Hand-Written and Automatically Extracted Rules for Polish Tagger

Maciej Piasecki

Institute of Applied Informatics, Wrocław University of Technology,
Wybrzeże Wyspiańskiego 27, Wrocław, Poland
maciej.piasecki@pwr.wroc.pl

Abstract. Stochastic approaches to tagging of Polish brought results far from being satisfactory. However, successful combination of hand-written rules and a stochastic approach to Czech, as well, as some initial experiments in acquisition of tagging rules for Polish revealed potential capabilities of a rule based approach. The goals are: to define a language of tagging constraints, to construct a set of reduction rules for Polish and to apply Machine Learning to extraction of tagging rules. A language of functional tagging constraints called JOSKIPI is proposed. An extension to the C4.5 algorithm based on introducing complex JOSKIPI operators into decision trees is presented. Construction of a preliminary hand-written tagging rules for Polish is discussed. Finally, the results of the comparison of different versions of the tagger are given.

1 Introduction

The statistical approach of [1] to tagging of Polish brought results far from being satisfactory. However the accuracy of statistical tagger for Czech, another inflective language, has been significantly improved after combination with hand-written rules [2]. Experience collected during experiments in acquisition of tagging rules for Polish [3] on the basis of Genetic Algorithms revealed potential capabilities of a rule based approach but showed also that the inefficient extraction algorithm had to be changed. This work follows this path. Similarly to [2], we want to combine hand-written rules with Machine Learning, but as in [3], we tend to base the whole tagging process on the rules. Our claim is that rules written in a symbolic language with high expressive power are a much better solution for tagging inflective languages than solutions relying on stochastic models.

The first goal is to define a symbolic language for expressing *tagging constraints*, in the sense similar to [4], i.e. constraints that must be preserved in all natural language expressions, or at least often enough. Such constraints must have local character and be verifiable without extensive parsing. But the main goal is to apply this language as a common tool for supporting automatic extraction of rules and expressing hand-written rules. We want to be able to express in this language both: simple constraints that are parts of automatically extracted rules and complicated premises of hand-written rules of tags reduction. Finally, we are going to present an algorithm for rules extraction and a preliminary set of hand-written rules for Polish.

Petr Sojka, Ivan Kopeček and Karel Pala (Eds.): TSD 2006, LNAI 4188, pp. 205–212, 2006.

2 JOSKIPI Language

The proposed language of constraints has a lot in common with its direct ancestors, namely [4,5], and [6]. The difference is in its primary application. As the intended Machine Learning algorithm for the extraction of rules should produce highly efficient rules on the basis of large corpus, we put the main stress on defining constraints working in a functional style. Such constraint, used as a part of extracted rules, should take context as input, check its state and return some information describing the state. Partial information delivered by component constraints is combined by the logic of the extracted rule. The intended algorithm was C4.5 [7]. Moreover, this work remotely inherits some ideas from Hermjakob and its work on parser learning [8].

A language called JOSKIPI (the acronym of a Polish name meaning 'the language of the description of the state in the IPI PAN Corpus' [9]) is proposed. The core of JOSKIPI is a set of predefined *operators* of the three main classes:

1. *simple operators* — atomic, return a set of symbolic values or strings,
2. *test operators* — compound, return a boolean value,
3. *conditional operators* — compound, include an operator and a test operator, the value of the operator is returned on the condition that the test is fulfilled.

The work of simple operators is closely related to the structure of a token description in the IPI PAN Corpus (further IPIC) XML format presented below:

<tok> <orth>*token*</orth>
<lex><base>*base_form*$_1$</base><ctag>*tag*$_n$</ctag></lex>
...
<lex><base>*base_form*$_n$</base><ctag>*tag*$_n$</ctag></lex> </tok>

Moreover, a *positional tag* in IPIC is a sequence of values:

<ctag>gramm_class:cat_val$_1$:...:cat_val$_k$</ctag>

The first value encodes one of the 32 *grammatical classes* (a more finer-grained division than parts of speech), e.g. *adjective*, *ad-adjectival adjective*, or *non-past form*. The following positions correspond to *grammatical categories* (12 in total), e.g. gender, aspect, or vocalicity, appropriate for the given grammatical class. For a given class each category is determined by position. An IPIC tag is considered to be *a structure of attributes* in JOSKIPI.

