Linguistic Kernels for Answer Re-ranking in Question Answering Systems

Alessandro Moschitti

University of Trento, Italy

Silvia Quarteroni

University of Trento, Italy

Email addresses: alessandro.moschitti@disi.unitn.it (Alessandro Moschitti), silvia.quarteroni@disi.unitn.it (Silvia Quarteroni)

Linguistic Kernels for Answer Re-ranking in Question Answering Systems

Alessandro Moschitti

University of Trento, Italy

Silvia Quarteroni

University of Trento, Italy

Abstract

Answer selection is the most complex phase of a Question Answering (QA) system. To solve this task, typical approaches use unsupervised methods such as computing the similarity between query and answer, optionally exploiting advanced syntactic, semantic or logic representations.

In this paper, we study supervised discriminative models that learn to select (rank) answers using examples of question and answer pairs. The pair representation is implicitly provided by kernel combinations applied to each of its members. To reduce the burden of large amounts of manual annotation, we represent question and answer pairs by means of powerful generalization methods, exploiting the application of structural kernels to syntactic/semantic structures.

We experiment with Support Vector Machines and string kernels, syntactic and shallow semantic tree kernels applied to part-of-speech tag sequences, syntactic parse trees and predicate argument structures on two datasets which we have compiled and made available. Our results on classification of correct and incorrect pairs show that our best model improves the bag-of-words model by 63% on a TREC dataset. Moreover, such a binary classifier, used as a re-ranker, improves the Mean Reciprocal Rank of our baseline QA system by 13%.

These findings demonstrate that our method automatically selects an appropriate representation of question-answer relations.

Key words: Question Answering, Information Retrieval, Kernel Methods, Predicate Argument Structures

Email addresses: alessandro.moschitti@disi.unitn.it (Alessandro Moschitti), silvia.quarteroni@disi.unitn.it (Silvia Quarteroni)

1. Introduction

Automatic Question Answering (QA) systems return concise answers - i.e. sentences or phrases - to questions in natural language. On one hand, QA is interesting from an Information Retrieval (IR) viewpoint as it studies means to satisfy the user's information needs; on the other, the high linguistic complexity of QA systems suggests a need for more advanced natural language techniques, that have been shown to be of limited use for more basic IR tasks, e.g. document retrieval [1].

As a matter of fact, the main source of complexity in QA lies in the question processing and answer extraction steps rather than in document retrieval, a step usually conducted using off-the shelf IR modules [2, 3].

In question processing, useful information is gathered from the question to create a query; the latter is submitted to the document retrieval module that provides the set of the most relevant documents. The latter are used by the answer extractor to provide a ranked list of candidate answers. In the answer extraction phase, unsupervised methods are usually applied: a similarity between query and answer (such that higher similarity results in higher rank), is computed using simple *bag-of-words* (BOW) models or more advanced syntactic, semantic or logic representations, e.g. [4, 5]. More recently, shallow semantic information has been successfully exploited for such an approach in terms of predicate argument structures (PASs) [6].

In contrast, supervised machine learning methods that learn to rank answers from examples of question and answer pairs [7, 8] rarely use representation more complex than BOW. This is a major drawback, since different questions need different training data, and the only solution to overcome the burden of manual annotation is to reduce it by generalizing such data in terms of syntactic/semantic structures. In previous work, this consideration led us to defining supervised approaches to answer extraction using syntactic and shallow semantic structures. In particular, we proposed two tree kernel functions, named Shallow Semantic Tree Kernel (SSTK) [9] and Partial Tree Kernel (PTK) [10], that exploit PASs in PropBank[1] format for automatic answering to description questions. The use of shallow semantics appears to be especially relevant in the case of non-factoid questions, such as those requiring definitions, where the answer can be a whole sentence or paragraph containing only one question word.

In this paper, we present a thorough study on the above ideas by focusing on the use of kernel functions to exploit syntactic/semantic structures for relational

[1]`www.cis.upenn.edu/~ace`

learning from questions and answers. We start our study from simple linguistic levels and gradually introduce more and more advanced language technology. In more detail, we: (i) model sequence kernels for words and Part of Speech Tags that capture basic lexical semantics and syntactic information, (ii) apply tree kernels to encode deeper syntactic information and more structured shallow semantics and (iii) analyze the proposed shallow semantic kernels in terms of both accuracy and efficiency. Finally, we carry out comparative experiments between the different linguistic/kernel models on question/answer classification by measuring the impact of the corresponding classifiers on answer re-ranking.

It is worth noting that, since finding a suitable Question Answering corpus for our study was difficult[2], we designed and made available two different corpora, named WEB-QA and TREC-QA. Their main characteristic is that they relate to description questions from TREC 2001 [12], whose answers, retrieved from Web and TREC data, respectively, were manually annotated by our group.

The extensive experiments carried out on such corpora show that the generalization ability of kernel functions, successfully used in previous approaches [13, 14, 15, 16, 17, 18, 19, 20], is essential. Indeed, a unique result of our approach is that kernels applied to pairs of questions and answers are effective for automatically learning their relations. This is a further step in automation with respect to previous work such as [11], that required human effort and intuition to design a structural representation of question-answer pairs and use the latter to extract an effective feature vector representation[3].

Our main findings are that (i) kernels based on PAS, POS-tag sequences and syntactic parse trees improve on the BOW approach on both datasets: on TREC-QA, the improvement is high (about 63% in F1 score), making its application worthwhile; (ii) PTK for processing PASs is more efficient and effective than SSTK and can be practically used in answer re-ranking systems; and (iii) our best question/answer classifier, used as a re-ranker, improves the Mean Reciprocal Rank (MRR) of our QA basic system by 13%, confirming its promising applicability. Such improvement is much larger on WEB-QA .

In the remainder of this paper, Section 2 presents our use of kernel functions for structural information and Section 3 introduces the data representations we use for question and answer pairs. Section 4 reports on our experiments with different learning models and representations. Finally, Section 5 discusses our approach with respect to related work and our final conclusions are drawn in Section 6.

[2]For supervised approaches we could use neither the Japanese corpus used in [7, 8] nor the corpus used in [11], since they are not publicly available.

[3]Machine Translation techniques were applied to make this task easier.

2. Kernel Methods for Structured Data

Kernel Methods refer to a large class of learning algorithms based on inner product vector spaces, among which Support Vector Machines (SVMs) are well-known algorithms. The main idea behind SVMs is to learn a hyperplane $H(\vec{x}) = \vec{w} \cdot \vec{x} + b = 0$, where $\vec{x}$ is the representation of a classifying object o as a feature vector, while $\vec{w} \in \Re^n$ (indicating that $\vec{w}$ belongs to a vector space of n dimensions built on real numbers) and $b \in \Re$ are parameters learnt from training examples by applying the *Structural Risk Minimization principle* [21]. Object o is mapped into $\vec{x}$ via a feature function $\phi : \mathcal{O} \rightarrow \Re^n$, where $\mathcal{O}$ is the set of objects; o is categorized in the target class only if $H(\vec{x}) \geq 0$.

By exploiting the "kernel trick", the decision hyperplane can be rewritten as:

$$H(\vec{x}) = \Big(\sum_{i=1..l} y_i \alpha_i \vec{x}_i \Big) \cdot \vec{x} + b =$$

$$= \sum_{i=1..l} y_i \alpha_i \vec{x}_i \cdot \vec{x} + b = \sum_{i=1..l} y_i \alpha_i \phi(o_i) \cdot \phi(o) + b,$$

where y_i is equal to 1 for positive examples and to -1 for negative examples, $\alpha_i \in \Re$ (with $\alpha_i \geq 0$, o_i $\forall i \in \{1,..,l\}$) are the training instances and the product $K(o_i, o) = \langle \phi(o_i) \cdot \phi(o) \rangle$ is the kernel function associated with the mapping ϕ.

Note that instead of applying the mapping ϕ, we can directly use $K(o_i, o)$. Under Mercer's conditions [22], this allows to define abstract kernel functions generating implicit feature spaces. In turn, this alleviates the feature extraction/design step and enables the use of potentially infinite feature spaces, since the scalar product $K(\cdot, \cdot)$ is implicitly evaluated.

In the remainder of this section, we extensively describe the following kernels: the String Kernel (SK) proposed in [22] to evaluate the number of subsequences between two sequences, the Syntactic Tree Kernel (STK) [13], that computes the number of syntactic tree fragments, the Shallow Semantic Tree Kernel (SSTK) [9], that considers fragments from PASs, and the Partial Tree Kernel (PTK) [23], that provides a yet more general representation of trees in terms of tree fragments.

2.1. String Kernels

The String Kernels that we work with count the number of substrings shared by two sequences containing gaps, i.e. some of the characters of the original string are skipped. Gaps modify the weights associated with target substrings as shown in the following. Let Σ be a finite alphabet: $\Sigma^* = \bigcup_{n=0}^{\infty} \Sigma^n$ is the set of all possible strings. Given a string $\sigma \in \Sigma^*$, $|\sigma|$ denotes the length of σ, that can be written as $s_1..s_{|s|}$ with $s_i \in \Sigma$ and $\sigma[i : j]$ selects the substring $s_i s_{i+1}..s_{j-1} s_j$ from the

i-th to the j-th character. Now, u is a subsequence of σ if there is a sequence of indices $\vec{I} = (i_1, ..., i_{|u|})$, with $1 \leq i_1 < ... < i_{|u|} \leq |\sigma|$, such that $u = s_{i_1}..s_{i_{|u|}}$ or $u = \sigma[\vec{I}]$ in short. Moreover, $d(\vec{I})$ is the distance between the first and last character of the subsequence u in σ, i.e. $d(\vec{I})$ = $i_{|u|} - i_1 + 1$. Finally, given σ_1, σ_2 $\in \Sigma^*$, $\sigma_1\sigma_2$ indicates their concatenation.

The set of all substrings of a text corpus forms a feature space denoted by $\mathcal{F} \subset \Sigma^*$. To map a string σ into $\mathbb{R}^\infty$ space, we can use the following functions:

$$\phi_u(\sigma) = \sum_{\vec{I}:u=s[\vec{I}]} \lambda^{d(\vec{I})}$$

for some $\lambda \leq 1$. These functions count the number of occurrences of u in the string σ and assign them a weight $\lambda^{d(\vec{I})}$ proportional to their length. Hence, the inner product of the feature vectors for two strings σ_1 and σ_2 returns the sum of all common subsequences weighted according to their length and occurrence frequency:

$$SK(\sigma_1, \sigma_2) = \sum_{u \in \Sigma^*} \phi_u(\sigma_1) \cdot \phi_u(\sigma_2) = \sum_{u \in \Sigma^*} \sum_{\vec{I}_1:u=\sigma_1[\vec{I}_1]} \lambda^{d(\vec{I}_1)} \sum_{\vec{I}_2:u=\sigma_2[\vec{I}_2]} \lambda^{d(\vec{I}_2)}$$

$$= \sum_{u \in \Sigma^*} \sum_{\vec{I}_1:u=\sigma_1[\vec{I}_1]} \sum_{\vec{I}_2:u=\sigma_2[\vec{I}_2]} \lambda^{d(\vec{I}_1)+d(\vec{I}_2)}$$

It is worth noting that: (a) longer subsequences receive lower weights; (b) sequences of the original string with some characters omitted, i.e. gaps, are valid substrings; (c) gaps determine the weighting function since $d(.)$ counts the number of characters in the substrings as well as the gaps that were skipped in the original string, and (d) symbols of a string can also be whole words, as in the Word Sequence Kernel [24].

