# CombiTagger: A System for Developing Combined Taggers

**Verena Henrich** and **Timo Reuter**
Department of Computer Science
UAS Darmstadt
Germany
{verenah08,timo08}@ru.is

**Hrafn Loftsson**
School of Computer Science
Reykjavik University
Iceland
hrafn@ru.is

## Abstract

The main task of part-of-speech (PoS) tagging is to assign the appropriate morphosyntactic category to each word in a sentence. A combination of different PoS taggers usually results in higher tagging accuracy than obtained by the use of only a single tagger. We present a new language and tagset independent system, CombiTagger, which combines automatically the output of several taggers. The system, which is open source, provides algorithms for simple and weighted voting, but it is extensible so that other combination algorithms can be added easily. We demonstrate the functionality of CombiTagger by using it to develop and evaluate combined taggers for Icelandic. The most accurate individual tagger obtains an accuracy of 91.83%. CombiTagger achieves 93.09%-93.41% accuracy by combining the output of five or six taggers using simple and weighted voting.

## Introduction

PoS tagging is the task of labelling words with the appropriate word class and morphological features. The string used as a label is called a *tag*, the set of labelling strings is called a *tagset*, and a program which performs tagging is called a *tagger*. Since a word can have several PoS tags, the main function of a tagger is to remove ambiguity. Tagging text is a useful preprocessing step in many natural language processing applications, i.e. in grammar checking, parsing, information extraction, and machine translation.

Tagging accuracy is usually measured as the number of correctly tagged tokens (words) divided by the total number of tokens. The accuracy of a particular text in a given language can usually be increased by combining taggers which are based on different tagging methods (see section "Combined Taggers"). In most cases, each combined tagger has been written from scratch, i.e. each developer has written the necessary program code to build the combined tagger. This is unfortunate because, generally, it entails the reproduction of code already written.

To tackle this problem, we introduce *CombiTagger*[1], a language and tagset independent system for developing and evaluating combined taggers. The system provides algorithms for simple and weighted voting, but it is extensible so that other combination algorithms can be added easily.

We demonstrate the functionality of CombiTagger by using it to develop and evaluate combined taggers for tagging Icelandic. We use the Icelandic Frequency Dictionary (IFD) corpus (Pind, Magnússon, and Briem 1991) as a gold standard. The most accurate individual tagger yields an accuracy of 91.83%. By combining the output of five or six taggers using simple and weighted voting, CombiTagger achieves 93.09%-93.41% accuracy.

The rest of this paper is organized as follows. First, we describe our motivation for developing CombiTagger. Second, we briefly describe the individual taggers used when demonstrating the system. Third, we elaborate on combined taggers and combination algorithms. Fourth, we describe the design of CombiTagger, and, fifth, we demonstrate the functionality of the system using several test cases. Lastly, we conclude with a summary.

## Motivation

Our motivation for the development of CombiTagger is twofold. First, to provide an open source utility for all researchers intending to develop a combined tagger for a given language. As discussed in the introduction, researchers developing combined taggers have usually reproduced functionality already developed by others. Even basic combination algorithms like simple voting have been reimplemented many times by different research groups. We maintain that it is especially important to develop combined taggers for other languages than English, for example, morphologically complex languages. The reason is that tagging accuracy obtained by individual taggers for morphologically complex languages is significantly lower than the accuracy obtained for English.

It has been shown that the best performing individual taggers have achieved around and above 97% accuracy on English text (Brill 1995; Daelemans et al. 1996; Ratnaparkhi 1996; Brants 2000; Toutanova et al. 2003; Shen, Satta, and Joshi 2007). In contrast, the state-of-the-art tagging accuracy obtained for many morphologically complex languages (using a large tagset) is well below the 97% level, e.g. about 89% for Slovene (Džeroski, Erjavec, and

Copyright © 2009, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

[1]CombiTagger is an open source system which can be obtained from `http://nlp.ru.is/combitagger`.

Zavrel 2000), about 92% for Icelandic (Dredze and Wallenberg 2008), and about 94% for Czech (Hajič and Kuboň 2003).

