32. R. Wille. Knowledge acquisition by methods of formal concept analysis. In E. Diday, ed. *Data Analysis, Learning Symbolic and Numeric Knowledge*. ISBN:0-941743-64-0. Nova Science Publishers, pp. 365–380, New York, 1989.
33. M. Zaki. SPADE: An efficient algorithm of mining frequent sequences. *Machine Learning Journal*. 42(1/2):31–60, 2001.
34. H. Zhou. Multiplicativity. Part I. Variations, multiplicative graphs and digraphs. *Journal of graph Theory*. 15(5):469–488, 1991.

11

KERNEL METHODS FOR GRAPHS

THOMAS GÄRTNER,[1] TAMÁS HORVÁTH,[1] QUOC V. LE,[2] ALEX J. SMOLA,[2] AND STEFAN WROBEL[1,3]

[1]*Fraunhofer AIS, Schloß Birlinghoven, Sankt Augustin, Germany*
[2]*Statistical Machine Learning Program, NICTA and ANU Canberra, Australia*
[3]*Department of Computer Science III, University of Bonn, Bonn Germany*

11.1 INTRODUCTION

Supervised learning is one of the most commonly considered data mining scenarios. The *supervised learning problem*—which we will concentrate on in this chapter—is to find a function that estimates a fixed but unknown functional or conditional dependence between objects and one of their properties—given some exemplary objects for which this property has been observed. The objects with observed property are called *training instances* and those for which the property has to be estimated are *test instances*. In the most common setting, known as *induction*, a good model of the dependence has to be found without knowing the test instances. A less common but nevertheless important problem is—given training and test instances—to find a model that has good predictive performance on the test data. This setting is known as *transduction*. In both cases, whenever the property takes one of a finite set of possible values, we speak of *classification*; whenever it takes real values, we speak of *regression*.

Mining Graph Data, Edited by Diane J. Cook and Lawrence B. Holder

Traditionally, machine learning research has focused on data that can directly be represented in a Euclidean space. Today, such methods are frequently applied in the analysis of business and scientific data. However, it has turned out that more and more of the collected data has a structure that hinders direct representation in a Euclidean space. Two powerful languages that can be used to represent such data are first- or higher-order logic [27] and graphs. Although graphs are special relational structures, for complexity reasons it is useful to consider them separately. In this chapter, we focus on graphs as our representation language only.

Two of the most frequently occurring machine learning problems involving graph structures are classification of vertices in a graph and classification of graphs in a graph database, that is, a set of disjoint graphs. A simple example of the first problem is the classification of Web pages on the World Wide Web, given the links between the pages. A simple example of the second problem is the classification of chemical compounds, given their atom bond structure. Both of these learning problems will be tackled in later parts of this chapter.

Our approach is to adapt kernel methods [32], a class of well-established learning algorithms, to graphs, rather than developing a new learning algorithm for graph-structured data from scratch. Today kernel methods are one of the most widely—and most successfully—applied classes of learning algorithms. Interestingly, one can look at two parts of kernel methods almost independently: a *kernel function* and a kernel-based *learning algorithm*. The kernel function is a positive definite function on the instances and "isolates" the learning algorithm from the instances, that is, the learning algorithm does not need to access any particular aspect of an instance—it relies on kernels between instances only. Hence, kernel methods can be applied to any data structure by defining a suitable kernel function. The main purpose of this chapter is to review some recent advances in the rapidly developing research area of *kernel methods for graphs*. This chapter is based on the work previously reported in [11–14, 18, 19] and is organized as follows: Section 11.2 presents recent results on making the classification of graphs tractable. Section 11.3 presents recent work on making transductive multiclass classification of vertices in a large graph tractable. Section 11.4 concludes.

11.2 GRAPH CLASSIFICATION

In this section we consider the classification of instances that have a natural representation as labeled graphs. That is, each instance is described by a graph $g = (V, E, \alpha, \beta)$, where the elements of V and E are labeled by the symbols of some appropriately chosen alphabets L_V and L_E, respectively (i.e., $\alpha : V \to L_V$ and $\beta : E \to L_E$). Depending on the set E of edges, g is either directed (i.e., when $E \subseteq V \times V$) or undirected (i.e., when $E \subseteq \{X \in 2^V : |X| = 2\}$). For a set G of graphs, we assume without loss of generality that the elements of G are all defined over the same vertex and edge alphabets, and that the graphs in G are either all directed or all undirected. For further basic notions and notations on graphs, the reader is referred to the introductory chapter of this book.

To classify graph-structured instances with kernel methods, for example, support vector machines [6], it suffices to define a valid (positive definite) kernel function $k : G \times G \to \mathbb{R}$, also referred to as *graph kernel*. In the design of practically useful graph kernels, one would require them to be computable in polynomial time in the size of the graphs and to distinguish between nonisomorphic graphs, that is, the underlying embedding function $\phi : g \mapsto k(g, \cdot)$ to be injective modulo isomorphism. Such graph kernels are called *complete* graph kernels. For the computation of complete graph kernels, the following result holds [13].

Proposition 11.1 Computing any complete graph kernel is at least as hard as deciding whether two graphs are isomorphic.

Since the graph isomorphism problem is believed to be not in P (and thus most likely not efficiently solvable, even though it is also probably not NP-complete), one has to resort to graph kernels where the underlying embedding functions may map some nonisomorphic graphs to the same point. Although this may seem to be a severe relaxation, empirical results indicate that several incomplete graph kernels have excellent predictive performance on real-world datasets. In the subsequent sections, for example, we also present some noncomplete graph kernels that have good predictive performance on a large benchmark chemical graph dataset.

The design of most graph kernels is based on the following general idea:

1. Map each graph to some distinguished, not necessarily finite, (multi)set of patterns occurring in the graph.
2. Define the graph kernel for each pair g_1, g_2 of graphs as a kernel on the (multi)sets assigned to g_1 and g_2.

As an example of this approach, we mention the family of graph kernels based on mapping each graph to a certain set of subgraphs occurring frequently in the set of graphs representing the instances. Encouraging empirical results have recently been reported for this approach (see [8] and Chapter 7 of this book). Such frequent pattern-based graph kernels, however, involve the problem of choosing a trade-off between predictive power and runtime, as both these conflicting requirements depend on the choice of the frequency threshold. As an alternative to graph kernels based on frequent subgraphs, in this section we describe two graphs kernels, the *walk kernel* (WK) [13] and the *cyclic pattern kernel* (CPK) [18, 19], which do not depend on such a frequency threshold.

11.2.1 Walk-Based Graph Kernels

Following the general methodology sketched above, in this section we define two kernels for labeled *directed* graphs. Both kernel definitions are based on mapping graphs to multisets of label sequences corresponding to *walks* in the graphs. Such graph kernels are called *walk kernels*.

To go into the technical details, we need some basic definitions. Let G be a set of labeled directed graphs (i.e., each graph in G is defined over the same alphabets of vertex and edge labels) and let $g = (V, E, \alpha, \beta)$ be a graph in G. A *walk* w in g is a sequence of vertices $w = v_1, v_2, \ldots v_{\ell+1}$ such that $(v_i, v_{i+1}) \in E$ for every $i = 1, \ldots, \ell$. The *length* of the walk is equal to the number of edges in this sequence, that is, ℓ in the above case. To compute walk kernels in a compact way, we use the *adjacency matrix* M_g of g, defined by

$$[M_g]_{ij} = \begin{cases} 1 & \text{if } (v_i, v_j) \in E \\ 0 & \text{otherwise} \end{cases}$$

for every $v_i, v_j \in V$.

Another central concept for the definition of walk kernels is the notion of products of labeled directed graphs. Let $g_1 = (V_1, E_1, \alpha_1, \beta_1)$ and $g_2 = (V_2, E_2, \alpha_2, \beta_2)$ be directed graphs over the same alphabets L_V and L_E of vertex and edge labels, respectively. Then the *direct product* of g_1 and g_2 is the directed graph $g_1 \times g_2 = (V, E, \alpha, \beta)$ over the alphabets L_V and L_E, where

$$\begin{aligned} V &= \{(v_1, v_2) \in V_1 \times V_2 : \alpha_1(v_1) = \alpha_2(v_2)\} \\ E &= \{((u_1, u_2), (v_1, v_2)) \in V \times V : \\ &\qquad (u_1, v_1) \in E_1, (u_2, v_2) \in E_2, \text{ and } \beta_1(u_1, v_1) = \beta_2(u_2, v_2)\} \end{aligned}$$

and α (resp. β) maps each vertex (resp. each edge) to the common label of its components.

The Direct Product Kernel. The walk kernel described first in this section is based on defining one feature for every possible label sequence and then counting how many walks in a graph match this label sequence. The inner product in this feature space can be computed with the following closed form: Let G be a set of directed graphs and let $g_1 = (V_1, E_1, \alpha_1, \beta_1)$ and $g_2 = (V_2, E_2, \alpha_2, \beta_2)$ be graphs in G. Let $M_{g_1 \times g_2}$ and $V_{g_1 \times g_2}$ denote the adjacency matrix and the vertex set of the direct product $g_1 \times g_2$, respectively. With a sequence of weights $\lambda_0, \lambda_1, \ldots$ ($\lambda_i \in \mathbb{R}$; $\lambda_i \geq 0$ for all $i \in \mathbb{N}$) the *direct product kernel* is defined as

$$k_\times(g_1, g_2) = \sum_{i,j=1}^{|V_{g_1 \times g_2}|} \left[\sum_{\ell=0}^{\infty} \lambda_\ell M_{g_1 \times g_2}^\ell \right]_{ij}$$

if the limit exists. Note that every entry $\left[M_{g_1 \times g_2}^\ell\right]_{ij}$ of the ℓth power of $M_{g_1 \times g_2}$ is the number of walks of length ℓ from $v_i = (v_{i,1}, v_{i,2})$ to $v_j = (v_{j,1}, v_{j,2})$ in the product graph $g_1 \times g_2$. This is in turn equal to the number of all possible pairs of walks of length ℓ from $v_{i,1}$ to $v_{j,1}$ in g_1 and from $v_{i,2}$ to $v_{j,2}$ in g_2 with the same label sequence.

To compute this kernel function one can make use of polynomial time-computable closed forms or resort to approximations by short walks on the graphs (see [13] for further details).

The Noncontiguous Sequence Kernel. A variant of the above kernel function that can be used whenever the label sequences are unlikely to match exactly is the following kernel. Let g_1 and g_2 be the graphs as defined above and let $g_1 \circ g_2$ be their direct product when ignoring the labels in g_1 and g_2. With a sequence of weights $\lambda_0, \lambda_1, \ldots$ ($\lambda_i \in \mathbb{R}$; $\lambda_i \geq 0$ for all $i \in \mathbb{N}$) and a factor $0 \leq \mu \leq 1$ penalizing gaps, the *noncontiguous sequence kernel* is defined as

$$k_*(g_1, g_2) = \sum_{i,j=1}^{|V_{g_1 \times g_2}|} \left[\sum_{\ell=0}^{\infty} \lambda_\ell \left((1-\mu) M_{g_1 \times g_2} + \mu M_{g_1 \circ g_2} \right)^\ell \right]_{ij}$$

if the limit exists.

