Machine Learning Approaches for Structured Data

構造データ解析のための機械学習法

Hisashi Kashima

Dissertation submitted to the Graduate School of Informatics

Kyoto University

Thesis Supervisor: Tatsuya Akutsu

ABSTRACT

Fast evolving Internet technologies including the WWW, Web Services, and blogs can hardly be discussed without referring to structured data such as HTML and XML. Such data has functions to link to each other, and meta-structures such as link structures emerge from those links. Meanwhile, the entire genome sequences of various species have been analysed and recorded, and the tertiary structures of proteins are being revealed. It is well known that there are relationships among these biological entities such as regulatory relationships among genes and physical interactions among proteins. In the area of bioinformatics, it is expected that analyzing such data will enable us to discover new biological and medical facts. We can say that all this data has internal or external "structures" in some sense or another. However, traditional machine learning methods have assumed that data are readily represented as vectors, and vector representations cannot be applied directly to such structured data. Recently, to break this barrier, many researchers are extensively studying how to extend machine learning methods to be able to handle structures hidden in data. In this thesis, we go further in this direction, and propose several approaches to handle structured data, especially, (I) methods to analyze internally structured data based on kernel methods, and (II) methods to predict the structures of externally structured data based on network evolution models.

(I) Methods to analyze internally structured data based on kernel methods

To handle internally structured data, it is essential to design kernel functions defining proper similarities between the structured data objects. Haussler proposed the convolution kernel, which is a framework for designing kernel functions that takes into account all substructures in internally structured data. However, when applying it to some class of structured data, we need appropriate definitions of the substructures of the data and efficient algorithms for computing the kernel functions. In this thesis, following the design scheme of the convolution kernel, we propose kernel functions

for labeled ordered trees and graphs. Although the naive computation of such a kernel causes a computational explosion, representing the problems in recursive forms enables us to show efficient algorithms by reducing them to dynamic programming or to simultaneous linear equations. Also, for tree-structured data, we show a result of the computational difficulty indicating that it is impossible to design kernels for more general trees with the same expressive power as the kernels for labeled ordered trees. Finally, we apply our kernel functions to the labeling problem of structured data, which is a generalized problem of the supervised classification problem.

(II) Methods to predict the structures of externally structured data based on network evolution models

For handling externally structured data, we consider the problem of link prediction which is a task to predict network structures such as the WWW, social networks, and biological networks. In this thesis, we especially focus on link prediction methods based only on structural information, not on node information. We assume an evolutiona model of network structures, and propose a link prediction method based on the model. Unlike the existing link metrics used for link prediction, our model has learnable parameters, and therefore we can use supervised learning to improve its predictive performance. In our model, the emergence and deletion of links are probabilistically determined by "copy and paste" mechanism. Assuming that the current network structure represents the stationary state of the network's evolution, we can predict unobserved links by fitting the marginal stationary distribution to the observed part of the network. We also propose an efficient on-line transductive estimation algorithm for solving the resulting optimization problem. Experimental results on a metabolic network and a protein network show that the proposed method outperforms the other link metrics.

Acknowledgments

First, I would like to thank gratefully and sincerely Dr. Tatsuya Akutsu who has been the supervisor of this study throughout my doctoral studies. He kindly gave me the opportunity to study this topic and many valuable suggestions. I would also like to thank Dr. Akihiro Yamamoto and Dr. Toshiyuki Tanaka, the referees of this thesis, for reviewing this thesis in detail and for giving me many valuable comments to improve this thesis.

Also, I would like to thank Dr. Norihiko Adachi and Dr. Takanori Fukao, the advisors in my master course in the department of Applied Systems Science in Kyoto University, and Mr. Tatsuo Hosotani, Chief Research Staff of the Shimane Institute for Industrial Technology, for giving me the opportunity to take the first steps towards becoming a researcher.

I would like to thank my co-authors in my earlier articles, Dr. Naoki Abe, Dr. Kiyoko F. Aoki-Kinoshita, Mr. Hiroaki Etoh, Dr. Takeshi Fukuda, Ms. Ryo Hirade, Dr. Kohichi Hirata, Dr. Tsuyoshi Ide, Dr. Akihiro Inokuchi, Mr. Makoto Kano, Dr. Akihiro Konagaya, Mr. Teruo Koyanagi, Dr. Tetsuji Kuboyama, Mr. Takahide Nogayama, Dr. Hiroshi Sakamoto, Dr. Tetsuo Shibuya, Dr. Kilho Shin, Mr. Yuta Tsuboi, Dr. Koji Tsuda, Mr. Tadashi Tsumura, and other collaborators, Dr. Tsuyoshi Kato, Dr. Hiroto Saigo, Dr. Nobuhisa Ueda, and Dr. Yoshihiro Yamanishi for valuable discussions, brilliant ideas, and helpful support.

I also would like to thank my managers in the IBM Tokyo Research Laboratory, Dr. Kazuyoshi Hidaka and Dr. Akira Tajima, for allowing and encouraging me to work on this doctoral thesis, and Mr. Shannon Jacobs for carefully checking the draft of this thesis. I also would like to thank IBM Japan, Ltd. for giving me various support for the doctoral course.

Finally, I would like to thank my family for their dedication and the many years of support.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

Fast evolving Internet technologies including the WWW, Web Services, and blogs can hardly be discussed without referring to structured data such as HTML and XML. Such data has functions to link to other data objects, and meta-structures such as link structures emerge from these links. Meanwhile, the entire genome sequences of various species have been analysed and recorded, and the tertiary structures of proteins are being revealed. It is well known that there are relationships among these biological entities, such as regulatory relationships among genes and physical interactions among proteins. In the field of bioinformatics, it is expected that analyzing such data will enable us to discover new biological and medical facts. We can say that all this data has internal or external "structures" in one sense or another.

For example, in the field of natural language processing [57], texts are represented as sequence structures, and parsed texts are represented as parse trees, i.e. tree structures. In the field of bioinformatics [19], we can represent sequences of DNA, of RNA, and of proteins as sequence structures, and secondary structures of RNA as tree structures. Also, the tertiary structures of proteins [31] and the structures of chemical compounds [33, 49] are sometimes represented as graph structures.

In Web data analysis, electronic documents are described as semi-structured data [1], such as XML and HTML, and shopping histories and transition histories in websites are represented as sequence-, tree-, or graph-structured data.

In addition, we can consider not only structures *in* the data, but also structures formed *among* data objects. For example, biological networks such as metabolic networks and protein-protein interaction networks, or link structures among Web pages can be interpreted as huge graph structures capturing the relationships among

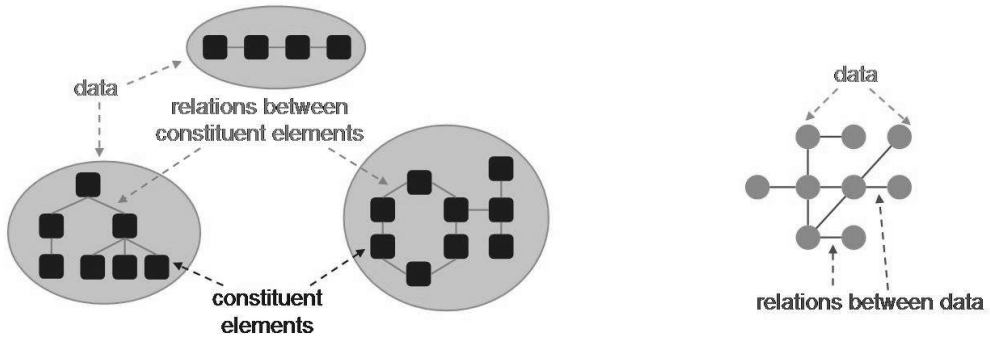

Figure. 1.1 Internal structure (left) and external structure (right).

genes, proteins, or hypertext documents.

With the massive amounts of structured data becoming readily available in electronic form today, it is naturally hoped that the analysis of such data will lead to new insights, possibly by applying machine learning methods. There has been a surge of interest in learning from structured data such as sequences, trees and graphs.

1.2 Roadmap

This thesis is organized into two main parts and a conclusion. We divide the structures in data into "internal structures" and "external structures" (See Figure 1.1.). Internally structured data is data with structures *in* the data, such as DNA sequences and HTML/XML. Externally structured data is data with structures *among* the data objects, such as biological networks and the WWW.

In Part I, we discuss approaches for data with internal structures. In Part II, we discuss approaches for data with external structures.

1.2.1 Part I: Kernel methods for structured data

In Part I, we consider applying *kernel methods* [70] to analyzing internally structured data.

In order to apply kernel methods to structured data objects, we first need to define a kernel function between two structured data. However, defining a kernel function is not an easy task, because it must be designed to be positive semidefinite, and ad

hoc similarity functions are not always positive semidefinite, e.g. [8] and [71].

Haussler [27] proposed a way to design kernel functions for discrete objects, where an object is decomposed into substructures (i.e. subsequences, subtrees, or subgraphs) and a feature vector is composed of the weighted counts of the substructures. Since the dimensionality of feature vectors is typically very high, they deliberately avoid explicit computations of feature values, and use efficient procedures such as dynamic programming or suffix trees. Chapter 2 of this dissertation is devoted to the introduction of kernel methods and the basic design schemes of kernels for structured data based on the convolution kernel.

Following the pioneering work by Haussler, a number of kernels were proposed for structured data. In Chapter 3, we design kernel functions for tree-structured data.

First, we propose an efficient algorithm to compute the kernel function for labeled ordered trees by using subtrees as their substructures in the framework of the convolution kernel, and extend the kernel to be able to handle flexible substructures. In addition, we show the #P-completeness of computing kernels for unordered trees.

In Chapter 4, we design kernel functions for directed graphs with vertex labels and edge labels. We define the kernels by using random walks on graphs in the framework of the marginalized kernel, and reduced the computation of the kernel to solving a system of linear simultaneous equations.

In Chapter 5, based on the techniques of the kernel functions we proposed in Chapter 3 and Chapter 4, we tackle the labeling problem, one of the important problems widely encountered in the areas of natural language processing (NLP), bioinformatics and Web data analysis. Labeling problems generalize supervised classification problems, since not only the label of one hidden variable, but the labels of a set of hidden variables are to be predicted. We introduce an efficient fully-kernelized learning algorithm for labeling structured data such as sequences, trees, and graphs.

1.2.2 Part II: Link prediction methods for network-structured data

In Part II, we consider analysis methods for external-structured data represented as network structures. The essence of the collective behavior of data is often embedded within the relationships among the data objects, such as co-occurrences, causality, and other types of interactions. Such relations can, for the most part, be represented as *networks* consisting of a set of entities and the links between them.

There has been a surge of interest in the study of analytical methods for network structured data, and *link mining* [25] has become a popular sub-area of data mining. Link mining includes several tasks such as link-based object classification/ranking/clustering, link prediction, and subgraph discovery [25]. In this dissertation, we consider the link prediction problem, which is the task of predicting unobserved portions of the network, such as hidden links, from the observed parts of the network (or alternatively, to predict the future structure of the network given the current structure of the network).

In Chapter 6, we introduce a new approach to the problem of link prediction for network-structured domains based on the topological features of network structure, not on the node features. We present a probabilistic evolution model of network structure which models probabilistic flips of the existence of edges depending on a "copy-and-paste" mechanism for the edges. Based on this model, we propose a transductive learning algorithm for link prediction based on an assumption of the stationarity of the network.

Part I

Kernel methods for structured data

Chapter 2

Kernel methods for structured data

In this chapter, we start with a basic introduction to kernel methods, and describe how we can apply kernel methods to structured data with concrete examples.

Existing methods in machine learning implicitly assume that data is represented as feature vectors [82]. A datum is usually represented as a point, i.e. a vector, in a feature space. For example, in the case of binary classification problems, a classifier learns rules such as the hyperplanes that can split positive examples and negative examples.

The definition of the feature space is assumed to be given beforehand. However, in the case of structured data, the definition is not trivial.

Probably one of the most natural ideas to handle structured data is to use substructures found in the data as features of the feature space. For example, a dimension of a feature vector may be a binary value, 0 or 1, indicating whether a particular substructure appear in a particular datum or not. Otherwise, the dimension may be the number of times the substructure appears in the datum. In the case of tree-structured data, it is natural to use paths or subtrees as features. However, since there can be exponentially many substrees in a tree according to its size, it causes a computational explosion to consider all of the possible substructures and construct feature vectors explicitly. Also, another problem is the so-called "curse of dimensionality" problem saying that predictive performance decreases significantly in very high dimensional feature spaces. These problems are traditionally tackled by selecting informative features. Kernel methods, especially the convolution kernel, is one of the promising approaches that gives an effective and elegant solution to this problem.

2.1 Kernel methods

We start with an introduction of kernel methods. Let us consider a binary classification problem, which is a task to predict whether or not a person is healthy based on the results of medical tests. Let us assume that we perform D medical tests on a person $\mathbf{x}$ in the set of all people X, and denote the test results by a D-dimensional vector $\Phi(\mathbf{x}) = (\phi_1(\mathbf{x}), \phi_2(\mathbf{x}), \ldots, \phi_D(\mathbf{x}))$. This vector is called *feature vector*. The variable y indicates whether or not the person is healthy. The value of $y = +1$ indicates "healthy", and $y = -1$ indicates "not healthy".

In the binary classification problem, the task is to estimate a function $h : \Re^D \rightarrow Y$ that maps from the D-dimensional feature space to the associated classes $Y = \{+1, -1\}$. To estimate h, the learner can exploit the training data consisting of N pairs of a feature vector and a class label, $(\Phi(\mathbf{x}^{(1)}), y^{(1)}), (\Phi(\mathbf{x}^{(2)}), y^{(2)}), \ldots, (\Phi(\mathbf{x}^{(N)}), y^{(N)})$ where $\mathbf{x}^{(i)} \in X$, $y^{(i)} \in Y$. The simplest choice of h is the linear classifier,

$$h(\mathbf{x}) = \mathrm{sign}\left(\langle \mathbf{w}, \Phi(\mathbf{x}) \rangle + b\right), \tag{2.1}$$

where the weight vector $\mathbf{w} \in \Re^D$ and the threshold $b \in \Re$ are the parameters, and $\langle \cdot, \cdot \rangle$ indicates the inner product, and 'sign' denotes the function returning the sign of its argument.

Perceptron is a well known learning algorithm that learns h with the parameters $\mathbf{w}$ and b from given examples. It is an incremental learning algorithm, which starts with the initial parameters $\mathbf{w} = \mathbf{0}$ and $b = 0$, and processes the examples one by one. For the i-th example, the perceptron guesses its class as $\hat{y}^{(i)} = h(\mathbf{x}^{(i)})$. It updates the parameters with the following update rules only when the guess is different from the true class label $y^{(i)}$.

$$\begin{aligned} \mathbf{w} &\leftarrow \mathbf{w} + y^{(i)}\Phi(\mathbf{x}^{(i)}) \\ b &\leftarrow b + y^{(i)}R^2, \end{aligned} \tag{2.2}$$

where R indicates the radius of the smallest hypersphere that includes all of the data. It is known that if at least one h classifying all examples exists, then the perceptron algorithm converges to the parameters with no errors to the given examples.

Let us consider the perceptron algorithm from another point of view. Noting that $y^{(i)} \in \{+1, -1\}$ in the update rules (2.2), we notice that the feature vector $\Phi(\mathbf{x}^{(i)})$

is always added or subtracted from the parameters, which implies that the weight vector is represented as a linear combination of the feature vectors,

$$\mathbf{w} = \sum_{i=1}^{N} \alpha_i \Phi(\mathbf{x}^{(i)}),$$

where α_i is the weight for the i-th example. For general kernel methods, the existence of this optimal linear representation is guaranteed by the *representer theorem*. By substituting this into (2.1), we obtain

$$h(\mathbf{x}) = \mathrm{sign}\left(\sum_{i=1}^{N} \alpha_i \langle \Phi(\mathbf{x}^{(i)}), \Phi(\mathbf{x}) \rangle + b\right). \tag{2.3}$$

Also (2.2) can be rewritten as

$$\alpha_i \leftarrow \alpha_i + y^{(i)}, \tag{2.4}$$

starting with $\alpha_i = 0$ for all i. Since (2.3) and (2.4) include no explicit expressions of the feature vectors, we can replace the inner product with the *kernel function* $K(\mathbf{x}, \mathbf{x}') = \langle \Phi(\mathbf{x}), \Phi(\mathbf{x}') \rangle$. This type of learning machines is called a *kernel method* since the data is accessed only via the kernel function. The dual representation by the kernel function has at least two merits. One is that even when the dimensionality of the feature space is very high (sometimes infinite) and it is infeasible to handle the feature vectors explicitly, a kernel method allows fast training and predictions if we have a black box K that somehow computes *just* the values of the inner products of the feature vectors. Another is that, on the other hand, it is possible to use a similarity function K defined by the user as a kernel function. In that case, we have to guarantee that K is a valid kernel function. It is known to be sufficient to show that K is positive semi-definite, i.e. K is symmetric $K(\mathbf{x}, \mathbf{x}') = K(\mathbf{x}', \mathbf{x})$ and

$$\sum_{\mathbf{x} \in X} \sum_{\mathbf{x}' \in X} w(\mathbf{x}) w(\mathbf{x}') K(\mathbf{x}, \mathbf{x}') \geq 0$$

for all w satisfying $\sum_{\mathbf{x} \in X} w(\mathbf{x}) < \infty$.

Also, we can easily integrate multiple kernel functions since the sums or products of kernel functions are still kernel functions.

2.2 Convolution kernels

Let us consider cases where $\mathbf{x}$ is a structured data object, e.g. a DNA sequence, an XML document, or a chemical compound. In most cases, structured data can be

represented as sequences, trees, or graphs. For example, a DNA sequence is represented as a sequence capturing the adjacency of bases, and a chemical compound is represented as a graph of the covalent bonds among atoms. Therefore, all we have to do is to define kernel functions between sequences, trees, or graphs. However, how can we design their kernel functions? If there are some kind of structures among the constituent elements in a datum, it is not obvious how to represent them as a feature vector $\Phi(\mathbf{x})$.

Although there are no absolute rules for designing the kernel functions, Haussler proposed *convolution kernels* as a general framework for designing kernel functions for discrete structures [27]. The basic idea behind the convolution kernels is that a datum is decomposed into its "parts" and the kernel function is defined by the sum of the kernel functions of the "parts".

As a concrete example, let us consider sequence-structured data. It is common to use n-gram models, i.e. Markov models, as probabilistic models for sequence-structured data. In n-gram models, $n-1$ consecutive characters determine the next character. This implies that several consecutive characters can represent the characteristics of the sequence. Generalizing this idea, it is natural to hypothesize that the characteristics of structured data are captured by their substructures, and the similarity between two structured data is defined by the similarities between their substructures. The convolution kernel proposed by Haussler reflects this idea for designing kernel functions for structured data. Convolution kernels are defined in the form of

$$K(\mathbf{x}, \mathbf{x}') = \sum_{\mathbf{s} \in S(\mathbf{x})} \sum_{\mathbf{s}' \in S(\mathbf{x}')} f(\mathbf{s}|\mathbf{x}) f(\mathbf{s}'|\mathbf{x}') K^S(\mathbf{s}, \mathbf{s}'), \tag{2.5}$$

where $S(\mathbf{x})$ represents a finite set of substructures extracted from $\mathbf{x}$, and K^S is the kernel function between two substructures. Also, $f(\mathbf{s}|\mathbf{x})$ is the weight function returning the weight for the substructure $\mathbf{s}$ extracted from $\mathbf{x}$. The convolution kernel is defined as a sum of the kernel functions for all possible pairs of substructures extracted from $\mathbf{x}$ and $\mathbf{x}'$. The similarity between two data is defined by the similarities between the substructures of the data (Figure 2.1). If K^S is a kernel function, (2.5) is guaranteed to be a kernel function. Note that the original definition of the convolution kernel [27] has no weight functions f.

The convolution kernel itself is no more than a framework. We have to design concrete kernel functions for various kinds of structured data, which is to define

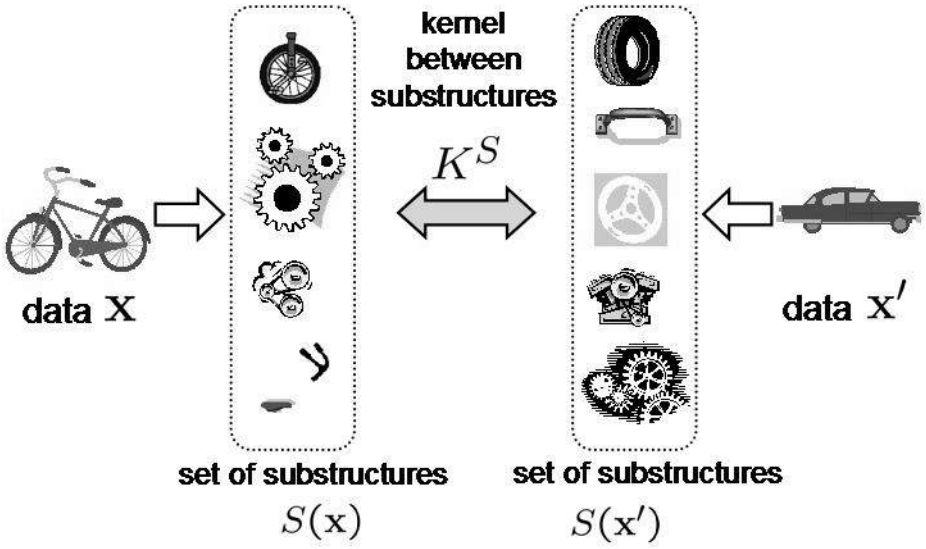

Figure. 2.1 The idea behind the convolution kernel.

substructures $S(\mathbf{x})$ and to design efficient algorithms for kernel computations [22, 70].

2.2.1 Convolution kernels for sequence-structured data

As a concrete example of the convolution kernels, we consider a convolution kernel for sequence-structured data. Let $\mathbf{x} = (x_1, x_2, \ldots, x_\ell)$ and $\mathbf{x}' = (x'_1, x'_2, \ldots, x'_{\ell'})$ be sequences of length ℓ and length ℓ', respectively. Also, let Σ be a set of labels, and $\sigma(x_i) \in \Sigma$ be the associated label of an element x_i.

We define the set of substructures $S(\mathbf{x})$ as the set of all subsequences of length n in $\mathbf{x}$, which is denoted by $S^{(n)}(\mathbf{x})$. In other words, the feature space is defined by subsequences of length n. We define the weight $f(\mathbf{s}|\mathbf{x})$ of subsequence $\mathbf{s}$ so that it decreases as the matched area becomes longer. By using a small constant $0 < \lambda < 1$, the weight is defined as $f(\mathbf{s}|\mathbf{x}) = \lambda^{g(\mathbf{s})}$ for a subsequence $\mathbf{s}$ matching an area of length $g(\mathbf{s})$. For example, we can find a subsequence "cat" by extracting $\mathbf{s} = (x_1, x_2, x_6) = (\mathrm{c}, \mathrm{a}, \mathrm{t})$ from $\mathbf{x} =$"cannot". Since the length of the matched area is $g(\mathbf{s}) = 6$, its weight becomes $f(\mathbf{s}|\mathbf{x}) = \lambda^6$.

The convolution kernel for sequence-structured data is defined as

$$K(\mathbf{x}, \mathbf{x}') = \sum_{\mathbf{s} \in S^{(n)}(\mathbf{x})} \sum_{\mathbf{s}' \in S^{(n)}(\mathbf{x}')} \lambda^{g(\mathbf{s})} \lambda^{g(\mathbf{s}')} K^S(\mathbf{s}, \mathbf{s}'), \tag{2.6}$$

where the kernel function between two subsequences $\mathbf{s} = (s_1, s_2, \ldots, s_n)$ and $\mathbf{s}' = (s'_1, s'_2, \ldots, s'_n)$ is defined as

$$K^S(\mathbf{s}, \mathbf{s}') = \prod_{i=1}^{n} K^\Sigma(\sigma(s_i), \sigma(s'_i)), \tag{2.7}$$

which is again defined as the product of the element-wise kernel function K^Σ. If we define the element-wise kernel between σ and $\sigma' \in \Sigma$ as

$$K^\Sigma(\sigma, \sigma') = \begin{cases} 1 & (\sigma = \sigma') \\ 0 & (\sigma \neq \sigma') \end{cases}, \tag{2.8}$$

then $K^S(\mathbf{s}, \mathbf{s}')$ will be 1 when $\mathbf{s}$ and $\mathbf{s}'$ are identical, and will be 0 otherwise.

Explicit enumeration of all the subsequences of length n from $\mathbf{x}$ and $\mathbf{x}'$ is prohibitive, since the numbers of such subsequence are too large, i.e. ${}_\ell C_n$ and ${}_{\ell'} C_n$, respectively. Therefore, we use dynamic programming for efficient kernel computations.

Let $S_t^{(n)}(\mathbf{x})$ be the set of subsequences of length n and with its rightmost element at position t. Then (2.6) becomes

$$\begin{aligned} K(\mathbf{x}, \mathbf{x}') &= \sum_{t=1}^{\ell} \sum_{t'=1}^{\ell'} k^{(n)}(t, t'), \\ k^{(n)}(t, t') &:= \sum_{\mathbf{s} \in S_t^{(n)}(\mathbf{x})} \sum_{\mathbf{s}' \in S_{t'}^{(n)}(\mathbf{x}')} \lambda^{g(\mathbf{s})} \lambda^{g(\mathbf{s}')} K^S(\mathbf{s}, \mathbf{s}'). \end{aligned} \tag{2.9}$$

(2.9) is interpreted as the kernel function when we focus only on such subsequences whose rightmost elements are $\mathbf{x}_t$ or $\mathbf{x}'_t$.

Based the definition of (2.7) and considering that the subsequences of length n with the rightmost element x_t are constructed by adding x_t to the subsequences of length $n-1$ with the rightmost element x_τ for $\tau < t$, then (2.9) can be written as

$$k^{(n)}(t, t') = \sum_{\tau=1}^{t-1} \sum_{\tau'=1}^{t'-1} \lambda^{t-\tau} \lambda^{t'-\tau'} k^{(n-1)}(\tau, \tau') K^\Sigma(\sigma(x_t), \sigma(x'_{t'})),$$

which is recursive with respect to n,t, and t'. With this, the computation time becomes $O(n\ell^2\ell'^2)$, but a few more tricks can decrease the computation time further.

Defining

$$J^{(n)}(t,t') := \sum_{\tau=1}^{t-1}\sum_{\tau'=1}^{t'-1} \lambda^{t-\tau}\lambda^{t'-\tau'} k^{(n-1)}(\tau,\tau'),$$

we have the following recursive form,

$$\begin{aligned} J^{(n)}(t,t') = \lambda J^{(n)}(t-1,t') + \lambda J^{(n)}(t,t'-1) \\ -\lambda^2 J^{(n)}(t-1,t'-1) + k^{(n-1)}(t-1,t'-1). \end{aligned} \tag{2.10}$$

By using (2.10) and

$$k^{(n)}(t,t') = J^{(n)}(t,t')K^{\Sigma}(\sigma(x_t),\sigma(x'_{t'})),$$

the kernel is computed in $O(n\ell\ell')$ time.

If we assume (2.8) and that the set of substructures is *substrings*, it is possible to compute the string kernel in linear time by using suffix trees [70].