The second motivation for the development of CombiTagger is that we need a tool which can locate error candidates in a PoS tagged corpus (as discussed in section "Combined Taggers").

## Individual Taggers Used

Various taggers have been developed based on different methods or models. We use the output from the following individual taggers to test the functionality of CombiTagger: *fnTBL* (Ngai and Florian 2001), *MXP* (Ratnaparkhi 1996), *MBT* (Daelemans et al. 1996), *TnT* (Brants 2000), *TreeTagger* (Schmid 1994), and *IceTagger* (Loftsson 2008). The first five taggers are data-driven (i.e. they learn from pretagged corpora), but the last one is a linguistic rule-based tagger.

The fnTBL tagger is a fast implementation (in C and Perl) of transformation-based error-driven learning (TBL) (Brill 1995). In TBL the training phase consists of, first, assigning each word its most likely tag without regard to context, and, second, learning a set of ordered rules which transform a tag X to a tag Y, with regard to context. New text is then tagged by applying the rules in the correct order.

The MXP tagger (implemented in Java) uses a binary feature representation to model tagging decisions, where each feature encodes any information that can be used to predict the tag for a particular word. The goal of the model is to maximize the entropy of a distribution, subject to certain feature constraints.

A memory-based model is used in the MBT tagger (implemented in C++). During training, a feature representation of an instance (word and its context) along with its correct tag (target class) is simply stored in memory. New instances are then tagged by similarity-based reasoning from these stored examples.

The TnT tagger (a very fast C implementation) uses a second order (trigram) probabilistic Hidden Markov Model (HMM). The probabilities of the model are estimated from a training corpus using maximum likelihood estimation. New assignments of PoS to words is found by optimizing the product of lexical probabilities ($p(w_i|t_j)$) and contextual probabilities ($p(t_i|t_{i-1}, t_{i-2})$) (where $w_i$ and $t_i$ are the $i^{th}$ word and tag, respectively).

TreeTagger is a probabilistic tagger (implemented in C) similar to a tagger based on an HMM. The main difference is that TreeTagger estimates contextual probabilities with a binary decision tree whereas an HMM tagger (like TnT) uses maximum likelihood estimation.

IceTagger (implemented in Java) is a linguistic rule-based tagger (the rules are hand-written) developed for tagging Icelandic text. It uses local (a window of 5 words) elimination rules for the initial disambiguation of tags. Thereafter, various heuristics are used to force feature agreement between words, effectively eliminating more tags. At the end, for a word not fully disambiguated, the default rule is to select the most frequent tag for the word.

## Combined Taggers

A combined tagger is built using the output of two or more individual taggers. It has been shown, for various languages, that a combined tagger usually obtains higher accuracy than the application of just a single tagger (van Halteren, Zavrel, and Daelemans 2001; Sjöbergh 2003; Kuba, Felföldi, and Kocsor 2005; Loftsson 2006). The reason is that different taggers tend to produce different (complementary) errors and the differences can be exploited to yield better results. When building combined taggers it is thus important to use taggers based on different methods.

Combined taggers are useful in many ways, for example when building tagged corpora or detecting errors in them. In the former task, a corpus is usually tagged with an automatic method and hand-corrected by humans afterwards. In order to minimize the hand-correction, it is thus important to tag the text with a high accuracy tagger, like a combined tagger.

In the latter task, a combined tagger can be used to point to possible error candidates in a tagged corpus. If a tag selected by the combined tagger does not agree with the corresponding corpus tag (the gold standard tag) then it may indicate an error in the corpus.

Various combination algorithms have been developed (see van Halteren, Zavrel, and Daelemans (2001) for a good overview). Here, we briefly review the two methods already implemented in CombiTagger: simple voting and weighted voting. In simple voting, equal weight is given to all taggers when voting for a tag. The votes from all taggers are summed up and the tag with the highest number of votes is selected as the output of the combined tagger. In the case of a tie, the tag proposed by the most accurate tagger(s) can be selected.

