iKernels-Core: Tree Kernel Learning for Textual Similarity

Aliaksei Severyn[1] and **Massimo Nicosia**[1] and **Alessandro Moschitti**[1,2]
[1]University of Trento, DISI, 38123 Povo (TN), Italy
{severyn,m.nicosia,moschitti}@disi.unitn.it
[2]Qatar Foundation, QCRI, Doha, Qatar
{amoschitti}@qf.org.qa

Abstract

This paper describes the participation of iKernels system in the Semantic Textual Similarity (STS) shared task at *SEM 2013. Different from the majority of approaches, where a large number of pairwise similarity features are used to learn a regression model, our model directly encodes the input texts into syntactic/semantic structures. Our systems rely on tree kernels to automatically extract a rich set of syntactic patterns to learn a similarity score correlated with human judgements. We experiment with different structural representations derived from constituency and dependency trees. While showing large improvements over the top results from the previous year task (STS-2012), our best system ranks 21st out of total 88 participated in the STS-2013 task. Nevertheless, a slight refinement to our model makes it rank 4th.

1 Introduction

Comparing textual data to establish the degree of semantic similarity is of key importance in many Natural Language Processing (NLP) tasks ranging from document categorization to textual entailment and summarization. The key aspect of having an accurate STS framework is the design of features that can adequately represent various aspects of the similarity between texts, e.g. using lexical, syntactic and semantic similarity metrics.

The majority of approaches to semantic textual similarity treat the input text pairs as feature vectors where each feature is a score corresponding to a certain type of similarity. This approach is conceptually easy to implement and STS-2012 (Agirre et al., 2012) has shown that the best systems were built following this idea, i.e. a number of features encoding similarity of an input text pair were combined in a single scoring model, such as Linear Regression or Support Vector Regression (SVR). One potential limitation of using only similarity features to represent a text pair is that of low representation power.

The novelty of our approach is that we encode the input text pairs directly into structural objects, e.g. trees, and rely on the power of kernel learning to extract relevant structures. This completely different from (Croce et al.,), where tree kernels where used to establish syntactic similarity and then plugged as similarity features. To link the documents in a pair we mark the nodes in the related structures with a special relational tag. In this way effective structural relational patterns are implicitly encoded in the trees and can be automatically learned by the kernel-based machine learning methods. We build our systems on top of the features used by two best systems from STS-2012 and combine them with the tree kernel models within the Support Vector Regression to derive a single scoring model. Since the test data used for evaluation in STS-2013 (Agirre et al., 2013) is different from the 2012 data provided for the system development, domain adaptation represents an additional challenge. To address this problem we augment our feature vector representation with features extracted from a text pair as a whole to capture individual properties of each dataset. Additionally, we experiment with a corpus type classifier and include its prediction score as additional features. Finally, we use stacking to combine several structural models into the feature vector representation.

*Second Joint Conference on Lexical and Computational Semantics (*SEM), Volume 1: Proceedings of the Main Conference and the Shared Task*, pages 53–58, Atlanta, Georgia, June 13-14, 2013.

In the following sections we describe our approach to combine structural representations with the pairwise similarity features in a single SVR learning framework. We then report results on both STS-2012 and 2013 tasks.

2 Structural Relational Similarity

In this section we first describe the kernel framework to combine structural and vector models, then we explain how to construct the tree models and briefly describe tree kernels we use to automatically extract the features.

2.1 Structural Kernel Learning

In supervised learning, given the labeled data $\{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}_{i=1}^{n}$, the goal is to estimate a decision function $h(\boldsymbol{x}) = \boldsymbol{y}$ that maps input examples to the target variables. A conventional approach is to represent a pair of texts as a set of similarity features $\{f_i\}$, s.t. the predictions are computed as $h(\boldsymbol{x}) = \boldsymbol{w} \cdot \boldsymbol{x} = \sum_i w_i f_i$, where $\boldsymbol{w}$ is the model weight vector. Hence, the learning problem boils down to estimating the individual weight of each of the similarity feature f_i. One downside of such approach is that a great deal of similarity information carried by a given text pair is lost when modeled by single real-valued scores.

