Transformation-Based Learning of Rules for Constraint Grammar Tagging

Torbjörn Lager
Department of Linguistics
Uppsala University, SWEDEN
Torbjorn.Lager@ling.uu.se

Abstract

If we conceive of a Constraint Grammar as an ordered sequence of transformation rules of a particular kind – as reduction rules rather than replacement rules – the transformation-based learning method used to train Brill taggers can, with minor modifications, be used to train Constraint Grammar taggers as well. This paper makes a few observations based on this approach, and presents some initial and rather promising experimental results.

1 Introduction

Two kinds of rule based taggers – Brill taggers (Brill 1995) and Constraint Grammar taggers (Karlsson et al. 1995) – have, in terms of accuracy, efficiency, compactness and intelligibility, quite successfully stood up to the competition from the statistical camp. How these rule-based taggers work, and how they differ in the way they work, can be briefly explained as follows.

In a Brill tagger, a lexical lookup module assigns exactly *one* tag to each occurrence of a word (usually the most frequent tag for that word type), disregarding context. A rule application module then proceeds to *replace* some of the tags with other tags, on the basis of what appears in the local context.

In a Constraint Grammar tagger, a lexical lookup module assigns *sets* of alternative tags to occurrences of words, disregarding context. A rule application module then *removes* tags from such sets, on the basis of what appears in the local context. However, in order to guarantee that each word token is left with at least one tag, the rule application module adheres to the following principle: *don't remove the last remaining tag*.

Whereas Brill taggers came with a novel and effective learning method which made it possible to learn rules for new languages very quickly, Constraint Grammar taggers were from the outset developed by hand only. Since then, several attempts have been made to induce Constraint Grammar rules – or 'Constraint Grammar'-like rules – from manually annotated corpora (Samuelsson et al. 1996; Cussens 1997; Eineborg & Lindberg 1998; Lindberg & Eineborg 1998; 1999).

In (Lager 1999) it was suggested that the transformation-based learning method used to train Brill taggers can, with minor modifications, be used to train Constraint Grammar taggers as well. This paper develops this suggestion further.

2 An Overview of the μ-TBL System

2.1 Background

The μ-TBL system – described in detail in (Lager 1999) – represents an attempt to use the search and database capabilities of the Prolog programming language to implement a generalized form of transformation-based learning.

The μ-TBL system is designed to be theoretically transparent, flexible and efficient. Transparency is achieved by performing a 'logical reconstruction' of transformation-based learning, and by deriving the system from there. Flexibility is achieved through the use of a compositional rule and template formalism, and 'pluggable' algorithms. As for the implementation, it turns out that transformation-based learning can be implemented very straightforwardly in a logic programming language such as Prolog. Efficient indexing of data, unification and backtracking search, as well as established Prolog programming techniques for building rule compilers and meta-interpreters, contribute to the making of a logically transparent, easily extendible, and fairly efficient system.

2.2 Transformation Rules

The object of TBL is to learn an ordered sequence of transformation rules. The μ-TBL system supports five kinds of transformation rules, but the relevant kind of rule in connection with learning Constraint Grammars is the *reduction rule*.

Online Proceedings of NODALIDA 2001

Reduction rules reduce the set of tags assigned to a word with a certain tag. An example would be "reduce a word's tag set with tag `vb` if the word immediately to the left is uniquely tagged as `dt`". Here is how this rule is represented in the μ-TBL system's formalism:

```
pos:red vb <- unique pos:dt@[-1]
```

This kind of rule will only remove a tag from a word if it is not the last tag for the word. If `vb` is the last value the above rule is not applicable and the reduction will not take place.

The use of the `unique/1` operator in a condition of a rule has the effect that the rule will trigger only if the assignments of tags to words in the relevant surroundings are non-ambiguous. (As Karlsson et al. (1995) put it, the rules are run in "careful application mode".)

Two or more rules may be connected into sequences – or *composed* – by means of the composition operator '`o`', where `(R o Rs)` basically means that the output of applying the rule `R` forms the input to the application of the rules `Rs`.

2.3 Rule Templates

Rules that can be learned in TBL are instances of templates, such as "reduce with tag `A` if the word immediately to the left is uniquely tagged as `B`", where `A` and `B` are variables. Here is how we write this template in the μ-TBL system:

```
pos:red A <- unique pos:B@[-1].
```

2.4 Rule Evaluation Measures

We now define two important rule evaluation measures. The *score* of a rule is the number of its positive instances minus the number of its negative instances:

$$score(R) = |positive(R)| - |negative(R)|$$

The *accuracy* of a rule is its number of positive instances divided by the total number of instances of the rule:

$$accuracy(R) = \frac{|positive(R)|}{|positive(R)| + |negative(R)|}$$

Score is a standard measure in TBL. The notion of rule accuracy is well-known in rule induction and inductive logic programming, and in this paper we will see that it has a role to play in the context of transformation-based learning too.

