

Tree Sequence Kernel for Natural Language

Jun Sun[†]

Min Zhang[‡]

Chew Lim Tan[†]

[†]School of Computing, National University of Singapore [‡]Institute for Infocomm Research
sunjun@comp.nus.edu.sg mzhang@i2r.a-star.edu.sg tancl@comp.nus.edu.sg

Abstract

We propose Tree Sequence Kernel (TSK), which implicitly exhausts the structure features of a sequence of subtrees embedded in the phrasal parse tree. By incorporating the capability of sequence kernel, TSK enriches tree kernel with tree sequence features so that it may provide additional useful patterns for machine learning applications. Two approaches of penalizing the substructures are proposed and both can be accomplished by efficient algorithms via dynamic programming. Evaluations are performed on two natural language tasks, i.e. Question Classification and Relation Extraction. Experimental results suggest that TSK outperforms tree kernel for both tasks, which also reveals that the structure features made up of multiple subtrees are effective and play a complementary role to the single tree structure.

Introduction

Kernel methods, which are able to efficiently evaluate the similarity between feature vectors, have been widely applied in many machine learning tasks. Basically, kernel methods employ a symmetric kernel function, of which the corresponding kernel matrix is positive semidefinite, to measure the similarity of dot product between two objects. By appropriately designing the kernel function, the dot product between the high dimensional feature vectors can be implicitly evaluated without enumerating all the features.

By applying in certain learning machines, such as Support Vector Machines, kernel methods can benefit many classification tasks in Natural Language Processing (NLP). In many cases, the input data of these tasks are expressed as sequences and trees. Consequently, kernel methods have been applied in these structures, i.e. sequence kernel (Lodhi et al. 2002) and tree kernel (Collins and Duffy 2002), both of which are instantiations of Convolution kernels (Hausler 1999). In terms of the tree structure, Collins and Duffy's tree kernel explores the feature space of all subtree types in the syntactic parse tree. To compute the similarity between two parse trees, the kernel function is evaluated by counting the number of common subtrees in each type. Tree kernel

has been successfully applied in many tasks such as syntactic parsing (Collins and Duffy 2002), question classification (Moschitti 2006), semantic parsing (Moschitti 2004), relation extraction (Zhang, Zhou, and Aw 2008) and bilingual alignment (Sun, Zhang, and Tan 2010). In view of the success of tree kernel, additional attempts have been made to enlarge the substructure space. Partial tree kernel (Moschitti 2006) allows partial rule matching by ignoring some child nodes in the original production rule. Grammar driven tree kernel (Zhang et al. 2008) modifies the original rules by generalizing certain grammar tags.

Alternatively, we propose Tree Sequence Kernel (TSK), which adopts the structure of a sequence of subtrees other than the single tree structure. Unlike previous works (Moschitti 2006; Zhang et al. 2008) which tend to modify the original rules in a parse tree, TSK strictly complies with the original production rules. Leveraging sequence kernel and tree kernel, TSK enriches sequence kernel with syntactic structure information and enriches tree kernel with disconnected subtree sequence structures. By combining the benefits of both structures, it is expected that TSK would be more powerful than the single tree based kernels.

To achieve the integration of sequence kernel into tree kernel, we propose Set Sequence Kernel (SSK) at first, which allows multiple choices of symbols in any position of a sequence. SSK is then combined with tree kernel to accomplish the evaluation of TSK. We propose two SSKs with different perspectives in penalizing the substructures. In consequence, two TSKs are correspondingly constructed. To verify their effectiveness, we apply TSK in two NLP tasks, i.e. Question Classification and Relation Extraction. The evaluation is conducted against Collins and Duffy's tree kernel. Experimental results suggest that TSK outperforms tree kernel, which consequently ensures the effectiveness of the tree sequence features in the corresponding tasks.

