RelTextRank: An Open Source Framework for Building Relational Syntactic-Semantic Text Pair Representations

Kateryna Tymoshenko[†] and **Alessandro Moschitti** and **Massimo Nicosia**[†] and **Aliaksei Severyn**[†]
[†]DISI, University of Trento, 38123 Povo (TN), Italy
Qatar Computing Research Institute, HBKU, 34110, Doha, Qatar
{kateryna.tymoshenko,massimo.nicosia}@unitn.it
{amoschitti,aseveryn}@gmail.com

Abstract

We present a highly-flexible UIMA-based pipeline for developing structural kernel-based systems for relational learning from text, i.e., for generating training and test data for ranking, classifying short text pairs or measuring similarity between pieces of text. For example, the proposed pipeline can represent an input question and answer sentence pairs as syntactic-semantic structures, enriching them with relational information, e.g., links between question class, focus and named entities, and serializes them as training and test files for the tree kernel-based reranking framework. The pipeline generates a number of dependency and shallow chunk-based representations shown to achieve competitive results in previous work. It also enables easy evaluation of the models thanks to cross-validation facilities.

1 Introduction

A number of recent works (Severyn et al., 2013; Tymoshenko et al., 2016b,a; Tymoshenko and Moschitti, 2015) show that tree kernel methods produce state-of-the-art results in many different relational tasks, e.g., Textual Entailment Recognition, Paraphrasing, question, answer and comment ranking, when applied to syntactico-semantic representations of the text pairs.

In this paper, we describe **RelTextRank**, a flexible Java pipeline for converting pairs of raw texts into structured representations and enriching them with semantic information about the relations between the two pieces of text (e.g., lexical exact match). The pipeline is based on the Apache UIMA technology[1], which allows for the creation of highly modular applications and analysis of large volumes of unstructured information.

RelTextRank is an open-source tool available at https://github.com/iKernels/RelTextRank. It contains a number of generators for shallow and dependency-based structural representations, UIMA wrappers for multi-purpose linguistic annotators, e.g., Stanford CoreNLP (Manning et al., 2014), question classification and question focus detection modules, and a number of similarity feature vector extractors. It allows for: (i) setting experiments with the new structures, also introducing new types of relational links; (ii) generating training and test data both for kernel-based classification and reranking, also in a cross-validation setting; and (iii) generating predictions using a pre-trained classifier.

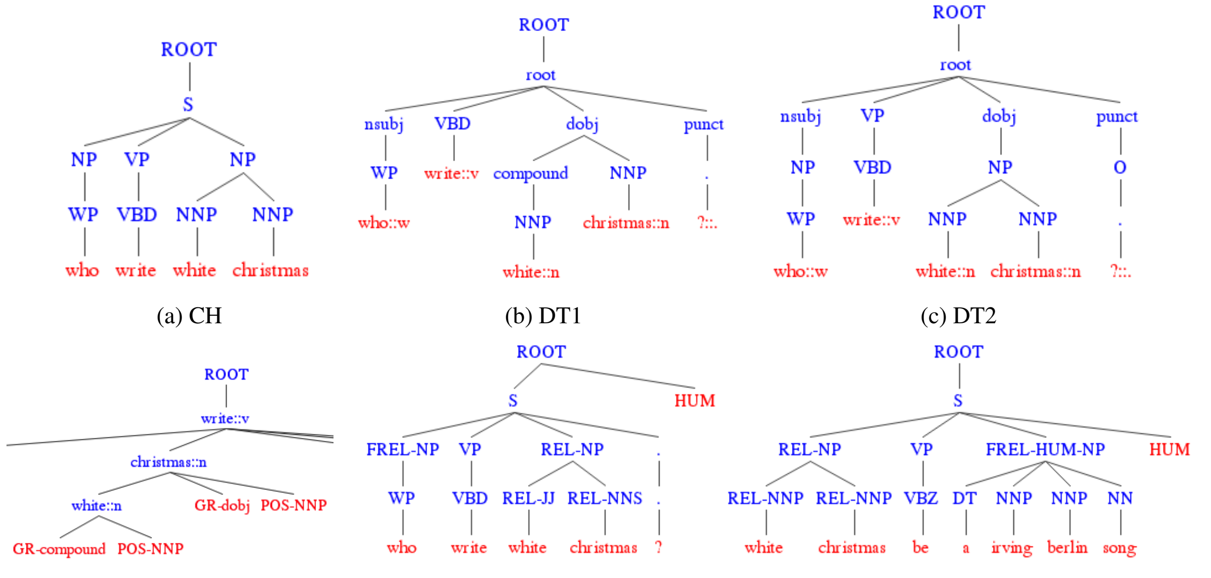
In the remainder of the paper, we describe the structures that can be generated by the system (Sec 2), the overall RelTextRank architecture (Sec 3) and the specific implementation of its components (Sec 4). Then, we provide some examples of how to run the pipeline from the command line (Sec 5)[2]. Finally, in Sec 6, we report some results using earlier versions of RelTextRank.

