Kernels for Graph Classification

Hisashi Kashima and Akihiro Inokuchi
IBM Research, Tokyo Research Laboratory
1623-14, Shimotsuruma, Yamato-shi, Kanagawa 242-8502, Japan
{hkashima, Inokouchi}@jp.ibm.com

Abstract

In this paper, we apply kernel methods to graph classification problems. To achieve the goal, we have to design an appropriate kernel for computing inner products for pairs of graphs represented in a feature space. We define a graph kernel by a random walk on a vertex product graph of two graphs. Some experiments on predicting properties of chemical compounds show encouraging results.

1 Introduction

Recently, it is needed to develop various kinds of data mining methods that can handle structural data. As semi-structured data such as XML and HTML are increasing, data mining methods that can handle not only relational data, but also for semi-structured data are attracting considerable attention. In pharmaceutical area, it is valuable for rationalization of drug discovery processes to predict the effectiveness or toxicity of drugs from their chemical structures since we can evaluate candidate compounds before we synthesize them.

In this paper, we aim to develop solutions to classification problems of graphs with vertex labels and edge labels (Figure 1). For example, semi-structured data and chemical compounds stated above can be represented as such graphs naturally.

In general learning problems, objects are represented as vectors in a feature space, and training classifiers is reduced to deciding on rules to separate vectors that belong to positive examples from vectors that belong to negative examples. However, when we handle more complex objects such as sequences, trees, and graphs that have structures among their constituent elements, design of a suitable feature space is not trivial. Probably, one of the sound strategies for handling such complex objects is to use local structures in them as features. However, in most cases, considering all possible local structures as features is inhibitive since it often leads to combinatorial explosion. Therefore, some mechanism is needed to select a subset of local structures that can contribute to classification. Relational learning [13] is a general method that can handle local structures in objects. In relational learning, several relationships among constituent elements are defined, and the relationships constitute local structures. The local structures used as features are incrementally built up in the process of training. However, since the problem of finding the best hypothesis is generally NP-hard, we must use heuristic methods. Another method is based on pattern discovery algorithms that find local structures appearing frequently [11, 6], and these structures are used as features. The pattern-discovery-based method has an advantage in that it can make use of unlabelled data. However, the process of discovering patterns is again almost always NP-hard.

Yet another approach is to use kernel methods such as support vector machines (SVMs) [15]. One of the important properties of kernel methods is their access to examples via kernels. In kernel methods, examples are mapped into a feature space implicitly, and only the inner products of the vector representations are used when learning machines access the examples. This means that even in cases where the dimension of the vector representations is extremely high, the dimensions do not explicitly appear in the process of training and classification as long as an efficient procedure to compute the inner products is available. The function giving the inner products is called the 'kernel', and kernel methods can work efficiently in high dimensional feature spaces by using kernels. Moreover, SVMs are known to have good generalization properties, both theoretically and experimentally, and overcome the 'curse of dimensionality' problem in high dimensional feature spaces [15].

Now, our task is to design suitable kernels that can classify structural objects, and that can be computed efficiently. We need a kernel function $K(G_1, G_2)$ that can be efficiently computed the inner product of two vectors represent two graphs G_1 and G_2 in a suitably defined feature space where graphs can be classified. There are several works that aim at classification of structural objects. Haussler [4] introduced 'convolution kernels', a general framework for han-

dling discrete data structures by kernel methods. In the context of the convolution kernels, Watkins [16] and Leslie et al. [12] proposed kernels for strings, and Collins et al. [1] and Kashima et. al. [9] proposed kernels for trees. Besides, Jaakkola et al. [7] proposed Fisher kernels that define kernels using given probabilistic models, and apply them to classification of protein sequences. Of special interest here, Kandola et al. [8] proposed diffusion kernels that define kernels when input spaces are represented as undirected graphs. They employed the idea of diffusion over given graphs to define similarity between arbitrary two vertices. Kondor et al. [10] applied this idea to document classification, where a document corresponds to a vertex. In diffusion kernels, graphs represent the structures of the input spaces, and the vertices are the objects to be classified, while in this paper, our aim is to classify graphs themselves.