The Tree Sequence Structure

A tree sequence embedded in a parse tree is the structure of an arbitrary number of subtrees. When the subtree number is restricted to 1, the tree sequence structure is equivalent to a single tree structure. In other words, single tree structure is a special case of the tree sequence structure. We present in Fig. 1(b,c,d) some examples of tree sequence structures embedded in the parse tree of Fig. 1(a). The subtree sequence

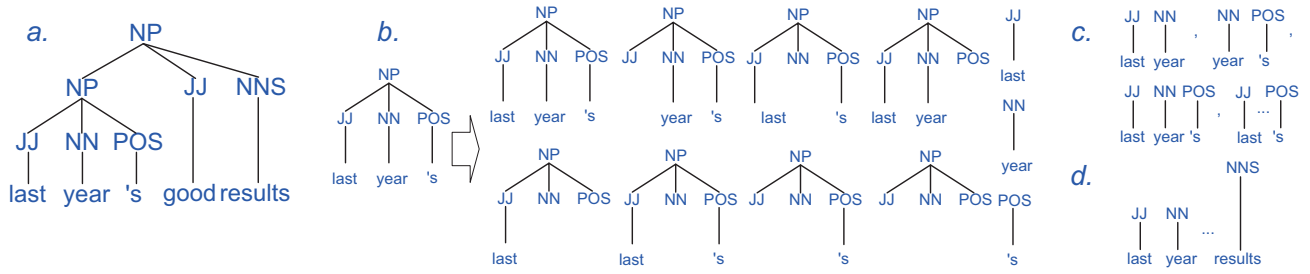


Figure 1: Illustration for Tree Sequence Structure

can be either contiguous, i.e. covering a contiguous context (Fig. 1(b) and first three in Fig. 1(c)) or be non-contiguous, i.e. with gaps between the adjacent subtrees (last one in Fig. 1(c) and Fig. 1(d)).

From Fig. 1, we may find that TSK are advantageous over tree kernel in certain aspects.

First, TSK may enrich the single tree representation with additional meaningful patterns. For the case of contiguous spans, tree kernel can only capture syntactic information when the span is covered by a single tree structure. As shown in Fig. 1(b), 11 types of subtree features over the context of “last year’s” can be captured by tree kernel. However, certain meaningful patterns like “last year” cannot be individually captured. On the contrary, when the tree sequence structure is allowed, additional subtree sequence features can be captured as shown in Fig. 1(c), where the syntactic structure over “last year” can be captured as a subtree sequence consisting of two subtrees. As for the non-contiguous case, since single tree based kernels require the structure features to be connected, it cannot deal with disconnected structures. Instead, TSK is able to employ the structure of multiple subtrees that cover a non-contiguous context as shown in example Fig. 1(d) and the last example of Fig. 1(c).

Second, TSK may help to match large structures. In single tree based kernels, matching large structures is hard to achieve, since large structures tend to be very sparse in the data set. In addition, the structure may not be entirely beneficial. It is quite likely that only certain parts of a large structure can be employed as meaningful patterns. TSK addresses the large structure matching problem by decomposing a large structure into multiple disconnected parts and matching certain parts each time. When the meaningful parts are matched together, additional useful patterns can be captured by TSK. For example, in Fig. 1(a), to match the pattern of “last year” and “results” together, tree kernel can only consult to the entire tree structure which may be too sparse to be matched. Alternatively, TSK can match this pattern by a non-contiguous subtree sequence which consisting of three subtrees in Fig. 1(d).

The structure of tree sequence has been found to be effective in syntactic driven tasks, such as syntax based statistical machine translation. Sun, Zhang, and Tan (2009) propose non-contiguous tree sequence based translation model which employs tree sequence as translation equivalences. In this model, the tree sequence structure relaxes the structural constraint of single tree based models without sacrific-

ing the grammatical information embedded in each subtree. Hence, the tree sequence structure is beneficial in capturing the structural divergence beyond the syntactic constraints across languages. Witnessing their success in applying tree sequence in multilingual parse trees, we attempt to use tree sequence in monolingual applications.

Preliminary Knowledge

Before presenting the algorithms for TSK, we briefly review tree kernel and sequence kernel, which will benefit understanding of TSK afterwards.

Tree Kernel

Tree Kernel (Collins and Duffy 2002) assesses the number of common subtrees between two parse trees by implicitly evaluating the dot product of the subtree feature vectors in high dimensional space. Given two parse trees T and T' , the tree kernel function can be computed as

$$\begin{aligned} K_t(T, T') &= \sum_{i=1}^{|T|} \left(\sum_{n \in N_T} I_i(n) \cdot \sum_{n' \in N_{T'}} I_i(n') \right) \\ &= \sum_{n \in N_T} \sum_{n' \in N_{T'}} \Lambda(n, n') \end{aligned}$$

where N_T and $N_{T'}$ refer to the node sets of the parse trees T and T' . Let \mathcal{T} refer to the substructure space. Let $I_i(n)$ refer to an indicator function which equals to 1 iff the corresponding subtree is rooted at the node n and 0 otherwise.