A more versatile approach in terms of the input representation relies on kernels. In a typical kernel machine, e.g. SVM, the prediction function for a test input $\boldsymbol{x}$ takes on the following form $h(\boldsymbol{x}) = \sum_i \alpha_i y_i K(\boldsymbol{x}, \boldsymbol{x}_i)$, where α_i are the model parameters estimated from the training data, y_i - target variables, $\boldsymbol{x}_i$ are support vectors, and $K(\cdot, \cdot)$ is a kernel function.

To encode both structural representation and similarity feature vectors of input text pairs $\boldsymbol{x}_i$ in a single model, we treat it as the following tuple: $\boldsymbol{x}_i = \langle \boldsymbol{x}_i^a, \boldsymbol{x}_i^b \rangle = \langle (\boldsymbol{t}_i^a, \boldsymbol{v}_i^a), (\boldsymbol{t}_i^b, \boldsymbol{v}_i^b) \rangle$, where $\boldsymbol{x}_i^a$ $\boldsymbol{x}_i^b$ are the first and the second document of $\boldsymbol{x}_i$, and $\boldsymbol{t}$ and $\boldsymbol{v}$ denote tree and vector representations respectively.

To compute a kernel between two text pairs $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ we define the following all-vs-all kernel, where all possible combinations of documents from each pair are considered: $K(\boldsymbol{x}_i, \boldsymbol{x}_j) = K(\boldsymbol{x}_i^a, \boldsymbol{x}_j^a) + K(\boldsymbol{x}_i^a, \boldsymbol{x}_j^b) + K(\boldsymbol{x}_i^b, \boldsymbol{x}_j^a) + K(\boldsymbol{x}_i^b, \boldsymbol{x}_j^b)$. Each of the kernel computations K between two documents $\boldsymbol{x}^a$ and $\boldsymbol{x}^b$ can be broken down into the following: $K(\boldsymbol{x}^a, \boldsymbol{x}^b) = K_{\text{TK}}(\boldsymbol{t}^a, \boldsymbol{t}^b) + K_{\text{fvec}}(\boldsymbol{v}^a, \boldsymbol{v}^b)$, where K_{TK} computes a tree kernel and K_{fvec} is a kernel over feature vectors, e.g. linear, polynomial or RBF, etc. Further in the text we refer to structural tree kernel models as `TK` and explicit feature vector representation as `fvec`.

Having defined a way to jointly model text pairs using structural `TK` representations along with the similarity features `fvec`, we next briefly review tree kernels and our relational structures derived from constituency and dependency trees.

2.2 Tree Kernels

We use tree structures as our base representation since they provide sufficient flexibility in representation and allow for easier feature extraction than, for example, graph structures. We use a Partial Tree Kernel (PTK) (Moschitti, 2006) to take care of automatic feature extraction and compute $K_{\text{TK}}(\cdot, \cdot)$.

PTK is a tree kernel function that can be effectively applied to both constituency and dependency parse trees. It generalizes a subset tree kernel (STK) (Collins and Duffy, 2002) that maps a tree into the space of all possible tree fragments constrained by the rule that the sibling nodes from their parents cannot be separated. Different from STK where the nodes in the generated tree fragments are constrained to include none or all of their direct children, PTK fragments can contain any subset of the features, i.e. PTK allows for breaking the production rules. Consequently, PTK generalizes STK generating an extremely rich feature space, which results in higher generalization ability.

2.3 Relational Structures

The idea of using relational structures to jointly model text pairs was previously proposed in (Severyn and Moschitti, 2012), where shallow syntactic structures derived from chunks and part-of-speech tags were used to represent question/answer pairs. In this paper, we define novel relational structures based on: (i) constituency and (ii) dependency trees.