2.2.2 What are the limits on the complexity of substructures?

One might wonder what substructures we should use. Although there are no principled methods to determine this, we should note that there are trade-offs between expressiveness and computational efficiency when we design kernels for structured data.

We can not use all substructures for give structured data. If the substructures are too complex, the kernel computations become intractable, i.e. not computable in polynomial time. On the other hand, if they are too simple, they do not have sufficient expressive power to capture the characteristics of the structured data.

As we have seen in the example of the previous subsection, we obtained a tractable string kernel by limiting the class of the substructures used and by assuming the exponentially decaying weight.

Therefore, when we apply kernel methods to structured data, we need to define substructures of as high an expressive power as possible and give an efficient algorithm for the kernel computations. Conversely, we need to select substructures whose resulting kernel computations are tractable. For example, in Chapter 4, we will consider designing kernel functions for graph-structured data. Although subgraphs are very expressive substructures for graph-structured data, those result in a #P-complete problem, and therefore we need to pursue simpler substructures such as paths.

2.3 Other approaches

Besides the convolution kernel, there are several ways to design kernel functions for structured data. If we have generative models of data as prior knowledge, such as hidden Markov models or probabilistic context free languages, we can utilize them to construct *Fisher kernels* [34, 35, 70]. Let us assume that the family of the generative models is a probabilistic distribution $P(\mathbf{x}|\theta)$ with parameters θ, and we are given a model $P(\mathbf{x}|\theta^*)$ specified by the particular parameters $\theta = \theta^*$ The Fisher kernel is defined as

$$K(\mathbf{x}, \mathbf{x}') = \Phi(\mathbf{x}) F^{-1} \Phi(\mathbf{x}')^\top,$$

where the feature vector $\Phi(\mathbf{x})$ is defined by

$$\Phi(\mathbf{x}) = \nabla_\theta \log P(\mathbf{x}|\theta)|_{\theta=\theta^*},$$

whose direction is the steepest gradient of the log-likelihood of the data $\mathbf{x}$ with respect to the model parameters. Intuitively, each dimension of $\Phi(\mathbf{x})$ indicates how much the corresponding parameter contributes to generating $\mathbf{x}$. Also, $F := E_\theta[\Phi(\mathbf{x})^\top \Phi(\mathbf{x})]$ is called the Fisher information matrix, but it is usually set to $F^{-1} = I$ for simplicity.

Although we focus on designing kernels for internally structured data in this chapter, there are several approaches for externally structured data. The best known approach is probably the diffusion kernel [47]. The diffusion kernel is a kernel function between two arbitrary nodes in an undirected graph that represents the external structures of the data. The definition of the diffusion kernel is given as

$$K(t) = \exp(tL) = \lim_{n\to\infty} \left(1 + \frac{tL}{n}\right)^n,$$

where L is the Laplacian matrix of the given graph. The diffusion kernel has various applications such as text classification [39] and protein function prediction in biological networks.

Apart from the kernel method, there are several approaches for handling structured data. Relational learning [59] is a general approach to handling data represented using predicate logic. In relational learning, several relationships among the constituent elements are defined, and features are defined as sets of the relationships. Only useful features are incrementally built up in the process of training. However, since the problem of searching for the best hypothesis is generally NP-hard, we must use

heuristic methods. Another approach is based on pattern discovery algorithms [33]. All substructures appearing frequently are enumerated beforehand and used as features. Although it is an advantage that we can make use of unlabelled data, the process of discovering patterns is again almost always #P-complete.

2.4 Concluding remarks

In this chapter, we reviewed the kernel method, which is one of the promising approaches for internally structured data, and the convolution kernel, which is a design framework for kernel functions. Also, we introduced the string kernel as an example of convolution kernels.

Based on the framework of convolution kernels, we will discuss kernels for tree-structured data in Chapter 3, and kernels for graph-structured data in Chapter 4.

Chapter 3

Kernel functions for tree-structured data

In this chapter, we consider designing kernel functions for tree-structured data.

Following the framework of the convolution kernel introduced in Chapter 2, Collins and Duffy proposed a kernel function for parse trees that can be seen in natural language processing [14]. They defined the kernel functions by using a limited class of subtrees as the substructures, and proposed an efficient algorithm for computing the kernel functions. However, in the subtrees they used, the child nodes for a node are labeled by their ordering, and the algorithm depends on that assumption. Therefore their kernel function is not applicable to more general classes of trees.

Here a question arises: What are the limits of the kinds of trees for which we can design kernel functions? This chapter attempts to answer this question. First, we show an algorithm for computing a kernel function for labeled ordered trees, and show that its computational time is the same as that for the parse tree kernel. Then we extend the proposed kernel to allow for structural ambiguities of the substructures. Also, as a limit of the generalization of tree kernels, we consider the computational difficulty of the kernel with subtrees as the substructures of unordered trees.

Finally, we demonstrate the proposed kernel functions with synthetic data and real HTML documents.

3.1 Convolution kernels for trees

3.1.1 Convolution kernels for trees

Let us denote two tree-structured data objects as $T = (V, E)$ and $T' = (V', E')$, where V and V' are sets of nodes and E and E' are sets of edges. The convolution kernel (2.5) for trees is defined as

$$K(T, T') = \sum_{\mathbf{s} \in S(T)} \sum_{\mathbf{s}' \in S(T')} K^S(\mathbf{s}, \mathbf{s}'). \tag{3.1}$$

We call this a *tree kernel*. Note that although we used the weights for the substructures f, here we set $f(\mathbf{s}|T) = 1$ for simplicity. Collins and Duffy represent the weight as decaying exponentially with respect to the depth of s [14].

We can define various tree kernels by limiting the class of tree-structured data, the set of substructures S, and the substructure-wise kernel K_S. In this chapter, we use all of the subtrees found in trees as S. (3.1) is rewritten as

$$\begin{aligned} K(T, T') &= \sum_{v \in V} \sum_{v' \in V'} \sum_{\mathbf{s} \in S_v(T)} \sum_{\mathbf{s}' \in S_{v'}(T')} K^S(\mathbf{s}, \mathbf{s}') \\ &= \sum_{v \in V} \sum_{v' \in V'} K^R(v, v'), \end{aligned} \tag{3.2}$$

where $S_v(T)$ is the set of subtrees with its root node at $v \in V$, and $K^R(v, v')$ is the kernel function focusing only on $S_v(T)$ and $S_{v'}(T')$, i.e.

$$K^R(v, v') = \sum_{\mathbf{s} \in S_v(T)} \sum_{\mathbf{s}' \in S_{v'}(T')} K^S(\mathbf{s}, \mathbf{s}'). \tag{3.3}$$

3.1.2 Parse tree kernel

Collins and Duffy proposed a convolution kernel for parse trees (Figure 3.1, left) found in the area of natural language processing [14]. A parse tree can be seen as a labeled rooted ordered tree, each of whose nodes has a label in Σ, and the child nodes each nodes have a left-to-right order. For the substructures $S(T)$, all subtrees are used, but we assume that each of the edges in the subtrees is labeled by the order of its child nodes (Figure 3.1, right). The kernel between two substructures $\mathbf{s} \in S(T)$ and $\mathbf{s}' \in S(t')$ is defined as

$$K^S(\mathbf{s}, \mathbf{s}') = I(\mathbf{s} = \mathbf{s}'), \tag{3.4}$$

where $I()$ is a function that returns 1 when its argument is true, and returns 0 otherwise. The $\mathbf{s} = \mathbf{s}'$ indicates that the two labeled ordered trees $\mathbf{s}$ and $\mathbf{s}'$ are identical.

Since it is prohibitive to compute (3.1) by enumerating exponentially many $S(T)$ and $S(T')$ explicitly, Collins and Duffy showed an efficient method for computing only the value of the kernel function by using the decomposition (3.2) and the fact that in (3.3) all pairs of v and v' are recursively computed in $O(|V||V'|)$ time with the following recursive equation.

- If either v or v' is a leaf node,

$$K^R(v, v') = I(\ell(v) = \ell(v')) \tag{3.5}$$

- If neither v or v' is a leaf node,

$$K^R(v, v') = I(\ell(v) = \ell(v')) \cdot \left(\prod_{i=1}^{\sharp ch(v)} (K^R(ch(v, i), ch(v', i)) + 1) \right) \tag{3.6}$$

In the recursive equation (3.6), $\sharp ch(v)$ indicates the number of child nodes of node v, and $ch(v, i)$ is the i-th child of v. Since subtrees with their roots at v are constructed by combining subtrees with their roots at the children of v and adding v above them, $K^R(v, v')$ is computed by enumerating all combinations of kernels for trees with their roots at the children of v and v' (the second factor in the right-hand side of (3.6)) and adding the node-wise kernels between v and v' to their tops (the first factor in the right-hand side of (3.6)). If the labels of v and v' are different, no two subtrees match with their roots at v and v', and then $K^R(v, v') = 0$. In the second factor on the right-hand side of (3.6), since all of the edges are labeled by their orders, we have only to consider the kernels between the trees with their roots at the i-th child of v and the trees with their roots at the i-th child of v'.

3.2 Labeled ordered tree kernels

3.2.1 A labeled ordered tree kernel

In this subsection, we remove the constraint that the parse tree kernel assumes, and introduce a more general kernel for labeled ordered trees.

Since the subtrees in $S(T)$ have edge labels indicating the child order in the parse tree kernels, there is a problem that two subtrees will not be considered to be identical

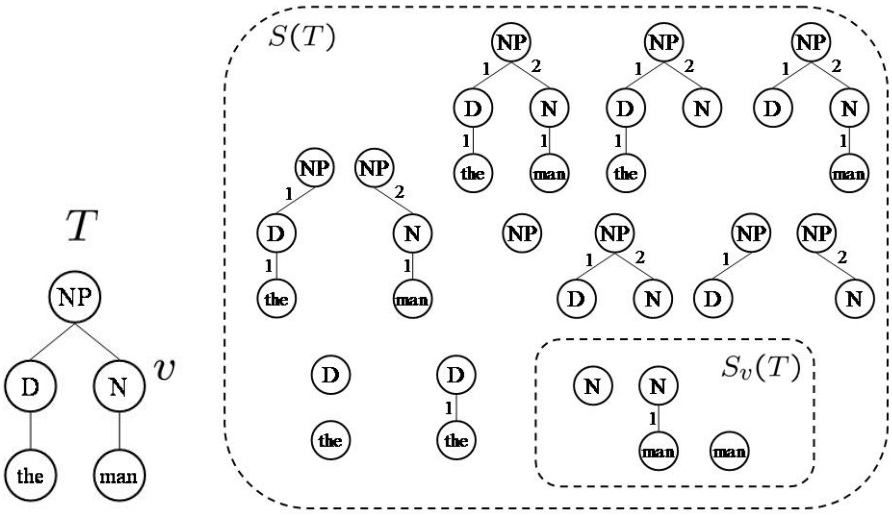

Figure. 3.1 (Left) An example of a parse tree T. (Right) The sets of substructures $S(T)$ and $S_v(T)$ in T.

unless the positions of the child nodes are exactly the same, even if the shapes and all of the node labels are identical. Therefore, the substructures in Figure 3.1 are useless for the classification of trees whose node labeled "man" is the second child. This constraint is problematic if we consider general labeled ordered trees such as HTML documents, since a node can have many child nodes. Therefore we remove this constraint, and use all subtrees without these position labels as the substructures.

Since we removed the position labels indicating the orders of the child nodes for each node, the second term in the right-hand side of (3.6) is no longer sufficient since we have to consider not only the kernels between the trees with their roots at the i-th child of v and the trees with their root at the i-th child of v', but also the kernels between the trees with their roots at the i-th child of v and the trees with their root at the j-th child of v' for all combinations of i and j. Therefore, we have to consider all possible matchings that preserve the order of the children. However, the numbers of subsets of the child nodes v and v' are $2^{\sharp ch(v)}$ and $2^{\sharp ch(v')}$, respectively. This implies that there exist exponentially many matchings, so it is difficult to evaluate them explicitly. As response to this problem, we use dynamic programming not only for the nodes, but also for the child nodes for each pair of nodes. Let $\bar{K}^R_{v,v'}(i,j)$ correspond to the second term on the right-hand side of (3.6), focusing only on the first child node to the i-th child node of v and the first child node to the j-th child

node of v'. Instead of (3.6), we have

$$K^R(v, v') = I(\ell(v) = \ell(v')) \cdot \bar{K}^R_{v,v'}(\sharp ch(v), \sharp ch(v')). \tag{3.7}$$

Then the following recursive equation holds,

$$\begin{aligned}\bar{K}^R_{v,v'}(i,j) = \bar{K}^R_{v,v'}(i-1,j) + \bar{K}^R_{v,v'}(i,j-1) - \bar{K}^R_{v,v'}(i-1,j-1) \\ + \bar{K}^R_{v,v'}(i-1,j-1) \cdot K^R(ch(v,i), ch(v',j)),\end{aligned} \tag{3.8}$$

where the boundary condition is $\bar{K}^R_{v,v'}(i,0) = \bar{K}^R_{v,v'}(0,j) = 1$. The recursive equation (3.8) can be interpreted as follows. The first line is considering all matchings without the i-th child of v and the j-th child of v', and the last term is subtracting the doubly-counted matchings (See Figure 3.2). The second line considers all matchings with the i-th child of v and the j-th child of v'. This is computed by adding the i-th child of v and the j-th child of v' to the matchings with the $1 \dots i-1$-th children of v and $1 \dots j-1$-th children of v' that have been already considered. This recursive equation allows us to compute $K^R(v, v')$ in $O(\sharp ch(v) \cdot \sharp ch(v'))$ time.

The computational complexity of computing this kernel is shown to be $O(|V||V'|)$ as follows, which is the same as for the parse tree kernel.

$$\begin{aligned}\sum_{v \in V} \sum_{v' \in V'} O(\sharp ch(v) \cdot \sharp ch(v')) &= \sum_{v \in V} O(\sharp ch(v)) \cdot \sum_{v' \in V'} O(\sharp ch(v')) \\ &= O(|V| \cdot |V'|)\end{aligned}$$

3.2.2 Extensions of the labeled ordered tree kernel

In the labeled tree kernel we proposed in the previous subsection, the kernel between two subtrees $\mathbf{s} \in S(T)$ and $\mathbf{s}' \in S(T')$ is 1 only when $\mathbf{s}$ and $\mathbf{s}'$ are completely identical. However, there may be some cases where we want to say they are effectively identical if the two subtrees are sufficiently similar. In this subsection, we will introduce two extensions to allow flexibility in the labeled ordered tree kernel.

First, we consider flexibility in the identity of node labels. Let $\mathbf{s}$ and $\mathbf{s}'$ have identical shapes. We define the kernel between $\mathbf{s}$ and $\mathbf{s}'$ as the product of the label-wise kernels,

$$K^S(\mathbf{s}, \mathbf{s}') = \prod_{i=1}^{|V_\mathbf{s}|} K^\Sigma(\ell(v_i), \ell(v'_i)),$$

where $V_\mathbf{s} = (v_1, v_2, \dots)$ is the ordered set of the preordered nodes included in $\mathbf{s}$, and K^Σ is the kernel between two node labels. This extension is equivalent to setting

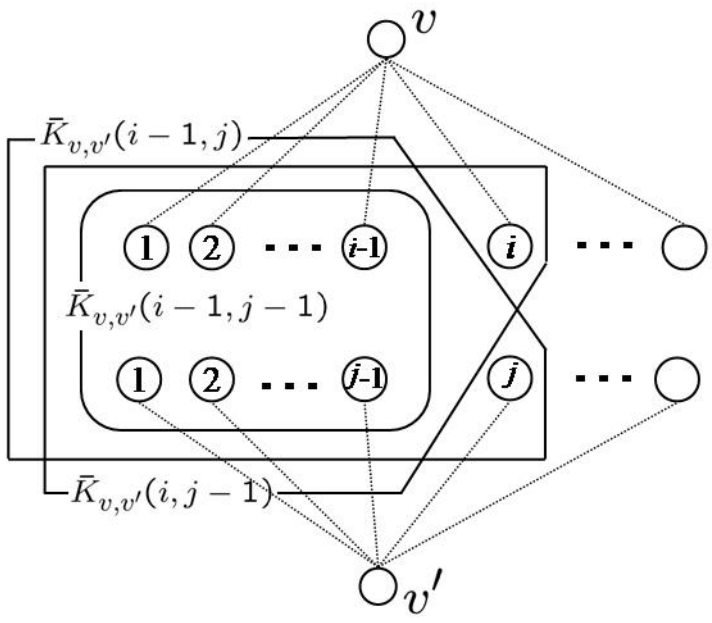

Figure. 3.2 An image of the recursive equation (3.8).

$K^{\Sigma}(\sigma,\sigma') = I(\sigma = \sigma')$ in the original labeled ordered tree kernel, and (3.7) is rewritten as

$$K^{R}(v,v') = K^{\Sigma}(\ell(v),\ell(v')) \cdot \bar{K}^{R}_{v,v'}(\sharp ch(v), \sharp ch(v')).$$

Next, we introduce flexibility in the identity of the subtree shapes. In this extension, the substructures $S(T)$ are not limited to subtrees, i.e. connected graphs, but to *elastic subtrees*. We allow the substructures to be trees that can be embedded into the original tree by extending their edges. In Figure 3.3, we show an example of extracting a substructure $\mathbf{s}$ of 5 nodes, $v_1, \ldots, v_5$, from T. Extending the edge between v_3 and v_4 and the edge between v_3 and v_5, $\mathbf{s}$ is embedded into T.

The extended kernel function can be efficiently computed by modifying the basic algorithm. The recursive equations in (3.8) and (3.6) exploit the construction approach that substructures $S_v(T)$ with their roots at v can be recursively constructed by combining subtrees with their roots at v's children and adding v to their tops. Allowing elastic subtrees as the substructures requires to considering not only substructures with their roots at v's children, but also substructures with their roots at v's descendants. Let $D(v)$ be the set of nodes including v and v's descendants, and

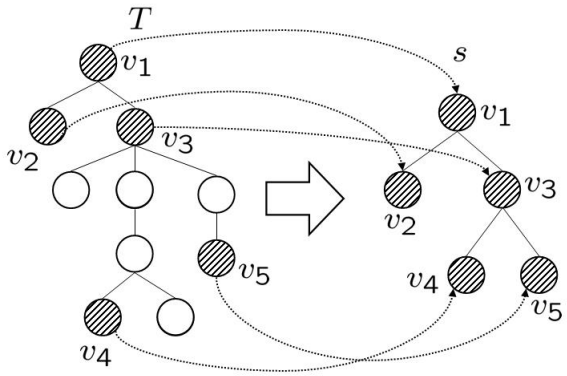

Figure. 3.3 An example of a substructure $\mathbf{s}$ of tree T when elastic subtrees are allowed as the substructures.

let the substructures using v and its descendants be

$$S_v^D(T) = \bigcup_{u \in D(v)} S_u(T),$$

where $S_u(T)$ is the set of subtrees embedded into T with their roots at u. Note that the definition of $S_u(T)$ is slightly changed. By using the kernel based on substructures using v, v' and their descendants

$$K^D(v, v') = \sum_{\mathbf{s} \in S_v^D(T)} \sum_{\mathbf{s}' \in S_{v'}^D(T')} K^S(\mathbf{s}, \mathbf{s}'). \tag{3.9}$$

(3.8) can be rewritten as

$$\begin{aligned} \bar{K}_{v,v'}^R(i,j) = \bar{K}_{v,v'}^R(i-1,j) + \bar{K}_{v,v'}^R(i,j-1) - \bar{K}_{v,v'}^R(i-1,j-1) \\ + \bar{K}_{v,v'}^R(i-1,j-1) \cdot K^D(ch(v,i), ch(v',j)). \end{aligned}$$

Since computing $K^D(v, v')$ by direct application of (3.9) needs $O(|S_v^D(T)| \cdot |S_{v'}^D(T')|)$ time, the total computation time becomes $O(|V|^2 \cdot |V'|^2)$. However, by using the following recursion, it is reduced to $O(\sharp ch(v) \cdot \sharp ch(v'))$

$$\begin{aligned} K^D(v, v') = \sum_{i=1}^{\sharp ch(v)} K^D(ch(v,i), v') + \sum_{j=1}^{\sharp ch(v')} K^D(v, ch(v',j)) \\ - \sum_{i=1}^{\sharp ch(v)} \sum_{j=1}^{\sharp ch(v')} K^D(ch(v,i), ch(v',j)) + K^R(v, v') \end{aligned}$$

3.3 Difficulty of computing tree kernels

In this section, we discuss whether or not it is possible to generalize the tree kernel we proposed in the previous section. A natural generalization of labeled ordered trees is labeled unordered trees whose child nodes at each node have no order. As we will show later, if we define a tree kernel for unordered trees with subtrees as their substructures, the kernel computations become #P-complete, which implies further generalization of the tree kernel is impossible in this direction.

First, we define the following problem, TREE KERNEL, as a problem to compute the kernel for labeled rooted unordered trees.

Problem 1: TREE KERNEL(T, T')
Input: two labeled rooted unordered trees, T and T'
Output: $K(T, T')$

Without loss of generality, we assume $|V| \leq |V'|$. Also, let $S(T)$ be the set of all subtrees in T, and $K^S(\mathbf{s}, \mathbf{s}') = I(\mathbf{s} = \mathbf{s}')$. Computing $K(T, T')$ is equivalent to counting the number of common subtrees shared by T and T'. Note that since we have no order of child nodes, it is permissible to change the orders of the child nodes when we check if two subtrees are identical.

Before evaluating the difficulty of Problem 1, we will consider the following limited version of the problem.

Problem 2: TREE KERNEL$^{(n)}(T, T')$
Input: two labeled rooted unordered trees, T and T'
Output: $K^{(n)}(T, T')$

In Problem 2, we define $S^{(n)}(T)$ as the set of subtrees with just n nodes, and $K^{(n)}(T, T')$ is the tree kernel whose underlying substructures are subtrees with n nodes, i.e.

$$K^{(n)}(T, T') = \sum_{\mathbf{s} \in S^{(n)}(T)} \sum_{\mathbf{s}' \in S^{(n)}(T')} K^S(\mathbf{s}, \mathbf{s}').$$

Note that, when $n = |V|$, $K^{(|V|)}(T, T')$ counts the number of times T appears in T' since $S^{(|V|)}(T) = \{T\}$. Similarly, $K^{(n)}(T, T')$ counts the number of subtrees of size n shared by T and T'.

Then the following lemma showing the difficulty of TREE KERNEL$^{(|V|)}(T, T')$ holds.

Lemma 1 *TREE KERNEL$^{(|V|)}(T, T')$ is #P-complete.*

(proof) We will show Lemma 1 based on the difficulty of the following problem on a bipartite graph $G = (X \cup Y, E)$.

Problem 2: #PERFECT MATCHINGS(G) [78, 80]
Input: A bipartite graph $G = (X \cup Y, E)$, s.t. $|X| = |Y|$
Output: Number of perfect matchings in G

Note that there are no edges among X, and there are no edges among Y, either. We call $M \subseteq E$ a *matching* if no pairs of nodes in M share nodes. Further, if M covers all of the nodes in V, we call M a *perfect matching*. It is known that #PERFECT MATCHINGS(G) is a $\#P$-complete problem.

Let the set of labels be $\Sigma = \{0, 1, 2, \ldots, |X|\}$. Let us consider T and T' shown in Figure 3.4. Here, t_j is a set of nodes defined by the following rule: t_j has the node labeled by i if and only if $(x_i, y_j) \in E$.

Because of the rule, the node labeled by i in T can be embedded to t_j. When T appears in T', T and T' are identical except for their leaves. Therefore, each of the leaves in T is embedded into $t_1, \ldots, t_{|X|}$, and each of $t_1, \ldots, t_{|X|}$ has just one of the leaves in T. This corresponds to a perfect matching in G, which means that the output of TREE KERNEL$^{(|V|)}(T, T')$ and the output of #PERFECT MATCHINGS(G) are identical. Therefore, it has been shown that #PERFECT MATCHINGS(G) is reduced to TREE KERNEL$^{(|V|)}(T, T')$ in polynomial time.

Next, we prove TREE KERNEL$^{(|V|)}(T, T')$ is in #P. We define an *embedding* E of T into T' as $E \subseteq V \times V'$ such that E includes all of the nodes of T, and for any (a, b) and $(c, d) \in E$, the parentage status between a and c is equivalent to that of b and d. From the definition of #P [48], it is sufficient to prove that (1) for any $E \subseteq V \times V'$, it can be checked whether or not E is an embedding in polynomial time with respect to $|V| + |V'|$, and (2) for any embedding E, there exists a constant c such that $|E| < (|V| + |V'|)^c$. (1) is equivalent to checking whether or not E includes all of the nodes of T and forms two isomorphic trees, which can be done in quadratic time. The condition (2) is obvious since $|E| \leq 2|V|$.

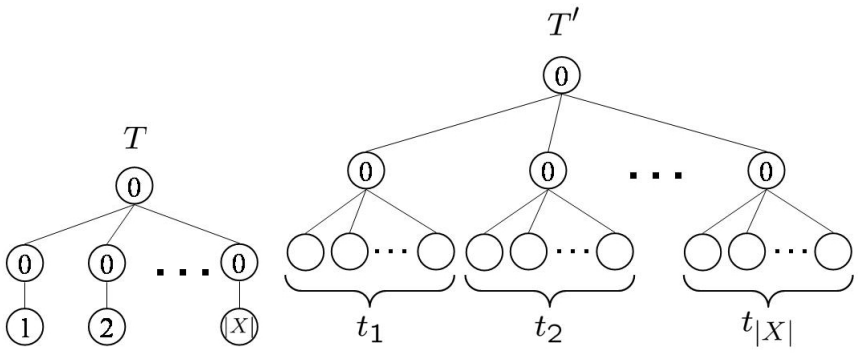

Figure. 3.4 T and T' used in the proof of Lemma 1.

□

By using Lemma 1, we can show the following theorem on the difficulty of TREE KERNEL(T, T'), which is our goal in this subsection.

Theorem 1 *TREE KERNEL*(T, T') *is #P-complete*

(proof) We will prove that TREE KERNEL(T, T') is as difficult as TREE KERNEL$^{(|V|)}(T, T')$ by using the Cook reduction [79]. In other words, we will show that TREE KERNEL$^{(|V|)}(T, T')$ can be solved in polynomial time by using an oracle to solve TREE KERNEL(T, T'). To do this, TREE KERNEL$^{(|V|)}(T, T')$ is reduced to a system of simultaneous linear equations by using the techniques introduced by Vadhan [78].

Let us introduce an extra label \$ that is not included in Σ, and $\Sigma' = \Sigma \cup \{\$\}$. Then, we extend T by the following procedure.

For each $m = 0, \ldots, |V|$, we create a chain $v_1 - v_2 - \cdots - v_m$ of length m, and add a label \$ to every node. The chain is added to every node in T (Figure 3.5). The resulting tree with the chains of length m is called T_m. Finally, we obtain $|V| + 1$ extended trees. Similarly, T' is extended to $T_m{}'$.

In the following, we will show TREE KERNEL$^{(|V|)}(T, T')$ is solved in polynomial time if we have an oracle to solve TREE KERNEL$(T_m, T_m{}')$ in polynomial time for T_m and $T_m{}'$ as obtained by this procedure.

As stated before, $K(T_m, T_m{}')$ is equal to the number of common subtrees of T_m and $T_m{}'$. We divide these common subtrees into the following three groups.