2.2. Tree Kernels

The main idea underlying tree kernels is to compute the number of common substructures between two trees T_1 and T_2 without explicitly considering the whole fragment space. Let $\mathcal{F} = \{f_1, f_2, \ldots, f_{|\mathcal{F}|}\}$ be the set of tree fragments and $\chi_i(n)$ an indicator function equal to 1 if the target f_i is rooted at node n and equal to 0 otherwise. A tree kernel function over T_1 and T_2 is defined as

$$TK(T_1, T_2) = \sum_{n_1 \in N_{T_1}} \sum_{n_2 \in N_{T_2}} \Delta(n_1, n_2),$$

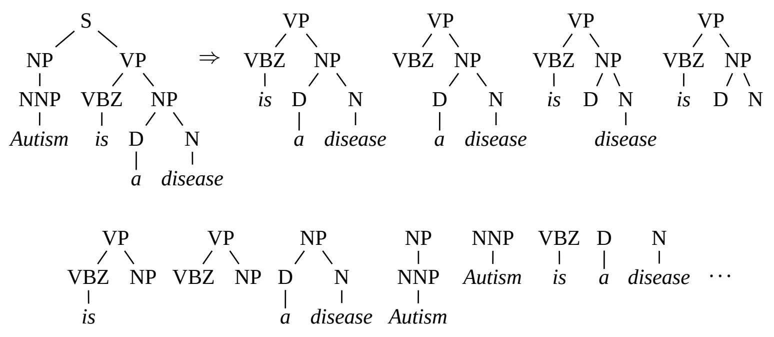


Figure 1: A tree for the sentence “Autism is a disease” (top left) with some of its syntactic tree fragments (STFs).

where N_{T_1} and N_{T_2} are the sets of nodes in T_1 and T_2, respectively, and

$$\Delta(n_1, n_2) = \sum_{i=1}^{|\mathcal{F}|} \chi_i(n_1)\chi_i(n_2).$$

The Δ function is equal to the number of common fragments rooted in nodes n_1 and n_2 and thus depends on the fragment type. Below, we report the algorithm to compute Δ for syntactic tree fragments (STFs) [13], shallow semantic tree fragments (SSTFs) [9], and partial tree fragments (PTFs) [23].

2.2.1. Syntactic Tree Kernel (STK)

A syntactic tree fragment (STF) is a set of nodes and edges from the original tree such that the fragment is still a tree, with the further constraint that any node must be expanded with either all or none of its children. This is equivalent to stating that the production rules contained in the STF cannot be partially applied.

To compute the number of common STFs rooted in n_1 and n_2, the Syntactic Tree Kernel (STK) uses the following Δ function [13]:

1. if the productions at n_1 and n_2 are different then $\Delta(n_1, n_2) = 0$;
2. if the productions at n_1 and n_2 are the same, and n_1 and n_2 have only leaf children (i.e. they are pre-terminal symbols) then $\Delta(n_1, n_2) = \lambda$;
3. if the productions at n_1 and n_2 are the same, and n_1 and n_2 are not pre-terminals then $\Delta(n_1, n_2) = \lambda \prod_{j=1}^{l(n_1)} (1 + \Delta(c_{n_1}(j), c_{n_2}(j)))$

where $l(n_1)$ is the number of children of n_1, $c_n(j)$ is the j-th child of node n and λ is a decay factor penalizing larger structures.

Figure 1 shows a tree and 10 out of its 17 STFs: note that STFs satisfy the constraint that grammatical rules cannot be broken. For example, [VP [VBZ NP]]

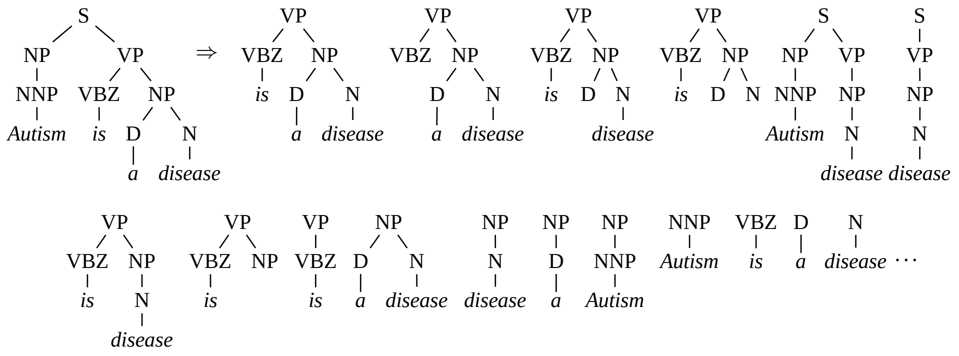


Figure 2: A tree for the sentence "Autism is a disease" (top left) with some of its partial tree fragments (PTFs).

is a STF which has two non-terminal symbols, VBZ and NP, as leaves while [VP [VBZ]] is not a STF. The computational complexity of STK is $O(|N_{T_1}||N_{T_2}|)$, although it is shown in [23, 25] that the average running time is linear in the number of tree nodes.

2.2.2. Shallow Semantic Tree Kernel (SSTK)

A shallow semantic tree fragment (SSTF) is almost identical to a STF, the difference being that the contribution of special nodes labeled with *null* should be zero. This is necessary as the Shallow Semantic Tree Kernel (SSTK) [9] is applied to special trees containing SLOT nodes that, when empty, have children labeled with *null*. Two steps are modified in the algorithm:

0. if n_1 (or n_2) is a pre-terminal node and its child label is *null*, $\Delta(n_1, n_2) = 0$;

3. $\Delta(n_1, n_2) = \prod_{j=1}^{l(n_1)} (1 + \Delta(c_{n_1}(j), c_{n_2}(j))) - 1$,

The above steps do not change the computational complexity of the original algorithm, which is therefore $O(|N_{T_1}||N_{T_2}|)$.

2.2.3. Partial Tree Kernel (PTK)

If we relax the production rule constraint over the STFs, we obtain a more general substructure type called partial tree fragment (PTF), generated by the application of partial production rules such as [VP [VBZ [is]]] in Figure 2. The Δ function for the Partial Tree Kernel (PTK) is the following. Given two nodes n_1 and n_2, STK is applied to all possible child subsequences of the two nodes, i.e. the String Kernel is applied to enumerate their substrings and the STK is applied on each of such child substrings. More formally:

1. if the node labels of n_1 and n_2 are different then $\Delta(n_1,n_2)=0$;
2. else

$$\Delta(n_1,n_2)=1+\sum_{\vec{I}_1,\vec{I}_2,l(\vec{I}_1)=l(\vec{I}_2)}\prod_{j=1}^{l(\vec{I}_1)}\Delta(c_{n_1}(\vec{I}_{1j}),c_{n_2}(\vec{I}_{2j}))$$

where $\vec{I}_1=\langle h_1,h_2,h_3,..\rangle$ and $\vec{I}_2=\langle k_1,k_2,k_3,..\rangle$ are index sequences associated with the ordered child sequences c_{n_1} of n_1 and c_{n_2} of n_2, respectively, $\vec{I}_{1j}$ and $\vec{I}_{2j}$ point to the j-th child in the corresponding sequence, and again, $l(\cdot)$ returns the sequence length, i.e. the number of children. Furthermore, we add two decay factors: μ for the depth of the tree and λ for the length of the child subsequences with respect to the original sequence, to account for gaps. It follows that

$$\Delta(n_1,n_2)=\mu\Big(\lambda^2+\sum_{\vec{I}_1,\vec{I}_2,l(\vec{I}_1)=l(\vec{I}_2)}\lambda^{d(\vec{I}_1)+d(\vec{I}_2)}\prod_{j=1}^{l(\vec{I}_1)}\Delta(c_{n_1}(\vec{I}_{1j}),c_{n_2}(\vec{I}_{2j}))\Big),$$

where $d(\vec{I}_1)=\vec{I}_{1l(\vec{I}_1)}-\vec{I}_{11}+1$ and $d(\vec{I}_2)+1=\vec{I}_{2l(\vec{I}_2)}-\vec{I}_{21}$+1. This way, both larger trees and child subsequences with gaps are penalized. An efficient algorithm for the computation of PTK is given in [23], where the worst case complexity is $O(\rho^3|N_{T_1}||N_{T_2}|)$, where ρ is the maximum branching factor of the two trees. Note that the average ρ in natural language parse trees is very small and the overall complexity can be reduced by avoiding the computation of node pairs with different labels [23].

2.3. *Kernel Engineering*

Kernel engineering can be carried out by combining basic kernels via additive or multiplicative operators or by designing specific data objects (vectors, sequences and tree structures) for the target task. It is worth noting that kernels applied to new structures produce new kernels. Indeed, 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:

$$K(s_1,s_2)=\phi(s_1)\cdot\phi(s_2)=\phi(\phi_M(t_1))\cdot\phi(\phi_M(t_2))=\phi'(t_1)\cdot\phi'(t_2)=K'(t_1,t_2),$$

that is a noticeably different kernel induced by the mapping $\phi'=\phi\circ\phi_M$. In this work, we use several such kernels, such as PAS$_{PTK}$ and POS$_{SK}$, obtained by applying PTK and SK to predicate argument structures and sequences of Part of Speech Tags, respectively.

3. Relational Representations for Question and Answer Pairs

Capturing the semantic relations between two text fragments is a complex task. In Question Answering, this task is carried out during answer extraction, where unsupervised approaches measure the similarity of questions and answers [4, 5, 26].

A key aspect of our work is that we apply supervised methods to learn such relations. More explicitly, we train classifiers for detecting whether an answer correctly responds to the corresponding question or not (the problem is formally defined in Sec. 3.1). This is a very different problem from typical answer extraction in that not only the relatedness between the target question and answer is taken into account, but also other question-answer training pairs are used. The similarity between pairs clearly depends on syntactic and semantic properties; thus, in addition to the usual bag-of-word approach (BOW), we study methods to capture Q/A structures using String Kernels over word and POS-tag sequences and tree kernels over full syntactic parse trees (PTs) and shallow semantic trees (PASs). The following sections describe the rationale behind our approach and the choice of such features.

