The Prague Bulletin of Mathematical Linguistics 76, 2001

A Machine Learning Approach to Automatic Functor Assignment in the Prague Dependency Treebank

Sašo Džeroski and Zdeněk Žabokrtský

Abstract

We use the machine learning approach of decision tree induction to automate the transition from analytic tree structures to tectogrammatical tree structures within the Prague Dependency Treebank. Rules are learned for assigning functors to autosemantic nodes given information about the dependent node and its governing node. We compare our approach to that of using hand-crafted rules and dictionaries and consider integrating the two approaches. Using the rules produced by machine learning in addition to hand-crafted rules and dictionaries increases cover and recall, while retaining the same precision.

1 Introduction

The *Prague Dependency Treebank* (PDT) is a research project at the Institute of Formal and Applied Linguistics (`http://ufal.mff.cuni.cz`), Faculty of Mathematics and Physics, Charles University, Prague. It is aimed at a complex annotation of a part of the Czech National Corpus, built at the Institute of the Czech National Corpus, Faculty of Philosophy, Charles University, Prague. The annotation scheme comprises three levels: morphological, analytical, and tectogrammatical.

At the *morphological level*, a morphological tag and a lemma is assigned to each word form in the input text; the annotation contains no syntactic structure. At the *analytical level*, the corresponding linear sequence of words and punctuation marks is enriched with a dependency structure representing the given sentence. Each node is assigned an analytical function (such as Subject, Object, Adverbial, Attribute...). The resulting structure is called an *analytic tree structure* (ATS).

During the transition from ATSs to *tectogrammatical tree structures* (TGTSs), the topology of the tree is slightly changed. *Synsemantic words* (functional words, nodes "without their own lexical meaning"), e.g., prepositions, auxiliaries, subordinating conjunctions, as well as punctuation marks, do not have their own nodes in TGTS, but are captured in the attributes of the remaining nodes representing the *autosemantic words.* At the *tectogrammatical level*, each autosemantic word of a sentence is annotated with its tectogrammatical function (functor) that represents its syntactic position within the sentence, e.g., Actor, Patient, Addressee, Effect, Origin, various types of spatial and temporal circumstantials, Means, Manner, Extent, Consequence, Condition. There are approximately 60 functors. Functors provide a more detailed information about the relation to the governing node than that covered by the analytical function. Figure 1 shows an example of a simplified TGTS, where each node in the figure is labeled with its lemma and functor.

At present, most of the functors have to be assigned manually, which is very time-consuming. The desire to save expert annotators' time and accelerate the growth of the PDT has motivated the development of a system for *Automatic Functor Assignment* (AFA) (i.e., a system which could automatically assign at least some of the functors). The AFA system (Žabokrtský, 2000) uses rule-based and dictionary-based methods, where the rules are hand-crafted and the dictionaries are extracted from a training set of manually annotated sentences.

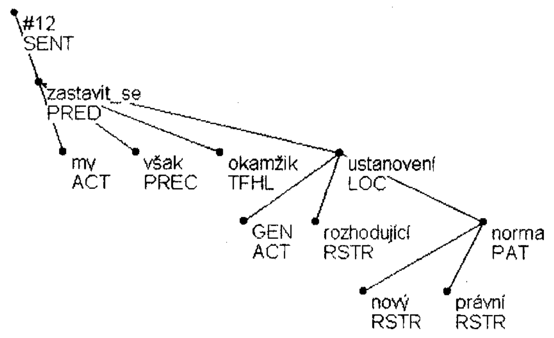

Figure 1: TGTS of the sentence *Zastavme se však na okamžik u rozhodujících ustanovení nové právní normy.* (Let's however stop for a moment at the most important paragraphs of the new legal norm.)

In this paper, we consider applying machine learning (ML) to a training set of manually annotated sentences in order to derive rules for automatic functor assignment. This would provide an alternative to the hand-crafted rules of the AFA system and might even outperform them. Better still, combining the two approaches might yield better performance than each of them individually.

