Abstract

This paper describes an approach to POS tagging based on the automatic refinement of manually written linguistic tagging rules. The refinement was carried out by means of a learning algorithm based on decision trees. The tagging rules work on ambiguity classes: each input word undergoes a morphological analysis and a set of possible tags is returned. The set of tags determines the ambiguity class: the suitable subset of tagging rules is applied and a single tag is returned. This method can be applied both with the original set of manual rules and with the set of refined rules. The experiments have been performed on the Italian language. The results show that the refined set reduces the tagging error of about 17%. The noticeable feature of the approach is that the original set of rules is not modified, so that they retain their linguistic value, but new rules are added to handle problematic cases.

Introduction

Some years ago, [Cutting et al. 92] listed a set of desiderata for a good POS tagger. They are: Robustness (ability not to collapse on titles, tables, etc.), Efficiency (linear tagging time and fast learning algorithms), Accuracy (high percentage of correct tagging), Tunability (ability to take advantage of linguistic insights), Reusability (possibility to apply to new corpora).

If we look at the literature on tagging it seems that the features that received most attention are accuracy (reasonably enough) and reusability. In particular, reusability has been addressed by the development of efficient learning algorithms which can be applied to any corpus large enough. With some approaches, the learning set (training data) must be tagged manually (supervised learning), with others this is not necessary (unsupervised learning). It has been observed that both these methods have some disadvantages: supervised learning requires large manually tagged corpora, which are not available for all languages (in particular, for Italian); unsupervised methods tend to build word clusters which are very coarse, thus lacking "fine distinctions found in the carefully designed tag sets used in the supervised methods" [Van Guilder 95].

Although the desiderata of efficiency and robustness come out as a side-effect in case the learning algorithm is designed carefully and the form of the learnt knowledge is suitable (as in the case of most probabilistic approach) for linear analysis of input text, it seems that tunability has been quite neglected: it can in fact be partially achieved by supervised methods, but it is much more difficult to obtain in case large tagged corpora are not available so that unsupervised methods need be used.

In contrast to automatically acquired knowledge, manually written rules can be used. Although this method has been claimed to require too much effort and to fail to meet the reusability requirement, it has been objected that it can in principle obtain the best results in terms of accuracy [Tapainen and Voutilainen 94, Samuelsson and Voutilainen 97]. However, their "tagger" (EngCG-2) includes about 3600 linguistic constraint and it appears to be more akin to a parser (CG stands for Constraint Grammar) than to a pure tagger. Although we agree that it is not easy to define a clear-cut distinction between a tagger and a parser, it seems that this kind of tool justifies the claim about the high cost of development of manual taggers.

The approach we followed faces the various problems mentioned above: absence of very large tagged corpora for Italian, great effort needed to build large set of constraints, desire for efficiency and accuracy. The basic idea is to build up a low-effort set of manual tagging rules that be linguistically motivated and to apply automatic learning methods to refine them. The hope was that a smaller tagged corpus is required for tuning than for learning: this hope was fulfilled partially. The methodology we adopted was:
- Build a set of tagging rules.
- Build a small tagged corpus (currently, about 13,400 words, split in a training set - 10,752 - and a test set - 2,641).

---

1 It seems reasonable that any tagger based on perspicuous linguistic knowledge could slowly turn into a parser, when the constraints on the kind of features used for choosing the correct tag become looser and looser. For instance, EngCG-2 uses information on subcategorization, and words far away from the one being disambiguated can be freely tested.
- Apply the manual rules to the test set and evaluate the results.
- Refine automatically the manual rules on the training set.
- Apply the refined rules to the test set and evaluate the results.

In sum, the refinement made the error rate drop from the original 4.7% (manual rules) to 3.9% (refined rules). This result needs many comments that will be set forth in the following, after the methodology has been discussed in more depth.

Architecture

The modules that compose the tagger are rather standard:

1) TOKENIZER
The raw ASCII text is fed to a tokenizer, which emits hypotheses about tokens and segments the text into sentences. The token classes include GW (general word), PN (Proper Noun), PUNCT (Punctuation Marks), NUM (numbers) and a few others. The output of the tokenizer can be ambiguous, in that a word can be labeled as both GW and PN in case it is capitalized or all-caps. Moreover, the segmentation in tokens can be ambiguous too, since something as week-end is ambiguously interpreted as either a single token or two tokens separated by a hyphen.

