A reconfigurable stochastic tagger for languages with complex tag structure

Łukasz Dębowski
Institute of Computer Science
Polish Academy of Sciences
ldebowsk@ipipan.waw.pl

Abstract

We present a case study of a complex stochastic disambiguator of alternatives of morphosyntactic tags which allows for using incomplete disambiguation, shorthand tag notation, external tagset definition and external definition of multivalued context features. The tagger bases on Naive Bayes modeling and allows for using almost as general context features as in classical trigram taggers as well as more specific ones. Its preliminary results for Polish still do not meet our expectations. Possible sources of the tagger's failures can be: inhomogeneity of the training corpus in preparation, lack of the automatic search of probability models, too general conditional independence assumptions in defining the class of interpretable models.

Automatization of high-quality morphosyntactic tagging for strongly inflective languages, such as Slavic languages, seems to be a much harder task than so called part-of-speech (POS) tagging for weaker inflective languages. An important factor increasing the complexity is the very design of the tagset. Usually, the tags assigned to word-like segments in the former task are long lists of subsequent attribute values, e.g. POS, number, case, gender, person etc. (Hajič and Hladká, 1998; Woliński and Przepiórkowski, 2001), so they provide much more information than almost atomic labels used for POS tagging (Manning and Schütze, 1999). To make the matter harder, many formal descriptions become easier when the tag attribute values are allowed to form RSRL-like type hierarchies (Przepiórkowski et al., 2002). Allowing the values to be partially ambiguous depending on a context raises questions what is *the* accurate level of disambiguation (Woliński and Przepiórkowski, 2001) and how to model it probabilistically in terms of random variables taking disjunctive values (Brew, 1995).

Working for a project aiming at building a large morphosyntactically tagged corpus of written Polish (information site http://dach.ipipan.waw.pl/CORPUS/), we have tried implementing a highly reconfigurable stochastic tagger addressing some of these problems. The main features of our software are as follows:

- The tagger is a contextual disambiguator: It only prunes the lists of tags admissible for successive word-segments, given by a separate morphological analyzer. Superiority of this approach over simulating a stochastic morphological analyzer by a tagger has been discussed in Hajič (2000).
- The tags processed by the tagger have form of short human-readable lists of attribute values. Especially, non-applicable attributes are omitted and multiple atomic values can be given for the same attribute.
- The tagger's internal representation of disambiguation alternatives (tagger's decisions) is different than the list of all admissible atomic tags. In this approach, some kind

of contextually-dependent incompleteness of disambiguation could be learned.

- Special configuration files inform the tagger which tag attributes are to be disambiguated and what multivalued context attribute are relevant for that. The tagger's inference uses a series of Naive-Bayes-like assumptions founded on joint distributions of disambiguation decisions for one tag attribute and values of one context attribute.

- The values of the context attributes are automatically instantiated and smoothed strings whose templates are given in a hand-made configuration file. This approach allows to combine strengths of generality of context attributes as in n-gram models (Brants, 2000; Megyesi, 2001) with their specificity as for binary features in MaxEnt taggers (Ratnaparkhi, 1996; Hajič and Hladká, 1998). Possibility of using alternative files defining the templates of context attributes eases constructive critiques of the particular definition of them.

- The tagger processes XML-formatted texts where input and output files have the same structure. Especially, it can be run on its own output to disambiguate some attributes in cascade rather than simultaneously.

- The tagger can be used for any other language supplied with morphological analyzer, training data and tagset-dependent configuration files.

In the following bulk of paper, we shall present the features of our tagger in more detail, we shall discuss its preliminary results for our Polish tagging project, as well as, we shall share remarks on possible extensions/improvements of the software behavior.