In weighted voting, more weight is given to taggers that have shown high accuracy, e.g. a tagger known to produce high overall accuracy gets more weight when voting. Otherwise, the voting mechanism works similarly as in simple voting.

## CombiTagger

CombiTagger is implemented in Java using the SWT toolkit[2]. The main purpose of the program is to read data files generated by individual taggers and use them to develop a combined tagger according to a specified algorithm. Note that CombiTagger supports any tagger, because it uses their output files but not the taggers themselves. Figure 1 shows an overview of CombiTagger's functionality, which will be explained in more detail below.

The graphical user interface consists of tabs to lead the user through the process of collecting information about the combined tagging approach. In the first tab, "Data Input", the user specifies the location of the output files already generated by the individual taggers. At least two tagger output files need to be specified and it is assumed that each line in a tagger output file contains a word and its corresponding tag, separated by a space or a tab. Figure 2 shows a screenshot after having added five tagger output files.

[2] `http://www.eclipse.org/swt/`

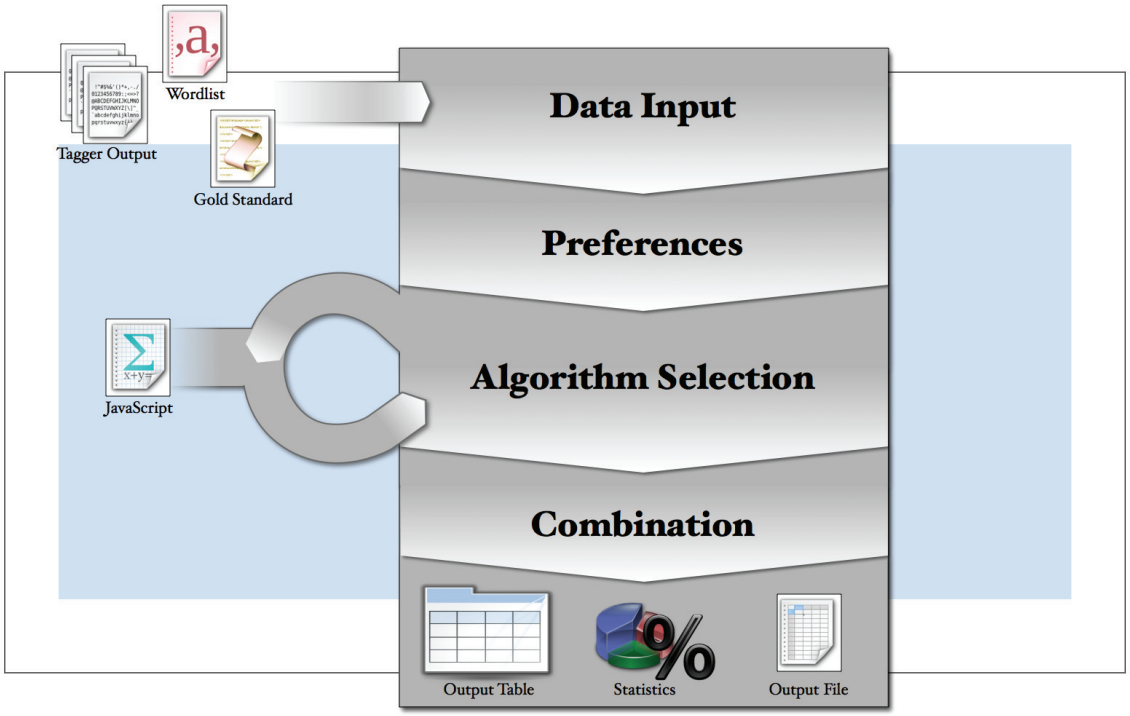



Figure 1: Overview of CombiTagger

The words of the input text can be provided by a separate wordlist file – containing one word per line. This option can be used if, for example, the words themselves do not appear in the output of the individual taggers. If no additional input file is provided, the program uses the words at the beginning of each line in the first specified tagger output file. A gold standard (i.e. a file containing correct PoS tagging) can also be provided. This file should be in the same format as the tagger output files described above.