2) MORPHOLOGICAL ANALYZER
The token hypotheses are input to the morphological analyzer, which emits hypotheses about suffixes and checks the hypotheses by looking for the root in the dictionary2. The morphology is rather complex in Italian, not only because verbs can have more than 40 inflected forms, but also because clitics can attach to verbal forms:

prendi: prenderemp-2nd-sing
(i.e. "to take" imperative, second person, singular)
prendilto: prenderemp-2nd-sing +
lo_m-sing-acc
(i.e. "take it" or "take him", where "lo" is masculine, singular and accusative)
prendiglielo: prendereimp-2nd-sing +
gli_sing-dat +
lo_m-sing-acc +
(i.e. "take it/him/it from him/her" where "gli" is singular and dative)

The morphological analyzer is implemented as a (augmented) deterministic automaton that collects morphological information while it scans the token from the end.

3) TAG DISAMBIGUATOR
The tagger proper, which collects the outputs of the morphological analyzer which were confirmed by the access to the lexicon and chooses the best tag. Actually, the tagger selects one of the possible interpretations, and, as a side effect, its associated tag. The tags are discussed below.

A note is required concerning unknown words. We kept apart three cases:
- The unknown word is not capitalized. In this case we exploit a class guesser based on learned rules (see below).
- The unknown word is capitalized. In this case a Proper Noun interpretation is added by the tag disambiguator (the morphological analyzer returns a warning of the form: "Does not exist; maybe a proper noun").

The class guesser for non-capitalized unknown words is based on rules learnt automatically by means of the same methodology used for the tagging rules (see the Learning section). Each word receives a set of possible tags among ADJ, ADV, NOUN and VERB (i.e. the open classes). These rules have the same form as the standard tagging rules and are based on the last five characters of the unknown word (where the suffix can possibly occur, although, as explained above, the suffix of an Italian word can sometimes be longer than that). A larger corpus of texts has been used as training set since for this task it is sufficient the information about the possible tags of each known word. This information is provided by the morphological analyzer.

Finally, it must be observed that, in case a "known" word which occurs in the mid of a sentence is capitalized, a Proper Noun hypothesis is added to the one(s) obtained from the morphological analysis. For instance, in "Mi piace la musica di Verdi" (I like the music by Verdi), Verdi is recognized as an adjective by the morphological analyzer (verdi = greenpl), but the Proper Noun tag is added by the tagger before entering the disambiguation phase. Notice that all these solutions are just patches, since we did not face yet the problem of Proper Nouns in its full complexity.

Tagging rules

The tagger receives in input a sequence of possibly ambiguous entries (PAE), each of which is composed of one or more lexical elements (LE1 LE2 ... LEn); for example, for the word data there are four interpretations: the participle past of the verb dare (to give), two forms of the verb datare (to date) and the noun data (date). The corresponding PAE is:

((DARE cat VERB transitive YES mood PARTICIPLE tense PAST gender F number SING)
 (DATARE cat VERB transitive YES mood IND tense PRES number SING person 3)
 (DATARE cat VERB transitive YES mood IMPER tense PRES number SING person 2)
 (DATA cat NOUN gender F number PL))

When n>1, the PAE is ambiguous and the following two-stage tagging process is applied:

- First, the primary tags (see Appendix 1) T1=Cat^+(LE1), T2=Cat^+(LE2), ... Tn=Cat^+(LEn), are

2 Currently, the dictionary includes about 24,000 roots, associated with morphological and syntactic features.
collected. These tags form the Primary Ambiguity Set (PAS=\{T_1\} \approx \{T_2\} \approx ... \approx \{T_n\}). Clearly, it may happen that \(T_i = T_j\) for some \(i \neq j\). The specific subset of rules devoted to the disambiguation within this set is retrieved\(^3\). The application of the rules always selects a single tag \(T_k\). The symbol \(\text{Cat}^+\) refers to the category of the word extended to form the primary tag, as shown in Appendix 1.

- Then, all lexical elements whose primary category is \(T_k\) (i.e. those elements \(\text{LE}_m\) such that \(\text{Cat}^+(\text{LE}_m)=T_k\)) are extracted from the PAE. If the cardinality of this new set is greater than one, then some intra-tag ambiguity has been left (forming the Secondary Ambiguity Set - SAS), otherwise the disambiguation process is completed. In the first case a new subset of tagging rules is retrieved, according to the type of ambiguity. For instance, if \(T_k=\text{Noun}\), then the rules Common-Proper, Mas-Fem and Sing-Pl can be applied, while if \(T_k=\text{Verb}\) (or Aux, or Modal), we have rules for Fin-Inf (choosing between finite and infinite tenses), Sing-Pl, etc. Again, a single choice results. Finally, if no single LE is selected according to the result of this step, as may happen in case of semantic ambiguity, a random choice is made.

