Extending Tree Kernels with Topological Information

Fabio Aiolli, Giovanni Da San Martino, and Alessandro Sperduti

Dept. of Pure and Applied Mathematics, University of Padova,
via Trieste 63, 35121, Padova, Italy
{aiolli,dasan,sperduti}@math.unipd.it

Abstract. The definition of appropriate kernel functions is crucial for the performance of a kernel method. In many of the state-of-the-art kernels for trees, matching substructures are considered independently from their position within the trees. However, when a match happens in similar positions, more strength could reasonably be given to it. Here, we give a systematic way to enrich a large class of tree kernels with this kind of information without affecting, in almost all cases, the worst case computational complexity. Experimental results show the effectiveness of the proposed approach.

Keywords: kernel methods; tree kernels; machine learning;

1 Introduction

Kernel based methods are recognized to be very effective methods to cope with data in non vectorial form. Indeed, many real world applications exist where data are more naturally represented in structured form, including XML documents for information retrieval tasks, protein sequences in biology, and parse trees in natural language applications.

The design of this type of kernels is still a challenging problem as they should be expressive enough (avoiding the loss of relevant structural information) while remaining computationally not too demanding.

In this paper we focus on kernels for trees, for which several kernels have been defined in the last few years. As an example, consider the Subtree kernel (ST) [10] and the Subset tree kernel [2]: the former counts the number of matching proper subtrees and the latter kernel extends this space by also considering all subset trees. We noted that possibly relevant topological information about the relative position in the trees of the matching substructures is not typically taken into account in state-of-the-art kernels. In fact, common kernels for trees can be considered position invariant kernels, that is, features represent parts of the tree but do not maintain information about the position of the features in the original tree. Our intuition is that a more satisfactory notion of similarity on trees should give higher values to those structures which present the same features also in the same positions. One example of a kernel that incorporates this kind of information has been recently proposed in [1]. However, both the type of topological information and the local kernels involved are different from the ones of the present paper (see section 3 for details).

Here we propose an operator which is applicable to a family of kernels for trees and is able to enrich these kernels with topological information while maintaining (with one exception) the same computational complexity in time. Experimental results obtained by this operator demonstrate its ability to improve the performance of baseline kernels on various datasets when topological information is really relevant for the domain at hand. More importantly, enriching fast tree kernels (such as the ST and SST kernels) allow them to reach accuracy values comparable to the ones of slower but more expressive tree kernels, such as the Partial Tree Kernel (PT) [7] even if enriched ST and SST kernels are faster to compute than PT.

The paper is organized as follows: section 2 gives a brief survey of kernels for trees. Section 3 describes an operator for extending tree kernels with topological information. Section 4 explains how to efficiently compute the novel kernels. Section 5 gives experimental evidence of the effectiveness of the extended kernels. Section 6 draws some conclusions and propose future extensions of the paper.

2 Kernels for Trees

This section describes some well known kernels for trees focusing on three of them which will be used as baselines in the experimental section: ST [10], SST [2] and PT [7] kernels. These kernels are all based on counting the number of parts (or substructures) which are shared by two trees. However, different kernels define in a different way the type of substructures that can be matched. Let us briefly present the them in decreasing order of expressivity. The PT kernel counts the number of matching *subtrees*, i.e. subsets of nodes of a tree (and edges that link them) which form a tree. The SST counts the number of matching *subset trees*, where subset trees are subtrees for which the following constraint is satisfied: either all of the children of a node belong to the subset tree or none of them. The ST kernel counts the number of matching *proper subtrees*, where proper subtrees are here defined as subtrees rooted at a node v and comprising all of its descendants. Note that all of the above kernels are members of the convolution kernel framework [4], that is they can be computed resorting to the following formula:

$$K(T_1, T_2) = \sum_{v_1 \in T_1} \sum_{v_2 \in T_2} C_{\mathcal{K}}(v_1, v_2), \tag{1}$$

where $\mathcal{K} \in \{ST, SST, PT\}$ (see below). $C(_{\mathcal{K}})$ can be computed according to three rules: *i*) if the productions[1] at v_1 and v_2 are different then $C_{\mathcal{K}}(v_1, v_2) = 0$; *ii*) if the productions at v_1 and v_2 are the same, and v_1 and v_2 have only leaf children (i.e. they are pre-terminals symbols) then $C_{\mathcal{K}}(v_1, v_2) = \lambda$, where λ is an external parameter which causes a downweighting of the influence of larger substructure matches; *iii*) if the productions at v_1 and v_2 are the same, and v_1

[1] A production is defined as the label of a node plus the labels of its children (if any).

