Convolution Kernel over Packed Parse Forest

Min Zhang Hui Zhang Haizhou Li
Institute for Infocomm Research
A-STAR, Singapore
{mzhang,vishz,hli}@i2r.a-star.edu.sg

Abstract

This paper proposes a convolution forest kernel to effectively explore rich structured features embedded in a packed parse forest. As opposed to the convolution tree kernel, the proposed forest kernel does not have to commit to a single best parse tree, is thus able to explore very large object spaces and much more structured features embedded in a forest. This makes the proposed kernel more robust against parsing errors and data sparseness issues than the convolution tree kernel. The paper presents the formal definition of convolution forest kernel and also illustrates the computing algorithm to fast compute the proposed convolution forest kernel. Experimental results on two NLP applications, relation extraction and semantic role labeling, show that the proposed forest kernel significantly outperforms the baseline of the convolution tree kernel.

1 Introduction

Parse tree and packed forest of parse trees are two widely used data structures to represent the syntactic structure information of sentences in natural language processing (NLP). The structured features embedded in a parse tree have been well explored together with different machine learning algorithms and proven very useful in many NLP applications (Collins and Duffy, 2002; Moschitti, 2004; Zhang et al., 2007). A forest (Tomita, 1987) compactly encodes an exponential number of parse trees. In this paper, we study how to effectively explore structured features embedded in a forest using convolution kernel (Haussler, 1999).

As we know, feature-based machine learning methods are less effective in modeling highly structured objects (Vapnik, 1998), such as parse tree or semantic graph in NLP. This is due to the fact that it is usually very hard to represent structured objects using vectors of reasonable dimensions without losing too much information. For example, it is computationally infeasible to enumerate all subtree features (using subtree a feature) for a parse tree into a linear feature vector. Kernel-based machine learning method is a good way to overcome this problem. Kernel methods employ a kernel function, that must satisfy the properties of being symmetric and positive, to measure the similarity between two objects by computing implicitly the dot product of certain features of the input objects in high (or even infinite) dimensional feature spaces without enumerating all the features (Vapnik, 1998).

Many learning algorithms, such as SVM (Vapnik, 1998), the Perceptron learning algorithm (Rosenblatt, 1962) and Voted Perceptron (Freund and Schapire, 1999), can work directly with kernels by replacing the dot product with a particular kernel function. This nice property of kernel methods, that implicitly calculates the dot product in a high-dimensional space over the original representations of objects, has made kernel methods an effective solution to modeling structured objects in NLP.

In the context of parse tree, convolution tree kernel (Collins and Duffy, 2002) defines a feature space consisting of all subtree types of parse trees and counts the number of common subtrees as the syntactic similarity between two parse trees. The tree kernel has shown much success in many NLP applications like parsing (Collins and Duffy, 2002), semantic role labeling (Moschitti, 2004; Zhang et al., 2007), relation extraction (Zhang et al., 2006), pronoun resolution (Yang et al., 2006), question classification (Zhang and Lee, 2003) and machine translation (Zhang and Li, 2009), where the tree kernel is used to compute the similarity between two NLP application instances that are usually represented by parse trees. However, in those studies, the tree kernel only covers the features derived from single 1-

Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics, pages 875–885,
Uppsala, Sweden, 11-16 July 2010.

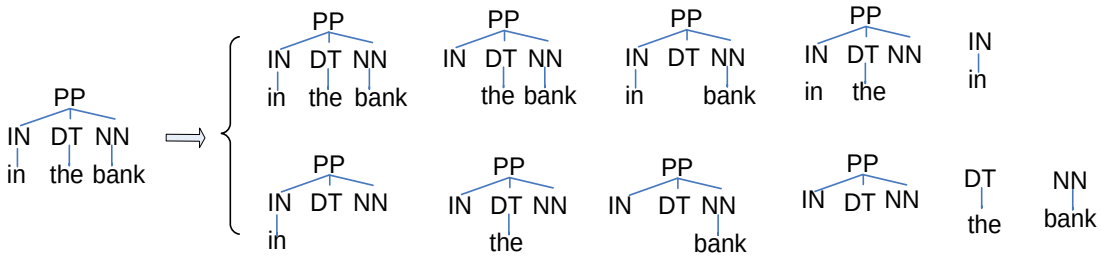

Figure 1. A parse tree and its 11 subtree features covered by convolution tree kernel

best parse tree. This may largely compromise the performance of tree kernel due to parsing errors and data sparseness.

To address the above issues, this paper constructs a forest-based convolution kernel to mine structured features directly from packed forest. A packet forest compactly encodes exponential number of n-best parse trees, and thus containing much more rich structured features than a single parse tree. This advantage enables the forest kernel not only to be more robust against parsing errors, but also to be able to learn more reliable feature values and help to solve the data sparseness issue that exists in the traditional tree kernel. We evaluate the proposed kernel in two real NLP applications, relation extraction and semantic role labeling. Experimental results on the benchmark data show that the forest kernel significantly outperforms the tree kernel.

The rest of the paper is organized as follows. Section 2 reviews the convolution tree kernel while section 3 discusses the proposed forest kernel in details. Experimental results are reported in section 4. Finally, we conclude the paper in section 5.

2 Convolution Kernel over Parse Tree

Convolution kernel was proposed as a concept of kernels for discrete structures by Haussler (1999) and related but independently conceived ideas on string kernels first presented in (Watkins, 1999). The framework defines the kernel function between input objects as the convolution of "sub-kernels", i.e. the kernels for the decompositions (parts) of the input objects.

