1 A Short Tour of Kernel Methods for Graphs

Thomas Gärtner
Fraunhofer AIS.KD
Schloß Birlinghoven
Sankt Augustin, Germany

Quoc V. Le, Alex J Smola
Statistical Machine Learning Program
NICTA and ANU Canberra
Australia

1.1 Introduction

Machine learning research has – apart from some exceptions – originally concentrated on learning from data that can naturally be represented in a single table without links between the instances. Due to the needs of many real-world applications, in recent years an increasing amount of reserch has been devoted to machine learning on relational data with more complex structure. This book is then concerned with a very popular subject among this research, graphical models for relational data. In this chapter now we will give a short introduction to another popular subject, kernel methods for relational data, in particular graph spaces.

Kernel Methods can loosely be characterised as learning algorithms that take as information about the data only the covariance structure of the dataset. This way we can look at two separate aspects of kernel methods almost independently, the kernel function defining the covariance structure and the kernel-based learning algorithm. Most popular kernel methods can naturally be derived from a regularised risk minimisation setting. In this setting, positive-definiteness of the kernel function has the additional benefit of rendering the optimisation problem convex such that the globally optimal solution can efficiently be found by appropriate algorithms.

The relation between kernel methods and graphical models can perhaps be described best by having a brief look at Gaussian process regression. This algorithm is based on the idea of modelling the distribution of the labels of any finite dataset by a multivariate normal distribution with given covariance function and additive Gaussian noise. If the data obeys the Markov properties relative to a graph G, the implied conditional independence restrictions are reflected by zero entries in the concentration matrix, i.e., in the inverse covariance matrix. This clearly limits the choise of potential kernel functions on such data. Gaussian processes are in turn directly related to kernel methods derived from the regularised risk minimisation setting. Assuming square loss we can derive a kernel method known as regularised least squares regression. Given the same kernel function and setting the regularisation parameter corresponding to the variance of the noise, regularised least squares regression predicts the target values for test data that are the maximum likelihood predictions of the Gaussian process.

This chapter is organised as follows: We first give a brief overview of kernel methods and kernel functions. We thereby focus on kernels on sets of graphs and on kernels between vertices of a graphs. We will observe that kernel methods for graphs can be more efficient in a transductive setting than in an inductive one. We then describe a recently develloped algorithm for multiclass transduction based on Gaussian processes that can be applied effectively to graphs. This algorithm expoits the (usually implicitly made) assumption that training and test data come from the same distribution. After that, we show encouraging initial empirical results on the webkb dataset. Last but not least we discuss some related work and conclude. Parts of this chapter are based on [Gärtner, 2003, Gärtner et al., 2003, Gärtner, 2005, Gärtner et al., 2006, Gärtner et al., 2006, Horváth et al., 2004]

1.2 Kernel Methods for Graphs

Kernel methods [Schölkopf and Smola, 2002] are a popular class of algorithms within the machine learning and data mining communities. Being on one hand theoretically well founded in statistical learning theory, they have on the other hand shown good empirical results in many applications. One particular aspect of kernel methods such as the support vector machine is the formation of hypotheses by linear combination of positive-definite kernel functions 'centred' at individual training instances. By the restriction to positive definite kernel functions, the regularised risk minimisation problem (we will define this problem once we have defined positive definite functions) becomes convex and every locally optimal solution is globally optimal.

We begin with the definition of 'positive definiteness'.

Definition 1.1

A symmetric $n \times n$ matrix K is *positive definite* if for all $c \in \mathbb{R}^n$ it holds that

$$c^\top Kc \geq 0$$

and it is *strictly positive definite* if additionally $c^\top Kc = 0$ implies $c = 0$.

Definition 1.2
Let $\mathcal{X}$ be a set. A symmetric function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is a *positive definite kernel* on $\mathcal{X}$ if, for all $n \in \mathbb{N}$, $x_1, \ldots, x_n \in \mathcal{X}$, the matrix K with $K_{ij} = k(x_i, x_j)$ is positive definite. A positive definite kernel k is a *strictly positive definite kernel*, if $x_i = x_j \Leftrightarrow i = j$ implies that K is strictly positive definite.

Loosely speaking, a kernel function can be seen as a kind of similarity measure that is not normalised. A better intuition about kernel functions can be obtained by looking at them as inner products. More formally, for every positive defininite kernel $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ there exists a map $\phi : \mathcal{X} \to \mathcal{H}$ into a Hilbert space $\mathcal{H}$ such that $\forall\, x, x' \in \mathcal{X} : k(x, x') = \langle \phi(x), \phi(x') \rangle$.

1.2.1 Regularised risk minimisation

The usual supervised learning model [Vapnik, 1995] considers a set $\mathcal{X}$ of individuals and a set $\mathcal{Y}$ of labels, such that the relation between individuals and labels is a fixed but unknown probability measure on the set $\mathcal{X} \times \mathcal{Y}$. The common theme in many different kernel methods such as support vector machines or regularised least squares regression is to find a hypothesis function that minimises not just the empirical risk (training error) but the *regularised risk*. This gives rise to the optimisation problem

$$\min_{f(\cdot) \in \mathcal{H}} \frac{C}{m} \sum_{i=1}^{m} V(y_i, f(x_i)) + \|f(\cdot)\|_{\mathcal{H}}^2$$

where C is a parameter, $\{(x_i, y_i)\}_{i=1}^m$ is a set of individuals with known label (the training set), $\mathcal{H}$ is a set of functions forming a Hilbert space (the hypothesis space), and V is a function that takes on small values whenever $f(x_i)$ is a good guess for y_i and large values whenever it is a bad guess (the loss function). The *representer theorem* [Wahba, 1990, Schölkopf et al., 2001] shows that under rather general conditions on V, solutions of the above optimisation problem have the form

$$f(\cdot) = \sum_{i=1}^{m} c_i k(x_i, \cdot) \tag{1.1}$$

and the norm of these functions can be computed as

$$\|f(\cdot)\|_{\mathcal{H}}^2 = c^\top K c$$

where $K_{ij} = k(x_i, x_j)$. Different kernel methods arise then from using different loss functions.

Regularised Least Squares
Choosing the square loss function, i.e., $V(y_i, f(x_i)) = (y_i - f(x_i))^2$, we obtain the optimisation problem of the regularised least squares algorithm [Rifkin, 2002,

Saunders et al., 1998]:

$$\min_{f(\cdot)\in\mathcal{H}} \frac{C}{m}\sum_{i=1}^{m}(y_i - f(x_i))^2 + \|f(\cdot)\|_{\mathcal{H}}^2 \tag{1.2}$$

Plugging in our knowledge about the form of solutions and taking the directional derivative with respect to the parameter vector c of the function (1.1), we can find the analytic solution to the optimisation problem as:

$$c = \left(K + \frac{m}{C}\mathbf{1}\right)^{-1} y$$

where $\mathbf{1}$ denotes the identity matrix of appropriate size.
Support Vector Machines Support vector machines [Boser et al., 1992] are a kernel method that can be applied to binary supervised classification problems. They are derived from the above optimisation problem by choosing the so-called *hinge loss* $V(y, f(x)) = \max\{0, 1 - yf(x)\}$. The motivation for support vector machines often taken in literature is that the solution can be interpreted as a hyperplane that separates both classes (if it exists) and is maximally distant from the convex hulls of both classes. A different motivation is the computational attractiveness of sparse solutions of the function (1.1) used for classification.
For support vector machines the problem of minimising the regularised risk can be transformed into the so-called 'primal' optimisation problem of soft-margin support vector machines:

$$\begin{array}{lll} \min\limits_{c\in\mathbb{R}^n} & \frac{C}{n}\sum\limits_{i=1}^{m}\xi_i + c^\top K c & \\ \text{subject to:} & y_i \sum\limits_j c_j k(x_i, x_j) \geq 1 - \xi_i & i = 1, \dots m \\ & \xi_i \geq 0 & i = 1, \dots m \end{array}$$

1.2.2 Kernel Functions

Positive definiteness of a matrix K is also reflected by its eigenvalues: K is positive definite if and only if K has only non-negative eigenvalues and K is strictly positive definite if and only if K has only positive eigenvalues, i.e., no zero eigenvalues. In turn K is indefinite if there are c_+, c_- such that $c_+^\top K c_+ > 0 > c_-^\top K c_-$. This is again equivalent to K having positive and negative eigenvalues.
Let us now have a quick look at which combinations of matrices are positive definite.