and v_2 are not pre-terminals, then the value of $C_{\mathcal{K}}()$ depends on the kernel. If $\mathcal{K} = \mathcal{ST}$ then $C_{ST}(v_1, v_2) = \lambda \prod_{j=1}^{nc(v_1)} (C(ch_j[v_1], ch_j[v_2]))$, where $nc(v_1)$ is the number of children of v_1 and $ch_j[v]$ denotes the j-th child of node v. If $\mathcal{K} = \mathcal{SST}$ then $C_{SST}(v_1, v_2) = \lambda \prod_{j=1}^{nc(v_1)} (1 + C(ch_j[v_1], ch_j[v_2]))$. The computational complexity in time of the above kernels is $O(|T_1||T_2|)$, where $|T_i|$ is the number of nodes of the tree T_i. Nevertheless, a faster algorithm for computing the ST kernel has also been proposed in [10] with complexity in time $O(N \log N)$, where $N = \max\{|T_1|, |T_2|\}$. Finally, for the PT kernel we have a slightly more complex formulation, i.e.

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

where $J_{11}, J_{12}, \ldots\ J_{21}, J_{22}, \ldots$ are sequences of indexes associated with the ordered sequences of children chs_{v_1} and chs_{v_2} respectively, J_{1i} and J_{2i} point to the i-th child in the two sequences and $|J_1|$ denotes the length of the sequence J_1. Finally, $d(J_1) = J_{1|J_1|} - J_{11}$ and $d(J_2) = J_{2|J_2|} - J_{21}$ (see [7] for details). The parameter μ penalizes subtrees built on subsequences of children that contain gaps. From this formulation one can see that the ST and SST kernels can be seen as special cases of the PT kernel. 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.

3 Injecting Positional Information into Tree Kernels

Tree kernels, such as ST, SST, and PT, are position invariant kernels, i.e. any match between two subtrees $t_1 \in T_1$ and $t_2 \in T_2$ does not consider where t_1 and t_2 occur within T_1 and T_2, respectively. While this feature may turn useful to avoid too sparse kernels (kernels in which matches barely occur), it may generate an unsatisfactory kernel matrix from the point of view of structural similarity. This point is illustrated in Fig. 1 where $K_{ST}(T_1, T_3) = K_{ST}(T_2, T_3)$, while clearly a better representation of the similarity among the trees would prescribe the constraint $K(T_1, T_3) > K(T_2, T_3)$ to hold. Then, a nice tradeoff aiming at using

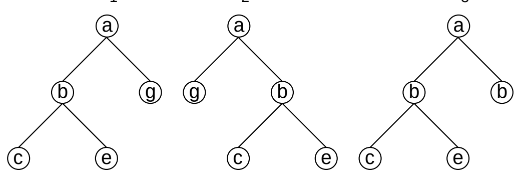

Fig. 1. Using the ST kernel we have $K_{ST}(T_1, T_3) = K_{ST}(T_2, T_3) = 3$, however it seems to be more reasonable to have $K(T_1, T_3) > K(T_2, T_3)$ since the matching subtrees (i.e. the leaf labeled **c**, the leaf labeled **e**, and the subtree **b(c,e)**) occur in the same positions within T_1 and T_3, while this is not the case for T_2.

this topological information while keeping low the sparsity of a kernel could be to extend the original feature space with new positional dependent features. We refer to the extended version of the kernels by applying the prefix PAK (Position Aware Kernel) to their names: PAK-ST, PAK-SST, PAK-PT. One simple way to do that is to define a kernel $K()$ which is the sum of local kernel $k()$ evaluations obtained for each pair of subtrees of the two trees sharing the same route. A route for a node $v \in T$, denoted by $\pi(v)$, is the sequences of indices of edges connecting the consecutive nodes in the path between $root(T)$, the root of the tree T, and v (for a more detailed description refer to [1]). The index of an edge is its position with respect to its siblings. The idea of the PAK extension is exemplified in Fig. 2 where the same color in the two trees correspond to those subtrees for which the kernel $k()$ have to be computed, i.e. those sharing the same route. Just to give an example, if we refer to trees T_1 and T_3 in Fig. 1, by

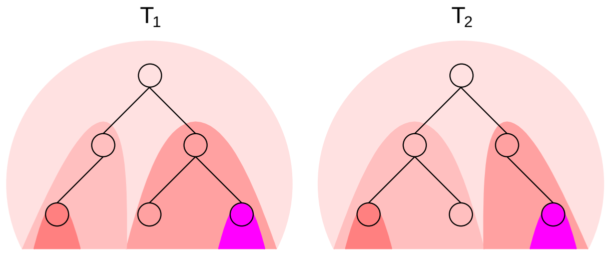

Fig. 2. A representation of which kernels $k()$ have to be computed to evaluate the kernel $K()$. Subtrees for which the kernel $k()$ is computed are the ones of the same color.