The parse tree kernel (Collins and Duffy, 2002) is an instantiation of convolution kernel over syntactic parse trees. Given a parse tree, its features defined by a tree kernel are all of its subtree types and the value of a given feature is the number of the occurrences of the subtree in the parse tree. Fig. 1 illustrates a parse tree with all of its 11 subtree features covered by the convolution tree kernel. In the tree kernel, a parse tree T is represented by a vector of integer counts of each *subtree type* (i.e., *subtree* regardless of its ancestors, descendants and span covered):

$$\phi(T) = (\#\ subtreetype_1(T),\ \ldots,\ \#\ subtreetype_n(T))$$

where $\#\ subtreetype_i(T)$ is the occurrence number of the i^{th} subtree type in T. The tree kernel counts the number of common subtrees as the syntactic similarity between two parse trees. Since the number of subtrees is exponential with the tree size, it is computationally infeasible to directly use the feature vector $\phi(T)$. To solve this computational issue, Collins and Duffy (2002) proposed the following tree kernel to calculate the dot product between the above high dimensional vectors implicitly.

$$\begin{aligned} K(T_1,T_2) &= <\phi(T_1),\phi(T_2)> \\ &= \sum_i \#subtreetype_i(T_1)\cdot\#subtreetype_i(T_2) \\ &= \sum_i \left(\sum_{n_1\in N_1} I_{subtree_i}(n_1) \right)\cdot\left(\sum_{n_2\in N_2} I_{subtree_i}(n_2) \right) \\ &= \sum_{n_1\in N_1}\sum_{n_2\in N_2} \Delta(n_1,n_2) \end{aligned}$$

where N_1 and N_2 are the sets of nodes in trees T_1 and T_2, respectively, and $I_{subtree_i}(n)$ is a function that is 1 iff the $subtreetype_i$ occurs with root at node n and zero otherwise, and $\Delta(n_1,n_2)$ is the number of the common *subtrees* rooted at n_1 and n_2, i.e.,

$$\Delta(n_1,n_2) = \sum_i I_{subtree_i}(n_1)\cdot I_{subtree_i}(n_2)$$

$\Delta(n_1,n_2)$ can be computed by the following recursive rules:

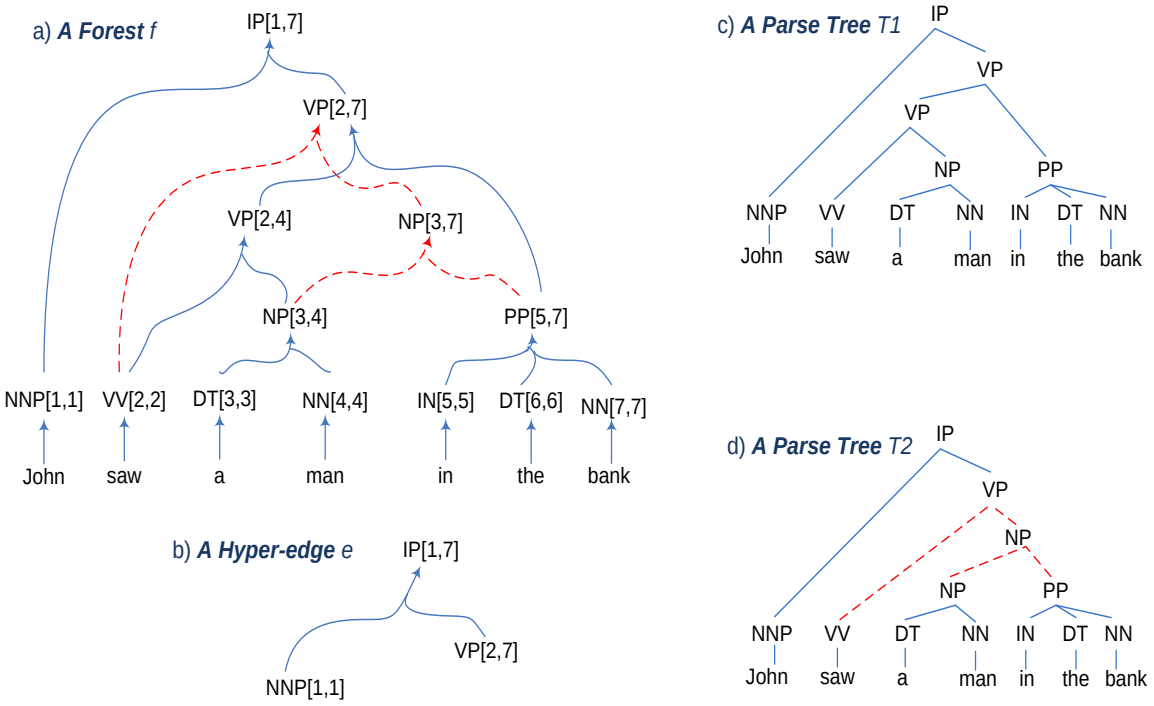

Figure 2. An example of a packed forest, a hyper-edge and two parse trees covered by the packed forest

Rule 1: if the productions (CFG rules) at n_1 and n_2 are different, $\Delta(n_1, n_2) = 0$;

Rule 2: else if both n_1 and n_2 are pre-terminals (POS tags), $\Delta(n_1, n_2) = 1 \times \lambda$;

Rule 3: else,

$$\Delta(n_1, n_2) = \lambda \cdot \prod_{j=1}^{nc(n_1)} (1 + \Delta(ch(n_1, j), ch(n_2, j)))\,,$$

where $nc(n_1)$ is the child number of n_1, $ch(n,j)$ is the j^{th} child of node n and λ ($0 < \lambda \leq 1$) is the decay factor in order to make the kernel value less variable with respect to the *subtree* sizes (Collins and Duffy, 2002). The recursive **Rule 3** holds because given two nodes with the same children, one can construct common subtrees using these children and common subtrees of further offspring. The time complexity for computing this kernel is $O(|N_1| \cdot |N_2|)$.

