Turku: Broad-Coverage Semantic Parsing with Rich Features

Jenna Kanerva[*]
Department of Information Technology
University of Turku
Finland
jmnybl@utu.fi

Juhani Luotolahti[*]
Department of Information Technology
University of Turku
Finland
mjluot@utu.fi

Filip Ginter
Department of Information Technology
University of Turku
Finland
figint@utu.fi

Abstract

In this paper we introduce our system capable of producing semantic parses of sentences using three different annotation formats. The system was used to participate in the SemEval-2014 Shared Task on broad-coverage semantic dependency parsing and it was ranked third with an overall F_1-score of 80.49%. The system has a pipeline architecture, consisting of three separate supervised classification steps.

1 Introduction

In the SemEval-2014 Task 8 on semantic parsing, the objective is to extract for each sentence a rich set of typed semantic dependencies in three different formats: *DM*, *PAS* and *PCEDT*. These formats differ substantially both in the assignment of semantic heads as well as in the lexicon of semantic dependency types. In the open track of the shared task, participants were encouraged to use all resources and tools also beyond the provided training data. To improve the comparability of the systems, the organizers provided ready-to-use dependency parses produced using the state-of-the-art parser of Bohnet and Nivre (2012).

In this paper we describe our entry in the open track of the shared task. Our system is a pipeline of three support vector machine classifiers trained separately for detecting semantic dependencies, assigning their roles, and selecting the top nodes of semantic graphs. In this, we loosely follow the architecture of e.g. the TEES (Björne et al., 2012) and EventMine (Miwa et al., 2012) systems, which were found to be effective in the structurally related task of biomedical event extraction. Similar classification approach is shown to be effective also in semantic parsing by e.g. Zhao et al. (2009), the winner of the CoNLL'09 Shared Task on Syntactic and Semantic Dependencies in Multiple Languages (SRL-only subtask) (Hajič et al., 2009), where semantic parsing is approached as a word-pair classification problem and semantic arguments and their roles are predicted simultaneously. In preliminary experiments, we also developed a joint approach to simultaneously identify semantic dependencies and assign their roles, but found that the performance of the joint prediction was substantially worse than for the current pipeline approach. As the source of features, we rely heavily on the syntactic parses as well as other external resources such as vector space representations of words and large-scale syntactic n-gram statistics.

In the following sections, we describe the three individual classification steps of our semantic parsing pipeline.

2 Detecting Semantic Dependencies

The first step of our semantic parsing pipeline is to detect semantic dependencies, i.e. governor-dependent pairs which has a semantic relation between them. The first stage covers only the identification of such dependencies; the labels describing the semantic roles of the dependents are assigned in a later stage.

The semantic dependencies are identified using a binary support vector machine classifier from the *LIBSVM* package (Chang and Lin, 2011). Each possible combination of two tokens in the sentence is considered to be a candidate for a semantic dependency in both directions, and thus also included as a training example. No beforehand pruning of possible candidates is performed during training. However, we correct for the overwhelming number of negative training examples

[*]These authors contributed equally.

by setting the weights of positive and negative examples used during training, so as to maximize the unlabeled F_1-score on the development set.

Increasing the recall of semantic dependency detection can be beneficial for the overall performance of the pipeline system, since a candidate lost in the dependency detection stage cannot be recovered later. We therefore tested the approach applied, among others by Björne et al. (2012), whereby the dependency detection stage heavily overgenerates candidates and the next stage in the pipeline is given the option to predict a negative label, thus removing a candidate dependency. In preliminary experiments we tried to explicitly overgenerate the dependency candidates by altering the classifier threshold, but noticed that heavy overgeneration of positive examples leads to a decreased performance in the role assigning stage. Instead, the above-mentioned optimization of the example weights during training results in a classifier which overgenerates positive examples by 4.4%, achieving the same objective and improving the overall performance of the system.

