Dependency Tree Kernels for Relation Extraction from Natural Language Text

Frank Reichartz, Hannes Korte, and Gerhard Paass

Fraunhofer IAIS, Schloss Birlinghoven, 53754 St. Augustin, Germany

Abstract. The automatic extraction of relations from unstructured natural text is challenging but offers practical solutions for many problems like automatic text understanding and semantic retrieval. Relation extraction can be formulated as a classification problem using support vector machines and kernels for structured data that may include parse trees to account for syntactic structure. In this paper we present new tree kernels over dependency parse trees automatically generated from natural language text. Experiments on a public benchmark data set show that our kernels with richer structural features significantly outperform all published approaches for kernel-based relation extraction from dependency trees. In addition we optimize kernel computations to improve the actual runtime compared to previous solutions.

1 Introduction

Current search engines usually are not effective for complex queries, e.g. "composers born in Berlin". The retrieved documents among others contain composers who stayed some time in Berlin or have the name "Berlin". Obviously the internal representation of text in a search index as a sequence of words is insufficient to recover semantics from unstructured text. An important step towards automatic knowledge discovery is to extract semantic relations between entities.

Information extraction tackles this goal in two steps. First entity or phrase taggers detect objects of different types, such as persons, descriptions or pronouns, mentioned in the text. Some of these techniques have reached a sufficient performance level on many datasets [18]. They offer the basis for the next step: the extraction of relations that exist between the recognized entities, e.g. composer-born-in(John White, Berlin).

An early approach to relation extraction is based on patterns [6], usually expressed as regular expressions for words with wildcards. The underlying hypothesis assumes that terms sharing similar linguistic contexts are connected by similar semantic relations. Various authors follow this approach, e.g. [1] use frequent itemset mining to extract word patterns and [7] employ logic-based frequent structural patterns for relation extraction.

Syntactic parse trees provide extensive information on syntactic structure and can, for instance, represent the relation between subject, verb and object in a sentence. For feature-based methods only a limited number of structural details may be compared. On the other hand, kernel-based methods offer efficient

W. Buntine et al. (Eds.): ECML PKDD 2009, Part II, LNAI 5782, pp. 270–285, 2009.

solutions that allow to explore a much larger (often exponential, or in some cases, infinite) characteristics of trees in polynomial time, without the need to explicitly represent the features. [20] and [4] proposed kernels for dependency trees inspired by string kernels. [2] investigated a kernel that computes similarities between nodes on the shortest path of a dependency tree that connect the entities. All these kernels are used as input for a kernel classifier.

In this paper we extend current dependency tree kernels by including richer structural features. To tackle the different shortcomings of previous work we use the ordering properties as well as the labeling of nodes in dependency trees in a novel fashion to create kernels which consider most of the available information in dependency trees. To allow the usage of more substructure properties while maintaining an acceptable runtime we propose two new computation algorithms tailored for relation extraction tree kernels. Our new kernels are shown to outperform all previously published kernels in classification quality by a significant margin on a public benchmark. Our kernels reach F-measures of $77\% - 80\%$ on selected relations which is sufficient for some applications like information retrieval.

The remainder of the paper is organized as follows. In the next section we describe dependency parse trees used for relation classification. Subsequently we give a generic description of the current dependency parse trees in the literature. The following two sections outline our new kernels for relation extraction, the All-Pairs Dependency Tree Kernel as well as the Dependency Path Tree Kernel. Next we describe different versions of the algorithms optimized for efficiency. For the experiments we re-implemented existing kernels and compare them to our new kernels on a benchmark dataset. We close with a summary and conclusions.

2 Dependency Parse Trees

A dependency tree is a structured representation of the grammatical dependency between the words of a sentence by a labeled directed tree [11]. Structure is determined by the relation between a word (a head) and its dependents. The dependent in turn can be the head for other words yielding a tree structure. Each node in the tree corresponds to a word in the sentence with arrows pointing from the head to the dependents.

Dependency trees may be *typed*, specializing the "dependent" relation into many subtypes, e.g. as "auxiliary", "subject", "object", while in the untyped case there is only a "dependent" relation. In this paper we consider untyped dependency trees only, generated by the *Stanford Parser* [10] from the sentences of a text. As an example consider the two sentences $a =$ "Recently Obama became the president of the USA" and $b =$ "Ballmer is the CEO of Microsoft" which have the tree representations as shown in figure 1.

2.1 Notation

Let $w = w_1\, w_2\, \ldots\, w_n$ be a sequence of words, a natural language sentence with words $w_j \in \mathcal{W}$. The parser will generate a representation of the sentence w as

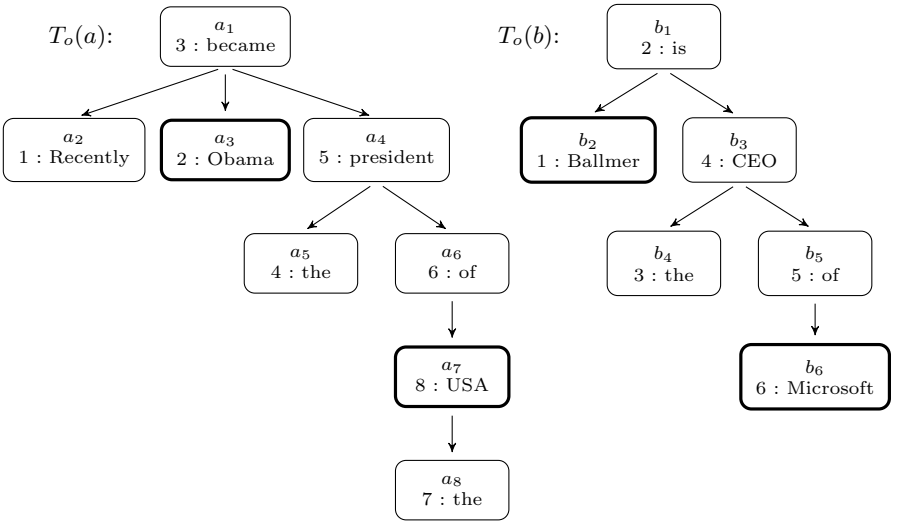

Fig. 1. The dependency trees of the two example sentences. The nodes are labeled with the position of the word in the sentence and the word itself. A thick border marks an entity mention.