Our general feeling is that the tagger as we have it implemented and configured now for Polish is not a very practical program: It works very slow and it makes much more mistakes than state-of-the-art taggers. In fact, the tagger gives its users so much freedom of manual configuration and feedback information in the error reports that they get lost. We hope that much accuracy can be earned when some automatic search for the optimal configuration files is implemented. On the hand, we are still afraid if we have not *underfitted* with various conditional independence assumptions, which restrain the tagger from seeing sequences of very specific tags as something systematic: Especially, each tag attribute is disambiguated probabilistically independently and the program does not allow for treating tags as atomic entities in probability modeling. This cannot be overcome without a major change in the program and, worse, in the already complex structure of its configuration files.

Despite all these drawbacks, we present some study of how one can think creatively about tagging and how one cannot find a quick break-even when trying to implement too many good guidelines in one piece. We report on lots of apparently technical details, hoping that their exposition can help see the tagger as something more than a black box and identify the sources of its errors.

1 Human-readable tags and tagger's decisions

Before we can define the probabilistic model of our tagger and the scheme of its training, we need to define some pretty abstract entities. These entities are tagger's disambiguating decisions whose conditional probability is maximized given the initial state of the text annotation. The decisions differ from human-readable tags in the corpus. To explain what they are and why they are there, we need to clarify how the tagger interprets the structure of text annotation in a way which allows for running the tagger on its earlier outputs.

Figure 1 presents a general scheme of using the tagger. Any plain text (format 0—possibly with some general XML tags) is first fed through script `analize.plain.text.pl` which identifies word-like segments and gives them their full morphological analysis, i.e. the list of all possible tags (`Morfeusz.pm` is a library serving the Polish morphological analyzer, written by our colleague). After the analysis, the text obtains format 2 which is conserved through multiple calls of the tagger (named `Cypher.pl`). The only thing which happens during these calls is reducing the

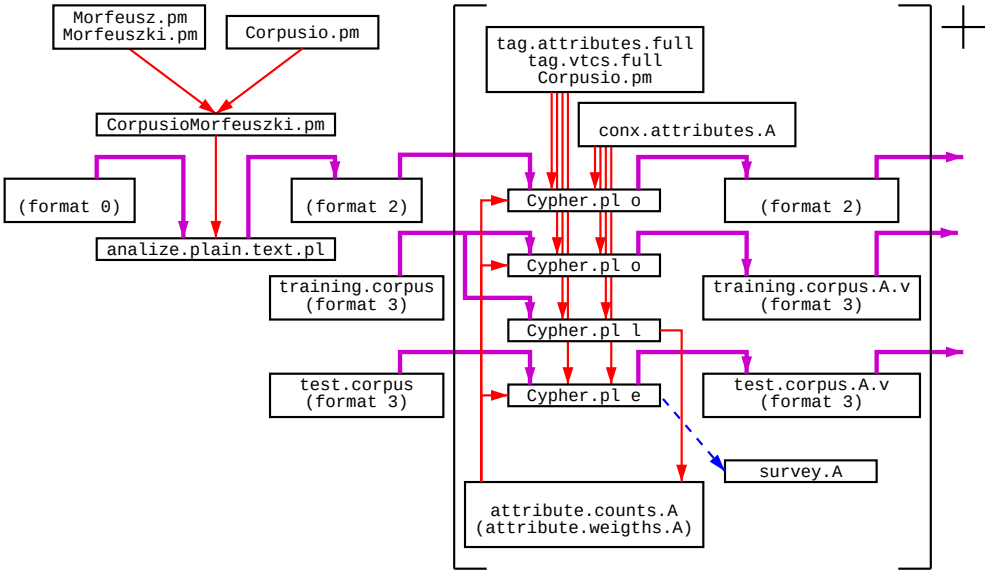

Figure 1: Structure of tagger running.

number of alternative tags initially given by the morphological analysis. Each tagger call can be used to disambiguate different sets of tag attributes depending on the given configuration files (such as `conx.attributes.*`).

