Gut, Besser, Chunker – Selecting the best models for text chunking with voting

Balázs Indig and István Endrédy

Pázmány Péter Catholic University, Faculty of Information Technology and Bionics
MTA-PPKE Hungarian Language Technology Research Group
50/a Práter Street, 1083 Budapest, Hungary
{indig.balazs,endredy.istvan}@itk.ppke.hu

Abstract. The CoNLL-2000 dataset is the de-facto standard dataset for measuring chunkers on the task of chunking base *noun phrases (NP)* and arbitrary phrases. The state-of-the-art tagging method is utilising TnT, an HMM-based *Part-of-Speech tagger (POS)*, with simple majority voting on different representations and fine-grained classes created by lexcialising tags. In this paper the state-of-the-art English phrase chunking method was deeply investigated, re-implemented and evaluated with several modifications. We also investigated a less studied side of phrase chunking, i.e. the voting between different currently available taggers, the checking of invalid sequences and the way how the state-of-the-art method can be adapted to morphologically rich, agglutinative languages. We propose a new, mild level of lexicalisation and a better combination of representations and taggers for English. The final architecture outperformed the state-of-the-art for arbitrary phrase identification and NP chunking achieving the F-score of 95.06% for arbitrary and 96.49% for noun phrase chunking.

Keywords: phrase chunking, voting, IOB labels, multiple IOB representations, sequential tagging, HMM, MEMM, CRF

1 Introduction

For chunking in English, the current state-of-the-art tagging method is SS05 [12]. It was tested on the *CoNLL-2000 dataset* [14] and achieves an F-score of 95.23% for chunking base noun phrases (NP) and 94.01% for arbitrary phrases. Its concept is based on the lexicalisation of Molina and Pla [6] and voting between multiple data representations (see details in Section 2.1). However, the paper leaves multiple questions unanswered. How well does a basic tagger (originally developed for POS tagging), like *TnT* [2], maintain well-formedness of chunks? How converters handle ill-formed input? And so on. In this paper, we also investigate the independent impact of voting without lexicalisation to F-scores.

We also evaluate multiple taggers (with different underlying methods), with respect to the number of invalid sequences created during tagging and converting between representations.

2 Introduction to chunking

The task of applying tags to each token in a sentence consecutively is called *sequential tagging*. In general, the tagger tries to assign labels to (neighbouring) tokens correctly. The well-known special cases of this task include Part-of-Speech tagging, Named-Entity Recognition (NER) and chunking. In the latter two *IOB tags* are used to determine a *well-formed one level bracketing* on the text. In each sentence each token has an assigned label, which indicates the *beginning (B)*, *inside (I)*, *end (E)* of a chunk. One may distinguish the *outside (O)* of a sought sequence and one-token-long, *single (S or 1)* chunks. In addition, each marked sequence (except the outside labels) may have a type that corresponds to the task (*typed* or *untyped* case).

Table 1. Multiple IOB representations: An example sentence from the training set represented with five different IOB label sets

word	**IOB1**	**IOB2**	**IOE1**	**IOE2**	**IOBES**
These	I	B	I	E	S
include	O	O	O	O	O
,	O	O	O	O	O
among	O	O	O	O	O
other	I	B	I	I	B
parts	I	I	I	E	E
,	O	O	O	O	O
each	I	B	I	I	B
jetliner	I	I	E	E	E
's	B	B	I	I	B
two	I	I	I	I	O
major	I	I	I	I	O
bulkheads	I	I	I	E	E
,	O	O	O	O	O

2.1 Representation variants

Numerous representations exist, which try to catch information differently (see table 1 and [15] for details). There are variants which are basically the same. We try to follow the most convenient form of these. For example, there exists the *BILOU format*[1] (B=B, I=I, L=E, O=O, U=S) or the *bracket variant* ([=B,

[1] Begin, Inside, Last, Outside, Unique

I=I,]=E, O=O, []=S) which are equivalent to the *SBIEO*[2] format. In this paper, we prefer the name *IOBES*. *IOB2* format is commonly referred to as *IOB* or *BIO* or *CoNLL format*. There also exists the *Open-Close* notation (O+C or OC for short), which is roughly the same as bracket variant ([=B, I=O,]=E, O=O, []=S). In the untyped case, however, where only one type of chunks is sought one can not make a difference between the outside and the inside chunks, so the whole representation relies on the right positioning of opening and closing tags, which makes this representation very fragile. Besides, there are two inferior representations the *Inside-Outside* notation (IO) and the *prefixless* notation, where the tag consists only of the chunk type. These variants can not distinguish between subsequent chunks of the same type, but is easy to use (for searching nonconsecutive chunks) due to their simplicity.

