Shallow Semantic in Fast Textual Entailment Rule Learners

Fabio Massimo Zanzotto
DISP
University of Rome "Tor Vergata"
Roma, Italy
zanzotto@info.uniroma2.it

Marco Pennacchiotti
Computerlinguistik
Universität des Saarlandes,
Saarbrücken, Germany
pennacchiotti@coli.uni-sb.de

Alessandro Moschitti
DIT
University of Trento
Povo di Trento, Italy
moschitti@dit.unitn.it

Abstract

In this paper, we briefly describe two enhancements of the *cross-pair similarity* model for learning textual entailment rules: 1) the typed anchors and 2) a faster computation of the similarity. We will report and comment on the preliminary experiments and on the submission results.

1 Introduction

Results of the second RTE challenge (Bar Haim et al., 2006) have suggested that both *deep semantic* models and *machine learning* approaches can successfully be applied to solve textual entailment. The only problem seems to be the size of the knowledge bases. The two best systems (Tatu et al., 2005; Hickl et al., 2005), which are significantly above all the others (more than +10% accuracy), use implicit or explicit knowledge bases larger than all the other systems. In (Tatu et al., 2005), a deep semantic representation is paired with a large amount of general and task specific semantic rules (*explicit knowledge*). In (Hickl et al., 2005), the machine learning model is trained over a large amounts of examples (*implicit knowledge*).

In contrast, Zanzotto&Moschitti (2006) proposed a machine-learning based approach which reaches a high accuracy by only using the available RTE data. The key idea is the *cross-pair similarity*, i.e. a similarity applied to two text and hypothesis pairs which considers the relations between the words in the two texts and between the words in the two hypotheses. This is obtained by using *placeholders* to link the related words. Results in (Bar Haim et al., 2006) are comparable with the best machine learning system when this latter is trained only on the RTE examples.

Given the high potential of the *cross-pair similarity* model, for the RTE3 challenge, we built on it by including some features of the two best systems: 1) we go towards a deeper semantic representation of learning pairs including shallow semantic information in the syntactic trees using *typed placeholders*; 2) we reduce the computational cost of the cross-pair similarity computation algorithm to allow the learning over larger training sets.

The paper is organized as follows: in Sec. 2 we review the cross-pair similarity model and its limits; in Sec. 3, we introduce our model for *typed anchors*; in Sec. 4 we describe how we limit the computational cost of the similarity; in Sec. 5 we present the two submission experiments, and in Sec. 6 we draw some conclusions.

2 Cross-pair similarity and its limits

2.1 Learning entailment rules with syntactic cross-pair similarity

The *cross-pair similarity* model (Zanzotto and Moschitti, 2006) proposes a similarity measure aiming at capturing *rewrite rules* from training examples, computing a *cross-pair similarity* $K_S((T', H'), (T'', H''))$. The rationale is that if two pairs are similar, it is extremely likely that they have the same entailment value. The key point is the use of *placeholders* to mark the relations between the sentence words. A *placeholder* co-indexes two substructures in the parse trees of text and hypothesis, indicating that such substructures are related. For example, the sentence pair, "*All companies file annual reports*" implies "*All insurance companies file annual reports*", is represented as follows:

T_1	(S (NP:$\boxed{1}$ (DT All) (NNS:$\boxed{1}$ companies)) (VP:$\boxed{2}$ (VBP:$\boxed{2}$ file) (NP:$\boxed{3}$ (JJ:$\boxed{3}$ annual) (NNS:$\boxed{3}$ reports))))	(E_1)
H_1	(S (NP:$\boxed{1}$ (DT All) (NNP Fortune) (CD 50) (NNS:$\boxed{1}$ companies)) (VP:$\boxed{2}$ (VBP:$\boxed{2}$ file) (NP:$\boxed{3}$ (JJ:$\boxed{3}$ annual) (NNS:$\boxed{3}$ reports))))	

where the placeholders $\boxed{1}$, $\boxed{2}$, and $\boxed{3}$ indicate the relations between the structures of T and of H.

