TASK QUARTERLY 11 No 1–2, 151–167

POLISH TAGGER TaKIPI: RULE BASED CONSTRUCTION AND OPTIMISATION

MACIEJ PIASECKI

Institute of Applied Informatics, Wroclaw University of Technology,
Wybrzeże Wyspiańskiego 27, 50-370 Wroclaw, Poland
maciej.piasecki@pwr.wroc.pl

(Received 27 December 2006; revised manuscript received 27 January 2007)

Abstract: A large number of different tags, limited corpora and the free word order are the main causes of low accuracy of tagging in Polish (automatic disambiguation of morphological descriptions) by applying commonly used techniques based on stochastic modelling. In the paper the rule-based architecture of the TaKIPI Polish tagger combining handwritten and automatically extracted rules is presented. The possibilities of optimisation of its parameters and component are discussed, including the possibility of using different methods of rules extraction, than C4.5 Decision Trees applied initially. The main goal of this paper is to explore a range of promising rule-based classifiers and investigate their impact on the accuracy of tagging. Simple techniques of combing classifiers are also tested. The performed experiments have shown that even a simple combination of different classifiers can increase the tagger's accuracy by almost one percent.

Keywords: morphosyntactic tagging, Polish, rule based tagging, decission trees

1. Introduction

In inflectional languages like Polish, Czech or Russian, the form of a word carries information not only about its Part of Speech (POS), but also about *morphological categories* characterizing the word, like: *number, gender* or *case*. POS and the values of the morphological categories determine to a large extent the possible grammatical relations of a given word in a text (an utterance). Thus, we will henceforth refer to POS and the morphological categories as *morphosyntactic features a word form*. Identifying the values of morphosyntactic features appropriate for an occurrence of a word in a text may be valuable for many applications in Natural Language Processing. However, it cannot be based on a dictionary only – many words are ambiguous with respect to the values of the features, *e.g.* the Polish word *nie* is ambiguous between a *particle* (English adverb *not*) and different forms of the personal pronoun ∼*he* (all forms in the accusative case). As a result, *nie* possesses 10 possible descriptions with different combinations of features.

It is typical for inflectional languages that possible morphosyntactic descriptions are first assigned to a word, independently of the context, by the *morphological*

analyzer, and only then a *tagger*[1] selects a description appropriate for the given context. The name *tagger* comes from the practice of describing words by attaching *tags* representing morphosyntactic descriptions; the process is called *tagging*.

The main idea of assessing the accuracy of tagging is to compare taggers' decisions with decisions made by a linguist for the same text. Technically, the accuracy may be measured in several ways. Mostly, the percentage of proper decisions in relation to all *tokens* in a text is given. A *token* is a word-level segment of text identified according to some assumed rules, *e.g.* a word, a punctuation mark, a suffix, a number, a symbol, *etc.* According to the rules of segmentation proposed for the largest corpus of Polish, namely IPI PAN Corpus (henceforth IPIC, see [1]), some word forms are not separated by two spaces, *e.g. chciałbym* (∼*I would like to*) is divided into three segments, called in [1] *tokens*: *chciał* + *by* + *m* (∼ $like_{\mathrm{sg,\,1st\,per,\,past}}$ + a particle + a form of *to be*). Henceforth, following the standard of IPIC, we will refer to the basic text segments as *tokens*. Some of them possess morphological descriptions, while others do not. The latter are assigned a special POS label written `ign` in IPIC.

The problem of tagging is almost solved for many natural languages as taggers typically achieve accuracy in excess of 95% (in relation to all tokens, see [2]). However, this is not the case with tagging in Polish. The task is substantially more difficult and there is much less available training data. There are 4179 theoretically possible tags in IPIC, but only 1642 of them occur in the manually disambiguated part of IPIC (henceforth MIPIC) *i.e.* 885 669 tokens [3]. The lower accuracy of Dębowski's statistical tagger [4] constructed on the basis of MIPIC was mainly attributable to sparseness of the data. For English, the number of tags is between 45 and 197 in different corpora, while the learning corpus usually in exceeds a million tokens (*e.g.* see [2]).

Taking into account the relatively small size of MIPIC, the free word order in Polish and the existing successful approaches to tagging of Czech utilizing *tagging rules*, we have assumed that the architecture of a Polish tagger should be based on rule-like representation of knowledge.

A tagger based on handwritten rules can achieve very good accuracy, *e.g.* EngCG [5], adopted for some languages, but manual construction of rules can be extremely laborious, as in the case of 134 rules to disambiguate the particle *se* in Czech [6]. Thus, our main goal has been to explore possibilities of automatic extraction of disambiguation rules and their combination with handwritten rules. We have attempted to reduce the workload, whilst maintaining a relatively high level of accuracy.

The problem of tagging can be perceived as a *classification problem*, *e.g.* [7]: tokens are *objects*, different tags (representing different descriptions) – *categories*, and a tagger *classifies* words into categories by assigning tags to them [8]. Thus, a tagger is a *classifier*, *i.e.* an implementation of a classification method. From the classification point of view, tagging of Polish is a very difficult task according to the number of categories – different tags, and the unfavorable proportion between the number of possible tags and the size of MIPIC. A possible way of tackling both problems

1. A distinction is sometimes made between morphosyntactic disambiguators, choosing appropriate descriptions, and taggers, performing the whole process including morphological analysis.

may be to decompose the overall classification problem into a collection of separable sub-problems for which the number of categories is smaller and the proportion is more favorable.

Thus, in order to solve the problem we need to:

- define the architecture allowing for combination of different types of rules,
- select a *Machine Learning* (ML) method of building a classifier (or classifiers),
- decompose the overall problem and define a way of combining classifiers, and
- optimize the system's parameters – learning and working.

