Chapter 11

GRAPH CLASSIFICATION

Koji Tsuda
Computational Biology Research Center, National Institute of Advanced Industrial Science and Technology (AIST)
Tokyo, Japan
koji.tsuda@aist.go.jp

Hiroto Saigo
Max Planck Institute for Informatics
Saarbrucken, Germany
hiroto@mpi-inf.mpg.de

Abstract Supervised learning on graphs is a central subject in graph data processing. In graph classification and regression, we assume that the target values of a certain number of graphs or a certain part of a graph are available as a training dataset, and our goal is to derive the target values of other graphs or the remaining part of the graph. In drug discovery applications, for example, a graph and its target value correspond to a chemical compound and its chemical activity. In this chapter, we review state-of-the-art methods of graph classification. In particular, we focus on two representative methods, graph kernels and graph boosting, and we present other methods in relation to the two methods. We describe the strengths and weaknesses of different graph classification methods and recent efforts to overcome the challenges.

Keywords: graph classification, graph mining, graph kernels, graph boosting

1. Introduction

Graphs are general and powerful data structures that can be used to represent diverse kinds of objects. Much of the real world data is represented not

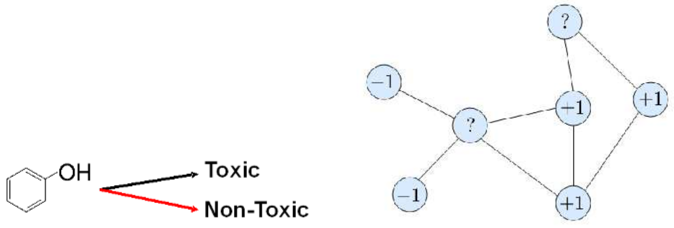


Figure 11.1. Graph classification and label propagation.

as vectors, but as graphs (including sequences and trees, which are specialized graphs). Examples include biological sequences, semi-structured texts such as HTML and XML, chemical compounds, RNA secondary structures, API call graphs, etc. The topic of graph data processing is not new. Over the last three decades, there have been continuous efforts in developing new methods for processing graph data. Recently we have seen a surge of interest in this topic, fueled partly by new technical advances, for example, development of graph kernels [21] and graph mining [52] techniques, and partly by demands from new applications, for example, chemical informatics. In fact, chemical informatics is one of the most prominent fields that deal with large repositories of graph data. For example, NCBI's PubChem has millions of chemical compounds that are naturally represented as molecular graphs. Also, many different kinds of chemical activity data are available, which provides a huge test-bed for graph classification methods.

This chapter aims at giving an overview of existing graph classification methods. The term "graph classification" can mean two different tasks. The first task is to build a model to predict the class label of a whole graph (Figure 11.1, left). The second task is to predict the class labels of nodes in a large graph (Figure 11.1, right). For clarity, we used the term to represent the first task, and we call the second task "label propagation"[6]. This chapter mainly deals with graph classification, but we will provide a short review of label propagation in Section 5.

Graph classification tasks can either be unsupervised or supervised. Unsupervised methods classify graphs into a certain number of categories by similarity [47, 46]. In supervised classification, a classification model is constructed by learning from training data. In the training data, each graph (e.g., a chemical compound) has a target value or a class label (e.g., biochemical activity). Supervised methods are more fundamental from a technical point of view, because unsupervised learning problems can be solved by supervised methods via probabilistic modeling of latent class labels [46]. In this chapter, we focus on two supervised methods for graph classification: graph kernels and graph boosting [40], which are similarity- and feature-based respectively. The two

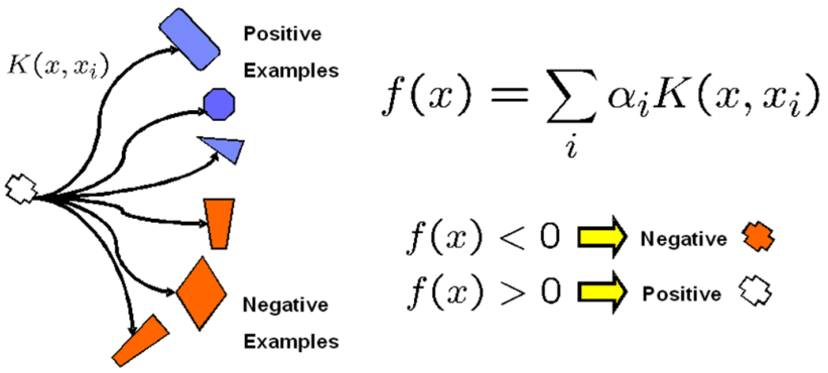


Figure 11.2. Prediction rules of kernel methods.

methods differ in many aspects, and a characterization of the difference of these two methods would be helpful in characterizing other methods.

Kernel methods, such as support vector machines, construct a prediction rule based on a similarity function between two objects [42]. Similarity functions which satisfy a mathematical condition called positive definiteness are called *kernel functions*. For example, in Figure 11.2, the similarity between two objects is represented by a *kernel function* $K(x, x')$. The prediction function $f(x)$ is a linear combination of x's similarities to each training example $K(x, x_i)$, $i = 1, \ldots, n$. In order to apply kernel methods to graph data, it is necessary to define a kernel function for graphs that can measure the similarity between two graphs. It is natural to use the number of shared substructures in two graphs as a similarity measure. However, the enumeration of subgraphs of a given graph is NP-hard [12]. Therefore, one needs to use simpler substructures such as paths and trees. Graph kernels [21] are based on the weighted counts of common paths. A clever recursive algorithm is employed to compute the similarity without total enumeration of substructures.

One obvious drawback of graph kernels is that it is not clear which substructures have the biggest contribution to classification. For a new graph classified by similarity, it is not always possible to know which part of the compound is essential in classification. In many chemical applications, the users are interested not only in accurate prediction of biochemical activities, but also in the mechanism creating the activities. This interpretation problem motivates us to reexamine the approach of subgraph enumeration. Recently, frequent subgraph enumeration algorithms such as AGM [18], Gaston [33] and gSpan [52] have been proposed. They can enumerate all the subgraph patterns that appear more than m times in a graph database. The threshold m is called *minimum support*. Frequent subgraph patterns are determined by branch-and-bound search in a tree shaped search space (Figure 11.7). The computational time crucially depends on the minimum support parameter. For larger values of the support parameter, the search tree can be pruned earlier. For chemical compound datasets, it is easy to mine tens of thousands of graphs on a commodity desktop computer, if the minimum support is reasonably high (e.g., 10% of the number of graphs). However, it is known that, to achieve the best accuracy, the minimum support has to be set to a small value (e.g., smaller than 1%) [51, 23, 16]. In such a setting, the graph mining becomes prohibitively inefficient, because the algorithm creates millions of patterns. This also makes subsequent processing very expensive. Graph boosting [40] progressively constructs the prediction rule in an iterative fashion, and in each iteration only a few informative subgraphs are discovered. In comparison to the na"ve method of using frequent mining and support vector machines, the graph mining routine has to be invoked multiple times. However, an additional search tree pruning condition can speed up each call, and the overall time is shorter than the na"ve method.

The rest of this chapter is organized as follows. In Section 2, we will explain graph kernels, and review its recent extensions for graph classification. In Section 3, we will discuss graph boosting and other methods based on explicit substructure mining. Applications of graph classification methods are reviewed in Section 4. Section 5 briefly presents the label propagation techniques. We conclude the chapter in Section 6.

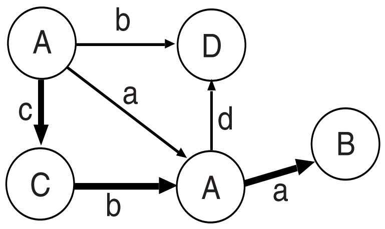
2. Graph Kernels

We consider a graph kernel as a similarity measure for two graphs whose nodes and edges are labeled (Figure 11.3). In this section, we present the most fundamental kernel called the marginalized graph kernel [21], which is based on graph paths. Recently, different versions of graph kernels have been proposed using different substructures. Examples include cyclic paths [17] and trees [29].