using the ST kernel K_{ST}, we would have

$$K(T_1, T_3) = K_{ST}(\mathbf{c}, \mathbf{c}) + K_{ST}(\mathbf{e}, \mathbf{e}) + K_{ST}(\mathbf{b(c,e)}, \mathbf{b(c,e)}) + K_{ST}(\mathbf{g}, \mathbf{b}) + \\ + K_{ST}(\mathbf{a(b(c,e),g)}, \mathbf{a(b(c,e),b)}). \tag{2}$$

Each route has associated its depth $d(v)$ where $d(root(T)) = 1$ and $d(v)$ is $1 + d(parent(v))$. The formal definition of the kernel is the following:

$$K_k(T_1, T_2) = \sum_{v_1 \in T_1, v_2 \in T_2} \gamma^{d(v_1)} \delta\left(\pi(v_1), \pi(v_2)\right) k(t_{v_1}, t_{v_2}), \tag{3}$$

where $\delta()$ is a function whose value is 1 whether the two input routes are identical, 0 otherwise. When $\gamma = 0$ computing eq. (3) is equivalent to computing the baseline kernel.

Eq. (3) considers topological similarity while avoiding to have a too sparse kernel since the computation of the position invariant kernel between the two trees is included. Moreover, from a computational point of view, the repeated computation of the same kernel on subtrees can be done efficiently by reusing the already performed calculations. The kernel of eq. (3) is different from the ones described in [1] in both the local kernel (simple kernels on the nodes in [1] compared to tree kernels in this paper) and the kernel on the route considered. Fig. 2 shows that the local kernel k between two subtrees is computed as many times as the length of the longest common prefix of their routes. For example in Fig. 1 the longest common prefix between the routes of nodes $\mathbf{e} \in T_1$ and $\mathbf{e} \in T_3$ is 2, the one between nodes $\mathbf{e} \in T_1$ and $\mathbf{e} \in T_2$ is 0, the one between nodes $\mathbf{e} \in T_1$ and $\mathbf{c} \in T_3$ is 1. On the contrary, in [1] the computation of the local kernel is repeated as many times as the length of the longest common suffix of the routes.

4 Algorithmic Issues

This section describes how to efficiently implement the kernel described in eq. 3. In fact, we will see that when using the SST or PT kernel as the "local" kernel k, the computational complexity of the extended kernel, i.e. eq. (3), is the same as the one of the local kernel. Unfortunately, the same idea would alter the complexity of ST, thus for the moment we'll focus on SST and PT.

Considering eq. (3), there are at most $n = \min(|T_1|, |T_2|)$ routes for which $\delta\left(\pi_{T_1}(v_1), \pi_{T_2}(v_2)\right) = 1$. A naive algorithm listing all the common routes and computing k for each of them, would have a complexity of $n \cdot Q$, where Q is the worst case complexity of the local kernel. Let us assume k is a convolution kernel, i.e. it can be written in the form of eq. (1). All C values can be computed in Q time, where $Q = O(|T_1| \cdot |T_2|)$ for SST and $Q = O(\rho^3 \cdot |T_1| \cdot |T_2|)$ for PT. We show that, by aggregating subsets of C values, it is actually possible to efficiently compute eq. (3). In fact, for each $v_1 \in T_1$ and $v_2 \in T_2$, let us define $S(v_1, v_2) = \sum_{v_2' \in t_{v_2}} C(v_1, v_2')$. By exploiting the recursive definition of C, it is easy to see that S can be computed as follows:

$$S(v_1, v_2) = C(v_1, v_2) + \sum_{j=1}^{nc(v_2)} S(v_1, ch_j(v_2)). \tag{4}$$

Note that, when v_2 is a leaf $nc(v_2) = 0$ and the second term of eq. (4) is 0. Assuming to have precomputed the C values, computing all of the S values related to a node v_1 requires $O(|T_2|)$. Thus computing all the S values for a pair of trees requires $O(|T_1| \cdot |T_2|)$, i.e. their computation does not affect the complexity of the SST and PT kernels. Plugging eq. (4) into eq. (3), we obtain:

$$K_k(T_1, T_2) = \sum_{v_1 \in T_1, v_2 \in T_2} \gamma^{d(v_1)} \delta\left(\pi(v_1), \pi(v_2)\right) \sum_{v_1' \in t_{v_1}} S(v_1', v_2). \tag{5}$$

Since, as already noted, there are at most $\min(|T_1|, |T_2|)$ common routes between T_1 and T_2 and those routes can be identified in $\min(|T_1|, |T_2|)$ steps by a simultaneous visit of both trees, the complexity of the kernel depends on the complexity of computing C and S values. Since the complexity of S is not greater than the complexity of computing C for SST and PT, the kernel we described in eq. (3) can be computed without altering the worst-case complexity of both kernels.