The remainder of this paper is organized as follows. Section 2 describes the AFA system. Section 3 briefly describes the decision tree learning system used in our experiments as well as the setup in which learning was taking place. Section 4 compares the performance of the machine learning approach and the original AFA system, as well as the performance of several different integrations of the two. Section 5 concludes with a discussion and an outline of directions for further work.

2 The System AFA for Automatic Functor Assignment

In this section, we describe the features of the AFA system (Žabokrtský, 2000) which are important for the comparison to and integration with our ML-based approach. We focus on the data-preprocessing procedure, on the training and testing set, on the classification of assigners, and on the rules that enable the symbiosis of the assigners.

Data preprocessing. If a program is to decide what the correct functor of a node is, it must be provided with sufficient information for such a decision. Naturally, from the implementation point of view it is desirable to minimize the amount of the required information. It is assumed that it is sufficient to take into account only the following set of attributes: word form, lemma, full morphological tag, analytical function of both the governing and dependent node, and the preposition or conjunction which connects the governing and the dependent node. In order to make the subsequent processing easier, three additional simple attributes (the parts of speech of both nodes and the morphological case of the dependent node) were extracted from these 10 attributes. A TGTS is transformed into a list of symbolic vectors, each vector corresponding to a dependent node. The task of AFA can be formulated as the classification of these vectors into functor classes, i.e., assigning a dependent node its functor, given the features listed above.

Available data. For the development of the AFA system, 18 files with TGTSs with manually assigned functors were available, each file containing up to 50 TGTSs. The available data were split into a *training set* and a *testing set*. The testing set consists of three randomly chosen files yielding 1089 testing vectors. The training set contains 15 files of TGTSs yielding 6049 training vectors. The training set was used to derive dictionaries for AFA, as described below. The testing set was used to assess the quality of the AFA system.

Families of assigners. Three families of assigners are used in AFA: rule-based methods, dictionary-based methods, and a method based on the notion of the nearest vector in the feature space.

The *rule-based methods* (RBMs) consist of simple hand-crafted rules, similar to decision trees. They do not use lexical attributes (word form, lemma) and they were either derived from observations of regularities in the training set (Žabokrtský, 2000), or from the instructions for annotators in the "Manual for tectogrammatical annotation" (Hajičová *et al.*, 1999). For example, the rules of the method `verbs_active` look as follows:

- `verbs_active`: if the governing node is a verb in an active form then
  - if the analytical function (afun) is subject, then the node is assigned the functor ACT (abbr. → ACT)
  - if the afun is object and the case is dative then → ADDR
  - if the afun is object and the case is accusative then → PAT

The rules for the remaining RBMs (`verbs_passive`, `adjectives`, `pronounpos`, `numerals`, `pnom`, and `pred`) are formulated in a similar way.

Unlike RBMs, *dictionary-based methods* (DBMs) profit from the value of the lexical attribute of the given node. If it is found in a dictionary, the corresponding functor is used. Two types of dictionaries are employed:

- `adverbs`, `subconj`: The couples *adverb–functor* (resp. *subordinating conjunction–functor*) were automatically extracted from the training set, and added to the list of adverbs (resp. sub. conjunctions, SC) from the manual; from the combined list, the *unambiguous* (accompanied always with the same functor) adverbs (resp. SC) were extracted.
- `prepnoun`: All the *preposition–noun* pairs (a preposition followed by a noun) were extracted from the training set. The unambiguous couples which occured at least twice were inserted into the dictionary. Examples: *v roce* (in year) TWHEN, *pro podnikatele* (for businessman) BEN, *v zemích* (in countries) LOC.

The method `sim` (similarity) is based on a function which defines the distance between two vectors in the symbolic feature space. When a node is to be assigned, the most similar (i.e., the nearest) vector from the training set is found and its functor is used.

How is it glued together? Each method has its own assigner (a script written in Perl). The unassigned data go serially through a sequence (pipeline) of assigners. No assigner can change a functor already assigned by a previous assigner. The overall performance can be "tuned" by reordering (or removing some) assigners. RBMs are applied in the order `pred`, `verbs_active`, `verbs_passive`, `pnom`, `adjectives`, `numerals`, `pronounpos`, and DBMs in the order `adverbs`, `subconj`, `prepnoun`.

