A Novel Composite Kernel for Finding Similar Questions in CQA Services

Jun Wang[1], Zhoujun Li[1], Xia Hu[2], and Biyun Hu[1]

[1] School of Computer Science and Engineering
Beihang University, 100191 Beijing, China
[2] School of Computing, National University of Singapore, 117590, Singapore
wangjun0706149@cse.buaa.edu.cn, lizj@buaa.edu.cn
huxia@comp.nus.edu.sg, hubiyun0706147@cse.buaa.edu.cn

Abstract. Finding similar questions in Community Question Answering (CQA) services plays more and more important role in current web and IR applications. The task aims to retrieve historical questions that are similar or relevant to new questions posed by users. However, traditional "bag-of-words" based models would fail to measure the similarity between question sentences, as they usually ignore sequential and syntactic information. In this paper, we propose a novel composite kernel to improve the accuracy in question matching. Our study illustrate that the composite kernel can efficiently capture both lexical semantics and syntactic information in a question sentence by leveraging word sequence kernel, POS tag sequence kernel and syntactic tree kernel. Experimental results on real world datasets show that our proposed method significantly outperforms the state-of-the-art models.

Keywords: question answering, similarity measure,tree kernel, string kernel.

1 Introduction

One of the emerging trends in Web information services is the growth of social collaborative applications, such as WikiAnswers, Baidu Knows, Yahoo!Answer etc. These applications bring together a network of self-declared "experts" to answer questions posted by other people. Community Question and Answering (CQA) service is one of these collaborative applications, which have attracted a large number of users both seeking and providing answers to a variety of questions on diverse subjects. These CQA services rapidly build up large number of question and answer (Q/A) pairs, and these pairs are valuable as linguistic resources for many researches. In order to analyze these Q/A pairs, there are many situations in which we wish to determine how similar two questions are. For example, "how can I lose weight in a few month?" and "are there any ways of losing pound in a short period?" are two questions asking for ways of losing weight [10], and we wish to determine that there is a high degree of semantic similarity between them, even though they may not share any actual terms in common.

Existing methods for computing text similarity have focused mainly on either large documents or individual words. However, they are not effective when using such methods to measure similarity between questions. One reason is that users form questions with natural language, where questions are encoded with various lexical, syntactic and semantic features. Some questions have identical meaning but are very different in lexical level, such as the examples given previously. Directly applying traditional document similarity measures based purely on the "bag-of-word"(BOW) may perform unsatisfactory in these cases.

Syntactic or semantic features hence become vital for such task. Syntactic analysis can provide more structured information not readily identified based on sequences of tokens alone. However, a general problem in Natural Language Processing is that as the processing gets deeper, it becomes less accurate. For example, the current accuracy of tokenization, chunking and sentence parsing for English is about 99%, 92%, and 90% respectively [13]. Algorithms based solely on deeper representations inevitably suffer from the errors in computing these representations. On the other hand, low level processing such as tokenization will be more accurate, and may contain useful information missed by deep processing of text. Systems based on a single level of representation are force to choose between shallower representations, which will have fewer errors, and deeper representations, which may be more general.

In this paper, we propose a composite kernel for finding similar questions in CQA archives. First we present a novel weighted tree kernel over full syntactic parsing trees. Based on the kernel proposed in [4], we present two extensions to the original kernel, making it more accurate when processing question sentences and more suitable for our application. Experimental results show that our proposed weighted tree kernel can capture structured syntactic information better and outperforms the original tree kernel. As the tree kernel based methods deeply depend on parsing tree, they inevitably suffer from the errors in parsing. To solve this problem, we combine our proposed weighted tree kernel with another two string kernels and develop a novel composite kernel. There are two reasons of doing so. First, the parsing errors can be compensated by combining with other kernels. Second, we can fully utilize diverse information from both shallow representation and deep representation. Compared with state-of-the-art methods, experiments show that our proposed composite kernel achieve significant improvements.