To define a kernel between arbitrary two graphs, we use a random walk on the vertex product graph of the two graphs. Precisely, the kernel is defined to be the probability with which two label sequences generated by two 'synchronized' random walks on the graphs are identical. In the feature space, each feature of the vector representation of a graph corresponds to a particular label path that can possibly be generated by a random walk on the graph. Although it is inhibitive to compute inner products explicitly since the number of possible paths is exponentially large, we show simultaneous linear equations to compute them. Therefore, we can compute the kernels efficiently by methods such as iterative methods.

Our kernel is closely related to the diffusion kernels, and we can show their structures are similar. However, while diffusion kernels are defined by a symmetric adjacent matrix which represents the input space graph, our kernel is defined by an asymmetric matrix which represents the vertex product graph of two graphs.

Finally, we perform some experiments on predicting properties of chemical compounds to investigate how our kernel performs well on real data, and the results show encouraging results.

This paper is organized as follows. In Section 2, we define our task, and introduce the idea of kernel methods. In Section 3, we propose a new kernel for graph classification. In Section 4, we summarize the results of our experiments on classification of chemical compounds. We conclude with Section 5 in which we provide a summary and duscussion.

2 Graph Classification Problems and Kernel Methods

In this section, we define the graph classification problem, and introduce the idea of the kernel methods. We define a graph classification problem as the followings. A learning machine receives a set of N training examples

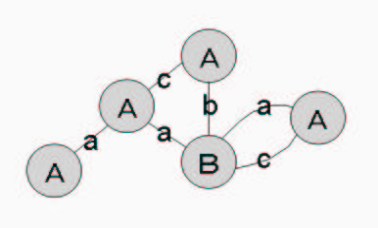

Figure 1. an example of graphs with labels

$\{(G_1, y_1), (G_2, y_2), \ldots, (G_N, y_N), \}$ where each example (G_i, y_i) is given as a pair of a graph $G_i = (V_i, E_i)$ and the class $y_i \in \{+1, -1\}$ that the graph belongs to. We assume that each vertex $v \in V_i$ is labeled by one of the possible vertex labels $\Sigma_V = \{\sigma_{V1}, \sigma_{V2}, \ldots\}$, and each edge is labeled by one of the possible edge labels in Σ_E. Figure 1 shows an example of graphs that we treat in this paper. The objective of the learning machine is to correctly predict the classes of test examples whose classes are unknown.

In this paper, we employ kernel methods for this task. One of the important properties of kernel methods is their access to examples via kernels. In kernel methods, examples are mapped into a feature space implicitly, and only the inner products of the vector representations are used when learning machines access the examples. For example, in the support vector machine that is a well-known kernel learning algorithm, training a classifier is reduced to the following quadratic programming problem,

$$\begin{aligned} \underset{\alpha_1,\ldots,\alpha_N}{\text{maximize}} \quad & \sum_{i=1}^{N} \alpha_i - \sum_{i=1}^{N}\sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j < \mathbf{x}_i, \mathbf{x}_j > \\ \text{s.t.} \quad & \alpha \geq 0 \\ & \sum_{i=1}^{N} \alpha_i y_i = 0 \end{aligned} \tag{1}$$

where $\mathbf{x}_i$ is the vector representation of the i-th training example. A text example $\mathbf{x}$ is classified by

$$f(\mathbf{x}) = sgn\left(\sum_{i=1}^{N} \alpha_i y_i < \mathbf{x}, \mathbf{x}_i > +b\right). \tag{2}$$

We can see all the accesses to the examples are done by inner products. This means that even in cases where the dimension of the vector representations is extremely large, the dimension does not explicitly appear in the process of training and classification as long as an efficient procedure to compute the inner products is available. The function giving the inner products is called 'kernel', and kernel methods can work efficiently in high dimensional feature spaces by using kernels. Therefore, now, our task is to design a suitable feature space where graphs can be classified, and to

give a kernel function $K(G_1, G_2)$ that can efficiently compute the inner product of the two vector representations of two graphs G_1 and G_2.

