Fast and Effective Kernels for Relational Learning from Texts

Alessandro Moschitti MOSCHITTI@DIT.UNITN.IT
Department of Information and Communication Technology, University of Trento, 38050 Povo di Trento, Italy

Fabio Massimo Zanzotto ZANZOTTO@INFO.UNIROMA2.IT
Department of Computer Science, Systems and Production, University of Rome "Tor Vergata", 00133 Rome, Italy

Abstract

In this paper, we define a family of syntactic kernels for automatic relational learning from pairs of natural language sentences. We provide an efficient computation of such models by optimizing the dynamic programming algorithm of the kernel evaluation. Experiments with Support Vector Machines and the above kernels show the effectiveness and efficiency of our approach on two very important natural language tasks, Textual Entailment Recognition and Question Answering.

1. Introduction

Statistical relational learning is a wide research area that includes many techniques and approaches. As pointed out in (Getoor, 2005), a statistical model depends on domain relational structures to which model parameters are often tied.

In the domain of natural language texts several applications of relational learning have been studied: from the simple extraction of word collocations, e.g. Named Entities and keyphrases (Bikel et al., 1999), to a higher level of semantic relation search, e.g. relation extraction between Named Entities (Zelenko et al., 2003; Cumby & Roth, 2003) or predicate argument relations (Moschitti, 2006). Lately, the most challenging natural language processing goal has been the extraction of complex relations between entire text fragments. On this subject two main related tasks have captured the attention of most researchers of the field: Question Answering and Textual Entailment recognition.

The former task has been widely studied in the TREC competitions, e.g. (Voorhees, 2003). It consists of a first phase in which text fragments related to the question are retrieved and a second phase in which the relatedness of question and answer has to be detected.

The latter is a traditional linguistic problem which has recently gained a considerable attention thanks to the Pascal's Recognizing Textual Entailment (RTE) challenges (Dagan et al., 2005; Bar Haim et al., 2006). Given two sentences, a text and a hypothesis, the task consists in determining whether the text implies the hypothesis. In both tasks, two very complex objects are involved in relations, e.g. at least two whole sentences. In these conditions typical language model approaches based on the so called bag-of-words (Ponte & Croft, 1998) turn out not to be applicable. Consequently, the solution should rely on the use of domain knowledge, e.g. WordNet (Miller, 1995) and syntactic information (Charniak, 2000).

To exploit the above features, we proposed a kernel function (Zanzotto & Moschitti, 2006) which evaluates the maximum similarity between two $\langle$text, hypothesis$\rangle$ pairs. These are represented by syntactic parse trees enriched with relational information, which is defined between the nodes (i.e. tree constituents) of a text and its corresponding hypothesis. Such relations are automatically tagged by a WordNet based labeler. In other words, some nodes in the first and second pair are tagged and the kernel function, by maximizing the tree similarity, attempts to find which tags of the first pair correspond to the tags of the second pair (i.e. a relation matching problem). This is done by performing a combinatorial search over matches.

In this paper, we generalize the above kernel function and provide an optimization for its computation which greatly reduces the learning and testing time. Our approach can be applied to an entire family of kernels involving pairs of trees whose nodes encode relational information. The key idea is to avoid carrying out the redundant computations of the original method by defining a more effective dynamic programming algorithm underlying the computation.

We apply such technique with different kernels that we designed to learn the relations between text pairs. Moreover,

Appearing in *Proceedings of the 24th International Conference on Machine Learning*, Corvallis, OR, 2007. Copyright 2007 by the author(s)/owner(s).

to apply the approach to domains where the number of related tree nodes may lead to a real combinatorial explosion, we also provide a version of the evaluation algorithm based on a beam search. The experiments with such models and two datasets AVE (Peñas et al., 2006) and RTE (Dagan et al., 2005; Bar Haim et al., 2006) show that our approach is efficient and reaches the state-of-the-art in entailment recognition task. This is an important result as it demonstrates that our models can deal with the important problems of noisy data and error propagation due to automatic syntactic parsing and automatic semantic labeling.

The rest of the paper is organized as follows: Sec. 2 shows our generalization of the similarity function between any pair of texts. Sec. 3 presents our optimization which reduces the number of kernel computations whereas Sec. 4 reports the running time and the accuracy evaluations of several family kernels. Finally, Sec. 5 draws the conclusions.

2. A Generalized Text Pair Kernel

In this section we give a generalized version of the *cross-pair similarity* model devised in (Zanzotto & Moschitti, 2006). First, we describe the idea to derive relations by means of such a similarity (Sec. 2.1), then we present our generalized kernel functions (Sec. 2.2 and Sec. 2.3).

2.1. Learning from Pairs of Text Fragments

Determining the relations between two generic texts T_1 and T_2 is a relational problem which not only depends on the lexical content but also on the way such content is structurally organized. For example, in the following text pairs:

T_1	*"All companies file annual reports."*	
T_2	*"All insurance companies file annual reports."*	(E_1)

and

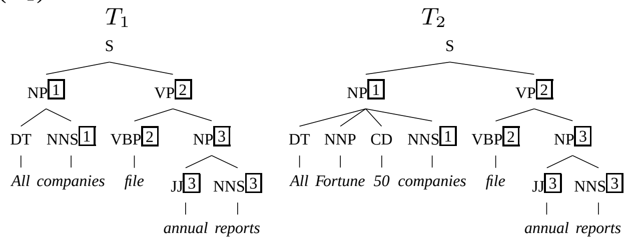
T_1	*"All companies file annual reports."*	
T_2	*"All companies file annual reports to the SEC."*	(E_2)

we observe that in E_1 the first sentence implies the second whereas no implication can be derived between sentences in E_2.