a labeled rooted connected tree $T(w) = (V, E)$ with nodes $V = \{v_1, \ldots, v_n\}$ and edges in $E \subset V \times V$. Each node v_i is labeled with a corresponding word $w_{\rho(v_i)}$ in the sentence w, where $\rho(v_i)$ is a bijective function mapping a node v_i to the index j of it corresponding word $w_j \in \mathcal{W}$ in w. For example in figure 1 we have $\rho(a_4) = 5$. For each node $v \in V$ the ordered sequence of its m children $ch(v) = (u_1, \ldots, u_m)$ with $u_i \in V$ satisfies $\rho(u_i) < \rho(u_{i+1})$ for all $i \in \{1, \ldots, m-1\}$. The node $a_1 =$"became" in figure 1, for instance, has

$$ch(a_1) = (a_2 = \text{“Recently”}, a_3 = \text{“Obama”}, a_4 = \text{“president”})$$

as ordered sequence of child nodes. With the order induced by ρ we get an ordered tree $T_o(w)$.

For a sequence $s = (s_j)_{j \in [1:k]}$ of length k the set of all possible subsequence index sets can be denoted as I_k. Each ordered sequence $o(i) = \mathbf{i} = (i_1, \ldots, i_m)$ of a subset $i \subset \mathbf{I}$ of the indices $\mathbf{I}$ of a sequence $s = (s_j)_{j \in \mathbf{I}}$ implies a subsequence $(s_{i_1}, \ldots, s_{i_m})$ which we denote by $s(\mathbf{i})$, which is inline with the notation of [17]. Therefore we can write the subsequence of children referenced by $\mathbf{i}$ from a node v of an ordered tree as $ch(v, \mathbf{i})$. In figure 1, for example, $ch(a_1) = (a_2, a_3, a_4)$ has the index set $\{1, 2, 3\}$. The subset $\{1, 3\}$ in its representation as ordered sequence $(1, 3)$ defines the subsequence $ch(a_1, (1, 3)) = (a_2 =$ "Recently", $a_4 =$ "president") of child nodes of a_1.

We consider relations connecting entities or objects, which in this paper are collectively called entities. We assume that these entities have been extracted in a prior step, e.g. by named entity recognition tools. They are treated as a single

word in the sentence w, e.g. the two words "Barack Obama" will be merged to a new word "Barack_Obama". Note that there may be different types τ of entities relevant for a relation, e.g. persons, locations or organisations which can be used as a feature associated with a entity.

Let $e(w) = (e_1(w), \ldots, e_m(w))$ be the subsequence of w of all entity mentions in w, i.e. each word $e_i(w)$ is an entity mention. Let R be a binary target relation between entities. An *instance* of the relation R in a sentence w is given by $(w, (e_i(w), e_j(w)))$, $i \neq j$, for two entity mentions $e_i(w)$ and $e_j(w)$ in $e(w)$ if the sentence semantically supports the relation between the two entity mentions, i.e. $(e_i(w), e_j(w)) \in R$. It is important to see that the same sentence w can support the same relation R between different pairs of entity mentions. The pairs $(e_i(w), e_j(w))$ of different components of $e(w)$ where the relation does not hold are negative examples of the target relation.

The example sentence a in figure 1 contains the two entities $e_1(a) = a_2 =$ "Obama" and $e_2(a) = a_8 =$ "USA" whereas sentence b contains $e_1(b) = b_1 =$ "Ballmer" and $e_2(b) = b_6 =$ "Microsoft". For the target relation *role*, where a person entity e_1 has some role in an organization entity e_2, both sentences contain a positive example e.g. $(e_1(a), e_2(a)) \in role$, which may be used as training examples.

In a rooted tree $T = (V, E)$ with root $r(T) \in V$ the *lowest common ancestor* $lca(v, w)$ of two nodes $v, w \in V$ is the lowest node in T which has both v, w as descendents. We define the set of nodes $sub(v, w)$ of two nodes v, w as the set containing all nodes of complete subtree of a tree T rooted in $lca(v, w)$. In a directed tree T we define the implied path $p(u, v)$ between two nodes u, v as the sequence $(u, \ldots, lca(u, v), \ldots, v)$ of nodes where $u, \ldots$ are the nodes on the path from the $lca(u, v)$ to u , analogous for v.

Relation extraction aims at learning a relation R from a number of positive and negative instances of the relation in natural language sentences. As a classifier we use Support Vector Machines (SVMs) [8]. They can compare complex structures, e.g. trees, by kernels, which efficiently evaluate the relevant differences. Given the kernel function, the SVM tries to find a hyperplane that separates positive from negative examples of the relation and at the same time maximizes the separation (margin) between them. This type of max-margin separator has been shown both empirically and theoretically resistant to overfitting and to provide good generalization performance on new examples.