The present paper is focused on the second and the third task, ML method selection and combining, as the others have already been discussed in previous papers [9–12]. However, a general review of the research performed so far is included as well.

2. Ideas

As handwritten rules are carefully tested on a corpus, their accuracy is usually very high and close to 100%. Thus, we have assumed that they can be initially applied as a filter deleting some contextually inappropriate tags, as was the case in [13].

A *positional tag* of IPIC [1] is a sequence of symbols describing various morphosyntactic features of a token. A tag is a record of *attributes* representing these features. Combinations of attribute values occur in MIPIC more frequently than whole tags. We can naturally decompose the general problem into sub-problems of *partial disambiguation* of subsets of attributes [11]. Obviously, the attributes are not mutually independent in that combinations of their values determine the structural relations of a token. However, a human performing manual disambiguation appears to do so in steps. He or she defines the POS of a token first, only then – the values of other attributes. We have assumed decomposition of the tagging process into *three phases of partial disambiguation* of *POS*, *number* together with *gender*, and *case*. According to experiments, the *person* category seems dependend on these four [11]. Moreover, from the twelve grammatical categories used in IPIC [1], the only ones independent of the four appeared to be:

- *non-past forms* of verbs in the present tense and the third person morphologically described as ambiguous in aspect, *e.g. razi* (*dazzles* or *offends*), *pozostaje* (*stays*, *remains*), *napotyka* (*encounters*), and
- *accentability* and *post-prepositionality* of personal pronouns in the third person, *e.g. on* (*he*), possessing four different combinations of values $\{$*accented*, *non-accented*$\} \times \{$*post-prepositional*, *non-post-prepositional*$\}$.

The values of these categories remain ambiguous to TaKIPI, *i.e.* more than one tag can be left for a token if the tags differ only in values of these categories. This simplifies the tagging problem as not all 1642 tags are distinguished from each other by TaKIPI. The lowest bound of the number of tags distinguished in MIPIC by TaKIPI according to the four attributes is 437, with additional simplification discussed in Section 3. However, the exact number is greater after adding person and other 'dependent' categories.

During each phase, tagging is based on automatically extracted rules. The proposed architecture of a tagger, called TaKIPI [11], is open for any types of classifiers

used in the subsequent phases. The phases' results are combined on the basis of probabilities of tags: each classifier updates the probabilities and some tags are eliminated at the end of each phase according to their probabilities. Many tags may remain for a token, but with different probabilities.

TaKIPI only uses the C4.5 [14] algorithm as an ML method and C4.5 Decision Trees as classifiers. However, TaKIPI initially achieved only 92.55% of accuracy. Many proposed ML algorithms have claimed better properties than C4.5. The goal of this work has been to analyze the possibility of applying other algorithms of rule extraction to the problem of tagging in Polish. As none of them had appeared to be superior, we performed preliminary experiments on the possibilities created by combining classifiers.

In each phase of partial disambiguation, the tagger has to classify a token into a small set of categories. The largest is the set of the first phase, in which 32 *grammatical classes* of IPIC [1] are used for a finer grained description than that of traditional POSs. For each phase, we could implement the tagger as one classifier. However, not all grammatical classes are possible for each token. Moreover, after analyzing MIPIC, we have identified about 200 different *classes of ambiguity* [15] for the first phase. A class of ambiguity (CoA) is a set of tokens such that each token is ambiguous among the same values of some tag attributes, in the case of the first phase, *e.g.* a token can be ambiguous among the grammatical classes of { `adj`, `subst`} (adjective and noun, the mnemonics coming from IPIC) or {`adj`, `conj`, `pred`, `qub`, `subst`} (`qub` = *particle-adverb*, `conj` = conjunction, `pred` = predicative), in the second phase, *e.g.* {`sg`, `m1`, `m2`, `m3`} (singular number, but all possible male genders). CoAs are disjoint and all tokens from a CoA express similar morphosyntactic features. A natural solution is to construct a separate classifier for each CoA. Such classifier can exploit specific features of tokens belonging to the given CoA in its decisions.

3. Architecture of the TaKIPI tagger

The algorithm of TaKIPI is outlined below (see [11] for details):

1. Morphological analysis based on the *Morfeusz* morphological analyzer [16], recognition of abbreviations and preliminary division into sentences
   → each token is assigned a set of all possible tags.
2. Application of the set of handwritten rules
   → elimination of some contextually inappropriate tags.
3. Assigning initial probabilities to tags on the basis of the *unigram dictionary* (*i.e.* statistics ⟨*token*, *tag*⟩ collected from MIPIC).
4. Partial disambiguation – repeated for subsequent phases:
   (a) application of classifiers to tokens according to their CoA
   → probabilities of the tags are updated;
   (b) elimination of tags with the lowest probability.
5. Annotation of tokens in the output text by tags with the highest probability
   → all tags for a token are restored, some chosen as appropriate.
6. Final segmentation into sentences.

The final decision concerning sentence boundaries cannot be made in step 2 but must be postponed to step 6 due to the ambiguity of some abbreviations (a full stop can be a part of an abbreviation).