Constituency tree. Each document in a given text pair is represented by its constituency parse tree. If a document contains multiple sentences they are merged in a single tree with a common root. To encode the structural relationships between docu-

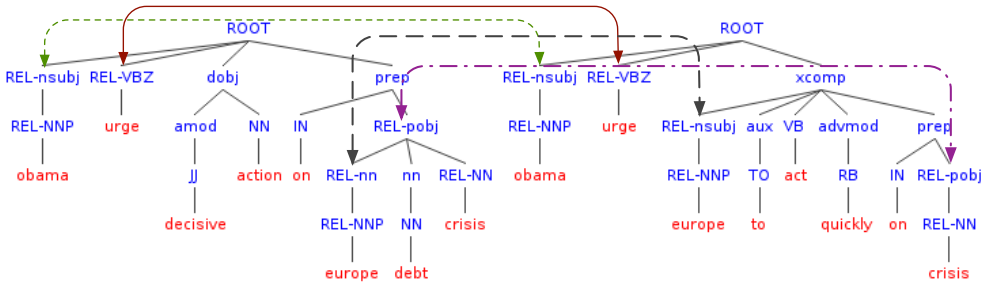

Figure 1: A dependency-based structural representation of a text pair. `REL` tag links related fragments.

ments in a pair a special `REL` tag is used to link the related structures. We adopt a simple strategy to establish such links: words from two documents that have a common lemma get their parents (POS tags) and grandparents, non-terminals, marked with a `REL` tag.

Dependency tree. We propose to use dependency relations between words to derive an alternative structural representation. In particular, dependency relations are used to link words in a way that words are always at the leaf level. This reordering of the nodes helps to avoid the situation where nodes with words tend to form long chains. This is essential for PTK to extract meaningful fragments. We also plug part-of-speech tags between the word nodes and nodes carrying their grammatical role. Again a special `REL` tag is used to establish relations between tree fragments. Fig. 1 gives an example of a dependency-based structure taken from STS-2013 *headlines* dataset.

3 Pairwise similarity features.

Along with the direct representation of input text pairs as structural objects our framework also encodes feature vectors (`base`), which we describe below.

3.1 Baseline features

We adopt similarity features from two best performing systems of STS-2012, which were publicly released: namely, the Takelab[1] system (Šarić et al., 2012) and the UKP Lab's system[2] (Bar et al., 2012). Both systems represent input texts with similarity features which combine multiple text similarity measures of varying complexity.

UKP provides metrics based on matching of character, word n-grams and common subsequences. It also includes features derived from Explicit Semantic Analysis vector comparisons and aggregation of word similarity based on lexical-semantic resources, e.g. WordNet. In total it provides 18 features.

Takelab includes n-gram matching of varying size, weighted word matching, length difference, WordNet similarity and vector space similarity where pairs of input sentences are mapped into Latent Semantic Analysis (LSA) space (Turney and Pantel, 2010). The features are computed over several sentence representations where stop words are removed and/or lemmas are used in place of raw tokens. The total number of Takelab's features is 21. Even though some of the UKP and Takelab features overlap we include all of them in a combined system with the total of 39 features.

3.2 iKernels features

Here we describe our additional features added to the `fvec` representation. First, we note that word frequencies used to compute weighted word matchings and the word-vector mappings to compute LSA similarities required by **Takelab** features are provided only for the vocabulary extracted from 2012 data. Hence, we use both STS-2012 and 2013 data to obtain the word counts and re-estimate LSA vector representations. For the former we extract unigram counts from Google Books Ngrams[3], while for the latter we use additional corpora as described below.

LSA similarity. To construct LSA word-vector mappings we use the following three sources: (i)

[1] http://takelab.fer.hr/sts/

[2] https://code.google.com/p/dkpro-similarity-asl/wiki/SemEval2013

[3] http://storage.googleapis.com/books/ngrams/books/datasetsv2.html

Aquaint[4], which consists of more than 1 million newswire documents, (ii) ukWaC (Baroni et al., 2009) - a 2 billion word corpus constructed from the Web, and (iii) and a collection of documents extracted from Wikipedia dump[5]. To extract LSA topics we use GenSim[6] software. We preprocess the data by lowercasing, removing stopwords and words with frequency lower than 5. Finally, we apply tf-idf weighting. For all representations we fix the number of dimensions to 250. For all corpora we use document-level representation, except for Wikipedia we also experimented with a sentence-level document representation, which typically provides a more restricted context for estimating word-document distributions.

