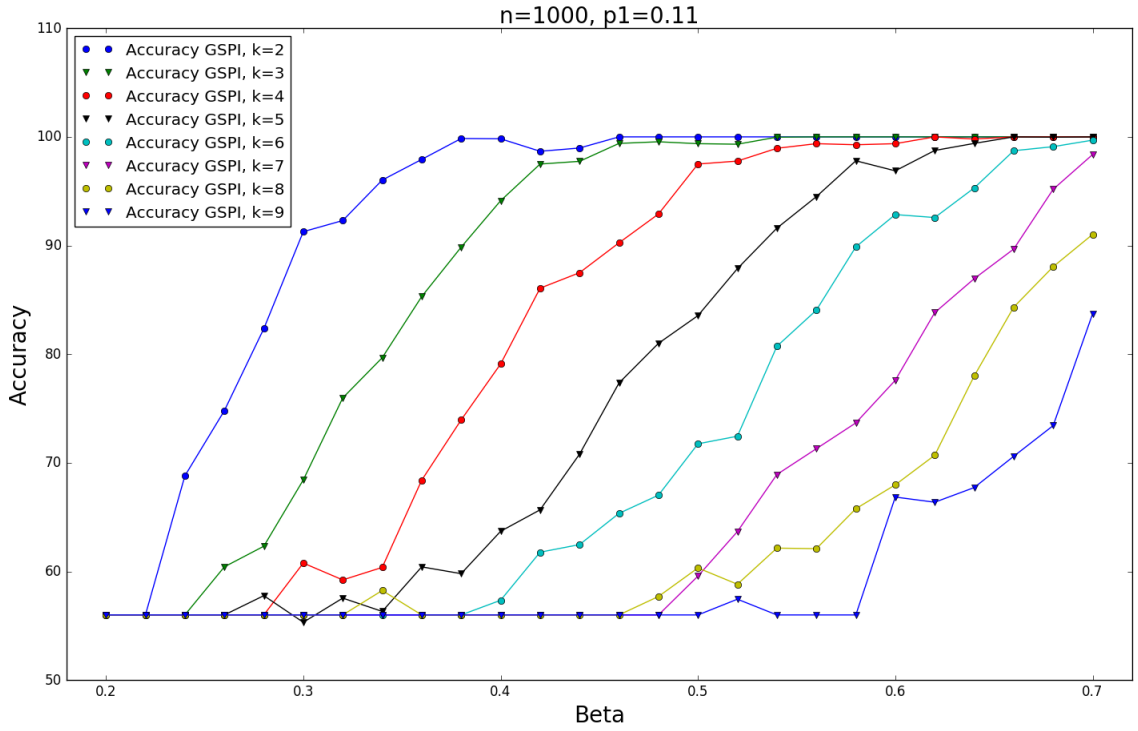

Figure 1.1: Accuracy of the GSPI kernel when $p = 0.11$.

Which means that the slope of the accuracy of the GSPI kernel (as a function of β) should be proportional to $\sqrt{p/k}$. We managed to find experimental results which supports this conjecture. In Fig. 1.2, we have plotted these results. Figure. 1.2 contains the slopes of the accuracies of the GSPI kernel for the datasets where $p \in \{0.11,0.12,0.13,0.14,0.15\}, k \in \{3,4,5,6\}$. These slopes are plotted against $\sqrt{p/k}$ which according to our conjecture should determine the slope of the accuracies. We also plotted a line which is the slope that we predict based on

$$\text{Slope} \propto \sqrt{p/k}.$$

Where we predict that the slope is a function $g(x) = cx$, and we fit it to the slopes from our experiments using the least squares method. As can be seen in Fig. 1.2, the slope values which we obtain from our experiments are reasonably close to what we predict based on our conjecture and we take this as evidence for our conjecture.

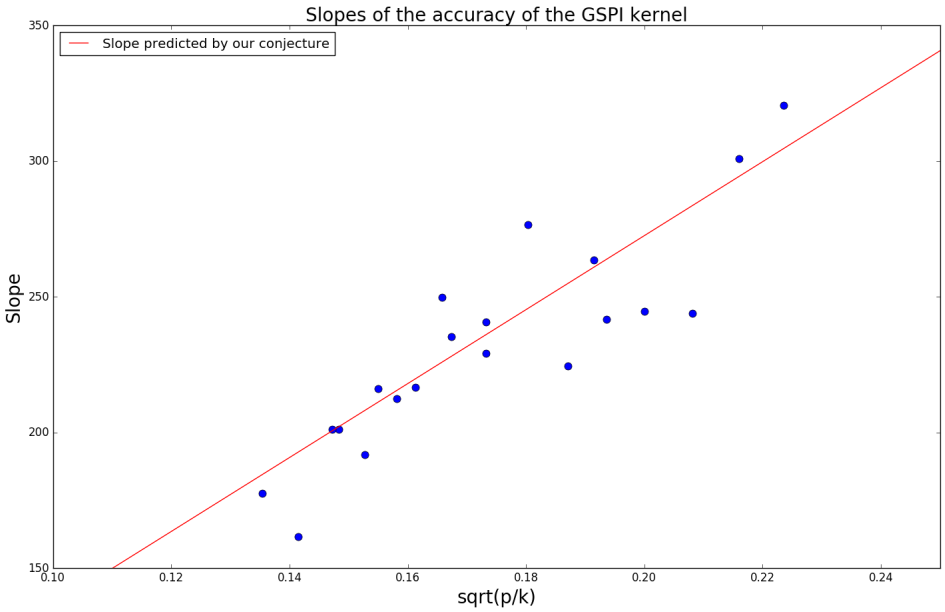

Figure 1.2: The slopes of the accuracy of the GSPI kernel for $p \in \{0.11,0.12,0.13,0.14,0.15\}$, $k \in \{3,4,5,6\}$ and the slope as predicted by our conjecture that the slope is proportional to $\sqrt{p/k}$.

1.5 Thesis Outline

The rest of the thesis is outlined as follows. Chapter 2 contains the preliminaries of the thesis. Chapter 3 contains a detailed summary of all the work by other researchers that this thesis builds upon, it also introduces our own original work, the generalized shortest path kernel. Chapter 4 contains the first main problem handled in the thesis, the problem of classifying if a graph is a one-cluster or a two-cluster graph. Chapter 5 contains the second main problem of the thesis, the problem of classifying if a graph is a k-cluster graph or a $k+1$-cluster graph, where k is greater than or equal to 2. Chapter 6 contains our conclusions and suggestions for future work.

2

Preliminaries

This chapter presents the preliminaries of the thesis.

2.1 Notation

In this section we define the notations used throughout the thesis. For a fixed graph, G,V, and E denote the graph, the set of vertices, and set of edges respectively. We denote the number of vertices and edges in a particular graph by the symbols n and m. We denote vectors using lower case bold characters, for instance $\mathbf{x}$. We denote a particular element of a vector using a subscript, for instance x_i is element number i of the vector $\mathbf{x}$. When we have several versions of related vectors, we distinguish them using a superscript $^{(i)}$. So that for instance $\mathbf{x}^{(i)}$ is vector number i of the $\mathbf{x}$ vectors. We denote the euclidean norm of a vector by $||\mathbf{x}||$. We denote the transpose of a vector with the superscript T, for instance $\mathbf{x}^T$. We denote matrices using upper case bold letters, for example $\mathbf{K}$. K_{ij} refers to the element at row i and column j of the matrix $\mathbf{K}$. By $\mathbf{e}$ we denote a vector of all ones, the length of which can be inferred from the context.

Since our approach is based on graph kernels, (see Sect. 3.2 for details) which count the number of vertex pairs in graphs that are at particular distances and have a particular number of shortest paths, we define necessary notations for the feature vectors of these graph kernels so that we can appropriately discuss the relevant

graph properties. We denote the number of vertex pairs, that have a shortest path of length $d \geq 1$, in a graph G, by n_d. For d,$x \geq 1$, by $n_{d,x}$, we denote the number of vertex pairs that are at distance d and have x number of shortest paths. Let $D(G)$ denote the multi set of shortest distances between all vertex pairs in the graph G. by $ND(G)$ we denote the multi set of numbers of shortest paths between all vertex pairs of G. For any given graph G, We call a vector $\boldsymbol{v}_{\text{sp}} = [n_1,n_2,\ldots]$ a *SPI feature vector*. For any given graph G, We call a vector $\boldsymbol{v}_{\text{gsp}} = [n_{1,1},n_{1,2},\ldots,n_{2,1}\ldots]$ a *GSPI feature vector*. Note that $n_d = \sum_x n_{d,x}$. We sometimes in our analysis use feature vectors where we consider shortest paths from a fixed vertex in a graph, instead of all vertex pairs. Whenever we use such a version of the feature vectors we point it out in the text. In this thesis we consider different random models for generating graphs. For any specific such model, $\mathrm{E}[\,\boldsymbol{v}_{\text{sp}}\,] = [\mathrm{E}[n_1],\mathrm{E}[n_2],\ldots]$, denotes the expected SPI feature vector. We denote the expected GSPI feature vector by $\mathrm{E}[\,\boldsymbol{v}_{\text{gsp}}\,] = [\mathrm{E}[n_{1,1}],\mathrm{E}[n_{1,2}]\ldots,\mathrm{E}[n_{2,1}]\ldots]$.

In this thesis we consider two main different problems, each with two different random models for generating random graphs (see Chap. 4 and 5) The first main problem (Chap. 4), is about predicting if a graph contains one or two clusters. For these random graphs, the SPI/GSPI feature vectors are random vectors. For each $z \in \{0,1\}$, we use $\boldsymbol{v}_{\text{sp}}^{(z)}$ and $\boldsymbol{v}_{\text{gsp}}^{(z)}$ to denote random SPI and GSPI feature vectors of a z-cluster graph. We use $n_d^{(z)}$ and $n_{d,x}^{(z)}$ to denote respectively the dth and (d,x)th component of $\boldsymbol{v}_{\text{sp}}^{(z)}$ and $\boldsymbol{v}_{\text{gsp}}^{(z)}$. For our experiments and analysis, we consider their expectations $\mathrm{E}[\,\boldsymbol{v}_{\text{sp}}^{(z)}\,]$ and $\mathrm{E}[\,\boldsymbol{v}_{\text{gsp}}^{(z)}\,]$, that is, $[\mathrm{E}[n_d^{(z)}]]_{d\geq 1}$ and $[\mathrm{E}[n_{d,x}^{(z)}]]_{d\geq 1,x\geq 1}$. Note that $\mathrm{E}[n_{d,x}^{(z)}]$ is the expected *number of vertex pairs* that have x number of shortest paths of length d; not to be confused with the expected number of distance d shortest paths. The second main problem (see Chap. 5), is about predicting if a graph contains k or $k+1$ number of clusters, where $k \geq 2$. Here again, the SPI/GSPI feature vectors are random vectors. In order to separate k and $k+1$-cluster graphs, we use a super script $^{(k)}$ or $^{(k+1)}$. So that for instance $n_{2,1}^{(k)}$ is the number of vertex pairs that have a shortest path of length 2 and exactly 1 shortest path, *in a k-cluster graph*. We use the same notation of separating the k and $k+1$ cluster graphs for the expected feature vectors so that $\mathrm{E}[\,\boldsymbol{v}_{\text{gsp}}^{(k)}\,]$ and $\mathrm{E}[\,\boldsymbol{v}_{\text{gsp}}^{(k+1)}\,]$ are the expected GSPI feature vectors for k-cluster graphs and $k+1$-cluster graphs, for any specific random model of generating the random graphs. We also consider vertices from particular clusters, we define that $n_d^{(y,k)}$ is the number of vertices at distance d, in a k cluster graph, *that are in cluster y only*.

2.2 Inclusion-Exclusion Principle

Here we give an approximation of the inclusion-exclusion principle, which is used in Chap. 4. This approximation comes from [12], here for completeness, we state this approximation as a lemma and give its proof that is outlined in [12].

Lemma 1. *Let E_1,E_2,...,E_l be mutually independent events such that* $\Pr[E_i] \leq \epsilon$ *holds for all i, $1 \leq i \leq l$. Then we have*

$$\Pr\left[\bigcup_{i=1}^{l} E_i\right] = 1 - \exp\left(-\sum_{i=1}^{l} \Pr[E_i]\right) - Q. \tag{2.1}$$

Where

$$-\sum_{k=0}^{l+1} \frac{(l\epsilon)^k}{k!} + (1+\epsilon)^l \leq Q \leq \sum_{k=0}^{l+1} \frac{(l\epsilon)^k}{k!} - (1+\epsilon)^l. \tag{2.2}$$

Remark. The above bound for the error term Q is slightly weaker than the one in [12], but it is sufficient enough for many situations, in particular for our usage.

Proof. Using the definition of the inclusion-exclusion principle we get

$$\Pr\left[\bigcup_{i=1}^{l} E_i\right] = \sum_{k=1}^{l} (-1)^{k+1} S(k), \tag{2.3}$$

where each $S(k)$ is defined by

$$S(k) = \sum_{1 \leq i_1 < \ldots < i_k \leq l} \Pr[E_{i_1}]\Pr[E_{i_2}] \cdots \Pr[E_{i_k}] = \sum_{1 \leq i_1 < \ldots < i_k \leq l} P_{i_1} P_{i_2} \cdots P_{i_k}.$$

Here and in the following we denote each probability $\Pr[E_i]$ simply by P_i.

First we show that

$$S(k) = \frac{1}{k!} \left(\sum_{i=1}^{l} P_i\right)^k - Q_k, \tag{2.4}$$

where

$$0 \leq Q_k \leq \left(\frac{l^k}{k!} - \binom{l}{k}\right) \epsilon^k. \tag{2.5}$$

To see this we introduce two index sequence sets Γ_k and Π_k defined by

$$\begin{aligned}\Gamma_k &= \{\,(i_1,\ldots,i_k) : i_j \in \{1,\ldots,l\} \text{ for all } j,\, 1 \le j \le k\,\},\\ \Pi_k &= \{\,(i_1,\ldots,i_k) \in \Gamma_k : i_j \neq i_{j'} \text{ for all } j,j',\, 1 \le j < j' \le k\,\}.\end{aligned}$$

Then it is easy to see that

$$\left(\sum_{i=1}^{l} P_i\right)^k = \sum_{(i_1,\ldots,i_k)\in\Gamma_k} P_{i_1}\cdots P_{i_k}, \quad \text{and} \quad k!S(k) = \sum_{(i_1,\ldots,i_k)\in\Pi_k} P_{i_1}\cdots P_{i_k}.$$

Thus, we have

$$\begin{aligned}k!Q_k = \left(\sum_{i=1}^{l} P_i\right)^k - k!S(k) &= \sum_{(i_1,\ldots,i_k)\in\Gamma_k\setminus\Pi_k} P_{i_1}\cdots P_{i_k}\\ &\le |\Gamma_k \setminus \Pi_k|\epsilon^k = \big(l^k - l(l-1)\cdots(l-k+1)\big)\epsilon^k,\end{aligned}$$

which gives bound (2.5) for Q_k.

Now from (2.3) and (2.4) we have

$$1 - \Pr\left[\bigcup_{i=1}^{l} E_i\right] = \sum_{k=0}^{l} \frac{(-1)^k}{k!}\left(\sum_{i=1}^{l} P_i\right)^k + \sum_{k=1}^{l}(-1)^{k+1}Q_k.$$

Here we note that the sum $\sum_{k=0}^{l}(-1)^k/k!(\sum_{i=1}^{l} P_i)^k$ is the first $l+1$ terms of the MacLaurin expansion of $\exp(-\sum_{i=1}^{l} P_i)$. Hence, the error term Q of (2.1) becomes

$$Q = -\sum_{k\ge l+1} \frac{(-1)^k}{k!}\left(\sum_{i=1}^{l} P_i\right)^k + \sum_{k=1}^{l}(-1)^{k+1}Q_k.$$

We now derive an upper bound for Q.

$$\begin{aligned}Q &\le -\sum_{k\ge l+1} \frac{(-1)^k}{k!}\left(\sum_{i=1}^{l} P_i\right)^k + \sum_{k=1}^{l} Q_k\\ &\le \frac{(l\epsilon)^{l+1}}{(l+1)!} + \sum_{k=1}^{l}\frac{(l\epsilon)^k}{k!} - \sum_{k=1}^{l}\binom{l}{k}\epsilon^k\\ &= \sum_{k=0}^{l+1}\frac{(l\epsilon)^k}{k!} - (1+\epsilon)^l.\end{aligned}$$

This proves the upper bound on Q, the proof for the lower bound of Q is completely analogous. Thus, the lemma holds. □

3

Related Work

This chapter gives a detailed summary of related works built upon by this thesis.

3.1 Support Vector Machines

A *support vector machine* (SVM) is a tool for supervised machine learning, where we want to separate two different classes of data. The modern form of the SVM was introduced in 1992 [6] and the currently most popular form (soft margin SVM) was introduced in 1995 [11].

