Learning Semantic Textual Similarity with Structural Representations

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

Abstract

Measuring semantic textual similarity (STS) is at the cornerstone of many NLP applications. Different from the majority of approaches, where a large number of pairwise similarity features are used to represent a text pair, our model features the following: (i) it directly encodes input texts into relational syntactic structures; (ii) relies on tree kernels to handle feature engineering automatically; (iii) combines both structural and feature vector representations in a single scoring model, i.e., in Support Vector Regression (SVR); and (iv) delivers significant improvement over the best STS systems.

1 Introduction

In STS the goal is to learn a scoring model that given a pair of two short texts returns a similarity score that correlates with human judgement. Hence, 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 treat 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 the STS shared task at SemEval 2012 (Agirre et al., 2012) (STS-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, e.g., SVR. Nevertheless, one 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 treat the input text pairs as structural objects and rely on the power of kernel learning to extract relevant structures. To link the documents in a pair we mark the nodes in the related structures with a special relational tag. This way effective structural relational patterns are implicitly encoded in the trees and can be automatically learned by the kernel-based machines. We combine our relational structural model with the features from two best systems of STS-2012. Finally, we use the approach of classifier stacking to combine several structural models into the feature vector representation.

The contribution of this paper is as follows: (i) it provides a convincing evidence that adding structural features automatically extracted by structural kernels yields a significant improvement in accuracy; (ii) we define a combination kernel that integrates both structural and feature vector representations within a single scoring model, e.g., Support Vector Regression; (iii) we provide a simple way to construct relational structural models that can be built using off-the-shelf NLP tools; (iv) we experiment with four structural representations and show that constituency and dependency trees represent the best source for learning structural relationships; and (v) using a classifier stacking approach, structural models can be easily combined and integrated into existing feature-based STS models.

2 Structural Relational Similarity

The approach of relating pairs of input structures by learning predictable syntactic transformations has shown to deliver state-of-the-art results in question answering, recognizing textual entailment, and paraphrase detection, e.g. (Wang et al., 2007; Wang and Manning, 2010; Heilman and Smith, 2010). Previous work relied on fairly complex approaches, e.g. applying quasi-synchronous grammar formalism and variations of tree edit distance alignments, to extract syntactic patterns relating pairs of input structures. Our approach is conceptually simpler, as it regards the problem within the kernel learning framework, where we first encode salient syntactic/semantic properties of the input text pairs into tree structures and rely on tree kernels to automatically generate rich feature spaces. This work extends in several directions our earlier work in question answering, e.g., (Moschitti et al., 2007; Moschitti and Quarteroni, 2008), in textual entailment recognition, e.g., (Zanzotto and Moschitti, 2006; Moschitti and Zanzotto, 2007), and more in general in relational text categorization (Moschitti, 2008; Severyn and Moschitti, 2012).

In this section we describe: (i) a kernel framework to combine structural and vector models; (ii) structural kernels to handle feature engineering; and (iii) suitable structural representations for relational learning.

2.1 Structural Kernel Learning

In supervised learning, given 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 their targets. 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 individual weights of each of the similarity features f_i. One downside of such approach is that a great deal of similarity information encoded in 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 learning approach, 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 are 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 a given text pair in a single model we define each document in a pair to be composed of a tree and a vector: $\langle \boldsymbol{t}, \boldsymbol{v} \rangle$. 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 components, $\boldsymbol{x}^{(1)}$ and $\boldsymbol{x}^{(2)}$, from each text pair are considered: $K(\boldsymbol{x}_i, \boldsymbol{x}_j) = K(\boldsymbol{x}_i^{(1)}, \boldsymbol{x}_j^{(1)}) + K(\boldsymbol{x}_i^{(1)}, \boldsymbol{x}_j^{(2)}) + K(\boldsymbol{x}_i^{(2)}, \boldsymbol{x}_j^{(1)}) + K(\boldsymbol{x}_i^{(2)}, \boldsymbol{x}_j^{(2)})$. Each of the kernel computations K can be broken down into the following: $K(\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}) = K_{\text{TK}}(\boldsymbol{t}^{(1)}, \boldsymbol{t}^{(2)}) + K_{\text{fvec}}(\boldsymbol{v}^{(1)}, \boldsymbol{v}^{(2)})$, where K_{TK} computes a structural 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.