Corresponding to these two measures are two *thresholds* that are used to control the behaviour of the system and influence the learning results. The *score threshold* and the *accuracy threshold* are the lowest score and the lowest accuracy, respectively, that the highest scoring rule must have in order to be considered.

2.5 Learning

Transformation-Based Learning is a matter of repeatedly instantiating rule templates in training data, scoring rules on the basis of counts of positive and negative evidence of them, selecting the highest scoring rule on the basis of this ranking, and applying it to the training data. When the highest scoring rule does not meet the thresholds, the learning algorithm is terminated.

3 Learning Constraint Grammars

Using transformation-based learning to train Brill taggers is a well-established practice, and since (Lager 1999) contains examples of how to do this with the μ-TBL system, no more will be said on this issue here. However, (Lager 1999) also reports on a small experiment on the learning of Constraint Grammar rules from tagged corpora, and this will be further elaborated in the present paper.

In particular, the following aspects of transformation-based learning of Constraint Grammars will be investigated. How do the use of the uniqueness condition, the setting of the accuracy threshold, and the size of the training corpus, effect the result of training? How does the fact that rules are ordered matter?

Apart from the use of templates for reduction rules rather than templates for replacement rules, the learning of Constraint Grammars contrasts with the learning of Brill tagger rules only in how the accuracy threshold is set.

It is well known that replacement rules do not have to be very accurate: if a rule early in a sequence of replacement rules makes some errors, the errors can often be corrected by rules later in the sequence. By contrast, in a sequence of reduction rules there are no rules that can add tags once they have been (falsely) removed. Therefore, in order to maximize the accuracy of the whole sequence of rules, it must be induced under a validation bias which sees to it that each rule is as accurate as possible. In the μ-TBL system, this is

taken care of by setting the accuracy threshold to a value of 1.0, or to a value close to 1.0.

3.1 Tagger Evaluation Measures

For all the experiments performed in the present paper, *recall* (*R*) and *precision* (*P*) will be calculated exactly as in (Karlsson et al. 1995:172), i.e. as follows.

$$R = \frac{\text{received appropriate tags}}{\text{intended appropriate tags}}$$

$$P = \frac{\text{received appropriate tags}}{\text{all received tags}}$$

The *F-score*, calculated as $F = (2RP)/(R+P)$ is used as a straightforward way of combining the measures of recall and precision. The *tags per word ratio* (*T/W*) is calculated as well.

3.2 Experiment 1

In the first experiment, each word token in a training corpus of 60,000 words was assigned the set of part of speech tags that it can have according to a lexicon. The data also indicated which member of this set was the correct one.

The score threshold and the accuracy threshold was set to 4 and 1.0, respectively, and the system was run with 26 templates which mixed in various ways conditions on part of speech tags with conditions on word forms. The uniqueness operator was not used in this experiment.

The system learned 902 rules, the first ten of which are shown below:

```
pos:red nn|dt <- pos:in@[-1,-2,-3] o
pos:ed rp <- wd:in@[0] & pos:nn@[-1] o
pos:ed rb <- wd:in@[0] & pos:nn@[-1] o
pos:ed nn|dt <- pos:nn@[1] o
pos:ed vb <- pos:dt@[-1] o
pos:ed vbn <- wd:said@[0] o
pos:ed vbp <- pos:to@[-1,-2] o
pos:ed vbp <- pos:md@[-1,-2,-3] o
pos:ed vbz <- wd:'s@[0] & pos:nn@[1] o
pos:ed rp <- wd:in@[0] & pos:nns@[-1] o
```

The graph in Figure 1 shows how recall and precision develops as the rules are applied to the test corpus.

As can be seen from the graph, precision increases with the number of rules that have been applied. It starts at 69.2% and reaches its maximum at 93.4%, when all rules have been applied. Recall decreases with the number of rules that are applied. It starts from 100% and reaches its minimum of 98.9% when all rules have been applied.

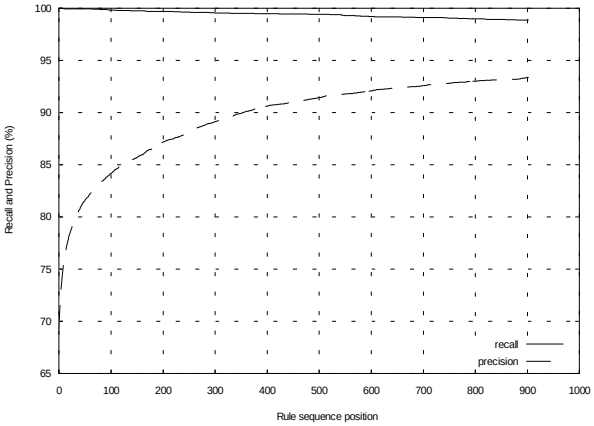

Figure 1: Recall and Precision as functions of the number of rules that have been applied

3.3 Experiment 2

In a second experiment, training was performed exactly as in Experiment 1, except that templates where the uniqueness condition was imposed on the context were used instead.