In its current form, and the form used in this thesis, the SVM separates examples by finding a separating hyperplane by solving an optimization problem. In particular, the SVM finds the maximum margin hyperplane for any given set of training examples. By a set of training examples we mean a set of pairs $(\mathbf{x}^{(i)}, y_i)$, where $\mathbf{x}^{(i)} \in \mathbb{R}^d$ is a vector and $y_i \in \{+1, -1\}$, is a label which contains the class of the training example. The hyperplane that the SVM finds can be described by $\mathbf{w}^T\mathbf{x} + b = 0$. Where $\mathbf{w}$ is the normal to the hyperplane, $\mathbf{x}$ is the set of points in the hyperplane, $\frac{b}{||\mathbf{w}||}$ is the perpendicular distance from the hyperplane to the origin and b is a bias term. The margin is then the distance from the hyperplane to the training examples that lie the closest to the margin (several of them can be at equal distance from the hyperplane). The training examples which lie exactly on the margin (the training examples closest to the hyperplane) are called the *support vectors*.

An SVM can be trained by solving the optimization problem given below, called the SVM primal problem. Note that in this thesis, we consider the soft margin version of the SVM, which allows some training examples to be misclassified, and we can control how important we consider it to be, to not misclassify training examples, using the variable C in the optimization problem below.

$$\begin{aligned} \underset{\mathbf{w},b,\xi}{\text{minimize}} \quad & \frac{1}{2}||\mathbf{w}||^2 + C\sum_{i=1}^{l}\xi_i \\ \text{subject to} \quad & y_i(\mathbf{w}^T\mathbf{x}^{(i)} + b) \geq 1 - \xi_i, \\ & \xi_i \geq 0, i = 1, \ldots, l. \end{aligned} \tag{3.1}$$

Where l is the number of examples in the training set and C is the regularization parameter which determines how much we penalize misclassifying training examples. It should be noted here that the separating hyperplane obtained by solving (3.1) will be linear, which can cause problems when we are trying to classify data which is not linearly separable in the given feature space. Because of this, something which is often used is the so called kernel trick.

3.1.1 Kernel Trick

A kernel is a function which can be used to help an SVM solve a non-linearly separable problem. A kernel function $K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \langle\phi(\mathbf{x}^{(i)}), \phi(\mathbf{x}^{(j)})\rangle_{\mathcal{H}} = K_{ij}$, is a function where $\phi(\mathbf{x})$ is a map from the feature space to a Hilbert space. Hilbert spaces can be of infinite dimension and also we may perform dot products in Hilbert spaces [26]. By using the mapping ϕ to a higher dimensional space, we are able to find a linear separating hyperplane in the space to which ϕ maps, instead of in the feature space itself. This means that as long as the problem is linearly separable in the space to which ϕ maps, the SVM will be able to find a linearly separable solution to the problem, which then can be seen as non-linear in the original feature space, see Fig. 3.1 and 3.2 for an example of such a solution. In Fig. 3.1, we see data points from two different classes, the two classes obviously cannot be linearly separated in the shown 2D space. Figure. 3.2 shows the same data points mapped into a 3D space, where they now can easily be linearly separated.

Because of this, instead of solving the original problem (3.1), it is useful to solve the same problem but include the fact that we may also use a kernel function, so that we

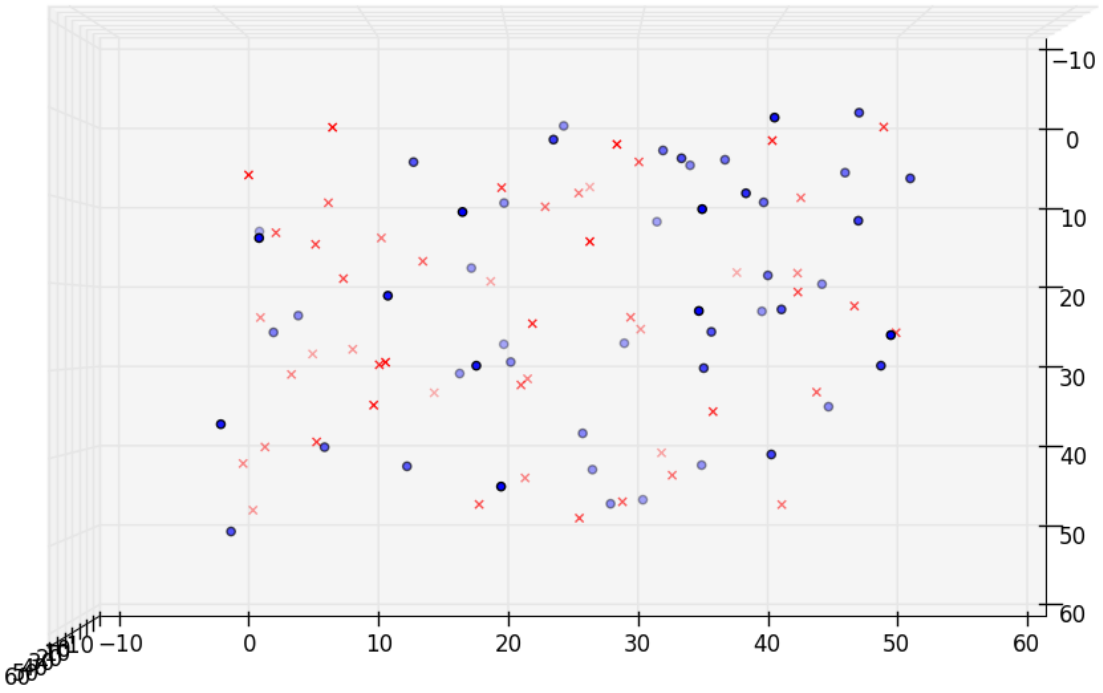

Figure 3.1: A set of points from two different classes, where points from class 1 are displayed as circles and points from class 2 are displayed as xs. In this image it appears as though all the points are in a 2D space, and thus, there is no linear separator for the two classes.

are able to map the data into a higher dimensional space. The optimization problem then becomes

$$\begin{aligned} \underset{\mathbf{w},b,\xi}{\text{minimize}} \quad & \frac{1}{2}||\mathbf{w}||^2 + C\sum_{i=1}^{l}\xi_i \\ \text{subject to} \quad & y_i(\mathbf{w}^T\phi(\mathbf{x}^{(i)}) + b) \geq 1 - \xi_i, \\ & \xi_i \geq 0, i = 1, \ldots, l. \end{aligned} \tag{3.2}$$

Where ϕ is some mapping into a higher dimensional space. Note that the kernel function K and the mapping ϕ discussed in this section are different from the main topic of this thesis, namely graph kernels (discussed in Sect. 3.2), which take graphs as input, not vectors.

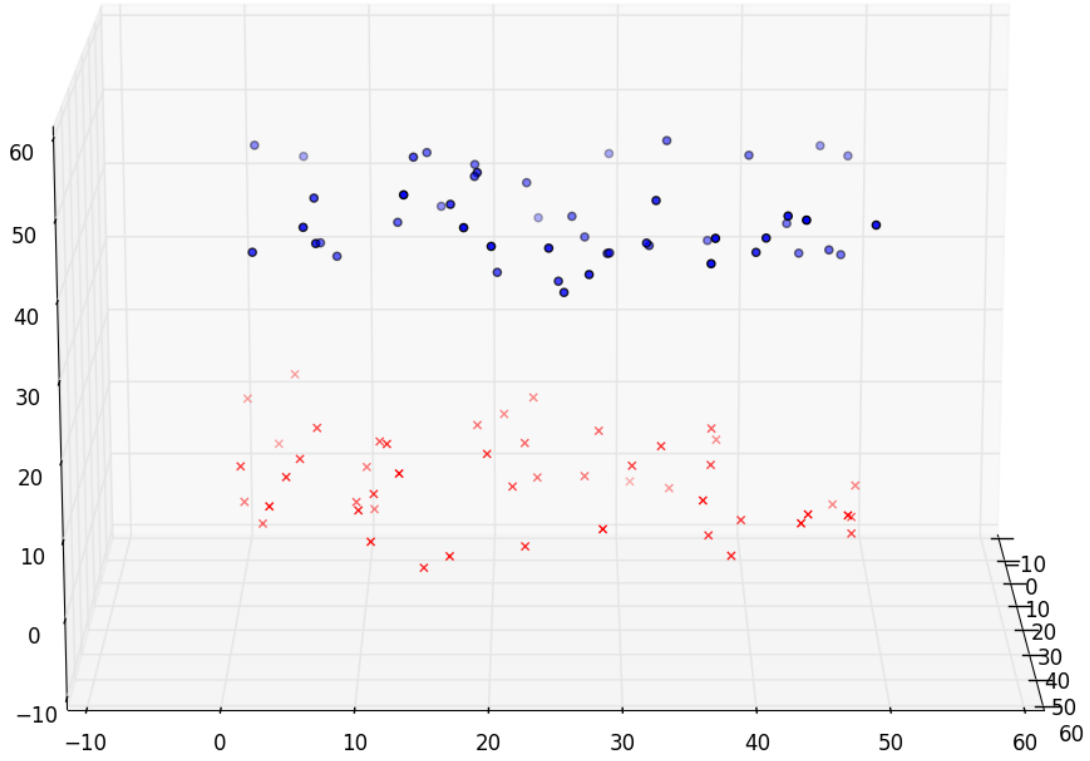

Figure 3.2: A set of points from two different classes, where points from class 1 are displayed as circles and points from class 2 are displayed as xs. In this image we see that the points are in a 3D space, with the circles on top and the xs at the bottom. This means that we can easily find a linear separator of the two classes.

3.1.2 Duality

One possible problem with solving (3.2) is that the dimensionality of $\mathbf{w}$ can be very high, which means that the problem takes a long time to solve. Also if ϕ maps to a very high dimensional space, then this is another reason why solving (3.2) will take a very long time. Because of this it is often more convenient to solve the Lagrangian dual of (3.2). We are able to transform (3.2) to its dual version using Lagrangian

duality [7]. The dual version of (3.2) can be written as the following [8].

$$\begin{aligned} \underset{\boldsymbol{\alpha}}{\text{minimize}} \quad & \frac{1}{2}\boldsymbol{\alpha}^T\mathbf{Q}\boldsymbol{\alpha} - \mathbf{e}^T\boldsymbol{\alpha} \\ \text{subject to} \quad & \sum_{i=1}^{l} y_i\alpha_i = 0, \\ & 0 \leq \alpha_i \leq C, i = 1, \ldots, l. \end{aligned} \tag{3.3}$$

Where $Q_{ij} \equiv y_i y_j K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)})$ and $K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) \equiv \phi(\mathbf{x}^{(i)})^T\phi(\mathbf{x}^{(j)})$ is the kernel function. The representer theorem then states that the optimal hyperplane $\mathbf{w}$ can be written as a linear combination of the training examples in some high dimensional space [19] i.e. we can recover the optimal $\mathbf{w}$ from 3.2 as

$$\mathbf{w} = \sum_{i=1}^{l} y_i\alpha_i\phi(\mathbf{x}^{(i)}). \tag{3.4}$$

Whenever we want to check which side of the hyperplane any particular example is we use the decision function, defined as follows

$$\operatorname{sgn}(\mathbf{w}^T\phi(\mathbf{x} + b)) = \operatorname{sgn}(\sum_{i=1}^{l} y_i\alpha_i K(\mathbf{x}^{(i)}, \mathbf{x}) + b) \tag{3.5}$$

Where $\operatorname{sgn}(b)$ is the sign function.

One thing to note is the fact that it is possible to choose many different versions of the kernel function K. In this thesis, we are mainly interested in analyzing different graph kernels (see Sect. 3.2), and do not want different kernels to influence the results of our graph kernels. Because of this, we do not consider any complicated version of the kernel function K for (3.3). Instead we always consider the basic kernel $K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \langle \mathbf{x}^{(i)}, \mathbf{x}^{(j)} \rangle$. Which is equivalent to not using a kernel at all. We do however keep the kernel notation in order to be consistent with other relevant material and for future extensions.

3.1.3 LIBSVM

How does one then go about solving problem (3.3)? For this task, several different free and proprietary software are available. One of the most popular solvers of the

optimization problem (3.3) is LIBSVM [8]. LIBSVM is an open source software which can freely be downloaded and modified by anybody who wishes to solve the SVM optimization problem (3.3). LIBSVM uses the fact that (3.3) is a concave optimization problem with linear constraints in order to find the solution. For a detailed description of the LIBSVM algorithm, please see [8].

3.2 Graph Kernels

For two graphs, G_1 and G_2, a graph kernel is a function $k(G_1,G_2)$ on pairs of graphs, which can be represented as an inner product $k(G_1,G_2) = \langle \phi(G_1), \phi(G_2) \rangle_{\mathcal{H}}$ for some mapping $\phi(G)$ to a Hilbert space $\mathcal{H}$, of possibly infinite dimension. It is convenient to think of graph kernels as similarity functions on graphs. Graph kernels have been used as tools for SVM classifiers for several graph classification problems [4, 5, 16]. It is important to note that the graph kernels investigated in this thesis all use a mapping ϕ to a *finite* dimensional space. Since this is the case, instead of thinking about inner products in infinite dimensions, we can simply think of our graph kernels as functions that take a graph (e.g. G_1) and produce a finite dimensional feature vector ($\phi(G_1)$). The kernel function itself (k), can then be computed by simply taking the inner product of any pair of these finite dimensional vectors.

We have now established that our graph kernels can simply be thought of as inner products between finite dimensional vectors, we note however that it is common to still call this technique graph kernels [4, 16, 18], and thus we keep to this convention and call these finite dimensional inner products, graph kernels, in order to maintain the connection to other researchers.

As explained above, our graph kernels are basically ways to represent a graph by a vector. It is possible to consider many different ways of how to represent any particular graph by a vector. It is generally regarded that the number of self-loop-avoiding paths between all pairs of vertices of a given graph is useful for understanding the structure of the graph [14, 23]. Computing the number of such paths between all vertices is however a computationally hard task (usually #P-hard). Counting only the number of shortest paths between vertex pairs is however possible in polynomial time and such paths at least avoid cycles, which is why some researchers have considered shortest paths a reasonable substitute.

In this thesis we consider two key properties of graphs in order to represent them as vectors, the shortest path length between vertex pairs, and the number of shortest

paths between vertex pairs. One simple way of calculating these two properties for any graph and for any pair of vertices, is to use a slightly modified version of Dijkstra's algorithm from each vertex in the graph. This algorithm is described in Alg. 1.

Algorithm 1 is a modified version of Dijkstra's algorithm where $\mathbf{L}$ is an $n \times n$ matrix which contains the shortest path length between all vertex pairs. $\mathbf{N}$ is an $n \times n$ matrix which contains the number of shortest paths between all vertex pairs. For simplicity we say that $\mathbf{L}$ and $\mathbf{N}$ are indexed from 1 to n. Meaning that L_{00} does not exist. When we loop over vertices we start from v_1 and end with v_n so that, for instance, in the first iteration of the loop on line 4, $v_i = v_1$. Q is a min priority queue with three operations, insert, get_min and decrease_priority. Typically we would use the fastest type of min priority queue for Q, which is a Fibonacci heap. The three operations on Q are standard, where insert(v, prio), inserts an element v into Q with a certain the priority prio. get_min(), extracts the element from Q which has the lowest priority. decrease_priority(v, prio) decreases the priority of an element v to a new value prio. We also use the notation distance(v_i,v_j) in Alg. 1, by this we simply mean the distance between the two neighboring vertices v_i and v_j. Note that this is always 1 for unweighted graphs, which is the type of graphs handled in our experiments.

The asymptotic running time of Alg. 1 is $\mathcal{O}(nm + n^2 \log n)$. A key point is that the asymptotic running time of Alg. 1 would have been the same even if we did not calculate the number of shortest paths between each vertex pair. In fact if we did not calculate the number of shortest paths between all vertex pairs, then Alg. 1 would simply be the standard Dijkstra's algorithm run from each vertex in the graph. This means that, if we use Dijkstra's algorithm from each vertex in order to get the shortest path length between all vertex pairs, we can get the number of shortest paths between all vertex pairs without any big increase in running time.

3.3 Shortest Path Kernel

A very popular graph kernel which we build upon in this thesis is the so called *shortest path* (SP) kernel. The SP kernel compares graphs based on their shortest paths and was first introduced in [4]. Let $D(G)$ denote the multi set of shortest distances between all vertex pairs in the graph G. For two given graphs G_1 and G_2,

Algorithm 1 MODIFIED DIJKSTRA'S($G = (V,E)$)

1: **Input:** $G = (V,E)$

2: $\mathbf{L} \leftarrow \underbrace{\begin{bmatrix} 0.0 & \dots & 0.0 \\ \vdots & \ddots & \vdots \\ 0.0 & \dots & 0.0 \end{bmatrix}}_{n}\Big\}_{n}$ Contains the SP length between all vertex pairs.

3: $\mathbf{N} \leftarrow \underbrace{\begin{bmatrix} 0.0 & \dots & 0.0 \\ \vdots & \ddots & \vdots \\ 0.0 & \dots & 0.0 \end{bmatrix}}_{n}\Big\}_{n}$ Contains the number of SPs between all vertex pairs.