3.1. Classification of Paired Texts

A Q/A classifier receives question-answer pairs $\langle q, a\rangle$ as input and judges whether the candidate answer a correctly responds to q. To design such a classifier, a set of examples of correct and incorrect pairs is needed. The learning algorithm operates by comparing the question and answer contents in a separate fashion rather than just comparing a question with its corresponding candidate answers. In a learning framework where kernel functions are deployed, given two pairs $p_1 = \langle q_1, a_1\rangle$ and $p_2 = \langle q_2, a_2\rangle$, a kernel function is defined as

$$K(p_1, p_2) = K_\tau(q_1, q_2) \oplus K_\alpha(a_1, a_2),$$

where K_τ and K_α are kernel functions defined over questions and over answers, respectively, and $\oplus$ is a valid operation between kernels, e.g. sum or multiplication.

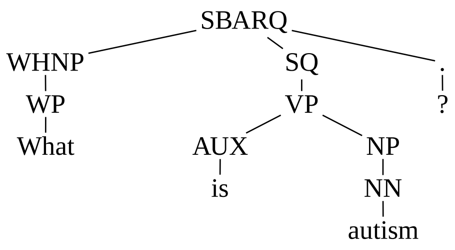
In Section 2, we described sequence and tree kernels, that can be applied to the sequential and tree representations of questions and answers, respectively. In the following sections we describe several of such linguistically motivated representations.

3.2. Representation via Word and POS-tag sequences and Trees

For a basic syntactic and semantic representation of both questions and answers, we adopt two different kernels: the Part of Speech Sequence Kernel (POS$_{SK}$)

and the Word Sequence Kernel (WSK). The former is obtained by applying the String Kernel on the sequence of POS-tags of a question or answer. For example, given the sentence s_0: *What is autism?*, the associated POS sequence is *WP AUX NN ?* and possible subsequences extracted by POS_{SK} are *WP NN* or *WP AUX*. Instead, WSK is applied to word sequences of questions or answers; given s_0, sample WSK substrings are: *What is autism*, *What is*, *What autism*, etc.

A more complete structure is the full parse tree (PT) of the sentence, that constitutes the input of the STK. For instance, the STK accepts the following syntactic parse tree for s_0:



3.3. *Shallow Semantic Representation*

Our semantic representation is motivated by the intuition - supported by computational linguistic evidence [27] - that definitions are characterized by a latent semantic structure, thanks to which *similar concepts* result in *structurally similar* formulations. Indeed, understanding whether a candidate answer is correct for a definition question would imply knowing the correct definition and comparing the current candidate to the former. When such information is unavailable (as in open domain QA) the learning algorithm must mimic the behavior of a human who does not know the exact definition but checks whether such an answer is formulated as a “typical” definition and possibly whether answers defining similar concepts are expressed in a similar way. A method to capture sentence structure is the use of predicate argument structures [28], described hereafter.

3.3.1. *Predicate Argument Structures*

Shallow approaches to semantic processing are making large strides in the direction of efficiently and effectively deriving tacit semantic information from text. Large data resources, annotated with semantic information as in the FrameNet [29] and ProbBank [30] projects, make it possible to design systems for the automatic extraction of predicate argument structures (PASs) [31]. Such systems identify predicates and their arguments in a sentence. For example, in the sentence, ‘John likes apples.', the predicate is 'likes' whereas 'John' and 'apples', bear the semantic role labels of *agent* ($A0$) and *theme* ($A1$). The crucial property about semantic roles is that regardless of the overt syntactic structure variation, the underlying predicates remain the same. For instance, given the sentences 'John found a bug in his code' and 'A bug was found in the code', although 'a bug' is the object of the first sentence and the subject of the second, it is the 'theme' in both sentences.

To represent PASs in the learning algorithm, we work with two types of trees: Shallow Semantic Trees for SSTK and Shallow Semantic Trees for PTK, both following PropBank definition, denoted by PAS$_{SSTK}$ and PAS$_{PTK}$, respectively. These are automatically generated by our system using the Semantic Role Labeling system described in [32]. As an example, let us consider sentence s_1: 'Autism is characterized by a broad spectrum of behavior that includes extreme inattention to surroundings and hypersensitivity to sound and other stimuli', resulting in the PropBank annotation a_1: [$_{A1}$ Autism] is [$_{rel}$ characterized] [$_{A0}$ by a broad spectrum of behavior] [$_{R-A0}$ that] [$_{rel}$includes] [$_{A1}$ extreme inattention to surroundings and hypersensitivity to sound and other stimuli].

Such an annotation can be used to design a shallow semantic representation to be matched against other semantically similar sentences, e.g. s_2: 'Panic disorder is characterized by unrealistic or excessive anxiety', resulting in a_2: [$_{A1}$ Panic disorder] is [$_{rel}$ characterized] [$_{A0}$ by unrealistic or excessive anxiety].

It can be observed that, although autism is a different disease from panic disorder, the structure of the two above definitions and the latent semantics they contain (inherent to behavior, disorder, anxiety) are similar. Indeed, s_2 would appear as a definition even to one who only knows what the definition of autism looks like.

The above annotation can be compactly represented by predicate argument structure (PAS) trees such as those in Figure 3. Here, we notice that the semantic similarity between sentences is explicitly visible in terms of common fragments extracted by PTK from their respective PASs, as illustrated in Figure 3(c). An equivalent PAS representation (PAS$_{SSTK}$) compatible with SSTK (see Section 2.2.2) was introduced in [9] (see Figure 4). Here, arguments follow a fixed ordering (i.e. rel, $A0$, $A1$, $A2$, $\ldots$) and a layer of SLOT nodes "artificially" allows SSTK to generate structures containing subsets of arguments.

3.3.2. PTK vs. SSTK applied to PAS

A comparison between SSTK and PTK suggests the following remarks: first, while PAS$_{PTK}$ is semantically equivalent to PAS$_{SSTK}$, PTK is able to extract a richer set of features which take gaps into account. This can be seen by comparing the first two fragments of Figures 3(c) and their equivalent in 4(b).

Second, PAS$_{PTK}$ does not need SLOT nodes to extract fragments containing argument subsets. This results in a visibly more compact representation (compare

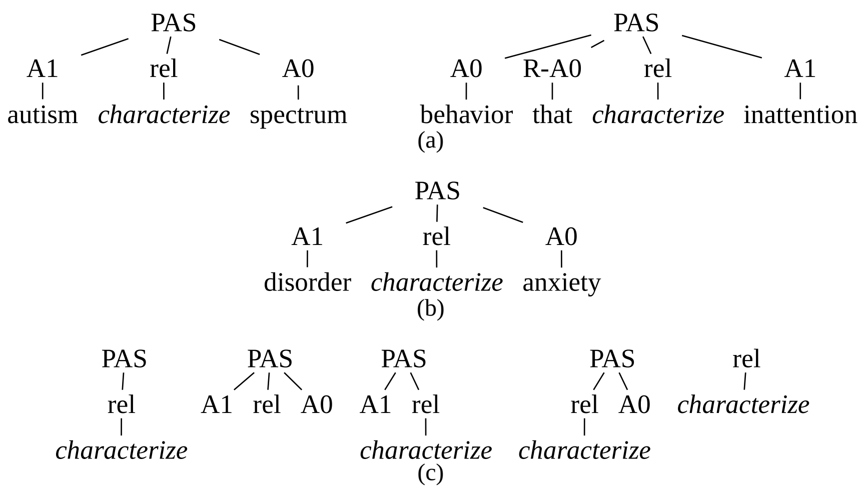


Figure 3: Compact PAS$_{PTK}$ structures of s_1 (a) and s_2 (b) and some fragments they have in common as produced by the PTK (c). Arguments are replaced with their most important word (or semantic head) to reduce data sparseness.

Figures 3(b) and 4(a)). Moreover, a more accurate computation of the matches between two PASs is performed, since only nodes that are actually useful are represented.

Third, although the computational complexity of PTK is greater than the one of SSTK, the structures to which PTK is applied are much smaller than those input to the SSTK. This makes PTK more efficient than SSTK. We show in the experiment section that the running time of PTK is much lower than that of SSTK (for both training and testing).

Next, another interesting difference between PTK and SSTK is that the latter requires an ordered sequence of arguments to evaluate the number of argument subgroups (arguments are sorted before running the kernel). This implies a loss of the natural argument order. In contrast, PTK is based on subsequence kernels thus it naturally takes order into account; this is very important as syntactic/semantic properties of predicates cannot be captured otherwise, e.g. passive and active forms have the same argument order in PAS$_{SSTK}$.

Finally, PTK weighs predicate substructures based on their length; this also accounts for gaps, e.g. the sequence $\langle$A0, A1$\rangle$ is more similar to $\langle$A0, A1, A-LOC$\rangle$ sequence than to $\langle$A0, A-LOC, A1$\rangle$, which in turn produces a better match than $\langle$A0, A-LOC, A2, A1$\rangle$ (cf. Section 2.1). This is another important property for modeling shallow semantic similarity.

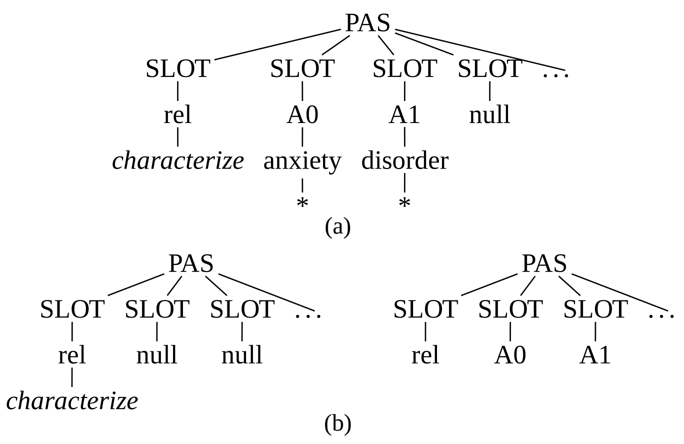


Figure 4: Compact PAS$_{SSTK}$ of s_2 (a) and some of its fragments produced by the SSTK (b).

3.4. YourQA, a baseline QA system

As mentioned earlier, our research focus is on non-factoid Question Answering, where the expected answer type mainly consists of definitions or descriptions. Non-factoid answer types are among the most complex and interesting in the literature [33, 34] as finding them requires deeper linguistic processing than for factoids.

Unfortunately, there has been limited interest this specific problem during official QA evaluation campaigns. TREC-10, the 2001 edition of the major QA evaluation campaign, remains to our knowledge the first and one of the few events where a large number of description or definition questions was included in the test set to be addressed by participant systems [12]. In a question classification taxonomy designed to account for this edition, 138 questions were labeled as "description"[4] [35]. We use the answers to such questions as a baseline to test our learning models.

In order to experiment with classifiers and re-rankers, an ordered list of candidate answers to each question is needed from an existing Question Answering system to obtain training instances for answer classifiers and evaluate their re-ranking abilities. To this end, we used YourQA [36], our Web-based Question Answering system, designed to address both factoid and non-factoid questions and return answers alternatively from the Web or from a closed corpus.