The ST kernel has $O(N \log(N))$ time complexity [10], thus computing S values as described in this section, would alter the total complexity of the kernel. We have derived an algorithm for computing PAK-ST which is faster than $O(N^2)$, but it won't be described here due to lack of space.

5 Experiments

Experiments were performed to test the effectiveness of the proposed operator in conjunction with the ST, SST and the PT kernels. The SVM-Light software has been used for the experiments [5, 8]. Our approach has been tested on the INEX 2005 dataset [3], the INEX 2006 dataset [3], the Propbank dataset [6].

The INEX 2005 dataset is a reduced version of the one used for the 2005 INEX competition [3] (for details of the preprocessing see [9]). It consists of $9,640$ xml documents describing movies from the IMDB database. The total number of tree nodes in the dataset is $247,128$, the average number of nodes in a tree is 25.63. The maximum outdegree of a node is 32. The task is an 11-class classification problem. The training and validation sets consist of 3397 and 1423 documents, respectively. The test set is formed by 4820 documents.

The INEX 2006 dataset [3] is derived from the IEEE corpus composed of 12000 scientific articles from IEEE journals in XML format. The total number of tree nodes in the dataset is $218,537$ and the average number of nodes in a tree is 18.05. The maximum outdegree of a tree is 66. In this case the training, validation and test sets consisted of 4237, 1816 and 6054 documents, respectively. The task is an 18-class classification problem.

The Propbank dataset [6] is derived from the Penn Tree Bank II dataset, which, in turns, consists of material from a set of Dow-Jones news articles. The corpus is divided into sections. In order to reduce the computational complexity of the task, we derived the training and validation sets from section 24 by selecting randomly, with uniform probability, a subset of 7000 and 2000 examples, respectively. The test set has been derived selecting randomly, with uniform probability, a subset of 6000 examples from section 23. The total number of tree nodes in the dataset are $209,251$ and the average number of nodes in a tree is 13.95. The maximum outdegree is 15. The task is a binary classification problem. The dataset is very unbalanced: the percentage of positive examples in each set is approximately 7%. Thus the F1 measure has been used for selecting the parameters on the validation set.

The procedure followed for the experiments on each dataset is the following. We first selected the best parameters of the baseline kernel on the validation set. Then, keeping them fixed, we applied the operator proposed in the paper selecting γ on the validation set. Finally, a model learned with the parameter setting on the union of the training and validation sets was tested on the test set. In the case of multiclass classification tasks, i.e. INEX 2005 and INEX 2006, the one against all methodology has been employed. Table 1 summarizes the results obtained. Note that the application of the operator always improves the accuracy of the baseline kernel on INEX 2005 and the improvement is impres-

Table 1. Comparison between the classification error of ST, SST, PT and their version with the proposed operator. The columns represent the lowest classification error on validation and the corresponding classification error on the test set. The performance measure employed for the Propbank dataset is the F1. In bold the best result between the baseline and the PAK extension on the test set.

Kernel	INEX 2005		INEX 2006		Propbank	
	valid. error %	test error %	valid. error %	test error %	valid. F1 (error %)	test F1 (error %)
ST	12.94	11.11	57.27	60.04	0.5078 (6.30)	0.5170 (6.60)
PAK-ST	3.52	**3.44**	57.27	60.04	0.5447 (5.85)	**0.5359** (6.23)
SST	12.51	11.17	57.72	60.40	0.5130 (5.60)	0.5420 (5.72)
PAK-SST	3.59	**3.31**	57.72	60.40	0.5431 (5.55)	**0.5477** (5.92)
PT	2.96	2.96	58.11	58.69	0.5488 (6.00)	0.5161 (7.00)
PAK-PT	2.96	**2.85**	57.55	58.85	0.5636 (5.65)	**0.5787** (6.07)