The proposed graph kernel is based on the idea of random walking. For the labeled graph shown in Figure 11.3a, a label sequence is produced by traversing the graph. A representative example is as follows:

$$(A, c, C, b, A, a, B), \tag{2.1}$$

The vertex labels A, B, C, D and the edge labels a, b, c, d appear alternately. By repeating random walks with random initial and end points, it is possible to obtain the probabilities for all possible walks (Figure 11.3b). The essential idea of the graph kernel is to derive a similarity measure of two graphs by comparing their probability tables. It is computationally infeasible to perform all possible random walks. Therefore, we employ a recursive algorithm which can estimate the underlying probabilities. The node and edge labels are either



Label path $\boldsymbol{h}$	Probability $p(\boldsymbol{h}\|G)$
AaA	0.001
⋮	⋮
AcDbE	0.000003
⋮	⋮
AeAdDaBcD	0.00000007
⋮	⋮

Figure 11.3. (a) An example of labeled graphs. Vertices and edges are labeled by uppercase and lowercase letters, respectively. By traversing along the bold edges, the label sequence (2.1) is produced. (b) By repeating random walks, one can construct a list of probabilities.

discrete symbols or vectors. In the latter case, it is necessary to define node kernels and edge kernels to specify the similarity of vectors.

Before describing technical details, we formally define a labeled graph. Let Σ_V denote the set of vertex labels, and Σ_E the set of edge labels. Let $\mathcal{X}$ be a finite nonempty set of vertices, v be a function $v : \mathcal{X} \to \Sigma_V$. Let $\mathcal{L}$ be a set of vertex pairs that denote edges, and e be a function $e : \mathcal{L} \to \Sigma_E$. (We assume that there are no multiple edges from one vertex to another.) Then $G = (\mathcal{X}, v, \mathcal{L}, e)$ is a labeled graph with directed edges. Our task is to construct a kernel function $k(G, G')$ between two labeled graphs G and G'.

2.1 Random Walks on Graphs

We extract features (labeled sequences) from a graph G by performing random walks. At the first step, we sample a node $x_1 \in \mathcal{X}$ from an initial probability distribution $p_s(x_1)$. Subsequently, at the ith step, the next vertex $x_i \in \mathcal{X}$ is sampled subject to a transition probability $p_t(x_i|x_{i-1})$, or the random walk ends at node x_{i-1} with probability $p_q(x_{i-1})$. In other words, at the ith step, we have:

$$\sum_{k=1}^{|\mathcal{X}|} p_t(x_k|x_{i-1}) + p_q(x_{i-1}) = 1 \tag{2.2}$$

that is, at each step, the probabilities of transitions and termination sum to 1.

When we do not have any prior knowledge, we can set the initial probability distribution p_s to be the uniform distribution, the transition probability p_t to be a uniform distribution over the vertices adjacent to the current vertex, and the termination probability p_q to be a small constant probability.

From the random walk, we obtain a sequence of vertices called a *path*:

$$\mathbf{x} = (x_1, x_2, \ldots, x_\ell), \tag{2.3}$$

where ℓ is the length of $\mathbf{x}$ (possibly infinite). The final probability of obtaining path $\mathbf{x}$ is the product of the probabilities that the path starts with x_1, transits from x_{i-1} to x_i for each i, and finally terminates with x_l:

$$p(\mathbf{x}|G) = p_s(x_1) \prod_{i=2}^{\ell} p_t(x_i|x_{i-1}) p_q(x_\ell).$$

Let us define a *label sequence* as sequence of alternating vertex labels and edge labels:

$$\mathbf{h} = (h_1, h_2, \ldots, h_{2\ell-1}) \in (\Sigma_V \Sigma_E)^{\ell-1} \Sigma_V.$$

Associated with a path $\mathbf{x}$, we obtain a label sequence

$$\mathbf{h_x} = (v_{x_1}, e_{x_1,x_2}, v_{x_2}, e_{x_2,x_3}, \ldots, v_{x_\ell}).$$

which is a sequence of alternating vertex and edge labels. Since multiple vertices (edges) may have the same label, multiple paths may map to one label sequence. The probability of obtaining a label sequence $\mathbf{h}$ is thus the sum of the probabilities of each path that emits $\mathbf{h}$. This can be expressed as

$$p(\mathbf{h}|G) = \sum_{\mathbf{x}} \delta(\mathbf{h} = \mathbf{h_x}) \cdot \left(p_s(x_1) \prod_{i=2}^{\ell} p_t(x_i|x_{i-1}) p_q(x_\ell) \right),$$

where δ is a function that returns 1 if its argument holds, 0 otherwise.

2.2 Label Sequence Kernel

We now define a kernel k_z between two label sequences $\mathbf{h}$ and $\mathbf{h}'$. The sequence kernel is defined based on kernels for vertex labels and edge labels.

We assume two kernel functions, $k_v(v, v')$ and $k_e(e, e')$, are readily defined between vertex labels and edge labels. We constrain both kernels to be nonnegative[1]. An example of a vertex label kernel is the identity kernel, that is, the kernel return 1 if the two labels are the same, 0 otherwise. It can be expressed as:

$$k_v(v, v') = \delta(v = v') \tag{2.4}$$

where $\delta(\cdot)$ is a function that returns 1 if its argument holds, and 0 otherwise. The above kernel (2.4) is for labels of discrete values. If the labels are defined in $\mathbb{R}$, then the Gaussian kernel can be used as a natural choice [42]:

$$k_v(v, v') = \exp(- \parallel v - v' \parallel^2 / 2\sigma^2), \tag{2.5}$$

Edge kernels can be defined in the same way as in (2.4) and (2.5).

Based on the vertex label and the edge label kernels, we defome the kernel for label sequences. If two sequences $\mathbf{h}$ and $\mathbf{h}'$ are of the same length, or

[1]This constraint will play an important role in proving the convergence of our kernel.

$\ell(\mathbf{h}) = \ell(\mathbf{h}')$, then the sequence kernel is defined as the product of the label kernels:

$$k_z(\mathbf{h}, \mathbf{h}') = k_v(h_1, h'_1) \prod_{i=2}^{\ell} k_e(h_{2i-2}, h'_{2i-2}) k_v(h_{2i-1}, h'_{2i-1}). \tag{2.6}$$

If the two sequences are of different length, or $\ell(\mathbf{h}) \neq \ell(\mathbf{h}')$, then the sequence kernel returns 0, that is, $k_z(\mathbf{h}, \mathbf{h}') = 0$.

Finally, our label sequence kernel is defined as the expectation of k_z over all possible $\mathbf{h} \in G$ and $\mathbf{h}' \in G'$.

$$k(G, G') = \sum_{\mathbf{h}} \sum_{\mathbf{h}'} k_z(\mathbf{h}, \mathbf{h}') p(\mathbf{h}|G) p(\mathbf{h}'|G'). \tag{2.7}$$

Here, $p(\mathbf{h}|G)p(\mathbf{h}'|G')$ is the probabilty that $\mathbf{h}$ and $\mathbf{h}'$ occur in G and G', respectively, and $k_z(\mathbf{h}, \mathbf{h}')$ is their similarity. This kernel is valid, as it is described as an inner product of two vectors $p(\mathbf{h}|G)$ and $p(\mathbf{h}'|G')$.