(a) subtrees containing only \$

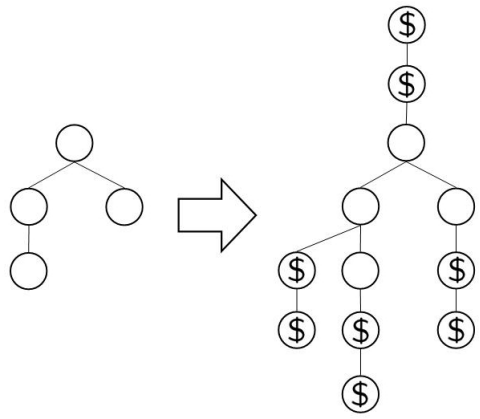

Figure. 3.5 Extension of T used in the proof of Theorem 1 (in the case of $m = 2$).

(b) subtrees containing no \$

(c) subtrees not included in (a) or (b)

Let $A^{(0)}$ be the number of common subtrees of type (a). $A^{(0)}$ can easily be computed as

$$\begin{aligned} A^{(0)} &= |V||V'| \sum_{k=1}^{m} (m-k+1)^2 \\ &= |V||V'|\ m(m+1)(2m+1)/6. \end{aligned}$$

Let $A^{(n)}$ be the number of common subtrees of type (b) with $1 \leq n \leq |V|$ nodes, and $\tilde{A}^{(n)}$ be the number of common subtrees of type (c) with $1 \leq n \leq |V|$ nodes whose labels are not \$. Then there exists a subtree of type (b) isomorphic to each subtree of type (c) whose nodes labeled with \$ have been removed. Also, since \$ does not exist in T nor T', the differences between $\tilde{A}^{(n)}$ and $A^{(n)}$ depend only on the added chains of length m. When we focus on a subtree of size n in (b), since the subtrees constructed by adding chains of length m or less than m to the subtree belong to (c), there are m^n subtrees in (c) that are isomorphic to the original subtree by removing nodes with label \$. Therefore, $\tilde{A}^{(n)} = m^n A^{(n)}$ holds.

Therefore, $K(T_m, T_m{}')$, the number of subtrees shared by T_m and $T_m{}'$ is written as

$$\begin{aligned} K(T_m, T_m{}') &= A^{(0)} + \sum_{n=1}^{|V|} (A^{(n)} + \tilde{A}^{(n)}) \\ &= A^{(0)} + \sum_{n=1}^{|V|} (1 + m^n) A^{(n)}. \end{aligned}$$

From the assumption, since $K(T_m, {T_m}')$ is computable in polynomial time, by varying m over $m = 1, \ldots, |V|$, we obtain a system of $|V|$ simultaneous linear equations with $|V|$ variables. $A^{(n)}$ is determined by solving these equations. Meanwhile, since $A^{(|V|)}$ is the solution of TREE KERNEL$^{(|V|)}(T, T')$, TREE KERNEL$^{(|V|)}(T, T')$ is solvable in polynomial time. Therefore, TREE KERNEL(T, T') has been shown to be #P-hard.

Also, TREE KERNEL(T, T') is proven to be in #P since embedding of a particular subtree into T and T' is evaluated in the same way as in the proof of TREE KERNEL$^{(|V|)}(T, T')$ being in #P.

□

3.4 Experiments

In this section, we demonstrate the proposed kernel function on synthetic data and real HTML documents. We used the kernel perceptron [20] as the learning algorithm because of its simplicity, and compared the basic version of the labeled ordered tree, and the extended version with elastic subtrees, and the 'bag of labels (BoL)' kernel, which uses only subtrees of size 1. Since the BoL kernel does not exploit any structural information, we used this kernel as our baseline. Each kernel was combined with the polynomial kernel [81],

$$K_d^{\mathrm{poly}}(T, T') = (1 + K(T, T'))^d,$$

where d is the degree of the polynomial kernel. We did not use the label ambiguity, i.e. $K^{\Sigma}(\sigma, \sigma') = I(\sigma = \sigma')$. All results were measured by leave-one-out cross validation.

3.4.1 Experiment on synthetic data

First, we conducted an experiment on a classification task using synthetic data. The synthetic data was generated so that classification was not possible without recognizing substructures. A tree was classified as positive data if it has both of the two subtrees shown in Figure 3.6, and classified into negative data otherwise. We generated 30 positive trees and 30 negative trees with 30 to 50 nodes and 10 labels.

Table 3.1 shows the averaged results for 5 trials. Bold numbers show the best performance by each kernel. Apparently, the proposed kernel outperforms the baseline kernel by about 20%, which indicates the kernel can recognize substructures. Also,

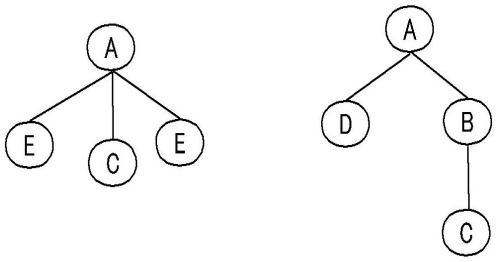

Figure. 3.6 Two substructures included in the trees in positive data.

Table. 3.1 Experimental results on synthetic dataset measured by leave-one-out cross validation.

d	BoL kernel	basic tree kernel
1	**57.8%**	80.5%
2	55.6%	**84.4%**
3	56.7%	80.6%
4	**57.8%**	76.1%
5	55.0%	76.1%

the best performance was obtained in combination with the polynomial kernel with degree $d = 2$. This is probably because the positive tree must have the two subtrees shown in Figure 3.6.

3.4.2 Experiment on HTML documents

Next, we performed an experiment on classification of real HTML documents based on their layouts. Usually, in test classification tasks, a "bag-of-words" representation [36, 67], a feature vector representation based on word counts, is used. In contrast, semi-structured documents such as HTML documents have structural information, e.g. visual information such as layouts, as tree structures by using tags. In this experiment, we classify HTML documents based on their tree structures.

In 2002, we collected 30 HTML pages from IBM Japan's website[*1] and IBM US's website[*2]. Although they are very similar to each other (Figure 3.7), it is expected that differences of the design templates and designers are reflected into subtle differences in the HTML tag structures.

Since we focus only on structural information, we removed the texts and used only

[*1] http://www.ibm.com/jp/
[*2] http://www.ibm.com/

Figure. 3.7 Examples of sampled HTML pages.

the tags[*3]. The collected trees had 10 to 1,500 nodes (but usually around 200 to 400 nodes), and 90 kinds of tags.

In this experiment, besides the BoL kernel and the basic labeled ordered tree kernel, we used the elastic subtree version of the labeled ordered tree kernel. Table 3.2 shows the experimental results. The basic tree kernel outperformed the BoL kernel by 20% and performed the best, which indicates that it captured differences of tree structures well.

Since the basic tree kernel performed best at $d = 4$, but the elastic tree kernel performed best at $d = 3$, it seems that 3 substructures with structural ambiguity are useful for classification.

In this experiment, employing the elastic tree kernel made the results worse. This is probably because of overfitting due to its excessive flexibility. One possible solution is to tune the weights f of the substructures in (2.5). Kashima and Koyanagi reported that the elastic kernel worked well for information extraction tasks [43].

[*3] If we use text, almost 100% performance will be obtained due to differences in the character codes.

Table. 3.2 Experimental results on HTML datasets measured by leave-one-out cross validation.

d	BoL kernel	basic tree kernel	elastic tree kernel
1	41.7%	63.3%	61.7%
2	55.0%	71.7%	60.0%
3	**58.3%**	75.0%	**66.7%**
4	51.7%	**80.0%**	60.0%
5	51.7%	71.7%	63.3%

3.5 Concluding remarks

In this chapter, we discussed kernel designs for tree-structured data. We proposed a new tree kernel for labeled ordered trees by using subtrees as the substructures, and developed an algorithm for computing the kernel. We also extended the proposed kernel function to be able to allow label ambiguities and structural ambiguities. Also, we showed that the kernel computations become #P-complete if we extend the kernel to be able to handle unordered trees,

Chapter 4

Kernel functions for graph-structured data

In this chapter, we discuss construction of kernel functions between two labeled graphs[*1]. Recently, several researchers have proposed kernels between two labeled graphs [42, 44, 21, 23]. One natural definition of substructures in graph-structured data is *subgraph*. However, Gärtner et al. [23] showed negative results for computing kernels based on subgraphs. Therefore, graph kernel based on *label path* would be a fair alternative. For the labeled graph shown in Figure 4.1, a label path is produced by traversing the nodes, and looks like

$$(A, e, A, d, D, a, B, c, D),$$

where the node labels A, B, C, D and the edge labels a, b, c, d, e appear alternately. Numerous label paths are produced by traversing the nodes in every possible way. Especially when the graph has a loop, the dimensionality of the count vector is *infinite*, because traversing may never end.

In the following sections, we will describe a kernel function based on infinite number of label paths. The label paths are produced by random walks on graphs, and thus is regarded as a random variable. Our kernel is defined by averaging a kernel function between labeled paths over all possible label paths, which is regarded as a special case of *marginalized kernels* [76]. The kernel computation is reduced to finding the stationary state of a discrete-time linear system [66], which can be done efficiently by solving simultaneous linear equations with a sparse coefficient matrix.

[*1] Note that they are distinguished from kernels in graph structured input spaces such as kernels between two nodes in a graph, e.g., diffusion kernels [47, 40, 51] or the kernel between two paths in a graph, e.g., path kernels [74].

We especially notice that this computational trick is fundamentally different from the dynamic programming techniques adopted in other kernels (e.g. [13], [43], [55], and [85]). These kernels deal with very large but still *finite* dimensional feature spaces and have parameters to constrain the dimensionality (e.g. the maximum length of subsequences [55]).

In order to investigate how our kernel performs well on the real data, we will show promising results on predicting the properties of chemical compounds.

4.1 Marginalized kernels

When designing kernel functions in the framework of the convolution kernel (2.5), (2.5) is not guaranteed to be finite if $S(\mathbf{x})$ is an infinite set. The *marginalized kernel* [76] is a special case of the convolution kernel with a probability measure $p(\mathbf{s}|\mathbf{x})$ as the weights for substructures $f(\mathbf{s}|\mathbf{x})$, which is described as

$$K(\mathbf{x}, \mathbf{x}') = \sum_{\mathbf{s} \in S(\mathbf{x})} \sum_{\mathbf{s}' \in S'(\mathbf{x})} p(\mathbf{s}|\mathbf{x}) p(\mathbf{s}'|\mathbf{x}') K^S(\mathbf{s}, \mathbf{s}'). \tag{4.1}$$

The marginalized kernel is guaranteed to be valid even in the case of infinite set $S(\mathbf{x})$.

In the original definition of the marginalized kernel [76] is

$$K(\mathbf{x}, \mathbf{x}') = \sum_{\mathbf{z}} \sum_{\mathbf{z}'} K^Z(\mathbf{z}, \mathbf{z}') p(\mathbf{h}|\mathbf{x}) p(\mathbf{h}'|\mathbf{x}'),$$

where $\mathbf{h}$ and $\mathbf{x}$ are the hidden variables and the observed variables in underlying probability models. $\mathbf{z} = [\mathbf{x}, \mathbf{h}]$ is the pair of the observed variables and hidden variables, and $K^Z(\mathbf{z}, \mathbf{z}')$ is the the kernel depending on both visible and hidden variables. The posterior probability $p(\mathbf{h}|\mathbf{x})$ can be interpreted as a feature extractor that extracts informative features for classification from $\mathbf{x}$. The kernel is defined as the expectation of the kernel over all possible values of $\mathbf{h}$ and $\mathbf{h}'$.

4.2 Graph kernel

In this section, we introduce a new kernel function between graphs with node labels and edge labels. At the beginning, let us formally define a labeled graph. Denote by $G = (V, E)$ a labeled directed graph where V is a set of nodes and E is a set of edges. Each node of the graph is uniquely indexed from 1 to $|V|$. Let $v_i \in \Sigma^V$ denote the label of node i and $e_{ij} \in \Sigma_E$ denote the label of the edge from i to j. Figure 4.1

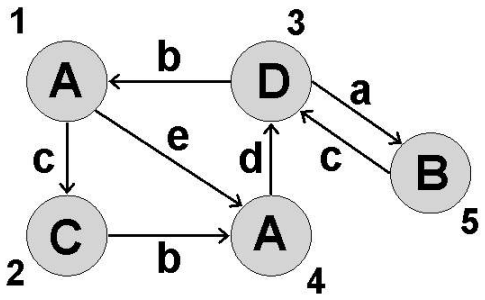

Figure. 4.1 An example of labeled directed graphs

shows an example of the labeled directed graphs that we deal with in this chapter. For the time being, we assume that there are no multiple edges from one node to another. Our task is to construct a kernel function $K(G, G')$ between two labeled graphs $G = (V, E)$ and $G' = (V', E')$.

The marginalized kernel (4.1) in our case is defined as

$$K(G, G') = \sum_{\mathbf{s} \in S(G)} \sum_{\mathbf{s}' \in S(G')} p(\mathbf{s}|G) p(\mathbf{s}'|G') K^S(\mathbf{s}, \mathbf{s}'). \tag{4.2}$$

The substructures $\mathbf{s}$ and $\mathbf{s}'$ are sequences of nodes traversed by random walks on graphs. Also the kernel between substructures K^S is defined as a kernel between two label sequences induced by the node sequences.

4.2.1 Random walks on graphs

A *path* $\mathbf{s} = (s_1, s_2, \ldots, s_\ell)$ associated with graph G is a sequence of the indices of nodes that are natural numbers from 1 to $|V|$. Given a graph G, $\mathbf{s}$ is generated by a random walk as follows: At the first step, s_1 is sampled from the initial probability distribution $p_b(s_1)$. Subsequently, at the i-th step, the next node x_i is sampled subject to the transition probability $p_t(s_i|s_{i-1})$, but the random walk may also end with probability $p_q(s_{i-1})$. Therefore, the following equation holds,

$$\sum_{s_i=1}^{|V|} p_t(s_i|s_{i-1}) + p_q(s_{i-1}) = 1. \tag{4.3}$$

The posterior probability for the path $\mathbf{x}$ is written as

$$p(\mathbf{s}|G) = p_b(s_1) \prod_{i=2}^{\ell} p_t(s_i|s_{i-1}) p_q(s_\ell),$$

where ℓ is the length of $\mathbf{s}$.
When we do not have any prior knowledge, we can set p_b to an uniform distribution over all of the nodes, the transition probability p_t to an uniform distribution over the nodes adjacent to the current node, and the termination probability p_q to a small constant probability. We set $p_q(i) = 1$ if node i has no leaving edges.

A *label path* $\mathbf{h} = (h_1, h_2, \ldots, h_{2\ell-1}) \in (\Sigma^V \Sigma_E)^{\ell-1} \Sigma^V$ is an alternating sequence of node labels and edge labels, that are generated by a random walk on a graph. The posterior probability for the label path $\mathbf{h}$ is described as the sum of the probabilities of the paths emitting $\mathbf{h}$,

$$p(\mathbf{h}|G) = \sum_{\mathbf{s}} \left(p_b(s_1) \prod_{i=2}^{\ell} p_t(s_i|s_{i-1}) p_q(s_\ell) \right) \cdot \left(\delta(v_{s_1} = h_1) \prod_{i=2}^{\ell} \delta(e_{s_{i-1}s_i} = h_{2i-2}) \delta(v_{s_i} = h_{2i-1}) \right),$$

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

4.2.2 Substructure kernel

Next, we define the kernel between substructures K^S assuming that the label sequences $\mathbf{h}$ and $\mathbf{h}'$ are given for two graphs G and G', respectively. Assume that two kernel functions, $K^{\Sigma^V}(v, v')$ and $K^{\Sigma^V}(e, e')$, are readily defined between node labels and edge labels, respectively. We constrain both kernels $K^{\Sigma^V}(v, v'), K^{\Sigma^V}(e, e') \geq 0$ to be nonnegative[*2]. An example of the node label kernels is

$$K^{\Sigma^V}(v, v') = \delta(v = v'). \tag{4.4}$$

If the labels are defined in $\mathbb{R}$, the Gaussian kernel

$$K^{\Sigma^V}(v, v') = \exp\left(\frac{- \parallel v - v' \parallel^2}{2\sigma^2} \right) \tag{4.5}$$

would be a natural choice [68]. Edge kernels are also defined similarly. The substructure kernel is defined as the product of the label kernels as follows.

$$K^S(\mathbf{h}, \mathbf{h}') = \begin{cases} 0 & (\ell \neq \ell') \\ K^{\Sigma^V}(h_1, h'_1) \prod_{i=2}^{\ell} K^{\Sigma^V}(h_{2i-2}, h'_{2i-2}) K^{\Sigma^V}(h_{2i-1}, h'_{2i-1}) & (\ell = \ell') \end{cases}.$$

[*2] This constraint will play an important role in proving the convergence of the marginalized kernel in Section 4.2.4.

4.2.3 Efficient computation

The marginalized kernel (4.2) is the expectation of K_z over all possible $\mathbf{h}$ and $\mathbf{h}'$, which is described as

$$\begin{aligned}
K(G,G') = \sum_{\ell=1}^{\infty} \sum_{\mathbf{h}} \sum_{\mathbf{h}'} K^{\Sigma^V}(h_1,h'_1) \prod_{i=2}^{\ell} K^{\Sigma^V}(h_{2i-2},h'_{2i-2}) K^{\Sigma^V}(h_{2i-1},h'_{2i-1}) \\
\sum_{\mathbf{s}} \left(p_b(s_1) \prod_{i=2}^{\ell} p_t(s_i|s_{i-1}) p_q(s_\ell) \right) \\
\cdot \left(\delta(v_{s_1} = h_1) \prod_{i=2}^{\ell} \delta(e_{s_{i-1}s_i} = h_{2i-2}) \delta(v_{s_i} = h_{2i-1}) \right) \\
\sum_{\mathbf{s}'} \left(p_b(s'_1) \prod_{i=2}^{\ell} p_t(s'_i|s'_{i-1}) p_q(s'_\ell) \right) \\
\cdot \left(\delta(v'_{s'_1} = h_1) \prod_{i=2}^{\ell} \delta(e'_{s'_{i-1}s'_i} = h'_{2i-2}) \delta(v'_{s'_i} = h'_{2i-1}) \right),
\end{aligned}$$

where $\sum_{\mathbf{h}} := \sum_{h_1 \in \Sigma^V} \sum_{h_2 \in \Sigma_E} \cdots \sum_{h_{2\ell-1} \in \Sigma^V}$ and $\sum_{\mathbf{s}} := \sum_{s_1=1}^{|V|} \cdots \sum_{s_\ell=1}^{|V|}$. The straightforward enumeration is obviously impossible, because ℓ spans from 1 to infinity. Nevertheless we have an efficient way to compute this kernel as shown below. By rearranging the terms, the kernel has the following nested structure.

$$\begin{aligned}
K(G,G') = \lim_{L\to\infty} \sum_{\ell=1}^{L} \sum_{s_1,s'_1} b(s_1,s'_1) \Bigg(\sum_{s_2,s'_2} t(s_2,s'_2,s_1,s'_1) \Bigg(\sum_{s_3,s'_3} t(s_3,s'_3,s_2,s'_2) \Bigg(\\
\cdots \Bigg(\sum_{s_\ell,s'_\ell} t(s_\ell,s'_\ell,s_{\ell-1},s'_{\ell-1}) q(s_\ell,s'_\ell) \Bigg) \Bigg) \cdots \Bigg)
\end{aligned} \tag{4.6}$$

where

$$\begin{aligned}
b(s_1,s'_1) &:= p_b(s_1) p'_b(s'_1) K^{\Sigma^V}(v_{s_1}, v'_{s'_1}) \\
t(s_i,s'_i,s_{i-1},s'_{i-1}) &:= p_t(s_i|s_{i-1}) p'_t(s'_i|s'_{i-1}) K^{\Sigma^V}(v_{s_i}, v'_{s'_i}) K^{\Sigma^V}(e_{s_{i-1}s_i}, e_{s'_{i-1}s'_i}) \\
q(s_\ell,s'_\ell) &:= p_q(s_\ell) p'_q(s'_\ell)
\end{aligned} \tag{4.7}$$

Acyclic graphs

Let us first consider the case where we have only directed acyclic graphs. Assume that the nodes in graph G are assigned natural numbers from 1 to $|V|$ in topological

order[*3] (Figure 4.2). Since there are no directed paths from node j to node i if $i < j$, we can employ dynamic programming. Since G and G' are directed acyclic graphs, Equation (4.6) can be written as

$$K(G,G') = \lim_{L\to\infty} \sum_{\ell=1}^{L} \sum_{s_1,s_1'} b(s_1,s_1') \cdot \Bigg(\sum_{s_2>s_1,s_2'>s_1'} t(s_2,s_2',s_1,s_1') \Bigg(\sum_{s_3>s_2,s_3'>s_2'} t(s_3,s_3',s_2,s_2') \Bigg(\cdots \Bigg(\sum_{s_\ell>s_{\ell-1},s_\ell'>s_{\ell-1}'} t(s_\ell,s_\ell',s_{\ell-1},s_{\ell-1}') q(s_\ell,s_\ell') \Bigg) \Bigg) \cdots \Bigg). \tag{4.8}$$

By defining that

$$r(s_1,s_1') := \lim_{L\to\infty} \sum_{\ell=1}^{L} \Bigg(\sum_{s_2>s_1,s_2'>s_1'} t(s_2,s_2',s_1,s_1') \Bigg(\sum_{s_3>s_2,s_3'>s_2'} t(s_3,s_3',s_2,s_2') \Bigg(\cdots \Bigg(\sum_{s_\ell>s_{\ell-1},s_\ell'>s_{\ell-1}'} t(s_\ell,s_\ell',s_{\ell-1},s_{\ell-1}') q(s_\ell,s_\ell') \Bigg) \Bigg) \cdots \Bigg), \tag{4.9}$$

we can rewrite Equation (4.8) as follows,

$$K(G,G') = \sum_{s_1,s_1'} b(s_1,s_1') r(s_1,s_1').$$

The merit of defining Equation (4.9) is that we can exploit the following recursive equation.

$$r(s_1,s_1') = q(s_1,s_1') + \sum_{j>s_1,j'>s_1'} t(j,j',s_1,s_1') r(j,j')$$

Since all nodes are topologically ordered, $r(s_1,s_1')$ for all s_1 and s_1' are efficiently computed by dynamic programming. The time complexity of computing $K(G,G')$ is $O(c\cdot c'\cdot|V|\cdot|V'|)$ where c and c' are the maximum out degree of G and G', respectively.

General directed graphs

When there are cycles in directed graphs, we can not employ the efficient dynamic programming scheme anymore. However, we can reduce the problem to solving a

[*3] The topological sort of graph G can be obtained in $O(|V|+|E|)$ time [15].

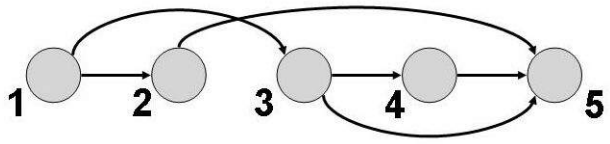

Figure. 4.2 A topologically sorted directed acyclic graph. Kernels can be efficiently computed by dynamic programming going through from right to left.

system of simultaneous linear equations.

Let us further simplify (4.6) as

$$K(G, G') = \lim_{L\to\infty} \sum_{\ell=1}^{L} \sum_{s_1, s_1'} b(s_1, s_1') r_\ell(s_1, s_1'),$$

where for $\ell \geq 2$,

$$r_\ell(s_1,_1') := \Bigg(\sum_{s_2, s_2'} t(s_2, s_2', s_1, s_1') \Bigg(\sum_{s_3, s_3'} t(s_3, s_3', s_2, s_2') \Bigg(\cdots \Bigg(\sum_{s_\ell, s_\ell'} t(s_\ell, s_\ell', s_{\ell-1}, s_{\ell-1}') q(s_\ell, s_\ell') \Bigg) \Bigg) \cdots \Bigg),$$

and $r_1(s_1, s_1') := q(s_1, s_1')$. Replacing the order of the summations, we have

$$\begin{aligned} K(G, G') &= \sum_{s_1, s_1'} b(s_1, s_1') \lim_{L\to\infty} \sum_{\ell=1}^{L} r_\ell(s_1, s_1') \\ &= \sum_{s_1, s_1'} b(s_1, s_1') \lim_{L\to\infty} R_L(s_1, s_1'), \end{aligned} \tag{4.10}$$

where

$$R_L(s_1, s_1') := \sum_{\ell=1}^{L} r_\ell(s_1, s_1').$$

Thus we need to compute $R_\infty(s_1, s_1')$ to obtain $K(G, G')$.

Now, let us restate this problem in terms of the linear system theory [66]. The following recursive relationship holds between r_k and r_{k-1} ($k \geq 2$):

$$r_k(s_1, s_1') = \sum_{i,j} t(i, j, s_1, s_1') r_{k-1}(i, j). \tag{4.11}$$

Using (4.11), the recursive relationship for R_L also holds as follows,

$$\begin{aligned} R_L(s_1, s_1') &= r_1(s_1, s_1') + \sum_{k=2}^{T} r_k(s_1, s_1') \\ &= r_1(s_1, s_1') + \sum_{k=2}^{T} \sum_{i,j} t(i, j, s_1, s_1') r_{k-1}(i, j) \\ &= r_1(s_1, s_1') + \sum_{i,j} t(i, j, s_1, s_1') R_{L-1}(i, j). \end{aligned} \tag{4.12}$$

Thus, R_L is perceived as a discrete-time linear system [66] evolving as the time L increases. Assuming that R_L converges (See Section 4.2.4 for the convergence condition), we have the following equilibrium equation,

$$R_\infty(s_1, s_1') = r_1(s_1, s_1') + \sum_{i,j} t(i, j, s_1, s_1') R_\infty(i, j). \tag{4.13}$$

Therefore, the computation of the marginalized kernel finally comes down to solving the simultaneous linear equations (4.13), and to substituting the solutions into (4.10).

Now, let us restate the above discussion by using matrix notations. Let $\mathbf{b}$, $\mathbf{q}$, $\mathbf{r}_1$, and $\mathbf{r}_\infty$ be $|V| \cdot |V'|$-dimensional vectors,

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

respectively. Let the transition probability matrix T be a $|V||V'| \times |V||V'|$ matirix,

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

Equation (4.6) can be rewritten as follows,

$$\begin{aligned} K(G, G') &= \lim_{L \to \infty} \sum_{\ell=0}^{L} \mathbf{q}^T T^\ell \mathbf{b} \\ &= \mathbf{q}^T \left(\lim_{L \to \infty} \sum_{\ell=0}^{L} T^\ell \right) \mathbf{b} \\ &= \mathbf{r}_\infty^T \mathbf{b}. \end{aligned} \tag{4.14}$$

Similarly, the recursive equation (4.13) is rewritten as

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

Solving this equation,

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

Finally, the matrix form of the kernel is

$$K(G, G') = (I - T)^{-1}\mathbf{r}_1\mathbf{b}. \tag{4.15}$$

Computing the kernel requires solving a linear equation or a matrix inversion with a $|V||V'| \times |V||V'|$ coefficient matrix. However, the matrix is actually sparse because the number of non-zero elements is less than $c \cdot c' \cdot |V| \cdot |V'|$ where c and c' are the maximum out degree of G and G', respectively. Therefore, we can employ various kinds of efficient numerical algorithms that exploit sparsity [10]. In our implementation, we employed a simple iterative method that updates current solutions by using (4.12) until convergence starting from $R_1(s_1, s_1') = r_1(s_1, s_1')$.

