Assessing the Impact of Syntactic and Semantic Structures for Answer Passages Reranking

Kateryna Tymoshenko
University of Trento
Povo (Trento), Italy
k.tymoshenko@gmail.com

Alessandro Moschitti*
Qatar Computing Research Institute
Doha, Qatar
amoschitti@qf.org.qa

ABSTRACT

In this paper, we extensively study the use of syntactic and semantic structures obtained with shallow and deeper syntactic parsers in the answer passage reranking task. We propose several dependency-based structures enriched with Linked Open Data (LD) knowledge for representing pairs of questions and answer passages. We use such tree structures in learning to rank (L2R) algorithms based on tree kernel. The latter can represent questions and passages in a tree fragment space, where each substructure represents a powerful syntactic/semantic feature. Additionally since we define links between structures, tree kernels also generate relational features spanning question and passage structures. We derive very important findings, which can be useful to build state-of-the-art systems: (i) full syntactic dependencies can outperform shallow models also using external knowledge and (ii) the semantic information should be derived by effective and high-coverage resources, e.g., LD, and incorporated in syntactic structures to be effective. We demonstrate our findings by carrying out an extensive comparative experimentation on two different TREC QA corpora and one community question answer dataset, namely Answerbag. Our comparative analysis on well-defined answer selection benchmarks consistently demonstrates that our structural semantic models largely outperform the state of the art in passage reranking.

Categories and Subject Descriptors

I.2.7 [**Artificial Intelligence**]: Natural Language Processing—*Text analysis; Language parsing and understanding*; H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval—*Retrieval models*

General Terms

Algorithms, Experimentation

Keywords

Question Answering; Learning to Rank; Kernel Methods; Structural Kernels; Linked Data

*Professor at University of Trento, DISI.

CIKM'15, October 19–23, 2015, Melbourne, VIC, Australia.

ACM 978-1-4503-3794-6/15/10 ...$15.00.
DOI: http://dx.doi.org/10.1145/2806416.2806490 .

1. INTRODUCTION

Previous work has shown that advanced natural language processing can positively impact the accuracy of Question Answering (QA) systems. As shown by the experience in the TREC QA task, e.g., [38], the selection of the right passage expressing the answer requires to consider the relation between the question and the passage text. In other words, it is not enough measuring the similarity between question and passage, but it is also important analyzing how some concepts in the questions, e.g., constituting the focus, are structured along with other concepts in the answer passage. For instance, in the following question/answer passage (Q/AP) pair[1]:

Q: *What sports stadium has been billed as "the eighth wonder of the world"?*

AP: *The Titans used to be called the Oilers and played in the dilapidated Astrodome; if this was the eighth wonder of the world, we live on a shabby little planet indeed.*

the question focus, *sports stadium* is linked to the property *the eighth wonder of the world*. Since in the AP the property above is also stated for *Astrodome*, we infer that the latter is the correct answer. The inference is about finding two related statements in Q and in AP regarding the Q focus, where the entities in the statements may not necessarily match. Automatically selecting the right properties to be considered is a difficult task that must exploit the relationship between Q and AP. For example, the property *Titans used to be called the Oilers* can be important but it is not useful for answering the question. It follows that the relation between Q and AP for property selection is necessary.

Since manually selecting properties, i.e., generating rules for any pairs of Q and AP is rather difficult or even impossible, automatic feature engineering approaches based on kernel methods, e.g., [31], have been developed, where syntactic and semantic tree representations of the Q/AP pairs are used in kernel-based L2R algorithms, e.g., SVMs [33]. The role of kernels was to implicitly generate syntactic patterns (i.e., tree fragments) to be used as features in SVMs.

However, this approach would not be able to solve the example above since, to answer the question, *Astrodome* has to be identified as a stadium. This is essential to derive that the focus of Q, *stadium*, and *Astrodome* in AP both own the same property, i.e., to be "*the eighth wonder of the world*". Without this link many incorrect entities may be selected as many entities enjoy the property above[2].

The solution (provided in [13]) for solving this case is the use of semantic resources: the lexical answer type (LAT) of the ques-

[1]This example from TREC QA corpus will be our a running example for the rest of the paper.

[2]For example see `http://en.wikipedia.org/wiki/Eighth_Wonder_of_the_World#Things_labeled_as_the_Eighth_Wonder_of_the_World`

tions, e.g., as provided by Wikipedia[3] category, can be matched against the one of the answers for improving candidate selection, e.g., for *Astrodome* the LAT is *stadium*. More in general, semantic resources help to reduce data sparseness and thus they enable matches between question and answer passages. However, finding the focus word and its category may be difficult and error prone and most importantly does not allow for solving non-factoid questions.

In this paper, we carried out an extensive study on the use of syntactic and semantic structures enriched with LD semantics for answer passage reranking. To make the feature design step easier, we adopt L2R models based on SVMs and structural kernels, which provide SVMs with structural patterns, automatically generated from Q/AP syntactic structures. The latter are enriched with relational semantic knowledge of LD by adding links between the entities in the Q and AP. In particular, we followed the steps below:

First, we design a representation for the Q/AP pair by engineering a pair of shallow syntactic trees connected with relational nodes (i.e., those matching the same words in the question and in the answer passages). This approach capitalizes on our successful models proposed in [31, 32, 29, 37] but we also explore deeper linguistic structure such as dependency trees.

Secondly, we use YAGO [16], DBpedia [4] and WordNet [12] to match constituents from QA pairs and use their generalizations in our semantic structures. Following our previous work in [37], we employ word sense disambiguation to match the right entities in YAGO and DBpedia, and consider all senses of an ambiguous word from WordNet. For example, our system automatically derives that *Astrodome* is a stadium and thus such concept is connected to the question structure: *stadium has been billed as "the eighth wonder of the world"*, through the word *stadium*. This way, we obtain connected structures of pairs of texts, which potentially contain patterns useful for capturing the relatedness of question and answer passage.

Thirdly, we apply structural kernels to the above structures by exploiting SVMs for automatically learning classification and ranking functions. In particular, we apply the Partial Tree Kernel (PTK) [24], which can generate the richest space of tree fragments.

Next, we experiment with three different corpora, (i) the standard TREC QA corpus for passage reranking, (ii) a QA benchmark built for testing sentence reranking [42], and (iii) a community QA dataset based on Answerbag[4]. We tested several models combining (i) traditional feature vectors, (ii) automatic semantic labels derived by statistical classifiers, e.g., question classifiers, and (iii) relational structures enriched with LD relations. The results show that our methods greatly improve on strong IR baseline, e.g., BM25, up to 101%, and on state-of-the-art reranking models, up to 16.0% (relative improvement), e.g., in MAP.

In the remainder of this paper, Sec. 2 reports on related work, Sec. 3 describes our proposed classification and reranking framework, Sec. 4 illustrates our basic representation approach. Sec. 5 describes our new algorithms to carry out semantic matching using LD, Sec. 6 shows how we use LD matches for defining relational structures, Sec. 7 presents our learning to rank models based on tree kernels, Sec. 8 illustrates our experiments and finally, Sec. 9 derives the conclusions.

2. RELATED WORK

A referring work for our research is the IBM Watson system [13] (hereafter referred as Watson). This is an advanced QA pipeline based on deep linguistic processing and semantic resources. Watson uses deep syntactic parsing components and a predicate-argument structure (PAS) builder [22]. Additionally, it uses several semantic resources, e.g., Wikipedia and PRISMATIC [11] combined in a machine learning-based reranker. The Watson system is very accurate and effective but it requires to hand-craft rules, which is typically very costly. Our approach instead can automatically build syntactic/semantic patterns also exploiting LD, which can be used both for matching and generalizing words.

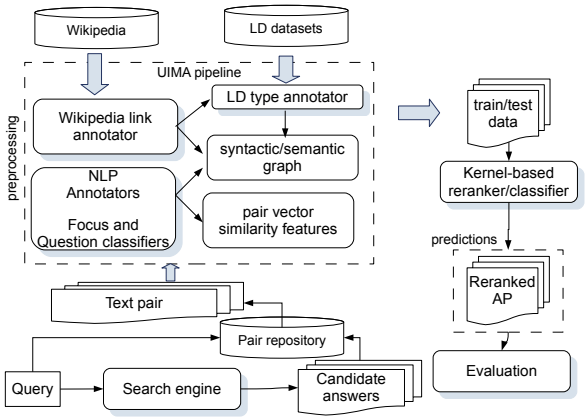

Figure 1: Kernel-based pair classification/reranking framework

Our baseline model is an updated version of our previous system developed in [31]. However, as pointed out in the introduction, such model does not encode dependency structure and the semantic information from LD. Moreover, our previous several attempts of using semantic information, e.g., Latent Dirichlet Allocation, WordNet, Latent Semantic Analysis, failed to improve the structural model. In contrast, we show that our LD approach can effectively encode knowledge improving on passage reranking.