2.3 Efficient Computation of Label Sequence Kernels

The label sequence kernel (2.7) defined above can be expanded as follows:

$$\begin{aligned} k(G, G') = & \textstyle\sum_{\ell=1}^{\infty} \sum_{\mathbf{h}} \sum_{\mathbf{h}'} k_v(h_1, h'_1) \times \\ & \left(\textstyle\prod_{i=2}^{\ell} k_e(h_{2i-2}, h'_{2i-2}) k_v(h_{2i-1}, h'_{2i-1}) \right) \times \\ & \left(\textstyle\sum_{\mathbf{x}} \delta(\mathbf{h} = \mathbf{h}_{\mathbf{x}}) \cdot \left(p_s(x_1) \prod_{i=2}^{\ell} p_t(x_i | x_{i-1}) p_q(x_\ell) \right) \right) \times \\ & \left(\textstyle\sum_{\mathbf{x}'} \delta(\mathbf{h} = \mathbf{h}_{\mathbf{x}'}) \cdot \left(p_s(x'_1) \prod_{i=2}^{\ell} p_t(x'_i | x'_{i-1}) p_q(x'_\ell) \right) \right). \end{aligned}$$

The straightforward enumeration of all terms to compute the sum has a prohibitive computational cost. In particular, for cyclic graphs, it is infeasible to perform this computation in an enumerative way, because the possible length of a sequence spans from 1 to infinity. Nevertheless, there is an efficient method to compute this kernel as shown below. The method is based on the observation that the kernel has the following nested structure.

$$
\begin{aligned}
k(G,G') = & \lim_{L\to\infty} \sum_{\ell=1}^{L} \\
& \sum_{x_1,x_1'} s(x_1,x_1') \times \\
& \left(\sum_{x_2,x_2'} t(x_2,x_2',x_1,x_1') \times \left(\sum_{x_3,x_3'} t(x_3,x_3',x_2,x_2') \times \right.\right. \\
& \left.\left. \cdots \times \sum_{x_\ell,x_\ell'} t(x_\ell,x_\ell',x_{\ell-1},x_{\ell-1}') q(x_\ell,x_\ell') \right) \cdots \right)
\end{aligned} \tag{2.8}
$$

where

$$
\begin{aligned}
s(x_1,x_1') &= p_s(x_1)p_s'(x_1')k_v(v_{x_1},v'_{x_1'}), \\
q(x_\ell,x_\ell') &= p_q(x_\ell)p_q'(x_\ell') \\
t(x_i,x_i',x_{i-1},x_{i-1}') &= p_t(x_i|x_{i-1})p_t'(x_i'|x_{i-1}')k_v(v_{x_i},v'_{x_i'})k_e(e_{x_{i-1}x_i},e_{x_{i-1}'x_i'})
\end{aligned}
$$

Intuitively, (2.8) computes the expectation of the kernel function over all possible pairs of paths of the same length l. Consider one of such pairs: $(x_1,\cdots,x_\ell)$ in G and $(x_1',\cdots,x_\ell')$ in G'. Here, p_s, p_t, and p_q denote the initial, transition, and termination probability of nodes in graph G, and p_s', p_t', and p_q' denote the initial, transition, and termination probability of nodes in graph G'. Thus, $s(x_1,x_1')$ is the probability-weighted similarity of the first elements in the two paths, $q(x_\ell,x_\ell')$ is the probability that the two paths end with x_ℓ and x_ℓ', and $t(x_i,x_i',x_{i-1},x_{i-1}')$ is the probability-weighted similarity of the ith node pair and edge pair in the two paths.

Acyclic Graphs. Let us first consider the case of acyclic graphs. In an acyclic graph, if there is a directed path from vertex x_1 to x_2, then there is no directed path from vertex x_2 to x_1. It is well known that vertices of a directed, acyclic graph can be numbered in a topological order[2] such that every edge from a vertex numbered i to a vertex numbered j satisfies $i<j$ (see Figure 11.4).

Since there are no directed paths from vertex j to vertex i if $i<j$, we can employ dynamic programming to achieve our goal. Given that both G and G'

[2]Topological sorting of graph G can be done in $O(|\mathcal{X}|+|\mathcal{L}|)$ [7].

are directed acyclic graphs, we can rewrite (2.8) into the following:

$$\begin{aligned} k(G,G') = \textstyle\sum_{x_1.x'_1} s(x_1,x'_1)q(x_1,x'_1) + \lim_{L\to\infty}\sum_{\ell=2}^{L}\sum_{x_1,x'_1} s(x_1,x'_1)\times \\ \left(\textstyle\sum_{x_2>x_1,x'_2>x'_1} t(x_2,x'_2,x_1,x'_1)\left(\sum_{x_3>x_2,x'_3>x'_2} t(x_3,x'_3,x_2,x'_2)\times\right.\right. \\ \left.\left.\left(\cdots\left(\textstyle\sum_{x_\ell>x_{\ell-1},x'_\ell>x'_{\ell-1}} t(x_\ell,x'_\ell,x_{\ell-1},x'_{\ell-1})q(x_\ell,x'_\ell)\right)\right)\cdots\right)\right). \end{aligned} \tag{2.9}$$

The first term corresponds to paths of length 1, and the second term corresponds to paths longer than 1. We define $r(\cdot,\cdot)$ as follows:

$$\begin{aligned} r(x_1,x'_1) := q(x_1,x'_1) + \lim_{L\to\infty}\textstyle\sum_{\ell=2}^{L}\left(\sum_{x_2>x_1,x'_2>x'_1} t(x_2,x'_2,x_1,x'_1)\times\right. \\ \left.\left(\cdots\left(\textstyle\sum_{x_\ell>x_{\ell-1},x'_\ell>x'_{\ell-1}} t(x_\ell,x'_\ell,x_{\ell-1},x'_{\ell-1})q(x_\ell,x'_\ell)\right)\right)\cdots\right), \end{aligned} \tag{2.10}$$

We can rewrite (2.9) as the follows:

$$k(G,G') = \sum_{x_1,x'_1} s(x_1,x'_1)r(x_1,x'_1).$$

The merit of defining (2.10) is that we can exploit the following recursive equation.

$$r(x_1,x'_1) = q(x_1,x'_1) + \sum_{j>x_1,j'>x'_1} t(j,j',x_1,x'_1)r(j,j'). \tag{2.11}$$

Since all vertices are topologically ordered, $r(x_1,x'_1)$ can be efficiently computed by dynamic programming (Figure 11.5) for all x_1 and x'_1. The worst-case time complexity of computing $k(G,G')$ is $O(c\cdot c'\cdot|\mathcal{X}|\cdot|\mathcal{X}'|)$ where c and c' are the maximum out-degree of G and G', respectively.

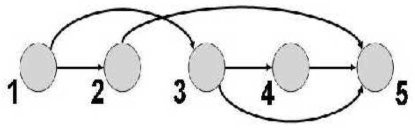


Figure 11.4. A topologically sorted directed acyclic graph. The label sequence kernel can be efficiently computed by dynamic programming running from right to left.

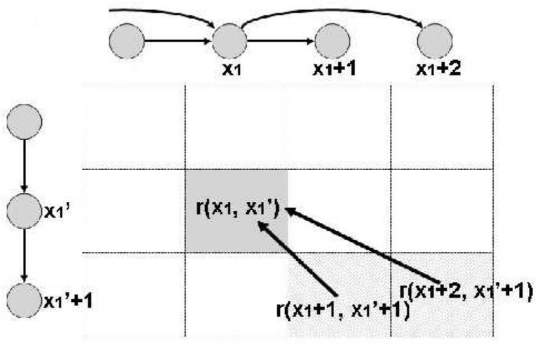


Figure 11.5. Recursion for computing $r(x_1, x_1')$ using recursive equation (2.11). $r(x_1, x_1')$ can be computed based on the precomputed values of $r(x_2, x_2'),\ x_2 > x_1,\ x_2' > x_1'$.

General Directed Graphs. For cyclic graphs, nodes cannot be topologically sorted. This means that we cannot employ a one-pass dynamic programming algorithm for acyclic graphs. However, we can obtain a recursive form of the kernel like (2.11), and reduce the problem to solving a system of simultaneous linear equations.

