Journal of Machine Learning Research 11 (2010) 555-580 Submitted 12/08; Revised 9/09; Published 2/10

Approximate Tree Kernels

Konrad Rieck RIECK@CS.TU-BERLIN.DE
Technische Universität Berlin
Franklinstraße 28/29
10587 Berlin, Germany

Tammo Krueger TAMMO.KRUEGER@FIRST.FRAUNHOFER.DE
Fraunhofer Institute FIRST
Kekuléstraße 7
12489 Berlin, Germany

Ulf Brefeld BREFELD@YAHOO-INC.COM
Yahoo! Research
Avinguda Diagonal 177
08018 Barcelona, Spain

Klaus-Robert Müller KLAUS-ROBERT.MUELLER@TU-BERLIN.DE
Technische Universität Berlin
Franklinstraße 28/29
10587 Berlin, Germany

Editor: John Shawe-Taylor

Abstract

Convolution kernels for trees provide simple means for learning with tree-structured data. The computation time of tree kernels is quadratic in the size of the trees, since all pairs of nodes need to be compared. Thus, large parse trees, obtained from HTML documents or structured network data, render convolution kernels inapplicable. In this article, we propose an effective approximation technique for parse tree kernels. The approximate tree kernels (ATKs) limit kernel computation to a sparse subset of relevant subtrees and discard redundant structures, such that training and testing of kernel-based learning methods are significantly accelerated. We devise linear programming approaches for identifying such subsets for supervised and unsupervised learning tasks, respectively. Empirically, the approximate tree kernels attain run-time improvements up to three orders of magnitude while preserving the predictive accuracy of regular tree kernels. For unsupervised tasks, the approximate tree kernels even lead to more accurate predictions by identifying relevant dimensions in feature space.

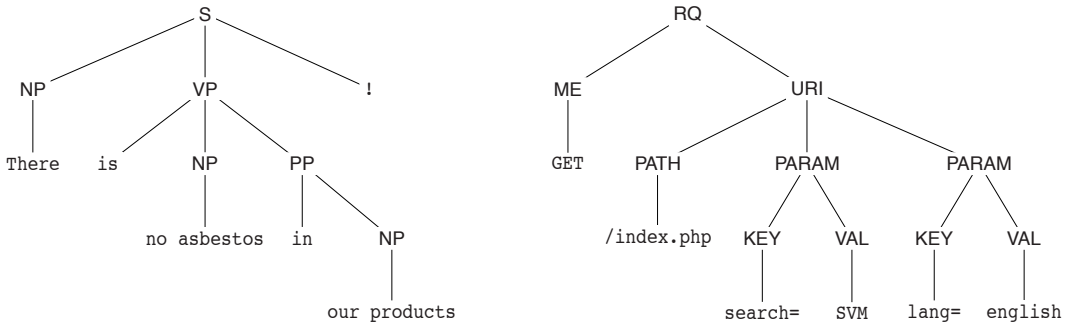
Keywords: tree kernels, approximation, kernel methods, convolution kernels

1. Introduction

Learning from tree-structured data is an elementary problem in machine learning, as trees arise naturally in many real-world applications. Exemplary applications involve parse trees in natural language processing, HTML documents in information retrieval, molecule structures in computational chemistry, and structured network data in computer security (e.g., Manning and Schütze, 1999; Kashima and Koyanagi, 2002; Moschitti, 2006b; Cilia and Moschitti, 2007; Düssel et al., 2008; Rieck et al., 2008; Bockermann et al., 2009).

In general, trees carry hierarchical information reflecting the underlying dependency structure of a domain at hand—an appropriate representation of which is often indispensable for learning accurate prediction models. For instance, shallow representations of trees such as flat feature vectors often fail to capture the underlying dependencies. Thus, the prevalent tools for learning with structured data are kernel functions, which implicitly assess pairwise similarities of structured objects and thereby avoid explicit representations (see Müller et al., 2001; Schölkopf and Smola, 2002; Shawe-Taylor and Cristianini, 2004). Kernel functions for structured data can be constructed using the convolution of local kernels defined over sub-structures (Haussler, 1999). A prominent example for such a convolution is the parse tree kernel proposed by Collins and Duffy (2002) which determines the similarity of trees by counting shared subtrees.

The computation of parse tree kernels, however, is inherently quadratic in the number of tree nodes, as it builds on dynamic programming to compute the contribution of shared subtrees. Allocating and updating tables for dynamic programming is feasible for small tree sizes, say less than 200 nodes, so that tree kernels have been widely applied in natural language processing, for example, for question classification (Zhang and Lee, 2003) and parse tree reranking (Collins and Duffy, 2002). Figure 1(a) shows an exemplary parse tree of natural language text. Large trees involve computations that exhaust available resources in terms of memory and run-time. For example, the computation of a parse tree kernel for two HTML documents comprising 10,000 nodes each, requires about 1 gigabyte of memory and takes over 100 seconds on a recent computer system. Given that kernel computations are performed millions of times in large-scale learning, it is evident that regular tree kernels are an inappropriate choice in many learning tasks.



(a) Parse tree for a sentence

(b) Parse tree for a HTTP request

Figure 1: Parse trees for natural language text and the HTTP network protocol.

The limitation of regular tree kernels becomes apparent when considering learning tasks based on formal grammars, such as web spam detection. In web spam detection one seeks to find arrays of linked fraudulent web pages, so-called link farms, that deteriorate the performance of search engines by manipulating search results (e.g., Wu and Davison, 2005; Drost and Scheffer, 2005; Castillo et al., 2006). Besides being densely linked, these web pages share an important property: they are generated automatically according to templates. Hence, a promising approach for web spam detection is the analysis of structure in web pages using parse trees of HTML documents. Unfortunately, HTML documents can grow almost arbitrarily large and render the computation of conventional tree kernels impossible.

Moreover, tree-structured data also arises as part of unsupervised learning problems, such as clustering and anomaly detection. For instance, a critical task in computer security is the automatic detection of novel network attacks (Eskin et al., 2002). Current detection techniques fail to cope with the increasing amount and diversity of network threats, as they depend on manually generated detection rules. As an alternative, one-class learning methods have been successfully employed for identifying anomalies in the context of intrusion detection (e.g., Eskin et al., 2002; Kruegel and Vigna, 2003; Rieck and Laskov, 2007). Due to the already mentioned run-time constraints these approaches focus on shallow features, although efficient methods for extracting syntactical structure from network data are available (e.g., Pang et al., 2006; Borisov et al., 2007; Wondracek et al., 2008). Similar to natural language, network protocols are defined in terms of grammars and individual communication can be represented as parse trees, see Figure 1(b).

To alleviate the limitations of regular tree kernels, we propose *approximate tree kernels* (ATKs) which approximate the kernel computation and thereby allow for efficient learning with arbitrary sized trees in supervised and in unsupervised settings. The efficacy gained by approximate tree kernels rests on a two-stage process: A sparse set of relevant subtrees rooted at appropriate grammar symbols is determined from a small sample of trees, *prior* to subsequent training and testing processes. By decoupling the selection of symbols from the kernel computation, both, run-time and memory requirements are significantly reduced. In the supervised setting, the subset of symbols is optimized with respect to its ability to discriminate between the involved classes, while for the unsupervised setting the optimization is performed with respect to node occurrences in order to minimize the expected run-time. The corresponding optimization problems are translated into linear programs that can be efficiently solved with standard techniques.

Experiments conducted on question classification, web spam detection and network intrusion detection demonstrate the expressiveness and efficiency of our novel approximate kernels. Throughout all our experiments, approximate tree kernels are significantly faster than regular convolution kernels. Depending on the size of the trees, we observe run-time and memory improvements up to 3 orders of magnitude. Furthermore, the approximate tree kernels not only consistently yield the same predictive performance as regular tree kernels, but even outperform their exact counterparts in some tasks by identifying informative dimensions in feature space.

The remainder of this article is organized as follows. We introduce regular parse tree kernels in Section 2 and present our main contribution, the approximate tree kernels, in Section 3. We study the characteristics of approximate kernels on artificial data in Section 4 and report on real-world applications in Section 5. Finally, Section 6 concludes.

2. Kernels for Parse Trees

Let $G = (S, P, s)$ be a grammar with production rules P and a start symbol s defined over a set S of non-terminal and terminal symbols (Hopcroft and Motwani, 2001). A tree X is called a parse tree of G if X is derived by assembling productions $p \in P$ such that every node $x \in X$ is labeled with a symbol $\ell(x) \in S$. To navigate in a parse tree, we address the i-th child of a node x by x_i and denote the number of children by $|x|$. The number of nodes in X is indicated by $|X|$ and the set of all possible trees is given by $\mathcal{X}$.

A kernel $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is a symmetric and positive semi-definite function, which implicitly computes an inner product in a reproducing kernel Hilbert space (Vapnik, 1995). A generic technique for defining kernel functions over structured data is the convolution of local kernels defined

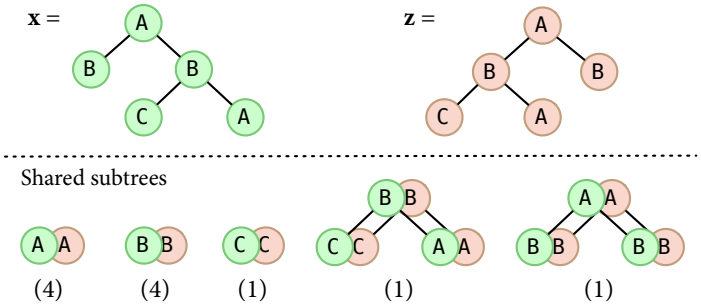


Figure 2: Shared subtrees in two parse trees. The numbers in brackets indicate the number of occurrences for each shared subtree pair.

over sub-structures (Haussler, 1999). Collins and Duffy (2002) apply this concept to parse trees by counting shared subtrees. Given two parse trees X and Z, their *parse tree kernel* is given by

$$k(X,Z) = \sum_{x \in X} \sum_{z \in Z} c(x,z), \tag{1}$$

where the counting function c recursively determines the number of shared subtrees rooted in the tree nodes x and z.