More traditional work in QA using semantics and syntax can be observed in [15, 35]. However, the complexity of the method in [15] along with an obscure fine manual tuning, have made adaption or just replication of such systems rather complex. Recent studies on passage reranking, exploiting structural information, were carried out in [19], whereas other methods explored soft matching (i.e., lexical similarity) based on NE types [1]. [28, 17] applied question and answer classifiers for passage reranking. In this context, several approaches focused on reranking the answers to definition/description questions, e.g., [34, 36].

Next, the models developed in [2, 3] demonstrate that linguistic structures improve QA but the proposed approaches again are based on handcrafted features and rules. In contrast, our method is based on automatic feature engineering, resulting rather adaptable to different application domains.

Regarding answer sentence rerankers, [42] proposed a probabilistic quasi-synchronous grammar, inspired by machine translation, which allows to model Q/AP relations by means of syntactic transformations. [41] designed a probabilistic model to learn tree-edit operations on dependency parse trees. [14] employed a computationally expensive tree kernel-based heuristic to identify tree edit sequences which could serve as good features for a logistic regression model. [43] further improved the [14] approach by proposing a faster dynamic-programming based algorithm for feature extraction and extending the feature set with WordNet features. [45] proposed a model which, in addition to syntax, incorporates features based on rich lexical semantic knowledge, including synonymy, antonymy, hypernymy and semantic similarity, obtained from a number of external systems and resources. In our work, we encode semantic knowledge directly into syntactic tree representations of Q/AP and use PTK to learn the syntactico-semantic

[3] http://www.wikipedia.org

[4] http://www.answerbag.com/

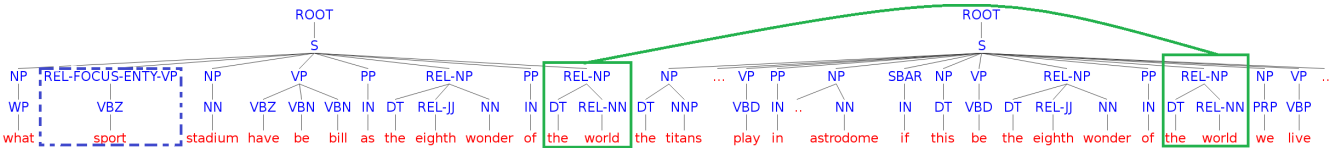

Figure 2: Shallow chunk-based tree (CH) for the Q and AP of the running example

patterns. More recently, the answer selection task was tackled with deep learning, e.g. convolutional deep neural networks [30] and stacked bidirectional Long Short-Term Memory model [40].

3. PASSAGE RERANKING FRAMEWORK

Our framework uses the relations between a question (Q) and its answer passage (AP) to rerank passages. The basic schema is displayed in Figure 1: given a Q we submit it as a query to a search engine, which retrieves a set of candidate AP. Each Q/AP text pair is processed by an NLP pipeline which performs basic tokenization, sentence splitting, lemmatization, stopword removal. Various NLP components, embedded in the pipeline as UIMA[5] annotators, perform more involved linguistic analysis, e.g., part-of-speech (POS) tagging, chunking, Named Entity (NE) recognition, constituency and dependency parsing, etc.

The Q/AP text pair is processed by a Wikipedia link annotator. It automatically recognizes n-grams in plain text, which may be linked to Wikipedia and disambiguates them to Wikipedia URLs. The input text is supposed to be short thus we concatenate the pair members together to provide a larger disambiguation context to the annotator. These annotations are then used to produce computational structures (see Sec. 4) input to pair classifiers. The semantics of such relational structures can be further enriched by adding links between the constituents of the two pieces of text. The relational links can be generated by: (i) matching lemmas as in [31]; (ii) matching the constituent types based on LD as in [37]; (iii) matching the question focus type derived by the question classifiers with the type of the target NE as in [31, 32, 29]. The resulting pairs of trees connected by semantic links are then used to train a kernel-based classifier Sec. 7.

4. TREES FOR Q/AP PAIRS

In our study, we design and compare several syntactic and semantic structural representations of pairs of short texts, including dependency- and shallow chunk-based tree representations.

Shallow chunk-based tree (CH). Similarly to [31, 32, 29], we represent a pair of short texts as two trees with lemmas at leaf level and their part-of-speech (POS) tags at the preterminal level. Preterminal POS-tags are grouped into the chunk nodes and the chunks are further grouped into sentences. For example, Figure 2 shows the shallow tree representation of the two Q and AP reported in the introduction.

Dependency-based tree (DT1) [32]. We structure the output of dependency parsers to design a new grammatical relation centered tree. This is a dependency tree altered so that grammatical relations become nodes. Lemmas and their POS tags are allocated at the leaf and the preterminal levels, respectively. Finally, we add '::' and the first letter of the respective POS tags to the leaves, e.g., "world::n". Figure 3 illustrates a DT1 representation of the question from the running example.

Dependency-phrase based tree (DT2). We further generalize DT1 with an additional layer of chunk label nodes between the POS and the grammatical relation node layers. Lemmas in the same chunk or in the "object of preposition" (**pobj**) or the "possession modifier"(**poss**) relations are grouped under the same chunk node. Figure 4 provides an example of a DT2 structure.

Lexical-centered dependency tree (DT3) [8]. Finally, we engineer DT3 in which dependency relations *rel(head,child)* are represented by a parent and a child node labeled *head::pos* and *child::pos*, respectively (lemmas are specialized with the first letter of their POS tags, i.e.,*::pos*). We add the information about the name of the relation, rel, and POS of its child as the rightmost children, with $GR-$ and $POS-$ tags prepended, respectively. For example, we encode the relation *nsubjpass(bill,stadium)* by creating a parent-child pair of nodes labeled *bill::v* and *stadium::n* and adding children labeled as *GR-nsubjpass* and *POS-NN* to the latter. Figure 5 provides an example of an DT3 structure.

4.1 Encoding Relations in the Tree Pairs

Previous work has shown the importance of encoding information about relations between question and answer passage into their structural representations, e.g., [2, 31]. Two basic methods to enrich our proposed structures with relational information are described hereafter.

Lexical relations (REL). Structural relations in both kinds of trees are encoded using the REL tag, which links the related structures in the two texts. Our previous work [31] used hard string match and soft-matching methods such as WordNet-based or Latent Dirichlet Allocation-based semantic relatedness metrics to find related lemmas. However, soft matching did not improve the models simply using hard-matching. The POS tag and the chunk pos tags in all representations are marked with REL label. In DT3 we mark the POS and grammar relation nodes. For example, Figure 2 shows that the lemma "world" occurs in both Q and AP, (we highlighted this with the solid line box), thus the related POS and chunk nodes are marked with REL.

Question Focus-based relations (FREL). Semantic relations specific to QA can be derived from the question focus and category. These are encoded using the REL-FOCUS-<QC> tag, where <QC> is substituted with a question class in the specific examples. As in [31], we use statistical classifiers to derive focus and categories of the question and of the NEs in the AP. We consider HUM, LOC, ENTY, NUM, ABBR and DESC question classes [20]. Question focus and AP chunks, which contain NEs of type compatible with the question class[6], are marked by prepending the above tags to their label. Figure 2 shows an example of such label in the dotted box, however, note that in this case the statistical classifier has determined the focus incorrectly, as it should be *stadium*, instead of *sport*. No named entities (NEs) of classes compatible with the question class ENTY are in AP, therefore there are no REL-FOCUS-ENTY tags in it.