1. For any matrix B, the matrix $B^\top B$ is positive definite.
2. For any two positive definite matrices G, H, the tensor product $G \otimes H$ is positive definite.
3. For any strictly positive definite matrix G and integer n, G^n is strictly positive definite. For any positive definite matrix G and integer $n > 0$, G^n is positive definite.
4. For any positive definite matrix G and real number $\gamma \geq 0$, the limit of the power

series $\sum_n^\infty \gamma^n G^n$ exists if γ is smaller than the inverse of the largest eigenvalue of G. Then also the limit is positive definite.

5. For any symmetric matrix G and real number β, the limit of the power series $\sum_n^\infty \frac{\beta^n}{n!} G^n$ exists and is positive definite.

An integral part of many kernels for structured data is the *decomposition* of an object into a (multi-) set of possibly overlapping parts and the computation of a kernel on (multi-) sets. We will thus next have a look at kernels for (multi-) sets.
The general case of interest for set kernels is when the instances X_i are elements of a semiring of sets $\mathfrak{S}$ and there is a measure μ with $\mathfrak{S}$ as its domain of definition. A natural choice of a kernel on such data is the following kernel function:

Definition 1.3
Let $\mu(\cdot)$ be a measure defined on the semiring of sets $\mathfrak{S}$. The *intersection kernel* $k_\cap : \mathfrak{S} \times \mathfrak{S} \to \mathbb{R}$ is defined as

$$k_\cap(X_i, X_j) = \mu(X_i \cap X_j); \qquad X_i, X_j \in \mathfrak{S} \ . \tag{1.3}$$

The intersection kernel is a positive definite kernel function and coincides in the simplest case (finite sets with $\mu(\cdot)$ being the set cardinality) with the inner product of the bitvector representations of the sets.
For nonempty sets we furthermore define the following kernels.

Definition 1.4
Let $\mu(\cdot)$ be a measure defined on the ring of sets $\mathfrak{S}$ with unit $\mathcal{X}$ such that $\mu(\mathcal{X}) < \infty$. We define functions $k_\cup, k_{\substack{\cap\\\cup}} : (\mathfrak{S} \setminus \{\emptyset\}) \times (\mathfrak{S} \setminus \{\emptyset\}) \to \mathbb{R}$ as

$$k_\cup(X_i, X_j) = \tfrac{1}{\mu(X_i \cup X_j)}; \tag{1.4}$$

$$k_{\substack{\cap\\\cup}}(X_i, X_j) = \tfrac{\mu(X_i \cap X_j)}{\mu(X_i \cup X_j)}; \tag{1.5}$$

The functions (1.4) and (1.5) are positive definite. The kernel function (1.5) is known as the *Tanimoto* or *Jaccard coefficient.* Its positive definiteness is shown in [Gower, 1971] and it has been used in kernel methods by Baldi and Ralaivola [2004].
In the remainder of this section we are more interested in the case that $\mathfrak{S}$ is a *Borel algebra* with unit $\mathcal{X}$ and measure μ which is countably additive and satisfies $\mu(\mathcal{X}) < \infty$. We then define a characteristic function of a set $X \subseteq \mathcal{X}$ by $\Gamma_X(x) = 1 \Leftrightarrow x \in X$ and $\Gamma_X(x) = 0$ otherwise. We can then write the intersection kernel as

$$k_\cap(X_i, X_j) = \mu(X \cap X') = \int_{\mathcal{X}} \Gamma_{X_i}(x)\Gamma_{X_j}(x) d\mu(x) \tag{1.6}$$

this shows the relation of the intersection kernel to the usual (L_2) inner product between the characteristic functions $\Gamma_X(\cdot), \Gamma_{X'}(\cdot)$ of the sets.
In the case that the sets X_i are finite or countable sets of elements on which a kernel has been defined, it is often benefitial to use set kernels other than the intersection kernel. For example, the following kernel function is also applicable in this case:

$$k_\times(X,X') = \int_X \int_{X'} k(x,x')d\mu(x')d\mu(x) = \int_{\mathcal{X}} \int_{\mathcal{X}} \Gamma_X(x)k(x,x')\Gamma_{X'}(x')d\mu(x')d\mu(x)$$

with any positive definite kernel k defined on the elements.
For knowledge representation often multisets are used instead of sets. The difference is that every element of a multiset is not required to be different from all other elements of that multiset.
For multisets, we can define a characteristic function $\Gamma_X : \mathcal{X} \to \mathbb{N}$ such that $\Gamma_X(x)$ is equal to the number of times x occurs in the multiset X. In this case we need to require that the multisets are finite in the sense that all characteristic functions have to be square integrable under the measure μ. This then immediatelly extends the above defined kernels for set also to multisets.

1.2.3 Kernels on Graphspaces

To apply kernel methods to the classification of graphs, it remains to define positive definte kernel functions on the set of all graphs or on application specific subsets of graphs. A typical application of this kind of kernels is the classification of chemical compounds, given their atom-bond structure. The strategy to define such kernel functions on graphs that has been followed mostly in literature is loosly speaking to decompose the graphs into possibly overlapping parts and then to apply one of the above described kernels on sets.
Let us first consider an intersection kernel with set cardinality as the measure and based on a decomposition that maps each graph into the set of all of its subgraphs. Using this kernel function, graphs satisfying certain properties can be identified. In particular, one could decide whether a graph has a Hamiltonian path, i.e., a sequence of adjacent vertices and edges that contains every vertex and edge exactly once. Now this problem is known to be *NP*-complete; therefore it is strongly believed that such kernels can not be computed in polynomial time. Furthermore, it can be shown that computing any graph kernel based on the intersection of injective decompositions is at least as hard as deciding graph isomorphism. We thus need to consider alternative, less expressive, graph kernels.
In literature different approaches have been tried to overcome this problem. [Graepel, 2002] restricted the decomposition to paths up to a given size, and [Deshpande et al., 2002] only consider the set of connected graphs that occur frequently as subgraphs in the graph database. The approach taken there to compute the decomposition of each graph is an iterative one
An alternative approach is based on measuring the number of walks in (directed or undirected) graphs with common label sequence. Although the set of common walks can be infinite, the inner product in this feature space can be computed in polynomial time by first building the product graph and then computing the limit of a matrix power series of the adjacency matrix [Gärtner et al., 2003]. An alternative walk-based kernel function exploits only the length of all walks between all pairs of

