Transition-based parsing with Confidence-Weighted Classification

Martin Haulrich
Dept. of International Language Studies and Computational Linguistics
Copenhagen Business School
mwh.isv@cbs.dk

Abstract

We show that using confidence-weighted classification in transition-based parsing gives results comparable to using SVMs with faster training and parsing time. We also compare with other online learning algorithms and investigate the effect of pruning features when using confidence-weighted classification.

1 Introduction

There has been a lot of work on data-driven dependency parsing. The two dominating approaches have been graph-based parsing, e.g. MST-parsing (McDonald et al., 2005b) and transition-based parsing, e.g. the MaltParser (Nivre et al., 2006a). These two approaches differ radically but have in common that the best results have been obtained using margin-based machine learning approaches. For the MST-parsing MIRA (McDonald et al., 2005a; McDonald and Pereira, 2006) and for transition-based parsing Support-Vector Machines (Hall et al., 2006; Nivre et al., 2006b).

Dredze et al. (2008) introduce a new approach to margin-based online learning called *confidence-weighted classification* (CW) and show that the performance of this approach is comparable to that of Support-Vector Machines. In this work we use confidence-weighted classification with transition-based parsing and show that this leads to results comparable to the state-of-the-art results obtained using SVMs.

We also compare training time and the effect of pruning when using confidence-weighted learning.

2 Transition-based parsing

Transition-based parsing builds on the idea that parsing can be viewed as a sequence of transitions between states. A transition-based parser (deterministic classifier-based parser) consists of three essential components (Nivre, 2008):

1. A parsing algorithm
2. A feature model
3. A classifier

The focus here is on the classifier but we will briefly describe the parsing algorithm in order to understand the classification task better.

The parsing algorithm consists of two components, a *transition system* and an *oracle*. Nivre (2008) defines a transition system $S = (C, T, c_s, C_t)$ in the following way:

1. C is a set of configurations, each of which contains a buffer β of (remaining) nodes and a set A of dependency arcs,
2. T is a set of transitions, each of which is a partial function $t : C \rightarrow C$,
3. c_s is a initialization function mapping a sentence $x = (w_0, w_1, \ldots, w_n)$ to a configuration with $\beta = [1, \ldots, n]$,
4. C_t is a set of terminal configurations.

A *transition sequence* for a sentence x in S is a sequence $C_{0,m} = (c_0, c_1 \ldots, c_m)$ of configurations, such that

1. $c_0 = c_s(x)$,
2. $c_m \in C_t$,
3. for every i $(1 \leq i \leq m) c_i = t(c_{i-1})$ for some $t \in T$

The oracle is used during training to determine a transition sequence that leads to the correct parse. The job of the classifier is to 'imitate' the oracle, i.e. to try to always pick the transitions that lead to the correct parse. The information given to the classifier is the current configuration. Therefore the training data for the classifier consists of a number of configurations and the transitions the oracle chose with these configurations.

Here we focus on stack-based parsing algorithms. A stack-based configuration for a sentence $x = (w_0, w_1, \ldots, w_n)$ is a triple $c = (\sigma, \beta, A)$, where

1. σ is a stack of tokens $i \leq k$ (for some $k \leq n$),
2. β is a buffer of tokens $j > k$,
3. A is a set of dependency arcs such that $G = (0, 1, \ldots, n, A)$ is a dependency graph for x. (Nivre, 2008)

In the work presented here we use the NivreEager algorithm which has four transitions:

Shift Push the token at the head of the buffer onto the stack.

Reduce Pop the token on the top of the stack.

Left-Arc$_l$ Add to the analysis an arc with label l from the token at the head of the buffer to the token on the top of the stack, and push the buffer-token onto the stack.

Right-Arc$_l$ Add to the analysis an arc with label l from the token on the top of the stack to the token at the head of the buffer, and pop the stack.

2.1 Classification

Transition-based dependency parsing reduces parsing to consecutive multiclass classification. From each configuration one amongst some predefined number of transitions has to be chosen. This means that any classifier can be plugged into the system. The training instances are created by the oracle so the training is offline. So even though we use online learners in the experiments these are used in a batch setting.

