Route Kernels for Trees

Fabio Aiolli AIOLLI@MATH.UNIPD.IT
Giovanni Da San Martino DASAN@MATH.UNIPD.IT
Alessandro Sperduti SPERDUTI@MATH.UNIPD.IT
Department of Pure and Applied Mathematics, University of Padua, Via Trieste 63, 35121, Padua, Italy

Abstract

Almost all tree kernels proposed in the literature match substructures without taking into account their relative positioning with respect to one another. In this paper, we propose a novel family of kernels which explicitly focus on this type of information. Specifically, after defining a family of tree kernels based on routes between nodes, we present an efficient implementation for a member of this family. Experimental results on four different datasets show that our method is able to reach state of the art performances, obtaining in some cases performances better than computationally more demanding tree kernels.

1. Introduction

In many real world applications involving machine learning tasks, such as supervised classification, data is naturally represented in structured form, e.g. XML documents, molecular structures in chemistry, parse trees in natural language processing, and protein sequences in biology, just to name a few. Among all the learning techniques for dealing with structured data, kernel methods are recognized to have a strong theoretical background and to be effective approaches. On the other hand, designing fast and good kernels for structured data is a challenging problem.

In the context of tree structured data, the first proposed kernels were the subtree kernel (Vishwanathan & Smola, 2002) and the subset tree kernel (Collins & Duffy, 2002). The first defines a feature space consisting of the set of all proper subtrees, while the second enlarges this set by also considering subset trees. While the subtree kernel is computationally more efficient than the subset tree kernel, it is less expressive and usually returns worse performances than the subset tree kernel.

One problem of the above kernels is that, in the case of large structures and many symbols, the feature space implicitly defined by the above kernels is very sparse, and consequently classification performance is quite poor, as clearly discussed in (Suzuki & Isozaki, 2006). Because of that, alternative tree kernels have been proposed, some of which tailored to specific application domains. For example the elastic tree kernel has been introduced for XML and HTML data in (Kashima & Koyanagi, 2002). It extends the subset tree kernel by allowing matching between nodes with different labels, and by allowing matchings between substructures built by combining subtrees with their descendants.

Other extensions of the subset tree kernel have been proposed, e.g. (Moschitti, 2006) permits a partial matching between subtrees, (Zhang et al., 2007) defines a grammar-driven convolution tree allowing approximate substructure and tree node matchings according to a given grammar, and (Bloehdorn & Moschitti, 2007) introduces a family of kernels specifically designed for being used in text categorization tasks, called Semantic Syntactic Tree Kernels.

Furthermore, the Spectrum Tree Kernel, counting common tree q-grams, i.e. subtrees isomorphic to paths with q nodes, was described in (Kuboyama et al., 2007), while Nicotra, Micheli and Starita (2004) present an application of the Fisher kernel (Jaakkola & Haussler, 1999) to labeled rooted positional ρ-ary trees, exploiting Hidden Tree Markov Models (Diligenti et al., 2003) as generative models. Finally, in (Rieck et al., 2008) the expressivity of the kernel has been reduced in favor of efficiency, obtaining an approximated tree kernel with linear time complexity.

In general, for a tree kernel to be effective, a trade-off between expressivity of the feature space and low computational complexity has to be found. In this paper, we propose a new tree kernel that we believe is able to reach a good balance between these two goals. Specifically, we noticed that almost all the existing tree

Appearing in *Proceedings of the 26th International Conference on Machine Learning*, Montreal, Canada, 2009.

kernels tend to discard explicit information about the "relative positioning" of the nodes in a tree. Even kernels based on counting the number of common paths such as the Spectrum Tree Kernel do not take into account the relative positioning of the nodes in the original tree. Here we propose a family of tree kernels, based on the definition of routes between nodes of the tree, that exploits this kind of information. In particular, we focus on a member of this family that, while returning state of the art performances on various datasets, is computationally not too expensive.