In the second tab, "Preferences", the behavior of the program can be adjusted. First, the user can specify a file containing all possible tags in the specific tagset. By explicitly specifying the tagset, CombiTagger is not dependent on the "Word Space Tag" format. Instead, CombiTagger uses the tagset information to search for tags in each line matching one of the tags in the given tagset. The Penn Treebank tagset (Santorini 1990) is provided with the program but other tagsets can be added. The second option in this tab is the selection of the output behavior. It is either possible to write the output to a file or to a table (described in the paragraph below about the "Result" tab).

In the third tab, "Algorithm", the combination algorithm is specified. Every algorithm is implemented in *JavaScript*. Two scripts, for simple and weighted voting, are already provided. In both these scripts, the resolving of ties depends on the exact order of the tagger output files. For example, if there is a voting tie between two tagger groups $A$ and $B$ then the tag proposed by group $A$ is selected if one of its taggers output has been loaded into CombiTagger before some output from group $B$.

Other user defined scripts can be added easily. The JavaScript files are divided into two functions:

1. `createAlgorithmSpecificGUI()`: used to extend the graphical user interface for giving information needed by the algorithm (e.g. the weight for each tagger output).
2. `runCombinedTaggingAlgorithm()`: the implementation of the algorithm itself.

CombiTagger stores the output of the different taggers in the two-dimensional Java string array `tagArray` and it requires the result of the combination algorithm in the one-dimensional string array `resultingTags`. With the help of a JavaScript engine, these objects (`tagArray` and `resultingTags`) can be accessed in the JavaScripts. Due to this functionality, the choice of a combination algorithm is very flexible.

In the fourth tab, "Result", the combination algorithm can be started with the specified preferences. When the algorithm terminates, the tab displays the settings and shows various statistical information (absolute and relative values) as: in how many cases i) do all the taggers agree, ii) do all the taggers except one agree, iii) do all the taggers agree with the gold standard, iv) does the combined tagger agree with the gold standard (and more).

If the option to create a table is chosen (in the "Preferences" tab), it appears in a new tab, "Output Table". An example output table is shown in Figure 3.