There are three primary research contributions in this paper:

1. Modified the original tree kernel by presenting two extensions.
2. Proposed a composite kernel by combining the modified tree kernel and two string kernels, and applied it to finding similar questions in CQA archives.
3. Compared the proposed composite kernel with several other methods for similar questions matching problem.

The rest of the paper is organized as follows. Section 2 provides a review of related work. In Section 3 we present a composite kernel by combining a modified tree kernel and two string kernels. Section 4 reports the experiments results and our observation. In section 5, we draw our conclusion in this work.

2 Related Work

Several techniques on similar question matching problem have been proposed in literatures. Representative related work can be classified into four classes: vector space model(VSM) based [1,6], language model(LM) based [2,5], and translation model based [9] and syntactic tree kernel-based [8].

Most of the existing work focuses on addressing the word mismatching problem between user questions and the questions in a QA archive. Burke et al. [1] combined lexical similarity and semantic similarity between questions to rank FAQs, where the lexical similarity was computed using a vector space model(VSM) and the semantic similarity was computed based on WordNet. Jijkoun and Rijke [6] used supervised learning methods to extract QA pairs from FAQ pages, and used a vector space model for QA pair retrieval. Cao et al. [2] used language models to exploit categories of questions for improving question search. Duan et al. [5] searched questions by identifying question topic and question focus with in a language modeling framework. Xue et al. [11] proposed a translation based language model for question search on CQA data.

Prior kernel-based methods for this task focus on using individual tree kernels to exploit tree structure-related features. Wang et al. [9] employed a syntactic tree kernel for finding similar questions in CQA archives. They parsed the questions into syntactic trees and used the similarity of the syntactic trees of the query question and the historical question to rank historical questions. However, as the language processing gets deeper, it becomes less accurate. The method proposed in [9] was solely based on syntactic tree (deeper representation), thus inevitably suffered from the errors in parsing.

In our study, we avoid this problem by using a novel composite kernel to cover more knowledge. Our method has the potential of effectively capturing not only most of the previous flat features, used in [1,2], and also the structured syntactic features, used in [9]. To our best knowledge, this is the first work of using a composite kernel method for finding similar questions in CQA archives.

3 A Novel Composite Kernel

In this section, we define a composite kernel by integrating a modified tree kernel, a word kernel and a POS tag kernel.

3.1 A Modified Tree Kernel

The main underlying idea of tree kernels is to compute the number of common substructures between two trees without explicitly considering the whole fragment space [8]. In order to capture more syntactic information between parsing trees, Collins [4] tried to consider all tree fragments that occur in a syntactic parsing tree. According to his definition, a tree fragment of a syntactic parsing tree, which we call syntactic tree fragments (STFs), is parts of a syntactic parsing tree which include at least one production rule, with the restriction that no production rules can be broken into incomplete parts.

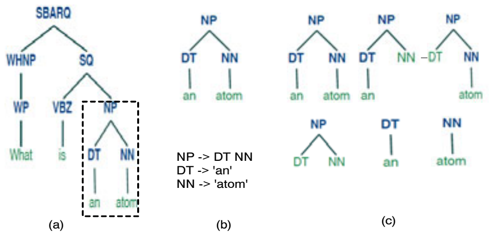

Fig. 1. (a) The syntactic tree of the sample question. (b) One sub-tree and its production rules. (c) All tree fragments in the sub-tree.

Take the question 'What is an atom?' for example. Figure 1(a) depicts the syntactic tree of the sentence, Figure 1(b) is one of its sub-trees and the production rules of the sub-tree, Figure 1(c) lists all tree fragments included in the sub-tree. In the original form of the tree kernel, it relies on mere common tree fragments counts, and lacks feature weighting schemes which have been deemed important by the IR community. In this section, we propose a weighted tree kernel by presenting two extensions to the original tree kernel. The first extension consists in adopting a depth decay factor for each syntactic tree fragments. The second one is that nodes in a tree are set different weight values according to its importance to the tree.