4: **for** $v_i \in V$ **do**
5: initialize Q to be an empty min priority queue
6: **for** $v_j \in V$ **do**
7: **if** $v_i \neq v_j$ **then**
8: $L_{ij} \leftarrow \infty$
9: **end if**
10: Q.insert(v_j, L_{ij})
11: **end for**
12: **while** Q is not empty **do**
13: $v_j \leftarrow Q$.get_min()
14: **for** each neighbor v_k of v_j **do**
15: new_dist = L_{ij}+ distance(v_j, v_k)
16: **if** new_dist < L_{ik} **then**
17: L_{ik} = new_dist
18: Q.decrease_priority(v_k,new_dist)
19: **if** $v_i = v_j$ **then**
20: $N_{ik} = 1$
21: **else**
22: $N_{ik} = N_{ij}$
23: **end if**
24: **else if** new_dist = L_{ik} **then**
25: **if** $v_i \neq v_j$ **then**
26: $N_{ik} = N_{ik} + N_{ij}$
27: **end if**
28: **end if**
29: **end for**
30: **end while**
31: **end for**
32: Output **L** and **N**.

the SP kernel is then defined as:

$$K_{\mathrm{SP}}(G_1, G_2) = \sum_{d_1 \in D(G_1)} \sum_{d_2 \in D(G_2)} k(d_1, d_2),$$

where k is a positive definite kernel [4]. It is possible to consider several different definitions of the function k. One of the most common version of k is the indicator function, as used in [4]. This kernel compares shortest distances for equality. Using this choice of k we obtain the following definition of the SP kernel:

$$K_{\mathrm{SPI}}(G_1, G_2) = \sum_{d_1 \in D(G_1)} \sum_{d_2 \in D(G_2)} \mathbb{1}\left[d_1 = d_2\right].$$

We call this the *shortest path index* (SPI) kernel. It is easy to check that $K_{\mathrm{SPI}}(G_1, G_2)$ is simply the inner product of the SPI feature vectors of G_1 and G_2. Here we will describe two real world problems where the SP kernel has proven to be very useful.

3.3.1 Enzyme Prediction

A classification task which comes up in bioinformatics is to classify which type of enzyme class a certain protein belongs to [4]. In such a task, we are provided with the spatial data for different proteins, which describes how the protein is built. In the problem which was considered in [4], this spatial data was translated into normal mathematical graphs of vertices and edges. In the graphs each vertex was connected to its three nearest neighbors in space and the edges received a weight which is equal to the distance they represent in Ångströms. This particular dataset consisted of 540 protein graphs, from 6 different enzyme classes, 90 protein graphs from each class.

The way the performance was tested was by using several different types of graph kernels together with an SVM and 10-fold cross validation. Since there are not just 2 types of enzyme classes, but 6, a "one-class vs. rest" classification was performed and the average accuracy over all 6 classes was used as the final accuracy. Several versions of the SP kernel were investigated, and also version of the random walk kernel (not investigated in this thesis). It was found that the SPI kernel achieved an accuracy of 93.52% on this classification task. Meaning that the SPI kernel was almost able to perfectly solve the real world bioinformatics problem of predicting the enzyme class of proteins, for this particular dataset.

3.3.2 Entity Disambiguation

Another real world problem which the SP kernel has proven to work well for is that of entity disambiguation. In the particular version of entity disambiguation considered in [16], we are given neighborhood graphs that either belongs to one person (entity) or several persons. Each vertex in a graph corresponds to one or several persons and each edge represents that two persons have been mentioned together in at least one article. The weights of the edges are equal to the number of times the persons were mentioned together in articles. Each graph which we are asked to classify is the neighborhood graph of one specific vertex (could be one or several persons). The goal is to classify if the neighborhood graph represents one or several persons.

The problem arises because when the database about which persons which are mentioned together in articles is generated, different persons can have the same name. Which means that when this data is gathered automatically, several persons could be regarded as just one person. One example is Chris Anderson. There is one Chris Anderson which was the editor-in-chief of Wired magazine and another Chris Anderson which is the curator of TED conferences. If information about which persons are mentioned together in articles is gather automatically, it is not easily possible to distinguish if the person being mentioned in a particular article is the former editor-in-chief of Wired magazine or the curator of TED conferences. As a result, it would be useful to be able to scan the database of persons mentioned together in articles, and classify which persons in the database actually refers to several persons. For this problem a version of the SP kernel was used in order to classify such neighborhood graphs of persons.

The dataset consisted of 91 neighborhood graphs with a label each, which contains the information if the neighborhood graph corresponds to one or several persons. The average size for the neighborhood graphs were 267 vertices, 5,830 edges and 39.6% of the graphs in the datasets were labeled as representing several persons. For this problem, the SPI kernel was able to achieve a 73.0% accuracy when using an SVM and 10-fold cross validation. Another, slightly modified, but closely related version of the SP kernel was able to achieve 82.0% accuracy [16].

3.3.3 Edge Density Comparison

Other researchers have considered problems which are similar to the classification tasks which we consider in Chap. 4 and 5. In [10], the problem of identifying the k

clusters of a k-cluster graph, where $k = 5$ and the graphs are generated according to the same model as in Chap. 5, is considered and various methods are investigated. From our experimental results given in Chap. 5, we found that $\beta = 0.5$ is enough for the generalized shortest path kernel to solve the classification problem given in Chap. 5 when $n = 1000, k = 5, p = 0.15$, with close to 100% accuracy. In [10] they consider the same parameters, but want to identify the actual 5 clusters, instead of classifying the number of clusters as either 5 or 6. For this problem they investigate 4 different algorithms, Single-Linkage clustering, spectral clustering, low-rank-plus-sparse, and their own algorithm. For this problem they found that Single-Linkage clustering and low-rank-plus-sparse were unable to solve the problem at all when $p = 0.15$, and required much larger values of p. Spectral clustering required $\beta > 0.8$ and their own method required $\beta > 0.6$. When considering the case when $p = 0.11$ the results were similar. Single-Linkage clustering and low-rank-plus-sparse were unable to solve the problem at all. Spectral clustering required $\beta > 0.86$ and their own method required $\beta > 0.68$. For these parameters our method solved the classification problem of determining if the graph contains 5 or 6 clusters, with close to 100% accuracy, when $\beta > 0.56$. It should be noted that for their problem they assume that it is known that the number of clusters is equal to 5. One possible useful application of our classification problem, as described in Chap. 5, is to actually classify the number of clusters in the graph in preparation for the algorithm given in [10].

It should be noted that the generalized shortest path kernel is able to solve our classification task for relatively low values of p as compared to other approaches which try to identify clusters in graphs. For instance, as stated above, for the problem of identifying the 5 clusters, both Single-Linkage clustering and low-rank-plus-sparse were unable to solve the problem at all when $p = 0.15$ or $p = 0.11$. In [24] they consider the case of identifying the two clusters in a two-cluster graph, generated according to the model given in Chap. 4. Where they consider the case where $n = 64$, $q = 0.15$, and vary p. Their approach is not able to identify the two clusters, with high accuracy, unless $p > 0.75$, which is indeed a lot higher than the values of p which we need to consider in our experiments.

3.4 Generalized Shortest Path Kernel

Although it is not a related work, rather it is our original work, we here give the definition of the generalized shortest path kernel. The *generalized shortest path* (GSP) kernel, is defined using the shortest path length and number of shortest paths be-

tween all vertex pairs. For a given graph G, by $ND(G)$ we denote the multi set of numbers of shortest paths between all vertex pairs of G. The GSP kernel is then defined as:

$$K_{\mathrm{GSP}}(G_1, G_2) = \sum_{d_1 \in D(G_1)} \sum_{d_2 \in D(G_2)} \sum_{t_1 \in ND(G_1)} \sum_{t_2 \in ND(G_2)} k(d_1,d_2,t_1,t_2),$$

where k is a positive definite kernel. Similarly to the SPI kernel, one obvious choice of k is to chose a function that consider vertex pairs as equal if they have the same shortest path length *and* the same number of shortest paths. This gives us the following definition, called the *generalized shortest path index* (GSPI) kernel.

$$K_{\mathrm{GSPI}}(G_1, G_2) = \sum_{d_1 \in D(G_1)} \sum_{d_2 \in D(G_2)} \sum_{t_1 \in ND(G_1)} \sum_{t_2 \in ND(G_2)} \mathbb{1}\left[d_1 = d_2\right] \mathbb{1}\left[t_1 = t_2\right].$$

It is easy to see that this is equivalent to the inner product of the GSPI feature vectors of G_1 and G_2.

Computing the SPI and GSPI feature vectors can be done efficiently using Alg. 1. Doing so takes $\mathcal{O}(nm + n^2 \log n)$ time for one graph and gives us the information needed to construct SPI and GSPI feature vectors. Note that with this method, the running time for computing an SPI feature vector is the same as the running time of computing an GSPI feature vector.

4

One-Cluster Graphs and Two-Cluster Graphs

In this chapter we consider random models for generating one-cluster graphs and two-cluster graphs. Our task is to distinguish such random graphs using our SVM based approach together with the SP kernel and the GSP kernel. We provide experimental results which show that the GSP kernel outperforms the SP kernel for several datasets. We also provide an analysis of the SP and GSP kernels for the relevant random graph generation models.

4.1 Random Graph Models

We investigate the advantage of our GSPI kernel over the SPI kernel for a synthetic random graph classification problem. Our target problem is to distinguish random graphs having two relatively "dense parts", from simple graphs generated by the Erdős-Rényi model. Here by "dense part" we mean a subgraph that has more edges in its inside compared with its outside.

For any edge density parameter p, $0 < p < 1$, the Erdős-Rényi model (with parameter p) denoted by $G(n,p)$ is to generate a graph G (of n vertices) by putting an edge between each pair of vertices with probability p independently at random. On the other hand, for any p and q, $0 < q < p < 1$, the *planted partition model* [20], denoted

by $G(n/2,n/2,p,q)$ is to generate a graph $G = (V^+ \cup V^-, E)$ (with $|V^+| = |V^-| = n/2$) by putting an edge between each pair of vertices u and v again independently at random with probability p if both u and v are in V^+ (or in V^-) and with probability q if $u \in V^+$ and $v \in V^-$ (or, $u \in V^-$ and $v \in V^+$).

In this chapter we use the symbol p_1 to denote the edge density parameter for the Erdős-Rényi model and p_2 and q_2 to denote the edge density parameters for the planted partition model. We want to have $q_2 < p_2$ while keeping the expected number of edges the same for both random graph models (so that one cannot distinguish random graphs by just couting the number of edges). It is easy to check that this requirement is satisfied by setting

$$p_2 = (1+\alpha)p_1, \quad \text{and} \quad q_2 = 2p_1 - p_2 - 2(p_1 - p_2)/n \tag{4.1}$$

for some constant α, $0 < \alpha < 1$. We consider the "sparse" situation for our experiments and analysis, and assume that $p_1 = c_0/n$ for a sufficiently large constant c_0. Note that we may expect with high probability, that when c_0 is large enough, a random graph generated by both models have a large connected component but might not be fully connected [3]. A random graph generated by $G(n,p_1)$ is called a *one-cluster graph* and a random graph generated by $G(n/2,n/2,p_2,q_2)$ is called a *two-cluster graph*.

For a random graph, the SPI/GSPI feature vectors are random vectors. For each $z \in \{0,1\}$, we use $\boldsymbol{v}_{\text{sp}}^{(z)}$ and $\boldsymbol{v}_{\text{gsp}}^{(z)}$ to denote random SPI and GSPI feature vectors of a z-cluster graph. We use $n_d^{(z)}$ and $n_{d,x}^{(z)}$ to denote respectively the dth and (d,x)th component of $\boldsymbol{v}_{\text{sp}}^{(z)}$ and $\boldsymbol{v}_{\text{gsp}}^{(z)}$. For our experiments and analysis, we consider their expectations $\mathrm{E}[\,\boldsymbol{v}_{\text{sp}}^{(z)}\,]$ and $\mathrm{E}[\,\boldsymbol{v}_{\text{gsp}}^{(z)}\,]$, that is, $[\mathrm{E}[n_d^{(z)}]]_{d\geq 1}$ and $[\mathrm{E}[n_{d,x}^{(z)}]]_{d\geq 1, x\geq 1}$. Note that $\mathrm{E}[n_{d,x}^{(z)}]$ is the expected *number of vertex pairs* that have x number of shortest paths of length d; not to be confused with the expected number of distance d shortest paths.

4.2 Experiments

In this section we compare the performance of the GSPI kernel with the SPI kernel on datasets where the goal is to classify if a graph is a one-cluster graph or a two-cluster graph.

4.2.1 Generating Datasets and Experimental Setup

All datasets in this chapter are generated using the models $G(n,p_1)$ and $G(n/2,n/2,p_2,q_2)$, described above. We generate 100 graphs from the two different classes in each dataset. q_2 is chosen in such a way that the expected number of edges is the same for both classes of graphs. Note that when $p_2 = p_1$, the two-cluster graphs actually become one-cluster graphs where all vertex pairs are connected with the same probability, meaning that the two classes are indistinguishable. The bigger difference there is between p_1 and p_2, the more different the one-cluster graphs are compared to the two-cluster graphs. In our experiments we generate graphs where $n \in \{200,400,600,800,1000\}$, $np_1 = c_0 = 40$ and $p_2 \in \{1.2p_1, 1.3p_1, 1.4p_1, 1.5p_1\}$. Hence $p_1 = 0.2$ for $n = 200$, $p_1 = 0.1$ for $n = 400$ etc.

In all experiments we calculate the normalized feature vectors for all graphs. By normalized we mean that each feature vector $\boldsymbol{v}_{\text{sp}}$ and $\boldsymbol{v}_{\text{gsp}}$ is normalized by its Euclidean norm. This means that the inner product between two feature vectors always is in [0,1]. We then train an SVM using 10-fold cross validation and evaluate the accuracy of the kernels. For the experiments in this chapter we used Pegasos [27], in order to solve the SVM.

4.2.2 Results

Table 4.1 shows the accuracy of both kernels, using 10-fold cross validation, on the different datasets. As can be seen neither of the kernels perform very well on the datasets where $p_2 = 1.2p_1$. This is because the two-cluster graphs generated in this dataset are almost the same as the one-cluster graphs. As p_2 increases compared to p_1, the task of classifying the graphs becomes easier. As can be seen in the table the GSPI kernel outperforms the SPI kernel on nearly all datasets. In particular, on datasets where $p_2 = 1.4p_1$, the GSPI kernel has an increase in accuracy of over 20% on several datasets. When $n = 200$ the increase in accuracy is over 40%! Although the shown results are only for datasets where $c_0 = 40$, experiments using other values for c_0 gave similar results.

One reason that our GSPI kernel is able to classify graphs correctly when the SPI kernel is not, is because the feature vectors of the GSPI kernel, for the two classes, are a lot more different than for the SPI kernel. In Fig. 4.1 we have plotted the SPI feature vectors, for a fixed vertex, for both classes of graphs and one particular dataset. By feature vectors for a fixed vertex we mean that the feature vectors

contains information for one fixed vertex, instead of vertex pairs, so that for example, n_d from $\boldsymbol{v}_{\text{sp}} = [n_1,n_2,\ldots]$, contains the number of vertices that are at distance d from one fixed vertex, instead of the number of vertex pairs that are at distance d from each other. The feature vector displayed in Fig. 4.1 is the average feature vector, for any fixed vertex, and averaged over the 100 randomly generated graphs of each type in the dataset. The dataset showed in the figure is when the graphs were generated with $n = 600$, the one-cluster graphs used $p_1 = 0.06667$, the two-cluster graphs used $p_2 = 0.08667$ and $q_2 = 0.04673$, this corresponds to, in Tbl. 4.1, the dataset where $n = 600$, $p_2 = 1.3p_1$, this dataset had an accuracy of 60.5% for the SPI kernel and 67.0% for the GSPI kernel. As can be seen in the figure there is almost no difference at all between the average SPI feature vectors for the two different cases. In Fig. 4.2 we have plotted the subvectors $[n^{(1)}_{2,x}]_{x\geq 1}$ of $\boldsymbol{v}^{(1)}_{\text{gsp}}$ and $[n^{(2)}_{2,x}]_{x\geq 1}$ of $\boldsymbol{v}^{(2)}_{\text{gsp}}$, for a fixed vertex, for the same dataset as in Fig. 4.1. The vectors contain the number of vertices at distance 2 from the fixed vertex with x number of shortest paths, for one-cluster graphs and two-cluster graphs respectively. The vectors have been averaged for each vertex in the graph and also averaged over the 100 randomly generated graphs, for both classes of graphs, in the dataset. As can be seen the distributions of such numbers of vertices are at least distinguishable for several values of x, when comparing the two types of graphs. This motivates why the SVM is able to distinguish the two classes better using the GSPI feature vectors than the SPI feature vectors.

4.3 Analysis

In this section we give some approximated analysis of random feature vectors in order to give theoretical support for our experimental observations. We first show that one-cluster and two-cluster graphs have quite similar SPI feature vectors (as their expectations). Then we next show some evidence that there is a non-negligible difference in their GSPI feature vectors. Throughout this section, we consider feature vectors defined by considering only paths from any fixed source vertex s. Thus, for example, $n^{(1)}_d$ is the number of vertices at distance d from s in a one-cluster graph, and $n^{(2)}_{d,x}$ is the number of vertices that have x shortest paths of length d to s in a two-cluster graph.

Here we introduce a way to state an approximation. For any functions a and b depending on n, we write $a \approx_{\text{rel}} b$ by which we mean

$$b\left(1-\frac{c}{n}\right) < a < b\left(1+\frac{c}{n}\right)$$

Table 4.1: The accuracy of the SPI kernel and the GSPI kernel using 10-fold cross validation. The datasets where $p_2 = 1.2p_1$ are the hardest and the datasets where $p_2 = 1.5p_1$ are the easiest. Very big increases in accuracy are marked in bold.