Brown Clusters. In addition to vector representations derived from LSA, we extract word-vector mappings using Brown word clusters[7] (Turian et al., 2010), where words are organized into a hierarchy and each word is represented as a bit-string. We encode each word by a feature vector where each entry corresponds to a prefix extracted from its bit-string. We use prefix lengths in the following range: $k = \{4, 8, 12, 16, 20\}$. Finally, the document is represented as a feature vector composed by the individual word vectors.

Term-overlap features. In addition to the word overlap features computed by **UKP** and **Takelab** systems we also compute a cosine similarity over the following representations: (i) n-grams of part-of-speech tags (up to 4-grams), (ii) SuperSense tags (Ciaramita and Altun, 2006), (iii) named entities, and (iv) dependency triplets.

PTK similarity. We use PTK to provide a syntactic similarity score between documents in a pair: $\texttt{PTK}(a, b) = PTK(a, b)$, where as input representations we use *dependency* and *constituency* trees.

Explicit Semantic Analysis (ESA) similarity. ESA (Gabrilovich and Markovitch, 2007) represents input documents as vectors of Wikipedia concepts. To compute ESA features we use Lucene[8] to index documents extracted from a Wikipedia dump. Given a text pair we retrieve k top documents (i.e. Wikipedia concepts) and compute the metric by looking at the overlap of the concepts between the documents: $\texttt{esa}_k(a, b) = \frac{|W_a \bigcap W_b|}{k}$, where W_a is the set of concepts retrieved for document a. We compute esa features with $k \in \{10, 25, 50, 100\}$.

[4]http://www.ldc.upenn.edu/Catalog/docs/LDC2002T31/

[5]http://dumps.wikimedia.org/

[6]http://radimrehurek.com/gensim/

[7]http://metaoptimize.com/projects/wordreprs/

[8]http://lucene.apache.org/

3.3 Corpus type features

Here we describe two complementary approaches (corpus) in an attempt to alleviate the problem of domain adaptation, where the datasets used for training and testing are drawn from different sources.

Pair representation. We treat each pair of texts as a whole and extract the following sets of corpus features: plain bag-of-words, dependency triplets, production rules of the syntactic parse tree and a length feature, i.e. a log-normalized length of the combined text. Each feature set is normalized and added to the fvec model.

Corpus classifier. We use the above set of features to train a multi-class classifier to predict for each instance its most likely corpus type. Our categories correspond to five dataset types of STS-2012. Prediction scores for each of the dataset categories are then plugged as features into the final fvec representation. Our multi-class classifier is a one-vs-all binary SVM trained on the merged data from STS-2012. We apply 5-fold cross-validation scheme, s.t. for each of the held-out folds we obtain independent predictions. The accuracy (averaged over 5-folds) on the STS-2012 data is 92.0%.

3.4 Stacking

To integrate multiple TK models into a single model we apply a classifier stacking approach (Fast and Jensen, 2008). Each of the learned TK models is used to generate predictions which are then plugged as features into the final fvec representation, s.t. the final model uses only explicit feature vector representation. We apply a 5-fold cross-validation scheme to obtain prediction scores in the same manner as described above.

4 Experimental Evaluation

4.1 Experimental setup

To encode TK models along with the similarity feature vectors into a single regression scoring model,

	base			corpus			TK							
	U	T	I	B	O	M	C	D	ALL	Mean	MSRp	MSRv	SMTe	OnWN	SMTn
	•								0.7060	0.6087	0.6080	0.8390	0.2540	0.6820	0.4470
		•							0.7589	0.6863	0.6814	0.8637	0.4950	0.7091	0.5395
	•	•							0.8079	0.7161	0.7134	0.8837	0.5519	0.7343	0.5607
	•	•	•						0.8187	0.7137	0.7157	0.8833	0.5131	0.7355	0.5809
	•	•	•				•		0.8458	0.7047	0.6935	0.8953	0.5080	0.7101	0.5834
	•	•	•					•	0.8468	0.6954	0.6717	0.8902	0.4652	0.7089	0.6133
	•	•	•	•			•		0.8539	0.7132	0.6993	0.9005	0.4772	0.7189	0.6481
	•	•	•	•				•	0.8529	0.7249	0.7080	0.8984	0.5142	0.7263	0.6700
Sys$_1$	•	•	•	•			•	•	0.8546	0.7156	0.6989	0.8979	0.4884	0.7181	0.6609
Sys$_3$	•	•	•		•		•	•	0.8810	0.7416	0.7210	0.8971	0.5912	0.7328	0.6778
Sys$_2$	•	•	•			•	•	•	0.8705	0.7339	0.7039	0.9012	0.5629	0.7376	0.6656
UKP$_{\text{best}}$									0.8239	0.6773	0.6830	0.8739	0.5280	0.6641	0.4937