The difference between E_1 and E_2 is independent of the meanings of the sentences. It only depends on the syntactic structures of the four sentences and the relations between sentences within pairs. For example, most texts matching the "*All NP(X) VP(Y)*" structure entail texts matching the "*All NP(Z X) VP(Y)*" structure, where NP and VP are noun and verbal phrases composed by the sequences of words X, Y or Z. A similar, but negative, relation can be inferred from the second example. The texts matching "*All NP(X) VP(Y)*" do not entail texts matching "*All NP(X) VP(Y) PP(Z)*".

The above example shows that, to automatically derive textual relations, we need to take syntax into account. For this purpose in (Zanzotto & Moschitti, 2006) *placeholders* were used to mark the sentence constituents, e.g. NP, VP and PP, in parse trees such that variables, e.g. X, Y and Z, can be assigned to them. For example, the parse tree of the sentence pair (E_1) is represented as follows:

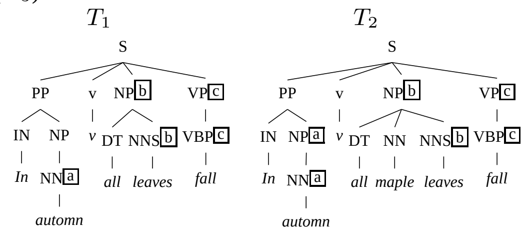
$(\overline{E}_1)$



where the placeholders $\boxed{1}$, $\boxed{2}$, and $\boxed{3}$ indicate the relations between the structures of T_1 and T_2. The relations between constituents are determined by using the lexical relations between their component words. These are usually available in external thesauri, e.g. WordNet (Miller, 1995).

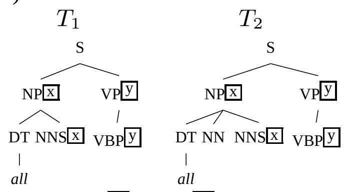
Placeholders on syntactic trees can help to determine if two text pairs share the same *relation* by looking at the subtrees that they have in common. For example, to determine if "*In autumn, all leaves fall.*" implies "*In autumn, all maple leaves fall.*", we can firstly consider their syntactic and co-indexed representation:

$(\overline{E}_3)$



If we change $\boxed{1}$ with $\boxed{x}$ and $\boxed{2}$ with $\boxed{y}$ in $\overline{E}_1$ and $\boxed{b}$ with $\boxed{x}$ and $\boxed{c}$ with $\boxed{y}$ in $\overline{E}_3$, we can simply compare $\overline{E}_1$ and $\overline{E}_3$ and we can discover that they share the following subtrees:

$(\overline{R})$



This is the relation that $\overline{E}_1$ and $\overline{E}_3$ have in common.

The above example shows that, syntactic trees enriched with placeholders may help to determine if two text pairs share the same relations. The next section shows how to process such data with kernel functions.

2.2. Tree Kernel Functions

Given the parse trees of text pairs, we may represent them by appropriately defining and extracting features which encode textual relations. However, such manual design appears to be very complex and it should be carried out for each different application of textual relational learning. A viable alternative for the representation of parse trees is the use of tree kernels (Collins & Duffy, 2002).

A syntactic tree kernel $K_T(\tau^\alpha, \tau^\beta)$ computes the number of common subtrees between τ^α and τ^β. Assuming that we indicate with n a node of the tree τ, with $ch(n, j)$ the j-th child of the node n, and with $nc(n)$ the number of children of n, the function K_T is computed by

$$K_T(\tau^\alpha, \tau^\beta) = \sum_{n^\alpha \in \tau^\alpha} \sum_{n^\beta \in \tau^\beta} \Delta(n^\alpha, n^\beta), \qquad (1)$$

where $\Delta(n^\alpha, n^\beta)$ is recursively defined as follows:

1. $\Delta(n^\alpha, n^\beta) = 0$ if the productions rooted in n^α and n^β are not equal
2. $\Delta(n^\alpha, n^\beta) = \lambda$ if n^α and n^β are preterminals with the productions $n^\alpha \rightarrow w^\alpha$ and $n^\beta \rightarrow w^\beta$ and $n^\alpha = n^\beta$ and $w^\alpha = w^\beta$.
3. $\Delta(n^\alpha, n^\beta) = \lambda \prod_{j=1}^{nc(n^\alpha)} (1 + \Delta(ch(n^\alpha, j), ch(n^\beta, j))$ otherwise,

where λ is the tree decay factor. The above $\Delta(n^\alpha, n^\beta)$ evaluation allows us to compute K_T in $O(|\tau^\alpha||\tau^\beta|)$ time.

By means of the above kernel, we may design a syntactic similarity between two text pairs (T_1^α, T_2^α) and (T_1^β, T_2^β) by a simple tree kernel sum, i.e. $K_s = K_T(T_1^\alpha, T_1^\beta) + K_T(T_2^\alpha, T_2^\beta)$. However, when placeholders are used to derive relational information, Eq. 1 will not be effective since those in pair α are different from those in pair β. The next section provides a solution to such problem.

2.3. Kernel Functions over Text Pairs

Since placeholders in two pairs are in principle different, when added to parse trees, the number of shared substructures between two texts decreases. Therefore, a substitution function which assigns the same name to related placeholders is needed.

Given a set c of correspondences between placeholders p^α of the first pair (T_1^α, T_2^α) and those p^β of the second pair (T_1^β, T_2^β), $t(\Gamma, c)$ substitutes placeholders with new names, where Γ is a text parse tree. If applied to the four trees of the two pairs, the substitution function $t(\Gamma, c)$ guarantees that the related placeholders have the same name.