Let us rewrite (2.8) as

$$k(G,G') = \lim_{L\to\infty} \sum_{\ell=1}^{L} \sum_{x_1,x_1'} s(x_1,x_1') r_\ell(x_1,x_1'), \tag{2.12}$$

where

$$\begin{aligned} r_1(x_1,x_1') &:= q(x_1,x_1') \\ &\text{and} \\ r_\ell(x_1,x_1') &:= \left(\sum_{x_2,x_2'} t(x_2,x_2',x_1,x_1') \left(\sum_{x_3,x_3'} t(x_3,x_3',x_2,x_2') \times \right.\right. \\ &\left.\left. \left(\cdots \left(\sum_{x_\ell,x_\ell'} t(x_\ell,x_\ell',x_{\ell-1},x_{\ell-1}') q(x_\ell,x_\ell')\right)\right)\cdots\right) \\ &\text{for } \ell \geq 2 \end{aligned}$$

Replacing the order of summation in (2.12), we have the following:

$$\begin{aligned} k(G,G') &= \sum_{x_1,x_1'} s(x_1,x_1') \lim_{L\to\infty} \sum_{\ell=1}^{L} r_\ell(x_1,x_1') \\ &= \sum_{x_1,x_1'} s(x_1,x_1') \lim_{L\to\infty} R_L(x_1,x_1'), \end{aligned} \tag{2.13}$$

where

$$R_L(x_1,x_1') := \sum_{\ell=1}^{L} r_\ell(x_1,x_1').$$

Thus we need to compute $R_\infty(x_1,x_1')$ to obtain $k(G,G')$.

Now let us restate this problem in terms of linear system theory [38]. The following recursive relationship holds between r_k and r_{k-1} $(k \geq 2)$:

$$r_k(x_1,x_1') = \sum_{i,j} t(i,j,x_1,x_1') r_{k-1}(i,j). \tag{2.14}$$

Using (2.14), the recursive relationship for R_L also holds as follows:

$$\begin{aligned} R_L(x_1, x_1') &= r_1(x_1, x_1') + \sum_{k=2}^{L} r_k(x_1, x_1') \\ &= r_1(x_1, x_1') + \sum_{k=2}^{L} \sum_{i,j} t(i, j, x_1, x_1') r_{k-1}(i, j) \\ &= r_1(x_1, x_1') + \sum_{i,j} t(i, j, x_1, x_1') R_{L-1}(i, j). \end{aligned} \tag{2.15}$$

Thus, R_L can be perceived as a discrete-time linear system [38] evolving as the time L increases. Assuming that R_L converges (see [21] for the convergence condition), we have the following equilibrium equation:

$$R_\infty(x_1, x_1') = r_1(x_1, x_1') + \sum_{i,j} t(i, j, x_1, x_1') R_\infty(i, j). \tag{2.16}$$

Therefore, the computation of the kernel finally requires solving simultaneous linear equations (2.16) and substituting the solutions into (2.13).

Now let us restate the above discussion in the language of matrices. Let s, $\mathbf{r}_1$, and $\mathbf{r}_\infty$ be $|\mathcal{X}| \cdot |\mathcal{X}'|$ dimensional vectors such that

$$\begin{aligned} \mathbf{s} &= (\cdots, s(i, j), \cdots)^\top \\ \mathbf{r}_1 &= (\cdots, r_1(i, j), \cdots)^\top \\ \mathbf{r}_\infty &= (\cdots, R_\infty(i, j), \cdots)^\top \end{aligned}$$

Let the transition probability matrix T be a $|\mathcal{X}||\mathcal{X}'| \times |\mathcal{X}||\mathcal{X}'|$ matrix,

$$[T]_{(i,j),(k,l)} = t(i, j, k, l).$$

Equation (2.13) can be rewritten as

$$k(G, G') = \mathbf{r}_\infty^T \mathbf{s} \tag{2.17}$$

Similarly, the recursive equation (2.16) is rewritten as

$$\mathbf{r}_\infty = \mathbf{r}_1 + T\mathbf{r}_\infty.$$

The solution of this equation is

$$\mathbf{r}_\infty = (I - T)^{-1}\mathbf{r}_1.$$

Finally, the matrix form of the kernel is

$$k(G, G') = (I - T)^{-1}\mathbf{r}_1\mathbf{s}. \tag{2.18}$$

Computing the kernel requires solving a linear equation or inverting a matrix with $|\mathcal{X}||\mathcal{X}'| \times |\mathcal{X}||\mathcal{X}'|$ coefficients. However, the matrix $I - T$ is actually sparse because the number of non-zero elements of T is less than $c \cdot c' \cdot |\mathcal{X}| \cdot |\mathcal{X}'|$ where c and c' are the maximum out degree of G and G', respectively. Therefore, we can employ efficient numerical algorithms that exploit sparsity [3]. In our implementation, we employed a simple iterative method that updates R_L by using (2.15) until convergence starting from $R_1(x_1, x'_1) = r_1(x_1, x'_1)$.

2.4 Extensions

Vishwanathan et al. [50] proposed a fast way to compute the graph kernel based on the Sylvestor equation. Let A_X, A_Y and B denote $M \times M$, $N \times N$ and $M \times N$ matrices, respectively. They have used the following equation to speed up the computation.

$$(A_X \otimes A_Y)\mathrm{vec}(B) = \mathrm{vec}(A_X B A_Y)$$

where $\otimes$ corresponds to the Kronecker product (tensor product) and vec is the vectorization operator. The left hand side requires $O(M^2N^2)$ time, while the right hand side requires only $O(MN(M + N))$ time. Notice that this trick ("vec-trick") has recently been used in link prediction tasks as well [20].

A random walk can trace the same edge back and forth many times ("tottering"), which could be harmful for similarity measurement. Mahe et al. [28] presented an extension of the kernel without tottering and applied it successfully to chemical informatics data.

3. Graph Boosting

Frequent pattern mining techniques are important tools in data mining [14]. Its simplest form is the classic problem of itemset mining [1], where frequent subsets are enumerated from a series of sets. The original work on this topic is for transactional data, and since then, researchers have applied frequent pattern mining to other structured data such as sequences [35] and trees [2]. Every pattern mining method uses a search tree to systematically organize the patterns. For general graphs, there are technical difficulties about duplication: it is possible to generate the same graph with different paths of the search tree. Methods such as AGM [18] and gspan [52] solve this duplication problem by pruning the search nodes whenever duplicates are found.

The simplest way to apply such pattern mining techniques to graph classification is to build a binary feature vector based on the presence or absence of frequent patterns and apply an off-the-shelf classifier. Such methods are employed in a few chemical informatics papers [16, 23]. However, they are obviously suboptimal because frequent patterns are not necessarily useful for

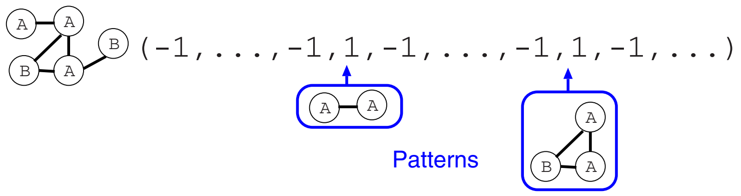


Figure 11.6. Feature space based on subgraph patterns. The feature vector consists of binary pattern indicators.

classification. In chemical data, patterns such as C-C or C-C-C are frequent, but have almost no significance.

To discuss pattern mining strategies for graph classification, let us first define the binary classification problem. The task is to learn a prediction rule from training examples $\{(G_i, y_i)\}_{i=1}^n$, where G_i is a training graph and $y_i \in \{+1, -1\}$ is its associated class label. Let $\mathcal{P}$ be the set of all patterns, i.e., the set of all subgraphs included in at least one training graph, and $d := |\mathcal{P}|$. Then, each graph G_i is encoded as a d-dimensional vector

$$x_{i,p} = \begin{cases} 1 & \text{if } p \subseteq G_i, \\ -1 & \text{otherwise,} \end{cases}$$

This feature space is illustrated in Figure 11.6.

Since the whole feature space is intractably large, we need to obtain a set of informative patterns without enumerating all patterns (i.e., discriminative pattern mining). This problem is close to feature selection in machine learning. The difference is that it is not allowed to scan all features. As in feature selection, we can consider the following three categories in discriminative pattern mining methods: filter, wrapper and embedded [24]. In filter methods, discriminative patterns are collected by a mining call before the learning algorithm is started. They employ a simple statistical criterion such as information gain [31]. In wrapper and embedded methods, the learning algorithm chooses features via minimization of a sparsity-inducing objective function. Typically, they have a high dimensional weight vector and most of these weights coverage to zero after optimization. In most cases, the sparsity is induced by L1-norm regularization [40]. The difference between wrapper and embedded methods are subtle, but wrapper methods tend to be based on heuristic ideas by reducing the features recursively (recursive feature elimination)[13]. Graph boosting is an embedded method, but to deal with graphs, we need to combine L1-norm regularization with graph mining.