2 Background

Recent work in text pair reranking and classification, e.g., answer sentence selection (AS) and community question answering (cQA), has studied a number of structures for representing text pairs along with their relational links, which provide competitive results when used in a standalone system and the state of the art when combined with feature vectors (Severyn et al., 2013; Tymoshenko et al., 2016b; Tymoshenko and Moschitti, 2015) and embeddings learned by the neural networks (Tymoshenko et al., 2016a). In this sec-

[1] https://uima.apache.org/

[2] The detailed documentation is available on the related GitHub project.



(e) CH^p structure enriched with the relational labels

Figure 1: Structures generated by *RelTextRank*

tion, we provide an overview of the structures and relational links that can be generated by RelTextRank (specific details are in the above papers).

2.1 RelTextRank Structures

RelTextRank can generate the following structural representations for a text pair $\langle T_1, T_2 \rangle$, where, e.g., T_1 can be a question and T_2 a candidate answer passage for an AS task.

Shallow pos-chunk tree representation (CH and CH^p). Here, T_1 and T_2 are both represented as shallow tree structures with lemmas as leaves, their POS-tags as their parent nodes. The POS-tag nodes are further grouped under chunk and sentence nodes. We provide two versions of this structure: CH^p (Fig. 1e), which keeps all information while CH (Fig. 1a) excludes punctuation marks and words outside of any chunk.

Constituency tree representation (CONST). Standard constituency tree representation.

Dependency tree representations. We provide three dependency-based relational structures: DT1, DT2 and LCT. DT1 (Fig. 1b) is a dependency tree in which grammatical relations become nodes and lemmas are located at the leaf level. DT2 (Fig. 1c) is DT1 modified to include the chunking information, and lemmas in the same chunk are grouped under the same chunk node. Finally, LCT (Fig. 1d) is a lexical-centered dependency tree (Croce et al., 2011) with the grammatical relation $REL(head, child)$ represented as *(head (child **GR**-REL **POS**-pos(head))*. Here REL is a grammatical relation, *head* and *child* are the head and child lemmas in the relation, respectively, and *pos (head)* is the POS-tag of the head lemma. *GR*- and *POS*- tags in the node name indicate that the node is a grammar relation or part-of-speech node, respectively.

2.2 RelTextRank relational links

Experimental results on multiple datasets (Severyn et al., 2013; Severyn and Moschitti, 2012) show that encoding information about the relations between T_1 and T_2 is crucial for obtaining the state of the art in text pair ranking and classification. RelTextRank provides two kinds of links: hard string match, `REL`, and semantic match, `FREL`.

REL. If some lemma occurs both in T_1 and T_2, we mark the respective POS-tag nodes and their parents in the structural representations of T_1 and T_2 with `REL` labels for all the structures, except for LCT. In LCT, we mark with `REL-` the POS (`POS-`) and grammar relation (`GR-`) nodes.

