University of Padua
Department of Mathematics
Doctorate Degree in Brain, Mind and Computer Science
Curriculum in Computer Science

KERNEL METHODS FOR LARGE-SCALE GRAPH-BASED HETEROGENEOUS BIOLOGICAL DATA INTEGRATION

Candidate
Dinh Tran Van

Supervisor
Prof. Alessandro Sperduti
Co-supervisor
Prof. Franca Stablum
University of Padova, Italy

October 26, 2017

Acknowledgments

Firstly, I would like to express my sincere gratitude to my advisor Prof A for the continuous support of my Ph.D study, for his patience, motivation and enthusiasm.

Besides my advisor, I would like to thank the rest of my thesis committee for their insightful and valuable comments and suggestions.

I would like to thank Dr. B for all his help and for his guidance before and during my Ph.D studies. Also, I would like to thank Prof. Peter Tino for the meaningful discussions and inputs he provided during my visit to University of Birmingham.

I would like to thank all my friends, colleagues and officemates (Moreno, Riccardo, Alberto, Daniele, Hossein and Ding Ding) here at the University of Padua for the stimulating discussions, and for all the fun we have had in the last three years.

Last but not the least, a huge thank goes to my family, that always supported me in these years. Words cannot express how grateful I am to my mother, and father for all of the sacrifices that they have made on my behalf.

Dinh Tran Van
Padova, October 26, 2017

Abstract

The last decades have been experiencing a rapid growth in volume and diversity of biological data, thanks to the development of high-throughput technologies related to web services and embeded systems. It is common that information related to a given biological phenomenon is encoded in multiple data sources. On the one hand, this provides a great opportunity for biologists and data scientists to have more unified views about phenomenon of interest. On the other hand, this presents challenges for scientists to find optimal ways in order to wisely extract knowledge from such huge amount of data which normally cannot be done without the help of automated learning systems. Therefore, there is a high need of developing smart learning systems, whose input as set of multiple sources, to support experts to form and assess hypotheses in biology and medicine. In these systems, the problem of combining multiple data sources or data integration needs to be efficiently solved to achieve high performances.

Biological data can naturally be represented as graphs. By taking graphs for data representation, we can take advantages from the access to a solid and principled mathematical framework for graphs, and the problem of data integration is converted into graph-based integration. In recent years, the machine learning community has witnessed the tremendous growth in the development of kernel-based learning algorithms. Kernel methods whose kernel functions allow to separate between the representation of the data and the general learning algorithm. Interestingly, kernel representation can be applied to any type of data, including trees, graphs, vectors, etc. For this reason, kernel methods are a reasonable and logical choice for graph-based inference systems. However, there is a number of challenges for graph-based systems using kernel methods need to be effectively solved,

including: *definition of node similarity measure*, *graph sparsity*, *scalability*, *integration methods*. The contributions of this thesis aim at investigating to propose solutions that overcome the challenges faced when constructing graph-based data integration learning systems.

The first contribution of the thesis is the definition of a novel decompositional graph node kernel, named conjunctive disjunctive node kernel, to measure graph node similarities. We first employ a network decomposition procedure to transform the network into a set of linked connected components in which we distinguish between *conjunctive* links whose endpoints are in the same connected components and *disjunctive* links that connect nodes located in different connected components. We then propose a graph node kernel that explicitly models the configuration of each nodes context.

The second contribution aims at dealing with the sparsity problem of graphs by introducing a link prediction method whose objective is to recover missing links of a graph. In this method we first represent each link connecting two nodes by a graph composed of their neighborhood subgraphs. We then cast the link prediction problem as a binary classification task over obtained graphs in which we employ an efficient decompositional graph kernel for graph similarity. Empirical evaluation proves the promissing of the method.

The third contribution targets to boost the performance of diffusion-based graph node kernels when the graph structure is affected by noise in the form of missing links, similarities are distorted proportionally to the sparsity of the graph and to the fraction of missing links. As a consequence, we propose a method named link enrichment for diffusion-based graph node kernels with the idea of carrying out the computation of information diffusion on a graph that contains edges identified by link prediction approaches. We discover a surprisingly robust signal that indicates that diffusion-based node kernels consistently benefit from the coupling with similarity-based link prediction techniques on large scale datasets in biological domains.

The fourth contribution copes with the scalability problem of graph-based data integration learning systems by proposing scalable kernel-based gene prioritization method (Scuba). Scuba is optimized to deal with strongly unbalanced setting and is able to deal with both large amount of candidate genes and arbitrary number of data sources. It enhances the

scalability and efficacy and outperforms existing methods for disease gene prioritization.

The last contribution is another approach for graph-based data integration, targeting to solve disease gene prioritization problem. In this approach, the common genes between graph layers, derived from biological sources, are connected by disjunctive links. Then a particular graph node kernel is adopted to exploit topological graph features from all layers for measuring gene similarities. The state of the art performance on different experimental settings confirms the strength of the method.

Contents

Chapter 1

Introduction

The release of advanced technologies is one of the main reasons for the revolution in various scientific research fields. In Biological and Medical domain, modern technologies are making it not only easier but also more economical than ever to undertake experiments and creating applications. As a consequence, a vast amount of biological data in terms of volume and type is generated through scientific experiments, published literature, high-throughput experiment technology, and computational analysis. This huge quantity of data are saved as biological datasets and made discoverable through web browsers, application programming interfaces, scalable search technology and extensive cross-referencing between databases. Biological databases normally contain information about gene function, structure, localization, clinical effects of mutations and similarities of biological sequences and structures.

The abundance of biological data, on the one hand, creates a golden chance for biologists to extract useful information. However, it, on the other hand, poses the challenge for scientists to wisely and effectively extract knowledge from such amount of data that normally cannot be done without the help of automated learning systems. Hence, the task of developing high performance learning systems, which help sciencists to form and assess hypotheses, plays an important role in the development of biology and medicine.

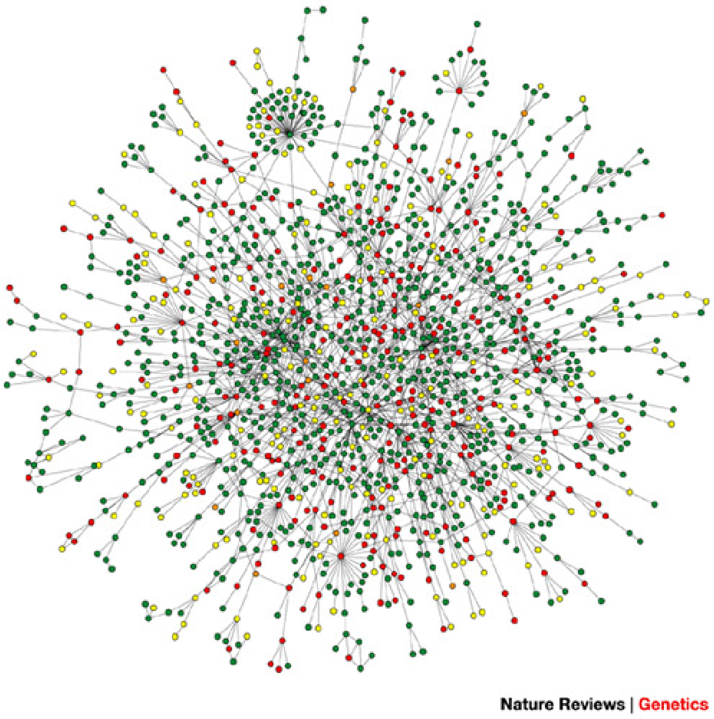

Figure 1.1: Yeast protein interaction network

1.1 Why graph-based biological data integration?

Biological knowledge is distributed among general and specialized sources, such as gene expression, protein interaction, gene ontology, etc. It is common that information of a biological phenomenon is encoded over various heterogeneous sources. Each source captures different aspects of the phenomenon. The distribution of information over sources provides us an unprecedented opportunity to understand the phenomenon from multiple angles. Therefore, the idea of data integration which allows multiple sources of information to be treated in a unified way is potential to improve the performance of biological learning systems. Despite the fact that data integration is a promising solution, it exposes a challenge for machine learning experts and data scientists to find out optimal solutions for combining multiple sources in a big space of solutions.

Relations between entities encoded in biological sources can be naturally represented in form of graphs (networks) whose vertices describe for biological entities and links characterize the relations between entities. An example is the protein-protein interaction network (Figure 1.1) where each vertex represents a protein and a link connecting two vertices if they interact. Graph theory provides a mathematical abstraction for the description of such relationships. Thus, using graphs to represent for biological data allows us to *i*) access to a principled and solid mathematical framework built for graphs that most scientists are familiar with, *ii*) develop concepts and tools which are independent of the concrete applications. By using a graph presentation, the problem of biological data integration now can be converted into graph-based data integration. The final representation of data obtained by integration is used as input for the construction of inference systems (graph-based learning systems).

1.2 Kernel methods for graph-based data integration and challenges

Recently, *kernel methods* whose best known member is support vector machine (SVM) [25], has emerged as one of the most famous and powerful frameworks in machine learning. Kernel methods with the use of kernels allow to decouple the representation of the data (via kernel function) from the specific learning algorithm. Kernel representation is flexible and efficient and it provides a principled framework that allows universal type of data to be represented, including images, graphs, vectors, strings, etc. As a consequence, kernel methods are the state-of-the-art learning technique for graph-based inference systems. However, there are a number of challenges that need to be efficiently solved, if we desire to have high performance graph-based data integration learning systems. Following are the main challenges: definition of node similarity measure, graph sparsity, data integration method.

1.2.1 Definition of node similarity measure

In machine learning, one of the main factors which impacts on the performance of learning systems is the definition of example similarity measure. In our context, large-scale graph-based inference systems, examples are nodes of graphs. Hence, it is necessary to have a good definition of node similarity measure. Node similarity is normally measured by *graph node kernels*. However, there is not a clear way to define a graph node kernel which can be efficiently applied to a wide range of graphs.

1.2.2 Sparsity

The input of a graph-based data integration system is a set of graphs which often contain sparse graphs whose number of links is much less than the number of possible links. This is typically due to the lack of information. For instance, in the disease gene network, links connecting genes are formed when genes are involved in the same diseases. However, new genes associated to a certain disease could be discovered over time. This means that new links could be added into the networks over time. At a give time, a number of discovered links can be very limited, so discovered links cause the sparsity problem. When working with sparse graphs, systems encounters difficulties in performing an effective training since not enough information is available to correctly learn the target function. As a consequence, an effective solution helping to overcome the sparsity problem is crucial and needs to be proposed.

1.2.3 Scalability

Given an adopted learning algorithm, the complexity of a large-scale graph-based data integration learning system incurs with the growth of the input graph set. In other words, the complexity of a graph-based learning system strongly depends on the size and the number of graphs used as its input. Therefore, scalability is an important property that a graph-based learning system is supposed to possess. It allows systems to run in reasonable time and with a reasonable memory consumption.

1.2.4 Data integration methods

Information encoded in multiple sources (graphs) provides a complementary views of phenomenon of interest. Combining information from collective sources helps to form a complete picture of the phenomenon or problem at hand. However, the search for integration methods that allow to improve the performance of the learning system with respect to the same system where a single source of information is used, is normally expensive.

1.3 Contributions

The contributions of the thesis focus on solutions to overcome the challenges faced when working with large-scale graph-based biological data integration.

The first contribution considers the problem of defining an effective node similarity measure by introducing a novel graph node kernel, named conjunctive disjunctive node kernel (CDNK). Most existing graph node kernels are based on a notion of information diffusion which can be applied to dense networks with high values of average node degree. However, a drawback of these approaches is their relatively low discriminative capacity. This is in part due to the fact that information is processed in an additive and independent fashion which prevents them from accurately modeling the configuration of each gene context. To address this issue, we propose to employ a decompositional graph kernel technique in which the similarity function between graphs can be formed by decomposing each graph into subgraphs and by devising a valid local kernel between the subgraphs. In CDNK, to exploit its higher discriminative capacity, first the network is decomposed into a collection of connected sparse graphs and then a suitable kernel is developed.

The second contribution is the introduction of a link prediction method, which is adopted later on for link enrichment with the aim of solving the problem of graph sparsity. We get the motivation from the current link prediction methods that do not effectively exploit the contextual information available in the neighborhood of each edge. In our method, we propose to cast the problem as a binary classification task over the union of the pairs of subgraphs located at the endpoints of each edge. We model the classification task using a support vector machine endowed with an efficient graph kernel and achieve state-of-the-art results on several benchmark datasets.

The third contribution proposes a method that boosts performance of diffusion-based kernels when working with sparse graphs by tackling them with link enrichments methods. In particular, given a sparse graph, our proposed method consists of two phases. In the first phase, a link prediction method is employed to rank unobserved links based on their probabilities to be related to missing links. The top links in the ranking are then added into the graph. In the second phase, diffusion-based graph node kernels are applied to the graph obtained from the first phase to compute the kernel matrix.

The fourth contribution is a method for disease gene prioritization named scalable kernel-based gene prioritization (Scuba). In Scuba, the scalability problem of graph-based data integration is effectively solved. Besides, Scuba is optimized to deal with strongly unbalanced setting. In particular, the method first employs different graph node kernels to compute a set of kernels from each input genetic graph. It then adopts an effective multiple kernel learning to find an optimal linear combination from these obtained kernels. The final kernel is then fed into a kernel machine (SVM) to output a ranking for candidate genes. Results from different empirical experiments show the outperformance of our method comparing with state-of-the-art ones.

In the last contribution, we propose another approach for graph-based data integration that we again target to solve for disease gene prioritization problem. Unlike Scuba whose the integration happens when multiple kernels computed from input graphs are combined, in this approach, data integration is performed by employing disjunctive interconnection graph procedure. In this procedure, the common genes between graph layers, derived from biological sources, are connected by disjunctive links. We then use CDNK to compute a kernel matrix that is later used as inpute of kernel machine to build model. The evaluation through two particular experimental settings proves that it is the state of the art method in disease gene prioritization.

1.4 Thesis roadmap

The thesis is organized as follows.

Chapter 2 presents preliminary concepts, notations. The rest of the thesis will follow these notations and conventions. Besides, it provides a comprehensive review of the state-of-the-art in the field.

Chapter 3 proposes conjunctive disjunctive graph node kernel.

Chapter **??** introduces a novel link prediction method.

Chapter 5 and **??** describe two different proposed approaches for large-scale graph-based biological data integration.

Chapter 6 summarizes the contributions of the thesis and discusses the directions for future work.

Chapter 2

Background

In this chapter, we describe preliminary knowledge and notaions used for the remaining parts of this thesis to make it easy for readers to follow.

2.1 Machine Learning

Recently, *machine learning* has become a must-know term not only in academia but also in daily life due to the popularity of it's applications in various fields. Machine learning can be considered as a branch of Artificial Intelligence which aims at providing systems the ability to automatically adapt to their environment and learn from experience without being explicitly programmed. According to [61], machine learning is formally defined as:

Definition 2.1.1. *A computer program is said to learn from experience E with respect to some task T and some performance measure P if its performance on T, as measured by P, improves with experience E.*

We denote $\mathbb{X}$ as domain dataset which encodes the complete information of a domain. For each domain, however, we are only able to collect a small fraction of domain dataset, $\mathbb{D}$, resulted from any observation, measurement or recording apparatus such that $\mathbb{D} \cup \mathbb{X}$. The set $\mathbb{D}$ is normally referred as the training dataset. Machine Leanring techniques desire to exploit $\mathbb{D}$ to get useful information for constructing a model that generalizes nature of data source. The model is then used to make prediction or inference in unseen dataset, $\mathbb{U} = \mathbb{X} - \mathbb{D}$.

Machine Learning algorithms can be classified into three groups: supervised learning, unsupervised learning and reinforcement learning. Supervised learning proceed with datasets whose objects are associated to labels, while unsupervised learning works with datasets consisting of input data without labeled responses. Reinforcement Learning aims at designing machines and software agents that can automatically determine the ideal behaviour within a specific context, in order to maximize its performance. Simple reward feedback is required for the agent to learn its behaviour; this is known as the reinforcement signal. In this thesis, we focus on supervised learning scenario.

We consider a training set $\mathbb{D}$ generated by an unknown probability distribution $\mathcal{P}$, $\mathbb{D} = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$ where $x_i \in \mathbb{X}$ are examples and $y_i \in \mathbb{Y}$ are labels. The relations between x_i and y_i are defined by a true function (target function) $f : \mathbb{X} \longmapsto \mathbb{Y}$. What we desire to do is to learn the function f. However, the only information we can access is from the training set. Therefore, a supervised learning method aims at estimating a function g based on $\mathbb{D}$ to be as close to f as possible. Depending on the domain of $\mathbb{Y}$, we can further group supervised learning methods into following sub-groups:

- if $\mathbb{Y} \subseteq \mathbb{R}$, the problem is called regression
- if $|\mathbb{Y}| = 2$, we have a binary classification problem
- if $|\mathbb{Y}| = n$ with $n > 2$, we have multi-class classification problem

Besides, a supervised learning algorithm is called multi-labels classification if an example could have more than one label associated with that. It is worth highlighting that there is normally more than one possible choice for g. We refer each choice of g as a hypothesis (h) or model and the set of all possible g as hypothesis space, $\mathbb{H}$. A hypothesis space need to be define in advance and it is necessary to contains good approximations to target function. In order to find the optimal hypothesis, h^*, one way is to employ a true loss function, $\mathcal{L}$, which measures how much a hypothesis fails to correctly map between examples and their corresponding labels.

$$\mathcal{R}(h) = \int_{\mathbb{X}\times\mathbb{Y}} \mathcal{L}(h(x), y) dP(x, y) \tag{2.1}$$

The optimal function with the least of mis-mappings (errors) is then the solution of following optimization:

$$h^* = \arg\min_{h\in\mathbb{H}} \mathcal{R}(h) \tag{2.2}$$

Unfortunately, it is impossible to directly solve the optimization 2.1 since the probability distribution $\mathcal{P}$ in the true loss function 2.2 is an unknown function and we only have access to a finite training set $\mathbb{D}$. In this case, an alternative approach is to use empirical loss instead of true loss function. The empirical loss function is defined over the training set as follow:

$$\mathcal{R}_{emp}(h) = \frac{1}{n}\sum_{n=1}^{n} |h(x_i) - y_i| \tag{2.3}$$

However, in order to use $\mathcal{R}_{emp}(h)$, we need to guarantee that the value of $\mathcal{R}_{emp}(h)$ converges to the value of $\mathcal{R}(h)$?. Using the law of large numbers, authors prove in [83] that the convergence happens when the number of examples is high enough. For more details we refer the reader an amazing book [83].

2.2 Kernel Methods

In classical machine learning techniques for binary classification, first the data presentation form is defined, strings, vectors for instances. It then is used to represent for each example, $x \in \mathbb{X} \longrightarrow \phi(x) \in \mathbb{F}$. After a linear function is learnt to separate positive examples from negative ones. Although these approaches have sucessfully applied in some cases, they share two common limitations: i) the high comlexity when working with high dimensional spaces. ii) the difficulty or impossiblity to find the vectorical form to represent data in many cases.

Recently, a new framework named Kernel method has been proposed and shown the state-of-the-art results in many cases of various fields. SVM [25] is a typical example of kernel methods. Unlike the presentation of data in traditional machine learning, data are not individually represented in kernel methods, but through a set of pairwise similarities. More precisely, a matrix whose each element is a real-valued comparision between two examples is used to represent for a data set. These real-valued elements are computed by using a kernel function: $k : \mathbb{X} \times \mathbb{X} \longmapsto \mathbb{R}$. By using matrix to represent for data set, the presentation of data in kernel methods does not depend on the nature of objects. That means the presentation of strings, images,... are the same. More interesting, it allows kernel machines to modularize into two components: the design of a specific kernel function and the design of a general learning algorithm (kernel machine).

2.3 Kernel functions

As stated in 2.2, in kernel methods, the definition of kernel functions is independent from the definition of general learning algorithms. Therefore, it provides kernel machines more options when employing kernels. A number of kernel functions have been proposed for different types of data. In this section, we first formally define what is a kernel function. We then introduce some kernels defined on graphs that later on are used in our experiments.

Definition 2.3.1. *Given a set of object $\mathbb{X}$, a function $k : \mathbb{X} \times \mathbb{X} \longmapsto \mathbb{R}$ is called a kernel on $\mathbb{X} \times \mathbb{X}$ iff k is*

- *symmetric: it means $k(x_1, x_2) = k(x_2, x_1)$, where $x_1, x_2 \in \mathbb{X}$.*
- *positive semi-definite: that is $\sum_{i=1}^{N} \sum_{j=1}^{N} c_i c_j k(x_i, x_j) \geq 0$ for any $N > 0$, $c_i, c_j \in \mathbb{R}$, and $x_i, x_j \in \mathbb{X}$.*

A kernel is usually represented as a matrix K which is called Kernel matrix (Gram matix) and K needs to be symmetric and positive semi-definite, i.e. its eigen values are non-negative. In this thesis, kernels and kernel matrcies are identical.

$$K = \begin{bmatrix} k(x_1, x_1) & k(x_1, x_2) & k(x_1, x_3) & \dots & k(x_1, x_n) \\ k(x_2, x_1) & k(x_2, x_2) & k(x_2, x_3) & \dots & k(x_2, x_n) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ k(x_n, x_1) & k(x_n, x_2) & k(x_n, x_3) & \dots & k(x_n, x_n) \end{bmatrix}$$

One of the most simple kernel, called Linear kernel which is defined on vectors, $\mathbb{X} \subseteq \mathbb{R}^n$.

$$k_L(x_1, x_2) = x_1^{\intercal} x_2, \tag{2.4}$$

where $x_1, x_2 \in \mathbb{X}$. This kernel suggests a systematic way to define kernels. Given a general set of object $\mathbb{X}$, we first project each element in $\mathbb{X}$ into a vector space, $x \in \mathbb{X} \longrightarrow \phi(x) \in \mathbb{R}^n$. Next, we define a kernel as:

$$k(x_1, x_2) = \phi(x_1)^{\intercal} \phi(x_2) \tag{2.5}$$

Interestingly, any kernels defined on $\mathbb{X}$, there exists a Hilbert space, $\mathbb{F}$ and a mapping $\phi : \mathbb{X} \longrightarrow \mathbb{F}$ such that $k(x_1, x_2) = \langle \phi(x_1), \phi(x_2) \rangle$, where $x_1, x_2 \in \mathbb{X}$.

There are two problems we might face with if we would like to explicitly embed objects into a vector space. i) We need to deal the high computation if we embed objects into high demensional spaces. ii) We do not have

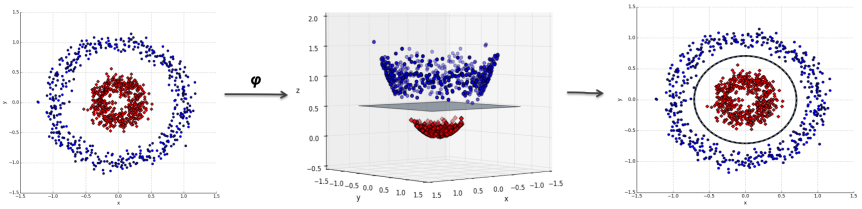

Figure 2.1: The kernel trick transforms the data in a feature space where the instances from the two classes may be linearly separable.

clear way to represent objects in vectorial forms. These limitations are effectively solved by using *Kernel trick*. The kernel trick 2.1 avoids the explicit mapping. Instead, it allows the operations (dot product) between vectors in the feature space to be done by computing only in the input space.

It is worth to notice that a kernel is considered as a similarity (proximity) measure since its value computed for two objects is proportional to their similarity.

Most kernels are defined on vectorial form of data in which Radial basis function kernel (RBF) [85] is the most used one. However, real-world data often cannot be represented in the vectorial form without lossing important information. Therefore, a high number of kernels are proposed to deal with structured data, including trees, graphs, etc.

Convolution kernels

On the development of kernels for structured data, R-convolution kernels [37] proposed by David Haussler can be considered as one of the most important frameworks. The basic idea of convolution kernels is that the semantics of composite objects can often be captrued by a relation R between the object and its parts. The kernel on the object is then made up from kernels defined on different parts.

Let x be a composite structure whose $x_1, x_2, \ldots, x_N = \overrightarrow{x}$ are parts of x, such that $x \in \mathbb{X}$, $x_i \in \mathbb{X}_\beth$, $i = \overline{1,N}$ and $\mathbb{X}, \mathbb{X}_1, \mathbb{X}_2, \ldots, \mathbb{X}_N$ are non-empty and separable metric spaces. We define a relation $R(\overrightarrow{x}, x)$ on $\mathbb{X}_1 \times \mathbb{X}_2 \times \ldots \times \mathbb{X}_N \times \mathbb{X}$ is true iif $x_1, x_2, \ldots, x_N$ are the parts of x. We denote R^{-1} as the inverse relation of R and it is defined as $R^{-1} = \{\overrightarrow{x} | R(\overrightarrow{x}, x)\}$.

If there exists kernel k_i defined on X_i, the similarity between $x, y \in \mathbb{X}$ is defined on $\mathbb{S} \times \mathbb{S}$, where $\mathbb{S} = \{x | R^{-1}(x) \neq \emptyset\}$, as:

$$K(x,y) = \sum_{\overrightarrow{x} \in R^1(x),\ \overrightarrow{y} \in R^1(y)} \prod_{i=1}^{N} k_i(x_i, y_i) \tag{2.6}$$

K is referred as finite convolution, if R is finite. The zero expansion of K to $\mathbb{X} \times \mathbb{X}$ is called R-convolution and it is denoted as $K_1 \star K_2 \star \ldots K_N(x,y)$.

Theorem 1. *If $K_1, K_2, \ldots, K_N$ are kernels on $\mathbb{X}_1, \mathbb{X}_2, \ldots, \mathbb{X}_N$, respectively, and R is a finite relation on $\mathbb{X}_1 \times \mathbb{X}_2 \times \ldots \times \mathbb{X}_N$, then $K_1 \star K_2 \star \ldots K_N(x,y)$ is a kernel on $\mathbb{X} \times \mathbb{X}$.*

Constructing kernels

Kernels can be constructed from predefined kernels. Let k_1, k_2 be kernels over $\mathbb{X} \times \mathbb{X}$, $\mathbb{X} \subseteq \mathbb{R}^n$, $\alpha_1, \alpha_2 \in \mathbb{R}^+$, $f(.)$ is a real valued function on $\mathbb{X}$, $\phi : \mathbb{X} \longmapsto \mathbb{R}^N$ with k_3 a kernel over $\mathbb{R}^N \times \mathbb{R}^N$, and $\mathbf{B}_{n \times n}$ is a symmetric, positive demi-definite. The following functions are kernels.