Syntactic Tree Fragment Depth Factor. Based on the original tree kernel framework, we consider a new decay factor: d for the depth of the syntactic tree fragments (STFs). The depth of each special STF tf_i is the level of tf_i root in the entire syntactic tree, with $d_{root} = 1$. Take the question 'what is an atom?' (Figure 1) for example again, the depth of the STF (Figure 1(b)) in the entire syntactic tree (Figure 1(a)) is 3.

Node Weighting Factor. Intuitionally, different parts of the sentence have different importance, and nouns and verbs are considered to be more important than other types of terms such as article, proposition and conjunction. Besides, as the data to be processed in our task are question sentences, we assume that the question *wh*-words have more impact on the performance. The *wh*-word is the question word in given questions. Here, the *wh*-words we considered are *what*, *why*, *where*, *when* and *how*. For example, the *wh*-word of question *'why does the moon turn orange? '* is *why*.

In our weighted tree kernel framework, we set different value for the importance of each none-terminal node in the entire syntactic tree for a special sentence. Particularly, we introduce the node weighting factor δ_i for the importance of node i in the entire syntactic tree. Here, we consider the following three situations:

1.$\delta_i = v_Q$*, where the POS tag label of node i is beginning with W(for question wh-word).*

2.$\delta_i = \upsilon_{\mathrm{NV}}$*, where the POS tag label of node* i *either beginning with* V *or* N *(for verbs or nouns).*
3.$\delta_i = 0.1$*, for all other types of POS tag label.*

where υ_Q is a tuning parameter indicating the importance of the *wh*-words nodes weighting factor and υ_{NV} is a tuning parameter indicating the importance of nouns or verbs nodes weighting factor. For each specific tree fragment i contained in the syntactic tree T, i is weighted by

$$\prod_{j=1}^{s(i)} \sqrt{\delta_j}\sqrt{\mu}^{d(i)}, \tag{1}$$

where $d(i)$ is the depth of i and μ is the weighting factor for the depth, $s(i)$ is the number of none terminal nodes included in i and δ_j is the weighting factor for each none terminal node in the syntactic tree fragment. The number of none terminal nodes included a syntactic tree fragment is equal to the number of production rules it contains.

Weighted Tree Kernel. Based on the above two extensions to the tree kernel, we reformulate the original tree kernel and name the new kernel as weighted tree kernel. Consider a syntactic tree T. Implicitly, we represent it by an m dimensional vector $V(T) = (v_1(T), v_2(T), \ldots, v_m(T))'$, where the i-th element $v_i(T)$ is the weight of the syntactic tree fragment in the i-th dimension if the syntactic tree fragment is in the tree T and zero otherwise.

Let N represent the set of nodes in T. Define the binary indicator function $I_i(n)$ to be 1 if syntactic tree fragment i is seen rooted at node n and 0 otherwise. Let $d(n)$ represent the depth of node n, if syntactic tree fragment i rooted at node n, then $d(i) = d(n)$,and then we can derive that :

$$v_i(T) = \sum_{n\in N}\prod_{j=1}^{s(i)} \sqrt{\delta_j}\sqrt{\mu}^{d(i)} I_i(n) = \sum_{n\in N}\prod_{j=1}^{s(i)} \sqrt{\delta_j}\sqrt{\mu}^{d(n)} I_i(n) \tag{2}$$

Given two syntactic tree T_1 and T_2, the weighted tree kernel function over T_1 and T_2 can be computed as

$$WTK(T_1, T_2) = \sum_i v_i(T_1)v_i(T_2) = \sum_{n_1\in N_1}\sum_{n_2\in N_2} \mu^{(d(n_1)+d(n_2))/2} K(n_1, n_2)\ , \tag{3}$$

where

$$K(n_1, n_2) = \sum_i \prod_{j=1}^{s(i)} \delta_j I_i(n_1) I_i(n_2) \tag{4}$$

In order to apply the weighted tree kernel function to measure the similarity of two sentences, we normalize it into the following forms:

$$sim(T_1, T_2) = \frac{WTK(T_1, T_2)}{\sqrt{WTK(T_1, T_1)}\sqrt{WTK(T_2, T_2)}} \tag{5}$$