vertices with given label.
To illustrate the walk based kernel, consider a simple graph with four vertices $1, 2, 3, 4$ labelled 'c', 'a', 'r', and 't', respectively. We also have four edges in this graph: one from the vertex labelled 'c' to the vertex labelled 'a', one from 'a' to 'r', one from 'r' to 't', and one from 'a' to 't'. The non-zero features in the label pair feature space are $\phi_{c,c} = \phi_{a,a} = \phi_{r,r} = \phi_{t,t} = \lambda_0$, $\phi_{c,a} = \phi_{a,r} = \phi_{r,t} = \lambda_1$, $\phi_{a,t} = \lambda_1 + \lambda_2$, $\phi_{c,r} = \lambda_2$, and $\phi_{c,t} = \lambda_2 + \lambda_3$. The non-zero features in the label sequence feature space are $\phi_c = \phi_a = \phi_r = \phi_t = \sqrt{\lambda_0}$, $\phi_{ca} = \phi_{ar} = \phi_{at} = \phi_{rt} = \sqrt{\lambda_1}$, $\phi_{car} = \phi_{cat} = \sqrt{\lambda_2}$, and $\phi_{cart} = \sqrt{\lambda_3}$. The λ_i are user defined weights and the square-roots appear only to make the computation of the kernel more elegant. In particular, inner product in this feature space can be computed efficiently for undirected graphs and exponential or geometric choices of λ_i .
Although the walk based graph kernel described above can be computed efficiently, for large-scale applications to, e.g., chemical compound databases, exact computation might still not be feasible. There one can either resort to approximations in terms of short walks only, or consider different graph kernels specialised to this kind of databases. In particular on can consider graph kernels for the class of undirected graphs which contain few cycles only. For this class of graphs a kernel function with time complexity polynomial in the number of vertices and cycles in the graph can be proposed. For some real-world dataset of molecules, this kernel function can be computed much faster than the walk-based graph kernels described above.
The key idea of cyclic pattern kernels [Horváth et al., 2004] is to decompose every undirected graph into the set of cyclic and tree patterns in the graph. A cyclic pattern is a unique representation of the label sequence corresponding to a simple cycle in the graph. A tree pattern in the graph is a unique representation of the label sequence corresponding to a tree in the forest made up by the edges of the graph that do not belong to any cycle. The cyclic-pattern kernel between two graphs is defined by the cardinality of the intersection of the pattern sets associated with each graph.
Consider a graph with vertices $1, \ldots, 6$ and labels (in the order of vertices) 'c', 'a', 'r', 't', 'e', and 's'. Let the edges be the set

$$\{\{1,2\},\{2,3\},\{3,4\},\{2,4\},\{1,5\},\{1,6\}\}.$$

This graph has one simple cycle, and the lexicographically smallest representation of the labels along this cycle is the string 'art'. The bridges of the graph are $\{1,2\},\{1,5\},\{1,6\}$ and the bridges form a forest consisting of a single tree. The lexicographically smallest representation of the labels of this tree (in pre-order notation) is the string 'aces'.

1.2.4 Kernels on Vertices in a Graph

To apply kernel methods to the classification of vertices in a graph it remains to define positive definte kernel functions on the vertex set. A typical application for this kind of kernels is the classification of webpages in the World-Wide-Web, given the links between the pages. The strategy to define such a kernel function on graphs that has been followed mostly in literature is loosly speaking to compare the sets of vertices reachable from these vertices.

To properly define these kernel functions it is benefitial to first introduce a representation for various kinds of graphs and some functions on them. General graphs consist of a set of vertices $\mathcal{V}$, a set of edges $\mathcal{E}$, a function t from the set of edges to a set or tuple of vertices, and are denoted by $G = (\mathcal{V}, \mathcal{E}, t)$. For directed graphs the range of t is the set of pairs of vertices, for undirected graphs the range of t is the set of sets of two vertices, and for hypergraphs t is the powerset of the vertices. Corresponding to t we can define an operator $T : (\mathcal{E} \cup \mathcal{V} \to \mathbb{N}) \to (\mathcal{E} \cup \mathcal{V} \to \mathbb{N})$ that maps a multiset of vertices or edges to a multiset of edges or vertices that can be reached by one step on the graph.

In what follows we denote a multiset by $\{a, \ldots\}_{\mathbb{N}}$, identify multisets and their characteristic function, and use $A \cup B$ for two multisets A, B as a shorthand for the multiset with $\Gamma_{A\cup B}(\cdot) = \Gamma_A(\cdot) + \Gamma_B(\cdot)$. For undirected graphs and hypergraphs we define $T(\{v\}) = \{e \in \mathcal{E} : v \in t(e)\}$ for $v \in \mathcal{V}$, $T(\{e\}) = t(e)$ for $e \in \mathcal{E}$, and $T(A \cup B) = T(A) \cup T(B)$ for larger sets. For directed graphs we define $T(\{v\}) = \{e \in \mathcal{E} : t(e) = (v, u), u \in \mathcal{V}\}$ for $v \in \mathcal{V}$, $T(\{e\}) = \{u \in \mathcal{V} : t(e) = (v, u), v \in \mathcal{V}\}$ for $e \in \mathcal{E}$, and $T(A \cup B) = T(A) \cup T(B)$ for larger sets.

Now we can recursively define operators mapping a multiset of vertices or edges to a multiset of edges or vertices that can be reached by n steps on the graph: $T_0(A) = A, T_{n+1}(\cdot) = T(T_n(\cdot))$. Given a kernel κ on multisets, we can then define a kernel on the vertices of the graph as

$$k(u, v) = \lim_{n\to\infty} \sum_{i=0}^{n} \lambda_i \kappa\left(T_i(\{v\}), T_i(\{u\})\right)$$

where λ_i has to be chosen such that convergence is guaranteed. For finite graphs, a simpler expression can be obtained by assuming wlog that $\mathcal{E} \cup \mathcal{V} = \{1, \ldots, |\mathcal{V}| + |\mathcal{E}|\}$. We can then identifying each multiset by the vector of counts of the elements (The multiset A can then be represented by the vector $a \in \mathbb{N}^{|\mathcal{V}|+|\mathcal{E}|}, a_i = \Gamma_A(i)$), identifying the operator T with the corresponding matrix $T \in \mathbb{N}^{(|\mathcal{V}|+|\mathcal{E}|)\times(|\mathcal{V}|+|\mathcal{E}|)}$, and using the canonical inner product in $\mathbb{R}^{|\mathcal{V}|+|\mathcal{E}|}$ for κ. The simpler form of the above kernel then becomes:

$$k(u, v) = \lim_{n\to\infty} \sum_{i=0}^{n} \lambda_i \left\langle T^i \mathbf{e}_v, T^i \mathbf{e}_u \right\rangle$$

where $\mathbf{e}_u, \mathbf{e}_v$ are the u-th and v-th unit vectors, respectively. To use this kernel function in applications one can either resort to approximations with small n, or make use of a closed form computation of the limit for the case of undirected graphs or hypergraphs. We will next discuss some closed form solutions.

For undirected graphs or hypergraphs the matrix T^2 is symmetric and each entry $\left[T^2\right]_{uv} = |\{e \in \mathcal{E} : u, v \in t(e)\}|$ for all pairs of vertices $u, v \in \mathcal{V}$. Let E now be the restriction of T^2 to vertices. The kernel matrix can then be written as

$$K_E = \lim_{n\to\infty} \sum_{i=0}^{n} \lambda_i E^i \ .$$

Note that many kernels defined in literature use a different "base matrix" E rather than the above described T^2. Variants are to use the negative Laplacian of the graph or the normalised negative Laplacian. Let the $n \times n$ matrix D by defined by $D_{ii} = \sum_j E_{ij} = \sum_j E_{ji}$. The matrix T^2 (and its restriction to vertices), the Laplacian $L = D - E$, and the normalised Laplacian $\tilde{L} = D^{-1/2} L D^{-1/2}$ are positive definite by construction.
Let us now have a look at the eigendecomposition of the base matrix$E = UDU^{-1}$ where D is diagonal. Now observe that K_E can be written as

$$K_E = \lim_{n\to\infty} \sum_{i=0}^{n} \lambda_i U D^i U^{-1} = U \left(\lim_{n\to\infty} \sum_{i=0}^{n} D^i \right) U^{-1}$$

and that powers and limits of powers of diagonal matrices can be computed componentwise. Frequent choises of λ_i are $\frac{\beta^i}{i!}$ as $\lim_{n\to\infty} \sum_{i=0}^{n} \frac{\beta^i}{i!} a = e^{\beta a}$ or $\lambda_i = \gamma^i$ as $\lim_{n\to\infty} \sum_{i=0}^{n} \gamma^i a = \frac{1}{1-\gamma a}$. Feasible computation in the latter case is also possible by inverting the matrix $\mathbf{1} - \gamma E$. To see this, let $(\mathbf{1} - \gamma E)x = 0$, thus $\gamma E x = x$ and $(\gamma E)^i x = x$. Now, note that $(\gamma E)^i \to 0$ as $i \to \infty$. Therefore $x = 0$ and $\mathbf{1} - \gamma E$ is regular. Then $(\mathbf{1} - \gamma E)(\mathbf{1} + \gamma E + \gamma^2 E^2 + \cdots) = \mathbf{1}$ and $(\mathbf{1} - \gamma E)^{-1} = (\mathbf{1} + \gamma E + \gamma^2 E^2 + \cdots)$ is obvious. Using this parameterisation has thus the added advantage that the inverse of the covariance matrix indeed reflects the conditional independence structure implied by a Markovian interpretation of the graph.
Examples of such kernel functions from literature are the *diffusion kernel* [Kondor and Lafferty, 2002]