As it has been noted elsewhere (Tzoukermann et al. 97), highly inflected languages, as French or Italian, have to rely heavily on the context to obtain good tagging performances. Since the starting set of tagging rules is hand-written, we felt free to include in them any condition on the features of the current and surrounding words we assumed to be useful for the disambiguation task (see [Lezius et al. 96] for a discussion of the respective merits of fixed-context vs. variable-context).

The general form of a tagging rule is\(^5\):

\[
\text{If } P_1 \sqcap P_2 \sqcap ... \sqcap P_r \text{ then } T_k \text{ } \text{CF} \text{ } N
\]

where \(P_i (1 \leq i \leq r)\) are predicates and \(T_k\) is a Primary Tag for rules operating on the PAS or a feature for rules to be applied to SAS (in the following, PAS and SAS will be collectively termed AS). \(N\) is a discrete value in the set \{Certain, Almost-certain, Uncertain\}. The certainty factor (CF) is used to order the application of the rules.

\(^3\) The rules associated with a given PAS will be referred to in the following as PAS-r1, PAS-r2, .... These are the actual names we use in the tagger (es. NOUN-VERB-R1, NOUN-VERB-R4, ADJ-ADV-NOUN-R2, ...)

\(^4\) In case no rule is found, the first tag is chosen. This happens, in the test set, for 91 words in 31 different ambiguity classes (so the total number of rules should be 83, instead of the 52 for which rules are currently defined). The results would have been clearer, but equivalent, with 31 new defaults, where the first element of each set is specified as default tag.

\(^5\) The form of the manual rules is more general than this, since any logical combination of predicates is allowed (see Appendix 2 for some examples). However, this is the form of the automatically learnt rules, and the predicates are the same.

The ordering is very rough, but it seemed to us that nothing better could be done with manually written rules. Rules having the same certainty factor are applied in a random order (usually they have mutually exclusive conditions, so that the order of application does not matter).

Among the predicates we currently use there are the following ones:

- \(\text{prevtag} (\text{TagSet})\): the primary tag of the previous word is in TagSet.
- \(\text{prev2tag} (\text{TagSet})\): as above, but for the word before the previous word.
- \(\text{prevagreement} (\text{Features})\): there is number and/or gender agreement between the current and the previous word.
- \(\text{prevmood} (\text{Mood})\): the previous word is a verb whose mood is Mood.
- \(\text{prevtense} (\text{Tense})\): the previous word is a verb whose tense is Tense.
- \(\text{curmood} (\text{Mood})\): in the verbal interpretation of the current word, the mood is Mood.
- \(\text{curtense} (\text{Tense})\): in the verbal interpretation of the current word, the tense is Tense.
- \(\text{nexttag} (\text{TagSet})\): one of the primary tags of the next word is in TagSet.
- \(\text{next2tag} (\text{TagSet})\): as above, but for the word after the next word.
- \(\text{nextmood} (\text{Mood})\): the next word has a verbal interpretation, and its mood is Mood.
- \(\text{nexttense} (\text{Tense})\): as above for Tense.
- \(\text{beforetag} (\text{Tag})\): within the previous n words, there is a word of tag Tag. The variable n is global; currently it is set to the value 10.
- \(\text{predicative} ()\): the nearest preceding verb belongs to the class of "predicative verbs". The set of predicative verbs is stored in a pre-defined list. This is a semantic feature of verbs that should be included in the dictionary, but the source of the information would not affect the behavior of the tagger.

The inspection of the set of predicates shows that most of the tests concern the previous one or two words, the features of the word to be disambiguated, and the next word. Only exceptionally, words in a wider context are taken into account (predicates "beforecat" and "predicative").

A noticeable feature of the tagger is that the tagging rules are applied strictly from left to right, starting from the first PAE of the text. This mode of operation has the consequence that any predicate applied to the preceding context refers to an already disambiguated element, while the predicates referring to the current and next words concern PAE's which are still possibly ambiguous. As discussed in the Conclusions section, this is one of the most problematic features of this approach, but it also has some advantages.

In addition to the rules, each ambiguity class is associated a default (referred to as PAS-default, see Note 3) which is used in case no rule in the set succeeds (which can sometimes happen in contexts which were
unforeseen or, perhaps, were deemed unable to bring information useful for the disambiguation).