5. SEMANTIC MATCH USING LD

Encoding relational information between Q and AP, i.e., links between words or constituents, is essential for improving passage reranking. Our previous work [32, 31, 29] has only defined two

[5]http://uima.apache.org/

[6]Compatibility is checked by means of a predefined compatibility table. We use the following mappings: **Person, Organization** → **HUM ,ENTY**; **Misc** → **ENTY**; **Location** →**LOC**; **Date, Time, Money, Percentage, Set, Duration** →**NUM**

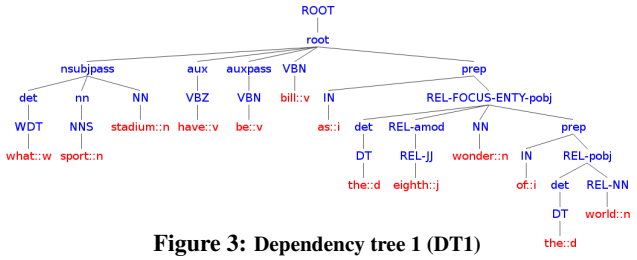

Figure 3: Dependency tree 1 (DT1)

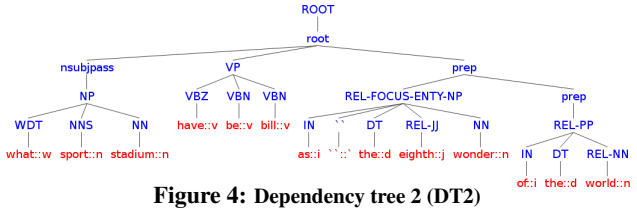

Figure 4: Dependency tree 2 (DT2)

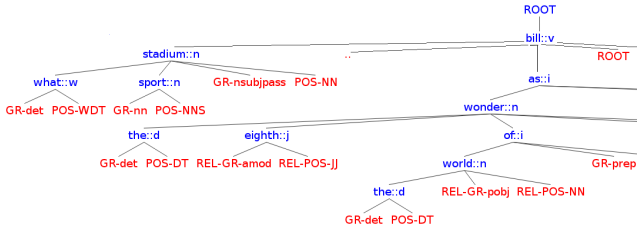

Figure 5: Dependency Tree 3 (DT3)

basic approaches based on string matching or question classifiers. The methods clearly suffer from (i) coverage problems, i.e., due to word mismatch and (ii) the non-perfect coverage and accuracy of classifiers and named entity recognizers (NER). Additionally, other attempts to use semantic matching, e.g., based on WordNet, have failed. Our approach targets entities defined in LD, it highly increases coverage and at the same time avoids errors of classifiers. In this work, we employ robust and accurate entity match algorithms we defined in [37].

More specifically, we detect semantic relations between word sequences in two texts, T_{ent} and T_{gen}. We look for word sequences, e.g., noun phrases, in T_{ent} and T_{gen}, which denote two classes or a class and an entity, which are in a *isSubclassOf* or *isa* relation. Here, class is a set of entities, where two classes, C_1 and C_2, are in a *subClassOf* relation if $C_1 \subseteq C_2$. Similarly, an entity e and class C are in *isa* relation if $e \in C$. We call such relation between word sequences a Type Match (TM). For instance, the word sequences *stadium* and *Astrodome* in the running Q/AP example are in the TM relation, since *stadium* denotes a class of all the stadiums, and *Astrodome* is one particular stadium. We extract knowledge about entities, classes and their relations from the DBpedia, YAGO and WordNet datasets available as LD.

5.1 Linked Data (LD)

LD is structured data published according to defined guidelines[7]. They suggest using HTTP Unique Resources Identifiers (URIs) for naming "things", i.e. *classes*, *entities*, *relationships*, and using standards such as Resource Description Framework (RDF)[8] and SPARQL query language for representing and querying knowledge. In RDF, knowledge is represented as a set of *(subject, predicate, object)* triples forming a directed graph, where subjects/objects are vertices and predicates are edges.

LD publishers reuse a number of vocabularies and ontologies when modeling their data. RDF Schema (RDFS)[9] is the core vocabulary, which contains basic elements for knowledge description. For example, its predicates `rdf:type` and `rdfs:SubClassOf`, denote the *isa* and *subClassOf* relationships that we described above. `rdf:label` predicate is utilized to describe human-readable names of the things uniquely identified by the URIs. For instance, in YAGO, the URIs for the *stadium* class and the *Astrodome* entity are `yago:wordnet_stadium_104295881`[10] and `yago:Reliant_Astrodome`, respectively, and the triple expressing their relationship is *(yago:Reliant_Astrodome, rdf:type, yago:wordnet_ stadium_ 104295881)*.

LD published under an open license is called Linked Open Data (LOD). In this work we use three large cross-domain datasets, namely WordNet, DBpedia and YAGO. WordNet is a manually created lexical database which groups synonymous words into synsets, and stores information about their relationships, e.g., hyponymy, hypernymy or meronymy. Since WordNet was created and is maintained manually, its coverage of NEs and domain-specific classes is limited.

YAGO was automatically created by combining WordNet and Wikipedia. YAGO construction algorithm converts WordNet synset hypernymy/hyponymy hierarchy into a class hierarchy and adds leaf Wikipedia categories as leaf classes to it using a heuristic category-to-synset mapping algorithm. The Wikipedia pages which belong to these categories become individuals of the resulting YAGO classes. A set of heuristics is employed to obtain facts about them from Wikipedia pages.

DBpedia was also automatically created using Wikipedia as a primary source of knowledge, and similarly to YAGO, Wikipedia pages are converted to individuals, and a set of heuristics is used to extract knowledge about them from Wikipedia infoboxes and categories. Hierarchy of classes in DBpedia was manually constructed. The classes are populated with individuals using infobox-class mappings.

Algorithm 1 Type Match algorithm

Require: T_{ent}, T_{gen} - short texts; LD_d - LD knowledge source
1: $TM \leftarrow \emptyset$
2: **for all** $a_{ent} \in$ getAnchors(T_{ent},LD_d) **do**
3: **for all** uri $\in$ getURIs($anchor$,T_{ent},T_{gen}, LD_d) **do**
4: **for all** $type \in$ getTypes(uri,LD_d) **do**
5: **for all** ch $\in$ getChunks(T_{gen}) **do**
6: $a_{gen} \leftarrow$ checkMatch(ch, type.label)
7: **if** $a_{gen} \neq \emptyset$ **then**
8: $TM \leftarrow TM \cup \{(a_{ent}, a_{gen})\}$

5.2 Type Match Detection Algorithm

In this work, we employ Algorithm 1 to detect token sequences (hereafter denoted as *anchors*) in TM relation. The algorithm takes two short texts, T_{ent} and T_{gen}, an LD dataset, LD_d, and an empty set, TM, as input. It (i) scans T_{ent} for the *anchors* that refer to classes or entities in LD_d (line 2); (ii) for each anchor, a_{ent} in T_{ent}, it detects the URIs of the corresponding entities in LD_d (line 3); (iii) for each uri the algorithm extracts a list of types that generalize[11] it along with their human-readable labels (line 4); (iv) finally, it iterates through chunks in T_{gen} and human-readable labels

[7] `http://www.w3.org/DesignIssues/LinkedData.html`
[8] `http://www.w3.org/TR/rdf-concepts/`
[9] `http://www.w3.org/TR/rdf-schema/`
[10] `yago:` is a shorthand for `http://yago-knowledge.org/resource/`
[11] Contain the entity or subsume the class denoted by uri

of types extracted in the previous step and checks them for string match (line 5-6). If the last tokens[12] in the chunk *ch*, a_{gen}, match the last tokens of the human-readable label of a type, $type.label$, extracted for a_{ent} from T_{ent}, we set a_{ent} and a_{gen} to be in TM relation, and add a tuple $\{(a_{ent}, a_{gen})\}$ to the TM set.

For instance, if $T_{ent} = AP$ and $T_{gen} = Q$ from our running example and $LD_d = YAGO$, we: (i) scan AP for all the anchors, which may denote YAGO classes or entities and extract their URIs. In our example, the *Astrodome* token in AP is an anchor with the respective YAGO URI `yago:Reliant_Astrodome`. (ii) We extract the URIs and human-readable names of the classes with which the URIs extracted in step (i) are in `rdf:type` or `rdfs:subClassOf` relation. For example, `yago:Reliant_Astrodome` is in `rdf:type` relation with the `yago:wordnet_stadium_104295881` class with the humane-readable label *stadium*. (iii) We look for the occurrences of the human-readable names of the classes extracted in step (ii) in the Q text. In our example, the human-readable label *stadium* occurs in the question. Therefore, we detect a TM relation between the *Astrodome* anchor from AP and the *stadium* anchor from Q, i.e., in line 8 of Algorithm 1 $(a_{ent}, a_{gen}) =$ ("$Astrodome$", "$stadium$"). We describe the details of the implementations of $getAnchors$, $getURIs$, $getTypes$ procedures in the subsection below.

Detecting anchors and referent entities/classes. We detect the sequences of tokens in text, a_{ent}, which constitute entities and classes in an external knowledge source, i.e., YAGO, DBpedia and WordNet. This is not as simple as a dictionary look-up as the a_{ent} are ambiguous. However, we benefit from the fact that (i) YAGO and DBpedia are aligned with Wikipedia pages on entity-level by construction; and (ii) there are several so-called *wikification* algorithms, which find references to Wikipedia pages in plain text, and disambiguate them to correct Wikipedia pages [9, 23]. Thus, we wikify text to both detect about anchors in T_{ent} and extract URLs of the Wikipedia pages they refer to. In order to have a richer disambiguation context, we concatenate T_{ent} with T_{gen} before passing it to the Wikification tool. We convert the Wikipedia page names to YAGO entities URIs by using YAGO `yago:hasWikipediaUrl` property. We obtain DBpedia URIs by the Wikipedia page prefix, `http://en.wikipedia.org/wiki/`, with `http://dbpedia.org/resource/`. For instance, in our running example the wikification tool maps the *astrodome* token from AP to the `http://en.wikipedia.org/wiki/Reliant_Astrodome` page and YAGO contains a triple *(yago:Reliant_Astrodome, yago:hasWikipediaURL, "http://en. wikipedia. org/ wiki/ Reliant_Astrodome")*.

