Encoding Semantic Resources in Syntactic Structures for Passage Reranking

Kateryna Tymoshenko
Trento RISE
38123 Povo (TN), Italy
k.tymoshenko@trentorise.eu

Alessandro Moschitti
Qatar Computing Research Instit.
5825 Doha, Qatar
amoschitti@qf.org.qa

Aliaksei Severyn
University of Trento
38123 Povo (TN), Italy
severyn@disi.unitn.it

Abstract

In this paper, we propose to use semantic knowledge from Wikipedia and large-scale structured knowledge datasets available as Linked Open Data (LOD) for the answer passage reranking task. We represent question and candidate answer passages with pairs of shallow syntactic/semantic trees, whose constituents are connected using LOD. The trees are processed by SVMs and tree kernels, which can automatically exploit tree fragments. The experiments with our SVM rank algorithm on the TREC Question Answering (QA) corpus show that the added relational information highly improves over the state of the art, e.g., about 15.4% of relative improvement in P@1.

1 Introduction

Past work in TREC QA, e.g. (Voorhees, 2001), and more recent work (Ferrucci et al., 2010) in QA has shown that, to achieve human performance, semantic resources, e.g., Wikipedia[1], must be utilized by QA systems. This requires the design of rules or machine learning features that exploit such knowledge by also satisfying syntactic constraints, e.g., the semantic type of the answer must match the question focus words. The engineering of such rules for *open domain* QA is typically very costly. For instance, for automatically deriving the correctness of the answer passage in the following question/answer passage (Q/AP) pair (from the TREC corpus[2]):

Q: *What company owns the soft drink brand "Gatorade"?*

A: *Stokely-Van Camp bought the formula and started marketing the drink as Gatorade in 1967. Quaker Oats Co. took over Stokely-Van Camp in 1983.*

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

[2] It will be our a running example for the rest of the paper.

we would need to write the following complex rules:

```
is(Quaker Oats Co.,company),
own(Stokely-Van Camp,Gatorade),
took_over(Quaker Oats Co.,Stokely-Van Camp),
took_over(Y, Z)→own(Z,Y),
```

and carry out logic unification and resolution. Therefore, approaches that can automatically generate patterns (i.e., features) from syntactic and semantic representations of the Q/AP are needed. In this respect, our previous work, e.g., (Moschitti et al., 2007; Moschitti and Quarteroni, 2008; Moschitti, 2009), has shown that tree kernels for NLP, e.g., (Moschitti, 2006), can exploit syntactic patterns for *answer passage reranking* significantly improving search engine baselines. Our more recent work, (Severyn and Moschitti, 2012; Severyn et al., 2013b; Severyn et al., 2013a), has shown that using automatically produced semantic labels in shallow syntactic trees, such as question category and question focus, can further improve passage reranking and answer extraction (Severyn and Moschitti, 2013).

However, such methods cannot solve the class of examples above as they do not use background knowledge, which is essential to answer complex questions. On the other hand, Kalyanpur et al. (2011) and Murdock et al. (2012) showed that semantic match features extracted from large-scale background knowledge sources, including the LOD ones, are beneficial for answer reranking.

In this paper, we tackle the candidate answer passage reranking task. We define kernel functions that can automatically learn structural patterns enriched by semantic knowledge, e.g., from LOD. For this purpose, we carry out the following steps: first, we design a representation for the Q/AP pair by engineering a pair of shallow syntactic trees connected with relational nodes (i.e.,

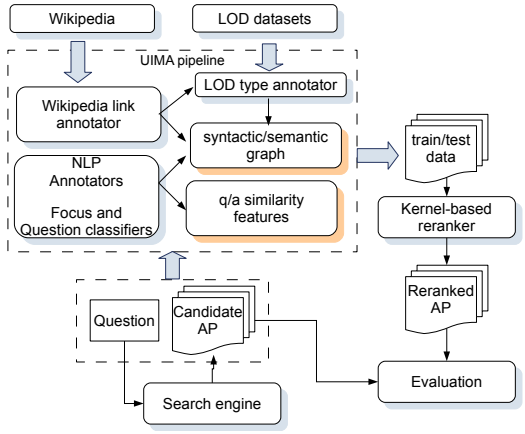

Figure 1: Kernel-based Answer Passage Reranking System

those matching the same words in the question and in the answer passages).

Secondly, we use YAGO (Suchanek et al., 2007), DBpedia (Bizer et al., 2009) and WordNet (Fellbaum, 1998) to match constituents from Q/AP pairs and use their generalizations in our syntactic/semantic structures. We employ word sense disambiguation to match the right entities in YAGO and DBpedia, and consider all senses of an ambiguous word from WordNet.

Finally, we experiment with TREC QA and several models combining traditional feature vectors with automatic semantic labels derived by statistical classifiers and relational structures enriched with LOD relations. The results show that our methods greatly improve over strong IR baseline, e.g., BM25, by 96%, and on our previous state-of-the-art reranking models, up to 15.4% (relative improvement) in P@1.