2. Notation

In this paper we focus on *labeled rooted positional ρ-ary trees*, i.e. graphs such that (i) the vertices are connected by exactly one path (a tree); (ii) a node has been designed to be the *root* of the tree and the edges have a natural orientation, i.e. away from the root; (iii) each node, which is not a leaf node, has at most ρ children (maximum out-degree); (iv) a unique positional index $P_v[e] \in \{1, \ldots, \rho\}$ can be assigned to each edge e leaving from a vertex v; (v) a label l from a fixed alphabet is associated to each vertex, for example l_i is the label of vertex v_i. For readability, we will refer to labeled rooted positional ρ-ary trees simply as trees in the following. Given a tree, we denote by $\mathrm{ch}(v)$ the set of children (or directed successors) of v, by $\mathrm{ch}_k(v)$ the k-th child of v, and $\mathrm{ch}_k(v) = \perp$ (indefinite) when there are no edges in position k. Similarly, we denote by $\mathrm{pa}(v)$ the parent (or directed predecessor) of a vertex v and $\mathrm{pa}(root) = \perp$. Furthermore, $pa^i(v)$ will indicate the ancestor of v obtained by applying the operator $pa(\cdot)$ exactly i times (assuming $pa(\perp) = \perp$). We also define the operator $chpos(v)$ that returns the position of a node v with respect to its parent, i.e. $chpos(v) = i$ if $P_u[(u, v)] = i$ and $\mathrm{pa}(v) = u$. When $pa(v)$ is undefined, $chpos(v)$ is also undefined. Given any two nodes and the path $p(v_i, v_j)$ connecting them, the *label path* $\xi(v_i, v_j)$ is the sequence of labels associated to the nodes in $p(v_i, v_j)$. For example, in Figure 1, $\xi(v_a, v_e) = [a, b, e]$ is the label path connecting nodes v_a and v_e (labeled a and e respectively). A function $\delta(x, x')$ will be used for comparing two generic objects. Specifically, we have $\delta(x, x') = 1$ whenever objects x and x' are both defined and equal, and 0 otherwise.

3. Convolution Tree Kernels

Kernel algorithms, such as the Support Vector Machine (SVM), require the definition of a kernel function, i.e. a similarity function between any two elements of a domain. A common approach to design kernel functions for tree structured data is the *convolution kernel* framework firstly introduced by Haussler (1999), which is a general methodology for computing kernels on complex discrete objects. By splitting the original object into parts and assuming to have at disposal a positive semidefinite kernel on the parts (called local kernel), Haussler (1999) describes a methodology for combining the kernels on the parts that preserves positive semidefiniteness. In particular, he proved that the kernel obtained by summing the local kernels computed on the Cartesian product of the sets of subparts of the original structures, is positive semidefinite.
Recently, Shin and Kuboyama (2008) have introduced a more general form of convolution kernel called *mapping kernel*. Basically, the idea of the mapping kernel is to restrict the set of substructure pairs over which the local kernel is computed. Formally speaking, the mapping kernel is defined as follows. Let χ be the domain of trees and each $x \in \chi$ be associated with the finite subset χ'_x of substructures associated with x. Now, let assume to have at disposal a (local) positive semidefinite kernel $k : \chi' \times \chi' \to \mathbb{R}$. Then the mapping kernel is defined as:

$$K(x_i, x_j) = \sum_{(x'_i, x'_j) \in \mathcal{M}_{x_i, x_j}} k(x'_i, x'_j), \qquad (1)$$

where $\mathcal{M}$ is part of a mapping system $\mathbb{M}$ defined as:

$$\mathbb{M} = \Big(\chi, \Big\{\chi'_x | x \in \chi\Big\}, \Big\{\mathcal{M}_{x_i,x_j} \subseteq \chi'_{x_i} \times \chi'_{x_j} | x_i, x_j \in \chi\Big\}\Big).$$

$\mathbb{M}$ is a triplet composed by the domain of the examples, the space of the substructures of the examples, and a function $\mathcal{M}$ specifying for which pairs of substructures the local kernel has to be computed. $\mathcal{M}$ is assumed to be finite and symmetric, i.e. $\forall x_i, x_j \in \chi, |\mathcal{M}_{x_i,x_j}| < \infty$ and $(x'_i, x'_j) \in \mathcal{M}_{x_i,x_j} \Rightarrow (x'_j, x'_i) \in \mathcal{M}_{x_j,x_i}$. In (Shin & Kuboyama, 2008) it is proved that the kernel K of eq. (1) is positive semidefinite *if and only if* the mapping system $\mathcal{M}$ is *transitive* (i.e. $\forall x_1, x_2, x_3 \in \chi . (x'_1, x'_2) \in \mathcal{M}_{x_1,x_2} \wedge (x'_2, x'_3) \in \mathcal{M}_{x_2,x_3} \Rightarrow (x'_1, x'_3) \in \mathcal{M}_{x_1,x_3}$).
Tree kernels which are used as baselines for the experiments of this paper are now sketched. We start by recalling the *partial tree kernel* (PT) (Moschitti, 2006). Then we recall the *subset tree kernel* (SST) (Collins & Duffy, 2002) and the *subtree kernel* (ST) (Vishwanathan & Smola, 2002) as instances of PT. The PT kernel allows partial matching between subtrees (a subset of nodes of a tree which forms a tree). A recursive formulation can be given as $K(T_1, T_2) = \sum_{v_1 \in T_1} \sum_{v_2 \in T_2} C(v_1, v_2)$, where $C(v_1, v_2)$ can be recursively computed according to the following rule: If the productions[1] at v_1 and v_2 are different then

[1] A production is defined as the label of a node plus the labels associated to its children.