The best results have been achieved using Support-Vector Machines placing the MaltParser very high in both the CoNNL shared tasks on dependency parsing in 2006 and 2007 (Buchholz and Marsi, 2006; Nivre et al., 2007) and it has been shown that SVMs are better for the task than Memory-based learning (Hall et al., 2006). The standard setting in the MaltParser is to use a 2nd-degree polynomial kernel with the SVM.

3 Confidence-weighted classification

Dredze et al. (2008) introduce confidence-weighted linear classifiers which are online-classifiers that maintain a confidence parameter for each weight and uses this to control how to change the weights in each update. A problem with online algorithms is that because they have no memory of previously seen examples they do not know if a given weight has been updated many times or few times. If a weight has been updated many times the current estimation of the weight is probably relatively good and therefore should not be changed too much. On the other hand if it has never been updated before the estimation is probably very bad. CW classification deals with this by having a confidence-parameter for each weight, modeled by a Gaussian distribution, and this parameter is used to make more aggressive updates on weights with lower confidence (Dredze et al., 2008). The classifiers also use Passive-Aggressive updates (Crammer et al., 2006) to try to maximize the margin between positive and negative training instances.

CW classifiers are online-algorithms and are therefore fast to train, and it is not necessary to keep all training examples in memory. Despite this they perform as well or better than SVMs (Dredze et al., 2008). Crammer et al. (2009) extend the approach to multiclass classification and show that also in this setting the classifiers often outperform SVMs. They show that updating only the weights of the best of the wrongly classified classes yields the best results. We also use this approach, called top-1, here.

Crammer et al. (2008) present different update-rules for CW classification and show that the ones based on standard deviation rather than variance yield the best results. Our experiments have confirmed this, so in all experiments the update-rule from equation 10 (Crammer et al., 2008) is used.

4 Experiments

4.1 Software

We use the open-source parser MaltParser[1] for all experiments. We have integrated confidence-weighted, perceptron and MIRA classifiers into the code. The code for the online classifiers has

[1]We have used version 1.3.1, available at `maltparser.org`

been made available by the authors of the CW-papers.

4.2 Data

We have used the 10 smallest data sets from CoNNL-X (Buchholz and Marsi, 2006) in our experiments. Evaluation has been done with the official evaluation script and evaluation data from this task.

4.3 Features

The standard setting for the MaltParser is to use SVMs with polynomial kernels, and because of this it uses a relatively small number of features. In most of our experiments the default feature set of MaltParser consisting of 14 features has been used.

When using a linear-classifier without a kernel we need to extend the feature set in order to achieve good results. We have done this very uncritically by adding all pair wise combinations of all features. This leads to 91 additional features when using the standard 14 features.

5 Results and discussion

We will now discuss various results of our experiments with using CW-classifiers in transition-based parsing.

5.1 Online classifiers

We compare CW-classifiers with other online algorithms for linear classification. We compare with perceptron (Rosenblatt, 1958) and MIRA (Crammer et al., 2006). With both these classifiers we use the same top-1 approach as with the CW-classifers and also averaging which has been shown to alleviate overfitting (Collins, 2002). Table 2 shows Labeled Attachment Score obtained with the three online classifiers. All classifiers were trained with 10 iterations.

These results confirm those by Crammer et al. (2009) and show that confidence-weighted classifiers are better than both perceptron and MIRA.

5.2 Training and parsing time

The training time of the CW-classifiers depends on the number of iterations used, and this of course affects the accuracy of the parser. Figure 1 shows Labeled Attachment Score as a function of the number of iterations used in training. The horizontal line shows the LAS obtained with SVM.

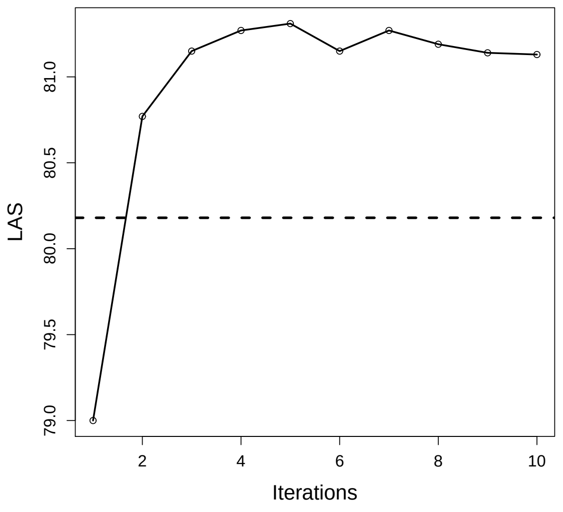

Figure 1: LAS as a function of number of training iterations on Danish data. The dotted horizontal line shows the performance of the parser trained with SVM.