- $k(x,y) = \alpha_1 k_1(x,y) + \alpha_2 k_2(x,y)$
- $k(x,y) = k_1(x,y) k_2(x,y)$
- $k(x,y) = f(x) f(y)$
- $k(x,y) = k_3(\phi(x), \phi(y))$
- $k(x,y) = x^{'} \mathbf{B} y$

For the proof of above kernels and other ways to form kernels from predefined kernels, we highly recommend a great book titled ”Kernel methods for pattern analysis” [74] to readers.

2.4 Kernel machine

Traditional machine learning techniques aim at finding linear relations in datasets which are presented in vectorical forms. However, there are many cases where expected linear relations does not exist in input dataset. A solution to overcome these situations is to first explicitly transform data into a higher dimensional space and then search for linear relations in that space. Unlike traditional machine methods, kernel methods with the use of kernel functions are able to operate in a high-dimensional space without ever computing the coordinates of the data in that space, but rather by simply computing the inner products between the images of all pairs of data in the feature space. This operation is often computationally cheaper than the explicit computation of the coordinates.

A number of kernel methods have been proposed. Examples of kernel methods include Perceptron, support vector machines (SVM), Gaussian processes, principal components analysis (PCA), etc. In the next sections, we will introduce in detail two famous algorithms: Perceptron and SVM. For the simplicity, we describe these algorithms in the context of binary classification.

2.4.1 Perceptron kernel algorithm

Perceptron is an old, online leanring algorithm which is based on error-driven learning. It desires to learn a hyper-plane, $wx + b$ or wx for simplicity, to separate positive examples from the negative ones in the training set. It is then used to predict a label for each unseen example, x, through *sign* function. If $wx \geq 0$, $\hat{y} = sign(x) = 1$, otherwise $\hat{y} = sign\,(x) = -1$.

Perceptron works by first initilizing values for weight vector, w. It then iteratively improves the performance by updating the weight vector whenever a misclassification is found in the training set. Consider y_i and $\hat{y}_i$ are true label and predicted label for x_i, if $y_i \neq \hat{y}_i$, w is updated as follow:

$$w \longleftarrow w + \alpha y_i x_i,$$

where $\alpha \in (0, 1]$ is the learning rate. Suppose that n misclassified examples are observed, the weight vector w can be presented as:

$$w = \sum_{i=1}^{n} \alpha y_i x_i.$$

The interation is done when there is no error found. This algorithm guarantees that a linear separation is found if it exists. When the linear separation does not exist, a possible solution is to embed input data into a higher dimensional space. By virtue of doing so, there is a higher chance to have linear separation. However, a problem of high computation is raised when the algorithm operates in a high dimensional space. As a consequence, Perceptron kernel method, an extension of original Perceptron, is proposed to deal with high dimensional space.

Suppose that $\phi(x)$ is the image of x in feature space. We rewrite the formula to compute the weight vector in feature space as

$$w = \sum_{i=1}^{n} \alpha y_i \phi(x_i).$$

The *sign* function is presented as:

$$sign(w\phi(x)) = \sum_{i=1}^{n} \alpha y_i x_i x_j$$

One limitation of both Perceptron and Perceptron kernel method is that they are not able to find the optimal linear separation. Normally, a certain linear separation is supposed to not show promissing predicting ability for unseen examples. In the next section, we describe SVM, a kernel method, which aims at finding an optimal hyperplane to separate between positive and negative examples.

2.4.2 Support vector machine

The original Support vector machine is a linear classifier and it was invented by Vladimir N. Vapnik [82]. SVM became popular when Vladimir et al introduced in [15] a way to create nonlinear classifiers by employing the notion of kernel trick. In particular, a SVM searches for an optimal hyperplane in feature space, $\mathbb{H}$, through operations in input space only.

Given a set of training examples $\{(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)\}$ in which $x_i \in \mathbb{R}^n$ and $y_i \in \{\pm 1\}$, we first assume that the examples in the training set are linear separable. SVM tries to learn a linear function

$$f(x) = w^{\intercal} x + b \tag{2.7}$$

where $b \in \mathbb{R}$ is the bias and w is the norm vector. This function forms two half-spaces $h^{+} = \{x : w^{\intercal} x \geq 1\}$ and $h^{-} = \{x : w^{\intercal} x \leq -1\}$. The distance between these two half-spaces is referred as *margin* and equal to $\frac{2}{\|w\|}$. To have all examples are correctly classified, the following condition needs to be satisfied

$$y_i(wx_i + 1) \geq 1 \tag{2.8}$$

The optimal hyperplane is the solution of the below quadratic optimization problem (primal form):

$$\begin{aligned} \underset{w,b}{\text{maximize}} \quad & \frac{2}{\|w\|} \\ \text{subject to} \quad & y_i(wx_i + 1) \geq 1 \end{aligned} \tag{2.9}$$

It is equivalent to

$$\begin{aligned} & \underset{w,b}{\text{minimize}} && \frac{1}{2}\|w\|^2 \\ & \text{subject to} && y_i(wx_i+1) \geq 1 \end{aligned} \tag{2.10}$$

The resulting Lagrange multiplier equation we desire to optimize is

$$L(w,b,\alpha) = \frac{1}{2}\|w\|^2 - \sum_{i=1}^{N} \alpha_i \left[y_i(wx_i+b) - 1\right], \tag{2.11}$$

where $\alpha_i \geq 0$ are Lagrange multipliers. Solving Lagrangian optimization 2.11, we obtain values for w, b and α which determin a unique hyperplane. The $x_i s$ corresponding to $\alpha_i s$ which differ from 0 are called support vectors.

The formula of hyperplane decision function 2.7 can be rewritten as:

$$f(x) = sign(\sum_{i=1}^{N} y_i \alpha_i \langle x, x_i \rangle + b) \tag{2.12}$$

In the case that the training set is not linearly separable, we can apply kernel trick to let SVM operates in Hilbert space through calculating in the input space only. We achieve the following decision function:

$$f(x) = sign(\sum_{i=1}^{N} y_i \alpha_i \langle \phi(x), \phi(x_i) \rangle + b) \tag{2.13}$$

There are usually very few $\alpha_i s$ which are equal to 0. Therefore, it requires a low computation to predict for unseen examples.

In practice, there are two problems that we need to take into account. First, in many cases, the separating hyperplane does not exist due to the high level of noise in data, that is, a large part of examples in one side are located in the other side. Second, the learning function is so complex that it not only fits the examples, but it also fits the noise. Therefore, the learning function is able to classify well for training examples, but it fails to generalize for unseen data. The latter problem is called overfitting. In order to solve such problems, a solution one may think is to allow examples to violate 2.8.

A soft margin SVM is introduced in which it allows to make a trade off between the mistakes on the training set and the complexity of the hypothesis. The optimization 2.10 is modified by introducing slack variables:

$$\begin{aligned} \underset{w,b,\xi}{\text{minimize}} \quad & \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{N}\xi_i \\ \text{subject to} \quad & y_i(wx_i + 1) \geq 1 - \xi_i \end{aligned} \tag{2.14}$$

where $\xi_i \geq 0$ and C is a constant which determines the trade-off between margin maximazation and the training error minimization.

2.5 Kernels on Graphs

Canonical machine learning methods take vectorial data, data that are represented by vectors of features, as their input. However, there are many fields where data are not naturally represented by vectors, but structured forms in which graph is one of the most popular representation. Therefore, the task of developing methods which are able to learn from structured data in general or graphs in particular is very important. In this thesis, we focus on graphs, a special type of structured data representation. An example of data represented by graph is the genetic network where each node represents for a gene and each link is formed between two genes if they encode common protein(s). Another example is the social network whose nodes are users and links depict friendship between users. There are many systems have been proposed that take graphs as their input. These systems can be called as graph-based systems.

One of the key points that determines the performance of a learning system is the similarity measure definition. In our context, that is the definition of similarity measure between graphs. An idea is to find ways that map graphs into vectorial forms: $\mathbb{X} \longmapsto \mathbb{R}^n$, and then employ similarity functions defined on vectors. However, the task of designing these mappings, which are able to encapsulate all information in a vectorial form, is a difficult task since the they need to:

- map isomorphic graphs into the same vector;
- non-isomorphic graphs into different vectors;
- be efficient in terms of time computation and memory consumption.

Recently, kernel methods, whose kernel functions, have emerged as one of the most powerful framework in machine learning. Kernel functions are considered as similarity functions which can be defined on any type of data representation. Therefore, similarity measure defined on graphs are mostly based on kernels. There are two groups of kernels defined on graphs. The first group consists of kernels that aim at measuring the similalrities between graphs and they are referred as graph kernels. To have an overview of graph kernels, we recommend readers to a survey on graph kernels presented in [86]. The second one includes kernels which intend to measure the similarities between nodes inside graphs and are called graph node kernels or node kernels in short. For analysis of different graph node kernels, we suggest to read the work proposed in [28].

In the following, we first give formal definitions and notations related to a graph. We then give an overview of graph kernels followed and graph node kernels.

Definition 2.5.1. A graph, notated as $G = (\mathbb{V}, \mathbb{E}, \mathcal{L}_1, \mathcal{L}_2)$, is a structure which consists of:

- a node (vertex) set $V = \{v_1, v_2, \ldots, v_n\}$,
- a link (edge) set $\mathbb{E} = \{(v_i, v_j)\} \subseteq (\mathbb{V} \times \mathbb{V})$,
- a discrete label function $\mathcal{L}_1 : \mathbb{V} \longmapsto \mathbb{L}$, where $\mathbb{L}$. $\mathcal{L}_1$ assigns a single discrete label $\ell \in \mathbb{L}$ for each ndoe $v \in \mathbb{V}$, $\mathcal{L}_1(v) = \ell$,
- a real vector label function $\mathcal{L}_2 : \mathbb{V} \longmapsto \mathbb{R}^n$. $\mathcal{L}_2$ assigns a single real vector label $(v_1, v_2, \ldots, v_n) \in \mathbb{R}^n$ for each ndoe $v \in \mathbb{V}$, $\mathcal{L}_2(v) = (v_1, v_2, \ldots, v_n)$.

Definition 2.5.2. *An undirected graph is a graph in which edges have no orientation. The edge (u, v) is identical to the edge (v, u), i.e., they are not ordered pairs, but sets $\{u, v\}$ (or 2-multisets) of vertices. The maximum number of edges in an undirected graph without a loop is $n \times (n-1)/2$.*

Definition 2.5.3. *The graph G is unweighted, if $w_{ij} \in \{0, 1\}$, otherwise it is weighted graph.*

Definition 2.5.4. *An adjacency matrix $\boldsymbol{A}$ is a symmetric matrix used to characterize the direct links between vertices v_i and v_j in the graph. Any entry A_{ij} is equal to w_{ij} when there exists a link connecting v_i and v_j, and is 0 otherwise..*

Definition 2.5.5. *The Laplacian matrix $\boldsymbol{L}$ is defined as $\boldsymbol{L} = \boldsymbol{D} - \boldsymbol{A}$, where $\boldsymbol{D}$ is the diagonal matrix with non-null entries equal to the summation over the corresponding row of the adjacency matrix, i.e. $\boldsymbol{D}_{ii} = \sum_j \boldsymbol{A}_{ij}$.*

Definition 2.5.6. *Transition matrix of a graph G, notated as P, is a matrix whose each element $P_{ij} = A_{ij}/\sum_i A_{ij}$ showing the probability of stepping on node j from node i.*

We define the *distance* $\mathcal{D}(u, v)$ between two nodes u and v, as the number of edges on the shortest path between them. The *neighborhood* of a node u with radius r, $N_r(u) = \{v \mid \mathcal{D}(u, v) \leq r\}$, is the set of nodes at distance no greater than r from u. The corresponding *neighborhood subgraph* $\mathcal{N}_r^u$ is the subgraph induced by the neighborhood (i.e. considering all the edges with endpoints in $N_r(u)$). The *degree* of a node u, $deg(u) = |\mathcal{N}_1^u|$, is the cardinality of its neighborhood. The maximum node degree in the graph G is $deg(G)$.

2.5.1 Graph Kernels

The task of designing effient and expressive graph kernels play an important role in the development of graph-based predictive systems. The research on graph kernels has been made possible thanks to the results of Haussler research, convolution or decomposition kernel, [37] (see 2.3). A decomposition kernel is sum over all possible ways to decompose a structured instance of the product of valid kernels over the "parts" of the instance. The "parts" are referred as features. Existing graph kernels are decompositional kernels and can be classified into two categories: sequence-based graph kernels and subgraph-based graph kernels. The sequence-based graph kernels decompose graphs into "parts" in sequence-based forms, such as paths and walks; meanwhile, the subgraph-based graph kernels dissolve graphs into subgraphs. Following we first introduce some consequence-based graph kernels: product graph kernel, shorstest path kernels, we then describe some subgraph-based graph kernels: Weisfeiler-Lehman kernels, The neighborhood subgraph pairwise distance kernel.

2.5.1.1 Product graph kernel

Product graph kernel was originaly proposed in [30] with the aim to measure the similarity between two labeled graphs by counting their common walks. To compute the similarity between two graphs (factor graphs), first a graph, called direct product graph, is constructed from two factor graphs. Then the similarity is then computed based on the obtained graph.

Formally, we consider two factor graphs G_1 and G_2 and $\mathcal{L}_{n1}$, $\mathcal{L}_{n2}$, $\mathcal{L}_{e1}$, $\mathcal{L}_{e2}$ as the node and edge labeling functions of G_1 and G_2, respectively. We define the direct product graph of G_1, G_2 as a graph $G_\times = (V_\times, V_\times)$ where

- $V_\times = \{(u, v) : u \in E(G_1) \wedge v \in V(G_2) \wedge \mathcal{L}_{n1}(u) = \mathcal{L}_{n2}(v)\}$
- $E_\times = \{((u, v), (u', v')) \in V_\times \times V_\times : (u, v) \in E(G_1) \wedge (u', v') \in E(G_2)$ $\wedge\, \mathcal{L}_{e1}((u, v)) = \mathcal{L}_{e2}((u', v'))\}$.

Given $\lambda_1, \lambda_2, \ldots (\lambda_i \in \mathbb{R}, \lambda_i \geq 0,\ \forall i \in \mathbb{N})$, the direct product kernel is defined as follow

$$k_(G_1, G_2) = \sum_{i,j=1}^{|V_\times|} \left[\sum_{n=0}^{\infty} \lambda_n A_\times^n \right]_{ij}, \tag{2.15}$$

if the limit exists, in which $A_\times$ is the adjacency matrix of the direct product graph $G_\times$. The computation of this limit is high, $O(n^6)$. There are different modifications of this kernel that can be more efficient to compute, such as the method proposed in [87] that has complexity of $O(n^3)$.

2.5.1.2 Shortest path kernels

The simple idea behind shortest path kernels is that they consider the common shortest paths between two graphs to measure their similarity. Although the computation of shortest paths in a graph can be computed in polinomial time, taking into account shortest paths leads to some problems. First, the shortest paths between two nodes normally are not unique and there has been a way to deterministically choose one shortest path among others. Second, if we keep all shortest paths into account, it could lead to NP-hard kernel. Although the shorstest paths are not unique, the length of them is unique. As the consequence, a shortest path graph kernel is proposed in [12] with total runtime of $O(n^4)$.

Given two graphs G_1, G_2, we first construct their two corresponding shortest graphs G_{s1}, G_{s2}, respectively. The shortest graph of a graph, G, is a labeled graph defined as $G_s = (V_s, E_s)$ where

- $V_s = V(G)$
- $E_s = \{(u, v) : u, v \in V(G) \wedge |p(u, v)| \geq 1\}$, where $p(u, v)$ is the number of paths connecting u and v.
- $\mathcal{L}_e((u, v)) = \mathcal{D}(u, v)$

We then define the shortest graph kernel for G_1 and G_2 as follow:

$$k_s(G_1, G_2) = \sum_{(u_1,v_1) \in E(G_{s1})} \sum_{(u_2,v_2) \in E(G_{s2})} k^1_{walk}((u_1, v_1), (u_2, v_2)), \tag{2.16}$$

where k^1_{walk} is a positive semi-definite kernel on 1-length walks.

2.5.1.3 Weisfeiler-Lehman kernels

The Weisfeiler-Lehman kernel framework is proposed in [76]. It is derived from the kernel proposed in [75]. The idea is to first decompose each graph into a sequence of graphs. Then the similarity between two graphs is the summation of values computed by a given kernel defined on the two corresponding sequences graphs.

Formally, the Weisfeiler-Lehman graph sequence of a given graph $G = (V, E, \mathcal{L})$ up to the height h is defined as:

$$\{G_0, G_1, \dots, G_h\} = \{(V, E, \mathcal{L}_0), (V, E, \mathcal{L}_1), \dots, (V, E, \mathcal{L}_h)\},$$

where $G_0 = G$, $\mathcal{L}_0 = \mathcal{L}$, $\mathcal{L}_i$s $(i = \overline{1, h})$ are different labeling function on G.

Given two graphs G, G' and a graph kernel k, the Weisfeiler-Lehman kernel, k_{WL} is defined as:

$$k_{WL}(G, G') = k(G_0, G_0^{'}) + k(G_1, G_1^{'}) + \dots + k(G_h, G_h^{'}) \tag{2.17}$$

2.5.1.4 The Neighborhood Subgraph Pairwise Distance Kernel

The NSPDK is an instance of convolution kernel [37] where given a graph $G \in \mathcal{G}$ and two rooted graphs A_u, B_v, the relation $R_{r,d}(A_u, B_v, G)$ is true *iff* $A_u \cong \mathcal{N}_r^u$ is (up to isomorphism $\cong$) a neighborhood subgraph of radius r of G and so is $B_v \cong \mathcal{N}_r^v$, with roots at distance $\mathcal{D}(u, v) = d$ (see the Fig 2.2). We denote R^{-1} as the inverse relation that returns all pairs of neighborhoods of radius r at distance d in G, $R_{r,d}^{-1}(G) = \{A_u, B_v | R_{r,d}(A_u, B_v, G) = \mathit{true}\}$. The kernel $\kappa_{r,d}$ over $\mathcal{G} \times \mathcal{G}$, counts the number of such fragments in common in two input graphs:

$$\kappa_{r,d}(G, G^{'}) = \sum_{\substack{A_u, B_v \in R_{r,d}^{-1}(G) \\ A'_{u'}, B'_{v'} \in R_{r,d}^{-1}(G')}} \mathbf{1}_{A_u \cong A'_{u'}} \cdot \mathbf{1}_{B_v \cong B'_{v'}} \tag{2.18}$$

where $\mathbf{1}_{A \cong B}$ is the *exact matching function* that returns 1 if A is isomorphic to B and 0 otherwise. Finally, the NSPDK is defined as

$$K(G, G') = \sum_r \sum_d \kappa_{r,d}(G, G'), \tag{2.19}$$

where for efficiency reasons, the values of r and d are upper bounded to a given maximal r^* and d^*, respectively.

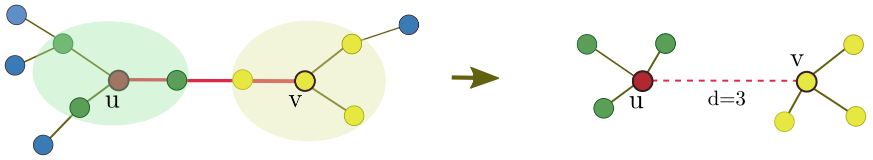

Figure 2.2: Example of a pairwise neighborhood subgraphs rooted at u with radius $r = 1$ and distance $d = 3$

2.5.2 Graph Node Kernels

It is different from graph kernels which aims at measuring similarities between graphs, graph node kernels intend to measure proximities between nodes in graphs. The ideas behind graph node kernels are similar to graph kernels. Therefore, they can be devided into sequence-based graph node kernels and subgraph-based graph node kernels. However, graph node kernels attempt to exploit the configuration concerning two nodes in the graph in order to define their similarity. Most available graph node kernels are sequence-based graph node kernels and they are based on the diffusion phenomenon. In other words, they consider paths connecting two given nodes in order to form their similarity. In the following, we introduce some of the most used graph node kernels.

2.5.2.1 Laplacian exponential diffusion kernel

One of the most well-known kernels for graphs is the Laplacian exponential diffusion kernel **LEDK**, as it is widely used for exploiting discrete structures in general and graphs in particular. On the basis of the heat diffusion dynamics, Kondor and Lafferty proposed **LEDK** in [47]: imagine to initialize each vertex with a given amount of heat and let it flow through the edges until an arbitrary instant of time. The similarity between any vertex couple v_i, v_j is the amount of heat starting from v_i and reaching v_j within the given time. Therefore, **LEDK** can capture the long range relationship between vertices of a graph to define the global similarities. Below is the formula to compute **LEDK** values:

$$K = e^{-\beta \mathbf{L}} = \mathbf{I} - \beta \mathbf{L} + \frac{\beta \mathbf{L}^2}{2!} - ... \tag{2.20}$$

where β is the diffusion parameter and is used to control the rate of diffusion and $\mathbf{I}$ is the identity matrix. Choosing a consistent value for β is very important: on the one side, if β is too small, the local information cannot be diffused effectively and, on the other side, if it is too large, the local information will be lost. **LEDK** is positive semi-definite as proved in [47].

2.5.2.2 Exponential diffusion kernel

In **LEDK**, the similarity values between high degree vertices are generally higher compared to those between low degree ones. Intuitively, the more paths connect two vertices, the more heat can flow between them. This could be problematic since peripheral nodes have unbalanced similarities with respect to central nodes. In order to make the strength of individual vertices comparable, a modified version of **LEDK** is introduced by Chen et al in [21], called Markov exponential diffusion kernel **MEDK** and given by the following formula:

$$\mathbf{K} = e^{-\beta \mathbf{M}} \tag{2.21}$$

The difference with respect to the Laplacian diffusion kernel is the replacement of **L** by the matrix $\mathbf{M} = (\mathbf{D} - \mathbf{A} - n\mathbf{I})/n$ where n is the total number of vertices in graph. The role of β is the same as for **LEDK**.

2.5.2.3 Markov diffusion kernel

The original Markov diffusion kernel **MDK** was introduced by Fouss et al. [29] and exploits the idea of diffusion distance, which is a measure of how similar the pattern of heat diffusion is among a pair of initialized nodes. In other words, it expresses how much nodes "influence" each other in a similar fashion. If their diffusion ways are alike, the similarity will be high and, vice versa, it will be low if they diffuse differently. This kernel is computed starting from the transition matrix **P** and by defining $\mathbf{Z}(t) = \frac{1}{t}\sum_{\tau=1}^{t} \mathbf{P}^{\tau}$, as follows:

$$\mathbf{K} = \mathbf{Z}(t)\mathbf{Z}^{\top}(t) \tag{2.22}$$

2.5.2.4 Regularized Laplacian kernel

Another popular graph node kernel function used in graph mining is the regularized Laplacian kernel **RLK**. This kernel function is introduced by Chebotarev and Shamis in [19] and represents a normalized version of the random walk with restart model. It is defined as follows:

$$\mathbf{K} = \sum_{n=0}^{\infty} \beta^n (-\mathbf{L})^n = (\mathbf{I} + \beta \mathbf{L})^{-1} \tag{2.23}$$

where the parameter β is again the diffusion parameter. **RLK** counts the paths connecting two nodes on the graph induced by taking **-L** as the adjacency matrix, regardless of the path length. Thus, a non-zero value is assigned to any couple of nodes as long as they are connected by any indirect path. **RLK** remains a relatedness measure even when diffusion factor is large, by virtue of the negative weights assigned to self-loops.

2.6 Disease Gene Prioritization

Disease-gene association recovery is a major goal in molecular biology and medical that has received much attention from many researchers. Despite that fact that a big progress has been made in the last decades, a number of genes known to be related to a genetic disease is normally limited. In order to find out the complement set of the known disease gene set, one way is to search for whole genome or specific regions that often contain a large number of suspected genes (candidate genes). This is obvious not a good idea as it is expensive not only in term of time consuming but also from financial aspect. For this reason, a considerable number of gene prioritization methods have been proposed. A gene prioritization method aims at ordering candidate genes from the most to the least probable to be associated to the disease. The top genes in the ranking are then sent to biologists and medical scientists for further studies to determine whether each gene is related to considered disease.

Let us formally define the problem of disease gene prioritization which is later on employed in our empirical experiments to evaluate of different adopted methods. We consider a list of genes $\mathcal{G} = \{g_1, g_2, ..., g_n\}$ that could either be the full list of human genes or a subset of it. Considering a specific disease, there exists a set $P_i \subseteq \mathcal{G}$ of genes known to be associated with it. Its complementary set $U_i = \mathcal{G} - P_i$ contains genes that are not a priori related to the disease, but we assume that inside U_i some positive genes are hidden. Gene priorization is a task that allows to rank the genes in U_i based on their likelihood to be related to P_i.

The identification of the genes underlying human diseases is a major goal in current molecular genetics research. Dramatic progresses have been made since the 1980s, when only a few DNA loci were known to be related to disease phenotypes. Nowadays opportunities for the diagnosis and the design of new therapies are progressively growing, thanks to several technological advances and the application of statistical or mathematical techniques. For instance, positional cloning has allowed to map a vast portion of known Mendelian diseases to their causative genes [77, 16]. However, despite the huge advances, much remains to be discovered. On December 21st 2016, the Online Mendelian Inheritance in Man database (OMIM) registered 4,908 Mendelian phenotypes of known molecular basis and 1,483 Mendelian phenotypes of unknown molecular origin [3]. Moreover, 1,677 more phenotypes were suspected to be Mendelian. But it is among oligogenic and poligenic (and multifactorial) pathologies that the most remains to be elucidated: for the majority of them, only a few genetic loci are known [77, 16].

Independently of the type of disease, the search of causative genes usually concerns a large number of suspects. It is therefore necessary to recognise the most promising candidates to submit to additional investigations, as experimental procedures are often expensive and time consuming. Gene prioritization is the task of ordering genes from the most promising to the least. In traditional genotype-phenotype mapping approaches - as well as in genome-wide association studies - the first step is the identification of the genomic region(s) wherein the genes of interest lie. Once the candidate region is identified, the genes there residing are prioritized and finally analysed for the presence of possible causative mutations [77]. More recently, in new generation sequencing studies this process is inverted as the first step is the identification of mutations, followed by prioritization and final validation [71]. Prioritization criteria are usually based on functional relationships, co-expression and other clues linking genes together. In general, all of them follow the “guilt-by-association” principle, i.e. disease genes are sought by looking for similarities to genes already associated to the pathology of interest [77].

In the last few years, computational techniques have been developed to aid researchers in this task, applying both statistics and machine learning [63]. Thanks to the advent of high-throughput technologies and new generation sequencing, a huge amount of data is in fact available for this kind of investigations. In particular, computational methods are essential for multi-*omics* data integration, that has been recognised as a valuable strategy for understanding genotype-phenotype relationships [70]. In fact, clues are often embedded in different data sources and only their combination leads to the emergence of informative patterns. Furthermore, incompleteness and noise of the single sources can be overcome by inference across multiple levels of knowledge.