3 Kernels for Graph Classification

3.1 Graph Kernels

Probably, the most simplest way of defining a vector representation X_G of a graph $G = (V, E)$ is to define each element of a vector using the number of times a particular vertex label appears in the graph.

$$\mathbf{x}_G = \left(\frac{\#(\sigma_{V1}, G)}{|V|}, \frac{\#(\sigma_{V2}, G)}{|V|}, \ldots, \frac{\#(\sigma_{V|\Sigma_V|}, G)}{|V|} \right) \quad (3)$$

where $\#(\sigma_{Vi}, G)$ is the number of times vertex label σ_{Vi} appears in graph G. This corresponds to the bag-of-words representation of a document which is usually used in information retrieval [2]. Suppose we want to calculate the kernel for a pair of graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$. Then the kernel is defined as

$$\begin{aligned} K(G_1, G_2) &:= \mathbf{x}_{G_1} \mathbf{x}_{G_2}^T & (4) \\ &= \frac{1}{|V_1||V_2|} \sum_{v_1 \in V_1} \sum_{v_2 \in V_2} k(v_1, v_2) & (5) \\ k(v_1, v_2) &:= I(v_1, v_2) & (6) \end{aligned}$$

where I is an indicator function that returns 1 when the labels of two arguments are identical, and returns 0 otherwise. The kernel for graphs can be considered to be decomposed in the kernels of pairs of vertices. The vertex-wise kernel of $k(v_1, v_2)$ checks if the labels of the vertices v_1 and v_2 are identical, and this can be seen as a kind of similarity between two vertices. However, $k(v_1, v_2)$ does not incorporate any local information around v_1 and v_2 at all, and therefore we modify $k(v_1, v_2)$ to consider the local structure of graphs. We redefine $k(v_1, v_2)$ so as to take higher scores when not only the labels of v_1 and v_2 are identical, but also the labels of the edges and vertices adjacent to v_1 and v_2, and the further edges and vertices are identical. Concretely, we redefine $k(v_1, v_2)$ as

$$\begin{aligned} & k(v_1, v_2) := (1-\lambda) \cdot k_0(v_1, v_2) & (7) \\ & + \lambda(1-\lambda) \sum_{\substack{e_1 \in A(v_1) \\ e_2 \in A(v_2)}} k_1(v_1, v_2, e_1, e_2) \\ & + \lambda^2(1-\lambda) \sum_{\substack{e_1 \in A(v_1) \\ e_2 \in A(v_2)}} \sum_{\substack{e'_1 \in A(\delta(v_1, e_1)) \\ e'_2 \in A(\delta(v_1, e_1))}} k_2(v_1, v_2, e_1, e_2, e'_1, e'_2) \\ & + \lambda^3(1-\lambda) \sum_{\substack{e_1 \in A(v_1) \\ e_2 \in A(v_2)}} \sum_{\substack{e'_1 \in A(\delta(v_1, e_1)) \\ e'_2 \in A(\delta(v_2, e_2))}} \sum_{\substack{e''_1 \in A(\delta(\delta(v_1, e_1), e'_1)) \\ e''_2 \in A(\delta(\delta(v_2, e_2), e'_2))}} \\ & \quad k_3(v_1, v_2, e_1, e_2, e'_1, e'_2, e''_1, e''_2) \\ & + \cdots \end{aligned}$$