The value of $K(n_1, n_2)$ can be computed recursively as follows,

1.$K(n_1, n_2) = 0$*, if* $n_1 \neq n_2$*;*
2.$K(n_1, n_2) = \delta_{n_1}$*, if* $n_1 = n_2$ *and* n_1,n_2 *are pre-terminal nodes;*
3.$K(n_1, n_2) = \delta_{n_1} \prod_{j=1}^{nc(n_1)}(1 + K(ch_{n_1}(j), ch_{n_2}(j)))$*, if* $n_1 = n_2$ *and* n_1,n_2 *are neither terminal nodes nor pre-terminal nodes.*

Here $n_1 = n_2$ if and only if the labels and production rules of node n_1 and n_2 are the same. The number of the node n's children is $nc(n)$, so if $n_1 = n_2$ then $nc(n_1) = nc(n_2)$. The j-th child of the node n is $ch_n(j)$. A terminal node means a leaf node, and a pre-terminal node means the immediate predecessor of a leaf node.

According to the dynamic programming idea, we can traverse T_1 and T_2 in post-order, fill in a $|N_1| \times |N_2|$ matrix of $K(n_1, n_2)$, then sum up the matrix entries weighted by $\mu^{(d(n_1)+d(n_2))/2}$. The time complexity of this algorithm turns out to be at most $o(|N_1| \times |N_2|)$.

3.2 POS Tag Kernel and Word Kernel

POS tag kernel (PK) and word kernel (WK) are two string kernels. String kernels [7] are similarity measures between documents seen as sequence of symbols (e.g., possible characters) over an alphabet. In general, similarity is assessed by the number of (possibly non-contiguous) matching subsequences share by two sequences.

Let $\sum$ be a finite alphabet, and $s = s_1 s_2 \ldots s_{|s|}$ a sequence over $\sum$ (i.e.$s_i \in \sum, 1 \leq i \leq |s|$). Let $\mathbf{i} = [i_1, i_2, \ldots, i_n]$, with $1 \leq i_2 \leq \ldots \leq i_n \leq |s|$, be a subset of the indices in s:we will indicate as $s[\mathbf{i}] \in \sum^n$ the subsequence $s_{i_1} s_{i_2} \ldots s_{i_n}$. Note that $s[\mathbf{i}]$does not necessarily from a contiguous subsequence of s. For example, if s is the sequence $CART$ and $\mathbf{i} = [2, 4]$, then $s[\mathbf{i}]$is AT. Let us write $l(\mathbf{i})$ the length spanned by $s[\mathbf{i}]$ in s, that is : $l(\mathbf{i}) = i_n - i_1 + 1$. The sequence kernel of two string s and t over $\sum$ is defined as:

$$K_n(s,t) = \sum_{\mu \in \sum^n} \sum_{\mathbf{i}:s[\mathbf{i}]=\mu} \sum_{\mathbf{j}:t[\mathbf{j}]=\mu} \lambda^{l(\mathbf{i})+l(\mathbf{j})}, \tag{6}$$

where $\lambda \in [0:1]$ is a decay factor used to penalize non-contiguous subsequences and the first sum refers to all possible subsequences of length n. Equation 6 defines a valid kernel as it amounts to performing an inner product in a feature space with one feature per ordered subsequence $\mu \in \sum^n$ with value:

$$\varPhi_\mu(s) = \sum_{\mathbf{i}:s[\mathbf{i}]=\mu} \lambda^{l(\mathbf{i})} \ . \tag{7}$$

Intuitively, this means that we match all possible subsequences of n symbols, even when these subsequences are not consecutive, and with each occurrence "discounted" according to its overall length. Choon et al. provide an efficient method for computing string kernels. For more information the readers may refer to [3].