3.1 Formulation of Graph Boosting

The name 'boosting' comes from the fact that linear program boosting (LPBoost) is used as a fundamental computational framework. In chemical informatics experiments [40], it was shown that the accuracy of graph boosting is better than graph kernels. At the same time, key substructures are explicitly discovered.

Our prediction rule is a convex combination of binary indicators $x_{i,j}$, and has the form

$$f(\boldsymbol{x}_i) = \sum_{p \in \mathcal{P}} \beta_p \boldsymbol{x}_{i,p}, \tag{3.1}$$

where $\boldsymbol{\beta}$ is a $|\mathcal{P}|$-dimensional column vector such that $\sum_{p \in \mathcal{P}} \beta_p = 1$ and $\beta_p \geq 0$.

This is a linear discriminant function in an intractably large dimensional space. To obtain an interpretable rule, we need to obtain a *sparse* weight vector $\boldsymbol{\beta}$, where only a few weights are nonzero. In the following, we will present a linear programming approach for efficiently capturing such patterns. Our formulation is based on that of LPBoost [8], and the learning problem is represented as

$$\min_{\boldsymbol{\beta}} \quad \|\boldsymbol{\beta}\|_1 + \lambda \sum_{i=1}^{n} \left[1 - \boldsymbol{y}_i f(\boldsymbol{x}_i)\right]_+ , \tag{3.2}$$

where $\|x\|_1 = \sum_{i=1}^{n} |\boldsymbol{x}_i|$ denotes the ℓ_1 norm of $\boldsymbol{x}$, λ is a regularization parameter, and the subscript "+" indicates positive part. A soft-margin formulation of the above problem exists [8], and can be written as follows:

$$\min_{\boldsymbol{\beta},\boldsymbol{\xi},\rho} \quad -\rho + \lambda \sum_{i=1}^{n} \xi_i \tag{3.3}$$

$$\text{s.t.} \quad \boldsymbol{y}^\top \boldsymbol{X} \boldsymbol{\beta} + \xi_i \geq \rho, \quad \xi_i \geq 0, \quad i = 1, \ldots, n \tag{3.4}$$

$$\sum_{p \in \mathcal{P}} \beta_p = 1, \quad \beta_p \geq 0,$$

where $\boldsymbol{\xi}$ are slack variables, ρ is the margin separating negative examples from positives, $\lambda = \frac{1}{\nu n}$, $\nu \in (0, 1)$ is a parameter controlling the cost of misclassification which has to be found using model selection techniques, such as cross-validation. It is known that the optimal solution has the following ν-property:

Theorem 11.1 ([36]). *Assume that the solution of (3.3) satisfies $\rho \geq 0$. The following statements hold:*

1 *ν is an upper-bound of the fraction of* margin errors*, i.e., the examples with*

$$\boldsymbol{y}^\top \boldsymbol{X} \boldsymbol{\beta} < \rho.$$

2 ν is a lower-bound of the fraction of the examples such that

$$\boldsymbol{y}^\top \boldsymbol{X}\boldsymbol{\beta} < \rho.$$

Directly solving this optimization problem is intractable due to the large number of variables in $\boldsymbol{\beta}$. So we solve the following *equivalent* dual problem instead.

$$\begin{aligned} \min_{\boldsymbol{u},v} \quad & v && (3.5)\\ \text{s.t.} \quad & \sum_{i=1}^{n} u_i y_i x_{i,p} \le v, \ \forall p \in \mathcal{P} && (3.6)\\ & \sum_{i=1}^{n} u_i = 1, \quad 0 \le u_i \le \lambda, \ i = 1, \ldots, n. \end{aligned}$$

After solving the dual problem, the primal solution $\boldsymbol{\beta}$ is obtained from the Lagrange multipliers [8]. The dual problem has a limited number of variables, but a huge number of constraints. Such a linear program can be solved by the *column generation* technique [27]: Starting with an empty pattern set, the pattern whose corresponding constraint is violated the most is identified and added iteratively. Each time a pattern is added, the optimal solution is updated by solving the restricted dual problem. Denote by $\boldsymbol{u}^{(k)}, v^{(k)}$ the optimal solution of the restricted problem at iteration $k = 0, 1, \ldots$, and denote by $\hat{\boldsymbol{X}}^{(k)} \subseteq \mathcal{P}$ the set at iteration k. Initially, $\hat{\boldsymbol{X}}^{(0)}$ is empty and $u_i^{(0)} = 1/n$. The restricted problem is defined by replacing the set of constraints (3.6) with

$$\sum_{i=1}^{n} u_i^{(k)} y_i x_{i,p} \le v, \ \forall p \in \hat{\boldsymbol{X}}^{(k)}.$$

The left hand side of the inequality is called as *gain* in boosting literature. After solving the problem, $\hat{\boldsymbol{X}}^{(k)}$ is updated to $\hat{\boldsymbol{X}}^{(k+1)}$ by adding a column. Several criteria have been proposed to select the new columns [10], but we adopt the most simple rule that is amenable to graph mining: We select the constraint with the largest gain.

$$p^* = \operatorname*{argmax}_{p \in \mathcal{P}} \sum_{i=1}^{n} u_i^{(k)} y_i x_{i,p}. \quad (3.7)$$

The solution set is updated as $\hat{\boldsymbol{X}}^{(k+1)} \leftarrow \hat{\boldsymbol{X}}^{(k)} \cup \boldsymbol{X}_{j^*}$. In the next section, we discuss how to efficiently find the largest gain in detail.

One of the big advantages of our method is that we have a stopping criterion that guarantees that the optimal solution is found: If there is no $p \in \mathcal{P}$ such

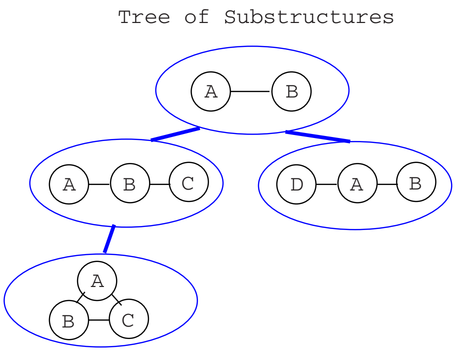


Figure 11.7. Schematic figure of the tree-shaped search space of graph patterns (i.e., the DFS code tree). To find the optimal pattern efficiently, the tree is systematically expanded by rightmost extensions.

that

$$\sum_{i=1}^{n} u_i^{(k)} y_i x_{i,p} > v^{(k)}, \tag{3.8}$$

then the current solution is the optimal dual solution. Empirically, the patterns found in the last few iterations have negligibly small weights. The number of iterations can be decreased by relaxing the condition as

$$\sum_{i=1}^{n} u_i^{(k)} y_i x_{i,p} > v^{(k)} + \epsilon, \tag{3.9}$$

Let us define the primal objective function as $V = -\rho + \lambda \sum_{i=1}^{n} \xi_i$. Due to the convex duality, we can guarantee that, for the solution obtained from the early termination (3.9), the objective satisfies $V \leq V^* + \epsilon$, where V^* is the optimal value with the exact termination (3.8) [8]. In our experiments, $\epsilon = 0.01$ is always used.

3.2 Optimal Pattern Search

Our search strategy is a branch-and-bound algorithm that requires a canonical search space in which a whole set of patterns are enumerated without duplication. As the search space, we adopt the DFS (depth first search) code tree [52]. The basic idea of the DFS code tree is to organize patterns as a tree, where a child node has a super graph of the parent's pattern (Figure 11.7). A pattern is represented as a text string called the DFS code. The patterns are enumerated by generating the tree from the root to leaves using a recursive algorithm. To avoid duplications, node generation is systematically done by rightmost extensions.