YourQA is organized according to three phases: question processing, document retrieval and answer extraction. During the first phase, the query is classified

[4]See `l2r.cs.uiuc.edu/~cogcomp/Data/QA/QC/`

according to a taxonomy of factoid or non-factoid answer types; the two top expected answer types are estimated and the query is submitted to the underlying IR engine. In the document retrieval phase, the top n documents found by the IR engine are retrieved and split into sentences. Finally, during answer extraction, a sentence-level similarity metric combining lexical, syntactic and semantic criteria is applied to the query and to each retrieved document sentence to identify candidate answer sentences; candidate answers are ordered by relevance to the query, while the IR engine rank of the answer source document is used as a tie-breaking criterion.

In particular, based on the outcome of the question classifier, the answer extraction module determines whether the expected answer belongs to the factoid group, i.e. PERS, ORG, LOC, QTY, or TIME. If this is the case, the required factoid contained in each candidate answer sentence is pinpointed down to the phrase or word level using relevant factoid QA techniques, involving the use of Named Entity recognizers and the use of regular expressions. In the case of non-factoid expected answer types, other similarity criteria are adopted to compute the similarity between the candidate answers and the original question; the final question-answer similarity metric $sim(q, a)$ results from a weighted combination of four simularity metrics, respectively based on bag-of-words (bow), $n-$grams (ng), syntactic chunks (chk), and head noun phrase-verb phrase-prepositional phrase (NP-VP-PP) groups (hd):

$$sim(q, a) = \alpha \times bow(q, a) + \beta \times ng(q, a) + \gamma \times chk(q, a) + \delta \times hd(q, a). \quad (1)$$

In particular, the bag-of-word similarity between the question q and a candidate answer a, $bow(q, a)$, is the number of matches between the question keywords q_i, with $i < |q|$, and the candidate answer keywords a_j, with $j < |a|$, normalized by dividing by the number of question keywords, $|q|$: $bow(q, a) = \frac{\sum_{i<|q|,j<|a|} match(q_i,a_j)}{|q|}$. As in many cases the presence of question keywords in a candidate answer is not a sufficient criterion to establish a strong similarity between the question and such an answer, we resort to n-gram similarity, defining $ng(q, a) = \frac{|commonN(q,a)|}{|ngrams(q)|}$, where $commonN(q, a)$ is the number of shared n-grams between q and a and $ngrams(q)$ is the set of question n-grams. In the current version of YourQA, $n = 2$.

Furthermore, chunk similarity $chk(q, a)$ is a function of the number of common sentence chunks[5] between q and a, $|commonC(q, a)|$. $|commonC(q, a)|$

[5]Chunks can be defined as groups of consecutive, semantically connected words in the sentence, which can be obtained using a shallow parser (in our case, the one provided by the OpenNLP chunker at `http://opennlp.sourceforge.net`)

is then divided by the total number of chunks in q, $|chunks(q)|$: $chk(q,a) = \frac{|commonC(q,a)|}{|chunks(q)|}$, where $commonC(q,a)$ is the number of shared chunks between q and a and $chunks(q)$ is the set of question chunks. Finally, $hd(q,a)$ is a variation of chunk similarity, where the focus is on word groups composed by a noun phrase, a verb phrase and a prepositional phrase (NP, VP and PP in short). Having identified the VPs in q and a that share the maximum number of tokens, named $maxVP_q$ resp. $maxVP_a$, we define $hd(q,a) = \mu \times HNP(q,a) + \nu \times VP(q,a) + \xi \times PP(q,a)$. Here, $VP(q,a)$ is the number of tokens shared between $maxVP_q$ and $maxVP_a$; $HNP(q,a)$ is the number of common tokens between the head NPs associated with $maxVP_q$ and $maxVP_a$, respectively, and $PP(q,a)$ is the number of common tokens between the PPs associated with $maxVP_q$ and $maxVP_a$, respectively; μ, ν and ξ are carefully chosen weights. The current version of YourQA uses $\mu = \nu = 0.4$, $\xi = 0.2$, while following empirical observation of YourQA's results, the α, β, γ and δ coefficients in (1) have been tuned to their current values of $\alpha = 0.6, \beta = 0.2, \gamma = \delta = 0.1$.

It must be noted that while the $sim(q,a)$ similarity metric in (1) takes as arguments a question q and one of its candidate answers a – as done by the vast majority of QA systems – the classification and re-ranking model proposed in Section 3.1 takes *question/answer pairs* as arguments (or learning instances). More concretely, the classifiers process two pairs at a time, $\langle q_1, a_1 \rangle$ and $\langle q_2, a_2 \rangle$, and compare q_1 with q_2 and a_1 with a_2 according to different functions, finally producing a combined similarity score. Such a comparison allows to determine whether an unknown question/answer pair contains a correct answer or not by assessing its distance from another question/answer pair with a known label. In particular, an unlabeled pair $\langle q_2, a_2 \rangle$ will be processed so that rather than "guessing" correctness based on words or structures shared by q_2 and a_2, both q_2 and a_2 will be compared to their correspondent components q_1 and a_1 of the labeled pair $\langle q_1, a_1 \rangle$ on the grounds of such words or structures.

To exemplify this, if q_1 is "What is autism?" and the candidate answers are a_1 "Autism may be defined as a mental disease" vs $a_1^{'}$ "Autism affects many people", comparison with the correct pair formed by q_2 "What is a golden parachute?" and a_2 "A golden parachute may be defined as a manager's privilege" will induce the kernel method to prefer a_1 to $a_1^{'}$. Indeed, a_1 has a similar wording and structure to a_2, hence $\langle q_1, a_1 \rangle$ will get a higher score than $\langle q_1, a_1^{'} \rangle$ using the kernel method; in contrast, this wouldn't be the case using a similarity score matching q_1 to a_1 resp. $a_1^{'}$ as both a_1 and $a_1^{'}$ contain the q_1 keyword "autism".

This intuitively explains why even a bag-of-words kernel adjusting its weights on question/answer pairs has a better chance to produce better results than a bag-of-words question/answer similarity (or a variation thereof as implemented by YourQA). This is experimentally proven in Sections 4.3 – 4.4. The above is even more true in the case of the tf*idf model implemented by the underlying document retrieval engine, as the latter similarity criterion is document-wide, as described in Section 4.4.

3.5. The YourQA corpora: WEB-QA and TREC-QA

In order to obtain our answer corpora, during the Document Retrieval phase, YourQA worked alternatively with two IR engines: Google[6], to retrieve Web documents, and Lucene[7], to retrieve news articles from the latest corpus released for the TREC competition, AQUAINT 6[8]. The two corpora are henceforth named WEB-QA and TREC-QA, respectively[9]. The WEB-QA corpus was especially interesting to test the abilities of a fully Web-based open domain QA system, a particularly challenging task. We also wanted to assess whether creating our relational data representations based on the use of "off-the-shelf" parsers and semantic role labelers (trained on "clean" data) on Web data would yield effective learning algorithms or not. However, the TREC-QA corpus was necessary to align with the methodology followed by traditional QA system evaluation drawn from IR on a closed corpus.

The answers returned by YourQA are in the form of sentences with relevant words or phrases highlighted and surrounded by their original passage. This choice is due to the fact that the system is intended to provide a context to the exact answer; moreover, our focus on non-factoids made it reasonable to provide answers in the form of sentences [36]. Each sentence of the top 20 paragraphs returned by YourQA was manually evaluated by two annotators based on whether or not it contained a correct answer to the corresponding question. The inter-annotator agreement was judged substantial (Cohen $\kappa = 0.635$).

To simplify the classification task, we isolated for each paragraph the sentence with the maximal judgment and labeled it as a positive instance if it answered the question, negative otherwise[10]. For instance, given the question 'What are invertebrates?', the sentence 'At least 99% of all animal species are invertebrates' was labeled -1 , while 'Invertebrates are animals without backbones' was labeled $+1$. The resulting WEB-QA corpus contains 1309 sentences, 416 of which are positive;

[6]`google.com`

[7]`lucene.apache.org`

[8]`trec.nist.gov/data/qa`

[9]Available at: `disi.unitn.it/~silviaq/resources.html`

[10]Positive judgments ranged from 3 to 5 to reflect increasing conciseness and correctness of the answers, while negative ones ranged from 1 to 2. In our experiments, these groups of judgments are further remapped to $+1$ resp. -1.

the TREC-QA corpus contains 2256 sentences, 261 of which are positive[11]. The difference in positive rate (31.8% and 11.6%, respectively) is due to the fact that finding an answer to a question is simpler on the Web than on the smaller TREC corpus.

4. Experiments

The aim of our experiments is twofold: on one hand, we demonstrate that our supervised approach applying kernels to pairs of questions and answers is effective for automatically learning their relation. On the other hand, we show that sequence, syntactic and shallow semantic tree kernels provide important linguistic information to describe the above-mentioned relations. As a general result, our models can successfully re-rank the output of a basic question answering system such as YourQA.

In more detail, we test the kernel functions elaborated for question and answer representation against the WEB-QA and TREC-QA corpora described in Section 3.5. We begin our illustration by discussing our experimental setup (Sec. 4.1). Then, we carry out a comparative analysis in terms of accuracy and efficiency of two different kernels for predicate argument structures: the Partial Tree Kernel (PTK) and the Shallow Semantic Tree Kernel (SSTK), respectively introduced in Sections 2.2.3 and 2.2.2. Next, we focus on the accuracy of different classifiers on both datasets in order to select the most promising combinations for complex QA (Sec. 4.3). Our results show that our POS$_{SK}$ jointly used with PAS$_{PTK}$ and STK highly improves on BOW. We finally discuss the impact of the above classifiers in re-ranking YourQA's initial results (Sec. 4.4).

4.1. Experimental Setup

To run our experiments, we implement the following functions:

- the BOW and POS linear kernels;
- the WSK and POS$_{SK}$ sequence kernels;
- the STK on syntactic parse trees, derived automatically via the Charniak parser [37];

[11]It can be noted that the number of instances in the WEB-QA and TREC-QA corpora do not amount to exactly 138 times 20 answer: indeed, this is due to the fact that not only the IR engine does not always find 20 relevant documents for the query, but also that the QA system does not always select as many as 20 answer paragraphs due to the low similarity score the latter may achieve with respect to the query.

- the SSTK and PTK on Predicate Argument Structures, derived automatically via the Semantic Role Labeling system described in [32].

Finally, we implement combinations of the above kernels in the SVM-light-TK toolkit[12], that supports the design of new functions in SVM-light [38].

