Procesamiento del Lenguaje Natural, Revista nº 51, septiembre de 2013, pp 119-126 recibido 18-04-2013 revisado 16-06-2013 aceptado 21-06-2013

Exploring Automatic Feature Selection for Transition-Based Dependency Parsing

Explorando la Selección Automática de Características para Analizadores Basados en Transiciones.

Miguel Ballesteros
Natural Language Processing Group,
Universitat Pompeu Fabra, Spain
miguel.ballesteros@upf.edu

Resumen: En este artículo se investigan técnicas automáticas para encontrar un modelo óptimo de características en el caso de un analizador de dependencias basado en transiciones. Mostramos un estudio comparativo entre algoritmos de búsqueda, sistemas de validación y reglas de decisión demostrando al mismo tiempo que usando nuestros métodos es posible conseguir modelos complejos que proporcionan mejores resultados que los modelos que siguen configuraciones por defecto.
Palabras clave: Análisis de dependencias, MaltOptimizer, MaltParser

Abstract: In this paper we investigate automatic techniques for finding an optimal feature model in the case of transition-based dependency parsing. We show a comparative study making a distinction between search algorithms, validation and decision rules demonstrating at the same time that using our methods it is possible to come up with quite complex feature specifications which are able to provide better results than default feature models.
Keywords: Dependency parsing, MaltOptimizer, MaltParser

1. Introduction

The choice of features to build data-driven NLP applications is something that needs to be done to produce good and competitive results. Besides application parameters, feature selection is the central way of tuning a system and it is not an easy task. It is difficult for researchers without specialized knowledge and it is also complicated for experienced researchers because it normally requires a search in a large space of possible cases. This is time consuming and demands deep knowledge of the task and the parsing algorithms involved.

Automatic feature selection is a process commonly used in machine learning, where the features that perform better are selected automatically for a single task. Since the inclusion of *MaltOptimizer*, it is not a matter of task expertise anymore (Ballesteros and Nivre, 2012), howewer for other tasks and parsing packages it still requires a lot of user action to produce a model capable of providing results that are comparable to the state of the art. We believe that this fact is still an issue in nowadays dependency parsing and Natural Language Processing (Smith, 2011), and this is why we took it as an inspiration.

In this paper we introduce and compare some automatic feature selection techniques, that are based on the ones implemented in MaltOptimizer. These techniques find an optimal feature set for a transition-based dependency parser: MaltParser (Nivre et al., 2007). Since MaltParser is based on support vector machines (henceforth, SVM[1]), there is no way to handle previously the weight of the features, because finding the appropriate weights for different features is exactly what a SVM does.

Therefore, we show firstly how it is possible to produce an optimal feature set for a transition-based dependency parser. Secondly, we compare a couple of algorithms and different criteria when selecting features, and we show how and why we get different results. Finally, we show some conclusions and ideas for further work.

[1] In our experiments we selected LIBLINEAR (Fan et al., 2008), to the detriment of LIBSVM (Chang and Lin, 2001), as training engine due to the inclusion of MaltOptimizer algorithms which are restricted to LIBLINEAR.

ISSN 1135-5948

2. Automatic Feature Selection

Automatic feature selection techniques have become a need in many natural language processing applications, but at this writing there is still not a significant amount of publications in the NLP community facing this problem (Smith, 2011). The objectives of an automatic feature selection technique are the following: (i) avoid overfitting and improve the performance, (ii) produce faster and more effective models and (iii) get more information from the annotated data.

There are two main approaches of finding an optimal feature set, others, as the ones that we show in the present paper, can be derived from these two:

Forward Selection. The process normally starts with zero features, and it adds them one by one, keeping them if they provide improvements. Besides the mixed backward-forward selection of MaltOptimizer (Ballesteros and Nivre, 2012), we can find an example on automatic feature selection carried out in this way for transition-based parsing (Nilsson and Nugues, 2010).

Backward Selection. The processes normally start with a big set of features, which is the case in transition-based parsing (or all the features, if possible) and remove them one by one, at each step removing the one that produces a feature set that performs better in a significant way. In this case, the concept of significant is normally more relaxed because a feature set with less features is less sensitive to overfitting and probably more efficient. As we already mentioned, MaltOptimizer also provides a backward selection of features but it is a merge between backward and forward.