Table 1: System configurations and results on STS-2012. Column set `base` lists 3 feature sets : UKP (U), Takelab (T) and iKernels (I); corpus type features (`corpus`) include plain features (B), corpus classifier (O), and manually encoded dataset category (M); `TK` contains constituency (C) and dependency-based (D) models. UKP$_{\text{best}}$ is the best system of STS-2012. First column shows configuration of our three system runs submitted to STS-2013.

we use an SVR framework implemented in SVM-Light-TK[9]. We use the following parameter settings `-t 5 -F 3 -W A -C +`, which specifies to use a combination of trees and feature vectors (`-C +`), PTK over trees (`-F 3`) computed in all-vs-all mode (`-W A`) and using polynomial kernel of degree 3 for the feature vector (active by default).

We report the following metrics employed in the final evaluation: *Pearson* correlation for individual test sets[10] and *Mean* – an average score weighted by the test set size.

4.2 STS-2012

For STS-2013 task the entire data from STS-2012 was provided for the system development. To compare with the best systems of the previous year we followed the same setup, where 3 datasets (*MSRp*, *MSRv* and *SMTe*) are used for training and 5 for testing (two "surprise" datasets were added: *OnWN* and *SMTn*). We use the entire training data to obtain a single model.

Table 1 summarizes the results using structural models (`TK`), pairwise similarity (`base`) and corpus type features (`corpus`). We first note, that combining all three features sets (`U`, `T` and `I`) provides a good match to the best system UKP$_{\texttt{best}}$. Next, adding TK models results in a large improvement beating the top results in STS-2012. Furthermore, using `corpus` features results in even greater improvement with the *Mean* $= 0.7416$ and Pearson *ALL* $= 0.8810$.

4.3 STS-2013

Below we specify the configuration for each of the submitted runs (also shown in Table 1) and report the results on the STS-2013 test sets: *headlines* (*head*), *OnWN*, *FNWN*, and *SMT*:

Sys$_1$: combines `base` features (`U`, `T` and `I`), TK models (`C` and `D`) and plain corpus type features (`B`). We use STS-2012 data to train a single model.

Sys$_2$: different from **Sys$_1$** where a single model trained on the entire data is used to make predictions, we adopt a different training/test setup to account for the different nature of the data used for training and testing. After performing manual analysis of the test data we came up with the following strategy to split the training data into two sets to learn two different models: *STMe* and *OnWN* (model$_1$) and *MSRp*, *SMTn* and *STMe* (model$_2$); model$_1$ is then used to get predictions for *OnWN*, *FNWN*, while model$_2$ is used for *SMT* and *headlines*.

Sys$_3$: same as **Sys$_1$** + a corpus type classifier `O` as described in Sec. 3.3.

Table 2 shows the resulting performance of our systems and the best UMBC system published in the final ranking. **Sys$_2$** appears the most accurate among our systems, which ranked 21st out of 88. Comparing to the best system across four datasets we observe that it performs reasonably well on the *headlines* dataset (it is 5th best), while completely fails on the *OnWN* and *FNWN* test sets. After performing

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

[10] for STS-2012 we also report the results for a concatenation of all five test sets (`ALL`)

error analysis, we found that TK models underperform on *FNWN* and *OnWN* sets, which appear underrepresented in the training data from STS-2012. We build a new system ($\mathbf{Sys}_2^*$), which is based on $\mathbf{Sys}_2$, by making two adjustments in the setup: (i) we exclude *SMTe* from training to obtain predictions on *SMT* and *head* and (ii) we remove all TK features to train a model for *FNWN* and *OnWN*. This is motivated by the observation that text pairs from STS-2012 yield a paraphrase model, since the texts are syntactically very similar. Yet, two datasets from STS-2013 *FNWN*, and *OnWN* contain text pairs where documents exhibit completely different structures. This is misleading for our syntactic similarity model learned on the STS-2012.

