Tree Kernels for Machine Translation Quality Estimation

Christian Hardmeier and **Joakim Nivre** and **Jörg Tiedemann**
Uppsala University
Department of Linguistics and Philology
Box 635, 751 26 Uppsala, Sweden
firstname.lastname@lingfil.uu.se

Abstract

This paper describes Uppsala University's submissions to the Quality Estimation (QE) shared task at WMT 2012. We present a QE system based on Support Vector Machine regression, using a number of explicitly defined features extracted from the Machine Translation input, output and models in combination with tree kernels over constituency and dependency parse trees for the input and output sentences. We confirm earlier results suggesting that tree kernels can be a useful tool for QE system construction especially in the early stages of system design.

1 Introduction

The goal of the WMT 2012 Quality Estimation (QE) shared task (Callison-Burch et al., 2012) was to create automatic systems to judge the quality of the translations produced by a Statistical Machine Translation (SMT) system given the input text, the proposed translations and information about the models used by the SMT system. The shared task organisers provided a training set of 1832 sentences drawn from earlier WMT Machine Translation test sets, translated from English to Spanish with a phrase-based SMT system, along with the models used and diagnostic output produced by the SMT system as well as manual translation quality annotations on a 1–5 scale for each sentence. Additionally, a set of 17 baseline features was made available to the participants. Systems were evaluated on a test set of 422 sentences annotated in the same way.

Uppsala University submitted two systems to this shared task. Our systems were fairly successful and achieved results that were outperformed by only one competing group. They improve over the baseline performance in two ways, building on and extending earlier work by Hardmeier (2011), on which the system description in the following sections is partly based: On the one hand, we enhance the set of 17 baseline features provided by the organisers with another 82 explicitly defined features. On the other hand, we use syntactic tree kernels to extract implicit features from constituency and dependency parse trees over the input sentences and the Machine Translation (MT) output. The experimental results confirm the findings of our earlier work, showing tree kernels to be a valuable tool for rapid prototyping of QE systems.

2 Features

Our QE systems used two types of features: On the one hand, we used a set of *explicit features* that were extracted from the data before running the Machine Learning (ML) component. On the other hand, syntactic parse trees of the MT input and output sentences provided *implicit features* that were computed directly by the ML component using tree kernels.

2.1 Explicit features

Both of the QE systems we submitted to the shared task used the complete set of 17 baseline features provided by the workshop organisers. Additionally, the `UU_best` system also contained all the features presented by Hardmeier (2011) with the exception

Proceedings of the 7th Workshop on Statistical Machine Translation, pages 109–113,
Montréal, Canada, June 7-8, 2012.

of a few features specific to the film subtitle genre and inapplicable to the text type of the shared task, as well as a small number of features not included in that work. Many of these features were modelled on QE features described by Specia et al. (2009). In particular, the following features were included in addition to the baseline feature set:

- number of words, length ratio (4 features)
- source and target type-token ratios (2 features)
- number of tokens matching particular patterns (3 features each):
  - numbers
  - opening and closing parentheses
  - strong punctuation signs
  - weak punctuation signs
  - ellipsis signs
  - hyphens
  - single and double quotes
  - apostrophe-s tokens
  - short alphabetic tokens (≤ 3 letters)
  - long alphabetic tokens (≥ 4 letters)
- source and target language model (LM) and log-LM scores (4 features)
- LM and log-LM scores normalised by sentence length (4 features)
- number and percentage of out-of-vocabulary words (2 features)
- percentage of source 1-, 2-, 3- and 4-grams occurring in the source part of the training corpus (4 features)
- percentage of source 1-, 2-, 3- and 4-grams in each frequency quartile of the training corpus (16 features)
- a binary feature indicating that the output contains more than three times as many alphabetic tokens as the input (1 feature)
- percentage of unaligned words and words with $1:1$, $1:n$, $n:1$ and $m:n$ alignments (10 features)
- average number of translations per word, unweighted and weighted by word frequency and reciprocal word frequency (3 features)
- translation model entropy for the input words, cumulatively per sentence and averaged per word, computed based on the SMT lexical weight model (2 features).