Placeholders help to determine if two pairs share the same *rewriting rule* by looking at the subtrees that they have in common. For example, suppose we have to determine if "*In autumn, all leaves fall*" implies "*In autumn, all maple leaves fall*". The related co-indexed representation is:

T_2	(S (PP (IN In) (NP (NN:$\boxed{a}$ automn))) (, ,) (NP:$\boxed{b}$ (DT all) (NNS:$\boxed{b}$ leaves)) (VP:$\boxed{c}$ (VBP:$\boxed{c}$ fall)))	(E_2)
H_2	(S (PP (IN In) (NP:$\boxed{a}$ (NN:$\boxed{a}$ automn))) (, ,) (NP:$\boxed{b}$ (DT all) (NN maple) (NNS:$\boxed{a}$ leaves)) (VP:$\boxed{c}$ (VBP:$\boxed{c}$ fall)))	

E_1 and E_2 share the following subtrees:

T_3	(S (NP:$\boxed{x}$ (DT all) (NNS:$\boxed{x}$)) (VP:$\boxed{y}$ (VBP:$\boxed{y}$)))	(R_3)
H_3	(S (NP:$\boxed{x}$ (DT all) (NN) (NNS:$\boxed{x}$)) (VP:$\boxed{x}$ (VBP:$\boxed{x}$)))	

This is the *rewrite rule* they have in common. Then, E_2 can be likely classified as a valid entailment, as it shares the rule with the valid entailment E_1.

The *cross-pair similarity* model uses: (1) a tree similarity measure $K_T(\tau_1, \tau_2)$ (Collins and Duffy, 2002) that counts the subtrees that τ_1 and τ_2 have in common; (2) a substitution function $t(\cdot, c)$ that changes names of the placeholders in a tree according to a set of correspondences between placeholders c. Given $\mathcal{C}$ as the collection of all correspondences between the placeholders of (T', H') and (T'', H''), the cross-pair similarity is computed as:

$$K_S((T', H'), (T'', H'')) = \max_{c \in \mathcal{C}}(K_T(t(T', c), t(T'', c)) + K_T(t(H', c), t(H'', c))) \tag{1}$$

The cross-pair similarity K_S, used in a kernel-based learning model as the support vector machines, allows the exploitation of implicit true and false entailment rewrite rules described in the examples.

2.2 Limits of the syntactic cross-pair similarity

Learning from examples using cross-pair similarity is an attractive and effective approach. However, the cross-pair strategy, as any machine learning approach, is highly sensitive on how the examples are represented in the feature space, as this can strongly bias the performance of the classifier.

Consider for example the following text-hypothesis pair, which can lead to an incorrect rule, if misused.

T_4	*"For my younger readers, Chapman killed John Lennon more than twenty years ago."*	(E_4)
H_4	*"John Lennon died more than twenty years ago."*	

In the basic cross-pair similarity model, the learnt rule would be the following:

T_5	(S (NP:$\boxed{x}$) (VP:$\boxed{y}$ (VBD:$\boxed{y}$) (NP:$\boxed{z}$) (ADVP:$\boxed{k}$)))	(R_5)
H_5	(S (NP:$\boxed{z}$) (VP:$\boxed{y}$ (VBD:$\boxed{y}$) (ADVP:$\boxed{k}$)))	

where the verbs *kill* and *die* are connected by the $\boxed{y}$ placeholder. This rule is useful to classify examples like:

T_6	*"Cows are vegetarian but, to save money on mass-production, farmers fed cows animal extracts."*	(E_6)
H_6	*"Cows have eaten animal extracts."*	

but it will clearly fail when used for:

T_7	*"FDA warns migraine medicine makers that they are illegally selling migraine medicines without federal approval."*	(E_7)
H_7	*"Migraine medicine makers declared that their medicines have been approved."*	

where *warn* and *declare* are connected as generically similar verbs.

The problem of the basic cross-pair similarity measure is that placeholders do not convey the semantic knowledge needed in cases such as the above, where the semantic relation between connected verbs is essential.