Kernel	n	p_2	Accuracy
SPI	200	$\{1.2p_1, 1.3p_1, 1.4p_1, 1.5p_1\}$	$\{52.5\%, 55.5\%, \mathbf{54.5}\%\mathbf{56.5}\%\}$
GSPI	200	$\{1.2p_1, 1.3p_1, 1.4p_1, 1.5p_1\}$	$\{52.5\%, 64.0\%, \mathbf{99.0}\%, \mathbf{100.0}\%\}$
SPI	400	$\{1.2p_1, 1.3p_1, 1.4p_1, 1.5p_1\}$	$\{55.5\%, 63.5\%, \mathbf{75.5}\%, 95.5\%\}$
GSPI	400	$\{1.2p_1, 1.3p_1, 1.4p_1, 1.5p_1\}$	$\{54.0\%, 62.0\%, \mathbf{96.5}\%, 100.0\%\}$
SPI	600	$\{1.2p_1, 1.3p_1, 1.4p_1, 1.5p_1\}$	$\{58.0\%, 60.5\%, \mathbf{75.5}\%, 93.5\%\}$
GSPI	600	$\{1.2p_1, 1.3p_1, 1.4p_1, 1.5p_1\}$	$\{58.0\%, 67.0\%, \mathbf{94.0}\%, 100.0\%\}$
SPI	800	$\{1.2p_1, 1.3p_1, 1.4p_1, 1.5p_1\}$	$\{57.5\%, 59.0\%, 72.0\%, 98.0\%\}$
GSPI	800	$\{1.2p_1, 1.3p_1, 1.4p_1, 1.5p_1\}$	$\{57.5\%, 58.0\%, 82.0\%, 100.0\%\}$
SPI	1000	$\{1.2p_1, 1.3p_1, 1.4p_1, 1.5p_1\}$	$\{53.5\%, 55.0\%, \mathbf{66.0}\%, 98.5\%\}$
GSPI	1000	$\{1.2p_1, 1.3p_1, 1.4p_1, 1.5p_1\}$	$\{55.0\%, 62.0\%, \mathbf{87.5}\%, 100.0\%\}$

holds for some constant $c > 0$ and sufficiently large n. We say that a and b are *relatively* $(1 \pm \mathcal{O}(1/n))$*-close* if $a \approx_{\mathrm{rel}} b$ holds. Note that this closeness notion is closed under constant number of additions/subtractions and multiplications. For example, if $a \approx_{\mathrm{rel}} b$ holds, then we also have $a^k \approx_{\mathrm{rel}} b^k$ for any $k \geq 1$ that can be regarded as a constant w.r.t. n. In the following we will often use this approximation.

4.3.1 Approximate Comparison of SPI Feature Vectors

We consider relatively small[1] distances d so that d can be considered as a small constant w.r.t. n. We show that $\mathrm{E}[n_d^{(1)}]$ and $\mathrm{E}[n_d^{(2)}]$ are similar in the following sense.
Theorem 1. *For any constant d, we have $\mathrm{E}[n_d^{(1)}] \in \mathrm{E}[n_d^{(2)}](1 \pm \frac{2}{c_0 - 1})$, holds within our $\approx_{\mathrm{rel}}$ approximation when $c_0 \geq 2 + \sqrt{3}$.*

Remark. For deriving this relation we assume a certain independence on the existence of two paths in G; see the argument below for the detail. Note that this difference between $\mathrm{E}[n_d^{(1)}]$ and $\mathrm{E}[n_d^{(2)}]$ vanishes for large values of c_0.

[1]This smallness assumption is for our analysis, and we believe that the situation is more or less the same for any d.

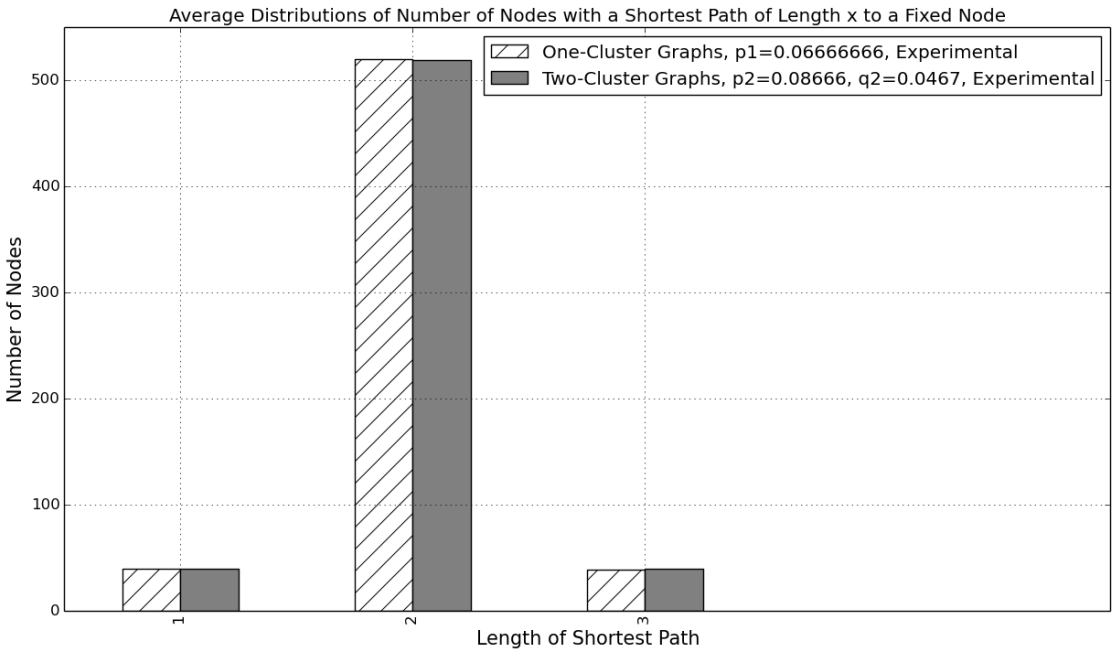

Figure 4.1: Average distributions of number of vertices with a shortest path of length x to a fixed vertex. The distributions have been averaged for each vertex in the graph and also averaged over 100 randomly generated graphs, for both classes of graphs. The graphs used the parameters $n = 600$, $p_1 = 0.06667$ for one-cluster graphs and $p_2 = 0.08667$, $q_2 = 0.04673$ for two-cluster graphs.

Proof. First consider a one-cluster graph $G = (V,E)$ and analyze $\mathrm{E}[n_d^{(1)}]$. For this analysis, consider any target vertex t ($\neq s$) of G (we consider this target vertex to be a fixed vertex to begin with), and estimate first the probability F_d that there exists at least one path of length d from s to t. Let $A_{u,v}$ be the event that an edge $\{u,v\}$ exists in G, and let W_d denote the set of all paths (from s to t) expressed by a permutation $(v_1,\ldots,v_{d-1})$ of vertices in $V \setminus \{s,t\}$. For each tuple $(v_1,\ldots,v_{d-1})$ of W_d, the event $A_{s,v_1} \wedge A_{v_1,v_2} \wedge \ldots \wedge A_{v_{d-1},t}$ is called the event that the path (from s to t) specified by $(s,v_1,\ldots,v_{d-1},t)$ exists in G (or, more simply, *the existence of one specific path*). Then the probability F_d is expressed by

$$F_d = \Pr\left[\bigvee_{(v_1,\ldots,v_{d-1})\in W_d} A_{s,v_1} \wedge A_{v_1,v_2} \wedge \ldots \wedge A_{v_{d-1},t}\right] \tag{4.2}$$

Clearly, the probability of the existence of one specific path is p_1^d, and the above probability can be calculated by using the inclusion-exclusion principle. Here we

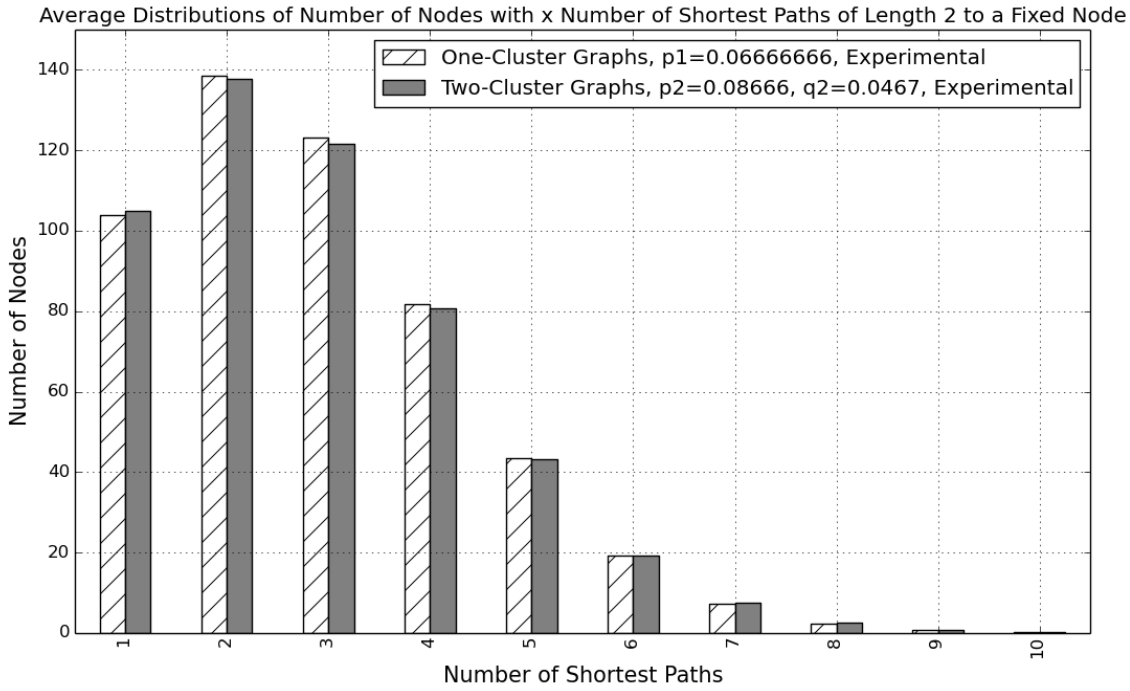

Figure 4.2: Average distributions of number of vertices with x number of shortest paths of length 2 to a fixed vertex. The distributions have been averaged for each vertex in the graph and also averaged over 100 randomly generated graphs, for both classes of graphs. The graphs used to generate this figure are the same as in Fig. 4.1.

follow the analysis of Fronczak et al. [12] and assume that every specific path exists independently. Note that the number of dependent paths can be big when the length of a path is long, therefore this assumption is only reasonable for short distances d. To simplify the analysis[2] we only consider the first term of the inclusion-exclusion principle. That is, we approximate F_d by

$$\begin{aligned} F_d &\approx \sum_{(v_1,\ldots,v_{d-1})\in W_d} \Pr\left[A_{s,v_1} \wedge A_{v_1,v_2} \wedge \ldots \wedge A_{v_{d-1},t} \right] \qquad (4.3) \\ &= |W_d| p_1^d = (n-2)(n-3)\cdots(n-(2+d-2)) p_1^d \approx_{\text{rel}} n^{d-1} p_1^d, \end{aligned}$$

where the last approximation relation holds since d is constant. From this approximation, we can approximate the probability f_d that t has a shortest path of length d

[2]Clearly, this is a rough approximation; nevertheless, it is enough for our asymptotic analysis w.r.t. the $(1 \pm \mathcal{O}(1/n))$-closeness. For smaller n, we may use the better approximation which was explained in Sect. 2.2.

to s. For any $d \geq 1$, let A_d be the event that there exists at least one path of length d, *or less*, between s and t, and let B_d be the event that there exists a shortest path of length d between s and t. Then

$$F_d \leq \Pr[A_d] \leq \sum_{i=1}^{d} F_i. \tag{4.4}$$

Note that

$$\begin{aligned} \sum_{i=1}^{d} F_i \quad &\approx_{\text{rel}} \quad \sum_{i=1}^{d} n^{i-1} p_1^i = n^{d-1} p_1^d (\sum_{i=0}^{d-1} \frac{1}{(np_1)^i}) \\ &\leq \quad n^{d-1} p_1^d (\frac{np_1}{np_1 - 1}) = n^{d-1} p_1^d (1 + \frac{1}{np_1 - 1}). \end{aligned}$$

Since $np_1 = c_0 \geq 1$, it follows that

$$n^{d-1} p_1^d (1 + \frac{1}{np_1 - 1}) = n^{d-1} p_1^d (1 + \frac{1}{c_0 - 1}).$$

While $F_d \approx_{\text{rel}} n^{d-1} p_1^d$. Thus we have within our $\approx_{\text{rel}}$ approximation, that

$$n^{d-1} p_1^d \leq \Pr[A_d] \leq n^{d-1} p_1^d (1 + \frac{1}{c_0 - 1}).$$

It is obvious that $f_d = \Pr[B_d]$, note also that $A_d = B_d \vee A_{d-1}$ and that the two events B_d and A_{d-1} are disjoint. Thus, we have $\Pr[A_d] = \Pr[B_d] + \Pr[A_{d-1}]$, which is equivalent to

$$n^{d-1} p_1^d - n^{d-2} p_1^{d-1} (1 + \frac{1}{c_0 - 1}) \leq f_d \leq n^{d-1} p_1^d (1 + \frac{1}{c_0 - 1}) - n^{d-2} p_1^{d-1}.$$

Since f_d is the probability that there is a shortest path of length d from s to any fixed t, it follows that $\mathrm{E}[n_d^{(1)}]$, i.e., the expected number of vertices that have a shortest path of length d to s, can be estimated by

$$\mathrm{E}[n_d^{(1)}] = (n-1) f_d \approx_{\text{rel}} n f_d.$$

Which gives that

$$n^d p_1^d - n^{d-1} p_1^{d-1} (1 + \frac{1}{c_0 - 1}) \leq \mathrm{E}[n_d^{(1)}] \leq n^d p_1^d (1 + \frac{1}{c_0 - 1}) - n^{d-1} p_1^{d-1}. \tag{4.5}$$

holds within our $\approx_{\text{rel}}$, approximation.

We may rewrite the above equation using the following equalities

$$\begin{aligned} n^d p_1^d - n^{d-1} p_1^{d-1}(1 + \frac{1}{c_0 - 1}) &= n^d p_1^d - n^{d-1} p_1^{d-1} - \frac{n^{d-1} p_1^{d-1}}{c_0 - 1} \\ &= (n^d p_1^d - n^{d-1} p_1^{d-1})(1 - \frac{\frac{n^{d-1} p_1^{d-1}}{c_0 - 1}}{n^d p_1^d - n^{d-1} p_1^{d-1}}) \\ &= (n^d p_1^d - n^{d-1} p_1^{d-1})(1 - \frac{1}{(c_0 - 1)^2}), \end{aligned} \tag{4.6}$$

and

$$\begin{aligned} n^d p_1^d (1 + \frac{1}{c_0 - 1}) - n^{d-1} p_1^{d-1} &= n^d p_1^d - n^{d-1} p_1^{d-1} + \frac{n^d p_1^d}{c_0 - 1} \\ &= (n^d p_1^d - n^{d-1} p_1^{d-1})(1 + \frac{\frac{n^d p_1^d}{c_0 - 1}}{n^d p_1^d - n^{d-1} p_1^{d-1}}) \\ &= (n^d p_1^d - n^{d-1} p_1^{d-1})(1 + \frac{c_0}{(c_0 - 1)^2}). \end{aligned} \tag{4.7}$$

Substituting (4.6) and (4.7) into (4.5) we get

$$(n^d p_1^d - n^{d-1} p_1^{d-1})(1 - \frac{1}{(c_0 - 1)^2}) \leq \mathrm{E}[n_d^{(1)}] \leq (n^d p_1^d - n^{d-1} p_1^{d-1})(1 + \frac{c_0}{(c_0 - 1)^2}). \tag{4.8}$$

We will later use these bounds to derive the theorem.