Handwritten rules express general necessary morphosyntactic constraints and eliminate all tags that do not fulfill them [10]. The language of the rules, called JOSKIPI, and the set of 24 rules used in TaKIPI are discussed in details in [10]. In summary, each rule consist of a *condition of application* and a *condition of elimination*. Both conditions may be applicable to all tokens and their tags in a sentence, but the condition of elimination is applied iteratively to all tags of a token being disambiguated, and only one tag of this token is visible to the condition during a single iteration (all tags of other tokens are visible), *e.g.* a rule eliminating two subsequent non-past forms of verb:

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

The condition of elimination (delete) deletes a tag of a non-past verb form (`fin`), only if the preceding token (`pos[-1]` = the set of grammatical classes from the tags of the first token to the left) is unambiguously `fin`. The `not` sub-condition expresses some exceptions found in MIPIC. In addition to simple operators of reading tag attributes and logical conjunctions, there are also simple means of defining iteration (search across a sentence), variables, and operators checking the potential morphosyntactic agreement of the selected attributes in JOSKIPI.

In step 3, probabilities for pairs not observed in the learning data but possible according to *Morfeusz* are calculated by smoothing based on Lidstone's law (inspired by [15]). Thus, probabilities of tags are associated with word forms. The method is efficient, but too simple. We hoped that further processing would compensate for its simplicity, but it did only to some extent.

In step 4, the processing is repeated for each of the phases defined in Section 1. During each phase, an ambiguous token belongs univocally to a CoA defined with respect to the attributes disambiguated in the given phase. Only tokens that are ambiguous with respect to these attributes are processed during the given phase. A classifier is constructed for each CoA (see Section 4). Generally, a token is processed by a classifier corresponding to its present CoA. (If there is no classifier constructed for a given class, see Section 4, several other classifiers can be applied). The probabilities of tags are updated. During each phase, tags are distinguished only on the basis of values of attributes disambiguated in that phase. A subset of tags of a token such that all of its members have identical values of attributes disambiguated in the given phase is called a *package* of tags. During each phase, the tagger chooses the best package according to the updated probabilities of tags and eliminates all packages except the best one. At the end of step 4, the probabilities of tags in a token are normalized to 1.

The architecture allows many types of classifiers, the only constraint being that a classifier should update the probabilities of tags. There are no restrictions on the ways of calculating probabilities. In the first version of TaKIPI [11] only classifiers based on the C4.5 algorithm of Induction of Decision Trees (DT) [14] were used. A DT classifier was converted to a classifier returning the probability of a *positive decision*, selection of values for disambiguated attributes, and a *negative decision*, a smoothed non-zero probability of other possible selections, in a way similar to that found in [15]. For each DT leaf, the probability of its decision is calculated on the basis of the number of examples attached to this leaf during the tree's construction. Each DT multiplies the probabilities of tags in a token by the probability of a decision.

In step 5, all morphologically possible tags for the token are restored and the XML attribute `dissamb` is set to 1 for the tags with the highest final probability. There can be more than one tag set appropriate for a token as probabilities of several tags may happen to be identical and TaKIPI does not distinguish certain types of tags (see Section 2). Besides the categories discussed in Section 2, TaKIPI does not disambiguate the `subst` (noun) and `ger` (gerund) grammatical classes. The difference between `subst` and `ger` seems to be of more semantic than syntactic nature in MIPIC and we have decided to keep it ambiguous if it is not solved by disambiguation of grammatical categories (*i.e.* gender, number, case). As a result of the simplifications, TaKIPI leaves an average of 1.03 tags per token (2.87 before disambiguation).

There are several parameters in the algorithm, including the number of additional internal iterations in step 4 and the cut-off level for tags' probabilities used after each iteration [9, 11]. Optimization of these parameters will be discussed together with optimization of the learning parameters in Section 4.

4. Learning and optimization of parameters

According to the assumed division of the tagging process into three phases and to the natural disjoint division of tokens into CoAs, we have decided to construct a separate classifier for CoA. Such classifiers process only tokens of the corresponding CoA and are applied during the appropriate phase. Each classifier was constructed on the basis of ML and examples collected from MIPIC. It appeared that not all CoAs included enough instances (*i.e.* corresponding tokens in MIPIC) to construct classifiers for them, *e.g.* C4.5 will not build a DT on the basis of only several learning examples. We selected some CoAs, called *supported classes*, that were sufficiently supported by examples (a heuristic criterion of having the size of about 100 examples) or were necessary as there was no other *similar* CoA. The notion of CoA similarity should be based on a careful linguistic analysis, which would be very laborious. Instead, following [15], we simplified CoA similarity to the inclusion of one CoA's description in the description of another CoA, *e.g.* {`adj fin`} and {`adj subst`} being both similar to {`adj fin subst`}. The *most similar* CoA is the *smallest similar* CoA (as there can be several). From the linguistic point of view, such similarity is very weak, but at least it expresses the similarity of decisions made by hypothetical classifiers of both CoAs.

DTs are constructed (as classifiers) only for the supported CoAs. The construction of a DT for a particular CoA enables accurate choice of components of learning

examples for the DT. Tokens from non-supported CoAs are processed by a number of DTs built for similar CoAs, the selection of DTs being controlled by a parameter, as explained below.

Sets of *learning examples* are generated only for supported classes, but according to CoA similarity, each token belonging to one of the non-supported classes is a source of learning examples added to the sets of examples of a number of similar CoAs. A *learning example* is a sequence of values produced by a sequence of *operators* expressed in JOSKIPI, the same language as for the handwritten rules. The operators can be all predefined, *e.g.* `pos[-2]`, `cas[2]` – reading, respectively, a set of grammatical classes and a set of case values from the tags of specified tokens, or *compound*, defined as JOSKIPI expressions testing morphosyntactic constraints on the context, *e.g.* an operator checking whether there is an adjectival token agreeing in number, gender and case, and located to the right:

```
!AdjPRight
or( and( inter(pos[1],{adj,pas,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}))))
```

The first *and* verifies the existence of an adjectival token in the first position to the right. If it is not found, we look at the sentence's end (`rlook`) searching for a token expressing agreement (`agrpp`) and separated only by tokens of certain grammatical classes (`only`). `$Adj` and `$Q` are variables.