$$\begin{aligned} & k_0(v_1, v_2) = I(v_1, v_2) & (8) \\ & k_1(v_1, v_2, e_1, e_2) = k_0(v_1, v_2) \\ & \qquad \cdot \frac{I(e_1, e_2) \cdot I(\delta(v_1, e_1), \delta(v_2, e_2))}{|A(v_1)||A(v_2)|} & (9) \\ & k_2(v_1, v_2, e_1, e_2, e'_1, e'_2) = k_1(v_1, v_2, e_1, e_2) \\ & \qquad \cdot \frac{I(e'_1, e'_2) \cdot I(\delta(\delta(v_1, e_1), e'_1), \delta(\delta(v_2, e_2), e'_2))}{|A(v'_1)||A(v'_2)|} & (10) \\ & k_3(v_1, v_2, e_1, e_2, e'_1, e'_2, e''_1, e''_2) = \\ & \qquad k_2(v_1, v_2, e_1, e_2, e'_1, e'_2) & (11) \\ & \cdot \frac{I(e''_1, e''_2) I(\delta(\delta(\delta(v_1, e_1), e'_1), e''_1), \delta(\delta(\delta(v_2, e_2), e'_2), e''_2))}{|A(v''_1)||A(v''_2)|} \\ & \ldots \end{aligned}$$

where $\lambda \in [0, 1]$ is a decaying constant, and $A(v)$ is a set of edges adjacent to v, and $\delta(v, e)$ is a transition function that returns the vertex at the other side of e adjacent to v. Note that Equation (7) is identical to Equation (6) when $\lambda = 0$. The new kernel $K(G_1, G_2)$ with modified $k(v_1, v_2)$ can be interpreted using a random walk on the vertex product graph $G_{1\times 2} = (V_1 \times V_2, E_{1\times 2} \subseteq E_1 \times E_2)$ of two graphs G_1 and G_2. Suppose that two 'synchronized' random walks are performed on G_1 and G_2 as the following. At first $v_1 \in V_1$ and $v_2 \in V_2$ are selected randomly as the starting points. At each round, both random walks are simultaneously halted with probability $1 - \lambda$, and continued with probability λ. If continued, in each graphs, a transition is made by randomly selecting an edge among the edges adjacent to the current vertex. When the random walks are halted, the trial succeeds if the two label sequences generated from two graphs are identical. $K(G_1, G_2)$ can be interpreted as the probability with that this trial succeeds.

From the viewpoint of constructing a feature space, each feature of graph G is the probability with which a particular label path is generated by a (single) random walk on G with halting probability λ.[1] Explicit Computation of Equation (7) is inhibitive because of the exponentially many label sequences. However, we can rewrite the equation as the following linear equations.

$$\begin{aligned} & k(v_1, v_2) = I(v_1, v_2) \{(1-\lambda) & (12) \\ & \quad + \lambda \sum_{\substack{e_1 \in A(v_1) \\ e_2 \in A(v_2)}} \frac{I(e_1, e_2)}{|A(v_1)||A(v_2)|} \cdot k(\delta(v_1, e_1), \delta(v_2, e_2)) \Bigg\} \end{aligned}$$

[1] For deriving Equations (7)-(11), each feature should be multiplied by $\sqrt{\frac{1-\lambda}{\lambda^i}}$ where i is the length of the random walk that generate the label sequence. However, these factors do not influence the kernel values after appropriate scaling.

3.2 Relation to Diffusion Kernels

In this subsection, we discuss the relationship between our kernel and von Neumann kernel proposed by Kandora et al. [8]. This kernel is a kind of diffusion kernels introduced by Kondor et al. [10], and Kandora et al. [8] applied the idea of diffusion kernels to document classification. They regard a document as a vertex in a graph, and represent the similarity between two documents as the weight of an edge. In other words, suppose that K is a symmetric adjacent matrix of the graph, the (i,j)-th element indicates the 'direct' similarity defined to be the inner product of the two bag-of-words vector representations of i-th document and the j-th document. Although K itself is a kernel matrix defined over the vertices as long as K is positive definite, they defined von Neumann kernel as

$$\begin{aligned} K_{vN} &:= L + \lambda K K_{vN} && (13)\\ &= K + \lambda K^2 + \lambda^2 K^3 + \cdots && (14)\\ &= (I - \lambda K)^{-1} K && (15) \end{aligned}$$

to incorporate the 'indirect' similarities. Intuitively, this kernel implements the idea that two documents are similar if both of them are similar to another document. The (i,j)-th element of K^d indicates the sum of the products of the edge weights in all possible paths of length d between the i-th vertex and the j-th vertex. Note that the paths can include a particular edge more than once.