As discussed in previous section, when convolution tree kernel is applied to NLP applications, its performance is vulnerable to the errors from the single parse tree and data sparseness. In this paper, we present a convolution kernel over packed forest to address the above issues by exploring structured features embedded in a forest.

3 Convolution Kernel over Forest

In this section, we first illustrate the concept of packed forest and then give a detailed discussion on the covered feature space, fractional count, feature value and the forest kernel function itself.

3.1 Packed forest of parse trees

Informally, a packed parse forest, or (packed) forest in short, is a compact representation of all the derivations (i.e. parse trees) for a given sentence under context-free grammar (Tomita, 1987; Billot and Lang, 1989; Klein and Manning, 2001). It is the core data structure used in natural language parsing and other downstream NLP applications, such as syntax-based machine translation (Zhang et al., 2008; Zhang et al., 2009a). In parsing, a sentence corresponds to exponential number of parse trees with different tree probabilities, where a forest can compact all the parse trees by sharing their common subtrees in a bottom-up manner. Formally, a packed forest F can be described as a triple:

$$F = < V, E, S >$$

where V is the set of non-terminal nodes, E is the set of hyper-edges and S is a sentence

represented as an ordered word sequence. A hyper-edge e is a group of edges in a parse tree which connects a father node and its all child nodes, representing a CFG rule. A non-terminal node in a forest is represented as a "label [start, end]", where the "label" is its syntax category and "[start, end]" is the span of words it covers. As shown in Fig. 2, these two parse trees ($T1$ and $T2$) can be represented as a single forest by sharing their common subtrees (such as NP[3,4] and PP[5,7]) and merging common non-terminal nodes covering the same span (such as VP[2,7], where there are two hyper-edges attach to it).

Given the definition of forest, we introduce the concepts of inside probability $\beta(.)$ and outside probability $\alpha(.)$ that are widely-used in parsing (Baker, 1979; Lari and Young, 1990) and are also to be used in our kernel calculation.

$$\beta(v[p,p]) = P(v \to S[p])$$

$$\beta(v[p,q]) = \sum_{\substack{e \text{ is a hyper-edge} \\ \text{attached to } v}} \left(P(e) \cdot \prod_{\substack{c_i[p_i,q_i] \text{is a leaf} \\ \text{node of } e}} \beta(c_i[p_i,q_i]) \right)$$

$$\alpha(root(f)) = 1$$

$$\alpha(v[p,q]) = \sum_{\substack{e \text{ is a hyper-edge and } v \\ \text{is its one leaf node}}} \left(\alpha(root(e)) \cdot P(e) \cdot \prod_{\substack{c_i[p_i,q_i] \text{is a children node} \\ \text{of e except } v}} \beta(c_i[p_i,q_i]) \right)$$

where v is a forest node, $S[p]$ is the p^{th} word of input sentence S, $P(v \to S[p])$ is the probability of the CFG rule $v \to S[p]$, $root(.)$ returns the root node of input structure, $[p_i, q_i]$ is a sub-span of $[p, q]$, being covered by c_i, and $P(e)$ is the PCFG probability of e. From these definitions, we can see that the inside probability is total probability of generating words $S[p,q]$ from non-terminal node $v[p,q]$ while the outside probability is the total probability of generating node $v[p,q]$ and words outside $S[p,q]$ from the root of forest. The inside probability can be calculated using dynamic programming in a bottom-up fashion while the outside probability can be calculated using dynamic programming in a top-to-down way.

3.2 Convolution forest kernel

In this subsection, we first define the feature space covered by forest kernel, and then define the forest kernel function.

3.2.1 Feature space, object space and feature value

The forest kernel counts the number of common subtrees as the syntactic similarity between two forests. Therefore, in the same way as tree kernel, its feature space is also defined as all the possible subtree types that a CFG grammar allows. In a forest kernel, forest *F* is represented by a vector of *fractional counts* of each *subtree type* (subtree regardless of its ancestors, descendants and span covered):

$$\begin{aligned} \phi(F) &= (\#\ subtreetype_1(F), \ldots, \\ &\quad \#\ subtreetype_n(F)) \\ &= (\#subtreetype_1(n\text{-}best\ parse\ trees), \ldots, \\ &\quad \#\ subtreetype_n(n\text{-}best\ parse\ trees)) \end{aligned} \quad (1)$$

where # $subtreetype_i(F)$ is the occurrence number of the i^{th} subtree type ($subtreetype_i$) in forest *F*, i.e., a *n*-best parse tree lists with a huge *n*.

Although the feature spaces of the two kernels are the same, their object spaces (tree *vs.* forest) and feature values (integer counts *vs.* fractional counts) differ very much. A forest encodes exponential number of parse trees, and thus containing exponential times more subtrees than a single parse tree. This ensures forest kernel to learn more reliable feature values and is also able to help to address the data sparseness issues in a better way than tree kernel does. Forest kernel is also expected to yield more non-zero feature values than tree kernel. Furthermore, different parse tree in a forest represents different derivation and interpretation for a given sentence. Therefore, forest kernel should be more robust to parsing errors than tree kernel.