Several popular algorithms for pattern analysis are based on *kernels*, which are mathematical transformations that permit to estimate the similarity among items (in our case genes) taking into account complex data relations [74]. Importantly, kernels provide a universal encoding for any kind of knowledge representation, e.g. vectors, trees or graphs. When data integration is required, a multiple kernel learning (MKL) strategy allows a data-driven weighting/selection of meaningful information [35]. The goal of MKL is indeed to learn optimal kernel combinations starting from a set of predefined kernels obtained by various data sources. Through MKL the issue of combining different data types is then solved by converting each dataset in a kernel matrix.

Numerous MKL approaches have been proposed for the integration of genomic data [89, 13] and some of them have been applied to gene prioritization [27, 95, 62, 96]. De Bie *et al* formulated the problem as a one-class support vector machine (SVM) optimization task [27], while Mordelet and Vert tackled it through a biased SVM in a *positive-unlabelled* framework [62, 17]. Recently, Zakeri *et al* proposed an approach for learning non-linear log-euclidean kernel combinations, showing that it can more effectively detect complementary biological information compared to linear combinations-based approaches [96]. However, as highlighted in a recent work by Wang *et al* [89], current methods share two limitations: high computational costs - given by a (at least) quadratic complexity in the number of training examples - and the difficulty to predefine optimal kernel functions to be fed to the MKL machine.

2.7 Biological Datasets

The development of computational biology makes a high number of biological datasets available. Many biological datasets can be naturally represented as networks which are later on used as the input of graph-based biological systems. In biological networks, vertices are biological entities (genes and proteins, etc) and links describe the relation between entities. The relations can be discovered by either physical experiments and results from inferring methods (systems). Following we describe the way to extract information from some biological datasets and transform them into unweighted, undirected networks, represented in the forms of adjacency matrix. These networks will be employed in our experiments for evaluating the performance of different algorithms.

Human Protein Reference Database (**HPRD**) a database of curated proteomic information pertaining to human proteins. It is derived from [44] with 9,465 vertices and 37,039 edges. We employ the HPRD version used in [18] that forms a graph which contains 7311 vertices (represent for genes) and 30503 links. In the graph, two vertices are linked if proteins encoded by their corresponding genes interact.

BioGPS [92]. It contains expression profiles for 79 human tissues, which are measured by using the Affymetrix U133A array. Gene co-expression, defined by pairwise Pearson correlation coefficients (PCC), is used to build an unweighted graph. A pair of genes are linked by an edge if the PCC value is larger than 0.5.

Pathways. Pathway datasets are obtained from the database of KEGG [66], Reactome [84], PharmGKB [91] and PID [72], which contain 280, 1469, 99 and 2679 pathways, respectively. A pathway co-participation network is constructed by connecting genes that co-participate in any pathway.

String [42]. The String database gathers protein information covering seven levels of evidence: genomic proximity in procaryotes, fused genes, co-occurrence in organisms, co-expression, experimentally validated physical interactions, external databases and text mining. Overall, these aspects focus on functional relationships that can be seen as edges of a weighted graph, where the weight is given by the reliability of that relationship. To perform the unbiased evaluation we employed the version 8.2 of String from which we extracted functional links among 17078 human genes.

Phenotype similarity: we use the OMIM [58] dataset and the phenotype similarity notion introduced by Van Driel et al. [80] based on the relevance and the frequency of the Medical Subject Headings (MeSH) vocabulary terms in OMIM documents. We built the graph linking those genes whose associated phenotypes have a maximal phenotypic similarity greater than a fixed cut-off value. Following [80], we set the similarity cut-off to 0.3. The resulting graph has 3393 nodes and 144739 edges.

Biogridphys: this dataset encodes known physical interactions among proteins. The idea is that mutations can affect physical interactions by changing the shape of proteins and their effect can propagate through protein graphs. We introduce a link between two genes if their products interact. The resulting graph has 15389 nodes and 155333 edges.

Biogridgen: Genetic interaction is the phenomenon through which the effects of a gene are modified by one or several other genes. This occurs in indirect way by means of knock-on effects of multiple physical interactions. In practice, this is observed when the effects of two mutations in distinct genes is not equal to the sum of the effects of the mutations alone. This kind of interaction is complementary in respect to the physical one and is important especially for complex diseases involving a large number of genes. In the adjacency matrix, the entries of coordinates (i, j) and (j, i) are equal to 1 if gene i and j interact. Otherwise, they are equal to 0.

Omim: OMIM is a public database of disease-gene association. Genes implicated in the same disease are more likely to be involved in other similar diseases as well. Therefore, Omim network is formed by connecting genes which are involved in common disease(s).

2.8 Link Prediction

We are witnessing a constant increase of the rate at which data is being produced and made available in machine readable formats. Interestingly it is not only the quantity of data that is increasing, but also its complexity, i.e. not only are we measuring a number of attributes or features for each data point, but we are also capturing their mutual relationships, that is, we are considering non independent and identically distributed (non i.i.d.) data. This yields collections that are best represented as graphs or relational data bases and requires a more complex form of analysis. As cursory examples of application domains that are social networks, where nodes are people and edges encode a type of association such as friendship or co-authorship, bioinformatics, where nodes are proteins and metabolites and edges represent a type of chemical interaction such as catalysis or signaling, and e-commerce, where nodes are people and goods and edges encode a "buy" or "like" relationship. A key characteristic of this type of data collections is the sparseness and dynamic nature, i.e. the fact that the number of recorded relations is significantly smaller than the number of all possible pairwise relations, and the fact that these relations evolve in time. A crucial computational task is then the "link prediction problem" which allows to suggest friends, or possible collaborators for scientists in social networks, or to discover unknown interactions between proteins to explain the mechanism of a disease in biological networks, or to suggest novel products to be bought to a customer in a e-commerce recommendation system. Many approaches to link prediction that exist in literature can be partitioned according to *i*) whether additional or "side" information is available for nodes and edges or rather only the network topology is considered and *ii*) whether the approach is unsupervised or supervised.

Unsupervised methods are non-adaptive (i.e. they do not have parameters that are tuned on the specific problem instance), and can therefore be computationally efficient. In general they define a score for any node pair that is proportional to the existence likelihood of an edge between the two nodes. *Adamic-Adar* [5, 54] computes the weighted sum over the common neighbors where the weight is inversely proportional to the (log of) each neighbor node degree. The *preferential attachment* [11] method computes a score simply as the product of the node degrees in an attempt to exploit the "rich get richer" property of certain network dynamics. *Katz* [43] takes into account the number of common paths with different lengths between two nodes, assigning more weight to shorter paths. The *Leicht-Holme-Newman* method [50] computes the number of intermediate nodes. In [59] the score is derived from the singular value decomposition of the adjacency matrix. Two methods, named Local Random Walk (LRW) and Superposed Random Walk (SRW), are proposed in [53]. These methods work based on random walk. Besides, graph node kernels, including [47, 21, 29, 19], can be applied to measure the similarities between nodes and link prediction is made based on these similarities. For more information of link prediction methods, we refer readers to [55, 59].

Supervised link prediction methods convert the problem into a binary classification task where links present in the network (at a given time) are considered as positive instances and a subset of all the non links are considered as negative instances. Following [59], we can further group these methods into four classes: feature-based models, graph regularization models, latent class models and latent feature models. A Bayesian nonparametric approach is used in [60] to compute a nonparametric latent feature model that does not need a user defined number of latent features but rather induces it as part of the training phase. In [59] a matrix factorization approach is used to extract latent features that can take into consideration the output of an arbitrary unsupervised method. The authors show a significant increase in predictive performance when considering a ranking loss function suitable for the imbalance problem, i.e. when the number of negative is much larger than the number of positive instances.

In general supervised methods exhibit better accuracies compared to unsupervised methods although incurring in much higher computational and memory complexity costs.

2.9 Biological Data Integration

Data integration has been attracted many researchers because of its important role in building high performance biological learning systems (as discussed in the Chapter 1). As a consequence, there is a high number of data integration methods which have been proposed in the last decades. According to [33], existing methods can be divided into three classes: early data integration, late data integration and intermediate data integration.

Early data integration first combines different data sources into a single one. It then builds a model for inference. Typical approaches are proposed in [48, 98, 62, 23, 45]. A common requirement for the methods in this class is that data sources need to be transformed into a common representation. This might lead to the problem of information loss.

Late data integration builds models for each data source separately. It then combines different obtained models to have a unified one. A common technique for combining is to use the majority voting policy. Late data integration methods often show relatively low performance since models are built from each dataset in isolation from others. Examples in this class are [31, 88, 57, 64].

Intermediate data integration combines data through inference of a joint model. An advantage of this strategy is that it does not require any data transformation. Therefore, it does not lead to the problem of information loss. Usually, it shows high performance in many applications, including [48, 31, 81, 98, 67].

2.10 Multiple Kernel Leanring

A common way to represent data in data integration is using kernels since kernels are well-known as universal methods for data representation (see 2.3). First kernels are defined on each data source. Then the obtained kernels are combined into a single one which has higher abstract level of data representation. The combination is often performed by using multiple kernel learning (MKL) algorithms [35, 89] (see [35] for a recent and quite exhaustive survey).

The task of Multiple Kernel Learning is to combine kernels derived from multiple sources in a data-driven way with the aim of improving the accuracy of a target kernel machine. MKL algorithms are normally in linear forms because of the two following reasons. First, the time required to solve the associated optimization problem grows, normally more than linearly, w.r.t the number of pre-defined kernels. Second, employing sophisticated algorithms often do not significantly outperform the simple average of kernels. However, most of them still require a long computation time and a high memory consumption, especially when the number of pre-defined kernels is high. To tackle these limitations, a scalable multiple kernel learning named EasyMKL has been previously proposed [9]. This method focuses on learning a linear combination of the input kernels with positive linear coefficients, namely

$$\mathbf{K} = \sum_{r=1}^{R} \eta_r \mathbf{K}_r, \ \ \eta_r \geq 0 \,, \tag{2.24}$$

where $\eta = (\eta_1, \ldots, \eta_R)$ is the coefficient vector. In a fully supervised binary task, EasyMKL computes the optimal kernel by maximizing the distance between positive and negative examples. The base learner is a kernel-based approach for the optimization of the margin distribution in binary classification or ranking [8].

In order to present its formulation, let us first define the probability distribution $\gamma \in \mathbb{R}_+^N$ representing weights assigned to training examples and living in the domain $\Gamma = \{\gamma \in \mathbb{R}_+^N | \sum_{i \in \mathcal{P}} \gamma_i = 1, \sum_{i \in \mathcal{N}} \gamma_i = 1\}$, where $\mathcal{N}$ is the set of negative examples. From this definition, it follows that any element $\gamma \in \Gamma$ represents a pair of points in the input space: the first one is constrained to the convex hull of positive training examples and the second one to the convex hull of negative training examples. As stated above, EasyMKL maximizes the distance between positive and negative examples, optimizing the margin distribution at the same time. Under this notation, the task can be posed as a min-max problem over variables γ and η as follows:

$$\max_{\eta : \|\eta\|_2 \leq 1} \min_{\gamma \in \Gamma} (1-\lambda)\gamma^\top \mathbf{Y}(\sum_r \eta_r \mathbf{K}_r)\mathbf{Y}\gamma + \lambda\, \gamma^\top \gamma \,. \tag{2.25}$$

Here $\mathbf{Y}$ is a diagonal matrix containing the vector of example labels, +1 for the positive and -1 for the negative. Optimization of the first term alone leads to an optimal probability distribution γ^* representing the two nearest points in the convex hulls of positive and negative examples, equally to a hard SVM task using a kernel $\mathbf{K}$ [8]. The second term represents a quadratic regularization over γ whose objective solution is the squared distance between positive and negative centroids in the feature space. The regularization parameter $\lambda \in [0, 1]$ permits to tune the objective to optimize, by balancing between the two critical values $\lambda = 0$ and $\lambda = 1$. When $\lambda = 0$ we obtain a pure hard SVM objective, while when $\lambda = 1$ we get a centroid-based solution.

It can be shown that this problem has analytical solution in the η variable, so that the previous expression can be reshaped into:

$$\min_{\gamma \in \Gamma} (1-\lambda)\gamma^\top \mathbf{Y}\mathbf{K}^s\mathbf{Y}\gamma + \lambda\, \gamma^\top \gamma \,, \tag{2.26}$$

where $\mathbf{K}^s = \sum_r^R \mathbf{K}_r$ is the sum of the pre-defined kernels. This minimization can be efficiently solved and only requires the sum of the kernels. The computation of the kernel summation can be easily implemented incrementally and only two matrices need to be stored in memory at a time. As shown in [9], EasyMKL can deal with an arbitrary number of kernels using a fixed amount of memory and a linearly increasing computation time.

Once the problem in Eq. 2.26 is solved, we have an optimal distribution γ^* and we are able to obtain the optimal kernel weights η_r^* by using the formula:

$$\eta_r^* = \frac{\gamma^* \mathbf{Y} \mathbf{K}_r \mathbf{Y} \gamma^*}{\sum_{r=1}^{R} \gamma^* \mathbf{Y} \mathbf{K}_r \mathbf{Y} \gamma^*} \,. \tag{2.27}$$

The optimal kernel is thus evaluated as $\mathbf{K}^* = \sum_r^R \eta_r^* \mathbf{K}_r$. Finally, by replacing $\mathbf{K}^s$ with $\mathbf{K}^*$ in Eq. 2.26, we can get the final probability distribution γ^*.

Chapter 3

Conjunctive Disjunctive Graph Node Kernel

In this chapter, we propose a graph node kernel, named conjunctive disjunctive node kernel (CDNK), which is an instance of convolutional kernels. To measure similarities between nodes in large-scale graphs, our graph node kernel not only effectively exploits graph structures, but also takes the side information (auxiliary information) associated to graph nodes into account. The empirical evaluation results on several datasets shows that CDNK significantly outperforms various famous graph node kernels.

3.1 Motivation

As discussed before, node similarity measure definition is the key that determines the performance of graph-based learning systems. The state of the art graph node kernels used to measure node similarity, are based on the notion of information diffusion, including LEDK [47], MEDK [21], MDK [29], RLK [19], etc. These graph node kernels often show relatively low discriminative capacity, especially in cases of working with sparse graphs, i.e, graphs whose high numbers of missing links, since they share following limitations. First information are processed in an additive and independent fashion which prevents them from accurately modeling the configuration of each node's context. Second, they do not take into account auxiliary information associated to nodes of graphs when they are available. These additional information normally provide a complement to graph topology.

Therefore, they are potential to use to improve the expressiveness of graph node kernels.

We propose an effective convolutional graph node kernel, *Conjunctive disjunctive node kernel*, which is able to *i*) effectively exploit the nodes' context, *ii*) process with side information sticked on nodes of graphs.

3.2 Conjunctive Disjunctive Graph Node Kernel

We start from the type of similarity notion computed by a neighborhood based decomposition kernel between graph instances [26], NSPDK, and adapt it to form a convolutional graph node kernel which aims at expressing the similarity between nodes in a single graph. Our intended graph node kernel takes an undirected, labeled graph as its input.

Given an input labeled graph $G = (\mathbb{V}, \mathbb{E}, \mathcal{L}_1, \mathcal{L}_2)$, our kernel consists of two phases. In the first phase, a network decomposition procedure is applied to transform the graph into a set of linked sparse connected components. In this procedure, we define two different kinds of link: *conjunctive* and *disjunctive* in which we treat them in distinct manners. In the second phase, the similarity between any node couple $(u,\ v)$, $u, v \in \mathbb{V}$ is computed by adopting NSPDK on two neighborhood subgraphs rooted as u and v which extracted from the resulted graph after decomposing. In the following, we describe each phase in detail.

3.2.1 Network Decomposition

In genetic networks, it is not uncommon to find nodes with high degrees. Unfortunately these cases cannot be effectively processed by a neighborhood based decomposition kernel (see 2.5.1.4) since they are based on the notion of exact matches. Neighborhood subgraphs rooted at high degree nodes are relatively big due to a high number of neighbors. It leads to a low likelihood of finding neighborhood subgraphs rooted at high degree nodes which are isomorphic. Thus, the probability of having identical neighborhoods decreases exponentially as the degree increases. This means that in a finite network it quickly becomes impossible to find any match and hence learn or generalize at all.

We propose a procedure to "sparsify" the network that is observed by the neighborhood kernel. In practice, we mostly keep the same the cardinality of the edge set. However we mark the edges with special attributes so that kernel is able to treat them differently when computing. The result is a procedure that decomposes the network in a linked collection of sparse

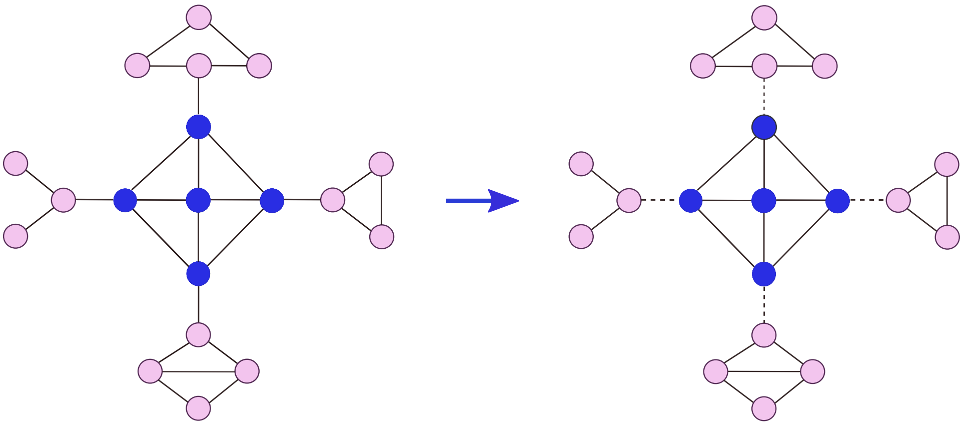

Figure 3.1: K-core decomposition with degree threshold $D = 4$

sub-networks where each node has a reduced connectivity when considering the edges of a specific type. However the other edges are still available to connect the various sub-networks. We distinguish two types of edges: *conjunctive* and *disjunctive* edges. Nodes linked by conjunctive edges are going to be used jointly to define the notion of context and will be visible to the neighborhood graph kernel. Nodes linked by disjunctive edges are instead used to define features based only on the pairwise co-occurrence of the genes at the endpoints and are processed by our novel kernel.

The network decomposition works by first applying the iterative k-core decomposition and then the clique decomposition. In what follows, we describe both k-core and clique decomposition procedure.

Iterative k-core decomposition [10]: The node set is partitioned in two groups on the basis of the degree of each node w.r.t. a threshold degree D, the first part contains all nodes with degree smaller than or equal to D and the second part the remaining ones. The node partition is used to induce the "conjunctive" vs "disjunctive" notion for the edge partition: edges that have both endpoints in the same part are marked as conjunctive, otherwise they are marked as disjunctive. We apply the k-core decomposition iteratively, where at each iteration we consider only the graph induced by the conjunctive edges. We stop iterating the decomposition after a user defined number of steps. Note that this decomposition does not alter the cardinality of the edge set, it is simply a procedure to mark each edge with the attribute conjunctive or disjunctive.

Clique decomposition [79]: To model the notion that nodes in a clique are tightly related, we summarize the whole clique with a new "representative" node. All the cliques (completely connected subgraphs) with a number

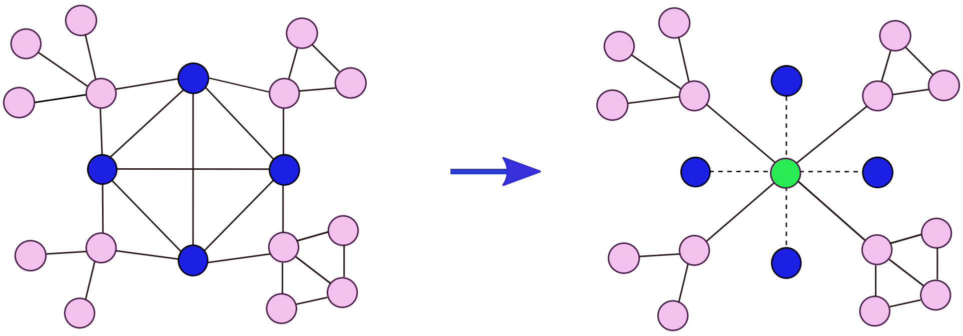

Figure 3.2: Clique decomposition with threshold $C = 4$

of nodes greater than or equal to a given threshold size C are identified. The endpoints of all edges incident on the clique's nodes are moved to the representative node. Disjunctive edges are introduced to connect each node in the clique to the representative. Finally all edges with both endpoints in the clique are removed.

3.2.2 Kernel Definition

We define a node kernel $K(G_u, G_{u'})$ between two copies of the same network G where we distinguish the nodes u and u' respectively. The idea is to define the features of a node u as the subset of NSPDK features that always have the node u as one of the roots. In addition we distinguish between two types of edges, called *conjunctive* and *disjunctive* edges. When computing distances to induce neighborhood subgraphs, only conjunctive edges are considered. When choosing the pair of neighborhoods to form a single feature, we additionally consider roots u and v that are not at distance d but such that u is connected to w via a disjunctive edge and such that w is at distance d from v (The Figure 3.3 is an illustration). In this way disjunctive edges can still allow an *information flow* even if their endpoints are only considered in a pairwise fashion and not jointly.

Formally, we define two relations: the *conjunctive relation* $R^{\wedge}_{r,d}(A_u, B_v, G_u)$ identical to the NSPDK relation $R_{r,d}(A_u, B_v, G)$, and (ii) $\mathcal{D}(u, v) = d$; the *disjunctive relation* $R^{\vee}_{r,d}(A_u, B_v, G_u)$ is true *iff* (i) $A_u \cong \mathcal{N}^u_r$ and $B_v \cong \mathcal{N}^v_r$ are true, (ii) $\exists w$ s.t. $\mathcal{D}(w, v) = d$, and (iii) (u, w) is a disjunctive edge. We define $\kappa_{r,d}$ on the inverse relations ${R^{\wedge}_{r,d}}^{-1}$ and ${R^{\vee}_{r,d}}^{-1}$:

$$\kappa_{r,d}(G_u, G_{u'}) = \sum_{\substack{A_u, B_v \in {R^{\wedge}_{r,d}}^{-1}(G_u) \\ A'_{u'}, B'_{v'} \in {R^{\wedge}_{r,d}}^{-1}(G_{u'})}} \mathbf{1}_{A_u \cong A'_{u'}} \cdot \mathbf{1}_{B_v \cong B'_{v'}} + \sum_{\substack{A_u, B_v \in {R^{\vee}_{r,d}}^{-1}(G_u) \\ A'_{u'}, B'_{v'} \in \, {R^{\vee}_{r,d}}^{-1}(G_{u'})}} \mathbf{1}_{A_u \cong A'_{u'}} \cdot \mathbf{1}_{B_v \cong B'_{v'}},$$

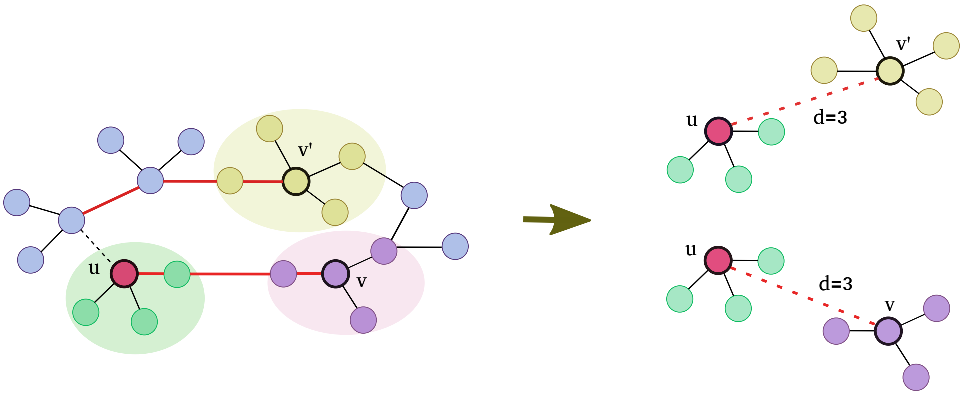

Figure 3.3: Pairwise neighborhood subgraphs for the “red” node with $r = 1$ and $d = 3$ using “*conjuction*” and “*disjuctive*” edges

where $\mathbf{1}_{A_u \cong B_v}$ is the matching function which returns 1 if A_u and B_v are isomorphic and 0 otherwise (see [26] for an efficient approximate function for graph isomorphism).

The CDNK is finally defined as $K(G_u, G_{u'}) = \sum_r \sum_d \kappa_{r,d}(G_u, G_{u'})$, where for efficiency reasons, the values of r and d are upper bounded to a given maximal r^* and d^*.

In order to integrate the information of real vector labels, we proceed as follows. We compute a sparse vector representation for the neighborhood graph rooted in node v following [26]: for each neighborhood subgraph we calculate the quasi-isomorphism certificate hash code; we then combine the hashes for the pair of neighborhoods and use the resulting integer as a feature indicator. This yields a direct sparse vector representation (associated to node u in graph G) $f : G_u \longmapsto I\!R^N$ where $N \approx 10K - 1M$. Given the real valued vector information (associated to node u in graph G) $g : G_u \longmapsto I\!R^P$ computed as the multi-class similarity to the P clusters (c.f.r. Section 3.4.2), we update the computation of CDNK considering the discrete convolution of the discrete information with the real valued information:

$$K(G_u, G_{u'}) = \left\langle f(G_u) \bigotimes g(G_u), f(G_{u'}) \bigotimes g(G_{u'}) \right\rangle, \tag{3.1}$$

where the discrete convolution is defined as: $(f \bigotimes g)[n] = \sum_{m=0}^{K-1} f[n - m]g[m]$. In words, we are starting a scaled copy of the real valued vector at the position indicated by each feature computed on the basis of the discrete information. Intuitively, when both the real valued and the discrete information match, the kernel computes a large similarity, but if there is a discrepancy in either one of the sources of information, the similarity will be penalized.

3.3 Parameter Space

Our kernel consists of five parameters: the threshold degree D, clique size threshold C, maximal radius r^*, maximal distance d^* and number of clusters P clusters in which their values are in $\mathbb{N}^*$. In order to choose the optimal tuple of parameters for kernel in a specific setting, a model selection procedure is normally adopted from a determined subset of parameter space. The parameter space of our kernel seems large due to the relatively high number of parameters. However, most parameter values are supposed to be in limited natural ranges. Therefore, it is actually not too big. In particular, the clique size threshold C is recommended to be in $\{4, 5\}$. The maximal radius r^* is set with a value smaller than or equal to 3 since it will lead to big neighborhood subgraphs (as discussed in 3.2.1). The maximal distance d^* is assigned with values less than or equal to 4 because the value of the maximal shortest distance between nodes in a connected component is often not too high. The values of the threshold degree D should neither be too high and too low. If it is high, we have to face with high degree node problem, and if it is too low, the obtained graph contains of too sparse connected components. Therefore, we suggest the value for D in $[6, 20]$.