Training resulted in 1,030 rules. The effect when evaluating them on the test corpus is shown in the table and the graph in Figure 2**.**

Unique	#(Rs)	R	P	F	T/W
yes	1030	98.7	92.6	95.6	1.066
no	902	98.9	93.4	96.0	1.059

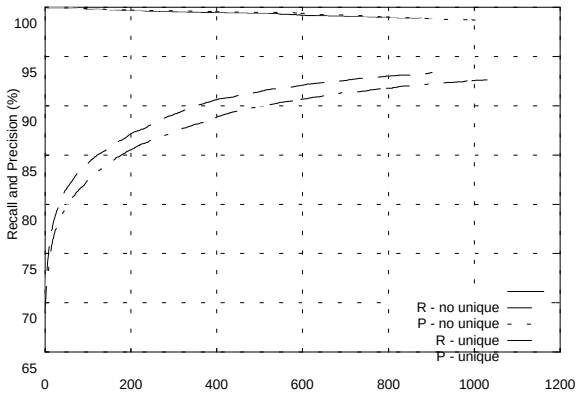

Figure 2: The effect of using the unique operator

Note that the tags per word ratio increases when the uniqueness operator is used. This is not hard to explain. The rules in the previous tagger were more ‘daring’, and therefore removed tags where the more careful tagger will not do so.

However, one would expect that there is something to be won by being careful, but surprisingly, the use of the uniqueness operator does not make any noticeable difference to recall. This is harder to explain, and we shall not attempt that here.

3.4 Experiment 3

In another experiment, the accuracy threshold (AT) was set to four different values: 1.0 (as before), 0.98, 0.95 and 0.90. Training was performed on the 60,000 word corpus, with templates and score threshold as before.

AT	#(Rs)	R	P	F	T/W
1.00	902	98.9	93.4	96.0	1.059
0.98	738	98.7	94.3	96.5	1.047
0.95	561	98.7	94.7	96.7	1.042
0.90	417	97.9	95.8	96.8	1.023

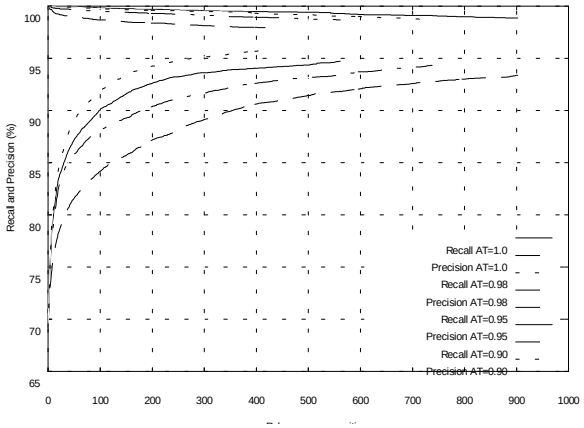

Figure 3: The effect of varying the accuracy threshold

The graph in Figure 3 illustrates nicely how the accuracy threshold can be used to control the recall-precision trade-off. Furthermore, it appears to be the case that an accuracy threshold slightly lower than 1.0 may have a beneficial effect on the overall result (in terms of F-score), perhaps by compensating for noise in the data.

4 Rule Order Dependency

Transformation rules are essentially ordered, with later transformations being dependent upon the outcome of applying earlier transformation rules.

Earlier reduction rule applications can affect what rules might later apply at a particular position P by removing a part of speech tag from the set of tags assigned to P. For example, given the (sketchy) 'text'

```
... question/{nn,vb} ...
```

and the (sketchy) rules

```
pos:red nn <- ... o
pos:red vb <- ...
```

it is clear that if the first rule is applicable to the word "question", the second is not, since that would have removed the last remaining tag.

The use of unique conditions introduces another kind of order dependency. Earlier reduction rule applications can also affect what rules might later apply at a particular position P by removing elements from the sets of parts of speech assigned to words in positions *local* to P. For example, given the (sketchy) 'text'

```
... what/{dt,pn} gave/{nn,vb} him/{pn} ...
```

and the rules

```
pos:red nn <- unique pos:pn@[1] o
pos:red dt <- unique pos:vb@[1]
```

the second rule is applicable only because the first rule is.

4.1 Experiment 4

To establish how much the order means in practice for sequences of reduction rules, a simple experiment was performed, in which a sequence of learned rules was reordered randomly, and then applied to the test corpus.