$C(v_1, v_2) = 0$; else

$$C(v_1, v_2) = \lambda\Big(\mu^2 + \sum_{J_1, J_2, |J_1|=|J_2|} \mu^{d(J_1)+d(J_2)} \cdot \prod_{i=1}^{|J_1|} (\sigma + C(chs_{v_1}[J_{1i}], chs_{v_2}[J_{2i}]))\Big) \quad (2)$$

where $J_{11}, J_{12}, J_{13}, \ldots$ $J_{21}, J_{22}, J_{23}, \ldots$ are index sequences associated with the ordered child sequences chs_{v_1} and chs_{v_2} respectively, J_{1i} and J_{2i} point to the i-th children in the two sequences, $|J_1|$ returns the length of the sequence J_1. $d(J_1) = J_{1|J_1|} - J_{11}$ and $d(J_2) = J_{2|J_2|} - J_{21}$. The parameter μ penalizes subtrees built on child subsequences that contain gaps. Note that in our formulation λ and μ are inverted with respect to (Moschitti, 2006) for being coherent with the semantic of λ in ST and SST. The partial tree kernel can be evaluated in $O(\rho^3|T_1||T_2|)$, where ρ is the maximum out-degree of the two trees and $|T|$ is the number of nodes of the tree T. The ST and SST kernels are a special case of the PT kernel. The SST kernel can be obtained back from eq. (2) by setting $\sigma = 1$, and considering only the contribution of the longest child sequence from node pairs with same children. The computational complexity in time of the SST kernel is $O(|T_1| \cdot |T_2|)$. The ST kernel can be obtained from the SST kernel by setting $\sigma = 0$. In (Vishwanathan & Smola, 2002) an efficient algorithm for computing the ST kernel is presented. Its computational complexity is $O(|T_i| \log |T_i|)$.

4. Generalized Route Kernel

This section formally describes the proposed Generalized Route Kernel. In particular, it will be gradually introduced by starting from a simple formulation, and then adding pieces with the aim of progressively enriching the feature space. In the end, we will obtain a kernel which is able to match set of routes ending up on nodes with the same label or production.

Let us start by introducing the concept of *route*. Intuitively, a route in a tree explicitly keeps information about the position of the nodes with respect to adjacent nodes.

Definition 1 (Route) *Let T be a (positional) tree, $v_1, v_2 \in T$ any two nodes in the tree, and $p(v_1, v_2) = [v_1, \ldots, v_2]$ the (shortest) path connecting v_1 and v_2 through the edges of T (not considering edge direction). Then the route from v_1 to v_2 in T, denoted by $\pi(v_1, v_2)$, is the sequence of indexes of edges connecting the consecutive nodes in the path $p(v_1, v_2)$. This indexes are taken positive (or negative) whenever edges are traversed away from (resp. towards) the root.*

Figure 1 gives an example of a tree and a route computed between nodes **a** and **e**. The nodes connected by dashed edges represent the shortest path connecting nodes **a** and **e**, i.e. $p(a, e)$. The route connecting nodes **a** and **e** is represented by the sequence $[2, 3]$, since node **b** is the second child of **a** and node **e** is the third child of **b**. The route connecting nodes **g** and **b** is $[-3, 2]$. Given any two trees, a first kernel can be

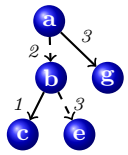

Figure 1. An example of a route connecting nodes labeled with **a** and **e**. The nodes connected by dashed edges are the ones comprising the path between the two nodes. The route is formed by the sequence $2, 3$ since node **b** is the second child of **a** and node **e** is the third child of **b**.

promptly defined by comparing the set of all routes in the trees:

$$K_\pi(T_1, T_2) = \sum_{v_i, v_j \in T_1} \sum_{v_l, v_m \in T_2} k_\pi((v_i, v_j), (v_l, v_m)),$$

where k_π is a local kernel defined on the routes, e.g. $k_\pi((v_i, v_j), (v_l, v_m)) = \delta(\pi(v_i, v_j), \pi(v_l, v_m))$. In this case, K_π counts the number of common routes of the two trees. It is straightforward to show that if k_π is a valid kernel then K_π is a valid kernel. Let χ be the set of trees, $\chi_T^{'}$ the set of routes in T,

$$\mathcal{M}_{T_1,T_2} = \chi_{T_1}^{'} \times \chi_{T_2}^{'} \quad (3)$$

the Cartesian product of the two sets of routes of T_1 and T_2, then K_π is an instance of the mapping kernel (see Section 3). $\mathcal{M}_{T_1,T_2}$ is clearly a transitive function and thus K_π is positive semidefinite.