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

the *von Neumann kernel* [Kandola et al., 2003]

$$K = \sum_{i=1}^{\infty} \gamma^{i-1} [T^2]^i_{\mathcal{V}\mathcal{V}} \ ,$$

and the *regularised Laplacian kernel* [Smola and Kondor, 2003]

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

A general framework and analysis of such kernels can be found in [Smola and Kondor, 2003]. To obtain some further intuition about the regularised Laplacian kernel, consider using regularised least squares regression (1.2) with the kernel

$K = (\mathbf{1} + \gamma L)^{-1}$. Recall the optimisation problem

$$\min_{f(\cdot)\in\mathcal{H}} \frac{C}{m}\|y - Kc\|^2 + c^\top Kc$$

and substitute $\hat{y} = Kc$

$$\min_{f(\cdot)\in\mathcal{H}} \frac{C}{m}\|y - \hat{y}\|^2 + \hat{y}^\top \left(\mathbf{1} + \gamma L\right)\hat{y} \ .$$

An equivalent formulation of this optimisation problem is

$$\min_{f(\cdot)\in\mathcal{H}} \frac{C}{m}\|y - \hat{y}\|^2 + \gamma\|\hat{y}\|^2 + \gamma \sum_{(u,v)=t(e),e\in\mathcal{E}} (\hat{y}_u - \hat{y}_v)^2 \ .$$

This shows that the regularised Laplacian kernel biases the predictions $\hat{y}$ such that connected vertices are likely to have the same label.

1.2.5 From Kernels on Vertices in a Graph to Transduction

While kernel functions between graphs (Section 1.2.3) can be used pretty directly with most available kernel methods, kernels for vertices (Section 1.2.4) 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.
Efficiency Issues Now consider for a moment the kernel to be partitioned according to the labelled/unlabelled split,i.e.

$$K = \begin{pmatrix} K_{\mathrm{ll}} & K_{\mathrm{lu}} \\ K_{\mathrm{ul}} & K_{\mathrm{uu}} \end{pmatrix} \ .$$

As described above, K is usually defined as the limit of some matrix power series of the adjacency matrix or the (normalised) graph Laplacian and can be computed by matrix exponentiation or inversion. In any case even when the graph is sparse, the kernel matrix rarely is. For inductive algorithms we could then use the reformulation in terms of the inverse of K_{ll}. Again even when the graph is sparse, the inverse of K_{ll} is unlikely to be sparse and usually expensive to obtain.
For transduction, however, the inverse of K can be used which is just the sum of the identity matrix and a multiple of the Laplacian. So for Gaussian Processes it might be computationally advantageous to perform transduction rather than induction.
Relation to the Cluster Assumption An assumption underlying most current transductive and semi-supervised approaches is the so-called "cluster assumption": "The decision boundary should not cross high-density regions" [Chapelle and Zien, 2005, e.g.]. From the above illustration of graph kernels we can directly see the relation between graph kernels and this cluster assumption. If we simply define

high-density regions to consist of all of those pairs of vertices that are connected by a lot of walks of short length, it becomes obvious that the correlation of these vertices, i.e., the value of the kernel function for these vertices, will be high. Every reasonable learning algorithm will then try to avoid classifying highly correlated vertices differently.

1.3 Gaussian Processes Induction

Supervised learning is one of the most commonly considered data mining scenarios. The *supervised learning problem* — which we will concentrate on — 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, 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.*

The usual supervised learning setting considers a set $\mathcal{X}$ of instances and a set $\mathcal{Y}$ of labels. The relation between instances and labels is assumed to be a fixed but unknown *probability measure* $p(\cdot, \cdot)$ on the set $\mathcal{X} \times \mathcal{Y}$. In other words, one assumes conditional dependence of labels on individuals only. Let now (X, Y) denote the training instances with their labels $\{(x_i, y_i)\}_{i=1}^m$.

The *non-probabilistic inductive* learning task is then — given a set of individuals with associated labels (X, Y) (observed according to $p(y_i, x_i) = p(y_i|x_i)p(x_i)$) — to find a function that estimates the label of instances drawn from $\mathcal{X}$.

The *probabilistic inductive* learning task is then — given a set of individuals with associated labels (X, Y) (observed according to $p(y_i, x_i) = p(y_i|x_i)p(x_i)$) — to estimate $p(y|x, X, Y)$.

1.3.1 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 respect to parameters θ.

Exponential Family Densities 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

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

is called the *log-partition function*, $\phi : \mathcal{X} \times \mathcal{Y} \to \mathcal{H}$ maps every pair (x, y) to its joint *sufficient statistics*, $\langle\cdot,\cdot\rangle$ denotes the inner product, and $\theta \in \mathcal{H}$ are parameters (here, random variables). It holds then that

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

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

Conditionally Exponential Family Densities From the joint exponential densities above, we can derive the conditional exponential densities as

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

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

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

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

1.3.2 Bayesian Estimation

To estimate the label y of a new test point x from data $(X, Y) = \{(x_1, y_1), \ldots, (x_m, y_m)\}$ under the assumption of a parameterised family of distributions like the exponential we need to compute

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

In order to avoid the integral over θ we can alternatively use

$$p(y|x,\theta^*) \quad \theta^* = \arg\max_{\theta} p(\theta|X,Y) \ .$$

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 maximise the posterior $p(\theta|X,Y)$ by maximising the likelihood $p(X,Y|\theta)$ times the prior $p(\theta)$ or – equivalently – minimise the negative log-posterior

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

where $c' = \log p(X,Y) - \log(X|\theta) = \log p(X,Y) - \log(X) = p(Y|X)$ is independent of θ and thus constant in the optimisation problem that can be ignored.
The representer theorem [Altun et al., 2004, e.g] shows that the minimising θ of the above negative log-posterior has the form

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

whenever the prior is such that $\forall j \in \{1,\dots m\}, y \in \mathcal{Y} \ : \ \Phi \perp \phi(x_j,y)$ implies

$$p(\theta) \geq p(\theta + \Phi) \ .$$

This is for instance the case for Gaussian priors. We thus obtain the objective function

$$\begin{aligned}
-\log p(\theta|X,Y) = &\sum_{i=1}^{m} g(\theta|x_i) - \sum_{i,j=1}^{m} \int_{\mathcal{Y}} \alpha_{jy} k\left((x_i,y_i),(x_j,y)\right) dy \\
&- \frac{1}{2\sigma^2}\sum_{i,j=1}^{m} \int_{\mathcal{Y}}\int_{\mathcal{Y}} \alpha_{jy}\alpha_{iy'} k\left((x_i,y'),(x_j,y)\right) dydy' + c''
\end{aligned} \qquad (1.7)$$

where c'' is independent of θ and

$$g(\theta|x) = \log \int_{\mathcal{Y}} \exp\left[\sum_{j=1}^{m}\int_{\mathcal{Y}} \alpha_{jy'} k\left((x,y),(x_j,y')\right) dy'\right] dy \ .$$

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 $k((x,y),(x',y')) = k_{\mathcal{X}}(x,x')k_{\mathcal{Y}}(y,y')$.