We can find some relevant work that solve and study the problem of feature selection in a similar way as in the present paper in research areas different than NLP. For instance, the work done by Das and Kempe (2011), in which they demonstrated that "greedy algorithms perform well even when the features are highly correlated", which is something that is inherent to transition-based dependency parsing. Similarly, Pahikkala et al. (2010) show how to speed up a forward feature selection by applying a greedy search, which motivated the experiments shown in the present paper.

In the case that occupies our study, which is transition-based dependency parsing, we can have in principle an infinite set of possible features, but it is possible to isolate a rather small pool (to be handled automatically) of potentially useful features for each window.[2] We are normally able to tune the part–of–speech window, morphology, such as gender or number and features based on the partially built dependency tree. We can also be able to provide very useful conjunction features, which means that two features are considered as single feature by the parser. All of this can be done normally within two data structures, the buffer and the stack, but in some cases, depending on the parsing algorithm, we can also have a third (or even fourth) data structure that can be included in the feature specification.

3. MaltParser Feature Language

A transition-based parser uses two data structures (as mentioned above, there could be some auxiliary data structures, but there are at least two), a *buffer* and a *stack*. The buffer provides the words that are going to be used during the parsing process and the stack stores the words that are producing arcs from/to them. MaltParser implements four families of transition-based parsers and all of them use features over the stack and the buffer, which basically means that the parsing algorithm would take into account the annotated info (or partially built trees) of the words that are in the first positions of the stack and the buffer. However it is possible to define features in any position of the data structures, and this is why the search may be very extensive. Figure 1 shows the transition system of one of the parsing algorithm families (Nivre's), and Figure 2 shows how a transition-based parser works for a given sentence following it.

MaltParser uses the CoNLL data format and it is therefore possible to generate features over the columns annotated in the CoNLL files. It is possible to define features over the stack and buffer slots, containing information about part-of-speech (fine-grained: POSTAG, and coarse-grained: CPOSTAG), simple word (FORM), stemmed version of the word (LEMMA), a list of morphosyntactic features (FEAT) and dependency struc-

[2]The concept 'window' refers to the different columns (POSTAG, CPOSTAG, LEMMA, FEATS, DEPREL) that a CoNLL file has. See `http://ilk.uvt.nl/conll/#dataformat` for more information.

$$\textsc{Shift}: \langle \Sigma, i|B, H, D\rangle \Rightarrow \langle \Sigma|i, B, H, D\rangle$$
$$\textsc{Reduce}: \langle \Sigma|i, B, H, D\rangle \Rightarrow \langle \Sigma, B, H, D\rangle$$
$$\textsc{Left-Arc}\ (r): \langle \Sigma|i, j|B, H, D\rangle \Rightarrow \langle \Sigma, j|B, H[i \rightarrow j], D[i \rightarrow \mathrm{r}]\}\rangle \quad \text{if h(i)} \neq 0.$$
$$\textsc{Right-Arc}\ (r): \langle \Sigma|i, j|B, H, D\rangle \Rightarrow \langle \Sigma|i|j, B, H[j \rightarrow i], D[j \rightarrow \mathrm{r}]\}\rangle \quad \text{if h(j)} = 0.$$

Figure 1: Transition System for Nivre's algorithms (Nivre et al., 2007). B refers to the buffer and Σ to the stack. H and D conform the partially built dependency structure referring to heads and dependency labels.