3.4 Empirical Evaluation

In this section, we desire to evaluate the performance of CDNK and other graph node kernels to answer for the two following questions:

- Does CDNK show better performance comparing to other graph node kernels?
- Does the use of side information (real vector labels) help to improve the performance of CDNK?

3.4.1 Experimental Settings and Evalution Method

We carry out experiments in the context of disease gene prioritization (see 2.6). Given a genetic graph and a list of training genes known to cause a genetic disease. We first apply a graph node kernel to compute kernel matrix. This kernel matrix together with training gene set are used as the input of a learning algorithm to construct a model. The obtained model is then used to prioritize for candidate genes.

The experiments are performed on two separate networks derived from BioGPS and Pathways datasets, described in 2.7, in which we follow the experiment procedure in [21] where 12 diseases [34] are used in which each disease is associated to at least 30 positive genes (see table A.1 for the list of genetic diseases and the number of positive genes in each disease). For each disease, we construct a positive set $\mathcal{P}$ with all positive (confirmed) disease genes, and a negative set $\mathcal{N}$ which contains random genes associated at least to one disease class which is not related to the class that is defining the positive set. In [21] the ratio between the dataset sizes is chosen as $|\mathcal{N}| = \frac{1}{2}|\mathcal{P}|$. This is due to the fact that genes known to be related to at least a genetic disease, but not to the considered one are well studied, so they have low probability to be associated to the current disease.

To compare the performance of graph node kernels, we fix the learning algorithm and sequentially substitute diffent graph node kernels. The performance of the system obtained in different cases are used to compare the performance of the employed graph node kernels: LEDK, MEDK, MDK, RLK and CDNK.

The predictive performance of the system using each kernel is evaluated via a leave-one-out cross validation: one gene is kept out in turn and the rest are used to train an SVM model. We compute a decision score q_i for the test gene g_i as the top percentage value of score s_i among all candidate gene scores. We collect all decision scores for every gene in the test set to form a global decision score list on which we compute the AUC-ROC.

3.4.2 Node Labeling

Our graph node kernel takes the labled graphs as its input in which labels can be discrete and/or real valued vectors. Therefore, to be able to carry out experiments on graphs derived from BioGPS and Pathways, we need ways to label for these genetic graphs. Following are our proposed node labeling methods for genetic networks.

Discrete labels: We cast two different approaches to associate genes with dicrete labels.

The first approach, we use a same label for every node of graphs. In this case, the configuration of nodes expressed by the pairwise neighborhood subgraphs in CDNK now turn to pairwise neighborhoods. Besides, since labels are identical, the cardinality of neighborhoods are took into account.

The second approach aims to use nodes labels to encode abstract information about the genes. In this way downstream machine learning algorithms can generalize from similar examples and allow the identification of overlooked but related genes. We employ a gene ontology [24] to construct binary vectors representing a bag-of-words encoding for each gene, i.e. each element of a binary vector is equal to 1 if its corresponding GO-term is associated to the gene, and is equal to 0 otherwise.). The resulting vectors are then clustered using the k-means algorithm into a user defined number of classes, P, so that genes with similar description profiles receive the same class identifier as label.

Real vector labels: In addition to encoding the functional information as a discrete label we add a richer description by computing the similarity vector w.r.t. to each cluster. In this way we can fully exploit the latent description of the genes in terms of the different functional groups captured by the clustering procedure. Formally, given a vector $v \in I\!R^{26501}$ we compute a similarity vector $S(v) = s_1, s_2, \dots s_P$ with entries $s_i = \frac{1}{1+\ell(v,c_i)}$ where $\ell(v, c_i)$ is the Euclidean distance of v from the center of the i^{th} cluster $c_i = \frac{1}{|C_i|}\sum_{x \in C_i} x$ computed as the geometric mean of the elements in the cluster C_i.

3.4.3 Model Selection

The hyper parameters of the various methods are tuned using a k-fold cross validation. However due to the non i.i.d. nature of the problem, we employ a stronger setup to ensure no information leakage. The dataset on which we are validating the performance is never subsequently used in the predictive performance estimation. The values for diffusion parameter β in LEDK and MEDK are sampled in $\{10^{-3}, 10^{-3}, 10^{-2}, 10^{-1}\}$, time steps t in MDK in $\{1, 10, 100\}$ and RLK parameter β in $\{1, 4, 7\}$. For CDNK, the degree threshold values, D, are sampled in $\{10,\ 15,\ 20\}$, clique size threshold, C, in $\{4,\ 5\}$, maximum radius, r^*, in $\{1,\ 2\}$, maximum distance, d^*, in $\{2,\ 3,\ 4\}$, number of clusters P in $\{5,\ 7\}$. Finally, the regularization trade off parameter C for the SVM is sampled in $\{10^{-5},\ 10^{-4},\ 10^{-3},\ 10^{-2},\ 10^{-1}, 1,\ 10,\ 10^{2}\}$.

3.5 Results and Discussion

Figure 3.4 and 3.5 show the AUC-ROC performance of different models using state of the art graph node kernels and different variations of CDNK on BioGPS and Pathways networks, respectively. In what follows, we analyze the results to answer for the two questions raised in 3.4.

Concering the performance of different graph node kernels, CDNK variations outperform diffusion-based graph node kernels in all cases on BioGPS dataset and 8 out of 11 cases on Pathways when compared to diffusion-based kernels. CDNK is ranked first when considering both the average AUC-ROC and the average rank (see table A.2 and A.3) with a difference, w.r.t. the best diffusion-based kernel, ranging from 5.4% to 10% and from 1.2% to 3.4% on BioGPS and Pathways, respectively. As a consequence, we can conclude that CDNK show state of the art performance in graph node kernels.

Regarding the variations of CDNK, the integration of real valued vectors improves the performance in most cases. In particular, considering the use of discrete labels based on ontology, using side information helps to increase the performance in all diseases on BioGPS and in 9 out of 11 diseases on Pathways. Similarly, by employing auxiliary information associated to graph nodes, the performance of CNDK in case of using uniform discrete labels is also improved in 9 and 8 out of 11 on BioGPS and Pathways, respectively.

3.6 Conclusion and Future Work

We have shown how decomposing a network in a set of connected sparse graphs allows us to take advantage of the discriminative power of CDNK, a novel decomposition kernel, to achieve state-of-the-art results. Moreover, we have also introduced the way to integrate "side" information in form of real valued vectors when it is available on graph to get even better performance of CDNK. In future work we will investigate how to *i*) decompose networks in a data driven way and *ii*) extend the CDNK approach to gene-disease association problems exploiting multiple heterogeneous information sources in a joint way.

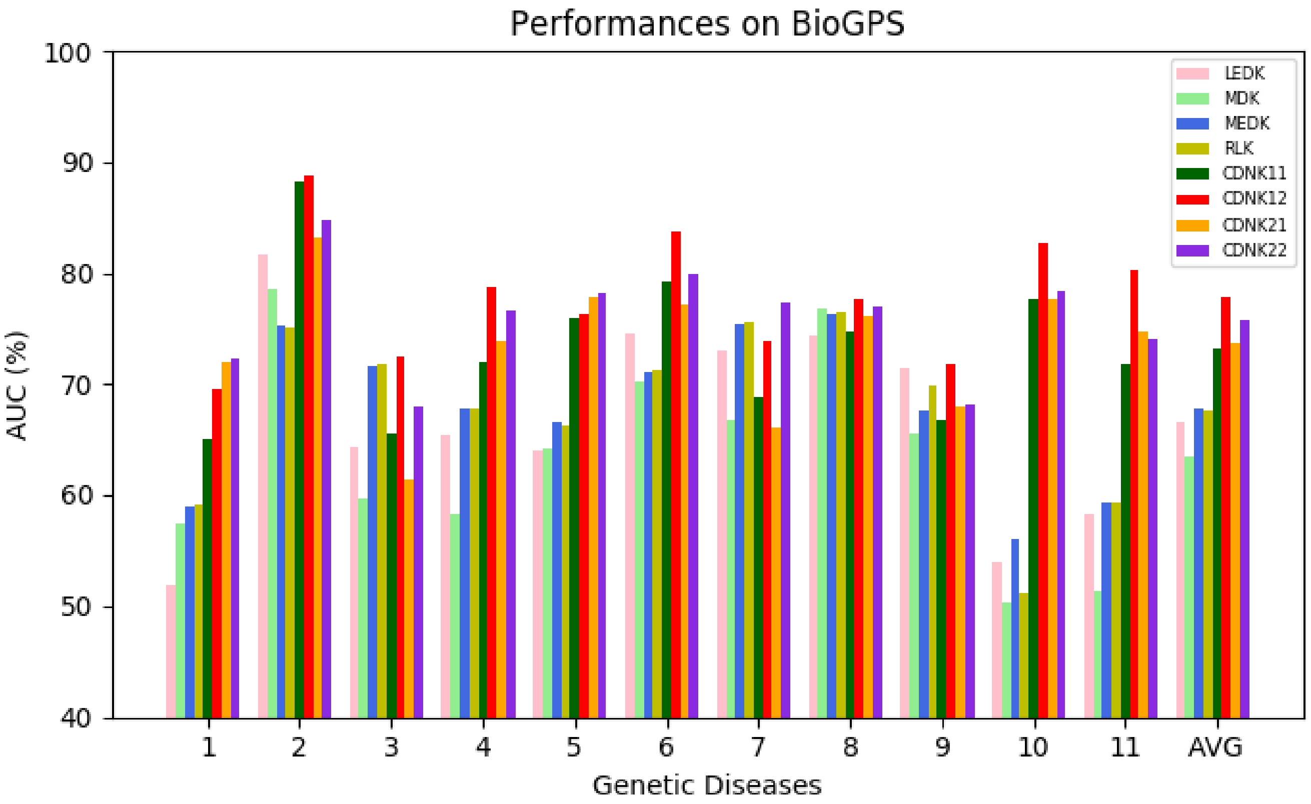

Figure 3.4: *AUC-ROC performance on 11 genetic diseases using network induced by the BioGPS. CDNK1: ontology for discrete labels, CDNK2: ontology for both discrete and vector labels, CDNK3: node degree for discrete labels and CDNK4: node degree for discrete labels and ontology for vector label.*

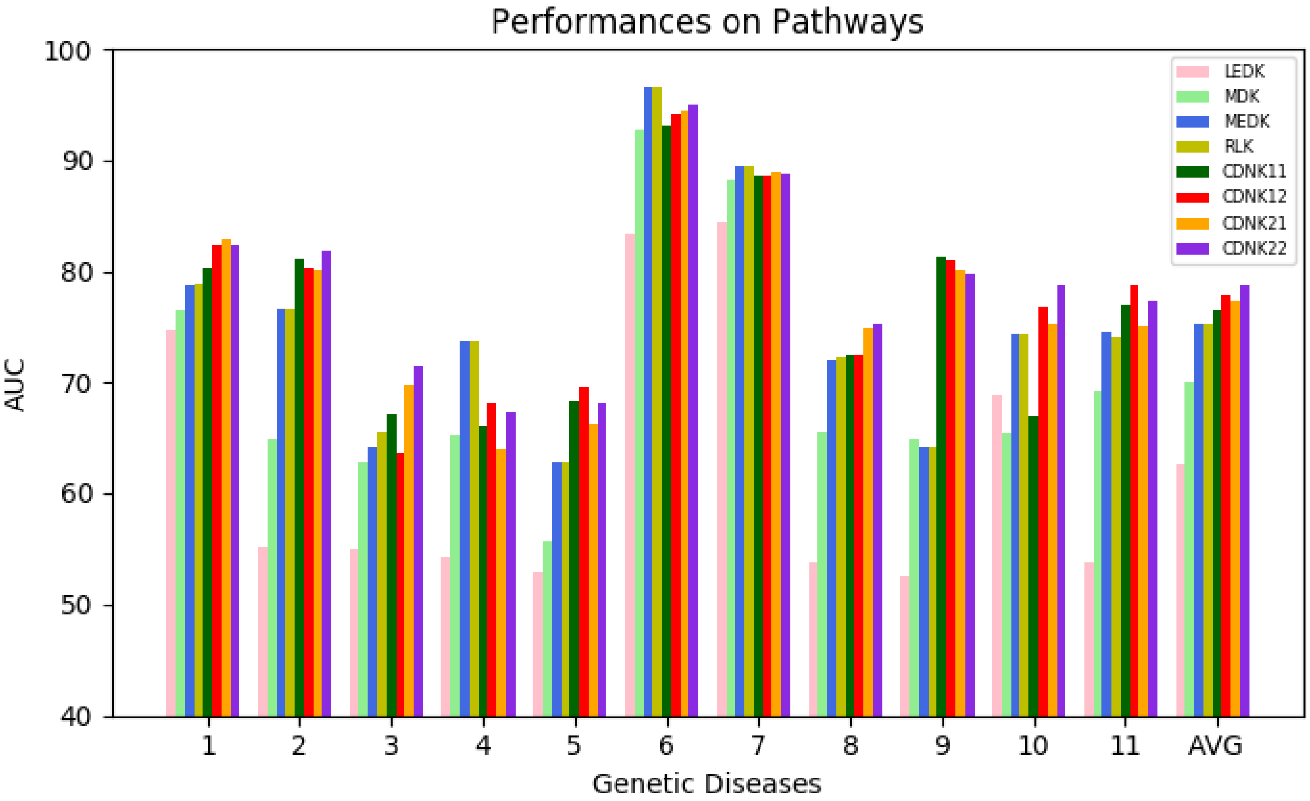

Figure 3.5: *AUC-ROC performance on 11 genetic diseases using network induced by the BioGPS. CDNK1: ontology for discrete labels, CDNK2: ontology for both discrete and vector labels, CDNK3: node degree for discrete labels and CDNK4: node degree for discrete labels and ontology for vector label.*

Chapter 4

Solutions for Graph Sparsity

One of the problems that we often face with when constructing graph-based learning systems is the graph sparsity where input graphs have high numbers of missing links. In this case, the systems can only access to limited information, so they are not able to learn effectively. This leads to low performance of the learning systems. In this chapter, we aim at investigating and proposing solutions to deal with graph sparsity problem.

4.1 Motivation

As we have discussed in the Chapter 1, the number of graph-based learning systems is getting higher over time. Despite the increase of relational data, which are best presented by graphs, in terms of diversity and amount, the relations of entities encoded in data are much smaller than the number of possible relations. This yields the problem of graph sparsity when we employ graphs to represent for the relations of entities. The graph sparsity problem often leads to low performance of graph-based learning systems since the information, which are available, for learning procedure is restricted. For this reason, we need effective solutions to overcome the graph sparsity issue.

An effective solution for graph sparsity is to use link enrichment methods whose the aim is to find top un-observed links, links whose highest probabilities of being missing links, to add into sparse graphs. However, the performance of link enrichment directly depends on the employed link prediction methods. Therefore, in the following sections, we first propose a novel, effective link prediction method, named Joint Neighborhood Subgraphs Link Prediction. We then propose a paradigm, called Link Enrichment for Diffusion-based Graph Node Kernels, to boost the performance of graph-based kernels that usually show low performance on sparse graphs.

4.2 Joint Neighborhood Subgraphs Link Prediction

Due to the importance of link prediction, there is a high number of link prediction approaches which have been proposed and applied in a wide range of fields (see 2.8). They can be classified into supervised and unsupervised learning group. Supervised methods normally show better accuracies compared to unsupervised methods. However, they face with much higher computational and memory complexity costs. Most link reprediction methods in both categories implicitly represent the link prediction problem and the inference used to tackle it as a disjunction over the edges, that is, information on edges is propagated in such a fashion so that for a node to have k neighbors or $k+1$ does not make a drastic difference. We claim that this hypothesis is likely putting a cap on the discriminative power of classifiers and therefore we propose a novel supervised method, JNSL, that employs a conjunctive representation. We call the method "joint neighborhood subgraphs link prediction" (JNSL). The key idea here is to transform the link prediction task into a binary classification on suitable small subgraphs which we then solve using an efficient convolutional graph kernel method.

Given a labeled, unweighted graph, $G = (\mathbb{V}, \mathbb{E}, \mathcal{L}_1, \mathcal{L}_2)$, the set $\mathbb{E}$ is partitioned into the subset of observed links ($\mathcal{O}$) and the subset of unobserved links (U). Like other approaches we assume that all un-observed links are indeed "non-links" and we therefore define the link prediction problem as the task of ranking candidate links from the most to the least probable to recover links in $\mathcal{O}$ but not in U exploiting only the network topology. In the next sections, we first describe how links are represented in our method, and we then show how we cast link prediction as a binary classication problem.

4.2.1 Link encoding as subgraphs union

Most methods for link prediction compute pairwise nodes similarities treating the nodes defining the candidate edge independently. Instead we propose to jointly consider both candidate endpoint nodes together with their extended "context". To do so we build a graph starting from the two nodes and the underlying network. Given nodes u and v, we first extract the two neighborhood sets with a user defined radius R rooted at u and v to obtain

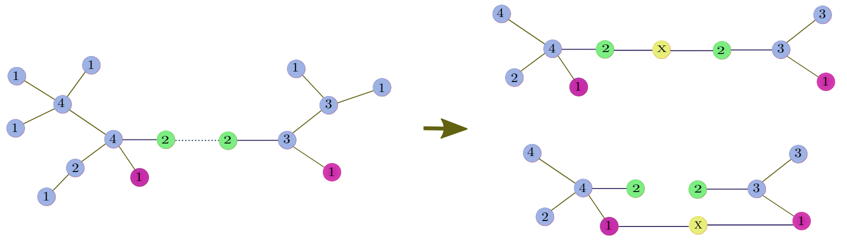

Figure 4.1: Left) We represent with solid lines edges belonging to the training material and with a dotted line edges belonging to the test material. Right) joint neighborhood subgraphs for an existing (green endpoints) (top) and a non existing (red endpoints) link (bottom). These graphs will receive respectively a positive and a negative target.

$N_R(u)$ and $N_R(v)$, respectively. We then consider the graph $\mathcal{J}$ induced by the set union $N_R(u) \cup N_R(u)$. Finally we add an auxiliary node w and the necessary edges to connect it to u and v (see Figure4.1).

4.2.2 Joint neighborhood subgraphs link prediction

In the link prediction problem we are given a graph $G = (\mathbb{V}, \mathbb{E}, \mathcal{L}_1, \mathcal{L}_2)$ and a binary target vector $Y = \{y_{(0,0)}, y_{(0,1)}, \cdots, y_{(|V|,|V|)}\}$ where $y_{(u,v)} = 1$ if $(u, v) \in E$ and 0 otherwise. The training data is obtained considering a random subset of edges in $E^{tr} \in E$ and inducing a training graph $G^{tr} = (V, E^{tr})$. Note that the graph used for training does not contain any of the edges that will be queried in the test phase. The remaining edges $E^{ts} = E \setminus E^{tr}$ are used to partition the target vectors: $Y^{tr} = \{y_{(u,v)} | (u, v) \in E^{tr}\}$, $Y^{ts} = \{y_{(u,v)} | (u, v) \in E^{ts}\}$. We can now cast the problem as a standard classification problem in the domain of graphs. Given G^{tr} we build a train and test set as the corresponding joint neighborhood subgraphs as detailed in Section 4.2.1. We can now compute a Gram matrix of the instances and solve the classification task using for example the efficient LinearSVC library [2].

4.2.3 Emperical Evaluation

In this section, we carry out two separate expermental settings to evaluate the performance of JNSL and compare it with various link prediction methods which only exploit topological features.

In the first setting, we follow the experimental setting and procedure performed in [59] in which 6 datasets belonging to different domains are employed.

- *Protein* [88]: nodes are proteins and edges encodes a thresholded interaction confidence between proteins. It has 2617 nodes and 11855 links with an average degree of 9.1.

- *Metabolic* [93]: nodes are enzyme and metabolites, edges are present if the enzyme catalyzes for a reaction that include those chemical compounds. It has 668 nodes and 2782 links with an average degree of 8.3.

- *Nips* [4]: nodes are authors at the NIPS conference from the first to the 12^{th} edition. Links encode the co-authorhip relation, i.e. if two authors have published a paper together. This network contains 2865 nodes and 4733 links with an average degree of 3.3.

- *Condmat* [52]: nodes are scientists working in condensed matter physics, edges encode co-authorship. This network has 14230 nodes and 1196 links with an average degree of 0.17.

- *Conflict* [32, 90]: nodes are countries and edges encode a conflict or a dispute. We have 130 nodes and 180 links in total in this network with an average degree of 2.5.

- *Powergrid* [41]: a network of electric powergrid in US. It has 4941 nodes and 6594 links. The average degree is 2.7.

We evaluate the performance of employed methods by splitting 10 times the data in a train and a test part. For *Protein*, *Metabolic*, *Nips* and *Conflict* networks, we use 10% of the edges to induce the training set while for *Condmat* and *Powergrid* we use 90% of the links. The performance of each method is computed as the average of the AUC-ROC over the 10 rounds.

In the second setting, we aims at applying diffusion-based graph node kernels LEDK, MEDK, MDK, RLK (see 2.5.2) for link prediction and comparing their performance with JNSL. We test on three datasets: BioGridgen, Omim and HPRD presented in 2.7. Similar to the first setting, we evaluate the performance of employed methods by splitting 10 times the data in a training (90%) and a testing part (10%). The performance of each method is computed as the average of the AUC-ROC over the 10 rounds.

Model Selection: The values of di

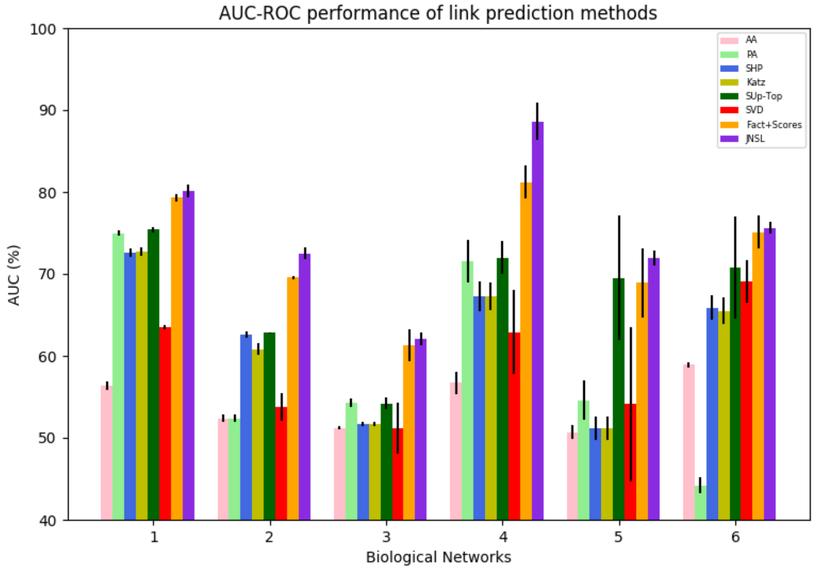

Figure 4.2: The performance of link prediction methods on 6 datasets. Legend: AA: Adamic-Adar, PA [11] preferential Attachment, SHP: Shortest Path, Sup-Top [59]: Liear regression running on unsupervised scores, SVD [59]: Singular value decomposition, Fact+Scores [59]: Factorization with unsupervised scores.

fferent hyper-parameters are set by using a 3-fold on the training set, that is, always considering only the training network, we use one fold for fitting the parameters and the rest two folds for validating the effect of the hyper parameter choice. We tune the values of radius for extracting subgraphs R in $\{1, 2\}$, λ in node label function in $\{10, 15\}$, for r and d parameters of NSPDK in $\{1, 2\}$ and $\{1, 2, 3\}$, respectively. The values for diffusion parameter β in LEDK and MEDK are sampled in $\{10^{-3}, 10^{-3}, 10^{-2}, 10^{-1}\}$, time steps t in MDK in $\{1, 10, 100\}$ and RLK parameter β in $\{1, 4, 7\}$. Finally, the regularization tradeoff C for the SVM is picked up in $\{10^{-4}, 10^{-3}, 10^{-2},\ 10^{-1}, 1,\ 10,\ 10^{2}, 10^{3},\ 10^{4}\}$.

4.2.4 Results and discussion

Figure 4.4 and 4.3 intuitively show the performance in AUC-ROC of different link prediction methods in the first and second settting, respectively. For the more detail of the results, we suggest to see in the Table B.2 and B.1.

Concerning the Figure 4.4, we can group methods into two groups based on their performances: supervised methods and unsupervised methods. The performance of supervised methods are considerably higher than unsuperivsed ones in most cases, except in the Conflict dataset where Sup-Top outperforms Fact+Scores, but with a very small difference. Concerning supervised methods, JNSL outperforms Fact-Scores in all cases. The difference between their performance is small in PowerGrid and Protein datasets with 0.5% and 0.8%, respectively. And the big gap is in the Condmat dataset with 7.4%.

In the Figure 4.3, it can be clearly seen that JNSL outperforms the compared methods that use diffusion-based kernels which are unsupervised methods. In particular, our method gets 10.2% 4.4% and 13.7% higher comparing to the second best performance method in BioGridgen, Omim and HPRD, respectively. Related to diffusion-based prediction methods, MDK and RLK present better results in all cases comparing to LEDK and MEDK.

4.2.5 Conclusion and Future Work

We have presented a novel approach to link prediction in absence of side information that can effectively exploit the topological contextual information available in the neighborhood of each edge. We have empirically shown that this approach achieves very competitive results compared to other state-of-the-art methods.

In future work, we will first investigate how to make use of multiple and heterogeneous information sources when these are available for nodes and edges. We then apply link prediction methods in general and JNSL in specific to link enrichment with the aim of improving performance of learning systems when dealing with sparse graphs.