The function c is defined as $c(x,z) = 0$ if x and z are not derived from the same production and $c(x,z) = \lambda$ if x and z are leave nodes of the same production. In all other cases, the definition of the counting function c follows a recursive rule given by

$$c(x,z) = \lambda \prod_{i=1}^{|x|} (1 + c(x_i, z_i)) \,. \tag{2}$$

The trade-off parameter $0 < \lambda \leq 1$ balances the contribution of subtrees, such that small values of λ decay the contribution of lower nodes in large subtrees (see Collins and Duffy, 2002). Figure 2 illustrates two simple parse trees and the corresponding shared subtrees.

Several extensions and refinements of the parse tree kernel have been proposed in the literature to increase its expressiveness for specific learning tasks. For example, setting the constant term in the product of Equation (2) to zero restricts the counting function to take only complete subtrees into account (Vishwanathan and Smola, 2003). Kashima and Koyanagi (2002) extend the counting function to generic trees—not necessarily derived from a grammar—by considering ordered subsets of child nodes for computing the kernel. Further extensions to the counting function proposed by Moschitti (2006b) allow for controlling the vertical as well as the horizontal contribution of subtree counts. Moreover, Moschitti and Zanzotto (2007) extend the parse tree kernel to operate on pairs of trees for deriving relations between sentences in tasks such as textual entailment recognition. However, all of the above mentioned extensions depend on dynamic programming over all pairs of nodes and thus yield prohibitive run-time and memory requirements if large trees are considered.

Selecting discriminative subtrees for tree kernels has been first studied by Suzuki et al. (2004) in the domain of natural language processing. A feature selection procedure based on statistical tests is embedded in the dynamic programming, such that relevant substructures are identified during computation of the parse tree kernel (see also Suzuki and Isozaki, 2005). While this procedure

significantly improves the expressiveness of corresponding tree kernels, the entanglement of feature selection and dynamic programming unfortunately prevents any improvement of run-time and memory requirements over regular tree kernels.

First steps toward run-time improvement of tree kernels have been devised by Moschitti (2006a), who introduces an alternative algorithm limiting computation to node pairs with matching grammar symbols. Although this extension reduces the run-time, it does not alleviate prohibitive memory requirements, as counts for all possible node pairs need to be stored. Also note, that for large trees with $|X| \gg |S|$, only a minor speed-up is gained, since only a small fraction of node pairs can be discarded from the kernel computation. Nevertheless, this algorithm is an efficient approach when learning with small trees as in natural language processing.

3. Approximate Tree Kernels

The previous section argues that the computation of regular tree kernels using dynamic programming is infeasible for large tree structures. In this section we introduce approximate tree kernels to significantly decrease this computational burden. Our approximation of tree kernels is based on the observation that trees often contain redundant parts that are not only irrelevant for the learning task but also slow-down the kernel computation unnecessarily. As an example for redundancy in trees let us consider the task of web spam detection. While few HTML elements, such as header and meta tags, are indicative for web spam templates, the majority of formatting tags is irrelevant and may even harm performance. We exploit this observation by restricting the kernel computation to a sparse set of subtrees rooted in only a few grammar symbols.

In general, selecting relevant subtrees for the kernel computation requires efficient means for enumerating subtrees. The amount of generic subtrees contained in a single parse tree is exponential in the number of nodes and thus intractable for large tree structures. Consequently, we refrain from exhaustive enumeration and limit the selection to subtrees rooted at particular grammar symbols. We introduce a selection function $\omega : S \to \{0,1\}$, which controls whether subtrees rooted in nodes with the symbol $s \in S$ contribute to the convolution ($\omega(s) = 1$) or not ($\omega(s) = 0$). By means of ω, approximate tree kernels are defined as follows.

Definition 1 *Given a selection function $\omega : S \to \{0,1\}$, the approximate tree kernel is defined as*

$$\hat{k}_\omega(X,Z) = \sum_{s \in S} \omega(s) \sum_{\substack{x \in X \\ \ell(x)=s}} \sum_{\substack{z \in Z \\ \ell(z)=s}} \tilde{c}(x,z), \tag{3}$$

where the approximate counting function $\tilde{c}$ is defined as (i) $\tilde{c}(x,z) = 0$ if x and z are not derived from the same production, (ii) $\tilde{c}(x,z) = 0$ if x or z has not been selected, that is, $\omega(\ell(x)) = 0$ or $\omega(\ell(z)) = 0$, and (iii) $c(x,z) = \lambda$ if x and z are leave nodes of the same production. In all other cases, the approximate counting function $\tilde{c}$ is defined as

$$\tilde{c}(x,z) = \lambda \prod_{i=1}^{|x|} (1 + \tilde{c}(x_i, z_i)) .$$

For the task at hand, the selection function ω needs to be adapted to the tree data before the resulting approximate tree kernel can be applied together with learning algorithms. Note that the exact parse tree kernel in Equation (1) is obtained as a special case of Equation (3) if $\omega(s) = 1$ for all symbols $s \in S$. Irrespectively of the actual choice of ω, Proposition 2 shows that the approximate kernel $\hat{k}$ is a valid kernel function.

Proposition 2 *The approximate tree kernel in Definition 1 is a kernel function.*

Proof Let $\phi(X)$ be the vector of frequencies of all subtrees occurring in X as defined by Collins and Duffy (2002). Then, by definition, $\hat{k}_\omega$ can always be written as

$$\hat{k}_\omega(X,Z) = \langle P_\omega \phi(X), P_\omega \phi(Z) \rangle,$$

where P_ω projects the dimensions of $\phi(X)$ on the subtrees rooted in symbols selected by ω. For any ω, the projection P_ω is independent of the actual X and Z, and hence $\hat{k}_\omega$ is a valid kernel. ■

Proposition 2 allows to view approximate tree kernels as feature selection techniques. If the selection function effectively identifies relevant symbols, the dimensionality of the feature space is reduced and the kernel provides access to a refined data representation. We will address this point in Section 4 and 5 where we study eigendecompositions in the induced features spaces. Additionally, we note that the approximate tree kernel realizes a run-time speed-up by a factor of q_ω, which depends on the number of selected symbols in ω and the amount of subtrees rooted at these symbols. For two particular trees X, Z we can state the following simple proposition.

Proposition 3 *The approximate tree kernel $\hat{k}_\omega(X,Z)$ can be computed q_ω times faster than $k(X,Z)$.*

Proof Let $\#_s(X)$ denote the occurrences of nodes $x \in X$ with $\ell(x) = s$. Then the speed-up q_ω realized by $\hat{k}_\omega$ is lower bounded by

$$q_\omega \geq \frac{\sum_{s \in S} \#_s(X) \, \#_s(Z)}{\sum_{s \in S} \omega(s) \, \#_s(X) \, \#_s(Z)} \tag{4}$$

as all nodes with identical symbols in X and Z are paired. For the trivial case where for all elements $\omega(s) = 1$, the factor q_ω equals 1 and the run-time is identical to the parse tree kernel. In all other cases $q_\omega > 1$ holds since at least one symbol is discarded from the denominator in Equation (4). ■

The quality of $\hat{k}_\omega$ and also q_ω depend on the actual choice of ω. If the selection function ω discards redundant and irrelevant subtrees from the kernel computation the approximate kernel can not only be computed faster but also preserves the discriminative expressiveness of the regular tree kernel. Sections 3.1 and 3.2 deal with adapting ω for supervised and unsupervised learning tasks, respectively. Although, the resulting optimization problems are quadratic in the number of instances, selecting symbols can be performed on a small fraction of the data prior to the actual learning process; hence, performance gains achieved in the latter are not affected by the initial selection process. We show in Section 5 that reasonable approximations can be achieved for moderate sample sizes.

3.1 The Supervised Setting

In the supervised setting, we are given n labeled parse trees $(X_1, y_1), \ldots, (X_n, y_n)$ with $y_i \in \mathcal{Y}$. For binary classification we may have $\mathcal{Y} = \{-1, 1\}$ while a multi-class scenario with κ classes gives rise to the set $\mathcal{Y} = \{1, 2, \ldots, \kappa\}$. In the supervised case, the aim of the approximation is to preserve the discriminative power of the regular kernel by selecting a sparse but expressive subset of grammar symbols. We first note that Y with elements $[Y_{ij}]_{i,j=1,\ldots,n}$ given by

$$Y_{ij} = [\![y_i = y_j]\!] - [\![y_i \neq y_j]\!], \tag{5}$$

represents an optimal kernel matrix where $[\![u]\!]$ is an indicator function returning 1 if u is true and 0 otherwise. For binary classification problems, Equation (5) can also be computed by the outer product $Y = yy^{\mathsf{T}}$, where $y = (y_1, \ldots, y_n)^{\mathsf{T}}$.

Inspired by kernel target alignment (Cristianini et al., 2001), a simple measure of the similarity of the approximate kernel $\hat{K}_\omega = [\hat{k}_\omega(X_i, X_j)]_{i,j=1,\ldots,n}$ and the optimal Y is provided by the Frobenius inner product $\langle \cdot, \cdot \rangle_{\mathcal{F}}$ that is defined as $\langle A, B \rangle_{\mathcal{F}} = \sum_{ij} a_{ij} b_{ij}$. We have,

$$\langle Y, \hat{K}_\omega \rangle_{\mathcal{F}} = \sum_{y_i = y_j} \hat{k}_{ij} - \sum_{y_i \neq y_j} \hat{k}_{ij}. \tag{6}$$

The right hand side of Equation (6) measures the within class (first term) and the between class (second term) similarity. Approximate kernels that discriminate well between the involved classes realize large values of $\langle Y, \hat{K}_\omega \rangle_{\mathcal{F}}$, hence maximizing Equation (6) with respect to ω suffices for finding approximate kernels with high discriminative power.