In order to add expressiveness to the kernel we consider a different local kernel k_ξ defined on label paths, i.e. $k_\xi((v_i, v_j), (v_l, v_m)) = \delta(\xi(v_i, v_j), \xi(v_l, v_m))$. A combined local kernel can then be defined based on the product between k_π and k_ξ, i.e.

$$k_{\pi\xi}\left((v_i, v_j), (v_l, v_m)\right) = k_\pi((v_i, v_j), (v_l, v_m)) k_\xi((v_i, v_j), (v_l, v_m))$$

Finally, a further extension can simply be obtained by considering a polynomial kernel on the previously defined kernel. Interestingly, this would allow to count as features the simultaneous presence of groups of routes:

$$K_{gr}(T_1, T_2) = \left(\sum_{((v_i, v_j), (v_l, v_m)) \in \mathcal{M}_{T_1,T_2}} k_{\pi\xi}\left((v_i, v_j), (v_l, v_m)\right) + e \right)^d, \quad (4)$$

where $e \in \mathbb{R}$, $d \in \mathbb{N}^+$ and $\mathcal{M}$ is defined as in eq. (3).

5. An instantiation of the Generalized Route Kernel

In this section, we discuss an instance of the generalized route kernel and give for it an efficient implementation. Our purpose here is to obtain a kernel function matching identical pairs of type (π, l), where l is the label of the last node in the path associated to the route π. The following modifications to eq. (4) are needed. Since $k_{\pi\xi}$ is defined in terms of k_π and k_ξ, we start by modifying these two functions:

$$k_\pi((v_i, v_j), (v_l, v_m)) = \delta(\pi(v_i, v_j), \pi(v_l, v_m))\lambda^{|\pi(v_i, v_j)|}, \tag{5}$$

where $\lambda \in \mathbb{R}$ is a user defined parameter and $|\pi(v_i, v_j)|$ is the length of the route $\pi(v_i, v_j)$, which corresponds to the length of the corresponding path. k_π in eq. (5) let match only identical routes. The value of each match is weighted according to a value λ dependent on the length of the route. The basic idea motivating the introduction of the λ is to downweight the influence of larger routes in the same way as described for the ST and SST kernels in Section 3. The function k_ξ is modified as follows:

$$k_\xi((v_i, v_j), (v_l, v_m)) = \delta(l_j, l_m). \tag{6}$$

k_ξ tests if the labels of the last nodes in the two paths are identical. Note that the use of the kernel k_ξ could have been avoided by imposing the following condition:

$$((v_i, v_j), (v_l, v_m)) \in \mathcal{M}_{T_1,T_2} \Leftrightarrow l_j = l_m. \tag{7}$$

In the following, we also experiment an alternative definition for the kernel k_ξ based on the production at the last node of the path:

$$k_{prod}(\xi(v_i, v_j), \xi(v_l, v_m)) = \delta(prod(v_j), prod(v_m)), \tag{8}$$

where $prod(v)$ is the subtree rooted at node v and composed by all the children of v, only.
The route kernel, K_r, is thus the following (for simplicity the decay factor λ is omitted):

$$K_r(T_1, T_2) = \left(\sum_{((v_i,v_j),(v_l,v_m)) \in \mathcal{M}_{T_1,T_2}} \delta(\pi(v_i, v_j), \pi(v_l, v_m)) + e \right)^d. \tag{9}$$

We further restrict the set of feasible routes by imposing the following condition to the sets $\chi_{_T}^{'}$:

$$\chi_{_T}^{'} = \left\{ p(v_i, v_j) | v_i, v_j \in T \wedge v_j \in \overset{v_i}{\triangle} \right\}, \tag{10}$$

where $v_j \in \overset{v_i}{\triangle}$ means that v_j is a descendant of v_i or v_i itself. In other words routes are allowed only between a node and its descendants or the node itself.
Given the definition of $\chi_{_T}^{'}$ in eq. (10), a route $\pi(v_i, v_j)$ can be recursively defined as:

$$\pi(v_i, v_j) = \begin{cases} \pi\left(v_i, pa(v_j)\right).chpos(v_j) & \text{if} \quad v_i \neq v_j \\ \epsilon & \text{if} \quad v_i = v_j \end{cases}$$

where v_i, v_j are nodes of a tree T, and the "." operator creates a sequence from two list of objects, and ϵ is a symbol for the empty sequence.
Since matches are allowed only with identical routes, a node v, at depth o, in the tree has associated at most o non null features: one feature related to the path composed only by the same node, one feature related to the path $p(pa(v), v)$, one feature related to the path $p(pa(pa(v)), v) = p(pa^2(v), v)$, and so on until the feature related to the path connecting the root of the tree to the node: $p(pa^o(v), v)$. The total number of non null features for a tree with $|T|$ nodes is less than or equal to $\sum_{i=1}^{|T|} depth(v_i) = avgdepth(T) \cdot |T|$, where $depth(v)$ is the depth of node v and $avgdepth(T)$ is the average depth of a node in T. Note that the total number of non null features is equal to $avgdepth(T) \cdot |T|$ when the labels of the nodes in T are all different. Figure 2 gives an example of the set of features associated with a simple tree. In the figure, features are grouped according to their length s.