FREL. When working in the question answering setting, i.e., when T_1 is a question and T_2 is a candidate answer passage, we encode the question focus and question class information into the structural representations. We use `FREL-<QC>` tag to mark question focus in T_1. Then, in T_2, we mark all the named entities T_2 of type compatible with the question class[3]. Here, `<QC>` is substituted

[3]We use the following mappings to check for compatibility(*Stanford named entity type* → *UIUC question class (Li and Roth, 2002)*): Person, Organization → HUM ,ENTY; Misc → ENTY; Location →LOC; Date, Time, Money, Percentage, Set, Duration →NUM

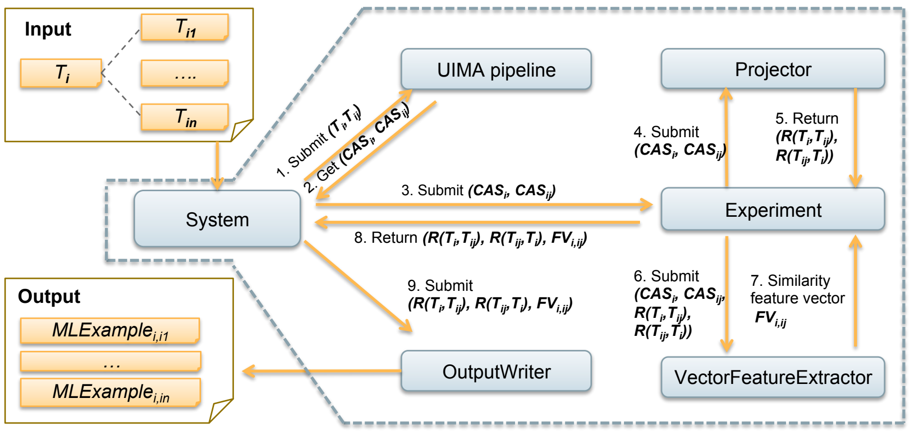

Input
T_i
T_{i1}
….
T_{in}
UIMA pipeline
Projector
1. Submit (T_i, T_{ij})
2. Get (CAS_i, CAS_{ij})
4. Submit (CAS_i, CAS_{ij})
5. Return ($R(T_i,T_{ij})$, $R(T_{ij},T_i)$)
3. Submit (CAS_i, CAS_{ij})
System
Experiment
8. Return ($R(T_i,T_{ij})$, $R(T_{ij},T_i)$, $FV_{i,ij}$)
Output
$MLExample_{i,i1}$
…
$MLExample_{i,in}$
9. Submit ($R(T_i,T_{ij})$, $R(T_{ij},T_i)$, $FV_{i,ij}$)
6. Submit (CAS_i, CAS_{ij}, $R(T_i,T_{ij})$, $R(T_{ij},T_i)$)
7. Similarity feature vector $FV_{i,ij}$
OutputWriter
VectorFeatureExtractor

Figure 2: Overall framework

with an actual question class. For all structures in Sec. 2.1, except LCT, we prepend the `FREL-<QC>` tag to the grand-parents of the lemma nodes. In case of LCT, we mark the child grammatical (`GR-`) and POS-tag (`POS-`) nodes of the lemma nodes. Finally, only in case of CH^p structure we also add `<QC>` to the `ROOT` tree node as its rightmost child, and mark the question focus node simply as `FREL`. Fig. 1e shows an example of a CH^p enriched with `REL` and `FREL` links.[4]

3 System Architecture

RelTextRank has a modular architecture easy to adjust towards new structures, features and relational links (see Fig. 2). The basic input of *RelTextRank* is a text T_i, and a list of n texts, $T_{i1}, ..., T_{in}$ to be classified or reranked as relevant or not for T_i. For example, in the QA setting, T_i would be a question and T_{ij}, $j = 1, ..., n$ would be a list of n candidate answer passages. The output of the system is a file in the SVMLight-TK[5] format containing the relational structures and feature vectors generated from the $\langle T_i, (T_{i1}, ..., T_{in})\rangle$ tuples.

When launched, the *RelTextRank* **System** module first initializes the other modules, such as the **UIMA** text analysis pipeline responsible for linguistic annotation of input texts, the **Experiment** module responsible for generating the structural representations enriched with the relational labels, and, finally, the **OutputWriter** module, which generates the output in the SVMLight-TK format.