This kernel is very similar to the direct product kernel. The only difference is that instead of $M_{g_1 \times g_2}$, the matrix $(1-\mu)M_{g_1 \times g_2} + \mu M_{g_1 \circ g_2}$ is used. The relationship can be seen by adding—parallel to each edge—a new edge labeled # with weight $\sqrt{\mu}$ in both factor graphs.

11.2.2 Cycle-Based Graph Kernels

In this section we present another graph kernel, called the cyclic pattern kernel (CPK), which is based on mapping graphs into a distinguished set of cycles and trees. Using the labels of vertices and edges, these cycles and trees are then mapped to strings called *cyclic* and *tree patterns*. For two graphs, CPK is defined as the cardinality of the intersection of their cyclic and tree patterns. Although below we define CPK for undirected graphs, we note that the approach can easily be adapted to directed graphs as well.

To simplify the description, we first define a function that will be used many times in what follows. Let U be a set and $k_\cap : 2^U \times 2^U \to \mathbb{R}$ be the function defined by

$$k_\cap : (S_1, S_2) \mapsto |S_1 \cap S_2| \tag{11.1}$$

for every $S_1, S_2 \subseteq U$. The proof of the next proposition follows directly from the definition.

Proposition 11.2 $k_\cap$ is a kernel.

In order to define CPK, we recall some basic notions related to graphs. Let $g = (V, E, \alpha, \beta)$ be an undirected graph. Two vertices of g are *adjacent* if they are connected by an edge. The *degree* of a vertex $v \in V$ is the number of vertices adjacent to v. A graph $g' = (V', E', \alpha', \beta')$ is a *subgraph* of g, if $V' \subseteq V$, $E' \subseteq E$,

$\alpha'(v) = \alpha(v)$ for every $v \in V'$, and $\beta'(e) = \beta(e)$ for every $e \in E'$. A walk of g is a sequence $w = v_1, v_2, \ldots, v_{\ell+1}$ of vertices of g such that $\{v_i, v_{i+1}\} \in E$ for every $i = 1, \ldots, \ell$. If there is a walk between any pair of its vertices, g is *connected*. A *connected component* of g is a maximal subgraph of g that is connected. A vertex $v \in V$ is an *articulation* vertex if its removal increases the number of connected components of g. If it contains no articulation vertex, g is *biconnected*. A *biconnected component* of g is a maximal subgraph that is biconnected. A subgraph c of g forms a *simple cycle* if it is connected and each of its vertices has degree 2. We denote by $\mathcal{S}(g)$ the set of simple cycles of g. We note that the number of simple cycles can grow *faster* than $2^{|V|}$.

It holds that the biconnected components of a graph g are pairwise edge disjoint and thus form a partition on the set of edges of g. This partition, in turn, corresponds to the following equivalence relation on the set of edges: Two edges are equivalent if and only if they belong to a common simple cycle. This property of biconnected components implies that an edge of a graph belongs to a simple cycle if and only if its biconnected component contains more than one edge. Edges not belonging to simple cycles are called *bridges*. The subgraph of a graph g formed by its bridges is denoted by $\mathcal{B}g$. Clearly, each bridge of a graph is a singleton biconnected component, and $\mathcal{B}(g)$ is a forest.

11.2.2.1 CPK Defined by the Set of All Simple Cycles. Let G be a set of undirected graphs over the vertex and edge alphabets L_V and L_E, respectively. We assume without loss of generality that L_V and L_E are disjoint. Let $\Sigma = L_V \cup L_E$, Γ be an alphabet, and let π be a mapping from the set of simple cycles and trees labeled by Σ to Γ^* such that

(i) π is injective modulo isomorphism on the set of cycles and trees,
(ii) π can be computed in polynomial time.

We note that such a π always exists and can easily be constructed (see, e.g., [19, 39]). Using π, the set of *cyclic* and *tree patterns* of a graph $g \in G$ is defined by

$$\mathcal{C}(g) = \{\pi(C) : C \in \mathcal{S}(g)\} \tag{11.2}$$

$$\mathcal{T}(g) = \{\pi(T) : T \text{ is a connected component of } \mathcal{B}(g)\} \tag{11.3}$$

respectively. The *cyclic pattern kernel* for G is then defined by

$$k_{\mathcal{S}}(g_1, g_2) = k_{\cap}(\mathcal{C}(g_1), \mathcal{C}(g_2)) + k_{\cap}(\mathcal{T}(g_1), \mathcal{T}(g_2)) \tag{11.4}$$

for every $g_1, g_2 \in G$. Since the sets $\mathcal{C}(g)$ and $\mathcal{T}(g)$ are disjoint for every $g \in G$, $k_{\mathcal{S}}$ is a kernel by Proposition 11.2.

As mentioned previously, kernels can often be computed efficiently using some closed form, that is, without explicitly performing the embedding into the feature space. Unfortunately, unless P = NP, $k_{\mathcal{S}}$ cannot be computed in polynomial time. In fact, one can show the following stronger negative result.

Algorithm* 11.1 *Computing CPK

Require: labeled undirected graphs g_1 and g_2
Ensure: $k_{\mathcal{S}}(g_1, g_2)$

1: **for** $i = 1, 2$ **do begin**
2: compute $\mathcal{B}(g_i)$
3: compute $\mathcal{T}(g_i)$ from $\mathcal{B}(g_i)$
4: compute $\mathcal{S}(g_i)$
5: compute $\mathcal{C}(g_i)$ from $\mathcal{S}(g_i)$
6: **end**
7: **return** $|\mathcal{C}(g_1) \cap \mathcal{C}(g_2)| + |\mathcal{T}(g_1) \cap \mathcal{T}(g_2)|$

THEOREM 11.1 Computing $k_{\mathcal{S}}$ is at least as hard as counting simple cycles of length k in a graph.

This problem is, however, unlikely to be fixed-parameter tractable [10]. That is, for any graph with n vertices, most likely there is no algorithm counting simple cycles of length k in time $f(k) \cdot n^c$, where $f : \mathbb{N} \to \mathbb{N}$ is some computable function and c is some constant.

Based on the above negative complexity result, one can consider Algorithm 11.1, which calculates CPK by explicitly performing the embedding of graphs into the feature space. Since the biconnected components of a graph can be computed efficiently [34] and their number is bounded by the size (i.e., number of vertices) of the graph, steps 2 and 3 of Algorithm 11.1 can be computed in polynomial time. Since the set of simple cycles of a graph can be enumerated with polynomial delay [29], steps 4 and 5 can be performed in time polynomial in the sizes of g_1 and g_2, and in their number of simple cycles, which in turn can be exponential in the size of these graphs. Thus Algorithm 11.1 provides an efficient way of computing $k_{\mathcal{S}}$ for well-behaved graph databases, that is, for sets of disjoint graphs containing graphs with a small number of simple cycles.

The restriction to such well-behaved graph databases is necessary as while $\mathcal{S}(g)$ can be listed in time polynomial in $|\mathcal{S}(g)|$ [29], for the enumeration of the set $\mathcal{C}(g)$ of cyclic patterns the following negative result holds [19].

Proposition 11.3 If P $\neq$ NP, the set of cyclic patterns of a labeled undirected graph cannot be enumerated in output-polynomial time.

Hence, although $k_{\mathcal{S}}$ depends only on *cyclic patterns* defined in (11.2), in steps 3 and 4 of Algorithm 11.1 we compute the set of all *simple cycles*. Algorithm 11.1 is thus polynomial in $|\mathcal{S}(g_1)|$ and $|\mathcal{S}(g_2)|$ rather than in $|\mathcal{C}(g_1)|$ and $|\mathcal{C}(g_2)|$.

Restricting CPK to well-behaved graph databases is a theoretically rather severe restriction because graphs containing exponentially many simple cycles may only have polynomially or even constantly many cyclic patterns. As an example, let g be the biconnected graph defined in Figure 11.1 such that g's vertices and edges

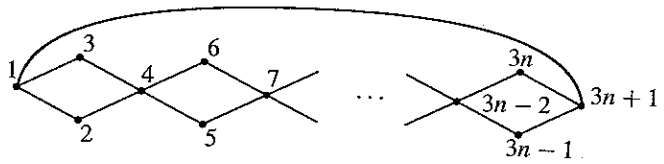

Figure 11.1. Graph of order $3n+1$ such that edges and vertices are labeled by the same symbol (not shown in the figure). It contains $2^n + n$ simple cycles but only 2 cyclic patterns.

are labeled by the same symbol. Also g is made up of $3n + 1$ vertices and contains $2^n + n$ simple cycles, which in turn form, however, only two different cyclic patterns. Despite the worst-case exponential number of simple cycles, it turns out in practice that there are large-scale real-world graph databases that are well-behaved, for example, the NCI-HIV dataset consisting of more than 42,000 molecules, that we will use to evaluate our graph kernels.

In the next sections we present two approaches relaxing the above computational limitation of CPK.

11.2.2.2 CPK for Graphs with Bounded Treewidth. The proof of Proposition 11.3 is based on a polynomial-time reduction from the NP-complete Hamiltonian cycle problem. This and many other NP-hard computational problems dealing with graphs become, however, polynomially solvable when restricted to graphs with *bounded treewidth* (see, e.g., [3] for an overview). Treewidth [31] is a measure of tree-likeness of graphs. More precisely, let $g = (V, E, \alpha, \beta)$ be a graph. A *tree decomposition* of g is a pair $(T, \mathcal{V})$, where $T = (I, F)$ is a tree and $\mathcal{V} = \{V_i \subseteq V : i \in I\}$ is a family of subsets of V such that

(i) $\bigcup_{i \in I} V_i = V$,
(ii) For every $\{u, v\} \in E$ there is an $i \in I$ such that $\{u, v\} \subseteq V_i$
(iii) For every $i, j, k \in I$ it holds that if j is on the path from i to k in T, then $V_i \cap V_k \subseteq V_j$.

The *width* of a tree decomposition $(T, \mathcal{V})$ of g is $\max_{i \in I} |V_i| - 1$, and the *treewidth* of g is the width of a tree decomposition of g with the smallest width. Clearly, the treewidth of a tree is 1, and the treewidth of a simple cycle is 2.

The class of bounded treewidth graphs includes many practically relevant graph classes (see, e.g., [4] for an overview). We also note that graphs with small treewidth may have exponentially many simple cycles. As an example, consider again the graph given in Figure 11.1. This graph has treewidth 2 for every $n > 0$ and one of its tree decompositions with minimum treewidth is shown in Figure 11.2.