In tree kernel, one occurrence of a subtree contributes 1 to the value of its corresponding feature (*subtree type*), so the feature value is an integer count. However, the case turns out very complicated in forest kernel. In a forest, each of its parse trees, when enumerated, has its own

probability. So one subtree extracted from different parse trees should have different *fractional count* with regard to the probabilities of different parse trees. Following the previous work (Charniak and Johnson, 2005; Huang, 2008), we define the *fractional count* of the occurrence of a *subtree* in a parse tree t_i as

$$c(subtree, t_i) = \begin{cases} 0 & if\ subtree \notin t_i \\ P(subtree, t_i|f,s) & otherwise \end{cases}$$
$$= \begin{cases} 0 & if\ subtree \notin t_i \\ P(t_i|f,s) & otherwise \end{cases}$$

where we have $P(subtree, t_i|f,s) = P(t_i|f,s)$ if $subtree \in t_i$. Then we define the fractional count of the occurrence of a subtree in a forest *f* as

$$\begin{aligned} c(subtree, f) &= P(subtree|f,s) \\ &= \textstyle\sum_{t_i} P(subtree, t_i|f,s) \\ &= \textstyle\sum_{t_i} I_{subtree}\ (t_i) \cdot P(t_i|f,s) \end{aligned} \quad (2)$$

where $I_{subtree}\ (t_i)$ is a binary function that is 1 iif the $subtree \in t_i$ and zero otherwise. Obviously, it needs exponential time to compute the above fractional counts. However, due to the property of forest that compactly represents all the parse trees, the posterior probability of a subtree in a forest, $P(subtree|f,s)$, can be easily computed in an Inside-Outside fashion as the product of three parts: the outside probability of its root node, the probabilities of parse hyperedges involved in the subtree, and the inside probabilities of its leaf nodes (Lari and Young, 1990; Mi and Huang, 2008).

$$\begin{aligned} c(subtree, f) &= P(subtree|f,s) \\ &= \frac{\alpha\beta(subtree)}{\alpha\beta(root(f))} \end{aligned} \quad (3)$$

where

$$\begin{aligned} \alpha\beta(subtree) &= \alpha\big(root(subtree)\big) \\ &\cdot \prod_{e \in subtree} P(e) \\ &\cdot \prod_{v \in leaf\ (subtree\ )} \beta(v) \end{aligned} \quad (4)$$

and

$$\begin{aligned} \alpha\beta\big(root(f)\big) &= \alpha\big(root(f)\big) \cdot \beta\big(root(f)\big) \\ &= \beta\big(root(f)\big) \end{aligned}$$

where $\alpha(.)$ and $\beta(.)$ denote the outside and inside probabilities. They can be easily obtained using the equations introduced at section 3.1.

Given a subtree, we can easily compute its fractional count (i.e. its feature value) directly using eq. (3) and (4) without the need of enumerating each parse trees as shown at eq. (2)[1]. Nonetheless, it is still computationally infeasible to directly use the feature vector $\phi(F)$ (see eq. (1)) by explicitly enumerating all subtrees although its fractional count is easily calculated. In the next subsection, we present the forest kernel that implicitly calculates the dot-product between two $\phi(F)$s in a polynomial time.

3.2.2 Convolution forest kernel

The forest kernel counts the *fractional numbers* of common subtrees as the syntactic similarity between two forests. We define the forest kernel function $K_f(f_1, f_2)$ in the following way.

$$\begin{aligned} K_f(f_1, f_2) &= <\phi(f_1), \phi(f_2)> \\ &= \sum_i \#subtreetype_i(f_1).\#subtreetype_i(f_2) \\ &= \sum_{\substack{subtree\ 1 \in f_1 \\ subtree\ 2 \in f_2}} \Big(I_{eq}(subtree1, subtree2) \\ &\qquad \cdot c(subtree1, f_1) \\ &\qquad \cdot c(subtree2, f_2) \Big) \\ &= \textstyle\sum_{v_1 \in N_1} \sum_{v_2 \in N_2} \Delta'(v_1, v_2) \end{aligned} \quad (5)$$

where

- $I_{eq}(\cdot,\cdot)$ is a binary function that is 1 iif the input two subtrees are identical (i.e. they have the same typology and node labels) and zero otherwise;
- $c(\cdot,\cdot)$ is the fractional count defined at eq. (3);
- N_1 and N_2 are the sets of nodes in forests f_1 and f_2;
- $\Delta'(v_1, v_2)$ returns the accumulated value of products between each two fractional counts of the common subtrees rooted at v_1 and v_2, i.e.,

$$\begin{aligned} &\Delta'(v_1, v_2) \\ &= \sum_{\substack{root\ (subtree\ 1)=v_1 \\ root\ (subtree\ 2)=v_2}} \Big(I_{eq}(subtree1, subtree2) \\ &\qquad \cdot c(subtree1, f_1) \\ &\qquad \cdot c(subtree2, f_2) \Big) \end{aligned}$$

[1] It has been proven in parsing literatures (Baker, 1979; Lari and Young, 1990) that eq. (3) defined by Inside-Outside probabilities is exactly to compute the sum of those parse tree probabilities that cover the subtree of being considered as defined at eq. (2).

We next show that $\Delta'(v_1, v_2)$ can be computed recursively in a polynomial time as illustrated at Algorithm 1. To facilitate discussion, we temporarily ignore all fractional counts in Algorithm 1. Indeed, Algorithm 1 can be viewed as a natural extension of convolution kernel from over tree to over forest. In forest[2], a node can root multiple hyper-edges and each hyper-edge is independent to each other. Therefore, Algorithm 1 iterates each hyper-edge pairs with roots at v_1 and v_2 (line 3-4), and sums over (eq. (7) at line 9) each recursively-accumulated sub-kernel scores of subtree pairs extended from the hyper-edge pair (e_1, e_2) (eq. (6) at line 8). Eq. (7) holds because the hyper-edges attached to the same node are independent to each other. Eq. (6) is very similar to the **Rule 3** of tree kernel (see section 2) except its inputs are hyper-edges and its further expansion is based on forest nodes. Similar to tree kernel (Collins and Duffy, 2002), eq. (6) holds because a common subtree by extending from (e_1, e_2) can be formed by taking the hyper-edge (e_1, e_2), together with a choice at each of their leaf nodes of simply taking the non-terminal at the leaf node, or any one of the common subtrees with root at the leaf node. Thus there are $\left(1 + \Delta'\left(leaf(e_1, j), leaf(e_2, j)\right)\right)$ possible choices at the j^{th} leaf node. In total, there are $\Delta''(e_1, e_2)$ (eq. (6)) common subtrees by extending from (e_1, e_2) and $\Delta'(v_1, v_2)$ (eq. (7)) common subtrees with root at (v_1, v_2).