In case of WordNet, we consider all noun phrases in T_{ent} to be the anchors and all the WordNet synsets containing them to be their references, i.e., we have more than one reference for an anchor.

Extracting generalization classes and their names. In case of YAGO and DBpedia, we employ RDFS predicates `rdf:type` and `rdfs:subClassOf` to extract URIs of classes which generalize the entities/classes returned by the $getURIs$ procedure directly or transitively. The URIs are HTTP unique identifiers. We extract their human-readable names by exploiting the `rdfs:label` property. For example, the output of $getTypes(uri =$`yago:Reliant_Astrodome`$, LD_d = YAGO)$ includes `yago:wordnet_stadium_104295881` with the human-readable labels such as *bowl*, *stadium* and *arena*.

In case of WordNet, generalizations are all the hypernyms of the uri-s output by the *getURIs* procedure, and their human-readable names are all words belonging to the hypernym synsets.

[12]Matching last tokens helps to increase the number of TM relations. For example, the human-readable name of one of the YAGO classes of *Astrodome* in the running example AP is "covered stadium". We would miss the match with the "stadium" in Q, if we only looked for the entire class name in it.

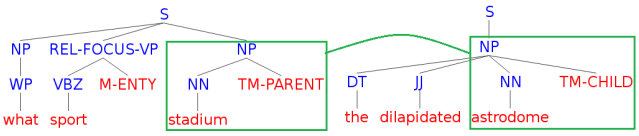

Figure 6: Fragments of a CH tree annotated in TM$_{ND}$ mode.

6. ENCODING RELATIONAL KNOWLEDGE

Encoding relational information in Q/AP is about establishing links between the Q and AP structures, in turn, this is about finding and characterizing matching between text constituents. The previous section has shown powerful methods using LD to establish a match between entities, possibly coming from two different pieces of text, e.g., Q and AP. Considering that in LD instances and classes are defined, we can also define different types of the match occurring between entities.

6.1 REL Based on LD Match Types

We start from the basic structures in Fig. 2 and we enrich them by adding TM[13] labels to their nodes. Let us define L_{ent} and L_{gen} as the sets of nodes corresponding to the tokens composing two anchors, a_{ent} and a_{gen} (from two texts T_{ent} and T_{gen}), respectively. We define the following different types of relational information:
(i) **Untyped (TM$_N$)**. We add a leaf sibling node labeled *TM* to the parent of each node in L_{ent} and L_{gen}.
(ii) **Direction-typed (TM$_{ND}$)**. By construction, one of the anchors in TM relation, a_{gen}, refers to a class in the LD dataset that contains an entity or generalizes a class to which the other anchor, i.e., a_{ent} refers to. We reflect this fact by adding sibling nodes labeled TM-PARENT to the parents of all nodes in L_{gen}, and sibling nodes labeled TM-CHILD to the parents of nodes in L_{ent}. In our running Q/AP example, *stadium* is the generalization of the anchor $Astrodome$. Therefore, as Figure 6 shows, we mark *stadium* as TM-PARENT and *Astrodome* as TM-CHILD.
(iii) **Focus-typed (TM$_{NF}$)**. If one of the anchors is also the question focus, we add sibling nodes, TM-FOCUS, to the parents of nodes in L_{ent} and L_{gen}. Otherwise, we use TM_N.
(iv) **Combo (TM$_{NDF}$)**. We apply both TM$_{ND}$ and TM$_{NF}$ strategies thus adding two different types of nodes.

TM$_{ND}$, TM$_{NF}$, TM$_{NDF}$ are more specific than TM$_N$, therefore we expect them to provide more discriminative patterns. We use the techniques described above for encoding the TM relations into CH, DT1 and DT2 structures, while in DT3 we add a TM node to matched lexicals as a child. Figure 6 illustrates the TM$_{ND}$ strategy applied to CH.

6.2 Wikipedia-based REL (wikiREL)

Section 3 has shown the importance of using REL tags. In our early work [31], these were generated using hard lemma matching. This may result in low coverage as, e.g., it does not capture synonyms or different variants of the same name. Wikification tools handle this problem, as typically they have precomputed sets of different variants of names for the same page, extracted from internal Wikipedia links and redirection pages. Therefore, if a pair of anchors in T_{ent} and T_{gen} are annotated with the link to the same

[13]It is important to use different tags than REL to convey a different type of information.

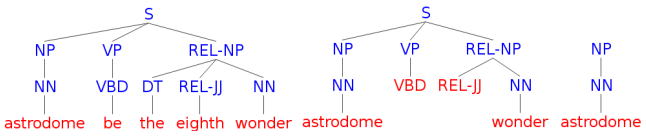

Figure 7: Some fragments generated by PTK or equivalently by SHTK (in faster time) when applied to the question tree of Fig. 2

page by a wikification tool, we consider them to be matching and we mark them with REL tags in the structural representations of T_{ent} and T_{gen} as described in Sec. 4.1.

7. L2R WITH TREE KERNELS

The above structures can be used by a domain expert to design machine learning features for training relational classifiers. Given the complexity of this task, we can rely on kernel machines, e.g., SVMs, for automatic feature generation. These classify a test input $\boldsymbol{x}$ using the function: $h(\boldsymbol{x}) = \sum_i \alpha_i y_i K(\boldsymbol{x}, \boldsymbol{x}_i)$, where α_i are the model parameters estimated from the training data, y_i are target variables, $\boldsymbol{x}_i$ are support vectors, and $K(\cdot, \cdot)$ is a kernel function. The latter computes the *similarity* between two objects. If we use kernel functions, we do not need to represent objects with features and thus we do not need to design features at all. In case of the convolution tree kernels, K counts the number of common subtrees between two trees T_1 and T_2 without explicitly considering the whole fragment space. The general equations for convolution tree kernels is: $TK(T_1, T_2) = \sum_{n_1 \in N_{T_1}} \sum_{n_2 \in N_{T_2}} \Delta(n_1, n_2)$, where N_{T_1} and N_{T_2} are the sets of the T_1's and T_2's nodes, respectively and $\Delta(n_1, n_2)$ is equal to the number of common fragments rooted in the n_1 and n_2 nodes, according to several possible definitions of the atomic fragments. We use the Δ constituting PTK [24], which defines any possible set of connected nodes as features. These capture dependencies between structure elements, e.g., Fig. 7 shows fragments generated for the question tree of Fig. 2.

7.1 Reranking with Kernels

The above section has shown how we can build classifiers based on kernels. However, the same framework can be easily adapted for learning to rank problems. In particular, we can design a function deciding which Q/AP pair is more probably correct than the others, where correct Q/AP pairs are formed by an AP containing a correct answer to Q and a supporting justification. For this purpose, we adopt the following kernel for (preference) reranking: $P_K(\langle o_1, o_2\rangle, \langle o_1', o_2'\rangle) = K(o_1, o_1') + K(o_2, o_2') - K(o_1, o_2') - K(o_2, o_1')$. In our case, $o_i = \langle Q_i, AP_i\rangle$ and $o_j' = \langle Q_j', AP_j'\rangle$, where Q and AP are the trees defined in the previous section, and $K(o_i, o_j') = TK(Q_i, Q_j') + TK(AP_i, AP_j')$. TK can be any tree kernel function, e.g., PTK. Finally, we also add $(\vec{FV}(o_1) - \vec{FV}(o_2)) \cdot (\vec{FV}(o_1') - \vec{FV}(o_2'))$ to P_K, where $\vec{FV}(o_i)$ is a traditional feature vector representing Q/AP pairs. This enables the use of standard features.

8. EXPERIMENTS

We evaluated our structural representations on the passage and sentence retrieval subtasks. We investigated the impact of the different structures described in this paper on several datasets of different nature. Moreover, we compared the impact of various preprocessing annotation pipelines on the final system accuracy.

8.1 Experimental Setup

We utilized three different datasets for testing our models:

TREC QA 2002/2003. TREC QA tasks provide questions along with the answer keys, which can be used to select the passages containing correct answers where the passages are extracted from a given text corpus. We used a total of 824 questions from years 2002 and 2003[14]. The AQUAINT[15] corpus is used for searching the supporting answer passages.