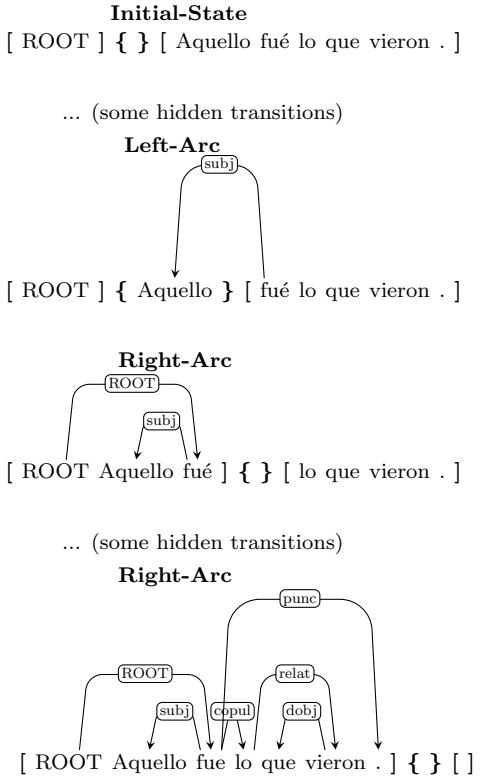

Figure 2: Parsing example for a sentence written in Spanish: *Aquello fue lo que vieron [That is what they saw].* The buffer is the structure on the right, and the stack is on the left.

tures that are being produced in the parsing process (DEPREL).

4. Feature Selection Algorithms

In this Section we describe the two implemented approaches that we are willing to test. Both approaches carry out the steps implemented in MaltOptimizer but they perform the search differently. The algorithms basically provide a backward and forward search of features by performing the following steps: (i) modify POSTAG and FORM features , (ii) modify DEPREL and POSTAG-DEPREL merge features over the partially built dependency structure, (iii) try with CPOSTAG, FEATS and LEMMA features if possible and (iv) add conjunctions of POSTAG and FORM features.

As mentioned by Ballesteros and Nivre (2012) the algorithm steps are not the same for all parsing algorithms; as shown in Section 3, the algorithms make use of different data structures, but the steps and the data structures are more or less equivalent.

As we mentioned in Section 2, these methods start with ***backward*** selection experiments removing features from the default feature model with the intention of testing whether they are useful. After that, they try with ***forward*** selection experiments, by testing features one by one and in combination. In this phase, a threshold of 0.05% LAS[3] (Labeled Attachment Score) is used to determine whether a feature is useful or not.

In this Section we describe the two implemented approaches that follow the steps presented above.

4.1. Relaxed Greedy Algorithm

The Relaxed Greedy approach traverses all the steps presented above adding one feature at a time and keeping the feature set that produces the best outcome. This Relaxed Greedy algorithm tries with all the backward and forward operations for all the steps shown at the beginning of Section 4, and it does not prune the search at all. Therefore, it can be understood as an exhaustive feature search that adds two, three or even more features at a time. We could think that an exhaustive feature search prevents getting stuck

[3] LAS = Percentage of scoring tokens for which the system has predicted the correct labeled attachments.

in local optima, which is something that intuitively could happen to the Greedy algorithm, presented in next subsection.

This algorithm implies running a high number of experiments because it just adds and tries with a big set of experiments, keeping the best feature model after each attempt. We could therefore expect that this algorithm overfits the performance in some cases providing a model with lower training error but higher test error.

Summing up, we have two different hypotheses for this algorithm: (1) it would not get stuck in local optima, (2) it could overfit the performance.

4.2. Greedy Algorithm

The Greedy algorithm is the one implemented and included in the MaltOptimizer distribution (Ballesteros and Nivre, 2012), it minimizes the number of experiments according to linguistic expert knowledge and experience (Nivre and Hall, 2010). It also follows the steps shown at the beginning of Section 4. However, in spite of trying with all the big set of possible features for each step as it is done in the Relaxed Greedy algorithm, it does the following:

1. It prunes the search when a backward feature selection of features provides improvements for a specific window, because it does not try with any forward selection experiments.
2. It prunes the search when a forward selection of features is not successful for a specific window, because it does not try with more forward selection experiments.

Therefore, it drastically reduces the number of iterations for backward and forward operations comparing with Relaxed Greedy.

For this algorithm we also have two different hypotheses: (1) it could intuitively get stuck in a local optima because it reduces the number of experiments and it could prune the search very early expecting that the search may not produce good results and (2) we could also expect that it underfits the performance due to (1). However, it is also worth remarking that the steps of this algorithm were developed with deep proven experience.

In the following Section we show an in-depth comparison between the Greedy and the Relaxed Greedy algorithms taking some experimental results into account, we therefore show which algorithm is the most accurate in order to get an optimal feature set.