Let $\Lambda(n, n') = \sum_{i=1}^{|\mathcal{T}|} (I_i(n) \cdot I_i(n'))$ evaluate the number of matched subtrees rooted at n and n' . $\Lambda(n, n')$ can be evaluated via dynamic programming as follows:

- (1) **If** the production rule rooted at n and n' are different, $\Lambda(n, n') = 0$;
- (2) **else if** both n and n' are POS tags, $\Lambda(n, n') = \lambda$;
- (3) **else** $\Lambda(n, n') = \lambda \prod_{j=1}^{nc(n)} (1 + \Lambda(c(n, j), c(n', j)))$.

where $nc(n)$ is the number of children of node n . Let $c(n, j)$ be the j -th child of node n . Let $\lambda \in [0, 1]$ be the decay factor for the depth of the subtree. Tree kernel can be evaluated in $O(|N_T| \cdot |N_{T'}|)$

Sequence Kernel

Sequence kernel applied in document classification explores the feature space with respect to bag of characters (Lodhi et

al. 2002) and bag of words (Cancedda et al. 2003). Given two sequence s and s' , sequence kernel match all identical subsequences shared by s and s' . The kernel function is then evaluated by the number of matched subsequences (both contiguous and non-contiguous).

Let Σ be a finite symbol set. Let $s = s_0 s_1 \dots s_{|s|-1}$ be a sequence with each item s_i to be a symbol, i.e. $s_i \in \Sigma, 0 \leq i \leq |s| - 1$. Let $\mathbf{i} = [i_1, i_2, \dots, i_m]$, where $0 \leq i_1 < i_2 < \dots < i_m \leq |s| - 1$, be a subset of indices in s . Let $s[\mathbf{i}] \in \Sigma^m$ refer to the subsequence of $s_{i_1} s_{i_2} \dots s_{i_m}$. Then for a certain subsequence length of m , the kernel function is defined as

$$\tilde{K}_m(s, s') = \sum_{\substack{u \in \Sigma^m \\ u' \in \Sigma^m \\ u=u'}} \left(\sum_{\mathbf{i}: u=s[\mathbf{i}]} p(u, \mathbf{i}) \cdot \sum_{\mathbf{j}: u'=s'[\mathbf{j}]} p(u', \mathbf{j}) \right)$$

where $p(u, \mathbf{i})$ and $p(u', \mathbf{j})$ are the respective weights contributed to the kernel value when u and u' are matched. The standard sequence kernel (Lodhi et al. 2002; Cancedda et al. 2003) employs a weighting function which penalizes both the number of *matching symbols* and the *length of gaps* using the *same* decay factor $\rho \in [0, 1]$. In other words, the penalty covers the whole span of a given subsequence from the very beginning symbol to the ending symbol.

$$p(u, \mathbf{i}) = \rho^{(i_m - i_1 + 1)} \\ p(u', \mathbf{j}) = \rho^{(j_m - j_1 + 1)}$$

Set Sequence Kernel

Before proposing TSK, we propose Set Sequence Kernel (SSK). Based on SSK, TSK can be well defined afterwards.

Let Σ be a finite symbol set. Let $S = S_0 S_1 \dots S_{|S|-1}$ be a *Set Sequence* with each item S_i to be an ordered symbol set, i.e. $S_i \subseteq \Sigma, 0 \leq i \leq |S| - 1$. Let $S_{(i, \hat{i})}$ be a symbol, i.e. $S_{(i, \hat{i})} \in S_i, 0 \leq \hat{i} < |S_i|$. Let $(\mathbf{i}, \hat{\mathbf{i}}) = [(i_1, \hat{i}_1), (i_2, \hat{i}_2), \dots, (i_m, \hat{i}_m)]$, where $0 \leq i_1 < i_2 < \dots < i_m \leq |S| - 1$ and $0 \leq \hat{i}_k < |S_k|$, be a subset of index pairs in S . In each index pair, the first element denotes the index of a symbol set and the second element denotes a specific element in this symbol set. Let $S[(\mathbf{i}, \hat{\mathbf{i}})] \in \Sigma^m$ refer to the subsequence $S_{(i_1, \hat{i}_1)} S_{(i_2, \hat{i}_2)} \dots S_{(i_m, \hat{i}_m)}$. Then for a certain subsequence of length m , the kernel is defined as