POS tag kernel (PK) is obtained by applying the string kernel on the sequence of POS tags of a question. For example: given the sentence s_0: *what is autism?*, the associated POS tag sequence is *WP AUX NN?* and some of subsequences extracted by PK are *WP NN* or *WP AUX*. Word kernel (WK) is applied to word sequences of questions. Given s_0, sample substring are *what is autism*, *what is*, *what autism*, *is autism*, etc.

3.3 A Composite Kernel for Finding Similar Questions

Kernel engineering can be carried out by combining basic kernels with additive or multiplicative operators or by designing specific data objects (vectors, sequences and tree structures) for the target tasks. In this section, we define a novel composite kernel by combining a POS tag kernel, a word kernel and our proposed weighted tree kernel.

Kernels applied to new structures produce new kernels. Let $K(t_1, t_2) = \phi(t_1) \cdot \phi(t_2)$ be a basic kernel, where t_1 and t_2 are two trees. If we map t_1 and t_2 into two new structures s_1 and s_2 with a mapping $\phi_M(\cdot)$, we obtain:$K(s_1, s_2) = \phi(s_1) \cdot \phi(s_2) = \phi(\phi_M(t_1)) \cdot \phi(\phi_M(t_2)) = \phi'(t_1) \cdot \phi'(t_2) = K'(t_1, t_2)$, which is a noticeably different kernel induced by the mapping $\phi' = \phi \cdot \phi_M$. In this way, we define a composite kernel by combining three different kernels with linear combination:

$$\begin{aligned} CK(s_1, s_2) &= \alpha_1 WTK(tree(s_1), tree(s_2)) \\ &+ \alpha_2 PK(pos(s_1), pos(s_2)) \\ &+ \alpha_3 WK(word(s_1), word(s_2)) \ , \end{aligned} \tag{8}$$

where s_i means a question sentence, $tree(s_i)$ means mapping a question sentence s_i into a syntactic tree, $pos(s_i)$ means mapping s_i into a POS tag sequence and $word(s_i)$ means mapping s_i into a word sequence. Thus, the combination kernel can explore syntactic features, POS tag sequence features and word sequence features for one question sentence. Some preliminary experiments on a validation set showed that the composite kernel yield the best performance with $\alpha = 0.4_1$, $\alpha_2 = 0.3$ and $\alpha_3 = 0.3$.

4 Experiments

In this section, we present empirical evaluation results to assess the effectiveness of our proposed composite kernel for the similar question matching problem.

4.1 Dataset and Metrics

We selected 50 questions from TREC_10 [1]as query questions. Each query question is submitted to the Yahoo!Answers web services[2] and the 1,000 top-ranked

[1] Available at http://l2r.cs.uiuc.edu:80/ cogcomp/Data/QA/QC/TREC_10.label
[2] http://developer.yahoo.com/answers/V1/questionSearch.html

questions according to Yahoo!Answer ranking are collected as candidate questions. Then we re-rank the 1,000 candidate questions according to their similarity to the query. The top 10 candidate questions after being re-ranked are deemed similar or relevant to the query question. Seven methods are used for comparison, 1,213 candidate questions are manully judeged and 199 semantically similar questions are found in total for 50 queries.

Precision@n and Mean Average of Precision(MAP) are used as the performance measures. Precision@n is the fraction of the top-n questions retrieved that are relevant. Mean Average of Precision(MAP)[3] rewards approaches returning relevant questions earlier, and also emphases the rank in returned lists. MAP is defined as the mean of the precision at N values calculated after each relevant answers was retrieved. The final MAP value is defined as the mean average of average precisions of all queries in the test set.