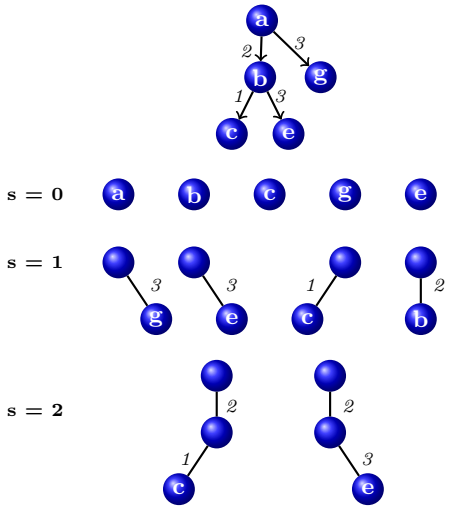

Figure 2. A tree (top) and its set of features according to the route kernel defined in eq. (9). Features are grouped by their length s.

5.1. Implementation

We now turn our attention to an efficient implementation of the kernel proposed in eq. (9) and eq. (10).

Without loss of generality it is assumed that parameters e and d are set to 0 and 1, respectively. Since no routes of different length can match, the definition of $\mathcal{M}$ given by eq. (7) and eq. (10), can be rewritten as

$$\mathcal{M}_{T_1,T_2} = \bigcup_{s=0}^{o} \mathcal{M}^{(s)}_{T_1,T_2},$$

where o is the smallest of the maximum depth of the trees T_1 and T_2. $\mathcal{M}^{(s)}_{T_1,T_2}$ is defined as:

$$((v_i, v_j), (v_l, v_m)) \in \mathcal{M}^{(s)}_{T_1,T_2} \Leftrightarrow \\ l_j = l_m \wedge |\pi(v_i, v_j)| = |\pi(v_l, v_m)| = s.$$

Clearly $\forall i, j, i \neq j, T_1, T_2 . \mathcal{M}^{(i)}_{T_1,T_2} \cap \mathcal{M}^{(j)}_{T_1,T_2} = \emptyset$. Eq. (9) can be rewritten as

$$K_r(T_1, T_2) = \left(\sum_{((v_i,v_j),(v_l,v_m)) \in \mathcal{M}_{T_1,T_2}} \delta(\pi(v_i, v_j), \pi(v_l, v_m)) \right) =$$

$$\sum_{s=0}^{o} \left(\sum_{((pa^s(v_j), v_j), (pa^s(v_m), v_m)) \in \mathcal{M}^s_{T_1,T_2}} \delta\left(\pi(pa^s(v_j), v_j), \pi(pa^s(v_m), v_m)\right) \right) =$$

$$\sum_{s=0}^{o} \left(\sum_{\substack{v_j \in T_1 \\ v_m \in T_2}} C^{(s)}(v_j, v_m) \right), \qquad (11)$$

where $C^{(s)}(v_i, v_j)$ can be computed according to the following rules: *i)* if $s = 1$ then $C^{(s)}(v_i, v_j) = \delta(v_i, v_j)$; *ii)* if $s > 1$ then $C^{(s)}(v_i, v_j) = C^{(s-1)}(v_i, v_j)\delta(chpos(pa^s(v_i)), chpos(pa^s(v_j)))$.

Eq. (11) suggests a strategy for computing the route kernel: by starting from the set of common labels of the two trees, identical routes of increasing length can be looked for. Clearly, since a common route of length s can have a match only if its subroute of length $s-1$ has a match, the proposed strategy can be stopped as soon as no more routes of current length are found.

The algorithm we propose (algorithm 1) assumes to treat trees with arbitrary but finite out-degree. It starts by computing the number of nodes with the same labels. It then proceeds by comparing the routes of length s, with s going to from 1 to the minimum of the maximum of the depths of the two trees. However, when no common routes of length i are found, the algorithm stops and avoids looking for routes of length greater than i. In the following the behavior of the algorithm is analyzed in detail.

Algorithm 1 Pseudo-Code for computing $K_r(T_1, T_2)$

```
    Input: trees T1 and T2, Kr parameter λ.
    k = 0; maxout =max{outdegree(T1),outdegree(T2)};
    List1 = sort {v ∈ T1}; //nodes sorted according to their labels
    List2 = sort {v ∈ T2}; //nodes sorted according to their labels
 5: for j = 1 to 2 do
      vj= 0; // vector of dimension ρ
      for all v ∈ Listj do
        v.label= l(v); vj += v.label;
      end for
10: end for
    k += λ v1^T v2; λ2 = λ · λ;
    create array F[] of dimension maxout;
    while |List1| > 0 ∧ |List2| > 0 do
      for i = 1 to maxout do
15:     v1,i= 0, F[1,i] = 0;
      end for
      for all node ∈ List1 do
        F[chpos(v)]= 1; v1,chpos(v) += v.label;
      end for
20:   for i = 1 to maxout do
        v2,i= 0; F[2,i] = 0;
      end for
      for all v ∈ List2 do
        if F[1,chpos(v)]= 0 then
25:       remove v from List2;
        else
          F[2,i] = 1;v2,chpos(v) += v.label;
          substitute v with w s.t. w ≡ pa(v) and w.label= v.label;
        end if
30:   end for
      for all v ∈ List1 do
        if F[2,chpos(v)]= 0 then
          remove v from List1;
        else
35:       substitute v with w s.t. w ≡ pa(v) and w.label= v.label;
        end if
      end for
      for i = 1 to maxout do
        k += λ2 v1,i^T v2,i;
40:   end for
      λ2 = λ2 · λ;
    end while
    Return k;
```