2 Reranking with Tree Kernels

In contrast to ad-hoc document retrieval, structured representation of sentences and paragraphs helps to improve question answering (Bilotti et al., 2010). Typically, rules considering syntactic and semantic properties of the question and its candidate answer are handcrafted. Their modeling is in general time-consuming and costly. In contrast, we rely on machine learning and automatic feature engineering with tree kernels. We used our state-of-the-art reranking models, i.e., (Severyn et al., 2013b; Severyn et al., 2013a) as a baseline. Our major difference with such approach is that we encode knowledge and semantics in different ways, using knowledge from LOD. The next sections outline our new kernel-based framework, although the detailed descriptions of the most innovative aspects such as new LOD-based representations are reported in Section 3.

2.1 Framework Overview

Our QA system is based on a rather simple reranking framework as displayed in Figure 1: given a question Q, a search engine retrieves a list of candidate APs ranked by their relevancy. Next, the question together with its APs are processed by a rich NLP pipeline, which performs basic tokenization, sentence splitting, lemmatization, stopword removal. Various NLP components, embedded in the pipeline as UIMA[3] annotators, perform more involved linguistic analysis, e.g., POS-tagging, chunking, NE recognition, constituency and dependency parsing, etc.

Each Q/AP 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. Given that question passages are typically short, we concatenate them with the candidate answers to provide a larger disambiguation context to the annotator.

These annotations are then used to produce computational structures (see Sec. 2.2) input to the reranker. The semantics of such relational structures can be further enriched by adding links between Q/AP constituents. Such relational links can be also generated by: (i) matching lemmas as in (Severyn and Moschitti, 2012); (ii) matching the question focus type derived by the question classifiers with the type of the target NE as in (Severyn et al., 2013a); or (iii) by matching the constituent types based on LOD (proposed in this paper). The resulting pairs of trees connected by semantic links are then used to train a kernel-based reranker, which is used to re-order the retrieved answer passages.

2.2 Relational Q/AP structures

We use the shallow tree representation that we proposed in (Severyn and Moschitti, 2012) as a baseline structural model. More in detail, each Q and its candidate AP are encoded into two trees, where lemmas constitute the leaf level, the part-of-speech (POS) tags are at the pre-terminal level and the sequences of POS tags are organized into the third level of chunk nodes. We encoded structural relations using the **REL** tag, which links the related structures in Q/AP, when there is a match

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

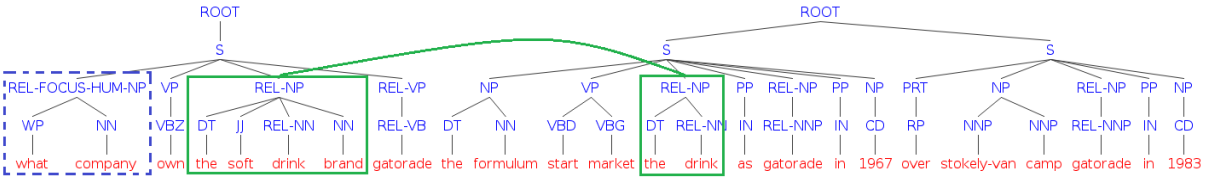

Figure 2: Basic structural representations using a shallow chunk tree structure for the Q/AP in the running example. Curved line indicates the tree fragments in the question and its answer passage linked by the relational **REL** tag.

between the lemmas in Q and AP. We marked the parent (POS tags) and grand parent (chunk) nodes of such lemmas by prepending a **REL** tag.

However, more general semantic relations, e.g., derived from the question focus and category, can be encoded using the **REL-FOCUS-**<**QC**> tag, where <**QC**> stands for the question class. In (Severyn et al., 2013b; Severyn et al., 2013a), we used statistical classifiers to derive question focus and categories of the question and of the named entities in the AP. We again mark (i) the focus chunk in the question and (ii) the AP chunks containing named entities of type compatible with the question class, by prepending the above tags to their labels. The compatibility between the categories of named entities and questions is evaluated with a lookup to a manually predefined mapping (see Table 1 in (Severyn et al., 2013b)). We also prune the trees by removing the nodes beyond a certain distance (in terms of chunk nodes) from the **REL** and *REL-FOCUS* nodes. This removes irrelevant information and speeds up learning and classification. We showed that such model outperforms bag-of-words and POS-tag sequence models (Severyn et al., 2013a).

An example of a Q/AP pair encoded using shallow chunk trees is given in Figure 2. Here, for example, the lemma "drink" occurs in both Q and AP (we highlighted it with a solid line box in the figure). "Company" was correctly recognized as a focus[4], however it was misclassified as "HUMAN" ("HUM"). As no entities of the matching type "PERSON" were found in the answer by a NER system, no chunks were marked as **REL-FOCUS** on the answer passage side.

We slightly modify the **REL-FOCUS** encoding into the tree. Instead of prepending **REL-FOCUS-**<**QC**>, we only prepend **REL-FOCUS** to the target chunk node, and add a new node *QC* as the rightmost child of the chunk node, e.g, in Figure 2, the focus node would be marked as **REL-FOCUS** and the sequence of its children would be *[WP NN HUM]*. This modification intends to reduce the feature sparsity.