4.2 Weighted Tree Kernel Performance Evaluation

In Section 3.1, we present two extensions to the original tree kernel, weighting factors υ_Q, υ_{NV} and the depth factor μ. If we set $\upsilon_Q = 0.1$, $\upsilon_{\mathrm{NV}} = 0.1$ and $\mu = 1$, our proposed weighted tree kernel (WTK) is equal to the original tree kernel(TK). In this section, we empirically evaluate the individual impact of these factors on the performance. First, we conduct experiment on the weighting factors: wh-word nodes weighting factor and noun or verb nodes weighting factor υ_{NV}. We evaluate the performance by setting υ_Q from 0.5 to 1.9 at increments of 0.1. The results are visually campared in Figure 2 (a) and Figure 2(b) . The *TK* line depicts the performance of the original tree kernel, while the $TK + upsilon\_Q / TK + upsilon\_\mathrm{NV}$ curve shows the impact of weighting factor $\upsilon_Q/\upsilon_{\mathrm{NV}}$ on the performance in MAP with different parameter settings.

Figure 2(a) shows the performance of $TK + \upsilon_Q$ with different wh-word nodes weighting factor υ_Q values while $\upsilon_{\mathrm{NV}} = 0.1$ and $\mu = 1$. The $TK + \upsilon_Q$ achives the best performance at the value of $\upsilon_Q = 0.05$. As the weight of *wh*-word nodes weighting factor grows, the performance declines. The reason may line in the corpus used in our experiment. We first submit our query question to Yahoo!Answer and the top-ranked 1,000 questions are used as our experiment corpus. The *wh*-word is usually the first word in the query and included in most of retrieved questions returned by Yahoo!Answer. So the *wh*-word has lower discrimination power in this situation. When we decline the impact of *wh*-word nodes weighting factor, The $TK + \upsilon_Q$ performance increases. Figure 2(b) depicts the performance of $TK + \upsilon_{NV}$ with different υ_{NV} values while $\upsilon_Q = 0.1$ and $\mu = 1$. The trend of $TK + \upsilon_{NV}$ reaches a peak at the value of $\upsilon_{\mathrm{NV}} = 0.13$. As the weight of noun or verb nodes grows, the performance of $TK + \upsilon_{\mathrm{NV}}$ declines. When $\upsilon_{\mathrm{NV}} > 0.16$, $TK + \upsilon_{\mathrm{NV}}$ performs worse than *TK*, which means the weighting factor brings in negative impact on performance.

We conducted extensive experiments on different parameter μ settings and found that when $\mu = 0.9$ $TK + \mu$ shows the best results. This parameter value is

[3] MAP_{10}: The MAP calculated on the returned top 10 questions.

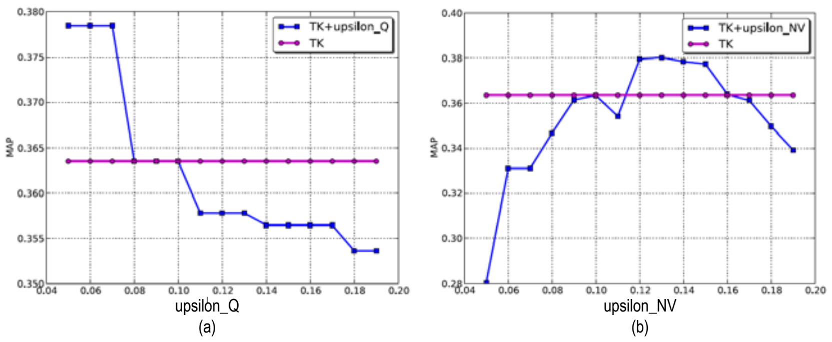

Fig. 2. (a) *wh*-word nodes weighting factor impact on performance. (b) Noun or verb nodes weighting factor impact on performance.

Table 1. Effect of depth factor

Method	**Precision@10** (*Impr*)	**MAP**$_{10}$ (*Impr*)
$TK_{tree}(baseline)$	0.285 $(N.A)$	0.364 $(N.A)$
$TK_{tree} + depthfactor$	0.346 (+%21.56)	0.372 (+%2.45)

same with [12]. In Table 1 we evaluate the performance of tree kernel(baseline) and our method with depth factor when $\upsilon_Q = 0.1$ and $\upsilon_{\mathrm{NV}} = 0.1$. As compared to baseline, we can achieve 21.56%, and 2.45% improvements in Precision@10, and MAP respectively. The overall performance of our proposed weighted tree kernel is listed in Table 4. Compared to TK, we can achieve 39.64%, and 9.06% improvements in Precision@10, and MAP respectively.