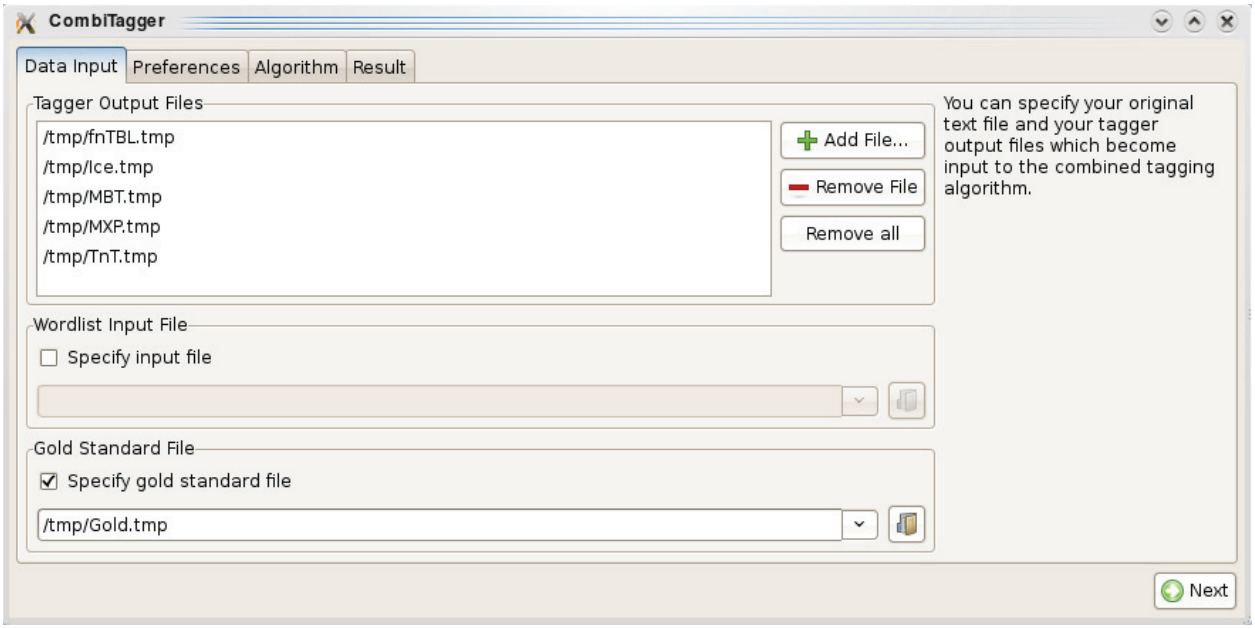
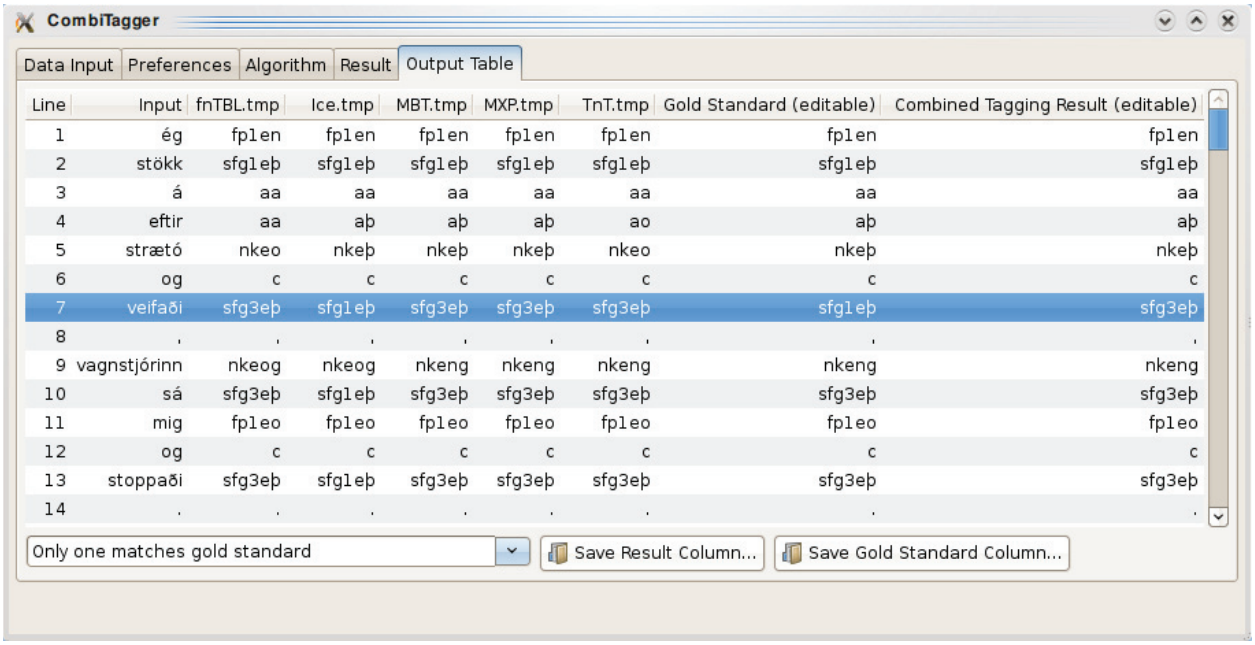



Figure 2: CombiTagger start screen. Five different tagger output files have been added as input data and a gold standard file has been specified.