4.2.4 Convergence condition

Since loops are allowed in general directed graphs, infinite number of path can be generated. Therefore, some convergence conditions are needed to justify Equation (4.13). The following theorem holds.

Theorem 2 *The infinite sequence* $\lim_{L\to\infty} R_L(s_1, s_1')$ *converges for any* $s_1 \in \{1, \cdots, |V|\}$ *and* $s_1' \in \{1, \cdots, |V'|\}$, *if the following inequality holds for* $i_0 \in \{1, \cdots, |V|\}$ *and* $j_0 \in \{1, \cdots, |V'|\}$,

$$\sum_{i=1}^{|V|}\sum_{j=1}^{|V'|} t(i, j, i_0, j_0)q(i, j) < q(i_0, j_0). \tag{4.16}$$

(proof) $R_\ell(s_1, s_1')$ can also be described as

$$\begin{aligned} R_\ell(s_1, s_1') \;=\; & q(s_1, s_1') + \sum_{i=2}^{\ell}\sum_{s_2, x_2'} t(s_2, x_2', s_1, s_1')\Bigg(\sum_{s_3, x_3'} t(s_3, x_3', s_2, x_2')\Bigg(\\ & \cdots\left(\sum_{s_i, x_i'} t(s_i, x_i', s_{i-1}, x_{i-1}')q(s_i, x_i')\right)\cdots\Bigg) \\ := \; & q(s_1, s_1') + \sum_{i=2}^{\ell} u_i(s_1, s_1'), \end{aligned}$$

where u_i $(i \geq 2)$ denotes the i-th term in the summation. According to the ratio test, this series converges if $\limsup_{i\to\infty} |u_i(s_1, s_1')|/|u_{i-1}(s_1, s_1')| < 1$. Since $K(v, v')$

and $K(e, e')$ are nonnegative as previously defined, and $u_i(s_1, s'_1) \geq 0$, what we need to prove is $\limsup_{i\to\infty} u_i(s_1, s'_1)/u_{i-1}(s_1, s'_1) < 1$. For this purpose, it is sufficient to show

$$\frac{u_i(s_1, s'_1)}{u_{i-1}(s_1, s'_1)} < 1 \tag{4.17}$$

for all i. The two variables in (4.17) are written as

$$u_i(s_1, s'_1) = \sum_{s_{i-1}, x'_{i-1}} a_{i-1}(s_{i-1}, x'_{i-1}, s_1, s'_1) \sum_{s_i, x'_i} t(s_i, x'_i, s_{i-1}, x'_{i-1}) q(s_i, x'_i), \tag{4.18}$$

$$u_{i-1}(s_1, s'_1) = \sum_{s_{i-1}, x'_{i-1}} a_{i-1}(s_{i-1}, x'_{i-1}, s_1, s'_1) q(s_{i-1}, x'_{i-1}), \tag{4.19}$$

where

$$a_{i-1}(s_{i-1}, x'_{i-1}, s_1, s'_1) := \sum_{s_2, x'_2} t(s_2, x'_2, s_1, s'_1) \Bigg(\sum_{s_3, x'_3} t(s_3, x'_3, s_2, s'_2) \Bigg(\cdots \Bigg(\sum_{s_{i-2}, x'_{i-2}} t(s_{i-2}, x'_{i-2}, s_{i-3}, s'_{i-3}) t(s_{i-1}, x'_{i-1}, s_{i-2}, x'_{i-2}) \Bigg) \cdots \Bigg).$$

Note that $a_{i-1}(s_{i-1}, x'_{i-1}, s_1, s'_1) \geq 0$ as well. Substituting (4.18) and (4.19) into (4.17), we have

$$\begin{aligned} &\left(\sum_{s_{i-1}, x'_{i-1}} a_{i-1}(s_{i-1}, x'_{i-1}, s_1, s'_1) \right) \left(\sum_{s_i, x'_i} t(s_i, x'_i, s_{i-1}, x'_{i-1}) q(s_i, x'_i) \right) \\ < &\sum_{s_{i-1}, x'_{i-1}} a_{i-1}(s_{i-1}, x'_{i-1}, s_1, s'_1) q(s_{i-1}, x'_{i-1}). \end{aligned}$$

Both sides of this inequality are the linear combinations of $a_{i-1}(s_{i-1}, x'_{i-1}, s_1, s'_1)$ with nonnegative coefficients. In order to prove the inequality, it is sufficient to prove the following inequalities for coefficients,

$$\sum_{i,j} t(i, j, i_0, j_0) q(i, j) < q(i_0, j_0), \quad \forall i_0, j_0,$$

which we have assumed to hold in (4.16).

□

The condition (4.16) seems rather complicated, but we can obtain a simpler condition, if the termination probabilities are constant over all nodes.

Lemma 2 *If $p_q(i) = p'_q(j) = \gamma$ for any i and j, the infinite sequence $\lim_{L\to\infty} R_L(s_1, s'_1)$ converges if*

$$K^{\Sigma^V}(v, v')K^{\Sigma^V}(e, e') < \frac{1}{(1-\gamma)^2}. \tag{4.20}$$

(proof) Due to the assumption, $q(i, j) = q(i_0, j_0) = \gamma^2$. Thus it is sufficient to prove $\sum_{i,j} t(i, j, i_0, j_0) < 1$, that is,

$$\sum_{i,j} p_t(i|i_0)p'_t(j|j_0)K^{\Sigma^V}(v_i, v_j)K^{\Sigma^V}(e_{i_0 i}, e_{j_0 j}) < 1. \tag{4.21}$$

Due to the relation (4.3), we observe that

$$\sum_{i,j} p_t(i|i_0)p'_t(j|j_0) = \sum_i p_t(i|i_0) \sum_j p'_t(j|j_0) = (1-\gamma)^2.$$

Thus (4.21) holds if all the coefficients satisfy $K^{\Sigma^V}(v_i, v_j)K^{\Sigma^V}(e_{i_0 i}, e_{j_0 j}) < \frac{1}{(1-\gamma)^2}$.

□

Apparently, the above lemma holds for $\lambda > 0$ if $K^{\Sigma^V}(\cdot, \cdot) \leq 1$ and $K^{\Sigma^V}(\cdot, \cdot) \leq 1$. Standard label kernels such as (4.4) and (4.5) satistfy this condition.

4.3 Extensions

In this section, we describe some extensions of our graph kernel to allow multiple edges between nodes and approximate path matching.

4.3.1 Allowing multiple edges between nodes

In the previous section, we assumed that there are no multiple edges from one node to another. However, slight modification makes the kernel allow more than two edges. Suppose that there are $M_{s_{i-1}s_i}$ directed edges from node x'_{i-1} to node x'_i, each of whose labels is $e^m_{s_{i-1}s_i}$, and each of whose transition probabilities is $p^m_t(s_i|s_{i-1})(m = 1, 2, \ldots, M_{s_{i-1}s_i})$. Instead of Equation (4.7), by considering all pair of $e^m_{s_{i-1}s_i}$ and $e^m_{x'_{i-1}x'_i}$, we have only to redefine $t(s_i, x'_i, s_{i-1}, x'_{i-1})$ as follows,

$$t(s_i, x'_i, s_{i-1}, x'_{i-1}) := K^{\Sigma^V}(v_{s_i}, v'_{x'_i}) \cdot \sum_{m=1}^{M_{s_{i-1}s_i}} \sum_{m'=1}^{M'_{s_{i-1}s_i}} p^m_t(s_i|s_{i-1})p'^{m'}_t(x'_i|x'_{i-1})K^{\Sigma^V}(e^m_{s_{i-1}s_i}, e^{m'}_{x'_{i-1}x'_i}).$$

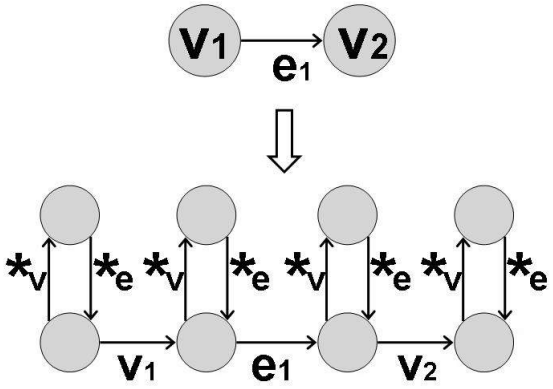

Figure. 4.3 An example of the conversion of a graph. Two new nodes are created from each of the original node, and the label of the original node is assigned as the edge label between the nodes. The loop labeled two contiguous wild card symbols '$*_e*_v$' is created for allowing insertions in label paths.

4.3.2 Approximate matching graph kernels

The graph kernels discussed so far do not consider noncontiguous label sequence on graphs. However, in biological sequence analysis, edit operations such as mutation, deletion, and insertion of symbols are considered to handle approximate matching of sequences that are not completely identical to each other [19]. This idea is also employed in designing the kernels between sequences, e.g. string kernels [55]. In our graph kernel, we can also apply the same idea by converting the graphs appropriately. Approximate path matching in the graph kernel is incorporated by introducing wild card symbols, '$*_v$' for nodes, and '$*_e$' for edges, and by converting the graphs into edge labeled graphs with extra nodes and extra edges to generate paths with the wild card symbols. The conversion of a graph is implemented as follows. Each of the nodes is split into two nodes, and let the transition between the nodes emit the label of the original node. Next, for considering gaps in label sequences, extra nodes are created to make loops that emit the wild card symbols, $*_e$ and $*_v$. Note that new transition matrix T is still sparse enough although this conversion results in $4|V|$ nodes and seems to make the kernel computations heavier. An example of the conversion of a graph is shown in Figure 4.3. Note that in the case where graphs have no node labels, this conversion is reduced to the one proposed in [23] that creates self loops emitting wild card symbols.

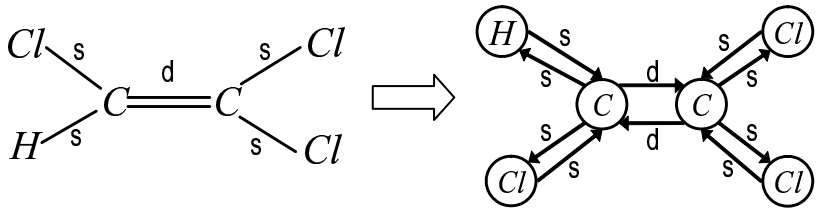

Figure. 4.4 A chemical compound is conventionally represented as an undirected graph (left). Atom types and bond types correspond to node labels and edge labels, respectively. The edge labels 's' and 'd' denote single and double bonds, respectively. As our kernel assumes a directed graph, undirected edges are replaced by directed edges (right).

4.4 Experiments

We applied our kernel to prediction of the properties of chemical compounds. A chemical compound can naturally be represented as an undirected graph by considering the atom types as the node labels, e.g. C, Cl and H, and the bond types as the edge labels, e.g. s (single bond) and d (double bond). To apply our graph kernel, we replaced each of the undirected edges by two directed edges (Figure 4.4) since the kernel assumes directed graphs.

4.4.1 Pattern discovery algorithm

We compare our graph kernel with the pattern-discovery (PD) method by Kramer and De Raedt [49] that is one of the state-of-the-art methods in predictive toxicology. As in our graph kernel, each feature is constructed as the count that a particular label path appears in the graph[*4]. There are other methods which count more complicated substructures such as subgraphs [33], but we focus only on paths [49] whose features are similar to ours.

Assume that we have n graphs $G_1, \ldots, G_n$. Also let us define $\#(\mathbf{h}, G)$ as the number of appearances of a label path $\mathbf{h}$ in G. The PD method identifies a set of all label paths $\mathcal{H}$ which appear in more than m graphs:

$$\mathcal{H} = \{\mathbf{h} \mid \sum_{i=1}^{n} \delta\left(\#\left(\mathbf{h}, G_i\right) > 0\right) \geq m\},$$

[*4] Notice that the definition of label paths is different from ours in several points, e.g. a node will not be visited twice in a path. See [49] for details.

where the parameter m is called the minimum support parameter. Furthermore, it is possible to add extra conditions, e.g., selecting only the paths frequent in a certain class, and scarce in the other classes. Then the feature vector of G based on the identified label paths is built as

$$G \to (\#(\mathbf{s}_1, G), \ldots, \#(\mathbf{s}_{|\mathcal{H}|}, G)), \tag{4.22}$$

whose dimensionality is the cardinality of $\mathcal{H}$. The PD method is useful for extracting comprehensive features. However, as the minimum support parameter gets smaller, the dimensionality of feature vectors becomes so huge that a prohibitive amount of computation is required. Therefore, the user has to control the minimum support parameter m, such that the feature space does not lose necessary information and, at the same time, computation stays feasible.

The PD method contrasts markedly with our method. Our kernel method puts emphasis on dealing with infinite, but less interpretable features, while the PD method trys to extract a relatively small number of meaningful features. Looking at the algorithms, our method is so simple that it is described by just one equation (4.13), while the PD method's algorithm is rather complicated [18].

4.4.2 Datasets

We used two datasets, the PTC dataset [28] and the Mutag dataset [73].

The PTC dataset is the results of the following pharmaceutical experiments. Each of 417 compounds is given to four types of test animals: Male Mouse (MM), Female Mouse (FM), Male Rat (MR) and Female Rat (FR). According to their carcinogenicity, each compound is assigned one of the following labels: $\{\mathrm{EE}, \mathrm{IS}, \mathrm{E}, \mathrm{CE}, \mathrm{SE}, \mathrm{P}, \mathrm{NE}, \mathrm{N}\}$, where CE, SE and P indicate "relatively active", and NE and N indicate "relatively inactive", and EE, IS and E indicate "cannot be decided". In order to simplify the problem, we relabeled CE, SE and P as "positive", and NE and N as "negative". The task is to predict whether a given compound is positive or negative for each type of test animals. Thus we eventually had four two-class classification problems.

In the Mutag dataset, the task is defined to be a two-class classification problem to predict whether each of the 188 compounds has mutagenicity or not.

The statistics of the datasets are summarized in Table 4.1.

Table. 4.1 Several statistics of the datasets, such as numbers of the positive examples (#positive) and the negative examples (#negative), the maximum degrees (max. degree), the maximum sizes of graphs (max. $|V|$), the average sizes of graphs (avg. $|V|$), and the numbers of the node labels ($|\Sigma^V|$) and the edge labels ($|\Sigma_E|$).

	MM	FM	MR	FR	MUTAG
#POSITIVE	129	143	152	121	125
#NEGATIVE	207	206	192	230	63
MAX. $\|V\|$	109	109	109	109	40
AVG. $\|V\|$	25.0	25.2	25.6	26.1	31.4
MAX. DEGREE	4	4	4	4	4
$\|\Sigma^V\|$	21	19	19	20	8
$\|\Sigma_E\|$	4	4	4	4	4

4.4.3 Experimental settings and results

Assuming no prior knowledge, we defined the probability distributions for random walks as follows. The initial probabilities were simply uniform, i.e. $p_b(s_1) = 1/|V|$, $\forall s_1$. The termination probabilities were set to a constant γ over all nodes. The transition probabilities $p_t(s_i|s_{i-1})$ were set to the uniform distibution over the adjacent nodes. We used Equation (4.4) as the label kernels. In solving the simultanous equations, we employed a simple iterative method (4.12). In our observation, 20-30 iterations were sufficient for convergence in all cases. As the classification algorithm, we used the voted kernel perceptron [20], whose the performance is known to be comparable to that of the Support Vector Machine. In the pattern discovery method, the minimum support parameter was set to one of $\{0.5\%, 1\%, 3\%, 5\%, 10\%, 20\%\}$ of the number of compounds, and the simple dot product in the feature space (4.22) was used as the kernel function. In our graph kernel, the termination probability γ was varied from 0.1 to 0.9.

Table 4.2 and Table 4.3 show the classification accuracies in the five two-class problems measured by leave-one-out cross validation. No general tendencies were found to conclude which method is better (the PD was better in MR, FR and Mutag, but our method was better in MM and FM). Thus it would be fair to say that the performances were comparable in this small set of experiments. Even though we could not show that our method is constantly better, this result is still appealing, because the advantage of our method lies in its simplicity both in concepts and in computational procedures.

Table. 4.2 Classification accuracies (%) by the pattern discovery method. 'MinSup' shows the ratio of the minimum support parameter to the number of compounds m/n.

MinSup	MM	FM	MR	FR	Mutag
0.5%	60.1	57.6	61.3	**66.7**	88.3
1 %	**61.0**	**61.0**	**62.8**	63.2	87.8
3 %	58.3	55.9	60.2	63.2	**89.9**
5 %	60.7	55.6	57.3	63.0	86.2
10 %	58.9	58.7	57.8	60.1	84.6
20%	**61.0**	55.3	56.1	61.3	83.5

Table. 4.3 Classification accuracies (%) by our graph kernel. The parameter γ is the termination probability of random walks, which controls the effect of the length of label paths.

γ	MM	FM	MR	FR	Mutag
0.1	62.2	59.3	57.0	62.1	84.3
0.2	62.2	61.0	57.0	62.4	83.5
0.3	64.0	61.3	56.7	62.1	**85.1**
0.4	**64.3**	61.9	56.1	63.0	**85.1**
0.5	64.0	61.3	56.1	64.4	83.5
0.6	62.8	61.9	54.4	65.8	83.0
0.7	63.1	62.5	54.1	63.2	81.9
0.8	63.4	**63.4**	54.9	64.1	79.8
0.9	62.8	61.6	**58.4**	**66.1**	78.7

4.5 Related work

In this section, we review related works, some of which explicitly aim to design graph kernels, and some do not. Also, we describe the relation between our graph kernel and them. They are very close to each other, and based on the same philosophy that similarity between graphs are reduced to the similarities between the paths on the graphs.

Gärtner et al. [23] also constructed graph kernels using path counts as the features of a graph. Since their kernel is not in probabilistic framework, p_t, p_b and p_q need not to satisfy the conditions for probability distributions. By setting

$$\begin{aligned} p_t(s_i|s_{i-1}) &\to \sqrt{\lambda_i} p_t(s_i|s_{i-1}) \\ p_b(s_1) &\to 1 \\ p_q(s_\ell) &\to 1, \end{aligned}$$

our kernel is equivalent to their kernel, and λ_i controls the convergence of the kernel function. One of the characteristics of the kernel is that λ_i varies with i. However, arbitrary choice of λ_i can not always be computed efficiently. When $\lambda_i = \lambda$, Equation (4.14) becomes *geometric kernels*,

$$K(G, G') = \sum_i \sum_j \left[\lim_{L \to \infty} \sum_{\ell=0}^{L} \lambda T^\ell \right]_{ij},$$

and the computation is reduced to matrix inversion like Equation (4.15). Similarly, setting

$$\lambda_i = \frac{\beta}{i}$$

leads to *exponential kernels*

$$K(G, G') = \sum_i \sum_j \left[\lim_{L \to \infty} \sum_{\ell=1}^{L} \frac{(\beta T)^\ell}{\ell!} \right]_{ij} = \sum_i \sum_j \left[e^{\beta T} \right]_{ij}.$$

By diagonalizing $T = V^{-1} D V$, this can be computed by exploiting $e^{\beta T} = V^{-1} e^{\beta D} V$. Gärtner et al. [23] also showed NP-completeness of computing graph kernels based not on paths, but on *subgraphs*. This fact gives us one rationale for using paths as the substructures of graphs.

Gebara and Kondor [24] proposed *probability product kernels* that define kernels between two probability distributions $\mathbf{p}$ and $\mathbf{p}'$.

$$K(p, p') = \int_\Omega p(\mathbf{x})^\rho p'(\mathbf{x})^\rho d\mathbf{x} \tag{4.23}$$

When $\rho = 1$, the kernel is called *expected likelihood kernel*, and becomes the probability with which two probability distributions generate $\mathbf{x}$ independently.

Also, when $\rho = 1/2$, the kernel is called Bhattacharrya kernel, and $H(p, p') = \sqrt{2 - 2K(p, p')}$ becomes Hellinger distance,

$$H(p, p') = \left[\int \left(\sqrt{p(\mathbf{x})} - \sqrt{p'(\mathbf{x})} \right)^2 d\mathbf{x} \right]^{1/2}.$$

Equation (4.23) can be written as

$$K(p, p') = \int_\Omega \int_{\Omega'} I(\mathbf{x} = \mathbf{y}) p(\mathbf{x})^\rho p'(\mathbf{y})^\rho d\mathbf{x} d\mathbf{y}.$$

We can say that our graph kernel marginalizes a string kernel for two independent Markov models defined over two graphs, and the expected likelihood kernel corresponds to using the identity kernel as the string kernel. Under such considerations, one of the possibilities of extending the marginalized kernel would be

$$K(\mathbf{x}, \mathbf{x}') = \sum_{\mathbf{h}} \sum_{\mathbf{h}'} K^Z(\mathbf{z}, \mathbf{z}') p(\mathbf{h}|\mathbf{x})^\rho p(\mathbf{h}'|\mathbf{x}')^\rho.$$

Although graph kernels are not intended, Lyngso et al. [56] gave an expected likelihood kernel for hidden Markov models (HMM), the probability with which two HMMs emit the same sequence independently. Their recursive equation is almost the same as ours (4.13) except for a few differences such as the existence of edge labels.

If we consider the kernel $K^Z(\mathbf{z}, \mathbf{z}')$ to be a joint distribution $q(\mathbf{h}, \mathbf{h}')$ that emits a pair of sequences $\mathbf{h}$ and $\mathbf{h}'$, it becomes an instance of *rational kernels* [16],

$$K(\mathbf{x}, \mathbf{x}') = \sum_{\mathbf{h}} \sum_{\mathbf{h}'} q(\mathbf{h}, \mathbf{h}') p(\mathbf{h}|\mathbf{x}) p'(\mathbf{h}'|\mathbf{x}'),$$

that define a kernel between two probabilistic automata $p(\mathbf{h}|\mathbf{x})$ and $p'(\mathbf{h}'|\mathbf{x}')$ via a probabilistic transducer $q(\mathbf{h}, \mathbf{h}')$. The rational kernel is not limited to probabilistic setting, and offers an unified framework for designing weighted automata via weighted transducers by using the notation of semirings. The above interpretation corresponds to using probability semiring. Changing semirings, e.g. changing probability semiring to Boolean semiring, i.e. $\times$ to $\wedge$ and $+$ to $\vee$, does not affect the algorithms. Although Cortes et al. [16] provided an algorithm only for directed acyclic cases, our algorithm can be easily modified for rational kernels for cyclic cases.

4.6 Concluding remarks

This chapter discussed designing kernels function between directed graphs with node labels and edge labels. We defined the kernel by using random walks on graphs in the framework of the marginalized kernel, and reduced the computation of the kernel to solving a system of linear simultaneous equations. As contrasted with the pattern-discovery method, our kernel takes into account all possible label paths without computing feature values explicitly.

Chapter 5

Kernel-based labeling of structured data

Sequence labeling is one of the important problems widely seen in the areas of natural language processing (NLP), bioinformatics and Web data analysis. Labeling problems generalize supervised classification problems, since not only the label of one hidden variable, but the labels of a set of hidden variables are to be predicted. Labeling problems for sequences have been extensively studied for years. However, there has been almost no significant work on labeling more general structured data such as trees and graphs. In this chapter, we consider kernel-based approaches for labeling problems with general structured data.

In some labeling problems, combinations of local features such as bi-grams are not sufficient for modeling long-distance dependencies such as idioms in NLP, or motifs in bioinformatics. However, in all of the methods proposed so far, there is a fundamental problem that they can handle only local features considering small numbers of hidden variables.

In this chapter, inspired by the idea of a point-wise log-loss function proposed by Kakade et al. [37], we propose *marginalized labeling perceptrons* that solve the label prediction problem by using the marginalized feature vectors. Marginalized labeling perceptrons realize point-wise label prediction, and are fully kernelized by using marginalized kernel functions [76] with long-distance dependencies among hidden variables.

Also, in this chapter, we propose several marginalized kernels used in the kernel marginalized labeling perceptrons for labeling various structured data such as sequences, trees, and graphs, by extending the kernels we developed in the previous

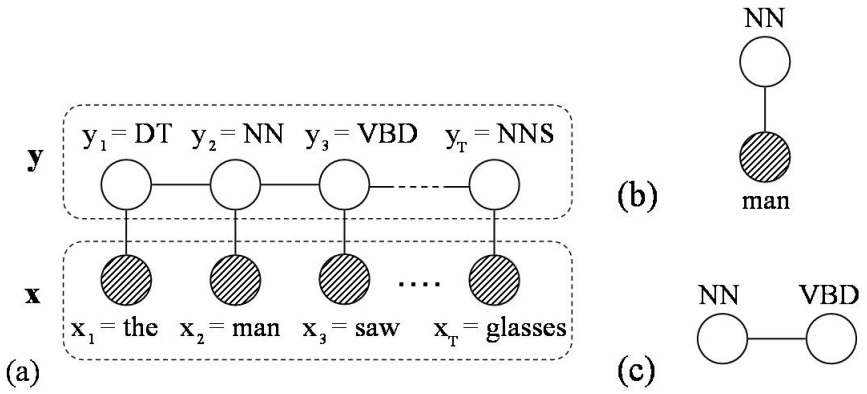

Figure. 5.1 (a) Graphical model representation of a labeled sequence under a first-order Markov assumption. Shadowed nodes indicate observable variables, and white nodes indicate hidden variables. The sentence "the, man, saw ,$\cdots$, glasses." is the label sequence $\mathbf{x}$ of the observable variables. The part of speech tag sequence "DT, NN, VBD, ..., NNS" is the label sequence $\mathbf{y}$ of the hidden variables. (b) A pair of an observable variable and a hidden variable. (c) A pair of two hidden variables.

chapters.

Finally, we show some promising results for experiments on sequence labeling and tree labeling for real-world tasks which require structure information.

5.1 Labeling problem

The labeling problem is defined to be the problem of learning a function that maps the observable variables $\mathbf{x} = (x_1, x_2, \ldots, x_T)$ to the hidden variables $\mathbf{y} = (y_1, y_2, \ldots, y_T)$ in structured data, where each $x_t \in \Sigma_x$ and each $y_t \in \Sigma_y$. For instance, in part-of-speech tagging, x_t represents the t-th word, and y_t represents the part-of-speech tag of the t-th word (Figure 5.1). The learner may exploit a set of training examples $E = (\mathbf{e}^{(1)}, \mathbf{e}^{(2)}, \ldots, \mathbf{e}^{(|E|)})$, where $\mathbf{e}^{(i)} = (\mathbf{x}^{(i)}, \mathbf{y}^{(i)})$ is the i-th example, and $|\mathbf{x}^{(i)}| = |\mathbf{y}^{(i)}| = T^{(i)}$.

Conventionally, in sequence labeling problems, generative models such as Hidden Markov Models (HMMs) have been used. However, they tend to need a lot of data since they target a more difficult problems of estimating joint probabilities of observable variables and hidden variables. Also, they can not handle overlapping features naturally, since observable variables must be independent of each other given the labels of the hidden variables. Recently, conditional models such as Maximum Entropy Markov Models (MEMMs) [58] and Conditional Random Fields (CRFs) [52] are

attracting considerable attention, since they are more suitable for the purpose of predicting the labels of hidden variables given the labels of observable variables. Recently, [12] proposed the Hidden Markov (HM) Perceptron, a more efficient sequence labeling algorithm based on perceptrons as an alternative to CRFs and MEMMs. SVM-based algorithms [6] and boosting-based algorithms [4] have also been proposed.