Similarly, we can rewrite Equation (12) by matrices. Let $\mathbf{k}$ be a vector whose dimension is $|V_1| \cdot |V_2|$, and whose $i_{v_1 v_2}$-th element is $I(v_1, v_2)$ be $k(v_1, v_2)$ where $i_{v_1 v_2}$ is the index for (v_1, v_2). Similarly, let $\mathbf{k}_0$ be a vector whose $i_{v_1 v_2}$-th element is $I(v_1, v_2)$. Using the $|V_1| \cdot |V_2| \times |V_1| \cdot |V_2|$ matrix K' defined as

$$[K']_{i_{v_1 v_2}, i_{v_1' v_2'}} := \sum_{\substack{e_1 \in A(v_1) \\ \delta(v_1, e_1) = v_1'}} \sum_{\substack{e_2 \in A(v_2) \\ \delta(v_2, e_2) = v_2'}} \frac{I(e_1, e_2)}{|A(v_1)||A(v_2)|}, \tag{16}$$

we rewrite Equation (12) as

$$\begin{aligned} \mathbf{k} &= (1-\lambda)\mathbf{k}^0 + \lambda K' \mathbf{k} && (17)\\ &= (1-\lambda)(\mathbf{k}^0 + \lambda K' \mathbf{k}^0 + \lambda^2 K'^2 \mathbf{k}^0 + \cdots) && (18)\\ &= (1-\lambda)(I - \lambda K')^{-1} \mathbf{k}^0. && (19) \end{aligned}$$

Apparently, Equations (13)-(15) and (17)-(19) have a common structure, and both can be interpreted as a random walks on graphs. However, we point some differences between them. The diffusion kernels are defined over undirected graphs, that is, K is symmetric. On the other hand, our kernel is defined over directed graphs, that is, K' is asymmetric. Moreover, in diffusion kernels, a kernel function defines the similarity between an arbitrary pair of objects in the input space represented as a graph, while in our kernel, an object itself is a graph, and the similarity between two arbitrary pairs of vertices is defined over the vertex product graph $G_{1\times 2} = (V_1 \times V_2, E_{1\times 2} \subseteq E_1 \times E_2)$ of two graph G_1 and G_2. In other words, while the (i,j)-th element of $(I - \lambda K)^{-1} K$ indicates the similarity between the i-th vertex and the j-th vertex, the $(1-\lambda)(i_{v_1 v_2}, i_{v_1' v_2'})$-th element of $(I - \lambda K')^{-1}$ indicates the contribution from the similarity of vertex pair v_1' and v_2' to the similarity of vertex pair v_1 and v_2.

4 Experiments

In this section, we apply our kernel to prediction of the properties of chemical compounds. A chemical compound can be represented as a graph by considering the names of atoms as vertex labels, and the types of bonds as edge labels. We used two datasets, mutag dataset [14] and PTC dataset [5]. In mutag dataset, the task is to predict mutagenicity, and 188 compounds are included, and the maximum number of vertices is 40, and the average number of vertices is 31.4. In PTC dataset, the task is to predict carcinogenicity, and 417 compounds are included, and the maximum number of vertices is 109, and the average number of vertices is 25.7. Each compound in PTC dataset is given four classes, MM(Male Mouse), FM(Female Mouse), MR(Male Rat) and FR(Female Rat), each of which takes one of $\{\mathrm{EE}, \mathrm{IS}, \mathrm{E}, \mathrm{CE}, \mathrm{SE}, \mathrm{P}, \mathrm{NE}, \mathrm{N}\}$. Therefore, PTC provides four classification problems. We use $\{\mathrm{CE}, \mathrm{SE}, \mathrm{P}\}$ as positive class, and $\{\mathrm{NE}, \mathrm{N}\}$ as negative class. In both datasets, four types of bond are included.