Obviously $\Delta'(v_1, v_2)$ calculated by Algorithm 1 is a proper convolution kernel since it simply counts the number of common subtrees under the root (v_1, v_2). Therefore, $K_f(f_1, f_2)$ defined at eq. (5) and calculated through $\Delta'(v_1, v_2)$ is also a proper convolution kernel. From eq. (5) and Algorithm 1, we can see that each hyper-edge pair (e_1, e_2) is only visited at most one time in computing the forest kernel. Thus the time complexity for computing $K_f(f_1, f_2)$ is $O(|E_1| \cdot |E_2|)$, where E_1 and E_2 are the set of hyper-edges in forests f_1 and f_2, respectively. Given a forest and the best parse trees, the number of hyper-edges is only several times (normally <=3 after pruning) than that of tree nodes in the parse tree[3].

[2] Tree can be viewed as a special case of forest with only one hyper-edge attached to each tree node.

[3] Suppose there are *K* forest nodes in a forest, each node has *M* associated hyper-edges fan out and each hyper-edge has *N* children. Then the forest is capable of encoding $M^{\frac{K-1}{N-1}}$ parse trees at most (Zhang et al., 2009b).

Algorithm 1.

Input:

f_1, f_2: *two packed forests*

v_1, v_2: *any two nodes of* f_1 *and* f_2

Notation:

$I_{eq}(\cdot,\cdot)$: *defined at* eq. (5)

$nl(e_1)$: *number of leaf node of* e_1

$leaf(e_1, j)$: *the* j^{th} *leaf node of* e_1

Output: $\Delta'(v_1, v_2)$

1. $\Delta'(v_1, v_2) = 0$
2. **if** $v_1.label \neq v_2.label$ **exit**
3. **for** each hyper-edge e_1 attached to v_1 **do**
4. **for** each hyper-edge e_2 attached to v_2 **do**
5. **if** $I_{eq}(e_1, e_2) == 0$ **do**
6. **goto** line 3
7. **else do**
8. $\Delta''(e_1, e_2) = \prod_{j=1}^{nl(e_1)}\left(1 + \Delta'\left(leaf(e_1, j), leaf(e_2, j)\right)\right)$ (6)
9. $\Delta'(v_1, v_2) \mathrel{+}= \Delta''(e_1, e_2)$ (7)
10. **end if**
11. **end for**
12. **end for**

Same as tree kernel, forest kernel is running more efficiently in practice since only two nodes with the same label needs to be further processed (line 2 of Algorithm 1).

Now let us see how to integrate fractional counts into forest kernel. According to Algorithm 1 (eq. (7)), we have (e_1/e_2 are attached to v_1/v_2, respectively)

$$\Delta'(v_1, v_2) = \sum_{e_1=e_2} \Delta''(e_1, e_2)$$

Recall eq. (4), a fractional count consists of outside, inside and subtree probabilities. It is more straightforward to incorporate the outside and subtree probabilities since all the subtrees with roots at (v_1, v_2) share the same outside probability and each hyper-edge pair is only visited one time. Thus we can integrate the two probabilities into $\Delta'(v_1, v_2)$ as follows.

$$\Delta'(v_1, v_2) = \lambda \cdot \alpha(v_1) \cdot \alpha(v_2) \cdot \sum_{e_1=e_2}\left(P(e_1) \cdot P(e_2) \cdot \Delta''(e_1, e_2)\right) \quad (8)$$

where, following tree kernel, a decay factor $\lambda(0 < \lambda \leq 1)$ is also introduced in order to make the kernel value less variable with respect to the subtree sizes (Collins and Duffy, 2002). It functions like multiplying each feature value by λ^{size_i}, where $size_i$ is the number of hyper-edges in $subtree_i$.

The inside probability is only involved when a node does not need to be further expanded. The integer 1 at eq. (6) represents such case. So the inside probability is integrated into eq. (6) by replacing the integer 1 as follows.

$$\Delta''(e_1, e_2) = \prod_{j=1}^{nl(e_1)} \left(\beta\big(leaf(e_1, j)\big) \cdot \beta\big(leaf(e_2, j)\big) + \frac{\Delta'\big(leaf(e_1, j), leaf(e_2, j)\big)}{\alpha\big(leaf(e_1, j)\big) \cdot \alpha\big(leaf(e_2, j)\big)} \right) \quad (9)$$

where in the last expression the two outside probabilities $\alpha\big(leaf(e_1, j)\big)$ and $\alpha\big(leaf(e_2, j)\big)$ are removed. This is because $leaf(e_1, j)$ and $leaf(e_2, j)$ are not roots of the subtrees of being explored (only outside probabilities of the root of a subtree should be counted in its *fractional count*), and $\Delta'\big(leaf(e_1, j), leaf(e_2, j)\big)$ already contains the two outside probabilities of $leaf(e_1, j)$ and $leaf(e_2, j)$.