Using the positive result on the regular-language-constrained simple path problem for bounded treewidth graphs given in [1], the following positive result can be shown.

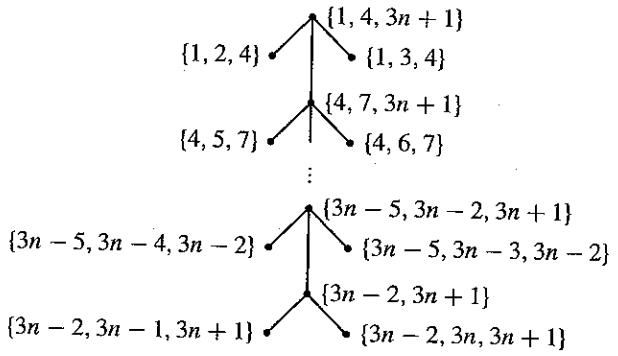

Figure 11.2. Tree decomposition of width 2 for the graph given in Figure 11.1.

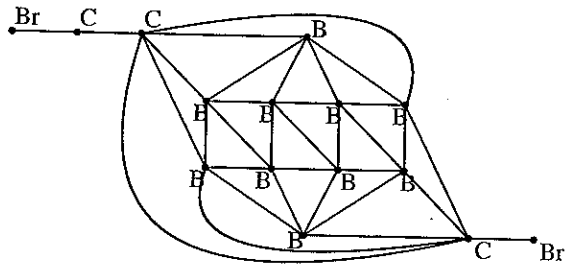

Figure 11.3. Chemical compound $C_3H_2B_{10}Br_2$ from the NCI-HIV dataset (NSC Id: 676529).

THEOREM 11.1 ([18]) For $i = 1, 2$, let $g_i = (V_i, E_i, \alpha_i, \beta_i)$ be graphs with constant bounded treewidth. Then $k_{\mathcal{S}}(g_1, g_2)$ can be computed in time polynomial in

$$\max\{|V_1|, |V_2|, |\mathcal{C}(g_1)|, |\mathcal{C}(g_2)|\}$$

We omit the proof of this theorem and refer the reader to [18] for further details.

11.2.2.3 CPK Based on Relevant Cycles. Consider the chemical structure graph depicted in Figure 11.3. This graph has a single biconnected component with 12 vertices, containing 12,878 simple cycles. To avoid the computation of such a big set of simple cycles, in this section we empirically investigate whether another, possibly smaller set of cyclic structures can be employed without significant loss of predictive performance. For that we consider cyclic patterns based on the *relevant* cycles [28] of a graph. For instance, the molecular graph in Figure 11.3 has only 19 relevant cycles, each with length 3. (Note that this graph contains altogether 20

To recall the definition of relevant cycles, we start by some basic notions from algebraic graph theory. Let $g = (V, E, \alpha, \beta)$ be a biconnected graph. A subgraph c of g is a *cycle* if each vertex of c has even degree. Note that simple cycles are a special case of cycles. Each cycle $c = (V', E', \alpha', \beta')$ of g can be represented by the incidence vector $\mathbf{c}$ of its edges. That is, the components of $\mathbf{c}$ are indexed by E, and for every $e \in E$, $\mathbf{c}_e = 1$ if $e \in E'$ and it is 0 otherwise. The vectors corresponding to cycles in g form a vector space over the binary field GF(2), called the *cycle space* (or *cycle vector space*). Addition in GF(2) is defined by $0 \oplus 0 = 0$, $0 \oplus 1 = 1$, and $1 \oplus 1 = 0$. Multiplication is given by $0 \otimes 0 = 0$, $0 \otimes 1 = 0$, and $1 \otimes 1 = 1$. Thus, vector addition in the cycle space corresponds to the symmetric difference of the sets of edges of the cycles represented by the vectors. The dimension of the cycle space of g is its *cyclomatic number* $\nu(g) = |E| - |V| + 1$ (g consists of a single connected component). To represent the cycle space of g, one can consider any of its bases with minimum length, where the length of a basis is the sum of the number of edges of the cycles represented by the vectors belonging to the basis. Since the minimum basis of a graph's cycle space is not necessarily unique, the cyclic structure of a graph is described by the union of all minimum bases of its cycle space [28]. This canonical set of cycles is called the set of *relevant cycles*. In [37], it is shown that *relevant cycles* can be enumerated with polynomial delay and counted in time polynomial in the order of G. Although the set of relevant cycles of a graph can be exponential in the number n of its vertices in the worst case, its cardinality is typically only cubic in n [15].

To measure the predictive performance of CPK based on relevant cycles, in our experiments we used monotone increasing subsets of simple cycles that can be generated by relevant cycles. More precisely, for a graph g and integer $k \geq 1$, let $\mathcal{R}_k(g)$ denote the set

$$\mathcal{R}_k(g) = \begin{cases} \text{set of relevant cycles of } g & \text{if } k = 1 \\ \{c \oplus c' \in \mathcal{S}(g) : c \in \mathcal{R}_{k-1}(g) \text{ and } c' \in \mathcal{R}_1(g)\} & \text{otherwise} \end{cases}$$

Since $\mathcal{R}_1(g) \subseteq \mathcal{S}(g)$, it holds that $\mathcal{R}_1(g) \subseteq \mathcal{R}_2(g) \subseteq \cdots \subseteq \mathcal{R}_{\nu(g)}(g) = \mathcal{S}(g)$. We note that the set of relevant cycles of a graph is the union of the relevant cycles of its biconnected components. Since biconnected components of a graph are enumerable in linear time [34], and relevant cycles of a biconnected graph are enumerable with polynomial delay [37], $\mathcal{R}_k(g)$ can be computed in time polynomial in $|\mathcal{R}_k(g)|$ for any arbitrary graph g.

For a graph database G and integer $k \geq 1$, the CPK based on $\mathcal{R}_k(g)$, denoted $k_{\mathcal{R}_k}$, is then defined by

$$k_{\mathcal{R}_k}(g_1, g_2) = k_\cap(P_{\mathcal{R}_k}(g_1), P_{\mathcal{R}_k}(g_2)) + k_\cap(\mathcal{T}(g_1), \mathcal{T}(g_2))$$

for every $g_1, g_2 \in G$, where $P_{\mathcal{R}_k}(g) = \{\pi(c) : c \in \mathcal{R}_k(g)\}$ and $\mathcal{T}(g)$ is defined by (11.3) for every $g \in G$. The remarks above along with Proposition 11.2 imply that $k_{\mathcal{R}_k}$ is a kernel that can be computed in time polynomial in $\max\{n_1, |\mathcal{R}_k(g_1)|, n_2, |\mathcal{R}_k(g_2)|\}$, where n_1 and n_2 denote the number of vertices of g_1 and g_2, respectively.

11.2.3 Empirical Evaluation

In this section, we evaluate the predictive power of walk-based and cyclic pattern kernels on the NCI-HIV dataset[1] of chemical compounds that has frequently been used in the empirical evaluation of graph mining approaches (see, e.g., [5, 7, 8, 25]). Each compound in this dataset is described by its chemical structure and classified into one of three categories based on its capability to inhibit the HIV virus: confirmed inactive (CI), moderately active (CM), or active (CA). The dataset contains 42,689 molecules, 423 of which are active, 1081 are moderately active, and 41,185 are inactive. Since more than 99% of the corresponding 42,689 chemical graphs contain less than 1000 cycles, the cyclic and tree patterns for the whole dataset can be computed in about 10 min.

Practical Considerations for the Walk Kernel. For molecule classification the number of vertex labels is limited by the number of elements occurring in natural compounds. It is therefore reasonable to not just use the element of the atom as its label. Instead, we use the pair consisting of the atom's element and the multiset of all neighbors' elements as the label.[2] In the HIV dataset this increases the number of different labels from 62 to 1391.

The size of this dataset, in particular the size of the graphs in this dataset, hinders the computation of walk-based graph kernels by means of eigendecompositions on the product graphs. The largest graph contains 214 atoms (not counting hydrogen atoms). If all had the same label, the product graph would have 45,796 vertices. As different elements occur in this molecule, the product graph has less vertices. However, it turns out that the largest product graph (without the vertex coloring step) still has 34,645 vertices. The vertex coloring above changes the number of vertices with the same label, thus the product graph is reduced to 12,293 vertices. For each kernel computation, either eigendecomposition or inversion of the adjacency matrix of a product graph has to be performed. With cubic time complexity, such operations on matrices of this size are not feasible.

The only chance to compute graph kernels in this application is to approximate them. There are two choices. First we consider counting the number of walks in the product graph up to a certain depth. In our experiments it turned out that counting walks with 13 or less vertices is still feasible. An alternative is to explicitly construct the image of each graph in the feature space. In the original dataset 62 different labels occur and after the vertex coloring 1391 different labels occur. The size of the feature space of label sequences of length 13 is then $62^{13} > 10^{23}$ for the original dataset and $1391^{13} > 10^{40}$ with the vertex coloring. We would also have to take into account walks with less than 13 vertices, but at the same time not all walks will occur in at least one graph. The size of this feature space hinders explicit computation. We thus resorted to counting walks with 13 or less vertices in the product graph.

[1] http://cactus.nci.nih.gov/ncidb/download.html.

[2] We note that more sophisticated vertex coloring algorithms are frequently employed in isomorphism tests.

Experimental Methodology and Results. We compare our approach to the results presented in [7] and [8]. The classification problems considered there were: (1) distinguish CA from CM, (2) distinguish CA and CM from CI, and (3) distinguish CA from CI. Additionally, we will consider (4) distinguish CA from CM and CI. For each problem, the area under the ROC curve (AUC), averaged over a five-fold cross validation, is given for different misclassification cost settings.

To choose the parameters of the walk-based graph kernel (to be precise, we use the direct product kernel) we proceeded as follows. We split the smallest problem (1) into 10% for parameter tuning and 90% for evaluation. First we tried different parameters for the exponential weight (10^{-3}, 10^{-2}, 10^{-1}, 1, 10) in a single nearest-neighbor algorithm (leading to an average AUC of 66, 66, 67.4, 75.9, and 33.8%) and decided to use 1 from now. Next we needed to choose the complexity (regularization) parameter of the SVM. Here we tried different parameters (10^{-3}, 10^{-2}, 10^{-1} leading to an average AUC of 69.4, 71.6, and 70.8%) and found the parameter 10^{-2} to work best. Evaluating with an SVM and these parameters on the remaining 90% of the data, we achieved an average AUC of 82.0% and standard deviation 0.024.

For cyclic pattern kernels, only the complexity constant of the support vector machine has to be chosen. Here, the heuristic as implemented in SVM light [20] is used. Also, we did not use any vertex coloring with cyclic pattern kernels.