We arrive at the following integer linear program that has to be maximized with respect to ω to align $\hat{K}_\omega$ to the labels y,

$$\max_{\omega \in \{0,1\}^{|S|}} \sum_{\substack{i,j=1 \\ i \neq j}}^{n} y_i y_j \hat{k}_\omega(X_i, X_j). \tag{7}$$

Note that we exclude diagonal elements, that is, $\hat{k}_\omega(X_i, X_i)$, from Equation (7), as the large self-similarity induced by the parse tree kernel impacts numerical stability on large tree structures (see Collins and Duffy, 2002).

Optimizing Equation (7) directly will inevitably reproduce the regular convolution kernel as all subtrees contribute positively to the kernel computation. As a remedy, we restrict the number of supporting symbols of the approximation by a pre-defined constant N. Moreover, instead of optimizing the integer program directly, we propose to use a relaxed variant thereof, where a threshold is used to discretize ω. Consequently, we obtain the following relaxed linear program that can be solved with standard solvers.

Optimization Problem 1 (Supervised Setting) *Given a labeled training sample of size n and let $N \in \mathbb{N}$. The optimal selection function ω^* can be computed by solving*

$$\begin{aligned} \omega^* = \operatorname*{argmax}_{\omega \in [0,1]^{|S|}} & \sum_{\substack{i,j=1 \\ i \neq j}}^{n} y_i y_j \sum_{s \in S} \omega(s) \sum_{\substack{x \in X_i \\ \ell(x) = s}} \sum_{\substack{z \in X_j \\ \ell(z) = s}} \tilde{c}(x, z) \\ \textit{subject to} \quad & \sum_{s \in S} \omega(s) \leq N, \end{aligned} \tag{8}$$

where the counting function $\tilde{c}$ is given in Definition (3).

3.2 The Unsupervised Setting

Optimization Problem 1 identifies N symbols providing the highest discriminative power given a labeled training set. In the absence of labels, for instance in an anomaly detection task, the operational goal needs to be changed. In these cases, the only accessible information for finding ω are the tree structures themselves. Large trees are often characterized by redundant substructures that strongly impact run-time performance while encoding only little information. Moreover, syntactical

structure present in all trees of a data set do not provide any information suitable for learning. As a consequence, we seek a selection of symbols using a constraint on the expected run-time and thereby implicitly discard redundant and costly structures with small contribution to the kernel value.

Given unlabeled parse trees $X_1, X_2, \ldots, X_n$, we introduce a function $f(s)$ which measures the average frequency of node comparisons for the symbol s in the training set, defined as

$$f(s) = \frac{1}{n^2} \sum_{i,j=1}^{n} \#_s(X_i) \#_s(X_j). \tag{9}$$

Using the average comparison frequency f, we bound the expected run-time of the approximate kernel by $\rho \in (0,1]$, the ratio of expected node comparisons with respect to the exact parse tree kernel. The following optimization problem details the computation of the optimal ω^* for unsupervised settings.

Optimization Problem 2 (Unsupervised Setting) *Given an unlabeled training sample of size n and a comparison ratio $\rho \in (0,1]$.*

$$\begin{aligned} \omega^* = \operatorname*{argmax}_{\omega \in [0,1]^{|S|}} & \sum_{\substack{i,j=1 \\ i \neq j}}^{n} \sum_{s \in S} \omega(s) \sum_{\substack{x \in X_i \\ \ell(x)=s}} \sum_{\substack{z \in X_j \\ \ell(z)=s}} \tilde{c}(x,z) \\ \text{subject to} \quad & \frac{\sum_{s \in S} \omega(s)\, f(s)}{\sum_{s \in S} f(s)} \leq \rho, \end{aligned} \tag{10}$$

where the counting function $\tilde{c}$ is given in Definition 3 and f defined as in Equation (9).

The optimal selection function ω^* gives rise to a tree kernel that approximates the regular kernel as close as possible, while on average considering a fraction of ρ node pairs for computing the similarities. Analogously to the supervised setting, we solve the relaxed variant of the integer program and use a threshold to discretize the resulting ω.

3.3 Extensions

The supervised and unsupervised formulations in Optimization Problem 1 and 2 build on different constraints for determining a selection of appropriate symbols. Depending on the learning task at hand, these constraints are exchangeable, such that approximate tree kernels in supervised settings may also be restricted to the ratio ρ of expected node comparisons and the unsupervised formulation can be alternatively conditioned on a fixed number of symbols N. Both constraints may even be applied jointly to limit expected run-time and the number of selected symbols. For our presentation, we have chosen a constraint on the number of symbols in the supervised setting, as it provides an intuitive measure for the degree of approximation. By contrast, we employ a constraint on the expected number of node comparisons in the unsupervised formulation, as in the absence of labels, it is easier to bound the run-time, irrespective of the number of selected symbols.

Further extensions incorporating prior knowledge into the proposed approximations are straight forward. For instance, the approximation procedure can be refined using logical compounds based on conjunctions and disjunctions of symbols. If the activation of symbol s_j requires the activation of s_{j+1}, the constraint $\omega(s_j) - \omega(s_{j+1}) = 0$ can be included in Equation (8) and (10). A conjunction (AND) of m symbols can then be efficiently encoded by $m-1$ additional constraints as

$$\forall_{j=1}^{m-1} \omega(s_j) - \omega(s_{j+1}) = 0.$$

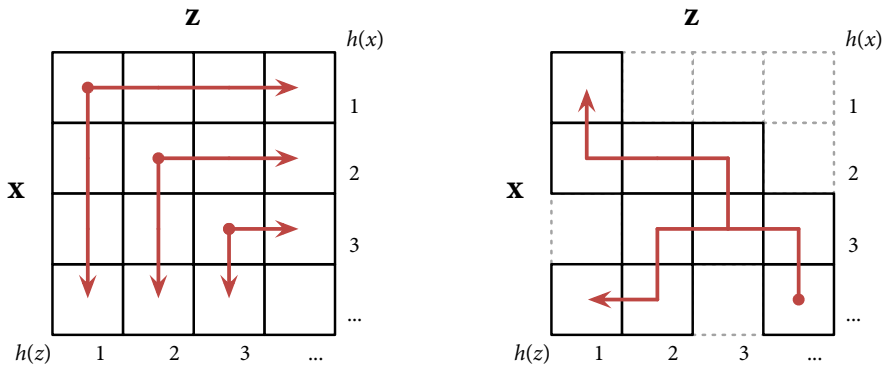
For a disjunction (OR) of symbols $s_j, \ldots, s_{j+m}$ the following constraint guarantees that at least one representative of the group is active in the solution

$$\omega(s_j) + \omega(s_{j+1}) + \ldots + \omega(s_{j+m}) \geq 1. \tag{11}$$

Alternatively, Equation (11) can be turned into an exclusive disjunction (XOR) of symbols by changing the inequality into an equality constraint.

3.4 Implementation

A standard technique for computing tree kernels is dynamic programming, where a table of $|X| \times |Z|$ elements is used to store the counts of subtrees during recursive evaluations. The kernel computation is then carried out in either a systematic or a structural fashion, where intermediate results are stored in the table as sketched in Figure 3. The systematic variant processes the subtrees with ascending height, such that at a particular height, all counts for lower subtrees can be looked up in the table (Shawe-Taylor and Cristianini, 2004). For the structural variant, the dynamic programming table acts as a cache, which stores previous results when computing the recursive counting directly (Moschitti, 2006a). This latter approach has the advantage that only matching subtrees are considered and mismatching nodes do not contribute to the run-time as in the systematic variant.



(a) Systematic mode (b) Structural mode

Figure 3: Dynamic programming for tree kernels. The convolution of subtree counts is computed in a systematic (Shawe-Taylor and Cristianini, 2004) or structural manner (Moschitti, 2006a). The term $h(x)$ denotes the height of the subtree rooted in $x \in X$.

We now provide details on an implementation of approximate tree kernels using the structural approach. The inputs of our implementation are parse trees X, Z and a selection function ω which has been determined in advance using the supervised or unsupervised approximation (Optimization Problems 1 and 2). The implementation proceeds by first generating pairs of matching nodes from the trees X, Z, similarly to the algorithm proposed by Moschitti (2006a). However, pairs whose symbols are not selected by ω are omitted. The computation of the tree kernel is then carried out by looping over the node pairs and counting the number of shared subtrees rooted at each pair. An exemplary implementation of the approximate kernel is given in Algorithms 1, 2 and 3.

Algorithm 1 Approximate Tree Kernel

```
1: function KERNEL(X, Z, ω)
2:     L ← GENERATEPAIRS(X, Z, ω)
3:     k ← 0
4:     for (x, z) ← L do                          ▷ Loop over selected pairs of nodes
5:         k ← k + COUNT(x, z)
6:     return k
```

Algorithm 1 realizes a generic tree kernel, which determines the number of shared subtrees by looping over a list of node pairs. Algorithm 2 shows the corresponding analogue of the counting function in Equation (2) which is called during each iteration of the loop. While the standard implementation of the parse tree kernel (e.g., Shawe-Taylor and Cristianini, 2004; Moschitti, 2006a) uses a dynamic programming table to store the contribution of subtree counts, we employ a hash table denoted by H. A hash table guarantees constant reading and writing of intermediate results yet it grows with the number of selected node pairs and thereby reduces memory in comparison to a standard table of all possible pairs. Note that if all symbols in ω are selected, H realizes the standard dynamic programming approach.

Algorithm 2 Counting Function

```
1: function COUNT(x, z)
2:     if x and z have different productions then
3:         return 0
4:     if x or z is a leaf node then
5:         return λ
6:     if (x, z) stored in hash table H then
7:         return H(x, z)                         ▷ Read dynamic programming cell
8:     c ← 1
9:     for i ← 1 to |x| do
10:        c ← c · (1 + COUNT(x_i, z_i))
11:    H(x, z) ← λc                               ▷ Write dynamic programming cell
12:    return H(x, z)
```

Algorithm 3 implements the function for generating pairs of nodes with selected symbols. The function first sorts the tree nodes using a predefined order in line 2–3. For our implementation we apply a standard lexicographic sorting on the symbols of nodes. Algorithm 3 then proceeds by generating a set of matching node pairs L, satisfying the invariant that included pairs $(x,z) \in L$ have matching symbols (i.e., $\ell(x) = \ell(z)$) and are selected via ω (i.e., $\omega(x) = 1$). The generation of pairs is realized analogously to merging sorted arrays (see Knuth, 1973). The function removes elements from the lists of sorted nodes N_X and N_Z in parallel until a matching and selected pair (x,z) is discovered. With a slight abuse of notation, all available node pairs (a,b) with label $\ell(x)$ are then added to L and removed from N_X and N_Z in lines 12–14 of Algorithm 3.