We see that after 4 iterations the CW-classifier has the best performance for the data set (Danish) used in this experiment. In most experiments we have used 10 iterations. Table 1 compares training time (10 iterations) and parsing time of a parser using a CW-classifiers and a parser using SVM on the same data set. We see that training of the CW-classifier is faster, which is to be expected given their online-nature. We also see that parsing is much faster.

	SVM	CW
Training	75 min	8 min
Parsing	29 min	1.5 min

Table 1: Training and parsing time on Danish data.

5.3 Pruning features

Because we explicitly represent pair wise combinations of all of the original features we get an extremely high number of binary features. For some of the larger data sets, the number of features is so big that we cannot hold the weight-vector in memory. For instance the Czech data-set has 16 million binary features, and almost 800 classes - which means that in practice there are 12 billion binary features[2].

[2]Which is also why we only have used the 10 smallest data sets from CoNNL-X.

	Perceptron	MIRA	CW, manual fs	CW	SVM
Arabic	58.03	59.19	60.55	†**60.57**	59.93
Bulgarian	80.46	81.09	82.57	†**82.76**	82.12
Danish	79.42	79.90	81.06	†**81.13**	80.18
Dutch	75.75	77.47	77.65	†**78.65**	77.76
Japanese	87.74	88.06	88.14	88.19	†**89.47**
Portuguese	85.69	85.95	86.11	86.20	**86.25**
Slovene	64.35	65.38	66.09	†**66.28**	65.45
Spanish	74.06	74.86	75.58	**75.90**	75.46
Swedish	79.79	80.31	81.03	†**81.24**	80.56
Turkish	46.48	47.13	46.98	47.09	**47.49**
All	78.26	79.00	79.68	†**79.86**	79.59

Table 2: LAS on development data for three online classifers, CW-classifiers with manual feature selection and SVM. Statistical significance is measuered between CW-classifiers without feature selection and SVMs.

To solve this problem we have tried to use pruning to remove the features occurring fewest times in the training data. If a feature occurs fewer times than a given cutoff limit the feature is not included. This goes against the idea of CW classifiers which are exactly developed so that rare features can be used. Experiments also show that this pruning hurts accuracy. Figure 2 shows the labeled attachment score as a function of the cutoff limit on the Danish data.

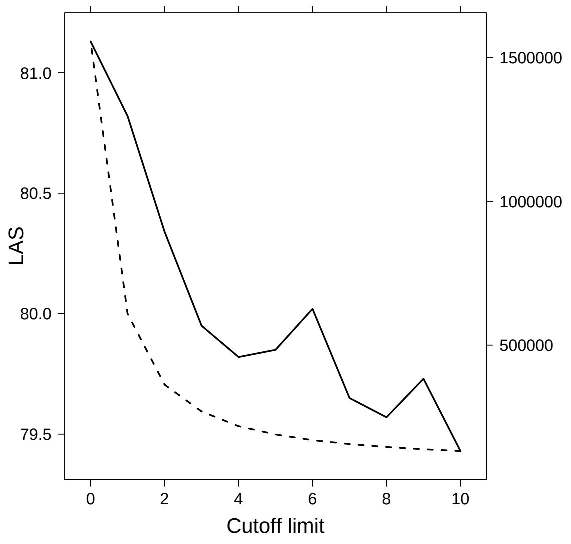

Figure 2: LAS as a function of the cutoff limit when pruning rare features. The dotted line shows the number of features left after pruning.

5.4 Manual feature selection

Instead of pruning the features we tried manually removing some of the pair wise feature combinations. We removed some of the combinations that lead to the most extra features, which is especially the case with combinations of lexical features. In the extended default feature set for instance we removed all combinations of lexical features except the combination of the word form of the token at the top of the stack and of the word form of the token at the head of the buffer.