For example, let p^α and p^β be placeholders of Γ^α and Γ^β, respectively, if $|p^\beta| < |p^\alpha|$, we can define a bijection between a subset $p'^\alpha \subseteq p^\alpha$ and p^β. Fig. 1 shows the tree Γ^α with its placeholders p^α $=\{\boxed{a},\boxed{b},\boxed{c},\boxed{d}\}$, the tree Γ^β with p^β $=\{\boxed{1},\boxed{2},\boxed{3}\}$ and a possible set of correspondences $c_1 = \{(a, 1), (b, 2), (c, 3)\}$.

Fig. 1 also reports how the substitution function works on the given trees, e.g. in the tree Γ^α, each placeholder $\boxed{a}$ is replaced with the new placeholder $\boxed{a{:}1}$ by $t(\cdot, c)$ obtaining the transformed tree $t(\Gamma^\alpha, c_1)$. Coherently, in the tree $t(\Gamma^\beta, c_1)$ each instance of the placeholder $\boxed{1}$ in Γ^β is replaced with $\boxed{a{:}1}$. After this substitution, the labels of the two trees can be matched and the tree kernel K_T can effectively be applied.

Having named $\mathcal{C}$ the collection of all the possible sets of correspondences between the placeholders of two pairs, a family of similarity functions can be computed as:

$$K_\Lambda((T_1^\alpha, T_2^\alpha), (T_1^\beta, T_2^\beta)) = \\ \Lambda_{c \in \mathcal{C}} \big(K_T(t(T_1^\alpha, c), t(T_1^\beta, c)) + K_T(t(T_2^\alpha, c), t(T_2^\beta, c))\big), \qquad (2)$$

where Λ expresses a function over $\mathcal{C}$.

For example, when Λ is the $\max$ function (Zanzotto & Moschitti, 2006), Eq. 2 finds the maximal similarity in terms of substructures when a certain number of variables (placeholders) are instantiated. Although this is not in general a valid kernel (Boughorbel et al., 2004), kernel machines like SVMs still solve a data separation problem in pseudo Euclidean spaces (Haasdonk, 2005) by finding a local optimum that can be satisfactory for the task.

Other kinds of valid kernels of the family can be designed by using valid kernel operations, e.g. $\Lambda_{c \in \mathcal{C}} = \sum_{c \in \mathcal{C}}$ applied to sums, $K_T(\cdot, \cdot) + K_T(\cdot, \cdot)$, or products, $K_T(\cdot, \cdot) \times K_T(\cdot, \cdot)$, of tree kernels.

From a computational complexity point of view, as $\mathcal{C}$ is combinatorial with respect to $|p^\alpha|$ and $|p^\beta|$, $|\mathcal{C}|$ rapidly grows. Assigning placeholders only to chunks helps in keeping their number controlled. In the RTE (Dagan et al., 2005) data for example, the number of placeholders is hardly larger than 7 as hypotheses are generally short sentences. Nevertheless, the number of K_T computations is still high.

To improve the running time, we observe that as the trees $t(\Gamma, c)$ are obtained from the initial tree Γ (containing placeholders) by applying different $c \in \mathcal{C}$, it is reasonable to assume that they can share common subparts. Thus, during the iterations of $c \in \mathcal{C}$, $K_T(t(\Gamma^\alpha, c), t(\Gamma^\beta, c))$ may compute the similarity between subtrees that have already been evaluated. In the next section, we exploit the above property to compute K_Λ more efficiently.

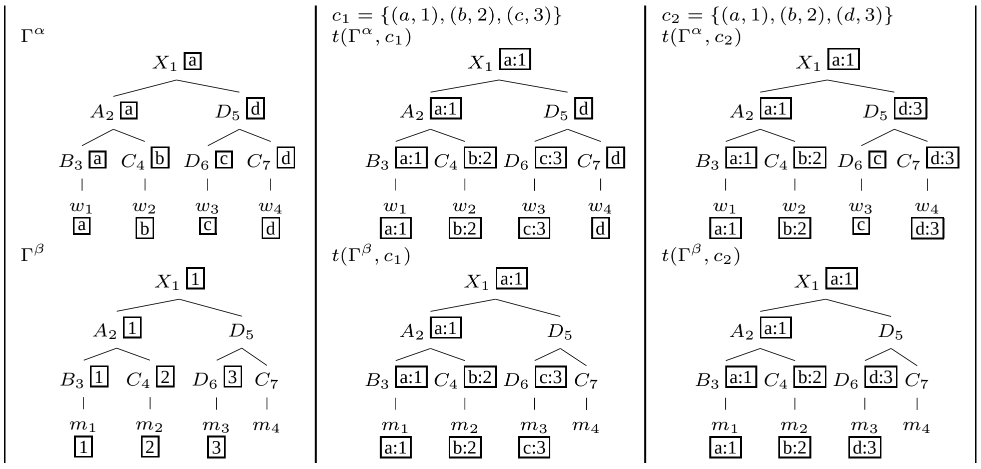


Figure 1. Tree pairs with placeholders and $t(T, c)$ transformation

3. Fast Kernels for Relational Learning

The previous section has shown that the similarity function K_Λ (Eq. 2) firstly applies a transformation $t(\cdot, c)$ and then computes the tree kernel K_T (Eq. 1) based on the Δ function (see Section 2.2). The overall process can be optimized if Δ is evaluated with respect to substitutions c and by factorizing redundant Δ computations. In this section we firstly motivate the previous idea (Sec. 3.1), then we introduce the notation needed to describe the fast kernels (Sec. 3.2) which are finally presented in (Sec. 3.3).