Features used during the dependency identification are derived from tokens and the syntactic parse trees provided by the organizers. Our primary source of features are the syntactic trees, since 73.2% of semantic dependencies have a corresponding undirected syntactic dependency in the parse tree. Further, the syntactic dependency path between the governor and the dependent is shorter than their linear distance in 48.8% of cases (in 43.4% of cases the distance is the same). The final feature set used in the identification is optimized by training models with different combinations of features and selecting the best combination based on performance on the held-out development set. Interestingly, the highest performance is achieved with a rather small set of features, whose full listing is shown in Table 1. The feature vectors are normalized to unit length prior to classification and the SVM regularization parameter c is optimized separately for each annotation format.

Feature	D	R	T
arg.pos	X	X	
arg.deptype	X	X	
arg.lemma	X	X	
pred.pos	X	X	X
pred.deptype	X	X	X
pred.lemma	X	X	X
pred.is_predicate		X	X
arg.issyntaxdep		X	
arg.issyntaxgov		X	
arg.issyntaxsibling		X	
path.length	X	X	
undirected_path.deptype	X	X	
directed_path.deptype	X	X	
undirected_path.pos	X	X	
extended_path.deptype	X	X	
simplified_path.deptype with len		X	
simplified_path.deptype wo len		X	
splitted_undirected_path.deptype	X		
arg.prev.pos	X	X	
arg.next.pos	X	X	
arg.prev+arg.pos	X	X	
arg.next+arg.pos	X	X	
arg.next+arg+arg.prev.pos	X	X	
pred.prev.pos	X	X	
pred.next.pos	X	X	
pred.prev+pred.pos	X	X	
pred.next+pred.pos	X	X	
pred.next+pred+pred.prev.pos	X	X	
linear_route.pos	X		
arg.child.pos		X	
arg.child.deptype		X	
arg.child.lemma		X	
pred.child.pos		X	
pred.child.deptype		X	X
pred.child.lemma		X	
syntaxgov.child.deptype		X	
vector_similarities		X	
n-gram_frequencies		X	
pred.sem_role			X
pred.child.sem_role			X
pred.syntaxsibling.deptype			X
pred.semanticsibling.sem_role			X

Table 1: Features used in the detection of semantic dependencies (D), assigning their roles (R) and top node detection (T). *path* refers to syntactic dependencies between the argument and the predicate, and *linear route* refers to all tokens between the argument and the predicate. In top node detection, where only one token is considered at a time, the *pred* is used to represent that token.

3 Role Assignment

After the semantic governor-dependent pairs are identified, the next step is to assign a role for each pair to constitute a full semantic dependency. This is done by training a multiclass support vector machine classifier implemented in the *SVM-multiclass* package by Joachims (1999). We iterate through all identified dependencies, and for each assign a role, or alternatively classify it as a negative example. This is to account for the 4.4% of overgenerated dependencies. However, the proportion of negative classifications should stay relatively low and to ensure this, we downsample the number of negative examples used in training to contain only 5% of all negative examples. The downsampling ratio is optimized on the development set using grid search and downsampled training instances are chosen randomly.

The basic features, shown in Table 1, follow the same style as in dependency identification. We also combine some of the basic features by creating all possible feature pairs in a given set, but do not perform this with the full set of features. In the open track, participants are also allowed to use additional data and tools beyond the official training data. In addition to the parse trees, we include also features utilizing syntactic n-gram frequencies and vector space similarities.

Google has recently released a large corpus of syntactic n-grams, a collection of dependency subtrees with frequency counts (Goldberg and Orwant, 2013). The syntactic n-grams are induced from the Google Books collection, a 350B token corpus of syntactically parsed text. In this work we are interested in *arcs*, which are $(governor, dependent, syntactic\ relation)$ triplets associated with their count.

For each governor-dependent pair, we generate a set of n-gram features by iterating through all known dependency types and searching from the syntactic n-grams how many times (if any) the governor-dependent pair with the particular dependency type is seen. A separate feature is then created for each dependency type and the counts are encoded in feature weights compressed using $w = \log_{10}(count)$. This approach gives us an opportunity to include statistical information about word relations induced from a very large corpus. Information is captured also outside the particular syntactic context, as we iterate through all known dependency types during the process.