During learning, values of all operators are computed for ambiguous tokens and stored as vectors in sets corresponding to CoAs. The sets are then used in the construction of C4.5 DTs for CoAs. During tagging, our own implementation of C4.5 DT reads the operators' identifiers and requests the application of a certain operator in the context by the tagger. A decision is made in a DT node on the basis of the value returned by the applied operator.

A set of operators is specified for each DT. Its core consists of simple operators that read values of various tag attributes, *e.g.* grammatical class, case, number, aspect, *etc.* (the values being sets, as tokens can be ambiguous). However, compound operators can deliver more abstract information specific to the given linguistic problem to DTs, *e.g.* the `!AdjPRight` operator, presented above, is a part of the specification of learning examples for the {`fin`, `subst`} CoA: the presence of an agreed adjective is an important indication for choosing a noun's grammatical class. Compound operators guides the decision process performed by DTs (classifiers); the accuracy of compound operators does not need to be very high, as they are building blocks from which automatically extracted rules are constructed.

During tagging, DTs used in the second phase (or classifiers in general) are applied to partially disambiguated tokens. During learning, we have to create a similar situation, achieved by learning partial taggers for subsequences of phases till the 'full tagger'. Before learning examples are prepared for the phase k, a partial tagger for phase $k-1$ is applied and the appropriate attributes are disambiguated. Such

gradual learning appeared to be superior to an 'ideal' disambiguation based on manual disambiguation of MIPIC.

In total, TaKIPI utilizes 143 classifiers during three subsequent phases. Thus, classifiers of one phase depend during *training* (in description of learning examples) and *tagging* (in combinations of attribute values) on classifiers of the previous phase, which complicates analyzing the influence of a particular classifier on the final accuracy. The classifiers are rule-based but they must produce probabilities of decisions. They do so on the basis of statistics collected during learning.

The performed tests [10], with the total result of 92.55% accuracy for all tokens, and 84.75% for ambiguous ones, have demonstrated positive influence of the 24 handwritten rules, without which the accuracy for all tokens was 91.60%, and the presence of compound operators in DTs (91.75%). However, mutual relations between the rules and the compound operators are more complex: without both the accuracy was 91.43%. Moreover, recent tests with an extended set of 29 rules (including one eliminating several hundred thousands of tags as tested on MIPIC) have produced an increase in accuracy to 93.3%. However, a similar result of 93.44% has been achieved with the 'old' set of rules by optimization of the learning and tagging parameters with the application of Genetic Algorithms (GA). With the extended set of rules, the optimization did not lead to any progress and the final accuracy after optimization was about 93.4%. It seems that, when using compound operators in DTs, we can automatically extract tagging rules of accuracy similar to that of manually constructed rules.

In GA-based optimization of TaKIPI [9], a chromosome encoded the values of parameters of TaKIPI architecture and the learning process. A specimen was an instance of the learning process and a complete tagger. The fitness functions of a specimen was equal to the tagger's accuracy measured in tests on MIPIC with the tagger parameters and learning parameters set to the values from the chromosome. The *Cut-off Level*, *CoA Similarity Level for learning and tagging* and *Pruning Confidence Levels* for DT appeared to be important parameters. The *Cut-off Level* defines the threshold of probability for removing tags after a phase of tagging. The *CoA Similarity Level for tagging*, in [9] referred to as the inheritance level, defines how many DTs are used in processing one token. The 0 value means that only the DT of the CoA of a given token is applied if this CoA is supported; otherwise DTs of the most similar CoAs are used. Each value greater than 0 increases the number of DTs used; it is correlated with the distance in similarity of CoAs. Similarly, the *Similarity Level for learning* defines in how many CoAs (*i.e.* appropriate sets of learning examples) a given learning example generated for a token will be included. For values greater than 0, a given learning example will be added to several sets corresponding to similar CoAs. During optimization, the Similarity Levels were set to the values of 0 or 1 for different phases. Lastly, the *Pruning Confidence Level* is a parameter of the C4.5 algorithm determining the depth of DT pruning [8, 14]. During optimization, we assigned a separate Pruning Confidence Level to each of the 141 DTs.

Full GA optimization of TaKIPI took about 3 days, but it increased the accuracy to 93.44% for all tokens (86.3% for ambiguous tokens). We repeated it several times, achieving almost identical results (or differences without statistical significance).

The repeated results of optimization demonstrate that the maximum accuracy of the present TaKIPI architecture had been reached. One of the possible modifications is to change the type of classifiers used; in the rest of the paper we will investigate how a simple replacement of the C4.5 algorithm with another ML algorithm may influence the tagger's accuracy and what would be the results of combining various classifiers.

5. Selected single classifiers

The first implementation of the TaKIPI architecture incorporated only classifiers based on the C4.5 DT, but as the architecture was open to other types of classifiers, we looked for classifiers that could offer better accuracy of tagging than C4.5. A version of TaKIPI utilizing only classifiers of one type will be referred to as a *single classifier tagger*.

Learning sets produced by various CoAs differ in size, balance and noise. Therefore, the selection of a type of classifier for the whole tagger cannot be based on any specific feature of the learning sets. Actually, the classifier should learn well on different types of learning sets. For reasons given in Section 1 above, we have limited ourselves to rule-based classifiers and further focused on the types of classifiers reported in the literature to be competitive with C4.5. We have also considered the technical constraints of classifiers so that they are able to process the largest learning sets used in TaKIPI during training. Finally, two rule-based ML algorithms have been selected: RIPPER [17] and PART [18]. The *C4.5 rules* algorithm has been intentionally omitted, in an effort to distance ourselves further from the C4.5 idea.