We compare our kernel with a pattern discovery-based method that uses frequent substructures as features of vector representations. Pattern discovery algorithms [11, 6] find all substructure patterns that appear more frequently than a given threshold in a dataset. In this paper, we use a frequent path finding algorithm [11] for constructing feature spaces. Suppose that the pattern discovery algorithm finds m frequent paths $\{path_1, path_2, \ldots, path_m\}$ in the dataset. The vector representation of a graph G is defined as

$$\begin{aligned} V_G^{num} &= (num(path_1, G), num(path_2, G), \ldots, \\ &\quad num(path_m, G)) && (20) \end{aligned}$$

where $num(path_i, G)$ is the number of times $path_i$ appears in graph G. The inner product of two vector representation $V_{G_1}^{num}$ and $V_{G_2}^{num}$ of two graphs G_1 and G_2 is defined as the following.

$$K^{num}(G_1, G_2) = \sum_{i=1}^{m} num(path_i, G_1) \cdot num(path_i, G_2) \tag{21}$$

Another possible vector representation uses a binary function $bin(path_i, G)$ which returns 1 when $path_i$ appears in

G, and returns 0 otherwise.

$$K^{bin}(G_1, G_2) = \sum_{i=1}^{m} bin(path_i, G_1) \cdot bin(path_i, G_2) \tag{22}$$

Note that our kernel also assumes the similar feature space, however, our kernel allows to use an edge more than once when checking whether a certain path appears in a graph. This implies that our kernel counts the appearances of paths approximately.

Computing our kernel needs to solve simultaneous linear equations (18) with a $|V_1| \cdot |V_2| \times |V_1| \cdot |V_2|$ matrix. However, the matrix is sparse since the number of non-zero elements is $|V_1| \cdot |V_2| \cdot \max_{v_1 \in V_1} |A(v_1)| \cdot \max_{v_2 \in V_2} |A(v_2)|$, and we can employ various kinds of efficient numerical algorithms. In this experiment, we just use an iterative method using the recursive equations (12).

As for the learning algorithm, for ease of implementation, we used the voted kernel perceptron [3] whose performance is known to be comparable to that of SVMs.

Table 1 - Table 4 show the test accuracy measured by leave-one-out cross validation. 'num' and 'bin' indicate the results for frequent-path-based kernels (21) and (22) respectively. 'MinSup' is a parameter for the pattern discovery algorithm that decides the minimum support. Although our graph kernel is not as well as the frequent-path-based kernels for mutag dataset, it is comparable to the frequent-path-based kernels for PTC dataset.

5 Conclusion

In this paper, we applied kernel methods to classification of graphs with vertex labels and edge labels. We defined a graph kernel for a pair of graphs by a random walk on a vertex product graph of the two graphs. Concretely, the kernel was defined to be the probability with which two label sequences generated by two synchronized random walks on the graphs were identical.

Next, we performed some experiments on predicting properties of chemical compounds to investigate how our kernel performed well on real data, and the results showed encouraging results.

Our kernel approximately counts all the appearances of the paths included in a graph, and at the same time, the whole process avoids NP-hard steps. This implies our kernel may be effective for larger graphs that pattern discovery algorithms suffer from, and we plan to apply our kernel to such datasets.

In this paper, we applied our kernel only to undirected graphs, however, we can naturally treat directed graphs such as semi-structured data and WWW structure by incorporating the edge directions into the edge labels. We plan to apply our kernel to such various datasets to investigate the ability of our kernel further.

References

[1] M. Collins and N. Duffy. Convolution kernel for natural language. In *Proc. of the 14th NIPS*, 2001.

[2] W. Frakes and R. Baeza-Yates.(eds.). *Information Retrieval: Data Structures and Algorithms*. Prentice Hall, 1992.

[3] Y. Freund and R. Shapire. Large margin classification using the perceptron algorithm. *Machine Learning*, 37(3), 1999.

[4] D. Haussler. Convolution kernels on discrete structures. Technical Report UCSC-CRL-99-10, University of Calfornia in Santa Cruz, 1999.