$$\tilde{K}_m(S, S') = \sum_{\substack{u \in \Sigma^m \\ u' \in \Sigma^m \\ u=u'}} \left(\sum_{\substack{(\mathbf{i}, \hat{\mathbf{i}}): \\ u=S[(\mathbf{i}, \hat{\mathbf{i}})]}} p(u, \mathbf{i}) \cdot \sum_{\substack{(\mathbf{j}, \hat{\mathbf{j}}): \\ u'=S'[(\mathbf{j}, \hat{\mathbf{j}})]}} p(u', \mathbf{j}) \right)$$

According to Sequence kernel, the penalizing function $p(u, \mathbf{i})$ can be the lexical span length. Although this may work well for character based sequence with each item covering exactly length of 1, it is very likely that the method would not adapt well on tree sequence, since the size of subtrees varies a lot. In addition, penalty on spans may violate the purpose of matching syntactic tree structures, that tree structures covering different length of spans are identically

considered. As a result, instead of adapting the standard sequence kernel function, we propose new kernel functions for Set Sequence using two alternative penalizing approaches. One is to penalize the *count of matching symbols*. The other is to penalize the *number of gaps* ignoring the span length i.e. each gap between two matched symbols only incurs the decay factor once, no matter how large span the gap covers.

To facilitate the illustration of TSK in later sections, we define $K_m(i, j)$ to be the kernel value that match all the m -length subsequences up to the indices i and j . Therefore, SSK that matches all the subsequences with length m can be defined as $K_m(|S| - 1, |S'| - 1) = \tilde{K}_m(S, S')$, which evaluates symbols from the 0-th set to the *last* set of S and S' . Thus we have transformed the kernel function from $\tilde{K}_m(S, S')$ to $K_m(i, j)$, where $0 \leq i < |S|$ and $0 \leq j < |S'|$ and evaluate the latter kernel using dynamic programming.

Penalizing Count of matching symbols (μ)

The kernel function $K_m(i, j)$ which penalizes the count of matching symbols can be evaluated as follows.

$$\begin{aligned} K_m''(i, j) &= 0 & \text{if } \min(i, j) < m - 1 \text{ or } i, j < 0 \\ K_m'(i, j) &= 0 & \text{if } \min(i, j) < m - 1 \text{ or } i, j < 0 \\ K_m'(i, j) &= 1 & \text{if } m = 0 \\ K_m''(i, j) &= K_m''(i, j - 1) + \sum_{\substack{s_i \in \Sigma \\ s_i \in S_i}} \sum_{\substack{s_j' \in \Sigma \\ s_j' \in S_j' \\ s_i = s_j'}} K_{m-1}'(i - 1, j - 1) \end{aligned}$$

$$\begin{aligned} K_m'(i, j) &= K_m'(i - 1, j) + K_m''(i, j) \\ K_m(i, j) &= \mu^m \cdot K_m'(i, j) \end{aligned} \quad (1)$$

To evaluate $K_m(i, j)$, we use two intermediate matching functions $K_m'(i, j)$ and $K_m''(i, j)$. Let $K_m'(i, j)$ refer to the kernel values when the penalty factor μ is not taken account. In other words, each pair of matching subsequence contributes exactly 1 to the kernel $K_m'(i, j)$. In Equation 1, $K_m'(i, j)$ is evaluated by the kernel value which does not match the symbols in the i -th set S_i , i.e. $K_m'(i - 1, j)$ and the value which matches the symbols in the i -th set S_i , i.e. $K_m''(i, j)$. Given that the i -th set S_i is matched, $K_m''(i, j)$ is evaluated by the kernel value which does not match the symbols in the j -th set S_j' , i.e. $K_m''(i, j - 1)$ and the kernel value which matches symbols in the i -th set S_i and the symbols in the j -th set S_j' .