However, as the probability of a decision is required from a classifier, we have also interested in (non-binary) classifiers combining directly rule extraction with probability estimation. The *Logistic Model Trees* algorithm (LMT) [19] fulfils these criteria. LMT is a decision tree with logistic models in its leaves, and logistic models define probability estimates directly.

RIPPER and PART had to be altered to return probabilistic estimates, too. It has been achieved by simply getting the number of positive (or negative) examples covered by a rule and dividing this number by the total number of examples in the learning set [12]. However, an additional assumption has been made that there always exist two bogus examples: one covered by all of the rules and the other not covered by any of the rules. It is an idea slightly similar to the smoothing used earlier for C4.5 trees[2]. We have to avoid returning zero probability; we cannot be sure that a decision is impossible and zero probability nullifies other decisions, for example made by the unigram classifier (as their probabilities are multiplied).

The training of the selected single classifiers was based on slightly modified implementations of PART, RIPPER and LMT from the WEKA system [20]. The WEKA output files were read and used in classification. We constructed our own implementations of the classifiers to be incorporated into the probabilistic TaKIPI architecture (written in C++).

2. In fact, the idea of smoothing as used in C4.5 was tested on PART and proved to result in slightly lower accuracy of tagging than the method described here. Therefore, the type of smoothing used in C4.5 was not used with PART or RIPPER.

All versions of single classifier taggers were tested using 10-fold cross validation on the MIPIC. Also the original C4.5-based architecture was tested in order to establish a reliable baseline in relation to the modified implementation and the applied parameters of learning and tagging.

The results are shown in Table 1. The tagging accuracy is presented in relation to all tokens (*all*) and ambiguous ones only (*amb*). Besides the average results calculated on 10 folds (*avg*), the minimum and maximum (*min* and *max*) across the folds is presented to give an impression of the variance of the results.

Table 1. Results of testing single classifiers [12]

	Phase 1 (≈ POS)					
	all			amb		
Classifier	**avg**	min	max	**avg**	min	max
C4.5	**98.72**	98.65	98.76	**91.12**	90.66	91.39
LMT	**98.78**	98.71	98.84	**91.54**	91.01	91.84
RIPPER	**98.73**	98.63	98.79	**91.17**	90.48	91.46
PART	**98.74**	98.63	98.78	**91.23**	90.51	91.46
	Phase 2 (L1 + number. gender)					
	all			amb		
Classifier	**avg**	min	max	**avg**	min	max
C4.5	**96.09**	95.88	96.26	**87.92**	87.33	88.36
LMT (& C4.5)	**95.02**	94.70	95.24	**84.62**	83.71	85.17
RIPPER	**95.82**	95.61	95.96	**87.08**	86.50	87.55
PART	**96.11**	96.00	96.24	**88.01**	87.70	88.30
	Phase 3 (L2 + case)					
	all			amb		
Classifier	**avg**	min	max	**avg**	min	max
C4.5	**93.11**	92.94	93.40	**85.71**	85.40	86.23
LMT (& C4.5)	**92.05**	91.79	92.29	**83.52**	83.02	83.92
RIPPER	**92.45**	92.11	92.79	**84.34**	83.68	85.02
PART	**93.08**	92.87	93.34	**85.64**	85.26	86.16

The tagging errors made in the first phase, roughly corresponding to parts of speech, are the most important for practical applications. In relation to the first phase, the best single classifier tagger was LMT-based. The statistical significance of improvement was greater than 99.9% according to the *T-test* applied to the mean difference computed for the results of the 10-folds (*cf.* [21]). The significances of all other results in this paper were calculated in the same way.

Unfortunately, LMT was extremely slow in training (weeks of computer work). For some of the largest learning sets, training could not be completed due to technical limitations – the memory needed for a Java Virtual Machine cannot exceed 1.5 GB and this was insufficient. There were many CoAs in the second phase for which we were unsuccessful with training LMT; C4.5 was used for these classes in the LMT-based

tagger. As a result, we did not apply LMT for the third phase at all. Consequently, the results described in Table 1 as *LMT* are results achieved by a tagger in which LMT classifiers were applied to some CoAs, while C4.5 classifiers were used for others. Neverthless, preliminary experiments have shown that LMT classifiers perform very poorly in the third phase. Moreover, the LMT performance for the second phase (supplemented by C4.5) decreases and is exceeded by other classifiers ($> 99.9\%$ of significance in comparison to C4.5 alone). A test case can have attributes with values not seen and, therefore, not predicted on the basis of learning. LMT seems to be more sensitive to this problem than the other classifiers. LMT is a DT with logistic models in the leaves and if an attribute with an unseen value occurs in a leaf, an inaccurate estimate of probability is generated by the model[3]. This is probably the reason why LMT accuracy is lower for the second and third phases. Accidental combinations of attributes may appear after partial disambiguation during the first phase. However, LMT is still the best classifier for the first phase for which the conditions are stable between learning and testing.