The current set of base (hand-written) rules covers 52 ambiguity classes and includes 134 rules (but see note 4). Although the figures seem rather low (cfr. the 304 ambiguity classes - called genotypes - in [Tzoukermann et al. 97]), the hierarchical organization of the decision problem (first the primary tag, and then the features) reduces the number considerably. For instance, the "possible" 256 subsets, from which the null set and the 8 singletons are excluded. Of course very few of them really occur in texts.

The base rules were originally used in a deterministic parser [Lesmo & Torasso 83]. Recently, they have been tuned manually on the basis of a small corpus of 5000 words (the first half of the training corpus we used in this work). After this step, they were used for tagging automatically the remaining part of the training corpus (totaling 10,752 words) and, again the errors were detected manually (this is the "bootstrapping" [Leon & Serrano 95] method we currently apply to tag any new testing text). In table 1, we report some statistics about the ambiguity present in the corpus. From Table 1, it can be seen that:

- Punctuation marks are not included in the overall evaluation. This seems fair because they are mostly unambiguous (a period is a period, after all). However, in the manually tagged corpus, we have sometimes changed the Punct category into Conj Coord (Coordinating conjunctions) for commas.

- The number of occurrences with 7 interpretations is very high. This depends on the fact that the word "che" (that, who, which, ...), which is very common, gets seven interpretations: question adjective, exclamative adjective, question pronoun, exclamative pronoun, relative pronoun, subordinating conjunction, comparative conjunction.

- The value that summarizes the degree of ambiguity is comparable to the one reported in other studies. For instance, [Tzoukermann et al. 97] reports an ambiguity factor ranging from 1.72 to 1.82 per word (depending on the corpus).

The table does not include ambiguities due to the presence of locutions (canned phrases). Locutions have been defined as belonging to two different types: fixed and flexible. They are included in a separate "locution" file, and appear as a possible output from the morphological analyzer. They must undergo morphological analysis since flexible locutions can vary in number and gender, but we adopted the approach that whenever the sequence of words forming the locution

<table>
<thead>
<tr>
<th></th>
<th>Tot. Words</th>
<th>13.771</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Punct</td>
<td>3.019</td>
</tr>
<tr>
<td>3</td>
<td>Tot. ok (1-2)</td>
<td>10.752</td>
</tr>
<tr>
<td>4</td>
<td>Not ambiguous</td>
<td>5.755</td>
</tr>
<tr>
<td></td>
<td>2 Interps</td>
<td>2.722</td>
</tr>
<tr>
<td></td>
<td>3 Interps</td>
<td>1.441</td>
</tr>
<tr>
<td></td>
<td>4 Interps</td>
<td>361</td>
</tr>
<tr>
<td></td>
<td>5 Interps</td>
<td>62</td>
</tr>
<tr>
<td></td>
<td>6 Interps</td>
<td>109</td>
</tr>
<tr>
<td></td>
<td>7 Interps</td>
<td>289</td>
</tr>
<tr>
<td></td>
<td>&gt; 7 Interps</td>
<td>13</td>
</tr>
<tr>
<td>6</td>
<td>av. interp /word</td>
<td>1.866</td>
</tr>
</tbody>
</table>

Table 1 - Degree of ambiguity of the input data in the training set

occurs in the input, it is definitely assigned the locution interpretation. For this reason, they were not included in table 1, whose figures would increase considerably.

Learning

The refinement algorithm is based on decision trees (Bahl et al. 89, Heeman & Allen 97). Decision trees are a classification method: at each level of the

---

6 The 256 subsets, from which the null set and the 8 singletons are excluded. Of course very few of them really occur in texts.

7 This appears to be a useless work. But the corpus we are tagging is also annotated for dependencies among words (i.e. it is a first version of a tree bank for Italian); so, we need to be careful about the syntactic role of the various elements. The distinction between punctuation marks and conjunctions is very useful for this task.

8 The treatment of locutions is not as flexible as it should be. "Flexible" locutions are characterized by the fact that some of their words can be morphologically inflected, but even in this case all the words must appear in sequence. Phrasal locutions are still waiting for an adequate treatment.
tree, one of the parameters describing the input element is inspected, in order to determine which daughter must be reached; the leaves of the tree are associated with the classes to which the input element may belong. In order to learn a decision tree from examples a balance must be found between a detailed inspection of all alternatives (if this is done, then the problem is intractable) and the use of heuristics, which allows to focus on a given parameter (at each node of the tree being built). In particular we used C4.5 (Ross Quinlan 93), a tool where the choice of the parameter is based on rather sophisticated measures of the information gain. One of the features that made C4.5 especially suitable to us, was its ability to construct production rules on the basis of the generated decision tree (the antecedents of the production rules are given by the predicates collected traversing the tree from the root to each leaf); this format corresponds exactly to the form of our tagging rules: a conjunction of predicates as premise, a resulting category and a (numeric) certainty factor. So, the application of C4.5 produces, in our case, new tagging rules, according to the methodology explained below.