At runtime, given an input $\langle T_i, (T_{i1}, ..., T_{in})\rangle$ tuple, *RelTextRank* generates (T_i, T_{ij}) pairs ($j = 1, ..., n$), and performs the following steps:

[4]Note that the structures in Fig. 1a- 1d here are depicted without `REL` and `FREL` links, however, at runtime the classes described in Sec. 4 do enrich them with the links

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

Step 1. Linguistic annotation. It runs a pipeline of UIMA Analysis Engines (AEs), which wrap linguistic annotators, e.g., Sentence Splitters, Tokenizers, Syntactic parsers, to convert the input text pairs (T_i, T_{ij}) into the UIMA Common Analysis Structures (CASes), i.e., $(CAS_i$ and $CAS_{ij})$. CASes contain the original texts and all the linguistic annotations produced by AEs. These produce linguistic annotations defined by a UIMA Type System. In addition, there is an option to persist the produced CASes, and not to rerun the annotators when re-processing a specific document.

Step 2. Generation of structural representations and feature vectors. The **Experiment** module is the core architectural component of the system, which takes CASes as input and generates the relational structures for T_i and T_{ij} along with their feature vector representation, $FV_{i,ij}$. $R(T_i, T_{ij})$ is the relational structure for T$_i$ enriched with the relational links towards T_{ij}, while $R(T_{ij}, T_i)$ is the opposite, i.e., the relational structure for T_{ij} with respect to T_i. Here, the **Projector** module generates $(R(T_i, T_{ij}), R(T_{ij}, T_i))$ and the **VectorFeatureExtractor** module generates $FV_{i,ij}$. In Sec. 4, we provide a list of Experiment modules that implement the representations described in Sec. 2.1, and a list of feature extractors to generate $FV_{i,ij}$.

Step 3. Generation of the output files. Once, all the pairs generated from the $\langle T_i, (T_{i1}, ..., T_{in})\rangle$ tuple have been processed, the **OutputWriter** module writes them into training/test files. We provide several output strategies described in Sec. 4.

4 Architecture Components

In order to generate the particular configuration of train/test data, one must specify which *System*, *Experiment* and *VectorFeatureExtractor* modules

Structure/previous usage	Class Name
CH (Severyn et al., 2013; Tymoshenko et al., 2016b)	`CHExperiment`
DT1 (Severyn et al., 2013; Tymoshenko and Moschitti, 2015)	`DT1Experiment`
DT2 (Tymoshenko and Moschitti, 2015)	`DT2Experiment`
LCT$_Q$-DT2$_A$ (Tymoshenko and Moschitti, 2015)	`LCTqDT2aExperiment`
CONST	`ConstExperiment`
CH^p (Tymoshenko et al., 2016a)	`CHpExperiment`
CH-cQA (Barrón-Cedeño et al., 2016)	`CHcQaExperiment`
CONST-cQA (Tymoshenko et al., 2016b)	`ConstQaExperiment`

Table 1: List of experimental configurations

to be used. In this section, we describe the implementations of the architectural modules currently available within RelTextRank.

System modules. These are the entry point to the pipeline, they initialize the specific structure and feature vector generation strategies (*Experiment* and *VectorFeatureExtractor*) and define the type (classification or reranking) and the format of the output file. Currently, we provide the system modules for generating classification (`ClassTextPairConversion`) and reranking training (`RERTextPairConversion`) files. Then, we provide a method to generate the cross-validation experimental data (`ClassCVTextPairConversion` and `CVRERTextPairConversion`). Additionally, we provide a method for generating training/test data for the answer comment reranking in cQA, `CQASemevalTaskA`. Finally, we provide a prediction module for classifying and reranking new text pairs with a pre-trained classifier (`TextPairPrediction`).

Every System module uses a single OutputWriter module, whose type of Experiment and FeatureVectorExtractor to be used are specified with command line parameters (see Sec. 5.)

UIMA pipeline. We provide the UIMA AEs (see `it.unitn.nlpir.annotators`), wrapping the components of the Stanford pipeline (Manning et al., 2014) and Illinois Chunker (Punyakanok and Roth, 2001). The UIMA pipeline takes a pair of texts, (T_i, T_{ij}) as input, and outputs their respective CASes, (CAS_i, CAS_{ij}).