[4]We used the same approach to focus detection and question classification used in (Severyn et al., 2013b)

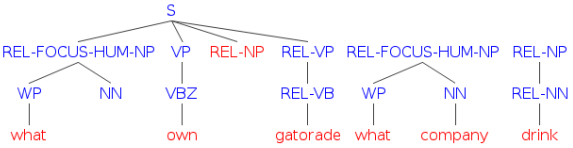

Figure 3: Some fragments (features) generated by PTK when applied to the question tree of Fig. 2

3 LOD for Semantic Structures

We aim at exploiting semantic resources for building more powerful rerankers. More specifically, we use structured knowledge about properties of the objects referred to in a Q/AP pair. A large amount of knowledge has been made available as LOD datasets, which can be used for finding additional semantic links between Q/AP passages.

In the next sections, we (i) formally define novel semantic links between Q/AP structures that we introduce in this paper; (ii) provide basic notions of Linked Open Data along with three of its most widely used datasets, YAGO, DBpedia and WordNet; and, finally, (iii) describe our algorithm to generate linked Q/AP structures.

3.1 Matching Q/AP Structures: Type Match

We look for token sequences (e.g., complex nominal groups) in Q/AP pairs that refer to entities and entity classes related by *isa* (Eq. 1) and *isSubclassOf* (Eq. 2) relations and then link them in the structural Q/AP representations.

$$isa : entity \times class \rightarrow \{true, false\} \quad (1)$$

$$isSubclassOf : class \times class \rightarrow \{true, false\} \quad (2)$$

Here, entities are all the objects in the world both real or abstract, while classes are sets of entities that share some common features. Information about entities, classes and their relations can be obtained from the external knowledge sources such as the LOD resources. *isa* returns *true* if an entity is an element of a class (*false* otherwise), while *isSubclassOf(class1,class2)* returns *true* if all elements of *class1* belong also to *class2*.

We refer to the token sequences introduced above as to *anchors* and the entities/classes they refer to as *references*. We define anchors to be in a Type Match (TM) relation if the entities/classes

they refer to are in *isa* or *isSubclassOf* relation. More formally, given two anchors a_1 and a_2 belonging to two text passages, p_1 and p_2, respectively, and given an $R(a, p)$ function, which returns a reference of an anchor *a* in passage *p*, we define $TM\,(r_1, r_2)$ as

$$\begin{cases} isa\,(r_1, r_2) : if\ isEntity\,(r_1) \wedge isClass\,(r_2) \\ subClassOf\,(r_1, r_2) : if\ isClass\,(r_1) \wedge isClass\,(r_2) \end{cases} \quad (3)$$

where $r_1 = R(a_1, p_1)$, $r_2 = R(a_2, p_2)$ and *isEntity(r)* and *isClass(r)* return true if *r* is an entity or a class, respectively, and *false* otherwise. It should be noted that, due to the ambiguity of natural language, the same anchor may have different references depending on the context.

3.2 LOD for linking Q/A structures

LOD consists of datasets published online according to the Linked Data (LD) principles[5] and available in open access. LOD knowledge is represented following the Resource Description Framework (RDF)[6] specification as a set of *statements*. A statement is a *subject-predicate-object* triple, where predicate denotes the directed relation, e.g., *hasSurname* or *owns*, between subject and object. Each object described by RDF, e.g., a class or an entity, is called a resource and is assigned a Unique Resource Identifier (URI).

LOD includes a number of common schemas, i.e., sets of classes and predicates to be reused when describing knowledge. For example, one of them is RDF Schema (RDFS)[7], which contains predicates `rdf:type` and `rdfs:SubClassOf` similar to the *isa* and *subClassOf* functions above. LOD contains a number of large-scale cross-domain datasets, e.g., YAGO (Suchanek et al., 2007) and DBpedia (Bizer et al., 2009). Datasets created before the emergence of LD, e.g., WordNet, are brought into correspondence with the LD principles and added to the LOD as well.

3.2.1 Algorithm for detecting TM

Algorithm 1 detects n-grams in the Q/AP structures that are in TM relation and encodes TM knowledge in the shallow chunk tree representations of Q/AP pairs. It takes two text passages, P_1 and P_2, and a LOD knowledge source, LOD_{KS}, as input. We run the algorithm twice, first with AP as P_1 and Q as P_2 and then vice versa. For example, P_1 and P_2 in the first run could be, according to our running example, Q and AP candidate, respectively, and LOD_{KS} could be YAGO, DBpedia or WordNet.

Algorithm 1 Type Match algorithm

Input: P_1, P_2 - text passages; LOD_{KS} - LOD knowledge source.

```
1: for all anchor ∈ getAnchors(P2,LOD_KS) do
2:     for all uri ∈ getURIs(anchor,P2,LOD_KS) do
3:         for all type ∈ getTypes(uri,LOD_KS) do
4:             for all ch ∈ getChunks(P1) do
5:                 matchedTokens ← checkMatch(ch, type.labels)
6:                 if matchedTokens ≠ ∅ then
7:                     markAsTM(anchor,P2.parseTree)
8:                     markAsTM(matchedTokens, P1.parseTree)
```

Detecting anchors. $getAnchors(P_2, LOD_{KS})$ in line 1 of Algorithm 1 returns all anchors in the given text passage, P_2. Depending on LOD_{KS} one may have various implementations of this procedure. For example, when LOD_{KS} is WordNet, *getAnchor* returns token subsequences of the chunks in P_2 of lengths *n-k*, where n is the number of tokens in the chunk and $k = [1, .., n)$.