3.1. Reusing Previous Computation

First of all, to simplify the description of our approach, we only focus on the computation of one tree kernel function in Eq 2, i.e. we only consider $K_\Lambda(\Gamma^\alpha, \Gamma^\beta) = \Lambda_{c \in \mathcal{C}}\big(K_T(t(\Gamma^\alpha, c), t(\Gamma^\beta, c))\big)$, where the $(\Gamma^\alpha, \Gamma^\beta)$ pair can be either (T_1^α, T_1^β) or (T_2^α, T_2^β).

Second, we observe that the two trees, $t(\Gamma, c_1)$ and $t(\Gamma, c_2)$, obtained by applying two sets of correspondences $c_1, c_2 \in \mathcal{C}$, may partially overlap since c_1 and c_2 can share a non-empty set of common elements. Indeed, if we define the set of subtree S shared by $t(\Gamma, c_1)$ and $t(\Gamma, c_2)$ and containing placeholders in $c_1 \cap c_2 = c$, then $t(\gamma, c) = t(\gamma, c_1) = t(\gamma, c_2)\ \forall \gamma \in S$. Therefore, when applying a tree kernel function K_T to a pair $(\Gamma^\alpha, \Gamma^\beta)$, it may be possible to find c such that subtrees of Γ^α and subtrees of Γ^β are invariant with respect to c_1 and c_2, i.e. $K_T(t(\gamma^\alpha, c), t(\gamma^\beta, c))$ = $K_T(t(\gamma^\alpha, c_1), t(\gamma^\beta, c_1))$= $K_T(t(\gamma^\alpha, c_2), t(\gamma^\beta, c_2))$. This suggests that to evaluate $K_\Lambda(\Gamma^\alpha, \Gamma^\beta)$ more efficiently, we can refine the dynamic programming algorithm for the Δ computation.

As an example, let us consider the two trees, Γ^α and Γ^β, represented in Fig. 1 and the application of two different transformations $c_1 = \{(a,1),(b,2),(c,3)\}$ and $c_2 = \{(a,1),(b,2),(d,3)\}$. Nodes are generally in the form $X_i\boxed{z}$ where X is the original node label, $\boxed{z}$ is the placeholder, and i is used to index nodes of the tree. Two nodes are equal if they have the same node label and the same placeholder. The first frame of the figure represents the original trees Γ^α and Γ^β. The second frame contains the transformed trees $t(\cdot, c_1)$ while the last frame shows the two trees derived applying $t(\cdot, c_2)$. We choose c_1 and c_2 such that they have a common non-empty subset $c = \{(a,1),(b,2)\}$.

Now, since the subtree of Γ^α rooted in $A_2\boxed{a}$ contains only placeholders that are in c, in the transformed trees, $t(\Gamma^\alpha, c_1)$ and $t(\Gamma^\alpha, c_2)$, the subtrees rooted in $A_2\boxed{a{:}1}$ are identical. The same happens for Γ^β with the subtree rooted in $A_2\boxed{1}$ as all its placeholders are contained in c. Thus, in the transformed trees, $t(\Gamma^\beta, c_1)$ and $t(\Gamma^\beta, c_2)$, the subtrees rooted in $A_2\boxed{a{:}1}$ are identical. As a result, K_T applied to the above subtrees gives an identical result.

3.2. Notations and Operators

To describe the algorithm that exploits the previous idea, we define three basic operators which extract specific information about trees and placeholders. We need to (1) know the placeholders contained in a given subtree, (2) project a set of correspondences onto two specific sets of placeholders and (3) redefine the transformation function t. In the definition of the above operators, we assume a tree with placeholders Γ, a node $n \in \Gamma$, and a set $\mathcal{C}$ of all the sets of correspondences between placeholders.

The first operator is $p(n)$ that extracts the set of placeholders from a subtree of Γ rooted in n. For example, in Fig. 1, the set of placeholders in the subtree of Γ^α rooted in A_2 is $p(A_2) = \{\boxed{a}, \boxed{b}\}$.

The second operator, the projector $c|_{X,Y} = \{(x,y) \in c | x \in X, y \in Y\}$, is defined on two sets of placeholders X and Y and a set of correspondences $c \in \mathcal{C}$. For example, the projection of $c_1 = \{(a,1),(b,2),(c,3)\}$ on the sets $X = \{\boxed{a}, \boxed{b}\}$ and $Y = \{\boxed{1}, \boxed{2}\}$ is $c_1|_{X,Y} = \{(a,1),(b,2)\}$.

The last operator, $t(n,c)$, applies to subtrees of a given tree. It returns the subtree of Γ rooted in n where the placeholders have been substituted according to c. For example, using the trees in Fig. 1 and this new operator, we can use $t(A_2, c_2)$ to indicate the subtree rooted in $A_2\boxed{a:1}$ node of Γ^α shown in the right frame of Fig. 1.

3.3. A Fast Evaluation of the K_Λ Function

To give a more efficient algorithm for the kernel K_Λ computation, we (a) integrate the transformation $t(\cdot, c)$ in the computation of the tree kernel K_T, (b) formalize when two different c's lead to the same Δ evaluation and (c) define an algorithm for K_Λ that takes into account the previous property. By introducing c in $K_T(t(\Gamma^\alpha, c), t(T^\beta, c))$, K_Λ can be rewritten as:

$$K_\Lambda(\Gamma^\alpha, \Gamma^\beta) = \Lambda_{c\in\mathcal{C}}\big(K_T(\Gamma^\alpha, \Gamma^\beta, c)\big) \qquad (3)$$

K_T can now be computed as:

$$K_T(\Gamma^\alpha, \Gamma^\beta, c) = \sum_{n^\alpha\in\Gamma^\alpha} \sum_{n^\beta\in\Gamma^\beta} \Delta(n^\alpha, n^\beta, c), \qquad (4)$$

where the new Δ function also depends on c as follows:

1. $\Delta(n^\alpha, n^\beta, c) = 0$ if the productions rooted in n^α and n^β after the application of the transformation $t(n^\alpha, c)$ and $t(n^\beta, c)$ are not equal
2. $\Delta(n^\alpha, n^\beta, c) = \lambda$ if n^α and n^β are preterminals and $t(n^\alpha, c) = t(n^\beta, c)$
3. $\Delta(n^\alpha, n^\beta, c) = \lambda \prod_{j=1}^{nc(n^\alpha)} (1 + \Delta(ch(n^\alpha, j), ch(n^\beta, j), c)$ otherwise.