1.3.3 Multiclass Gaussian Processes

For multiclass classification we have a finite label set and wlog we can assume $\mathcal{Y} = \{1, \ldots, n\}$. Together with the normal prior $\theta \sim \mathcal{N}(0, \sigma^2 \mathbf{1})$ we obtain a Gaussian process classifier where the Gaussian process is on $u : u_{(x,y)} = \langle \phi(x, y), \theta \rangle$ where for the restriction $\mathbf{u}$ of u to the training instances X it holds that $\mathbf{u} \sim \mathcal{N}(0, \sigma^2 K)$ with $K_{(x_i,y),(x_j,y')} = k((x_i, y), (x_j, y'))$.
To see this we assume a Gaussian process multiclass classifier

$$p(y|x, u)p(u) \propto \exp\left[u_{(x,y)} - \log \sum_{y'} \exp[u_{(x,y')}]\right] \exp\left[-\frac{1}{2}\mathbf{u}^\top \Sigma^{-1} \mathbf{u}\right]$$

and relate this to the exponential family model

$$p(y|x, \theta)p(\theta) \propto \exp\left[\langle \phi(x, y), \theta \rangle - \log \sum_{y'} \exp\left[\langle \phi(x, y'), \theta \rangle\right]\right] \exp\left[-\frac{1}{2\sigma^2}\|\theta\|^2\right] .$$

A short computation then shows

$$\sigma^2 K = \Sigma .$$

With

$$\mathbf{y}_{jy} = \begin{cases} 1 & \text{if } y_j = y \\ 0 & \text{otherwise} \end{cases}$$

and assuming $k_{\mathcal{Y}}(y, y') = \delta_{y,y'}$ we can write (1.7) as

$$-\log p(\theta|X, Y) = \sum_{i=1}^{m} \log \sum_{y=1}^{n} \exp\left([K\boldsymbol{\alpha}]_{iy}\right) - \operatorname{tr} \mathbf{y}^\top K \boldsymbol{\alpha} + \frac{1}{2\sigma^2} \operatorname{tr} \boldsymbol{\alpha}^\top K \boldsymbol{\alpha} + c'' . \tag{1.8}$$

Equivalently we can expand (1.8) in terms of $t = K\boldsymbol{\alpha}$ as

$$-\log p(\theta|X, Y) = \sum_{i=1}^{m} \log \sum_{y=1}^{n} \exp\left([t]_{iy}\right) - \operatorname{tr} \mathbf{y}^\top t + \frac{1}{2\sigma^2} \operatorname{tr} t^\top K^{-1} t + c'' . \tag{1.9}$$

This is useful for the case that the inverse kernel is easier to obtain and has less non-zero entries than the kernel matrix itself.
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

$$\partial_\alpha \mathcal{P} = K(\pi - \mu + \sigma^{-2}\alpha) \tag{1.10a}$$
$$\partial_t \mathcal{P} = \pi - \mu + \sigma^{-2} K^{-1} t. \tag{1.10b}$$

To avoid spelling out tensors of fourth order for the second derivatives (since

$\alpha \in \mathbb{R}^{m\times n}$) we state the action of the latter as bilinear forms on vectors $\beta, \gamma, u, v \in \mathbb{R}^{m\times n}$:

$$\partial_\alpha^2 \mathcal{P}[\beta,\gamma] = \operatorname{tr}(K\gamma)^\top(\pi.*(K\beta)) - \operatorname{tr}(\pi.*K\gamma)^\top(\pi.*(K\beta)) + \sigma^{-2}\operatorname{tr}\gamma^\top K\beta \tag{1.11a}$$

$$\partial_t^2 \mathcal{P}[u,v] = \operatorname{tr} u^\top(\pi.*v) - \operatorname{tr}(\pi.*u)^\top(\pi.*v) + \sigma^{-2}\operatorname{tr} u^\top K^{-1} v. \tag{1.11b}$$

We used the "Matlab" notation of '.$*$' to denote element-wise multiplication of matrices.
Let $L \cdot n$ be the computation time required to compute $K\alpha$ and $K^{-1}t$ respectively. One may check that $L = O(m)$ implies that each conjugate gradient (CG) descent step can be performed in $O(m)$ time. Combining this with rates of convergence for Newton-type or nonlinear CG solver strategies yields overall time costs in the order of $O(m \log m)$ to $O(m^2)$ worst case, a significant improvement over conventional $O(m^3)$ methods.

1.4 Balanced Gaussian Process Transduction

1.4.1 Transduction

For transduction the labels Y decompose into $Y_{\text{train}} \cup Y_{\text{test}}$ and the instances X decompose into $X_{\text{train}} \cup X_{\text{test}}$.
The *probabilistic transductive* learning task is then — given a set of individuals with associated labels $(X_{\text{train}}, Y_{\text{train}})$ (observed according to $p(y,x) = p(y \mid x)p(x)$) and a set of individuals X_{test} — to estimate $p(Y_{\text{test}} \mid X, Y_{\text{train}})$.
Balancing Constraints To achieve better predictive accuracy of our transductive approach, we impose a balancing constraint on the considered probability distributions rather than just integrating out θ or maximising the joint probability. In particular we would like the class marginals of the distributions over Y_{test} to (approximately) match the observed class frequencies in Y_{train}. Let $\mathcal{M}$ be the set of probability distributions that (approximately) match the observed class frequencies. The problem of finding the MAP estimate of θ subject to the balancing constraint becomes:

$$\begin{aligned} \min\nolimits_\theta \quad & -\log p(\theta \mid X, Y_{\text{train}}) \\ \text{s.t.} \quad & p(Y_{\text{test}} \mid X, Y_{\text{train}}, \theta) \in \mathcal{M} \end{aligned}$$

which we can rewrite for the case of exact match as

$$\begin{aligned} \min\nolimits_\theta \quad & -\log p(\theta \mid X, Y_{\text{train}}) \\ \text{s.t.} \quad & \mathbf{E}_{Y_{\text{test}} \sim p(Y_{\text{test}} \mid X, Y_{\text{train}}, \theta)}\left[\psi(Y_{\text{test}})\right] = \mu \end{aligned}$$

where ψ maps Y_{test} to a vector of class counts and μ is the corresponding vector of class counts in the training data.

Variational Transduction Recall Entropy $H(q) = -\int q(x) \log q(x) dx$ and KL-divergence $D(q\|p) = \int q(x) \log \frac{q(x)}{p(x)} dx$. Now consider the following optimisation problem:

$$\min_{q,\theta} -\log p(\theta \mid X, Y_{\text{train}}) + D(q(Y_{\text{test}}) \| p(Y_{\text{test}} \mid X, Y_{\text{train}}, \theta)) \; .$$

If we put no restrictions on the choice of q, q will simply become equal to $p(Y_{\text{test}} \mid X, Y_{\text{train}}, \theta)$ and θ will be the maximum a posteriori (MAP) parameters. Once we constrain q, the objective function $D(q(Y_{\text{test}}) \| p(Y_{\text{test}}, \theta \mid X, Y_{\text{train}}))$ is still an upper bound on $p(\theta \mid X, Y_{\text{train}})$ and we optimise a trade-off between $-\log p(\theta \mid X, Y_{\text{train}})$ and the divergence from the nearest distribution that obeys the balancing constraints. A short calculation

$$\begin{aligned}
& -\log p(\theta \mid X, Y_{\text{train}}) && (1.12)\\
\leq & -\log p(\theta \mid X, Y_{\text{train}}) + D(q(Y_{\text{test}}) \| p(Y_{\text{test}} \mid X, Y_{\text{train}}, \theta)) && (1.13)\\
= & -\log p(\theta \mid X, Y_{\text{train}}) + \sum_{Y_{\text{test}}} q(Y_{\text{test}}) \log q(Y_{\text{test}}) \\
& - \sum_{Y_{\text{test}}} q(Y_{\text{test}}) \log p(Y_{\text{test}} \mid X, Y_{\text{train}}, \theta) \\
= & -H(q) - \sum_{Y_{\text{test}}} q(Y_{\text{test}}) \log p(Y_{\text{test}}, \theta \mid X, Y_{\text{train}}) \; . && (1.14)
\end{aligned}$$

provides alternative formulations of the upper bound that we will use below to simplify the optimisation. Indeed we would like to minimise (1.12) over θ subject to the balancing constraints on $p(Y_{\text{test}} \mid X, Y_{\text{train}}, \theta)$. To simplify this problem instead we iteratively minimise (1.13) over q subject to the balancing constraints on q and minimise (1.14) over θ.