In case when LOD_{KS} is YAGO or DBpedia, we benefit from the fact that both YAGO and DBpedia are aligned with Wikipedia on entity level by construction and we can use the so-called *wikification* tools, e.g., (Milne and Witten, 2009), to detect the anchors. The *wikification* tools recognize n-grams that may denote Wikipedia pages in plain text and disambiguate them to obtain a unique Wikipedia page. Such tools determine whether a certain n-gram may denote a Wikipedia page(s) by looking it up in a precomputed vocabulary created using Wikipedia page titles and internal link network (Csomai and Mihalcea, 2008; Milne and Witten, 2009).

Obtaining references. In line 2 of Algorithm 1 for each anchor, we determine the URIs of entities/classes it refers to in LOD_{KS}. Here again, we have different strategies for different LOD_{KS}. In case of WordNet, we use the all-senses strategy, i.e., getURI procedure returns a set of URIs of synsets that contain the anchor lemma.

In case when LOD_{KS} is YAGO or DBpedia, we use *wikification* tools to correctly disambiguate an anchor to a Wikipedia page. Then, Wikipedia page URLs may be converted to DBpedia URIs by substituting the `en.wikipedia.org/wiki/` prefix to the `dbpedia.org/resource/`; and YAGO URIs by querying it for subjects of the

[5] `http://www.w3.org/DesignIssues/LinkedData.html`

[6] `http://www.w3.org/TR/rdf-concepts/`

[7] `http://www.w3.org/TR/rdf-schema/`

RDF triples with `yago:hasWikipediaUrl`[8] as a predicate and Wikipedia URL as an object.

For instance, one of the anchors detected in the running example AP would be "Quaker oats", a wikification tool would map it to `wiki:Quaker_Oats_Company`[9], and the respective YAGO URI would be `yago:Quaker_Oats_Company`.

Obtaining type information. Given a *uri*, if it is an entity, we look for all the classes it belongs to, or if it is a class, we look for all classes for which it is a subclass. This process is incorporated in the *getTypes* procedure in line 3 of Algorithm 1. We call such classes *types*. If LOD_{KS} is WordNet, then our types are simply the URIs of the hypernyms of *uri*. If LOD_{KS} is DBpedia or YAGO, we query these datasets for the values of the `rdf:type` and `rdfs:subClassOf` properties of the *uri* (i.e., objects of the triples with *uri* as subject and *type/subClassOf* as predicates) and add their values (which are also URIs) to the *types* set. Then, we recursively repeat the same queries for each retrieved type URI and add their results to the *types*. Finally, the *getTypes* procedure returns the resulting *types* set.

The extracted URIs returned by *getTypes* are HTTP ids, however, frequently they have human-readable names, or labels, specified by the `rdfs:label` property. If no label information for a URI is available, we can create the label by removing the technical information from the type URI, e.g., *http* prefix and underscores. *type.labels* denotes a set of type human-readable labels for a specific *type*. For example, one of the types extracted for `yago:Quaker_Oats_Company` would have label "company".

Checking for TM. Further, the *checkMatch* procedure checks whether any of the labels in the *type.label* matches any of the chunks in P_1 returned by *getChunks*, fully or partially (line 5 of Algorithm 1). Here, *getChunks* procedure returns a list of chunks recognized in P_1 by an external chunker.

More specifically, given a chunk, ch, and a type label, $type.label$, *checkMatch* checks whether the ch string matches[10] $type.label$ or its last word(s). If no match is observed, we remove the first token from ch and repeat the procedure. We stop when the match is observed or when no tokens in ch are left. If the match is observed *checkMatch* returns all the tokens, remaining in ch as $matchedTokens$. Otherwise, it returns an empty set. For example, the question of the running example contains the chunk "what company", which partially matches the human readable "company" label of one of the types retrieved for the "Quaker oats" anchor from the answer. Our implementation of the $checkMatch$ procedure would return "company" from the question as one of the *matchedTokens*.

If the $matchedTokens$ set is not empty, this means that $TM\big(R\big(anchor, P_2\big), R\big(matchedTokens, P_1\big)\big)$ in Eq. 3 returns *true*. Indeed, a_1 is an $anchor$ and a_2 is the *matchedTokens* sequence (see Eq. 3), and their respective references, i.e., URI assigned to the anchor and URI of one of its types, are either in *subClassOf* or in *isa* relation by construction. Naturally, this is only one of the possible ways to evaluate the TM function, and it may be noise-prone.

Marking TM in tree structures. Finally, if the TM match is observed, i.e., *matchedTokens* is not an empty set, we mark tree substructures corresponding to the anchor in the structural representation of P_2 ($P_2.parseTree$) and those corresponding to *matchedTokens* in that of P_1 ($P_1.parseTree$) as being in a TM relation. In our running example, we would mark the substructures corresponding to "Quaker oats" anchor in the answer (our P_2) and the "company" *matchedToken* in the question (our P_1) shallow syntactic tree representations. We can encode TM match information into a tree in a variety of ways, which we describe below.