All embeddings of a pattern in the graphs $\{G_i\}_{i=1}^n$ are maintained in each node. If a pattern matches a graph in different ways, all such embeddings are stored. When a new pattern is created by adding an edge, it is not necessary to perform full isomorphism checks with respect to all graphs in the database. A new list of embeddings are made by extending the embeddings of the parent [52]. Technically, it is necessary to devise a data structure such that the embeddings are stored incrementally, because it takes a prohibitive amount of memory to keep all embeddings independently in each node. As mentioned in (3.7), our aim is to find the optimal hypothesis that maximizes the gain $g(p)$.

$$g(p) = \sum_{i=1}^{n} u_i^{(k)} y_i x_{i,p}. \tag{3.10}$$

For efficient search, it is important to minimize the size of the actual search space. To this aim, *tree pruning* is crucially important: Suppose the search tree is generated up to the pattern p and denote by g^* the maximum gain among the ones observed so far. If it is guaranteed that the gain of any super graph p' is not larger than g^*, we can avoid the generation of downstream nodes without losing the optimal pattern. We employ the following pruning condition.

Theorem 11.2. *[30, 26] Let us define*

$$\mu(p) = 2 \sum_{\{i|y_i=+1, p\subseteq G_i\}} u_i^{(k)} - \sum_{i=1}^{n} y_i u_i^{(k)}.$$

If the following condition is satisfied,

$$g^* > \mu(p), \tag{3.11}$$

the inequality $g(p') < g^*$ holds for any p' such that $p \subseteq p'$.

The gBoost algorithm is summarized in Algorithms 12 and 13.

3.3 Computational Experiments

In [40], it is shown that graph boosting performs better than graph kernels in classification accuracy in chemical compound datasets. The top 20 discriminative subgraphs for a mutagenicity dataset called CPDB are displayed in Figure 11.8. We found that the top 3 substructures with positive weights (0.0672,0.0656, 0.0577) correspond to known *toxicophores* [23]. They correspond to *aromatic amine*, *aliphatic halide*, and *three-membered heterocycle*, respectively. In addition, the patterns with weights 0.0431, 0.0412, 0.0411 and 0.0318 seem to be related to *polycyclic aromatic systems*. Only from this result, we cannot conclude that graph boosting is better in general data. However, since important chemical substructures cannot be represented in paths, it would be reasonable to say that subgraph features are better in chemical data.

Algorithm 12 gBoost algorithm: main part

```
1: $\hat{\boldsymbol{X}}^{(0)} = \emptyset$, $\boldsymbol{u}_i^{(0)} = 1/n$, $k = 0$
2: loop
3:     Find the optimal pattern $p^*$ based on $\boldsymbol{u}^{(k)}$
4:     if termination condition (3.9) holds then
5:         break
6:     end if
7:     $\hat{\boldsymbol{X}} \leftarrow \hat{\boldsymbol{X}} \cup \boldsymbol{X}_{j^*}$
8:     Solve the restricted dual problem (3.5) to obtain $\boldsymbol{u}^{(k+1)}$
9:     $k = k + 1$
10: end loop
```

Algorithm 13 Finding the Optimal Pattern

```
1: Procedure OPTIMAL PATTERN
2: Global variables: $g^*, p^*$
3: $g^* = -\infty$
4: for $p \in$ DFS codes with single nodes do
5:     project($p$)
6: end for
7: return $p^*$
8: EndProcedure
9:
10: Function PROJECT($p$)
11: if $p$ is not a minimum DFS code then
12:     return
13: end if
14: if pruning condition (3.11) holds then
15:     return
16: end if
17: if $g(p) > g^*$ then
18:     $g^* = g(p)$, $p^* = p$
19: end if
20: for $p' \in$ rightmost extensions of $p$ do
21:     project($p'$)
22: end for
23: EndFunction
```

3.4 Related Work

Graph algorithms can be designed based on existing statistical frameworks (i.e., mother algorithms). It allows us to use theoretical results and insights

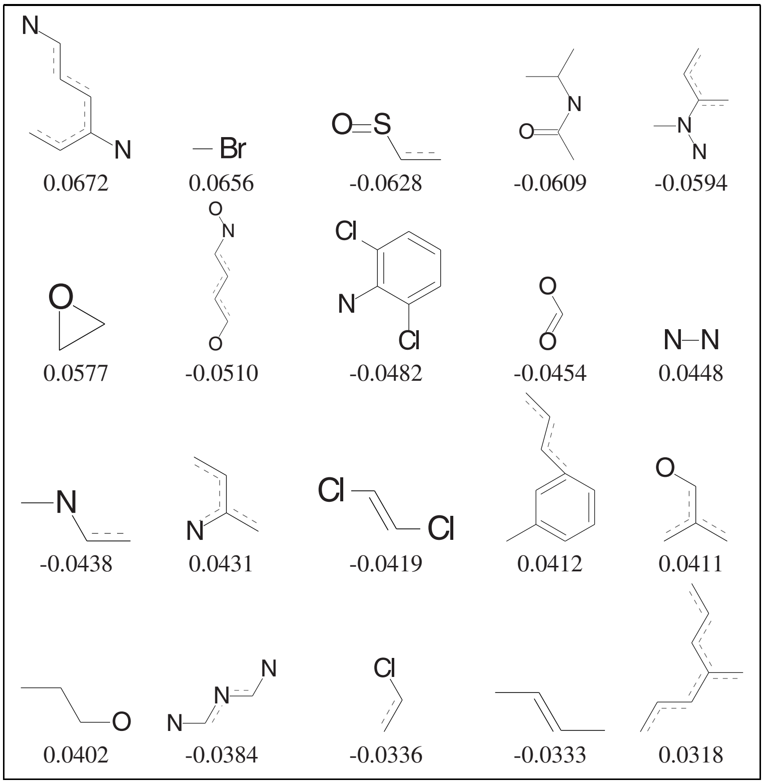


Figure 11.8. Top 20 discriminative subgraphs from the CPDB dataset. Each subgraph is shown with the corresponding weight, and ordered by the absolute value from the top left to the bottom right. H atom is omitted, and C atom is represented as a dot for simplicity. Aromatic bonds appeared in an open form are displayed by the combination of dashed and solid lines.

accumulated in the past studies. In graph boosting, we employed LPboost as a mother algorithm. It is possible to employ other algorithms such as partial least squares regression (PLS) [39] and least angle regression (LARS) [45].

When applied to ordinary vectorial data, partial least squares regression extracts a few orthogonal features and perform least squares regression in the projected space [37]. A PLS feature is a linear combination of original features, and it is often the case that correlated features are summarized into a PLS feature. Sometimes, the subgraph features chosen by graph boosting is not robust against bootstrapping or other data perturbations, whereas the classification accuracy is quite stable. It is due to strong correlation among features corresponding to similar subgraphs. The graph mining version of PLS, gPLS [39], solves this problem by summarizing similar subgraphs into each feature (Figure 11.9). Since only one graph mining call is required to construct each