2.3 Computational cost of the cross-similarity measure

Let us go back to the computational cost of K_S (eq. 1). It heavily depends on the size of $\mathcal{C}$. We define p' and p'' as the placeholders of, respectively, (T', H') and (T'', H''). As $\mathcal{C}$ is combinatorial with respect to $|p'|$ and $|p''|$, $|\mathcal{C}|$ rapidly grows. Assigning placeholders only to chunks helps controlling their number. For example, in the RTE data the number of placeholders hardly goes beyond 7, as hypotheses are generally short sentences. But, even in these cases, the number of K_T computations grows. As the trees $t(\Gamma, c)$ are obtained from a single tree Γ (containing placeholder) applying different $c \in \mathcal{C}$, it is reasonable to think that they will share common subparts. Then, during the iterations of $c \in \mathcal{C}$, $K_T(t(\Gamma', c), t(\Gamma'', c))$ will compute the similarity between subtrees that have already been evaluated. The reformulation of the *cross-pair similarity* function we present takes advantage of this.

3 Adding semantic information to cross-pair similarity

The examples in the previous section show that the cross-pairs approach lacks the lexical-semantic knowledge connecting the words in a placeholder. In the examples, the missed knowledge is the type of semantic relation between the main verbs. The relation that links *kill* and *die* is not a generic similarity, as a WordNet based similarity measure can suggest, but a more specific causal relation. The learnt rewrite rule R_5 holds only for verbs in such relation. In facts, it is correctly applied in example E_6, as *feed* causes *eat*, but it gives a wrong suggestion in example E_7, since *warn* and *declare* are only related by a generic similarity relation.

We then need to encode this information in the syntactic trees in order to learn correct rules.

3.1 Defining anchor types

The idea of introducing anchor types should be in principle very simple and effective. Yet, this may be not the case: simpler attempts to introduce semantic information in RTE systems have often failed. To investigate the validity of our idea, we then need to focus on a small set of relevant relation types, and to carefully control ambiguity for each type.

A valuable source of relation types among words is WordNet. We choose to integrate in our system three important relation standing at the word level: *part-of*, *antinomy*, and *verb entailment*. We also define two more general anchor types: *similarity* and the *surface matching*. The first type links words which are similar according to some WordNet similarity measure. Specifically, this type is intended to capture the semantic relations of *synonymy* and *hyperonymy*. The second type is activated when words or lemmas match: then, it captures cases in which words are semantically equivalent. The complete set of relation types used in the experiments is given in Table 1.

Rank	***Relation Type***	***Symbol***
1.	*antinomy*	$\leftrightarrow$
2.	*part-of*	$\subset$
3.	*verb entailment*	$\leftarrow$
4.	*similarity*	$\approx$
5.	*surface matching*	$=$

Table 1: Ranked anchor types

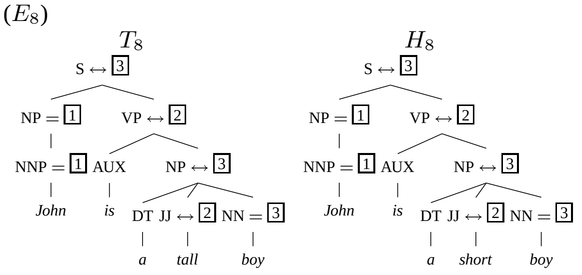
3.2 Type anchors in the syntactic tree

To learn more correct *rewrite rules* by using the anchor types defined in the previous section, we need to add this information to syntactic trees. The best position would be in the same nodes of the anchors. Also, to be more effective, this information should be inserted in as many subtrees as possible. Thus we define the typed-anchor climbing-up rules. We then implement in our model the following climbing up rule:

> *if two typed anchors climb up to the same node, give precedence to that with the highest ranking in Tab. 1.*

This rule can be easily showed to be consistent with common sense intuitions. For an example like "*John is a tall boy*" that does not entail "*John is a short boy*", our strategy will produce these trees:

(E_8)



This representation can be used to derive a correct rewrite rule, such as:

if two fragments have the same syntactic structure $S(NP_1, VP(AUX, NP_2))$, and there is an antonym type ($\leftrightarrow$) on the S and NP_2 , then the