RelTextRank also includes AEs for question classification (`QuestionClassifier`) and question focus detection (`QuestionFocusAnnotator`). Focus classification module employs a model pre-trained as in (Severyn et al., 2013). `QuestionClassifier` can be run with coarse- and fine-grained question classification models trained on the UIUC corpus by (Li and Roth, 2002) as described in (Severyn et al., 2013). The coarse-grained classier uses the following categories, HUMan, ENTitY, LOCation, ABBR, DESCription, NUMber classes, whereas the fine-grained classifier splits the NUM class into DATE, CURRENCY and QUANTITY.

Algorithm 1 Generating training data for reranking

```
Require: $S_{q+}$, $S_{q-}$ - $(T_i, T_{ij})$ pairs with positive and negative labels, respectively
 1: $E_+ \leftarrow \emptyset$, $E_- \leftarrow \emptyset$, $flip \leftarrow$ true
 2: for all $s_+ \in S_{q+}$ do
 3:     for all $s_- \in S_{q-}$ do
 4:         if $flip$ == true then
 5:             $E_+ \leftarrow E_+ \cup (s_+, s_-)$
 6:             $flip \leftarrow$ false
 7:         else
 8:             $E_- \leftarrow E_- \cup (s_-, s_+)$
 9:             $flip \leftarrow$ true
10: return $E_+$, $E_-$
```

Currently, we use a custom UIMA type system defined for our pipeline, however, in future we plan to use the type systems used in other widely used UIMA pipelines, e.g., DKPro (de Castilho and Gurevych, 2014).

Experiment modules. All the Experiment module implementations are available as classes in the `it.unitn.nlpir.experiment.*` package. Tab. 1 provides an overview of the structures currently available within the system. Here LCT$_Q$-DT2$_A$ represents T_i and T_{ij} as LCT and DT2 structures, respectively. CH-cQA and CONST-cQA are the CH and CONST structures adjusted for cQA (see (Tymoshenko et al., 2016b)).

OutputWriter modules. In the `it.unitn.nlpir.system.datagen` package, we provide the OutputWriters, which output the data in the SVMLight-TK format in the classification (`ClassifierDataGen`) and the reranking (`RerankingDataGenTrain` and `RerankingDataGenTest`) modes. Currently, the type of the OutputWriter can only be specified in the code of the System module. It is possible to create a new System module starting from the existing one and code a different OutputWriter.

In the classification mode, one OutputWriter generates one example for each text pair (T_i, T_{ij}). Another OutputWriter implementation generates input data for kernel-based reranking (Shen et al., 2003) using the strategy described in Alg. 1.

VectorFeatureExtractors. RelTextRank contains feature extractors to compute: (i) *cosine similarity* over the text pair: $sim_{COS}(T_1, T_2)$, where the input vectors are composed of word lemmas, bi-, three- an four-grams, POS-tags; *similarity based on the PTK score* computed for the structural rep-

Command 1. Generate training data

```
java -Xmx5G -Xss512m it.unitn.nlpir.system.core.RERTextPairConversion -questionsPath data/wikiQA/WikiQA-train.questions.txt
-answersPath data/wikiQA/WikiQA-train.tsv.resultset -outputDir data/examples/wikiqa -filePersistence CASes/wikiQA
-candidatesToKeep 10 -mode train -expClassName it.unitn.nlpir.experiment.fqa.CHExperiment
-featureExtractorClass it.unitn.nlpir.features.presets.BaselineFeatures
```

Command 2. Use pre-trained model to do classification

```
java -Xmx5G -Xss512m it.unitn.nlpir.system.core.TextPairPrediction -svmModel data/wikiQA/wikiqa-ch-rer-baselinefeats.model
-featureExtractorClass it.unitn.nlpir.features.presets.BaselineFeatures -questionsPath data/wikiQA/WikiQA-test.questions.txt
-answersPath data/wikiQA/WikiQA-test.tsv.resultset -outputFile data/examples/wikiqa/wikiqa-ch-rer-baselinefeats.pred
-expClassName it.unitn.nlpir.experiment.fqa.CHExperiment -mode reranking -filePersistence CASes/wikiQA/test
```

Table 2: Example commands to launch the pipeline

resentations of T_1 and T_2: sim_{PTK} $(T_1, T_2) = PTK(T_1, T_2)$, where the input trees can both be the dependency trees and/or the shallow chunk trees; (ii) *IR score*, which is a normalized score assigned to the answer passage by an IR engine, if available; (iii) *question class* as a binary feature.