3 Dependency Tree Kernel

The *Dependency Tree Kernel* (DTK) [4] is based on the work of [20]. A *node kernel* $\Delta(u, v)$ measures the similarity of two tree nodes u, v and its substructures. Then the tree kernel can be expressed as the node kernel of the two corresponding root nodes. Nodes may be described by different features, which are compiled in table 1. This is the feature set of [4], where *word* is the word itself, *POS* and *general-POS* are part-of-speech-tags, the *chunk tag* is the phrase the word is contained in, like noun or verb phrases. If the word describes an entity, the corresponding entity type and the mention is provided. For example "Obama" is

Table 1. List of features which may be assigned to every node in a tree. The index corresponds to k in f_v^k, hence f_v^5 denotes the "entity type" of the node v.

Feature Index k	Feature Name	Possible Feature Values
1	word	*Obama, president,...*
2	POS	NN, NNP,...
3	general-POS	noun, verb, adj,...
4	chunk tag	NP, VP, ADJP,...
5	entity type	person, location, GPE,...
6	mention type	name, nominal, pronoun
7	WordNet hypernym	corporate executive, city,...
8	relation argument	1, 2 or n/a

an entity with entity type "person" and mention type "name". If this entity is part of the current relation, the *relation argument* is set to 1 or 2 depending on the entity's role in the relation. Finally the WordNet hypernym is provided if there is one. That is a superordinate grouping word, e.g. the hypernym of "red" is "color". Unfortunately, in [4] the authors do not describe how the hypernyms are selected if there are more than one, which is a problem, because most words have different meanings, e.g. "bank" (credit institute or furniture). To tackle this problem an automatic disambiguation of words[15] is needed. So, in our experiments we exclude this feature. A specific subset of feature indices may be designated by the set f^m.

To compare the relations in two instances $X = (x, (x_1, x_2)), Y = (y, (y_1, y_2))$ [4] proposes to compare the subtrees induced by the relation arguments x_1, x_2 and y_1, y_2 in the two sentences, i.e. the least common ancestor of the relation argument nodes

$$K_{\mathrm{DTK}}(u, v) = \Delta(lca(x_1, x_2), lca(y_1, y_2)) \tag{1}$$

In order to compute the node kernel Δ, we first define a matching function $mat(u, v) \rightarrow \{0, 1\}$ which checks whether two nodes u, v are comparable at all.

$$mat(u, v) = \begin{cases} 1 & \text{if } \forall k \in f^m : f_u^k = f_v^k \\ 0 & \text{otherwise} \end{cases}$$

where f_v^k is the k-th feature of the node v. The set f^m consists of the indices of the features used in the matching function. Based on observations in [4] this set is defined as $f^m = \{3, 5, 8\}$. Thus, two nodes u and v match, if the features *general-POS*, *entity type* and *relation argument* are equal.

The similarity function $sim(u, v) \rightarrow (0, \infty]$ counts the number of common features of the two nodes. In [4] there are 8 features (Table 1), hence $d = 8$ but in our setup $d = 7$ because the hypernym feature is not considered.

$$sim(u, v) = \sum_{k=1}^{d} comp(f_u^k, f_v^k) \qquad comp(f_u^k, f_v^k) = \begin{cases} 1 & \text{if } f_u^k = f_v^k \\ 0 & \text{otherwise} \end{cases}$$

We define the node kernel $\Delta(u, v)$ over two nodes u and v as the sum of the node similarity and their children similarity $C(ch(u), ch(v))$

$$\Delta(u,v) \begin{cases} 0 & \text{if } mat(u,v) = 0 \\ sim(u,v) + C(ch(u), ch(v)) & \text{otherwise} \end{cases} \tag{2}$$

The *child subsequence kernel* $C(s, t)$ uses a modified version of the *String Subsequence Kernel* of [17] to recursively compute the sum of node kernel values of subsequences of node sequences s and t:

$$C(s,t) = \sum_{\substack{\mathbf{i} \in I_{|s|},\ \mathbf{j} \in I_{|t|}, \\ |\mathbf{i}| = |\mathbf{j}|}} \lambda^{d(\mathbf{i})+d(\mathbf{j})} \quad \Delta'(s(\mathbf{i}), t(\mathbf{j})) \quad M(s(\mathbf{i}), t(\mathbf{j})) \tag{3}$$

where s, t are sequences of nodes and $s(\mathbf{i})$ and $t(\mathbf{j})$ are the subsequences of nodes implied by the index sets $\mathbf{i}, \mathbf{j}$. Furthermore the parameter $0 < \lambda \leq 1$ is a penalty factor for lengths and gaps of the subsequences, and $d(\mathbf{i}) = max(\mathbf{i}) - min(\mathbf{i}) + 1$ is the covered distance of the index sequence $\mathbf{i}$. The function Δ' over two node sequences of length n computes the sum over the node kernels

$$\Delta'\left((s_1, \ldots, s_n), (t_1, \ldots, t_n)\right) = \sum_{k=1}^{n} \Delta\left(s_k, t_k\right)$$

Furthermore, M represents a filter for non-matching sequences:

$$M((s_1, \ldots, s_n), (t_1, \ldots, t_n)) = \prod_{k=1}^{n} mat(s_k, t_k)$$

Less formally, the function $C(s, t)$ sums up the similarities of all subsequences in which every node matches its corresponding node. The similarity of a sequence is given by the sum of the single node similarities provided by $\Delta(u, v)$.