TREC13. Factoid open-domain TREC QA corpus prepared by [42]. The training data was assembled from the 1,229 TREC8-12 questions. The answers for the training questions were automatically marked in sentences by applying regular expressions, therefore the dataset can be noisy. The test data contains 100 questions, whose answers were manually annotated.[16] Consistently with the previous work, we remove the questions that have only correct/only incorrect candidate APs, thus reducing the dataset to 68 questions. We used 10 answer passages for each question for training our classifiers and all the available answer passages for testing.[17]

Answerbag. It is a collaboratively constructed question-answer resource. Some of the answers are marked as "professionally researched" meaning that they have been professionally edited and fact-checked thus making them high-quality QA training data. We used 2,000 questions for training and 1,000 for testing, using 10 and 50 passages, respectively.

LD datasets. We used the core RDF distribution of YAGO2[18], WordNet 3.0 in RDF,[19] and the datasets from the DBpedia 3.9[20].

Feature Vectors. We used several similarity functions between the pairs of texts, computed over various input representations to form a feature vector, as described hereafter: *Term-overlap features:* a cosine similarity over the text pair: $sim_{COS}(T_1, T_2)$. Input vectors are composed of word lemmas, bi-, three- an four-grams, POS-tags. *PTK over tree representations:* similarity based on the PTK score computed for the structural representations of T_1 and T_2: $sim_{PTK}(T_1, T_2) = PTK(T_1, T_2)$, where the input trees can be both the dependency trees and/or the shallow chunk trees. *Search engine ranking score:* when experimenting with TREC QA and Answerbag, we also use a ranking score of our search engine assigned to AP.

Learning Models. We used SVM-Light-TK[21] to train our models. The toolkit enables the use of structural kernels [24] in SVM-Light [18]. We used default PTK parameters as described in [31] and the polynomial kernel of degree 3 on standard features.

Pipeline. We built the entire processing pipeline on top of the UIMA framework.We included many off-the-shelf NLP tools wrapping them as UIMA annotators. We use the Apache OpenNLP[22] and Stanford CoreNLP [21] tools for sentence detection, tokenization, POS-tagging and NE recognition; Illinois chunker [27], Stanford CoreNLP Lemmatizer, and question class and focus classifiers trained as in [31]. For dependency parsing we used Stanford dependency parser (version 2.0.3) and UIMA wrappers provided by the DKPro toolset [10] for the Mate [6] (v3.5), ClearNLP [7] (v2.0.2,

[14] http://trec.nist.gov/data/qamain.html

[15] https://catalog.ldc.upenn.edu/LDC2002T31

[16] We downloaded the distribution made available by [44] in https://code.google.com/p/jacana/.

[17] In order to obtain results comparable to those in the previous works experimeting on TREC13 [45, 40, 42], we used the same evaluation setting as described in footnote 7 in [45]. In this setting the gold judgment file contains 4 extra questions not covered by the test set, thus resulting in lowering the upper bound of MAP and MRR to 94.44 instead of 100

[18] http://www.mpi-inf.mpg.de/yago-naga/yago1_yago2/download/yago2/yago2core_20120109.rdfs.7z

[19] http://semanticweb.cs.vu.nl/lod/wn30/

[20] http://dbpedia.org/Downloads39

[21] http://disi.unitn.it/moschitti/Tree-Kernel.htm

[22] https://opennlp.apache.org/index.html

Table 1: Comparison of different syntactic structures combined with vectors and basic relations

	TREC QA 2002/2003			Trec13		Answerbag		
	MRR	**MAP**	**P@1**	**MRR**	**MAP**	**MRR**	**MAP**	**P@1**
BM25	28.02±2.94	21.90±1.67	18.17±3.79	n/a	n/a	65.96	66.24	55.60
V	32.70±2.25	25.76±1.08	23.05±2.91	74.06	64.32	67.94	68.22	58.60
CH+V+FREL [31]	39.49	32.00	30.00	73.58	67.81	n/a	n/a	n/a
DT1+V+FREL [31]	38.05	31.00	28.39	n/a	n/a	n/a	n/a	n/a
CH+FREL	38.85±1.07	31.87±1.29	28.41±0.92	82.29	73.34	63.20	63.53	53.10
CH+V+FREL	41.51±1.65	33.54±1.35	32.20±1.90	81.30	72.65	69.90	70.07	60.80
DT1+V+FREL	41.73±1.71	33.55±1.60	32.44±1.99	78.47	70.56	69.33	69.52	60.20
DT2+V+FREL	42.75±2.43	34.05±2.18	33.66±2.97	78.45*	71.21*	68.97	69.17	59.50
$DT3_Q$+$DT2_A$+FREL	39.80±2.67	31.75±1.66	29.63±2.94	79.98	72.19	62.91	63.26	53.20
$DT3_Q$+$DT2_A$+V+FREL	43.22±2.57	34.37±1.78	34.39±2.91	80.57	73.40	69.51	69.70	60.20

Table 2: Comparing the impact of the preprocessing pipelines in building structures on TREC QA 2002/2003 data.

	Config	MRR	MAP	P@1
CH+V+REL FREL	$OpenNLP_{pos}$	40.75±2.35	32.75±1.79	31.46±2.81
	$Stanford_{pos}$	41.51±1.65	33.54±1.35	32.20±1.90
CH_{pr}+ V+REL+FREL	$OpenNLP_{pos}$	39.81±1.80	32.28±0.98	29.51±2.22
	$Stanford_{pos}$	41.21±1.36	33.40±1.84	32.32±1.93
$DT3_Q$+ $DT2_A$ +V+FREL	Stanford	43.22±2.57	34.37±1.78	34.39±2.91
	Mate	41.31±2.22	33.38±1.30	31.46±3.44
	ClearNLP	40.72±1.64	32.05±1.54	30.98±2.71
	Malt	41.78±1.67	33.25±2.03	32.68±1.40

ontonotes model) and Malt [25] (v1.7.2, linear model) dependency parsers. Moreover, we used annotators for building new sentence representations starting from tools' annotations. For example, we generated annotations with shallow chunk-based structural presentations using POS-tag and chunk annotation output by the previous annotators in the pipeline.

Search engines. We retrieved answer passages for TREC QA 2002/ 2003 from the AQUAINT corpus and Answerbag answer collection, respectively. We used Terrier[23] along with the accurate BM25 scoring model. We retrieved 50 passages for each question.

Wikification tools. We used the Wikipedia Miner [24] (WM) [23] and the Machine Linking (ML)[25] tools. These return links along with their confidences ranging from 0 to 1. To have high coverage with Wikipedia links, we used all the links with confidence exceeding 0.2 and 0.05 for WM and ML, respectively.

QA metrics. We used common QA metrics: Precision at rank 1 (P@1) i.e., the percentage of questions with a correct answer ranked at the first position, and Mean Reciprocal Rank (MRR). We also report the Mean Average Precision (MAP).

Significance tests. We used paired two-tailed t-test for evaluating the statistical significance of the cross-validation experiments. ‡ and † indicates the significance levels of 0.05 and 0.1, respectively.

8.2 Comparing Shallow and Deep Structures

Table 1[26] reports the performance of the relational syntactic representations enriched with question class, NE and question focus information provided by the statistical classifiers.

Here, BM25 refers to the performance of the Terrier search engine, **V** is a baseline reranker only employing the feature vectors described in Section 8.1. In the following lines, CH, DT1, DT2, DT3 correspond to the eponymous structures described in Section 4. **CH+V+REL+FREL** and **DT1+V+REL+FREL** indicate the systems employing, **CH** and **DT1** representations, respectively whereas **+REL**, **+FREL** indicate that the structures are linked with the REL and FREL approaches (Sec. 4.1). **$DT3_Q$+$DT2_A$** means that question and answer passage are represented by DT3 and DT2, respectively. **+V** indicates that a feature vector is added to the tree representation.

In our experiments on TREC 2002/2003, CH+V+FREL performs comparably to DT1+V+FREL and is outperformed by the other dependency structures. In our intuition, this new outcome may be due to the performance of the dependency parser employed for preprocessing [26].

This intuition motivated us to evaluate the impact of four different parsers for building $DT3_Q$+$DT2_A$. The results in Table 2 show that ClearNLP parser is outperformed by Mate, Malt and Stanford, where the latter always obtains the best result. Additionally, we compared the impact of using sentence splitters, tokenizers and POS-taggers from two basic preprocessing pipelines, Stanford CoreNLP ($Stanford_{pos}$) and OpenNLP ($OpenNLP_{pos}$), on the final performance of the simpler CH structure. The results show that $Stanford_{pos}$ outperforms $OpenNLP_{pos}$.

8.3 Measuring the Impact of LD Semantics

In these experiments, we evaluated the accuracy achieved by the baseline systems enriched with wikiREL (Section 6.2), and TM-relations. First, we ran an evaluation of all possible LD sources combinations with different TM-match knowledge techniques in cross-validation on TREC 2002/2003 and then we ran the resulting best systems on TREC13 and Answerbag.