Algorithm 3 Node Pair Generation

```
1: function GENERATEPAIRS(X, Z, ω)
2:     N_X ← SORTNODES(X)
3:     N_Z ← SORTNODES(Z)
4:     while N_X and N_Z not empty do
5:         x ← head of N_X
6:         z ← head of N_Z
7:         if ℓ(x) < ℓ(z) or ω(x) = 0 then
8:             remove x from N_X                    ▷ x mismatches or not selected
9:         else if ℓ(x) > ℓ(z) or ω(z) = 0 then
10:            remove z from N_Z                    ▷ y mismatches or not selected
11:        else
12:            N ← { (a,b) ∈ N_X × N_Z with label ℓ(x) }
13:            L ← L ∪ N                            ▷ Add all pairs with label ℓ(x)
14:            remove N from N_X and N_Z
15:    return L
```

3.5 Application Setup

In contrast to previous work on feature selection for tree kernels (see Suzuki et al., 2004), the efficiency of our approximate tree kernels is rooted in *decoupling* the selection of symbols from later application of the learned kernel function. In particular, our tree kernels are applied in a two-stage process as detailed in the following.

1. *Selection stage.* In the first stage, a sparse selection ω of grammar symbols is determined on a sample of tree data, where depending on the learning setting either Optimization Problem 1 or 2 is solved by linear programming. As solving both problems involves computing exact tree kernels, the selection is optimized on a small fraction of the trees. To limit memory requirements, the sample may be further filtered to contain only trees of reasonable sizes.

2. *Application stage.* In the subsequent application stage, the approximate tree kernels are employed together with learning algorithms using the efficient implementation detailed in the previous section. The optimized ω reduces the run-time and memory requirements of the kernel computation, such that learning with trees of almost arbitrary size becomes feasible.

The approximate tree kernels involve the parameter λ as defined in Equation (2). The parameter controls the contribution of subtrees; values close to zero emphasize shallow subtrees and $\lambda = 1$ corresponds to a uniform weighting of all subtrees. To avoid repeatedly solving Optimization Problem 1 or 2 for different values of λ, we fix $\lambda = 1$ in the selection stage and perform model selection only in the application stage for $\lambda \in [10^{-4}, 10^{0}]$. This procedure ensures that the approximate tree kernels are first determined over equally weighted subtrees, hence allowing for an unbiased optimization in the selection phase. A potential refinement of λ is postponed to the application stage to exploit performance gains of the approximate tree kernel. Note that if prior knowledge is available, this may be reflected by a different choice of λ in the selection stage.

4. Experiments on Artificial Data

Before studying the expressiveness and performance of approximate tree kernels in real-word applications, we aim at gaining insights into the approximation process. We thus conduct experiments using artificial data generated by the following probabilistic grammar, where A, B, C, D denote non-terminal symbols and a, b terminal symbols. The start symbol is S.

$$S \xrightarrow{[1.0]} A\ B \tag{*1}$$
$$A \xrightarrow{[0.2|0.2|0.6]} A\ A \mid C\ D \mid a \tag{*2}$$
$$B \xrightarrow{[0.2|0.2|0.6]} B\ B \mid D\ C \mid b \tag{*3}$$
$$C \xrightarrow{[0.3|0.3|0.3]} A\ B \mid A \mid B \tag{*4}$$
$$D \xrightarrow{[0.3|0.3|0.3]} B\ A \mid A \mid B \tag{*5}$$

Parse trees are generated from the above grammar by applying the rule $S \to AB$ and randomly choosing matching production rules according to their probabilities until all branches end in terminal nodes. Recursions are included in (*2)–(*5) to ensure that symbols occur at different positions and depths in the parse trees.

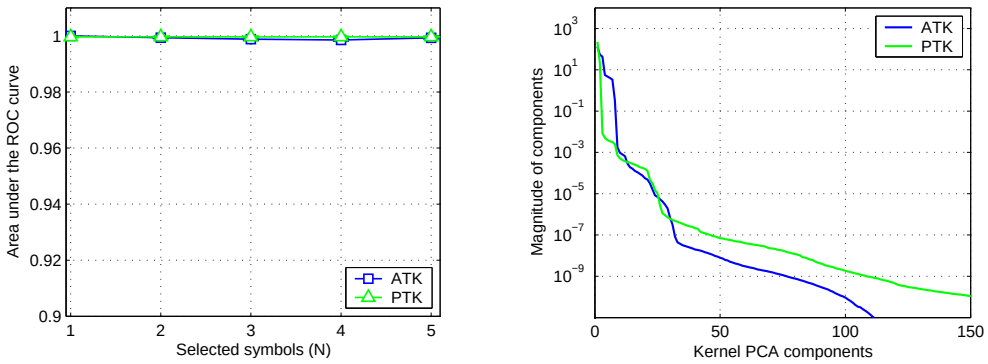
4.1 A Supervised Learning Task

To generate a simple supervised classification task, we assign the first rule in (*4) as an indicator of the positive class and the first rule in (*5) as one of the negative class. We then prepare our data set, such that one but not two of the rules are contained in each parse tree. That is, positive examples possess the rule $C \to AB$ and negative instances exhibit the rule $D \to BA$. Note that due to the symmetric design of the production rules, the two classes can not be distinguished from the symbols C and D alone but from the respective production rules.

Using this setup, we generate training, validation, and test sets consisting of 500 positive and negative trees each. We then apply the two-stage process detailed in Section 3.5: First, the selection function ω is adapted by solving Optimization Problem 1 using a sample of 250 randomly drawn trees from the training set. Second, a Support Vector Machine (SVM) is trained on the training data and applied to the test set, where the optimal regularization parameter of the SVM and the depth parameter λ are selected using the validation set. We report on averages over 10 repetitions and error bars indicate standard errors.

The classification performance of the SVM for the two kernel functions is depicted in Figure 4, where the number of selected symbols N for the approximate kernel is given on the x-axis and the attained area under the ROC curve (AUC) is shown on the y-axis. The parse tree kernel (PTK) leads to a perfect discrimination between the two classes, yet the approximate tree kernel (ATK) performs equally well, irrespectively of the number of selected symbols. That is, the approximation captures the discriminant subtrees rooted at either the symbol C or D in all settings. This selection of discriminative subtrees is also reflected in the optimal value of the depth parameter λ determined during the model selection. While for the exact tree kernel the optimal λ is 10^{-2}, the approximate kernel yields best results with $\lambda = 10^{-3}$, thus putting emphasis on shallow subtrees and the discriminative production rules rooted at C and D.

To analyze the feature space induced by the selection of subtrees, we perform a kernel principle component analysis (PCA) (see Schölkopf et al., 1998; Braun et al., 2008) for the exact and the



(a) Classification performance

(b) Kernel PCA plot for $N = 1$

Figure 4: Classification performance and kernel PCA plot for the supervised toy data.

approximate tree kernel. Figure 4(b) shows the sorted magnitudes of the principal components in feature space. Although the differences are marginal, comparing the spectra allows for viewing the approximate kernel as a denoised variant of the regular parse tree kernel. The variance of smaller components is shifted towards leading principle components, resulting in a dimensionality reduction in feature space (see Mika et al., 1999).

4.2 An Unsupervised Learning Task

In order to obtain an unsupervised learning task, we modify the artificial grammar to reflect the notion of anomaly detection. First, we incorporate redundancy into the grammar by increasing the probability of irrelevant production rules in (*4)–(*5) as follows

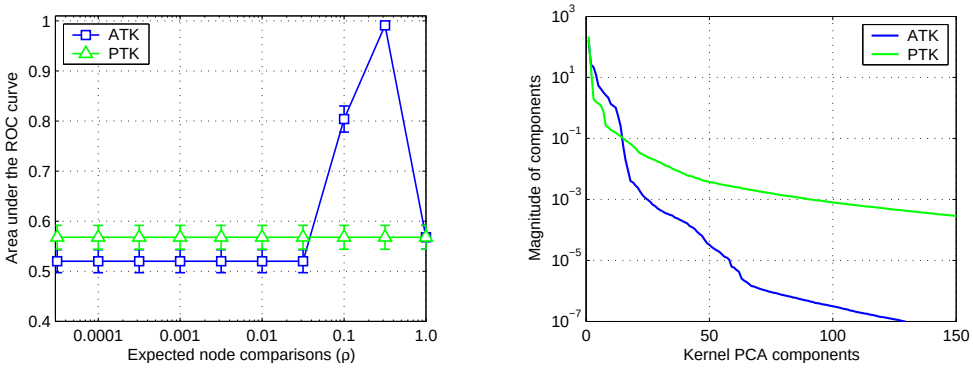
$$C \xrightarrow{[0.1|0.4|0.4]} A\ B \mid A \mid B \qquad (*4)$$

$$D \xrightarrow{[0.1|0.4|0.4]} B\ A \mid A \mid B \qquad (*5)$$

Second, we sample the parse trees such that training, validation, and testing sets contain 99% positive and 1% negative instances each, thus matching the anomaly detection scenario. We pursue the same two-stage procedure as in the previous section but first solve Optimization Problem 2 for adapting the approximate tree kernel to the unlabeled data and then employ a one-class SVM (Schölkopf et al., 1999) for training and testing.

Figure 5(a) shows the detection performance for the parse tree kernel and the approximate tree kernel for varying values of ρ. The parse tree kernel reaches an AUC value of 57%. Surprisingly, we observe a substantial gain in performance for the approximate kernel, leading to an almost perfect separation of the two classes for $\rho = 0.3$. Moreover, for the approximate kernel shallow subtrees are sufficient for detection of anomalies which is indicated by an optimal $\lambda = 10^{-3}$, whereas for the exact kernel subtrees of all depths need to be considered due to an optimal $\lambda = 1$.