In fact, in format 2, each tagged segment (roughly orthographic word) is given two identically formatted lists of potentially ambiguous linguistic annotation: the full morphological analysis and the reduced morphological analysis as pruned by the tagger (initially equal to the full morphological analysis). In format 3, used for training and test corpora, there is a third list containing the morphological analysis as pruned by the human annotators (it shares the same format and it can be only partially disambiguated). During any kind of the tagger call (training, testing, or tagging new texts), it is only the tagger-pruned analyses that are pruned against values of the call-dependent tag attributes. Full morphological analyses and human-pruned analyses remain untouched. Unfortunately, tagger-pruned analyses must be pruned also in the original training and test data to obtain the relevant test and training data for the next tagger call run in a pipe.

Each morphological analysis, full or reduced, appearing in the annotated texts can be ambiguous and it has a form of shorthand human-readable tags grouped by lemmas. Here is an example of the full morphological analysis for the Polish word `kurze` (dust, cock, hen, hen:Adj):

```
<l>kurz<t>subst:pl:nom.acc:m3
<l>kur<t>subst:sg:loc.voc:m2
<l>kura<t>subst:sg:dat.loc:f
<l>kurzy<t>adj:sg:nom.acc:n1.n2:
pos<t>adj:pl:nom.acc:m2.m3.f.n1.
n2.p2.p3:pos\n
```

The tags are lists of values separated by colons (`:`) and dots (`.`). Colons separate tag attributes and dots separate alternative values for the same attribute. When the dot is used for more than one attribute, the tag means the full Cartesian product of alternatives. Values of non-applicable attributes are omitted.

The omission of attributes is not used in the tagger's internal representation of the reduced morphological analysis (called tag list). Here, each attribute has its fixed position, and our example of full morphological analysis is transformed one-to-one into list:

```
[[subst,pl,nom.acc,m3,-,-,...,kurz],
 [subst,sg,loc.voc,m2,-,-,...,kur],
 [subst,sg,dat.loc,f,-,-,...,kura],
 [adj,sg,nom.acc,n1.n2,-,pos,...,
  kurzy],
 [adj,pl,nom.acc,m2.m3.f.n1.n2.p2.p3,
  -,pos,...,kurzy]]
```

This transformation is controlled by file `tag.attributes.*` which specifies names

of consequent tag attributes and enumerations of their values

```
<POS> conj prep subst fin ...
<number> sg pl
<case> nom gen dat acc inst loc voc
...
```

(The last tag attribute is always lemma, which is not enumerative but formally useful as a kind of tag attribute.)

Human-reduced analyses in the training data are allowed to be ambiguous, so the tagger could learn not to disambiguate in some contexts. In this case, one needs to differentiate tagger's lists of disjunctive disambiguation decisions (decision lists) from the tagger-pruned tag lists. In fact, each decision list is a very simple functions of the tagger-reduced tag list. E.g., if it is specified that only number and case can be disambiguated in a given tagger call, the list of decisions for our continued example is:

```
[[LEAVE,LEAVE,LEAVE,LEAVE,...,LEAVE],
 [LEAVE,sg,nom,LEAVE,...,LEAVE],
 [LEAVE,sg,dat,LEAVE,...,LEAVE],
 [LEAVE,sg,acc,LEAVE,...,LEAVE],
 [LEAVE,sg,loc,LEAVE,...,LEAVE],
 [LEAVE,sg,voc,LEAVE,...,LEAVE],
 [LEAVE,pl,nom,LEAVE,...,LEAVE],
 [LEAVE,pl,acc,LEAVE,...,LEAVE]]
```

Decision list contains all choices of atomic values of disambiguated attributes plus special decision `[LEAVE,LEAVE,...,LEAVE]`.

When a decision is eventually selected from the list, the new reduced tag list becomes former reduced tag list unified with the decision against all tag attributes. `LEAVE` and any value `X` unifies downto `X`. Atomic value `Y` and any value `X` unifies downto `Y` if `X` contains `Y` and downto empty set otherwise. For learning and testing purposes, the right decision to be expected from the tagger is computed as follows: When any manually disambiguated tag attribute has just one value, the same attribute value is chosen for the right decision. When the attribute is ambiguous, same attribute for the right decision equals to `LEAVE`.