Simple operators allow for reading the state of a specified token. The *position* of the token can be directly specified by an *offset* from the *centre of the context*, e.g. `orth[-1]` returns a *string* (a sequence of characters) — a token first to the left from the centre. The centre of the context is always the token being currently disambiguated. A position can also be specified by a *variable*[1], e.g. `base[$P]` returns a *set of strings*. The size of the returned set is one if the token in the position stored in `$P` is non-ambiguous according to its base form. Other simple operators return sets of symbolic values of other attributes of tags assigned to the pointed token, e.g. grammatical class, case, number, gender etc. If there is no token in the pointed position or the pointed token does not possess the given attribute in any of its tags, the *empty value* `none` is returned. Names of the simple operators follow exactly the mnemonics used in the query language of IPIC. Filtered simple operators allow for reading values from attributes of tags fulfilling specified condition, e.g. `catflt(0,{nom,acc},{nmb})` reads

[1] Only variables over positions exists in JOSKIPI.

the number of the token 0 from its tags with case values in the given set, when there are no such tags the operator returns `none`.

Test operators construct compound expressions. There are five subclasses:

1. *sets comparison*: `equal(`o_1`,`o_2`)`, `inter(`o_1`,`o_2`)`, `in(`o_1`,`o_2`)`, where o_i can be an operator or a constant value, e.g. `equal(pos[0],pos[-2])`, where `pos` returns a set of grammatical classes, or `in(case[1], {nom, gen, acc})`;
2. *logical conjunctions*: `and`, `or`, and `not`, where the last one means *not or* when applied to more than one argument,
3. *search*: `llook(`pos_{st}`,`pos_{end}`,$Var,`*test*`)` and `rlook`, where positions can be variables or constants (e.g. the begin and the end of a sentence), `$Var` is an obligatory iterator, both operators look for the first token fulfilling *test*, set `$Var` to its position and return `true`, otherwise return `false`;
4. *condition fulfilment over a sequence*: `only(`pos_{st}`,`pos_{end}`,$Var,`*test*`)` and a similar `atleast`;
5. and *agreement*: `agrpp(`pos_1`,`pos_2`,{`*mask*`},`n`)` and `agr` with the same syntax, where *mask* is a set of names of attributes and/or values, `agrpp` checks agreement between a pair of tokens, while `agr` over a sequence, e.g. `agrpp(0,-1,{pl,gnd,gen},3)` checks *possible* agreement the two tokens: 0 and -1 on gender but only for tags with *number=pl* and *case=gen*.

Search operators setting some variables, when used in combination with other operators, can construct complex conditions over a large sequences of tags. The agreement operators add also a lot to the large expressive power of JOSKIPI.

Conditional operators, of the scheme: *op* `?` *test*, combine any operator with a test operator and return `none` when the test is not fulfilled, e.g.
`cas[$Cs]?rlook(1,end,$Cs, not(equal(cas[$Cs],{none}) ))`

JOSKIPI has been implemented directly in C++ in order to achieve high efficiency. Especially, the implementation of `agr` had to be very careful, as in a sequence of tokens each can have many tags and the number of possible paths of agreement across the tags is huge. Instead of searching, the problem was considered as to be a *Constraint Solving Problem*.

3 Preliminary Rules

In JOSKIPI reduction rules have the following scheme:
`delete(` $test_{del}$ `) # ` p `:-` $test_{app}$, where $test_{del}$ and $test_{app}$ are any test operators and p is the priority of the rule.

The value of $test_{app}$ decides about applicability of the rule. If the rule is applicable, then $test_{del}$ is evaluated for each tag of the 0 token i.e. the 0 token is presented several times to $test_{del}$ as possessing only one tag. Tags for which $test_{del}$ returns `true` are deleted from the 0 token. In order to protect against errors in IPIC (or text), if all tags are removed by a rule, then all are restored.

The staring point for the construction of our set of rules was the work of Rudolf [6] and a few rules in [3]. Unfortunately, Rudolf's rules are expressed in natural language, and in this form most of them are very imprecise and can be only guidelines. From the 17 rules presented in [6] only one (pp. 95 in [6]), presented below, survived in its original form in tests on IPIC:

```
delete(equal(pos[0],{fin})) # 150 :-
and( inter(pos[0],{fin}), equal(pos[1],{fin}) )
```

The above rule states that an ambiguous token cannot be a verb non-past form (`fin`) if followed by another non-past form. Oliva and Petkevič [10] claim that this rule is universal for all Slavic languages and our research supports this hypothesis, as it produces no errors tested on IPIC.