From the above equations, it follows that $\Delta(n^\alpha, n^\beta, c)$ is applied to the two subtrees rooted in $n^\alpha \in \Gamma^\alpha$ and $n^\beta \in \Gamma^\beta$ after their placeholders are transformed by $t(n^\alpha, c)$ and $t(n^\beta, c)$. Of course, this transformation involves only the subset of placeholders of c which are in the two subtrees, i.e. $c|_{p(n^\alpha),p(n^\beta)}$. Therefore, the following property

$$\begin{array}{c}\Delta(n^\alpha, n^\beta, c_1) = \Delta(n^\alpha, n^\beta, c_2) \\ \text{if } c_1|_{p(n^\alpha),p(n^\beta)} = c_2|_{p(n^\alpha),p(n^\beta)}\end{array} \qquad (5)$$

holds.

According to Eq. 5, K_T can be defined as follows:

$$K_T(\Gamma^\alpha, \Gamma^\beta, c) = \sum_{n^\alpha\in\Gamma^\alpha} \sum_{n^\beta\in\Gamma^\beta} \Delta(n^\alpha, n^\beta, c|_{p(n^\alpha),p(n^\beta)}), \qquad (6)$$

$c_1 = \{(a,1),(b,2),(c,3)\}$
$c_1|_{p(n^\alpha),p(n^\beta)} =$

		Γ^α						
		X_1	A_2	B_3	C_4	D_5	C_6	C_7
	X_1	(a,1),(b,2),(c,3)	(a,1),(b,2)	(a,1)	(b,2)	(c,3)	(c,3)	$\emptyset$
	A_2	(a,1),(b,2)	(a,1),(b,2)	(a,1)	(b,2)	$\emptyset$	$\emptyset$	$\emptyset$
	B_3	(a,1)	(a,1)	(a,1)	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
Γ^β	C_4,	(b,2)	(b,2)	$\emptyset$	(b,2)	$\emptyset$	$\emptyset$	$\emptyset$
	D_5	(c,3)	$\emptyset$	$\emptyset$	$\emptyset$	(c,3)	(c,3)	$\emptyset$
	D_6	(c,3)	$\emptyset$	$\emptyset$	$\emptyset$	(c,3)	(c,3)	$\emptyset$
	C_7	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$

$c_2 = \{(a,1),(b,2),(d,3)\}$
$c_2|_{p(n^\alpha),p(n^\beta)} =$

		Γ^α						
		X_1	A_2	B_3	C_4	D_5	C_6	C_7
	X_1	(a,1),(b,2),(d,3)	(a,1),(b,2)	(a,1)	(b,2)	(d,3)	$\emptyset$	(d,3)
	A_2	(a,1),(b,2)	(a,1),(b,2)	(a,1)	(b,2)	$\emptyset$	$\emptyset$	$\emptyset$
	B_3	(a,1)	(a,1)	(a,1)	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
Γ^β	C_4	(b,2)	(b,2)	$\emptyset$	(b,2)	$\emptyset$	$\emptyset$	$\emptyset$
	D_5	(d,3)	$\emptyset$	$\emptyset$	$\emptyset$	(d,3)	$\emptyset$	(d,3)
	D_6	(d,3)	$\emptyset$	$\emptyset$	$\emptyset$	(d,3)	$\emptyset$	(d,3)
	C_7	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$

Common submatrix: $c_1|_{p(n^\alpha),p(n^\beta)} \cap c_2|_{p(n^\alpha),p(n^\beta)} =$

		Γ^α						
		X_1	A_2	B_3	C_4	D_5	C_6	C_7
	X_1	-	(a,1),(b,2)	(a,1)	(b,2)	-	-	-
	A_2	(a,1),(b,2)	(a,1),(b,2)	(a,1)	(b,2)	$\emptyset$	$\emptyset$	$\emptyset$
	B_3	(a,1)	(a,1)	(a,1)	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
Γ^β	C_4	(b,2)	(b,2)	$\emptyset$	(b,2)	$\emptyset$	$\emptyset$	$\emptyset$
	D_5	-	$\emptyset$	$\emptyset$	$\emptyset$	-	-	-
	D_6	-	$\emptyset$	$\emptyset$	$\emptyset$	-	-	-
	C_7	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$

Figure 2. Projection of c during the Δ computations

where the last step of $\Delta(n^\alpha, n^\beta, c)$ definition changes in:

3. $\Delta(n^\alpha, n^\beta, c) = \lambda \prod_{j=1}^{nc(n^\alpha)} \Big(1 + \Delta(ch(n^\alpha, j), ch(n^\beta, j), c|_{p(ch(n^\alpha,j)),p(ch(n^\beta,j))}\Big)$ otherwise.

The above definition reduces the computational cost since $\Delta(n^\alpha, n^\beta, c)$ are used more than once in the computation of $\Lambda_{c\in\mathcal{C}}\big(K_T(\Gamma^\alpha, \Gamma^\beta, c)\big)$.