The high detection performances can be explained by considering a kernel PCA of the two tree kernels in Figure 5(b). The redundant production rules introduce irrelevant and noisy dimensions into the feature space induced by the parse tree kernel. Clearly, for $\rho = 0.3$, the approximate tree kernel effectively reduces the intrinsic dimensionality by shifting the variance towards leading



(a) Detection performance (b) Kernel PCA plot for $\rho = 0.3$

Figure 5: Detection performance and kernel PCA plot for the unsupervised toy data.

components. Compared to the exact kernel, the resulting eigenspectrum of the approximate kernel possesses more explanatory and fewer noisy components.

5. Real-world Experiments

We now proceed to study the expressiveness, stability, and run-time performance of approximate tree kernels in real-world applications, namely supervised learning tasks dealing with question classification and web spam detection, respectively, and an unsupervised learning task on intrusion detection for HTTP and FTP traffic. In all experiments we employ the exact parse tree kernel and state-of-the-art implementations as baseline methods.

- **Question Classification.** Question classification is an important step for automatic answering of questions (Voorhees, 2004). The task is to categorize a user-supplied question into predefined semantic categories. We employ the data collection by Li and Roth (2002) consisting of 6,000 English questions assigned to six classes (abbreviation, entity, description, human, location, numeric value). Each question is transformed to a respective parse tree using the MEI Parser[1] (Charniak, 1999). For simplicity, we learn a discrimination between the category "entity" (1,339 instances) and all other categories using a two-class Support Vector Machine (SVM).

- **Web Spam Detection.** Web spam refers to fraudulent HTML documents, which yield high ranks in search engines through massive amounts of links. The detection of so-called link farms is essential for providing proper search results and protecting users from fraud. We use the web spam data as described by Castillo et al. (2006). The collection consists of HTML documents from normal and spam websites in the UK. All sites are examined by humans and manually annotated. We use a fault-tolerant HTML parser[2] to obtain parse trees from HTML documents. From the top 20 sites of both classes we sample 5,000 parse trees covering 974 web spam documents and 4,026 normal HTML pages. Again, we use a two-class SVM as the underlying learning algorithm.

1. Maximum-Entropy-Inspired Parser, see `ftp://ftp.cs.brown.edu/pub/nlparser`.
2. Beautiful Soup Parser, see `http://www.crummy.com/software/BeautifulSoup`.

- **Intrusion Detection.** Intrusion detection aims to automatically identify unknown attacks in network traffic. As labels for such data are hard to obtain, unsupervised learning has been a major focus in intrusion detection research (e.g., Eskin et al., 2002; Kruegel and Vigna, 2003; Rieck and Laskov, 2007; Laskov et al., 2008). Thus, for our experiments we employ a one-class SVM (Schölkopf et al., 1999) in the variant of Tax and Duin (1999) to detect anomalies in network traffic of the protocols HTTP and FTP. Network traffic for HTTP is recorded at the Fraunhofer FIRST institute, while FTP traffic is obtained from the Lawrence Berkeley National Laboratory [3] (see Paxson and Pang, 2003). Both traffic traces cover a period of 10 days. Attacks are additionally injected into the traffic using popular hacking tools.[4] The network data is converted to parse trees using the protocol grammars provided in the specifications (see Fielding et al., 1999; Postel and Reynolds, 1985). From the generated parse trees for each protocol we sample 5,000 instances and add 89 attacks for HTTP and 62 for FTP, respectively. This setting is similar to the data sets used in the DARPA intrusion detection evaluation (Lippmann et al., 2000).

Figure 6 shows the distribution of tree sizes in terms of nodes for each of the three learning tasks. For question classification, the largest tree comprises 113 nodes, while several parse trees in the web spam and intrusion detection data consist of more than 5,000 nodes.

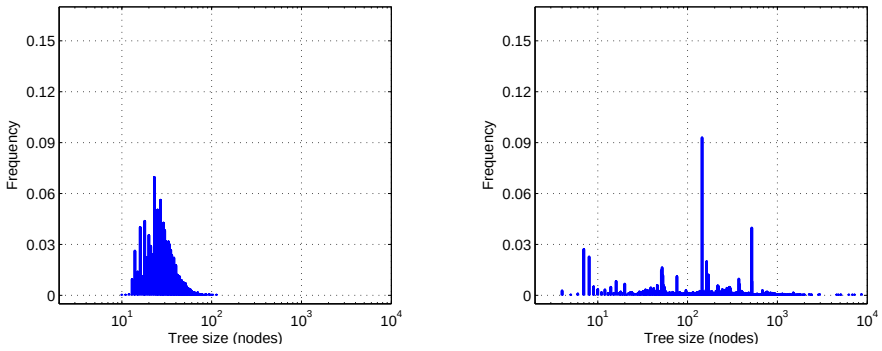
For each learning task, we pursue the two-stage procedure described in Section 3.5 and conduct the following experimental procedure: parse trees are randomly drawn from each data set and split into training, validation and test partitions consisting of 1,000 trees each. If not stated otherwise, we first draw 250 instances at random from the training set for the selection stage, where we solve Optimization Problem 1 or 2 with fixed $\lambda = 1$. In the application stage, the resulting approximate kernels are then compared to exact kernels using SVMs as underlying learning methods. Model selection is performed for the regularization parameter of the SVM and the depth parameter λ. We measure the area under the ROC curve of the resulting classifiers and report on averages over 10 repetitions with error bars indicate standard errors. In all experiments we make use of the LIBSVM library developed by Chang and Lin (2000).

5.1 Results for Question Classification

We first study the expressiveness of the approximate tree kernel and the exact parse tree kernel for the question classification task. We thus vary the number of selected symbols in Optimization Problem 1 and report on the achieved classification performance for the approximate tree kernel for varying N and the exact tree kernel in Figure 7(a).

As expected, the approximation becomes more accurate for increasing values of N, meaning that the more symbols are included in the approximation, the better is the resulting discrimination. However, the curve saturates to the performance of the regular parse tree kernel for selecting 7 and more symbols. The selected symbols are NP, VP, PP, S1, SBARQ, SQ, and TERM. The symbols NP, PP, and VP capture the coarse semantics of the considered text, while SBARQ and SQ correspond to the typical structure of questions. Finally, the symbol TERM corresponds to terminal symbols and contains the actual sequence of tokens including interrogative pronouns. The optimal depth λ for the approximate kernel is again lower with 10^{-2} in comparison to the optimal value of 10^{-1} for the exact kernel, as discriminative substructures are close to the selected symbols.

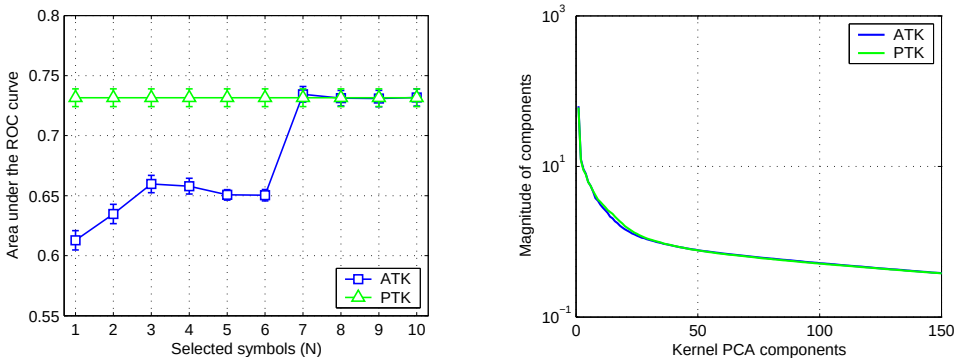
3. LBNL-FTP-PKT, http://www-nrg.ee.lbl.gov/anonymized-traces.html.
4. Metasploit Framework, see http://www.metasploit.org.



(a) Question classification (b) Web spam detection

(c) Intrusion detection (HTTP) (d) Intrusion detection (FTP)

Figure 6: Tree sizes for question classification, web spam detection and intrusion detection.

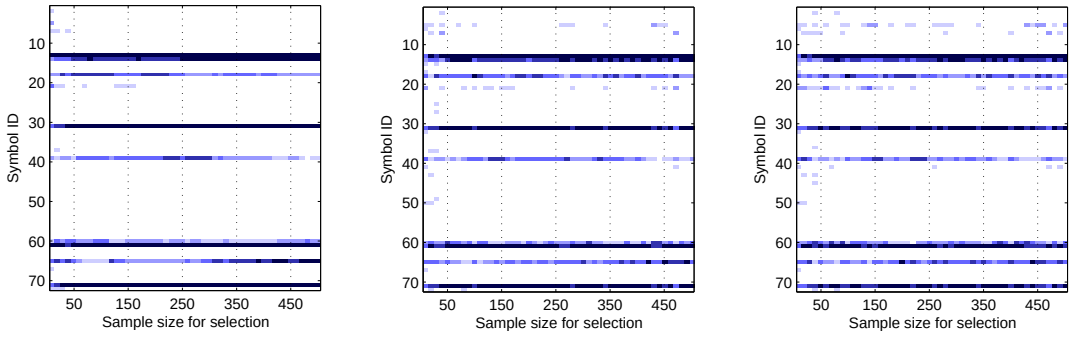


(a) Classification performance (b) Kernel PCA plot for $N = 7$

Figure 7: Classification performance and kernel PCA plot for the question classification task.

Figure 7(b) shows the eigenspectra of the parse tree kernel and its approximate variant with the above 7 selected symbols. Even though the proposed kernel is only an approximation of the regular tree kernel, their eigenspectra are nearly identical. That is, the approximate tree kernel leads to a nearly identical feature space to its exact counterpart.