Penalizing Count of Gaps (τ)

The kernel function $K_m(i, j)$, which penalizes the count of gaps in the sequence, can be evaluated as follows.

$$\begin{aligned} K_m^{(l)}(i, j) &= 0 & \text{if } \min(i, j) < m - 1 \text{ or } i, j < 0 \\ & & \text{for all } l \in \{1, 2, 3, 4\} \\ K_m^{(4)}(i, j) &= 1 & \text{if } m = 0 \\ K_m^{(1)}(i, j) &= K_m(i - 1, j - 1) \\ K_m^{(2)}(i, j) &= K_m^{(2)}(i, j - 1) + K_m^{(4)}(i, j - 1) \\ K_m^{(3)}(i, j) &= K_m^{(3)}(i - 1, j) + K_m^{(4)}(i - 1, j) \end{aligned}$$

$$K_m^{(4)}(i, j) = \sum_{\substack{s_i \in \Sigma \\ s_i \in S_i}} \sum_{\substack{s'_j \in \Sigma \\ s'_j \in S'_j \\ s_i = s'_j}} \tau^2 K_{m-1}^{(1)}(i-1, j-1) + \tau K_{m-1}^{(2)}(i-1, j-1) + \tau K_{m-1}^{(3)}(i-1, j-1) + K_{m-1}^{(4)}(i-1, j-1)$$

$$K_m(i, j) = \sum_{l=1}^4 K_m^{(l)}(i, j)$$

Similar with the previous section, we also propose certain intermediate matching functions, i.e. $K_m^{(l)}(i, j)$, $l \in \{1, 2, 3, 4\}$, to accomplish efficient evaluation of $K_m(i, j)$. Let $K_m^{(1)}(i, j)$ refer to the case where neither the symbols in set S_i or the symbols in set S'_j are matched in the subsequence. Let $K_m^{(2)}(i, j)$ refer to the case where a symbol in set S_i is matched but none of the symbols in set S'_j are matched in the subsequence. Let $K_m^{(3)}(i, j)$ refer to the case where none of the symbols in set S_i are matched but a symbol in set S'_j is matched in the subsequence. Let $K_m^{(4)}(i, j)$ refer to the case where a symbol in set S_i and a symbol in set S'_j match to each other in the subsequence. The decay factor τ is incurred in computing $K_m^{(4)}(i, j)$, when both symbols are matched. The evaluation of $K_m(i, j)$ can be achieved by the sum of all the four cases.

Generally, the evaluation of both kernel functions can be accomplished in $O(m|\Sigma|^{S_i}|S_i| \cdot |\Sigma|^{S'_j}|S'_j|)$, where m is maximal number of symbols allowed in a matched subsequence.

Tree Sequence Kernel

In order to evaluate TSK, we integrate the algorithms of SSK and tree kernel by means of certain modifications.

To apply the SSK algorithm, we first transform the parse tree structure into a valid Set Sequence structure. Given a parse tree T with span length L , the node set S_i for all the indices $0 \leq i < L$ can be constructed by delivering all the nodes n to set S_i , when the end index of the span covered by n equals to i . In Fig. 2, we construct the Set Sequence for a parse tree with span $[0, 4]$ by sending the internal nodes to the set S_0, \dots, S_4 . For example, since NN^1 is ended with index 1, we put NN^1 in S_1 .

The general idea of the TSK evaluation is to match the subtrees in a subtree sequence from left to right and from top to bottom. Given a subtree sequence to be matched, the key issue to achieve this goal is the root node rooted at each subtree. When the root node n of a subtree t is matched, we need to *vertically* move downwards and match the entire subtree structure t . In addition, we also need to *horizontally* move to the right side of t and match the root node n' of the subtree t' that is adjacent to t . A gap is allowed between the subtree t and the subtree t' . Consequently, the subtree sequence matching problem is transformed into a problem of matching the root node sequences and the single tree structures. To implement this idea in dynamic programming, the *horizontal* evaluation is achieved by incurring SSK on the node sequence set of the given parse tree pair. In other words, any matched node in the node set will be considered as the root of the subtree to be matched. At the same time, the *verti-*

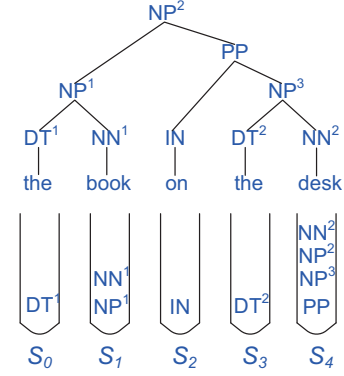


Figure 2: Construction of Node Set Sequence

cal evaluation is achieved by using the tree kernel function $\Lambda(\cdot, \cdot)$ within the *root-matched* subtrees.