The differentiation between all the tag attributes and the disambiguated tag attributes was motivated by the occurrence of tag lists close to large Cartesian products: Some Polish participle forms contain alternative values for so many tag attributes, that one would get > 90 disjunctive decisions for one word if disambiguating all attributes simultaneously. Sequential disambiguation reduces polynomial complexity downto linear one and may enable better probability estimation. Additionally, disambiguation of only 4 out of 14 identified morphosyntactic tag attributes usually suffices in Polish for the full tag disambiguation.

2 Probability modeling

The general problem of probabilistic modeling of decision processes is simple to phrase: If we want a particular decision T to depend on all its context, we find that pairs (decision, context state) are unique events and we cannot generalize beyond them. If we group the contexts into too few equivalence classes and make the same decision for all contexts in a class, then we lose sensitivity of the decisions. Finding the optimal equivalence classes can need large amounts of domain knowledge or extensive search. Probability modeling gives additional hint: Define several not so sophisticated partitions of contexts into equivalence classes. Find the joint probabilities of pairs (decision, equivalence class of i-th context partition). From these, compute the joint probability of tuples (decision, equivalence classes for all context partitions) assuming some regularity of this distribution. Depending on the size of available training data, it can give better results than direct estimation of the tuple distribution.

In stochastic text tagging by our tagger, there is a string of decisions for successive word segments (text positions) rather than one decision. Let T_i be the tagger's decision at text position i, $i \in \mathcal{I}$. The admissible disjunctive choices for T_i are $\mathbf{t}_{ij}$, $j \in \mathcal{J}_i$. Both T_i and $\mathbf{t}_{ij}$ are vectors of values T_{it} and $\mathbf{t}_{ijt}$ for disambiguated tag attributes t, $t \in \mathcal{T}$ ($\mathbf{t}_{ijt}$ equals to an atomic tag attribute value or `LEAVE`). On the other hand, there is a set of random variables, called context attributes C_{itc}, $c \in \mathcal{C}_t$, which take values identifying disjunctive equivalence classes of successive context partitions. During tagging, context attribute values are computed using scheme: $C_{itc} = f_{tc}(\Omega, i, \{T_{i't'}\}_{(i',t')\in\mathcal{P}(i,t)})$, where Ω represents the whole input text

(list of word-segments and their input tag lists). $\mathcal{P}(i,t) = (\{i-k,...,i-1\} \times \mathcal{T}) \cup (\{i\} \times (\{0,...,t-1\} \cap \mathcal{T}))$. f_{tc} are fast-computable functions taking repeatable values we describe later.

For a formulated probability model $P\left(T_i = \cdot \,|\, \{C_{itc} = \cdot\}_{t\in\mathcal{T},c\in\mathcal{C}_t}\right)$, the tagger uses Viterbi search to find the optimal string of decisions $\mathbf{t}_{ij_i}$, $i \in \mathcal{I}$, maximizing product

$$\prod_{i\in\mathcal{I}} P(T_i = \mathbf{t}_{ij_i} | \{C_{itc} = f_{tc}(\Omega, i, \left\{\mathbf{t}_{i'j_{i'}t'}\right\}_{(i',t')\in\mathcal{P}(i,t)})\}_{t\in\mathcal{T},c\in\mathcal{C}_t}).$$