3.2.2 Encoding TM knowledge in the trees

a_1 and a_2 from Eq. 3 are n-grams, therefore they correspond to the leaf nodes in the shallow syntactic trees of p_1 and p_2. We denote the set of their preterminal parents as N_{TM}. We considered the following strategies of encoding TM relation in the trees: (i) **TM node** (TM_N). Add leaf sibling tagged with *TM* to all the nodes in N_{TM}. (ii) **Directed TM node** (TM_{ND}). Add leaf sibling tagged with **TM-CHILD** to all the nodes in N_{TM} corresponding to the anchor, and leaf siblings tagged with **TM-PARENT** to the nodes corresponding to *matchedTokens*. (iii) **Focus** TM (TM_{NF}). Add leaf siblings to all the nodes in

[8] `yago:` is a shorthand for the http prefix `http://yago-knowledge.org/resource/`

[9] `wiki:` is a shorthand for the http prefix `http://en.wikipedia.org/wiki/`

[10] case-insensitive exact string match

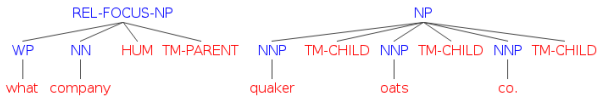

Figure 4: Fragments of a shallow chunk parse tree annotated in TM_{ND} mode.

N_{TM}. If *matchedTokens* is a part of a question focus label them as **TM-FOCUS**. Otherwise, label them as **TM**. (iv) **Combo** TM_{NDF}. Encode using the TM_{ND} strategy. If *matchedTokens* is a part of a question focus label then also add a child labeled *FOCUS* to each of the TM labels. Intuitively, $TM_{ND}, TM_{NF}, TM_{NDF}$ are likely to result in more expressive patterns. Fig. 4 shows an example of the TM_{ND} annotation.

3.3 Wikipedia-based matching

Lemma matching for detecting **REL** may result in low coverage, e.g., it is not able to match different variants for the same name. We remedy this by using Wikipedia link annotation. We consider two word sequences (in Q and AP, respectively) that are annotated with the same Wikipedia link to be in a matching relation. Thus, we add new **REL** tags to Q/AP structural representations as described in Sec. 2.2.

4 Experiments

We evaluated our different rerankers encoding several semantic structures on passage retrieval task, using a factoid open-domain TREC QA corpus.

4.1 Experimental Setup

TREC QA 2002/2003. In our experiments, we opted for questions from years 2002 and 2003, which totals to 824 factoid questions. The AQUAINT corpus[11] is used for searching the supporting passages.

Pruning. Following (Severyn and Moschitti, 2012) we prune the shallow trees by removing the nodes beyond distance of 2 from the **REL**, **REL-FOCUS** or **TM** nodes.

LOD datasets. We used the core RDF distribution of YAGO2[12], WordNet 3.0 in RDF[13], and the datasets from the 3.9 DBpedia distribution[14].

Feature Vectors. We used a subset of the similarity functions between Q and AP described in (Severyn et al., 2013b). These are used along with the structural models. More explicitly: *Term-overlap features:*, i.e., a cosine similarity over question/answer, $sim_{COS}(Q, AP)$, where the input vectors are composed of lemma or POS-tag n-grams with $n = 1, .., 4$. *PTK score:* i.e., the PTK output, when applied to the structural representations of Q and AP: $sim_{PTK}(Q, AP) = PTK(Q, AP)$ (note that, this is computed within a pair). *Search engine ranking score:* the ranking score of our search engine assigned to AP divided by a normalizing factor.

SVM re-ranker. To train our models, we use SVM-light-TK[15], which enables the use of structural kernels (Moschitti, 2006) in SVM-light (Joachims, 2002). We use default parameters and the preference reranking model described in (Severyn and Moschitti, 2012; Severyn et al., 2013b). We used PTK 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 to perform sentence detection, tokenization, NE Recognition, parsing, chunking and lemmatization. Moreover, we used annotators for building new sentence representations starting from tools' annotations and classifiers for question focus and question class.

Search engines. We adopted Terrier[16] using the accurate BM25 scoring model with default parameters. We trained it on the TREC corpus (3Gb), containing about 1 million documents. We performed indexing at the paragraph level by splitting each document into a set of paragraphs, which are then added to the search index. We retrieve a list of 50 candidate answer passages for each question.