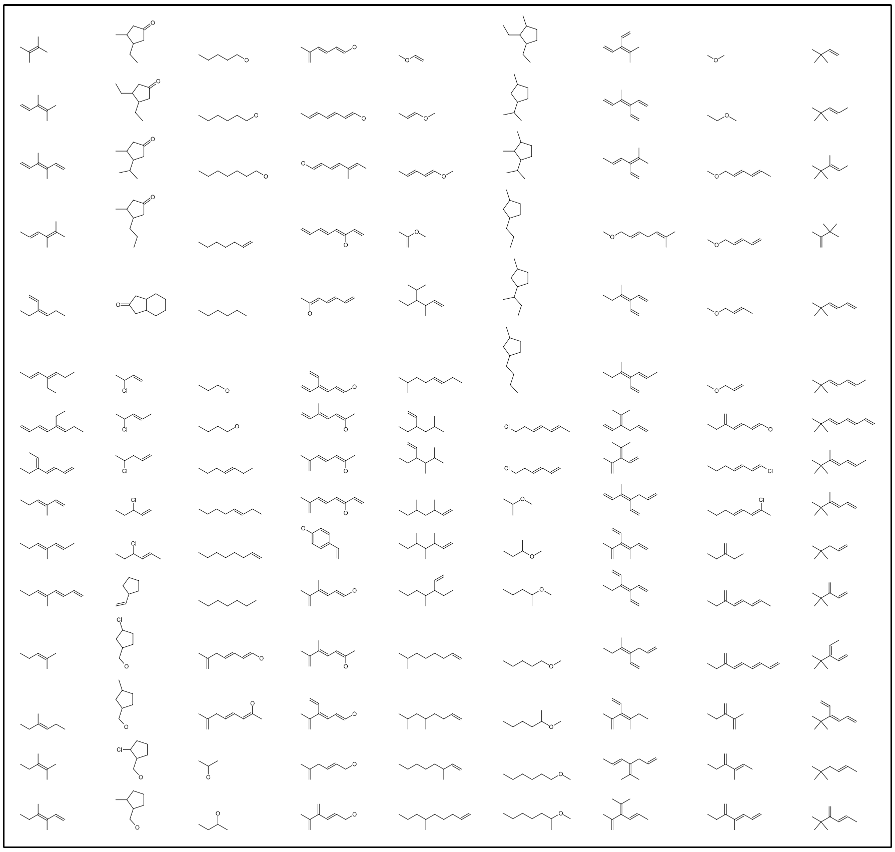


Figure 11.9. Patterns obtained by gPLS. Each column corresponds to the patterns of a PLS component.

feature, gPLS can build the classification rule more quickly than graph boosting.

In graph boosting, it is necessary to set the regularization parameter λ in (3.2). Typically it is determined by cross validation, but there is a different approach called "regularization path tracking". When $\lambda = 0$, the weight vector converges to the origin. As λ is increased continuously, the weight vector draws a piecewise linear path. Because of this property, one can track the whole path by repeating to jump to the next turning point. We combined the tracking with graph mining in [45]. In ordinary tracking, a feature is added or removed at each turning point. In our graph version, a subgraph to add or remove is found by a customized gSpan search.

The examples shown above were for supervised classification. For unsupervised clustering of graphs, the combinations with the EM algorithm [46] and the Dirichlet process [47] have been reported.

4. Applications of Graph Classification

Borgwardt et al. [5] applied the graph kernel method to classify protein 3D structures. It outperformed classical alignment-based approaches. Karklin et al. [19] built a classifier for non-coding RNAs employing a graph representation of RNAs. Outside biology and chemistry, Harchaoui and Bach [15] applied graph kernels to image classification where each region corresponds to a node and their positional relationships are represented by edges.

Traditionally, graph mining methods are mainly used for small chemical compounds [28, 9]. However, new application areas are emerging. In image processing [34], geometric relationships between points are represented as edges. Software bug detection is an interesting area, where the relationships of APIs are represented as directed graphs and anomalous patterns are detected to identify bugs [11]. In natural language processing, the relationships between words are represented as a graph (e.g., predicate-argument structures) and key phrases are identified as subgraphs [26].

5. Label Propagation

In the previous discussion, the term graph classification means classifying an entire graph. In many applications, we are interested in classifying the nodes. For example, in large-scale network analysis for social networks and biological networks, it is a central task to classify unlabeled nodes given a limited number of labeled nodes (Figure 11.1, right). In FaceBook, one can label people who responded to a certain advertisement as positive nodes, and people who did not respond as negative nodes. Based on these labeled nodes, our task is to predict other people's response to the advertisement.

In earlier studies, diffusion kernels are used in combination with support vector machines [25, 48]. The basic idea is to compute the closeness between two nodes in terms of commute time of random walks between the nodes. Though this approach gained popularity in the machine learning community, a significant drawback is that the derived kernel matrix is dense. For large networks, the diffusion kernel is not suitable because it takes $O(n^3)$ time and $O(n^2)$ memory. In contrast, label propagation methods use simpler computational strategies that exploit sparsity of the adjacency matrix [54, 53]. The label propagation method of Zhou et al.[53] is achieved by solving simultaneous linear equations with a sparse coefficient matrix. The time complexity is nearly linear to the number of non-zero entries of the coefficient matrix [49], which is much more efficient than the diffusion kernels. Due to its efficiency, label propagation is gaining popularity in applications with biological networks, where web servers should return the propagation result without much delay [32]. However, the classification performance is quite sensitive to methodological details. For example, Shin et al. pointed out that the introduction of directional propagation can increase the performance significantly [43]. Also, Mostafavi et al. [32] reported that their engineered version has outperformed the vanilla version [53]. Label propagation is still an active research field. Recent extensions include automatic combination of multiple networks [49, 22] and the introduction of probabilistic inference in label propagation [54, 44].

6. Concluding Remarks

We have covered the two different methods for graph classification. Graph kernel is a similarity measure between two graphs, while graph mining methods can derive characteristic subgraphs that can be used for any subsequent machine learning algorithms. We have the impression that so far graph kernels are more frequently applied. Probably it is due to the fact that graph kernels are easier to implement and currently used graph datasets are not so large. However, graph kernels are not suitable for very large data, because it takes $O(n^2)$ time to derive the kernel matrix of n training graphs, which is very hard to improve. Toward large scale data, graph mining methods seem more promising because it requires only $O(n)$ time. Nevertheless, there remains much to be done in graph mining methods. Existing methods such as gSpan enumerate all subgraphs satisfying a certain frequency-based criterion. However, it is often pointed out that, for graph classification, it is not always necessary to enumerate all subgraphs. Recently, Boley and Grosskreutz proposed a uniform sampling method of frequent itemsets [4]. Such theoretically guaranteed sampling procedures will certainly contribute to graph classification as well.

One fact that hinders the further popularity of graph mining methods is that it is not common to make the code public in the machine learning and data mining community. We have made several easy-to-use code available: SPIDER (`http://www.kyb.tuebingen.mpg.de/bs/people/spider/`) contains codes for graph kernels and the gBoost package contains codes for graph mining and boosting (`http://www.kyb.mpg.de/bs/people/nowozin/gboost/`).

References

[1] R. Agrawal and R. Srikant. Fast algorithms for mining association rules in large databases. In *Proc. VLDB 1994*, pages 487–499, 1994.

[2] T. Asai, K. Abe, S. Kawasoe, H. Arimura, H. Sakamoto, and S. Arikawa. Efficient substructure discovery from large semi-structured data. In *Proc 2nd SIAM Data Mining Conference (SDM)*, pages 158–174, 2002.

[3] R. Barrett, M. Berry, T. F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H. Van der Vorst. *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods, 2nd Edition.*

SIAM, Philadelphia, PA, 1994.

[4] M. Boley and H. Grosskreutz. A randomized approach for approximating the number of frequent sets. In *Proceedings of the 8th IEEE International Conference on Data Mining*, pages 43–52, 2008.

[5] K. M. Borgwardt, C. S. Ong, S. Schөnauer, S. V. N. Vishwanathan, A. J. Smola, and H.-P. Kriegel. Protein function prediction via graph kernels. *Bioinformatics*, 21(suppl. 1):i47–i56, 2006.

[6] O. Chapelle, A. Zien, and B. Schөlkopf, editors. *Semi-Supervised Learning*. MIT Press, Cambridge, MA, 2006.

[7] T. Cormen, C. Leiserson, and R. Rivest. Introduction to Algorithms. MIT Press and McGraw Hill, 1990.

[8] A. Demiriz, K.P. Bennet, and J. Shawe-Taylor. Linear programming boosting via column generation. *Machine Learning*, 46(1-3):225–254, 2002.

[9] M. Deshpande, M. Kuramochi, N. Wale, and G. Karypis. Frequent substructure-based approaches for classifying chemical compounds. *IEEE Trans. Knowl. Data Eng.*, 17(8):1036–1050, 2005.

[10] O. du Merle, D. Villeneuve, J. Desrosiers, and P. Hansen. Stabilized column generation. *Discrete Mathematics*, 194:229–237, 1999.

[11] F. Eichinger, K. Bөhm, and M. Huber. Mining edge-weighted call graphs to localise software bugs. In *Proceedings of the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML PKDD)*, pages 333–348, 2008.