Since the goal was not to build the rules from scratch, but to refine the manual ones, the following approach was adopted. For each ambiguity class, and each rule within them, the set of cases when it was applied is collected. Let's focus on the rule ASk-rki. This rule was applied on the test corpus nki times, producing, for nki tokens, the tag assignment Tki. These assignments are compared with the correct tags, thus producing two values n+ki (the rule firing produced the correct assignment) and n-ki (the tag Tki is not the right one), where n+ki + n-ki =nki. In case n-ki ≠0, the rule is a candidate for being refined.

The main problem now is that the number of times the rule is applied (in the small training set) is often not sufficient for the learning procedure to be applied successfully. So, we tried to expand the training set of each rule in the following way. Let's consider the Ambiguity Set AS={T1, T2, ..., Tn}. Even if there are a few cases where a given AS-rj rule fires, there are many more contexts where items correctly tagged as T1, or T2, ... or Tn occur. For instance, if AS={Noun, Verb-Main}, there may be a few items ambiguous between these two tags, but in the whole corpus there are a lot of nouns and main verbs (not ambiguous in this way, but appearing in a well-defined context). So, the training set for the refinement of the rule AS-rj is enlarged by adding all of these contexts in which the rule fires; these contexts presumably keep apart noun form verbs. This method worked well provided that we disable the tests on the features of the current word from being used in the rules. For instance, in the Noun-Verb example, the presence of a Tense feature classifies an (unambiguous) verb as such, thus excluding from the decision tree all the verbs via a feature which is useless in truly ambiguous cases.

After the learning phase, new rules, more detailed than the original ASk-rki are produced; these new rules are appended below the original one. It must be observed that the original rule is maintained, and the tagger applies it exactly as before the learning; the difference is that, after the learning, it does not return a class name (a tag), but the name of the set of newly learnt rules associated with it. This process is sketched in fig.1.

It must be observed that the refinement of a rule does not affect the behavior of the other rules of the same AS. This is guaranteed by the fact that the original rule acts as a precondition for the application of the refined set.

The same procedure is applied to "default" rules (i.e. in cases when no rule for a given AS applies) as well as to those PAS for which no set of rules have been defined. In these cases, some rules are learnt which are really "new", i.e. not subject to the previous enablement of an existing manual rule.

The total amount of learned rules is 200 distributed on 26 different PAS: 5 of them concern default cases and for 3 of them there was no previous set of manually written rules.

**Results**

The results obtained are summarized in Table 2, where the performances of the manual rules and of the refined rules on the training corpus are compared.

These results seem encouraging, but more need to be done. The first, obvious, limitation is the very limited size of the manually tagged (learning and test) corpora. We are proceeding to tag more texts; we foresee to have available for the end of July 1998 about 1000 tagged sentences (approximately 30.000 words, including punctuation marks). Although the work appears to proceed rather slowly, we recall (see footnote 7) that the tagged sentences are also being annotated for dependencies, which is a more time-consuming task. The growth of the tagged corpus seem required, in particular with respect to the results reported in table 3.
From table 3, it can be appreciated the real performance of the learning procedure (instead of the tagger). Type A errors and type B errors are kept apart on the basis of the tuning of some learning parameters. In particular the learning program was not applied to rules that was applied less than 10 times in the training corpus. This means that some rules were never refined because of lack of data. In these cases the errors of the manual rules had no chance of being corrected (this happened for 47 errors out of 124). So, the refinement produced the correction of 27.3% of the errors to which it was applied.

The two major problems we observed in the experiments are common to most left-to-right taggers: error clusters and limited ability to take into account the right context.

Concerning clusters, it must be recalled that, of course, the training was made on the correct tags, so that a tagging error propagates forward: the presence of a wrong tag influences the application of the rules trying to disambiguate the next words. It must be noted, however, that similar clusters arise not only after the application of the refined rules, but also when the original rules are applied. This suggests that the source of the problem does not stand in the learning procedure, but in the tagging procedure.