8.3.1 TREC QA Answer Passage Reranking

We tested the impact of LD relations in structures using (i) basic lexical relations, i.e., CH+V+REL, and (ii) the lexical and semantic relations, i.e., CH+V+REL+FREL, described in Section 4.1. For these experiments, we (i) performed 5-fold cross-validation on TREC 2002/2003 and (ii) employed the preprocessing pipeline based on OpenNLP sentence splitter, tokenizer, POS tagger and Illinois chunker.

Tables 3 and 4 report the results using CH+V+REL and CH+V+ REL+FREL, respectively. Since wikiREL improves coverage, we preferred to add it to all TM types of relations. Thus, after the double line, the tables display the baselines (in bold) enriched ("+") with different models based on TM_N, TM_{ND}, TM_{NF}, TM_{NDF}, i.e., different TM-encoding strategies (see Section 6). Additionally, **D**, **Y**, **W** denote that DBpedia, YAGO and WordNet, respectively, were used as the LD dataset, when detecting TM matches. We use **+** to indicate a combination of several LD datasets. We will reuse such notation in all the following tables. We evaluate statistical significance of the results obtained when using TM as compared to the results reported in the CH+V+REL+wikiREL and CH+V+REL+FREL+wikiREL lines for tables 3 and 4.

The tables show that all the systems exploiting LD knowledge, excluding those using DBpedia only, outperform the strong CH+V +REL and CH+V+REL+FREL baselines. Note that CH+V+REL+

[23] http://terrier.org/

[24] http://sourceforge.net/projects/wikipedia-miner/files/wikipedia-miner/wikipedia-miner_1.1

[25] http://www.machinelinking.com/wp

[26] In [31], we reported MAP values rounded to decimals as 0.xx. As we do not have more decimal digits for such results, we converted them to xx.00.

Table 3: Results in 5-fold cross-validation on TREC QA corpus, each TM model is added to CH+V+REL+wikiREL

System	MRR	MAP	P@1
CH+V+REL	36.82±2.68	30.08±1.63	26.34±2.17
CH+V+REL+wikiREL	39.17±1.29‡	31.34±1.34‡	28.66±1.43 ‡
+TM$_N$:D	40.60±1.88	33.28±1.11‡	31.10±2.99 †
+TM$_N$:W	41.39±1.96 ‡	33.43±1.15‡	31.34±2.94
+TM$_N$:W+D	40.85±1.52 ‡	33.41±1.21‡	30.37±2.34
+TM$_N$:Y	40.71±2.07	33.27±2.53†	30.24±2.09‡
+TM$_N$:Y+D	41.25±1.57‡	33.78±1.92‡	31.10±1.88 ‡
+TM$_N$:Y+W	42.01±2.26 ‡	34.38±2.39 ‡	32.07±3.04 ‡
+TM$_N$:Y+W+D	41.52±1.85‡	33.78±1.80 ‡	30.98±2.71‡
+TM$_{NF}$:D	40.67±1.94†	33.46±1.28‡	30.85±2.22 †
+TM$_{NF}$:W	40.95±2.27 ‡	33.27±1.37‡	30.98±3.74
+TM$_{NF}$:W+D	40.84±2.18 †	33.73±1.36‡	30.73±3.04
+TM$_{NF}$:Y	42.01±2.44 ‡	34.38±2.39‡	32.07±3.01 ‡
+TM$_{NF}$:Y+D	41.32±1.70 †	34.32±1.79 ‡	31.10±2.48 ‡
+TM$_{NF}$:Y+W	41.69±1.66 ‡	33.95±1.96 ‡	31.10±2.44 ‡
+TM$_{NF}$:Y+W+D	41.56±1.41‡	34.41±1.80‡	30.85±2.22 †
+TM$_{ND}$:D	40.37±1.87	33.36±0.94‡	30.37±2.17
+TM$_{ND}$:W	41.13±2.14‡	33.34±0.77‡	30.73±2.75
+TM$_{ND}$:W+D	41.28±1.03‡	33.80±0.88‡	30.73±0.82 ‡
+TM$_{ND}$:Y	42.11±3.24 ‡	34.41±2.19‡	32.07±4.06 ‡
+TM$_{ND}$:Y+D	42.28±2.01‡	34.76±1.18‡	32.44±1.99‡
+TM$_{ND}$:Y+W	42.96±1.45‡	34.68±1.49	33.05±2.04 ‡
+TM$_{ND}$:Y+W+D	42.56±1.25 ‡	34.78±1.11 ‡	32.56±1.91‡
+TM$_{NDF}$:D	40.35±1.72 †	33.42±1.06‡	30.49±1.78 ‡
+TM$_{NDF}$:W	40.98±1.96	33.25±0.58†	30.85±2.05
+TM$_{NDF}$:W+D	41.37±1.05‡	34.14±0.78‡	31.34±1.40‡
+TM$_{NDF}$:Y	42.02±2.38‡	34.77±2.23‡	32.07±2.78‡
+TM$_{NDF}$:Y+D	42.07±2.40 ‡	35.10±1.70 ‡	32.44±2.91 ‡
+TM$_{NDF}$:Y+W	43.12±1.44 ‡	34.71±2.30 †	33.78±2.88 ‡
+TM$_{NDF}$:Y+W+D	**43.38±1.58**‡	**35.27±1.61**‡	**34.02±2.81**‡
Severyn et al. [31]	39.49	32.00	30.00

Table 4: Results in 5-fold cross-validation on TREC QA corpus, all models are build on top of CH+V+REL+FREL system

System	MRR	MAP	P@1
CH+V+REL+FREL	40.50±2.35	33.07±1.75	31.46±2.42
CH+V+REL+ FREL+wikiREL	41.33±1.17	34.23±1.10	31.46±1.40
+TM$_N$:D	40.80±1.01	34.43±1.04	30.37±1.90
+TM$_N$:W	42.43±0.56	35.07±0.61	32.80±0.67
+TM$_N$:W+D	42.37±1.12	35.46±1.15	32.44±2.64
+TM$_N$:Y	43.28±1.91†	36.01±1.33†	33.90±2.75
+TM$_N$:Y+D	42.39±1.83	35.21±1.42	32.93±3.14
+TM$_N$:Y+W	43.98±1.08 ‡	36.33±0.57‡	35.24±1.46‡
+TM$_N$:Y+W+D	43.13±1.38	35.62±0.98	33.66±2.77
+TM$_{NF}$:D	41.43±0.70	35.08±0.85	31.22±1.09
+TM$_{NF}$:W	42.37±0.98	35.10±0.95	32.56±1.76
+TM$_{NF}$:W+D	43.08±0.83	36.24±1.36	33.54±1.29
+TM$_{NF}$:Y	43.82±2.36 †	36.32±1.54‡	34.88±3.35
+TM$_{NF}$:Y+D	43.19±1.17 †	36.30±1.16†	33.90±1.86
+TM$_{NF}$:Y+W	44.32±0.70‡	36.49±0.82‡	35.61±1.11‡
+TM$_{NF}$:Y+W+D	43.79±0.73 ‡	36.56±1.18†	34.88±1.69‡
+TM$_{ND}$:D	41.58±1.02	35.23±0.95 †	31.46±1.59
+TM$_{ND}$:W	42.19±1.39	34.94±0.65	32.32±1.36
+TM$_{ND}$:W+D	42.37±1.16†	35.90±1.11†	32.44±2.71 †
+TM$_{ND}$:Y	44.04±2.05 ‡	36.47±1.18‡	34.63±2.17‡
+TM$_{ND}$:Y+D	43.77±2.02 †	36.55±1.35 †	34.27±2.42
+TM$_{ND}$:Y+W	44.25±1.32 ‡	36.71±0.37 ‡	34.76±1.61 ‡
+TM$_{ND}$:Y+W+D	43.91±1.01‡	36.51±0.84 †	34.63±1.32 ‡
+TM$_{NDF}$:D	41.56±1.10	35.12±1.05†	31.59±1.46
+TM$_{NDF}$:W	41.97±0.96	34.75±0.73	31.71±1.14
+TM$_{NDF}$:W+D	42.12±1.08	35.74±1.46	32.20±2.09
+TM$_{NDF}$:Y	**44.99±2.45** ‡	**37.16±1.69** ‡	**36.59±3.02** ‡
+TM$_{NDF}$:Y+D	44.14±1.85	36.66±1.22	35.00±2.05
+TM$_{NDF}$:Y+W	44.55±1.42 ‡	37.01±1.21 ‡	35.73±2.22 ‡
+TM$_{NDF}$:Y+W+D	43.83±0.95 ‡	36.73±1.29 †	34.51±1.96 †
V	32.67±1.74	26.08±0.83	22.68±2.67
V+VL	35.39±2.49	28.34±1.81	25.73±3.54
CH+V+VL+REL+FREL	41.36±2.17	34.04±1.19	31.46±2.81

wikiREL systems enriched with TM tags perform comparably to CH+V+REL+FREL, i.e., using question and focus classifiers, and in some cases even outperform it. Thus LD models can avoid the use of training data and language/domain specific classifiers.