5.2 Hidden Markov perceptrons

Collins [12] introduced the HM-Perceptron, a discriminative learning algorithm for sequence labeling as an alternative to CRFs and MEMMs. It can be trained efficiently by processing the training examples one by one. Its ability was shown to be comparable to CRFs [5]. The key idea of the HM-Perceptron is to interpret the mapping as a binary classification[*1], i.e. ${\Sigma_x}^T \times {\Sigma_y}^T \rightarrow \{+1, -1\}$. Let F be a set of features for vector representation of $(\mathbf{x}, \mathbf{y})$. Let $\phi_f(\mathbf{x}, \mathbf{y})$ be the number of times a feature $f \in F$ appears in $(\mathbf{x}, \mathbf{y})$, and $\Phi(\mathbf{x}, \mathbf{y})$ be their vector form. Given $\mathbf{x}$, the HM-Perceptron outputs the prediction $\hat{\mathbf{y}}$ according to the current weights of the features:

$$\hat{\mathbf{y}} = \underset{\mathbf{y} \in {\Sigma_y}^T}{\operatorname{argmax}} \sum_{f \in F} w_f \phi_f(\mathbf{x}, \mathbf{y}) = \underset{\mathbf{y} \in {\Sigma_y}^T}{\operatorname{argmax}} \big\langle \mathbf{w}, \Phi(\mathbf{x}, \mathbf{y}) \big\rangle, \tag{5.1}$$

where w_f is the weight of a feature f, and $\mathbf{w}$ is the vector form of the weights. Under a first-order Markov assumption, two types of features, such as pairs of an observable variable and a hidden variable (Figure 5.1(b)), and pairs of two hidden variables (Figure 5.1(c)) are used.

Starting at $\mathbf{w} = 0$, the weights are updated by using

$$\mathbf{w}^{new} = \mathbf{w}^{old} + \Phi(\mathbf{x}^{(i)}, \mathbf{y}^{(i)}) - \Phi(\mathbf{x}^{(i)}, \hat{\mathbf{y}}^{(i)}), \tag{5.2}$$

when the prediction for $\mathbf{e}^{(i)}$ is wrong, i.e. $\hat{\mathbf{y}}^{(i)} \neq \mathbf{y}^{(i)}$. Equation (5.1) is also written in a dual form as follows by using the weights of the examples α.

$$\hat{\mathbf{y}} = \underset{\mathbf{y} \in {\Sigma_y}^T}{\operatorname{argmax}} \sum_{j=1}^{|E|} \sum_{\tilde{\mathbf{y}} \in {\Sigma_y}^T} \alpha_j(\tilde{\mathbf{y}}) \big\langle \Phi(\mathbf{x}^{(j)}, \tilde{\mathbf{y}}), \Phi(\mathbf{x}, \mathbf{y}) \big\rangle \tag{5.3}$$

Starting with $\alpha = \mathbf{0}$, the updating rules are rewritten as

- $\alpha_i^{new}(\mathbf{y}^{(i)}) = \alpha_i^{old}(\mathbf{y}^{(i)}) + 1$

[*1] Weston et al. [86] have proposed another general framework that aims at learning the direct mapping from $\Phi_x(\mathbf{x})$ to $\Phi_y(\mathbf{y})$ with arbitrary kernels.

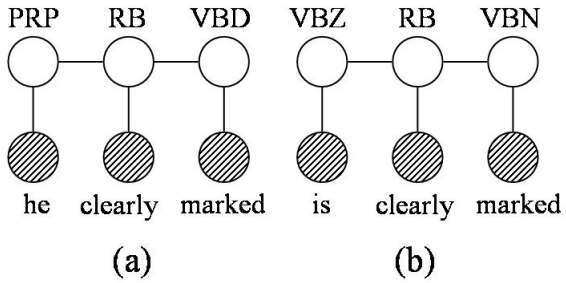

Figure. 5.2 Features for labeling sequences (of length 3). Bi-gram features are not sufficient for disambiguation of the label of "marked".

- $\alpha_i^{new}(\hat{\mathbf{y}}^{(i)}) = \alpha_i^{old}(\hat{\mathbf{y}}^{(i)}) - 1,$

when $\hat{\mathbf{y}}^{(i)} \neq \mathbf{y}^{(i)}$. $(\mathbf{x}^{(i)}, \hat{\mathbf{y}}^{(i)})$ is called a pseudo-negative example since it acts like an negative example.

Now, we discuss using features that consider long-distance dependencies in the HM-Perceptron. Sometimes, bi-gram features (Figure 5.1(b)(c)) are not sufficient for modeling long-distance dependencies such as idioms in NLP, or motifs in bioinformatics. What if we employ longer features such as in Figure 5.2? Let us assume that we allow features of lengths up to d. Let $\mathbf{x}_{t_1}^{t_2}$ be the subset of the observable variables from position t_1 to position t_2 of $\mathbf{x}$, and $\mathbf{y}_{t_1}^{t_2}$ is defined accordingly. Taking into account that all features depend on at most d consecutive hidden variables, $\Phi(\mathbf{x}, \mathbf{y})$ is decomposed as

$$\begin{aligned}\Phi(\mathbf{x}, \mathbf{y}) = &\Phi(\mathbf{x}_1^d, \mathbf{y}_1^d) - \Phi(\mathbf{x}_2^d, \mathbf{y}_2^d) \\ &+\Phi(\mathbf{x}_2^{d+1}, \mathbf{y}_2^{d+1}) - \Phi(\mathbf{x}_3^{d+1}, \mathbf{y}_3^{d+1}) \\ &\cdots \\ &+\Phi(\mathbf{x}_{T-d}^{T-1}, \mathbf{y}_{T-d}^{T-1}) - \Phi(\mathbf{x}_{T-d+1}^{T-1}, \mathbf{y}_{T-d+1}^{T-1}) \\ &+\Phi(\mathbf{x}_{T-d+1}^{T}, \mathbf{y}_{T-d+1}^{T}).\end{aligned} \tag{5.4}$$

Substituting Equation (5.4) into Equation (5.1), the argmax operation in Equation (5.1) is performed via the following dynamic programming.

$$\begin{aligned}\max_{\mathbf{y}} \left\langle \mathbf{w}, \Phi(\mathbf{x}, \mathbf{y}) \right\rangle &= \max_{y_1, \ldots, y_d} s(y_1, \ldots, y_d) \\ s(y_t, \ldots, y_{t+d-1}) &= \max_{y_{t+d}} s(y_{t+1}, \ldots, y_{t+d}) \\ &\quad + \left\langle \mathbf{w}, \Phi(\mathbf{x}_t^{t+d-1}, \mathbf{y}_t^{t+d-1}) - \Phi(\mathbf{x}_{t+1}^{t+d-1}, \mathbf{y}_{t+1}^{t+d-1}) \right\rangle \\ s(y_{T-d+1}, \ldots, y_T) &= \left\langle \mathbf{w}, \Phi(\mathbf{x}_{T-d+1}^{T}, \mathbf{y}_{T-d+1}^{T}) \right\rangle\end{aligned}$$

However, the computation time in each step of the recursion grows exponentially with respect to the maximum feature length d. Even the dual form (5.3) still has the same problem, because we have to evaluate all of the features explicitly when predicting labels. This is a serious problem when we want to employ kernels with arbitrarily long features (possibly of infinite length). Also, CRFs have the same problem since they employ dynamic programming procedures based on the same decomposition in learning and prediction. In addition, even the decomposition (5.4) is not applicable to kernels enumerating the occurrences of features when allowing gaps [55], since a feature of length of d can occur over an interval of length longer than d. [11] avoids this problem by a reranking approach where a pre-trained labeler using local features generates an affordable number of candidates for $\mathbf{y}$, and the candidates are reranked by using global features. However, the correct answer is not guaranteed to be contained in the candidates based on the local features.

5.3 Marginalized labeling perceptrons

5.3.1 Marginalized labeling perceptron

We now propose a new kernel-based labeler that solves the problems of the previous section. Usually, in MEMMs and CRFs, model $P_s(\mathbf{y}|\mathbf{x})$ is trained by maximizing the sum of the log-likelihood $\sum_i \log P_s(\mathbf{y}^{(i)}|\mathbf{x}^{(i)})$. This objective function aims to label an entire sequence correctly. On the other hand, Kakade et al. [37] noticed that it suffices to maximize the number of individually correct labels in many labeling tasks. They proposed to use the marginalized $P_s(\mathbf{y}|\mathbf{x})$ over all possible assignments of labels for the hidden variables with the t-th hidden variable fixed as $\hat{y}_t \in \Sigma_y$.

$$P_p(y_t = y_t^{(i)}|\mathbf{x}^{(i)}) = \sum_{\mathbf{y}:y_t=y_t^{(i)}} P_s(\mathbf{y}^{(i)}|\mathbf{x}^{(i)})$$

The point-wise objective function $\sum_i \sum_t \log P_p(y_t = y_t^{(i)}|\mathbf{x}^{(i)})$ is shown to be comparable to the original sequential log-loss function [5]. The important point is that label prediction is performed in a point-wise manner since P_p depends only on y_t. We combine this idea with perceptron-based labelers. We propose a *marginalized labeling perceptron* defined as follows.

$$\hat{y}_t = \operatorname*{argmax}_{\tilde{y}_t \in \Sigma_y} \sum_{\mathbf{y}:y_t=\tilde{y}_t} P(\mathbf{y}|\mathbf{x}) \big\langle \mathbf{w}, \Phi(\mathbf{x}, \mathbf{y}) \big\rangle. \tag{5.5}$$

Note that the point-wise label prediction works since the argmax operation depends only on $\tilde{y}_t$. The main idea is that, when y_t is predicted, the original output $\langle \mathbf{w}, \Phi(\mathbf{x}, \mathbf{y}) \rangle$ is marginalized over all possible assignments of labels for the hidden variables with the t-th hidden variable fixed as $\hat{y}_t$. $P(\mathbf{y}|\mathbf{x})$ is some prior distribution over hidden variables that might be pre-trained probabilistic models such as MEMMs or CRFs, or might be designed manually. Since $\sum_{\mathbf{y}:y_t=\tilde{y}_t} P(\mathbf{y}|\mathbf{x})\Phi(\mathbf{x}, \mathbf{y})$ can be considered as a new feature vector, the weight vector $\mathbf{w}$ is updated by

$$\mathbf{w}^{new} = \mathbf{w}^{old} + \sum_{\mathbf{y}:y_t=y_t^{(i)}} P(\mathbf{y}|\mathbf{x})\Phi(\mathbf{x}, \mathbf{y}) - \sum_{\mathbf{y}:y_t=\hat{y}_t^{(i)}} P(\mathbf{y}|\mathbf{x})\Phi(\mathbf{x}, \mathbf{y}), \tag{5.6}$$

when the prediction for the t-th position of $\mathbf{e}^{(i)}$ is wrong, i.e. $\hat{y}_t^{(i)} \neq y_t^{(i)}$.

5.3.2 Kernel marginalized labeling perceptron

Next, we derive a dual form of the marginalized labeling perceptron, and obtain a fully-kernelized labeler. Let $\phi_f(\mathbf{x}, \mathbf{y}; t)$ be the number of times a feature f appears in $(\mathbf{x}, \mathbf{y})$ with including the t-th position of $(\mathbf{x}, \mathbf{y})$, and $\Phi_f(\mathbf{x}, \mathbf{y}; t)$ be the vector form. First, we rewrite the primal form (5.5) as follows by adding terms that do not depend on $\tilde{y}_t$.

$$\begin{aligned}
\hat{y}_t &= \operatorname*{argmax}_{\tilde{y}_t \in \Sigma_y} \sum_{\mathbf{y}:y_t=\tilde{y}_t} P(\mathbf{y}|\mathbf{x})\big\langle \mathbf{w}, \Phi(\mathbf{x}, \mathbf{y}) \big\rangle + \sum_{\tilde{y}_t \in \Sigma_y} \sum_{\mathbf{y}:y_t=\tilde{y}_t} P(\mathbf{y}|\mathbf{x})\big\langle \mathbf{w}, \Phi(\mathbf{x}, \mathbf{y}; t) \big\rangle \\
&\qquad - \sum_{\mathbf{y} \in {\Sigma_y}^T} P(\mathbf{y}|\mathbf{x})\big\langle \mathbf{w}, \Phi(\mathbf{x}, \mathbf{y}) \big\rangle \\
&= \operatorname*{argmax}_{\tilde{y}_t \in \Sigma_y} \sum_{\mathbf{y}:y_t=\tilde{y}_t} P(\mathbf{y}|\mathbf{x})\big\langle \mathbf{w}, \Phi(\mathbf{x}, \mathbf{y}; t) \big\rangle.
\end{aligned} \tag{5.7}$$

The update rule is then rewritten as

$$\mathbf{w}^{new} = \mathbf{w}^{old} + \sum_{\mathbf{y}:y_t=y_t^{(i)}} P(\mathbf{y}|\mathbf{x})\Phi(\mathbf{x}, \mathbf{y}; t) - \sum_{\mathbf{y}:y_t=\hat{y}_t^{(i)}} P(\mathbf{y}|\mathbf{x})\Phi(\mathbf{x}, \mathbf{y}; t).$$

By introducing example weights α, $\mathbf{w}$ is rewritten as a linear combination,

$$\mathbf{w} = \sum_{j=1}^{|E|} \sum_{\tau=1}^{T^{(j)}} \sum_{\breve{y}_\tau \in \Sigma_y} \alpha_{j\tau}(\breve{y}_\tau) \sum_{\mathbf{y}:y_\tau=\breve{y}_\tau} P(\mathbf{y}|\mathbf{x})\Phi(\mathbf{x}^{(j)}, \mathbf{y}; \tau). \tag{5.8}$$

```
// Initialization
α := 0
// Training
for round = 1, ..., max_round,
  for i = 1, ..., |E|,
    // kernel computation
    for j = 1, ..., |E|,
      for t = 1, ..., T^(i), and τ = 1, ..., T^(j),
        for y̆_τ ∈ Σ_y, and ỹ_τ ∈ Σ_y,
          compute K(x^(j), x^(i), τ, t, y̆_τ, ỹ_t)
        end_for
      end_for
    end_for
    // prediction & update
    for t = 1, ..., T^(i),
      predict ŷ_t^(i) by using Equation (5.9)
      if ŷ_t^(i) ≠ y_t^(i),
        α_it(y_t^(i)) := α_it(y_t^(i)) + 1
        α_it(ŷ_t^(i)) := α_it(ŷ_t^(i)) − 1
      end_if
    end_for
  end_for
end_for
```

Figure. 5.3 Kernel Marginalized Labeling Perceptron

Finally, substituting Equation (5.8) into Equation (5.7), we obtain the *kernel marginalized labeling perceptron*,

$$\hat{y}_t = \operatorname*{argmax}_{\tilde{y}_t \in \Sigma_y} \sum_{j=1}^{|E|} \sum_{\tau=1}^{T^{(j)}} \sum_{\breve{y}_\tau \in \Sigma_y} \alpha_{j\tau}(\breve{y}_\tau) K(\mathbf{x}^{(j)}, \mathbf{x}, \tau, t, \breve{y}_\tau, \tilde{y}_t), \tag{5.9}$$

where

$$K(\mathbf{x}^{(j)}, \mathbf{x}, \tau, t, \breve{y}_\tau, \tilde{y}_t) = \sum_{\mathbf{y}: y_\tau = \breve{y}_\tau} \sum_{\mathbf{y}': y'_t = \tilde{y}_t} P(\mathbf{y}|\mathbf{x}^{(j)}) P(\mathbf{y}'|\mathbf{x}) \big\langle \Phi(\mathbf{x}^{(j)}, \mathbf{y}; \tau), \Phi(\mathbf{x}, \mathbf{y}'; t) \big\rangle \tag{5.10}$$

is called the marginalized kernel since the kernel is marginalized over all possible assignments of labels for hidden variables with fixed labels at τ and t. The algorithm is described in Figure 5.3.

5.4 Kernel computation

In this section, we propose several instances of the marginalized kernel (5.10) for labeling sequences, ordered trees, and directed acyclic graphs. Recently, kernels for structured data such as sequences, trees, and graphs have been studied actively [22]. We extend these kernels for labeling problems. In all kernels, for the sake of simplicity, we suppose that the prior $P(\mathbf{y}|\mathbf{x})$ can be decomposed as

$$P(\mathbf{y}|\mathbf{x}) = \prod_{t=1}^{T} P(y_t|x_t).$$

However, the following discussion can be generalized to exploit more general models like CRFs. In the remainder of this section, we use $K(\tau, t, \breve{y}_\tau, \tilde{y}_t)$ to refer the kernel function given i and j. The important point in computing our kernels is that all features can be constructed by combining smaller features. For instance, in Figure 5.4, suppose that the feature (b) appears with its rightmost position at t, and the feature (c) appears with its leftmost position at t. Then, the feature (a), the combination of the two features, appears with its second variable at t. All the marginalized kernels that we propose in this chapter are decomposed into the upstream kernels $K_U(\tau, t)$, the downstream kernels $K_D(\tau, t)$, and the point-wise kernels $K_P(\tau, t, \breve{y}_\tau, \tilde{y}_t)$ as follows.

$$\begin{aligned}
K(\tau, t, \breve{y}_\tau, \tilde{y}_t) &= K_U(\tau, t) \cdot K_P(\tau, t, \breve{y}_\tau, \tilde{y}_t) \cdot K_D(\tau, t) \\
K_U(\tau, t) &= \sum_{\mathbf{y}_U(\tau)} \sum_{\mathbf{y}'_U(t)} P(\mathbf{y}_U(\tau)|\mathbf{x}_U^{(j)}(\tau)) P(\mathbf{y}'_U(t)|\mathbf{x}_U^{(i)}(t)) \\
&\qquad \cdot \big\langle \Phi(\mathbf{x}_U^{(j)}(\tau), \mathbf{y}_U(\tau); \tau), \Phi(\mathbf{x}_U^{(i)}(t), \mathbf{y}'_U(t); t) \big\rangle \qquad (5.11) \\
K_D(\tau, t) &= \sum_{\mathbf{y}_D(\tau)} \sum_{\mathbf{y}'_D(t)} P(\mathbf{y}_D(\tau)|\mathbf{x}_D^{(j)}(\tau)) P(\mathbf{y}'_D(t)|\mathbf{x}_D^{(i)}(t)) \\
&\qquad \cdot \big\langle \Phi(\mathbf{x}_D^{(j)}(\tau), \mathbf{y}_D(\tau); \tau), \Phi(\mathbf{x}_D^{(i)}(t), \mathbf{y}'_D(t); t) \big\rangle \qquad (5.12)
\end{aligned}$$

$$K_P(\tau, t, \breve{y}_\tau, \tilde{y}_t) = \frac{P(\breve{y}_\tau|x_\tau^{(j)}) P(\tilde{y}_t|x_t^{(i)}) \big\langle \Phi(x_\tau^{(j)}, \breve{y}_\tau; \tau), \Phi(x_t^{(i)}, \tilde{y}_t; t) \big\rangle}{\big(\sum_{\breve{y}_\tau} \sum_{\tilde{y}_t} P(\breve{y}_\tau|x_\tau^{(j)}) P(\tilde{y}_t|x_t^{(i)}) \big\langle \Phi(x_\tau^{(j)}, \breve{y}_\tau; \tau), \Phi(x_t^{(i)}, \tilde{y}_t; t) \big\rangle \big)^2} \qquad (5.13)$$

where $\mathbf{y}_U(\tau)$ is the set of the hidden variables lying upstream of τ, $\mathbf{y}_D(\tau)$ is the set of the hidden variables lying downstream of τ, and $\mathbf{y}_U(\tau) \cap \mathbf{y}_D(\tau) = \{y_\tau\}$. The

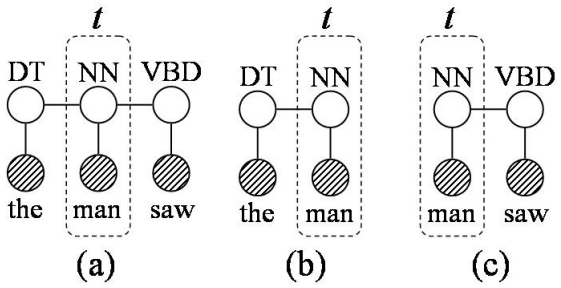

Figure. 5.4 Combining upstream features and downstream features to make larger features. Appearance of the feature (a) with its second variable at t is guaranteed by the appearances of the feature (b) with its rightmost position at t and the feature (c) with its leftmost position at t.

denominator of $K_P(\tau, t, \breve{y}_\tau, \tilde{y}_t)$ cancels the overlap of $K_U(\tau, t)$ and $K_D(\tau, t)$. Although explicit computation of the marginalization in K_U and K_D is prohibitive, we can efficiently compute them by using dynamic programming in many cases. The major advantage of the decomposition is that, given the i-th example and the j-th example, marginalized kernels for all possible values of $(\tau, t, \breve{y}_\tau, \tilde{y}_t)$ can be computed at the same time.

5.4.1 Sequence labeling

Now we introduce kernels for sequence labeling. To consider the long-distance dependencies in sequences, we define each feature as a consecutive pair of a hidden variable and an observable variable of arbitrary length, as in Figure 5.2. Since the lengths of the features are not restricted, and features of various lengths are mixed, we vary the weight ϕ_f according to the length of the feature f by using the parameter $c > 0$. If the length of the feature is d, and if we observe the feature N times in $(\mathbf{x}, \mathbf{y})$, $\phi_f(\mathbf{x}, \mathbf{y})$ is defined as $N \cdot c^d$.

$K_U(\tau, t)$ for labeling sequences is the kernel that considers only the appearances of features with their rightmost positions at τ and t, and which is marginalized over all possible assignments for $\mathbf{y}_U(\tau)$ and $\mathbf{y}_U(t)$. Similarly, $K_D(\tau, t)$ is the kernel that considers only the appearances of features with their leftmost positions at τ and t, and which is marginalized over all possible assignments for $\mathbf{y}_D(\tau)$ and $\mathbf{y}_D(t)$.

The algorithm consists of two dynamic programming loops, one for K_U, and the other

for K_D, just like the forward-backward algorithm.

$$K_U(\tau,t) = c^2 k(x_\tau^{(j)}, x_t^{(i)})(K_U(\tau-1,t-1)+1) \tag{5.14}$$

$$K_D(\tau,t) = c^2 k(x_\tau^{(j)}, x_t^{(i)})(K_D(\tau+1,t+1)+1), \tag{5.15}$$

where

$$k(x_\tau^{(j)}, x_t^{(i)}) = \sum_{\breve{y}_\tau \in \Sigma_y} \sum_{\tilde{y}_t \in \Sigma_y} P(\breve{y}_\tau | x_\tau^{(j)}) P(\tilde{y}_t | x_t^{(i)}) k_y(\breve{y}_\tau, \tilde{y}_t) k_x(x_\tau^{(j)}, x_t^{(i)})$$

Note that $K_U(\tau,0) = K_U(0,t) = K_D(\tau, |T^{(i)}|+1) = K_D(|T^{(j)}|+1, t) = 0$. K_P is computed as follows.

$$K_P(\tau,t,\breve{y}_\tau,\tilde{y}_t) = \frac{c^2 P(\breve{y}_\tau | x_\tau^{(j)}) P(\tilde{y}_t | x_t^{(i)}) k_y(\breve{y}_\tau, \tilde{y}_t) k_x(x_\tau^{(j)}, x_t^{(i)})}{\left(c^2 k(x_\tau^{(j)}, x_t^{(i)})\right)^2}, \tag{5.16}$$

where k_x and k_y are functions that return 1 when the arguments are identical, and return 0 elsewhere. However, we can replace them with kernels between the two variables. Apparently, the time complexity of computing the kernels is $O(T^{(i)}T^{(j)})$.[*2]

5.4.2 Tree labeling

The kernel for labeling ordered trees (Figure 5.5) is based on the labeled ordered tree kernel [43]. We allow arbitrary tree-structured features such as in Figure 5.5. By using the mixing parameter c as in the sequence labeling, $\phi_f(\mathbf{x},\mathbf{y})$ is defined to be $N \cdot c^d$ where N is the number of times the feature f appears as subgraphs in $(\mathbf{x},\mathbf{y})$, and d is the size of f. $K_U(\tau,t)$ for labeling trees is the kernel that considers only the appearances of features with their leaf positions at τ and t, and that is marginalized over all possible assignments for $\mathbf{y}_U(\tau)$ and $\mathbf{y}_U(t)$. Similarly, $K_D(\tau,t)$ is the kernel that considers only appearances of features with their root positions at τ and t, and that is marginalized over all possible assignments for $\mathbf{y}_D(\tau)$ and $\mathbf{y}_D(t)$.
The algorithm consists of two dynamic programming loops, one for K_U, and the other

[*2] Gappy matching as in [55] is allowed by introducing a penalty parameter $0 \le \lambda \le 1$. An appearance of a feature is counted as λ^g if g gaps are inserted. The recursive equation (5.14) is modified by using the accumulated upstream kernel S_F as follows.

$$\begin{aligned} K_U(\tau,t) &= c^2 k(x_\tau^{(j)}, x_t^{(i)})(S_U(\tau-1,t-1)+1) \\ S_U(\tau,t) &= K_U(\tau,t) + \lambda S_U(\tau,t-1) \\ &\quad + \lambda S_U(\tau-1,t) + \lambda^2 S_U(\tau-1,t-1) \end{aligned}$$

The downstream kernel (5.15) is modified accordingly.

for K_D, just as in the inside-outside algorithm. Let $ch(\tau, \upsilon)$ be the index of the υ-th child node of the τ-th node, $pa(\tau)$ be the index of the parent node of the τ-th node, $\#ch(\tau)$ be the number of the child nodes of the τ-th node, and $chID(\tau)$ be the index that means the τ-th node is the $chID(\tau)$-th child of its parent node.
Similar to the recursive computation of the labeled ordered tree kernel [43], K_D is computed by dynamic programming in a post-order traversal by using

$$K_D(\tau, t) = c^2 k(x_\tau^{(j)}, x_t^{(i)}) S_F(\tau, t, \#ch(\tau), \#ch(t)),$$

where S_F is also defined recursively as follows,

$$\begin{aligned} S_F(\tau, t, \upsilon, u) = {} & K_D(ch(\tau, \upsilon), ch(t, u)) S_F(\tau, t, \upsilon - 1, u - 1) \\ & + S_F(\tau, t, \upsilon - 1, u) + S_F(\tau, t, \upsilon, u - 1) - S_F(\tau, t, \upsilon - 1, u - 1), \end{aligned}$$

where $S_F(\tau, t, 0, u) = S_F(\tau, t, \upsilon, 0) = 1$. Intuitively, $S_F(\tau, t, \upsilon, u)$ is the sum of the contributions of all the way of matching between the nodes indexed from $ch(\tau, 1)$ to $ch(\tau, \upsilon)$ and the nodes indexed from $ch(t, 1)$ to $ch(t, u)$.
At the same time, for the sake of the computation of K_U, we also define $S_B(\tau, t, \upsilon, u)$, the sum of the contributions of all the way of matching between the nodes indexed from $ch(\tau, \upsilon)$ to $ch(\tau, \#ch(\tau))$ and the nodes indexed from $ch(t, u)$ to $ch(t, \#ch(t))$, as

$$\begin{aligned} S_B(\tau, t, \upsilon, u) = {} & K_D(ch(\tau, \upsilon), ch(t, u)) S_B(\tau, t, \upsilon + 1, u + 1) \\ & + S_B(\tau, t, \upsilon + 1, u) + S_B(\tau, t, \upsilon, u + 1) - S_B(\tau, t, \upsilon + 1, u + 1), \end{aligned}$$

where $S_B(\tau, t, \#ch(\tau) + 1, u) = S_B(\tau, t, \upsilon, \#ch(t) + 1) = 1$.
K_U is computed by dynamic programming in a pre-order traversal. $K_U(\tau, t)$ is computed by combining the upstream kernel of the parents and the downstream kernels of the siblings (Figure 5.6),

$$\begin{aligned} K_U(\tau, t) = c^2 k(x_\tau^{(j)}, x_t^{(i)}) \Big(& 1 + K_U(pa(\tau), pa(t)) \\ & \cdot S_F(pa(\tau), pa(t), chID(\tau) - 1, chID(t) - 1) \\ & \cdot S_B(pa(\tau), pa(t), chID(\tau) + 1, chID(t) + 1) \Big), \end{aligned}$$

Note that the point-wise kernel K_P is the same as in Equation (5.16). A similar analysis to that of [43] can show that the time complexity of computing the kernel is $O(T^{(i)} T^{(j)})$.