We now analyze a two-cluster graph and $\mathrm{E}[n_d^{(2)}]$. Recall that, for two-cluster graphs, we use the notation $V = V^+ \cup V^-$ (with $|V^+| = |V^-| = n/2$). Let us assume first that s is in V^+. Again we fix a target vertex t to begin with. Here we need to consider the case that the target vertex t is also in V^+ and the case that it is in V^-. Let F_d^+ and F_d^- be the probabilities that t has at least one path of length d to s in the two cases. Then for the first case, the path starts from $s \in V^+$ and ends in $t \in V^+$, meaning that the number of times that the path crossed from one cluster to another (either from V^+ to V^- or V^- to V^+) has to be even. Thus the probability of one specific path existing is $p_2^{d-k} q_2^k$ for some even k, $0 \leq k \leq d$. Thus, the first term of the inclusion-exclusion principle (the sum of the probabilities of all possible paths) then becomes

$$F_d^+ \approx_{\text{rel}} \left(\frac{n}{2}\right)^{d-1} \sum_{\text{even } k=0}^{d} \binom{d}{k} p_2^{d-k} q_2^k,$$

where the number of paths is approximated as before, i.e., $|V^+ \setminus \{s,t\}| \cdot (|V^+ \setminus \{s,t\}| - 1) \cdots ((|V^+ \setminus \{s,t\}| - (d-2))$ is approximated by $(n/2)^{d-1}$. We can similarly analyze the case where t is in V^- to obtain

$$F_d^- \approx_{\text{rel}} \left(\frac{n}{2}\right)^{d-1} \sum_{\text{odd } k=1}^{d} \binom{d}{k} p_2^{d-k} q_2^k.$$

Since both cases ($t \in V^+$, or $t \in V^-$) are equally likely, the average probability of there being a path of length d, between s and t, in a two-cluster graph is

$$\begin{aligned} \frac{F_d^+ + F_d^-}{2} &\approx_{\text{rel}} \left(\frac{n}{2}\right)^{d-1} \sum_{\text{even } k=0}^{d} \binom{d}{k} \frac{p_2^{d-k} q_2^k}{2} + \left(\frac{n}{2}\right)^{d-1} \sum_{\text{odd } k=1}^{d} \binom{d}{k} \frac{p_2^{d-k} q_2^k}{2} \\ &= \left(\frac{n}{2}\right)^{d-1} \sum_{k=0}^{d} \binom{d}{k} \frac{p_2^{d-k} q_2^k}{2} = \left(\frac{n}{2}\right)^{d-1} \frac{(p_2+q_2)^d}{2}. \end{aligned}$$

Note here that $p_2 + q_2 \approx_{\text{rel}} 2p_1$ from our choice of q_2 (see (4.1)). Thus, we have

$$\frac{F_d^+ + F_d^-}{2} \approx_{\text{rel}} \left(\frac{n}{2}\right)^{d-1} \frac{(2p_1)^d}{2} = n^{d-1} p_1^d.$$

Which is exactly the same as in the one cluster case, see (4.3). Thus we have

$$(n^d p_1^d - n^{d-1} p_1^{d-1})(1 - \frac{1}{(c_0-1)^2}) \leq \mathrm{E}[n_d^{(2)}] \leq (n^d p_1^d - n^{d-1} p_1^{d-1})(1 + \frac{c_0}{(c_0-1)^2}). \quad (4.9)$$

Using this we now prove the main statement of the theorem, namely that

$$\mathrm{E}[n_d^{(1)}] \in \mathrm{E}[n_d^{(2)}](1 \pm \frac{2}{c_0 - 1}).$$

To prove the theorem we need to prove the following two things

$$\mathrm{E}[n_d^{(1)}] \leq \mathrm{E}[n_d^{(2)}](1 + \frac{2}{c_0 - 1}), \text{ and} \quad (4.10)$$

$$\mathrm{E}[n_d^{(1)}] \geq \mathrm{E}[n_d^{(2)}](1 - \frac{2}{c_0 - 1}) \quad (4.11)$$

The proof of (4.10) can be done by using (4.8) and (4.9).

$$\begin{aligned}
&\mathrm{E}[n_d^{(1)}] \le \mathrm{E}[n_d^{(2)}] + \frac{2\mathrm{E}[n_d^{(2)}]}{c_0 - 1} \\
&\Leftrightarrow (n^d p_1^d - n^{d-1} p_1^{d-1})(1 + \frac{c_0}{(c_0-1)^2}) \le (n^d p_1^d - n^{d-1} p_1^{d-1})(1 - \frac{1}{(c_0-1)^2}) \\
&\quad + \frac{2(n^d p_1^d - n^{d-1} p_1^{d-1})(1 - \frac{1}{(c_0-1)^2})}{c_0 - 1} \\
&\Leftrightarrow 1 + \frac{c_0}{(c_0-1)^2} \le 1 - \frac{1}{(c_0-1)^2} + \frac{2}{c_0-1} - \frac{2}{(c_0-1)^3} \\
&\Leftrightarrow \frac{c_0+1}{c_0-1} + \frac{2}{(c_0-1)^2} \le 2. \qquad (4.12)
\end{aligned}$$

Which holds when $c_0 \ge 2 + \sqrt{3} \approx 3.7$. The proof of (4.11) is similar and shown below.

$$\begin{aligned}
&\mathrm{E}[n_d^{(1)}] \ge \mathrm{E}[n_d^{(2)}] - \frac{2\mathrm{E}[n_d^{(2)}]}{c_0 - 1} \\
&\Leftrightarrow (n^d p_1^d - n^{d-1} p_1^{d-1})(1 - \frac{1}{(c_0-1)^2}) \ge (n^d p_1^d - n^{d-1} p_1^{d-1})(1 + \frac{c_0}{(c_0-1)^2}) \\
&\quad - \frac{2(n^d p_1^d - n^{d-1} p_1^{d-1})(1 - \frac{1}{(c_0-1)^2})}{c_0 - 1} \\
&\Leftrightarrow 1 - \frac{1}{(c_0-1)^2} \ge 1 + \frac{c_0}{(c_0-1)^2} - \frac{2}{c_0-1} + \frac{2}{(c_0-1)^3} \\
&\Leftrightarrow 2 \ge \frac{c_0+1}{c_0-1} + \frac{2}{(c_0-1)^2}. \qquad (4.13)
\end{aligned}$$

Which again holds when $c_0 \ge 2 + \sqrt{3} \approx 3.7$. This completes the proof of the theorem. □

4.3.2 Heuristic Comparison of GSPI Feature Vectors

We compare in this section the expected GSPI feature vectors $\mathrm{E}[\,\boldsymbol{v}_{\mathrm{gsp}}^{(1)}\,]$ and $\mathrm{E}[\,\boldsymbol{v}_{\mathrm{gsp}}^{(2)}\,]$, that is, $[\mathrm{E}[n_{d,x}^{(1)}]]_{d\ge1,x\ge1}$ and $[\mathrm{E}[n_{d,x}^{(2)}]]_{d\ge1,x\ge1}$, and show evidence that they have some non-negligible difference. Here we focus on the distance $d = 2$ part of the GSPI feature vectors, i.e., subvectors $[\mathrm{E}[n_{2,x}^{(z)}]]_{x\ge1}$ for $z \in \{1,2\}$. Since it is not so easy to

analyze the distribution of the values $\mathrm{E}[n_{2,1}^{(z)}]$,$\mathrm{E}[n_{2,2}^{(z)}]$, . . ., we introduce some "heuristic" analysis.

We begin with a one-cluster graph G, and let V_2 denote the set of vertices of G with distance 2 from the source vertex s. Consider any t in V_2, and for any $x \geq 1$, we estimate the probability that it has x number of shortest paths of length 2 to s. Let V_1 be the set of vertices at distance 1 from s. Recall that G has $(n-1)f_1 \approx_{\mathrm{rel}} np_1$ vertices in V_1 on average, and we assume that t has an edge from some vertex in V_1 each of which corresponds to a shotest path of distance 2 from s to t. We now assume for our "heuristic" analysis that $|V_1| = np_1$ and that an edge between each of these distance 1 vertices and t exists with probability p_1 independently at random. Then x follows the binomial distribution $\mathrm{Bin}(np_1,p_1)$, where by $\mathrm{Bin}(N,p)$ we mean a random number of heads that we have when flipping a coin that gives heads with probability p independently N times. Then for each $x \geq 1$, $\mathrm{E}[n_{2,x}^{(1)}]$, the expected number of vertices of V_2 that have x shortest paths of length 2 to s, is estimated by

$$\mathrm{E}[n_{2,x}^{(1)}] \approx \sum_{t \in V_2} \Pr\big[\,\mathrm{Bin}(np_1,p_1) = x\,\big] = \mathrm{E}[n_2^{(1)}] \cdot \Pr\big[\,\mathrm{Bin}(np_1,p_1) = x\,\big],$$

by assuming that $|V_2|$ takes its expected value $\mathrm{E}[n_2^{(1)}]$. Clearly the distribution of values of vector $[\mathrm{E}[n_{2,x}^{(1)}]]_{x \geq 1}$ is proportional to $\mathrm{Bin}(np_1,p_1)$, and it has one peak at $x_{\mathrm{peak}}^{(1)} = np_1^2$, since the mean of a binomial distribution, $\mathrm{Bin}(N,p)$ is Np.

Consider now a two-cluster graph G. We assume that our start vertex s is in V^+. For $d \in \{1,2\}$, let V_d^+ and V_d^- denote respectively the set of vertices in V^+ and V^- with distance d from s. Let $V_2 = V_2^+ \cup V_2^-$. Again we assume that V_1^+ and V_1^- have respectively $np_2/2$ and $nq_2/2$ vertices and that the numbers of edges from V_1^+ and V_1^- to a vertex in V_2 follow binomial distributions. Note that we need to consider two cases here, $t \in V_2^+$ and $t \in V_2^-$. First consider the case that the target vertex t is in V_2^+. In this case there are two types of shortest paths. The first type of paths goes from s to V_1^+ and then to $t \in V_2^+$. The second type of shortest path goes from s to V_1^- and then to $t \in V_2^+$. Based on this we get

$$\begin{aligned}
\widetilde{f}_{2,x}^{(2,+)} &:= \Pr\big[\,t \text{ has } x \text{ shortest paths}\,\big] \\
&= \Pr\left[\,\mathrm{Bin}\left(\frac{n}{2}p_2,p_2\right) + \mathrm{Bin}\left(\frac{n}{2}q_2,q_2\right) = x\,\right] \\
&\approx \Pr\left[\,\mathrm{N}\left(\frac{n}{2}p_2^2,\sigma_1^2\right) + \mathrm{N}\left(\frac{n}{2}q_2^2,\sigma_2^2\right) \in [x-0.5,x+0.5]\,\right] \\
&= \Pr\left[\,\mathrm{N}\left(\frac{n(p_2^2+q_2^2)}{2},\sigma_1^2+\sigma_2^2\right) \in [x-0.5,x+0.5]\,\right],
\end{aligned}$$

where we use the normal distribution $\mathrm{N}(\mu,\sigma^2)$ to approximate each binomial distribution so that we can express their sum by a normal distribution (here we omit specifying σ_1 and σ_2). For the second case where $t \in V_2^-$, with a similar argument, we derive

$$\widetilde{f}_{2,x}^{(2,-)} := \Pr\big[\, t \text{ has } x \text{ shortest paths} \,\big] = \Pr\left[\, \mathrm{N}\left(np_2q_2,\sigma_3^2+\sigma_4^2\right) \in [x-0.5,x+0.5] \,\right].$$

Note that the first case ($t \in V_2^+$), happens with probability $|V_2^+|/(|V_2^+|+|V_2^-|)$. The second case ($t \in V_2^-$), happens with probability $|V_2^-|/(|V_2^+|+|V_2^-|)$. Then again we may approximate the xth component of the expected feature subvector by

$$\mathrm{E}[n_{2,x}^{(2)}] \approx \frac{\mathrm{E}[|V_2^+|]}{\mathrm{E}[|V_2^+|]+\mathrm{E}[|V_2^-|]}\mathrm{E}[|V_2^+|]\widetilde{f}_{2,x}^{(2,+)} + \frac{\mathrm{E}[|V_2^-|]}{\mathrm{E}[|V_2^+|]+\mathrm{E}[|V_2^-|]}\mathrm{E}[|V_2^-|]\widetilde{f}_{2,x}^{(2,-)}.$$

We have now arrived at the key point in our analysis. Note that the distribution of values of vector $[\mathrm{E}[n_{2,x}^{(2)}]]_{x\geq 1}$ follows the mixture of two distributions, namely, $\mathrm{N}(n(p_2^2+q_2^2)/2,\sigma_1^2+\sigma_2^2)$ and $\mathrm{N}(np_2q_2,\sigma_3^2+\sigma_4^2)$, with weights $\mathrm{E}[|V_2^+|]/(\mathrm{E}[|V_2^+|]+\mathrm{E}[|V_2^-|])$ and $\mathrm{E}[|V_2^-|]/(\mathrm{E}[|V_2^+|]+\mathrm{E}[|V_2^-|])$. Now we estimate the distance between the two peaks $x_{\mathrm{peak}}^{(2,+)}$ and $x_{\mathrm{peak}}^{(2,-)}$ of these two distributions. Note that the mean of a normal distribution $\mathrm{N}(\mu,\sigma^2)$ is simply μ. Then we have

$$\begin{aligned} x_{\mathrm{peak}}^{(2,+)} - x_{\mathrm{peak}}^{(2,-)} &= \frac{n}{2}(p_2^2+q_2^2) - np_2q_2 = \frac{n}{2}(p_2-q_2)^2 \\ &\approx_{\mathrm{rel}} \frac{n}{2}(p_2-(2p_1-p_2))^2 = \frac{n}{2}(2p_2-2p_1)^2 \\ &\approx_{\mathrm{rel}} 2n(p_1(1+\alpha)-p_1)^2 = 2n(p_1\alpha)^2 = 2n\alpha^2p_1^2 \end{aligned}$$

Note that $q_2 \approx_{\mathrm{rel}} 2p_1 - p_2$ holds (from (4.1)); hence, we have $p_1 \approx_{\mathrm{rel}} (p_2+q_2)/2 \geq \sqrt{p_2q_2}$, and we approximately have $p_1^2 \geq p_2q_2$. By using this, we can bound the difference between these peaks by

$$x_{\mathrm{peak}}^{(2,+)} - x_{\mathrm{peak}}^{(2,-)} \approx_{\mathrm{rel}} 2n\alpha^2p_1^2 \geq 2\alpha^2 x_{\mathrm{peak}}^{(2,-)}.$$

That is, these peaks have a non-negligible relative difference.

From our heuristic analysis we may conclude that the two vectors $[\mathrm{E}[n_{2,x}^{(1)}]]_{x\geq 1}$ and $[\mathrm{E}[n_{2,x}^{(2)}]]_{x\geq 1}$ have different distributions of their component values. In particular, while the former vector has only one peak, the latter vector has a double peak shape (for large enough α). Note that this difference does not vanish even when c_0 is big. This means that the GSPI feature vectors are different for one-cluster graphs and

two-clusters graphs, even when c_0 is big, which is not the case for the SPI feature vectors, since their difference vanishes when c_0 is big. This provides evidence as to why our GSPI kernel performs better than the SPI kernel

Though this is a heuristic analysis, we can show some examples that our observation is not so different from experimental results. In Fig. 4.3 we have plotted both our approximated vector of $[\mathrm{E}[n_{2,x}^{(2)}]]_{x\geq 1}$ (actually we have plotted the mixed normal distribution that gives this vector) and the corresponding experimental vector obtained by generating graphs according to our random model. In this figure the double peak shape can clearly be observed, which provides empirical evidence supporting our analysis. This experimental vector is the average vector for each fixed source vertex in random graphs, which is averaged over 500 randomly generated graphs with the parameters $n = 400$, $p_2 = 0.18$, and $q_2 = 0.0204$. (For these parameters, we used a better approximation of (4.2) explained in Sect. 2.2 to derive the normal distribution of this figure.)

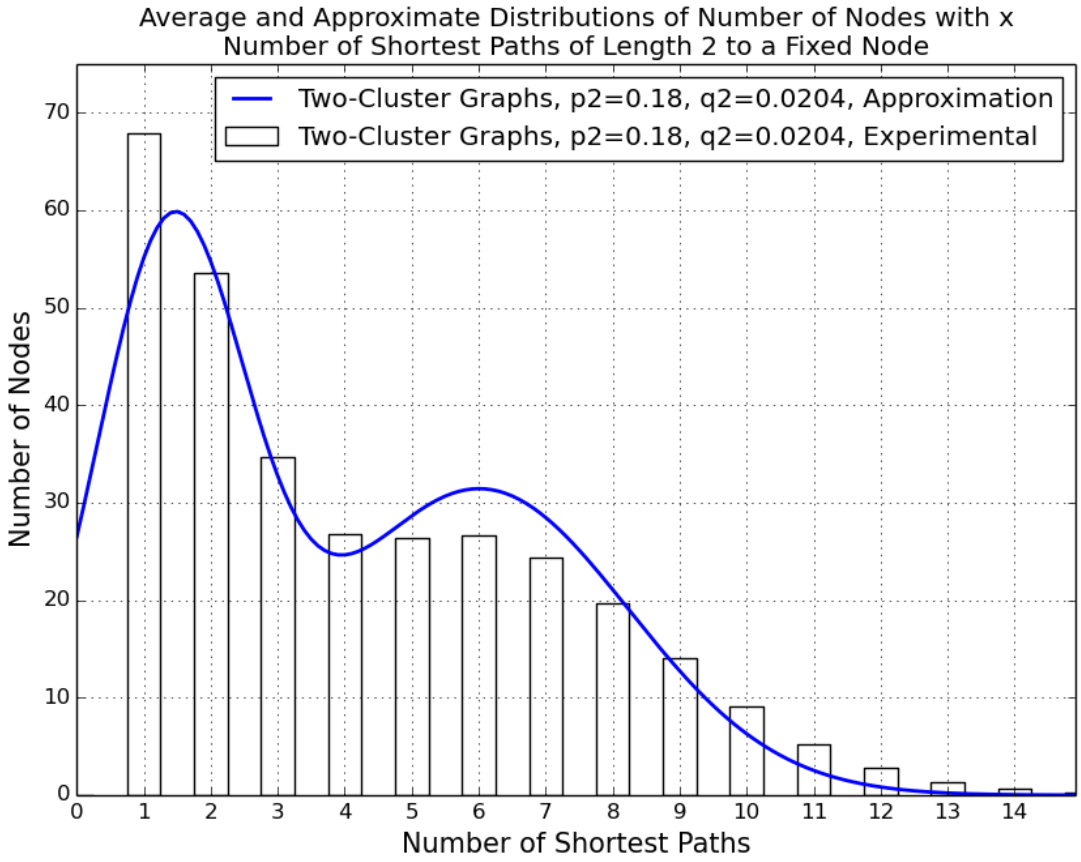

Figure 4.3: Average experimental and approximate distributions of number of vertices with x number of shortest paths of length 2 from a fixed vertex. The experimental distribution has been averaged for each vertex in the graph and also averaged over 500 randomly generated graphs. Graphs used had parameters $n = 400$, $p_2 = 0.18$ and $q_2 = 0.0204$. (For computing the graph of the approximate distribution, we used a more precise approximation for F_d because our n is not large enough, see Sect. 2.2 for details.)