Similar comments apply to the exploitation of right context: a rule can include in its preconditions the requirement that the tag of the following word be $T_x$. But $T_x$ could be just one of the possible tags of the next word, without being the preferred one. With our control strategy, the rule in question fires even if the next word is subsequently tagged as $T_y$ ($\neq T_x$).

Left-to-right processing is the best way to meet the desideratum of Efficiency, since the production rules can be (rather) easily translated into a deterministic automaton; the only alternative solution currently working (to the best of our knowledge) is the well known Brill's tagger [Brill 92]. Although we followed him in adopting a rule-based approach, we tried to apply the constraints immediately, instead of using them to correct previous errors. Consequently, we are now inspecting a different solution to the problem, i.e. variable-length lookahead. However, many of the current rules include both forward-looking and backward-looking preconditions; so, when a non-ambiguous word is reached in the lookahead process, it may happen that the last word of the ambiguous cluster cannot be disambiguated because the rules need the tag of the previous word. Experiments are under way to keep apart the two kinds of preconditions (forward and backward) and to exploit the certainty factors to balance the suggestion coming from the two directions.

## Conclusions

We have presented a rule-based tagger that exploits both a manual method and an automatic method for building the set of tagging rules. This approach seems suitable in case one wants to limit the effort for developing linguistic knowledge and no large corpus of tagged data is available.

The results of the application of the automatically refined rules to the test set are in line with the standard results of automatic tagging. More tests, however, are required to verify whether the availability of a larger training set could produce even better results.

An advantage of the structure of the tagging rules (both the manual ones and the learnt ones) stands in the kind of predicates used in their left-hand side. In many cases (as for instance, the agreement checks, or the test for the presence of a noun enabling a word to be tagged as a relative pronoun) the conditions of the rules are based on the existence of some specific relationship between the words. These tests could be exploited in order to provide a parser with suggestions about the structure it must build. It must be observed that this type of interaction between tagger and parser is substantially different from the one envisaged in [Charniak et al. 94]. In that case, in fact, the task of the tagger is just to restrict the category of the input word in order to reduce the degree of ambiguity and, consequently, to make the parser more efficient. What we are suggesting, on the contrary, is that the tagger does not feed the parser only the tags, but also the reasons for having chosen a given tag. We are currently pursuing this line of research, which is more in line with the approach of EngCG, although we are moving from tagging to parsing instead of the other way around.

All the modules described in this paper are implemented in Common Lisp. The compiled tag disambiguator, run on a SUN SparcStation 4, took 4.67 seconds on the test corpus. This makes about 700 words per second (including punctuation marks), without any
kind of optimization on the procedures. Notice that UNIX wc command returns 2745 words for the input text, instead of our 3286. The time for writing the output on a file is not included in the figure, as well as the time for morphological analysis (which is largely the slowest module - about 90 word per second).

Acknowledgments
We are very grateful to Cristina Baroglio for the fruitful discussions and good advice that she provided us with about all Machine Learning issues.

References


Appendix 1
(List of 35 primary tags)
The tags in the left column are the ones actually used in the tagging rules.

### Appendix 2
(Some example rules)

**Manually written rules:**

**Rule adjc-nounr1:**
```
 :if (and (prevtag 'art adjc)
          (prevagreement 'gender number)
          (nexttag 'verb chesub cherel qadj adv prep nil)))
 :then 'noun
 :cf 'A
```

where A means "Almost certain", and nil indicates a punct able to act as a sentence boundary. The rules are implemented as Lisp structures, so we maintained their original syntax.

**Rule adjc-nounr2:**
```
 :if (and (not (prevtag 'noun))
          (not (nexttag 'noun))
          (not (predicative)))
 :then 'noun
 :cf 'U
```

**Rule adjc-nounr3:**
```
 :if (and (or (not (prevtag '(art adjc)))
          (not (prevagreement 'gender number)))
          (predicative))
 :then 'adjc
 :cf 'U
```

**Automatically learned rules:**

**Rule adjc-nounr2,1:**
```
 :if (and (prevtag 'conjunct)
          (prev2tag 'adj))
 :then 'adjc
 :cf '84.1
```

**Rule adjc-nounr2,2:**
```
 :if (and (nextagreement 'gender number)
          (nexttense 'nil))
 :then 'adjc
 :cf '70,7
```

**Rule adjc-nounr2,3:**
```
 :if (beforetag 'quantadj)
 :then 'noun
 :cf '95,5
```

where the Certainty Factors of the Automatic Rules are the ones obtained by the C4.5 tool.