4 Extended Kernels

The Dependency Tree kernel forms the foundation for our proposed kernels considering a richer feature space described in the following section.

4.1 All-Pairs Dependency Tree Kernel

The dependency tree kernel [4] discards a possible relation if only one node in the subsequence does not "match" its corresponding node. We propose to use all pairs of possible combinations in the subtrees to tackle this restriction of the dependency tree kernel which can be done by generalizing the approaches of [14].

We define the All-Pairs Dependency Tree Kernel (All-Pairs-DTK), which sums up the node kernels of all possible combinations of nodes contained in the two subtrees implied by the relation argument nodes as

$$K_{\text{All-Pairs}}(X, Y) = \sum_{u \in V_x} \sum_{v \in V_y} \Delta(u, v) \tag{4}$$

where $V_x = sub(x_1, x_2)$ and $V_y = sub(y_1, y_2)$ are sets containing the nodes of the complete subtrees rooted at the respective lowest common ancestors. The consideration of all possible pairs of nodes ensure that no valuable information which resides in subtrees in which only the corresponding root does not match is discarded. However this could lead to the problem that some irrelevant subtrees are also compared which in turn leads to some noise in the kernel computation.

This Kernel is structurally comparable to the Partial Tree Kernel (PTK) described in [14]. The main difference is that the PTK does not support feature vectors linked to nodes and therefore discards important information available for our task (see table 1 for features associated with nodes). Another important difference is that our node kernel computes the sum of all node kernels of the matching subsequences, while the PTK builds a product of the node kernels.

Because Δ computes the similarity of two nodes and their substructures, Δ is likely to be called multiple times for the same nodes during the computation of $K_{\text{All-Pairs}}$. Due to its computational cost we implemented a cache to limit the number of times Δ has to be calculated to exactly $|V_x| \cdot |V_y|$ times.

It is clear that $K_{\text{All-Pairs}}$ is a kernel, because it is a sum and product of valid kernels, which has been shown to be also a valid kernel. [17]

4.2 Dependency Path Tree Kernel

In [2] the authors argue that the information expressing a relation between two entities is almost exclusively concentrated in the dependency path between them. Motivated by this they propose the Shortest Path Kernel for relation extraction. This kernel compares the nodes on the path between the two relation argument entities in the dependency tree. Our experiments revealed that this kernel performs almost as well as the Dependency Tree Kernel which shows that the path contains almost as much information as the whole subtree. However this kernel has some restrictions, like the condition that the path in both trees needs to have the same length, as otherwise the relation is discarded. Moreover the kernel computes a product of the node similarities, yielding zero similarity if only one node pair is completely dissimilar.

To avoid these constraints we propose a second new kernel combining the SPK with the DTK utilizing the main ideas of the Subsequence Kernel [3],[17]. The Dependency Path Tree Kernel (Path-DTK) not only measures the similarity of the root nodes and its descendents as in [4] or only the similarities of nodes on the path [2], it considers the similarities of all nodes (and substructures) using the node kernel on the path connecting the two relation argument entity nodes. To this end the pairwise comparison is performed using the ideas of the

subsequence kernel from [17], therefore relaxing the "same length" restriction of the SPK.

In order to integrate these features into our kernel we need to define how the nodes on the dependency path contribute to the overall kernel. We use a variant of the Subsequence Kernel for the computation of the path subsequence sums of the single node kernels. In our benchmark training data (ACE 2003) the dependency paths include up to 28 nodes. So for a computationally feasible solution we need a limitation on the maximum subsequence length and hence the number of possible subsequences. We introduce an additional parameter q which acts as an upper bound on the index distance $d(\mathbf{i})$. In contrast to a limitation on the length of a sequence, this allows us to compute much longer subsequences because the number of possible combinations is very much reduced.

Besides q we define another parameter μ with $0 < \mu \leq 1$ as a factor that penalizes gaps on the path. The Dependency Path Tree Kernel is then defined as:

$$K_{\text{Path-DTK}}(X,Y) = \sum_{\substack{i \in I_{|x|},\, j \in I_{|y|},\\ |\mathbf{i}|=|\mathbf{j}|,\, d(\mathbf{i}),d(\mathbf{j}) \leq q}} \mu^{d(\mathbf{i})+d(\mathbf{j})} \quad \Delta'(x(\mathbf{i}), y(\mathbf{j})) \quad M(x(\mathbf{i}), y(\mathbf{j})) \quad (5)$$

where x and y are the implied paths $path(x_1, x_2)$ and $path(y_1, y_2)$ between the relation arguments of the instances and $x(\mathbf{i})$ is the subsequence of x implied by the ordered sequence $\mathbf{i} = o(i)$ of the index set i analogous for j. In the examples in figure 1 this leads to the dependency paths $x = (a_3, a_1, a_4, a_6, a_7)$ and $y = (b_2, b_1, b_3, b_5, b_6)$.

The Dependency Path Tree Kernel effectively compares the nodes from paths with different lengths while maintaining the ordering information and considering the similarities of substructures. The path as well as the substructures turned out to hold valuable information in dependency trees. The Dependency Path Tree Kernel accounts for both which makes it a flexible kernel for relation extraction.