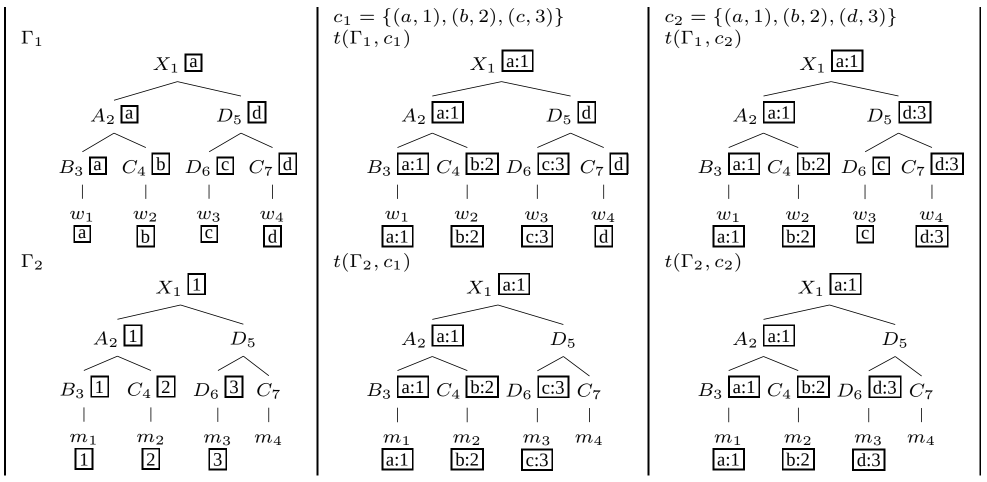


Figure 1: Tree pairs with placeholders and $t(T, c)$ transformation

entailment does not hold.

4 Reducing computational cost of the cross-pair similarity computation

4.1 The original kernel function

In this section, we describe more in detail the similarity function K_S (Eq. 1). To simplify, we focus on the computation of only one K_T of the kernel sum.

$$K_S(\Gamma', \Gamma'') = \max_{c \in \mathcal{C}} K_T(t(\Gamma', c), t(\Gamma'', c)), \quad (2)$$

where the (Γ', Γ'') pair can be either (T', T'') or (H', H''). We apply this simplification since we are interested in optimizing the evaluation of the K_T with respect to different sets of correspondences $c \in \mathcal{C}$.

To better explain K_S, we need to analyze the role of the substitution function $t(\Gamma, c)$ and to review the tree kernel function K_T.

The aim of $t(\Gamma, c)$ is to coherently replace placeholders in two trees Γ' and Γ'' so that these two trees can be compared. The substitution is carried out according to the set of correspondences c. Let p' and p'' be placeholders of Γ' and Γ'', respectively, if $p'' \subseteq p'$ then c is a bijection between a subset $\widehat{p'} \subseteq p'$ and p''. For example (Fig. 1), the trees Γ_1 has $p_1 = \{\boxed{a}, \boxed{b}, \boxed{c}, \boxed{d}\}$ as placeholder set and Γ_2 has $p_2 = \{\boxed{1}, \boxed{2}, \boxed{3}\}$. In this case, a possible set of correspondence is $c_1 = \{(a, 1), (b, 2), (c, 3)\}$. In Fig. 1 the substitution function replaces each placeholder $\boxed{a}$ of the tree $\Gamma_1$with the new placeholder $\boxed{a{:}1}$ by $t(\cdot, c)$ obtaining the transformed tree $t(\Gamma_1, c_1)$, and each placeholder $\boxed{1}$ of Γ_2 with $\boxed{a{:}1}$. After these substitutions, the labels of the two trees can be matched and the similarity function K_T is applicable.

$K_T(\tau', \tau'')$, as defined in (Collins and Duffy, 2002), computes the number of common subtrees between τ' and τ''.

4.2 An observation to reduce the computational cost

The above section has shown that the similarity function K_S firstly applies the transformation $t(\cdot, c)$ and then computes the tree kernel K_T. The overall process can be optimized by factorizing redundant K_T computations.

Indeed, two trees, $t(\Gamma, c')$ and $t(\Gamma, c'')$, obtained by applying two sets of correspondences $c', c'' \in \mathcal{C}$, may partially overlap since c' and c'' can share a non-empty set of common elements. Let us consider the subtree set S shared by $t(\Gamma, c')$ and $t(\Gamma, c'')$ such that they contain placeholders in $c' \cap c'' = c$, then $t(\gamma, c) = t(\gamma, c') = t(\gamma, c'') \; \forall \gamma \in S$. Therefore if we apply a tree kernel function K_T to a pair (Γ', Γ''), we can find a c such that subtrees of Γ' and subtrees of Γ'' are invariant with respect to c' and c''. Therefore, $K_T(t(\gamma', c), t(\gamma'', c)) = K_T(t(\gamma', c'), t(\gamma'', c'))$ $= K_T(t(\gamma', c''), t(\gamma'', c''))$. This implies that it is possible to refine the dynamic programming algorithm used to compute the Δ matrices while computing the kernel $K_S(\Gamma', \Gamma'')$.