As an example of the usefulness of Eq. 5, let us consider Fig. 2 which reports the projection of the sets of correspondences c_1 and c_2 according to the nodes of Γ^α and Γ^β of Fig. 1. The three matrices represent the projections $c_1|_{p(n^\alpha),p(n^\beta)}$, $c_2|_{p(n^\alpha),p(n^\beta)}$ and $c_1 \cap c_2|_{p(n^\alpha),p(n^\beta)}$ for the node pair (n^α, n^β). The first row shows the nodes of Γ^α while the first column represents those of the second tree, Γ^β. For the sake of clarity, we report here the values of the projection function $p(n)$ for each node n of both Γ^α and Γ^β:

Γ^α n	X_1	A_2	B_3	C_4	D_5	C_6	C_7
$p(n)$	$\{\boxed{a},\boxed{b},\boxed{c},\boxed{d}\}$	$\{\boxed{a},\boxed{b}\}$	$\{\boxed{a}\}$	$\{\boxed{b}\}$	$\{\boxed{c},\boxed{d}\}$	$\{\boxed{c}\}$	$\{\boxed{d}\}$

Γ^β n	X_1	A_2	B_3	C_4	D_5	C_6	C_7
$p(n)$	$\{\boxed{1},\boxed{2},\boxed{3}\}$	$\{\boxed{1},\boxed{2}\}$	$\{\boxed{1}\}$	$\{\boxed{2}\}$	$\{\boxed{3}\}$	$\{\boxed{3}\}$	$\emptyset$

Note that the bottom matrix in Fig. 2 contains all the pairs for which $c_1|_{p(n^\alpha),p(n^\beta)} = c_2|_{p(n^\alpha),p(n^\beta)}$ holds. For this example, this property holds 37 times over 49 computations. Therefore, to evaluate $\Delta(n^\alpha, n^\beta, c_1)$, we need 12 computations after we have already computed $\Delta(n^\alpha, n^\beta, c_2)$.

3.4. A Beam Search for the Max Function Computation

The previous section has shown an efficiency optimization of redundant evaluation specific to the target kernel family K_Λ. However, traditional search algorithms can be applied jointly. In the following, we show a beam search algorithm designed to improve the computation speed of $K_{\max}$:

```
$kernel\_computation(\Gamma_\alpha, \Gamma_\beta)$
begin
  $agenda = \emptyset$
  $insert(agenda, \Delta(\Gamma_\alpha, \Gamma_\beta, \emptyset))$
  for each $a_1 \in p^\alpha$
    for each $a_2 \in p^\beta$
      for each $\Delta(\Gamma_\alpha, \Gamma_\beta, c) \in agenda$
        $insert(agenda', \Delta(\Gamma_\alpha, \Gamma_\beta, c \cup \{(a_1, a_2)\}))$
    $agenda = agenda'$
  return $peek\_first\_value(agenda)$
end
```

The $agenda$ retains the partial set k of correspondences c that have the best value of the partial kernel $K_{\max}$ computed over the partial matrix Δ. The partial matrix Δ has null values for pairs of nodes (n_α, n_β) if the subtrees rooted in n_α and n_β contain placeholders that are not in the partial set of correspondences c. The procedure $insert(agenda, \Delta(\Gamma_\alpha, \Gamma_\beta, c))$ puts in the k-size $agenda$ the set c if $K_{\max}$ is higher than the minimum value in $agenda$. The procedure $peek\_first\_value(agenda)$ takes the highest score of $agenda$. The algorithm complexity is $O(k|p^\alpha||p^\beta||\Gamma_\alpha||\Gamma_\beta|)$

4. The Experiments

The aim of the experiments is twofold: (a) we show the speed-up that our fast evaluation of K_Λ produces in both learning and testing phase of Support Vector Machines and (b) we illustrate the potentiality of the K_Λ family for relational learning tasks such as Textual Entailment Recognition and Question Answering.

4.1. Experimental Setup

We implemented K_Λ with the naïve (Eq. 2) and fast (Eq. 3) computation approaches in the SVM-light-TK software available at `http://ai-nlp.info.uniroma2.it/moschitti`. This encodes different tree kernels in SVM-light (Joachims, 1999). We used the default cost factor and trade-off parameters and we set λ to 0.4.

To experiment with entailment relations, we used the data sets made available by the first (Dagan et al., 2005) and second (Bar Haim et al., 2006) Recognizing Textual Entailment Challenge. These corpora are divided in development $D1$ and $D2$ and test sets $T1$ and $T2$. $D1$ contains 567 examples whereas $T1$, $D2$ and $T2$ all have the same size, i.e. 800 instances. Each example is an ordered pair of texts for which the entailment relation has to be decided.

To experiment with relations between question and answer, we used the Answer Validation Exercise (AVE) dataset (Peñas et al., 2006). This contains pairs of questions and answers for which the correctness, i.e. if the answer correctly responds to the associated question, has to be predicted. The AVE development set contains 2,870 instances. Here, the positive and negative examples are not equally distributed. It contains 436 positive and 2,434 negative examples.

Since all the above three datasets were used in international competitions, we could exactly compare with state-of-the-art approaches.

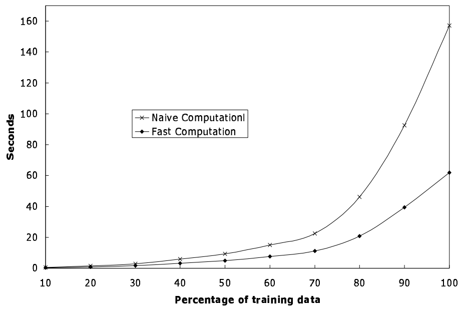
4.2. Running Time Experiments

Analytically determining the complexity of our fast computation (FC) is difficult as it depends on the variability and type of the application data. Since this comes from natural language sentences, defining models that quantify the number of elementary operations in the kernel function is quite complex. However, we can provide an empirical evaluation of our algorithm in terms of its impact on training and classification running time.