This kernel $K_{\text{Path-DTK}}$ also consists of only sums and products of a valid kernel [20], so $K_{\text{Path-DTK}}$ is also a valid kernel. [17]

5 Efficient Computation

The runtime of the kernels is highly dependent on the algorithm used to count the matching subsequences. To reduce the exponential runtime needed for naive enumeration [20] adapted the recursive algorithm of the String Subsequence Kernel [12] to dependency trees. This algorithm needs $O(mn^3)$ operations for sequence lengths n, m with $n \leq m$. Another approach to the subsequence computation for strings running in $O(mn^2)$ was proposed by [17]. We adapted this solution for the computation of $C(s,t)$ as denoted in pseudocode in algorithm 1 which leads to an $O(mn^2)$ solution for the computation of the child node subsequences $C(s,t)$.

However, because we observed in our benchmark data an average inner node child count of 1.72, another approach based on a caching strategy is useful. Let

Algorithm 1. Compute $C(s,t)$ in $\mathrm{O}(mn^2)$ with sequence lengths $n \leq m$

$k \leftarrow 0,\ \ m \leftarrow |s|,\ \ n \leftarrow |t|,\ \ p \leftarrow min(m,n)$
for $i = 1 : m$ **do**
  for $j = 1 : n$ **do**
    if $mat(s(i), t(j)) = 1$ **then**
      $DPS(i,j) \leftarrow \lambda^2$
      $DQS(i,j) \leftarrow \lambda^2\ \Delta(s(i), t(j))$
      $k \leftarrow k + DQS(i,j)$
    else
      $DPS(i,j) \leftarrow 0$
      $DQS(i,j) \leftarrow 0$
    end if
  end for
end for
$DP(0 : m, 0) \leftarrow 0,\ \ DQ(0 : m, 0) \leftarrow 0,\quad DP(0, 1 : n) \leftarrow 0,\quad DQ(0, 1 : n) \leftarrow 0$
for $l = 2 : p$ **do**
  for $i = 1 : m$ **do**
    for $j = 1 : n$ **do**
      $DP(i,j) \leftarrow DPS(i,j) + \lambda\ DP(i-1,j) + \lambda\ DP(i,j-1) - \lambda^2\ DP(i-1,j-1)$
      $DQ(i,j) \leftarrow DQS(i,j) + \lambda\ DQ(i-1,j) + \lambda\ DQ(i,j-1) - \lambda^2\ DQ(i-1,j-1)$
      if $mat(s(i), t(j)) = 1$ **then**
        $DPS(i,j) \leftarrow \lambda^2\ DP(i-1, j-1)$
        $DQS(i,j) \leftarrow \lambda^2\ DQ(i-1, j-1) + \Delta(s(i), t(j))\ DPS(i,j)$
        $k \leftarrow k + DQS(i,j)$
      end if
    end for
  end for
end for
return k

Algorithm 2. Compute $C(s,t)$ with prepared index sequences SI

$k \leftarrow 0,\ \ m \leftarrow |s|,\ \ n \leftarrow |t|,\ \ p \leftarrow min(m,n)$
for $q = 1 : p$ **do**
  for $\mathbf{i} \in SI_{m,q}$ **do**
    for $\mathbf{j} \in SI_{n,q}$ **do**
      $x \leftarrow 0$
      for $r = 1 : q$ **do**
        if $mat(s(\mathbf{i}(r)), t(\mathbf{j}(r))) = 0$ **then**
          goto 4 // jump to next index sequence **j**
        end if
        $x \leftarrow x + \Delta(s(\mathbf{i}(r)), t(\mathbf{j}(r)))$
      end for
      $k \leftarrow k + x\ \lambda^{d(\mathbf{i})+d(\mathbf{j})}$
    end for
  end for
end for
return k

Table 2. Averaged over 5 measurements, this table shows the relative empirical runtimes of the DTK with the different subsequence algorithms

Algorithm	Theoretical runtime	Relative empirical runtime	Std. dev.
Algorithm 2	$\sum_{q=1}^{n} q \binom{m}{q} \binom{n}{q}$	1	–
Algorithm 1	$mn + mn^2$	3.2204	0.0345
Zelenko	mn^3	2.9338	0.0363

$SI_{p,q} = \{o(i)|i \in I_p \wedge |i| \leq q\}$ be the set of all ordered index subsequences of length q with highest index p. We can write the child subsequence kernel $C(s,t)$ as

$$C(s,t) = \sum_{q=1}^{min(m,n)} \sum_{\mathbf{i} \in SI_{m,q}} \sum_{\mathbf{j} \in SI_{n,q}} \lambda^{d(\mathbf{i})+d(\mathbf{j})} \; \Delta'(s(\mathbf{i}), t(\mathbf{j})) \; M(s(\mathbf{i}), t(\mathbf{j})) \quad (6)$$

with $m = |s|$ and $n = |t|$. This reformulation of the child subsequence kernel allows an efficient computation by utilizing a cache for the Δ function. The computation procedure is described in algorithm 2. It is easy to see that the worst case runtime (if all nodes match) of this algorithm for $n \leq m$ is

$$\mathrm{O}\left(\sum_{q=1}^{n} q \cdot \binom{n}{q} \cdot \binom{m}{q}\right)$$

However, for $n, m \leq 4$ this approach needs less operations than the $\mathrm{O}(mn^3)$ solution proposed by [20]. For $n, m \leq 3$ it also needs less operations than algorithm 1, which has an exact runtime of $\mathrm{O}(mn + mn^2)$. For 95% of the occurring calculations of $C(s,t)$ during the kernel computations on the benchmark dataset it holds that $n, m \leq 3$. Our experiments on the benchmark data set show that – though it explicitly enumerates all subsequence combinations – algorithm 2 has a better practical runtime than the two other solutions, as can be seen in table 2. Another interesting result is that the theoretically fastest approach (Algorithm 1) is even slower than the Zelenko algorithm on our dataset.