Table 2 shows that this consistently leads to a small decreases in LAS.

5.5 Results without optimization

Table 2 shows the results for the 10 CoNNL-X data sets used. For comparison we have included the results from using the standard classifier in the MaltParser, i.e. SVM with a polynomial kernel. The hyper-parameters for the SVM have not been optimized, and neither has the number of iterations for the CW-classifiers, which is always 10. We see that in many cases the CW-classifier does significantly [3] better than the SVM, but that the opposite is also the case.

5.6 Results with optimization

The results presented above are suboptimal for the SVMs because default parameters have been used for these, and optimizing these can improve ac-

[3]In all tables statistical significance is marked with †. Significance is calculated using McNemar's test ($p = 0.05$). These tests were made with MaltEval (Nilsson and Nivre, 2008)

	SVM			CW		
	LAS	UAS	LA	LAS	UAS	LA
Arabic	66.71	77.52	80.34	**67.03**	77.52	†**81.20**
Bulgarian*	**87.41**	**91.72**	**90.44**	87.25	91.56	89.77
Danish	†**84.77**	†**89.80**	**89.16**	84.15	88.98	88.74
Dutch*	†**78.59**	†**81.35**	†**83.69**	77.21	80.21	82.63
Japanese	†**91.65**	†**93.10**	†**94.34**	90.41	91.96	93.34
Portuguese*	†**87.60**	†**91.22**	†**91.54**	86.66	90.58	90.34
Slovene	**70.30**	78.72	**80.54**	69.84	†**79.62**	79.42
Spanish	81.29	84.67	90.06	**82.09**	†**85.55**	**90.52**
Swedish*	†**84.58**	**89.50**	**87.39**	83.69	89.11	87.01
Turkish	†**65.68**	†**75.82**	†**78.49**	62.00	73.15	76.12
All	†**79.86**	†**85.35**	†**86.60**	79.04	84.83	85.91

Table 3: Results on the CoNNL-X evaluation data. Manuel feature selection has been used for languages marked with an *.

curacy a lot. In this section we will compare results obtained with CW-classifiers with the results for the MaltParser from CoNNL-X. In CoNNL-X both the hyper parameters for the SVMs and the features have been optimized. Here we do not do feature selection but use the features used by the MaltParser in CoNNL-X[4].

The only hyper parameter for CW classification is the number of iterations. We optimize this by doing 5-fold cross-validation on the training data. Although the manual feature selection has been shown to decrease accuracy this has been used for some languages to reduce the size of the model. The results are presented in table 3.

We see that even though the feature set used are optimized for the SVMs there are not big differences between the parses that use SVMs and the parsers that use CW classification. In general though the parsers with SVMs does better than the parsers with CW classifiers and the difference seems to be biggest on the languages where we did manual feature selection.

6 Conclusion

We have shown that using confidence-weighted classifiers with transition-based dependency parsing yields results comparable with the state-of-the-art results achieved with Support Vector Machines - with faster training and parsing times. Currently we need a very high number of features to achieve these results, and we have shown that pruning this big feature set uncritically hurts performance of the confidence-weighted classifiers.

7 Future work

Currently the biggest challenge in the approach outlined here is the very high number of features needed to achieve good results. A possible solution is to use kernels with confidence-weighted classification in the same way they are used with the SVMs.

Another possibility is to extend the feature set in a more critical way than what is done now. For instance the combination of a POS-tag and CPOS-tag for a given word is now included. This feature does not convey any information that the POS-tag-feature itself does not. The same is the case for some word-form and word-lemma features. All in all a lot of non-informative features are added as things are now. We have not yet tried to use automatic features selection to select only the combinations that increase accuracy.

We will also try to do feature selection on a more general level as this can boost accuracy a lot. The results in table 3 are obtained with the features optimized for the SVMs. These are not necessarily the optimal features for the CW-classifiers.

Another comparison we would like to do is with linear SVMs. Unlike the polynomial kernel SVMs used as default in the MaltParser linear SVMs can be trained in linear time (Joachims, 2006). Trying to use the same extended feature set we use with the CW-classifiers with a linear SVM would provide an interesting comparison.