Before presenting the algorithms, we define certain notations. Let Π be a finite node symbol set. For any node n , let n_b refer to the beginning index of the span covered by n , while n_e refer to the end index. Take the node NN^3 in Fig. 2 for example. We have $NN_b^3 = 3$ and $NN_e^3 = 4$.

Penalizing Count of matching subtrees (μ)

At the beginning, we modify certain kernel functions in the context of SSK to adapt the SSK computation to TSK with the penalty of the count of matching subtrees. Briefly, the adaptation of SSK to TSK is achieved by two major modifications:

First, TSK incurs the tree kernel computation of $\Lambda(n, n')$ when the node n ended with i matches the node n' ended with j . This is achieved by modifying $K_m''(i, j)$ in the context of SSK.

Second, each symbol matched in SSK is considered as the root node of a subtree in the matching subtree sequence. When a node pair (n, n') is matched in $K_m''(i, j)$, it reuses the kernel values before the beginning indices of (n, n') . This avoids the matching of other nodes in the span covered by (n, n') , since any two subtrees in a given subtree sequence cannot overlap. This again requires modifying $K_m''(i, j)$ in the context of SSK by changing $K'_{m-1}(i-1, j-1)$ to $K'_{m-1}(n_b-1, n'_b-1)$.

Therefore, with other equations to be the same, we only need to modify $K_m''(i, j)$ in SSK as follows:

$$K_m''(i, j) = K_m''(i, j-1) + \sum_{\substack{n \in \Pi \\ n_e = i}} \sum_{\substack{n' \in \Pi \\ n'_e = j \\ n = n'}} K'_{m-1}(n_b-1, n'_b-1) \cdot \Lambda(n, n')$$

Penalizing Count of Gaps (τ)

Similar modification is required for TSK evaluation when penalizing the count of gaps. In this kernel, the tree kernel computation $\Lambda(n, n')$ is incurred at the evaluation of $K_m^{(4)}(i, j)$. Therefore, with other equations to be the same,

the modification is only required for $K_m^{(4)}(i, j)$ as follows:

$$K_m^{(4)}(i, j) = \sum_{\substack{n \in \Pi \\ n_e = i}} \sum_{\substack{n' \in \Pi \\ n'_e = j \\ n = n'}} \left(\begin{array}{l} \tau^2 K_{m-1}^{(1)}(n_b-1, n'_b-1) \\ + \tau K_{m-1}^{(2)}(n_b-1, n'_b-1) \\ + \tau K_{m-1}^{(3)}(n_b-1, n'_b-1) \\ + K_{m-1}^{(4)}(n_b-1, n'_b-1) \end{array} \right) \cdot \Lambda(n, n')$$

In fact, if the minimum matching structure is restricted to a CFG rule with the height of 2, the set sequence can be constructed by the set of production rules, instead of simple nodes. In real implementation, we also rank the elements by alphabetical order for each node set (as shown in Fig. 2) to achieve fast matching (Moschitti 2006). The time complexity of TSK is incurred by the time cost of SSK and tree kernel. If we consider that tree kernel is evaluated before SSK. The cost is the sum of both kernels to be $O(m|N_T| \cdot |N_{T'}| + |N_T| \cdot |N_{T'}|)$, where m is the maximal number of subtrees allowed to be matched. In fact, we evaluate TSK by incurring SSK and tree kernel simultaneously, since partial results of both kernels can be reused. Briefly, the overall cost is $O(m|N_T| \cdot |N_{T'}|)$.

Experiments

To assess the effectiveness of TSK, we apply it in two NLP applications, i.e. Question Classification and Relation Extraction. Both tasks are addressed as a multi-class classification problem. We use the one vs. others strategy to choose the category with the largest margin. In our implementation, TSK is integrated into the tree kernel tool (Moschitti 2006), which employs SVM as the learning machine. To be consistent with previous work, the experimental results of Question Classification are demonstrated with classification accuracy, while for Relation Extraction evaluations are carried out with respect to Precision (P), Recall (R) and F-measure (F). We compare TSK with Collins and Duffy’s Tree kernel (TK) for both tasks.