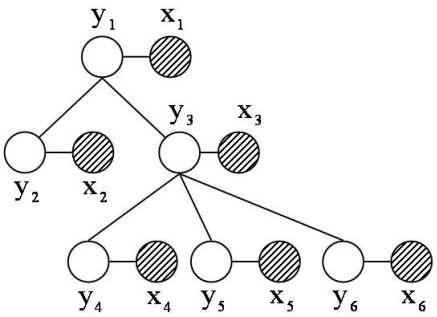

Figure. 5.5 Ordered trees: shadowed nodes indicate observable variables, and white nodes indicate hidden variables.

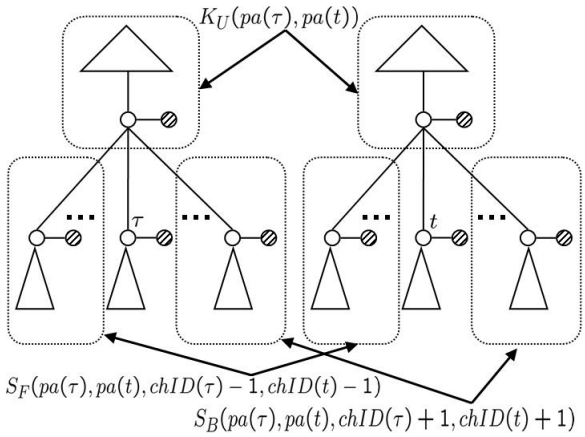

Figure. 5.6 Intuitive image for computing $K_U(\tau, t)$.

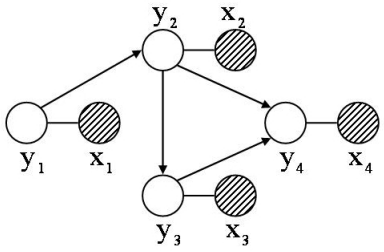

Figure. 5.7 Directed acyclic graphs: shadowed nodes indicate observable variables, and white nodes indicate hidden variables.

5.4.3 Graph labeling

Finally, we propose a kernel for labeling directed acyclic graphs (DAGs) (Figure 5.7). Unfortunately, it has been shown to be NP-hard to use arbitrary graph-

structured features [23]. Therefore, we propose a marginalized kernel using directed path features like Figure 5.2, based on the DAG kernel [69].
$K_U(\tau, t)$ for labeling DAGs is the kernel that considers only the appearances of features with their end positions at τ and t, and which is marginalized over all possible assignments for $\mathbf{y}_U(\tau)$ and $\mathbf{y}_U(t)$. Similarly, $K_D(\tau, t)$ is the kernel that considers only the appearances of features with their start positions at τ and t, and which is marginalized over all possible assignments for $\mathbf{y}_D(\tau)$ and $\mathbf{y}_D(t)$. As in the kernels for sequence labeling, we can write K_U and K_D recursively,

$$K_U(\tau, t) = c^2 k(x_\tau^{(j)}, x_t^{(i)}) \left(1 + \sum_{v \in Pa(\tau)} \sum_{u \in Pa(t)} K_U(v, u) \right)$$

$$K_D(\tau, t) = c^2 k(x_\tau^{(j)}, x_t^{(i)}) \left(1 + \sum_{v \in Ch(\tau)} \sum_{u \in Ch(t)} K_D(v, u) \right),$$

where $Pa(\tau)$ and $Ch(\tau)$ are the set of indices of the parent nodes and the child nodes of the τ-th node, respectively. Note that the point-wise kernel K_P is the same as Equation (5.16). The time complexity of computing this kernel is $O\big(|T^{(i)}||T^{(j)}|(\max_{\tau,t} |Pa(\tau)||Pa(t)| + \max_{\tau,t} |Ch(\tau)||Ch(t)|)\big)$.

5.5 Experiments

Finally, we will show the result for experiments on NLP data to assess the importance of large structural features in a real-life situation. We conducted two experiments on Named Entity Recognition (NER) and Product Usage Information Extraction tasks. We compared the performance of the kernel marginalized labeling perceptron with the HM-Perceptrons, which is limited to smaller features in nature.

5.5.1 Named entity recognition

NER is a kind of information extraction task that deals with identifying proper names such as person names and organization names in sentences. We used a subcorpus consisting of the first 300 sentences (8,541 tokens) from the Spanish corpus provided for the Special Session of CoNLL2002 on NER. The task is to label each word with one of the nine labels, i.e. $|\Sigma_y| = 9$, that represents the types and boundaries of named entities, including a dummy label for non-named entities.
The kernel between two observable labels is designed by using words and their spelling

Table. 5.1 Named Entity Recognition task result

	Accuracy	Precision	Recall	F1
Sequence Kernel	**88.4**% (3.9)	**52.3**% (17.5)	**19.3**% (2.2)	**27.9** (5.1)
HM-Perceptron	82.9% (7.4)	21.8% (9)	15.6%(4.5)	17.2 (4)

Table. 5.2 Product Usage Information Extraction task result

	Accuracy	Precision	Recall	F1
Sequence Kernel	88.4% (1.4)	45.7% (10)	**35.2**% (17.7)	36.7 (5.8)
Tree Kernel	**89.8**% (1.2)	**51.5**% (5.2)	32.3% (17.4)	**37.9** (12.1)
HM-Perceptron	87.7%(2.0)	40%(16.5)	29%(11.4)	30.7(9.3)

features, which are described in [5] as the S2 features. In both algorithms, it is combined with a second degree polynomial kernel. A window of size 3 is also used in HM-Perceptrons according to [6]. For the proposed perceptron algorithm, we used the sequence kernel with the mixing parameter $c = 1$ which, was determined in preliminary experiments. The prior over hidden variables is modeled as a uniformed distribution, i.e. $P(y_t|x_t) = 1/|\Sigma_y|$. The performances are evaluated according to the labeling accuracies (including correct answers for dummy labels), precision and recall of named entity labels, and the F1 measure which is the harmonic mean of the precision and recall.

Table 5.1 shows the results of the NER task in a 3-fold cross-validation. The values in parentheses represent the standard deviations on each cross-validation. The proposed perceptron algorithm with the sequence kernel outperforms the other method. Note that the F1 measure score is a more suitable measure for evaluating NER tasks than the labeling accuracy because the majority of the annotated labels are dummy labels. The good performance of the proposed sequence kernel is due to the nature of NER tasks. NER tasks are known as one of the NLP problems which require considering a wide context. For instance, the same phrase "White house" can be a location or an organization entity in different contexts. This empirical result supports the advantages of the fully-kernelized labeler that can handle rich contextual information efficiently without any manual selection of an appropriate feature size.

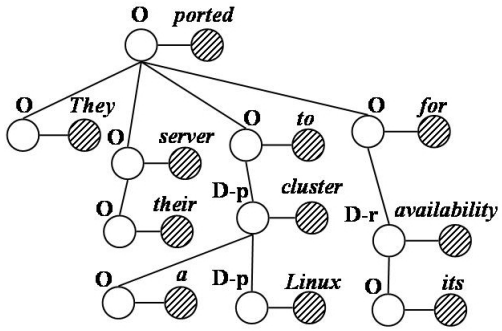

Figure. 5.8 An example of lexicalized dependency trees for the sentence "They ported their server to a linux cluster for its availability" where the hidden variable label "D-p" is for the name of products which a customer uses, "D-r" is for the reason of use, and "O" is dummy label.

5.5.2 Product usage information extraction

Product Usage Information Extraction is a special case of NER tasks which deals with extracting information about the usage of products from a sales log written by sales representatives. We used 184 sentences (3,570 tokens) from an annotated sales log written in Japanese. This task aims to identify the product name (p), its vendor (v), the number of products bought (n), and the reason for buying (r) linked to a classification of the customer state that indicates whether the costomer is already using (D), wishes to use (P), or rejects (R) using the products. In total, there are $|\Sigma_y| = 12$ different labels including a dummy label (O).

All of the Japanese sentences were previously segmented into words, and these words were annotated with the part-of-speech and base form. In all of the algorithms, we designed the kernel between two observable labels by using words, part-of-speech tags, base forms, and character type features. We used it by combining with the second degree polynomial kernel.

In this task, we use not only the sequence kernel for the proposed perceptron, but also the tree kernel. For the tree kernel, we used (lexicalized) dependency trees that represent linguistic structures of the sentences in terms of the dependencies between words. Figure 5.8 shows an example of a dependency tree in English. To build the dependency trees for the data set, we used a Japanese statistical parser [38].*3 For

*3 The accuracy of the parser is 88%.

both kernels, the mixing parameter is $c = 0.8$.
Table 5.2 shows the result of the Product Usage Information Extraction task in 3-fold cross-validation. The parameter settings and the evaluation measures are the same as in the NER task. Again, we can see the advantage of the proposed perceptron algorithm against the HM-Perceptrons. Furthermore, the tree kernel utilizing dependency trees shows higher performance than the sequence kernel, which elaborates the advantage of using structured information.

5.6 Concluding remarks

In this chapter, we introduced an efficient fully-kernelized learning algorithm for labeling structured data such as sequences, trees, and graphs. Our approach can handle large features including arbitrary numbers of variables by using the pointwise label prediction [37], and the marginalized kernels [76]. We also proposed several instances of the marginalized kernels for labeling sequences, ordered trees and directed acyclic graphs. In the preliminary experiments on information extraction tasks using real-world data, our approach was shown to be promising.

Part II

Link prediction methods for network-structured data

Chapter 6

A supervised link prediction method based on network structure evolution models

For modeling and analyzing various phenomena concerning a collection of entities, it is rarely sufficient to examine the properties of the entities themselves. The essence of their collective behavior is often embedded within the relationship among them, such as co-occurrences, causality, and other types of interactions. Such relations can, for the most part, be represented as *networks* consisting of a set of entities and the links between them.

Examples of network data around us that call for analysis are abundant. The world wide web consists of pages and hyperlinks among them. Social Networks consist of individuals and relations among them, such as friendship. They are attracting considerable attention from various business perspectives, such as marketing and business process modeling. In the field of bioinformatics, network structures among biological entities such as genes and proteins, representing physical interactions and gene regulation, are studied extensively. (See Figure 6.1 for an example biological network.) Links among entities are not limited to static relations, but may vary dynamically. For example, e-mail exchanges and cooperative interactions are temporal relations. In the field of sociometry, there has been a long history of social network analysis, in which relations among social entities are analyzed [84].

With the large amount of network data becoming readily available in electronic form today, along with the advances in computing power and algorithmic techniques that enable handling of such massive data, there has been a surge of interest in the

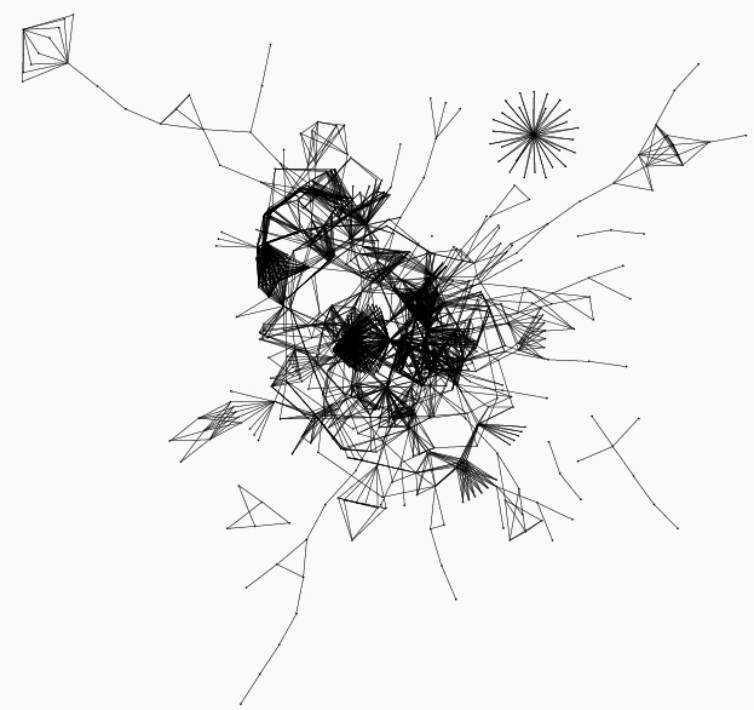

Figure. 6.1 A metabolic network of S. Cerevisiae. Proteins are represented as nodes, and an edge represents catalyzation of successive reactions by two proteins.

study of analytical methods for network-structured data, and *link mining* [25] has become a popular sub-area of data mining. Link mining includes several tasks such as link-based object classification/ranking/clustering, link prediction, and subgraph discovery [25]. In the present chapter, we consider the link prediction problem, which is the task of predicting unobserved portions of the network, such as hidden links, from the observed part of the network (or alternatively, to predict the future structure of the network given the current structure of the network).

In this chapter, we introduce a new approach to the problem of link prediction for network-structured domains based on the topological features of network structure, not on the node features. We present a probabilistic evolution model of network structure which models probabilistic flips of the existence of edges depending on a "copy-and-paste" mechanism of edges. Based on this model, we propose a transductive learning algorithm for link prediction based on an assumption of the stationarity of the network.

6.1 Supervised link prediction problem

Link prediction has several applications including predicting relations among participants such as friendship and pecking order, and predicting their future behavior such as communications and collaborations. In the field of bioinformatics, predicting protein-protein interactions and regulatory relationships can provide guidance on the design of experiments for discovering new biological facts.

We start with defining the supervised link prediction problem we consider in this chapter. The link prediction problem is usually formalized as a binary classification problem or a ranking problem on the node pairs.

Let the data domain be represented as a graph $G = (V, \Phi)$, where $V = \{1, 2, \ldots, |V|\}$ is the set of indices of nodes, and $\Phi : V \times V \to \{0, 1\}$ is a set of *edge labels*. Each node $v \in V$ represents an entity, for example, a person in the case of social networks, and a protein in the case of protein-to-protein interaction networks. An *edge label* $\phi(i, j)$ indicates whether or not an edge exists between any pair of nodes. In particular, $\phi(i, j) = 1$ if an edge exists between i and j, and $\phi(i, j) = 0$ if an edge does not exist between i and j. Note that ϕ is symmetric, i.e. $\phi(i, j) = \phi(j, i)$.

Let $E^L \subset V \times V$ be the *labeled pairs*, which is a set of pairs of nodes whose edge labels are known. They will serve as the training data set for our problem. The *Link prediction problem* is the task of predicting the values of edge labels for $E^U := (V \times V) - E^L$, given V and E^L as training data.

There are two types of information that may be useful for this objective – information on the nodes themselves and that on the network topology. For example, there may be information about each person in a social network, such as the name, age, hobbies, etc. There have been several studies of link prediction with node features, but comparatively little work has been done with topological features [65, 26, 75, 64]. Topological features can be derived from generative models of network structures [62]. Perhaps the most well-known model is the preferential attachment model proposed by Barabási et al. [3]. Liben-Nowelly and Kleinberg proposed and compared a number of metrics derived from various network models [54].

Certainly, the ultimate goal would be to utilize all available information that may be helpful for prediction. In this chapter, however, we do not assume the existence of any features on the nodes that can be used for inference. We focus on the use of

topological features having to do with the structure of the network, and consider the link prediction problem based solely on them.

6.2 A link prediction method based on a network structure evolution model

In this section, we propose a novel parameterized model for network evolution, which is then used to derive a method of link prediction.

In our model, probabilistic flips of the existence of edges are modeled by a certain "copy-and-paste" mechanism between the edges. Our link prediction algorithm is derived by assuming that the network structure is generated by the *stationary distribution* of the network evolution. This allows us to formalize the inference of the marginal stationary distribution as a transduction problem, which is an inference problem from partially observed edge labels. Then we propose an on-line transduction method for solving this problem. The algorithm embodies an approximate maximum likelihood estimation procedure using (approximate) exponentiated gradient ascent [29].

6.2.1 An edge label copying model of network structure evolution

In this section, we propose a parameterized probabilistic model of network evolution, in which the structure of a network probabilistically changes over time. Based on this model, we predict whether an edge exists between two nodes or not, at any given point in time. By the "structure of network" we mean the edge label function ϕ that indicates the existence of edges among nodes.

We denote by $\Phi^{(t)}$ the set of edge labels at time t. We assume that only $\Phi^{(t)}$ changes over time, and V, the members in the network, are fixed at all times, as we are primarily interested in the link structure among them.

In our model, we model probabilistic flips of the value of ϕ. Flips of the edge labels do not occur uniformly at random, but occur depending on some characteristics of the network. We characterize this by the process of "copy-and-paste" of edge labels between the nodes. The probability of copying differs from one node pair to another. We assume a Markov model in which $\Phi^{(t+1)}$ depends only on $\Phi^{(t)}$.

In the proposed model, an edge label is copied from node ℓ to node m randomly with probability $w_{\ell m}$. Once it has been decided that an edge label is to be copied from node ℓ to m, node ℓ copies one of its $|V| - 1$ edge labels (excluding $\phi^{(t)}(\ell, m)$)

to node m uniformly at random. We have the following probability constraints.

$$\sum_{\ell m} w_{\ell m} = 1,\ w_{\ell m} \geq 0. \tag{6.1}$$

Also, we assume that $w_{\ell\ell} = 0$, which indicates copying from a node to the node itself does not occur. Let W denote the matrix whose (ℓ, m)-th element is $w_{\ell,m}$.

The basic idea behind our model is as follows; if you have a friend who has a strong influence on you, your association will be highly affected by the friend's association. Also, if a gene is duplicated in the course of genetic evolution, the copied gene will have similar characteristics to the original one. Assume that node k has a strong influence on node i, and there is an edge between node k and node j. Following the above hypothesis, there will likely be an edge between node i and node j. Similarly, if there are no edges between k and j, there will likely be no edge between i and j.

There are two possible ways for $\phi^{(t)}(i,j)$ to assume a particular edge label. One possibility is that node k has copied an edge label to node i or to node j. The other is that $\phi^{(t)}(i,j) = \phi^{(t+1)}(i,j)$ and nothing has happened (indicating that a copy happened somewhere else in the network).

Following the above discussion, the probability of an edge existing between node i and node j at time $t+1$, can be written as

$$\begin{aligned}\Pr[\phi^{(t+1)}(i,j) = 1 | \Phi^{(t)}] = \frac{1}{|V|-1}\Big(\sum_{k\neq i,j} w_{kj}\phi^{(t)}(k,i) + w_{ki}\phi^{(t)}(k,j)\Big) \\ + \Big(1 - \frac{1}{|V|-1}\sum_{k\neq i,j} w_{kj} + w_{ki}\Big)\phi^{(t)}(i,j),\end{aligned} \tag{6.2}$$

where the first term indicates the probability that the edge label for (i,j) is changed by copy-and-pasting, and the second term indicates the probability that the edge label for (i,j) is left unchanged since copy-and-pasting happened somewhere else. In the first term, node k must have an edge to node j at time t (i.e. $\phi^{(t)}(k,j) = 1$) for node k to make node i span an edge to node j. Similarly, node k must have an edge to node i (i.e. $\phi^{(t)}(k,i) = 1$) for node k to make node j span an edge to node i. We assume that the edge label for (i,j) cannot be copied from node i to node j, and vice versa.

Note that (6.2) is evaluating the marginal probability, in other words, it is evaluating the probability of each edge label independently, and therefore the product of (6.2) is not equal to the joint probability of all edge labels.

6.2.2 Stationary distribution of the network structure

Since we do not know the true parameters in (6.2), and do not in general observe the history of evolution of the network structure, it is unlikely that we can predict the existence of edges just based on this model. Therefore, we make an additional assumption that the current network that we are seeing is in some sense "typical" of the network structure averaged over time [32]. More precisely, we assume a stationary distribution of the network structure.

Assume that there exists a stationary state of network structure evolution,

$$\Phi^{(\infty)} := \lim_{t\to\infty} \Phi^{(t)},$$

and its stationary distribution, $\Pr[\Phi^{(\infty)}]$.

Although it is difficult to obtain the stationary distribution, it is rather easy to derive the marginal stationary distribution. By taking the expectation of (6.2) with respect to $P(\Phi^{(t)})$, the probability distribution for $\Phi^{(t)}$,

$$\begin{aligned}\Pr[\phi^{(t+1)}(i,j)=1] &= \sum_{\Phi^{(t)}} \Pr[\phi^{(t+1)}(i,j)=1|\Phi^{(t)}]P(\Phi^{(t)}) \\ &= \frac{1}{|V|-1}\Big(\sum_{k\neq i,j} w_{kj}\Pr[\phi^{(t)}(k,i)=1] + w_{ki}\Pr[\phi^{(t)}(k,j)=1]\Big) \\ &\quad +\Big(1-\frac{1}{|V|-1}\sum_{k\neq i,j} w_{kj}+w_{ki}\Big)\Pr[\phi^{(t)}(i,j)=1]. \qquad (6.3)\end{aligned}$$

Letting $t\to\infty$, we solve (6.3) to obtain the the marginal stationary distribution,

$$\rho(i,j|W) = \frac{\sum_{k\neq i,j} w_{kj}\rho(k,i|W) + w_{ki}\rho(k,j|W)}{\sum_{k\neq i,j} w_{kj}+w_{ki}}, \qquad (6.4)$$

where we let

$$\rho(i,j|W) := \Pr[\phi^{(\infty)}(i,j)=1].$$

6.2.3 An EM-like estimation algorithm

Now we formalize our goal as an optimization problem and propose an Expectation-Maximization(EM)-type algorithm to solve it. We let the objective function be the log-likelihood of the stationary distribution of the network structure with respect to

the known part of the network E^L. Letting $\Phi^{(\infty)^L}$ be the set of edge labels for the known part and $\Phi^{(\infty)^U}$ be that for the unknown part, the objective function is written as $\log P(\Phi^{(\infty)^L}|W)$.

Let us derive an algorithm for maximizing this. In the EM algorithm, given the current parameters $\widehat{W}$, we try to update them to W so that the objective function is improved. By taking the lower bound,

$$\begin{aligned}\log P(\Phi^{(\infty)^L}|W) &= \log \sum_{\Phi^{(\infty)^U}} P(\Phi^{(\infty)^L}, \Phi^{(\infty)^U}|W) \\ &= \log \sum_{\Phi^{(\infty)^U}} P(\Phi^{(\infty)^L}, \Phi^{(\infty)^U}|W) \frac{P(\Phi^{(\infty)^U}|\Phi^{(\infty)^L}, \widehat{W})}{P(\Phi^{(\infty)^U}|\Phi^{(\infty)^L}, \widehat{W})} \\ &\geq \sum_{\Phi^{(\infty)^U}} P(\Phi^{(\infty)^U}|\Phi^{(\infty)^L}, \widehat{W}) \log \frac{P(\Phi^{(\infty)^L}, \Phi^{(\infty)^U}|W)}{P(\Phi^{(\infty)^U}|\Phi^{(\infty)^L}, \widehat{W})},\end{aligned}$$

it amounts to maximizing the term including W, i.e.

$$Q(W, \widehat{W}) := \sum_{\Phi^{(\infty)^U}} P(\Phi^{(\infty)^U}|\Phi^{(\infty)^L}, \widehat{W}) \log P(\Phi^{(\infty)^L}, \Phi^{(\infty)^U}|W).$$

In the exact EM algorithm, one would like to maximize $Q(W, \widehat{W})$ with respect to W for a fixed $\widehat{W}$. Performing the exact EM algorithm is, however, infeasible since we do not have the explicit representation of the distribution of $\Phi^{(\infty)}$. Therefore, we approximate this by the product of the marginal distributions that we already have as follows,

$$\begin{aligned}Q(W, \widehat{W}) &\approx Q^{\text{approx}}(W, \widehat{W}) \\ &:= \sum_{\Phi^{(\infty)^U}} \prod_{(i,j)\in E^U} P(\phi(i,j)^{(\infty)}|\Phi^{(\infty)^L}, \widehat{W}) \log \prod_{(k,l)\in E} P(\phi^{(\infty)}(k,l)|W)\end{aligned}$$

The approximation adopted here amounts to ignoring statistical correlations between different links, so that one could call the approximation the naive mean-field approximation.

$Q^{\text{approx}}(W, \widehat{W})$ is further simplified as follows,

$$\begin{aligned}Q^{\text{approx}}(W, \widehat{W}) &= \sum_{(k,l)\in E} \sum_{\Phi^{(\infty)^U}} \prod_{(i,j)\in E^U} P(\phi(i,j)^{(\infty)}|\Phi^{(\infty)^L}, \widehat{W}) \log P(\phi^{(\infty)}(k,l)|W) \\ &= \sum_{(i,j)\in E^U} \sum_{\phi^{(\infty)}(i,j)\in\{0,1\}} P(\phi(i,j)^{(\infty)}|\Phi^{(\infty)^L}, \widehat{W}) \log P(\phi^{(\infty)}(i,j)|W)\end{aligned}$$

$$
\begin{aligned}
&+ \sum_{(i,j)\in E^L} \log P(\phi^{(\infty)}(i,j)|W) \\
&= \sum_{(i,j)\in E^U} \Big(\rho(i,j|\Phi^{(\infty)^L}, \widehat{W}) \log \rho(i,j|W) + (1-\rho(i,j|\Phi^{(\infty)^L}, \widehat{W})) \log(1-\rho(i,j|W)) \Big) \\
&+ \sum_{(i,j)\in E^L} \phi(i,j) \log \rho(i,j|W) + (1-\phi(i,j)) \log(1-\rho(i,j|W)), \qquad (6.5)
\end{aligned}
$$

where $\phi(i,j)$ is the known link label, and $\rho(i,j|\Phi^{(\infty)^L}, \widehat{W})$ is

$$\rho(i,j|\Phi^{(\infty)^L}, \widehat{W}) := \Pr[\phi^{(\infty)}(i,j) = 1|\Phi^{(\infty)^L}].$$

Note that (6.4) also holds for $\rho(i,j|\Phi^{(\infty)^L}, \widehat{W})$.