Decomposing the Variational Bound To simplify the optimisation problem, note that the second part of (1.14) can be written as

$$\begin{aligned}
& -\sum_{Y_{\text{test}}} q(Y_{\text{test}}) \log p(Y_{\text{test}}, \theta \mid X, Y_{\text{train}}) \\
= & -\sum_{Y_{\text{test}}} q(Y_{\text{test}}) \log p(Y_{\text{test}}, Y_{\text{train}}, \theta \mid X) + \sum_{Y_{\text{test}}} q(Y_{\text{test}}) \log p(Y_{\text{train}} \mid X) \\
= & -\sum_{Y_{\text{test}}} q(Y_{\text{test}}) \log p(Y_{\text{test}}, Y_{\text{train}}, \theta \mid X) + \log p(Y_{\text{train}} \mid X) \; .
\end{aligned}$$

Here $-\log p(Y_{\text{test}}, Y_{\text{train}}, \theta \mid X)$ is the joint likelihood of θ and Y that we already looked at above and $\log p(Y_{\text{train}} \mid X)$ is independent of θ and Y_{test}. We can write this in terms of expectations as

$$\begin{aligned}
&-\sum_{Y_{\text{test}}} q(Y_{\text{test}}) \log p(Y_{\text{test}}, Y_{\text{train}}, \theta \mid X) && (1.15)\\
=&\mathbf{E}_{Y_{\text{test}}\sim q}\left[\sum_{i=1}^{m} \log \sum_{y=1}^{n} \exp\left([K\boldsymbol{\alpha}]_{iy}\right) - \operatorname{tr} \mathbf{y}^\top K\boldsymbol{\alpha} + \frac{1}{2\sigma^2} \operatorname{tr} \boldsymbol{\alpha}^\top K \boldsymbol{\alpha}\right]\\
=&\sum_{i=1}^{m} \log \sum_{y=1}^{n} \exp\left([K\boldsymbol{\alpha}]_{iy}\right) - \mathbf{E}_{Y_{\text{test}}\sim q} \operatorname{tr} \mathbf{y}^\top K\boldsymbol{\alpha} + \frac{1}{2\sigma^2} \operatorname{tr} \boldsymbol{\alpha}^\top K \boldsymbol{\alpha}\\
=&\sum_{i=1}^{m} \log \sum_{y=1}^{n} \exp\left([K\boldsymbol{\alpha}]_{iy}\right) - \operatorname{tr} \nu(q)^\top K\boldsymbol{\alpha} + \frac{1}{2\sigma^2} \operatorname{tr} \boldsymbol{\alpha}^\top K \boldsymbol{\alpha} && (1.16)
\end{aligned}$$

where $\nu(q) = \mathbf{E}_{Y_{\text{test}}\sim q}[\mathbf{y}]$ or simply $[\nu(q)]_{iy} = \delta_{y_i,y}$ for training instances and $[\nu(q)]_{iy} = \mathbf{q}_{iy} = q(y_i = y)$.
So the two stages of the iterative procedure are

- Given q find θ that minimises $E_{Y_{\text{test}}\sim q} - \log p(Y, \theta \mid X)$.
- Given θ find q that respects the balancing constraints and minimises $D(q(Y_{\text{test}}) \| p(Y_{\text{test}} \mid X, Y_{\text{train}}, \theta))$.

both steps minimise the same upper bound on $-\log p(\theta \mid X, Y_{\text{train}})$ so the procedure will converge to a (local) optimum.
Inverse Formulation As above we can expand (1.15) in terms of $t = K\boldsymbol{\alpha}$ as

$$\sum_{i=1}^{m} \log \sum_{y=1}^{n} \exp\left([t]_{iy}\right) - \operatorname{tr} \nu(q)^\top t + \frac{1}{2\sigma^2} \operatorname{tr} t^\top K^{-1} t\ .$$

1.4.2 Optimising wrt the Balancing Constraints

We would now like to solve

$$\begin{array}{ll} \min\limits_{q} & D(q(Y_{\text{test}}) \| p(Y_{\text{test}} \mid X, Y_{\text{train}}, \theta)) \\ \text{s.t.} & \mathbf{E}_{Y_{\text{test}}\sim q}[\psi(Y_{\text{test}})] = \mu \end{array} . \qquad (1.17)$$

However, quite often we are not really sure that the test data has exactly the same class distribution than the training data. In such cases we do not want to strictly enforce the balancing constraints but instead only enforce them approximately. The problem with strict balancing is illustrated in Figure 1.1 for some toy problems. An alternative to (1.4.2) is to introduce slack variables ξ and solve

$$\begin{array}{ll} \min\limits_{q,\xi} & D(q(x)\|p(x)) + \frac{1}{2\beta}\|\xi\|^2 \\ \text{s.t.} & \mathbf{E}_{x\sim q}\psi(x) = c + \xi \\ & \int dq(x) = 1 \\ & q(x) \geq 0\ . \end{array} \qquad (1.18)$$

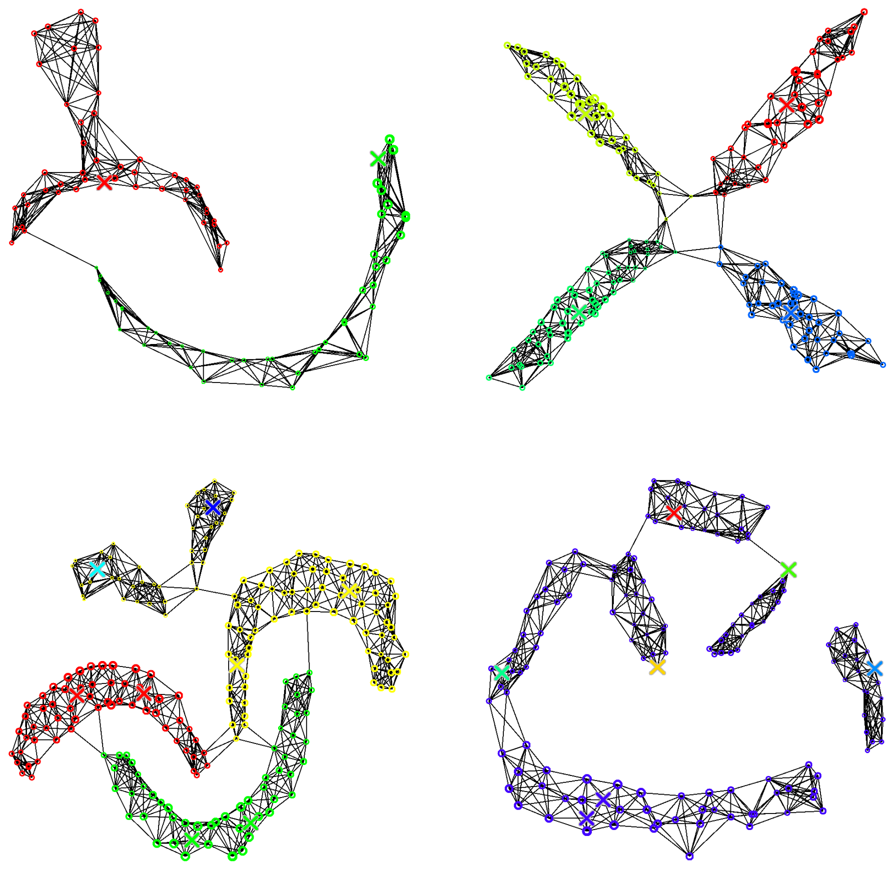

Figure 1.1 Strict Balancing on different toy datasets. Crosses indicate labelled examples, circles indicate unlabelled examples. The colour of the circles indicates the predicted class, the size and thickness of the circles indicate the probability of the predicted class.