[4]Available at `http://maltparser.org/conll/conllx/`

8 Acknowledgements

The author thanks three anonymous reviewers and Anders Søgaard for their helpful comments and the authors of the CW-papers for making their code available.

References

Sabine Buchholz and Erwin Marsi. 2006. Conll-x shared task on multilingual dependency parsing. In *Proceedings of the Tenth Conference on Computational Natural Language Learning (CoNLL-X)*, pages 149–164, New York City, June. Association for Computational Linguistics.

Michael Collins. 2002. Discriminative training methods for hidden markov models: theory and experiments with perceptron algorithms. In *EMNLP '02: Proceedings of the ACL-02 conference on Empirical methods in natural language processing*, pages 1–8, Morristown, NJ, USA. Association for Computational Linguistics.

Koby Crammer, Ofer Dekel, Joseph Keshet, Shai Shalev-Shwartz, and Yoram Singer. 2006. Online passive-aggressive algorithms. *J. Mach. Learn. Res.*, 7:551–585.

Koby Crammer, Mark Dredze, and Fernando Pereira. 2008. Exact convex confidence-weighted learning. In Daphne Koller, Dale Schuurmans, Yoshua Bengio, and Léon Bottou, editors, *NIPS*, pages 345–352. MIT Press.

Koby Crammer, Mark Dredze, and Alex Kulesza. 2009. Multi-class confidence weighted algorithms. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing*, pages 496–504, Singapore, August. Association for Computational Linguistics.

Mark Dredze, Koby Crammer, and Fernando Pereira. 2008. Confidence-weighted linear classification. In *ICML '08: Proceedings of the 25th international conference on Machine learning*, pages 264–271, New York, NY, USA. ACM.

Johan Hall, Joakim Nivre, and Jens Nilsson. 2006. Discriminative classifiers for deterministic dependency parsing. In *Proceedings of the COLING/ACL 2006 Main Conference Poster Sessions*, pages 316–323, Sydney, Australia, July. Association for Computational Linguistics.

Thorsten Joachims. 2006. Training linear svms in linear time. In *KDD '06: Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 217–226, New York, NY, USA. ACM.

Ryan T. McDonald and Fernando C. N. Pereira. 2006. Online learning of approximate dependency parsing algorithms. In *EACL*. The Association for Computer Linguistics.

Ryan T. McDonald, Koby Crammer, and Fernando C. N. Pereira. 2005a. Online large-margin training of dependency parsers. In *ACL*. The Association for Computer Linguistics.

Ryan T. McDonald, Fernando Pereira, Kiril Ribarov, and Jan Hajic. 2005b. Non-projective dependency parsing using spanning tree algorithms. In *HLT/EMNLP*. The Association for Computational Linguistics.

Jens Nilsson and Joakim Nivre. 2008. Malteval: An evaluation and visualization tool for dependency parsing. In *Proceedings of the Sixth International Language Resources and Evaluation*, Marrakech, Morocco, May. LREC.

Joakim Nivre, Johan Hall, and Jens Nilsson. 2006a. Maltparser: A data-driven parser-generator for dependency parsing. In *Proceedings of the fifth international conference on Language Resources and Evaluation (LREC2006)*, pages 2216–2219, May.

Joakim Nivre, Johan Hall, Jens Nilsson, Gülşen Eryiğit, and Svetoslav Marinov. 2006b. Labeled pseudo-projective dependency parsing with support vector machines. In *Proceedings of the Tenth Conference on Computational Natural Language Learning (CoNLL-X)*, pages 221–225, New York City, June. Association for Computational Linguistics.

Joakim Nivre, Johan Hall, Sandra Kübler, Ryan McDonald, Jens Nilsson, Sebastian Riedel, and Deniz Yuret. 2007. The CoNLL 2007 shared task on dependency parsing. In *Proceedings of the CoNLL Shared Task Session of EMNLP-CoNLL 2007*, pages 915–932.

Joakim Nivre. 2008. Algorithms for deterministic incremental dependency parsing. *Computational Linguistics*, 34(4):513–553.

Frank Rosenblatt. 1958. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408.