| Line | Input | fnTBL.tmp | Ice.tmp | MBT.tmp | MXP.tmp | TnT.tmp | Gold Standard (editable) | Combined Tagging Result (editable) |
|---|---|---|---|---|---|---|---|---|
| 1 | ég | fp1en | fp1en | fp1en | fp1en | fp1en | fp1en | fp1en |
| 2 | stökk | sfg1eþ | sfg1eþ | sfg1eþ | sfg1eþ | sfg1eþ | sfg1eþ | sfg1eþ |
| 3 | á | aa | aa | aa | aa | aa | aa | aa |
| 4 | eftir | aa | aþ | aþ | aþ | ao | aþ | aþ |
| 5 | stræto | nkeo | nkeþ | nkeþ | nkeþ | nkeo | nkeþ | nkeþ |
| 6 | og | c | c | c | c | c | c | c |
| 7 | veifaði | sfg3eþ | sfg1eþ | sfg3eþ | sfg3eþ | sfg3eþ | sfg1eþ | sfg3eþ |
| 8 | , | , | , | , | , | , | , | , |
| 9 | vagnstjórinn | nkeog | nkeog | nkeng | nkeng | nkeng | nkeng | nkeng |
| 10 | sá | sfg3eþ | sfg1eþ | sfg3eþ | sfg3eþ | sfg3eþ | sfg3eþ | sfg3eþ |
| 11 | mig | fpleo | fpleo | fpleo | fpleo | fpleo | fpleo | fpleo |
| 12 | og | c | c | c | c | c | c | c |
| 13 | stoppaði | sfg3eþ | sfg1eþ | sfg3eþ | sfg3eþ | sfg3eþ | sfg3eþ | sfg3eþ |
| 14 | . | . | . | . | . | . | . | . |

Figure 3: Example of an output table using five different taggers and a gold standard. The second column contains the words (tokens) and columns 3-6 contain the tags proposed by the five taggers, respectively. A highlight function has been used to show those rows where there is only one match with the gold standard.

| No. | Tagger | Accuracy (%) |
|---|---|---|
| 1. | fnTBL* | 90.15 |
| 2. | Ice* | 91.83 |
| 3. | MBT | 89.28 |
| 4. | MXP | 89.03 |
| 5. | TnT* | 91.25 |
| 6. | TreeTagger | 89.30 |

Table 1: The average tagging accuracy of the individual taggers

In this table, it is possible to highlight rows that match the different statistical aspects described above. Furthermore, the user has the possibility to edit the result column of the combined tagging as well as the gold standard column. The changes can be saved to a file. This can, for example, be used to produce a new gold standard.

## Test Cases

PoS taggers for Icelandic have been evaluated by applying 10-fold cross-validation on the IFD corpus (Helgadóttir 2005; Loftsson 2006; Dredze and Wallenberg 2008). In our experiments described below, we follow Loftsson (2006) by using the output of individual taggers for the first nine test files and present accuracy numbers as averages from these nine runs. To test the functionality of CombiTagger and the two provided combination algorithms, we used CombiTagger for developing and evaluating combined taggers for Icelandic. We present the combined taggers in five test cases below.

As input to CombiTagger, we used the output of the six individual taggers: fnTBL, IceTagger, MBT, MXP, TnT, and TreeTagger (described in section "Individual Taggers Used"). We used enhanced versions of the taggers fnTBL, TnT, and IceTagger – called fnTBL*, TnT* and Ice*, respectively (Loftsson 2006). Table 1 shows the average tagging accuracy of the individual taggers when tagging the first nine test files.

In the first test case, we used the simple voting algorithm of CombiTagger. We loaded the output files of the first five taggers listed in Table 1 in alphabetical order (this effectively means that ties are resolved in random order). This resulted in an accuracy of 93.09%. Interestingly, according to CombiTagger, 2.29% of all tokens are not tagged correctly by any of the taggers. This means that the best simple or weighted combination can only reach 97.71% accuracy.

In the second test case, we rearranged the order of the five individual tagger output files, i.e. we loaded them into CombiTagger using descending order of accuracy: Ice*, TnT*, fnTBL*, MBT, and MXP. Thus, in the case of a tie, the tag proposed by the most accurate tagger in the tie is selected. This resulted in an accuracy of 93.35%, which is consistent with the results obtained by Loftsson (2006) using the same taggers.