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. Hence, we rely on tree kernels to compute $K_{\text{TK}}(\cdot,\cdot)$. Given two trees it evaluates the number of substructures (or *fragments*) they have in common, i.e., it is a measure of their overlap. Different TK functions are characterized by alternative fragment definitions. In particular, we focus on the Syntactic Tree kernel (STK) (Collins and Duffy, 2002) and a Partial Tree Kernel (PTK) (Moschitti, 2006).

STK generates all possible substructures rooted in each node of the tree with the constraint that production rules can not be broken (i.e., any node in a tree fragment must include either all or none of its children).

PTK can be more effectively applied to both constituency and dependency parse trees. It generalizes STK as the fragments it generates can contain any subset of nodes, i.e., PTK allows for breaking the production rules and generating an extremely rich feature space, which results in higher generalization ability.

2.3 Structural representations

In this paper, we define simple-to-build relational structures based on: (i) a shallow syntactic tree, (ii) constituency, (iii) dependency and (iv) phrase-dependency trees.

Shallow tree is a two-level syntactic hierarchy built from word lemmas (leaves), part-of-speech tags (preterminals) that are further organized into chunks. It was shown to significantly outperform feature vector baselines for modeling relationships between question answer pairs (Severyn and Moschitti, 2012).

Constituency tree. While shallow syntactic parsing is very fast, here we consider using constituency structures as a potentially richer source of syntactic/semantic information.

Dependency tree. We propose to use dependency relations between words to derive an alter-

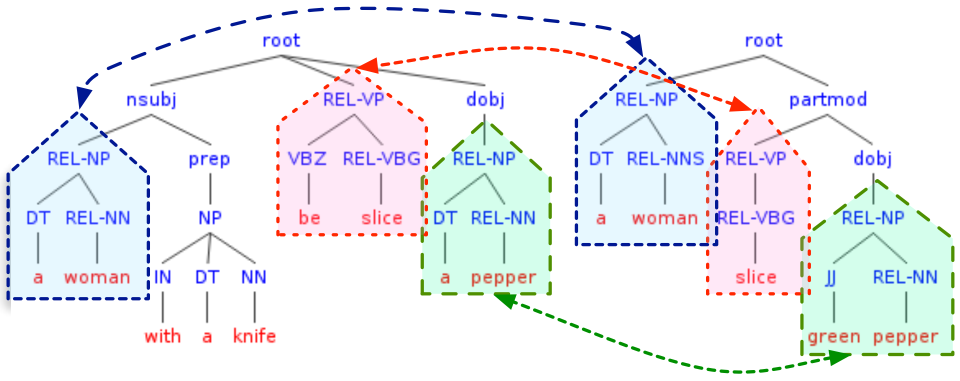

Figure 1: A phrase dependency-based structural representation of a text pair (s1, s2): *A woman with a knife is slicing a pepper* (s1) vs. *A women slicing green pepper* (s2) with a high semantic similarity (human judgement score 4.0 out of 5.0). Related tree fragments are linked with a `REL` tag.

native structural representation. In particular, dependency relations are used to link words in a way that they 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.

Phrase-dependency tree. We explore a phrase-dependency tree similar to the one defined in (Wu et al., 2009). It represents an alternative structure derived from the dependency tree, where the dependency relations between words belonging to the same phrase (chunk) are collapsed in a unified node. Different from (Wu et al., 2009), the collapsed nodes are stored as a shallow subtree rooted at the unified node. This node organization is particularly suitable for PTK that effectively runs a sequence kernel on the tree fragments inside each chunk subtree. Fig 1 gives an example of our variation of a phrase dependency tree.

As a final consideration, if a document contains multiple sentences they are merged in a single tree with a common root. To encode the structural relationships between documents 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.

3 Pairwise similarity features.

Along with the direct representation of input text pairs as structural objects our framework is also capable of encoding pairwise similarity feature vectors (`fvec`), which we describe below.

Baseline features. (`base`) We adopt similarity features from two best performing systems of STS-2012, which were publicly released[1]: namely, the Takelab[2] system (Šarić et al., 2012) and the UKP Lab's system[3] (Bar et al., 2012). Both systems represent input texts with similarity features combining multiple text similarity measures of varying complexity.