Referring to eq. (3), each fractional count needs to be normalized by $\alpha\beta(root(f))$. Since $\alpha\beta(root(f))$ is independent to each individual fractional count, we do the normalization outside the recursive function $\Delta''(e_1, e_2)$. Then we can re-formulize eq. (5) as

$$K_f(f_1, f_2) = < \phi(f_1), \phi(f_2) > = \frac{\left(\sum_{v_1 \in N_1} \sum_{v_2 \in N_2} \Delta'(v_1, v_2)\right)}{\alpha\beta\big(root(f_1)\big) \cdot \alpha\beta\big(root(f_2)\big)} \quad (10)$$

Finally, since the size of input forests is not constant, the forest kernel value is normalized using the following equation.

$$\widehat{K}_f(f_1, f_2) = \frac{K_f(f_1, f_2)}{\sqrt{K_f(f_1, f_1) \cdot K_f(f_2, f_2)}} \quad (11)$$

From the above discussion, we can see that the proposed forest kernel is defined together by eqs. (11), (10), (9) and (8). Thanks to the compact representation of trees in forest and the recursive nature of the kernel function, the introduction of fractional counts and normalization do not change the convolution property and the time complexity of the forest kernel. Therefore, the forest kernel $\widehat{K}_f(f_1, f_2)$ is still a proper convolution kernel with quadratic time complexity.

3.3 Comparison with previous work

To the best of our knowledge, this is the first work to address convolution kernel over packed parse forest.

Convolution tree kernel is a special case of the proposed forest kernel. From feature exploration viewpoint, although theoretically they explore the same subtree feature spaces (defined recursively by CFG parsing rules), their feature values are different. Forest encodes exponential number of trees. So the number of subtree instances extracted from a forest is exponential number of times greater than that from its corresponding parse tree. The significant difference of the amount of subtree instances makes the parameters learned from forests more reliable and also can help to address the data sparseness issue. To some degree, forest kernel can be viewed as a tree kernel with very powerful *back-off* mechanism. In addition, forest kernel is much more robust against parsing errors than tree kernel.

Aiolli *et al.* (2006; 2007) propose using Direct Acyclic Graphs (DAG) as a compact representation of tree kernel-based models. This can largely reduce the computational burden and storage requirements by sharing the common structures and feature vectors in the kernel-based model. There are a few other previous works done by generalizing convolution tree kernels (Kashima and Koyanagi, 2003; Moschitti, 2006; Zhang et al., 2007). However, all of these works limit themselves to single tree structure from modeling viewpoint in nature.

From a broad viewpoint, as suggested by one reviewer of the paper, we can consider the forest kernel as an alternative solution proposed for the general problem of noisy inference pipelines (eg. speech translation by composition of FSTs, machine translation by translating over 'lattices' of segmentations (Dyer *et al*., 2008) or using parse tree info for downstream applications in our cases) . Following this line, Bunescu (2008) and Finkel et al. (2006) are two typical related works done in reducing cascading noisy. However, our works are not overlapped with each other as there are two totally different solutions for the same general problem. In addition, the main motivation of this paper is also different from theirs.

4 Experiments

Forest kernel has a broad application potential in NLP. In this section, we verify the effectiveness of the forest kernel on two NLP applications, semantic role labeling (SRL) (Gildea, 2002) and relation extraction (RE) (ACE, 2002-2006).

In our experiments, SVM (Vapnik, 1998) is selected as our classifier and the *one vs. others* strategy is adopted to select the one with the

largest margin as the final answer. In our implementation, we use the binary SVMLight (Joachims, 1998) and borrow the framework of the Tree Kernel Tools (Moschitti, 2004) to integrate our forest kernel into the SVMLight. We modify Charniak parser (Charniak, 2001) to output a packed forest. Following previous forest-based studies (Charniak and Johnson, 2005), we use the marginal probabilities of hyper-edges (i.e., the Viterbi-style inside-outside probabilities and set the pruning threshold as 8) for forest pruning.

4.1 Semantic role labeling

Given a sentence and each predicate (either a target verb or a noun), SRL recognizes and maps all the constituents in the sentence into their corresponding semantic arguments (roles, e.g., A0 for *Agent*, A1 for *Patient* …) of the predicate or *non*-argument. We use the CoNLL-2005 shared task on Semantic Role Labeling (Carreras and Màrquez, 2005) for the evaluation of our forest kernel method. To speed up the evaluation process, the same as Che *et al.* (2008), we use a subset of the entire training corpus (WSJ sections 02-05 of the entire sections 02-21) for training, section 24 for development and section 23 for test, where there are 35 roles including 7 Core (A0–A5, AA), 14 Adjunct (AM-) and 14 Reference (R-) arguments.

The state-of-the-art SRL methods (Carreras and Màrquez, 2005) use constituents as the labeling units to form the labeled arguments. Due to the errors from automatic parsing, it is impossible for all arguments to find their matching constituents in the single 1-best parse trees. Statistics on the training data shows that 9.78% of arguments have no matching constituents using the Charniak parser (Charniak, 2001), and the number increases to 11.76% when using the Collins parser (Collins, 1999). In our method, we break the limitation of 1-best parse tree and regard each span rooted by a single forest node (i.e., a *sub-forest* with one or more roots) as a candidate argument. This largely reduces the unmatched arguments from 9.78% to 1.31% after forest pruning. However, it also results in a very large amount of argument candidates that is 5.6 times as many as that from 1-best tree. Fortunately, after the pre-processing stage of argument pruning (Xue and Palmer, 2004)[4], although the amount of unmatched argument increases a little bit to 3.1%, its generated total candidate amount decreases substantially to only 1.31 times of that from 1-best parse tree. This clearly shows the advantages of the forest-based method over tree-based in SRL.

The best-reported tree kernel method for SRL $K_{hybrid} = \theta \cdot K_{path} + (1-\theta) \cdot K_{cs}$ $(0 \le \theta \le 1)$, proposed by Che et al. (2006)[5], is adopted as our baseline kernel. We implemented the K_{hybrid} in tree case ($K_{T-hybrid}$, using tree kernel to compute K_{path} and K_{cs}) and in forest case ($K_{F-hybrid}$, using tree kernel to compute K_{path} and K_{cs}).