However, the other variant of this rule, where an unambiguous `fin` precedes a token which can be `fin` , have had to be refined:

```
delete(equal(pos[0],{fin})) # 140 :-
and( inter(pos[0],{fin}), equal(pos[-1],{fin}),
not( and( in(orth[-2],{‘‘jest’’,‘‘znaczy’’}),equal(orth[-3],{‘‘to’’})
) ) )
```

The additional constraints protect a little outdated constructions of the type *to znaczy* `fin` (gloss. *it means* `fin`) appearing several times in IPIC.

Inspired by [6] we tried to explore all cases of obligatory agreement in Polish and formulated a *preliminary set of tagging rules for Polish*. The set is called ‘preliminary’ as the work was time consuming, and we concentrated on very efficient and general rules first. The number of rules completely tested is 24. The work on rules is performed according to the following scheme:

1. An initial naive version of a rule is formulated.
2. The rule is tested on IPIC with the help of a special *Rule Debugger* tool.
3. *Rule Debugger* records positions of all cases in which a tag selected by a human was deleted by the rule.
4. The recorded exceptions are analysed in IPIC with the help of a constructed editor called *Manufakturzysta*. The errors are immediately corrected in IPIC.
5. If the rule produces some exceptions, it is refined and the process is repeated.

As we have no space for a detailed presentation of the rules, only the main groups of rules are described (all used in tests, Sec. 5).

Separation of two non-past forms — the two members of this group have been already presented. The other rules of this group block the occurrence of two verbs in non-past forms even if there are one or two *particle-adverbs* (`qub`) between the two tokens. However, if we add adverbs, we will encounter too many errors (i.e. mistakenly deleted tags).

Case after preposition — only tags agreeing in case with a preposition are left after the preposition. This almost always working rule had to be divided into several at least. Firstly, potential possessive pronoun *jego* (*his*) had to be treated separately. Secondly, numerals in dates or currencies break this rule (separate rules needed). Thirdly, genitive case can always appear as the indicator of possessive construction. And finally, adjective after preposition needs separate complicated rule taking into account some collocations.

Token “z”/“z” as preposition — cannot be a preposition if there is no following token with instrumental case as it is claimed in [6]. We added genitive case to the list and a collocation *z tak*.

Genitive case after numerals — in the preliminary set only a rule for indefinite numerals like *mało* (*little*) or *mnóstwo* (*plenty*) (being adverbs in IPIC) has been formulated. A set of rules for main numerals is in development.

Plural number after numeral — works well for nouns except cases of names of units, but adjectives in IPIC include ordinals which complicates the picture.

*Agreement of relative pronoun "który" (*which/who*) and noun in number and gender* — mainly works with the additional condition of the presence of ',', but anyway produces some small error rate.

4 Operator Based Learning

Large workload on manual construction of rules was predicated before we started, and from the very beginning we wanted to extract the bulk of rules automatically. As the application of rules should be efficient, we were looking for some simple form. Encouraged by the positive results in [8], as the first attempt, we decided to extract rules in the form of Decision Trees (DTs) by the C4.5 algorithm [7]. The rules of both types were applied in the reductionistic tagger described in [11]. The tagger works in three phases. During each phase a partial disambiguation is performed: firstly *grammatical class*, secondly *number and gender* and thirdly *case*. Most of the other attributes are dependent on those four, with the exception of: *aspect* in case of non-past forms of verbs ('present tense', third person) and *accentability* and *post-prepositionality* in the case of 3rd person pronouns.

DTs are constructed not for the whole phases but for *ambiguity classes*, following [12]. An ambiguity class is a set of tokens possessing the same set of possible values of some tag attribute or attributes, e.g. `{adj fin subst}` is an example of ambiguity class of the first phase. There are 143 DTs constructed: 61 (1st phase), 48, and 34. DTs encode from several to many thousands of rules.