5

k-Cluster Graphs and k+1-Cluster Graphs

In this chapter we analyze the problem of classifying random graphs as having either k or $k+1$ number of clusters, where k is an integer greater than or equal to 2. As in previous chapters, we use our SVM based approach to achieve this, in combination with the SP kernel and the GSP kernel. In this chapter we consider two different random models for generating random graphs, one for the k-cluster graphs and one for the $k+1$-cluster graphs. We give experimental results which show in which situations the GPS kernel outperforms the SP kernel and we also give an analysis about when the GSP kernel is able to solve the problem and when it is not able to do so.

5.1 Random Graph Models and Our Graph Classification Task

A cluster is a subgraph of a graph where the edge density is higher than between the clusters. The graphs which we use in our experiments, in this chapter, are generated according to the *planted partition model* [20], which is an extension of the well known Erdős-Rényi model [3]. In the planted partition model, for a k-cluster graph, all vertices belong to one of k clusters. Each cluster contains n/k number of vertices.

The presence of an edge between any vertex pair is determined randomly. Two vertices that are from the same cluster are connected with the probability p, while vertices that are from different clusters are connected with probability q_1. Where we define q_1 as

$$q_1 = p(1 - \beta).$$

with parameter β which is one of the key parameters in our experiments. We denote the model for generating such graphs as $G(n,k,p,q_1)$.

Each dataset consists of k-cluster graphs and $k+1$-cluster graphs, For the $k+1$-cluster graphs, two vertices that are from the same cluster are connected with probability p, which is the same probability as for the k-cluster graphs. Two vertices that are from different clusters are connected with probability q_2. In order that it is not possible to simply distinguish the different graph types based on the number of edges, we fix q_2 so that the expected number of edges in the k-cluster graphs and the $k+1$-cluster graphs are the same. It is easy to verify that this implies that we have to choose q_2 as

$$q_2 = q_1 + \frac{1}{k^2}(p - q_1) = q_1 + p\frac{\beta}{k^2}.$$

We call the model for generating such graphs $G(n,k+1,p,q_2)$.

We consider the problem to train an SVM using graphs from both models (for a fixed set of parameters p,q_1,q_2,k). We then want to be able to use this SVM to predict, for random graphs which are generated from one of the two models, which of the two models was used to generate each particular graph. I.e. we want our SVM to correctly be able to classify, after training, if a given random graph is a k-cluster graph or a $k+1$-cluster graph.

5.2 Analysis

In this section we analyze in which situations, and why, the GSP kernel outperforms the SP kernel. Further evidence for our analysis can be found in Sect. 5.3, which contains our experimental results. In this section, whenever average feature vectors are mentioned, the average is always over 200 i.i.d. feature vectors.

In this section, instead of considering the number of vertex *pairs* which are at distance d from each other, we consider the number of vertices at a particular distance from a fixed source vertex s. By $n_{d,x}^{(k)}$ we mean the number of vertices that are at distance d from s and have exactly x number of shortest paths to s, in a k-cluster graph.

5.2.1 Preliminary Approximations

Here we give some preliminary approximations that we use in order to derive our main result. We estimate the expected number of vertices at distances 1 and 2, from a fixed vertex s, in a k-cluster graph. Our analysis closely parallels the one in Sect. 4.3.1 Consider the probability that s is connected to some other fixed vertex t. Let $F_{t,1}$ be the probability that s and t are connected. $F_{t,1}$ is obviously p if t is in the same cluster as s and q_1 otherwise. Let $F_{t,2}$ be the probability that there exists at least one path of length 2 between s and t. Let $A_{u,v}$ be the event that u and v are connected. $F_{t,2}$ is then equal to

$$F_{t,2} = \Pr\left[\bigvee_{v \in V \setminus \{s,t\}} A_{s,v} \wedge A_{v,t}\right]. \tag{5.1}$$

$F_{t,2}$ can be calculated using the inclusion-exclusion principle and will give a different result depending on if t is in the same cluster as s or not. Let $f_{t,2}$ denote the probability that s and t are at distance 2. This happens if s and t have a path of length 2 and no path of length 1. I.e. $f_{t,2} = F_{t,2} - F_{t,1}$. Note that $f_{t,2}$,$F_{t,2}$ and $F_{t,1}$ vary depending on where t is. Let f_2 denote the probabillity that s is at distance 2 to a *random* vertex t. f_2 is simply the weighted average sum over the cases when t is in the same cluster as s and when t is not in the same cluster as s. The probability of t being in the same cluster as s is simply $1/k$ and the probability that t is not in the same cluster as s is $(k-1)/k$. With this f_2, we are able to estimate that the expected number of vertices at distance 2 is $(n-1)f_2$.

In order to simplify (5.1), we use an approximation of the inclusion-exclusion principle from Sect. 2.2. I.e. we approximate

$$\Pr\left[\bigcup_{i=1}^{l} E_i\right] \approx 1 - \exp\left(-\sum_{i=1}^{l} \Pr[E_i]\right).$$

Where E_i, in the case of (5.1), is $A_{s,v} \wedge A_{v,t}$.

5.2.2 Analysis of GSPI feature vectors

Here we analyze the expected GSPI feature vectors, $\mathrm{E}[\,\boldsymbol{v}_{\mathrm{gsp}}^{(k)}\,]$ and $\mathrm{E}[\,\boldsymbol{v}_{\mathrm{gsp}}^{(k+1)}\,]$, where the graphs are generated as specified in Sect. 5.1. Here we consider the number of

shortest paths between a fixed source vertex s and a random target vertex t. We focus on the part of the GSPI feature vectors that corresponds to vertices at distance 2 from s, namely the subvectors $[\mathrm{E}[n_{2,x}^{(k)}]]_{x\geq 1}$ and $[\mathrm{E}[n_{2,x}^{(k+1)}]]_{x\geq 1}$. Since we in this section consider feature vectors for a fixed s, note that for instance, $\mathrm{E}[n_{2,4}^{(k)}]$ is the expected number of vertices, that are at distance 2 from s and have 4 number of shortest paths to s in a k-cluster graph.

We first consider a k-cluster graph but we use q instead of q_1 so that we are later able to replace q by either q_1 or q_2. Assume, without loss of generality, that s is in cluster 1. Note that s has an expected $(n/k)p$ number of vertices at distance 1, in cluster 1. s also has an expected $(n/k)q$ number of vertices at distance 1 in each of the other $k-1$ clusters. For our analysis, we assume that the number of vertices at distance 1, in each cluster, is actually equal to the expected number of such vertices. We also assume that each vertex at distance 1 is connected to any vertex at distance 2 *independently* at random. We give evidence later in this section that these are actually reasonable assumptions.

We now analyze the number of shortest paths between s and t, when s and t are at distance 2. We first consider the case when t is in cluster 1. In this case we know that s has an expected $(n/k)p$ vertices at distance 1 in cluster 1. Due to our assumption of independence of each of those $(n/k)p$ vertices having an edge to t, we say that each such vertex is connected to t with probability p independently at random. Which means that if we consider the number of shortest paths between s and t, that goes through cluster 1, the number of shortest paths is distributed according to the binomial distribution $\mathrm{Bin}((n/k)p,p)$. There could also be shortest paths that go from s to cluster 2 then to t, from s to cluster 3 then to t and so on. For any of these situations, the number of shortest paths is distributed according to the distribution $\mathrm{Bin}((n/k)q,q)$. There are obviously $k-1$ such cases. The final distribution of the number of shortest paths when s and t are both in cluster 1 is then

$$\mathrm{Bin}(\frac{np}{k},p) + (k-1)\mathrm{Bin}(\frac{nq}{k},q). \tag{5.2}$$

Similarly, we can calculate that, when t is not in cluster 1, the number of shortest paths between s and t is distributed according to the distribution

$$\mathrm{Bin}(\frac{np}{k},q) + \mathrm{Bin}(\frac{nq}{k},p) + (k-2)\mathrm{Bin}(\frac{nq}{k},q). \tag{5.3}$$

Note here that we have two different cases, the first case where t is in cluster 1 and the second case where t is in any other cluster. Since the position of t is randomly

determined, the final distribution of the number of shortest paths between s and t is a *mixture distribution* [25] of (5.2) and (5.3). Where the weight of the first distribution is the number of vertices at distance 2 in cluster 1 over the total number of vertices at distance 2. Which we write as

$$\frac{n_2^{(1,k)}}{n_2^{(k)}} = w_1^{(k)}.$$

We call this weight $w_1^{(k)}$. The weight of the second distribution is the sum of the number of vertices at distance 2 in all clusters except cluster 1, over the total number of vertices at distance 2. Written as

$$\sum_{i=2}^{k} \frac{n_2^{(i,k)}}{n_2^{(k)}} = w_2^{(k)}.$$

We call this weight $w_2^{(k)}$. Note that $w_1^{(k)} + w_2^{(k)} = 1$. The final mixture distribution, for the number of shortest paths between s and a random vertex t, that is at distance 2 from s, is then

$$\begin{aligned} &w_1^{(k)}(\text{Bin}(\frac{np}{k},p) + (k-1)\text{Bin}(\frac{nq}{k},q)) + \\ &w_2^{(k)}(\text{Bin}(\frac{np}{k},q) + \text{Bin}(\frac{nq}{k},p) + (k-2)\text{Bin}(\frac{nq}{k},q)). \end{aligned} \tag{5.4}$$

This mixture distribution must then have two peaks, the first one being

$$x_{\text{peak}}^{(1,k)} = \frac{n}{k}(p^2 + (k-1)q^2)$$

and the second one being

$$x_{\text{peak}}^{(2,k)} = \frac{n}{k}(2pq + (k-2)q^2).$$

When using the above formulas for a k-cluster graph, we simply replace q by q_1. To obtain the relevant formulas for a $k+1$-cluster graph, we need to replace q by q_2 and k by $k+1$.

We denote the difference between the peaks for a k-cluster graph as

$$\Delta x_{\text{peak}}^{(k)} = x_{\text{peak}}^{(1,k)} - x_{\text{peak}}^{(2,k)} = \frac{np^2\beta^2}{k}. \tag{5.5}$$

Note that the two weights in (5.4) will be different for the k-cluster graphs and the $k+1$-cluster graphs. In particular, for fixed values of p, β and k, we always have $w_2^{(k)} < w_2^{(k+1)}$, since the weight of the second distribution increases when the number of clusters increases (since then there is a lower chance of t being in cluster 1). Also note that $w_1^{(k)}$ decreases when k grows. If $w_1^{(k)}$ becomes negligible, we will not have a two peak shape even if the two peaks are separated by a wide margin. This is however never the case for the parameters we consider in our experiments, i.e. $k \in [2,9]$.

In the above semi-theoretical analysis, we made two noteworthy assumptions. First that the number of vertices at distance 1 from a fixed vertex s, is equal to its expected value. We also assumed that all of the distance 1 vertices are connected to distances 2 vertices independently at random, which finally gave us the distribution (5.4) for the number of shortest paths between s and a random vertex t at distance 2 from s. In order to provide some evidence that this model is actually reasonably close to what really happens, we provide two figures where we compare the subvector $[\mathrm{E}[n_{2,x}^{(k)}]]_{x\geq 1}$ from the experiments and the subvector which we obtain by using (5.4) multiplied by the number of vertices at distance 2. We use the approximation from Sect. 5.2.1 to approximate the number of vertices at distance 2 and substitute the binomial distributions in (5.4) for normal distributions in order to simplify summing the distributions. The results can be seen in Fig. 5.1 and 5.2. As can be seen, the distributions obtained from our analysis are very close to the experimental values, no matter if the distribution appears to have only one peak (the two peaks are very close), or if the two peaks are far away.

It is important to note that if the difference between the peaks is large enough, the distribution will look very different compared to when the difference between the peaks is close to 0. If the difference between the peaks is large, the distribution will look skewed compared to a standard normal distribution. This can be seen in Fig. 5.3, which shows the average subvectors $[\mathrm{E}[n_{2,x}^{(k)}]]_{x\geq 1}$, when $n = 1000$, $p = 0.15$, $k = 2$ for $\beta = 0.2$ and $\beta = 0.7$. Note that the difference between the peaks is a lot greater when $\beta = 0.7$ than when $\beta = 0.2$. In fact the difference between the two peaks is equal to 0.45 when $\beta = 0.2$ and 5.51 when $\beta = 0.7$. The distributions of the k and $k+1$-cluster graphs also appears to look more different when the difference between the peaks is large. Figure 5.4, contains the average subvectors $[\mathrm{E}[n_{2,x}^{(k)}]]_{x\geq 1}$ and $[\mathrm{E}[n_{2,x}^{(k+1)}]]_{x\geq 1}$, when $n = 1000$, $p = 0.15$, $k = 2$, $\beta = 0.7$. The difference between the peaks for the k-cluster graphs is 5.51 and the difference between the peaks for the $k+1$-cluster graphs is 2.07. If the distance between the peaks is low, the feature vectors tend to look the same for k-cluster graphs and $k+1$-cluster graphs. An example of this can be seen in Fig. 5.5. Obviously, for such datasets, it is not

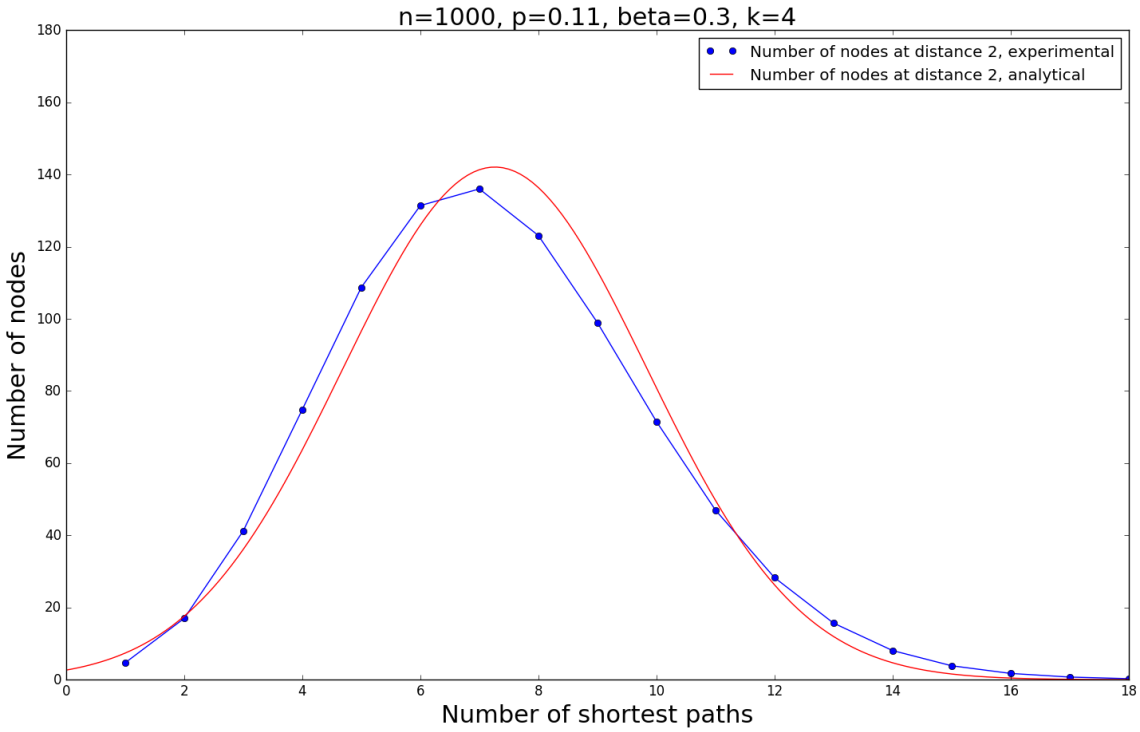

Figure 5.1: Comparison between the average experimental and analytical feature subvectors $[\mathrm{E}[n_{2,x}^{(k)}]]_{x\geq 1}$. For n=1000, p=0.11, beta=0.3, k=4.

possible to distinguish the feature subvectors.