[5] C. Helma, R. King, S. Kramer, and A. Srinivasan. The predictive toxicology challenge 2000-2001. *Bioinformatics*, 17(1):107–108, 2001.

[6] A. Inokuchi, T. Washio, and H. Motoda. An Apriori-based algorithm for mining frequent substructures from graph data. In *Proc. of the 4th PKDD*, pages 13–23, 2000.

[7] T. Jaakkola, M. Diekhans, and D. Haussler. A discriminative framework for detecting remote protein homologies. *Journal of Computational Biology*, 7(1,2):95–114, 2000.

[8] J. Kandola, J. Shawe-Taylor, and N. Cristianini. On the application of diffusion kernel to text data. Technical report, Neurocolt, 2002. NeuroCOLT Technical Report NC-TR-02-122.

[9] H. Kashima and T. Koyanagi. Kernels for semi-structured data. In *Proc. of the 19th ICML*, 2002.

[10] R. I. Kondor and J. Lafferty. Diffusion kernels on graphs and other discrete input spaces. In *Proc. of the 19th ICML*, 2002.

[11] S. Kramer and L. D. Raedt. Feture construction with version spaces for biochemical application. In *Proc. of the 18th ICML*, 2001.

[12] C. Leslie, E. Eskin, and W. S. Noble. The spectrum kernel: a string kernel for SVM protein classification. In *Proc. of PSB 2002*, pages 564–575, 2002.

[13] T. Mitchell. *Machine Learning.* McGraw-Hill, 1997.

[14] A. Srinivasan, S. Muggleton, R. D. King, and M. Sternberg. Theories for mutagenicity: a study of first-order and feature based induction. *Artificial Intelligence*, 85(1-2):277–299, 1996.

[15] V. Vapnik. *The Nature of Statistical Learning Theory*. Springer Verlag, 1995.

[16] C. Watkins. Kernels from matching operations. Technical Report CSD-TR-98-07, University of London, Computer Science Department, Royal Holloway, 1999.

MinSup	bin	num
0.5 %	89.4%	88.3%
1 %	88.3%	87.8%
3 %	**89.9%**	**89.9%**
5 %	89.4%	86.2%
10 %	84.0%	84.6%
20 %	85.1%	83.5%

Table 1. Result for mutag (frequent-path-based kernel)

λ	Graph Kernel
0.1	78.7%
0.2	79.8%
0.3	81.9%
0.4	83.0%
0.5	83.5%
0.6	**85.1%**
0.7	**85.1%**
0.8	83.5%
0.9	84.e%

Table 2. Result for mutag (graph kernel)

	MM		FM		MR		FR	
MinSup	bin	num	bin	num	bin	num	bin	num
0.5%	61.0%	60.1%	57.3%	57.6%	59.0%	61.3%	63.8%	**66.7%**
1 %	59.8%	**61.0%**	59.0%	**61.0%**	59.3%	**62.8%**	**64.7%**	63.2%
3 %	59.2%	58.3%	59.6%	55.9%	57.8%	60.2%	63.2%	63.2%
5 %	56.8%	60.7%	58.2%	55.6%	55.5%	57.3%	64.1%	63.0%
10 %	57.4%	58.9%	**61.0%**	58.7%	58.4%	57.8%	60.1%	60.1%
20%	**61.6**%	**61.0%**	57.0%	55.3%	**60.2%**	56.1%	60.7%	61.3%

Table 3. Result for PTC (frequent-path-based kernel)

λ	MM	FM	MR	FR
0.1	62.8%	61.6%	**58.4%**	**66.1%**
0.2	63.4%	**63.4%**	54.9%	64.1%
0.3	63.1%	62.5%	54.1%	63.2%
0.4	62.8%	61.9%	54.4%	65.8%
0.5	64.0%	61.3%	56.1%	64.4%
0.6	**64.3%**	61.9%	56.1%	63.0%
0.7	64.0%	61.3%	56.7%	62.1%
0.8	62.2%	61.0%	57.0%	62.4%
0.9	62.2%	59.3%	57.0%	62.1%

Table 4. Result for PTC (graph kernel)