To compare our results to those achieved in previous work, we fixed these parameters and reran the experiments on the full data of all four problems. Table 11.1 summarizes these results and the results with FSG reported in [7]. In [8] the authors of [7] describe improved results (FSG*). There, the authors report results obtained with an optimized threshold on the frequency of patterns.[3] Clearly, the graph kernels proposed here outperform FSG and FSG* over all problems and misclassification cost settings.

To evaluate the significance of our results we proceeded as follows: As we did not know the variance of the area under the ROC curve for FSG, we assumed the same variance as obtained with graph kernels. Thus, to test the hypothesis that graph kernels significantly outperform FSG, we used a pooled sample variance equal to the variance exhibited by graph kernels. As FSG and graph kernels were applied in a five-fold cross validation, the estimated standard error of the average difference is the pooled sample variance times $\sqrt{\frac{2}{5}}$. The test statistic is then the average difference divided by its estimated standard error. This statistic follows a t distribution. The null hypothesis—graph kernels perform no better than FSG—can be rejected at the significance level α if the test statistic is greater than the corresponding percentile of the t distribution.

Table 11.1 shows the detailed results of this comparison. Walk-based graph kernels perform always better or at least not significantly worse than any other kernel. Cyclic pattern kernels are sometimes outperformed by walk-based graph kernels but can be computed much more efficiently. For example, in the classification problem (4) fivefold cross validation with walk-based graph kernels finished in about 8 h while cyclic pattern kernels needed only 20 min.

[3] In [8] also including a description of the three-dimensional shape of each molecule is considered. We do not compare our results to those obtained using the three-dimensional information. We are considering to also include three-dimensional information in our future work and expect similar improvements.

TABLE 11.1 Area under the ROC Curve (AUC) in Percentage for Different Costs and Problems[a]

Task	Cost	Walk-Based Kernels (AUC in %)	Cyclic Pattern Kernels (AUC in %)	FSG (AUC in %)	FSG* (AUC in %)
(1)	1.0	81.8(±2.4)	81.3(±1.4)	77.4 $\ll_w \ll_c$	81.0
(1)	2.5	82.5(±3.2)	82.7(±1.3)	78.2 $<_w \ll_c$	79.2 $<_w \ll_c$
(2)	1.0	81.5(±1.5)	77.5(±1.7) $\ll_w$	74.2 $\ll_w \ll_c$	76.5 $\ll_w$
(2)	35.0	79.9(±1.1)	80.1(±1.7)	77.8 $\ll_w <_c$	79.4
(3)	1.0	94.2(±1.5)	91.9(±1.1) $<_w$	86.8 $\ll_w \ll_c$	83.9 $\ll_w \ll_c$
(3)	100.0	94.4(±1.5)	92.9(±1.0) $<_w$	91.4 $\ll_w <_c$	90.8 $\ll_w \ll_c$
(4)	1.0	92.6(±1.5)	90.8(±2.4) $<_w$	—	—
(4)	100.0	92.8(±1.3)	92.1(±2.6)	—	—

[a] $<_w$, significant loss against walk-based kernels at 10%; $\ll_w$, significant loss against walk-based kernels at 1%; $<_c$, significant loss against cyclic pattern kernels at 10%; $\ll_c$, significant loss against cyclic pattern kernels at 1%.

TABLE 11.2 Area under the ROC Curve (AUC) in Percentage for Different Costs and Problems [$\gg_X$ (resp. $>_X$) Significant Win at 5% (resp. 10%) Level wrt $k_{\mathcal{R}_X}$]

Task	Cost	$k_{\mathcal{R}_1}$ (AUC in %)	$k_{\mathcal{R}_2}$ (AUC in %)	$k_{\mathcal{R}_3}$ (AUC in %	$k_{\mathcal{R}_\infty} = k_S$ (AUC in %)
(1)	1.0	80.1(±4.5)	81.5(±3.1) $\gg_1$	81.4(±3.2)	81.3(±3.3)
(1)	2.5	82.1(±4.6)	83.0(±4.1) $\gg_1$	82.9(±3.9)	82.7(±4.2)
(2)	1.0	75.4(±2.2)	77.1(±2.3) $\gg_1$	77.8(±1.8) $\gg_1$	77.8(±1.9) $\gg_1$
(2)	35.0	79.5(±2.8)	80.0(±2.7)	80.4(±2.4) $\gg_1$	80.5(±2.5) $\gg_{1,2}$
(3)	1.0	91.1(±3.4)	92.5(±2.9) $\gg_1$	92.5(±2.7) $\gg_1$	92.6(±2.8) $\gg_1$
(3)	100.0	93.7(±2.4)	93.9(±1.9)	93.6(±1.8)	93.4(±2.0)
(4)	1.0	89.2(±3.2)	90.7(±2.5) $\gg_1$	90.8(±2.7) $\gg_1$	90.8(±2.7) $\gg_1$
(4)	100.0	92.9(±2.6)	92.9(±3.2) $>_\infty$	92.6(±2.9) $\gg_\infty$	92.2(±3.0)

Empirical Evaluation of $k_{\mathcal{R}_k}$. To evaluate CPK based on relevant cycles, we looked at the predictive performance of $k_{\mathcal{R}_1}$,$k_{\mathcal{R}_2}$,$k_{\mathcal{R}_3}$, and k_S. We used the above-described dataset and compared the different kernels with a paired t test on the same folds.[4] The results are given in Table 11.2. They indicate that $k_{\mathcal{R}_2}$ can be used in most of the cases without a significant loss of predictive performance wrt (with respect to) k_S and that $k_{\mathcal{R}_3}$ was never outperformed by k_S. Hence, the alternative definition of CPK allows one to apply it to graph databases containing graphs even with exponentially many simple cycles.

[4] However, note that the folds were different from the ones used in the above experiments.

11.3 VERTEX CLASSIFICATION

In this section we consider the classification vertices in a graph. That is, the vertices represent the instances of the learning problem, and we want to make use of our knowledge about common properties or relations between instances. While kernel functions between graphs (Section 11.2) can be used pretty directly with most available kernel methods, kernels for vertices raise somewhat different computational challenges. In this section we first give an overview of several kernels for vertices that have been defined in the literature. It turns out that for some of these more efficient learning is possible in a transductive setting. We thus develop an algorithm for *transductive Gaussian processes classification* of vertices that exploits this and perform an empirical evaluation on several datasets.

11.3.1 Kernels for Vertices

To apply Bayesian and other kernel methods to instances represented as vertices in a graph, we (only) need to define a covariance kernel on these instances. Hence, in this section we describe kernels that are based on knowledge about common properties of instances or relations between instances. We begin with a review of several kernels defined in the literature and discuss some efficiency issues later.

Kernel Definitions. The best known kernel in this class is the *diffusion kernel* [24]. The motivation behind this and similar kernel functions is that it is often easier to describe the local neighborhood of an instance than to describe the structure of the whole instance space. For example, the neighborhood of an instance could be all instances that differ from this one only by the presence or absence of one property. When working with molecules such properties might be bonds, functional groups, or particular chemical properties. The neighborhood relation obviously induces global information about the make up of the instance space. The approach taken in the kernels described in this section is to try to capture this global information in a kernel function merely based on the neighborhood description. To model the structure of instance spaces, undirected graphs or hypergraphs can be used. While the use of hypergraphs is less common in the literature, it appears more natural.

A hypergraph is defined by a set of vertices V—the instances—and a set of edges E, where each edge is a subset of vertices. Each edge of the hypergraph can be interpreted as some property that all vertices of the edge have in common. For documents, for example, the edges could correspond to words or citations that they have in common; in a metric space the hyperedge could include all vertices with distance less than a given threshold from some point. By a walk in a hypergraph we understand a sequence of vertices and edges $v_1, e_1, v_2, \ldots v_{\ell+1}$ where $v_i, v_{i+1} \in e_i \in E$ for all $1 \leq i \leq \ell$.

For a hypergraph we define the $|V| \times |E|$ matrix B by $B_{ij} = 1$ if and only if $v_i \in e_j$ and $B_{ij} = 0$, otherwise. Let then the $|V| \times |V|$ matrix D be defined by $D_{ii} = \sum_j \left[B^\top B\right]_{ij} = \sum_j \left[B^\top B\right]_{ji}$. The matrices $B^\top B$ and $L = D - B^\top B$ are

positive definite by construction. The matrix L is known as the graph Laplacian. Often the normalized Laplacian is used.

Conceptually, kernel matrices are then defined as the limits of matrix power series of the form

$$K = \sum_{i=0}^{\infty} \lambda_i \left(B^\top B\right)^i \quad \text{or} \quad K = \sum_{i=0}^{\infty} \lambda_i \left(-L\right)^i$$

with suitable parameters λ_i. These power series can be interpreted as measuring the number of walks of different lengths between given vertices. Sometimes the limit is approximated by a finite sum.

Limits of such power series can be computed by means of an eigenvalue decomposition of $B^\top B$ or $-L$, and a "recomposition" with modified eigenvalues. The modification of the eigenvalues is usually such that the order of eigenvalues is kept, while all eigenvalues are forced to become positive.

Examples for such kernel functions are the *diffusion kernel* [24]

$$K = \sum_{i=0}^{\infty} \frac{\beta^i}{i!} (-L)^i$$

the *von Neumann kernel* [23]

$$K = \sum_{i=1}^{\infty} \gamma^{i-1} \left(B^\top B\right)^i$$

and the *regularized Laplacian kernel* [33]

$$K = \sum_{i=1}^{\infty} \gamma^i (-L)^i$$

For exponential power series like the diffusion kernel, the limit can be computed by exponentiating the eigenvalues, while for geometrical power series, the limit can be computed by the formula $1/(1 - \gamma e)$, where e is an eigenvalue of $B^\top B$ or $-L$, respectively. Instead of computing K by manipulating the eigenvalues, one can alternatively compute kernels like the regularized Laplacian by inverting the matrix $\mathbf{1} + \gamma L$ ($\mathbf{1}$ denotes the identity matrix). A general framework and analysis of these kernels can be found in [33].

Efficiency Issues. While kernel functions between graphs (Section 11.2) can be used pretty directly with most available kernel methods, kernels for vertices raise somewhat different computational challenges. If the instance space is big, the computation of the kernels as defined above might be too expensive. Most kernel methods rely on computing Kv for some vector v several times in the course

of the algorithm. Obtaining K by matrix inversion or eigenvalue decomposition is expensive and even if L is sparse, K hardly is, making Kv expensive as well.