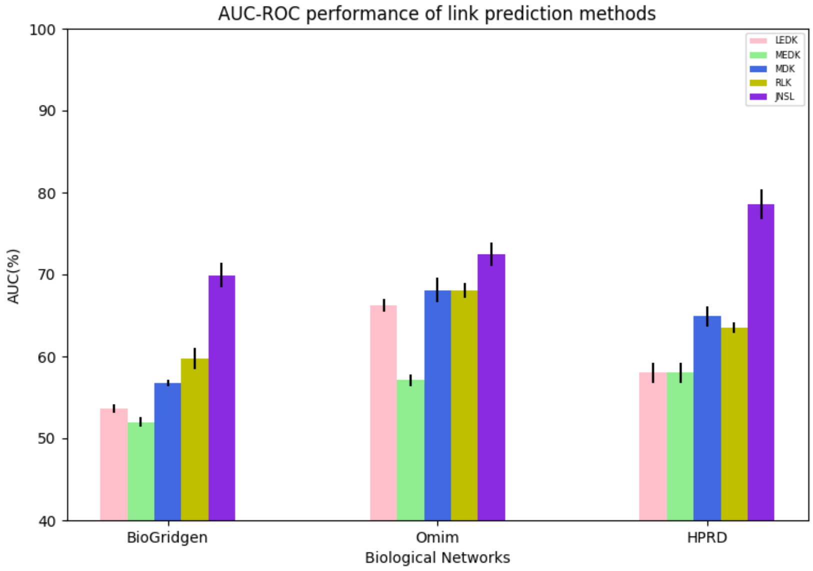

Figure 4.3: The performance link prediction methods on 3 datasets

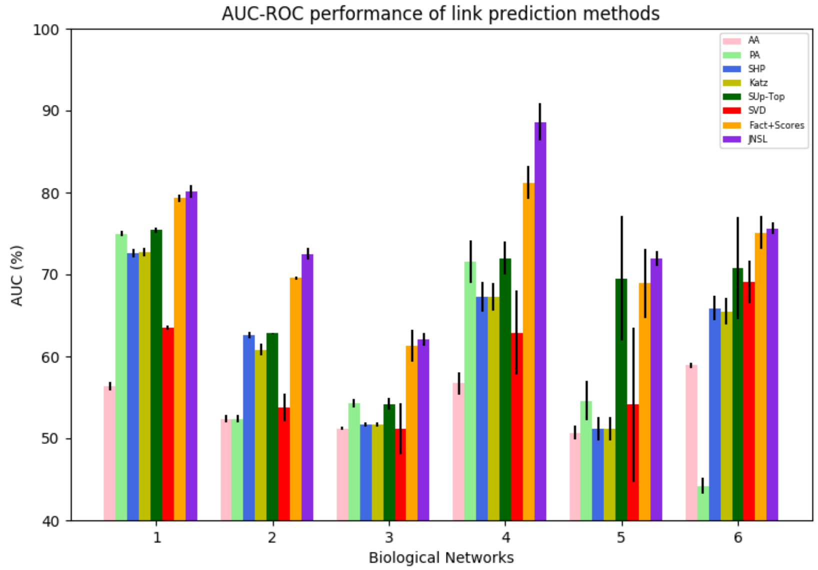

Figure 4.4: The performance of link prediction methods on 6 datasets. Legend: AA: Adamic-Adar, PA [11] preferential Attachment, SHP: Shortest Path, Sup-Top [59]: Liear regression running on unsupervised scores, SVD [59]: Singular value decomposition, Fact+Scores [59]: Factorization with unsupervised scores.

4.3 Link Enrichment for Diffusion-based Graph Node Kernels

It is worth to highlight that a powerful approach to process large heterogeneous sources of data is to use graph encodings [39, 69, 56, 38] and then use graph-based learning systems. In these systems the notion of node similarity is key. A common approach is to resort to graph node kernels in which diffusion-based graph node kernels are commonly used [78, 62, 51, 21, 49]. The diffusion-based graph node kernels measure the proximity between any pair of nodes by taking into account the paths that connect them. However, when the graph structure is affected by noise in the form of missing links, node similarities are distorted proportionally to the sparsity of the graph and to the fraction of missing links. Two of the main reasons for this are that *i*) the lower the average node degree is, the smaller the number of paths through which information can travel, and *ii*) missing links can end up separating a graph into multiple disconnected components. In this case, since information cannot travel across disconnected components, the similarity of nodes belonging to different components is null. To address these problems we propose to solve a link prediction task prior to the node similarity computation and start studying the question: *how can we improve node similarity using link prediction?* In this section, we take into account both the link prediction literature (presented in 2.8) and the diffusion-based graph node kernel literature (described in 2.5.2), select a subset of approaches in both categories that seem well suited, focus on a set of node predicting problems in the bioinformatics domain and empirically investigate the effectiveness of the combination of these approaches on the given predictive tasks. The encouraging result that we find is that all the strategies for link prediction we examined consistently enhance the performance on downstream predictive tasks, often significantly improving state of the art results.

4.3.1 Method

Often the relational information that defines the graph structure is incomplete because certain relations are not known at a given time or have not been yet investigated. When this happens the resulting graphs tend to become sparse and composed of several disconnected components. Diffusion-based kernels are not suited in these cases and so they show a degraded predictive capacity. Our key idea is to introduce a *link enrichment phase* that can address both issues and enhance the performance of diffusion-based systems.

Given a link prediction algorithm M, a diffusion-based graph node kernel K and a sparse graph $G = (V, E, \mathbf{L}_1, \mathbf{L}_2)$ in which $|V| = n$ and $|E| = m$, with $m \approx n$ the link enrichment method consists of two phases:

- enrichment: the link prediction algorithm M is used to score all possible $\frac{n(n-1)}{2} - m$ missing links. A score associated to a link represents for its likelihood to be a missing link. The top scoring t links are added to E to obtain E' that defines the new graph $G' = (V, E')$.
- kernel computation: the diffusion-based graph node kernel K is applied to graph $G^{'}$ to compute the kernel matrix K' which captures the similarities between any couple of nodes, possibly belonging to different components in the graph G.

The kernel matrix K' can be used directly by a kernelized learning algorithm, such as a support vector machine, to make predictive inferences.

4.3.2 Empirical evaluation

To empirically study the answer to the question: *how can we improve node similarity using link prediction?* we would need to define a taxonomy of prediction problems on graphs that make use of the notion of node similarity and analyze which link prediction strategies can be effectively coupled with specific node similarity computation techniques for each given class of problems. In addition we should also study the quantitative relation between the degree of missingness and the size of the improvement offered by prepending the link prediction to the node similarity assessment. In this paper we start such endeavor restricting the type of predictive problems tothat of node ranking in the sub-domain of gene-disease association studies with a fixed but unknown degree of missingness given by the current medical knowledge. More in detail, the task, known as *gene prioritization*, consists in ranking candidate genes based on their probabilities to be related to a disease on the basis of a given a set of genes experimentally known to be associated to the disease of interest. We have studied the proposed approach on the following 4 datasets: **BioGPS**, **HPRD**, **Phenotype similarity** and **Biogridphys** which we described in 2.7.

4.3.3 Evaluation Method

To evaluate the performance of the diffusion kernels, we follow the experimetal setting from [21] and used in 3.4. However, we employ 14 disease-gene associations with at least 30 confirmed genes (see the Table B.3). For each disease, we construct a positive set $\mathcal{P}$ with all confirmed disease genes. To build the negative set $\mathcal{N}$, we randomly sample a set of genes that are associated at least to one disease class, but not related to the class which defines the positive set such that $|\mathcal{N}| = \frac{1}{2}|\mathcal{P}|$. We replicate this procedure 5 times such that the positive set is held constant, while the negative set varies. We assess the performance of kernels via a 3-fold CV, where, after partitioning the dataset $\mathcal{P} \cup \mathcal{U}$ in 3 folds, we use one fold for training model using SVM and the two remaining folds for testing. For each test gene g_i, the model returns a score s_i proportional to the likelihood of being associated to the disease. Next a decision score q_i is computed as the top percentage value of s_i among all candidate gene scores. We collect all decision scores for every test genes to compute the area under the curve for the receiver operating characteristic (AUC-ROC). The final performance on the disease class is obtained by taking average over 3 folds × 5 trials.

Model Selection: The hyper-parameters of the various methods are set using a 3-fold on training set in which one fold is used for training the model and two remaining folds are used for validation. We try the values for LEDK and MEDK in $\{0.01, 0.05, 0.1\}$, time steps in MDK in $\{3, 5, 10\}$ and RLK parameter in $\{0.01, 0.1, 1\}$. For CDNK, we try for the degree threshold value in $\{10, 15, 20\}$, clique size threshold in $\{4, 5\}$, maximum radius in $\{1, 2\}$, maximum distance in $\{2, 3, 4\}$. The number of links used for the enrichment are chosen in $\{40\%, 50\%, 60\%, 70\%\}$ of the number of existing links. Finally, the regularization tradeoff C for the SVM is chosen in $\{10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, 1, 10, 10^2, 10^3,\ 10^4\}$.

4.4 Results and Discussion

The Figure 4.5, 4.6, 4.7 and 4.8 show the average performance of the system on BioGPS, Biogridphys, HPRD and Omim, respectively. Note that each resource yields a graph with different characteristic sparsity and number of components. We compare the average AUC-ROC scores in two cases: plain diffusion kernel (in green color) and diffusion kernel on a modified graph G' (in red color) which includes the novel edges identified by a link prediction system. Here we report the aggregated results (a detailed breakdown is available in Table B.4 - B.12 in Appendices).

The noteworthy result is how consistent the result is: each link prediction method improves each diffusion kernel algorithm, and on average using link prediction yields a 15% to 20% relative error reduction for diffusion-based methods. What varies is the amount of improvement, which depends on the coupling between the four elements: the disease, the information source, the link prediction method and the diffusion kernel algorithm. In specific we obtain that the largest improvement is obtained for disease 3 (connective) where we have a maximum improvement of 20% ROC points, while the minimum improvement is for disease 8 (immunological) with a minimal improvement of 0% ROC points (see in the detailed report). On average the largest improvement is of 13% ROC points, while the smallest improvement is on average of 1% ROC point. Such stable results are of interest since diffusion-based methods are currently state-of-the-art for gene-disease prioritization tasks, and hence a technique that can offer a consistent and relatively large improvement can have important practical consequences in the understanding of disease mechanisms.

4.5 Conclusion and Future Work

In this section, we have proposed the notion of *link enrichment* for diffusion-based graph node kernels, that is, the idea of carrying out the computation of information diffusion on a graph that contains edges identified by link prediction approaches. We have discovered a surprisingly robust signal that indicates that diffusion-based node kernels consistently benefit from the coupling with similarity-based link prediction techniques on large scale datasets in biological domains.

In future work we will carry out a more fine grained analysis, defining a taxonomy of prediction problems on graphs that make use of the notion of node similarity and analyze which link prediction strategies can be effectively coupled with specific node similarity computation techniques for a given problem class. In addition we will study the quantitative relation between the degree of missingness and the size of the improvement offered by prepending the link prediction to the node similarity assessment. Finally, we will extend the analysis to the more complex case of kernel integration and data fusion, i.e. when multiple heterogeneous information sources are used jointly to define the predictive task.

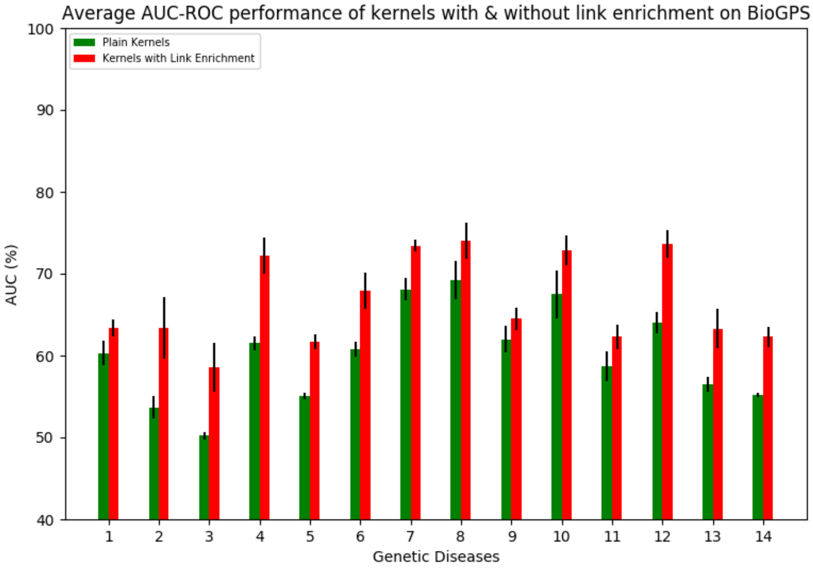

Figure 4.5: The AUC-ROC performance using kernels with and without using link enrichment on BioGPS

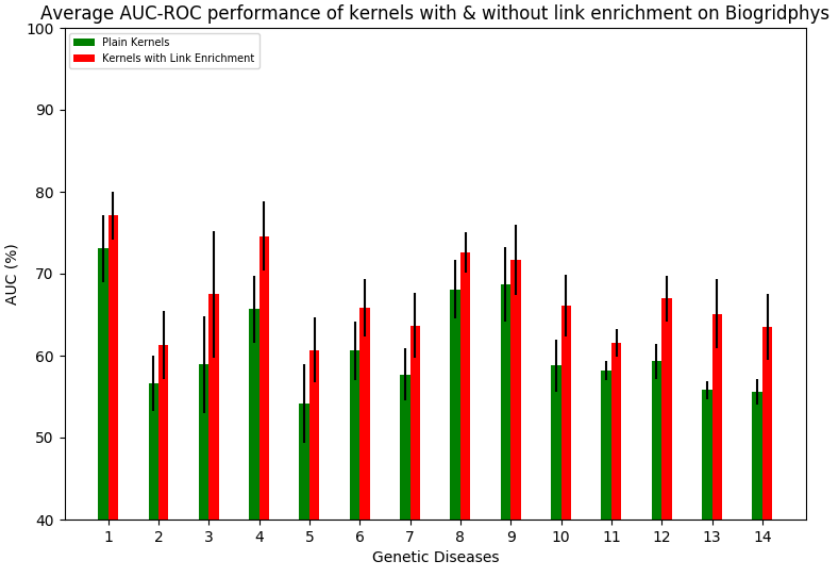

Figure 4.6: The AUC-ROC performance using kernels with and without using link enrichment on Biogridphys

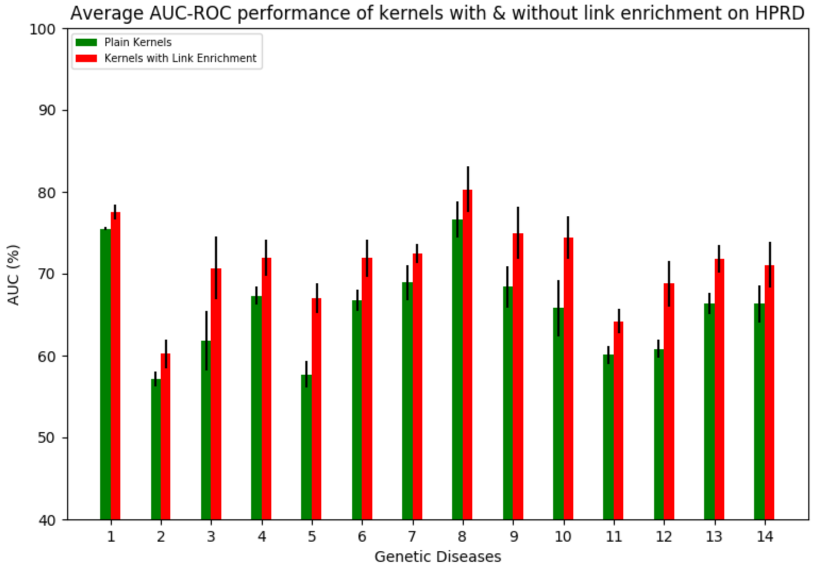

Figure 4.7: The AUC-ROC performance using kernels with and without using link enrichment on HPRD

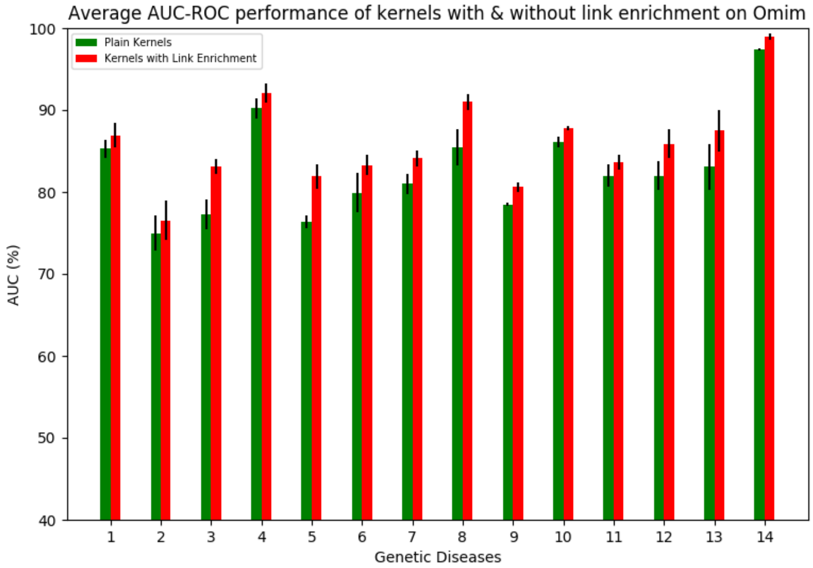

Figure 4.8: The AUC-ROC performance using kernels with and without using link enrichment on Omim

Chapter 5

Graph-based Data Integration Approaches

As discussed in the Chapter 1, there is a high need of proposing graph-based integration methods which allow to build high performance graph-based predictive systems. The task of designing such methods is not trivial due to its requirements. In particular, a good graph-based integration should possess the following features. First, it needs to effectively exploit the complement property of graph combination. Indeed, each graph encodes an aspect of a considered phenomenon. Therefore, combining graphs together would bring a more unified view of that phenomenon which is useful for leanring systems. Second, it is required to be scalable to deal with the constant increase of data sources resulting from different researches in which each research views the interest phenomenon from a certain angle. Third, the mothod should be efficient, i.e the algorithm is required to execute in a reasonable time and memory consumption. Last, the performance of graph-based systems using graph integration are expected to be better than the systems which only use a single graph.

In this chapter, we propose a general kernel-based framework for graph-based biological data integration, named Graph-one, that meets the above requirements. In what follows, we first show the flow of Graph-one, followed by description for each step of the flow. We then evaluate Graph-one by introducing different variations of Graph-one created under the general framework solving disease gene prioritization problem.

5.1 Graph-one: A General Kernel-based Framework for Graph-based Data Integration

In this section, we describe Graph-one, a kernel-based general framework for graph-based data integraiton.

5.1.1 Graph-one Flow

The Figure 5.1.2 shows the flow of Graph-one which aims at finding an effective and efficient kernel that represents for a set of data sources. Precisely, our framework cosists of four main steps: graph construction, graph layer combination, kernel definition and kernel learning. Given a set of of data sources that encode information of a certain phenomenon, $S = \{S_1, S_2, \ldots, S_n\}$. Following, we explain each step in detail.

- Graph construction: for each data source $S_i \in S$, we build an undirected, unweighted, labeled graph G_i. As a consequence, we obtain a set of graphs $\{G_1, G_2, \ldots, G_n\}$. We consider this graph set as a single graph G whose every single graph as its layer, $G = \{L_1, L_2, \ldots, L_n\}$ in which L_i is corresponding to L_i.

- Graph layer combination: we allow different graph layers to be *combined* in specific ways. This is due to the fact that combining graph layers would enrich information with respect to a single one and it can be useful for learning process. For example, sparse graph layers can be combined to get a denser layer which allows diffusion-based kernels to perform better. As a consequence of the combining process, we obtain a graph $G' = \{L_1, L_2, \ldots, L_m\}$ such that $m \leq n$.

- Kernel definition: we apply suitable graph node kernels to each graph layer $L_i \in G'$ (see the section 2.5.2) with various selected parameter values. Thus, we obtain a set of kernel matrices in which each matrix encodes a different node similarity measure.

- Kernel learning: we employ UEasyMKL (see 5.1.2), a scalable MKL algorithm that is able to effficiently deal with unbalanced settings, to get an optimal positive linear combination of predefined computed kernels in a data driven way. As a result, we achieve a kernel which is likely to be better than every base kernel.

The final kernel can be fed into any kernel machine to build a model for different tasks of interest: classification, regression, etc.

Graph-one can be considered as a member of early data integration class since its target is to find a kernel that represents the abstraction of a given data sources. It is worth to state that Graph-one meet the requirements for graph-based data integration described above. First, it allows to define various kernels from each graph layer in which each kernel would capture the node similarity in a specific way. The obtained kernels from defined on all graph layers are combined into a single one. Therefore, the final kernel encodes the unified view of information from all layers. Second, Graph-one is scalable and efficient framework due to the properties of the adopted multiple kernel learning, UEasyMKL. UEasyMKL allows to deal with a high number of defined kernels to combine in a constant memory consumption and linear time computation w.r.t the number of kernels. Interestingly, Graph-one is desinged to handle with unbalanced settings are problematic for many learning algorithms. Moreover, the graph layer combination step in the framework lets users to decide which graph layers should be combined. This is important since the decision of combination depends on the constructed graphs and the problem context. The performance of the proposed framework is evaluated through disease gene prioritization presented in the section 5.2.

5.1.2 Scalable Unbalanced Multiple Kernel Learning: UEasyMKL

In 2.10, we introduce EasyMKL, a scalable, efficient kernel integration approach. However, in many applications, we often observe the strong imbalance between the number of positive and negative examples, making it more difficult to build high performance learning systems. Disease gene prioritization is a typical example where given a genetic disease, the number of positive genes known to be associated to the disease is much smaller than the number of the unknown genes. For this reason, inspired by a previous work proposed in [68], and we adapt EasyMKL to propose a novel MKL algorithm that we name as UEasyMKL. UEasyMKL is not only scalable and efficient, as it inherits from EasyMKL, but also able to deal with unbalanced settings.

In order to clearly present our method, we first need to highlight the different contributions given by positive and unlabelled examples. Therefore, we define $\mathbf{K}^+$, $\mathbf{K}^-$ and $\mathbf{K}^{+-}$ the sub-matrices of $\mathbf{K}^s$ pertaining to positive-positive, unlabelled-unlabelled and positive-unlabelled example pairs, respectively. Schematically, we have:

$$\mathbf{K}^s = \begin{pmatrix} \mathbf{K}^+ & \mathbf{K}^{+-} \\ \mathbf{K}^{-+} & \mathbf{K}^- \end{pmatrix},$$

where $\mathbf{K}^{-+}$ the transpose of $\mathbf{K}^{+-}$. In other words, $\mathbf{K}^+$ contains similarities among positive examples, $\mathbf{K}^-$ contains similarities among unlabelled examples and $\mathbf{K}^{+-}$ includes similarities between positive-unlabelled example pairs. In the same way, we define γ_+ and γ_- as the probability vectors associated to positive and unlabelled examples, respectively.

Under this change of variables, we reformulate the problem as:

$$\min_{\gamma\in\Gamma} \gamma_+^\top \mathbf{K}^+ \gamma_+ \; - \; 2\,\gamma_+^\top \mathbf{K}^{+-} \gamma_- \; + \; \gamma_-^\top \mathbf{K}^- \gamma_- \; + \; \lambda_+ \gamma_+^\top \gamma_+ \; + \; \lambda_- \gamma_-^\top \gamma_- \, .$$

In this new formulation, the original EasyMKL problem is obtained by setting $\lambda_+ = \lambda_- = \frac{\lambda}{1-\lambda}$. However, due to the unbalanced PU nature of the problem, we are interested in using two different regularizations among positive and unlabelled examples. In our case, we decide to fix *a priori* the regularization parameter $\lambda_- = +\infty$, corresponding to fixing $\lambda = 1$ over unlabelled examples only. Then, the solution of part of the objective function is defined by the uniform distribution $\gamma_- = (\frac{1}{n}, \frac{1}{n}, \dots \frac{1}{n}) \equiv u$, where n is the number of unlabelled examples.

We inject this analytic solution of part of the problem in our objective function as

$$\min_{\gamma\in\Gamma^+} \gamma_+^\top \mathbf{K}^+ \gamma_+ \; - \; 2\,\gamma_+^\top \mathbf{K}^{+-} u \; + \; u^\top \mathbf{K}^- u \; + \; \lambda_+ \gamma_+^\top \gamma_+ \; + \; \lambda_- u^\top u,$$

where $\Gamma^+ = \{\gamma \in \mathcal{R}_+^m | \sum_{i:y_i=1} \gamma_i = 1, \gamma_j = 1/n \, \forall j : y_j = -1\}$ is the probability distribution domain where the distributions over the unlabelled examples correspond to the uniform distribution. It is trivial that $u^\top \mathbf{K}^- u$ and $\lambda_- u^\top u$ are independent from the γ_+ variable. Then, they can be removed from the objective function obtaining

$$\min_{\gamma\in\Gamma^+} \gamma_+^\top \mathbf{K}^+ \gamma_+ - 2\,\gamma_+^\top \mathbf{K}^{+-} u + \lambda_+ \gamma_+^\top \gamma_+. \tag{5.1}$$

In this expression, we only need to consider the entries of the kernel $\mathbf{K}^s$ concerning the positive set, avoiding all the entries with indices in the unlabelled set. The complexity becomes quadratic in the number of positive examples m, which is always much smaller than the number of examples to prioritize. Moreover, this algorithm still depends linearly on the number of kernels R and the overall time complexity is then $\mathcal{O}(m^2 \cdot R)$. In this way, we greatly simplify the optimization problem, while being able to take into account the diverse amount of noise present in positive and unlabelled example sets.

Like in the previous section, after solving the problem of Eq. 5.1 we use Eq. 2.27 to compute the optimal kernel weights. Next, we solve again the Scuba optimization problem to get the final optimal probability distribution γ^*. The likelihood to be positive of every test example is given by the vector of scores s defined as

$$s = \mathbf{K}^* \mathbf{Y} \gamma^*, \tag{5.2}$$

where $\mathbf{K}^*$ is the final kernel matrix, computed by $\mathbf{K}^* = \sum_r^R \eta_r^* \mathbf{K}_r$. We apply the formula 5.3 to compute scores for the test examples as:

$$s = \mathbf{K}^{t*} \mathbf{Y} \gamma^*\,, \tag{5.3}$$

where $\mathbf{K}^{t*}$ is the final test kernel matrix, obtained by taking the weighted sum over all test kernel matrices:

$$\mathbf{K}^{t*} = \textstyle\sum_r \eta_r^* \mathbf{K}_r^t,$$

$\mathbf{K}^{t*}, \mathbf{K}_r^t$s are matrices of size (p, q) with p, q are the number of test examples and training examples, respectively. More particularly, a score for s_i of the test example g_i is computed as:

$$s_i = \textstyle\sum_j y_j \gamma_j^* \mathbf{K}_{ij}^{t*}.$$

In words, s_i is the weighted sum over the weighted similarities between the test example g_i and all examples in the training set g_js. Once we get the scores for test examples, candidate examples are then prioritized based on the score values.