(Viterbi search is done as for a non-stationary hidden Markov model of order k). Direct estimation of $P\left(T_i = \cdot \,|\, \{C_{itc} = \cdot\}_{t\in\mathcal{T},c\in\mathcal{C}_t}\right)$ is assumed unfeasible. The tagger approximates it as

$$P\left(T_i = \mathbf{t}_{ij} | \{C_{itc} = \mathbf{c}_{tc}\}_{t\in\mathcal{T},c\in\mathcal{C}_t}\right) = \frac{\prod_{t\in\mathcal{T}} P(T_{\cdot t} = \mathbf{t}_{ijt}) \prod_{c\in\mathcal{C}_t} P(C_{\cdot tc} = \mathbf{c}_{tc} | T_{\cdot t} = \mathbf{t}_{ijt})}{\sum_{j'\in\mathcal{J}_i} \prod_{t\in\mathcal{T}} P(T_{\cdot t} = \mathbf{t}_{ij't}) \prod_{c\in\mathcal{C}_t} P(C_{\cdot tc} = \mathbf{c}_{tc} | T_{\cdot t} = \mathbf{t}_{ij't})}, \quad (1)$$

where numerical values of $P(T_{\cdot t} = \cdot)$, $P(C_{\cdot tc} = \cdot \,|\, T_{\cdot t} = \cdot)$ do not depend explicitly on i. If the decision lists were Cartesian products: $\bigcup_{j\in\mathcal{J}_i} \times_{t\in\mathcal{T}} \mathbf{t}_{ijt} = \times_{t\in\mathcal{T}} \bigcup_{j\in\mathcal{J}_i} \mathbf{t}_{ijt}$, then approximation (1) would be equivalent to assuming that (i) variables $\{C_{itc}\}_{c\in\mathcal{C}}$ are independent given T_{it} (naive Bayes), and (ii) given all $\{C_{itc}\}_{t\in\mathcal{T},c\in\mathcal{C}}$, variables $\{T_{it}\}_{t\in\mathcal{T}}$ are also independent (compare Hajič and Hladká (1998)).

Estimation of probabilities $P(T_{\cdot t} = \cdot)$, $P(C_{\cdot tc} = \cdot \,|\, T_{\cdot t} = \cdot)$ is done using formula

$$P(T_{\cdot t} = \mathbf{t}_{ijt} | C_{\cdot tc} = \mathbf{c}_{tc}) = \frac{\#\text{ of positions } i \text{ such that } T_{it} = \mathbf{t}_{ijt} \text{ i } C_{itc} = \mathbf{c}_{tc}}{\#\text{ of positions } i \text{ such that } C_{itc} = \mathbf{c}_{tc}},$$

where the counts of positions are faked by smoothing procedure described in the next section.

3 Definition of context partitions

Functions f_{tc}, defining the context partitions, are specified in file `conx.attributes.*`. The file is merely read by the tagger and it can be prepared by hand. Table 1 shows an example of this file. First column is the name of some tag attribute, say t-th one. Then, the second column is a string defining f_{tc}. The string defining f_{tc} resembles definition of a decision tree with identical structure of all branches. It represents a succession of tests and variable instantiations evaluated at each text position with decision ambiguous for t-th tag attribute. Larger and larger prefixes of the string are taken into account as long as no false condition is demanded or no instantiation results in disallowed prefixes. (Smoothing procedure provides a list of allowed instantiated prefixes, which is substantially smaller than the list of all syntactically correct instantiations). The longest achieved prefix with instantiated variables is returned as $f_{tc}(\Omega, i, \{T_{i't'}\}_{(i',t')\in\mathcal{P}(i,t)})$.

The meaning of currently recognized commands in the string defining f_{tc} is such:

- `rp:INTEGER_NUMBER:` sets an auxiliary text position as current text position plus `INTEGER_NUMBER`.
- `wd:SOME_STRING:` checks if the segment at auxiliary position equals to `SOME_STRING`.
- `ac:`, `ex:`, `al:`, `my:` followed by `NAME_OF_A_TAG_ATTR:` and `SOME_STRING:` check if `SOME_STRING` for tag attribute `NAME_OF_A_TAG_ATTR` at auxiliary position is: the alternative of available values (`ac:`), one of available values (`ex:`), the only available value (`al:`), the value of decision chosen by the tagger (`my:`).
- `<>:` and `<0>:` are variables replaced for demanded kind of entity with its value at the auxiliary text position (`<>:`) or at the current position (`<0>:`).