However, there is another issue: Assume that we are given the inverse kernel matrix K^{-1} on training and test set and we wish to perform induction only. In this case we need to compute the kernel matrix (or its inverse) restricted to the training set. Let

$$K^{-1} = \begin{bmatrix} A & B \\ B^\top & C \end{bmatrix}$$

Then the upper left-hand corner (representing the training set part only) of K is given by the Schur complement $\left(A - B^\top C^{-1} B\right)^{-1}$. Computing the latter is costly as it involves the inversion of C and we usually assume a very large amount of unlabeled data. Moreover, neither the Schur complement nor its inverse are typically sparse.

Here we have a nice connection between graphical models and graph kernels. Assume that the target property t is a normal random variable with conditional independence properties. In this case the inverse covariance matrix has nonzero entries only for variables with a direct dependency structure. This follows directly from an application of the Clifford–Hammersley theorem to Gaussian random variables [26]. In other words, if we are given a graphical model of normal random variables, their conditional independence structure is reflected by K^{-1}.

In the same way as in graphical models marginalization may induce dependencies, computing the kernel matrix on the training set only may lead to dense matrices, even when the inverse kernel on training and test data combined is sparse. The bottom line is there are cases where it is computationally cheaper to take both training and test set into account and optimize over a larger set of variables rather than dealing with a smaller dense matrix (which is expensive to obtain).

Later in this chapter we develop a transductive algorithm that can use the inverse kernel matrix instead of the kernel matrix—thus avoiding expensive matrix inversion or eigenvalue decompositions with graph kernels—and exploit the sparsity of the graph.

11.3.2 Bayesian Kernel Methods

Before we describe our transductive Gaussian processes classifier in detail, in this section we give a brief introduction to Bayesian kernel methods.

Exponential Family Distributions. We begin with a brief review of exponential family distributions. For the purpose of learning algorithms we are usually interested in the joint density $p(x, y|\theta)$ or the conditional density $p(y|\theta, x)$ of random variables x, y with parameters θ. A density $p(x, y|\theta)$ with $(x, y) \in \mathcal{X} \times \mathcal{Y}$ is in the exponential family whenever it can be expressed as

$$p(x, y|\theta) = \exp\left[\langle \phi(x, y), \theta \rangle - g(\theta)\right]$$

where

$$g(\theta) = \log \int_{\mathcal{X}\times\mathcal{Y}} \exp\left[\langle \phi(x, y), \theta \rangle\right] dx\, dy$$

is called the *log-partition function*, $\phi(x, y)$ are *sufficient statistics* of (x, y), and $\langle \cdot, \cdot \rangle$ denotes the inner product. It holds then that

$$\begin{aligned} \frac{\partial}{\partial \theta} g(\theta) &= \mathbf{E}_{p(x,y|\theta)}\left[\phi(x, y)\right] \\ \frac{\partial^2}{\partial \theta\, \partial \theta^t} g(\theta) &= \mathbf{E}_{p(x,y|\theta)}\left[\phi(x, y)\phi(x, y)^t\right] - \mathbf{E}_{p(x,y|\theta)}\left[\phi(x, y)\right]\mathbf{E}_{p(x,y|\theta)}\left[\phi(x, y)^t\right] \\ &= \mathbf{Cov}_{p(x,y|\theta)}\left[\phi(x, y)\right] \end{aligned}$$

and it can directly be seen that $p(x, y|\theta)$ is convex in θ.

From the joint exponential densities above, we can derive the conditional exponential densities as

$$p(y|x, \theta) = \exp\left[\langle \phi(x, y), \theta \rangle - g(\theta|x)\right]$$

where

$$g(\theta|x) = \log \int_{\mathcal{Y}} \exp\left(\langle \phi(x, y), \theta \rangle\right)\, dy$$

is the *conditional log-partition function*. It holds then that

$$\begin{aligned} \frac{\partial}{\partial \theta} g(\theta|x) &= \mathbf{E}_{p(y|x,\theta)}\left[\phi(x, y)\right] \\ \frac{\partial^2}{\partial \theta\, \partial \theta^t} g(\theta|x) &= \mathbf{Cov}_{p(y|x,\theta)}\left[\phi(x, y)\right] \end{aligned}$$

and it can directly be seen that $p(y|x, \theta)$ is convex in θ.

Bayesian Parameter Estimation. To estimate the label y of a new test point x from data $(X, Y) = \{(x_1, y_1), \ldots, (x_{|\mathcal{X}|}, y_{|\mathcal{X}|})\}$ under the assumption of a parameterized family of distributions, like the exponential, we need to compute

$$p(y|x, X, Y) = \int p(y|x, \theta) p(\theta|X, Y)\, d\theta$$

To avoid the integral over θ, we can alternatively use

$$p(y|x, \theta^*)$$

where

$$\theta^* = \arg\max_\theta p(\theta|X, Y)$$

The quantity $p(\theta|X, Y)$ is known as the *posterior* of the parameters and is related to the *likelihood* of the parameters $p(X, Y|\theta)$ as follows:

$$p(\theta|X, Y) = p(X, Y|\theta)\frac{p(\theta)}{p(X, Y)}$$

As $p(X, Y)$ is independent of θ, we can maximize the posterior $p(\theta|X, Y)$ by maximizing the likelihood $p(X, Y|\theta)$ times the prior $p(\theta)$ or—equivalently—minimize the negative log-posterior:

$$\begin{aligned} -\log p(\theta|X, Y) &= -\log \prod_{i=1}^{n} \exp\left[\langle \phi(x_i, y_i), \theta\rangle - g(\theta|x_i)\right] - \log p(\theta) + c' \\ &= \sum_{i=1}^{n} g(\theta|x_i) - \sum_{i=1}^{n} \langle \phi(x_i, y_i), \theta\rangle - \log p(\theta) + c' \end{aligned}$$

where $c' = \log p(X, Y) - \log(X|\theta) = \log p(X, Y) - \log(X) = p(Y|X)$ is independent of θ. Hence, it is a constant in the optimization problem and can be ignored.

Bayesian Kernel Methods. It turns out that under certain—rather general—conditions on the prior of the above minimization problem, it can be shown that the optimal parameters lie in the span of the sufficient statistics of the training data. This observation is known as the representer theorem. Thus we can replace θ by

$$\theta = \sum_j \int_{\mathcal{Y}} \alpha_{jy} \phi(x_j, y)\, dy$$

and minimize

$$\sum_{i=1}^{n} g(\theta|x_i) - \sum_{i,j=1}^{n} \int_{\mathcal{Y}} \alpha_{jy} \left\langle \phi(x_i, y_i), \phi(x_j, y)\right\rangle\, dy - \log p(\theta)$$

Assuming, for example, an uncorrelated Gaussian prior $p(\theta) \sim \mathcal{N}(0, \sigma^2 \mathbf{1})$ implies a Gaussian process on the random variable $t(x, y) := \langle \phi(x, y), \theta\rangle$ with covariance kernel k

$$\sigma^2 k((x, y), (x', y')) := \sigma^2 \left\langle \phi(x, y), \phi(x', y')\right\rangle = \mathrm{Cov}\left[t(x, y), t(x', y')\right] \qquad (11.5)$$

We thus obtain the minimization problem

$$\begin{aligned} &\sum_{i=1}^{n} g(\theta|x_i) - \sum_{i,j=1}^{n} \int_{\mathcal{Y}} \alpha_{jy} k\left((x_i, y_i), (x_j, y)\right)\, dy \\ &\quad - \sum_{i,j=1}^{n} \int_{\mathcal{Y}\times\mathcal{Y}} \alpha_{iy'} \alpha_{jy} k\left((x_i, y'), (x_j, y)\right)\, dy\, dy' \end{aligned}$$

with

$$g(\theta|x) = \log \int_{\mathcal{Y}} \exp\left[\sum_j \int_{\mathcal{Y}} \alpha_{jy} k\left((x_i, y'), (x_j, y)\right)\, dy\right] dy'$$

It remains to define a suitable joint covariance kernel $k : (\mathcal{X}\times\mathcal{Y}) \times (\mathcal{X}\times\mathcal{Y}) \to \mathbb{R}$. Usually this problem is simplified to the problem of defining the covariance of the instances $k_{\mathcal{X}} : \mathcal{X}\times\mathcal{X} \to \mathbb{R}$ based on the domain and the problem of defining the covariance of the labels based on the learning task $k_{\mathcal{Y}} : \mathcal{Y}\times\mathcal{Y} \to \mathbb{R}$. For regression often $k_{\mathcal{Y}}(y, y') = yy'$ is used; for classification often $k_{\mathcal{Y}}(y, y') = \delta_{yy'}$ is used. The joint covariance kernel is then simply the product $k((x, y), (x', y')) = k_{\mathcal{X}}(x, x')k_{\mathcal{Y}}(y, y')$. In the remainder of this section we will focus on $k_{\mathcal{X}}(\cdot, \cdot) = \langle \phi_{\mathcal{X}}(\cdot), \phi_{\mathcal{X}}(\cdot)\rangle$ and usually refer to it as k and ϕ.

11.3.3 Multiclass Transduction

A common problem of many transductive approaches is that they scale badly with the amount of unlabeled data, which prohibits the use of massive sets of unlabeled data. In this section, we thus present a transductive Gaussian process classifier for multiclass estimation problems. It performs particularly effective on graphs and other data structures for which the kernel matrix or its inverse have special numerical properties that allow fast matrix vector multiplication. An empirical evaluation of this algorithm shows improved predictive performance compared to both standard Gaussian process classification as well as transductive algorithms. We perform classification on a dataset consisting of a massive digraph with 75,888 vertices and 508,960 edges. To the best of our knowledge it has so far not been possible to perform transduction on graphs of this size in reasonable time (with standard hardware). On standard data our method shows competitive or better performance.

11.3.3.1 Gaussian Processes. We begin with a brief overview over Gaussian process multiclass classification [38] recast in terms of exponential families. Denote by $\mathcal{X}\times\mathcal{Y}$ with $\mathcal{Y} = \{1, \ldots, |\mathcal{Y}|\}$ the domain of observations and labels. Moreover let $X := \{x_1, \ldots, x_{|\mathcal{X}|}\}$ and $Y := \{y_1, \ldots, y_{|\mathcal{X}|}\}$ be the set of observations. It is our goal to estimate $y|x$ via

$$p(y|x, \theta) = \exp\left(\langle \phi(x, y), \theta\rangle - g(\theta|x)\right)$$

where

$$g(\theta|x) = \log \sum_{y\in\mathcal{Y}} \exp\left(\langle \phi(x, y), \theta\rangle\right)$$

$\phi(x, y)$ are the joint sufficient statistics of x and y and $g(\theta|x)$ is the conditional log-partition function that takes care of the normalization. We impose a normal prior

on θ, leading to the following negative joint likelihood in θ and Y:

$$\mathcal{P} := -\log p(\theta, Y|X) = \sum_{i=1}^{|\mathcal{X}|} \left[g(\theta|x_i) - \langle \phi(x_i, y_i), \theta \rangle \right] + \frac{1}{2\sigma^2} \|\theta\|^2 + c \quad (11.6)$$

for some constant c independent of θ and Y. For transduction purposes, $p(\theta, Y|X)$ will prove more useful than $p(\theta|Y, X)$.