Then, RelTextRank includes feature extractors based on the DKPro Similarity tool (Bär et al., 2013) for extracting (iv) *longest common substring/subsequence measure*; (v) *Jaccard similarity* coefficient on 1,2,3,4-grams; (vi) *word containment measure* (Broder, 1997); (vii) *greedy string tiling* (Wise, 1996); and (viii) *ESA similarity* based on Explicit Semantic Analysis (ESA) (Gabrilovich and Markovitch, 2007),

We provide several predefined VectorFeatureExtractors: `BaselineFeatures`, `AllFeatures` and `NoESAAllFeatures`[6], which incorporate feature groups: (i)-(iii), (i)-(viii) and (i)-(vii), respectively. Finally, we provide a feature extractor that reads the pre-extracted features from file, `FromFileVectorFeature`. The full list of features can be found by exploring the contents and documentation of the `it.unitn.nlpir.features` package.

5 Running the pipeline

Documentation on the RelTextRank GitHub explains how to install and run the pipeline with the various reranking and classification configurations. Due to the space limitations, here we only provide the sample commands for running the training file generation for the reranking mode (Tab. 2, Command 1) and using a pretrained model to rank the produced data (Tab. 2, Command 2).

Command 1 runs the RERTextPairConversion system (see Sec. 4), using the files specified as *-questionsPath* and *-answersPath* parameters to read the questions (T_i in Fig 2) and their corresponding candidate answers (T_{ij}, in Fig 2 with $j = 1, ..., n$; *-candidatesToKeep* parameter specifies the value of n), respectively. *-outputDir* is a path to the folder that will contain the resulting training file, while *-filePersistence* indicates where to persist the UIMA CASes containing the linguistic annotations produced by the UIMA pipeline (this is optional). *-mode train* indicates that we are generating the training file. *-expClassName* is a mandatory parameter, which indicates the Experiment module (Fig. 2) we want to invoke, i.e., which structure we wish to generate. In this specific example, we build a CH structure (see Tab. 1). Finally, *-featureExtractorClass* specifies which features to include into the feature vector.

Command 2 runs a pipeline that uses a pre-trained SVMLight-TK model (*-svmModel* parameter) to rerank the candidate answers (*-answerPath*) for the input questions (*-questionsPath*), and stores them into a prediction file (*-outputFile*). Here, we also indicate which structure generator and feature extractors to be used (*-expClassName* and *-featureExtractorClass*). Note that *-expClassName* and *-featureExtractorClass* must be exactly the same as the ones used when generating the data for training the model specified by *svmModel*.

6 Previous uses of RelTextRank

Tab. 3 reports the results of some of the state-of-the-art AS and cQA systems that employ `RelTextRank` as a component and combine the structures produced by it with the feature vectors of different nature, V. Here feature vectors are either manually handcrafted thread-level features, V_t, or word and phrase vector features, V_b, for cQA; or embeddings of T_i, T_{ij} learned by Convolutional Neural Networks, V_{CNN}, for the AS task.

Due to space limitations, we do not describe every system in detail, but provide link to a reference paper with the detailed setup description, and

[6]The motivation behind this feature extractor is that ESA feature extraction process is time-consuming

Corpus	Reference paper	Struct.	Feat.	V		+ Rel.Structures	
				MRR	MAP	MRR	MAP
WikiQA	(Tymoshenko et al., 2016a)*	CHp	V_{CNN}	67.49	66.41	73.88	71.99
TREC13	(Tymoshenko et al., 2016a)	CH	V_{CNN}	79.32	73.37	85.53*	75.18*
SemEval-2016, 3.A, English	(Tymoshenko et al., 2016b)	CONST	V_t	82.98	73.50	86.26	78.78
SemEval-2016, 3.D, Arabic	(Barrón-Cedeño et al., 2016)	CONST	V_b	43.75	38.33	52.55	45.50

Table 3: Previous uses of RelTextRank

mention which of the structures described in Sec. 2 they employ. (Tymoshenko et al., 2016a)* is a new structure and embedding combination approach.