Additionally, the tables show that we typically obtain better results when using YAGO2 and/or WordNet. In our intuition this is due to the fact that these resources are large-scale, have fine-grained class taxonomy and contain many synonymous labels per class/entity thus allowing us to have a good coverage with TM-links. The DBpedia ontology that we employed in the **D** experiments is more shallow and contains fewer labels for classes, therefore the amount of discovered TM matches is not always sufficient for increasing performance. YAGO2 provides better coverage for TM relations between entities and their classes, while WordNet contains more relations between classes.[27] Note that, we also used WordNet in our experiments in [31] but we employed supersenses whereas in this paper we use hypernymy relations. Moreover, we used a different technique to incorporate semantic match into the tree structures. This allowed our new models to relatively improve the old ones by about 16%, e.g., 37.16 vs. 32.0 in MAP.

Next, using different TM-knowledge encoding strategies, i.e., TM$_N$, TM$_{ND}$, TM$_{NF}$, TM$_{NDF}$, results in small performance variations. This means that encoding LD information is basically simpler than what we thought. We further tested the impact of the different encoding strategies when using the Stanford parser. Table 5 reports results of the structures built by such parser and enriched with LD, according to different TM encoding techniques.

Finally, the last three lines of Tab. 4 show our attempt to encode LD information in a feature vector, i.e., VL. This refers to the number of TM matches between the Q and AP, for different types of TM. It is basically the unstructured version of our LD models. As it can be seen, V+VL only slightly improves V and CH+V+VL+REL+FREL is about 4 points absolute less than the best model using LD in structures, i.e., +TM$_{NDF}$. This confirms that semantic knowledge requires to be used in syntactic structures.

Comparison with TREC challenge. An approximate (as we used five-fold cross-validation) comparison can be attempted with the results from TREC 2003 for the "best passages tasks" described in TREC-overview [39]. Thanks to LD our system achieves an accuracy (Precision@1) of 36.59, which would allow it to be ranked 3rd in the official evaluation, i.e., higher than MultiText-system (accuracy=35.1) and below the systems of LCC and Singapore (68.5 and 41.9, respectively). However, such top two systems used many handcrafted rules, resources and heuristics, which also prevent researchers to replicate them. In contrast, we generated features automatically, we do not design rules, and all our technology is already off-the-shelf, except for some missing components that we will make freely available to facilitate replicability of our results.

8.3.2 Answerbag

The first two lines of Table 6 report our chosen baselines whereas the last six lines show the accuracy of the systems that typically exhibited the best accuracy on our experiments on TREC QA 2002/2003 corpus. The relative improvement when using LD and Wikipedia is lower. This is likely due to the fact that the textual overlap between questions and answers on Answerbag is higher than that on TREC QA 2002/2003, thus the generalization provided by our structures is less important. Note, that we also experiment with DT1 structure built with OpenNLP pipeline and it is outperformed by the one constructed using Stanford preprocessing tools (see Table 1).

8.3.3 TREC13: Sentence Reranking

Table 7 reports the accuracy of the baseline CH+V+REL+FREL, along with the combinations with wikiREL and TM knowledge. The results on the TREC13 data further confirm the usefulness of LD-based TM knowledge, which allows us to obtain a statistically

[27] We consider the WordNet synsets to be classes in the scope of our experiments

Table 5: Syntactic structures using Stanford parser and LD

Model	LD	MRR	MAP	P@1
CH+V+REL+FREL	+ TM$_N$:Y+W	43.12±2.70	35.56±2.12	33.05±4.08
	+TM$_{ND}$:Y+W	43.55±1.81	35.72±1.52	33.90±2.64
	+TM$_{NF}$:Y+W	43.54±1.74	36.22±1.16	33.78±2.31
	+TM$_{NDF}$:Y	42.57±1.67	35.91±1.94	32.20±2.45
DT1+V+REL+FREL	+TM$_{NDF}$:Y	42.98±1.32	35.47±0.62	32.68±2.09
DT2+V+REL+FREL	+TM$_{NDF}$:Y	43.32±2.24	35.96±1.94	33.17±3.49
$DT3_Q$+$DT2_A$ +V+REL+FREL	+TM$_{NDF}$:Y	43.77±1.79	36.14±1.40	34.15±1.83

Table 6: Experiments of LD on Answerbag data

Model	MRR	MAP	P@1
CH+V+REL+FREL	69.29	69.50	60.00
DT1+V+REL+FREL	62.76	63.08	52.50
CH+V+REL+FREL+wikiREL	70.17	70.31	60.80
$+TM_N$:Y+W	70.66	70.77	61.10
$+TM_N$:Y	70.35	70.46	60.60
$+TM_{ND}$:Y+W	70.62	70.76	61.10
$+TM_{NF}$:Y+W	70.44	70.58	60.50
$+TM_{NDF}$:Y	70.64	70.75	60.80

significant improvement of around 3.5 points in terms of MRR as compared to the baseline CH+V+REL+FREL. The comparison of our models with the state-of-the-art results, reported by Tab. 8, shows that our kernel-based rerankers outperforms the very recent best model, i.e., Wang and Nyberg (2015), by 2.17% absolute in MRR and 1.31% in MAP. Moreover, when we add LD information, the improvement in MAP increases to 2.73% absolute.

8.4 Efficiency

Computational complexity. The complexity of the type match procedure for a short text pair, T_{ent} and T_{gen}, described in Section 5.2 is $O(k \times |T_{gen}| \times |T_{ent}|)$. $|T|$ is the length of text T in terms of words. k is a LD-dataset specific constant denoting the maximal amount of generalizations per all *uri*-s corresponding to a word. For example, in case of WordNet this would be the maximal amount of hypernyms for all the senses of a given word.

Wikipedia annotation time. Wikipedia annotation time depends on the algorithms used to perform wikification, tools and techniques for Wikipedia data storage and retrieval, and hardware capacity. In our experiments we used (i) a local installation of Wikipedia Miner (WM) on a DELL N5110 machine[28] and a (ii) commercial web-service optimized for fast text processing, ML, which we access using REST API. We store preprocessed version of Wikipedia to be used by WM 1.0 in a 5.5.17 MySQL database.

Table 9 provides information about the running time of Wikipedia annotation on two randomly selected subsets of text pairs with different length. *Size* column reports number of Q/AP pairs processed, *Avg length* reports their average length in words, and the two last columns report the average time required to process one pair in seconds (and total time required to process the full corpus in minutes in parentheses). The speed exhibited by the ML service shows that the methods relying on linking to Wikipedia are scalable for large amounts of data given the efficient implementation of the wikification tool.

Type data extraction time. We use Jena TDB 0.9.0[29] RDF triple store to store local YAGO, WordNet and DBpedia data. Jena TDB efficiency[30] was evaluated on a number of benchmarks with

[28]6GB RAM, four core Intel(R) Core(TM) i7-2630M CPU @ 2.00GHz processor, 64-bit operating system, 640gb 2.5" Sata II Hard Drive with 5400 RPM

[29]`http://jena.apache.org/documentation/tdb/index.html`

[30]We installed Jena TDB on a server machine with 12*Intel®Xeon®Processor X5670* processors, with 94GB RAM under Linux, with 64-bit *jdk1.7.0_02* and 1TB SATA 6.0Gb/s 2.5" hard drive with 7200 RPM.

Table 7: Results of CH+FREL plus the best LD models on TREC13

Model	MAP	MRR
CH+REL+FREL	72.65	81.30
CH+REL+FREL+wikiREL	72.99	80.37
+TM$_N$:Y	**74.07**	81.10
+TM$_{NF}$:Y	73.73	79.52
+TM$_{ND}$:Y	**74.05**	80.19
+TM$_{NDF}$:Y	72.94	79.78

Table 8: Previous work results on the TREC13.

Model	MRR	MAP
Wang et al. (2007)[42]	68.52	60.29
Heilman and Smith (2010)[14]	69.17	60.91
Wang and Manning (2010)[41]	69.51	59.51
Yao et al. (2013)[43]	74.77	63.07
Severyn and Moschitti (2013)[31]	75.20	68.29
Yih et al. (2013)[45]	77.00	70.92
Wang and Nyberg (2015) [40]	79.13	71.34
CH+FREL (this work)	**81.30**	72.65
CH+FREL+wikiREL+TM$_N$:Y	81.10	**74.07**

results reported, for example, in [5]. We store YAGO2 and WordNet+DBpedia data in separate triple stores.

In our case, extracting all the generalizations of the URI, i.e. sending a set of SPARQL queries to a triple store, took 18.22, 36.28 and 68.59 milliseconds for WordNet, YAGO and DBpedia, respectively. In average, we extracted 32 *type-type label* pairs per anchor from WordNet, 49 from YAGO and 26 from DBpedia.