Finding W that maximizes (6.5) does not guarantee to find the local optimum which is guaranteed in the EM-algorithm, since $Q^{\text{approx}}(W, \widehat{W})$ is the mean field approximation and does not evaluate $Q(W, \widehat{W})$ exactly.

Now we describe the detail of the Expectation-step (E-step) and the Maximization-step (M-step) of our EM-like optimization algorithm.

E-step

The step to obtain the probability for unknown edge labels $\rho(i,j|\Phi^{(\infty)^L}, \widehat{W})$ for $(i,j) \in E^U$ is rather easy. As we have seen in the previous subsection, they must satisfy the following stationary equation,

$$\rho(i,j|\Phi^{(\infty)^L}, \widehat{W}) = \frac{\sum_{k\neq i,j} w_{kj}\rho(k,i|\Phi^{(\infty)^L}, \widehat{W}) + w_{ki}\rho(k,j|\Phi^{(\infty)^L}, \widehat{W})}{\sum_{k\neq i,j} w_{kj} + w_{ki}}, \qquad (6.6)$$

Therefore, we solve the simultaneous linear equations to get the unknown edge labels. Note that $\rho(i,j|\Phi^{(\infty)^L}, \widehat{W}) = \phi(i,j)$ for $(i,j) \in E^L$.

M-step

Next, we consider the more complicated step to maximize L with respect to W while all $\{\rho(i,j)|(i,j) \in E^U\}$ fixed.

This optimization problem is nonlinear, and does not have a closed form solution. Therefore, we employ a gradient-based optimization to obtain a solution numerically. We employ the exponentiated gradient algorithm [29] since it ensures positivity of all elements of W and the probability constraint (6.1), and converges fast in the case of sparse W, which is the case for most of the real world networks.

Suppose that we have parameters W at a particular time of iterations, we want to update W to W' to increase $Q^{\text{approx}}(W', \widehat{W})$. Also, the new W' is chosen to be sufficiently close to W by making the distance penalty $-d(W', W)$. We choose the distance as the relative entropy,

$$d(W', W) = \sum_{\ell,m} w'_{\ell m} \log \frac{w'_{\ell m}}{w_{\ell m}},$$

which leads to the exponentiated gradient algorithm.

At each iteration step, we determine the new W' that maximizes the following objective function with respect to (6.1).

$$\eta Q^{\text{approx}}(W', \widehat{W}) - d(W', W),\ \eta > 0,$$

where $\eta > 0$ is a constant that balances the two terms. Using Lagrangean multiplier γ, the objective function we wish to maximize is

$$F(W') := \eta Q^{\text{approx}}(W', \widehat{W}) - d(W', W) - \gamma\Big(\sum_{\ell,m} w_{\ell m} - 1\Big)$$

Approximating $Q^{\text{approx}}(W', \widehat{W})$ by using $L(W)$ as

$$Q^{\text{approx}}(W', \widehat{W}) = Q^{\text{approx}}(W, \widehat{W}) + \sum_{\ell,m} \frac{\partial Q^{\text{approx}}(W, \widehat{W})}{\partial w_{\ell m}}(w'_{\ell m} - w_{\ell m}),$$

and setting the gradient of $F(W')$ with respect to $w'_{\ell m}$ to be zero, the Lagrangean that we maximize is

$$\frac{\partial F(W')}{\partial w'_{\ell m}} = \eta \frac{\partial Q^{\text{approx}}(W, \widehat{W})}{\partial w_{\ell m}} - \Big(\log \frac{w'_{\ell m}}{w_{\ell m}} + 1\Big) + \gamma = 0.$$

Solving this equation with $\sum_{\ell,m} w_{\ell m} = 1$ in mind, we obtain the following exponentiated gradient update,

$$w'_{\ell m} = Z^{-1} w_{\ell m} \exp\Big(\eta \frac{\partial Q^{\text{approx}}(W, \widehat{W})}{\partial w_{\ell m}}\Big), \tag{6.7}$$

where

$$Z := \sum_{\ell,m} w_{\ell m} \exp\Big(\eta \frac{\partial Q^{\text{approx}}(W, \widehat{W})}{\partial w_{\ell m}}\Big).$$

Now we compute the gradient of the objective function with respect to $w_{\ell m}$ becomes

$$\frac{\partial Q^{\text{approx}}(W, \widehat{W})}{\partial w_{\ell m}} = \sum_{i,j} \frac{\partial Q_{ij}^{\text{approx}}(W, \widehat{W})}{\partial w_{\ell m}},$$

where $Q_{ij}^{\mathrm{approx}}(W, \widehat{W})$ denotes the term in (6.5) corresponding to the note pair (i, j).

We obtain

$$\frac{\partial Q_{ij}^{\mathrm{approx}}(W, \widehat{W})}{\partial w_{\ell m}} = \left\{ \rho(i,j|\Phi^{(\infty)^L}, \widehat{W}) \frac{1}{\rho(i,j|W)} - [1 - \rho(i,j|\Phi^{(\infty)^L}, \widehat{W})] \frac{1}{1-\rho(i,j|W)} \right\} \frac{d\rho(i,j|W)}{dw_{\ell m}},$$

for $(i, j) \in E^U$, and

$$\frac{\partial Q_{ij}^{\mathrm{approx}}(W, \widehat{W})}{\partial w_{\ell m}} = \left\{ \phi(i,j) \frac{1}{\rho(i,j|W)} - [1 - \phi(i,j)] \frac{1}{1-\rho(i,j|W)} \right\} \frac{d\rho(i,j|W)}{dw_{\ell m}},$$

for $(i, j) \in E^L$, where

$$\frac{d\rho(i,j|W)}{dw_{\ell m}} = \left(\frac{\delta(m=j)\rho(\ell,i|W) + \delta(m=i)\rho(\ell,j|W)}{\sum_{k \neq i,j} w_{kj} + w_{ki}} - \frac{\delta(m=j) + \delta(m=i)}{\left(\sum_{k \neq i,j} w_{kj} + w_{ki} \right)^2} \right) + \frac{\sum_{k \neq i,j} w_{kj} \frac{d\rho(k,i|W)}{dw_{\ell m}} + w_{ki} \frac{d\rho(k,j|W)}{dw_{\ell m}}}{\sum_{k \neq i,j} w_{kj} + w_{ki}}. \tag{6.8}$$

Note that $\rho(i, j|W)$ depends on the current W, therefore we have to solve (6.4) at every time we take the derivatives. Also in (6.8), the derivatives are depending on each other, so strictly, we have to solve the large system of the simultaneous linear equations to obtain these derivatives, possibly by the power method. However, it is not efficient to solve them at every M-step, so we just approximate them by dropping those interacting terms, which is equivalent to the initial value for the power method for solving the simultaneous linear equations.

$$\frac{d\rho(i,j|W)}{dw_{\ell m}} \approx \left(\frac{\delta(m=j)\rho(\ell,i|W) + \delta(m=i)\rho(\ell,j|W)}{\sum_{k \neq i,j} w_{kj} + w_{ki}} - \frac{\delta(m=j) + \delta(m=i)}{\left(\sum_{k \neq i,j} w_{kj} + w_{ki} \right)^2} \right)$$

The above discussion is summarized as the procedure described in Figure 6.2. [Step:4] corresponds to the step to obtain the marginal stationary probability of unknown edge labels and [Step:3] corresponds to the step to maximize $Q^{\mathrm{approx}}(W, \widehat{W})$ with respect to W. Note that, this procedure is similar to that of the Expectation-Maximization (EM)-based transduction method [63], but different from the EM algorithm in a strict sense. When we see our algorithm as an EM algorithm, the

Algorithm: Batch
[**Step:1**] Set $w_{\ell m} := \frac{1}{|V|-1}$ for all (ℓ, m) such that $\ell \neq m$.
[**Step:2**] Continue [Step:3] and [Step:4], alternately.
[**Step:3**] Solve (6.6) to obtain $\rho(i,j|\Phi^{(\infty)^L}, \widehat{W})$ for $(i,j) \in E^U$.
[**Step:4**] Find W that maximizes $Q^{\text{approx}}(W, \widehat{W})$, by using (6.7).

Figure. 6.2 A batch transduction algorithm

lower bound of the objective function is also approximated by the mean field approximation. And also, in the M-step, we approximate the derivatives of the objective function. Therefore, the convergence to local optimum solutions is not guaranteed.

6.2.4 Scaling up the estimation algorithm by sequential updating

The transduction algorithm presented in the previous subsection runs in a batch manner, so can be somewhat inefficient for large data sizes. Therefore, we approximate this by a sequential version of the learning algorithm.

We perform the iteration of [Step:3] and [Step:4] in Figure 6.2, not with the entire data, but with only one datum at a time. Assume that (i,j) is the node pair that we currently focus on.

As for [Step:3] in Figure 6.2, instead of solving (6.6) as simultaneous linear equations, we process only one step of the power method for one unobserved edge label by

$$\rho(i,j|\Phi^{(\infty)^L}, \widehat{W}) := \frac{\sum_{k \neq i,j} w_{kj}\rho(k,i|\Phi^{(\infty)^L}, \widehat{W}) + w_{ki}\rho(k,j|\Phi^{(\infty)^L}, \widehat{W})}{\sum_{k \neq i,j} w_{kj} + w_{ki}}. \quad (6.9)$$

Similarly, $\rho(i,j|\widehat{W})$ is updated by

$$\rho(i,j|\widehat{W}) := \frac{\sum_{k \neq i,j} w_{kj}\rho(k,i|\widehat{W}) + w_{ki}\rho(k,j|\widehat{W})}{\sum_{k \neq i,j} w_{kj} + w_{ki}}. \quad (6.10)$$

As for [Step:4] in Figure 6.2, instead of (6.7), we employ stochastic approximation which processes one datum at a time,

$$w'_{\ell m} = Z^{-1} w_{\ell m} \exp\left(\gamma \frac{\partial Q_{ij}^{\text{approx}}(W, \widehat{W})}{\partial w_{\ell m}}\right) \quad (6.11)$$

Algorithm: Sequential
[**Step:1**] Set $w_{\ell m} := \frac{1}{|V|-1}$ for all (ℓ, m) such that $\ell \neq m$.
[**Step:2**] Solve (6.4) to obtain $\rho(i,j)$ for $(i,j) \in E^U$.
[**Step:3**] Continue [Step:4].
[**Step:4**] Sample (i,j) uniformly at random, and
[**Step:4-a**] update $\rho(i,j|\Phi^{(\infty)^L}, \widehat{W})$ by using (6.9) if $(i,j) \in E^U$,
[**Step:4-a**] update $\rho(i,j|\widehat{W})$ by using (6.10), and
[**Step:4-b**] update $w_{\ell m}$ for $\{(\ell,m)|\ell \in \{1,2,\ldots,|V|\}, m \in \{i,j\}\}$ by using (6.11).

Figure. 6.3 A sequential transduction algorithm

for $\{(\ell,m)|\ell \in \{1,2,\ldots,|V|\}, m \in \{i,j\}\}$, where γ is the coefficient for stochastic approximation. Note that Z includes the summation of $w_{\ell m}$ over all (ℓ,m), but we only compute the differences of Z, so each update is done in $O(|V|)$ time.

Also we execute the two steps in parallel. We randomly pick an (i,j) pair, and execute (6.11), and execute (6.9) if $(i,j) \in E^U$,

The description of the sequential version of the transduction algorithm is exhibited in Figure 6.3.

6.3 Experiments

In this section, we evaluate the effectiveness of the proposed approach by empirically comparing its predictive performance with link prediction methods based on the various topological metrics. The results of our experiments indicate that the predictive performance of the proposed method, in terms of precision recall curves, significantly out-performs these existing methods, for the two biological network datasets.

6.3.1 Review of comparison methods

Before describing the experimental setting, we review the existing methods from earlier works [54, 88] we compare against the proposed method. These methods are based on node similarity metrics derived from certain network evolution models. Since each of these metrics described below quantifies the potential of an edge existing between a pair of nodes, they give rise to a *ranking* over node pairs and thus can be directly used to perform link prediction. (Link predictions can be given as ranking over the set of all possible edges.)

Note that all metrics defined below consider only the positive edge labels. In the

definitions to follow, we let $\Gamma(i)$ denote the set of neighbor nodes connected to node i with edge label 1.

- Common neighbors [61]

$$\text{common} := |\Gamma(i)| \cap |\Gamma(j)|$$

  Common neighbors metric is based on the idea that if two nodes i and j have many common neighbor nodes, they are likely to be linked.

- Jaccard's coefficient [7, 54]

$$\text{Jaccard's} := \frac{|\Gamma(i)| \cap |\Gamma(j)|}{|\Gamma(i)| \cup |\Gamma(j)|}$$

  Jaccard's coefficient is a normalized version of the common neighbors metric, and is used as a similarity metric in the field of information retrieval. In Jaccard's coefficient, if two pairs of nodes have the same number of shared neighbor nodes, since in some sense their links are considered more "precious".

- Adamic/Adar [2]

$$\text{Adamic/Adar} := \sum_{k \in |\Gamma(i)| \cap |\Gamma(j)|} \frac{1}{\log |\Gamma(k)|}$$

  Adamic/Adar metric is a variant of the common neighbors metric. The idea is similar to Jaccard's coefficient in the sense that a link owned by nodes with a smaller number of neighbors is considered more important, but Adamic/Adar assigns different weights among the neighbors. Common neighbors with a small number of neighbors are weighted more highly.

- Katz_β [45]

$$\text{Katz}_\beta := \sum_{l=1}^{\infty} \beta^l |\text{paths}_{i,j}^{(l)}|$$

  Katz_β is a generalization of the common neighbors metric to account for more distant relations. Here $\text{paths}_{i,j}^{(l)}$ is the number of paths of length l from node i to node j. It is essentially identical to the diffusion kernel used in kernel methods to define the similarity between two nodes on a graph. In our experiment, we set $\beta = 0.05$.

- Preferential attachment [9, 61]

$$\text{preferential} := |\Gamma(i)| \cdot |\Gamma(j)|$$

Preferential attachment is based on a different idea from those of the above variants of common neighbors metric. It originates with a generative model of scale free networks, where nodes with many neighbors are likely to obtain new neighbors.

6.3.2 Experimental setting

We used two biological network datasets in our experiments. One is a medium sized metabolic network dataset, and the other is a larger protein-protein interaction network dataset.

- Metabolic network
  The first dataset [87] contains metabolic pathways of the yeast S. Cerevisiae in KEGG/PATHWAY database [41]. Figure 6.1 shows an overview of this network. In this network, proteins are represented as nodes, and an edge indicates that the two proteins are enzymes that catalyze successive reactions between them.
  The number of nodes in the network is 618, and the number of links is 2782. Therefore the number of data (i.e. node pairs) to be classified is 190653, and the ratio of positive and negative data is 0.015 : 1.
- Protein-protein interaction network
  The second dataset is a protein-protein interaction network dataset constructed by von Mering et al. [83]. We followed Tsuda and Noble [77], and used the medium confidence network, containing 2617 nodes and 11855 links. Therefore the number of data (i.e. node pairs) to be classified is 3423036, and the ratio of positive and negative data is 0.003 : 1.

In both datasets, the number of edges are very small as compared to the number of node pairs, so the ratio of positive and negative data is highly skewed. Therefore, in [Step:4] in the learning algorithm of Figure 6.3, we use weighted sampling using the data distribution, that is, the positive data are sampled with probability proportional to the ratio of the number of negative data to the total number of data (and similarly for the negative data).

We used 66% of the data as training data and the rest as test data, and evaluated the performance by 3-fold cross validation. The performance of competing methods

is compared using precision-recall curves since the dataset are highly skewed, with less than 2% of the edge labels being positive.

6.3.3 Experimental Results

First, we will investigate the relative performance of the various methods we consider. Figure 6.4 exhibits the precision-recall curves for the metabolic network data, and Figure 6.4 shows those for the protein-protein interaction network data. Table 6.1 summarizes these results, in terms of the break-even points of precision and recall by the respective methods.

Overall, the proposed method achieves better performance than all other methods, particularly with the metabolic network data. With the protein-protein network data, the improvement is not as dramatic, especially in terms of the break-even points.

In link prediction, however, it is generally considered important to achieve high precision in the low recall area (i.e. the left-half of the precision-recall curve). This is because, in actual applications, link prediction is often used to recommend a small number of promising pairs from among a large set of candidates, e.g. recommending new friends in social network services or potential protein-protein interactions that have not been found experimentally. From this perspective, we conclude, the proposed method enjoys a significantly higher performance.

Next, we see whether or not the transductive formulation results in significant improvement in predictive performance. We can obtain a non-transduction version of our link prediction method by simply setting $\phi(i, j)$ for $(i, j) \in E^U$ to be identically 0, and not using the constraints on the unknown edge labels. Figure 6.6 shows the precision recall curves of the proposed method, with and without transduction, for the metabolic network data. We observe that adoption of transductive formulation does boost the predictive performances for these datasets.

Finally, we examine the relationship among the competing methods by evaluating the similarity of predictions output by them. In particular, we compare them by using Spearman rank correlation [30] on the positive test data. Spearman rank correlation is a measure of how two rankings of a given set are similar to each other. The similarity is the highest when it is 1, and the lowest when it is 0. Table 6.2 shows the Spearman rank correlations for all pairs of the methods we consider, for the metabolic network data. Also, Figure 6.7 visualizes them in two dimensions by multi-

dimensional scaling [17]. Note that they are essentially comparisons for only the positive test data, since the datasets are highly skewed. Therefore the prediction performance can differ significantly even when the two rankings are highly correlated. (It is the ranking on the positive test data that matters in attaining high predictive performance, of which there are few.) Now, let us examine the results. We can observe that the predictions made by common neighbor, Jaccard's coefficient, and Adamic/Adar are very similar to one another, which is actually implied by their definitions. Since Katz_β can be regarded as an extension of these metrics, it is no surprise that it also has high correlations with them, but at the same time, it is also similar to preferential attachment.

It is interesting to observe that the mean correlation for the proposed method is the lowest among all methods, implying that the proposed method is rather different from all the other methods, in the way it predicts. It is consistent with the fact that our model is based on an evolution model with different characteristics from the models that the other metrics/methods are based upon. This means that adding the proposed method to the pool of existing methods, for example, when using their predictions as part of input features to a subsequent classifier, would be beneficial.

6.3.4 Discussion

In our experiments, our model attained good performance on biological network data. But, when considering applications of our method to various kinds of data, the following two questions about generality and limitations of our method arise. The first question is whether we can apply the method to data other than biological networks or not. The results of the experiment are thought to be, in part, attributable to the fact that our network evolution model matches the characteristics of biological networks. In [62], it is mentioned that there have been proposed several models based on copying mechanism as models for biological networks. Actually, in our preliminary experiments, our model did not work well for a kind of social networks that represents co-authoring relationships of academic publications, which implies that the model is not appropriate for modeling collaborative networks. Barabási et al. [9] propose the preferential attachment model as one of such models.

The second question is whether or not our evaluation method is appropriate in real settings. In our experiments, we performed cross-validation for evaluation, i.e. we

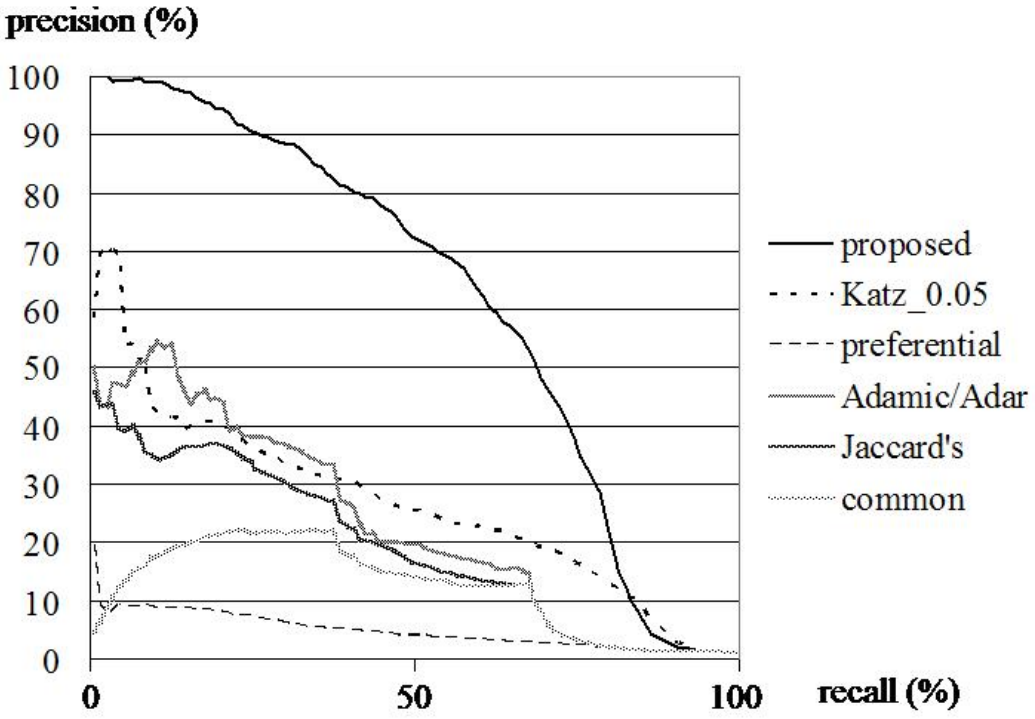

Figure. 6.4 Precision-Recall curve for metabolic network with 66% training data.

	metabolic	protein-protein
common	21.1%	37.9%
Jaccard's	30.2%	47.9%
Adamic/Adar	34.9%	50.3%
preferential	9.2%	25.4%
$\text{Katz}_{0.05}$	32.8%	27.6%
proposed	61.2%	52.6%

Table. 6.1 Summary of results for both dataset measured by break-even point of precision and recall.

randomly chose the link labels used for training and testing, which indicates that our knowledge about data is uniform. However, in protein-to-protein interaction networks, it is more realistic to consider such cases where relatively many relationships are known for a certain number of proteins, and almost no relationships are unknown for the proteins. Our method does not work for such cases, since the other proteins with no known relationships have no structural clues on which our method depends.

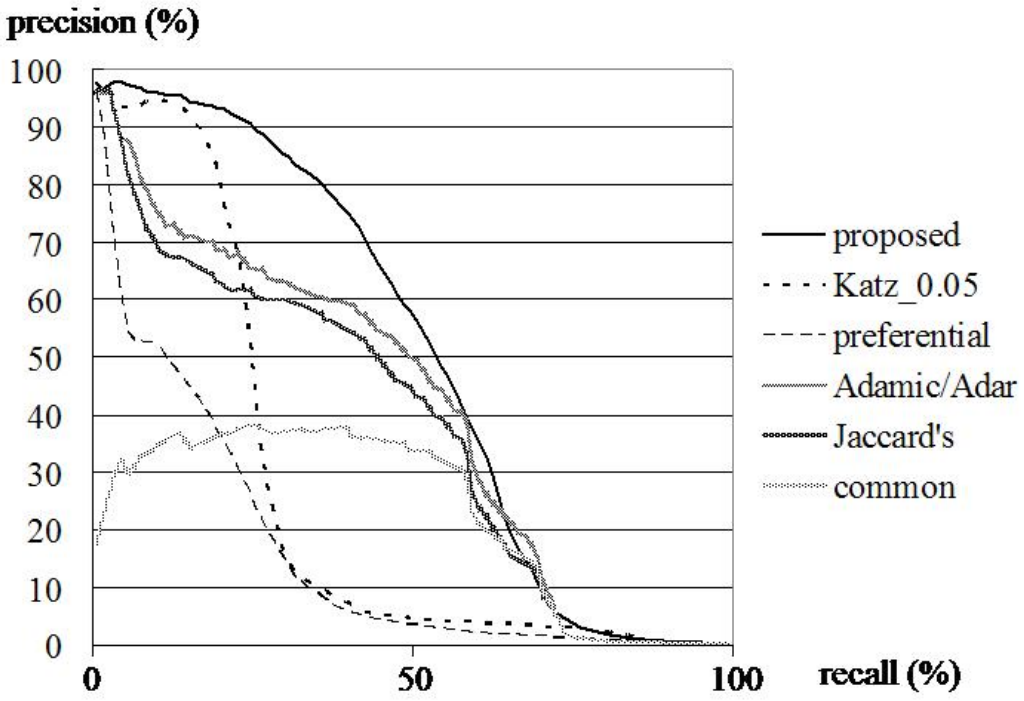

Figure. 6.5 Precision-Recall curve for protein-to-protein interaction network with 66% training data.

6.4 Related work

As we pointed out in Introduction, link prediction is naturally cast as a classification/ranking problem for node pairs, and two types of information, node features and topological features, are used for this task.

Topological features are often derived from generative models of network structure [2, 7, 9, 45, 54, 61], and our model can be interpreted as a parameterized version of the "node copying model" due to Kleinberg et al. [46, 50], To the best of our knowledge, there been no probabilistic models with tunable parameters derived from a network evolution model, nor have there been any topological metrics defined based on a node copying model. We also point out that our generative model differs from theirs, in that in our model the edge labels (either positive or negative) are copied, not the edges (positive labels only). Also, we note that our model does not consider node addition and deletion, since our interest is in link prediction. Another difference is

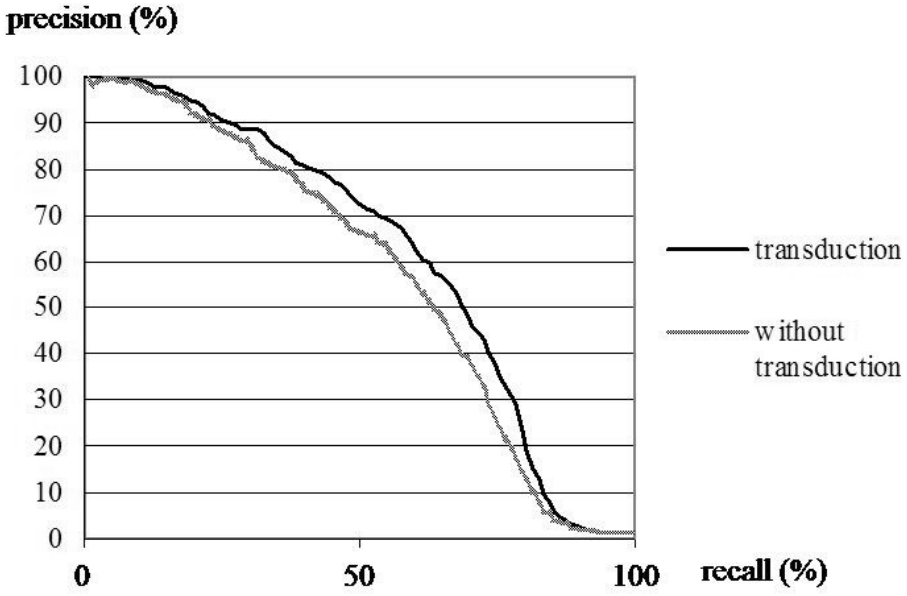

Figure. 6.6 Comparison of Precision-Recall curves of transductive inference and non-transductive inference for metabolic network. The break-even points are 61.2% with transduction, and 58.0% without transduction, respectively.

	c	J	A	p	K	proposed
common	1	0.92	0.94	0.31	0.61	0.20
Jaccard's	0.92	1	0.97	0.53	0.75	0.35
Adamic/Adar	0.94	0.97	1	0.49	0.70	0.31
preferential	0.31	0.53	0.49	1	0.84	0.69
$\text{Katz}_{0.05}$	0.61	0.75	0.70	0.84	1	0.70
proposed	0.20	0.35	0.31	0.69	0.70	1
mean	0.80	0.91	0.88	0.77	0.92	0.65

Table. 6.2 Spearman rank correlations among rankings by various methods for positive test data in the metabolic network dataset. Note that the Spearman rank correlation is symmetric. Columns indicated by c, J, A, p, and K denote common neighbors, Jaccard's coefficient, Adamic/Adar, preferential attachment, and $\text{Katz}_{0.05}$, respectively.

that Kleinberg et al's model allows copying multiple edges at a time, our model copies one edge label at a time, where copy probabilities are parameterized and tunable in the model.