For any definition of f_{tc}, such as `rp:-1:wd:nie:rp:0:ac:<POS>:<>:ac:<lemma>:<>`, and each of its maximally long values, such as `C:rp:-1:wd:nie:rp:0:ac:<POS>:inf.impt:ac:<lemma>:iść.iścić`, there is a list prefixes representing less

and less specific information on the context: `C:rp:-1:wd:nie:rp:0:ac:<POS>:inf.impt:ac:<lemma>:iść.ić`, `C:rp:-1:wd:nie:rp:0:ac:<POS>:inf.impt`, `C:rp:-1:wd:nie`, `C:`. For each of these prefixes, except empty one—`C`, there is exactly one prefix immediately shorter. Basing on this property, we have implemented the following smoothing of f_{tc} values:

1. During learning, the alphabet of allowed instantiated prefixes is not closed, so for each f_{tc}, counts of all its maximally long values are collected.

2. Each f_{tc} value passes all its counts to its immediate prefix if it was seen less than threshold (3 times).

3. The alphabet of allowed values of f_{tc} is fixed as the set of all prefixes of f_{tc} values currently counted positively.

4. For each f_{tc} value in the allowed alphabet, it is faked it was also seen (once) with special decision value T_{it} =`SMOOTH`.

5. During tagging, f_{tc} yields the longest allowed and matching instantiated value $\mathbf{c}_{tc}$. If $\mathbf{c}_{tc}$ has no positive count with particular asked decision value $T_{it} = \mathbf{t}_{ijt}$, $\mathbf{t}_{ijt}$ is treated as it were `SMOOTH`.

In the adopted disambiguation scheme, the order k of nonstationary Markov model to be used in Viterbi search is the negative of minimal argument of `rp:` commands followed by `my:` command. Thanks to that, given file `conx.attributes.*`, our tagger identifies k automatically and accepts any value of k.

4 First results

In this section, we would like to present several scores of our tagger on processing Polish texts. The scores should be considered preliminary due to several causes: (i) we believe that our `conx.attributes.*` file is not optimal yet, (ii) improving the tagger's code continually, we have had too little time to test its behavior on large training data sufficiently, (iii) the training corpus is still in a mix of two different degrees of disambiguation of gender values, (iv) morphological analyzer contains no guesser and unrecognized strings are not considered ambiguous by the tagger.

In our current Polish annotation scheme, tags consist of the following attributes: POS, number, case, gender, person, degree, aspect, negation, depreciation, accommodability, accentability, post-prepositionality, vocalicity, punctuation, lemma (Woliński and Przepiórkowski, 2001). We have observed that for given morphological analysis, any atomic tag is almost always identified given its values for just 4 attributes: POS, number, case, gender, lemma, and decided to disambiguate directly only these attributes.

Testing several *simple* versions of `conx.attributes.*` file, we have observed that smaller error on POS is obtained when POS is disambiguated simultaneously with number, case, and gender while lemma is to be disambiguated afterwards.[1] The best `conx.attributes.*` file we have found so far is presented in table 1. The general structure of this model of f_{tc}s resembles a product of trigram models for each of the disambiguated attributes. Some modification of `conx.attributes.*` file might be adding POS values at positions -1,0,+1 to f_{tc} values for number, case, and gender. Theoretically, this modification could be more sensitive to a regular grammar syntax of simple phrase structures. Strangely, this modification yields twice as high error rate as the original `conx.attributes.*` file.