Since answers often contain more than one PAS (see Figure 3(a)), we sum PTK (or SSTK) applied to all pairs $P_1 \times P_2$, where P_1 and P_2 are the set of PASs of the first two answers. More formally, let P_t and $P_{t'}$ be the sets of PASs extracted from text fragments t and t' by the PTK; the resulting kernel is

$$K_{\texttt{all}}(P_t, P_{t'}) = \sum_{p \in P_t} \sum_{p' \in P_{t'}} PTK(p, p').$$

Although different kernels can be used for questions and for answers, we use (and combine) the same sets of kernels on both questions and answers; the only exception are PAS$_{PTK}$ and PAS$_{SSTK}$, that are only evaluated on answers.

We train and test our classifiers and answer re-rankers on the two datasets described in Section 3.5. The accuracy of our classifiers is evaluated in terms of F1 score, whereas the QA system performance is measured in terms of Mean Reciprocal Rank (MRR). This is defined as: $MRR = \frac{1}{n}\sum_{i=1}^{n}\frac{1}{rank_i}$, where n is the number of questions in the corpus and $rank_i$ is the rank of the first correct answer to question i. We consider the top 5 available ranks returned by YourQA in MRR computation. Moreover, each reported value in our figures refers to the average over 5 different samples using five-fold cross-validation.

4.2. PTK vs SSTK: Performance and Efficiency

In a first set of experiments, we compare the performance of PTK with respect to SSTK for predicate argument structures. We compute the classification accuracy of SVMs by using either the PAS$_{SSTK}$ or PAS$_{PTK}$ data representations alone on both the WEB-QA and TREC-QA datasets. Figure 5 shows the obtained F1 (average on 5 folds) according to different values of the cost-factor parameter used for learning: higher values of the latter increase the cost of mistaking the positive examples, in turn increasing classifier Recall[13]. We note that while on WEB-QA the models are very close, PTK is slightly better than SSTK on TREC-QA. The fact that both classifiers achieve much higher F1 on WEB-QA is not surprising, as

[12]available at `dit.unitn.it/moschitti/`

[13]This parameter (`-j` option in SVM-light) multiplies the summation of the positive slack variables $\sum_i \xi_i^+$, where ξ_i is roughly the error in mistaking the example x_i. Since such summation is added to the objective functions of SVM optimization problem, an optimal solution tends to reduce the mistakes of positive examples.

this dataset contains many more correct answers (balanced classification problems are generally easier to solve).

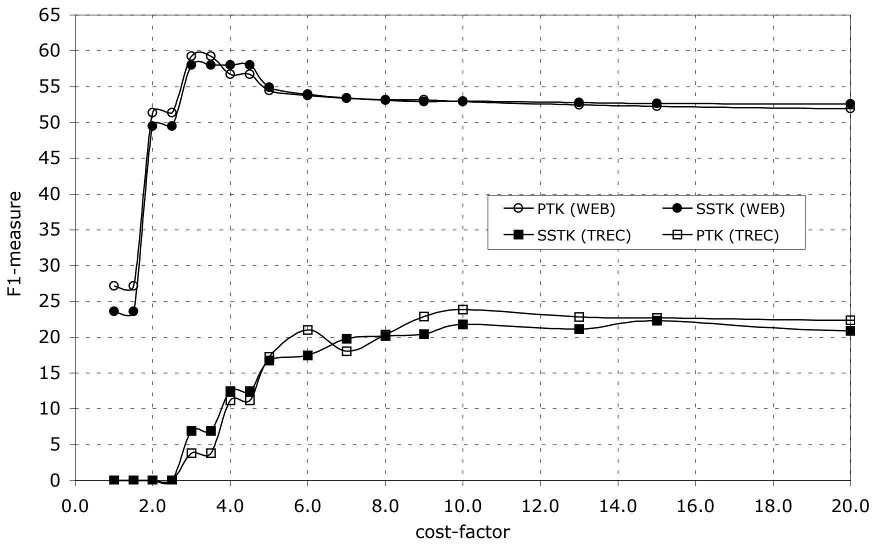


Figure 5: Impact of PAS$_{PTK}$ (PTK) and PAS$_{SSTK}$ (SSTK) on answer classification

Another interesting test concerns kernel efficiency. SSTK runs on large structures containing as many slots as the number of possible predicate argument types. This affects both memory occupancy and kernel computation speed. In contrast, PTK is able to process the same information with much smaller structures. To test the above characteristics, we divide the training data into 9 bins of increasing size (with a step of 200 instances between two contiguous bins) and we measure the training and test time[14] for each bin. Figure 6 shows that in both the training and test phases PTK is much faster than SSTK. In training, PTK is 40 times faster, making the experimentation of SVMs with large datasets feasible. This is an interesting result since for SSTK as well as for PTK we use the fast algorithms proposed in [25, 23], typically denoting a linear average running time.

4.3. Results for Question-Answer Classification

In these experiments, we test different kernels and some of their most promising combinations. Since the nature of the applied kernels strongly depends on the

[14]Processing time in seconds of a Mac-Book Pro 2.4 Ghz.

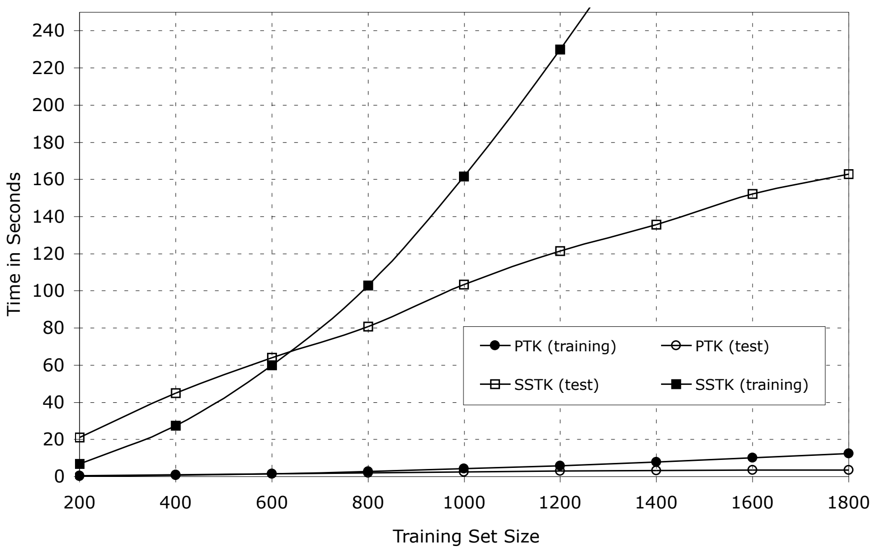


Figure 6: Efficiency of PTK and SSTK

data they operate on, we simplify our notation by only using the name of the representation instead of the more appropriate name combination (representation and kernel). In other words, we use BOW, POS and PT to indicate that a linear kernel is applied to bag-of-words and POS vectors and the syntactic tree kernel is applied to parse tree (PT). In the other notations, i.e. POS$_{SK}$, PAS$_{SSTK}$ and PAS$_{PTK}$, the subscript indicates the applied kernel: this suggests that SK is applied to POS sequences and that SSTK and PTK are applied to the PAS structures. The only exception is WSK, indicating the Word Sequence Kernel, i.e. a string kernel applied to word sequences.

To produce kernel combinations, we use the sum between kernels[15] since this yields the joint feature space of the individual kernels [22].

First, we compute the F1 of our answer classifiers for different values of the cost-factor parameter adjusting the ratio between Precision and Recall; this is in order to verify whether any difference between models is systematically observed regardless of the classifier parameters (Section 4.3.1). Furthermore, we examine the differences between models for a fixed value of the cost-factor parameter (estimated from a held out set) to measure any significant difference (Section 4.3.2).

[15]All additive kernels are normalized to have a similarity score between 0 and 1, i.e. $K'(X_1, X_2) = \frac{K(X_1,X_2)}{\sqrt{K(X_1,X_1) \times K(X_2,X_2)}}$.

Finally, to complete our analysis, we compute the Precision-Recall curves for a number of models on a fixed fold of our cross-validation splits (Section 4.3.3).

4.3.1. F1 curves

Figure 7 shows the F1-plots of several kernels[16] according to different values of the above-mentioned cost-factor parameter.

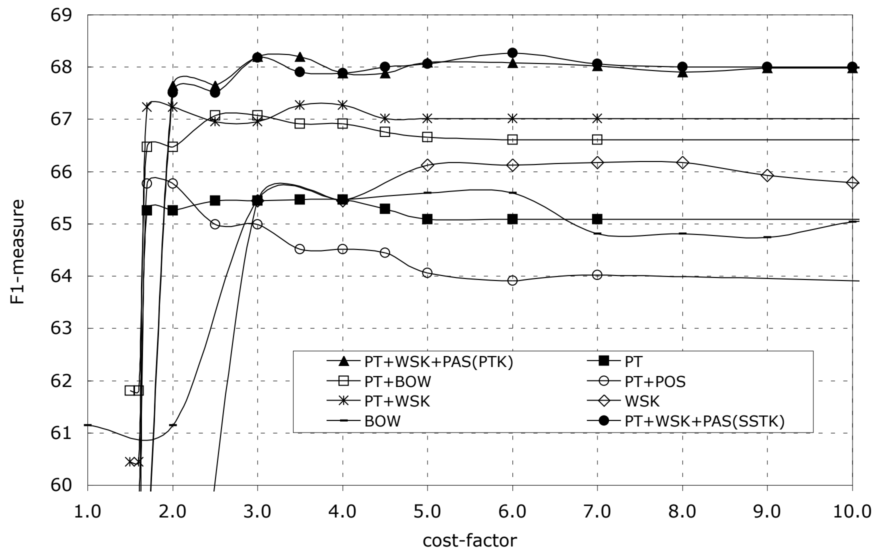


Figure 7: Impact of different feature sets on the WEB-QA dataset

First, we note that BOW achieves very high accuracy, comparable to the accuracy of PT; this is surprising when considering that at test time, instances of the training models (e.g. support vectors) are compared to different test examples since questions cannot be shared between training and test set (indeed, sharing questions between test and training sets would be an error from a machine learning viewpoint as we cannot expect new questions to be identical to those in the training set). Thus, we would expect answer wordings to be different and of low contribution to generalize rules for answer classification. However, error analysis reveals a number of common patterns in the answers due to typical Web page phrasings that indicate if a retrieved passage is an incorrect answer, e.g. `Learn more about X`. Although the ability to detect these patterns is beneficial for a QA system as

[16]In order to meet a trade-off between the readability of the plots and the representation of all interesting systems, we always give the priority to the top accurate systems.

it improves its overall accuracy, it is slightly misleading for the study that we are carrying out. This further motivates our experiments with the TREC-QA dataset, which is cleaner from a linguistic viewpoint and also more complex from a QA perspective as it contains fewer positive instances.