5. Experiments and Results

As we mentioned in Sections 4.1 and 4.2, we expect that the Greedy and the Relaxed Greedy algorithms could underfit and overfit the performance respectively, we took these two facts as hypotheses. Therefore, we try to extract conclusions by running and comparing the outcomes of both algorithms. We train models using data sets for 6 different languages that were included in the CoNLL-X Shared Task (Buchholz and Marsi, 2006) (Arabic, Dutch, Slovene, Spanish, Swedish and Turkish), and we also test the models produced over the separate test-sets not used during the optimization.

With the intention of having a starting point of comparison we run the first phases of MaltOptimizer in order to set some parameters (phase 1) and select the parsing algorithm (phase 2) that performs the best over each data set.

In the rest of this section we firstly show the results of each of the algorithms implemented during the training phase in which we get the optimal feature set, afterwards, we show the test results when we test the outcome model with an unseen test set not used during the optimization (Section 5.1). Finally, and considering the results of the first two experiments, we perform a 5-fold cross validation strategy to demonstrate its usefulness (Section 5.2). We also show three different kind of sentence selection strategies for the folds.

It is worth mentioning that we always compare our results with the results given by default feature models to ensure the usefulness of our methods.

5.1. Results and Comparisons between Greedy and Relaxed Greedy

Table 1 shows the results of the Greedy algorithm and the Relaxed Greedy algorithm for a selection of languages. Note that these results are obtained using 80% of the training set for training and 20% as development test set, which were obtained using the entire training set and a separate held-out test set for evaluation. Therefore, these are the

results obtained during the optimization process.

Language	DefaultFM	Greedy	Relaxed Greedy
Arabic	63.84	65.56 (+1.72)	**66.00 (+2.16)**
Dutch	78.02	**82.63 (+4.61)**	82.49 (+4.47)
Slovene	68.40	71.71 (+3.31)	**72.43 (+4.03)**
Spanish	76.64	79.38 (+2.74)	**79.62 (+2.98)**
Swedish	83.50	84.09 (+0.59)	**84.20 (+0.70)**
Turkish	58.29	66.92 (+8.63)	**67.19 (+8.90)**

Table 1: Labeled attachment score with comparison to default feature model (MaltParser in its default settings without feature selection) and the greedy approach during the optimization process.

We can observe how Relaxed Greedy seems to beat the results of Greedy, with the exception of Dutch. Nevertheless, the differences are not very remarkable. Relaxed Greedy always carries out more than 100 different experiments, and Greedy between 40 and 50, depending on the pruning decisions and results.

This fact means, that the decisions taken during the development of the Greedy algorithm seem to be the correct ones. This fact is also evidenced in the Figure 5.1 in which we show the results of the Greedy and the Relaxed Greedy algorithms for the Slovene example.[4] We can see how the Greedy algorithm achieves an optimal accuracy faster than Relaxed Greedy, but in some cases it seems that it gets stuck (in optimization time) in local optima because the Relaxed Greedy approach beats these results finding eventually a more accurate feature configuration. And finally, it is also interesting to remark that the Greedy algorithm rarely produce results that are worse than the baseline or default feature model in none of its steps, however Relaxed Greedy does.

In order to find out the second hypothesis, whether the algorithms overfit or underfit the performance, and also whether our methods are really useful or not, we tested the obtained feature model with the real testing data set used in the CoNLL-X Shared Task, the Table 2 shows the results obtained.

In this case, most of the differences are indeed statistically significant comparing with the default models results.[5] According to McNemar's test we got significant improvements for Dutch, Slovene, Spanish and Turkish while the ones obtained for Arabic and Swedish are not better enough. Moreover, for the languages in which we have statistically significant improvements, these ones are for $p<0.01$ and for $p<0.05$ and the Z value varies from 2.687 in the case of Spanish to 12.452 in the case of Turkish. Taking into account the size of the testing data sets these results are quite remarkable.

Language	DefaultFM	Greedy	Relaxed Greedy
Arabic	64.93	**66.01**	65.71
Dutch	72.63	**77.23**	76.89
Slovene	69.66	**73.68**	73.26
Spanish	78.68	**80.00**	79.84
Swedish	83.50	83.81	**83.85**
Turkish	56.32	64.11	**64.31**

Table 2: Labeled attachment score with comparison to default feature models and the greedy approach for a selection of languages using the optimized models.