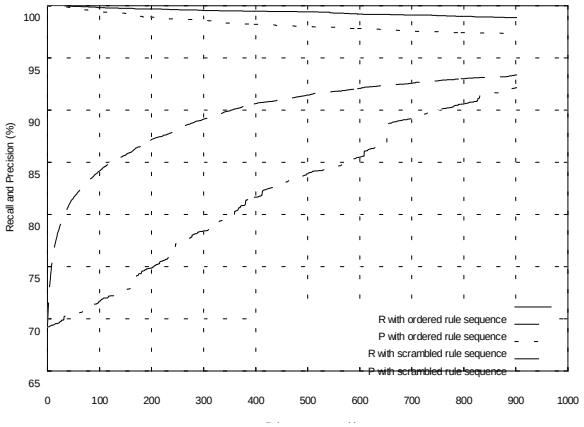

Figure 4: Comparing an ordered and a scrambled rule sequence

Recall decreased from 98.7% to 97.4% when the previous sequence of 902 rules was randomly reordered, and (as a consequence) precision dropped too (from 93.4% to 92.1%), but the number of tags per word stayed roughly the same.

Note that the precision curve corresponding to the randomly reordered rules sequence is flatter than the original curve. The explanation for this is that the really powerful rules are no longer applied first, but spread out randomly. This also accounts for the jaggedness of the curve.

How can the drop in recall be explained? Intuitively, if we change the order of rules, some rules apply too early, and others too late. For example, when a given CG rule is placed late in the sequence, the 'don't remove the last tag'-principle stops it from applying in (most) cases where it would have hurt, if it was placed earlier in the sequence. 'Being a bad rule' (or 'being a good rule') is not a property of a rule just by itself, but of a rule in a particular place in a sequence.

4.2 Discussion

Whereas order dependency is a well-known property of a sequence of Brill tagging rules, it may be argued that this is a point where our CG 'reconstruction' departs from the spirit of original Constraint Grammar. Whether this change of 'semantics' is for good or for bad, remains to be investigated further, but it does seem to be the case that the fact that a rule can leverage off the work performed by rules earlier in the sequence can boost the overall performance of a tagger.

Our Constraint Grammar rules – in contrast with rules in the original Constraint Grammar framework – share another interesting property with Brill tagging rules. The sequence of rules is optimized in the sense that it need only be applied once. This has been verified empirically by applying the sequence of rules twice on the same data, and the second pass has always turned out to have no effect at all.

5 Scaling Up

In an attempt to see how well transformation-based learning of Constraint Grammars works for a larger training corpus, 240,000 words of Swedish data was used. The tag set was also much larger: 156 tags instead of the 42 tags used before.

Training with 15 templates – which took three weeks to complete (*sic*!) – resulted in a sequence of 4,866 rules. These rules were applied to a 30,000 word test corpus. This resulted in 393 errors, which corresponds to a recall of 98.7%. The initial number of tags per word was 1.46 (which corresponds to a precision of 68.4%). After applying the rules there were only 1.10 tags per word left, corresponding to a precision of 89.8%.

6 Summary and Conclusions

This paper has shown that if we conceive of a Constraint Grammar as an ordered sequence of transformation rules of a particular kind – as reduction rules rather than as replacement rules – the transformation-based learning method used to train Brill taggers can, with minor modifications, be used to train Constraint Grammar taggers as well.

The performance of the taggers – around 99% recall and 90% precision – is probably good enough to warrant further research. First and foremost, TBL algorithms which are more efficient for the particular task of learning reduction rules should be sought for. Secondly, an attempt to make better use of complex morphological features should be made, so that more general rules can be learned. Finally, work has already begun to develop ways to learn other kinds of Constraint Grammar rules (e.g. *selection rules)*, and this may eventually improve the performance further.

References

Brill, E., 1995, Transformation-Based Error-Driven Learning and Natural Language Processing: A Case Study in Part of Speech Tagging. *Computational Linguistics*, December 1995.

Cussens, J., 1997, Part of speech tagging using Progol, In *Proceedings of the 7th International Workshop on Inductive Logic Programming* (ILP-97), Prague.

Karlsson, F., Voutilainen, A., Heikkilä, J., Anttila, A. (eds.), 1995, *Constraint Grammar. A Language-Independent System for Parsing Unrestricted Text*. Mouton de Gruyter.

Lager, T., 1999, The μ-TBL System: Logic Programming Tools for Transformation-Based Learning, Paper presented at the *Third International Workshop on Computational Natural Language Learning* (CoNLL-99), Bergen, 1999.

Lindberg, N. and Eineborg M., 1998, Learning Constraint Grammar-style disambiguation rules using Inductive Logic Programming. In *Proceedings of COLING/ACL'98.*

Samuelsson, C., Tapanainen, P. and Voutilainen, A., 1996, Inducing Constraint Grammars, In: Laurent, M. and de la Higuera, C. (eds.) *Grammatical Inference: Learning Syntax from Sentences*, Springer Verlag.