Figure 7 also shows that that the BOW+PT combination improves on both individual models; however, POS+PT produces a lower F1 than PT alone, indicating that POS does not provide useful information for this dataset. Furthermore, WSK improves on BOW and is further improved by WSK+PT, demonstrating that word sequences and parse trees are very relevant for this task. Finally, both PAS$_{SSTK}$ and PAS$_{PTK}$ improve on previous models, yielding the highest result (PT+WSK+ PAS$_{PTK}$). These findings are interesting as they suggest that the syntactic information provided by STK and the semantic information brought by WSK and PAS$_{PTK}$ (or even PAS$_{SSTK}$) considerably improves on BOW.

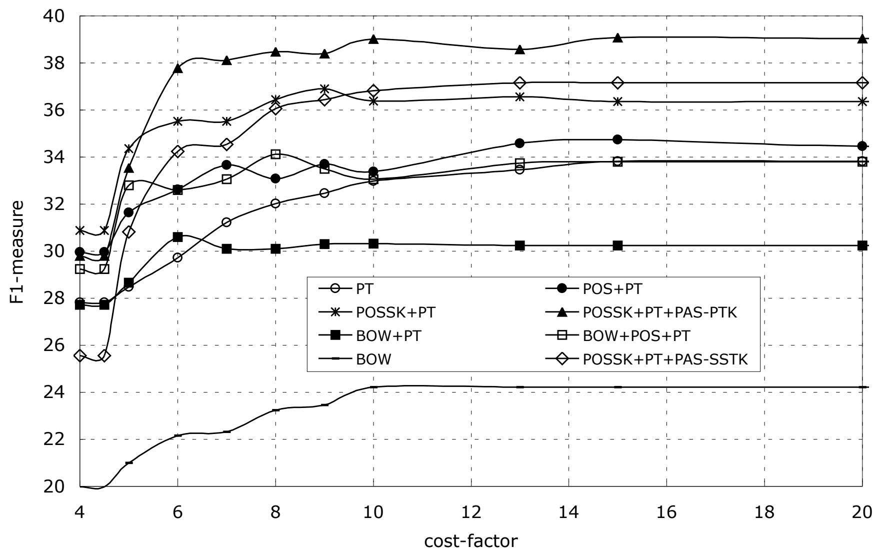


Figure 8: Impact of different feature sets on the TREC-QA dataset

In summary, our results for WEB-QA strengthen the expectation that BOW may be outperformed by structural information in the TREC-QA dataset, where the task is more complex and the data is less noisy. To this purpose, Figure 8 shows the plots of different classification models on the TREC-QA dataset. An initial glance suggests that the F1 of all models is much lower than for the WEB-QA dataset. Indeed, BOW shows the lowest accuracy and also the accuracy of its combination with PT is lower than the one of PT alone. Interestingly, this time POS seems helpful since its combination with PT improves on PT alone;

WEB Question/Answer Classification Corpus						
BOW	POS	POS$_{SK}$	WSK	PT	PAS$_{SSTK}$	PAS$_{PTK}$
65.3±2.9	56.8±0.8	62.5±2.3	65.7±6.0	65.1±3.9	52.9±1.7	50.8±1.2
	BOW+POS	BOW+PT	POS$_{SK}$+PT	WSK+PT	PT+PAS$_{SSTK}$ +WSK	PT+PAS$_{PTK}$ +WSK
	63.7±1.6	66.0±2.7	65.3±2.4	66.6±3.0	68.0±2.7	**68.2**±4.3

Table 1: F1 ± std. dev. of the question/answer classifier using several kernels on the WEB corpus

however, again, summing BOW to POS+PT produces a decrease. Moreover, SK is beneficial for exploiting POS information as POS$_{SK}$+PT improves on POS+PT, yet PAS adds further useful information as the best models are POS$_{SK}$+PT+PAS$_{PTK}$ and POS$_{SK}$+PT+PAS$_{SSTK}$.

In order to gain better numerical insights on our results, we provide further analysis in the following section, where we compare the accuracy of different models at a fixed cost-factor parameter.

4.3.2. Pointwise estimation and significance of results

The plots representing F1 versus the cost-factor parameter suggest that the value of such parameter maximizing F1 can be reliably estimated. Thus, for each model, we selected the minimum cost-factor associated with maximum F1 value on a held-out set[17]. This provides a single performance index for each system, that can be used to compare the five different models obtained via cross-validation using the paired $t-$test.

Table 1 reports the average F1 ± the standard deviation over 5 folds achieved by the different kernels on the WEB-QA corpus. When examining our results, we note that:

- BOW achieves very high accuracy on the WEB dataset, comparable to the one produced by PT, i.e. 65.3 vs 65.1;
- the BOW+PT combination reaches 66.0 accuracy, improving on both BOW and PT alone; however, BOW+POS produces a lower F1, i.e. 63.7, than PT+BOW, indicating that POS does not provide useful information for this dataset;
- WSK achieves 65.7 accuracy, thus improving on BOW; furthermore, WSK is enhanced by WSK+PT (66.6). This demonstrates that word sequences and

[17]It turned out that a value of 10 is roughly the best for any kernel.

TREC Question/Answer Classification Corpus						
BOW	POS	POS$_{SK}$	WSK	PT	PAS$_{SSTK}$	PAS$_{PTK}$
24.2±5.0	26.5±7.9	31.6±6.8	4.0±4.2	33.1±3.8	21.8±3.7	23.6±4.7
	BOW+POS	BOW+PT	POS$_{SK}$+PT	WSK+PT	PT+PAS$_{SSTK}$ +POS$_{SK}$	PT+PAS$_{PTK}$ +POS$_{SK}$
	31.9±7.8	30.2±5.3	36.4±9.3	23.7±3.9	36.2±7.1	**39.1**±6.9

Table 2: F1 ± std. dev. of the question/answer classifier using several kernels on the TREC-QA corpus

parse trees are very relevant for this task;

- finally, the highest performing combinations of features are PAS$_{SSTK}$ + WSK + BOW and PAS$_{PTK}$ + WSK + BOW, which reach 68.2 accuracy, further improving on the already high performance of BOW as a standalone (65.3).

Despite the observed improvement on BOW in terms of F1 averaged over five folds, none of the results achieved on the WEB-QA corpus have registered a sufficiently small p value to reach statistical significance in the $t-$test. Indeed, even the most performing combinations of syntactic and shallow semantic information, exhibiting an improvement up to 3 points in F1 on the BOW feature, are affected by the fact that the corpus contains a number of patterns indicating wrong answers that – as stated earlier – can easily be captured by word-level features only. For this reason, we now focus on the results obtained on the TREC-QA corpus, reported in Table 2. A comparative analysis with respect to Table 1 suggests that:

- as observed in the curves, we can immediately register that the F1 of all models is much lower than for the WEB-QA dataset, due to the presence of fewer positive instances in the training corpus;
- BOW denotes the lowest accuracy (a F1 of 24.2), and also the accuracy of its combination with PT (30.2) is lower than the accuracy of PT alone (33.1);
- Sequence Kernels are beneficial for exploiting POS information, as POS$_{SK}$ + PT reaches 36.4, improving on POS (99% significance, $p < 0.01$) and PT.
- Finally, Predicate Argument Structures add further information, as the best model is POS$_{SK}$ + PT + PAS$_{PTK}$. The latter improves on BOW from 24.2 to 39.1, i.e. by 63%; this result is 95% significant ($p < 0.05$).

Our first conclusion is that on this "cleaner" corpus, BOW does not prove very relevant to learn re-ranking functions from examples; while it is useful to establish the initial ranking by measuring the similarity between question and answer, it is almost irrelevant to capture typical rules that suggest whether a description is valid or not. Indeed, since there is no trace of test questions in the training set, their words as well as those of candidate answers are different. Most crucially, since a question with its answer set originates training pairs with a large word overlap, BOW tends to overfit.

Secondly, the results show that PT is important to detect typical description patterns, however POS sequences provide additional information since they are less sparse than tree fragments. Such patterns improve on the bag of POS-tags feature by about 4%. This is a relevant result considering that in standard text classification bigrams or trigrams are usually ineffective.

Third, although POS$_{SK}$+PT generates a very rich feature set, consisting of POS patterns provided by SK and tree fragments generated by STK, PAS$_{PTK}$ is still capable to improve on the POS$_{SK}$+PT combination by about 3% in F1. This suggests that shallow semantics can be very useful to detect whether an answer is well formed and related to a question.

Furthermore, error analysis reveals that PAS can provide patterns like:

- `A1(X) R-A1(that) rel(result) A1(Y)` and
- `A1(X) rel(characterize) A0(Y)`,

where `X` and `Y` need not necessarily be matched. Finally, the best model, POS$_{SK}$ + PT + PAS$_{PTK}$, improves on BOW by 63%; as mentioned above, this result is significant at 95% according to the $t-$test. This is strong evidence showing that complex natural language tasks require advanced linguistic information that should be exploited by powerful algorithms such as SVMs, and using effective feature engineering techniques such as kernel methods.

4.3.3. Precision/Recall Curves

To better study the benefit of the proposed linguistic structures, we also report Precision/Recall curves. Figure 9 displays the curves of some interesting kernels for one of the five folds of the WEB-QA dataset. As expected, BOW shows the lowest curves; moreover, WSK, able to exploit n-grams (with gaps), produces very high curves when summed to PT. In general, all kernel combinations tend to achieve only slightly higher results than BOW. Again, the cause is the high contribution of BOW, which prevents other models from clearly emerging.

The results on TREC-QA, reported in Figure 10 (for one of the five dataset folds), are more interesting. Here, the contribution of BOW remains very low and thus the difference in accuracy with the other linguistic models is more evident. In

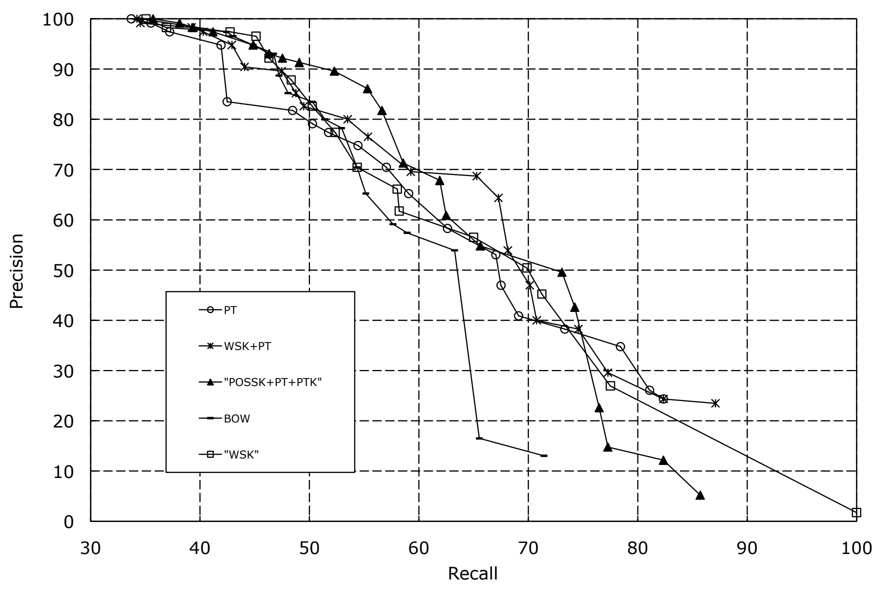


Figure 9: Precision/Recall curves of some kernel combinations over one fold of the WEB dataset.

particular, POS$_{SK}$ +PT+PAS$_{PTK}$, that encodes the most advanced syntactic and semantic information, shows a very high curve outperforming all the others.