In the third test case, we added the sixth tagger, TreeTagger, to the combination pool, hoping for an increase in tagging accuracy relative to the previous text case. We loaded the tagger output files into CombiTagger using descending order of accuracy: Ice*, TnT*, fnTBL*, TreeTagger, MBT, and MXP. This test, however, resulted in an decrease in accuracy to 93.24%. Thus, the combined tagger does not benefit from adding TreeTagger to the combination pool. The reason seems to be that there are too many incorrect tags proposed by TreeTagger that become part of the "winner vote". Adding a sixth tagger to the combination pool is thus probably only beneficial if the given tagger is relatively accurate. For the remaining test cases, we therefore left TreeTagger out and only used the first five taggers.

| No. | Combination | Voting method | Accuracy (%) |
|---|---|---|---|
| 1. | fnTBL*, Ice*, MBT, MXP, TnT* | Simple | 93.09 |
| 2. | Ice*, TnT*, fnTBL* MBT, MXP | Simple | 93.35 |
| 3. | Ice*, TnT*, fnTBL*, TreeTagger, MBT, MXP | Simple | 93.24 |
| 4. | fnTBL*, Ice*, MBT, MXP, TnT* | Weighted | 93.33 |
| 5. | Ice*,TnT*,fnTBL*, MBT, MXP | Weighted | 93.41 |

Table 2: The average tagging accuracy of the combined taggers

The remaining two test cases were carried out using the weighted voting algorithm, in which the results depend more on the given weights and less on the order of the tagger output files. In the fourth test case, we weighted each of the five tagger output files with its corresponding tagging accuracy (from Table 1) and ordered them alphabetically. This resulted in an accuracy of 93.33%, which is 0.24 percentage points higher than using the simple voting algorithm with the same ordering of the tagger output files. Note that when all the given weights are close to 1.0, and random order of tagger output files is used, this test case is more or less equivalent to ordering the tagger output files using descending order of accuracy, as carried out in the second test case.

Finally, in the fifth test case, we again rearranged the order of the five individual tagger output files using descending order of accuracy. Furthermore, we weighted Ice* with 2.0, MXP with 1.1, but the three other taggers with 1.0. The reason for doing this is that we had noticed that in some cases Ice* and MXP agree on a correct tag, but are outvoted when the other three taggers agree on an incorrect tag. The given weight allocation will thus result in 3.1 votes to the joint tag proposed by Ice* and MXP, but 3.0 votes for the joint tag proposed by the other taggers. Applying this last combined tagger resulted in an accuracy of 93.41%.

To summarize, the difference between the best individual tagger and our best combined tagger is 1.58 percentage points, which amounts to an error reduction rate of 19.3%. Table 2 shows the results of the five test cases.

## Conclusion

In this paper, we have argued that it is important to develop combined taggers for morphologically complex languages, where tagging accuracy (using a single tagger) is low. We have described CombiTagger, an open source system for developing and evaluating combined taggers. CombiTagger is a language and tagset independent tool, which could encourage the development of combined taggers for various languages.

We have demonstrated that CombiTagger is flexible in the sense that different combination algorithms can be applied and that (voting) ties can be handled in an appropriate manner. Moreover, we have demonstrated the functionality of CombiTagger by using it to develop and evaluate combined taggers for tagging Icelandic text.

The current version of CombiTagger calculates tagging accuracy for all words. For future work, we propose an addition to CombiTagger to handle unknown words separately.

## Acknowledgements

We would like to thank the Árni Magnússon Institute for Icelandic Studies for providing access to the IFD corpus.

## References

Brants, T. 2000. TnT: A statistical part-of-speech tagger. In *Proceedings of the* $6^{th}$ *Conference on Applied natural language processing*, 224–231. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Brill, E. 1995. Transformation-Based Error-Driven Learning and Natural Language Processing: A Case Study in Part of Speech Tagging. *Computational Linguistics* 21(4):543–565.

Daelemans, W.; Zavrel, J.; Berck, P.; and Gillis, S. 1996. MBT: a Memory-Based Part of Speech Tagger-Generator. In *Proceedings of the* $4^{th}$ *Workshop on Very Large Corpora*, 14–27. Morristown, NJ, USA: Association for Computational Linguistics.