Wikipedia link annotators. We use the Wikipedia Miner (WM) (Milne and Witten, 2009)[17] tool and the Machine Linking (ML)[18] web-service to annotate Q/AP pairs with links to Wikipedia. Both tools output annotation confidence. We use all WM and ML annotations with

[11] http://catalog.ldc.upenn.edu/LDC2002T31

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

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

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

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

[16] http://terrier.org

[17] http://sourceforge.net/projects/wikipedia-miner/files/wikipedia-miner/wikipedia-miner_1.1, we use only topic detector module which detects and disambiguates anchors

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

System	MRR	MAP	P@1
BM25	28.02±2.94	0.22±0.02	18.17±3.79
CH+V (CoNLL, 2013)	37.45	0.3	27.91
CH+V+QC+TFC (CoNLL, 2013)	39.49	0.32	30
CH + V	36.82±2.68	0.30±0.02	26.34±2.17
CH + V+ QC+TFC	40.20±1.84	0.33±0.01	30.85±2.35
CH+V+QC+TFC*	40.50±2.32	0.33±0.02	31.46±2.42

Table 1: Baseline systems

confidence exceeding 0.2 and 0.05, respectively. We obtained these figures heuristically, they are low because we aimed to maximize the Recall of the Wikipedia link annotators in order to maximize the number of TMs. In all the experiments, we used a union of the set of the annotations provided by WM and ML.

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). We perform 5-fold cross-validation and report the metrics averaged across all the folds together with the std.dev.

4.2 Baseline Structural Reranking

In these experiments, we evaluated the accuracy of the following baseline models: **BM25** is the BM25 scoring model, which also provides the initial ranking; **CH+V** is a combination of tree structures encoding Q/AP pairs using relational links with the feature vector; and **CH+V+QC+TFC** is **CH+V** extended with the semantic categorial links introduced in (Severyn et al., 2013b).

Table 1 reports the performance of our baseline systems. The lines marked with *(CoNLL, 2013)* contain the results we reported in (Severyn et al., 2013b). Lines four and five report the performance of the same systems, i.e., **CH+V** and **CH+V+QC+TFC**, after small improvement and changes. Note that in our last version, we have a different set of **V** features than in *(CoNLL, 2013)*. Finally, **CH+V+QC+TFC*** refers to the performance of **CH+V+QC+TFC** with question type information of semantic **REL-FOCUS** links represented as a distinct node (see Section 2.2). The results show that this modification yields a slight improvement over the baseline, thus, in the next experiments, we add LOD knowledge to **CH+V+QC+TFC***.

4.3 Impact of LOD in Semantic Structures

These experiments evaluated the accuracy of the following models (described in the previous sections): (i) a system using Wikipedia to establish the REL links; and (ii) systems which use LOD knowledge to find type matches (TM).

The first header line of the Table 2 shows which baseline system was enriched with the TM knowledge. *Type* column reports the TM encoding strategy employed (see Section 3.2.2). *Dataset* column reports which knowledge source was employed to find TM relations. Here, *yago* is YAGO2, *db* is DBpedia, and *wn* is WordNet 3.0. The first result line in Table 2 reports the performance of the strong **CH+V** and **CH+V+QC+TFC*** baseline systems. Line with the "wiki" dataset reports on **CH+V** and **CH+V+QC+TFC*** using both Wikipedia link annotations provided by ML and MW and hard lemma matching to find the related structures to be marked by **REL** (see Section 3.3 for details of the Wikipedia-based REL matching). The remainder of the systems is built on top of the baselines using both hard lemma and Wikipedia-based matching. We used bold font to mark the top scores for each encoding strategy.

The tables show that all the systems exploiting LOD knowledge, excluding those using DBpedia only, outperform the strong **CH+V** and **CH+V+QC+TFC*** baselines. Note that **CH+V** enriched with TM tags performs comparably to, and in some cases even outperforms, **CH+V+QC+TFC***. Compare, for example, the outputs of **CH+V+**TM_{NDF} using YAGO, WordNet and DBpedia knowledge and those of **CH+V+QC+TFC*** with no LOD knowledge.

Adding TM tags to the top-performing baseline system, **CH+V+QC+TFC***, typically results in further increase in performance. The best-performing system in terms of MRR and P@1 is **CH+V+QC+TFC*+**TM_{NF} system using the combination of WordNet and YAGO2 as source of TM knowledge and Wikipedia for REL-matching. It outperforms the **CH+V+QC+TFC*** baseline by 3.82% and 4.15% in terms of MRR and P@1, respectively. Regarding MAP, a number of systems employing YAGO2 in combination with WordNet and Wikipedia-based REL-matching obtain 0.37 MAP score thus outperforming the **CH+V+QC+TFC*** baseline by 4%.

We used paired two-tailed t-test for evaluating the statistical significance of the results reported in Table 2. ‡ and † correspond to the significance levels of 0.05 and 0.1, respectively. We compared (i) the results in the *wiki* line to those in the *none* line; and (ii) the results for the *TM* systems to those in the *wiki* line.