In summary, the Precision/Recall figures further corroborate our observations concerning classification accuracy and the role of structural data representations in complex Question Answering.

4.4. Answer Re-ranking

The obvious application of an answer classifier consists in re-ranking the initial answers extracted by a baseline Question Answering system: indeed, re-ranking can be regarded as the real testbed of any QA classifier. We have been running a number of re-ranking tests by taking the top classifiers obtained on the WEB-QA and TREC-QA corpus, respectively, and using their binary output to rearrange the answers returned by the YourQA system. Our re-ranking algorithm starts from the top of the answer list and leaves the corresponding answer's rank unchanged if the answer is classified as correct by the binary classifier; otherwise, the rank is pushed down, until a lower ranked incorrect answer is found.

In order to compare the above re-ranking strategy to a reasonable baseline, we first measure the Q/A classification ability of YourQA and its underlying IR engine by examining the F1 and MRR of the answers corresponding to the top five documents returned by the IR engine and the top five answers as ranked by

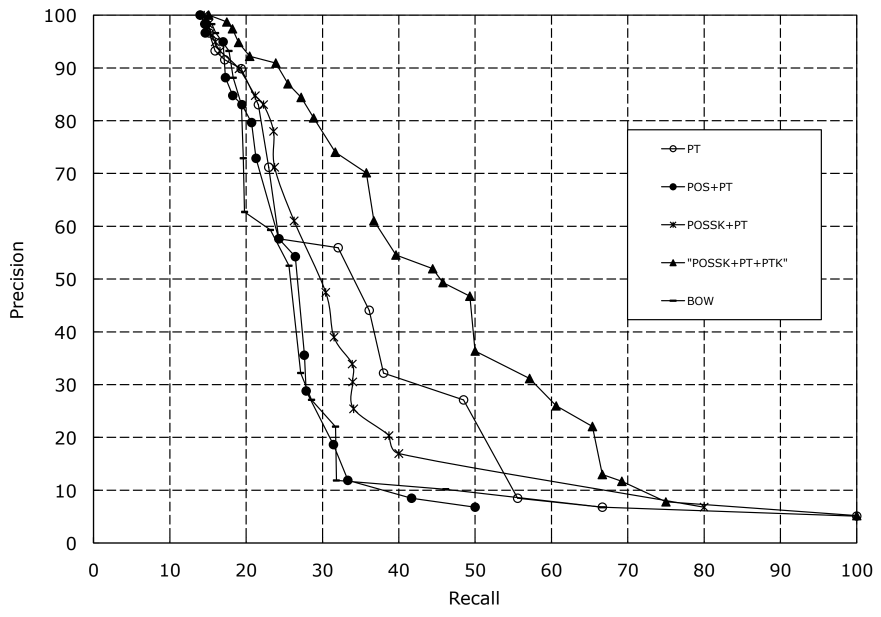


Figure 10: Precision/Recall curves of some kernel combinations over one fold of the TREC dataset.

YourQA. In particular, the classification accuracy of the above systems is computed by labeling each of the five retrieved answers as correct.

Our results, reported in the *Classifier F1* row of Table 3, show that the accuracy of YourQA is slightly higher than the IR accuracy. Indeed, in the WEB-QA dataset, the IR engine (Google) is outperformed by YourQA since its ranks are based on whole documents, not on single passages. Hence, Google may rank a document containing several sparsely distributed question words higher than documents with several words concentrated in one passage, which are more interesting. This is reflected by the fact that, as visible in the *MRR* rows of Table 3, the ranking deriving naturally from YourQA's answer extractor improves on the original IR ranking by gaining 7 points in MRR on the WEB-QA corpus (56.2 vs 49.0). Furthermore, on the TREC-QA corpus, YourQA almost doubles the IR engine (Lucene) MRR, taking it from 16.2 to 30.3. This result can be explained by the complexity of the answer extraction task, as fewer documents are relevant to the question in the TREC-QA corpus and for such documents, the emphasis on Precision provided by the YourQA answer extractor yields an increase in answer relevance.

By now considering the performance of our best Q/A classifiers ("Re-ranker (Best)" column), learned in Section 4, we can observe that the latter greatly outperform the F1 of YourQA, i.e. 68.6 vs 36.8 (on WEB-QA) and 39.0 vs 22.9 (on

		IR engine	YourQA	Re-ranker (BOW)	Re-ranker (Best)
WEB-QA	*Classifier F1*	35.9±4.0	36.8±3.6	65.3±2.9	68.6±2.3
	MRR	49.0±3.8	56.2±3.2	77.4±2.7	81.1±2.1
TREC-QA	*Classifier F1*	21.3±1.0	22.9±1.5	24.2±3.1	39.1±6.9
	MRR	16.2±3.4	30.3±8.9	32.8±7.7	34.2±10.6

Table 3: Classifier F1 and MRR@5 (± std. dev.) of the IR engine (Google resp. Lucene), YourQA, the BOW re-ranker and the best re-ranker on the WEB-QA resp. TREC-QA datasets

TREC-QA). This suggests that more information is exploited by the re-ranking classifier. Indeed, when the re-ranking algorithm is applied, YourQA achieves an MRR of 81.1%, i.e. a 45% improvement, on WEB-QA. On the TREC-QA dataset, the IR engine is also outperformed by YourQA and the re-ranker produces a further improvement by about 13%. Such a smaller difference compared to the Web re-ranker depends on the lower classification accuracy of the re-ranker, due in turn to the higher complexity of the TREC dataset.

It may be noted that having proven that re-ranking is effective in improving QA systems does not imply that structural features are useful. To show this, we need to compare a re-ranker based on BOW against those based on linguistic structures. The BOW result is reported in Table 3 ("Re-ranker (BOW)" column). It shows that the F1 (*Classifier F1* row) of the BOW classifier is lower than that of the "best" classifier in both the WEB-QA and TREC-QA cases: on the TREC-QA dataset, the F1 of the BOW classifier is 24.2±3.1 while it is 39.1±6.9 for the "best" classifier. This translates into a slightly lower MRR (*MRR* rows) obtained with the BOW re-ranker in comparison to the re-ranker using structural features. We can therefore conclude that not only the simple fact of using an answer re-ranker is beneficial in terms of answer accuracy, but also that tree kernels applied on structural features are yet more effective than simpler bag-of-word features in identifying correct answers to complex questions.

4.4.1. Discussion

It should be noted that re-ranking approaches more elaborate than ours can be applied, such as a "true" re-ranking based on pairs of instances [13, 39]. Although this should in principle produce better results, it also has the drawback of doubling the number of structures required to represent such pairs of pairs. Since we have carried out a study on about twenty-five different kernels, we preferred to keep our models simpler.

A second option to improve our methods while keeping the model complexity low would be the use of the classifier score to establish the relative ranking between different answers classified as correct. One problem with this approach is that SVM score, i.e. the margin of the classifying point, is not a probability. This means that the relative distance between two scores is not a good indicator of the reliability of a classification over the other. For example, for a given question, a difference of 0.5 between the scores of two candidate answers may not indicate a high reliability for the higher-scored classification, whereas for another question a difference of 0.005 may indicate very high reliability for the higher-scored classification.

This problem is very critical in our context since the rankers (i.e. the classifiers) may reach low F1, e.g. about 40% for TREC dataset. This causes both a high variability and a limited reliability of results. Under such conditions, re-ranking should be carefully carried out. The *status quo*, i.e. the initial ranking provided by the basic QA system, should only be changed when there is strong indication of a misplaced answer (i.e. incorrect answer), as, for example, can be an exceeded classification threshold. Thus, we argue that our heuristic approach of pushing answers down in the ranking when they are labeled as incorrect is more conservative and has a higher chance to improve the basic QA.

A possible alternative would be the conversion of SVM scores into actual probabilities, however once again these would not be reliable due to the scarcity of available training data. A more effective solution would be the adoption of meta-classifiers to decide whether the current scores/probabilities within a given context suggest a valuable change in the position of the target answer. The above approaches are interesting research directions, albeit beyond the aim of this paper.

5. Related Work

Early work on the use of syntax and semantics in Information Retrieval was carried out in [40, 41, 42] and in [43, 44]. The results showed that the use of advanced linguistic information was not effective for document retrieval. In contrast, Question Answering work shows that semantics and syntax are essential to retrieve punctual answers, e.g [45, 46, 47]. However, successful approaches in TREC-style systems were based on several interconnected modules exploiting complex heuristics and fine tuning. The effective combination of such modules strongly depended on manual setting, which was often not disclosed.

In our study, we avoid this problem by focusing on a single phase of Question Answering, namely answer extraction. The latter can be seen as a typical text categorization task, i.e. the classification of pairs of text fragments constituted by question and answer. Since some types of questions can be solved with relatively simple representations, i.e. without the use of syntactic and semantic structures, we focus on the more complex task of processing description (often called definition) questions [48, 2, 6, 28, 9, 49].

In [2], answer ranks were computed based on the probabilities of bigram language models generating candidate answers; language modelling was also applied to definitional QA in [34] to learn soft pattern models based on bigrams. Other related work, such as [7, 8], was also very tied to bag-of-words features.

Our approach is different from the above in that we attempt to capture structural information, which has proven to be very effective in our experiments, yielding a very high MRR. In contrast to [11], our approach does not require the creation of ad-hoc joint question-answer representations. In particular, we compare to previous work [6, 28, 9, 49] using predicate argument structures for re-ranking candidate answer lists and reporting significant improvement. To our knowledge, our work in [9] was the first to use kernel methods for answer re-ranking. We used a syntactic tree kernel and a shallow semantic tree kernel based on predicate argument structures for the design of answer re-rankers. However, as we only experimented with a Question Answering corpus derived from Web documents and the reported improvement, although significant, did not justify the adoption of computationally expensive approaches like SVMs and kernel methods. In this paper, developing with respect to subsequent work [10], we have experimented with many more kernel types and with both Web and TREC documents and we could show that the potential improvement reachable by our approach is much higher (about 63% over BOW). Moreover, we have designed a faster kernel for the processing of semantic information.

In summary, the main property of our approach with respect to previous work adopting syntactic and semantic structures is that we can define the latter without requiring a thorough manual linguistic analysis. We do not carry out feature engineering since we simply let kernel functions generate a large feature set (tree fragments or substrings) that represents semantic/syntactic information effectively. The feasibility of this approach is due to the SVM theory which makes the learning algorithm robust to many irrelevant features (often produced by NLP errors).