UKP (`U`) provides metrics based on matching of character, word n-grams and common subsequences. It also includes features derived from Explicit Semantic Analysis (Gabrilovich and Markovitch, 2007) and aggregation of word similarity based on lexical-semantic resources, e.g., WordNet. In total it provides 18 features.

Takelab (`T`) 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. 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. The combined system consists of 39 features.

Additional features. We also augment the `U` and `T` feature sets, with an additional set of features (`A`) which includes: a cosine similarity scores computed over (i) n-grams of part-of-speech tags (up to 4-grams), (ii) SuperSense tags (Ciaramita and Altun, 2006), (iii) named entities, (iv) dependency

[1]Note that only a subset of the features used in the final evaluation was released, which results in lower accuracy when compared to the official rankings.

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

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

triplets, and (v) PTK syntactic similarity scores computed between documents in a pair, where as input representations we use raw *dependency* and *constituency* trees. To alleviate the problem of domain adaptation, where datasets used for training and testing are drawn from different sources, we include additional features to represent the combined text of a pair: (i) bags (`B`) of lemmas, dependency triplets, production rules (from the constituency parse tree) and a normalized length of the entire pair; and (ii) a manually encoded corpus type (`M`), where we use a binary feature with a non-zero entry corresponding to a dataset type. This helps the learning algorithm to learn implicitly the individual properties of each dataset.

Stacking. To integrate multiple `TK` representations 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. To obtain prediction scores, we apply 5-fold cross-validation scheme, s.t. for each of the held-out folds we obtain independent predictions.

4 Experiments

We present the results of our model tested on the data from the Core STS task at SemEval 2012.

4.1 Setup

Data. To compare with the best systems of the STS-2012 we followed the same setup used in the final evaluation, where 3 datasets (*MSRpar*, *MSRvid* and *SMTeuroparl*) are used for training and 5 for testing (two "surprise" datasets were added: *OnWN* and *SMTnews*). We use the entire training data to obtain a single model for making predictions on each test set.

Software. To encode TK models along with the similarity feature vectors into a single regression scoring model, we use an SVR framework implemented in SVM-Light-TK[4]. We use the following parameter settings `-t 5 -F 1 -W A -C +`, which specifies a combination of trees and feature vectors (`-C +`), STK over trees (`-F 1`) (`-F 3` for PTK) computed in all-vs-all mode (`-W A`) and polynomial kernel of degree 3 for the feature vector (active by default).

Metrics. We report the following metrics employed in the final evaluation: *Pearson* correlation for individual test sets[5] and *Mean* – an average score weighted by the test set size.

4.2 Results

Table 1 summarizes the results of combining TK models with a strong feature vector model. We test structures defined in Sec. 2.3 when using STK and PTK. The results show that: (i) combining all three features sets (`U`, `T`, `A`) provides a strong baseline system that we attempt to further improve with our relational structures; (ii) the generality of PTK provides an advantage over STK for learning more versatile models; (iii) constituency and dependency representations seem to perform better than shallow and phrase-dependency trees; (iv) using structures with no relational linking does not work; (v) `TK` models provide a far superior source of structural similarity than `U` + `T` + `A` that already includes PTK similarity scores as features, and finally (vi) the domain adaptation problem can be addressed by including corpus specific features, which leads to a large improvement over the previous best system.

5 Conclusions and Future Work

We have presented an approach where text pairs are directly treated as structural objects. This provides a much richer representation for the learning algorithm to extract useful syntactic and shallow semantic patterns. We have provided an extensive experimental study of four different structural representations, e.g. shallow, constituency, dependency and phrase-dependency trees using STK and PTK. The novelty of our approach is that it goes beyond a simple combination of tree kernels with feature vectors as: (i) it directly encodes input text pairs into relationally linked structures; (ii) the learned structural models are used to obtain prediction scores thus making it easy to plug into existing feature-based models, e.g. via stacking; (iii) to our knowledge, this work is the first to apply structural kernels and combinations in a regression setting; and (iv) our model achieves the state of the art in STS largely improving the best previous systems. Our structural learning approach to STS is conceptually simple and does not require additional linguistic sources other than off-the-shelf syntactic parsers. It is particularly suitable for NLP tasks where the input domain comes as pairs of objects, e.g., question answering, paraphrasing and recognizing textual entailment.