3 Learning an assigner with C4.5

Decision tree induction is one of the most popular machine learning approaches. It takes as input a set of examples (represented as vectors of feature values and their classifications) and produces a tree-like structure (called decision tree) that can be used for classifying new examples. Internal nodes in the tree correspond to features (also called attributes), branches correspond to feature values, and leaves of the tree correspond to classifications (predicting specific class values).

In our case, the features (attributes) were **gov_morph**, **gov_pos**, **gov_afun**, **dep_morph**, **dep_pos**, **dep_case**, **conj_prep**, and **dep_afun**. These include the morphological tags (or rather their prefixes, e.g., VP for a verb in indicative mood and present tense), the corresponding parts-of-speech, and the analytical functions of the governing and dependent node. In addition, the case of the dependent node is considered, as well as the conjunction/preposition connecting the two nodes. The class variable is the tectogrammatical function (functor) of the dependent node.

An excerpt from the file with training examples is given below: each row corresponds to an example; feature values are listed first; the class value is the last entry in each row.

```
n, n, adv, a, a, 0, null, atr, rstr.
vs, v, obj, n, n, 2, do, adv, dir3.
vp, v, pred, vs, v, 0, z_e, obj, pat.
znum, z, sb, dg, d, 0, null, auxz, ext.
n, n, atr, znum, z, 0, null, sb, rstr.
```

```
dep_afun = sb:
|   gov_pos = a: rstr (1.0/0.8)
|   gov_pos = j: pat (1.0/0.8)
|   gov_pos = n: rstr (21.0/8.0)
|   gov_pos = null: act (1.0/0.8)
|   gov_pos = z: act (19.0/5.9)
|   gov_pos = v:
|   |   gov_morph = vp: act (463.0/25.9)
|   |   gov_morph = vr: act (133.0/12.9)
|   |   gov_morph = vs: pat (28.0/8.2)  *
|   |   gov_morph = vf:
|   |   |   dep_case = 0: pat (2.0/1.0)
|   |   |   dep_case = 1: act (6.0/3.3)
|   |   |   dep_case = 4: pat (1.0/0.8)
```

Figure 2: An excerpt from a simplified decision tree (text output from C4.5).

C4.5 (Quinlan, 1993) is probably the most widely used program for inducing decision trees. It implements the TDIDT (Top Down Induction of Decision Trees) approach, where a feature is first selected that discriminates best among the class values of the given training examples. Once this feature is selected, it is assigned to the root of the tree; the examples are partitioned according to the values of this feature and tree construction is repeated recursively; the resulting subtrees are attached to the branches of the root node. If the set of examples contains examples of only one class or if no good attribute can be found to split on, a leaf is created which predicts a specific class value. This is the criterion for terminating the recursion. C4.5 also performs pruning (simplification) of the induced decision trees. The subtrees that are built on too small number of examples to be statistically reliable are removed and replaced with leaves (in other words, they are pruned). Several parameters control the construction of a decision tree with C4.5 (e.g., by setting the degree of pruning): we left these parameters at their default values.

An excerpt from the text representation of the simplified tree produced by C4.5 from the training examples is given in Figure 2. A graphical representation of this excerpt is given in Figure 3. In the text representation, the indentation of a node represents the length of the

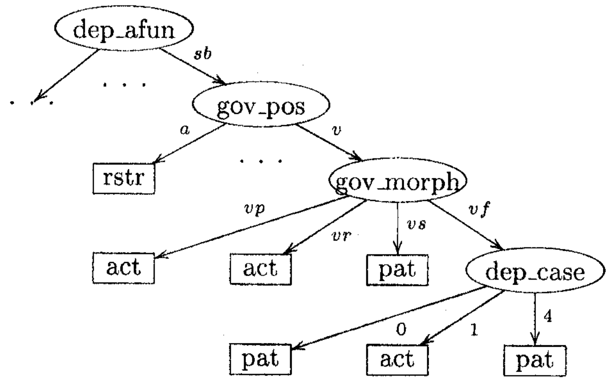

Figure 3: A graphical representation of the decision tree excerpt from Figure 2

path from the root of the tree (dep_afun) to that node. A leaf contains a class value, e.g., pat, as well as the number of examples that fall within the leaf: 28 examples fall within the leaf marked with an asterisk in Figure 2, of which 8.2 are classified incorrectly. Fractional numbers of examples can result if unknown feature values are present in the training examples.