6 Related Work

There are several ways to tackle the problem of relation extraction. Prominent solution strategies have been mentioned in the introduction. Among the first who proposed kernel methods for relation extraction on shallow parse trees was [20]. The resulting kernel was then generalized to be able to handle dependency trees by [4]. [2] suggested a kernel for relation extraction which only considers the shortest path between the two entity nodes which nevertheless yielded good performance. Besides work on tree kernels for relation extraction there have

been tree kernels proposed for other tasks like question classification [14] which have shown improvements over bag-of-words approaches. Considering features associable with words [5] proposed to use semantic background knowledge from various sources like WordNet, FrameNet and PropBank to enhance the DTK of [4] and showed good results.

7 Experiments

In this section we present the results of the experiments with kernel-based methods for relation extraction. Throughout this section we will compare our approaches with other state-of-the-art kernels considering their classification quality on the publicly available benchmark dataset ACE-2003 [13] which has been commonly used for the evaluation of relation extraction systems in previous work.

7.1 Technical Realization

We implemented our tree-kernels in Java and used Joachim's [9] SVM^{light} with the JNI Kernel Extension[1]. Each experiment was splitted into different tasks to allow distributed processing for better efficiency, e.g. each fold of the cross validation was computed on a different machine. Depending on the kernel and the relation this resulted in training times of minutes for the simple kernels to 12 hours for the advanced kernels. We also employed a standard grid-search to optimize the parameters of all kernels as well as the SVM-parameter C. The parameter optimization followed a strict protocol of exclusively utilizing training data to avoid a biased comparison. We only report the best configurations in the result tables. Because the implementations of the other kernels were unavailable and the experimental setup is critical for a fair comparison of the different kernels we implemented all other state of the art kernels for relation extraction, using the implementation details from the original papers. This allows for a comprehensive comparison of our approach with state-of-the-art kernels.

7.2 Benchmark Dataset

The ACE-2003 corpus consists of 519 news documents from different sources splitted in a test and training set containing 176825 words in 9256 sentences. In all documents entities and the relations between them were annotated by humans annotators. The entities are annotated by the types *named* (e.g. "Barack Obama") , *nominal* (e.g. "government") and *pronominal* (e.g. "he"). There are 5 top level relation types *role*, *part*, *near*, *social* and *at* (see table 3), which are further differentiated into 24 subtypes.

[1] Available from www.aifb.uni-karlsruhe.de/WBS/sbl/software/jnikernel/

Table 3. A list of relation types and the respective frequencies in the ACE-2003 corpus. On the left side the number of relations between named entities is counted.

Relation	Named–Named		All Entity Types	
	# Training	# Test	# Training	# Test.
At	481	106	1602	389
Near	44	14	220	70
Part	265	64	749	163
Role	732	155	2927	712
Social	55	4	611	112

7.3 Experimental Setup

As currently neither nominal nor pronominal co-reference resolution can be done with sufficient quality we restricted our experiments to *named–named* relations, where named entity recognition approaches may be used to extract the arguments. Nevertheless our kernels without any modification could also applied to the all types setting as well. Throughout our experiments we conducted classification tests on the five top level relations of the dataset. For each class in each fold we trained a separate SVM following the one vs. all scheme for multi-class classification.

7.4 Evaluation Metrics

We use the standard evaluation measures for classification accuracy: precision, recall and f-measure defined as follows:

$$Prec = \frac{TP}{TP + FP} \qquad Rec = \frac{TP}{TP + FN} \qquad F = \frac{2 \cdot Prec \cdot Rec}{Prec + Rec}$$

with the number of *true positive* (TP), *false positive* (FP) and *false negative* (FN) of the classification. Because we have a multi class classification problem we report macro (per class) and micro (per instance) averaged evaluation scores as well as results for each individual class of the different kernels [19]. For each experiment we used a 5-times repeated 5-fold cross-validation. Based on this we can compute the average for all metrics and report the standard error for the F-scores. The use of a 5-times repeated CV allows to estimate the significance of the difference in classification performance of the different kernels.

7.5 Results

Table 4 gives a quick overview of the overall classification performance of the different kernels on the benchmark dataset. In this table the classification performance of the Dependency Tree Kernel (DTK)[4], Shortest Path Kernel (SPK)[2], All-Pairs Dependency Tree Kernel (All-Pairs-DTK) and Dependency Path Tree Kernel (Path-DTK) on the 5-times repeated 5-Fold CV is shown. The All-Pairs-DTK outperforms the SPK by 5.9% F_{micro} and 23.5% F_{macro} with a significance

Table 4. Table of results with 5-times repeated 5 fold cross validation. The values in parenthesis denote the standard error over the five different CV runs.