Lines from 1 to 4 initialize internal variables and create a sorted list of nodes for each tree. The procedure costs $O(|T| \log |T|)$. Lines from 5 to 11 compute the number of matchings due to the routes of length 1, i.e. the number of common labels. The computational complexity of the procedure is $O(\rho|T|)$. Line 12 creates an array which will contain information about the matchings between routes at current level. Its cost is $O(maxout)$. Each while iteration (lines 13 to 42) costs $O(|T|)$, since L lists may not contain more than $O(|T|)$ elements and for each list $O(1)$ operations are performed. Lines $14-16$ have a computational complexity of $O(maxout)$. However, it can be skipped by making use of a variable and an array of the same size of F. The variable, say t, is initially set to 0 and it is incremented every time the while loop is entered. Whenever F is written to (lines 18 and 27), the current value of t is recorded in the auxiliary array, say F'. When values from F are read (lines 24 and 32), they are considered valid if the corresponding value of F' is equal to t, otherwise the read operation returns a value of 0. Each while iteration counts the number of common routes of length s. Initially $s = 1$; If at

least one common route of length s is found (there is a non empty list L), then, in the next while iteration, routes of length $s+1$ are looked for. Clearly, there are no more than the length of the longest path in the smallest tree, i.e. min(maxdepth(T_1),maxdepth(T_2)), of such paths. By summing up the cost of all lines, the total cost of the algorithm in the worst case becomes: $O(|T| * avgdepth(T) + |T| \log |T|)$. Note that the average depth of a node can be at most $O(|T|)$, thus the complexity of the algorithm can be at most $O(|T|^2)$.

5.2. Relationship with other Kernels

It is known that, with respect to the feature space, $ST \subseteq SST \subseteq PT$. Given a tree T, ST associates to T at most a linear number of non-null features. SST and PT, with respect to T, have at most an exponential number of non-null features. The number of non-null features of the Route kernel is at most $avgdepth(T)|T|$. The features space of the Route kernel is not directly comparable with the one of the PT kernel. However if all labels of a tree were identical, the feature space of the Route kernel would properly be included into the feature space of the PT kernel. This is not the common case: two matching nodes for the Route kernel having different ascendants would not match for the PT kernel. Regarding computational complexity, ST is faster than SST, which in turn is faster than PT. The Route kernel has a lower computational complexity than the PT kernel and, in the worst case, the same quadratic complexity of the SST. Note that, when $avgdepth(T)$ is $O(1)$, the computational complexity of the Route kernel is equivalent to the one of the ST kernel. The sparsity of the kernel, with respect to a dataset, has been analyzed by considering the percentage of null entries of the Gram matrix. The sparsity of ST (those of SST are similar) is 54.71% on INEX 2005, 0.0024% on INEX 2006, 96.12% on LOGML, 44.45% on Propbank. The sparsity of the partial tree kernel is 0% for all datasets but Propbank (0.4388%). The sparsity of the route kernel, we consider here the version giving best results on each dataset, is 45.64% on INEX 2005, 44.50% on Propbank and 0% on all other datasets. Apart from INEX 2005 and Propbank, the sparsity of the route kernel is identical to the one of the partial tree. Thus, sparsity alone can not explain the difference in accuracy (see Section 6). We hypothesize that this different is due to the nature of the feature space of the route kernels.

6. Experiments

Experiments were performed to test the effectiveness of the proposed kernel with respect to ST, SST, the polynomial version of SST (obtained by exponentiating the kernel value in the same way as described for the route kernel in eq. (4)) and the PT kernel. The implementation of these kernels is available as part of the SVM-Light software[2]. Our approach has been tested on the INEX 2005 dataset (Section 6.1), the INEX 2006 dataset (Section 6.2), on part of the Propbank dataset (Section 6.3), and the LOGML dataset (Section 6.4). In some cases the training procedure was stopped due to excessive training times. Specifically we set a 24 hours time out for each single learning procedure. The time out was necessary because of the number of parameters evaluated.