	Precision	**Recall**	**F-Score**
$K_{T-hybrid}$ (Tree)	76.02	67.38	71.44
$K_{F-hybrid}$ (Forest)	79.06	69.12	73.76

Table 1: Performance comparison of SRL (%)

Table 1 shows that the forest kernel significantly outperforms (χ^2 test with p=0.01) the tree kernel with an absolute improvement of 2.32 (73.76-71.42) percentage in F-Score, representing a relative error rate reduction of 8.19% (2.32/(100-71.64)). This convincingly demonstrates the advantage of the forest kernel over the tree kernel. It suggests that the structured features represented by subtree are very useful to SRL. The performance improvement is mainly due to the fact that forest encodes much more such structured features and the forest kernel is able to more effectively capture such structured features than the tree kernel. Besides F-Score, both precision and recall also show significantly improvement (χ^2 test with p=0.01). The reason for recall improvement is mainly due to the lower rate of unmatched argument (3.1% only) with only a little bit overhead (1.31 times) (see the previous discussion in this section). The precision improvement is mainly attributed to fact that we use *sub-forest* to represent argument instances, rather than sub-tree used in tree kernel, where the sub-tree is only one tree encoded in the *sub-forest*.

[4] We extend (Xue and Palmer, 2004)'s argument pruning algorithm from tree-based to forest-based. The algorithm is very effective. It can prune out around 90% argument candidates in parse tree-based SRL and thus makes the amounts of positive and negative training instances (arguments) more balanced. We apply the same pruning strategies to forest plus our heuristic rules to prune out some of the arguments with span overlapped with each other and those arguments with very small inside probabilities, depending on the numbers of candidates in the span.

[5] K_{path} and K_{cs} are two standard convolution tree kernels to describe predicate-argument path substructures and argument syntactic substructures, respectively.

4.2 Relation extraction

As a subtask of information extraction, relation extraction is to extract various semantic relations between entity pairs from text. For example, the sentence "Bill Gates is chairman and chief software architect of Microsoft Corporation" conveys the semantic relation "EMPLOYMENT.executive" between the entities "Bill Gates" (person) and "Microsoft Corporation" (company). We adopt the method reported in Zhang et al. (2006) as our baseline method as it reports the state-of-the-art performance using tree kernel-based composite kernel method for RE. We replace their tree kernels with our forest kernels and use the same experimental settings as theirs. We carry out the same five-fold cross validation experiment on the same subset of ACE 2004 data (LDC2005T09, ACE 2002-2004) as that in Zhang et al. (2006). The data contain 348 documents and 4400 relation instances.

In SRL, constituents are used as the labeling units to form the labeled arguments. However, previous work (Zhang et al., 2006) shows that if we use complete constituent (MCT) as done in SRL to represent relation instance, there is a large performance drop compared with using the path-enclosed tree (PT)[6]. By simulating PT, we use the minimal fragment of a forest covering the two entities and their internal words to represent a relation instance by only parsing the span covering the two entities and their internal words.

	Precision	Recall	F-Score
Zhang et al. (2006):Tree	68.6	59.3	63.6
Ours: Forest	70.3	60.0	64.7

Table 2: Performance Comparison of RE (%) over 23 subtypes on the ACE 2004 data

Table 2 compares the performance of the forest kernel and the tree kernel on relation extraction. We can see that the forest kernel significantly outperforms (χ^2 test with p=0.05) the tree kernel by 1.1 point of F-score. This further verifies the effectiveness of the forest kernel method for modeling NLP structured data. In summary, we further observe the high precision improvement that is consistent with the SRL experiments. However, the recall improvement is not as significant as observed in SRL. This is because unlike SRL, RE has no un-matching issues in generating relation instances. Moreover, we find that the performance improvement in RE is not as good as that in SRL. Although we know that performance is task-dependent, one of the possible reasons is that SRL tends to be long-distance grammatical structure-related while RE is local and semantic-related as observed from the two experimental benchmark data.

[6] MCT is the minimal constituent rooted by the nearest common ancestor of the two entities under consideration while PT is the minimal portion of the parse tree (may not be a complete subtree) containing the two entities and their internal lexical words. Since in many cases, the two entities and their internal words cannot form a grammatical constituent, MCT may introduce too many noisy context features and thus lead to the performance drop.

5 Conclusions and Future Work

Many NLP applications have benefited from the success of convolution kernel over parse tree. Since a packed parse forest contains much richer structured features than a parse tree, we are motivated to develop a technology to measure the syntactic similarity between two forests.

To achieve this goal, in this paper, we design a convolution kernel over packed forest by generalizing the tree kernel. We analyze the object space of the forest kernel, the fractional count for feature value computing and design a dynamic programming algorithm to realize the forest kernel with quadratic time complexity. Compared with the tree kernel, the forest kernel is more robust against parsing errors and data sparseness issues. Among the broad potential NLP applications, the problems in SRL and RE provide two pointed scenarios to verify our forest kernel. Experimental results demonstrate the effectiveness of the proposed kernel in structured NLP data modeling and the advantages over tree kernel.

In the future, we would like to verify the forest kernel in more NLP applications. In addition, as suggested by one reviewer, we may consider rescaling the probabilities (exponentiating them by a constant value) that are used to compute the fractional counts. We can sharpen or flatten the distributions. This basically says "how seriously do we want to take the very best derivation" compared to the rest. However, the challenge is that we compute the fractional counts together with the forest kernel recursively by using the Inside-Outside probabilities. We cannot differentiate the individual parse tree's contribution to a fractional count on the fly. One possible solution is to do the probability rescaling off-line before kernel calculation. This would be a very interesting research topic of our future work.