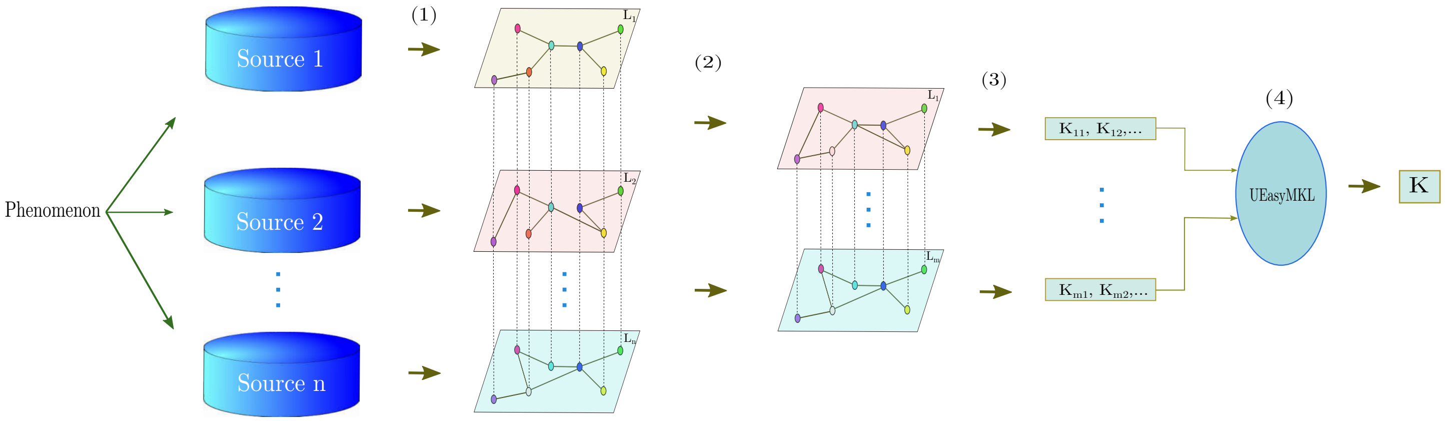

Figure 5.1: Graph-one general framework flow for graph-based data integration where (1) Graph construction, (2) Layer combination , (3) Kernel definition, (4) Kernel learning

5.2 Graph-one for Disease Gene Prioritization

In this section, we evaluate the performance of Graph-one through the disease gene prioritization, one of the main tasks in Biomedicine as discussed in 2.6. Since Graph-one is a general framework which allow the user to define their preference in some steps. Different choices leads to different variations of Graph-one. Here, we propose three variations for disease gene prioritization: Scuba, PLC and DIGI. These variations are evaluated and compared with various methods by using two separate experimental settings: Cross-validation and Unbiased evaluation. Through performing these experiments, we would like to have answers for following questions.

- Does Graph-one show better performance comparing to other methods for disease gene prioritization?
- Does Graph-one with graph integration show better results in comparison with the use of single graph?
- Is Graph-one stable in term of performance between its different variations?

5.2.1 Graph Variations for Disease Gene Prioritization

Graph-one: Scuba

We introduce Scuba as an variation of Graph-one. Consider a n-layer graph $G = G_1, G_2, \ldots, G_n$ where $G_i \in G$ is derived from the source $S_i \in S$. In this method, kernels are defined on every single graph layer, i.e, different kernels with different parameter values are applied on each $G_i \in \mathcal{G}$. As a consequence, we get a set of kernel matrices $\mathcal{K}_i = \{K_{i1}, K_{i2}, \ldots, K_{iH}\}$ for each layer G_i. By collecting all kernels from all $\mathcal{K}_i$, we achieve a final kernel matrix set $\mathcal{K} = \bigcup_{i \in I}^{n} \mathcal{K}_i$ comprising $L \cdot H$ matrices. Next, all matrices in $\mathcal{K}$ and the training are fed into UEasyMKL to obtain the final kernel K. In this way, we directly use UEasyMKL to perform an automatic selection of optimal kernel parameters. As the method is an instance of Graph-one, it inherits the strength of the general graph integration framework Graph-one.

We apply Scuba for experiments in 5.2.4 with three diffusion-based kernels. Each kernel is used to define three kernel matrices from each graph layer. The value for λ_+ is chosen by employing k-fold cross validation on the training set, using the the grid of values $\{0,\ 0.1,\ 0.2,\ \ldots, 1\}$. Kernel parameter values are set as follows: $\{0.01,\ 0.04,\ 0.07\}$ for MEDK, as suggested in [23] and used in [21], $\{2,\ 4,\ 6\}$ for MDK and $\{1,\ 10,\ 100\}$ for RLK, as suggested in [29]. Besides, we use Scuba in two cases: using plain diffusion-based kernels (Scuba1), using diffusion-based kernels with association with link enrichment (Scuba2).

5.2.2 Graph-one: DIGI

We consider a graph including n layers $G = \{G_1, G_2, \ldots, G_n\}$ where $G_i \in G$ is derived from $S_i \in S$. In this method, we first process and combine all graph layers in a specific way. We then apply CDNK, which is defined in Chapter 3, with different parameter value tuples on the obtained graph to get a set of kernel matrices. These kernel matrices are then fed into UEasyMKL to get a final kernel matrix. More precisely, the method consists of the following steps:

- Graph decomposition: for each graph layer $G_i \in G$, we apply the network decomposition procedure proposed in 3.2.1 to have a graph composed of linked sparse connected components.

- Graph combination: consider a layer tuple G_i, G_j $(G_i, G_j \in G)$ and two nodes $u \in G_i$, $v \in G_j$ such that u and v are both represent for a same entity, we connect u and v by a "disjuctive" link. As the consequence, we achieve a graph $G^{'}$ whose nodes represeting the same entities are linked by disjuctive links. Figure 5.2 is a visualization for this graph integration idea.

- Similarity definition: we employ CDNK to define similarity for the similarity for any couple of genes. The similarity between two genes g_i and g_j is the summation of similarity of between their corresponding nodes on all layers.

For experiments in 5.2.4, we use values for parameters of graph decomposition procedure and CDNK the same as experiments in 3.4

5.2.3 Graph-one: PLC

Starting from The idea of PLC is similar to Scuba. The only difference is that PLC combines some graph layers together in the same way with DIGI.

We propose a Graph-one variation named PLC to apply the second

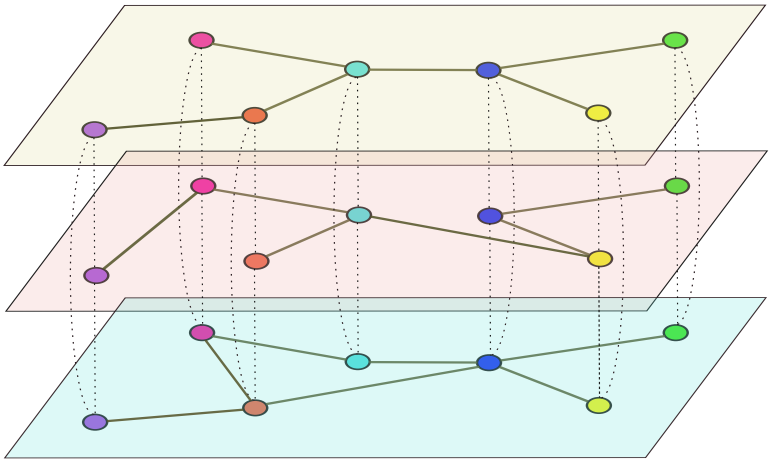

Figure 5.2: Graph Disjunctive Interconnection Illustration

5.2.4 Experiments and Results

Cross-validation

As a first evaluation of Graph-one, we followed the experimental protocol used by Chen *et al* to test predictive performance of other prioritization methods [20]. In this setting, we employed three data sets: BioGPS, HPRD and Pathways (see the section 2.7), which we borrowed from the authors of the work. To perform the experiments, we employed known gene-disease associations from OMIM, grouped into 20 classes on the basis of disease relatedness by Goh *et al* [34]. Among those classes we selected the 12 with at least 30 confirmed genes. We then built a training set consisting of a positive set $\mathcal{P}$ and an unlabelled set $\mathcal{U}$ for each of them. $\mathcal{P}$ contains all its disease gene members. $\mathcal{U}$ is constructed by randomly picking genes from known disease genes such that $|\mathcal{U}| = \frac{1}{2}|\mathcal{P}|$. The unlabelled genes relate to at least one disease class, but do not relate to the current class. We chose the genes in $\mathcal{U}$ from the other disease genes because we assume that they were less likely to be associated to the considered class. In fact, disease genes are generally more studied and a potential association has more chances to have already been identified.

After that, leave-one-out cross validation was used to evaluate the performance of the algorithm. Iteratively, every gene in the training set was selected to be the test gene and the remaining genes in $\mathcal{P}$ and $\mathcal{U}$ were used to train the model. Once the model was trained, a score list for the test gene and all genes associated to no disease was computed. Then, we computed a decision score for each test gene representing the percentage of candidate genes ranked lower than it. We collected all decision scores for every gene in all disease classes to form a global decision score list. The performance of Scuba was measured by calculating the area under the curve (AUC) in the receiver-operating-characteristic plot obtained from the decision score list. The AUC expresses the probability that a randomly chosen disease gene is ranked above a randomly picked non-disease gene for any disease class.

In this setting, we compare not only the performance between Graph-one variations, but also the performance between Graph-one with other methods for disease gene prioritization, including F3C3 [20], MRF [21], DIR [23], GeneWanderer [45] and Avg, an implementation that takes an average of kernels defined on graph layers as the final representation for datasets).

The Figure 5.3 shows the performance of Graph-one variations (Scuba1, Scuba2, PLC and DIGI) along with other methods, while the Figure 5.4 presents the performance of Scuba in two cases: using single graph and graph integration. It can be seen from the Table 5.3 that all variations of Graph-one perform significantly better than the other methods (also see the Table C.1), getting an AUC from 3.8% and 4.9% greater than the F3PC and Avg, respectively. Among Graph-one variations, DIGI and Scuba2 show slightly better than Scuba1 and PLC. It illustrates the stability of performance between Graph-one variations. Besides, the effectiveness of Graph-one for data integration is shown in the Table 5.4. Graph-one with data integration presents higher performance in most of diseases (11 out of 12) w.r.t its that uses single data source.

Unbiased evaluation

Although the previous evaluation is useful to compare Graph-one with other methods, predictive performance in cross-validation experiments may be inflated compared to real applications. Indeed, the retrieval of known disease genes can be facilitated by various means. One mean is the crosstalk between data repositories: for example, KEGG [66] draws its information also from medical literature. Moreover, often the discovery of the link between a gene and a disease coincides with the discovery of a functional annotation or of a molecular interaction. In practice, instead, researchers are interested in novel associations, which in most cases are harder to find due to a lack of information around them.

To achieve a thorough evaluation of Graph-one, we test it in a more realistic setting, following the work of Börnigen *et al* [14]. In this study, eight gene prioritization web tools were benchmarked as follows. Newly discovered gene-disease associations were collected over a timespan of six months, gathering 42 test genes associated to a range of disorders. As soon as a new association was discovered, each web tool was queried with a disorder-specific set of positive genes $\mathcal{P}$ to prioritize a set of candidates $\mathcal{U}$ containing the corresponding test gene (or to prioritize the whole genome where possible). In other words, the test gene was treated as unlabelled to simulate the re-discovery of its association with the disease. Rank positions of the 42 test genes were ultimately used to assess the ability of the tools to successfully prioritize disease genes. The idea behind this procedure is to anticipate the integration of the associations in the data sources and so avoid biased predictions.

In order to test Graph-one in this setting, we backdate our data to a time prior to May 15, 2010 by employing String v8.2 data [42] and Omim (see the section 2.7). After that, we recovere positive sets and test genes from the original publication and we followe its experimental protocol as follows [14]. We performe prioritizations for each test gene in two distinct cases: genome-wide and candidate set-based prioritizations. In any genome-wide prioritization all genes in the String dataset - except those in $\mathcal{P}$ - belong to $\mathcal{U}$ and were prioritized. In any candidate set-based prioritization, the set of candidates $\mathcal{U}$ was constructed by considering all genes with Ensembl [1] gene identifier within the chromosomal regions around the test gene, in order to get on average 100 candidates. In both cases, we normalize the ranking positions over the total number of considered genes in order to get median, mean and standard deviation of the normalize ranks for test genes. We also compute the true positive rate (TPR) relatively to some representative thresholds (5%, 10% and 30% of the ranking) and the AUC obtained by averaging over the 42 prioritizations.

Along with Graph-one, we evaluated in this setting also MKL1class [95] and ProDiGe [62], two state-of-the-art kernel based gene prioritization methods. In Table 5.1, it is possible to see performances for all three methods. The significance of rank median differences between Scuba and competing methods was assessed by Wilcoxon signed rank tests, one for each comparison. At a significance threshold of 0.05, Scuba achieves significantly higher performances in genome-wide tasks compared to both baselines. In the candidate set-based setting, it performs significantly better than ProDiGe and better, although not significantly, than MKL1class. These differences can be visually appreciated in Figure 5.5, where we compare the rank distributions of test genes obtained by the three methods. Scuba and MKL1class present moderate rank differences, particularly in the central region of the ranks. On the other hand, differences between Scuba and ProDiGe are smaller (Pearson $r = 0.98$ in both cases) and almost all in favour of Scuba.

In Table 5.2 we show results for Graph-one compared to the results obtained in the work of Börnigen *et al*, pertaining to eight prioritization systems [14]. In genome-wide predictions, Scuba dominates over the other tools. On predictions over smaller candidate sets, it is still competitive although best results are achieved by GeneDistiller [73], Endeavour [7] and ToppGene [22]. It is important to underline that in this case considered tools rely on different data sources, so we are comparing different prioritization systems rather different algorithms. Furthermore, tools are in some cases unable to provide an answer to a given task, depending on the underlying data sources (for more details see the original work [14]). We report the fraction of prioritizations on which tools are actually evaluated as response rate. This table has the purpose of showing the potentiality of Scuba relatively to what is easily accessible by non-bioinformaticians. However, since we used the String data for instance Graph-one:Scuba1 is directly comparable with Pinta [14, 65].

Next, we expanded this validation by employing gene-disease annotations derived from the Human Phenotype Ontology (HPO) [46]. This resource gathers information from several databases and makes available its monthly updates, permitting to trace the annotations history. We downloaded the HPO build 29 - dating March 2013 - and build 117 of February 2017. We compared the two annotations corresponding to these versions of HPO and extracted the gene-disease associations that were added in this time gap. We concentrated on the multifactorial diseases covered in the previous analysis, that could possibly have some previously undiscovered associations. We thus analyzed how the obtained genes are ranked in genome-wide prioritizations of the previous analysis, applying the same performance measures as before. The outcome is an analogous evaluation, but this time target genes are those extracted from HPO.

In Table 5.3 results for Scuba, MKL1class and ProDiGe are shown. We can observe a slightly different trend compared to previous results, with Scuba and ProDiGe having very close performance and MKL1class being significantly worse than Scuba. As a confirmation, in Figure 5.5 we can see that there is no clear difference between test genes rank distributions for Scuba and ProDiGe. Instead, MKL1class ranks several test genes neatly lower compared to Scuba, with the associated Pearson correlation coefficient dropping to $r = 0.85$.

5.2.5 Discussion

Gene prioritization is progressively becoming essential in molecular biology studies. In fact, we are assisting to a continuous proliferation of a variety of *omic* data brought by technological advances. In the near future it is then likely that more heterogeneous knowledge will have to be combined. Moreover, the classes of biological agents to be prioritized are going to enlarge. For instance, we are only beginning to understand the complex regulation machinery involving non-coding RNA and epigenetic agents. It is estimated that around 90.000 human long non coding genes exist, whose functional implications are progressively emerging [97]. Facing this challenge, the development of novel methods is still strongly needed in order to enhance predictive power and efficiency.

Compared to the considered benchmark kernel methods - MKL1class and ProDiGe - Scuba has some important advantages. ProDiGe is one of the first proposed kernel-based PU learning method for gene prioritization [62]. It implements a PU learning strategy based on a biased SVM, which over-weights positive examples during training. In order to reach scalability to large datasets, it leverages a bagging procedure. Like ProDiGe, Scuba implements a learning strategy based on a binary classification set up, but from a different perspective. In a PU problem, the information on positive labels is assumed secure, while the information on negative labels is not. In terms of margin optimization, this translates in unbalanced entropy on the probability distributions associated to the two sets of training examples. It is then required to regularize more on the unlabelled class - having higher entropy - and in the limit of maximum uncertainty we get the uniform distribution.

MKL1class implements another effective approach for data integration, namely single class learning. This means that the model is obtained solely based on the distribution of known disease genes, disregarding unlabelled ones. Scuba has enhanced scalability compared to MKL1class, as it involves the optimization of the 1-norm of the margin vector from the different kernels. In contrast, MKL1class optimizes its 2-norm, which is more computationally demanding. Importantly, another distinctive feature of Scuba is a time complexity dependent on the number of positive examples and not on the number of total examples. As a consequence, Scuba can exploit the information on the whole data distribution and at the same time scale to large datasets without the need of sub-sampling the examples. This may be of great advantage as typically disease genes are orders of magnitude less numerous than the candidates.

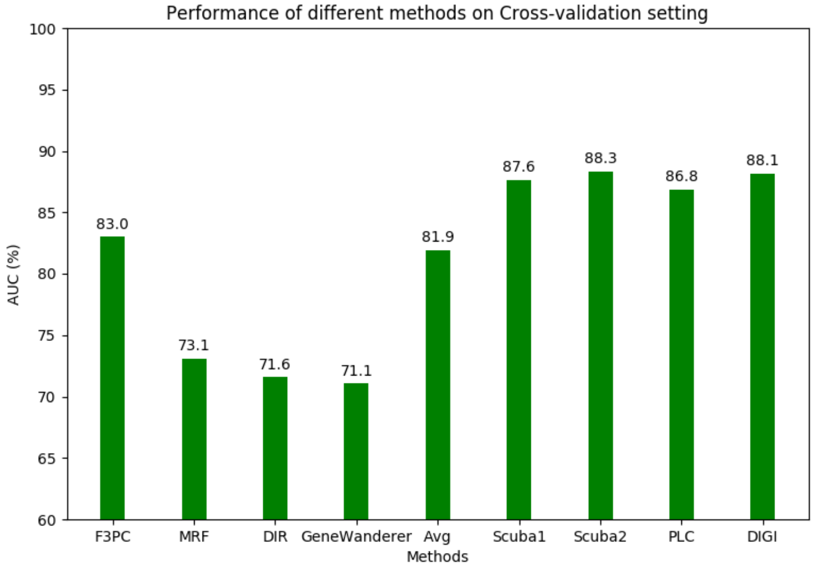

Figure 5.3: The performance of different techniques in the experimental setting of Chen *et al* [20] expressed in terms of AUC. Except for our Graph-one's variations, and the average kernels (Avg), these results were taken from that work.

Results from two different evaluation settings show that our proposed method Scuba outperforms many existing methods, particularly in genome-wide analyses. Compared to the two considered existing kernel-based methods, Scuba performances (considering AUC) are always higher, and often significantly higher. Moreover, Scuba has two main levels of scalability that make it particularly suitable for gene prioritization

Altogether, our results show that Scuba is a valuable tool to achieve efficient prioritizations, especially in large-scale investigations.

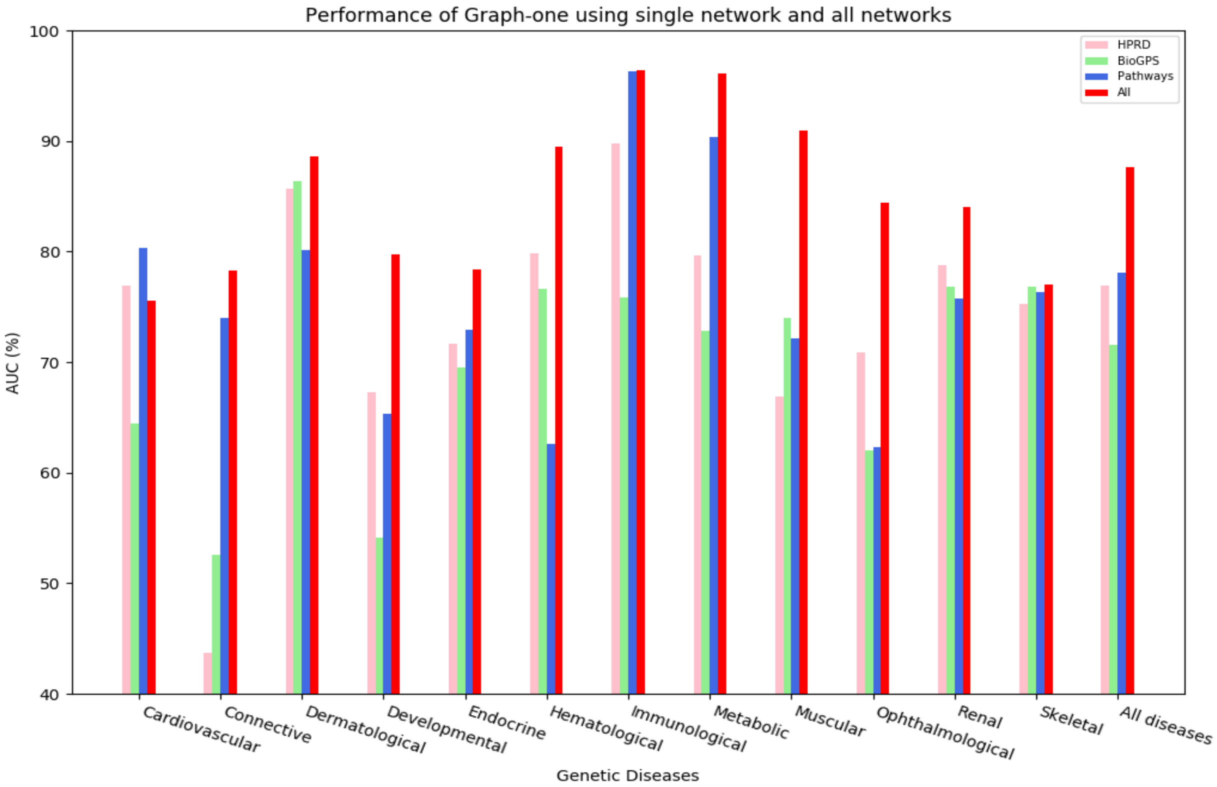

Figure 5.4: The performance of Graph-one (Scuba1): using single layer and combine all layers

Table 5.1: Performances of Graph-one, MKL1class and ProDiGe in the unbiased setting of Börnigen *et al* [14]. Values refer to predictions on all the 42 gene-disease associations. Rank difference p-values were obtained using Wilcoxon signed rank tests comparing separately Scuba1/MKL1class and Scuba1/ProDiGe ranks differences. Asterisks indicate significance of the tests at a threshold of 0.05.

Tool/Method	Rank median	Rank average	TPR in top 5% (%)	TPR in top 10% (%)	TPR in top 30% (%)	AUC	Rank difference p-value
Genome-wide prioritization methods							
Scuba1	**10.55**	**20.48**±23.53	**33.3**	**47.6**	**78.6**	**0.80**	-
MKL1class [95]	13.30	23.42±23.23	21.4	**47.6**	69.0	0.77	$2.5 \cdot 10^{-2}$ *
ProDiGe [62]	11.73	24.45±27.33	31.0	45.2	71.4	0.76	$3.0 \cdot 10^{-7}$ *
Candidate set-based prioritization methods							
Scuba1	**12.95**	**23.32**±25.46	**28.6**	**45.2**	**73.8**	**0.78**	-
MKL1class [95]	15.07	25.63±24.73	23.8	40.5	61.9	0.76	$9.7 \cdot 10^{-2}$
ProDiGe [62]	14.41	26.39±29.09	26.2	40.5	71.4	0.75	$2.7 \cdot 10^{-3}$ *

Table 5.2: Performances of Graph-one (Scuba1) and of some gene prioritization web tools in the unbiased setting of Börnigen *et al* [14]. Response rate is the percentage of gene-disease associations considered by each tool. Values for Suspects were computed on the first 27 associations only (highlighted by [a]).

Tool/Method	Response rate (%)	Rank median	Rank average	TPR in top 5% (%)	TPR in top 10% (%)	TPR in top 30% (%)	AUC
Genome-wide prioritization methods							
Scuba1	100	**10.55**	**20.48**±23.53	**33.3**	**47.6**	**78.6**	**0.80**
DIGI	100	**4.73**	**12.72**±18.20	**52.4**	**59.5**	**85.7**	**0.87**
Candid [40]	100	18.10	27.35±24.62	21.4	33.3	64.3	0.73
Endeavour [7]	100	15.49	21.47±22.37	28.6	38.1	71.4	0.79
Pinta [65]	100	19.03	23.52±23.58	26.2	31.0	71.4	0.77
Candidate set-based prioritization methods							
Scuba1	100	12.95	23.32±25.46	28.6	45.2	73.8	0.78
DIGI	100	6.10	13.70±17.98	50.0	59.5	88.1	0.86
Suspects [6]	88.9[a]	12.77[a]	24.64±26.42[a]	33.3[a]	33.3[a]	63.0[a]	0.76[a]
ToppGene [22]	97.6	16.80	34.53±35.31	**35.7**	42.9	52.4	0.66
GeneWanderer-RW [45]	88.1	22.10	29.55±26.28	16.7	26.2	61.9	0.71
Posmed-KS [94]	47.6	31.44	42.07±30.98	4.7	7.1	23.8	0.58
GeneDistiller [73]	97.6	**11.11**	**15.37**±13.77	26.2	**47.6**	78.6	**0.85**
Endeavour [7]	100	11.16	18.41±21.39	26.2	42.9	**90.5**	0.82
Pinta [65]	100	18.87	25.23±24.72	28.6	31.0	71.4	0.75

Table 5.3: Performances of Graph-one, MKL1class and ProDiGe in the expanded unbiased setting involving seven multifactorial diseases. Values refer to predictions on 48 gene-disease associations. Rank difference p-values were obtained using Wilcoxon signed rank tests comparing separately Scuba/MKL1class and Scuba/ProDiGe ranks differences. Asterisks indicate significance of the tests at a threshold of 0.05.