The above experiments demonstrate the ability of the approximation to select discriminative symbols, yet it is not clear how the expressiveness of the approximate kernels depends on the re-



(a) Selection (b) Selection with 5% label noise (c) Selection with 10% label noise

Figure 8: Stability plot for the question classification task.

duced sample size in the selection stage. To examine this issue, we keep $N = 7$ fixed and vary the amount of data supplied for adapting the approximate tree kernel. Figure 8(a) displays the assignments of the selection function ω, where the size of the provided data is shown on the x-axis and the IDs of the grammar symbols are listed on the y-axis. The intensity of each point reflects the average number of times the corresponding symbol has been chosen in five repetitions. The selection remains stable for sample sizes of 150 parse trees, where consistently the correct 7 symbols are identified. Even if label noise is injected in the data, the approximation remains stable if at least 150 trees are considered for the selection as depicted in Figure 8(b) and 8(c).

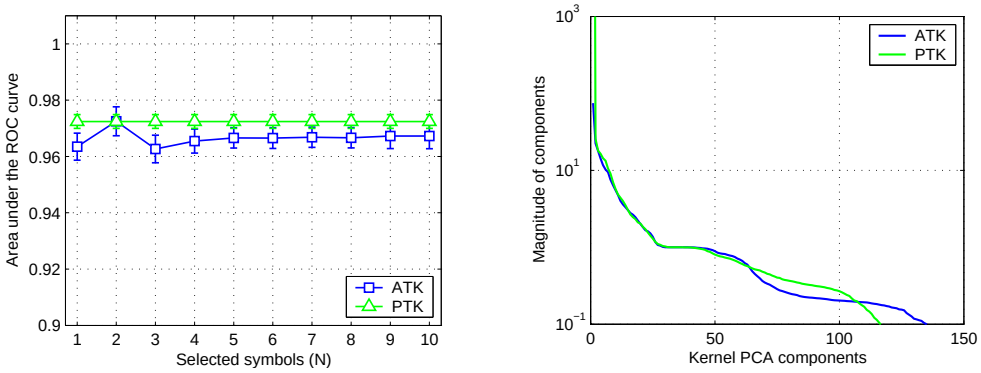
The results on question classification show that exploiting the redundancy in parse trees can be beneficial even when dealing with small trees. Approximate tree kernels identify a simplified representation that proves robust against label noise and leads to the same classification rate compared to regular parse tree kernels.

5.2 Results for Web Spam Detection

We now study approximate tree kernels for web spam detection. Unfortunately, training SVMs using the exact parse tree kernel proves intractable for many large trees in the data due to their excessive memory requirements. We thus exclude trees from the web spam data set with more than 1,500 nodes for the following experiments. Again, we vary the number of symbols to be selected and measure the corresponding AUC value over 10 repetitions.

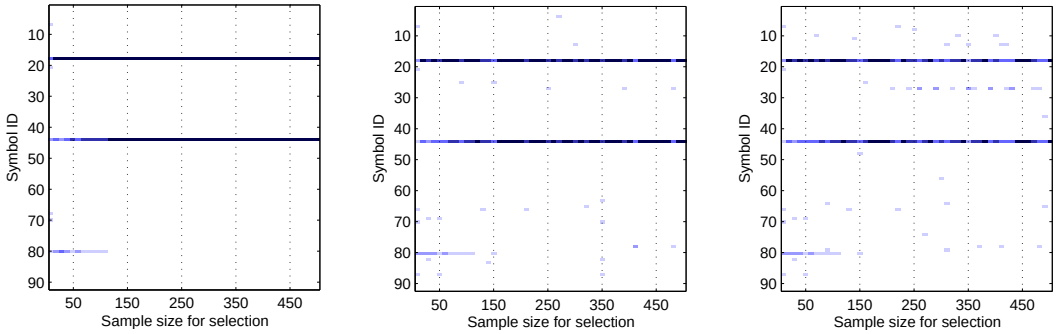
The results are shown in Figure 9(a). The approximation is consistently on par with the regular parse tree kernel for four and more selected labels, as the differences in this interval are not significant. However, the best result is obtained for selecting only two symbols. The approximation picks the tags HTML and BODY. We credit this finding to the usage of templates in spam websites inducing a strict order of high-level tags in the documents. In particular, header and meta tags occurring in subtrees below the HTML tag are effective for detecting spam templates. As a consequence, the optimal $\lambda = 10^{-1}$ for the approximate kernel effectively captures discriminative features reflecting web spam templates rooted at the HTML and BODY tag. The eigendecomposition of the two kernels in Figure 9(b) hardly show any differences. As for the question classification task, the exact and approximate tree kernels share the same expressiveness.

Figure 10 shows the stability of the selection function for varying amounts of data considered in the selection stage where N is fixed to 2. The selection saturates for samples containing at least 120



(a) Classification performance (b) Kernel PCA plot for $N = 2$

Figure 9: Classification performance and kernel PCA plot for the web spam detection task.



(a) Selection (b) Selection with 5% label noise (c) Selection with 10% label noise

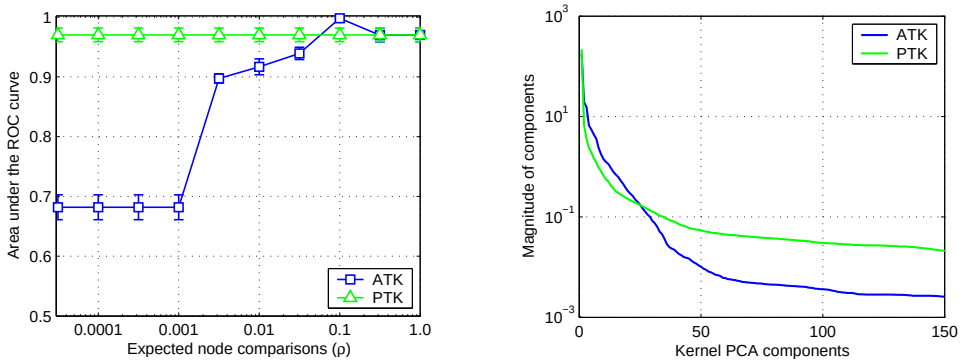
Figure 10: Stability plot for the web spam detection task.

parse trees and the two symbols HTML and BODY are chosen consistently. Moreover, the selection of symbols for web spam detection proves robust against label noise. Even for a noise ratio of 10% which corresponds to flipping every tenth label, the same symbols are selected. This result confirms the property of web spam to be generated from templates which provides a strong feature for discrimination even in presence of label noise.

5.3 Results for Intrusion Detection

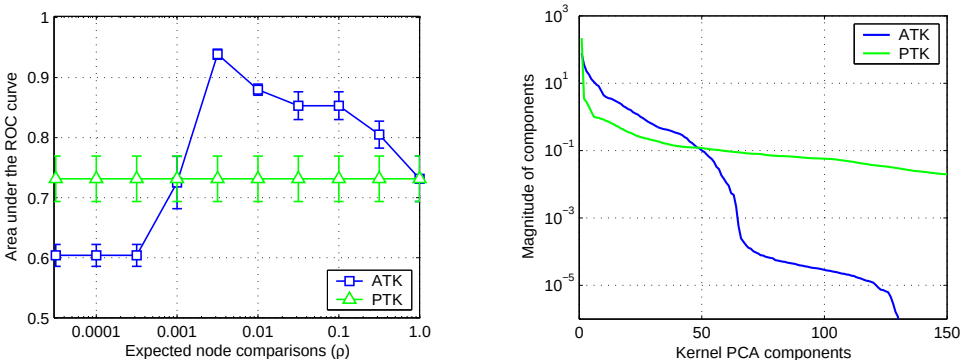
In this section, we study the expressiveness of approximate kernels for unsupervised intrusion detection. Since label information is not available, we adapt the selection function to the data using Optimization Problem 2. The resulting approximate tree kernels are then employed together with a one-class SVM for the detection of attacks in HTTP and FTP parse trees. To determine the impact of the approximation on the detection performance, we vary the number of expected node comparisons, that is, variable ρ in Optimization Problem 2. We again exclude trees comprising more than 1,500 nodes due to prohibitive memory requirements for the exact tree kernel.

Figures 11 (HTTP) and 12 (FTP) show the observed detection rates on the left for the approximate and the exact tree kernel. Clearly, the approximate tree kernel performs identically to its exact counterpart if the ratio of node comparisons ρ equals 100%. However, when the number of



(a) Detection performance (b) Kernel PCA plot for $\rho = 0.1$

Figure 11: Detection performance and analysis of the intrusion detection task (HTTP).



(a) Detection performance (b) Kernel PCA plot for $\rho = 0.003$

Figure 12: Detection performance and analysis of the intrusion detection task (FTP).

comparisons is restricted to only a fraction, the approximate kernel significantly outperforms the exact parse tree kernel and leads to a superior detection rate. The approximate tree kernel realizes an AUC improvement of 1% for HTTP data. For the FTP protocol, the differences are more severe: the approximate kernel outperforms its exact counterpart and yields an AUC improvement of 20%. The optimal depth parameter λ for the approximate kernel is 10^{-2} for HTTP and 10^{-2} for FTP, while the exact tree kernel requires $\lambda = 10^{-1}$ in the optimal setting. This result demonstrates that the approximation identifies relevant grammar symbols by focusing on shallow subtrees comprising discriminative patterns.

These gains in performance can be explained by looking at the respective eigenspectra, depicted in Figures 11(b) and 12(b). Compared to the regular kernel, the approximate kernel yields remarkably fewer noisy components. This is particularly the case for FTP traffic. Moreover, the variance is shifted toward only a few leading components. The approximate tree kernel performs a dimensionality reduction by suppressing noisy and redundant parts of the feature space. For HTTP traffic, such redundancy is for instance induced by common web browsers like Internet Explorer and Mozilla Firefox whose header attributes constitute a good portion of the resulting parse trees. This syntactical information is delusive in the context of intrusion detection and hence their removal

improves the detection performance. Note that the observed gain in performance is achieved using only less than 10% of the grammar symbols. That is, the approximation is not only more accurate than the exact parse tree kernel but also concisely represented.