Parametric Optimization Problem. In the following we assume isotropy[5] among the class labels, that is, $\langle \phi(x, y), \phi(x', y') \rangle = \delta_{y,y'} \langle \phi(x), \phi(x') \rangle$ where δ is the Kronecker δ. This allows us to decompose θ into $\theta_1, \ldots, \theta_{|\mathcal{Y}|}$ such that

$$\langle \phi(x, y), \theta \rangle = \langle \phi(x), \theta_y \rangle \text{ and } \|\theta\|^2 = \sum_{y=1}^{|\mathcal{Y}|} \|\theta_y\|^2 \quad (11.7)$$

Applying the representer theorem in conjunction with (11.7) gives

$$\theta_y = \sum_{i=1}^{|\mathcal{X}|} \alpha_{iy} \phi(x_i) \quad (11.8)$$

where $\alpha \in \mathbb{R}^{|\mathcal{X}| \times |\mathcal{Y}|}$.

Let $\mu \in \mathbb{R}^{|\mathcal{X}| \times |\mathcal{Y}|}$ with $\mu_{ij} = 1$ if $y_i = j$ and $\mu_{ij} = 0$ otherwise, and $K \in \mathbb{R}^{|\mathcal{X}| \times |\mathcal{X}|}$ with $K_{ij} = \langle \phi(x_i), \phi(x_j) \rangle$. The joint log-likelihood (11.6) can then be expressed in terms of α and K, yielding

$$\mathcal{P} := \sum_{i=1}^{|\mathcal{X}|} \log \sum_{y=1}^{|\mathcal{Y}|} \exp\left([K\alpha]_{iy}\right) - \operatorname{tr} \mu^\top K\alpha + \frac{1}{2\sigma^2} \operatorname{tr} \alpha^\top K\alpha + c \quad (11.9)$$

where tr denotes the trace. Equivalently we could expand (11.6) in terms of $t := K\alpha$:

$$-\log p(\theta|X, Y) = \sum_{i=1}^{|\mathcal{X}|} \log \sum_{y=1}^{|\mathcal{Y}|} \exp\left([t]_{iy}\right) - \operatorname{tr} \mu^\top t + \frac{1}{2\sigma^2} \operatorname{tr} t^\top K^{-1} t + c \quad (11.10)$$

This is commonly done in Gaussian process literature, and we will use both formulations, depending on the problem we need to solve: If $K\alpha$ can be computed effectively, as is the case with string kernels, we use the α parameterization. Conversely, if $K^{-1}\alpha$ is cheap, as, for example, with graph kernels, we use the t parameterization.

[5]This is not a necessary requirement for the efficiency of our algorithm, however, it greatly simplifies the presentation.

DERIVATIVES. Second-order methods such as conjugate gradient require the computation of derivatives of $-\log p(\theta, Y|X)$ with respect to θ in terms of α or t. Using the shorthand $\pi \in \mathbb{R}^{m \times n}$ with $\pi_{ij} := p(y = j|x_i, \theta)$ we have

$$\partial_\alpha \mathcal{P} = K(\pi - \mu + \sigma^{-2}\alpha) \quad (11.11a)$$

$$\partial_t \mathcal{P} = \pi - \mu + \sigma^{-2} K^{-1} t \quad (11.11b)$$

To avoid spelling out tensors of fourth order for the second derivatives (since $\alpha \in \mathbb{R}^{|\mathcal{X}| \times |\mathcal{Y}|}$), we state the action of the latter as bilinear forms on vectors $\beta, \gamma, u, v \in \mathbb{R}^{|\mathcal{X}| \times |\mathcal{Y}|}$:

$$\partial_\alpha^2 \mathcal{P}[\beta, \gamma] = \operatorname{tr}(K\gamma)^\top(\pi .* (K\beta)) - \operatorname{tr}(\pi .* K\gamma)^\top(\pi .* (K\beta)) + \sigma^{-2} \operatorname{tr} \gamma^\top K\beta \quad (11.12a)$$

$$\partial_t^2 \mathcal{P}[u, v] = \operatorname{tr} u^\top(\pi .* v) - \operatorname{tr}(\pi .* u)^\top(\pi .* v) + \sigma^{-2} \operatorname{tr} u^\top K^{-1} v. \quad (11.12b)$$

We used the Matlab notation of $.*$ to denote element-wise multiplication of matrices.

Let $L \cdot |\mathcal{Y}|$ be the computation time required to compute $K\alpha$ and $K^{-1}t$, respectively. One may check that $L = O(|\mathcal{X}|)$ implies that each conjugate gradient (CG) descent step can be performed in $O(|\mathcal{X}|)$ time. Combining this with rates of convergence for Newton-type or nonlinear CG solver strategies yields overall time costs in the order of $O(|\mathcal{X}| \log |\mathcal{X}|)$ to $O(|\mathcal{X}|^2)$ worst case, a significant improvement over conventional $O(|\mathcal{X}|^3)$ methods.

11.3.3.2 *Transductive Inference by Variational Methods.*

As we are interested in transduction, the labels Y (and analogously the data X) decompose as $Y = Y_{\text{train}} \cup Y_{\text{test}}$. To directly estimate $p(Y_{\text{test}}|X, Y_{\text{train}})$ we would need to integrate out θ, which is usually intractable. Instead, we now aim at estimating the mode of $p(\theta|X, Y_{\text{train}})$ by variational means. With the KL divergence D and an arbitrary distribution q the well-known bound (see, e.g., [22])

$$-\log p(\theta|X, Y_{\text{train}}) \leq -\log p(\theta|X, Y_{\text{train}}) + D(q(Y_{\text{test}}) \| p(Y_{\text{test}}|X, Y_{\text{train}}, \theta)) \quad (11.13)$$

$$= -\sum_{Y_{\text{test}}} \left(\log p(Y_{\text{test}}, \theta|X, Y_{\text{train}}) - \log q(Y_{\text{test}})\right) q(Y_{\text{test}}) \quad (11.14)$$

holds. This bound (11.14) can be minimized with respect to θ and q in an iterative fashion. The key trick is that while using a factorizing approximation for q we restrict the latter to distributions that satisfy balancing constraints. That is, we require them to yield marginals on the unlabeled data, which are comparable with the labeled observations.

DECOMPOSING THE VARIATIONAL BOUND. To simplify (11.14) observe that

$$p(Y_{\text{test}}, \theta | X, Y_{\text{train}}) = p(Y_{\text{train}}, Y_{\text{test}}, \theta | X) / p(Y_{\text{train}} | X) \tag{11.15}$$

In other words, the first term in (11.14) equals (11.9) up to a constant independent of θ or Y_{test}. With $q_{ij} := q(y_i = j)$, we define $\mu_{ij}(q) = q_{ij}$ for all $i > |\mathcal{X}_{\text{train}}|$ and $\mu_{ij}(q) = 1$ if $y_i = 1$ and 0 otherwise for all $i \leq |\mathcal{X}_{\text{train}}|$. In other words, we are taking the expectation in μ over all unobserved labels Y_{test} with respect to the distribution $q(Y_{\text{test}})$. We have

$$\sum_{Y_{\text{test}}} q(Y_{\text{test}}) \log p(Y_{\text{test}}, \theta | X, Y_{\text{train}}) = \sum_{i=1}^{|\mathcal{X}|} \log \sum_{j=1}^{|\mathcal{Y}|} \exp\left([K\alpha]_{ij}\right)$$

$$- \operatorname{tr} \mu(q)^\top K\alpha + \frac{1}{2\sigma^2} \operatorname{tr} \alpha^\top K \alpha + c \tag{11.16}$$

which we can of course also expand in $t = K\alpha$ as in (11.10). The second term in (11.14) is the entropy of q. Since q factorizes, we have

$$\sum_{Y_{\text{test}}} q(Y_{\text{test}}) \log q(Y_{\text{test}}) = \sum_{i=|\mathcal{X}_{\text{train}}|+1}^{|\mathcal{X}|} q_{ij} \log q_{ij} \tag{11.17}$$

For fixed q the optimization over θ proceeds as in the previous section for fully observed models. We now discuss optimization over q.

MARGINAL CONSTRAINTS ON q. It is unreasonable to assume that q may be chosen freely from all factorizing distributions (the latter would lead to a straightforward EM algorithm for transductive inference): If we observe a certain distribution of labels on the training set, for example, for binary classification we see 45% positive and 55% negative labels, then it is very unlikely that the label distribution on the test set deviates significantly. Hence we should make use of this information.

If $|\mathcal{X}| \gg |\mathcal{X}_{\text{train}}|$, a naive application of the variational bound can lead to cases where q is concentrated on one class—the increase in likelihood for a resulting very simple classifier completely outweighs any balancing constraints implicit in the data. This is confirmed by experimental results. It is, incidentally, also the reason why SVM transduction optimization codes [21] impose a balancing constraint on the assignment of test labels. We impose the following conditions:

$$r_j^- \leq \sum_{i=|\mathcal{X}_{\text{train}}|+1}^{|\mathcal{X}|} q_{ij} \leq r_j^+ \quad \text{for all } j \in \mathcal{Y}$$

and

$$\sum_{j=1}^{|\mathcal{Y}|} q_{ij} = 1 \quad \text{for all } i \in \left\{|\mathcal{X}|_r, \ldots, |\mathcal{X}|\right\}$$

Here the constraints $r_j^- = p_{\text{emp}}(y = j) - \epsilon$ and $r_j^+ = p_{\text{emp}}(y = j) + \epsilon$ are chosen such as to correspond to confidence intervals given by finite sample size tail bounds. In other words we set $p_{\text{emp}}(y = j) = |\mathcal{X}_{\text{train}}|^{-1} |\{i : 1 \leq i \leq |\mathcal{X}_{\text{train}}| \wedge y_i = j\}|$ and ϵ such as to satisfy

$$\Pr\left\{\left||\mathcal{X}_{\text{train}}|^{-1} \sum_{i=1}^{|\mathcal{X}_{\text{train}}|} \xi_i - |\mathcal{X}_{\text{test}}|^{-1} \sum_{i=1}^{|\mathcal{X}_{\text{test}}|} \xi_i'\right| > \epsilon\right\} \leq \delta \tag{11.18}$$

for iid $\{0, 1\}$ random variables ξ_i and ξ_i' with mean p. This is a standard ghost-sample inequality. It follows directly from [17, Eq. (2.7)] after application of a union bound over the class labels that

$$\epsilon \leq \sqrt{\log(2|\mathcal{Y}|/\delta)|\mathcal{X}| / (2|\mathcal{X}_{\text{train}}||\mathcal{X}_{\text{test}}|)} \tag{11.19}$$

11.3.3.3 Optimization. *Optimization in α and t:* $\mathcal{P}$ is convex in α (and in t since $t = K\alpha$). This means that Newton–conjugate gradient (NCG) and Newton–Raphson (NR) can be used for optimization. We use the following procedure [9]:

- Compute updates $\alpha \longleftarrow \alpha - \eta \partial_\alpha^2 \mathcal{P}^{-1} \partial_\alpha \mathcal{P}$ via
  - Solve the linear system approximately by conjugate-gradient iterations.
  - Find optimal η by line search.
- Repeat until the norm of the gradient is sufficiently small.