Another approach to link prediction that exists in the literature is that of applying supervised learning methods using node features as well as topological features. (See,

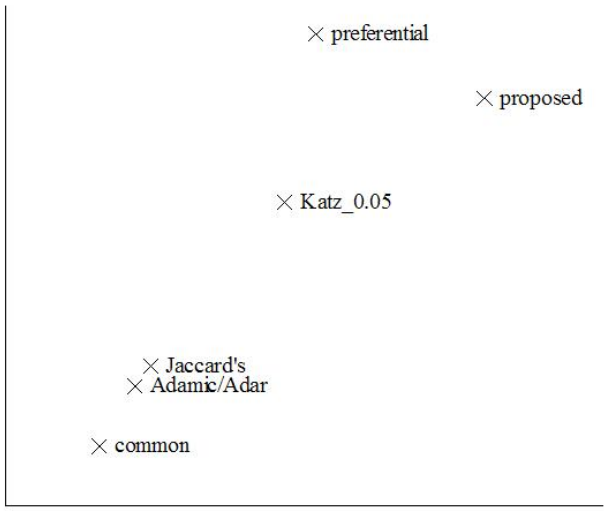

Figure. 6.7 Visualization of the Spearman correlation matrix of Table 6.2 in two dimensions by multi-dimensional scaling.

for example, Hasan et al. [26] and O'Madadhain et al. [64].) We point out that their work differs from ours in that their learning algorithms do not learn parameters within the network model. In order to examine the merit of applying a supervised learning method on topological features, we conducted a preliminary experiment in which a linear predictor was applied on the topological features considered in our experiments to perform link prediction. We did not observe any accuracy improvement over the direct methods that base their predictions solely on the topological metrics.

There has also been some works that apply the framework of statistical relational learning to link prediction. For example, Popescul and Ungar [65] and Taskar et al. [75] have applied such approach, using both node features and topological features, including those we used in our experiments. It is worth noting that these are applications of a more general paradigm (of relational learning) to the link prediction problem, and are to be contrasted with approaches such as ours that develop tailored models and methods for link prediction per se.

Link prediction has strong resemblance to the problem of collaborative filtering, if we regard it as a matrix completion problem. For example, Huang et al. [88] applied the metrics we used in our experiments to collaborative filtering. On the other hand, application of collaborative filtering techniques to link prediction would also make

sense. Indeed, an approach by Nakamura and Abe [60], which completes an user-item matrix by learning similarities among rows and columns, has some superficial resemblance to our approach and motivated our present work to some extent. Applications of other types of collaborative filtering techniques would be worth exploring.[*1]

6.5 Concluding remarks

In this chapter, we introduced a new approach to the problem of link prediction for network-structured domains based on the topological features of network structure, not on the node features. We presented a probabilistic evolution model of network structure which models probabilistic flips of the existence of edges depending on a "copy-and-paste" mechanism for the edges. Based on this model, we proposed a transductive learning algorithm for link prediction based on an assumption of the stationarity of the network. Finally, we show some promising experimental results using real network data. We used biological network data in our experiments, and attained good performance.

[*1] Although we did not show the results in this chapter, the matrix factorization approach [72] performed poorly on link prediction tasks in our preliminary experiments.

Part III

Conclusions and future directions

Chapter 7

Conclusions and future directions

In this thesis, we proposed several approaches of machine learning to structured data. The first part of this thesis discussed approaches for internally structured data, and the second part discussed approaches for externally structured data.

In the first part, we focused especially on applying kernel methods, and discussed the design of kernel functions for various kind of internally structured data based on the framework of the convolution kernel.

In Chapter 3, we discussed convolution kernels for tree-structured data. Our contributions in this chapter are twofold. First, we introduced a new kernel function for labeled ordered trees by extending Collins and Duffy's parse tree kernel, and showed an efficient algorithm for computing the kernel. Also, we proposed some extensions to allow flexibility in the substructures used in the kernel without increasing the computational complexity of the extended kernels. The new kernels we proposed allow working on more general trees such as semi-structured data that parse tree kernels cannot handle. The second contribution is the hardness result for designing kernels for unordered labeled trees. By showing that computing the unordered tree kernels that use subtrees as their substructures is #P-complete, we showed a limit of generalization in designing kernels of the same type as the labeled ordered tree kernel.

One of the possible directions of future research is the design of more efficient tree kernels. Tree kernels based on dynamic programming need at least quadratic computation time. However, if more scalability is needed, quadratic computation time is still too slow, and near-linear time algorithms are desirable. Limiting the class of substructures to simpler ones such as substrings [53], and balancing the expressive power and computational efficiency will be important for this goal.

In Chapter 4 we discussed the convolution kernels for graph-structured data. Our

contributions in this chapter are twofold. The first contribution is to introduce the kernel for graph-structured data by using the idea of random walks on graphs. The second contribution is a technique which reduces the kernel computation to solving a system of simultaneous linear equations, which allows computing the graph kernels in realistic times.

The structure we dealt with is fairly general, and promising in a wide variety of problems in bioinformatics and other areas. Potential targets would be DNA and RNA sequences with remote correlations, HTML and XML documents in MEDLINE, topology graphs and distance graphs of 3-D protein structures, and so on.

One of the possible directions of future research is to tune the parameters of the random walk, which was set just to uniform distributions in our example. This corresponds to an implicit feature selection problem, and both wrapper approaches and filter approaches will be applicable to this purpose.

In Chapter 5, we applied kernel methods to the labeling problem, which is a generalized problem of the supervised classification problems. Our contribution in this chapter is to introduce an efficient fully kernelized learning algorithm for labeling structured data such as sequences, trees, and graphs. Contrasted with the existing label prediction methods that can handle only features of a small number of hidden variables, our approach can handle large features, including arbitrary numbers of variables, by using pointwise label prediction and marginalized kernels. We also proposed several types of marginalized kernels for labeling sequences, ordered trees and directed acyclic graphs.

One of the possible directions of future research is to incorporate more complicated prior distributions such as MEMMs and CRFs in the marginalized kernels, although we only used a very simple form of prior distribution in this chapter. It is an interesting question whether our approach will improve such probabilistic models.

Another direction would be a large-margin version of our approach. Since HM-SVMs [6] use an entire label sequence as an example, maximization of the margin between correct label sequences and the second best label sequences is not always acceptable in problems where each sequence has some difficult labels. On the other hand, our approach avoids this problem by considering the label at each position as an individual example.

In the second part, we discussed prediction of the network structure of externally structured data.

In Chapter 6, we discussed the link prediction problem based only on structural information. Our contributions in this chapter are twofold. First, we presented a probabilistic evolution model of network structures that models probabilistic flips of the existence of edges depending on a "copy-and-paste" mechanism for the edges. The second contribution is the transductive learning algorithm for link prediction based on an assumption of the stationary distribution of the network evolution model. Our work differs from these earlier efforts in that the existence of tunable parameters within the network model naturally gives rise to a *learning* algorithm for link prediction, which leads to improved accuracy of prediction. To the best of our knowledge, our model is the first structural-information-based model that has learnable parameters, and therefore provides a new idea to approaches for link prediction.

One of the possible directions of future research is generalization to scenarios in which there are more than two edge labels. Let Σ be a set of label types, and $\phi_a^{(t)}(i,j)$ be the edge label for each edge type $a \in \Sigma$. Then we could modify the model (6.2) as

$$\Pr[\phi_a^{(t+1)}(i,j) = 1 | \Phi^{(t)}] = \frac{1}{|V|-1}\Big(\sum_{k \neq i,j} w_{kj}\phi_a^{(t)}(k,i) + w_{ki}\phi_a^{(t)}(k,j)\Big) + \Big(1 - \frac{1}{|V|-1}\sum_{k \neq i,j} w_{kj} + w_{ki}\Big)\phi_a^{(t)}(i,j).$$

For example, this generalized model would include directed graphs. There are numerous domains in the real world for which directed networks can be applied. In the future, we hope to apply our approach to such networks and to compare its performance with the directed versions of the other approaches considered in the experiments.

Bibliography

[1] S. Abiteboul, P. Buneman, and D. Suciu. *Data on the Web: from relations to semistructured data and XML*. Morgan Kaufmann, 2000.

[2] L. A Adamic and E.n Adar. Friends and neighbors on the Web. *Social Networks*, 25(2):211–230, 2003.

[3] R. Albert, A. Barabási, and H. Jeong. Mean-field theory for scale-free random networks. *Physica A*, 272:173–187, 1999.

[4] Y. Altun, T. Hofmann, and M. Johnson. Discriminative learning for label sequences via boosting. In S. Becker, S. Thrun, and K. Obermayer, editors, *Advances in Neural Information Processing Systems 15*, Cambridge, MA, 2003. MIT Press.

[5] Y. Altun, M. Johnson, and T. Hofmann. Investigating loss functions and optimization methods for discriminative learning of label sequences. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 2003.

[6] Y. Altun, I. Tsochantaridis, and T. Hofmann. Hidden Markov support vector machines. In *Proceedings of the Twentieth International Conference on Machine Learning*, 2003.

[7] R. A. Baeza-Yates and B. A. Ribeiro-Neto. *Modern Information Retrieval*. ACM Press / Addison-Wesley, 1999.

[8] C. Bahlmann, B. Haasdonk, and H. Burkhardt. On-line handwriting recognition with support vector machines – a kernel approach. In *Proc. of the 8th Int. Workshop on Frontiers in Handwriting Recognition (IWFHR)*, pages 49–54, 2002.

[9] A. L. Barabási, J. Jeong, Z. Néda., E. Ravasz, A. Shubert, and T. Vicsek. Evolution of the social network of scientific collaborations. *Physica A*, 311(3-4):590–614, 2002.

[10] 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.

[11] M. Collins. Discriminative reranking for natural language parsing. In *Proceedings of the Seventeenth International Conference on Machine Learning*, pages 175–182, San Francisco, CA, 2000. Morgan Kaufmann.

[12] M. Collins. Discriminative training methods for hidden Markov models: Theory and experiments with perceptron algorithms. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 2002.

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

[14] M. Collins and N. Duffy. Convolution kernels for natural language. In *Advances in Neural Information Processing Systems 14*, Cambridge, MA, 2002. MIT Press.

[15] T. C. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms.* MIT Press, Cambridge, MA, 1990.

[16] C. Cortes, P. Haffner, and M. Mohri. Rational kernels. In S. Becker, S. Thrun, and K. Obermayer, editors, *Advances in Neural Information Processing Systems 15*, Cambridge, MA, 2003. MIT Press.

[17] T. F. Cox and M. A. A. Cox. *Multidimensional Scaling, 2nd ed.* Chapman and Hall, 2004.

[18] L. De Raedt and S. Kramer. The levelwise version space algorithm and its application to molecular fragment finding. In B. Nebel, editor, *Proceedings of the 17th International Joint Conference on Artificial Intelligence*, pages 853–862. Morgan Kaufmann, 2001.

[19] R. Durbin, S. Eddy, A. Krogh, and G. Mitchison. *Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids.* Cambridge University Press, 1998.

[20] Y. Freund and R. Shapire. Large margin classification using the perceptron algorithm. *Machine Learning*, 37(3):277–296, 1999.

[21] T. Gärtner. Exponential and geometric kernels for graphs. In *NIPS*02 Workshop on Unreal Data: Principles of Modeling Nonvectorial Data*, 2002. Available from http://mlg.anu.edu.au/unrealdata/.

[22] T. Gärtner. A survey of kernels for structured data. *SIGKDD Explorations*, 5(1):S268–S275, 2003.

[23] T. Gärtner, P. Flach, and S. Wrobel. On graph kernels: Hardness results and efficient alternatives. In *Proceedings of the Sixteenth Annual Conference on Com-*

putational Learning Theory, 2003.

[24] T. Gebara and R. Kondor. Bhattacharyya and expected likelihood kernels. In *Proc. of the 16th COLT*, 2003.

[25] L. Getoor and C. P. Diehl. Link mining: a survey. *SIGKDD Explorations*, 7(2):3–12, 2005.

[26] M. A. Hasan, V. Chaoji, S. Salem, and M. Zaki. Link prediction using supervised learning. In *Workshop on Link Discovery: Issues, Approaches and Applications (LinkKDD-2005)*, 2005.

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

[28] C. Helma, R. D. King, S. Kramer, and A. Srinivasan. The Predictive Toxicology Challenge 2000-2001. *Bioinformatics*, 17(1):107–108, 2001.

[29] D. Helmbold, R. Schapire, Y. Singer, and M. Warmuth. A comparison of new and old algorithms for a mixture estimation problem. In *Proceedings of the Eighth Annual Workshop on Computational Learning Theory (COLT)*, pages 69–78, 1995.

[30] R. V. Hogg and Craig. *Introduction to Mathematical Statistics, 5th ed.* Macmillan, 1995.

[31] L. Holm and C. Sander. Protein structure comparison by alignment of distance matrices. *Journal of Molecular Biology*, 233(1):123–38, 1993.

[32] T. Ide and H. Kashima. Eigenspace-based anomaly detection in computer systems. In *Proceedings of the Tenth ACM SIGKDD Conference (KDD)*, 2004.

[33] A. Inokuchi, T. Washio, and H. Motoda. An Apriori-based algorithm for mining frequent substructures from graph data. In *The Fourth European Conference on Principles and Practice of Knowledge Discovery in Databases*, pages 13–23, 2000.

[34] 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.

[35] T. S. Jaakkola and D. Haussler. Exploiting generative models in discriminative classifiers. In M. S. Kearns, S. A. Solla, and D. A. Cohn, editors, *Advances in Neural Information Processing Systems 11*, Cambridge, MA, 1999. MIT Press.

[36] T. Joachim. Text categorization with support vector machines. In *Proceedings of the tenth European Conference on Machine Learning*, 1998.

[37] S. Kakade, Y. W. Teh, and S. Roweis. An alternative objective function for

Markovian fields. In *Proceedings of the Nineteenth International Conference on Machine Learning*, pages 275–282, San Francisco, CA, 2002. Morgan Kaufmann.

[38] H. Kanayama, K. Torisawa, Y. Mitsuishi, and J. Tsujii. A hybrid Japanese parser with hand-crafted grammar and statistics. In *Proceedings of the 18th International Conference on Computational Linguistics*, pages 411–417, 2000.

[39] 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.

[40] J. Kandola, J. Shawe-Taylor, and N. Cristianini. Learning semantic similarity. In S. Becker, S. Thrun, and K. Obermayer, editors, *Advances in Neural Information Processing Systems 15*. MIT Press, 2003. In press.

[41] M. Kanehisa, S. Goto, S. Kawashima, Y. Okuno, and M. Hattori. The KEGG resources for deciphering the genome. *Nucleic Acids Research*, 32:D277–D280, 2004.

[42] H. Kashima and A. Inokuchi. Kernels for graph classification. In *IEEE ICDM Workshop on Active Mining*. Maebashi, Japan, 2002.

[43] H. Kashima and T. Koyanagi. Kernels for semi-structured date. In *Proceedings of the Nineteenth International Conference on Machine Learning*, pages 291–298, San Francisco, CA, 2002. Morgan Kaufmann.

[44] H. Kashima, K. Tsuda, and A. Inokuchi. Marginalized kernels between labeled graphs. In *Proceedings of the Twentieth International Conference on Machine Learning*, San Francisco, CA, 2003. Morgan Kaufmann.

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

[46] J. M. Kleinberg, R. Kumar, P. Raghavan, S. Rajagopalan, and A. S. Tomkins. The Web as a graph: Measurements, models and methods. *Lecture Notes in Computer Science*, 1627:1–17, 1999.

[47] R. I. Kondor and J. Lafferty. Diffusion kernels on graphs and other discrete input spaces. In *Proceedings of the Nineteenth International Conference on Machine Learning*, pages 315–322, 2002.

[48] D. C. Kozen. *The design and analysis of algorithms, chapter 26: counting problems and #P*. Springer, 1992.

[49] S. Kramer and L. De Raedt. Feature construction with version spaces for biochemical application. In *Proc. of the 18th ICML*, pages 258–265, 2001.

[50] R. Kumar, P. Raghavan, S. Rajagopalan, D. Sivakumar, A. Tomkins, and E. Upfal. Stochastic models for the web graph. In *IEEE Symposium on Foundations of Computer Science (FOCS)*, 2000.

[51] J. Lafferty and G. Lebanon. Information diffusion kernels. In S. Becker, S. Thrun, and K. Obermayer, editors, *Advances in Neural Information Processing Systems 15*. MIT Press, 2003. In press.

[52] J. Lafferty, A. McCallum, and F. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the Eighteenth International Conference on Machine Learning*, pages 282–289, San Francisco, CA, 2001. Morgan Kaufmann.

[53] C. Leslie, E. Eskin, and W. S. Noble. The spectrum kernel: A string kernel for SVM protein classification. In R. B. Altman, A. K. Dunker, L. Hunter, K. Lauerdale, and T. E. Klein, editors, *Proceedings of the Pacific Symposium on Biocomputing*, pages 566–575. World Scientific, 2002.

[54] D. Liben-Nowelly and J. Kleinberg. The link prediction problem for social networks. In *Proceedings of the Twelfth International Conference on Information and Knowledge Management (CIKM)*, pages 556–559, 2004.

[55] H. Lodhi, C. Saunders, J. Shawe-Taylor, N. Cristianini, and C. Watkins. Text classification using string kernels. *Journal of Machine Learning Research*, 2:419–444, 2002.

[56] R. B. Lyngso, C. N. S. Pedersen, and H. Nielsen. Metrics and similarity measures for hidden markov models. In *Proc. of the 7th ISMB*, 1999.

[57] C. D. Manning and H. Schütze. *Foundations of Statistical Natural Language Processing*. The MIT Press, Cambridge, MA, 1999.

[58] A. McCallum, D.Freitag, and F. Pereira. Maximum entropy Markov models for information extraction and segmentation. In *Proceedings of the Seventeenth International Conference on Machine Learning*, pages 591–598, San Francisco, CA, 2000. Morgan Kaufmann.

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

[60] A. Nakamura and N. Abe. Collaborative filtering using weighted majority prediction algorithms. In *Proceedings of the Fifteenth International Conference on Machine Learning (ICML)*, pages 395–403, 1998.

[61] M. E. J. Newman. Clustering and preferential attachment in growing networks. *Physical Review Letters E*, 64(025102), 2001.

[62] M. E. J. Newman. The structure and function of complex networks. *SIAM Review*, 45(2):167–256, 2003.

[63] K. Nigam, A. K. McCallum, S. Thrun, and T. M. Mitchell. Text classification from labeled and unlabeled documents using EM. *Machine Learning*, 39(2/3):103–134, 2000.

[64] J. O'Madadhain, J. Hutchins, and P. Smyth. Prediction and ranking algorithms for event-based network data. *SIGKDD Explorations*, 7(2):23–30, 2005.

[65] A. Popescul and L. H. Ungar. Statistical relational learning for link prediction. In *IJCAI 2003 Workshop on Learning Statistical Models from Relational Data*, 2003.

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

[67] G. Salton and C. Buckley. Term weighting approaches in automatic text retrieval. *Information Processing and Management*, 24(5):513–523, 1988.

[68] B. Schölkopf and A. J. Smola. *Learning with Kernels*. MIT Press, Cambridge, MA, 2002.

[69] B. Schölkopf, K. Tsuda, and J.-P. Vert, editors. *Kernel Methods in Bioinformatics*. MIT Press, Cambridge, MA, 2004.

[70] J. Shawe-Taylor and N. Cristianini. *Kernel Methods for Pattern Analysis*. Cambridge University Press, 2004.

[71] H. Shimodaira, K.-I. Noma, M. Nakai, and S. Sagayama. Dynamic time-alignment kernel in support vector machine. In T. G. Dietterich, S. Becker, and Z. Ghahramani, editors, *Advances in Neural Information Processing Systems 14*, pages 921–928, Cambridge, MA, 2002. MIT Press.

[72] N. Srebro. *Learning with Matrix Factorization*. PhD thesis, Massachusetts Institute of Technology, 2004.

[73] 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.

[74] E. Takimoto and M. K. Warmuth. Path kernels and multiplicative updates. In *Proc. of the 15th Annual Conference on Computational Learning Theory*, pages 74–89, 2002.

[75] B. Taskar, M. Wong, P. Abbeel, and D. Koller. Link prediction in relational data. In *Neural Information Processing System*, 2003.

[76] K. Tsuda, T. Kin, and K. Asai. Marginalized kernels for biological sequences.

Bioinformatics, 18(Suppl. 1):S268–S275, 2002.

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

[78] S. P. Vadhan. The complexity of counting in sparse, regular, and planar graphs. *SIAM Journal on Computing*, 31(2):398–427, 2001.

[79] L. G. Valiant. The complexity of computing the permanent. *Theoretical Computer Science*, 8:189–201, 1979.

[80] L. G. Valiant. The complexity of enumeration and reliability problems. *SIAM Journal on Computing*, 8(3):410–421, 1979.

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

[82] V.N. Vapnik. *Statistical Learning Theory*. Wiley, New York, 1998.

[83] C. von Merin, R. Krause, B. Snel, M. Cornell, S. G. Olivier, S. Fields, and P. Bork. Comparative assessment of large-scale data sets of protein-protein interactions. *Nature*, 417:399–403, 2002.

[84] S. Wasserman and K. Faust. *Social Network Analysis*. Cambridge University Press, 1994.

[85] C. Watkins. Dynamic alignment kernels. In A.J. Smola, P.L. Bartlett, B. Schölkopf, and D. Schuurmans, editors, *Advances in Large Margin Classifiers*, pages 39–50. MIT Press, Cambridge, MA, 2000.

[86] J. Weston, O. Chapelle, A. Elisseeff, B. Schölkopf, and V. Vapnik. Kernel dependency estimation. In S. Becker, S. Thrun, and K. Obermayer, editors, *Advances in Neural Information Processing Systems 15*, Cambridge, MA, 2003. MIT Press.

[87] Y. Yamanishi, J.-P. Vert, and M. Kanehisa. Supervised enzyme network inference from the integration of genomic data and chemical information. *Bioinformatics*, 21:i468–i477, 2005.

[88] H. Chen Z. Huang, X. Li. Link prediction approach to collaborative filtering. In *Proceedings of the Fifth ACM/IEEE-CS joint conference on Digital libraries (JCDL)*, 2005.

List of Publications by the Author

Publications included in this thesis

Journal Papers

1. 鹿島 久嗣, 坂本 比呂志, 小柳 光生. 木構造データに対するカーネル関数の設計と解析, **人工知能学会論文誌**, 21(1):113-121, 2006.
2. 鹿島 久嗣, 安倍直樹. ネットワーク構造の確率的な時変モデルに基づく教師ありリンク予測, **人工知能学会論文誌**, 22(2):209-217, 2007

Book Chapters

1. Hisashi Kashima , Koji Tsuda, and Akihiro Inokuchi: Kernels for Graphs, In *Kernel Methods in Computational Biology*, pages 155-170, MIT press, Cambridge, MA, 2004.

Peer-reviewed Conference Papers

1. Hisashi Kashima and Naoki Abe. A Parameterized Probabilistic Model of Network Evolution for Supervised Link Prediction, In *Proceedings of the Sixth IEEE International Conference on Data Mining (ICDM2006)*, pages 340-349, Hong Kong, China, 2006.
2. Hisashi Kashima and Yuta Tsuboi: Kernel-Based Discriminative Learning Algorithms for Labeling Sequences, Trees and Graphs, In *Proceedings of the Twenty-first International Conference on Machine Learning (ICML2004)*, pages 457-464, Banff, Alberta, Canada, 2004.
3. Hisashi Kashima, Koji Tsuda, and Akihiro Inokuchi: Marginalized Kernels Between Labeled Graphs, In *Proceedings of the Twentieth International Conference on Machine Learning (ICML2003)*, pages 321-328, Washington, DC USA,

2003.

4. Hisashi Kashima and Akihiro Inokuchi: Kernels for Graph Classification, In *Proceedings of the First ICDM Workshop on Active Mining (AM-2002)*, pages 31-36, Maebashi, Japan, 2002.
5. Hisashi Kashima and Teruo Koyanagi: Kernels for Semi-Structured Data, In *Proceedings of the Nineteenth International Conference on Machine Learning (ICML2002)*, pages 291-298, Sydney, Australia, 2002.

Other Publications by the author

Journal Papers

1. Tetsuji Kuboyama, Hisashi Kashima, Kiyoko F. Aoki-Kinoshita, Kouichi Hirata, and Hiroshi Yasuda, A Spectrum Tree Kernel, **人工知能学会論文誌**, 22(2):, 2007.
2. 鹿島 久嗣, 津村 直史, 井手 剛, 野ヶ山 尊秀, 平出 涼, 江藤 博明, 福田 剛志, ネットワークデータを用いた分散システムにおける異常検出, **電子情報通信学会論文誌**, J89-D(2):183-198, 2006.
3. Tetsuo Shibuya, Hisashi Kashima, and Akihiko Konagaya: Efficient Filtering Methods for Clustering cDNAs with Spliced Sequence Alignment, *Bioinformatics*, 20(1):29-39, 2004.

Peer-reviewed Conference Papers

1. Tetsuji Kuboyama, Hisashi Kashima, Kiyoko F. Aoki-Kinoshita, Koichi Hirata, and Hiroshi Yasuda, A Spectrum Tree Kernel, In *Proceedings of The International Workshop on Data-Mining and Statistical Science (DMSS2006)*, Sapporo, Japan, 2006.
2. Tetsuji Kuboyama, Kilho Shin, and Hisashi Kashima, Flexible Tree Kernels Based on Counting the Number of Tree Mappings, In *Proceedings of Workshop on Mining and Learning* (held with *ECML/PKDD 2006*), Berlin, Germany, 2006.
3. Hisashi Kashima, Risk-Sensitive Learning via Expected Shortfall Minimization, In *Proceedings of 2006 SIAM Conference on Data Mining (SDM06)*, Bethesda, Maryland, USA, 2006.
4. Hisashi Kashima, Tadashi Tsumura, Tsuyoshi Ide, Takahide Nogayama, Ryo Hirade, Hiroaki Etoh, and Takeshi Fukuda, Network-Based Problem Detection for Distributed Systems, In *Procedings of the Twenty-first International Conference on Data Engineering (ICDE2005)*, Tokyo, Japan, 2005.
5. Tsuyoshi Ide and Hisashi Kashima, Eigenspace-based Anomaly Detection in Computer Systems, In *Proceedings of the Tenth ACM SIGKDD Conference (KDD2004)*, Seattle, WA, USA, 2004.

6. Akihiro Inokuchi and Hisashi Kashima, Mining Significant Pairs of Patterns from Graph Structures with Class Labels, In *Proceedings of the Third IEEE International Conference on Data Mining (ICDM2003)*, Melbourne, Florida, USA, 2003.