Observed error rates for f_{tc}s as in table 1 are:

- Training data 01k (784 word segments)

Error rate:	(all tokens)	(ambiguous)
Overall	0.25	0.45
on POS	0.06	0.10
on number	0.06	0.10
on case	0.16	0.28
on gender	0.13	0.22

- Training data 10k (8118 word segments)

[1] Some major ambiguity left for lemma disambiguation in this case are homonymous present tense forms for *musieć* – 'to have' and *musić* – 'to compel'.

```
<POS>       ac:<POS>:<>:ac:<lemma>:<>
<POS>       ac:<POS>:<>:rp:-1:my:<POS>:ac:<lemma>:<>
<POS>       my:<POS>:<>:rp:+1:ac:<POS>:ac:<lemma>:<>
#
<number>    ac:<number>:<>:rp:-1:my:<number>:<>
<number>    ac:<number>:<>:rp:+1:ac:<number>:<>
#
<case>      rp:-1:my:<case>:<>
<case>      rp:+1:ac:<case>:<>
#
<gender>    ac:<gender>:<>:rp:-1:my:<gender>:<>
<gender>    ac:<gender>:<>:rp:+1:ac:<gender>:<>
```

Table 1: Preliminary `conx.attributes.*` file used for Polish.

Error rate:	(all tokens)	(ambiguous)
Overall	0.23	0.41
on POS	0.06	0.11
on number	0.06	0.10
on case	0.14	0.24
on gender	0.11	0.19

- Training data 50k (41208 word segments)

Error rate:	(all tokens)	(ambiguous)
Overall	0.20	0.34
on POS	0.05	0.08
on number	0.04	0.08
on case	0.12	0.20
on gender	0.07	0.13

- Full training data (558224 word segments)

Error rate:	(all tokens)	(ambiguous)
Overall	0.22	0.35
on POS	0.05	0.09
on number	0.05	0.08
on case	0.12	0.21
on gender	0.10	0.17

All trained models were tested on test data 5k (4117 word segments).

The error rate for full training data is larger than for 50k training data, especially on gender. This is probably due to the inhomogenous manual annotation of gender values we had already mentioned. Comparing with publications on similar tasks, our minimal overall error rate is twice as big as for Slovene (Džeroski et al., 2000) and 3 times as big as for Czech (Hajič and Hladká, 1998).

5 Comments and possible extensions

It has been remarked that HMM trigram taggers using single multivalued context features can perform better and run faster than MaxEnt taggers trying to combine conditional probabilities for a multitude of binary features (Brants, 2000; Megyesi, 2001), even for large structured tagsets and Slavonic free word-order (Hajič et al., 2001). Our intention was to try out a hybrid approach in which a small number of multivalued context features is used. We have thought it can help solving data sparseness problems. Now we do not know exactly what is the most important cause of our present high error rate: inhomogenous Polish corpus annotation, deficiencies of morphological analysis, low efficiency of the manual model search, or the assumption of probabilistic independence of different tag attributes (used succesfully by Hajič and Hladká (1998)). Due to the last thing, we cannot either simulate a simple trigram tagger interpreting tags as single entities.

So as to overcome the manual model search, we have thought of writing another program which could read the error reports of our tagger and search for the optimal `conx.attributes.*` file in a fixed space of context functions f_{tc}s. For binary context functions, Hajič and Hladká (1998) truncated the length of their definitions and searched extensively through a very large finite space of them. We think that optimal f_{tc} definitions may be so long that genetic algorithm search through the infinite space of f_{tc}s may be a better approach.

One might also search for more powerful se-

mantics of context functions f_{tc}. As long as f_{tc}s do not depend on too distant previous tagger decisions (increase the order k of Markov model), or do not need too much computation on their own, one can freely decide what functions of morphologically analysed text f_{tc}s are. Better disambiguation of Slavic nominative/accusative ambiguity may need testing if there is a possible verbal form in the left/right context of a word (Hajič and Hladká, 1998). Disambiguation of complex gender ambiguities may need testing unifiability of the attribute values.