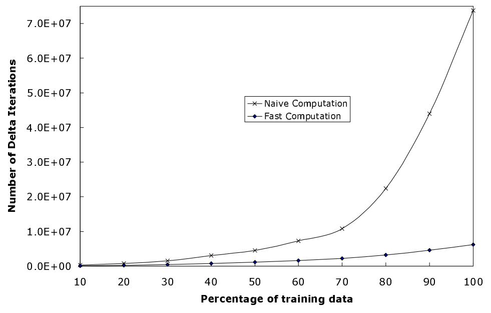
To give a finer evaluation, we studied the number of Δ (defined in Sec. 3.3) iterations according to different data sets. This is better suited to compare between the naïve computation (NC) and FC as it is not biased by the processing time required by the learning and classification code not correlated with the kernel functions.

To study the relation between the size of training data and the learning time needed by NC and FC, we divided D1 and D2 in bins of increasing sizes (from 10% to 100% with step 10%). Plot (a) in Figure 3 shows the time required for training an SVM with bins extracted from RTE2. Subfigure (b) illustrates the plot of Δ iterations instead of execution time. We note that FC greatly reduces the computation time and the number of Δ iterations (not biased by additional time) is decreased by about 10 times (see when using all training data).

To study the classification time, we classified data bins of increasing size with the same SVM model trained on the whole D1. Figure 4 shows that NC requires about 12 times the number of iterations of FC to classify T1, this produces a much higher testing time.

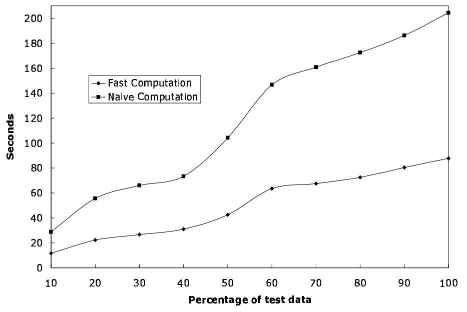


(a) RTE2



(b) RTE2

Figure 3. Training time and number of Δ iterations according to different data bins



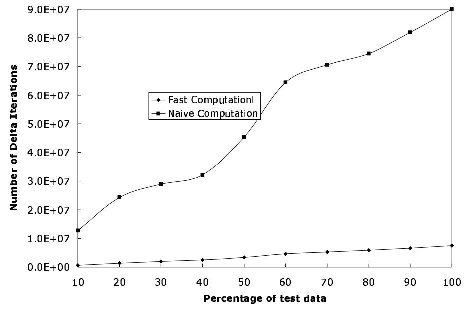


Figure 4. Testing time and number of Δ iterations according to different test data bins and fixed training data from RTE1.

Finally, when the beam search is applied along with FC on the RTE, SVMs decrease their learning time from 61.77 to 54.04 seconds (on RTE2 data) and their testing time from 87.72 to 71.17 seconds (on RTE1 data). We note that there is not much speed improvement since the number of placeholders is small and practically constant in such datasets. However other application domains different from text may show an unbounded number of placeholders. In such case algorithms like beam search would be the only feasible solutions.

4.3. Accuracy Evaluation

To verify the quality of our relational learning approach, we measured the accuracy of three different kernels:

- $K_B((T_1^\alpha, T_2^\alpha),(T_1^\beta, T_2^\beta)) = K_T(T_1^\alpha, T_1^\beta) + K_T(T_2^\alpha, T_2^\beta)$,
- $K_{max}((T_1^\alpha, T_2^\alpha),(T_1^\beta, T_2^\beta)) = \max_{c \in \mathcal{C}} \big(K_T(t(T_1^\alpha, c), t(T_1^\beta, c)) + K_T(t(T_2^\alpha, c), t(T_2^\beta, c))\big)$
- $K_\Sigma((T_1^\alpha, T_2^\alpha),(T_1^\beta, T_2^\beta)) = \sum_{c \in \mathcal{C}} \big(K_T(t(T_1^\alpha, c), t(T_1^\beta, c)) + K_T(t(T_2^\alpha, c), t(T_2^\beta, c))\big)$,

where (T_1^α, T_2^α) and (T_1^β, T_2^β) are two text pairs. Note that K_B and K_Σ are two valid kernels whereas K_{max} is not. It should be noted that, along with syntactic information, RTE systems usually use a kernel based on lexical similarity between words, see e.g. (Corley & Mihalcea, 2005). This lexical kernel is built on external resources, e.g. WordNet, which are able to generalize words and provide a higher recall. To have a fair comparison with such systems, we added a similar lexical kernel (LK) to the above models.

Table 1 shows the results of the above kernels on the split used for the RTE and AVE competitions. We note that (i) the tree kernel K_B does not significantly improve LK as, without the use of placeholders for linking constituents, few relations could be derived; (ii) $K_{\max}$ relevantly improves LK. Although it is not a valid kernel, it intuitively selects the best correspondence between placeholders; (iii) K_Σ also improves LK but it shows a lower accuracy than $K_{\max}$ and (iv) as expected, the beam search applied to $K_{\max}$ decreases its accuracy.

Finally, $K_{\max}$ model improves the average result of the systems participating in RTE1, RTE2 and AVE of about 9 (i.e. 0.63 vs 0.54), 5 (i.e. 0.64 vs 0.59) and 8 (i.e. 0.43 vs 0.35) absolute percent points, respectively. It should be noted that the best two systems of RTE outperformed all the others of about 10-5 absolute percent points. However, they cannot be taken as the reference systems since they use lexical resources and training data not available to the other participants. The third best system was the $K_{\max}$ model which would also have been the best approach in AVE if all the participants had had the same resources, i.e. a fair comparative setting.