Comparing the Greedy algorithm and the Relaxed Greedy algorithm we can conclude that the Greedy one (which is much faster[6]) is more capable of providing a competitive feature model for the real case (in which the user would need to parse sentences that are not included neither in the test set nor in the training set) because the Relaxed Greedy models seem to be overfitted to the test set in most of the cases. Running the McNemar's test most of the differences are not statistically significant neither for $p<0.01$ nor for $p<0.05$, but for the Slovene treebank there is a statistically significant difference for $p<0.05$ with a Z value of 2.171, taking into account that the test sets are small - 5000 tokens- this is an interesting result. In summary, the outcomes given over most of languages nevertheless strongly suggests that the simple Greedy algorithm is more accurate and it does not underfit the performance.

These results led us to think that we should consider more conservative criteria for accepting improvements during feature selection. Therefore, in the following section we show a more informative approach, a K-Fold cross validation experiment for the Greedy al-

[4] In the Figure we simulate that the Greedy algorithm is waiting for the Relaxed Greedy before taking another step.

[5] We run the statistically significant tests by using MaltEval (Nilsson and Nivre, 2008)

[6] Every step of the algorithm requires to train a parsing model and test with the separate held-out test set, and depending on the size of the training treebank it could take a while

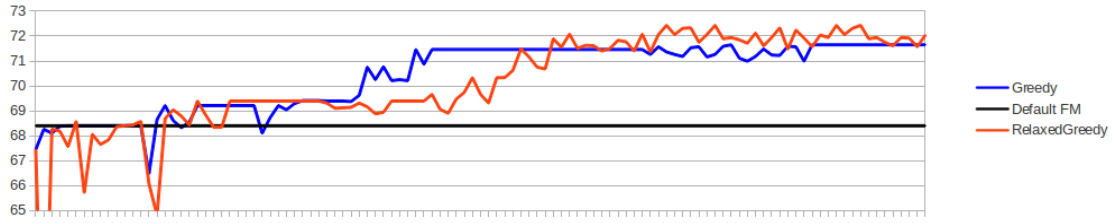

Figure 3: Results obtained by Greedy and Relaxed Greedy in every step of the algorithms for the Slovene treebank. The X axis shows the number of steps in the optimization process and the Y axis shows the performance achieved.

gorithm, because it is the one that provides better results in the present experiment and it is the only one that provides a statistically significant difference compared to Relaxed Greedy.

5.2. 5-Fold Cross Experiment

We decided to carry out a 5-fold cross validation experiment to be included in the validation step of the Greedy algorithm due to the results obtained in the real case with Greedy and Relaxed Greedy and taking into account that one of the best ways of estimating the generalization performance of a model trained on a subset of features is to use cross-validation, as shown in (John, Kohavi, and Pfleger, 1994).

We divided the corpus in 5 folds in order to have similar number of sentences in the folds as we had in the previous experiments, when we divided the corpus in 80% for training and 20% for testing.

It is well known that there are various ways of extracting the folds from the training corpora. For the present experiment and in order to get a more complex and interesting comparison we try two different approaches: (i) Extracting the sentences in an iterative way, by doing a simple split, firstly the sentences for fold 1, then sentences for fold 2 and so on and (ii) a pseudo randomize selection of sentences which provides more heterogeneous folds. We could come up with the following hypotheses: we could expect that the simple split selection of sentences will underfit the performance and we could also expect that the pseudo randomize selection will provide better results.

We also decided to implement three different criteria in order to decide whether a feature set is worth to be included in the final feature model: (i) considering that the average LAS over all folds must beat the result of the best feature model so far, (ii) considering that the majority of folds (in this case 3 of 5) must beat the result of the best feature model so far, and (iii) considering that all the folds must beat the result of the best feature model so far. Therefore, we could come up with the following hypotheses regardless or whether we use the simple split selection of sentences or the pseudo-randomize selection of sentences:

1. We could expect that (i) and (ii) will provide similar results, and it seems that both of them will neither underfit nor overfit the performance.
2. We could also expect that (iii) is going to underfit the performance in most of the cases.

In the following Subsections we show a set of experiments in which we discuss whether our hypotheses are corroborated or falsified.