For that, we need the following theorem

Theorem 1.5
The problem of finding a probability distribution q which

$$\begin{aligned} \min_{q,\xi} \quad & D(q(x)\|p(x)) + \tfrac{1}{2\beta}\|\xi\|^2 \\ \text{s.t.} \quad & \mathbf{E}_{x\sim q}\psi(x) = c + \xi \end{aligned}$$

has as solution

$$q(x) = p(x)\exp\left(\langle\psi(x), \Theta\rangle - g(\Theta)\right)$$

where $\partial_\Theta g(\Theta) = c$ and Θ can be found as

$$\min_\Theta g(\Theta) - \langle \Theta, c \rangle + \frac{\beta}{2} \|\Theta\|^2 \ .$$

Proof

$$\begin{aligned} \min_{q,\xi} \quad & \textstyle\int [\log q(x) - \log p(x)]\, dq(x) + \frac{1}{2\beta} \|\xi\|^2 \\ \text{s.t.} \quad & \textstyle\int \psi(x) dq(x) = c + \xi \\ & \textstyle\int dq(x) = 1 \\ & q(x) \geq 0 \end{aligned}$$

has Lagrange function

$$\mathcal{L} = \int dq(x) \left[\log q(x) - \log p(x) - \nu_x - \lambda - \Theta^\top (\psi(x) - c - \xi) \right] + \lambda + \frac{1}{2\beta} \|\xi\|^2$$

where $\nu_x \geq 0$, $\lambda \in \mathbb{R}$ and $\Theta \in \mathbb{R}^n$ are Lagrange multipliers. We now need to find the saddle point of $\mathcal{L}$ that is minimum with respect to q, ξ and maximum with respect to the Lagranian multipliers. Differentiating with respect to ξ and noting that q is constrained to valid probability distributions, we obtain

$$\partial_\xi \mathcal{L} = \Theta + \frac{1}{\beta} \xi = 0 \ \Rightarrow \ \xi = -\beta \Theta \ .$$

Plugging this in and differentiating now with respect to q we must have

$$\partial_q \mathcal{L} = \int dx \left[\log q(x) + 1 - \log p(x) - \nu_x - \lambda - \Theta^\top (\psi(x) - c + \beta\Theta) \right] = 0$$

for all $\nu_x \geq 0$, $\lambda \in \mathbb{R}$ and $\Theta \in \mathbb{R}^n$. Thus the solution has the form

$$q(x) = p(x) \exp\left[\langle \psi(x), \Theta \rangle - g(\Theta) \right]$$

where $g(\Theta)$ collects all terms independent of x. As q has to be a probability distribution, we know that $g(\Theta)$ can be written as

$$g(\Theta) = \log \int p(x) \exp\left(\Theta^\top \psi(x) \right) dx \ .$$

If we plug the form of the solution into the objective function we get

$$\mathcal{L} = \int dq(x) \left[-g(\Theta) - \nu_x - \lambda + \Theta^\top c - \beta \|\Theta\|^2 \right] + \lambda + \frac{\beta}{2} \|\Theta\|^2 = -g(\Theta) + \Theta^\top c - \frac{\beta}{2} \|\Theta\|^2$$

as $q(x)$ is by the form of the solution guaranteed to be a valid probability distribution. ∎

Multiclass Solution Recall our minimisation problem (1.4.2). Due to the above theorem it has the solution

$$q(Y_{\text{test}}) = p(Y_{\text{test}} \mid X, Y_{\text{train}}, \theta) \exp[\langle \psi(Y_{\text{test}}), \Theta \rangle - g(\Theta)]$$

where

$$g(\Theta) = \log \mathbf{E}_{Y_{\text{test}} \sim p(Y_{\text{test}}|X, Y_{\text{train}}, \theta)} \exp[\langle \psi(Y_{\text{test}}), \Theta \rangle] = \mu \ .$$

As Y_{test} decomposes into different instances so do $q(Y_{\text{test}})$ and $g(\Theta)$:

$$q(Y_{\text{test}}) = \prod_i q(y_i); \quad g(\Theta) = \sum_i g_i(\Theta) \ .$$

With $\pi_{ij} = p(y_i = j \mid x_i, X_{\text{train}}, Y_{\text{train}}, \theta)$ we then have

$$q(y_i = j) = \pi_{ij} \exp\left[\langle \psi(j), \Theta \rangle - g_i(\Theta)\right] = \pi_{ij} \exp\left[\Theta_j - g_i(\Theta)\right]$$

and

$$g_i(\Theta) = \log \sum_j \pi_{ij} \exp(\Theta_j) \ .$$

As the form of the solution (above) already guarantees that $q(y_i = j)$ is a valid probability distribution and that it is optimal, it remains to make sure that the moments really match. Let $\mathbf{q}$ denote the matrix with $\mathbf{q}_{ij} = q(y_i = j)$.
Thus for approximate balancing it is sufficient to solve

$$\min_\Theta \left[\frac{1}{m} \sum_{i=1}^m g_i(\Theta)\right] - \langle \Theta, \mu \rangle + \frac{\beta}{2} \|\Theta\|^2$$

with

$$\partial_\Theta(\cdot) = \frac{1}{m} \sum_{i=1}^m \mathbf{E}_{j \sim q(y_i)}[\psi(j)] - \mu + \beta\Theta = \frac{1}{m} \sum_{i=1}^m \mathbf{q}_{i\cdot}^\top - \mu + \beta\Theta$$

and

$$\partial_\Theta^2(\cdot) = \frac{1}{m} \sum_{i=1}^m \left[\text{diag}[\mathbf{q}_{i\cdot}] - \mathbf{q}_{i\cdot}^\top \mathbf{q}_{i\cdot}\right] + \beta \mathbf{1} \ .$$

Approximate balancing is illustrated in Figure 1.2 for some toy problems.

1.4.3 Related Work

String Kernels: Efficient computation of string kernels using suffix trees was described in [Vishwanathan and Smola, 2004]. In particular, it was observed that expansions of the form $\sum_{i=1}^m \alpha_i k(x_i, x)$ 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