[12] T. Gartner, P. Flach, and S. Wrobel. On graph kernels: Hardness results and efficient alternatives. In *Proc. of the Sixteenth Annual Conference on Computational Learning Theory*, 2003.

[13] I. Guyon, J. Weston, S. Bahnhill, and V. Vapnik. Gene selection for cancer classification using support vector machines. *Machine Learning*, 46(1-3):389–422, 2002.

[14] J. Han and M. Kamber. *Data Mining: Concepts and Techniques*. Morgan Kaufmann, 2000.

[15] Z. Harchaoui and F. Bach. Image classification with segmentation graph kernels. In *2007 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. IEEE Computer Society, 2007.

[16] C. Helma, T. Cramer, S. Kramer, and L.D. Raedt. Data mining and machine learning techniques for the identification of mutagenicity inducing substructures and structure activity relationships of noncongeneric compounds. *J. Chem. Inf. Comput. Sci.*, 44:1402–1411, 2004.

[17] T. Horvath, T. Gartner, and S. Wrobel. Cyclic pattern kernels for predictive graph mining. In *Proceedings of the 10th ACM SIGKDD International*

Conference on Knowledge Discovery and Data Mining, pages 158–167, 2004.

[18] A. Inokuchi. Mining generalized substructures from a set of labeled graphs. In *Proceedings of the 4th IEEE Internatinal Conference on Data Mining*, pages 415–418. IEEE Computer Society, 2005.

[19] Y. Karklin, R.F. Meraz, and S.R. Holbrook. Classification of non-coding rna using graph representations of secondary structure. In *Pacific Symposium on Biocomputing*, pages 4–15, 2005.

[20] H. Kashima, T. Kato, Y. Yamanishi, M. Sugiyama, and K. Tsuda. Link propagation: A fast semi-supervised learning algorithm for link prediction. In *2009 SIAM Conference on Data Mining*, pages 1100–1111, 2009.

[21] H. Kashima, K. Tsuda, and A. Inokuchi. Marginalized kernels between labeled graphs. In *Proceedings of the 21st International Conference on Machine Learning*, pages 321–328. AAAI Press, 2003.

[22] T. Kato, H. Kashima, and M. Sugiyama. Robust label propagation on multiple networks. *IEEE Trans. Neural Networks*, 20(1):35–44, 2008.

[23] J. Kazius, S. Nijssen, J. Kok, T. Back, and A.P. Ijzerman. Substructure mining using elaborate chemical representation. *J. Chem. Inf. Model.*, 46:597–605, 2006.

[24] R. Kohavi and G. H. John. Wrappers for feature subset selection. *Artificial Intelligence*, 1-2:273–324, 1997.

[25] R. I. Kondor and J. Lafferty. Diffusion kernels on graphs and other discrete input. In *ICML 2002*, 2002.

[26] T. Kudo, E. Maeda, and Y. Matsumoto. An application of boosting to graph classification. In *Advances in Neural Information Processing Systems 17*, pages 729–736. MIT Press, 2005.

[27] D. G. Luenberger. *Optimization by Vector Space Methods*. Wiley, 1969.

[28] P. Mahe, N. Ueda, T. Akutsu, J.-L. Perret, and J.-P. Vert. Graph kernels for molecular structure - activity relationship analysis with support vector machines. *J. Chem. Inf. Model.*, 45:939–951, 2005.

[29] P. Mahe and J.P. Vert. Graph kernels based on tree patterns for molecules. *Machine Learning*, 75:3–35, 2009.

[30] S. Morishita. Computing optimal hypotheses efficiently for boosting. In *Discovery Science*, pages 471–481, 2001.

[31] S. Morishita and J. Sese. Traversing itemset lattices with statistical metric pruning. In *Proceedings of ACM SIGACT-SIGMOD-SIGART Symposium on Database Systems (PODS)*, pages 226–236, 2000.

[32] S. Mostafavi, D. Ray, D. Warde-Farley, C. Grouios, and Q. Morris. GeneMANIA: a real-time multiple association network integration algorithm for predicting gene function. *Genome Biology*, 9(Suppl. 1):S4, 2008.

[33] S. Nijssen and J.N. Kok. A quickstart in frequent structure mining can make a difference. In *Proceedings of the 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 647–652. ACM Press, 2004.

[34] S. Nowozin, K. Tsuda, T. Uno, T. Kudo, and G. Bakir. Weighted substructure mining for image analysis. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE Computer Society, 2007.

[35] J. Pei, J. Han, B. Mortazavi-asl, J. Wang, H. Pinto, Q. Chen, U. Dayal, and M. Hsu. Mining sequential patterns by pattern-growth: The prefixspan approach. *IEEE Transactions on Knowledge and Data Engineering*, 16(11):1424–1440, 2004.

[36] G. Ratsch, S. Mika, B. Schelkopf, and K.-R. Muller. Constructing boosting algorithms from SVMs: an application to one-class classification. *IEEE Trans. Patt. Anal. Mach. Intell.*, 24(9):1184–1199, 2002.

[37] R. Rosipal and N. Kramer. Overview and recent advances in partial least squares. In *Subspace, Latent Structure and Feature Selection Techniques*, pages 34–51. Springer, 2006.

[38] W.J. Rugh. *Linear System Theory*. Prentice Hall, 1995.

[39] H. Saigo, N. Kramer, and K. Tsuda. Partial least squares regression for graph mining. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 578–586, 2008.

[40] H. Saigo, S. Nowozin, T. Kadowaki, T. Kudo, and K. Tsuda. GBoost: A mathematical programming approach to graph classification and regression. *Machine Learning*, 2008.

[41] A. Sanfeliu and K.S. Fu. A distance measure between attributed relational graphs for pattern recognition. *IEEE Trans. Syst. Man Cybern.*, 13:353–362, 1983.

[42] B. Schelkopf and A. J. Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. MIT Press, 2002.

[43] H. Shin, A.M. Lisewski, and O. Lichtarge. Graph sharpening plus graph integration: a synergy that improves protein functional classification. *Bioinformatics*, 23:3217–3224, 2007.

[44] A. Subramanya and J. Bilmes. Soft-supervised learning for text classification. In *Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing*, pages 1090–1099, 2008.

[45] K. Tsuda. Entire regularization paths for graph data. In *Proceedings of the 24th International Conference on Machine Learning*, pages 919–926, 2007.

[46] K. Tsuda and T. Kudo. Clustering graphs by weighted substructure mining. In *Proceedings of the 23rd International Conference on Machine Learning*, pages 953–960. ACM Press, 2006.

[47] K. Tsuda and K. Kurihara. Graph mining with variational dirichlet process mixture models. In *SIAM Conference on Data Mining (SDM)*, 2008.

[48] K. Tsuda and W.S. Noble. Learning kernels from biological networks by maximizing entropy. *Bioinformatics*, 20(Suppl. 1):i326–i333, 2004.

[49] K. Tsuda, H.J. Shin, and B. Schølkopf. Fast protein classification with multiple networks. *Bioinformatics*, 21(Suppl. 2):ii59–ii65, 2005.

[50] S.V.N. Vishwanathan, K.M. Borgwardt, and N.N. Schraudolph. Fast computation of graph kernels. In *Advances in Neural Information Processing Systems 19*, Cambridge, MA, 2006. MIT Press.

[51] N. Wale and G. Karypis. Comparison of descriptor spaces for chemical compound retrieval and classification. In *Proceedings of the 2006 IEEE International Conference on Data Mining*, pages 678–689, 2006.

[52] X. Yan and J. Han. gSpan: graph-based substructure pattern mining. In *Proceedings of the 2002 IEEE International Conference on Data Mining*, pages 721–724. IEEE Computer Society, 2002.

[53] D. Zhou, O. Bousquet, J. Weston, and B. Schølkopf. Learning with local and global consistency. In *Advances in Neural Information Processing Systems (NIPS) 16*, pages 321–328. MIT Press, 2004.

[54] X. Zhu, Z. Ghahramani, and J. Lafferty. Semi-supervised learning using gaussian fields and harmonic functions. In *Proc. of the Twentieth International Conference on Machine Learning (ICML)*, pages 912–919. AAAI Press, 2003.