To better explain this idea let us consider Fig. 1 that represents two trees, Γ_1 and Γ_2, 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 column of the figure represents the original trees Γ_1 and Γ_2. The second and third columns contain respectively the transformed trees $t(\Gamma, c_1)$ and $t(\Gamma, c_2)$

Since the subtree of Γ_1 starting from $A_2\boxed{a}$ contains only placeholders that are in c, in the transformed trees, $t(\Gamma_1, c_1)$ and $t(\Gamma_1, c_2)$, the subtrees rooted in $A_2\boxed{\text{a:1}}$ are identical. The same happens for Γ_2 with the subtree rooted in $A_2\boxed{1}$. In the transformed trees, $t(\Gamma_2, c_1)$ and $t(\Gamma_2, c_2)$, subtrees rooted in $A_2\boxed{\text{a:1}}$ are identical. The computation of K_T applied to the above subtrees gives an identical result. Then, this computation can be avoided. If correctly used in a dynamic programming algorithm, the above observation can produce an interesting decrease in the time computational cost.

5 Experimental Results

5.1 Experimental Setup

We implemented the novel cross-similarity kernel in the SVM-light-TK (Moschitti, 2006) that encodes the basic syntactic kernel K_T in SVM-light (Joachims, 1999).

To assess the validity of the typed anchor model (**tap**), we evaluated two sets of systems: the *plain* and *lexical-boosted* systems. The *plain* systems are:
-tap: our tree-kernel approach using typed placeholders with climbing in the syntactic tree;
-tree: the cross-similarity model described in Sec.2. Its comparison with *tap* indicates the effectiveness of our approaches;
The *lexical-boosted* systems are:
-lex: a standard approach based on *lexical overlap*. The classifier uses as the only feature the lexical overlap similarity score described in (Corley and Mihalcea, 2005);
-lex+tap: these configurations mix lexical overlap and our typed anchor approaches;
-lex+tree: the comparison of this configuration with *lex+tap* should further support the validity of our intuition on typed anchors;

Preliminary experiments have been performed using two datasets: **RTE2** (the 1600 entailment pairs from the RTE-2 challenge) and **RTE3d** (the development dataset of this challenge). We randomly divided this latter in two halves: $RTE3d_0$ and $RTE3d_1$.

5.2 Investigatory Results Analysis and Submission Results

Table 2 reports the results of the experiments. The first column indicates the training set whereas the second one specifies the used test set. The third and the forth columns represent the accuracy of basic models: the original *tree* model and the enhanced *tap* model. The latter three columns report the basic *lex* model and the two combined models, *lex+tree* and *lex+tap*. The second and the third rows represent the accuracy of the models with respect to the first randomly selected half of $RTE3d$ whilst the last two rows are related to the second half.

The experimental results show some interesting facts. In the case of the **plain systems** (*tree* and *tap*), we have the following observations:
- The use of the *typed anchors* in the model seems to be effective. All the *tap* model results are higher than the corresponding *tree* model results. This suggests that the method used to integrate this kind of information in the syntactic tree is effective.
- The claim that *using more training material helps* seems not to be supported by these experiments. The gap between $tree$ and tap is higher when learning with $RTE2 + RTE3d_0$ than when learning with $RTE3_0$. This supports the claim. However, the result is not kept when learning with $RTE2 + RTE3d_1$ with respect to when learning with $RTE3_1$. This suggests that adding not very specific information, i.e. derived from corpora different from the target one (RTE3), may not help the learning of accurate rules.