Our idea of smoothing f_{tc} values resembles HMM reconstruction algorithm proposed by Shalizi, Shalizi and Crutchfield (2003), which uses additionally Kolmogorov-Smirnov test to check if extending the context depth is necessary for adequate prediction. Similarly, we could replace f_{tc} value $\mathbf{c}_{tc}$ by its prefix $\mathbf{c'}_{tc}$ not only when $\mathbf{c}_{tc}$ is rare but also when $P(T_{\cdot t} = \cdot | C_{\cdot tc} = \mathbf{c}_{tc})$ does not differ significantly from $P(T_{\cdot t} = \cdot | C_{\cdot tc} = \mathbf{c'}_{tc})$. Such technique could decrease memory usage and slightly speed up the tagger.

Graña, Alonso and Vilares (2002) proposed a modification of Viterbi search for unknown text segmentation as a common solution for disambiguation of segmentation and tagging. Our Polish tagset introduces some very rare ambiguities of segmentation but we have consciously decided not to touch the segmentation since extending the modification of Viterbi search to multiattribute tags with independent tag attributes needs a very different structure of training corpus and probability estimation. Such tool would be formally capable of performing functionality of a cascade phrase parser and would be a good subject of another project.

The research reported here was partly supported by the KBN grant 8 T11C 043 20.

References

Thorsten Brants. 2000. TnT —a statistical part-of-speech tagger. In *Proceedings of the Sixth Applied Natural Language Processing Conference (ANLP-2000)*. Seattle, WA, USA.

Chris Brew. 1995. Stochastic HPSG. In *Proceedings of the 7th Conference of the European Chapter of the Association for Computational Linguistics*.

Sašo Džeroski, Tomaž Erjavec, and Jakub Zavrel. 2000. Morphosyntactic Tagging of Slovene: Evaluating PoS Taggers and Tagsets. In *Second International Conference on Language Resources and Evaluation, LREC'00*, pages 1099–1104, Paris. ELRA. `http://nl.ijs.si/et/Bib/LREC00/lrec-tag-www/`.

Jorge Graña, Miguel A. Alonso, and Manuel Vilares. 2002. A common solution for tokenization and part-of-speech tagging. In *Proceedings of Text, Speech and Dialogue 2002. Lecture Notes in Artificial Intelligence 2448*. Springer Verlag.

Jan Hajič and Barbora Hladká. 1998. Tagging inflective languages: Prediction of morphological categories for a rich, structured tagset. In *Proceedings of COLING-ACL Conference*. Montréal.

Jan Hajič, Pavel Krbec, Pavel Květoň, Karel Oliva, and Vladimír Petkevič. 2001. Serial combination of rules and statistics: A case study in czech tagging. In *Proceedings of ACL'01*. Toulouse, France.

Jan Hajič. 2000. Morphological tagging: Data vs. dictionaries. In *Proceedings of ANLP-NAACL Conference*. Seattle.

Christopher D. Manning and Hinrich Schütze. 1999. *Foundations of Statistical Natural Language Processing*. The MIT Press.

Beáta Megyesi. 2001. Comparing data-driven learning algorithms for PoS tagging of Swedish. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP 2001)*. Carnegie Mellon University, Pittsburgh, PA, USA.

Adam Przepiórkowski, Anna Kupść, Małgorzata Marciniak, and Agnieszka Mykowiecka. 2002. *Formalny opis języka polskiego: Teoria i implementacja*. Akademicka Ofi cyna Wydawnicza EXIT, Warszawa.

Adwait Ratnaparkhi. 1996. A maximum entropy model for part-of-speech tagging. In *Proceedings of the First Conference on Empirical Methods in Natural Language Processing (EMNLP-1996)*. University of Pennsylvania, PA, USA.

Cosma Rohilla Shalizi, Kristina Lisa Shalizi, and James P. Crutchfi eld. 2003. An algorithm for pattern discovery in time series. `http://www.arxiv.org/abs/cs.LG/0210025`.

Marcin Woliński and Adam Przepiórkowski. 2001. Projekt anotacji morfosynktaktycznej korpusu języka polskiego. Technical Report 938, Institute of Computer Science, Polish Academy of Sciences.