Another source of additional data used in role classification is a publicly available Google News vector space model[1] representing word similarities. The vector space model is induced from the Google News corpus with the *word2vec* software (Mikolov et al., 2013) and negative sampling architecture, and each vector have 300 dimensions. The vector space representation gives us an opportunity to measure word similarities using the standard cosine similarity function.

The approach to transforming the vector representations into features varies with the three different annotation formats. On *DM* and *PAS*, we follow the method of Kanerva and Ginter (2014), where for each role an average argument vector is calculated. This is done by averaging all word vectors seen in the training data as arguments for the given predicate with a particular role. For each candidate argument, we can then establish a set of similarity values to each possible role by taking the cosine similarity of the argument vector to the role-wise average vectors. These similarities are then turned into separate features, where the similarity values are encoded as feature weights.

On *PCEDT*, preliminary experiments showed that the best strategy to include word vectors into classification is by turning them directly into features, so that each dimension of the word vector is represented as a separate feature. Thus, we iterate through all 300 vector dimensions and create a separate feature representing the position and value of a particular dimension. Values are again encoded in feature weights. These features are created separately for both the argument and the predicate. The word vectors are pre-normalized to unit length, so no additional normalization of feature weights is needed.

Both the n-gram– and vector similarities–based features give a modest improvement to the classification performance.

4 Detecting Top Nodes

The last step in the pipeline is the detection of *top nodes*. A top node is the semantic head or the structural root of the sentence. Typically each sentence annotated in the *DM* and *PAS* formats contains one top node, whereas *PCEDT* sentences have on average 1.12 top nodes per sentence.

As in the two previous stages, we predict top nodes by training a support vector machine classifier, with each token being considered a candidate. Because the top node prediction is the last step performed, in addition to the basic information available in the two previous steps, we are able to use also predicted arguments as features. Otherwise, the feature set used in top node detection follows the same style as in the two previous

[1]`https://code.google.com/p/word2vec/`

	LP	LR	LF	UF
DM	80.94	82.14	81.53	83.48
PAS	87.33	87.76	87.54	88.97
PCEDT	72.42	72.37	72.40	85.86
Overall	80.23	80.76	80.49	86.10

Table 2: Overall scores of whole task as well as separately for each annotation format in terms of labeled precision (LP), recall (LR) and F_1-score (LF) as well as unlabeled F_1-score (UF).

tasks, but is substantially smaller (see Table 1). We also create all possible feature pairs prior to classification to simulate the use of a second-degree polynomial kernel.

For each token in the sentence, we predict whether it is a top node or not. However, in *DM* and *PAS*, where typically only one top node is allowed, we choose only the token with the maximum positive value to be the final top node. In *PCEDT*, we simply let all positive predictions act as top nodes.

5 Results

The primary evaluation measure is the labeled F_1-score of the predicted dependencies, where the identification of top nodes is incorporated as an additional dummy dependency. The overall semantic F_1-score of our system is 80.49%. The prediction performance in *DM* is 81.53%, in *PAS* 87.54% and in *PCEDT* 72.40%. The top nodes are identified with an overall F_1-score of 87.05%. The unlabeled F_1-score reflects the performance of the dependency detection in isolation from labeling task and by comparing the labeled and unlabeled F_1-scores from Table 2 we can see that the most common mistake relates to the identification of correct governor-dependent pairs. This is especially true with the *DM* and *PAS* formats where the difference between labeled and unlabeled scores is very small (1.9pp and 1.4pp), reflecting high performance in assigning the roles. Instead, in *PCEDT* the role assignment accuracy is substantially below the other two and the difference between unlabeled and labeled F_1-score is as much as 13.5pp. One likely reason is the higher number of possible roles defined in the *PCEDT* format.