Here we propose a "hidden parameter", a major factor in determining the accuracy of the GSPI based SVM classifier. It is well known that the difficulty of distinguishing between a simple one peak distribution and a mixture distribution with two peaks, is related to the ratio of the distance of the two peaks and the standard deviation of the distributions [25]. Here we propose to use this ratio for the hidden parameter. Note that the standard deviation of both (5.2) and (5.3) can be approximated as $\sqrt{nq_1^2(1-q_1)}$. Thus, the ratio can be approximated by

$$\begin{aligned} R_{n,p,k,\beta} &= \frac{np^2\beta^2}{k}/\sqrt{nq_1^2(1-q_1)} \\ &\approx \sqrt{n}\; p\; \beta^2/(k(1-\beta)). \end{aligned} \tag{5.6}$$

Which we conjecture to be the major factor determining the accuracy of the GSPI based SVM, for the graph classification problem of deciding if a graph contains k

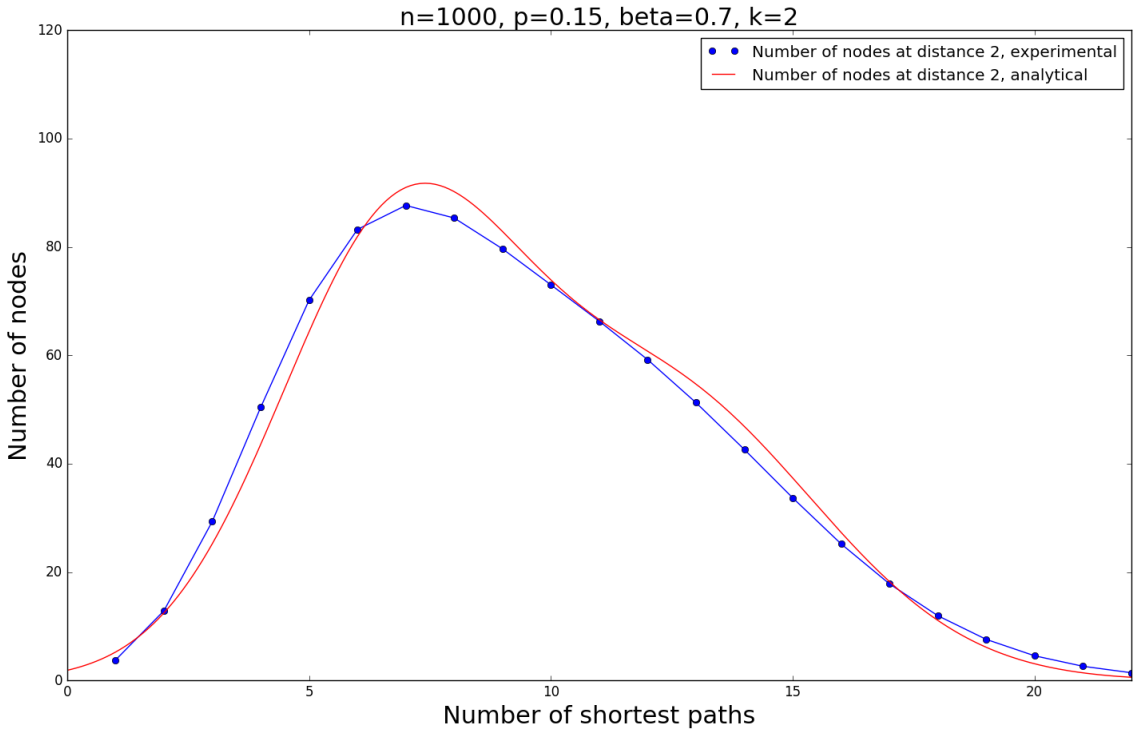

Figure 5.2: Comparison between the average experimental and analytical feature subvectors $[\mathrm{E}[n_{2,x}^{(k)}]]_{x\geq 1}$. For n=1000, p=0.15, beta=0.7, k=2.

or k+1 clusters. In Sect. 5.3.3 we derive some evidence supporting this conjecture based on experimental results.

5.3 Experiments

In this section we show and explain experimental results.

5.3.1 Dataset Specifications and Experiment Parameters

All datasets are generated according to the models $G(n,k,p,q_1)$ and $G(n,k+1,p,q_2)$. Where each dataset consists of 200 graphs from each model (400 in total). We performed the experiments for the parameters $n = 1000$,

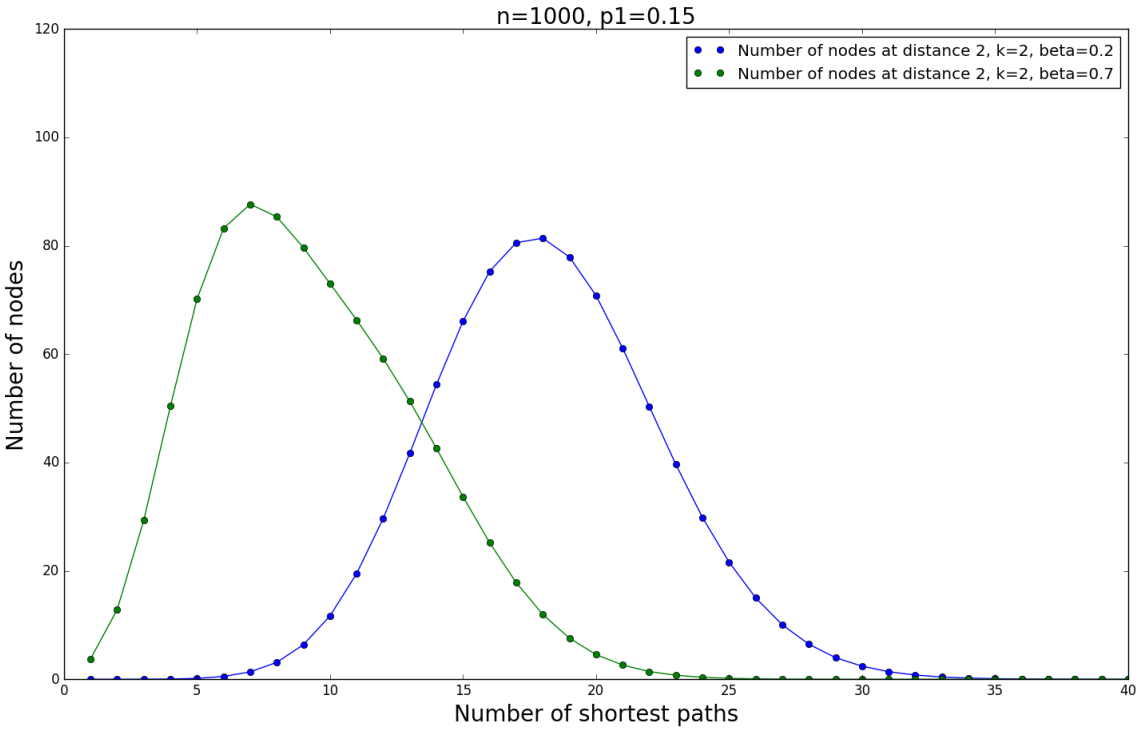

Figure 5.3: Average subvectors $[\mathrm{E}[n_{2,x}^{(k)}]]_{x\geq 1}$, when $n = 1000$, $p = 0.15$, $k = 2$ for $\beta = 0.2$ and $\beta = 0.7$. The difference between the peaks when $\beta = 0.2$ is 0.45 and the difference between the peaks when $\beta = 0.7$ is 5.51.

$p \in \{0.07,0.09,0.11,0.12,0.13,0.14,0.15\}$, $k \in \{2,3,\ldots,9\}$ and
$\beta \in \{0.2, 0.22, 0.24, \ldots,0.7\}$. For the SVM, we used *LIBSVM* [9], with 10-fold cross validation. For the cross validation we tried $C = 2^i$, for $i = [0,49]$ and the result from the best C was used as the accuracy. Each experiment was repeated 10 times, with different random seeds, and the final accuracy is the average of the 10 runs. If any of the 10 runs resulted in less than 56% accuracy, the final accuracy for that dataset and kernel was set to 56% in order to save running time. Preliminary experiments showed that $i < 1$ never gave any competitive accuracy. We used the SPI and GSPI kernels, with their feature vectors as defined in Sect. 3.3 and 3.4, with the modification that each feature vector is normalized by its Euclidean norm, this means that all inner products are in [0,1].

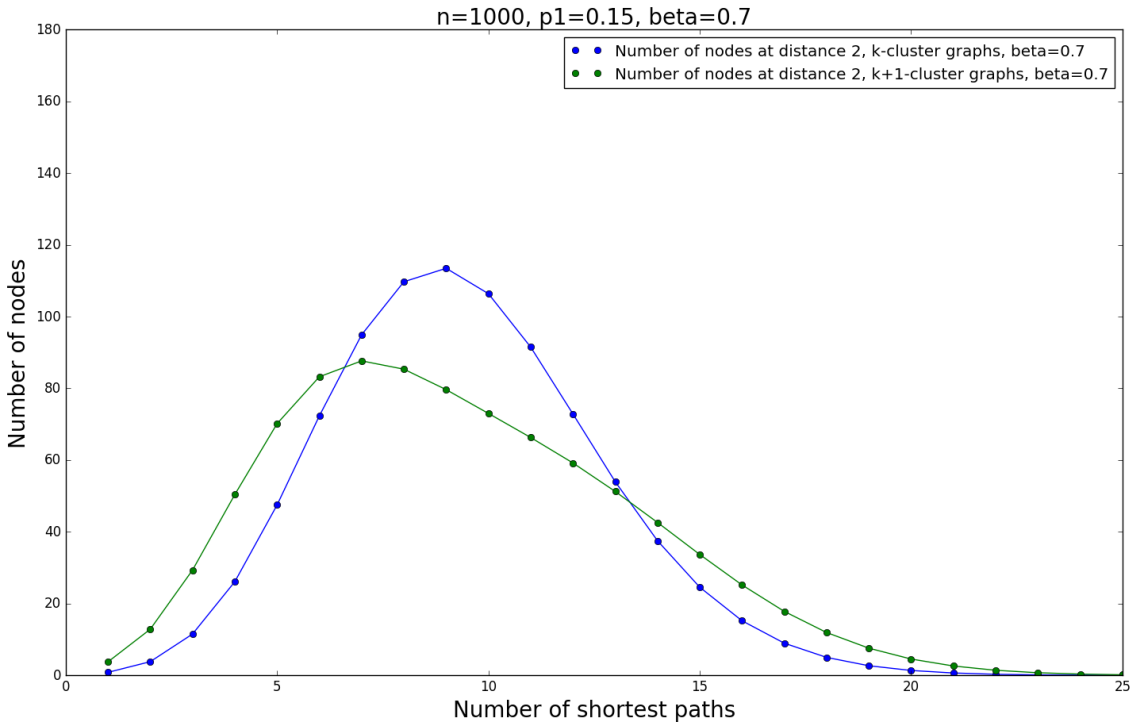

Figure 5.4: Average subvectors $[\mathrm{E}[n_{2,x}^{(k)}]]_{x\geq 1}$ and $[\mathrm{E}[n_{2,x}^{(k+1)}]]_{x\geq 1}$, when $n = 1000$, $p = 0.15$, $\beta = 0.7$, $k = 2$. The difference between the peaks for the k-cluster graphs is 5.51 and the difference between the peaks for the $k+1$-cluster graphs is 2.07.

5.3.2 Results

Both kernels increase in accuracy when β increases. The SPI kernel does not perform good when p is very large, this is because of the fact that for large values of p, all vertex pairs are at very close distances. For none of the datasets did the SPI kernel significantly outperform the GSPI kernel, although for very low values of p, the kernels perform similarly in terms of accuracy. For larger values of p, the GSPI significantly outperforms the SPI kernel. A comparison between the accuracies of the kernels can be seen in Tbl. 5.1, 5.2, 5.3 and 5.4.

The SPI kernel is only able to detect changes in the number of vertices at certain distances, i.e. n_1,n_2 .... Also note that the expected number of edges is the same for the k-cluster graphs and the $k+1$-cluster graphs, which means $\mathrm{E}[n_1^{(k)}] = \mathrm{E}[n_1^{(k+1)}]$. Because of this, for datasets where all vertices are at distances 1 and 2, the SPI kernel will not be able to gain any advantage over a random classifier. The number

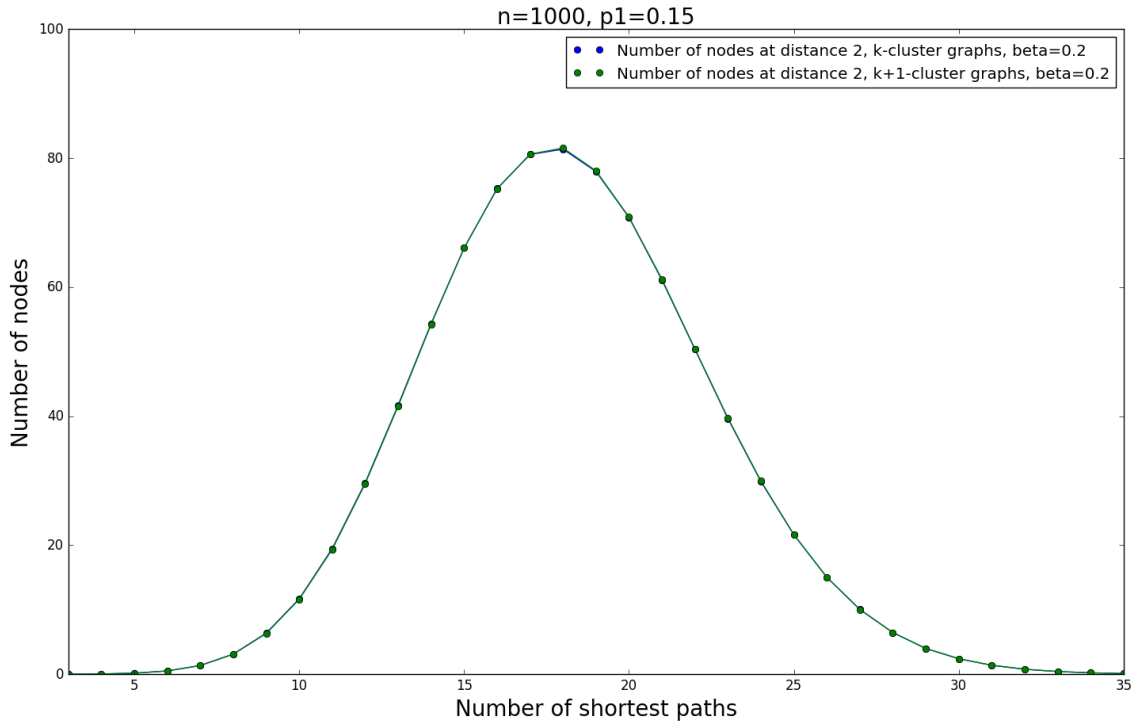

Figure 5.5: Average subvectors $[\mathrm{E}[n_{2,x}^{(k)}]]_{x\geq 1}$ and $[\mathrm{E}[n_{2,x}^{(k+1)}]]_{x\geq 1}$, when $n = 1000$, $p = 0.15$, $\beta = 0.2$, $k = 2$. The difference between the peaks for the k-cluster graphs is 0.45 and the difference between the peaks for the $k+1$-cluster graphs is 0.17. The distributions in this picture are indistinguishable.

of vertices at distances 3 and greater, is larger for smaller values of p and larger values of β.

5.3.3 The Accuracy of the GSPI kernel

We plot the accuracies of the GSPI kernel for $p = 0.11$ and $p = 0.15$. These results can be seen in Fig. 5.6 and 5.7. Our analysis is focused on the subvectors at distance 2 for analyzing the accuracy of the GSPI kernel. The reason for this is because of the fact that for most of the datasets, the number of vertex pairs that are at distances 1 and 2 is over 50%, meaning that the number of such vertex pairs is never negligible. In fact, for certain datasets, nearly all the vertex pairs are at distances 1 and 2. Such a range of parameters is for instance, $p = 0.15, \beta \in [0.2,0.4], k = 2,3$, where

Table 5.1: Comparison of accuracy for the SPI kernel and the GSPI kernel. $p = 0.15$, $k = 2$.

β	SPI accuracy	GSPI accuracy
0.2	56%	63.6%
0.22	56%	70.2%
0.24	56%	78.2%
0.26	56%	85.3%
0.28	56%	92.0%
0.3	56%	95.2%
0.32	56%	98.4%

Table 5.2: Comparison of accuracy for the SPI kernel and the GSPI kernel. $p = 0.09$, $k = 2$.

β	SPI accuracy	GSPI accuracy
0.26	60.3%	71.3%
0.28	61.7%	74.7%
0.3	60.3%	82.3%
0.32	62.2%	84.1%
0.34	68.1%	91.9%
0.36	74.5%	92.8%
0.38	78.9%	98.5%

the number of vertex pairs at distance more than 2 is always less than 0.1% of the total number of vertex pairs. This fact, that the number of vertex pairs at distances greater than 2 is very small, is one reason that the SPI kernel does not perform well for those datasets.

Now let us derive some evidence from our experimental results supporting our conjecture, that is, the ratio $R_{n,p,k,\beta}$ is the major factor determing the accuracy of the GSPI based SVM classifier. Unfortunately, it is computationally hard to get data for various graph sizes n. Thus, in this chapter, we consider one reasonably large graph size, namely, $n = 1000$, fixed, and obtain experimental results by changing the other

Table 5.3: Comparison of accuracy for the SPI kernel and the GSPI kernel. $p = 0.15$, $k = 4$.