Key is the fact that the arising linear system is only solved approximately, which can be done using very few CG iterations. Since each of them is $O(|\mathcal{X}|)$ for fast kernel-vector computations the overall cost is a subquadratic function of $|\mathcal{X}|$.

OPTIMIZATION IN q. Is somewhat less straightforward: We need to find the optimal q in terms of KL divergence subject to the marginal constraint. Denote by τ the part of $K\alpha$ pertaining to test data, or more formally $\tau \in \mathbb{R}^{|\mathcal{X}_{\text{test}}| \times |\mathcal{Y}|}$ with $\tau_{ij} = [K\alpha]_{i+|\mathcal{X}_{\text{train}}|,j}$. We have

$$\underset{q}{\text{Minimize}} \quad \operatorname{tr} q^\top \tau + \sum_{i,j} q_{ij} \log q_{ij} \tag{11.20a}$$

$$\text{Subject to} \quad q_j^- \leq \sum_i q_{ij} \leq q_j^+ \text{ for all } j \in \mathcal{Y} \tag{11.20b}$$

$$\sum_j q_{ij} = 1 \text{ for all } i \in \{1, \ldots, |\mathcal{X}|\} \text{ and } q_{ij} \geq 0 \tag{11.20c}$$

Using Lagrange multipliers, one can show that q needs to satisfy $q_{ij} = \exp(-\tau_{ij}) b_i c_j$ where $b_i, c_j \geq 0$. Solving for $\sum_j^{|\mathcal{Y}|} q_{ij} = 1$ yields

$$q_{ij} = \frac{\exp(-\tau_{ij}) c_j}{\sum_{l=1}^{|\mathcal{Y}|} \exp(-\tau_{il}) c_l}$$

This means that instead of an optimization problem in $|\mathcal{X}_{\text{test}}| \times |\mathcal{Y}|$ variables, we only need to optimize over $|\mathcal{Y}|$ variables subject to $2|\mathcal{Y}|$ constraints. Using $\gamma_j = \log c_j$, it is

$$\underset{c}{\text{Minimize}} \quad \sum_{i,j} \frac{\exp(\gamma_j - \tau_{ij})}{\sum_{l=1}^{|\mathcal{Y}|} \exp(\gamma_l - \tau_{il})} \left(\gamma_j - \log \sum_{l=1}^{|\mathcal{Y}|} \exp(\gamma_l - \tau_{il}) \right) \tag{11.21a}$$

$$\text{Subject to} \quad q_j^- \leq \sum_i \frac{\exp(\gamma_j - \tau_{ij})}{\sum_{l=1}^{|\mathcal{Y}|} \exp(\gamma_l - \tau_{il})} \leq q_j^+ \tag{11.21b}$$

This problem now only depends on $|\mathcal{Y}|$ variables. It can be solved by standard second-order methods. As initialization we choose γ_l such that the per class averages match the marginal constraint while ignoring the per sample balance. After that a small number of Newton steps suffices for optimization.

11.3.4 Related Work

String Kernels:. Efficient computation of string kernels using suffix trees was described in [36]. In particular, it was observed that expansions of the form $\sum_{i=1}^{|\mathcal{X}|} \alpha_i k(x_i, x)$ can be evaluated in linear time in the length of x, provided some preprocessing for the coefficients α and observations x_i is performed. This preprocessing is independent of x and can be computed in $O(\sum_i |x_i|)$ time. The efficient computation scheme covers all kernels of type

$$k(x, x') = \sum_s w_s \#_s(x) \#_s(x') \tag{11.22}$$

for arbitrary $w_s \geq 0$. Here, $\#_s(x)$ denotes the number of occurrences of s in x and the sum is carried out over all substrings of x. This means that computation time for evaluating $K\alpha$ is again $O(\sum_i |x_i|)$ as we need to evaluate the kernel expansion for all $x \in X$. Since the average string length is independent of m, this yields an $O(m)$ algorithm for $K\alpha$.

VECTORS. If $k(x, x') = \phi(x)^\top \phi(x')$ and $\phi(x) \in \mathbb{R}^d$ for $d \ll |\mathcal{X}|$, it is possible to carry out matrix vector multiplications in $O(|\mathcal{X}|d)$ time. This is useful for cases where we have a sparse matrix with a small number of low-rank updates (e.g., from low-rank dense fill-ins).

EXISTING TRANSDUCTIVE APPROACHES. For SVMs use nonlinear programming [2] or EM-style iterations for binary classification [21]. Moreover, on graphs various methods for unsupervised learning have been proposed [40, 41], all of which are mainly concerned with computing the kernel matrix on training and test set jointly. Other formulations impose that the label assignment on the test set be consistent with the assumption of confident classification [35]. Others again exploit the fact that training and test set have similar marginal distributions [21].

The approach described in this chapter takes advantage of all three properties. Our formulation is particularly efficient whenever $K\alpha$ or $K^{-1}\alpha$ can be computed in linear time, where K is the kernel matrix and α is a coefficient vector. We approach the problem as follows:

- We require consistency of training and test marginals. This avoids problems with overly large majority classes and small training sets.
- Kernels (or their inverses) are computed on training and test set simultaneously. On graphs this can lead to considerable computational savings.
- Self-consistency of the estimates is achieved by a variational approach. This allows us to make use of Gaussian process multiclass formulations.

11.3.5 Empirical Evaluation

Unfortunately, we are not aware of other multiclass transductive learning algorithms. To still be able to compare our approach to other transductive learning algorithms, we performed experiments on some benchmark datasets. To investigate the performance of our algorithm in classifying vertices of a graph, we choose the WebKB dataset.

BENCHMARK DATASETS. Table 11.3 reports results on some benchmark datasets. To be able to compare the error rates of the transductive multiclass Gaussian process classifier proposed in this chapter, we also report error rates from [2] and compare to a simple Gaussian process classifier. The reported error rates are for 10-fold cross validations. Parameters were chosen on a single split inside each training fold.

GRAPH MINING. To illustrate the effectiveness of our approach on graphs we performed experiments on the well-known WebKB dataset. This dataset consists of 8275 Web pages classified into 7 classes. Each Web page contains textual content and/or links to other Web pages. As we are using this dataset to evaluate our graph mining algorithm, we ignore the text on each Web page and consider the dataset as a labeled directed graph. To have the data set as large as possible, in contrast to most other work, we did not remove any Web pages.

Table 11.4 reports the results of our algorithm on different subsets of the WebKB data as well as on the full data. We use the co-linkage graph and report results for "inverse" 10-fold stratified cross validations, that is, we use 1 fold as training data and 9 folds as test data. Parameters are the same for all reported experiments and were found by experimenting with a few parameter sets on the "Cornell" subset only. It turned out that the class membership probabilities are not well-calibrated on this dataset. To overcome this, we predict on the test set as follows: For each class the instances that are most likely to be in this class are picked (if they have not been picked for a class with lower index) such that the fraction of instances assigned to this class is the same on the training and test set. We will investigate the reason for this in future work.

The setting most similar to ours is probably the one described in [40]. Although a directed graph approach outperforms an undirected approach, we resorted to kernels

TABLE 11.3 Error Rates on Some Benchmark Datasets (Mostly from UCI)

Dataset	#Inst.	#Attr	Ind. GP (%)	Transd. GP (%)	S^3VMmip (%)[a]
cancer	699	9	3.4 ± 4.1	2.1 ± 4.7	3.4
cancer (progn.)	569	30	6.1 ± 3.7	6.0 ± 3.7	3.3
heart (cleave.)	297	13	15.0 ± 5.6	13.0 ± 6.3	16.0
housing	506	13	7.0 ± 1.0	6.8 ± 0.9	15.1
ionosphere	351	34	8.6 ± 6.3	6.1 ± 3.4	10.6
pima	769	8	19.6 ± 8.1	17.6 ± 8.0	22.2
sonar	208	60	10.5 ± 5.1	8.6 ± 3.4	21.9
glass	214	10	20.5 ± 1.6	17.3 ± 4.5	—
wine	178	13	19.4 ± 5.7	15.6 ± 4.2	—
tictactoe	958	9	3.9 ± 0.7	3.3 ± 0.6	—
cmc	1473	10	32.5 ± 7.1	28.9 ± 7.5	—
USPS	9298	256	5.9	4.8	—[b]

[a]The last column is the error rates reported in [2].
[b]In [2] only subsets of USPS were considered due to the size of this problem.

TABLE 11.4 Results on WebKB for "inverse" 10-fold Cross Validation

Dataset	$\|V\|$	$\|E\|$	Error (%)	Dataset	$\|V\|$	$\|E\|$	Error (%)
Cornell	867	1793	10	Misc	4113	4462	66
Texas	827	1683	8	All	8275	14370	53
Washington	1205	2368	10	Universities	4162	9591	12
Wisconsin	1263	3678	15				

for undirected graphs, as those are computationally more attractive. We will investigate computationally attractive digraph kernels in future work and expect similar benefits as reported by [40]. Though we are using more training data than [40], we are also considering a more difficult learning problem (multiclass without removing various instances). To investigate the behavior of our algorithm with less training data, we performed a 20-fold inverse cross validation on the "wisconsin" subset and observed an error rate of 17% there.

To show that the runtime performance of our algorithm is sufficient for classifying the vertices of massive graphs, we also performed initial experiments on the Epinions dataset [30]. The dataset is a social network consisting of 75,888 people connected by 508,960 "trust" edges. Additionally the dataset comes with a list of 185 "to previewers" for 25 topic areas. We tried to predict these but only got 12% of the top reviewers correct. As we are not aware of any predictive results on this task, we suppose this low accuracy is inherent to this task. However, the main point is that the experiments prove that the algorithm can be run on very large graph datasets.

11.4 CONCLUSIONS AND FUTURE WORK

In this chapter we reviewed some recent advances in the area of *kernel methods for graphs*. On the one hand we described kernels that can be used for graphs in a graph database. On the other hand we reviewed kernels that can be used for vertices in a single graph and introduced a transductive algorithm that allows efficient classification of vertices in a graph.

11.4.1 Graph Classification

The obvious approach to define kernels on objects that have a natural representation as a graph is to map each graph to a set of subgraphs and measure the intersection of the two sets. As mentioned above, such graph kernels cannot be computed efficiently if the mapping is required to be unique up to isomorphism.