System	head	OnWN	FNWN	SMT	Mean	Rank
UMBC	0.7642	0.7529	0.5818	0.3804	0.6181	1
$\mathbf{Sys}_2$	0.7465	0.5572	0.3875	0.3409	0.5339	21
$\mathbf{Sys}_1$	0.7352	0.5432	0.3842	0.3180	0.5188	28
$\mathbf{Sys}_3$	0.7395	0.4228	0.3596	0.3294	0.4919	40
$\mathbf{Sys}_2^*$	0.7538	0.6872	0.4478	0.3391	0.5732	4*

Table 2: Results on STS-2013.

5 Conclusions and Future Work

We have described our participation in STS-2013 task. Our approach treats text pairs as structural objects which provides much richer representation for the learning algorithm to extract useful patterns. We experiment with structures derived from constituency and dependency trees where related fragments are linked with a special tag. Such structures are then used to learn tree kernel models which can be efficiently combined with the a feature vector representation in a single scoring model. Our approach ranks 1st with a large margin w.r.t. to the best systems in STS-2012 task, while it is 21st according to the final rankings of STS-2013. Nevertheless, a small change in the system setup makes it rank 4th. Clearly, domain adaptation represents a big challenge in STS-2013 task. We plan to address this issue in our future work.

6 Acknowledgements

This research has been supported by the European Community's Seventh Framework Program (FP7/2007-2013) under the #288024 LIMOSINE project.

References

Eneko Agirre, Daniel Cer, Mona Diab, and Gonzalez-Agirre. 2012. Semeval-2012 task 6: A pilot on semantic textual similarity. In *First Joint Conference on Lexical and Computational Semantics (*SEM)*.

Eneko Agirre, Daniel Cer, Mona Diab, Aitor Gonzalez-Agirre, and Weiwei Guo. 2013. *sem 2013 shared task: Semantic textual similarity, including a pilot on typed-similarity. In **SEM 2013: The Second Joint Conference on Lexical and Computational Semantics*.

Daniel Bar, Chris Biemann, Iryna Gurevych, and Torsten Zesch. 2012. Ukp: Computing semantic textual similarity by combining multiple content similarity measures. In *Proceedings of the Sixth International Workshop on Semantic Evaluation (SemEval 2012)*.

Marco Baroni, Silvia Bernardini, Adriano Ferraresi, and Eros Zanchetta. 2009. The wacky wide web: a collection of very large linguistically processed web-crawled corpora. *Language Resources and Evaluation*, 43(3):209–226.

Massimiliano Ciaramita and Yasemin Altun. 2006. Broad-coverage sense disambiguation and information extraction with a supersense sequence tagger. In *EMNLP*.

Michael Collins and Nigel Duffy. 2002. New Ranking Algorithms for Parsing and Tagging: Kernels over Discrete Structures, and the Voted Perceptron. In *ACL*.

Danilo Croce, Paolo Annesi, Valerio Storch, and Roberto Basili. Unitor: Combining semantic text similarity functions through sv regression. In *SemEval 2012*.

Andrew S. Fast and David Jensen. 2008. Why stacked models perform effective collective classification. In *ICDM*.

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

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

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

Joseph Turian, Lev Ratinov, and Yoshua Bengio. 2010. Word representations: a simple and general method for semi-supervised learning. In *ACL*.

Peter D. Turney and Patrick Pantel. 2010. From frequency to meaning: vector space models of semantics. *J. Artif. Int. Res.*, 37(1):141–188.

Frane Šarić, Goran Glavaš, Mladen Karan, Jan Šnajder, and Bojana Dalbelo Bašić. 2012. Takelab: Systems for measuring semantic text similarity. In *Proceedings of the Sixth International Workshop on Semantic Evaluation (SemEval 2012)*.