References

ACE (2002-2006). *The Automatic Content Extraction Projects*. http://www.ldc.upenn.edu/Projects/ACE/

Fabio Aiolli, Giovanni Da San Martino, Alessandro Sperduti and Alessandro Moschitti. 2006. *Fast Online Kernel Learning for Trees*. ICDM-2006

Fabio Aiolli, Giovanni Da San Martino, Alessandro Sperduti and Alessandro Moschitti. 2007. *Efficient Kernel-based Learning for Trees*. IEEE Symposium on Computational Intelligence and Data Mining (CIDM-2007)

J. Baker. 1979. *Trainable grammars for speech recognition*. The 97th meeting of the Acoustical Society of America

S. Billot and S. Lang. 1989. *The structure of shared forest in ambiguous parsing*. ACL-1989

Razvan Bunescu. 2008. *Learning with Probabilistic Features for Improved Pipeline Models*. EMNLP-2008

X. Carreras and Lluıs Màrquez. 2005. *Introduction to the CoNLL-2005 shared task: SRL*. CoNLL-2005

E. Charniak. 2001. *Immediate-head Parsing for Language Models*. ACL-2001

E. Charniak and Mark Johnson. 2005. *Corse-to-fine-grained n-best parsing and discriminative reranking*. ACL-2005

Wanxiang Che, Min Zhang, Ting Liu and Sheng Li. 2006. *A hybrid convolution tree kernel for semantic role labeling*. COLING-ACL-2006 (poster)

WanXiang Che, Min Zhang, Aiti Aw, Chew Lim Tan, Ting Liu and Sheng Li. 2008. *Using a Hybrid Convolution Tree Kernel for Semantic Role Labeling*. ACM Transaction on Asian Language Information Processing

M. Collins. 1999. *Head-driven statistical models for natural language parsing*. Ph.D. dissertation, Pennsylvania University

M. Collins and N. Duffy. 2002. *Convolution Kernels for Natural Language*. NIPS-2002

Christopher Dyer, Smaranda Muresan and Philip Resnik. 2008. *Generalizing Word Lattice Translation.* ACL-HLT-2008

Jenny Rose Finkel, Christopher D. Manning and Andrew Y. Ng. 2006. *Solving the Problem of Cascading Errors: Approximate Bayesian Inference for Linguistic Annotation Pipelines.* EMNLP-2006

Y. Freund and R. E. Schapire. 1999. *Large margin classification using the perceptron algorithm*. Machine Learning, 37(3):277-296

D. Guldea. 2002. *Probabilistic models of verb-argument structure*. COLING-2002

D. Haussler. 1999. *Convolution Kernels on Discrete Structures.* Technical Report UCS-CRL-99-10, University of California, Santa Cruz

Liang Huang. 2008. *Forest reranking: Discriminative parsing with non-local features*. ACL-2008

Karim Lari and Steve J. Young. 1990. *The estimation of stochastic context-free grammars using the inside-outside algorithm*. Computer Speech and Language. 4(35–56)

H. Kashima and T. Koyanagi. 2003. *Kernels for Semi-Structured Data*. ICML-2003

Dan Klein and Christopher D. Manning. 2001. *Parsing and Hypergraphs*. IWPT-2001

T. Joachims. 1998. *Text Categorization with Support Vecor Machine: learning with many relevant features*. ECML-1998

Haitao Mi and Liang Huang. 2008. *Forest-based Translation Rule Extraction*. EMNLP-2008

Alessandro Moschitti. 2004. *A Study on Convolution Kernels for Shallow Semantic Parsing*. ACL-2004

Alessandro Moschitti. 2006. *Syntactic kernels for natural language learning: the semantic role labeling case*. HLT-NAACL-2006 (short paper)

Martha Palmer, Dan Gildea and Paul Kingsbury. 2005. *The proposition bank: An annotated corpus of semantic roles*. Computational Linguistics. 31(1)

F. Rosenblatt. 1962. *Principles of Neurodynamics: Perceptrons and the theory of brain mechanisms*. Spartan Books, Washington D.C.

Masaru Tomita. 1987. *An Efficient Augmented-Context-Free Parsing Algorithm*. Computational Linguistics 13(1-2): 31-46

Vladimir N. Vapnik. 1998. *Statistical Learning Theory*. Wiley

C. Watkins. 1999. *Dynamic alignment kernels*. In A. J. Smola, B. Sch ̈olkopf, P. Bartlett, and D. Schuurmans (Eds.), Advances in kernel methods. MIT Press

Nianwen Xue and Martha Palmer. 2004. *Calibrating features for semantic role labeling.* EMNLP-2004

Xiaofeng Yang, Jian Su and Chew Lim Tan. 2006. *Kernel-Based Pronoun Resolution with Structured Syntactic Knowledge*. COLING-ACL-2006

Dell Zhang and W. Lee. 2003. *Question classification using support vector machines*. SIGIR-2003

Hui Zhang, Min Zhang, Haizhou Li, Aiti Aw and Chew Lim Tan. 2009a. *Forest-based Tree Sequence to String Translation Model*. ACL-IJCNLP-2009

Hui Zhang, Min Zhang, Haizhou Li and Chew Lim Tan. 2009b. *Fast Translation Rule Matching for*

Syntax-based Statistical Machine Translation. EMNLP-2009

Min Zhang, Jie Zhang, Jian Su and GuoDong Zhou. 2006. *A Composite Kernel to Extract Relations between Entities with Both Flat and Structured Features*. COLING-ACL-2006

Min Zhang, W. Che, A. Aw, C. Tan, G. Zhou, T. Liu and S. Li. 2007. *A Grammar-driven Convolution Tree Kernel for Semantic Role Classification*. ACL-2007

Min Zhang, Hongfei Jiang, Aiti Aw, Haizhou Li, Chew Lim Tan and Sheng Li. 2008. *A Tree Sequence Alignment-based Tree-to-Tree Translation Model*. ACL-2008

Min Zhang and Haizhou Li. 2009. *Tree Kernel-based SVM with Structured Syntactic Knowledge for BTG-based Phrase Reordering*. EMNLP-2009