The simplified tree generated by C4.5 on the training set is referred to as the assigner `ml`. Given that many of the leaves in this tree still cover only few examples and are not very accurate, we postprocessed the tree by removing the leaves the accuracy of which is less than 80%. For example, the leaf marked with an asterisk in Figure 2 was removed, as its accuracy is (1-8.2/28)*100%=70.7%. The remaining decision tree was semi-automatically translated into Perl code - this assigner is named `ml80`. The Perl code resulting from the part of the decision tree in Figure 2 is given below.

```
if ($dep_afun eq "sb") {
   if ($gov_pos eq "v") {
    if ($gov_morph eq "vp") {$functor="act"};
    if ($gov_morph eq "vr") {$functor="act"};
   }};
```

This code can be read as follows: if the analytical function of a node is Subject and its governing node is a verb in the active voice, then assign the functor ACT (Actor) to the dependent node. This is exactly the same as the first rule in the RBMs `verbs_active` and is in agreement with common sense.

Some rules which are part of the Manual for tectogrammatical annotation have also been "rediscovered" by C4.5. For instance, the leaf marked with an asterisk in Figure 2 corresponds to the following rule from the manual: "if a subject is dependent on a verb in the passive voice, then its functor is PAT (Patient)". Note, however, that this rule has not been included in the assigner `ml80` because of its insufficient accuracy on the training set.

This demonstrates that C4.5 can find rules that are valid on the training set, agree with common sense and may have already been formulated by humans. As evidenced by the performance of the assigners that include the C4.5 rules (cf. Section 4), it can also discover "new" regularities that are not to be found in the annotation manual or the RBMs.

The assigners `ml` and `ml80` can be inserted into different positions within the sequence of assigners in the AFA system. Figure 4 depicts the data flow diagram of the ML-AFA system where `ml80` is executed first, followed by all the assigners from the AFA system.

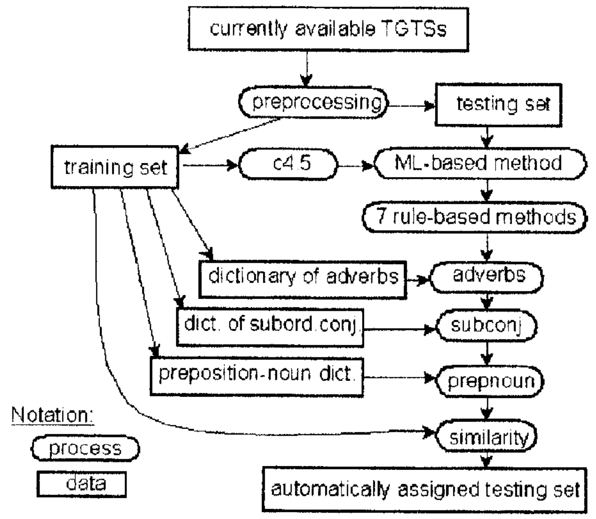

Figure 4: Data flow diagram of the ML-AFA system containing the assigner ml80.

4 Performance Evaluation

In this section, we evaluate the performance of the assigners `ml` and `ml80` in comparison with different subsets of the AFA assigners. We also consider several different ways of integrating the assigner `ml80` with the AFA assigners. In particular, we consider different subsets of the AFA assigners plus `ml80`, as well as different orderings thereof.

We evaluate the performance of the assigners and their combinations in terms of the following metrics:

Cover (C) = number of all nodes assigned by the given method (C = E + H)

Relative cover (C') = cover divided by the number of all functors to be assigned (1089 in the testing set). It also reflects the frequency of phenomenona (e.g., the occurrence of possessive pronouns).

Errors (E) = number of incorrectly assigned functors

Hits (H) = number of correctly assigned functors

Recall (R) = percentage of correct functor assignments by the given method among all the functors to be assigned (R = 100% · H / 1089)

Precision (P) = percentage of correct functor assignments by the given method among all functors assigned by this method (P = 100% · H / *cover*)

Table 1: Performance of single assigners.