Method	Rank median	Rank average	TPR in top 1% (%)	TPR in top 5% (%)	TPR in top 10% (%)	TPR in top 30% (%)	AUC	Rank difference p-value
Genome-wide prioritizations								
Scuba1	8.13	**17.45**±22.33	**10.4**	41.7	**58.3**	**79.2**	**0.83**	-
MKL1class [95]	14.28	25.79±26.96	2.1	27.1	45.8	66.7	0.74	$1.2 \cdot 10^{-5}$ *
ProDiGe [62]	**7.89**	18.40±23.77	**10.4**	**43.8**	54.2	**79.2**	0.82	$9.5 \cdot 10^{-2}$

5.3 Conclusion and Future Work

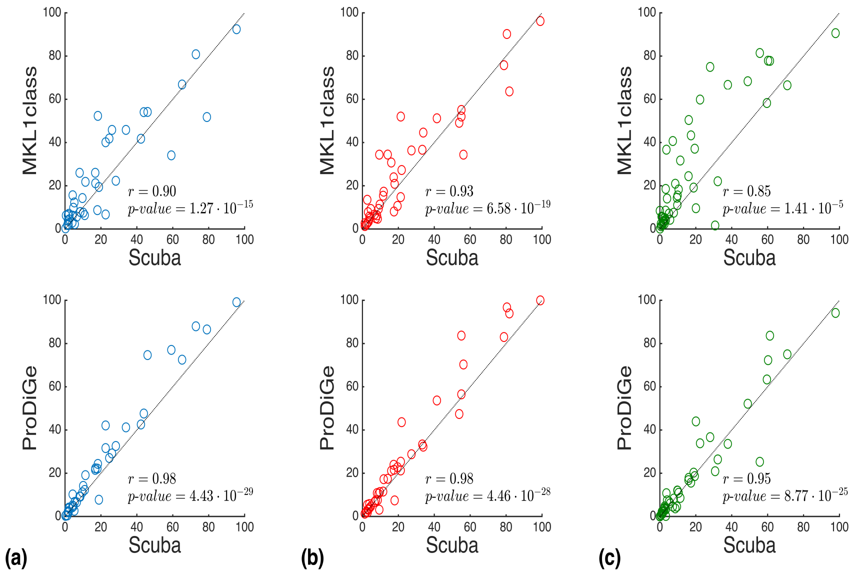

Figure 5.5: Comparison of normalized ranks predicted by Scuba and competing kernel methods. Normalized test genes rank distributions predicted by Scuba, MKL1class and ProDiGe for test genes in **(a)** genome-wide prioritizations in the unbiased evaluation of Table 5.1 - **(b)** candidate set-based prioritizations in the unbiased evaluation of the Table 5.1 - **(c)** genome-wide prioritizations in the expanded unbiased evaluation of the Table 5.3. In all cases, each point represents a test gene and lower values on the axes indicate better predictions. Genes lying on a diagonal have the same rank according to both methods considered on a plot. The further a gene lies above (below) a diagonal and the better it was ranked by Scuba (MKL1class/ProDiGe) compared to MKL1class/ProDiGe (Scuba). In each plot we show the Pearson correlation coefficient r between the test genes rank distributions and its associated p-value.

Chapter 6

Conclusion and Future Work

Bibliography

[1] Ensembl, http://www.ensembl.org/.

[2] http://www.scikit-learn.org/.

[3] Online mendelian inheritance in man, http://omim.org/.

[4] Rowies, s.: Nips dataset, http://www.cs.nyu.edu/rwoeis/data.html.

[5] Lada A Adamic and Eytan Adar. Friends and neighbors on the web. *Social networks*, 25(3):211–230, 2003.

[6] Euan A Adie, Richard R Adams, Kathryn L Evans, David J Porteous, and Ben S Pickard. Suspects: enabling fast and effective prioritization of positional candidates. *Bioinformatics*, 22(6):773–774, 2006.

[7] Stein Aerts, Diether Lambrechts, Sunit Maity, Peter Van Loo, Bert Coessens, Frederik De Smet, Leon-Charles Tranchevent, Bart De Moor, Peter Marynen, Bassem Hassan, et al. Gene prioritization through genomic data fusion. *Nature biotechnology*, 24(5):537–544, 2006.

[8] Fabio Aiolli, Giovanni Da San Martino, and Alessandro Sperduti. A kernel method for the optimization of the margin distribution. *Artificial Neural Networks-ICANN 2008*, pages 305–314, 2008.

[9] Fabio Aiolli and Michele Donini. Easymkl: a scalable multiple kernel learning algorithm. *Neurocomputing*, 169:215–224, 2015.

[10] José Ignacio Alvarez-Hamelin, Luca Dall'Asta, Alain Barrat, and Alessandro Vespignani. k-core decomposition: A tool for the visualization of large scale networks. *arXiv preprint cs/0504107*, 2005.

[11] Albert-László Barabási and Réka Albert. Emergence of scaling in random networks. *science*, 286(5439):509–512, 1999.

[12] Karsten M Borgwardt and Hans-Peter Kriegel. Shortest-path kernels on graphs. In *Data Mining, Fifth IEEE International Conference on*, pages 8–pp. IEEE, 2005.

[13] 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.

[14] Daniela Börnigen, Léon-Charles Tranchevent, Francisco Bonachela-Capdevila, Koenraad Devriendt, Bart De Moor, Patrick De Causmaecker, and Yves Moreau. An unbiased evaluation of gene prioritization tools. *Bioinformatics*, 28(23):3081–3088, 2012.

[15] 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.

[16] David Botstein and Neil Risch. Discovering genotypes underlying human phenotypes: past successes for mendelian disease, future approaches for complex disease. *Nature genetics*, 33(3s):228, 2003.

[17] Olivier Chapelle, Bernhard Scholkopf, and Alexander Zien. Semi-supervised learning (chapelle, o. et al., eds.; 2006)[book reviews]. *IEEE Transactions on Neural Networks*, 20(3):542–542, 2009.

[18] Andrew Chatr-Aryamontri, Bobby-Joe Breitkreutz, Rose Oughtred, Lorrie Boucher, Sven Heinicke, Daici Chen, Chris Stark, Ashton Breitkreutz, Nadine Kolas, Lara O'donnell, et al. The biogrid interaction database: 2015 update. *Nucleic acids research*, 43(D1):D470–D478, 2014.

[19] Pavel Chebotarev and Elena Shamis. The matrix-forest theorem and measuring relations in small social groups. *arXiv preprint math/0602070*, 2006.

[20] Bolin Chen, Min Li, Jianxin Wang, Xuequn Shang, and Fang-Xiang Wu. A fast and high performance multiple data integration algorithm for identifying human disease genes. *BMC medical genomics*, 8(3):S2, 2015.

[21] BoLin Chen, Min Li, JianXin Wang, and Fang-Xiang Wu. Disease gene identification by using graph kernels and markov random fields. *Science China. Life Sciences*, 57(11):1054, 2014.

[22] Jing Chen, Huan Xu, Bruce J Aronow, and Anil G Jegga. Improved human disease candidate gene prioritization using mouse phenotype. *BMC bioinformatics*, 8(1):392, 2007.

[23] Yixuan Chen, Wenhui Wang, Yingyao Zhou, Robert Shields, Sumit K Chanda, Robert C Elston, and Jing Li. In silico gene prioritization by integrating multiple data sources. *PloS one*, 6(6):e21137, 2011.

[24] Gene Ontology Consortium et al. The gene ontology (go) database and informatics resource. *Nucleic acids research*, 32(suppl 1):D258–D261, 2004.

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

[26] Fabrizio Costa and Kurt De Grave. Fast neighborhood subgraph pairwise distance kernel. In *Proceedings of the 26th International Conference on Machine Learning*, pages 255–262. Omnipress, 2010.

[27] Tijl De Bie, Léon-Charles Tranchevent, Liesbeth MM Van Oeffelen, and Yves Moreau. Kernel-based data fusion for gene prioritization. *Bioinformatics.*

[28] François Fouss, Kevin Francoisse, Luh Yen, Alain Pirotte, and Marco Saerens. An experimental investigation of kernels on graphs for collaborative recommendation and semisupervised classification. *Neural networks*, 31:53–72, 2012.

[29] Francois Fouss, Luh Yen, Alain Pirotte, and Marco Saerens. An experimental investigation of graph kernels on a collaborative recommendation task. In *Data Mining, 2006. ICDM'06. Sixth International Conference on*, pages 863–868. IEEE, 2006.

[30] Thomas Gärtner. A survey of kernels for structured data. *ACM SIGKDD Explorations Newsletter*, 5(1):49–58, 2003.

[31] Olivier Gevaert, Frank De Smet, Dirk Timmerman, Yves Moreau, and Bart De Moor. Predicting the prognosis of breast cancer by integrating clinical and microarray data with bayesian networks. *Bioinformatics*, 22(14):e184–e190, 2006.

[32] Faten Ghosn, Glenn Palmer, and Stuart A Bremer. The mid3 data set, 1993–2001: Procedures, coding rules, and description. *Conflict Management and Peace Science*, 21(2):133–154, 2004.

[33] Vladimir Gligorijević and Nataša Pržulj. Methods for biological data integration: perspectives and challenges. *Journal of the Royal Society Interface*, 12(112):20150571, 2015.

[34] Kwang-Il Goh, Michael E Cusick, David Valle, Barton Childs, Marc Vidal, and Albert-László Barabási. The human disease network. *Proceedings of the National Academy of Sciences*, 104(21):8685–8690, 2007.

[35] Mehmet Gönen and Ethem Alpaydın. Multiple kernel learning algorithms. *Journal of machine learning research*, 12(Jul):2211–2268, 2011.

[36] James A Hanley and Barbara J McNeil. The meaning and use of the area under a receiver operating characteristic (roc) curve. *Radiology*, 143(1):29–36, 1982.

[37] David Haussler. Convolution kernels on discrete structures. Technical report, Technical report, Department of Computer Science, University of California at Santa Cruz, 1999.

[38] Lawrence B Holder, Diane J Cook, et al. Learning patterns in the dynamics of biological networks. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 977–986. ACM, 2009.

[39] Zan Huang, Wingyan Chung, Thian-Huat Ong, and Hsinchun Chen. A graph-based recommender system for digital library. In *Proceedings of the 2nd ACM/IEEE-CS joint conference on Digital libraries*, pages 65–73. ACM, 2002.

[40] Janna E Hutz, Aldi T Kraja, Howard L McLeod, and Michael A Province. Candid: a flexible method for prioritizing candidate genes for complex human traits. *Genetic epidemiology*, 32(8):779–790, 2008.

[41] Koki Ichinose, Jihoon Park, Yuji Kawai, Junichi Suzuki, Minoru Asada, and Hiroki Mori. Local over-connectivity reduces the complexity of neural activity: Toward a constructive understanding of brain networks in patients with autism spectrum disorder.

[42] Lars J Jensen, Michael Kuhn, Manuel Stark, Samuel Chaffron, Chris Creevey, Jean Muller, Tobias Doerks, Philippe Julien, Alexander Roth,

Milan Simonovic, et al. String 8a global view on proteins and their functional interactions in 630 organisms. *Nucleic acids research*, 37(suppl_1):D412–D416, 2008.

[43] Leo Katz. A new status index derived from sociometric analysis. *Psychometrika*, 18(1):39–43, 1953.

[44] TS Keshava Prasad, Renu Goel, Kumaran Kandasamy, Shivakumar Keerthikumar, Sameer Kumar, Suresh Mathivanan, Deepthi Telikicherla, Rajesh Raju, Beema Shafreen, Abhilash Venugopal, et al. Human protein reference database2009 update. *Nucleic acids research*, 37(suppl_1):D767–D772, 2008.

[45] Sebastian Köhler, Sebastian Bauer, Denise Horn, and Peter N Robinson. Walking the interactome for prioritization of candidate disease genes. *The American Journal of Human Genetics*, 82(4):949–958, 2008.

[46] Sebastian Köhler, Nicole A Vasilevsky, Mark Engelstad, Erin Foster, Julie McMurry, Ségolène Aymé, Gareth Baynam, Susan M Bello, Cornelius F Boerkoel, Kym M Boycott, et al. The human phenotype ontology in 2017. *Nucleic acids research*, 45(D1):D865–D876, 2017.

[47] Risi Imre Kondor and John Lafferty. Diffusion kernels on graphs and other discrete input spaces. In *ICML*, volume 2, pages 315–322, 2002.

[48] Gert RG Lanckriet, Tijl De Bie, Nello Cristianini, Michael I Jordan, and William Stafford Noble. A statistical framework for genomic data fusion. *Bioinformatics*, 20(16):2626–2635, 2004.

[49] Hyunju Lee, Zhidong Tu, Minghua Deng, Fengzhu Sun, and Ting Chen. Diffusion kernel-based logistic regression models for protein function prediction. *Omics: a journal of integrative biology*, 10(1):40–55, 2006.

[50] Elizabeth A Leicht, Petter Holme, and Mark EJ Newman. Vertex similarity in networks. *Physical Review E*, 73(2):026120, 2006.

[51] Xin Li and Hsinchun Chen. Recommendation as link prediction in bipartite graphs: A graph kernel-based machine learning approach. *Decision Support Systems*, 54(2):880–890, 2013.

[52] Ryan N Lichtenwalter, Jake T Lussier, and Nitesh V Chawla. New perspectives and methods in link prediction. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 243–252. ACM, 2010.

[53] Weiping Liu and Linyuan Lü. Link prediction based on local random walk. *EPL (Europhysics Letters)*, 89(5):58007, 2010.

[54] Linyuan Lü, Ci-Hang Jin, and Tao Zhou. Similarity index based on local paths for link prediction of complex networks. *Physical Review E*, 80(4):046122, 2009.

[55] Linyuan Lü and Tao Zhou. Link prediction in complex networks: A survey. *Physica A: statistical mechanics and its applications*, 390(6):1150–1170, 2011.

[56] Jose Lugo-Martinez and Predrag Radivojac. Classification in biological networks with hypergraphlet kernels. *arXiv preprint arXiv:1703.04823*, 2017.

[57] Edward M Marcotte, Matteo Pellegrini, Ho-Leung Ng, Danny W Rice, Todd O Yeates, and David Eisenberg. Detecting protein function and protein-protein interactions from genome sequences. *Science*, 285(5428):751–753, 1999.

[58] Victor A McKusick. Mendelian inheritance in man and its online version, omim. *The American Journal of Human Genetics*, 80(4):588–604, 2007.

[59] Aditya Krishna Menon and Charles Elkan. Link prediction via matrix factorization. In *Joint european conference on machine learning and knowledge discovery in databases*, pages 437–452. Springer, 2011.

[60] Kurt Miller, Michael I Jordan, and Thomas L Griffiths. Nonparametric latent feature models for link prediction. In *Advances in neural information processing systems*, pages 1276–1284, 2009.

[61] Tom MITCHELL and McGraw HILL. *Machine Learning.* McGraw-Hill Science/Engineering/Math, 1997.

[62] Fantine Mordelet and Jean-Philippe Vert. Prodige: Prioritization of disease genes with multitask machine learning from positive and unlabeled examples. *BMC bioinformatics*, 12(1):389, 2011.

[63] Yves Moreau and Léon-Charles Tranchevent. Computational tools for prioritizing candidate genes: boosting disease gene discovery. *Nature reviews. Genetics*, 13(8):523, 2012.

[64] Ralf Mrowka, Wolfram Liebermeister, and Dirk Holste. Does mapping reveal correlation between gene expression and protein–protein interaction? *Nature genetics*, 33(1):15–16, 2003.

[65] Daniela Nitsch, Joana P Gonçalves, Fabian Ojeda, Bart De Moor, and Yves Moreau. Candidate gene prioritization by network analysis of differential expression using machine learning approaches. *BMC bioinformatics*, 11(1):460, 2010.

[66] Hiroyuki Ogata, Susumu Goto, Kazushige Sato, Wataru Fujibuchi, Hidemasa Bono, and Minoru Kanehisa. Kegg: Kyoto encyclopedia of genes and genomes. *Nucleic acids research*, 27(1):29–34, 1999.

[67] Paul Pavlidis, Jason Weston, Jinsong Cai, and William Stafford Noble. Learning gene functional classifications from multiple data types. *Journal of computational biology*, 9(2):401–411, 2002.

[68] Mirko Polato and Fabio Aiolli. Kernel based collaborative filtering for very large scale top-n item recommendation. In *Proceedings of the European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning (ESANN)*, pages 11–16, 2016.

[69] Emad Ramadan, Sadiq Alinsaif, and Md Rafiul Hassan. Network topology measures for identifying disease-gene association in breast cancer. *BMC bioinformatics*, 17(7):274, 2016.

[70] Marylyn D Ritchie, Emily R Holzinger, Ruowang Li, Sarah A Pendergrass, and Dokyoon Kim. Methods of integrating data to uncover genotype-phenotype interactions. *Nature reviews. Genetics*, 16(2):85, 2015.

[71] David Salgado, Matthew I Bellgard, Jean-Pierre Desvignes, and Christophe Béroud. How to identify pathogenic mutations among all those variations: variant annotation and filtration in the genome sequencing era. *Human mutation*, 2016.

[72] Carl F Schaefer, Kira Anthony, Shiva Krupa, Jeffrey Buchoff, Matthew Day, Timo Hannay, and Kenneth H Buetow. Pid: the pathway interaction database. *Nucleic acids research*, 37(suppl_1):D674–D679, 2008.

[73] Dominik Seelow, Jana Marie Schwarz, and Markus Schuelke. Genedistillerdistilling candidate genes from linkage intervals. *PLoS One*, 3(12):e3874, 2008.

[74] John Shawe-Taylor and Nello Cristianini. *Kernel methods for pattern analysis*. Cambridge university press, 2004.

[75] Nino Shervashidze and Karsten M Borgwardt. Fast subtree kernels on graphs. In *Advances in neural information processing systems*, pages 1660–1668, 2009.

[76] Nino Shervashidze, Pascal Schweitzer, Erik Jan van Leeuwen, Kurt Mehlhorn, and Karsten M Borgwardt. Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research*, 12(Sep):2539–2561, 2011.

[77] T Strachan and AP Read. Human molecular genetics (new york: Garland science, taylor & francis group). 2011.

[78] Liang Sun, Shuiwang Ji, and Jieping Ye. Adaptive diffusion kernel learning from biological networks for protein function prediction. *BMC bioinformatics*, 9(1):162, 2008.

[79] Robert E Tarjan. Decomposition by clique separators. *Discrete mathematics*, 55(2):221–232, 1985.

[80] Marc A Van Driel, Jorn Bruggeman, Gert Vriend, Han G Brunner, and Jack AM Leunissen. A text-mining analysis of the human phenome. *European journal of human genetics: EJHG*, 14(5):535, 2006.

[81] Martin H Van Vliet, Hugo M Horlings, Marc J Van De Vijver, Marcel JT Reinders, and Lodewyk FA Wessels. Integration of clinical and gene expression data has a synergetic effect on predicting breast cancer outcome. *PloS one*, 7(7):e40358, 2012.

[82] Vladimir Vapnik. Pattern recognition using generalized portrait method. *Automation and remote control*, 24:774–780, 1963.

[83] Vladimir Naumovich Vapnik and Vlamimir Vapnik. *Statistical learning theory*, volume 1. Wiley New York, 1998.

[84] Imre Vastrik, Peter D'Eustachio, Esther Schmidt, Geeta Joshi-Tope, Gopal Gopinath, David Croft, Bernard de Bono, Marc Gillespie, Bijay Jassal, Suzanna Lewis, et al. Reactome: a knowledge base of biologic pathways and processes. *Genome biology*, 8(3):R39, 2007.

[85] Jean-Philippe Vert, Koji Tsuda, and Bernhard Schölkopf. A primer on kernel methods. *Kernel Methods in Computational Biology*, pages 35–70, 2004.

[86] S Vichy N Vishwanathan, Nicol N Schraudolph, Risi Kondor, and Karsten M Borgwardt. Graph kernels. *Journal of Machine Learning Research*, 11(Apr):1201–1242, 2010.

[87] SVN Vishwanathan, Karsten M Borgwardt, and Nicol N Schraudolph. Fast computation of graph kernels. In *Proceedings of the 19th International Conference on Neural Information Processing Systems*, pages 1449–1456. MIT Press, 2006.

[88] Christian Von Mering, Roland Krause, Berend Snel, Michael Cornell, Stephen G Oliver, Stanley Fields, and Peer Bork. Comparative assessment of large-scale data sets of protein–protein interactions. *Nature*, 417(6887):399–403, 2002.

[89] Xuefeng Wang, Eric P Xing, and Daniel J Schaid. Kernel methods for large-scale genomic data analysis. *Briefings in bioinformatics*, 16(2):183–192, 2014.

[90] Michael D Ward, Randolph M Siverson, and Xun Cao. Disputes, democracies, and dependencies: A reexamination of the kantian peace. *American Journal of Political Science*, 51(3):583–601, 2007.

[91] Michelle Whirl-Carrillo, EM McDonagh, JM Hebert, Li Gong, K Sangkuhl, CF Thorn, RB Altman, and Teri E Klein. Pharmacogenomics knowledge for personalized medicine. *Clinical Pharmacology & Therapeutics*, 92(4):414–417, 2012.

[92] Chunlei Wu, Camilo Orozco, Jason Boyer, Marc Leglise, James Goodale, Serge Batalov, Christopher L Hodge, James Haase, Jeff Janes, Jon W Huss, et al. Biogps: an extensible and customizable portal for querying and organizing gene annotation resources. *Genome biology*, 10(11):R130, 2009.

[93] Yoshihiro Yamanishi, Jean-Philippe Vert, and Minoru Kanehisa. Supervised enzyme network inference from the integration of genomic data and chemical information. *Bioinformatics*, 21(suppl_1):i468–i477, 2005.

[94] Yuko Yoshida, Yuko Makita, Naohiko Heida, Satomi Asano, Akihiro Matsushima, Manabu Ishii, Yoshiki Mochizuki, Hiroshi Masuya, Shigeharu Wakana, Norio Kobayashi, et al. Posmed (positional medline): prioritizing genes with an artificial neural network comprising medical documents to accelerate positional cloning. *Nucleic acids research*, 37(suppl_2):W147–W152, 2009.

[95] Shi Yu, Tillmann Falck, Anneleen Daemen, Leon-Charles Tranchevent, Johan AK Suykens, Bart De Moor, and Yves Moreau. L2-norm multiple kernel learning and its application to biomedical data fusion. *BMC bioinformatics*, 11(1):309, 2010.

[96] Pooya Zakeri, Sarah Elshal, and Yves Moreau. Gene prioritization through geometric-inspired kernel data fusion. In *Bioinformatics and Biomedicine (BIBM), 2015 IEEE International Conference on*, pages 1559–1565. IEEE, 2015.

[97] Yi Zhao, Hui Li, Shuangsang Fang, Yue Kang, Yajing Hao, Ziyang Li, Dechao Bu, Ninghui Sun, Michael Q Zhang, Runsheng Chen, et al. Noncode 2016: an informative and valuable data source of long noncoding rnas. *Nucleic acids research*, 44(D1):D203–D208, 2016.

[98] Marinka Žitnik and Blaž Zupan. Data fusion by matrix factorization. *IEEE transactions on pattern analysis and machine intelligence*, 37(1):41–53, 2015.

Appendices

Chapter A

Conjunctive Disjunctive Graph Node Kernel

Table A.1: Indices and gene number of genetic disease classes

Index	Disease	
0	Connective	35
1	Cardiovascular	75
2	Dermatological	54
3	Developmental	37
4	Endocrine	62
5	Hematological	106
6	Immunological	94
7	Metabolic	159
8	Muscular	55
9	Ophthalmological	61
10	Renal	42
11	Skeletal	35

Table A.2: *Predictive performance on 11 gene-disease associations in percentage using network induced by the BioGPS. Best results in bold. We report the AUC ROC and the rank for each kernel method. CDNK11 = ontology for discrete labels, CDNK12 = ontology for both discrete and vector labels, CDNK21 = uniform discrete labels and CDNK22 = uniform discrete labels and ontology for real vector labels.*

	BioGPS							
Disease	LEDK	MDK	MEDK	RLK	CDNK11	CDNK12	CDNK21	CDNK22
1	51.9/8	57.4/7	59.0/6	59.2/5	65.1/4	69.5/3	72.0/2	**72.3/1**
2	81.7/5	78.5/6	75.2/7	75.0/8	88.3/1	**88.8/1**	83.2/4	84.8/3
3	64.3/6	59.6/8	71.6/3	71.8/2	65.5/5	**72.5/1**	61.4/7	67.9/4
4	65.3/7	58.2/8	67.8/6	67.8/5	71.9/4	**78.7/1**	73.8/3	76.7/2
5	64.0/8	64.1/7	66.5/5	66.2/6	75.9/4	76.2/3	77.8/2	**78.2/1**
6	74.6/5	70.2/8	71.0/7	71.2/6	79.3/3	**83.7/1**	77.2/4	80.0/2
7	73.0/5	66.7/7	75.4/3	75.6/2	68.8/6	73.9/4	66.1/8	**77.4/1**
8	74.4/8	76.8/3	76.2/5	76.4/4	74.7/7	77.7/1	76.1/6	76.9/2
9	71.5/2	65.6/8	67.7/6	69.9/3	66.8/7	**71.7/1**	67.9/5	68.1/4
10	54.0/6	50.3/8	56.1/5	51.1/7	77.6/4	**82.7/1**	77.7/3	78.3/2
11	58.2/7	51.3/8	59.3/6	59.3/5	71.8/4	**80.2/1**	74.8/2	74.0/3
$\overline{AUC}$	66.6	63.5	67.8	67.6	73.2	**77.8**	73.5	75.9
$\overline{Rank}$	6.09	7.09	5.36	4.82	4.45	**1.64**	4.18	2.27

Table A.3: *Predictive performance on 11 gene-disease associations in percentage using network induced by the Pathways. Best results in bold. We report the AUC ROC and the rank for each kernel method. CDNK11 = ontology for discrete labels, CDNK12 = ontology for both discrete and vector labels, CDNK21 = uniform discrete labels and CDNK22 = uniform discrete labels and ontology for real vector labels.*

	Pathways							
Disease	LEDK	MDK	MEDK	RLK	CDNK11	CDNK12	CDNK21	CDNK22
1	74.7/8	76.4/7	78.7/6	78.8/5	80.2/4	82.4/2	**82.8/1**	82.3/3
2	55.1/8	64.9/7	76.6/6	76.6/5	81.1/2	80.3/4	80.1/3	**81.9/1**
3	55.0/8	62.7/7	64.1/5	65.6/4	67.1/3	63.6/6	69.7/2	**71.4/1**
4	54.3/8	65.2/6	**73.7/1**	73.7/2	66.1/5	68.1/3	64.0/7	67.3/4
5	52.9/8	55.7/7	62.7/5	62.7/6	68.3/2	**69.6/1**	66.2/4	68.1/3
6	83.4/8	92.7/7	96.5/2	**96.5/1**	93.0/6	94.1/5	94.5/4	94.9/3
7	84.5/7	88.3/8	89.4/2	**89.5/1**	88.5/6	88.5/5	89.0/3	88.7/4
8	53.7/8	65.6/7	72.0/6	72.3/5	72.5/4	72.5/3	74.9/2	**75.3/1**
9	52.5/8	64.9/5	64.2/7	64.2/6	**81.3/1**	81.0/3	80.1/2	79.8/4
10	68.8/6	65.4/8	74.4/5	74.4/4	66.9/7	76.8/2	75.2/3	**78.7/1**
11	53.7/8	69.2/7	74.6/5	74.1/6	77.0/3	**78.7/1**	75.1/4	77.3/2
$\overline{AUC}$	62.6	70.1	75.2	75.3	76.5	77.8	77.4	**78.7**
$\overline{Rank}$	7.73	6.82	4.55	4.09	3.91	**2.27**	3.18	2.45

Chapter B

Solutions for Graph Sparsity

B.1 Joint Neighborhood Subgraphs Link Prediction

Table B.1: The AUC-ROC performance of link prediction methods on 3 datasets. In bold the highest score.