In the literature different approaches have been tried to overcome this problem. Graepel [16] restricted the image of the graph to paths up to a given size, and [7] only considers the set of connected graphs that occur frequently as subgraphs in the graph database.

In this work we presented two effectively computable kernels for graphs. Although the underlying decompositions are not unique up to isomorphism, our experiments on a large chemical dataset indicate that the above complexity limitation does not hinder successful classification of molecules.

In future work we consider investigating more specialized kernels for molecules: Often more important than the chemical structure graph for the activity of a molecule is the three-dimensional (3D) structure of the molecule. Depending on the energy level, graphs can take different conformations and thus different 3D structures. Developing kernels that take 3D information into account might facilitate better predictive performance in such domains.

11.4.2 Vertex Classification

Current kernel methods for graphs do not scale very well with the available amount of unlabeled data. It turned out that for certain graph kernels it is more efficient to perform transduction than induction. We presented a transductive Gaussian process classifier for multiclass estimation problems. It performs particularly effectively on graphs and other data structures for which the kernel matrix or its inverse has special numerical properties that allow fast matrix vector multiplication. That said, also on standard dense problems we observed very large improvements (typically a 10% reduction of the training error) over standard induction.

Structured Labels and Conditional Random Fields are a clear area where one could extend the transductive setting. The key obstacle to overcome in this context is to find a suitable marginal distribution: With increasing structure of the labels the confidence bounds per subclass decrease dramatically. A promising strategy is to use only partial marginals on maximal cliques and enforce them directly, similarly to an unconditional Markov network.

Other Marginal Constraints than matching marginals are also worth exploring. In particular, constraints derived from exchangeable distributions such as those used by latent dirichlet allocation are a promising area to consider. This may also lead to connections between GP classification and clustering.

Sparse $O(m^{1.3})$ Solvers for Graphs have recently been proposed by the theoretical computer science community. It is worth exploring their use for inference on graphs.

Acknowledgments

The authors thank Mathew Richardson and Pedro Domingos for collecting the Epinions data and Deepayan Chakrabarti and Christos Faloutsos for providing a preprocessed version. Parts of this work were carried out when TG was visiting NICTA. National ICT Australia is funded through the Australian government's *Backing Australia's Ability* initiative, in part through the Australian Research Council. This work was partially supported by grants of the ARC, the Pascal Network of Excellence, and by the DFG project (WR 40/2-1) *Hybride Methoden und Systemarchitekturen für heterogene Informationsräume*.

REFERENCES

1. C. Barrett, R. Jacob, and M. Marathe. Formal-language-constrained path problems. *SIAM Journal on Computing*, 30(3):809–837, 2000.
2. K. Bennett. Combining support vector and mathematical programming methods for classification. In *Advances in Kernel Methods—Support Vector Learning*, pp. 307–326, MIT Press, Cambridge, 1998.
3. H. L. Bodlaender. A tourist guide through treewidth. *Acta Cybern.*, 11(1–2):1–22, 1993.
4. H. L. Bodlaender. A partial k-arboretum of graphs with bounded treewidth. *Theoretical Computer Science*, 209(1–2):1–45, 1998.
5. C. Borgelt and M. R. Berthold. Mining molecular fragments: Finding relevant substructures of molecules. In *Proceedings of the 2002 IEEE International Conference on Data Mining*. IEEE Computer Society, 2002.
6. B. E. Boser, I. M. Guyon, and V. N. Vapnik. A training algorithm for optimal margin classifiers. In D. Haussler, ed. *Proceedings of the Annual Conference on Computational Learning Theory*, pp. 144–152, ACM Press, New York, 1992.
7. M. Deshpande, M. Kuramochi, and G. Karypis. Automated approaches for classifying structures. In Proceedings of the 2nd ACM SIGKDD Workshop on Data Mining and Bioinformatics, 2002.
8. M. Deshpande, M. Kuramochi, and G. Karypis. Frequent sub-structure based approaches for classifying chemical compounds. In *Proceedings of the 3rd IEEE International Conference on Data Mining*, pp. 35–42. IEEE Computer Society, 2003.
9. R. Fletcher. *Practical Methods of Optimization*. Wiley, New York, 1989.
10. J. Flum and M. Grohe. The parameterized complexity of counting problems. *SIAM Journal on Computing*, 33(4):892–922, 2004.
11. T. Gärtner. A survey of kernels for structured data. *SIGKDD Explorations*, 5(1):49–58, July 2003.
12. T. Gärtner. Predictive graph mining with kernel methods. In *Advanced Methods for Knowledge Discovery from Complex Data*. Springer, Berlin, 2005.

13. T. Gärtner, P. A. Flach, and S. Wrobel. On graph kernels: Hardness results and efficient alternatives. In Proceedings of the 16th Annual Conference on Computational Learning Theory and the 7th Kernel Workshop, 2003.
14. T. Gärtner, Q. V. Le, S. Burton, A. J. Smola, and S. V. N. Vishwanathan. Large-scale multiclass transduction. *Advances in Neural Information Processing Systems, 18*: 411–418, 2006.
15. P. M. Gleiss and P. F. Stadler. Relevant cycles in biopolymers and random graph. In Proceedings of the 4th Slovene International Conference in Graph Theory, 1999.
16. T. Graepel. PAC-Bayesian pattern classification with kernels. Ph.D. thesis, TU Berlin, 2002.
17. W. Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58:13–30, 1963.
18. T. Horváth. Cyclic pattern kernels revisited. In *Proceedings of Advances in Knowledge Discovery and Data Mining, 9th Pacific-Asia Conference, PAKDD 2005*, Vol. 3518 of *LNAI*, pp. 791–801. Springer, Berlin, 2005.
19. T. Horváth, T. Gärtner, and S. Wrobel. Cyclic pattern kernels for predictive graph mining. In Proceedings of the 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 158–167, 2004.
20. T. Joachims. Making large-scale SVM learning practical. In B. Schölkopf, C. J. C. Burges, and A. J. Smola, eds. *Advances in Kernel Methods—Support Vector Learning*, pp. 169–184, MIT Press, Cambridge, MA, 1999.
21. T. Joachims. *Learning to Classify Text Using Support Vector Machines: Methods, Theory, and Algorithms*. The Kluwer International Series in Engineering and Computer Science. Kluwer Academic, Boston, 2002.
22. M. I. Jordan, Z. Ghahramani, Tommi S. Jaakkola, and L. K. Saul. An introduction to variational methods for graphical models. *Machine Learning*, 37(2):183–233, 1999.
23. J. Kandola, J. Shawe-Taylor, and N. Cristianini. Learning semantic similarity. In *Advances in Neural Information Processing Systems*, Vol. 15. MIT Press, Cambridge, MA, 2003.
24. I. R. Kondor and J. D. Lafferty. Diffusion kernels on graphs and other discrete structures. In Proceedings of the 19th International Conference on Machine Learning, pp. 315–312, 2002.
25. S. Kramer, L. De Raedt, and C. Helma. Molecular feature mining in HIV data. In Proceedings and the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 136–143, 2002.
26. S. L. Lauritzen. *Graphical Models*. Oxford University Press, Oxford, 1996.
27. J. W. Lloyd. *Logic for Learning*. Springer, Berlin, 2003.
28. M. Plotkin. Mathematical basis of ring-finding algorithms at CIDS. *J. Chem. Doc.*, 11:60–63, 1971.
29. R. C. Read and R. E. Tarjan. Bounds on backtrack algorithms for listing cycles, paths, and spanning trees. *Networks*, 5(3):237–252, 1975.
30. M. Richardson and P. Domingos. Mining knowledge-sharing sites for viral marketing. In Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2002.
31. N. Robertson and P. D. Seymour. Graph minors. II. Algorithmic aspects of tree-width. *Journal of Algorithms*, 7(3):309–322, 1986.
32. B. Schölkopf and A. J. Smola. *Learning with Kernels*. MIT Press, Cambridge, MA, 2002.

33. A. J. Smola and I. R. Kondor. Kernels and regularization on graphs. In B. Schölkopf and M. K. Warmuth, eds. *Proceedings of the Annual Conference on Computational Learning Theory*, Lecture Notes in Computer Science. Springer, Berlin, 2003.
34. R. Tarjan. Depth-first search and linear graph algorithms. *SIAM Journal on Computing*, 1(2):146–160, 1972.
35. V. Vapnik. *Statistical Learning Theory*. Wiley, New York, 1998.
36. S. V. N. Vishwanathan and A. J. Smola. Fast kernels for string and tree matching. In K. Tsuda, B. Schölkopf, and J.P. Vert, eds. *Kernels and Bioinformatics*, MIT Press, Cambridge, MA, 2004.
37. P. Vismara. Union of all the minimum cycle bases of a graph. *Electronic Journal of Combinatorics*, 4(1):73–87, 1997.
38. C. K. I. Williams and D. Barber. Bayesian classification with Gaussian processes. *IEEE Transactions on Pattern Analysis and Machine Intelligence PAMI*, 20(12):1342–1351, 1998.
39. M. Zaki. Efficiently mining frequent trees in a forest. In *Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 71–80, ACM Press, New York, 2002.
40. D. Zhou, J. Huang, and B. Schölkopf. Learning from labeled and unlabeled data on a directed graph. In International Conference on Machine Learning, 2005.
41. X. Zhu, J. Lafferty, and Z. Ghahramani. Semi-supervised learning using gaussian fields and harmonic functions. In International Conference on Machine Learning ICML'03, 2003.

12

KERNELS AS LINK ANALYSIS MEASURES

MASASHI SHIMBO AND TAKAHIKO ITO

Nara Institute of Science and Technology
Ikoma, Nara 630-0192, Japan

12.1 INTRODUCTION

Citations have been a major source of information for analyzing the relationship between scientific papers, authors, and journals from the early days of bibliometric studies. With the recent proliferation of hypertext documents on the Web, many attempts have been made to apply the methods proposed in bibliometrics to the structural analysis of the Web and also to develop new methods. Two notable methods, PageRank [5] and HITS (Hypertext-Induced Topic Search) [15], have emerged from the research and are used extensively to evaluate the *importance*, or popularity, of Web pages.

Another type of link analysis measure has been studied in bibliometrics, with an intention to quantify the *relatedness*, or similarity, of two given documents. Co-citation coupling [22] is one such classical measure of relatedness. This method is still widely used, for example, by the well-known scientific literature search system CiteSeer [4] to recommend related papers to users.

These two lines of link analysis measures, namely, importance and relatedness, have been discussed independently in the literature. One of the objectives of this chapter is to present a unified framework that accounts for both importance and relatedness and to make it possible to define measures intermediate between the two.

Mining Graph Data, Edited by Diane J. Cook and Lawrence B. Holder