Kernel	q	μ	λ	C	Precision	Recall	F_{micro}	F_{macro}
SPK [2]	–	–	–	1	79.8%	46.4%	58.6%(0.14)	34.2%(0.14)
DTK [4]	–	–	0.65	100	71.7%	53.7%	61.4%(0.32)	44.5%(0.63)
All-Pairs-DTK	–	–	0.6	60	73.1%	57.8%	64.5%(0.26)	**57.7%**(0.54)
Path-DTK	1	0.5	0.5	10	**80.2%**	**61.2%**	**69.4%**(0.09)	57.3%(0.40)

Table 5. Table of results on the pre-specified testset

Kernel	q	μ	λ	C	Precision	Recall	F_{micro}	F_{macro}
SPK [2]	–	–	–	1	74.7%	34.4%	47.1%	25.0%
DTK [4]	–	–	0.65	100	79.5%	44.0%	56.7%	36.9%
All-Pairs-DTK	–	–	0.6	60	**80.2%**	49.6%	61.3%	40.6%
Path-DTK	1	0.5	0.5	10	76.7%	**52.8%**	**62.5%**	**41.3%**

Table 6. Table of results for the single top-level relations with 5-times repeated 5 fold cross validation

Kernel	Relation	Precision	Recall	F-score	Std. error
SPK [4]	At	78.1%	41.8%	54.5%	(0.34)
	Near	10.0%	0.5%	0.9%	(0.78)
	Part	76.8%	25.4%	38.1%	(0.76)
	Role	81.4%	62.9%	71.0%	(0.16)
	Social	54.0%	3.6%	6.8%	(0.06)
DTK [2]	At	65.2%	47.4%	54.9%	(0.52)
	Near	52.9%	29.1%	37.3%	(2.20)
	Part	71.8%	41.8%	52.8%	(0.67)
	Role	78.4%	67.1%	72.3%	(0.30)
	Social	11.5%	3.6%	5.5%	(0.73)
All-Pairs-DTK	At	65.2%	54.0%	59.1%	(0.28)
	Near	93.2%	71.8%	**80.9%**	(1.57)
	Part	70.0%	43.4%	53.6%	(0.48)
	Role	79.1%	67.8%	73.0%	(0.44)
	Social	39.6%	15.6%	**22.2%**	(2.08)
Path-DTK	At	73.4%	58.0%	**64.8%**	(0.05)
	Near	90.9%	50.5%	64.9%	(2.19)
	Part	85.5%	49.8%	**62.9%**	(0.67)
	Role	83.4%	71.9%	**77.2%**	(0.13)
	Social	42.4%	10.6%	16.8%	(1.34)

Table 7. The results of different parameters with 5-fold CV on a single seed on the training data in the optimization phase

Kernel	q	μ	λ	C	Prec	Rec	F_{micro}	F_{macro}
SPK				1	79.1%	**46.6%**	**58.7%**	**33.9%**
				10	**79.2%**	46.5%	58.6%	**33.9%**
				60	78.8%	46.6%	58.6%	**33.9%**
				100	9.9%	46.0%	16.3%	21.4%
DTK			0.65	1	**88.3%**	35.4%	50.5%	26.9%
			0.65	10	73.7%	53.1%	61.7%	43.7%
			0.65	60	72.7%	54.8%	**62.5%**	45.4%
			0.65	100	72.7%	**54.9%**	**62.5%**	**45.5%**
			0.6	1	87.8%	34.3%	49.3%	26.3%
			0.6	10	73.0%	52.9%	61.3%	43.8%
			0.6	60	72.3%	54.9%	62.4%	45.4%
			0.6	100	72.1%	55.0%	62.4%	45.4%
			0.5	1	86.1%	31.5%	46.1%	24.1%
			0.5	10	71.5%	51.5%	59.9%	41.3%
			0.5	60	69.8%	53.7%	60.7%	43.1%
			0.5	100	69.8%	54.2%	61.0%	44.2%
All-Pairs-DTK			0.65	1	89.0%	36.0%	51.2%	30.2%
			0.65	10	78.4%	54.4%	64.2%	53.1%
			0.65	60	74.4%	57.4%	64.8%	54.8%
			0.65	100	73.8%	57.6%	64.7%	54.8%
			0.6	1	89.2%	35.1%	50.3%	28.4%
			0.6	10	79.7%	53.9%	64.3%	53.3%
			0.6	60	74.3%	**58.2%**	**65.3%**	**56.7%**
			0.6	100	73.5%	58.1%	64.9%	56.5%
			0.5	1	**89.4%**	33.5%	48.8%	25.2%
			0.5	10	81.3%	51.9%	63.4%	54.3%
			0.5	60	73.6%	57.1%	64.3%	57.2%
			0.5	100	71.8%	58.1%	64.2%	58.3%
Path-DTK	1	0.5	0.65	1	86.8%	40.3%	55.1%	29.2%
	1	0.5	0.65	10	79.8%	60.6%	68.9%	55.7%
	1	0.5	0.65	60	78.4%	61.1%	68.7%	56.4%
	1	0.5	0.65	100	78.2%	61.1%	68.6%	56.3%
	1	0.5	0.6	1	**86.9%**	39.7%	54.5%	28.7%
	1	0.5	0.6	10	80.3%	61.3%	69.5%	55.6%
	1	0.5	0.6	60	77.9%	61.8%	68.9%	56.6%
	1	0.5	0.6	100	77.9%	61.8%	68.9%	56.6%
	1	0.5	0.5	1	86.8%	38.4%	53.3%	27.1%
	1	0.5	0.5	10	80.6%	61.4%	**69.7%**	56.1%
	1	0.5	0.5	100	77.4%	61.8%	68.7%	57.4%
	1	0.5	0.5	60	77.4%	61.6%	68.6%	57.3%
	2	0.5	0.5	10	79.4%	61.1%	69.1%	52.8%
	3	0.5	0.5	10	79.7%	61.3%	69.3%	52.7%
	4	0.5	0.5	10	80.1%	61.6%	69.6%	52.8%