On the other hand, in the case of the **lexical-boosted** systems (*lex*, *lex+tree*, and *lex+tap*), we see that:
- There is an extremely high result for the pure *lex* model. Even if this model makes use of semantic information and word relevance information (i.e.,

Train	*Test*	**tree**	**tap**	**lex**	**lex+tree**	**lex+tap**
$RTE3d_0$	$RTE3d_1$	62.97	64.23	69.02	68.26	69.02
$RTE2 + RTE3d_0$	$RTE3d_1$	62.22	62.47	71.03	71.28	71.79
$RTE3d_1$	$RTE3d_0$	62.03	62.78	70.22	70.22	71.22
$RTE2 + RTE3d_0$	$RTE3d_0$	63.77	64.76	71.46	71.22	72.95

Table 2: Accuracy of the systems on two folds of RTE3 development

the inverse document frequency), this result is counterintuitive. Examples on the RTE3d set are extracted from NLP systems solving a particular task (e.g., QA and IE). We expect that these systems use models like the *lex* approach to select relevant passages. Then, positive and negative examples should have similar values for this *lex* distance indicator. It is then not clear why this model results in so high accuracy.
- Given the high results of the *lex* model, the model *lex+tree* does not increase the performances.
- On the contrary, the model *lex+tap* is always better (or equal) than the *lex* model. This suggests that for this particular set of examples the *typed anchors* are necessary to effectively use the *rewriting rules* implicitly encoded in the examples.
- When the *tap* model is used in combination with the *lex* model, it seems that the claim "*the more training examples the better*" is valid. The gaps between *lex* and *lex+tap* are higher when the $RTE2$ is used in combination with the $RTE3d$ related set.

Given this analysis we submitted two systems both based on the *lex+tap* model. We did two different training: one using $RTE3d$ and the other using $RTE2 + RTE3d$. Results are reported in the Table below:

Train	*Accuracy*
$RTE3d$	66.75%
$RTE2 + RTE3d$	65.75%

Such results seem too low to be statistically consistent with our development outcome. This suggests that there is a clear difference between the content of $RTE3d$ and the $RTE3$ test set. Moreover, in contrast with what expected, the system trained with only the $RTE3d$ data is more accurate than the others. Again, this suggests that the RTE corpora (from all the challenges) are most probably very different.

6 Conclusions and final remarks

This paper demonstrates that it is possible to effectively include shallow semantics in syntax-based learning approaches. Moreover, as it happened in RTE2, it is not always true that more learning examples increase the accuracy of RTE systems. This claim is still under investigation.

References

Roy Bar Haim, Ido Dagan, Bill Dolan, Lisa Ferro, Danilo Giampiccolo, Bernardo Magnini, and Idan Szpektor. 2006. The II PASCAL RTE challenge. In *PASCAL Challenges Workshop*, Venice, Italy.

Michael Collins and Nigel Duffy. 2002. New ranking algorithms for parsing and tagging: Kernels over discrete structures, and the voted perceptron. In *Proceedings of ACL02*.

Courtney Corley and Rada Mihalcea. 2005. Measuring the semantic similarity of texts. In *Proc. of the ACL Workshop on Empirical Modeling of Semantic Equivalence and Entailment*, pages 13–18, Ann Arbor, Michigan, June. Association for Computational Linguistics.

Andrew Hickl, John Williams, Jeremy Bensley, Kirk Roberts, Bryan Rink, and Ying Shi. 2005. Recognizing textual entailment with LCCs GROUNDHOG system. In *Proceedings of the Second PASCAL Challenges Workshop on Recognising Textual Entailment*, Venice, Italy.

Thorsten Joachims. 1999. Making large-scale svm learning practical. In B. Schlkopf, C. Burges, and A. Smola, editors, *Advances in Kernel Methods-Support Vector Learning*. MIT Press.

Alessandro Moschitti. 2006. Making tree kernels practical for natural language learning. In *Proceedings of EACL'06*, Trento, Italy.

Marta Tatu, Brandon Iles, John Slavick, Adrian Novischi, and Dan Moldovan. 2005. COGEX at the second recognizing textual entailment challenge. In *Proceedings of the Second PASCAL Challenges Workshop on Recognising Textual Entailment*, Venice, Italy.

Fabio Massimo Zanzotto and Alessandro Moschitti. 2006. Automatic learning of textual entailments with cross-pair similarities. In *Proceedings of the 21st Coling and 44th ACL*, pages 401–408, Sydney, Australia, July.