Dredze, M., and Wallenberg, J. 2008. Icelandic Data Driven Part of Speech Tagging. In *Proceedings of the* $46^{th}$ *Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, 33–36. Morristown, NJ, USA: Association for Computational Linguistics.

Džeroski, S.; Erjavec, T.; and Zavrel, J. 2000. Morphosyntactic Tagging of Slovene: Evaluating Taggers and Tagsets. In *Proceedings of the* $2^{nd}$ *International Conference on Language Resources and Evaluation*, 1099–1104. Paris, France: European Language Resources Association.

Hajič, J., and Kuboň, V. 2003. Tagging as a Key to Successful MT. In Obdržálek, D., and Tesková, J., eds., *Proceedings of the MIS*, 56–65. Prague, Czech Republic: MATFYZPRESS.

Helgadóttir, S. 2005. Testing Data-Driven Learning Algorithms for PoS Tagging of Icelandic. In Holmboe, H., ed., *Nordisk Sprogteknologi 2004*. Copenhagen, Denmark: Museum Tusculanums Forlag.

Kuba, A.; Felföldi, L.; and Kocsor, A. 2005. POS tagger combinations on Hungarian text. In Dale, R.; Wong, K.-F.; Su, J.; and Kwong, O., eds., *Proceedings of the* $2^{nd}$ *International Joint Conference on Natural Language Processing (IJCNLP-05)*, 191–196. Heidelberg, Germany: Springer.

Loftsson, H. 2006. Tagging Icelandic text: An experiment with integrations and combinations of taggers. *Language Resources and Evaluation* 40(2):175–181.

Loftsson, H. 2008. Tagging Icelandic text: A linguistic rule-based approach. *Nordic Journal of Linguistics* 31(1):47–72.

Ngai, G., and Florian, R. 2001. Transformation-based learning in the fast lane. In *Proceedings of the* $2^{nd}$ *meeting of the North American Chapter of the Association for Computational Linguistics on Language technologies 2001*, 1–8. Morristown, NJ, USA: Association for Computational Linguistics.

Pind, J.; Magnússon, F.; and Briem, S. 1991. *Íslensk orðtíðnibók [The Icelandic Frequency Dictionary]*. Reykjavik, Iceland: The Institute of Lexicography, University of Iceland.

Ratnaparkhi, A. 1996. A Maximum Entropy Model for Part-Of-Speech Tagging. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 133–142. Morristown, NJ, USA: Association for Computational Linguistics.

Santorini, B. 1990. Part-of-Speech Tagging Guidelines for the Penn Treebank Project. Technical report, Department of Computer and Information Science, University of Pennsylvania.

Schmid, H. 1994. Probabilistic Part-of-Speech Tagging Using Decision Trees. In *Proceedings of International Conference on New Methods in Language Processing*, 44–49. Manchester, United Kingdom: University of Manchester.

Shen, L.; Satta, G.; and Joshi, A. 2007. Guided Learning for Bidirectional Sequence Classification. In *Proceedings of the* $45^{th}$ *Annual Meeting of the Association of Computational Linguistics*, 760–767. Morristown, NJ, USA: Association for Computational Linguistics.

Sjöbergh, J. 2003. Combining POS-taggers for improved accuracy on Swedish text. In *Proceedings of the* $14^{th}$ *Nordic Conference of Computational Linguistics (NoDaLiDa 2003)*.

Toutanova, K.; Klein, D.; Manning, C.; and Singer, Y. 2003. Feature-Rich Part-of-Speech Tagging with a Cyclic Dependency Network. In *Proceedings of the 2003 Conference of the North American Chapter of the ACL on Human Language Technology*, 173–180. Morristown, NJ, USA: Association for Computational Linguistics.

van Halteren, H.; Zavrel, J.; and Daelemans, W. 2001. Improving Accuracy in Wordclass Tagging through Combination of Machine Learning Systems. *Computational Linguistics* 27(2):199–230.