Type	Dataset	CH + V			CH + V + QC + TFC*		
		MRR	**MAP**	**P@1**	**MRR**	**MAP**	**P@1**
-	none	36.82±2.68	0.30±0.02	26.34±2.17	40.50±2.32	0.33±0.02	31.46±2.42
-	wiki	39.17±1.29‡	0.31±0.01‡	28.66±1.43‡	41.33±1.17	0.34±0.01	31.46±1.40
TM_N	db	40.60±1.88	0.33±0.01‡	31.10±2.99†	40.80±1.01	0.34±0.01	30.37±1.90
TM_N	wn	41.39±1.96‡	0.33±0.01‡	31.34±2.94	42.43±0.56	0.35±0.01	32.80±0.67
TM_N	wn+db	40.85±1.52‡	0.33±0.01‡	30.37±2.34	42.37±1.12	0.35±0.01	32.44±2.64
TM_N	yago	40.71±2.07	0.33±0.03†	30.24±2.09‡	43.28±1.91†	**0.36±0.01**†	33.90±2.75
TM_N	yago+db	41.25±1.57‡	0.34±0.02‡	31.10±1.88‡	42.39±1.83	0.35±0.01	32.93±3.14
TM_N	yago+wn	**42.01±2.26**‡	**0.34±0.02**‡	**32.07±3.04**‡	**43.98±1.08**‡	**0.36±0.01**‡	**35.24±1.46**‡
TM_N	yago+wn+db	41.52±1.85‡	**0.34±0.02**‡	30.98±2.71‡	43.13±1.38	**0.36±0.01**	33.66±2.77
TM_{NF}	db	40.67±1.94†	0.33±0.01‡	30.85±2.22†	41.43±0.70	0.35±0.01	31.22±1.09
TM_{NF}	wn	40.95±2.27‡	0.33±0.01‡	30.98±3.74	42.37±0.98	0.35±0.01	32.56±1.76
TM_{NF}	wn+db	40.84±2.18†	**0.34±0.01**‡	30.73±3.04	43.08±0.83†	0.36±0.01†	33.54±1.29†
TM_{NF}	yago	**42.01±2.44**†	**0.34±0.02**‡	**32.07±3.01**‡	43.82±2.36†	0.36±0.02‡	34.88±3.35
TM_{NF}	yago+db	41.32±1.70†	**0.34±0.02**‡	31.10±2.48‡	43.19±1.17†	0.36±0.01†	33.90±1.86
TM_{NF}	yago+wn	41.69±1.66‡	**0.34±0.02**‡	31.10±2.44‡	**44.32±0.70**‡	0.36±0.01‡	**35.61±1.11**‡
TM_{NF}	yago+wn+db	41.56±1.41‡	**0.34±0.02**‡	30.85±2.22†	43.79±0.73‡	**0.37±0.01**†	34.88±1.69‡
TM_{ND}	db	40.37±1.87	0.33±0.01‡	30.37±2.17	41.58±1.02	0.35±0.01†	31.46±1.59
TM_{ND}	wn	41.13±2.14‡	0.33±0.01‡	30.73±2.75	42.19±1.39	0.35±0.01	32.32±1.36
TM_{ND}	wn+db	41.28±1.03‡	0.34±0.01‡	30.73±0.82‡	42.37±1.16	0.36±0.01	32.44±2.71
TM_{ND}	yago	42.11±3.24‡	0.34±0.02‡	32.07±4.06†	44.04±2.05‡	0.36±0.01‡	34.63±2.17‡
TM_{ND}	yago+db	42.28±2.01‡	**0.35±0.01**‡	32.44±1.99‡	43.77±2.02†	**0.37±0.01**†	34.27±2.42
TM_{ND}	yago+wn	**42.96±1.45**‡	**0.35±0.01**‡	**33.05±2.04**‡	**44.25±1.32**‡	**0.37±0.00**‡	**34.76±1.61**‡
TM_{ND}	yago+wn+db	42.56±1.25‡	**0.35±0.01**‡	32.56±1.91‡	43.91±1.01‡	**0.37±0.01**†	34.63±1.32‡
TM_{NDF}	db	40.40±1.93†	0.33±0.01‡	30.49±1.78‡	41.85±1.05	0.35±0.01‡	31.83±0.80
TM_{NDF}	wn	40.84±1.69‡	0.33±0.01‡	30.49±2.24	41.89±0.99	0.35±0.01	31.71±0.86
TM_{NDF}	wn+db	41.14±1.29‡	0.34±0.01‡	30.73±1.40‡	42.31±0.92	0.36±0.01	32.32±2.36
TM_{NDF}	yago	42.31±2.57‡	**0.35±0.02**‡	32.68±3.01‡	**44.22±2.38**‡	**0.37±0.02**‡	**35.00±2.88**‡
TM_{NDF}	yago+db	41.96±1.82‡	**0.35±0.01**‡	32.32±2.24‡	43.82±1.95‡	**0.37±0.01**‡	34.51±2.39†
TM_{NDF}	yago+wn	42.80±1.19‡	**0.35±0.01**‡	33.17±1.86‡	43.91±0.98‡	**0.37±0.01**‡	34.63±0.90‡
TM_{NDF}	yago+wn+db	**43.15±0.93**‡	**0.35±0.01**‡	**33.78±1.59**‡	43.96±0.94‡	**0.37±0.01**‡	34.88±1.69‡

Table 2: Results in 5-fold cross-validation on TREC QA corpus

The table shows 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. DBpedia ontology that we employed in the *db* 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[19]. Note that in (Severyn and Moschitti, 2012), we also used supersenses of WordNet (unsuccessfully) whereas here we use hypernymy relations and a different technique to incorporate semantic match into the tree structures.

Different TM-knowledge encoding strategies, TM_N, TM_{ND}, TM_{NF}, TM_{NDF} produce small changes in accuracy. We believe, that the difference between them would become more significant when experimenting with larger corpora.