Table 9: Wikipedia annotation time

Dataset	Size	Average length	Average (total) time	
			ML	MW
Sample1	1599	36.78	0.14s (3.6m)	2.84s (75.7m)
Sample2	1146	107.30	0.23s (4.3m)	9.02s (172.7m)

9. CONCLUSIONS

This paper proposes a study on syntactic structures enriched with semantic information from statistical classifiers and knowledge from LD for passage reranking. In particular, YAGO, DBpedia and WordNet are used to match constituents from QA pairs. Such matches are used to enrich semantic structures. The experiments with TREC QA and the above models also combining traditional feature vectors and the improved relational structures greatly outperform a strong IR baseline, i.e., BM25, by 101%, and previous state-of-the-art reranking models, e.g., up to 16% in MAP. Differently from previous work, our models can effectively use semantic knowledge in statistical learning to rank methods. It should be stressed that our experiments have shown that simply using semantic information as features (even if extracted from a powerful resource as LD) does not significantly improve BM25. It is really necessary to encode semantic features in syntactic structures and then generate syntactic/semantic relational patterns between question and answer passage (to be used as features in the reranker).

Our promising results open interesting future directions in designing novel semantic structures and using innovative semantic representations in learning algorithms for IR applications. Additionally, jointly using deep neural networks with our approach is an interesting and promising research direction.

Acknowledgements

This work has been partially supported by the EC project CogNet, 671625 (H2020-ICT-2014-2, Research and Innovation action) and by an IBM Faculty Award. Many thanks to the Center for Advanced Studies of Trento of IBM Italia for the fruitful discussions on the early draft of this manuscript.

10. REFERENCES

[1] E. Aktolga, J. Allan, and D. A. Smith. Passage reranking for question answering using syntactic structures and answer types. In *ECIR*, 2011.

[2] M. W. Bilotti, J. L. Elsas, J. Carbonell, and E. Nyberg. Rank learning for factoid question answering with linguistic and semantic constraints. In *CIKM*, 2010.

[3] M. W. Bilotti and E. Nyberg. Improving text retrieval precision and answer accuracy in question answering systems. In *(IR4QA) Workshop at COLING*, 2008.

[4] C. Bizer, J. Lehmann, G. Kobilarov, S. Auer, C. Becker, R. Cyganiak, and S. Hellmann. Dbpedia - a crystallization point for the web of data. *Web Semantics*, 7(3), 2009.

[5] C. Bizer and A. Schultz. The berlin sparql benchmark. *IJSWIS*, 5(2):1–24, 2009.

[6] B. Bohnet. Top accuracy and fast dependency parsing is not a contradiction. In *COLING*, 2010.

[7] J. D. Choi and M. Palmer. Getting the most out of transition-based dependency parsing. In *ACL*, 2011.

[8] D. Croce, A. Moschitti, and R. Basili. Structured lexical similarity via convolution kernels on dependency trees. In *Proceedings of EMNLP*, 2011.

[9] A. Csomai and R. Mihalcea. Linking documents to encyclopedic knowledge. *IEEE Intelligent Systems*, 23(5):34–41, 2008.

[10] R. E. de Castilho and I. Gurevych. A broad-coverage collection of portable NLP components for building shareable analysis pipelines. In *OIAF4HLT Workshop (COLING)*, 2014.

[11] J. Fan, D. Ferrucci, D. Gondek, and A. Kalyanpur. Prismatic: Inducing knowledge from a large scale lexicalized relation resource. In *NAACL HLT*, 2010.

[12] C. Fellbaum, editor. *WordNet: An Electronic Lexical Database*. MIT Press, 1998.

[13] D. Ferrucci, E. Brown, J. Chu-Carroll, J. Fan, D. Gondek, A. Kalyanpur, A. Lally, J. W. Murdock, E. Nyberg, J. Prager, N. Schlaefer, and C. Welty. Building Watson: An Overview of the DeepQA Project. *AI Magazine*, 31(3), 2010.

[14] M. Heilman and N. A. Smith. Tree edit models for recognizing textual entailments, paraphrases, and answers to questions. In *NAACL*, 2010.

[15] A. Hickl, J. Williams, J. Bensley, K. Roberts, Y. Shi, and B. Rink. Question answering with lcc chaucer at trec 2006. In *TREC*, 2006.

[16] J. Hoffart, F. Suchanek, K. Berberich, and G. Weikum. Yago2: a spatially and temporally enhanced knowledge base from wikipedia. *Artificial Intelligence*, 2012.

[17] J. Jeon, W. B. Croft, and J. H. Lee. Finding similar questions in large question and answer archives. In *CIKM*, 2005.

[18] T. Joachims. Optimizing search engines using clickthrough data. In *ACM SIGKDD KDD*, 2002.

[19] B. Katz and J. Lin. Selectively using relations to improve precision in question answering. In *Workshop on NLP for QA (EACL)*, 2003.

[20] X. Li and D. Roth. Learning question classifiers. In *Proceedings of ACL*, 2002.

[21] C. D. Manning, M. Surdeanu, J. Bauer, J. Finkel, S. J. Bethard, and D. McClosky. The Stanford CoreNLP natural language processing toolkit. In *ACL*, 2014.

[22] M. C. McCord, J. W. Murdock, and B. Boguraev. Deep parsing in Watson. *IBM Journal*, 56(3), 2012.

[23] D. Milne and I. Witten. An open-source toolkit for mining wikipedia. In *NZCSRSC*, volume 9, 2009.

[24] A. Moschitti. Efficient convolution kernels for dependency and constituent syntactic trees. In *ECML*, 2006.

[25] J. Nivre, J. Hall, J. Nilsson, A. Chanev, G. Eryigit, S. Kübler, S. Marinov, and E. Marsi. Maltparser: A language-independent system for data-driven dependency parsing. *Natural Language Engineering*, 2007.

[26] M. Pasca. Open-Domain Question Answering from Large Text Collections. CSLI Publications, 2003.

[27] V. Punyakanok and D. Roth. The use of classifiers in sequential inference. In *NIPS*, pages 995–1001, 2001.

[28] F. Radlinski and T. Joachims. Query chains: Learning to rank from implicit feedback. *CoRR*, 2006.

[29] A. Severyn and A. Moschitti. Automatic feature engineering for answer selection and extraction. In *EMNLP*, 2013.

[30] A. Severyn and A. Moschitti. Learning to rank short text pairs with convolutional deep neural networks. In *SIGIR*, 2015.

[31] A. Severyn, M. Nicosia, and A. Moschitti. Building structures from classifiers for passage reranking. In *CIKM*, 2013.

[32] A. Severyn, M. Nicosia, and A. Moschitti. Learning adaptable patterns for passage reranking. *CoNLL-2013*, page 75, 2013.

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

[34] D. Shen and M. Lapata. Using semantic roles to improve question answering. In *EMNLP-CoNLL*, 2007.

[35] S. Small, T. Strzalkowski, T. Liu, S. Ryan, R. Salkin, N. Shimizu, P. Kantor, D. Kelly, and N. Wacholder. Hitiqa: Towards analytical question answering. In *COLING*, 2004.

[36] M. Surdeanu, M. Ciaramita, and H. Zaragoza. Learning to rank answers on large online QA collections. In *ACL*, 2008.

[37] K. Tymoshenko, A. Moschitti, and A. Severyn. Encoding semantic resources in syntactic structures for passage reranking. In *Proceedings of EACL*, 2014.

[38] E. M. Voorhees. Overview of the TREC 2001 Question Answering Track. In *Proceedings of TREC*, 2001.

[39] E. M. Voorhees. Overview of the trec 2003 question answering track. In *TREC 2003*, 2003.

[40] D. Wang and E. Nyberg. A long short-term memory model for answer sentence selection in question answering. In *ACL*, July 2015.

[41] M. Wang and C. D. Manning. Probabilistic tree-edit models with structured latent variables for textual entailment and question answer- ing. In *ACL*, 2010.

[42] M. Wang, N. A. Smith, and T. Mitamura. What is the jeopardy model? a quasi-synchronous grammar for qa. In *EMNLP-CoNLL*, 2007.

[43] P. C. Xuchen Yao, Benjamin Van Durme and C. Callison-Burch. Answer extraction as sequence tagging with tree edit distance. In *NAACL*, 2013.

[44] X. Yao, B. Van Durme, and P. Clark. Answer extraction as sequence tagging with tree edit distance. In *Proceedings of NAACL-HLT*, pages 858–867, 2013.

[45] W.-t. Yih, M.-W. Chang, C. Meek, and A. Pastusiak. Question answering using enhanced lexical semantic models. In *ACL*, 2013.