Question Classification

Question Classification is a crucial sub-task for implementing question answering systems. Given a question sentence, it is necessary to identify what specific issues the question concerns. In our experiment, we follow the standard experimental setup in the previous work (Moschitti 2006). The experiments are conducted on the coarse grained question taxonomy with six major categories: Abbreviations, Descriptions, Entity, Human, Location and Number. We use the TREC data (Li and Roth 2002). The corpus consists of 5.5k labeled questions for training and 0.5k questions for testing. Stanford parser (Klein and Manning 2003) is used to generate the phrasal parse tree structures.

In experiments, different penalty factors for TSK are demonstrated and compared. Additionally, we also construct experiments using a polynomial kernel $d = 2$ over the bag of words (BoW) and bag of N-grams (BoN) respectively. All the experiments are conducted on different scale of training data from 1k to the 5.5k with respect to the original division in the data set.

	1k	2k	3k	4k	5.5k
BoW	78.0	84.0	85.8	86.6	87.2
BoN	74.8	81.6	82.8	86.4	88.0
TK	83.8	87.8	90.2	89.4	91.0
TSK μ	83.6	87.4	90.0	89.6	91.4
TSK τ	84.6	88.4	90.0	91.2	92.4

Table 1: Accuracy of Question Classification

We choose $C = 10$ for the SVM classifier. By 5-fold cross validation on the 5.5k training data, we choose the parameters for the corresponding kernels: $\lambda = 0.2$ for TK; $\lambda = 0.25$ and $\mu = 0.05$ for TSK μ ; $\lambda = 0.15$ and $\tau = 0.25$ for TSK τ .

As shown in Table 1, we can observe that TSK τ achieves the optimal performance. It outperforms TK by 1.4 point of accuracy when using the entire training corpus. This indicates the advantage of the additional tree sequence features in question classification. On the other hand, TSK μ outperforms TK by a little amount of accuracy when using larger training data (4k & 5.5k). However, when using small training data (1k & 2k), TSK μ underperforms TK. It’s easy to attribute this result to the fact that the additional large structures matched by TSK tend to be very sparse in small data. However, we also notice that TSK τ outperforms TK in small data. The inconsistency between the performance of TSK μ and TSK τ in small data when compared with TK indicates that the tree sequence with many gaps may not well contribute to the classification accuracy and the penalty of τ is essential for penalizing those structures.

Furthermore, both TK and TSK achieve better performance than the polynomial kernel only using semantic information. This indicates the syntactic features in phrasal parse tree are very useful for detecting question types.

Relation Extraction

Relation Extraction is a task to find various semantic relations between entity pairs in the document. For example, the sentence “*Larry Page was Google’s founding CEO*” conveys a semantic relation of “EMPLOYMENT.executive” between the entity pairs “*Larry Page*” (entity type: person) and “*Google*” (entity type: company).

We follow the experimental setup of Zhang, Zhou, and Aw (2008) as our baseline. We use the corpus (ACE 2004, LDC2005T09) with 348 documents and 4400 relation instances. The corpus consists of 7 types of entities, 7 major relation types and 23 relation subtypes. Since Zhang, Zhou, and Aw (2008) reported that using the path-enclosed (PT) tree structure instead of the complete constituent (MCT¹), achieves more improvement, we also follows them to use the PT structure as our tree structure. We adopt Charniak parser (Charniak 2001) to generate the phrasal parse trees. Before entity identification, we extract all entity mentions appeared in the same sentences as relation candidate pairs.

¹MCT 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.

	P	R	F	λ	μ	τ
TK _{flat}	71.47	56.55	63.14	0.3	–	–
TSK _{μ.flat}	72.39	58.25	64.55	0.1	0.7	–
TSK _{τ.flat}	72.75	58.00	64.55	0.1	–	0.8
TK _{multi}	70.98	57.88	63.77	0.3	–	–
TSK _{μ.multi}	74.35	59.83	66.30	0.2	0.7	–
TSK _{τ.multi}	74.92	59.83	66.53	0.2	–	0.7

Table 2: Performance for Relation Extraction

Originally, the candidate entities are encapsulated with five entity types (i.e. person, organization, location, facility and geo-political entity) and three entity mentions (i.e. names, nominal expressions and pronouns). We propose two methods to integrate the entity information. One is to use the combination of the two levels of annotation as a flat grammar tag (*flat*). The other is to interpret it as a multi-layer tree structure (*multi*):

e.g. (NN Google (ORG, NAME, Entity))

flat: (Entity-NAME-ORG (NN Google))

multi: (Entity (NAME (ORG (NN Google))))

We choose $C = 7$ for SVM. By a 5-fold cross validation on the training data, we choose all the kernel parameters as listed with the corresponding results in Table 2.