6.1. Experiments on INEX 2005

We have used a modified version of the 2005 INEX Competition dataset(Denoyer & Gallinari, 2007), processed in order to reduce its maximum out-degree. The 2005 INEX Competition corpus we have used is the (`m-db-s-0`), which consists of 9,640 documents containing XML tags only. All documents belong to one out of 11 possible classes. For the learning phase, we used 3397 documents as training examples, while 1423 documents constitute the validation set. All remaining documents form the test set. For each setting of the hyper-parameters, SVM-based multi-class classification was performed by using the one-against-all methodology. Best hyper-parameters for the different methods were selected comparing classification accuracies on the validation set. The model related to the best hyper-parameters setting was trained on the union of the train and validation sets, and finally tested on the test set.
Only a subspace of the parameters of the kernels were evaluated. Specifically, experiments were performed with both normalized and not normalized kernels. The e parameter in eq. (9) was set to 0 in all experiments. The parameter d in eq. (9) was set to $1, 2, 3$. The PT kernel has an additional parameter, μ, which has been set to $0.1, 0.5, 1.0$. Route kernel experiments were carried out with the local kernel defined on node labels, eq. (6) and with the local kernel defined on productions, eq. (8). For each combination of the previous parameters, the λ and c parameter of the SVM were selected in validation among the values: $\lambda = \{0.05, 0.1, 0.25, 0.50, 0.75, 1.0, 2.0\}$ and $c = \{0.001, 0.01, 0.1, 1, 10, 100, 1000\}$. Table 1 summarizes the results obtained. The PT kernel has lowest classification error, 2.96%, while the route kernel with local kernel defined on labels places second with 3.06% classification error. A significance test shows that the difference between PT and route accuracies

[2] http://www.math.unipd.it/∼dasan/routekernel.htm

Table 1. Comparison between the classification error of ST, SST, the polynomial SST, the Partial Tree kernel and the Route kernel, respectively. Data refer to four different datasets: INEX 2005 and 2006, Propbank (sec. 23 and 24), and LOGML. The columns represent, respectively, the type of kernel, the lowest classification error on validation, the corresponding classification error on the test set. The last column refers to 3-fold cross-validation error over the LOGML dataset.

Kernel Type	INEX 2005		INEX 2006		Propbank		LOGML
	validation error %	test error %	validation error %	test error %	validation error %	test error %	cross-valid. error %
ST	13.15	11.27	68.32	67.98	5.43	5.71	16.72
SST	12.79	11.21	56.55	59.56	4.65	4.62	16.84
Polynomial SST	12.09	10.67	55.55	59.88	4.65	4.62	16.82
Partial Tree	2.96	2.96	57.83	58.87	4.71	4.55	16.40
Route, $k_\xi \equiv$ eq. (6)	3.10	3.06	58.72	59.94	5.07	4.90	16.20
Route, $k_\xi \equiv$ eq. (8)	3.31	3.52	55.55	58.09	4.92	4.76	16.79

is not significant.

6.2. Experiments on INEX 2006

The INEX 2006 dataset (Denoyer & Gallinari, 2007) is derived from the IEEE corpus composed of 12000 scientific articles from IEEE journals in XML format. It includes XML formatted documents, each from one of 18 different journals. In this case the training, validation and test sets consisted of 4237, 1816 and 6054 documents, respectively. Each document belongs to 1 out of 18 classes. By applying the same methodology as in the previous experiment and keeping the same experimental setting, the results summarized in Table 1 were obtained.
The lowest classification error is obtained by the route kernel with local kernel defined on node productions, 58.09%. The partial tree kernel reaches a 58.17%. Other kernels have a classification error going from 1.47% worse than the route kernel to 9.89%. The INEX 2006 is a harder task than INEX 2005. Nonetheless the route kernel, in conjunction with the local kernel defined on productions, is able to improve on the classification error of all the other techniques we compared to. A significance test shows that the improvement is indeed significant.

6.3. Experiments on Propbank

The Propbank dataset (Kingsbury & Palmer, 2002) is derived from a set of Dow-Jones news articles. It proposes argument structures to encode shallow semantics from texts. Only verbs are considered as predicates whereas arguments are labeled sequentially from Arg0 to Arg5 plus ArgMs including several type of adjuncts. The corpus is divided into sections. In order to reduce the computational complexity of the task, we derived the training (75314 examples) and validation (40495 examples) sets from section 24 and the test set from section 23 (184273 examples). The task is a binary classification problem. By applying the same methodology as in the previous experiment and keeping the same experimental setting, the results summarized in Table 1 were obtained.
The lowest classification error is obtained by the PT kernel, 4.55%. The route kernel with local kernel defined on node productions reaches a 4.76% and places 4th.