$$k(x, x') = \sum_s w_s \#_s(x) \#_s(x') \tag{1.19}$$

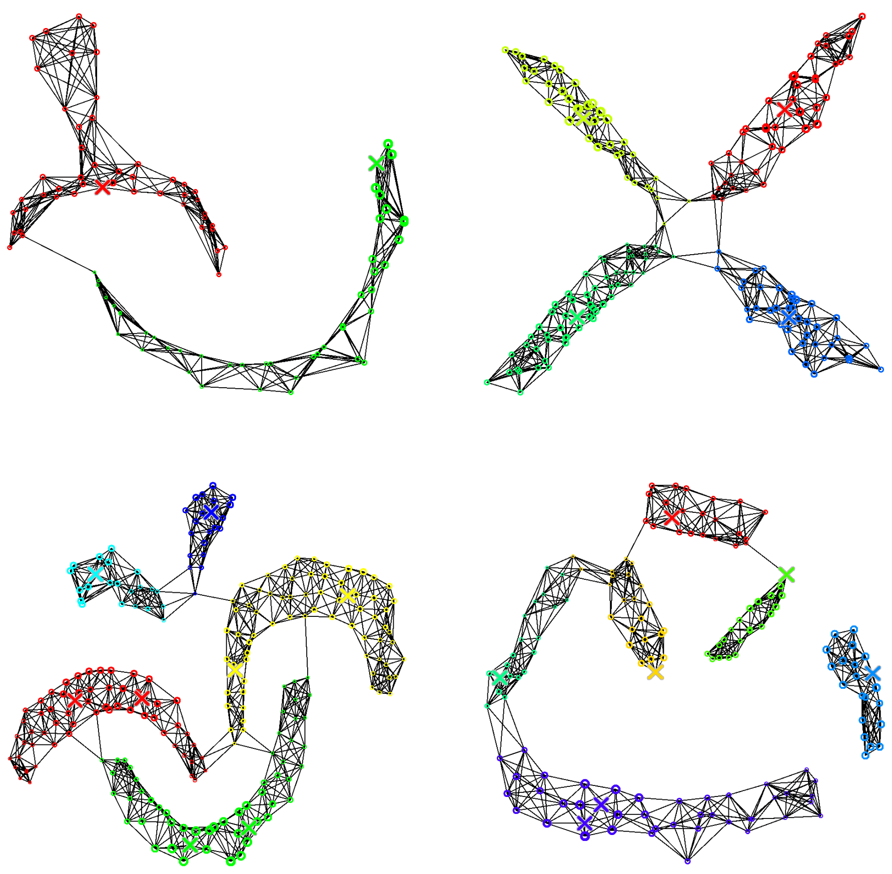

Figure 1.2 Approximate Balancing on different toy datasets.

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 $d \ll m$, it is possible to carry out matrix vector multiplications in $O(md)$ 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 [Bennett, 1998] or EM-style iterations for binary classification [Joachims, 2002]. Moreover, on graphs various methods for unsupervised learning have been proposed [Zhu et al., 2003, Zhou et al., 2005], 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 [Vapnik, 1998]. Others again exploit the fact that training and test set have similar marginal distributions [Joachims, 2002].

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.

1.5 Empirical results

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

Table 1.1 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 crossvalidations, i.e., 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 haven't 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 [Zhou et al., 2005]. Although a directed graph approach outperforms there an undirected approach, we resorted to kernels for undirected graphs, as those are computationally more attractive. We will investigate computationally attractive digraph kernels in future

0. In [Bennett, 1998] only subsets of USPS were considered due to the size of this problem.

Table 1.1 Results on WebKB for ‘inverse’ 10-fold crossvalidation

DATASET	$\lvert V\rvert$	$\lvert E\rvert$	ERROR	DATASET	$\lvert V\rvert$	$\lvert E\rvert$	ERROR
Cornell	867	1793	10%	Misc	4113	4462	78%
Texas	827	1683	7%	all	8275	14370	50%
Washington	1205	2368	12%	Universities	4162	9591	18%
Wisconsin	1263	3678	28%				

work and expect similar benefits as reported by [Zhou et al., 2005]. Though we are using more training data than [Zhou et al., 2005] we are also considering a more difficult learning problem (multiclass without removing various instances).

1.6 Conclusions and Future Work

This chapter gave a short overview of kernel methods and how kernel methods can be applied to data repesented by a graph.
Current Kernel Methods for graphs do not scale very well with the available amount of unlabelled data. It turns out that for certain graph kernels it is more efficient to perform transduction than induction. We thus presented 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 which allow fast matrix vector multiplication.
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.

Acknowledgements 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

Y. Altun, T. Hofmann, and A.J. Smola. Gaussian process classification for segmenting and annotating sequences. In *Proceedings of the 21th International Conference on Machine Learning*, 2004.

P. Baldi and L. Ralaivola. Graph kernels for molecular classification and prediction of mutagenicity, toxicity, and anti-cancer activity. Presentated at the Computational Biology Workshop of NIPS, 2004.

Kristin Bennett. Combining support vector and mathematical programming methods for classification. In *Advances in Kernel Methods - -Support Vector Learning*, pages 307 – 326. MIT Press, 1998.

B. E. Boser, I. M. Guyon, and V. N. Vapnik. A training algorithm for optimal margin classifiers. In David Haussler, editor, *Proceedings of the 5th Annual ACM Workshop on Computational Learning Theory*, pages 144–152. ACM Press, July 1992. ISBN 0-89791-498-8.

O. Chapelle and A. Zien. Semi-supervised classification by low density separation. In *Tenth International Workshop on Artificial Intelligence and Statistics*, 2005.

M. Deshpande, M. Kuramochi, and G. Karypis. Automated approaches for classifying structures. In *Proceedings of the 2nd ACM SIGKDD Workshop on Data Mining in Bioinformatics*, 2002.

T. Gärtner. A survey of kernels for structured data. *SIGKDD Explorations*, 2003.

T. Gärtner. Predictive graph mining with kernel methods. In *Advanced Methods for Knowledge Discovery from Complex Data*. Springer-Verlag, 2005. To appear.

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.

T. Gärtner, T. Horváth, Q.V. Le, A.J. Smola, and S. Wrobel. Kernel methods for graphs. John Wiley and Sons, 2006. To appear.

T. Gärtner, Q.V. Le, S. Burton, A.J. Smola, and SVN. Vishwanathan. Large-scale multiclass transduction. In *Advances in Neural Information Processing Systems 18*, 2006. To appear.

J. C. Gower. A general coefficient of similarity and some of its properties. *Biometrics*, 1971.

T. Graepel. *PAC-Bayesian Pattern Classification with Kernels.* PhD thesis, TU

Berlin, 2002.

T. Horváth, T. Gärtner, and S. Wrobel. Cyclic pattern kernels for predictive graph mining. In *Proc. of the 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 158–167, 2004.

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 Publishers, Boston, May 2002. ISBN 0 - 7923 - 7679-X.

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

R. I. Kondor and J. Lafferty. Diffusion kernels on graphs and other discrete input spaces. In C. Sammut and A. Hoffmann, editors, *Proceedings of the 19th International Conference on Machine Learning*, pages 315–322. Morgan Kaufmann, 2002.

R. M. Rifkin. *Everything Old is new again: A fresh Look at Historical Approaches to Machine Learning.* PhD thesis, MIT, 2002.

C. Saunders, A. Gammerman, and V. Vovk. Ridge regression learning algorithm in dual variables. In *Proceedings of the Fifteenth International Conference on Machine Learning*. Morgan Kaufmann, 1998.

B. Schölkopf, R. Herbrich, and A. J. Smola. A generalized representer theorem. In *Proceedings of the 14th annual conference on learning theory*, 2001.

B. Schölkopf and A. J. Smola. *Learning with Kernels.* MIT Press, 2002.

A. J. Smola and R. Kondor. Kernels and regularization on graphs. In *Proceedings of the 16th Annual Conference on Computational Learning Theory and the 7th Kernel Workshop*, 2003.

V. Vapnik. *The Nature of Statistical Learning Theory.* Springer-Verlag, 1995.

V. Vapnik. *Statistical Learning Theory.* John Wiley and Sons, New York, 1998.

S. V. N. Vishwanathan and A. J. Smola. Fast kernels for string and tree matching. In K. Tsuda, B. Schölkopf, and J.P. Vert, editors, *Kernels and Bioinformatics*, Cambridge, MA, 2004. MIT Press. URL `http://users.rsise.anu.edu.au/ vishy/papers/VisSmo04.pdf`.

G. Wahba. *Spline Models for Observational Data*, volume 59 of *CBMS-NSF Regional Conference Series in Applied Mathematics*. SIAM, Philadelphia, 1990.

D. Zhou, J. Huang, and B. Schölkopf. Learning from labeled and unlabeled data on a directed graph. In *International Conference on Machine Learning*, 2005.

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.