The other classifiers are more suitable for the situation in which unseen values are encountered in data. Rules focus on values of attributes seen in the learning data and, at worst, the last most generic rule is applied, the one covering all examples that are not covered by the previous rules (both RIPPER and PART produce ordered sets of rules, where rules are applied one by one in a sequence). However, there is still a notion, that if the disambiguation during the previous phase is stronger, *i.e.* less tags are left for a token in a learning set, it is more difficult to train the classifiers for the next phase on the basis of these sets. PART seems to cope best with this problem and it seems to be the best classifier from the tested ones, as it outruns C4.5 during the first and the second phase and is only slightly worse during the third phase. However, the significance of the PART results is low, about 90% in comparison to C4.5. Moreover, C4.5 still yields the best result for all three phases, but on the same level as PART for the third phase – the better result of C4.5 is statistically insignificant. The overall good result of C4.5 has been obtained in inspite of its inferiority in the first phase, the most important one from the practical point of view.

RIPPER also seems to have better accuracy for the first phase in comparison to the C4.5 baseline, but without statistical significance. PART is slightly better when used alone. The authors of PART claim that their approach can be better than RIPPER because PART avoids what they refer to as *hasty generalization*, which is actually overpruning of RIPPER's rules. These results suggest that both classifiers can perform better than C4.5 for some CoAs (or even some cases).

We should bear in mind that no optimization of learning parameters has been performed. Adjusting the parameters may lead to improvements, so a tuned classifier other than C4.5 may be better for all phases when used as a single classifier. Some general tests were performed for the first phase. In C4.5, we changed the pruning level and in RIPPER – the number of optimization iterations (from 2 to 10). In LMT, there is heuristics that controls the number of optimizations by setting it to a value

3. If an attribute with an unseen value is used as a test in a decision node, we can forfeit the classification which probably has lesser impact than disrupting the model in a leaf.

computed only for the root. The heuristics is used by default; when it is turned off, the learning is much slower (because the computation of the best number of iterations is performed for every node) but the accuracy of classification can be improved. LMTs with the heuristics turned off were also tested. All the above mentioned changes to C4.5, RIPPER and LMT offered better overall results. This proves that optimization is applicable with good results, and possibly the best way would be actually tuning the parameters for each CoA separately (we optimized globally by setting the same parameters for all classifiers of all classes).

Verification of the results of disambiguation demonstrated that various classifiers perform differently for different CoAs. For a selected CoA, a tagger based on a single type of classifiers disambiguates some tokens correctly, while an other tagger, based on a different classifier, fails to disambiguate them properly. At the same time, this tagger can disambiguate other tokens better, even for the same selected CoA. When such situation occurs, we say that taggers (or classifiers) are *complementary*.

The fact that the application of possibly complementary classifiers can improve the ultimate accuracy is well known in literature. However, as we observed many times in the case of TaKIPI architecture, formulating predictions concerning the results of the process is always risky. It is worth emphasizing here that classifiers calculate the probability of a decision in relation to the learning data and other possible decisions, not in relation to the current context of a word. The computation of probability depends also on the structure of a classifier. Thus, a comparison in TaKIPI of probabilities produced by classifiers (even of the same type) for the same word is difficult. Therefore, the secondary goal of this work is to test the idea of simultaneous combination of different classifiers in a given architecture. We wanted to start with simple combination methods (*e.g.* in comparison to [22]) for which the results are easier to be analyzed in relation to the TaKIPI architecture[4], hoping that the collected results would help in defining the direction of further developments.

6. The multiclassifier approach

The idea of a combination of classifiers of the same type was already tested in a limited scope in [15, 23], where C4.5 trees were used in an ensemble for selected CoAs. However, we preferred to use different types of classifiers and our goal was to test simple ways of joining them.

In one of the most obvious approaches, which can be called *OneShot*, one has to simply select a classifier for a given CoA that expressed the highest accuracy for this class in tests. It seems quite natural that this approach leads to better results. Unfortunately, it is difficult obtain results comparable with other approaches and with single classifier taggers. Therefore, we chose first to test approaches that work without reference to previous tests. This means that they should be based only on probabilities returned by classifiers during the disambiguation process. All approaches were tested using the same 10-fold cross validation approach as single classifier taggers, so that the results are comparable. We used previously learned classifiers to speed up testing

4. It is worth mentioning here that training a complete set of classifiers for TaKIPI can take even several weeks of continuous computer work.

process. However, the tagging during the first phase is now different, as a combination of classifiers is used. So, new data is produced for the second and the same goes for the third phase. With more ample resources, one could try to train all single classifiers from new data, although this would be a lengthy process of several weeks. Fortunately, the usage of already learned classifiers appeared to be successful (at least for the preliminary tests).

Another natural way to use a number of single classifiers simultaneously is to apply them one by one. This corresponds to multiplying returned probabilities, so the order of the classifiers application is actually irrelevant. This also fits nicely the TaKIPI architecture, in which the probabilities calculated for the tags of a token according to the classifiers used in the subsequent phases are multiplied (starting with the unigram classifier). An approach in which probabilities returned by classifiers of different types (assigned to the same CoA) are *simply multiplied* is here called the *Bag* approach.

Another tested approach is a version of the *Winner Takes All* (*WTA*) approach. In WTA we apply all classifiers for a selected token, but use only the highest probability returned in updating the probabilities of tags. This approach attempts to mimic *OneShot* by assuming that the highest probability returned is the right one. Of course this assumption is not expected to be always right, mainly because it is difficult to compare probabilities returned by classifiers (see Section 3 above).

Other two simple approaches are attempts to produce mutual agreement between the classifiers applied to a token. This is done by either computing the *arithmetic average*, hereinafter referred to as *AVN*, or *geometric average*, *AVG*. The average of positive and negative decisions is computed from probabilities returned by all classifiers for a given token and these values are used for updating the tags' probabilities.

The results of 10-fold testing of multiclassifier taggers based on algorithms described above are shown in Table 2 (identical in form to Table 1).