5.1 Discussion

Naturally, our system generally performs better with frequently seen semantic roles than roles that are seen rarely. In the case of *DM*, the 4 most common semantic roles cover over 87% of the gold standard dependencies and are predicted with a macro F_1-score of 85.3%, while the remaining 35 dependency labels found in the gold standard are predicted at an average rate of 49.4%. To give this a perspective, the most common 4 roles have on average 121K training instances, while the remaining 35 roles have on average about 2000 training instances. For *PAS*, the 9 most common labels, which comprise over 80% of all dependencies in the gold standard data and have on average about 66K training instances per role, are predicted with an F_1-score of 87.6%, while the remaining 32 labels have on average 4200 training instance and are predicted with an F_1-score of 57.8%. The *PCEDT* format has the highest number of possible semantic roles and also lowest correlation between the frequency in training data and F_1-score. For *PCEDT*, the 11 most common labels, which cover over 80% of all dependencies in the gold standard, are predicted with an F_1-score of 69.6%, while the remaining 53 roles are predicted at an average rate of 46.6%. The higher number of roles also naturally affects the number of training instances and the 11 most common labels in *PCEDT* have on average 35K training instances, while the remaining 53 roles have on average 1600 instances per role.

Similarly, the system performs better with semantic arguments which are nearby the governor. This is true for both linear distance between the two tokens and especially for distance measured by syntactic dependency steps. For example in the case of *DM*, semantic dependencies shorter than 3 steps in the syntactic tree cover more than 95% of the semantic dependencies in the gold standard and have an F_1-score of 75.1%, while the rest have only 32.6%. The same general pattern is also evident in the other formats.

6 Conclusion

In this paper we presented our system used to participate in the SemEval-2014 Shared Task on broad-coverage semantic dependency parsing. We built a pipeline of three supervised classifiers to identify semantic dependencies, assign a role for each dependency and finally, detect the top nodes.

In addition to basic features used in classification we have shown that additional information, such as frequencies of syntactic n-grams and word similarities derived from vector space representations, can also positively contribute to the classification performance.

The overall F_1-score of our system is 80.49% and it was ranked third in the open track of the shared task.

Acknowledgments

This work was supported by the Emil Aaltonen Foundation and the Kone Foundation. Computational resources were provided by CSC – IT Center for Science.

References

Jari Björne, Filip Ginter, and Tapio Salakoski. 2012. University of Turku in the BioNLP'11 shared task. *BMC Bioinformatics*, 13(Suppl 11):S4.

Bernd Bohnet and Joakim Nivre. 2012. A transition-based system for joint part-of-speech tagging and labeled non-projective dependency parsing. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 1455–1465.

Chih-Chung Chang and Chih-Jen Lin. 2011. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27.

Yoav Goldberg and Jon Orwant. 2013. A dataset of Syntactic-Ngrams over time from a very large corpus of English books. In *Second Joint Conference on Lexical and Computational Semantics (*SEM), Volume 1: Proceedings of the Main Conference and the Shared Task: Semantic Textual Similarity*, pages 241–247.

Jan Hajič, Massimiliano Ciaramita, Richard Johansson, Daisuke Kawahara, Maria Antònia Martí, Lluís Màrquez, Adam Meyers, Joakim Nivre, Sebastian Padó, Jan Štěpánek, et al. 2009. The CoNLL-2009 shared task: Syntactic and semantic dependencies in multiple languages. In *Proceedings of the Thirteenth Conference on Computational Natural Language Learning: Shared Task*, pages 1–18.

Thorsten Joachims. 1999. Making large-scale SVM learning practical. In *Advances in Kernel Methods - Support Vector Learning*, pages 169–184. MIT Press.

Jenna Kanerva and Filip Ginter. 2014. Post-hoc manipulations of vector space models with application to semantic role labeling. In *Proceedings of the 2nd Workshop on Continuous Vector Space Models and their Compositionality (CVSC)@ EACL*, pages 1–10.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. In *Workshop Proceedings of International Conference on Learning Representations*.

Makoto Miwa, Paul Thompson, John McNaught, Douglas Kell, and Sophia Ananiadou. 2012. Extracting semantically enriched events from biomedical literature. *BMC Bioinformatics*, 13(1):108.

Hai Zhao, Wenliang Chen, Chunyu Kit, and Guodong Zhou. 2009. Multilingual dependency learning: a huge feature engineering method to semantic dependency parsing. In *Proceedings of the Thirteenth Conference on Computational Natural Language Learning: Shared Task*, pages 55–60.