Based on these observations, we can see that our two extensions to the original tree kernel can both achive better performance than original tree kernel. It indicates that our proposed weighted tree kernel can capture structured syntactic information better than the original tree kernel.

4.3 Composite Kernel Performance Evaluation

To evaluate the performance of our composite kernel, we use seven different methods for comparison. Table 2 describes these methods and the parameter settings are listed in Table 3. In TK_{tree}, WTK_{tree},PK_{POS} and $CK_{word+POS+tree}$, question sentences are parsed into syntactic tree and POS tag sequence by using Stanford Parsers[4].

The experimental results are illustrated in Table 4. From the table, we draw the following observation:

• TK_{tree} performs better in $Precision$@10 and worse in MAP than VSM_{BOW}. The reasons may be that TK_{tree} ignores the word sequence information. The parsing error may be another reason that results in disappointed performance.

[4] http://nlp.stanford.edu/software/lex-parser.shtml

Table 2. Methods and description

Method	**Description**
VSM_{BOW}(baseline1)	Traditional vector space model with bag-of-words representation
LM_{BOW}(baseline2)	Language model with bag-of-words representation
TK_{tree}(baseline3)	Tree kernel with parse tree representation
WTK_{tree}	Weighted tree kernel with parse tree representation(proposed)
WK_{word}	Word kernel:String kernel on word sequences
PK_{POS}	POS tag kernel:String kernel on POS tag sequences
$CK_{word+POS+tree}$	Composite kernel(proposed)

Table 3. Methods and parameter setting

Method	**Parameter Setting**
VSM_{BOW}(baseline1)	-
LM_{BOW}(baseline2)	-
TK_{tree}(baseline3)	$\mu = 0.9$
WTK_{tree}	$\mu = 0.9$,$\upsilon_Q = 0.05$,$\upsilon_{NV} = 0.13$
WK_{word}	$\lambda = 0.9$, $n = 1$
PK_{POS}	$\lambda = 0.9$,$n = 3$
$CK_{word+POS+tree}$	$\alpha_1 = 0.4$, $\alpha_2 = 0.3$,$\alpha_3 = 0.3$

Table 4. Performance of different approaches measured by different metrics

Method	**Precision**@10 (*Impr*)	**MAP**$_{10}$ (*Impr*)
VSM_{BOW}	0.268 (*N.A.*)	0.403 (*N.A.*)
LM_{BOW}	0.330 (+%23.13)	0.445 (+%10.37)
TK_{tree}	0.285 (+%6.20)	0.364 (−%9.78)
WTK_{tree}	0.398 (+%48.51)	0.397 (−%1.36)
WK_{word}	0.359 (+%34.33)	0.465 (+%15.45)
PK_{POS}	0.350 (+%30.56)	0.483 (+%19.76)
$CK_{word+POS+tree}$	**0.400** (**+%49.25**)	**0.499** (**+%24.02**)

• WTK_{tree} performs better than TK_{tree} in all metrics. This suggests that our proposed weighted tree kernel can do better than original tree kernel in capturing structured syntactic information. However, it still performs worse than VSM_{BOW}. The reasons may be the same with TK_{tree}. This suggests that only considering the structured syntactic information does harm performance in finding similar questions.
• WK_{word} and PK_{POS} perform a bit better than VSM_{BOW}. This suggests that word sequence information and POS tag sequence have good predication power for finding similar questions.

• $CK_{word+POS+tree}$ achieves significant performance improvement over all the other mehtods. This indicates that structured syntactic information, word sequence and POS tag sequence information are complementary and the composite kernel can well integrate them: 1) the structured syntactic information captured by WTK_{tree}; 2) word sequence information captured by WK_{word}; 3) POS tag sequence information captured by PK_{POS}.