sive for the ST and SST kernels. For what concerns INEX 2006, in the case of ST and SST, adding positional information to the matchings does not improve the accuracy. The PT kernel improves its accuracy on the validation set, but it does not on the test set. This may be due to the fact that, in order to reduce the time required for the whole experimentation, we do not reselect the c together with γ. The application of the operator improves the F1 of each of the baseline kernels for the Propbank dataset. Although the worst-case computational complexity of PAK-SST and PAK-PT is the same with respect to the corresponding baselines, we computed the execution time overhead due to the PAK extension. The comparison has been performed with respect to the time (in seconds) required for computing the kernel matrices on a Intel(R) Xeon(R) 2.33GHz processor. All PAK extensions has been implemented as modules of the SVM-Light Software [8]. Table 2 reports the ratio between execution times and the ratio between test errors (F1 in the case of Propbank) of the PAK extensions with respect to the baseline kernels. While a ratio lower than 1 for the execution time or the test error means that the first method is better than the second, in the case of the F1, a ratio higher than 1 means that the first method is better than the second. Notice that the execution time ratios of both PAK-ST/PT and PAK-SST/PT are always lower than 1, thus PAK-ST and PAK-SST are faster, up to 4.76 times (0.21^{-1}), than PT. While being faster, PAK-SST has only slightly worse test error on INEX 2006, with ratio 1.03, and better performances on INEX 2005 and Propbank, (with ratios 0.29 and 1.06, respectively).

6 Conclusion and Future Work

In this paper we proposed a general operator for extending convolution tree kernels with positional features. It has quadratic computational complexity in time and thus does not alter the worst-case complexity of most of state-of-the-art tree kernels. Experimental results show that, when positional information is relevant for a specific task, this extension significantly improves on the baseline

Table 2. Ratio between the execution times and the test errors or the F1 of the PAK extensions with respect to the baseline kernels.

		PAK-ST			PAK-SST			PAK-PT		
		ST	SST	PT	ST	SST	PT	ST	SST	PT
INEX 2005	time	2.33	2.29	0.21	2.33	2.31	0.21	14.06	13.86	1.32
	err	0.30	0.30	1.16	0.29	0.28	0.29	0.25	0.25	0.96
INEX 2006	time	2.5	2.42	0.54	2.65	2.44	0.54	6.21	6.04	1.34
	err	1	0.99	1.02	1.01	1	1.03	0.98	0.97	1.01
Propbank	time	2.63	2.63	0.71	2.65	2.65	0.72	5.83	5.82	1.59
	err	1.03	0.98	1.03	1.05	1.01	1.06	1.12	1.06	1.12

kernels. Moreover, less effective tree kernels, if enriched with topological information, may achieve accuracy values comparable to ones of the most effective tree kernels, while being faster to compute. Future works will study the injection of other kinds of relationships which can be defined on substructures and the application of the same operator to other convolution kernels.

References

1. Aiolli, F., Da San Martino, G., Sperduti, A.: Route kernels for trees. In: International Conference on Machine Learning. pp. 17–24 (2009)
2. Collins, M., Duffy, N.: New ranking algorithms for parsing and tagging: Kernels over discrete structures, and the voted perceptron. In: Proceedings of the Fortieth Annual Meeting on Association for Computational Linguistics. pp. 263–270. Philadelphia, PA, USA (2002)
3. Denoyer, L., Gallinari, P.: Report on the XML mining track at INEX 2005 and INEX 2006: categorization and clustering of XML documents. ACM SIGIR Forum 41(1), 79–90 (2007)
4. Haussler, D.: Convolution kernels on discrete structures. Tech. Rep. UCSC-CRL-99-10, University of California, Santa Cruz (July 1999)
5. Joachims, T.: Making large-scale support vector machine learning practical. MIT Press, Cambridge, MA, USA (1999)
6. Kingsbury, P., Palmer, M.: From Treebank to PropBank. In: Proceedings of the 3rd International Conference on Language Resources and Evaluation. pp. 1989–1993. Las Palmas, Spain (2002)
7. Moschitti, A.: Efficient convolution kernels for dependency and constituent syntactic trees. In: Proceedings of the European Conference on Machine Learning. pp. 318–329 (2006)
8. Moschitti, A.: Making tree kernels practical for natural language learning. In: Proceedings of EACL06. Trento, Italy (2006)
9. Trentini, F., Hagenbuchner, M., Sperduti, A., Scarselli, F., Tsoi, A.: A self-organising map approach for clustering of xml documents. In: Proceedings of the WCCI. IEEE Press, Vancouver, Canada (July 2006)
10. Vishwanathan, S., Smola, A.J.: Fast kernels on strings and trees. In: Proceedings of Neural Information Processing Systems 2002. pp. 569–576 (2002)