The general idea for encoding more sophisticated rules by DTs is that each node of DT corresponds to the application of some JOSKIPI operator. During learning all operators of DT are applied in advance for example tokens and the returned values are stored as sequences. The sequences are next passed to the implementation of C4.5 as learning examples. Created DTs encodes identifiers of operators in their nodes. During tagging our implementation of DT while is entering some node requests from the environment application of the appropriate operator to the current context. After the value is returned, our DT compares the returned value with values assigned to the branches leaving the node. One of the branches is chosen and DT traverses to the next node corresponding to the next operator. The important limitation of this mechanism is that operators in DTs work independently. A highly expressive mechanism of joining by position variables is not avaible. But the operator encoded in the nodes can be of any complexity.

Each DT is specified by a JOSKIPI expression called a *pattern*:

ambiguity_class_specification # a_sequence_of_operators ,

where the specification of an ambiguity class is written as a set of mnemonics of possible values. The backbone of each DT is a set of simple operators called a *standard vector*, e.g. for DTs of the first phase the standard vector includes:

```
pos[-3]...pos[1] pos[2] cas[-3]...cas[2]
gnd[-3]...gnd[2] nmb[-3]...nmb[2]
```

We have been extending patterns with more complex operators for each DT individually. Firstly by manual inspection of members of the given ambiguity class in IPIC, we tried to

identify some distinguishing features, e.g. `equal(orth[0],{“kiedy”})` in the pattern for `{conj qub}` points to the behaviour of the specific word, for which DT should build some specific rules. Secondly, we tried to formulate some relaxed linguistic constraints concerning agreement or some common word order, e.g. whether there is a particle *się* somewhere to the left and between it and the centre there are only tokens of some specified classes. Such constraints were tested by building pseudo-rules and running them by *Rule Debugger* on IPIC and next analysing the exceptions. Finally, the statistics of a set of learning examples generated by some preliminary version of a pattern was analysed. We could see how promising are the different versions of operators for C4.5. For example, we formulated an conditional operator checking whether two sequences of tokens agreeing on case, number and gender that are joined by a conjunction have a common case value. Mostly the operator returns `none`, but in some situations its boolean values are nicely correlated with a choice of some class. Unfortunately, for C4.5 this happens to rarely to include this operator in the given DT.

An example of a more complex operator (but a shorter one) is an operator looking for an adjective somewhere to the right which agrees with the 0 token:

```
!AdjPRight
or( and( inter(pos[1],adj,ppas,pact),
agrpp(0,1,cas,gnd,nmb,3) ),
and( rlook(2,end,$Adj,inter(pos[$Adj],{adj,ppas,pact})),
agrpp(0,$Adj,{cas,gnd,nmb},3),
only(1,$-1Adj,$Q,inter(pos[$Q],{adv,qub}) ) )
)
```

There are 11 operators of similar complexity applied in DTs of the first phase (grammatical class). The most sophisticated, but extensively used by C4.5 is the operator testing whether the 0 token can be a potential subject of some verb in the context. DTs of the second phase (number and gender) utilise several simpler operators and 8 complex. DTs of the third phase (case) use 10 complex operators, and a lot of simpler ones. Obviously, for each phase a version of a standard vector is defined.

5 Results and Conclusions

The tagger using the rules and DTs based on JOSKIPI operators achieved the accuracy of 92.55% (84.75% for ambiguos words). However, it must be emphasised that in addition to the simplifications disscussed in Sec. 4, the tagger does not distinguish *nouns* from *gerunds* on the other base than number, gender and case. All tokens *noun*/*gerund* ambiguous, if not disambiguated on the basis of the three attributes, are assigned two tags at the end. The accuracy of the tagger has been evaluated in ten-fold test on the learning part IPIC (LIPIC) including 885 669 tokens. All cases in which the set of tags assigned by the tagger has a non-empty intersection with the set assigned by a human were counted as proper decisions. In order to analyse what is the influence of hand-written rules and of the use of sophisticated operators in DTs, we prepared and tested 4 different versions of the tagger (Tab. 1)[2]:

[2] The results reported earlier were increased due to a programmer error in the test.

- *a full tagger* (T): hand-written rules and all types of operators in DTs,
- *a tagger without hand-written rules* (T-HR) but still DTs are constructed with the use of complex operators,
- *a tagger without hand-written rules and complex operators in DTs* (T-HR-C) in which DTs are constructed on the basis of standard vectors plus some simple operators checking existence of some particular words in the fixed positions,
- and *a tagger applying hand-written rules, but not using complex operators in DTs* (T+HR-C).