5.4 Run-time Performance

As we have seen in the previous sections, approximate kernels can lead to a concise description of the task at hand by selecting discriminative substructures in data. In this section we compare the run-time and memory requirements of the approximate tree kernel with state-of-the-art implementations of the exact tree kernels. We first measure the time for selection, training, and testing phases using SVMs as underlying learning methods. For all data sets, we use 250 randomly drawn trees in the selection stage where training and test sets consist of 1,000 instances each. Again, we exclude large trees with more than 1,500 nodes because of excessive memory requirements.

Selection stage on 250 parse trees	
Question classification	17s $\pm$0
Web spam detection	144s $\pm$28
Intrusion detection (HTTP)	43s $\pm$7
Intrusion detection (FTP)	31s $\pm$2

Table 1: Selection stage prior to training and testing phase.

The run-time for the selection of symbols prior to application of the SVMs are presented in Table 1. For all three data sets, a selection is determined in less than 3 minutes, demonstrating the advantage of phrasing the selection as a simple linear program. Table 5.4 lists the training and testing times using the approximate tree kernel (ATK) and a fast implementation for the exact tree kernel (PTK2) devised by Moschitti (2006a). As expected, the size of the trees influences the observed results. For the small trees in the question classification task we record run-time improvements by a factor of 1.7 while larger trees in the other tasks give rise to speed-up factors between $2.8 - 13.8$. Note that the total run-time of the application stage is only marginally affected by the initial selection stage that is performed only once prior to the learning process. For example, in the task of web spam detection a speed-up of roughly 10 is attained for the full experimental evaluation, as the selection is performed once, whereas 25 runs of training and testing are necessary for model selection.

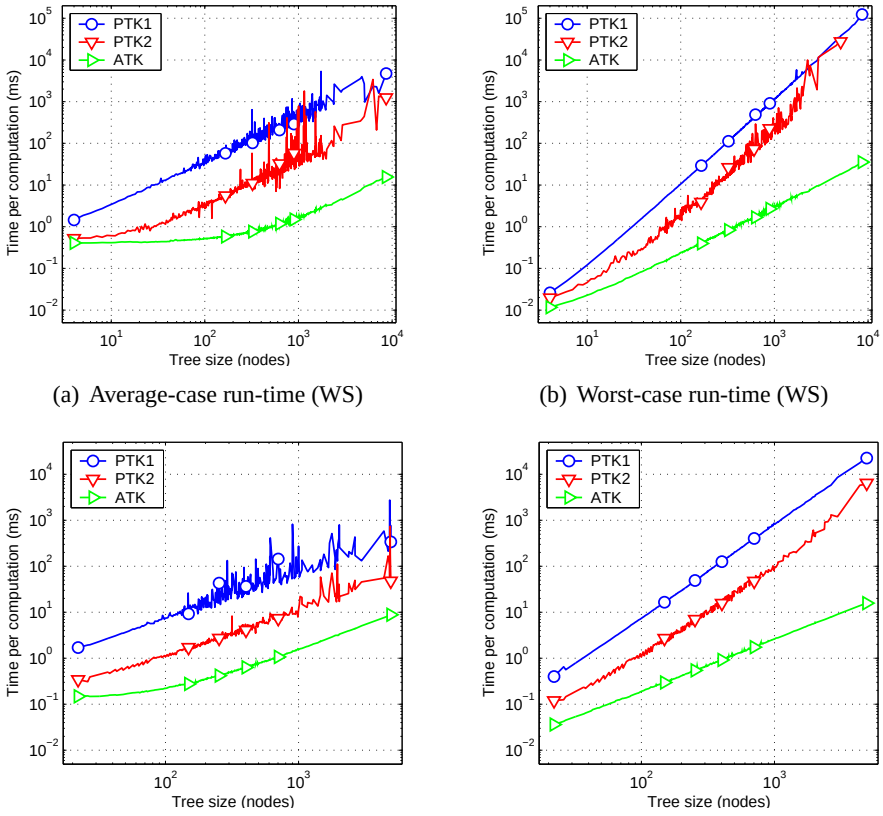
However, the interpretability of the results reported in Table 5.4 is limited because parse trees containing more than 1,500 nodes have been excluded from the experiment and the true performance gain induced by approximate tree kernels cannot be estimated. Moreover, the reported training and testing times refer to a particular learning method and cannot be transferred to other methods and applications, such as clustering and regression tasks. To address these issues, we study the run-time performance and memory consumption of tree kernels explicitly—independently of a particular learning method. Notice that for these experiments we include parse trees of all sizes. As baselines, we include a standard implementation of the parse tree kernel (PTK1) detailed by Shawe-Taylor and Cristianini (2004) and the improved variant (PTK2) proposed by Moschitti (2006a).

For each kernel, we estimate the average run-time and memory requirements by computing kernels between reference trees of fixed sizes and 100 randomly drawn trees. We also consider the worst-case scenario for each data set, which occurs if kernels are computed between identical parse trees, thus realizing the maximal number of matching node pairs. We focus in our experiments on

	ATK	*PTK2*	*Speed-up*
Training time on 1,000 parse trees			
Question classification	42s ±4	72s ±7	1.7×
Web spam detection	111s ±17	1,487s ±435	13.4×
Intrusion detection (HTTP)	123s ±20	349s ±80	2.8×
Intrusion detection (FTP)	125s ±14	517s ±129	5.8×
Testing time on 1,000 parse trees			
Question classification	40s ±4	70s ±2	1.8×
Web spam detection	112s ±18	1,542s ±471	13.8×
Intrusion detection (HTTP)	81s ±14	225s ±71	2.8×
Intrusion detection (FTP)	107s ±15	455s ±112	4.1×

Table 2: Training and testing time of SVMs using the exact and the approximate tree kernel.

the learning tasks of web spam and intrusion detection (HTTP), where results for question classification and FTP are analogous.



(a) Average-case run-time (WS)

(b) Worst-case run-time (WS)

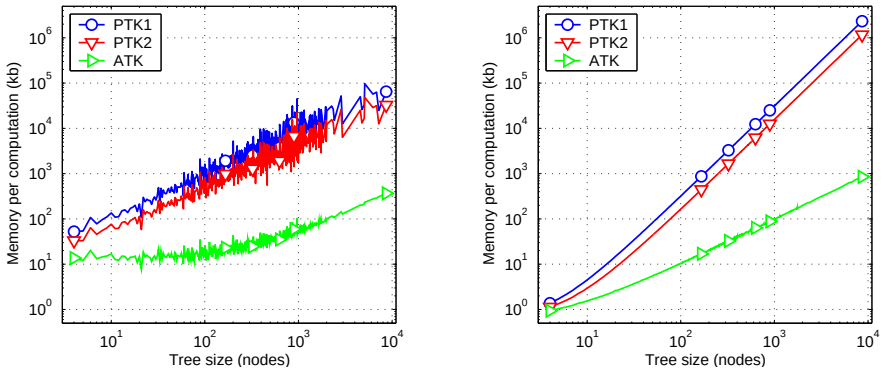
(c) Average-case run-time (ID)

(d) Worst-case run-time (ID)

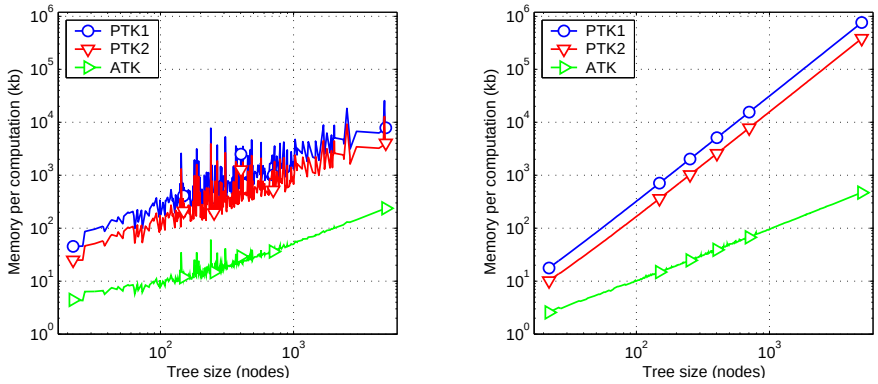
Figure 13: Run-times for web spam (WS) and intrusion detection (ID).

Figure 13 illustrates the run-time performance of the approximate and the two exact tree kernels. The run-time is given in milliseconds (ms) per kernel computation on the y-axis and the size of the

considered trees is shown on the x-axis. Both axes are presented in log-scale. Although the improved variant by Moschitti (PTK2) is significantly faster than the standard implementation, neither of the two show compelling run-times in both tasks. For both implementations of the regular tree kernel, a single kernel computation can take more than 10 seconds, thus rendering large-scale applications infeasible. By contrast, the approximate tree kernel computes similarities between trees up to three orders of magnitude faster and yields a worst-case computation time of less than 40 ms for the web spam detection task and less than 20 ms for the intrusion detection task. The worst-case analysis shows that the exact tree kernel scales quadratically in the number of nodes whereas the approximate tree kernel is computed in sub-quadratic time in the size of the trees.



(a) Average-case memory (WS) (b) Worst-case memory (WS)



(c) Average-case memory (ID)) (d) Worst-case memory (ID)

Figure 14: Memory requirements for web spam (WS) and intrusion detection (ID).

Figure 14 reports on average and worst-case memory requirements for the web spam detection and intrusion detection task. The memory consumption in kilobytes is depicted on the y-axis and the size of the considered trees is shown on the x-axis. Both axes are given in logarithmic scale. In all figures, the curves of the approximate kernel are significantly below the variants of the parse tree kernel. The allocated memory for the regular tree kernel exceeds 1 gigabytes in both learning tasks, which is clearly prohibitive for a single kernel computation. In contrast the approximate tree kernel requires at most 800 kilobytes. For the worst-case estimation, the memory consumption of the exact kernel scales quadratically in the number of tree nodes while the approximate tree kernel scales sub-quadratically due to the sparse selection of symbols.