5.2.1. Simple Split Selection of Sentences

The simple split selection of sentences only provides improvements for Slovene and Turkish for the *average* and the *majority* criteria, producing 70.24 LAS in the case of Slovene and 66.00 LAS in the case of Turkish. It seems that this selection of sentences is not very representative of the data set and this fact misleads the results when considering 5-fold cross validation.

The *average* and the *majority* criteria even come up with the same feature set and in the real case (training with the whole training set and testing with the test set) they got 73.52 LAS in the case of Slovene, and 64.45 LAS in the case of Turkish. These results compared with the ones that we got applying the simple Greedy step wise approach

are better for Turkish (+0.3) and worse for Slovene (-0.2). These differences are not statistically significant according to McNemar's test, neither for $p<0.01$ nor for $p<0.05$.

5.2.2. Pseudo Randomize Selection of Sentences

We believe that the pseudo randomize selection of sentences is more representative of the real case. Our methods provide the results of Table 3, which also shows the results of the Greedy algorithm without making use of the 5-fold cross validation. Moreover, Figure 5.2.2 shows the results of the 5 folds, with *average* criterion, pseudo randomize selection of sentences and the Slovene corpus, we can see how all the folds produce high results if we compare with the simple split selection.

Language	DefaultFM	Greedy	Average	Majority	All
Arabic	63.84	65.56	66.44	**66.62**	65.33
Dutch	78.02	**82.63**	82.32	82.29	81.42
Slovene	68.40	71.71	**72.00**	**72.00**	69.46
Spanish	76.64	**79.38**	79.29	79.29	76.64
Swedish	83.50	**84.09**	83.50	83.50	83.50
Turkish	58.29	66.92	67.11	67.01	**67.37**

Table 3: Labeled attachment score with comparison to default feature model and the greedy approach for a selection of languages from the CoNLL-X shared task (Buchholz and Marsi, 2006), reporting the results of the 5-fold cross validation.

As we can see the results of the 5 fold cross validation strategy are more informative, intuitively, we can rely more in the feature models obtained during the process because they have been tested over 5 different folds and represent the real case in a more accurate way. In order to demonstrate this fact, we set up Table 4 which shows the results of the obtained feature model when we test them with the test set of the CoNLL-X Shared Task.

As observed in Table 4, the 5-fold cross validation produces higher results for Arabic, Spanish and Turkish, while the simple Greedy algorithm produces better results in the other 3. The *All* criterion seems to be very restrictive because it leads to underfitting, however, *average* and *majority* produce similar and robust results.

Nevertheless, the differences for Slovene are statistically significant according to McNemar's test in favor for the Greedy algorithm for $p<0.01$ and for $p<0.05$. But, the differences for the Turkish algorithm are statistically significant in favor of the 5-fold cross experiment (for the three cases) running McNemar's test only for $p<0.05$ and a Z value of 2.296. The rest of the differences are not significant.

Language	DefaultFM	Greedy	Average	Majority	All
Arabic	64.93	66.01	66.21	**66.27**	65.61
Dutch	72.63	**77.23**	76.97	76.39	75.73
Slovene	69.66	**73.68**	73.32	73.32	71.64
Spanish	78.68	80.00	**80.46**	**80.46**	78.68
Swedish	83.50	**83.81**	83.59	83.59	83.59
Turkish	56.32	64.11	64.85	**65.01**	64.99

Table 4: Labeled attachment score with comparison to default feature model and the greedy approach for a selection of languages from the CoNLL-X shared task (Buchholz and Marsi, 2006), reporting the results of the 5-fold cross validation, making use of the training and test set of the CoNLL-X Shared Task

We can conclude that the Greedy algorithm by itself can provide results as good as an approach that follows more informative criteria, in this case, K-Fold cross validation. Nonetheless, it seems worth to carry out both experiments because in some cases we can find statistically significant improvements when we check over 5 different divisions of the corpus (or folds) and vice versa.

It is also worth noting that comparing the results of the simple split selection of sentences (shown in Section 5.2.1) for Slovene and Turkish (which are the ones that provide improvements) with the corresponding outputs produced by the pseudo randomize selection of sentences by running McNemmar's test. We get a statistically significant different in favor of the pseudo randomize selection for $p<0.05$. Therefore, we can also conclude that the results produced by the pseudo randomize selection are not overfitted and the ones produced by the simple split selection of sentences are underfitted by a misleading selection of sentences.