Comparing the results with the results of single classifier taggers, we can clearly see that any multiclassifier tagger is better in the first phase than any of the single classifier ones (with significance $> 99.9\%$). This suggests that even simple ways of combining classifiers in the TaKIPI architecture increase its accuracy and is worth further effort to developing more sophisticated algorithms. However, in the next phases the accuracy of WTA and AVN taggers is lower than that of single classifier taggers (with significance $> 99.9\%$). These approaches may lack the ability to cope with the differences in processing the input data by different classifiers. The possibly mistaken decisions of individual classifiers have great impact on the whole combination. The influence of the most extreme decisions must be more balanced. Moreover, as argued in Section 3, different classifiers have a different sensitivity to the differences between learning and tagging. Especially WTA proves that taking into account only the highest probability is no good, most notably because of differences among the ways of producing the estimates. Additionally, WTA tends to ignore classifiers other than the selected one. AVN is only slightly better (significantly in comparison to WTA), as it attempts to produce common data from the returned probabilities for all of the classifiers used, but the calculation of a simple average seems to exert too much strength on the extreme values of probability.

Table 2. Results of testing multiclassifier approaches [12]

	Phase 1 (≈ POS)					
	all			amb		
Classifier	**avg**	min	max	**avg**	min	max
Bag	**98.83**	98.72	98.89	**91.86**	91.08	92.15
WTA	**98.83**	98.73	98.88	**91.86**	91.18	92.24
AVN	**98.83**	98.73	98.88	**91.88**	91.18	92.16
AVG	**98.88**	98.78	98.92	**92.19**	91.54	92.41
	Phase 2 (L1 + number. gender)					
	all			amb		
Classifier	**avg**	min	max	**avg**	min	max
Bag	**96.27**	96.12	96.46	**88.48**	88.11	88.98
WTA	**94.79**	94.54	95.00	**83.90**	83.23	84.43
AVN	**95.64**	95.46	95.80	**86.54**	86.11	86.97
AVG	**96.39**	96.25	96.54	**88.86**	88.46	89.18
	Phase 3 (L2 + case)					
	all			amb		
Classifier	**avg**	min	max	**avg**	min	max
Bag	**93.40**	93.25	93.69	**86.30**	86.03	86.82
WTA	**91.48**	91.19	91.85	**82.32**	81.78	83.06
AVN	**92.35**	92.04	92.74	**84.13**	83.54	84.90
AVG	**93.53**	93.36	93.76	**86.57**	86.28	86.98

The implicit assumption is that the probabilities returned by different classifiers are directly comparable, while we know that it is not always true.

The Bag approach appears to be quite successful (with significance $> 99.9\%$ in all three phases). In this approach, every classifier simply does what it is supposed to do, basing on its own learning examples and its own estimation, and does not interfere directly with other classifiers. Combining classifiers follows directly the procedure used while combining subsequent phases in the TaKIPI architecture. The Bag tagger offers better accuracy than any of the single classifier taggers for all phases. Thus, very good agreement is produced in a rather indirect fashion.

The AVG approach appears to be even better. The use of AVG is possible, as the probabilities of a token's tags are normalized after each phase. In general, AVG follows the Bag approach and fits the TaKIPI architecture, as computing the geometric average is after all based on multiplication of the values. Obviously, AVG has no probabilistic sense, but as it sometimes yields better results [2], it is used as a kind of heuristics. Unlike Bag, AVG tries to directly produce agreement between classifiers. AVG is the best of the tested approaches. It is also significantly better than Bag: $> 99.9\%$ of significance for all three phases. Moreover, the obtained result of 93.53% accuracy on all tags for all phases (86.57% for ambiguous ones) is currently the best result obtained for Polish.

Having established that AVG offers the best results, we returned for a while to the OneShot idea. We could then consider merely replacing AVG with classifier had it been proven to be better than AVG for a given CoA. However, in a test for the first phase, AVG produced average results comparable with the best single classifier tagger for a given CoA. Only sometimes the Bag approach gave considerably better results for selected CoA. So, we could still consider replacing AVG with Bag, but in the end this approach was abandoned in tests, as the test should have been changed to produce trustworthy and comparable values without seeing any of the testing data in the learning phase before the ultimate test. However, it is our intuition that replacing AVG with Bag for some CoAs, on the basis of tests, should yield slightly better results for real-world applications than using AVG alone at all times.

7. Conclusions

Experiments with handwritten rules and automatically extracted rules based on JOSKIPI and compound operators have demonstrated that these two sources of knowledge can be combined. In comparison to statistical approaches to tagging, TaKIPI lacks information concerning particular tokens in DTs. The unigram dictionary is too simple a method for expressing lexical information.

Experiments with single-classifier and multi-classifier taggers have demonstrated the possibilities of extending the TaKIPI architecture. Different CoAs present different classification problems. Thus, contrary to our preliminary expectations, a simple replacement of one rule-based classifier with a 'better' one will not solve the problem. The initially applied, well-known C4.5 algorithm appeared to be more successful than we had expected when used in the overall problem of tagging at all three levels. Some bias in supporting C4.5 may be due to the TaKIPI architecture being open and general, but it had been developed and modified on the basis of experienced gained in experiments with the application of C4.5 as the only type of classifier. However, even a simple modification of the algorithms' parameters has shown that there are possibilities of improvement to be achieved only at the cost of extreme complexity of optimization.