	RTE1	RTE2	AVE		
	Acc	Acc	$Prec$	Rec	$F1$
LK	0.597	0.617	0.251	0.823	0.385
LK + K_B	0.619	0.616	0.398	0.384	0.391
LK + $K_{\max}$	0.631	0.640	0.386	0.495	0.434
LK + K_Σ	0.605	0.621	0.364	0.478	0.414
LK + $K_{\max}$ + beam	0.621	0.617	0.310	0.571	0.401
Avg others	0.54	0.59	-	-	0.35

Table 1. Accuracy of Relational Kernels on RTE1 and RTE2. Precision, Recall, and F1-measure on AVE.

LK	LK + K_B	LK + $K_{\max}$	LK + K_Σ	LK + $K_{\max}$ +beam
0.619±0.022	0.619±0.013	0.642±0.018	0.635±0.028	0.639±0.019

Table 2. Accuracy of Relational Kernels derived on 5-fold cross validation on RTE2.

Some of the above results do not fully reveal the role of the beam search and K_Σ, e.g. in one case the latter performs lower than K_B. The problem here is that the results derived on a single split may not be statistically significant. To overcome this problem, we also ran a 5-fold cross validation on RTE2[1].

The results reported in Table 2 are the average over 5 samples $\pm$ the confidence limits at 90%. We note that (i) K_B+LK performs as LK alone (i.e. 0.619). This suggests that K_B does not add any information to LK. (ii) K_Σ is slightly less accurate than $K_{\max}$ (i.e. 0.635 vs 0.642). (iii) The beam search slightly deteriorates the accuracy of $K_{\max}$ (0.642 vs 0.639).

5. Conclusions

In this paper, we have presented a new family of kernels for relational learning from texts and we have provided fast algorithms for their evaluation. We have empirically demonstrated that our approach reduces the number of computations of the Δ matrix employed for the dynamic computation of the above kernel functions. This allows us to experiment with very large datasets, from Text Entailment Recognition and Question Answering challenges, and other kernels of the proposed family. The results show that our relational kernels obtain state-of-the-art accuracy by using limited resources, i.e. a syntactic parser and a lexical semantic network.

Finally, promising future research could be devoted (a) to study innovative relational kernels of the proposed family, (b) to use our approach for the design of large-scale applications like web question answering, e.g. by re-ranking the n-best answers and (c) to apply our methods to new domains different from text, in which syntactic information is provided, e.g. XML documents.

[1]We did not run n-fold validation on the other datasets as they contain repeated instances and mixing them between train and testing would have highly biased the results

Acknowledgments

Alessandro Moschitti would like to thank the AMI2 lab at the University of Trento and the EU project LUNA ”spoken Language UNderstanding in multilinguAl communication systems” contract n^o 33549 for supporting part of his research. Many thanks to the anonymous reviewers for their helpful suggestions.

References

Bar Haim, R., Dagan, I., Dolan, B., Ferro, L., Giampiccolo, D., Magnini, B., & Szpektor, I. (2006). The II PASCAL RTE challenge. *PASCAL Challenges Workshop*. Venice, Italy.

Bikel, D., Schwartz, R., & Weischedel, R. (1999). An Algorithm that Learns What’s in a Name. *Machine Learning, Special Issue on Natural Language Learning*.

Boughorbel, S., Tarel, J.-P., & Fleuret, F. (2004). Non-mercer kernel for SVM object recognition. *Proceedings of BMVC 2004*. London, England.

Charniak, E. (2000). A maximum-entropy-inspired parser. *Proc. of the 1st NAACL*. Seattle, Washington, USA.

Collins, M., & Duffy, N. (2002). New ranking algorithms for parsing and tagging: Kernels over discrete structures, and the voted perceptron. *Proceedings of ACL02*. Morristown, NJ, USA.

Corley, C., & Mihalcea, R. (2005). Measuring the semantic similarity of texts. *Proc. of the ACL Workshop on Empirical Modeling of Semantic Equivalence and Entailment*. Ann Arbor, Michigan, USA.

Cumby, C., & Roth, D. (2003). Kernel methods for relational learning. *Proceedings of ICML 2003*. Washington, DC, USA.

Dagan, I., Glickman, O., & Magnini, B. (2005). The PASCAL RTE challenge. *PASCAL Challenges Workshop*. Southampton, U.K.

Getoor, L. (2005). Tutorial on statistical relational learning. *ILP* (p. 415).

Haasdonk, B. (2005). Feature space interpretation of SVMs with indefinite kernels. *IEEE Trans Pattern Anal Mach Intell*, *27*.

Joachims, T. (1999). Making large-scale svm learning practical. *Advances in Kernel Methods-Support Vector Learning*. MIT Press.

Miller, G. A. (1995). WordNet: A lexical database for English. *Communications of the ACM*.

Moschitti, A. (2006). Efficient convolution kernels for dependency and constituent syntactic trees. *Proceedings of ECML*, Berlin, Germany.

Peñas, A., Rodrigo, A., Sama, V., & Verdejo, F. (2006). Overview of the answer validation exercise 2006. *Working Notes for the CLEF 2006 Workshop*. Alicante, Spain.

Ponte, J. M., & Croft, W. B. (1998). A language modeling approach to information retrieval. *Proceedings of SIGIR ’98*. New York, NY, USA.

Voorhees, E. M. (2003). Overview of TREC 2003. *TREC*.

Zanzotto, F. M., & Moschitti, A. (2006). Automatic learning of textual entailments with cross-pair similarities. *Proceedings of the 21st Coling and 44th ACL*. Sydney, Australia.

Zelenko, D., Aone, C., & Richardella, A. (2003). Kernel methods for relation extraction. *Journal of Machine Learning Research*.