6. Conclusions

Learning with large trees render regular parse tree kernels inapplicable due to quadratic run-time and memory requirements. As a remedy, we propose to approximate regular tree kernels. Our approach splits into a selection and an application stage. In the selection stage, the computation of tree kernels is narrowed to a sparse subset of subtrees rooted in appropriate grammar symbols. The symbols are chosen according to their discriminative power for supervised settings and to minimize the expected number of node comparisons for unsupervised settings, respectively. We derive linear programming approaches to identify such symbols, where the resulting optimization problems can be solved with standard techniques. In the subsequent application stage, learning algorithms benefit from the initial selection because run-time and memory requirements for the kernel computation are significantly reduced.

We evaluate the approximate trees kernels with SVMs as underlying learning algorithms for question classification, web spam detection and intrusion detection. In all experiments, the approximate tree kernels not only replicate the predictive performances of exact kernels but also provide concise representations by operating on only 2–10% of the available grammar symbols. The resulting approximate kernels lead to significant improvements in terms of run-time and memory requirements. For large trees, the approximation reduces a single kernel computation from 1 gigabyte to less than 800 kilobytes, accompanied by run-time improvements up to three orders of magnitude. We also observe improvements for parse trees generated for sentences in natural language, however, at a smaller scale. The most dramatic results are obtained for intrusion detection. Here, a kernel PCA shows that approximate tree kernels effectively identify relevant dimensions in feature space and discard redundant and noisy subspaces from the kernel computation. Consequently, the approximate kernels perform more efficiently and more accurately than their exact counterparts achieving AUC improvements of up to 20%.

To the best of our knowledge, we present the first efficient approach to learning with large trees containing thousands of nodes. In view of the many large-scale applications comprising structured data , the presented work provides means for efficient and accurate learning with large structures. Although we focus on classification, approximate tree kernels are easily leveraged to other kernel-based learning tasks, such as regression and clustering, using the introduced techniques. Moreover, the devised approximate tree kernels build on the concept of convolution over local kernel functions. Our future work will focus on transferring attained performance gains to the framework of convolution kernels, aiming at rendering learning with various types of complex structured data feasible in large-scale applications.

Acknowledgments

The authors would like to thank the anonymous reviewers for helpful comments and suggestions. Furthermore, we like to thank Patrick Düssel and René Gerstenberger for providing efficient implementations of network protocol parsers. The authors gratefully acknowledge the funding from the Bundesministerium für Bildung und Forschung under the project REMIND (FKZ 01-IS07007A) and from the FP7-ICT Program of the European Community under the PASCAL2 Network of Excellence, ICT-216886. Most of the work was done when UB was at TU Berlin.

References

C. Bockermann, M. Apel, and M. Meier. Learning SQL for database intrusion detection using context-sensistive modelling. In *Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA)*, 2009. to appear.

N. Borisov, D.J. Brumley, H. Wang, J. Dunagan, P. Joshi, and C. Guo. Generic application-level protocol analyzer and its language. In *Proc. of Network and Distributed System Security Symposium (NDSS)*, 2007.

M. L. Braun, J. Buhmann, and K.-R. Müller. On relevant dimensions in kernel feature spaces. *Journal of Machine Learning Research*, 9:1875–1908, Aug 2008.

C. Castillo, D. Donato, L. Becchetti, P. Boldi, S. Leonardi, M. Santini, and S. Vigna. A reference collection for web spam. *SIGIR Forum*, 40(2):11–24, 2006. URL `http://portal.acm.org/citation.cfm?id=1189703`.

C.-C. Chang and C.-J. Lin. LIBSVM: Introduction and benchmarks. Technical report, Department of Computer Science and Information Engineering, National Taiwan University, Taipei, 2000.

E. Charniak. A maximum-entropy-inspired parser. Technical Report CS-99-12, Brown University, 1999.

E. Cilia and A. Moschitti. Advanced tree-based kernels for protein classification. In *Artificial Intelligence and Human-Oriented Computing (AI*IA), 10th Congress*, LNCS, pages 218–229, 2007.

M. Collins and N. Duffy. Convolution kernel for natural language. In *Advances in Neural Information Proccessing Systems (NIPS)*, volume 16, pages 625–632, 2002.

N. Cristianini, J. Shawe-Taylor, A. Elisseeff, and J. S. Kandola. On kernel target alignment. In *Advances in Neural Information Proccessing Systems (NIPS)*, volume 14, pages 367–737, 2001.

I. Drost and T. Scheffer. Thwarting the nigritude ultramarine: Learning to identify link spam. In *Proc. of the Eurpoean Conference on Machine Learning (ECML)*, 2005.

P. Düssel, C. Gehl, P. Laskov, and K. Rieck. Incorporation of application layer protocol syntax into anomaly detection. In *Proc. of International Conference on Information Systems Security (ICISS)*, pages 188–202, 2008.

E. Eskin, A. Arnold, M. Prerau, L. Portnoy, and S. Stolfo. *Applications of Data Mining in Computer Security*, chapter A geometric framework for unsupervised anomaly detection: detecting intrusions in unlabeled data. Kluwer, 2002.

R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. RFC 2616 (Draft Standard), June 1999. URL `http://www.ietf.org/rfc/rfc2616.txt`. Updated by RFC 2817.

D. Haussler. Convolution kernels on discrete structures. Technical Report UCSC-CRL-99-10, UC Santa Cruz, July 1999.

J.E. Hopcroft and J.D. Motwani, R. Ullmann. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 2 edition, 2001.

H. Kashima and T. Koyanagi. Kernels for semi-structured data. In *International Conference on Machine Learning (ICML)*, pages 291–298, 2002.

D. Knuth. *The Art of Computer Programming*, volume 3. Addison-Wesley, 1973.

C. Kruegel and G. Vigna. Anomaly detection of web-based attacks. In *Proc. of 10th ACM Conf. on Computer and Communications Security*, pages 251–261, 2003.

P. Laskov, K. Rieck, and K.-R. Müller. Machine learning for intrusion detection. In *Mining Massive Data Sets for Security*, pages 366–373. IOS press, 2008.

X. Li and D. Roth. Learning question classifiers. In *International Conference on Computational Linguistics (ICCL)*, pages 1–7, 2002. doi: http://dx.doi.org/10.3115/1072228.1072378.

R. Lippmann, J.W. Haines, D.J. Fried, J. Korba, and K. Das. The 1999 DARPA off-line intrusion detection evaluation. *Computer Networks*, 34(4):579–595, 2000.

C. Manning and H. Schütze. *Foundations of Statistical Natural Language Processing*. MIT Press, 1999.

S. Mika, B. Schölkopf, A.J. Smola, K.-R. Müller, M. Scholz, and G. Rätsch. Kernel PCA and de–noising in feature spaces. In M.S. Kearns, S.A. Solla, and D.A. Cohn, editors, *Advances in Neural Information Processing Systems*, volume 11, pages 536–542. MIT Press, 1999.

A. Moschitti. Making tree kernels practical for natural language processing. In *Conference of the European Chapter of the Association for Computational Linguistics (EACL)*, 2006a.

A. Moschitti. Efficient convolution kernels for dependency and constituent syntactic trees. In *European Conference on Machine Learning (ECML)*, 2006b.

A. Moschitti and F.M. Zanzotto. Fast and effective kernels for relation learning from texts. In *International Conference on Machine Learning (ICML)*, 2007.

K.-R. Müller, S. Mika, G. Rätsch, K. Tsuda, and B. Schölkopf. An introduction to kernel-based learning algorithms. *IEEE Neural Networks*, 12(2):181–201, May 2001.

R. Pang, V. Paxson, R. Sommer, and L.L. Peterson. binpac: a yacc for writing application protocol parsers. In *Proc. of ACM Internet Measurement Conference*, pages 289–300, 2006.

V. Paxson and R. Pang. A high-level programming environment for packet trace anonymization and transformation. In *Proc. of Applications, Technologies, Architectures, and Protocols for Computer Communications SIGCOMM*, pages 339 – 351, 2003.

J. Postel and J. Reynolds. File Transfer Protocol. RFC 959 (Standard), October 1985. URL `http://www.ietf.org/rfc/rfc959.txt`. Updated by RFCs 2228, 2640, 2773, 3659.

K. Rieck and P. Laskov. Language models for detection of unknown attacks in network traffic. *Journal in Computer Virology*, 2(4):243–256, 2007.

K. Rieck, U. Brefeld, and T. Krueger. Approximate kernels for trees. Technical Report FIRST 5/2008, Fraunhofer Institute FIRST, September 2008.

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

B. Schölkopf, A.J. Smola, and K.-R. Müller. Nonlinear component analysis as a kernel eigenvalue problem. *Neural Computation*, 10:1299–1319, 1998.

B. Schölkopf, J. Platt, J. Shawe-Taylor, A.J. Smola, and R.C. Williamson. Estimating the support of a high-dimensional distribution. TR 87, Microsoft Research, Redmond, WA, 1999.

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

J. Suzuki and H. Isozaki. Sequence and tree kernels with statistical feature mining. In *Advances in Neural Information Proccessing Systems (NIPS)*, volume 17, 2005.

J. Suzuki, H. Isozaki, and E. Maeda. Convolution kernels with feature selection for natural language processing tasks. In *Annual Meeting on Association for Computational Linguistics (ACL)*, 2004.

D.M.J. Tax and R.P.W. Duin. Support vector domain description. *Pattern Recognition Letters*, 20 (11–13):1191–1199, 1999.

V.N. Vapnik. *The Nature of Statistical Learning Theory*. Springer, New York, 1995.

S.V.N. Vishwanathan and A.J. Smola. Fast kernels for string and tree matching. In *Advances in Neural Information Proccessing Systems (NIPS)*, pages 569–576, 2003.

E.M. Voorhees. Overview of the trec 2004 question answering track. In *Proc. of the Thirteenth Text Retreival Conference (TREC)*, 2004.

G. Wondracek, P.M. Comparetti, C. Kruegel, and E. Kirda. Automatic network protocol analysis. In *Proc. of Network and Distributed System Security Symposium (NDSS)*, 2008.

B. Wu and B. D. Davison. Identifying link farm spam pages. In *WWW '05: Special Interest Tracks, 14th International Conference on World Wide Web*, 2005.

D. Zhang and W. S. Lee. Question classification using support vector machines. In *Annual International ACM SIGIR Conference*, pages 26–32, 2003.