6.4. Experiments on LOGML

The LOGML dataset consists of user sessions of the Rensselaer Polytechnic Institute Computer Science Department website[3], collected over a period of three weeks. Each user session consists of a graph and contains the websites a user visited on the Computer Science domain. These graphs were transformed to trees by only enabling forward edges starting from the root node. The goal of the classification task is to discriminate between users who come from the edu domain and users from other domains, based upon the users browsing behavior. Three datasets are available. They comprises 8074, 7409 and 7628 examples, respectively. The maximum out-degree of the trees is 137.
Because of the availability of the three datasets, it was natural to compute the classification error of the kernels by performing a 3-fold cross-validation considering, in each round, one of the dataset as the test set. The set of parameters involved is the same as the one for the INEX 2005 experiments (see Section 6.1). Table 1 summarizes the result obtained. The values listed are the mean of the classification error on the three folds. Note that the Route kernel with local kernel defined on node labels has the lowest mean classification

[3] http://www.cs.rpi.edu

error, 16.20%.

7. Conclusions

We have proposed Tree Route kernels, a new family of tree kernels based on the definition of route between nodes. A member of this family where routes are matched if and only if the label of the arriving node is the same, has been studied in detail. The computational complexity of the procedure for computing the kernel is $O(avgdepth \cdot |T| + |T| \log(|T|))$. Experimental results obtained on supervised classification for four non-trivial datasets have shown that the feature space induced by the proposed kernel is rich enough to give state of the art performances with respect to the most widely used tree kernels. The proposed kernel is thus able to reach quite good results while keeping a reasonable computational complexity. In addition to that, the proposed algorithm for computing the kernel can be easily adapted to parallel computation. In fact, each tread of computation could take responsibility for computing the contribution to the kernel given by the matchings between routes that end up on specific nodes (of the two trees) with identical label. The same approach cannot be applied to the other kernels because of the strong dependencies among nodes.

References

Bloehdorn, S., & Moschitti, A. (2007). Structure and semantics for expressive text kernels. *Proceedings of the sixteenth ACM Conference on Information and Knowledge Management* (pp. 861–864). Lisbon, Portugal.

Collins, M., & Duffy, N. (2002). New ranking algorithms for parsing and tagging: Kernels over discrete structures, and the voted perceptron. *Proceedings of the Fortieth Annual Meeting on Association for Computational Linguistics* (pp. 263–270). Philadelphia, PA, USA.

Denoyer, L., & Gallinari, P. (2007). Report on the XML mining track at INEX 2005 and INEX 2006: categorization and clustering of XML documents. *ACM SIGIR Forum*, *41*, 79–90.

Diligenti, M., Frasconi, P., & Gori, M. (2003). Hidden tree markov models for document image classification. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *25*, 519 – 523.

Haussler, D. (1999). *Convolution kernels on discrete structures* (Technical Report UCSC-CRL-99-10). University of California, Santa Cruz.

Jaakkola, T. S., & Haussler, D. (1999). Exploiting generative models in discriminative classifiers. *Proceedings of the 1998 conference on Advances in Neural Information Processing Systems II* (pp. 487–493). Cambridge, MA, USA: MIT Press.

Kashima, H., & Koyanagi, T. (2002). Kernels for semi-structured data. *Proceeding of the International Conference on Machine Learning* (pp. 291–298).

Kingsbury, P., & Palmer, M. (2002). From Treebank to PropBank. *Proceedings of the 3rd International Conference on Language Resources and Evaluation* (pp. 1989–1993). Las Palmas, Spain.

Kuboyama, T., Hirata, K., Kashima, H., Aoki-Kinoshita, K. F., & Yasuda, H. (2007). A spectrum tree kernel. *Information and Media Technologies*, *2*, 292–299.

Moschitti, A. (2006). Efficient convolution kernels for dependency and constituent syntactic trees. *Proc. of the European Conference on Machine Learning* (pp. 318–329).

Nicotra, L., Micheli, A., & Starita, A. (2004). Tree fisher kernel. *Proceedings. 2004 IEEE International Joint Conference on Neural Networks* (pp. 1917 – 1922).

Rieck, K., Brefeld, U., & Krger, T. (2008). *Approximate kernels for trees* (Technical Report). Fraunhofer Publica [http://publica.fraunhofer.de/oai.har] (Germany).

Shin, K., & Kuboyama, T. (2008). A generalization of haussler's convolution kernel: mapping kernel. *Proceeding of the International Conference on Machine Learning* (pp. 944–951).

Suzuki, J., & Isozaki, H. (2006). Sequence and tree kernels with statistical feature mining. In Y. Weiss, B. Schölkopf and J. Platt (Eds.), *Advances in neural information processing systems 18*, 1321–1328. Cambridge, MA: MIT Press.

Vishwanathan, S., & Smola, A. J. (2002). Fast kernels on strings and trees. *Proceedings of Neural Information Processing Systems 2002* (pp. 569–576).

Zhang, M., Che, W., Aw, A., Tan, C. L., Zhou, G., Liu, T., & Li, S. (2007). A grammar-driven convolution tree kernel for semantic role classification. *Proc. of the 45th Annual Meeting of the Association for Computational Linguistics* (pp. 200–2007).