6. Conclusions

We have demonstrated that different criteria for automatic feature selection in the case of transition based dependency parsing can be accomplished successfully and produce variant and strong results in the final performance. Moreover, both search algorithms presented produce consistent improvements over the default settings using different validation procedures. According to our results

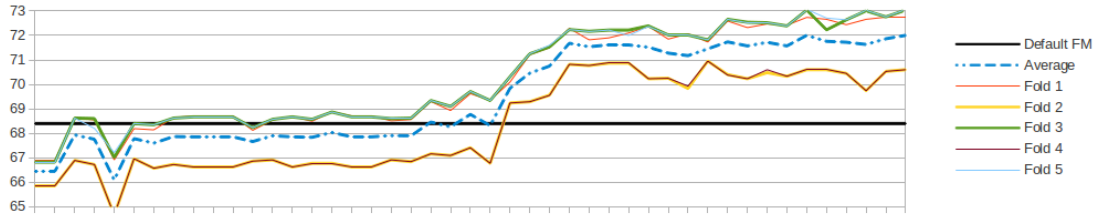

Figure 4: Results obtained by the 5 fold cross experiments with pseudo randomize selection of sentences in every step of the algorithm for the Slovene treebank.

and taking into account that all of our results are inherently based on the same Greedy algorithm, we believe that it is better to follow proven experience and linguistic expertise in this kind of experiments.

It is worth mentioning that we tried to alter the order between the different steps shown in Section 4, but we did not get any improvement nor any significant differences between the different feature sets and the different algorithms. A specific order for a treebank was useful and better, but it was not the same for a different treebank. Therefore, we plan to carry out an in-depth comparison following different experiment orders, not simply altering the order between the steps but making a cross experimental comparison.

Acknowledgments

Thanks to Joakim Nivre, who guided me in the development of MaltOptimizer and the algorithms that are explained in the present paper.

References

Ballesteros, Miguel and Joakim Nivre. 2012. MaltOptimizer: A System for MaltParser Optimization. In *Proceedings of the Eighth International Conference on Language Resources and Evaluation (LREC)*.

Buchholz, Sabine and Erwin Marsi. 2006. CoNLL-X shared task on multilingual dependency parsing. pages 149–164.

Chang, Chih-Chung and Chih-Jen Lin, 2001. *LIBSVM: A Library for Support Vector Machines*. Software available at http://www.csie.ntu.edu.tw/∼cjlin/libsvm.

Das, Abhimanyu and David Kempe. 2011. Submodular meets spectral: Greedy algorithms for subset selection, sparse approximation and dictionary selection. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, ICML '11, pages 1057–1064, New York, NY, USA, June. ACM.

Fan, R.-E., K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin. 2008. LIBLINEAR: A library for large linear classification. *Journal of Machine Learning Research*, 9:1871–1874.

John, George H., Ron Kohavi, and Karl Pfleger. 1994. Irrelevant Features and the Subset Selection Problem. In *International Conference on Machine Learning*, pages 121–129.

Nilsson, Jens and Joakim Nivre. 2008. Malteval: an evaluation and visualization tool for dependency parsing. In *(LREC'08)*, Marrakech, Morocco, may.

Nilsson, Peter and Pierre Nugues. 2010. Automatic discovery of feature sets for dependency parsing. In *COLING*, pages 824–832.

Nivre, Joakim and Johan Hall. 2010. A quick guide to maltparser optimization. Technical report.

Nivre, Joakim, Johan Hall, Jens Nilsson, Atanas Chanev, Gülşen Eryiğit, Sandra Kübler, Svetoslav Marinov, and Erwin Marsi. 2007. Maltparser: A language-independent system for data-driven dependency parsing. *Natural Language Engineering*, 13:95–135.

Pahikkala, Tapio, Antti Airola, and Tapio Salakoski. 2010. Speeding up greedy forward selection for regularized least-squares. In *ICMLA*, pages 325–330.

Smith, Noah A. 2011. *Linguistic Structure Prediction*. Synthesis Lectures on Human Language Technologies. Morgan and Claypool.