A comparison of results achieved by various classifiers has shown that their performance can be very different in relation to various CoAs and various tagging phases, where the learning conditions (*i.e.* the degree of match between learning and tagging) are changeable. The differences in the classifiers' accuracy have been explored successfully even with application of simple techniques of classifier combination. For the best technique of AVG, the tagging error calculated for ambiguous tokens has been reduced by almost one percent, which is a significant number. The multi-classifier approach should be further extended to different types of classifiers, *e.g.* memory-based ones, but especially by application of more sophisticated techniques of classifier ensembles. The achieved results support efforts being put into the very costly process of learning of partial multi-classifier taggers in order to reduce the difference between the environments of learning and tagging. Efforts should also be made to unify the calculation of decision probabilities between different classifiers.

The final accuracy of the modified TaKIPI, *viz.* 93.53% for all tags and 86.57% for ambiguous ones, is still much below the results obtained for Czech [13] (95.16%

for all tokens) and English [2] (more than 97%). This means that TaKIPI makes an average of one mistake per sentence, but the error in grammatical classes is much lower. The accuracy of TaKIPI may be insufficient for precise, deep syntactic analysis, *e.g.* for Machine Translation. However, TaKIPI is very efficient (4000 words per second on a PC of 512 MB RAM, 2.41 GHz.) and may be very useful in tasks requiring shallow processing of large amounts of text, *e.g.* in Information Extraction [21]. TaKIPI has been successfully applied to identify base forms for the purposes of constructing a semantic similarity function for Polish nouns [24]. TaKIPI has also been used in automatic disambiguation of the present version of IPIC [1].

Acknowledgements

This work was financed by the Ministry of Education and Science project No T11C 018 29.

References

[1] Przepiórkowski A 2004 *The IPI PAN Corpus. Preliminary Version*, Institute of Computer Science PAS
[2] Manning Ch. D and Schütze H 1999 *Foundations of Statistical Natural Language Processing*, The MIT Press
[3] Przepiórkowski A 2006 *The Potential of the IPI PAN Corpus. Poznań Studies in Contemporary Linguistics* **41** 31
[4] Dębowski Ł 2004 *Proc. Int. Conf. on Intelligent Information Processing and Web Mining*, Zakopane, Poland (Kłopotek M A, Wierzchoń S T and Trojanowski K, Eds), Springer Verlag, pp. 409–413
[5] Voutilainen A 1997 *EngCG tagger, Version 2* (Bronsted T and Lytje I, Eds), Sprog og Multimedier, Aalborg Universitetsforlag
[6] Oliva K 2003 *Contributions 4th Eur. Conf. on Formal Description of Slavic Languages* (Kosta P *et al.*, Eds), Peter Lang, pp. 299–314
[7] Dębowski Ł 2001 *Internal Report IPI PAN*, Instytut Podstaw Informatyki PAN, **934** www.ipipan.waw.pl/staff/l.debowski/raporty/kropka934.pdf
[8] Mitchell T M 1997 *Machine Learning*, WCB/McGraw-Hill
[9] Godlewski G and Piasecki M 2006 *Proc. Artificial Intelligence Studies* (Kłopotek M and Tchórzewski J, Eds), Publishing House of University of Podlasie, pp. 157–164
[10] Piasecki M 2006 *Text, Speech Dialogue. Proc. 9th Int. Conf.*, Brno, Czech Republic (Sojka P, Kopeček I and Pala K, Eds), Springer Verlag, **LNAI4188**, pp. 205–212
[11] Piasecki M and Godlewski G 2006 *Text, Speech Dialogue. Proc. 9th Int. Conf.*, Brno, Czech Republic (Sojka P, Kopeček I and Pala K, Eds), Springer Verlag, **LNAI4188**, pp. 213–220
[12] Piasecki M and Wardyński A 2006 *Proc. 1st Int. Symposium Advances in Artificial Intelligence and Applications*, Wisła, Poland, pp. 169–178
[13] Hajič J, Krbec P, Květoň P, Oliva K, Petkevič V 2001 *Proc. 39th Annual Meeting of ACL*, Morgan Kaufmann Publishers, pp. 260–267
[14] Quinlan J R 1993 *C4.5: Programs for Machine Learning*, Morgan Kaufmann, San Mateo
[15] Márquez L 1999 *Part-of-speech Tagging: A Machine Learning Approach based on Decision Trees*, PhD Thesis, Universitat Politècnica de Catalunya
[16] Woliński M 2006 *Proc. Int. Conf. Intelligent Information Processing and Web Mining*, Ustroń, Poland (Kłopotek M A, Wierzchoń S T and Trojanowski K, Eds), Springer Verlag, pp. 511–520
[17] Cohen W W 1995 *Machine Learning: Proc. 12th Int. Conf.*, Lake Tahoe, California, Morgan Kaufmann, pp. 115–123
[18] Frank E and Witten I H 1998 *Proc. 15th Int. Conf. on Machine Learning*, Morgan Kaufmann, pp. 144–151
[19] Landwehr N, Hall M and Frank E 2005 *Machine Learning* **59** (1–2) 161

[20] Witten I H and Frank E 2005 *Data Mining: Practical Machine Learning Tools and Techniques*, 2nd Edition, Morgan Kaufmann, San Francisco

[21] Konchady M 2006 *Text Mining Application Programming*, Charles River Media

[22] Kuncheva L 2004 *Combining Pattern Classifiers: Methods and Algorithms*, John Wiley & Sons, New Jersey

[23] Márquez L, Rodríguez H, Carmona J and Montolio J 1999 *Proc. 1999 Joint SIGDAT Conf. on Empirical Methods in Natural Language Processing and Very Large Corpora*, Maryland, USA, pp. 53–62

[24] Piasecki M 2006 *Proc. Multimedia and Network Information Systems* (Zgrzywa A, Ed.), Wyd. PWr., pp. 99–107