The main difference between our proposed composite kernel and traditional vector space model or language model lies in the feature represtation and similarity calculation mechanism(kernel function vs. dot-product). In composite kernel, we empoly three feature representation methods: parse tree, POS tag sequence and word sequence. While in traditional vector space model or language model, only bag of words representation is used. Therefore our method has the potential of effectively capturing not only most of the previous word level features but also the ussful syntactic structure features. Moreover, we present two extensions to the original tree kernel according to our application. Experimental results shows that our proposed weighted tree kernel can capture structured syntactic informatin better than original tree kernel and more suitable for processing question sentences in finding similar questions.

5 Conclusion

In this paper, we design a syntactic parsing tree kernel to exploit useful syntactic structure features. We present two extensions to the original tree kernel with a novel weighting scheme and a depth decay factor. A composite kernel is developed by combining our proposed weighted tree kernel, a POS tag kernel and a word kernel. Our study illustrates that the composite kernel could effectively capture both sequential and structured features. Experimental results show that our proposed model significantly outperforms the state-of-the-art methods.

The Stanford Parser is directly used in our experiments. However, this parser, including most off-the-shelf parsers, are not well-trained to parse question sentences. A more targetd parser which is trained on question sets may gain beeter accuracy. Besides, our proposed composite kernel only explores parse tree, POS tag sequence and word sequence information. Capturing more features may gain further improvements and this can be done by including more individual kernels such as WordNet-based semantic kernel and other feature-based kernels.

References

1. Burke, R.D., Hammond, K.J., Kulyukin, V.A., Lytinen, S.L., Tomuro, N., Schoenberg, S.: Question answering from frequently asked question files: Experiences with the faq finder system. AI Magazine 18(2), 57–66 (1997)
2. Cao, X., Cong, G., Cui, B., Jensen, C.S., Zhang, C.: The use of categorization information in language models for question retrieval. In: CIKM 2009, pp. 265–274. ACM, New York (2009)

3. Choon Hui Teo, S.: Fast and space efficient string kernels using suffix arrays. In: ICML 2006, pp. 929–936. ACM, New York (2006)
4. Collins, M., Duffy, N.: New ranking algorithms for parsing and tagging: kernels over discrete structures and the voted perceptron. In: ACL 2002, pp. 263–270. Association for Computational Linguistics (2002)
5. Duan, H., Cao, Y., Lin, C.-Y., Yu, Y.: Searching questions by identifying question topic and question focus. In: Proceedings of ACL 2008: HLT, pp. 156–164. Association for Computational Linguistics (June 2008)
6. Jijkoun, V., de Rijke, M.: Retrieving answers from frequently asked questions pages on the web. In: CIKM 2005, pp. 76–83. ACM, New York (2005)
7. Lodhi, H., Saunders, C., Shawe-Taylor, J., Cristianini, N., Watkins, C.: Text classification using string kernels. J. Mach. Learn. Res. 2, 419–444 (2002)
8. Moschitti, A.: Kernel methods, syntax and semantics for relational text categorization. In: CIKM 2008, pp. 253–262. ACM Press, New York (2008)
9. Wang, K., Ming, Z., Chua, T.-S.: A syntactic tree matching approach to finding similar questions in community-based qa services. In: SIGIR 2009, pp. 187–194. ACM Press, New York (2009)
10. Wang, X.-J., Tu, X., Feng, D., Zhang, L.: Ranking community answers by modeling question-answer relationships via analogical reasoning. In: SIGIR 2009, pp. 179–186. ACM, New York (2009)
11. Xue, X., Jeon, J., Croft, W.B.: Retrieval models for question and answer archives. In: SIGIR 2008, pp. 475–482. ACM, New York (2008)
12. Zhang, D., Lee, W.S.: Question classification using support vector machines. In: SIGIR 2003, pp. 26–32. ACM, New York (2003)
13. Zhao, S., Grishman, R.: Extracting relations with integrated information using kernel methods. In: ACL 2005, pp. 419–426. Association for Computational Linguistics (2005)