Table 1. Comparison of the accuracy [%] of different versions of the tagger

tagger	**all tokens**	all max.	all min.	**ambiguous**	amb. max.	amb. min.
T	**92.55**	93.04	91.98	**84.75**	86.12	83.4
T-HR	**91.60**	92.03	90.97	**82.70**	83.27	81.94
T-HR-C	**91.43**	91.86	90.76	**82.54**	83.16	81.38
T+HR-C	**91.75**	91.94	91.48	**82.8**	83.31	82.39

In Tab. 1 we can observe, that switching off both: the hand-written rules and the complex operators decreases significantly the accuracy. The mutual relation between both changes is less clear, as the decrease of accuracy in the case of T-HR-C is marginal in comparison to T-HR. But the tagger T+HR-C shows that the lack of the complex operators in DTs is as important, as the lack of the hand-written rules! The small difference between T-HR and T-HR-C can be the result of construction by DTs some general rules with the help of the complex operators. Such rules can express some important linguistic constraints. The complex operators are often used in the top parts of DTs.

The preliminary set of rules is very small, but very significant, see the result of T-HR. The rules when applied to LIPIC activated for 76 543 tokens and removed 187 895 tags. The tagger leaves at average **1.03 tags per token** (initially 2.87). The estimated speed of the full tagger is about 4000 tokens per second on the PC 512 MB RAM 2.41 GHz.

The accuracy is lower than [2], but it seems to be significantly improved in comparison to the previous Polish taggers [1,3]. Further work must be done on enlarging the set of hand-written rules. We plan also to look for an algorithm of extraction of more expressive rules based on JOSKIPI than DTs.

Acknowledgement. This work was financed by the Ministry of Education and Science projects No 3 T11C 003 28 and No 3 T11C 018 29.

References

1. Łukasz Dębowski: Trigram morphosyntactic tagger for Polish. In Mieczysław A. Kłopotek, Wierzchoń, S.T., Trojanowski, K., eds.: Proceedings of Intelligent Information Processing and Web Mining. Proceedings of the International IIS:IIPWM'04 Conference held in Zakopane, Poland, May 17–20, 2004. Springer Verlag (2004) 409–413.
2. Hajič, J., Krbec, P., Květoň, P., Oliva, K., Petkevič, V.: Serial combination rules and statistics: A case study in czech tagging. In: Proceedings of The 39th Annual Meeting of ACL, Morgan Kaufmann Publishers (2001) 260–267.

3. Piasecki, M., Gaweł, B.: A rule-based tagger for Polish based on Genetic Algorithm. [13].
4. Karlsson, F., Voutilainen, A., Heikkil *a*, J., Anttila, A., eds.: Constraint Grammar: A Language-Independent System for Parsing Unrestricted Text. Mouton de Gruyter, Berlin and New York (1995).
5. Květoň, P.: Language for grammatical rules. Report TR-2003-17, ÚFAL/CKL MFF UK, Prague (2003).
6. Rudolf, M.: Metody automatycznej analizy korpusu tekstów polskich. Uniwersytet Warszawski, Wydz. Polonistyki (2004).
7. Quinlan, J.: C4.5: Programms for Machine Learning. Morgan Kaufmann (1993).
8. Hermjakob, U.: Learning Parse and Translation Decisions From Examples With Rich Context. PhD thesis, University of Texas, Austin (1997).
9. Przepiórkowski, A.: The IPI PAN Corpus Preliminary Version. Institute of Computer Science PAS (2004).
10. Oliva, K., Petkevič, V.: Morphological and syntactic tagging of slavonic languages. Lecture Notes for Empirical Linguistics and Natural Language, Fall School, Sozopol (2002).
11. Piasecki, M., Godlewski, G.: Reductionistic, Tree and Rule Based Tagger for Polish. [14].
12. Márquez, L.: Part-of-speech Tagging: A Machine Learning Approach based on Decision Trees. PhD thesis, Universitat Politécnica de Catalunya (1999).
13. Mieczysław A. Kłopotek, Wierzchoń, S.T., Trojanowski, K., eds.: Proceedings of Intelligent Information Processing and Web Mining, 2005. Advances in Soft Computing. Springer, Berlin (2005).
14. Mieczysław A. Kłopotek, Wierzchoń, S.T., Trojanowski, K., eds.: Proceedings of Intelligent Information Processing and Web Mining 2006. Advances in Soft Computing. Springer, Berlin (2006).