	C	C'	H	R	E	P
RBMs	558	51.2 %	524	48.1 %	34	93.9 %
DBMs	46	4.2 %	41	3.8 %	5	89.1 %
`sim`	1089	100 %	796	73.0 %	293	73.0 %
`ml`	1089	100 %	819	75.2 %	268	75.2 %
`ml80`	406	37.3 %	384	35.3 %	22	94.6 %

The performance of the individual assigners is listed in Table 1. The performance of t combinations of classifiers is given in Table 2. AFA refers to the sequence of assigners RBM DBMs, `sim`, while ML-AFA refers to the sequence `ml80`, RBMs, DBMs, `sim`. The results th do not involve `ml` or `ml80` have been taken from (Žabokrtský, 2000).

Table 2: Performance of assigner sequences.

	C	C'	H	R	E	P
R+D	604	55.5 %	565	51.9 %	39	93.6 %
R+D+`ml80`	686	63.0 %	641	58.9 %	45	93.4 %
`ml80`+R+D	686	63.0 %	641	58.9 %	45	93.4 %
AFA	1089	100 %	852	78.2 %	237	78.2 %
ML-AFA	1089	100 %	856	78.6 %	233	78.6 %

Considering the performance of the individual assigners, we note that `ml` and `sim` assig functors to all nodes in the testing set (C'=100 %). The recall and precision for each are equ in this case. `ml` performs slightly better. However, the precisions of both `ml` and `sim` are belo 80 %, making these assigners unacceptable for automated annotation if used by themselves. precision of at least 90 % is expected from an automated assigner.

The remaining three assigners have good precision, with `ml80` being slightly more precis than the hand-crafted RBMs. The quality of the learned decision tree is thus similar to tha of the hand-crafted rules. However, the RBMs have much larger coverage than `ml80` (51.2 % vs. 37.3 %). Details on the RBMs performance are given in Table 3 (taken from Žabokrtský 2000).

Adding `ml80` to the sequence of assigners RBMs+DBMs increases the relative cover by 7. %, while retaining practically the same precision (R+D vs. R+D+`ml80` entries in Table 2) This indicates that `ml80` contains some rules that are essentially different from those presen in the hand-crafted RBMs, but overlaps with them for the most part. The overlap betwee `ml80`, RBMs and DBMs is illustrated in Figure 5. The intersection of the domains of (the set of nodes annotated by) `ml80` and RBMs is more than one half of the domain of hand-crafte rules.

Overall, the performance of the sequence `ml80`+RBMs+DBMs is the same as that of th sequence RBMs+DBMs+`ml80` (the details of the performance are in Tables 4 and 5). This fact indicates that the quality of the learned decision tree is similar to that of hand-crafted rules. It also suggests (but does not prove) that the two methods are mostly consistent (they assign the same functors to the same nodes) in the overlapping part.

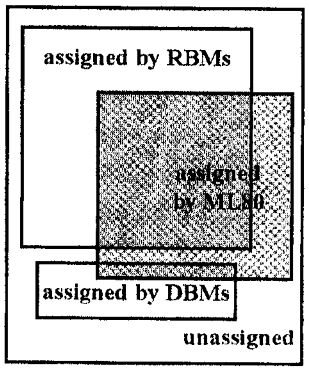

Figure 5: The relationships among the covers of the assigners `ml80`, RBMs, and DBMs. The outermost rectangle depicts the set of all functors to be assigned.

Table 3: Evaluation of the performance of the rule-based methods.

Method	Cover	Rel. cover	Hits	Recall	Errors	Precision
pred	104	9.6 %	104	9.6 %	0	100 %
verbs_active	199	18.3 %	184	16.9 %	15	92.5 %
verbs_passive	7	0.6 %	6	0.6 %	1	85.7 %
pnom	34	3.1 %	32	2.9 %	2	94.1 %
adjectives	177	16.2 %	170	15.6 %	7	96.0 %
numerals	21	1.9 %	15	1.4 %	6	71.4 %
pronounpos	16	1.5 %	13	1.2 %	3	81.3 %
Total	Σ 558	Σ 51.2 %	Σ 524	Σ 48.1 %	Σ 34	93.9 %

As was already mentioned, it is possible to select and apply only a subset of the available methods and thus to control the characteristics of the AFA system. In general, the higher the recall achieved, the lower the precision. The optimal compromise between precision and recall is reached for the sequences containing `ml80`, RBMs, and DBMs.