β	SPI accuracy	GSPI accuracy
0.26	56%	64.4%
0.28	56%	68.0%
0.30	56%	73.0%
0.32	56%	87.2%
0.34	56%	92.1%
0.36	56%	94.4%
0.38	56%	97.8%

Table 5.4: Comparison of accuracy for the SPI kernel and the GSPI kernel. $p = 0.09$, $k = 4$.

β	SPI accuracy	GSPI accuracy
0.34	57.9%	64.2%
0.36	60.8%	74.7%
0.38	64.2%	80.0%
0.4	70.0%	86.2%
0.42	73.6%	90.3%
0.44	78.9%	92.5%
0.46	84.9%	97.1%

parameters, p, k, and β. Thus, in the following discussion, we regard n in the ratio $R_{n,p,k,\beta}$ as a constant.

First we note the relation between β and the accuracy of the GSPI kernel; see, Fig. 5.6 and 5.7. From the figures it seems that the accuracy increases linearly in β for the interval of β where the accuracy has started to increase above 60% until it reaches close to 100%. That is, for such an interval of β, the accuracy could be written as $a\beta + b$ at least approximately. Note on the other hand that

$$\sqrt{R_{n,p,k,\beta}} \propto \left(\sqrt{p/k}\right) \beta/\sqrt{1-\beta},$$

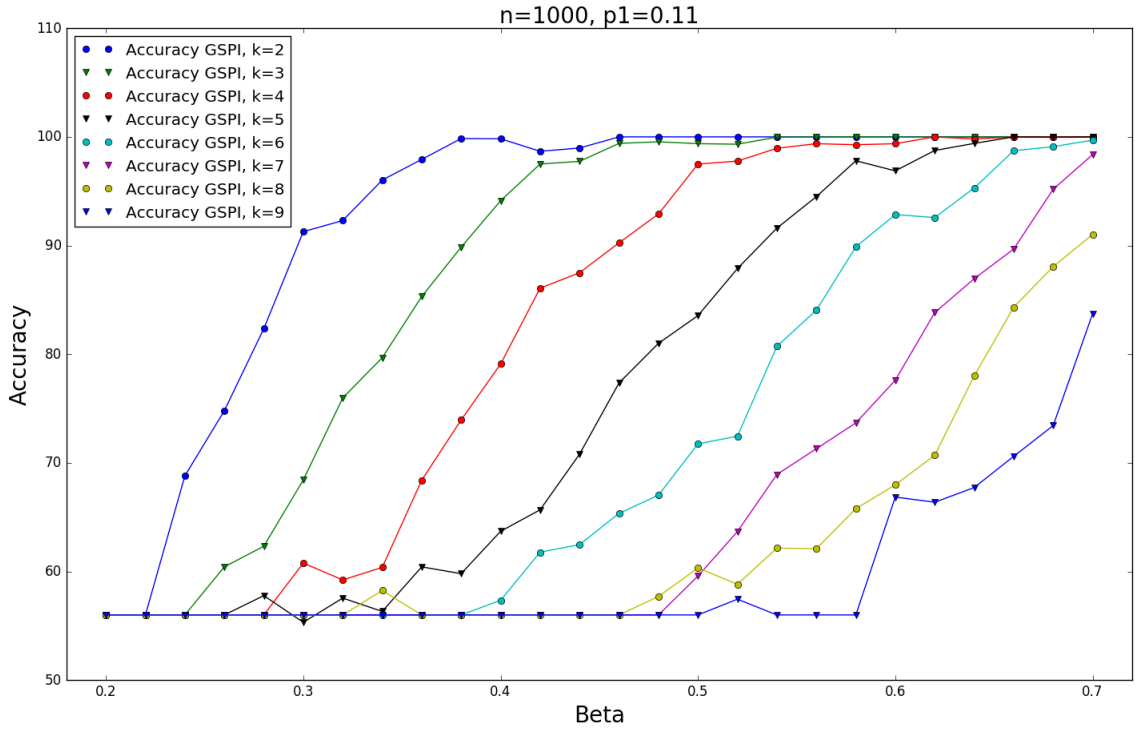

Figure 5.6: Accuracy of the GSPI kernel when $p = 0.11$.

and that $\beta/\sqrt{1-\beta}$ is close to linear for that interval of β. Thus, if $R_{n,p,k,\beta}$ were the major factor of the accuracy, the accuracy can be expressed as

$$\sqrt{R_{n,p,k,\beta}} + b \approx c\left(\sqrt{p/k}\right)\beta + b,$$

and hence, the slopes of graphs, e.g., in Fig. 5.6 and 5.7 (i.e., how quickly the accuracy increases when β increases) should be proportional to $\sqrt{p/k}$. We can verify this point from our experimental results.

More specifically, we check whether the slopes are proportional to $\sqrt{p/k}$ as follows. First calculate the slope, from the experiments, of the accuracy of the GSPI kernel for several combinations of the parameters p and k. The slopes were obtained by picking the intervals of β values for which the GSPI accuracy seems to increase linearly, where we considered $p \in \{0.11,0.12,0.13,0.14,0.15\}$, $k \in \{3,4,5,6\}$. We then fitted a line $a\beta + b$ using the least squares method in order to obtain the slope a. We then plotted the obtained slopes (values of a) against $\sqrt{p/k}$ and fitted it by a function $g(x) = cx$, again using the least squares method. The result of this can

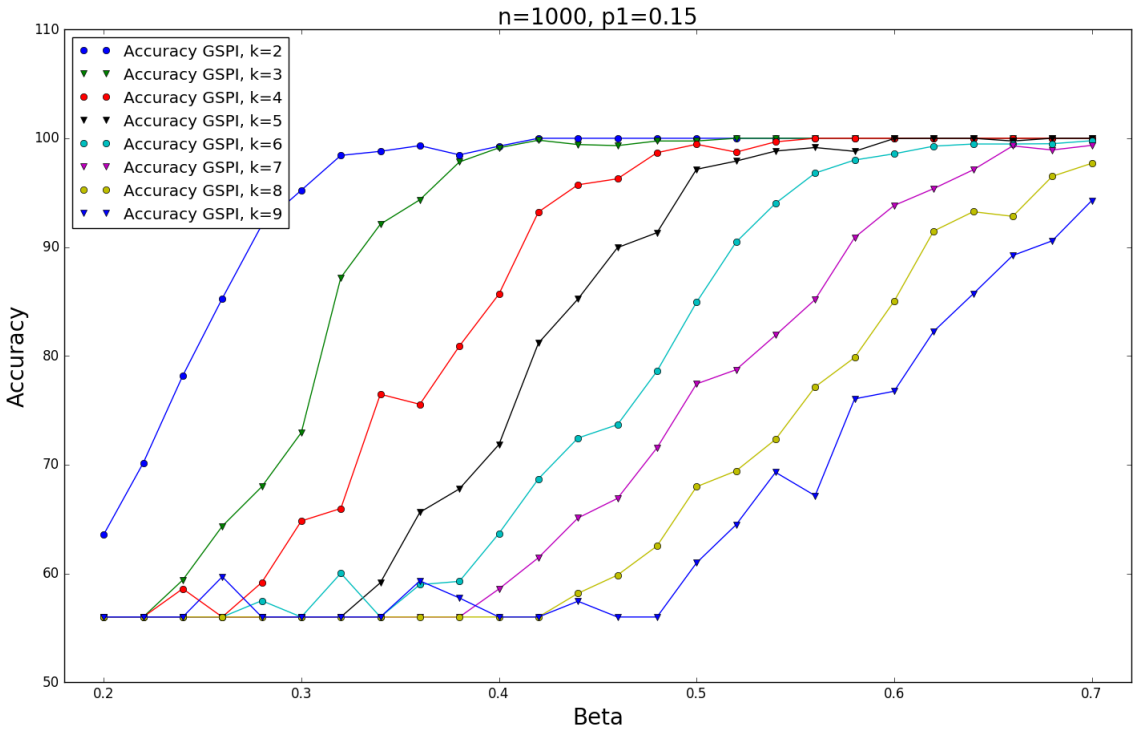

Figure 5.7: Accuracy of the GSPI kernel when $p = 0.15$.

be seen in Fig. 5.8, where c was determined to be 1362.8. In the figure, each dot represents an experimental slope value obtained from one combination of p and k and the line is the slope that we predict based on

$$\text{Slope} \propto \sqrt{p/k}.$$

As can be seen in the figure, the slope obtained from our conjecture is reasonably close to the real values.

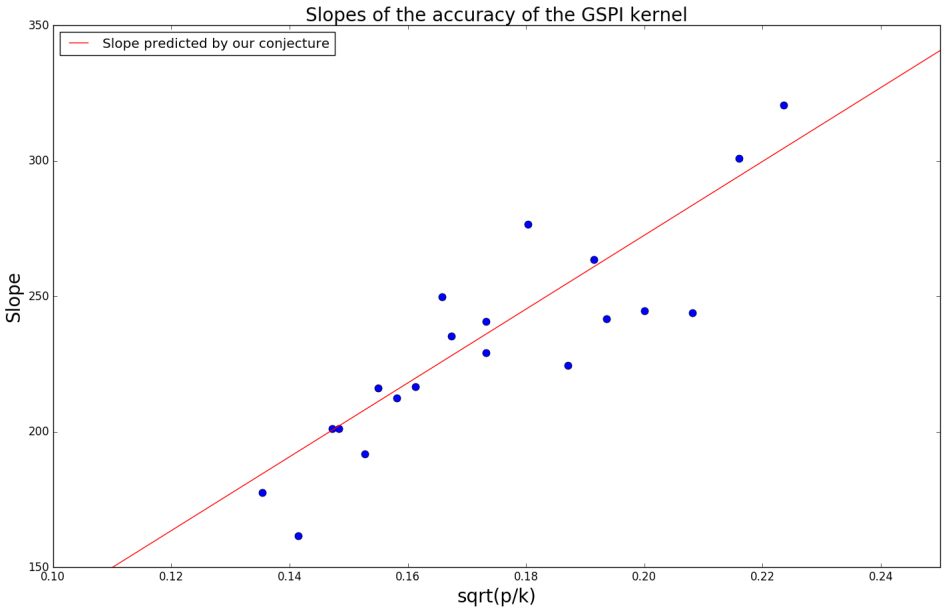

Figure 5.8: The slopes of the accuracy of the GSPI kernel for $p \in \{0.11,0.12,0.13,0.14,0.15\}$, $k \in \{3,4,5,6\}$ and the slope as predicted by our conjecture that the slope is proportional to $\sqrt{p/k}$.

6

Conclusions and Future Work

In this thesis we have introduced the *generalized shortest path* (GSP) kernel and compared it with the *shortest path* (SP) kernel. We applied a supervised machine learning approach for classifying cluster graphs, where we used graph kernels in order to represent graphs as vectors and then classified graphs using a *support vector machine* (SVM). For this approach, we experimentally evaluated the performances of the SP and the GSP kernels.

We introduced several random models for generating random cluster graphs, giving us the possibility of creating a wide variety of datasets where we have the ability to control the difficulty of the classification task for any given dataset. Using the random graph models we generated a large number of different datasets which we then tested both the performance of the SP kernel and the GSP kernel on.

We gave analyses of the feature vectors of the SP kernel and the GSP kernel, for the random graph models which we investigated. These analyses resulted in formulas for predicting the accuracies of the SP and the GSP kernel, based on certain parameters used when generating the datasets. We performed a heuristic analysis but motivated our assumptions whenever they were made and showed evidence that the analyses were not far from reality. Finally we evaluated our analysis by performing experiments, where we were able to confirm that our analyses in fact accurately were able to predict the accuracies of the SP kernel and the GSP kernel. These experiments also provided information about how well the SP kernel performed, compared with our new generalized shortest path kernel. In our experiments we found that the GSP

kernel outperforms the SP kernel on a wide range of datasets. When the GSP kernel did not outperform the SP kernel it still gave an accuracy that was competitive with the SP kernel. Since it is possible to compute the GSP kernel feature vectors in the same time as the SP kernel feature vectors, we are able to get an increase in accuracy when using the GSP kernel instead of the SP kernel, even though the running time does not increase significantly.

Future work could focus on applying the generalized shortest path kernel to various real world datasets. For most real world datasets, in order to achieve high accuracy, the graph kernels used typically needs to be modified slightly. This was done for instance for the random walk kernel when classifying proteins [5], where the basic random walk kernel was used, but was modified as to take into account certain properties unique to graphs of proteins. Similar modifications could be considered for the GSP kernel, when we want to classify particular real world datasets.

Bibliography

[1] Alex Bewley and Ben Upcroft. Advantages of exploiting projection structure for segmenting dense 3d point clouds. In *Australian Conference on Robotics and Automation*, 2013.

[2] Cagatay Bilgin, Cigdem Demir, Chandandeep Nagi, and Bulent Yener. Cell-graph mining for breast tissue modeling and classification. In *Engineering in Medicine and Biology Society, 2007. EMBS 2007. 29th Annual International Conference of the IEEE*, pages 5311–5314. IEEE, 2007.

[3] Béla Bollobás. *Random graphs*. Springer, 1998.

[4] Karsten M Borgwardt and Hans-Peter Kriegel. Shortest-path kernels on graphs. In *Prof. of ICDM*, 2005.

[5] Karsten M Borgwardt, Cheng Soon Ong, Stefan Schönauer, SVN Vishwanathan, Alex J Smola, and Hans-Peter Kriegel. Protein function prediction via graph kernels. *Bioinformatics*, 21(suppl 1):i47–i56, 2005.

[6] Bernhard E Boser, Isabelle M Guyon, and Vladimir N Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the fifth annual workshop on Computational learning theory*, pages 144–152. ACM, 1992.

[7] S. Boyd and L. Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[8] Chih-Chung Chang and Chih-Jen Lin. LIBSVM: a library for support vector machines. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 2(3):27, 2011.

[9] Chih-Chung Chang and Chih-Jen Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27, 2011. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

[10] Yudong Chen, Sujay Sanghavi, and Huan Xu. Clustering sparse graphs. In *Advances in neural information processing systems*, pages 2204–2212, 2012.

[11] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

[12] Agata Fronczak, Piotr Fronczak, and Janusz A Hołyst. Average path length in random networks. *Physical Review E*, 70(5):056110, 2004.

[13] Thomas Gärtner, Peter Flach, and Stefan Wrobel. On graph kernels: Hardness results and efficient alternatives. *Learning Theory and Kernel Machines*, pages 129–143, 2003.

[14] S Havlin and D Ben-Avraham. Theoretical and numerical study of fractal dimensionality in self-avoiding walks. *Physical Review A*, 26(3):1728, 1982.

[15] Linus Hermansson, Fredrik D Johansson, and Osamu Watanabe. Generalized shortest path kernel on graphs. In *International Conference on Discovery Science*, pages 78–85. Springer, 2015.

[16] Linus Hermansson, Tommi Kerola, Fredrik Johansson, Vinay Jethava, and Devdatt Dubhashi. Entity disambiguation in anonymized graphs using graph kernels. In *Proceedings of the 22nd ACM international conference on Conference on information & knowledge management*, pages 1037–1046. ACM, 2013.

[17] Radan Huth, Christoph Beck, Andreas Philipp, Matthias Demuzere, Zbigniew Ustrnul, Monika Cahynova, Jan Kyselỳ, and Ole Einar Tveito. Classifications of atmospheric circulation patterns. *Annals of the New York Academy of Sciences*, 1146(1):105–152, 2008.

[18] Fredrik Johansson, Vinay Jethava, Devdatt Dubhashi, and Chiranjib Bhattacharyya. Global graph kernels using geometric embeddings. In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pages 694–702, 2014.

[19] George Kimeldorf and Grace Wahba. Some results on tchebycheffian spline functions. *Journal of Mathematical Analysis and Applications*, 33(1):82–95, 1971.

[20] Sanjoy Dasgupta1 Alexandra Kolla and Konstantinos Koiliaris. Spectra of random graphs with planted partitions.

[21] Xiangnan Kong and Philip S Yu. Semi-supervised feature selection for graph classification. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 793–802. ACM, 2010.

[22] Taku Kudo, Eisaku Maeda, and Yuji Matsumoto. An application of boosting to graph classification. In *Advances in neural information processing systems*, pages 729–736, 2004.

[23] Maciej Liśkiewicz, Mitsunori Ogihara, and Seinosuke Toda. The complexity of counting self-avoiding walks in subgraphs of two-dimensional grids and hypercubes. *Theoretical Computer Science*, 304(1):129–156, 2003.

[24] Samet Oymak and Babak Hassibi. Finding dense clusters via" low rank+ sparse" decomposition. *arXiv preprint arXiv:1104.5186*, 2011.

[25] Mark F Schilling, Ann E Watkins, and William Watkins. Is human height bimodal? *The American Statistician*, 56(3):223–229, 2002.

[26] Bernhard Schölkopf and Alexander J Smola. *Learning with kernels: Support vector machines, regularization, optimization, and beyond*. MIT press, 2001.

[27] Shai Shalev-Shwartz, Yoram Singer, Nathan Srebro, and Andrew Cotter. Pegasos: primal estimated sub-gradient solver for SVM. *Mathematical Programming*, 127(1):3–30, 2011.

[28] Nino Shervashidze, SVN Vishwanathan, Tobias Petri, Kurt Mehlhorn, and Karsten M Borgwardt. Efficient graphlet kernels for large graph comparison. In *Proc. of AISTATS*, 2009.