Whenever applicable, features were computed for both the source and the target language, and additional features were added to represent the squared difference of the source and target language feature values. All feature values were scaled so that their values ranged between 0 and 1 over the training set.

The total number of features of the `UU_best` system amounted to 99. It should be noted, however, that there is considerable redundancy in the feature set and that the 82 features of Hardmeier (2011) overlap with the 17 baseline features to some extent. We did not make any attempt to reduce feature overlap and relied on the learning algorithm for feature selection.

2.2 Parse trees

Both the English input text and the Spanish Machine Translations were annotated with syntactic parse trees from which to derive implicit features. In English, we were able to produce both constituency and dependency parses. In Spanish, we were limited to dependency parses because of the better availability of parsing models. English constituency parses were produced with the Stanford parser (Klein and Manning, 2003) using the model bundled with the parser. For dependency parsing, we used MaltParser (Nivre et al., 2006). POS tagging was done with HunPOS (Halácsy et al., 2007) for English and SVMTool (Giménez and Márquez, 2004) for Spanish, with the models provided by the OPUS project (Tiedemann, 2009). As in previous work (Hardmeier, 2011), we treated the parser as a black box and made no attempt to handle the fact that parsing accuracy may be decreased over malformed SMT output.

To be used with tree kernels, the output of the dependency parser had to be transformed into a single tree structure with a unique label per node and unlabelled edges, similar to a constituency parse tree. We followed Johansson and Moschitti (2010) in using a tree representation which encodes part-of-speech tags, dependency relations and words as sequences of child nodes (see fig. 1).

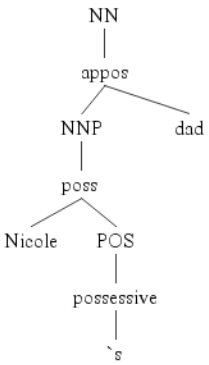
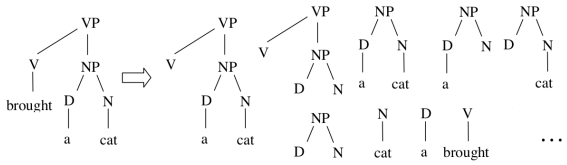
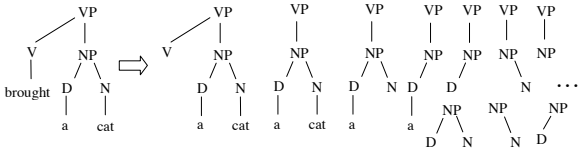


Figure 1: Representation of the dependency tree fragment for the words *Nicole 's dad*



A tree and some of its Subset Tree Fragments



A tree and some of its Partial Tree Fragments

Figure 2: Tree fragments extracted by the Subset Tree Kernel and by the Partial Tree Kernel. Illustrations by Moschitti (2006a).

3 Machine Learning component

3.1 Overview

The QE shared task asked both for an estimate of a 1–5 quality score for each segment in the test set and for a ranking of the sentences according to quality. We decided to treat score estimation as primary and address the task as a regression problem. For the ranking task, we simply submitted the ranking induced by the regression output, breaking ties randomly.

Our system was based on SVM regression as implemented by the SVMlight software (Joachims, 1999) with tree kernel extensions (Moschitti, 2006b). Predicted scores less than 1 were set to 1 and predicted scores greater than 5 were set to 5 as this was known to be the range of valid scores. Our learning algorithm had some free hyperparameters. Three of them were optimised by joint grid search with 5-fold cross-validation over the training set: the SVM training error/margin trade-off (C parameter), one free parameter of the explicit feature kernel and the ratio between explicit feature and tree kernels (see below). All other parameters were left at their default values. Before running it over the test set, the system was retrained on the complete training set using the parameters found with cross-validation.

3.2 Kernels for explicit features

To select a good kernel for our explicit features, we initially followed the advice given by Hsu et al. (2010), using a Gaussian RBF kernel and optimising the SVM C parameter and the γ parameter of the RBF with grid search. While this gave reasonable results, it turned out that slightly better prediction could be achieved by using a polynomial kernel, so we chose to use this kernel for our final submission and used grid search to tune the degree of the polynomial instead. The improvement over the Gaussian kernel was, however, marginal.