We show the results on two AS corpora, WikiQA (Yang et al., 2015) and TREC13 (Wang et al., 2007). Then, we report the results obtained when using `RelTextRank` in a cQA system for English and Arabic comment selection tasks in the SemEval-2016 competition, Tasks 3.A and 3.D (Nakov et al., 2016).

V column reports the performance of the systems that employ feature vectors only, while $+Rel.Structures$ corresponds to the systems using a combination of relational structures generated by the earlier versions of `RelTextRank` and feature vectors. The numbers marked by * were obtained using relational structures only, since combining features and trees decreased the overall performance in that specific case. *Rel.Structures* always improves the performance.

7 Conclusions

In this demonstration paper we have provided an overview of the architecture and the particular components of the RelTextRank pipeline for generating structural relational representations of text pairs. In previous work, these representations have shown to achieve the state of the art for factoid QA and cQA. In the future, we plan to further evolve the pipeline, improving its code and usability. Moreover, we plan to expand the publicly available code to include more relational links, e.g., Linked Open Data-based relations described in (Tymoshenko et al., 2014). Finally, in order to enable better compatibility with publicly available tools, we plan to adopt the DKPro type system (de Castilho and Gurevych, 2014).

Acknowledgments

This work has been partially supported by the EC project CogNet, 671625 (H2020-ICT-2014-2, Research and Innovation action).

References

D. Bär, T. Zesch, and I. Gurevych. 2013. Dkpro similarity: An open source framework for text similarity. In *ACL: System Demonstrations*.

A. Barrón-Cedeño, G. Da San Martino, S. Joty, A. Moschitti, F. Al-Obaidli, S. Romeo, K. Tymoshenko, and A. Uva. 2016. Convkn at semeval-2016 task 3: Answer and question selection for question answering on arabic and english fora. In *SemEval-2016*.

A. Z Broder. 1997. On the resemblance and containment of documents. In *SEQUENCES*.

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

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

E. Gabrilovich and S. Markovitch. 2007. Computing semantic relatedness using wikipedia-based explicit semantic analysis. In *IJCAI*.

X. Li and D. Roth. 2002. Learning question classifiers. In *COLING*.

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

P. Nakov, L. Màrquez, A. Moschitti, W. Magdy, H. Mubarak, A. Freihat, J. Glass, and B. Randeree. 2016. Semeval-2016 task 3: Community question answering. In *SemEval*.

V. Punyakanok and D. Roth. 2001. The use of classifiers in sequential inference. In *NIPS*.

A. Severyn and A. Moschitti. 2012. Structural relationships for large-scale learning of answer re-ranking. In *SIGIR*.

A. Severyn, M. Nicosia, and A. Moschitti. 2013. Learning adaptable patterns for passage reranking. In *CoNLL*.

L. Shen, A. Sarkar, and A. Joshi. 2003. Using LTAG Based Features in Parse Reranking. In *EMNLP*.

K. Tymoshenko, D. Bonadiman, and A. Moschitti. 2016a. Convolutional neural networks vs. convolution kernels: Feature engineering for answer sentence reranking. In *NAACL-HLT*.

K. Tymoshenko, D. Bonadiman, and A. Moschitti. 2016b. Learning to rank non-factoid answers: Comment selection in web forums. In *CIKM*.

K. Tymoshenko and A. Moschitti. 2015. Assessing the impact of syntactic and semantic structures for answer passages reranking. In *CIKM*.

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

Mengqiu Wang, Noah A. Smith, and Teruko Mitaura. 2007. What is the Jeopardy model? A quasi-synchronous grammar for QA. In *EMNLP-CoNLL*.

Michael J. Wise. 1996. Yap3: improved detection of similarities in computer program and other texts. In *ACM SIGCSE Bulletin*.

Y. Yang, W. Yih, and C. Meek. 2015. Wikiqa: A challenge dataset for open-domain question answering. In *EMNLP*.