5 Conclusions

This paper proposes syntactic structures whose nodes are enriched with semantic information from statistical classifiers and knowledge from LOD. In particular, YAGO, DBpedia and WordNet are used to match and generalize constituents from QA pairs: such matches are then used in syntactic/semantic structures. The experiments with TREC QA and the above representations also combined with traditional features greatly improve over a strong IR baseline, e.g., 96% on BM25, and on previous state-of-the-art reranking models, up to 15.4% (relative improvement) in P@1. In particular, differently from previous work, our models can effectively use semantic knowledge in statistical learning to rank methods. These promising results open interesting future directions in designing novel semantic structures and using innovative semantic representations in learning algorithms.

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

Acknowledgments

This research is partially supported by the EU's 7th Framework Program (FP7/2007-2013) (#288024 LIMOSINE project) and by a Shared University Research award from the IBM Watson Research Center - Yorktown Heights, USA and the IBM Center for Advanced Studies of Trento, Italy. The third author is supported by the Google Europe Fellowship 2013 award in Machine Learning.

References

Matthew W. Bilotti, Jonathan L. Elsas, Jaime Carbonell, and Eric Nyberg. 2010. Rank learning for factoid question answering with linguistic and semantic constraints. In *Proceedings of the 19th ACM international Conference on Information and Knowledge Management (CIKM)*, pages 459–468.

Christian Bizer, Jens Lehmann, Georgi Kobilarov, Sören Auer, Christian Becker, Richard Cyganiak, and Sebastian Hellmann. 2009. Dbpedia - a crystallization point for the web of data. *Web Semantics: Science, Services and Agents on the World Wide Web*, 7(3):154–165, September.

Andras Csomai and Rada Mihalcea. 2008. Linking documents to encyclopedic knowledge. *IEEE Intelligent Systems*, 23(5):34–41.

Christiane Fellbaum, editor. 1998. *WordNet: An Electronic Lexical Database*. MIT Press.

David Ferrucci, Eric Brown, Jennifer Chu-Carroll, James Fan, David Gondek, Aditya Kalyanpur, Adam Lally, J. William Murdock, Eric Nyberg, John Prager, Nico Schlaefer, and Chris Welty. 2010. Building watson: An overview of the deepqa project. *AI Magazine*, 31(3).

Thorsten Joachims. 2002. Optimizing search engines using clickthrough data. In *ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD)*, pages 133–142. ACM.

Aditya Kalyanpur, J William Murdock, James Fan, and Christopher Welty. 2011. Leveraging community-built knowledge for type coercion in question answering. In *The Semantic Web–ISWC 2011*, pages 144–156. Springer.

David Milne and Ian H Witten. 2009. An open-source toolkit for mining wikipedia. In *New Zealand Computer Science Research Student Conference (NZCSRSC)*.

Alessandro Moschitti and Silvia Quarteroni. 2008. Kernels on linguistic structures for answer extraction. In *Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics on Human Language Technologies: Short Papers (ACL)*, pages 113–116.

Alessandro Moschitti, Silvia Quarteroni, Roberto Basili, and Suresh Manandhar. 2007. Exploiting syntactic and shallow semantic kernels for question/answer classification. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics (ACL)*, pages 776–783.

Alessandro Moschitti. 2006. Efficient convolution kernels for dependency and constituent syntactic trees. In *ECML*, pages 318–329. Springer.

Alessandro Moschitti. 2009. Syntactic and semantic kernels for short text pair categorization. In *Proceedings of the 12th Conference of the European Chapter of the Association for Computational Linguistics (EACL)*, pages 576–584. Association for Computational Linguistics.

J William Murdock, Aditya Kalyanpur, Chris Welty, James Fan, David A Ferrucci, DC Gondek, Lei Zhang, and Hiroshi Kanayama. 2012. Typing candidate answers using type coercion. *IBM Journal of Research and Development*, 56(3.4):7–1.

Aliaksei Severyn and Alessandro Moschitti. 2012. Structural relationships for large-scale learning of answer re-ranking. In *Proceedings of the 35th international ACM SIGIR conference on Research and development in information retrieval (SIGIR)*, pages 741–750. ACM.

Aliaksei Severyn and Alessandro Moschitti. 2013. Automatic feature engineering for answer selection and extraction. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 458–467.

Aliaksei Severyn, Massimo Nicosia, and Alessandro Moschitti. 2013a. Building structures from classifiers for passage reranking. In *Proceedings of the 22nd ACM international conference on Conference on information & knowledge management (CIKM)*, pages 969–978. ACM.

Aliaksei Severyn, Massimo Nicosia, and Alessandro Moschitti. 2013b. Learning adaptable patterns for passage reranking. In *Proceedings of the Seventeenth Conference on Computational Natural Language Learning (CoNLL)*.

Fabian M. Suchanek, Gjergji Kasneci, and Gerhard Weikum. 2007. Yago: a core of semantic knowledge. In *Proceedings of the 16th international conference on World Wide Web (WWW)*, pages 697–706. ACM Press.

Ellen M Voorhees. 2001. Overview of the TREC 2001 Question Answering Track. In *Proceedings of TREC*, pages 42–51.