6. Conclusions

We have approached answer selection, the most complex phase of a QA system. To solve this task, typical approaches use unsupervised methods that involve computing the similarity between query and answer in terms of lexical, syntactic, semantic or logic representations. In contrast, we study supervised discriminative models that learn to select (rank) answers from examples of question and answer pairs, where the representation of the pair is implicitly provided by kernel combinations applied to each of its components. To reduce the burden of manual annotation of such pairs, we use kernel functions applied to syntactic/semantic structures as powerful generalization methods. The combination of the generalization properties of such structures with the exponential space of substructures generated by kernel functions provides an advanced form of back-off model in a discriminative setting, that we have proved to be effective.

In particular, we use POS-tag sequences, syntactic parse trees and predicate argument structures (PASs) along with sequence kernels and syntactic and shallow semantic tree kernels. Extensive experiments on two different corpora that we have collected and made available show that: (i) on TREC data, the improvement on the bag-of-words feature (BOW) is very high (about 63% in F1 score) confirming that our kernels/structures provide the right level of generalization; (ii) the Partial Tree Kernel (PTK) for processing PASs is efficient and effective and can be practically used to design answer re-ranking models; and (iii) our best question/answer classifier, used as a re-ranker, significantly improves the QA system MRR, confirming its promising applicability.

Regarding PAS, deeper analysis reveals that PTK can learn definition patterns such as: `A1(X) R-A1(that) rel(result) A1(Y)` (e.g. ‘German measles, *that result* in red marks on the skin, are a common disease’) and:
`A1(X) rel(characterize) A0(Y)` (e.g. ‘Autism is *characterized* by the inability to relate to other people’).

We believe that these are strong arguments in favor of the exploitation of advanced linguistic information by using powerful discriminative models such as SVMs and effective feature engineering techniques such as kernel methods in challenging natural language tasks.

In the future, we would like to experiment with our model on larger and different datasets and compare with (or better re-rank) more advanced QA systems. Moreover, an interesting open problem is how to jointly exploit the set of PASs of a sentence/paragraph in a more effective and compositional semantics-driven approach.

References

[1] J. Allan, Natural Language Processing for Information Retrieval, in: NAACL/ANLP (tutorial notes), 2000.

[2] Y. Chen, M. Zhou, S. Wang, Reranking answers from definitional QA using language models, in: Proceedings of ACL, 2006.

[3] K. Collins-Thompson, J. Callan, E. Terra, C. L. Clarke, The effect of document retrieval quality on factoid QA performance, in: Proceedings of SIGIR, 2004.

[4] H. Yang, T. Chua, QUALIFIER: Question Answering by Lexical Fabric and External Resources, in: Proceedings of EACL, 363–370, 2003.

[5] E. Hovy, U. Hermjakob, C. Lin, The Use of External Knowledge of Factoid QA, in: Proceedings of TREC, Gaithersburg, MD, U.S.A., 2001.

[6] D. Shen, M. Lapata, Using Semantic Roles to Improve Question Answering, in: Proceedings of EMNLP-CoNLL, 2007.

[7] Y. Sasaki, Question Answering as Question-Biased Term Extraction: A New Approach toward Multilingual QA, in: Proceedings of ACL, 215–222, 2005.

[8] J. Suzuki, Y. Sasaki, E. Maeda, SVM Answer Selection for Open-Domain Question Answering, in: Proceedings of Coling, 974–980, 2002.

[9] A. Moschitti, S. Quarteroni, R. Basili, S. Manandhar, Exploiting Syntactic and Shallow Semantic Kernels for Question/Answer Classification, in: Proceedings of ACL, Prague, Czech Republic, 2007.

[10] A. Moschitti, S. Quarteroni, Kernels on Linguistic Structures for Answer Extraction, in: Proceedings of ACL, Columbus, OH, USA, 2008.

[11] A. Echihabi, D. Marcu, A Noisy-Channel Approach to Question Answering, in: Proceedings of ACL, 2003.

[12] E. M. Voorhees, Overview of the TREC 2001 Question Answering Track, in: Proceedings of TREC, 2001.

[13] M. Collins, N. Duffy, New Ranking Algorithms for Parsing and Tagging: Kernels over Discrete Structures, and the Voted Perceptron, in: Proceedings of ACL, 2002.

[14] T. Kudo, Y. Matsumoto, Fast Methods for Kernel-Based Text Analysis, in: E. Hinrichs, D. Roth (Eds.), Proceedings of ACL, 24–31, 2003.

[15] C. Cumby, D. Roth, Kernel Methods for Relational Learning, in: Proceedings of ICML, Washington, DC, USA, 107–114, 2003.

[16] A. Culotta, J. Sorensen, Dependency Tree Kernels for Relation Extraction, in: Proceedings of ACL04, Barcelona, Spain, 423–429, 2004.

[17] T. Kudo, J. Suzuki, H. Isozaki, Boosting-based Parse Reranking with Subtree Features, in: Proceedings of ACL, Ann Arbor, MI, USA, 2005.

[18] K. Toutanova, P. Markova, C. Manning, The Leaf Path Projection View of Parse Trees: Exploring String Kernels for HPSG Parse Selection, in: Proceedings of EMNLP, Barcelona, Spain, 2004.

[19] J. Kazama, K. Torisawa, Speeding up Training with Tree Kernels for Node Relation Labeling, in: Proceedings of EMNLP, Toronto, Canada, 137–144, 2005.

[20] M. Zhang, J. Zhang, J. Su, Exploring Syntactic Features for Relation Extraction using a Convolution Tree Kernel, in: Proceedings of NAACL, New York City, USA, 288–295, 2006.

[21] V. Vapnik, The Nature of Statistical Learning Theory, Springer, 1995.

[22] J. Shawe-Taylor, N. Cristianini, Kernel Methods for Pattern Analysis, Cambridge University Press, 2004.

[23] A. Moschitti, Efficient Convolution Kernels for Dependency and Constituent Syntactic Trees, in: Proceedings of ECML, 2006.

[24] N. Cancedda, E. Gaussier, C. Goutte, J. M. Renders, Word sequence kernels, J. Mach. Learn. Res. 3 (2003) 1059–1082, ISSN 1533-7928.

[25] A. Moschitti, Making Tree Kernels Practical for Natural Language Learning, in: Proceedings of EACL2006, 2006.

[26] Y. Wu, R. Zhang, X. Hu, H. Kashioka, Learning Unsupervised SVM Classifier for Answer Selection in Web Question Answering, in: Proceedings of EMNLP-CoNLL, 2007.

[27] K. Deschacht, M.-F. Moens, Using the Latent Words Language Model for Semi-Supervised Semantic Role Labeling, in: Proceedings of EMNLP, 2009.

[28] M. Bilotti, P. Ogilvie, J. Callan, E. Nyberg, Structured Retrieval for Question Answering, in: Proceedings of ACM SIGIR, 2007.

[29] C. R. Johnson, C. J. Fillmore, The FrameNet tagset for frame-semantic and syntactic coding of predicate-argument structures, in: Proceedings of ANLP-NAACL, 56–62, 2000.

[30] P. Kingsbury, M. Palmer, From Treebank to PropBank, in: Proceedings of LREC, 2002.

[31] X. Carreras, L. Màrquez, Introduction to the CoNLL Shared Task: SRL, in: Proceedings of CoNLL, 2005.

[32] A. Moschitti, B. Coppola, A. Giuglea, R. Basili, Hierarchical Semantic Role Labeling, in: Proceedings of the CoNLL 2005 shared task, 2005.

[33] H. Kazawa, H. Isozaki, E. Maeda, NTT Question Answering system in TREC 2001, in: Proceedings of TREC, 2001.

[34] H. Cui, M. Kan, T. Chua, Generic soft pattern models for definitional QA, in: Proceedings of SIGIR, ACM, Salvador, Brazil, 2005.

[35] X. Li, D. Roth, Learning Question Classifiers, in: Proceedings of ACL, 2002.

[36] S. Quarteroni, S. Manandhar, Designing an Interactive Open Domain Question Answering System, Natural Language Engineering 15 (1) (2009) 73–95.

[37] E. Charniak, A Maximum-Entropy-Inspired Parser, in: Proceedings of NAACL, 2000.

[38] T. Joachims, Making large-Scale SVM Learning Practical, in: B. Schölkopf, C. Burges, A. Smola (Eds.), Advances in Kernel Methods, 1999.

[39] L. Shen, A. K. Joshi, An SVM-based voting algorithm with application to parse reranking, in: Proceedings of CoNLL HLT-NAACL 2003, 9–16, 2003.

[40] E. M. Voorhees, Using WordNet to Disambiguate Word Senses for Text Retrieval, in: R. Korfhage, E. M. Rasmussen, P. Willett (Eds.), Proceedings of ACM-SIGIR, ACM, ISBN 0-89791-605-0, 171–180, 1993.

[41] E. M. Voorhees, Query Expansion Using Lexical-Semantic Relations, in: W. B. Croft, C. J. van Rijsbergen (Eds.), Proceedings of ACM-SIGIR, ACM/Springer, ISBN 3-540-19889-X, 61–69, 1994.

[42] A. F. Smeaton, Using NLP or NLP resources for information retrieval tasks, in: T. Strzalkowski (Ed.), Natural language information retrieval, Kluwer Academic Publishers, Dordrecht, NL, 99–111, 1999.

[43] T. Strzalkowski, G. C. Stein, G. B. Wise, J. P. Carballo, P. Tapanainen, T. Jarvinen, A. Voutilainen, J. Karlgren, Natural Language Information Retrieval: TREC-7 Report, in: Proceedings of TREC, 164–173, 1998.

[44] T. Strzalkowski, J. P. Carballo, J. Karlgren, A. H. P. Tapanainen, T. Jarvinen, Natural Language Information Retrieval: TREC-8 Report, in: Proceedings of TREC, 1999.

[45] A. Hickl, J. Williams, J. Bensley, K. Roberts, Y. Shi, B. Rink, Question Answering with LCC CHAUCER at TREC 2006, in: Proceedings of TREC, 2006.

[46] E. M. Voorhees, Overview of the TREC 2004 question answering track, in: Proceedings of TREC 2004, 2004.

[47] S. Small, T. Strzalkowski, T. Liu, S. Ryan, R. Salkin, N. Shimizu, P. Kantor, D. Kelly, N. Wacholder, HITIQA: Towards Analytical Question Answering, in: Proceedings of COLING, 2004.

[48] S. Blair-Goldensohn, K. R. McKeown, A. H. Schlaikjer, Answering Definitional Questions: A Hybrid Approach, AAAI Press, 2004.

[49] M. Surdeanu, M. Ciaramita, H. Zaragoza, Learning to Rank Answers on Large Online QA Collections, in: Proceedings of ACL-HLT, Columbus, Ohio, 2008.