3.3 Tree kernels

To exploit parse tree information in our Machine Learning (ML) component, we used tree kernel functions. Tree kernels (Collins and Duffy, 2001) are kernel functions defined over pairs of tree structures. They measure the similarity between two trees by counting the number of common substructures. Implicitly, they define an infinite-dimensional feature space whose dimensions correspond to all possible tree fragments. Features are thus available to cover different kinds of abstract node configurations that can occur in a tree. The important feature dimensions are effectively selected by the SVM training algorithm through the selection and weighting of the support vectors. The intuition behind our use of tree kernels is that they may help us identify constructions that are difficult to translate in the source language, and doubtful syntactic structures in the output language. Note that we do not currently compare parse trees across languages; tree kernels

					Cross-validation				*Test set*			
	Features	T	C	d	Δ	ρ	MAE	RMS	Δ	ρ	MAE	RMS
`UU_best`	99 explicit + TK	0.05	4	2	0.506	0.566	0.550	0.692	0.56	0.62	0.64	0.79
(a)	99 explicit + TK	0.03	8	3	0.502	0.564	0.552	0.700	0.56	0.61	0.63	0.78
(b)	17 explicit + TK	0.05	4	2	0.462	0.530	0.568	0.714	0.57	0.61	0.65	0.79
`UU_bltk`	17 explicit + TK	0.03	8	3	0.466	0.534	0.566	0.712	0.58	0.61	0.64	0.79
(c)	99 explicit	0	8	2	0.492	0.560	0.554	0.700	0.56	0.59	0.65	0.80
(d)	17 explicit	0	8	2	0.422	0.466	0.598	0.748	0.52	0.55	0.70	0.83
(e)	TK only	–	4	–	0.364	0.392	0.632	0.782	0.51	0.51	0.70	0.85

T: Tree kernel weight C: Training error/margin trade-off d: Degree of polynomial kernel
Δ: DeltaAvg score ρ: Spearman rank correlation MAE: Mean Average Error
RMS: Root Mean Square Error TK: Tree kernels

Table 1: Experimental results

are applied to trees of the same type in the same language only.

We used two different types of tree kernels for the different types of parse trees (see fig. 2). The Subset Tree Kernel (Collins and Duffy, 2001) considers tree fragments consisting of more than one node with the restriction that if one child of a node is included, then all its siblings must be included as well so that the underlying production rule is completely represented. This kind of kernel is well suited for constituency parse trees and was used for the source language constituency parses. For the dependency trees, we used the Partial Tree Kernel (Moschitti, 2006a) instead. It extends the Subset Tree Kernel by permitting also the extraction of tree fragments comprising only part of the children of any given node. Lifting this restriction makes sense for dependency trees since a node and its children do not correspond to a grammatical production in a dependency tree in the same way as they do in a constituency tree (Moschitti, 2006a). It was used for the dependency trees in the source and in the target language.

The explicit feature kernel and the three tree kernels were combined additively, with a single weight parameter to balance the sum of the tree kernels against the explicit feature kernel. This coefficient was optimised together with the other two hyperparameters mentioned above. It turned out that best results could be obtained with a fairly low weight for the tree kernels, but in the cross-validation experiments adding tree kernels did give an improvement over not having them at all.

4 Experimental Results

Results for some of our experiments are shown in table 1. The two systems we submitted to the shared task are marked with their system identifiers. A few other systems are included for comparison and are numbered (a) to (e) for easier reference.

Our system using only the baseline features (d) performs a bit worse than the reference system of the shared task organisers. We use the same learning algorithm, so this seems to indicate that the kernel and the hyperparameters they selected worked slightly better than our choices. Using only tree kernels with no explicit features at all (e) creates a system that works considerably worse under cross-validation, however we note that its performance on the test set is very close to that of system (d).