	Methods				
Datasets	LEDK (%)	MEDK (%)	MDK (%)	RLK (%)	JNSL (%)
BioGridgen	53.6±0.5	52.0±0.6	56.7±0.4	59.7±1.3	**69.9**±**1.5**
Omim	66.2±0.8	57.1±0.7	68.1±1.5	68.1±0.9	**72.5**±**1.4**
HPRD	58.0±1.2	58.0±1.2	64.9±1.2	63.5±0.7	**78.6**±**1.8**

Table B.2: The AUC-ROC performance of link prediction methods on 6 datasets. Legend: AA: Adamic-Adar, PA [11] preferential Attachment, SHP: Shortest Path, Sup-Top [59]: Liear regression running on unsupervised scores, SVD [59]: Singular value decomposition, Fact+Scores [59]: Factorization with unsupervised scores.

	Datasets					
Methods	Protein (%)	Metabolic (%)	Nips (%)	Condmat (%)	Conflict (%)	PowerGrid (%)
AA	56.4±0.5	52.4±0.5	51.2±0.2	56.7±1.4	50.7±0.8	58.9±0.3
PA	75.0±0.3	52.4±0.5	54.3±0.5	71.6±2.6	54.6±2.4	44.2±01.0
SHP	72.6±0.5	62.6±0.4	51.7±0.3	67.3±1.8	51.2±1.4	65.9±1.5
Katz	72.7±0.5	60.8±0.7	51.7±0.3	67.3±1.7	51.2±1.4	65.5±1.6
Sup-Top	75.4±0.3	62.8±0.1	54.2±0.7	72.0±2.0	69.5±7.6	70.8±6.2
SVD	63.5±0.3	53.8±1.7	51.2±3.1	62.9±5.1	54.1±9.4	69.1±2.6
Fact+Scores	79.3±0.5	69.6±0.2	61.3±1.9	81.2±2.0	68.9±4.2	75.1±2.0
JNSL	**80.1±0.8**	**72.5±0.7**	**62.1±0.8**	**88.6±2.3**	**72.0±0.9**	**75.6±0.7**

B.2 Link Enrichment for Diffusion-based Graph Node Kernels

In this section, we present in detail the results obtained from the experiment described in 4.3.2. We report the performance on 14 disease gene classes and four different biological graphs induced by BioGPS, Biogridphys, HPRD and Omim. For each disease class and graph, we show the average AUC(%) for each diffusion-based graph node kernel in two cases: plain diffusion kernel (denoted by ”-” and followed by kernel notation) and diffusion kernel on a modified graph obtained by link enrichment process (denoted by + followed by notation of kernel which is used for link enrichment). We notate each kernel as follow: A: LEDK, B: MEDK, C: MDK, D: RLK, E: CDNK.

Table B.3: Indices of genetic disease classes

Index	Disease
1	Cancer
2	Cardiovascular
3	Connective
4	Dermatological
5	Developmental
6	Endocrine
7	Hematological
8	Immunological
9	Metabolic
10	Muscular
11	Neurological
12	Ophthalmological
13	Renal
14	Skeletal

Table B.4: *Average predictive performance on 14 gene-disease associations using four different graphs induced by the BioGPS, Biogridphys, Hprd and Omim. We report the average AUC-ROC (%) and standard deviations for all difussion-based kernels with (+) and without (-) link enrichment.*

	BioGPS		**Biogridphys**		**Hprd**		**Omim**	
Disease	-	+	-	+	-	+	-	+
1	60.3±1.5	63.4±1.0	73.1±4.1	77.1±2.9	75.5±0.2	77.5±0.9	85.3±1.1	86.9±1.5
2	53.7±1.4	63.4±3.8	56.6±3.4	61.3±4.1	57.1±0.9	60.2±1.8	75.0±2.2	76.5±2.4
3	50.2±0.4	58.6±3.0	58.9±5.9	67.5±7.7	61.8±3.6	70.7±3.8	77.3±1.8	83.1±0.9
4	61.5±0.9	72.2±2.2	65.7±4.1	74.6±4.2	67.3±1.1	71.9±2.2	90.2±1.2	92.1±1.2
5	55.1±0.4	61.7±0.9	54.2±4.8	60.7±4.0	57.7±1.6	67.0±1.8	76.4±0.8	81.9±1.5
6	60.8±0.9	67.9±2.2	60.6±3.6	65.9±3.5	66.8±1.3	71.9±2.3	79.9±2.4	83.3±1.2
7	68.1±1.4	73.4±0.7	57.7±3.2	63.7±4.0	68.9±2.1	72.5±1.2	81.0±1.2	84.1±1.0
8	69.2±2.3	74.0±2.2	68.1±3.6	72.6±2.5	76.6±2.2	80.3±2.8	85.4±2.2	91.0±1.0
9	62.0±1.6	64.5±1.4	68.7±4.6	71.7±4.3	68.4±2.5	75.0±3.2	78.5±0.2	80.6±0.6
10	67.5±2.9	72.9±1.8	58.8±3.2	66.1±3.8	65.8±3.4	74.4±2.6	86.1±0.6	87.8±0.3
11	58.7±1.8	62.3±1.5	58.2±1.2	61.6±1.7	60.1±1.1	64.2±1.5	82.0±1.4	83.6±0.9
12	64.0±1.3	73.6±1.7	59.3±2.1	67.0±2.8	60.8±1.1	68.8±2.8	82.0±1.8	85.9±1.7
13	56.5±0.9	63.3±2.4	55.8±1.1	65.1±4.2	66.4±1.3	71.8±1.7	83.1±2.8	87.5±2.5
14	55.2±0.3	62.3±1.2	55.6±1.6	63.5±4.0	66.3±2.3	71.1±2.8	97.4±0.1	99.0±0.4
$\overline{AUC}$	60.2±0.3	66.7±1.2	60.8±1.6	67.0±4.0	65.7±2.3	71.2±2.8	82.8±0.1	86.0±0.4

Table B.5: *Predictive performance on 14 gene-disease associations using network induced by the BIOGPS*

Disease	-A	+A	+B	+C	+D	+E	-B	+A	+B	+C	+D	+E
1	61.8	63.8	62.5	63.7	63.9	63.4	57.9	62.8	62.8	62.2	62.6	62.2
2	55.6	67.5	62.3	70.4	69.5	63.5	52.0	62.8	57.3	57.9	60.1	58.3
3	50.3	62.3	61.2	63.6	61.9	60.0	49.5	57.0	55.1	56.1	57.2	55.2
4	61.3	73.9	73.1	73.3	73.7	73.9	62.7	72.7	67.9	72.5	74.7	68.5
5	55.7	62.8	60.6	63.2	62.9	62.9	55.1	61.7	61.0	62.9	62.6	60.5
6	60.1	68.8	66.8	70.4	69.4	67.4	59.7	66.1	62.2	68.2	67.0	63.4
7	68.8	73.6	73.0	73.6	74.2	73.4	67.7	73.5	73.1	74.2	74.8	72.1
8	71.1	77.9	72.3	76.5	77.8	73.2	65.6	71.7	70.8	70.4	71.8	70.9
9	62.4	63.8	63.1	64.4	64.3	63.8	63.7	66.9	65.7	67.2	66.7	66.5
10	69.4	72.9	71.2	74.5	72.6	71.8	62.5	73.6	70.4	72.7	71.9	71.1
11	60.0	63.9	63.1	64.6	63.6	63.3	58.4	61.4	61.9	61.4	61.8	61.7
12	64.4	76.2	73.5	76.1	75.2	73.2	61.9	71.2	70.3	71.7	72.3	70.5
13	57.2	66.1	62.4	65.8	63.0	59.3	55.1	63.7	63.5	62.4	61.8	60.6
14	55.4	61.7	61.7	61.6	63.0	60.1	55.0	63.7	64.8	62.1	63.1	62.5
$\overline{AUC}$	61.0	68.2	66.2	68.7	68.2	66.4	59.1	66.3	64.8	65.8	66.3	64.6

Table B.6: *Predictive performance on 14 gene-disease associations using network induced by the BIOGPS.*

Disease	-C	+A	+B	+C	+D	+E	-D	+A	+B	+C	+D	+E
1	60.4	65.2	63.6	64.7	65.3	64.6	61.1	63.0	62.2	63.0	63.3	62.5
2	53.0	67.5	64.2	66.8	66.9	65.1	54.4	63.7	59.7	59.8	63.8	61.0
3	50.4	62.7	52.8	61.8	62.3	57.1	50.4	57.4	56.7	56.9	58.6	56.1
4	60.2	74.0	72.4	72.8	74.3	74.9	61.9	72.0	68.2	69.6	71.7	69.9
5	54.8	60.8	61.1	61.4	61.5	61.9	54.8	61.2	60.2	61.9	61.5	61.3
6	62.0	70.6	67.6	70.8	69.5	66.1	61.3	70.0	67.0	69.3	69.5	67.6
7	66.0	73.4	71.7	72.5	72.9	73.2	69.8	73.7	73.3	73.7	74.1	73.4
8	71.4	76.9	73.9	75.3	76.0	74.0	68.7	75.0	73.3	74.4	74.2	72.8
9	59.3	64.2	62.1	63.6	63.3	62.4	62.4	64.4	63.7	65.0	64.7	64.3
10	69.6	75.9	74.4	76.1	76.2	73.9	68.4	72.8	70.2	73.5	71.8	70.5
11	56.0	61.4	58.1	61.7	61.9	60.0	60.6	63.5	63.1	63.5	63.6	62.8
12	65.4	74.2	72.7	73.6	74.2	74.0	64.5	76.2	73.4	74.3	74.6	74.3
13	57.3	67.2	60.5	67.3	67.7	63.5	56.4	63.3	61.7	61.7	63.3	60.5
14	55.5	63.1	60.0	62.3	63.8	63.2	54.9	62.0	61.8	62.0	62.4	61.1
$\overline{AUC}$	60.1	68.4	65.4	67.9	68.3	66.7	60.7	67.0	65.3	66.3	66.9	65.6

Table B.7: *Predictive performance on 14 gene-disease associations using network induced by the BIOGRIDphys.*

Disease	-A	+A	+B	+C	+D	+E	-B	+A	+B	+C	+D	+E
1	76.4	79.2	77.8	77.8	79.6	78.3	73.2	79.7	75.0	73.9	79.5	74.9
2	60.5	66.4	61.6	60.8	67.3	63.4	52.3	63.3	55.5	53.0	61.3	53.8
3	62.2	77.2	67.7	67.4	77.2	69.2	49.1	65.2	53.7	51.8	70.0	54.2
4	69.2	79.7	75.4	75.0	79.1	76.3	63.5	77.5	73.9	71.9	77.9	72.7
5	59.2	66.6	63.3	62.1	67.5	62.7	50.2	59.5	56.2	53.4	58.7	55.0
6	63.1	69.7	66.7	66.7	70.1	67.1	54.4	62.9	60.3	58.7	63.0	59.0
7	60.1	67.2	62.3	61.8	72.2	61.8	53.2	64.1	60.3	59.1	67.8	59.1
8	70.6	75.1	71.0	71.1	75.7	71.8	62.9	73.6	68.0	67.4	74.5	67.6
9	69.1	72.1	70.5	70.5	72.3	73.7	61.3	69.2	63.6	62.1	69.0	64.9
10	61.3	70.4	65.6	66.7	70.4	68.1	57.3	67.4	60.8	60.3	68.6	60.3
11	58.9	64.0	61.4	60.6	64.1	62.1	56.2	61.5	59.3	58.4	61.4	59.6
12	61.0	68.3	70.8	66.3	69.5	67.4	59.6	68.6	67.6	67.1	69.4	66.7
13	55.4	66.3	64.7	57.3	69.1	61.7	57.6	71.5	67.2	66.2	71.9	68.3
14	57.4	68.7	61.4	60.9	72.4	64.3	53.2	64.7	60.0	57.8	64.5	58.9
$\overline{AUC}$	63.2	70.8	67.2	66.1	71.9	67.7	57.4	67.8	63.0	61.5	68.4	62.5

Table B.8: *Predictive performance on 14 gene-disease associations using network induced by the BIOGRIDphys.*

Disease	-C	+A	+B	+C	+D	+E	-D	+A	+B	+C	+D	+E
1	66.4	75.6	69.9	72.1	79.2	73.5	76.6	80.0	78.2	78.1	80.2	78.7
2	54.4	63.6	57.5	57.8	65.9	58.7	59.3	66.5	61.8	59.5	65.9	62.8
3	60.0	74.0	60.6	60.3	74.0	65.5	64.4	76.3	68.7	68.4	76.3	71.4
4	60.2	73.3	65.1	67.3	71.8	67.3	70.1	79.5	76.9	75.8	79.7	76.7
5	48.6	61.9	56.5	55.4	62.7	57.4	58.7	66.0	62.6	60.9	64.6	61.1
6	61.7	68.8	65.2	64.5	68.8	65.2	63.2	69.5	67.1	67.1	70.1	67.3
7	56.4	65.5	58.5	58.8	68.7	60.3	61.3	66.8	63.1	62.3	71.1	62.5
8	66.8	74.0	73.1	74.0	74.6	72.5	72.3	75.3	72.3	72.3	75.4	72.7
9	74.0	78.5	75.1	75.6	79.1	75.1	70.4	73.0	71.6	71.7	72.6	74.8
10	54.4	66.9	64.1	61.3	68.2	58.7	62.3	71.0	66.5	67.3	70.8	68.3
11	58.4	64.5	60.0	60.5	63.9	61.4	59.1	62.9	61.0	60.3	63.2	61.8
12	55.9	66.9	56.9	65.4	66.7	64.4	60.8	67.2	70.1	66.1	68.5	67.1
13	54.8	66.2	64.4	58.7	69.4	63.5	55.2	65.5	64.1	57.0	68.1	60.9
14	55.1	66.6	59.7	57.5	67.6	62.6	56.4	67.5	61.7	60.7	68.7	64.1
$\overline{AUC}$	59.1	69.0	63.3	63.5	70.0	64.7	63.6	70.5	67.6	66.3	71.1	67.9

Table B.9: *Predictive performance on 14 gene-disease associations using network induced by the HPRD.*

Disease	-A	+A	+B	+C	+D	+E	-B	+A	+B	+C	+D	+E
1	75.5	77.7	77.8	76.7	77.9	76.1	75.1	77.2	77.0	76.5	76.0	75.9
2	56.6	61.1	58.6	59.6	63.5	58.6	55.9	59.5	56.6	58.0	60.2	58.0
3	61.2	73.0	69.0	71.3	76.1	67.9	56.1	70.0	61.1	67.7	71.1	63.3
4	67.3	74.9	71.0	70.9	74.1	69.7	65.5	72.7	68.6	68.0	71.8	67.5
5	57.6	66.9	65.6	68.6	68.4	68.2	55.1	67.3	62.9	64.5	66.5	62.6
6	67.1	73.7	72.0	72.1	73.5	69.3	64.8	71.8	69.8	69.1	68.6	66.4
7	68.8	73.6	72.4	72.7	73.7	71.9	65.5	71.9	69.4	71.6	72.4	69.8
8	76.2	82.4	76.7	81.9	83.3	77.1	73.4	80.7	73.6	79.5	82.3	74.8
9	68.2	76.2	75.6	71.2	74.5	77.3	64.4	72.2	70.9	68.6	69.4	76.0
10	66.0	76.2	75.9	74.6	76.6	73.8	60.4	74.9	71.8	69.2	71.0	69.4
11	60.5	65.2	63.2	64.4	65.0	63.8	58.3	64.1	60.7	62.7	63.2	61.5
12	61.9	69.8	69.3	68.5	72.1	64.3	60.5	68.8	68.8	67.6	68.0	63.0
13	67.6	72.2	71.9	73.3	72.6	71.4	65.1	71.5	69.4	69.4	69.7	67.1
14	68.4	74.1	72.7	71.8	74.1	71.2	64.6	70.5	66.7	67.8	66.9	67.5
$\overline{AUC}$	65.9	72.6	70.8	71.3	73.2	70.0	63.2	70.9	67.7	68.6	69.8	67.3

Table B.10: *Predictive performance on 14 gene-disease associations using network induced by the HPRD.*

Disease	-C	+A	+B	+C	+D	+E	-D	+A	+B	+C	+D	+E
1	75.6	78.2	78.0	78.8	78.6	77.1	75.6	78.2	78.1	77.7	78.8	77.7
2	58.0	62.5	60.6	60.7	63.0	60.7	58.0	60.4	59.2	60.0	63.5	59.2
3	65.5	73.4	70.7	73.5	76.6	69.4	64.4	72.0	70.1	73.7	74.6	70.2
4	68.1	73.9	73.2	72.0	74.3	71.3	68.3	75.1	72.1	71.2	74.0	71.5
5	59.0	68.1	67.1	67.2	68.3	66.9	59.0	68.5	66.7	69.0	68.2	67.8
6	68.3	75.5	72.2	75.3	73.8	71.9	67.1	73.4	72.3	72.1	73.6	70.7
7	71.1	73.5	72.7	72.8	74.2	73.1	70.1	73.3	72.3	72.9	73.5	72.4
8	77.6	82.3	78.8	81.4	83.3	79.2	79.3	82.7	80.0	82.9	83.2	80.3
9	71.0	77.5	77.1	77.2	77.5	81.0	69.8	76.0	75.6	72.5	75.3	78.8
10	67.1	77.0	69.9	73.8	77.5	74.8	69.6	76.3	75.2	76.8	77.1	75.2
11	60.3	65.2	63.4	67.2	65.6	64.4	61.3	65.8	63.5	65.1	65.1	64.5
12	59.2	71.7	70.9	68.6	73.0	64.2	61.7	70.9	69.7	69.7	72.0	64.3
13	65.3	73.3	71.1	73.3	73.3	71.8	67.7	74.1	72.5	74.0	73.2	71.7
14	63.4	72.4	67.1	67.3	73.7	72.1	68.7	74.3	72.5	71.8	74.5	73.9
$\overline{AUC}$	66.4	73.2	70.9	72.1	73.8	71.3	67.2	72.9	71.4	72.1	73.3	71.3

Table B.11: *Predictive performance on 14 gene-disease associations using network induced by the OMIM.*

Disease	-A	+A	+B	+C	+D	+E	-B	+A	+B	+C	+D	+E
1	84.9	86.9	85.6	87.4	86.7	85.8	84.4	86.3	85.3	85.9	85.9	85.4
2	76.6	78.4	78.4	78.8	78.4	78.4	75.7	77.1	77.4	77.3	77.1	77.1
3	78.4	83.3	83.3	83.8	83.2	84.6	74.4	82.1	80.8	82.5	82.4	82.2
4	91.3	93.0	93.0	93.0	93.0	93.1	89.3	93.0	91.5	92.6	93.0	92.1
5	76.1	82.3	79.7	84.1	80.4	80.7	75.6	83.9	79.6	84.3	82.4	79.9
6	81.9	84.7	83.7	83.9	84.6	83.8	79.1	82.7	80.8	83.6	84.1	80.8
7	81.2	85.2	84.5	84.1	84.9	83.2	79.5	84.3	82.8	82.2	84.3	81.9
8	84.3	90.5	91.4	91.0	90.5	92.2	83.9	90.1	89.8	89.7	90.1	90.9
9	78.8	80.6	80.4	80.8	80.4	80.4	78.2	79.9	79.9	80.1	80.3	80.4
10	86.3	87.6	87.6	87.6	87.6	87.6	86.7	88.1	88.1	88.1	88.1	88.1
11	83.3	84.6	84.6	84.6	84.6	84.6	79.8	83.0	81.7	83.1	82.9	82.2
12	82.0	87.3	85.2	87.0	86.8	85.3	79.8	85.3	81.9	87.2	85.4	81.8
13	85.0	88.6	88.9	88.6	89.4	89.2	84.1	88.9	88.6	89.4	89.6	89.1
14	97.4	99.2	98.6	99.2	99.2	99.5	97.2	99.3	98.2	99.1	99.1	99.6
$\overline{AUC}$	83.4	86.6	86.1	86.7	86.4	86.3	82.0	86.0	84.8	86.1	86.0	85.1

Table B.12: *Predictive performance on 14 gene-disease associations using network induced by the OMIM.*

Disease	-C	+A	+B	+C	+D	+E	-D	+A	+B	+C	+D	+E
1	87.3	89.7	88.7	90.2	89.5	88.4	84.7	86.8	85.5	86.5	86.5	85.7
2	71.2	72.9	72.1	72.1	72.8	72.1	76.4	78.0	78.0	78.2	78.0	78.0
3	78.9	84.2	83.3	84.2	84.0	84.2	77.5	82.7	82.6	83.2	82.9	83.6
4	88.8	90.2	90.0	90.2	90.0	90.0	91.4	92.9	92.8	92.8	92.9	93.1
5	77.7	82.2	80.6	83.5	83.1	82.5	76.3	82.3	80.1	83.8	81.3	80.9
6	76.3	82.4	81.9	82.0	83.7	82.2	82.2	84.9	83.8	83.8	84.6	84.5
7	82.7	85.4	84.1	85.4	85.8	84.1	80.6	84.6	83.9	84.3	83.9	83.3
8	89.3	91.6	91.3	91.8	91.7	92.4	84.3	90.5	90.1	90.5	90.3	93.5
9	78.5	81.5	81.5	81.5	81.5	81.5	78.6	80.4	80.4	80.8	80.4	80.4
10	85.1	88.0	88.0	88.1	88.3	88.0	86.0	87.5	87.5	87.6	87.5	87.5
11	82.0	82.9	82.8	82.7	82.8	82.6	83.0	84.4	84.4	84.4	84.4	84.4
12	84.7	87.8	86.5	88.2	87.7	85.9	81.3	86.5	83.8	86.4	86.3	85.0
13	78.2	83.1	83.1	83.1	83.8	83.1	85.0	88.5	88.5	88.5	89.0	88.5
14	97.5	98.7	98.2	98.4	98.6	98.7	97.4	99.2	98.3	99.2	99.2	99.5
$\overline{AUC}$	82.7	85.7	85.2	85.8	86.0	85.4	83.2	86.4	85.7	86.4	86.2	86.3

Chapter C

Graph-based Data Integration Approaches

Table C.1: The performance of different techniques in the experimental setting of Chen *et al* [20] expressed in terms of AUC. Except for our Graphone's variations, these results were taken from that work. The p-values indicate significance of the pairwise AUC differences with respect to Scuba AUC [36]. Asterisks indicate significance of the test (p-value < 0.05) of Scuba1 with compared methods.

Method	AUC(%)	p-value
F3PC [20]	0.830	$1.39 \cdot 10^{-4}$ *
MRF [21]	0.731	$<10^{-6}$ *
DIR [23]	0.716	$<10^{-6}$ *
GeneWanderer [45]	0.711	$<10^{-6}$ *
Avg	81.9	$<10^{-6}$ *
Scuba1	87.6	-
Scuba2	88.3	-
PLC	86.8	-
DIGI	88.1	-

Table C.2: Scuba performance for single disorders considered by Börnigen *et al* in their evaluation of gene prioritization tools [**?**].

Disease	**Associated genes**	**genome-wide AUC**	**candidate set AUC**
Abdominal aortic aneurysm	ENSG00000136848	0.77	0.84
Alcohol dependence	ENSG00000148680	0.98	0.98
Arthrogryposis	ENSG00000152818	0.98	1.0
Asthma	ENSG00000182578	0.93	0.94
Autosomal recessive primary microcephaly	ENSG00000075702	0.41	0.44
Behcet's disease	ENSG00000136634	0.98	0.97
Bipolar schizoaffective disorder	ENSG00000146276 ENSG00000139618	0.97	0.98
Complex heart defect	ENSG00000121068	0.98	1.0
Congenital anomalies of the kidney and the urinary tract	ENSG00000164736 ENSG00000178188	0.97	0.96
Congenital diaphragmatic hernia	ENSG00000004961 ENSG00000154309	0.86	0.87
Crohn's disease	ENSG00000176920 ENSG00000185651 ENSG00000069399	0.89	0.89
Dursun syndrome	ENSG00000141349	0.58	0.46
Ehlers-Danlos syndrome	ENSG00000169105	0.99	1.0
Esophageal squamous cell carcinoma	ENSG00000138193 ENSG00000101276	0.3	0.23

Disease	Associated genes	genome-wide AUC	candidate set AUC
Leprosy	ENSG00000111537	0.96	0.9
Lung adenocarcinoma	ENSG00000073282	0.89	0.84
Methylmalonic aciduria	ENSG00000167775	0.9	0.93
Metopic craniosynostosis	ENSG00000106571	0.98	0.98
Mitochondrial complex I deficiency	ENSG00000177646	0.95	0.96
Multiple sclerosis	ENSG00000120088	0.83	0.84
Myelodysplastic syndromes	ENSG00000106462	0.81	0.83
Nasopharyngeal carcinoma	ENSG00000085276 ENSG00000127863	0.81	0.8
Nonsyndromic cleft lip/palate	ENSG00000148175	0.82	0.8
Parkinson's disease	ENSG00000175104	0.82	0.8
Periventricular heterotopia	ENSG00000102103	0.54	0.45
Primary biliary cirrhosis	ENSG00000142606 ENSG00000142539	0.82	0.77
Psoriasis	ENSG00000056972	0.96	1.0
Retinal-renal ciliopathy	ENSG00000054282	1.0	1.0
Single-suture craniosynostosis	ENSG00000124813	0.98	0.98
Smooth pursuit eye movement abnormality	ENSG00000099901	0.27	0.2
Testicular germ cell tumor	ENSG00000137090 ENSG00000171681	0.5	0.41
Tetralogy of Fallot	ENSG00000145012	0.74	0.67
Type 2 diabetes	ENSG00000182247	0.21	0.19

Table C.3: Scuba performance for single disorders considered in Table 4 in the main text. These are the multifactorial diseases employed by Börnigen *et al* [**?**] with at least a new gene annotation between March 2013 and February 2017 as reported by the Human Phenotype Ontology [**?**].

Disease	**Associated genes**	**genome-wide AUC**
Behcet's disease	ENSG00000162594 ENSG00000168811 ENSG00000138378 ENSG00000163823 ENSG00000136869 ENSG00000183542 ENSG00000164307 ENSG00000026103 ENSG00000206340 ENSG00000206450 ENSG00000134882	0.87
Bipolar schizoaffective disorder	ENSG00000175344 ENSG00000138592 ENSG00000151702 ENSG00000124782 ENSG00000171988 ENSG00000176986	0.68
Crohn's disease	ENSG00000140368	0.90
Parkinson's disease	ENSG00000064692 ENSG00000153234 ENSG00000116675 ENSG00000159082 ENSG00000184381 ENSG00000138246	0.89
Primary biliary cirrhosis	ENSG00000128604 ENSG00000181634 ENSG00000105329 ENSG00000110777 ENSG00000064419 ENSG00000016602 ENSG00000141076 ENSG00000106089 ENSG00000132912	0.76
Psoriasis	ENSG00000206237 ENSG00000196126 ENSG00000179344 ENSG00000206306 ENSG00000163599 ENSG00000206240 ENSG00000077150 ENSG00000141527 ENSG00000198246	0.94
Smooth pursuit eye movement abnormality	ENSG00000104133 ENSG00000171385 ENSG00000020922 ENSG00000070610 ENSG00000013503 ENSG00000167658	0.73