Table 4: Evaluation of the performance of the sequence RBMs, DBMs, and `ml80`.

Method	Cover	Rel. cover	Hits	Recall	Errors	Precision
RBMs	558	51.2 %	524	48.1 %	34	93.9 %
DBMs	46	4.2 %	41	3.8 %	5	89.1 %
ml80	82	7.5 %	76	7.0 %	6	92.7 %
Total	Σ 686	Σ 63.0 %	Σ 641	Σ 58.9 %	Σ 45	Σ 93.4 %

Table 5: Evaluation of the performance of the sequence `ml80`, RBMs, and DBMs.

Method	Cover	Rel. cover	Hits	Recall	Errors	Precision
ml80	406	37.3 %	384	35.3 %	22	94.6 %
RBMs	242	22.2 %	224	20.5 %	18	92.5 %
DBMs	38	3.5 %	33	3.0 %	5	86.7 %
Total	Σ 686	Σ 63.0 %	Σ 641	Σ 58.9 %	Σ 45	93.4 %

5 Conclusions and further work

We have used the machine learning approach of decision tree induction to generate rules that assign the tectogrammatical functions to dependent nodes. The rules generated in this fashion, cover more than one half of the domain of hand-crafted rules when applied as a standalone system. If they are applied together with the hand-coded rule-based methods and dictionary methods, an overall improvement of the relative cover of 7.5 % is achieved, while maintaining the precision level of 93.4 %. While this may not seem to be a dramatic change of performance, in the context of the whole PDT it corresponds to automatically annotating about additional 20 000 nodes, which would save a significant amount of annotators' time. Since that the precision is not 100 %, annotators still have to verify the functors assigned by the ML-AFA system, i.e., they cannot be completely replaced. However, given the precision level of 93.4 %, verification is easier and less time consuming than annotating from scratch.

The performance of rules generated by machine learning can be improved in several ways. A larger training set could be used. Also, machine learning approaches other than decision tree induction (such as rule induction in propositional and first-order logic) might be more suitable. These avenues will be explored in further work.

References

Böhmová, A., Panevová, J., Sgall, P. (1999) Syntactic Tagging: Procedure for the Transition from the Analytic to the Tectogrammatical Tree Structures. In *Proceedings of the Second International Workshop on Text, Speech and Dialogue*, pages 34–38. Springer, Berlin.

Böhmová, A., Hajičová, E. (1999) How much of the underlying syntactic structure can be tagged automatically? *The Prague Bulletin of Mathematical Linguistics*, 71: 5–12. Charles University Press, Prague.

Hajič, J. (1998) *Building a Syntactically Annotated Corpus: The Prague Dependency Treebank.* In E. Hajičová, ed., *Issues of Valency and Meaning*, pages 106–132. Karolinum, Charles University Press, Prague.

Hajičová, E., Panevová, J., Sgall, P. (1999) *Manuál pro tektogramatické značkování.* (Manual for tectogrammatical annotation.) Technical Report ÚFAL-TR-7, Charles University, Prague.

Hajičová, E., Pajas, P. (2000) Evaluation of Tectogrammatical Annotation of PDT. In *Proceedings of the Third International Workshop on Text, Speech and Dialogue*, pages 75–80. Springer, Berlin.

Panevová, J. (1980) *Formy a funkce ve stavbě české věty.* (Forms and functions in the structure of the Czech sentence.) Academia, Prague.

Quinlan, J. R. (1993) *C4.5: Programs for Machine Learning.* Morgan Kaufmann, San Mateo, CA.

Žabokrtský, Z. (2000) Automatic Functor Assignment in the Prague Dependency Treebank. In *Proceedings of the Third International Workshop on Text, Speech and Dialogue*, pages 45–50. Springer, Berlin.