In Table 2, we find that both TSKs outperform TK in all metrics for both representations. Specifically, The optimal performance is accomplished by TSK penalizing the count of gaps (TSK τ) in *multi*-layer style, which outperforms TK_{multi} by 2.76 point of F-score. In addition, we also observe that the multi-layer representation is more effective than the flat representation for integration of the entity information. Furthermore, by introducing more structure information from the multi-layer entity, TSK also gains more improvement than using the flat integration.

Discussion

Compared with Question Classification, Relation Extraction gains more improvement by using TSK. The effectiveness of TSK on Relation Extraction can be attributed to the fact that TSK can capture multiple items in a non-contiguous tree sequence simultaneously. Relation Extraction is a task focusing on paired relational constituents, which are highly likely to be disconnected. Therefore, it will be useful to capture patterns consisting of information over both entities as a single structure, which is hard to be captured by the single tree based kernels. As a result, TSK, which is able to capture patterns across both entities may benefit Relation Extraction. Unlike Relation Extraction, Question Classification may consider non-contiguous patterns to be less important, since it is sometimes unnecessary to deeply explore the non-contiguous patterns for certain question sentences. For example, the question sentences beginning with single inquiry words “where” and “when” are easy to be correctly categorized. On the other hand, the improvement on classification accuracy still suggests that some question sentences indeed receive benefit from TSK.

Conclusion

In this paper, we propose Tree Sequence Kernel (TSK), which combines the advantages of both sequence kernel and tree kernel. The key characteristic of the proposed kernel is that the captured structure features are enlarged from the structure of a single tree to the structure of multiple trees (tree sequence). Two kernel functions with different penalty factors are presented. Both TSKs can be efficiently evaluated within $O(m|N_T| \cdot |N_{T'}|)$, where m is the maximal number of subtrees allowed in a matched tree sequence and N_T is the number of tree nodes. Experimental results on Question Classification and Relation Extraction suggest that the proposed TSKs outperform the tree kernel in both applications. Specifically, the structure of tree sequence more facilitates the task that focuses on disconnected constituents modeling, i.e. Relation Extraction.

References

- Cancedda, N.; Gaussier, E.; Goutte, C.; and Renders, J. 2003. Word sequence kernels. *The Journal of Machine Learning Research* 3:1059–1082.
- Charniak, E. 2001. Immediate-head parsing for language models. In *Proceedings of ACL*, 124–131.
- Collins, M., and Duffy, N. 2002. Convolution kernels for natural language. In *Proceedings of NIPS*, 625–632.
- Haussler, D. 1999. Convolution kernels on discrete structures.
- Klein, D., and Manning, C. 2003. Accurate unlexicalized parsing. In *Proceedings of ACL*, 423–430.
- Li, X., and Roth, D. 2002. Learning question classifiers. In *Proceedings of COLING*, 1–7.
- Lodhi, H.; Saunders, C.; Shawe-Taylor, J.; Cristianini, N.; and Watkins, C. 2002. Text classification using string kernels. *The Journal of Machine Learning Research* 2:419–444.
- Moschitti, A. 2004. A study on convolution kernels for shallow semantic parsing. In *Proceedings of ACL*, 335–342.
- Moschitti, A. 2006. Efficient convolution kernels for dependency and constituent syntactic trees. In *Proceedings of ECML*, 318–329.
- Sun, J.; Zhang, M.; and Tan, C. 2009. A non-contiguous tree sequence alignment-based model for statistical machine translation. In *Proceedings of ACL*, 914–922.
- Sun, J.; Zhang, M.; and Tan, C. 2010. Exploring syntactic structural features for sub-tree alignment using bilingual tree kernels. In *Proceedings of ACL*, 306–315.
- Zhang, M.; Che, W.; Zhou, G.; Aw, A.; Tan, C.; Liu, T.; and Li, S. 2008. Semantic role labeling using a grammar-driven convolution tree kernel. *IEEE transactions on audio, speech, and language processing* 16(7):1315–1329.
- Zhang, M.; Zhou, G.; and Aw, A. 2008. Exploring syntactic structured features over parse trees for relation extraction using kernel methods. *Information Processing & Management* 44(2):687–701.