Adding the 82 additional features of Hardmeier (2011) to the system without tree kernels slightly improves the performance both under cross-validation and on the test set (c). Adding tree kernels has a similar effect, which is a bit less pronounced for the cross-validation setting, but quite comparable on the test set (`UU_bltk`, b). Finally, combining the full feature set with tree kernels results in an additional gain under cross-validation, but unfortunately the improvement does not carry over to the test set (`UU_best`, a).

5 Conclusions

In sum, the results confirm the findings made in our earlier work (Hardmeier, 2011). They show that tree kernels can be a valuable tool to boost the initial

performance of a Quality Estimation system without spending much effort on feature engineering. Unfortunately, it seems that the gains achieved by tree kernels over simple parse trees and by the additional explicit features used in our systems do not necessarily add up. Nevertheless, comparison with other participating systems shows that either of them is sufficient for state-of-the-art performance.

References

Chris Callison-Burch, Philipp Koehn, Christof Monz, Matt Post, Radu Soricut, and Lucia Specia. 2012. Findings of the 2012 Workshop on Statistical Machine Translation. In *Proceedings of the Seventh Workshop on Statistical Machine Translation*, Montreal, Canada, June. Association for Computational Linguistics.

Michael Collins and Nigel Duffy. 2001. Convolution kernels for natural language. In *Proceedings of NIPS 2001*, pages 625–632.

Jesús Giménez and Lluís Márquez. 2004. SVMTool: A general POS tagger generator based on Support Vector Machines. In *Proceedings of the 4th Conference on International Language Resources and Evaluation (LREC-2004)*, Lisbon.

Péter Halácsy, András Kornai, and Csaba Oravecz. 2007. HunPos – an open source trigram tagger. In *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics. Companion Volume: Proceedings of the Demo and Poster Sessions*, pages 209–212, Prague, Czech Republic, June. Association for Computational Linguistics.

Christian Hardmeier. 2011. Improving machine translation quality prediction with syntactic tree kernels. In Mikel L. Forcada, Heidi Depraetere, and Vincent Vandeghinste, editors, *Proceedings of the 15th conference of the European Association for Machine Translation (EAMT 2011)*, pages 233–240, Leuven, Belgium.

Chih-Wei Hsu, Chih-Chung Chang, and Chih-Jen Lin. 2010. A practical guide to support vector classification. Technical report, Department of Computer Science, National Taiwan University.

Thorsten Joachims. 1999. Making large-scale SVM learning practical. In B. Schölkopf, C. Burges, and A. Smola, editors, *Advances in Kernel Methods – Support Vector Learning*. MIT Press.

Richard Johansson and Alessandro Moschitti. 2010. Syntactic and semantic structure for opinion expression detection. In *Proceedings of the Fourteenth Conference on Computational Natural Language Learning*, pages 67–76, Uppsala, Sweden, July. Association for Computational Linguistics.

Dan Klein and Christopher D. Manning. 2003. Accurate unlexicalized parsing. In *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics*, pages 423–430, Sapporo, Japan, July. Association for Computational Linguistics.

Alessandro Moschitti. 2006a. Efficient convolution kernels for dependency and constituent syntactic trees. In *Proceedings of the 17th European Conference on Machine Learning*, Berlin.

Alessandro Moschitti. 2006b. Making tree kernels practical for natural language learning. In *Proceedings of the Eleventh International Conference of the European Association for Computational Linguistics*, Trento.

Joakim Nivre, Johan Hall, and Jens Nilsson. 2006. MaltParser: A language-independent system for data-driven dependency parsing. In *Proceedings of the 5th Conference on International Language Resources and Evaluation (LREC-2006)*, pages 2216–2219, Genoa.

Lucia Specia, Craig Saunders, Marco Turchi, Zhuoran Wang, and John Shawe-Taylor. 2009. Improving the confidence of Machine Translation quality estimates. In *Proceedings of MT Summit XII*, Ottawa.

Jörg Tiedemann. 2009. News from OPUS – a collection of multilingual parallel corpora with tools and interface. In N. Nicolov, K. Bontcheva, G. Angelova, and R. Mitkov, editors, *Recent Advances in Natural Language Processing*, pages 237–248. John Benjamins, Amsterdam.