of 99%. The All-Pairs-DTK also beats the DTK by 3.1% F_{micro} and 13.2% F_{macro} again with a significance of 99%. It has also the best F_{macro} score of all kernels although the difference is not significant. The Path-DTK exceeds all other kernels by at least 4.9% F_{micro} measure with a significance of 99%. It also beats DTK and SPK by at least 8.0% F_{micro} measure and 12.8% F_{macro} again with a significance of 99%. This experiment clearly shows that our kernels are superior to the previously published kernels by a large margin with a high significance. In table 6 we present the results of the different kernels on each of the 5 top level relations. We see that the All-Pairs-DTK outperforms all other kernels on the "Near" relation by 16.0% F-score and on the "Social" relation at least by about 5.4% F-score. This is an interesting result because for these two relations the least training data is available. This indicates that All-Pairs-DTK might need less training data than the other kernels. The Path-DTK outperforms all other kernels on the "At", "Part" and "Role" relations with a significance of 99%. These results allow the conclusion that the Path-DTK performs best on relations where many training examples are available. However it is not clear from our experiments on the benchmark data set which of our proposed kernels is suited best for an arbitrary relation. It is clear that more experiments on different data sets are needed to make such a judgement. Table 7 shows the results for all different parameter settings. Because the benchmark data set consists of a separate test set we also conducted experiments where we trained the classifier only on the training set. We show these results in table 5. The Path-DTK performs best on the test set outperforming the All-Pairs-DTK by 1.2% F_{micro} and the other published kernels by at least 5.8% F_{micro}. This further supports the conclusion that the Path-DTK is a good general kernel for relation extraction.

8 Conclusion and Future Work

In this paper we presented two new tree kernels for relation extraction using dependency parse trees. They use richer structural features and yield significantly better results than previosly published approaches. We optimized kernel computations for dependency trees reducing the runtime considerably. For some relations the accuracy level of our methods seem to be good enough for practical application, e.g. for information retrieval. An examination on how those kernels can be combined with other kernel is meaningful [16]. Another promising direction for future work is the usage of more sophisticated features which aim at capturing the semantics of words. Here word sense disambiguation approaches [15] might be employed or topic modeling algorithms may be used to assign broader semantic information to the words relevant for a relation.

Acknowledgement

The work presented here was funded by the German Federal Ministry of Economy and Technology (BMWi) under the THESEUS project. We would like to thank the reviewers for their detailed comments.

References

1. Blohm, S., Cimiano, P.: Scaling up pattern induction for web relation extraction through frequent itemset mining. In: Proc. KI 2008 WS on Ontology-Based IE Systems (2008)
2. Bunescu, R.C., Mooney, R.J.: A shortest path dependency kernel for relation extraction. In: Proc. EMNLP 2005 (2005)
3. Bunescu, R.C., Mooney, R.J.: Subsequence kernels for relation extraction. In: Proc. Neural Information Processing Systems, NIPS 2005 (2005)
4. Culotta, A., Sorensen, J.: Dependency tree kernels for relation extraction. In: Proc. ACL 2004 (2004)
5. Harabagiu, S., Bejan, C.A., Morarescu, P.: Shallow semantics for relation extraction. In: Proc. IJCAI 2005 (2005)
6. Hearst, M.: Automatic acquisition of hyponyms from large text corpora. In: Proc. COLING 1992 (1992)
7. Horvath, T., Paass, G., Reichartz, F., Wrobel, S.: A logic-based approach to relation extraction from texts. In: ILP 2009 (2009)
8. Joachims, T.: Text categorization with support vector machines: learning with many relevant features. In: Nédellec, C., Rouveirol, C. (eds.) ECML 1998. LNCS, vol. 1398. Springer, Heidelberg (1998)
9. Joachims, T.: Making large-scale SVM learning practical. In: Advances in Kernel Methods - Support Vector Learning. MIT Press, Cambridge (1999)
10. Klein, D., Manning, C.D.: Accurate unlexicalized parsing. In: Proc. ACL 2003 (2003)
11. Klein, D., Manning, C.D.: Corpus-based induction of syntactic structure: Models of dependency and constituency. In: Proc. ACL 2004 (2004)
12. Lodhi, H., Saunders, C., Shawe-Taylor, J., Cristianini, N., Watkins, C.: Text classification using string kernels. JMLR (2), 419–444 (2002)
13. Mitchell, A.: ACE-2 Version 1.0; corpus LDC2003T11. Linguistic Data Consortium, Philadelphia (2003), `http://www.ldc.upenn.edu`
14. Moschitti, A.: Efficient convolution kernels for dependency and constituent syntactic trees. In: Fürnkranz, J., Scheffer, T., Spiliopoulou, M. (eds.) ECML 2006. LNCS (LNAI), vol. 4212, pp. 318–329. Springer, Heidelberg (2006)
15. Paaß, G., Reichartz, F.: Exploiting semantic constraints for estimating supersenses with crfs. In: Proc. SDM 2009 (2009)
16. Reichartz, F., Korte, H., Paass, G.: Composite kernels for relation extraction. In: Proc. ACL 2009 (2009)
17. Shawe-Taylor, J., Cristianini, N.: Kernel Methods for Pattern Analysis. Cambridge University Press, Cambridge (2004)
18. Tjong, E.F., Sang, K., De Meulder, F.: Language-independent named entity recognition. CoRR cs.CL/0306050 (2003)
19. Yang, Y.: An evaluation of statistical approaches to text categorization. Information Retrieval 1(1-2), 69–90 (1999)
20. Zelenko, D., Aone, C., Richardella, A.: Kernel methods for relation extraction. J. Mach. Learn. Res. 3, 1083–1106 (2003)