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

[5]we also report the results for a concatenation of all five test sets (`ALL`)

Experiment	U	T	A	S	C	D	P	STK	PTK	B	M	ALL	Mean	MSRp	MSRv	SMTe	OnWN	SMTn
fvec model	●											.7060	.6087	.6080	.8390	.2540	.6820	.4470
		●										.7589	.6863	.6814	.8637	.4950	.7091	.5395
	●	●										.8079	.7161	.7134	.8837	.5519	.7343	.5607
	●	●	●									.8187	.7137	.7157	.8833	.5131	.7355	.5809
TK models with STK and PTK	●	●	●	●				●				.8261	.6982	.7026	.8870	.4807	.7258	.5333
	●	●	●		●			●				.8326	.6970	.7020	.8925	.4826	.7190	.5253
	●	●	●			●		●				.8341	.7024	.7086	.8921	.4671	.7319	.5495
	●	●	●				●	●				.8211	.6693	.6994	.8903	.2980	.7035	.5603
	●	●	●	●					●			.8362	.7026	.6927	.8896	.5282	.7144	.5485
	●	●	●		●				●			.8458	.7047	.6935	.8953	.5080	.7101	.5834
	●	●	●			●			●			.8468	.6954	.6717	.8902	.4652	.7089	.6133
	●	●	●				●		●			.8326	.6693	.7108	.8879	.4922	.7215	.5156
REL tag	●	●	●		○							.8218	.6899	.6644	.8726	.4846	.7228	.5684
	●	●	●			○						.8250	.7000	.6806	.8822	.5171	.7145	.5769
domain adaptation	●	●	●		●					●		.8539	.7132	.6993	.9005	.4772	.7189	.6481
	●	●	●			●				●		.8529	.7249	.7080	.8984	.5142	.7263	.6700
	●	●	●		●	●				●		.8546	.7156	.6989	.8979	.4884	.7181	.6609
	●	●	●		●	●					●	.8810	.7416	.7210	.8971	.5912	.7328	.6778
	UKP (best system of STS-2012)											.8239	.6773	.6830	.8739	.5280	.6641	.4937

Table 1: Results on STS-2012. First set of experiments studies the combination of `fvec` models from UKP (U), Takelab (T) and (A). Next we show results for four structural representations: shallow (S), constituency (C), dependency (D) and phrase-dependency (P) trees with STK and PTK; next row set demonstrates the necessity of relational linking for two best structures, i.e. C and D (empty circle denotes a structures with no relational linking.); finally, domain adaptation via bags of features (B) of the entire pair and (M) manually encoded dataset type show the state of the art results.

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 **SEM*.

Daniel Bar, Chris Biemann, Iryna Gurevych, and Torsten Zesch. 2012. Ukp: Computing semantic textual similarity by combining multiple content similarity measures. In *SemEval*.

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*.

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*.

Michael Heilman and Noah A. Smith. 2010. Tree edit models for recognizing textual entailments, paraphrases, and answers to questions. In *NAACL*.

Alessandro Moschitti and Silvia Quarteroni. 2008. Kernels on linguistic structures for answer extraction. In *ACL*.

Alessandro Moschitti and Fabio Massimo Zanzotto. 2007. Fast and effective kernels for relational learning from texts. In *ICML*.

Alessandro Moschitti, Silvia Quarteroni, Roberto Basili, and Suresh Manandhar. 2007. Exploiting syntactic and shallow semantic kernels for question/answer classification. In *ACL*.

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

Alessandro Moschitti. 2008. Kernel methods, syntax and semantics for relational text categorization. In *CIKM*.

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

Frane Šarić, Goran Glavaš, Mladen Karan, Jan Šnajder, and Bojana Dalbelo Bašić. 2012. Takelab: Systems for measuring semantic text similarity. In *SemEval*.

Mengqiu Wang and Christopher D. Manning. 2010. Probabilistic tree-edit models with structured latent variables for textual entailment and question answering. In *ACL*.

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

Yuanbin Wu, Qi Zhang, Xuanjing Huang, and Lide Wu. 2009. Phrase dependency parsing for opinion mining. In *EMNLP*.

Fabio Massimo Zanzotto and Alessandro Moschitti. 2006. Automatic Learning of Textual Entailments with Cross-Pair Similarities. In *ACL*.