2.2 Conversion of (possibly invalid) IOB sequences

POS taggers and IOB taggers might have many characteristics in common, but they are significantly different. IOB tags have an important substantial property: *well-formedness*, which relate tags at the intra-token level and can easily be inspected, while POS tags have no such property. In the literature, there is no mention of the proper handling of invalid sequences, especially during conversion between formats. However, lexicalising increases the number of invalid tag sequences assigned by simple taggers, because as the number of labels grows the sparse data problem arises.

At the time of writing this paper, we only found one tool that is publicly available and able to convert between multiple IOB representations: *IOBUtils*[3], which is the part of the *Stanford CoreNLP tools* written by Christopher Manning [5]. This converter is written in *JAVA 1.8* and has no external interface to use by its own. It can handle labelsets IOB1, IOB2, IOE1, IOE2, IO, SBIEO/IOBES and BILOU with or without type (e.g. B-VP) and it seemed robust and fault-tolerant. IOBUtils also has some similarities with the official evaluation script of CONLL-2000 task. First, it converts the chunks from the IOB labels into an intermediate representation and then transforms them into the required IOB labelset.

The original source of the SS05 approach (see Section 2.3 for details) served as another, independent implementation written in *Perl*. We still decided to create our own tool for conversion (inspired by the Perl version) in *Python 3* (to enable us to verify thoroughly the speed and robustness of the method). We also examined IOBUtils, a converter based on a rather different concept, jointly with the original SS05 and our approach. The latter two converters use the fact that two neighboring tags of each label and the current tag can unambiguously determine the result of the conversion. This method seemed to be very clumsy and fragile compared to IOBUtils because a big number of corner cases needed to be handled properly.

[2] Single, Begin, Inside, End, Outside

[3] `https://github.com/stanfordnlp/CoreNLP/blob/master/src/edu/stanford/nlp/sequences/IOBUtils.java`

In our measurements, we compared the three converters on a realistic data which could be malformed (the intermediate stages of the SS05 algorithm). To help later reproduction and application of our results, we made all three converters available freely along with our whole pipeline.

2.3 State-of-the-art chunker for English

For chunking English, the CoNLL-2000 shared task [14] is the de-facto standard for measuring and comparing taggers. The current state-of-the-art method, *SS05* [12], uses this dataset with every 10th sentence separated from the training set for development. We used this method as our baseline.

The concept of SS05 has three basic steps. First, the *lexicalisation*: every IOB label is augmented with POS tags and with words which are more frequent than threshold of 50[4] (see table 2 for details). Second, *the conversion and tagging*: the lexicalised tags converted to the five IOB formats, then tagging is performed with each format separately with TnT tagger [2]. In the third step, the five outputs are converted to a common format and *voted* by simple majority voting and the resulting tag becomes the final output. This method yields an F-score of 95.23% for NPs and 94.01% for arbitrary phrases. We mainly followed this concept, if it is not indicated otherwise.

Table 2. Lexicalisation: every IOB label is augmented with the POS tag, and (above a given frequency threshold that was set to 50 in SS05) also with the word as well (Full), and our lighter, mild version with less labels (just words) where just the labels of the frequent words is modified. We use '+' for separator because it is easier to parse than '-' used originally in SS05.

	Unlexicalised		**Lexicalised**			
	Original format		***Full***		***Mild (just words)***	
Word	**POS**	**IOB Label**	**POS**	**IOB Label**	**POS**	**IOB Label**
Rockwell	NNP	B-NP	NNP	NNP+B-NP	NNP	B-NP
said	VBD	O	VBD	O	VBD	O
the	DT	B-NP	the+DT	the+DT+B-NP	the+DT	the+DT+B-NP
agreement	NN	I-NP	NN	NN+I-NP	NN	I-NP

We were given the original Perl source code of SS05[5], which helped to understand better the workflow and the undisclosed details. Our first step was to reproduce their results, but the package did not contain the specific version of TnT that they were using, so we applied a potentially different version of the original Brants implementation that we had access to. The review of their code

[4] The authors did not disclose why they chosen 50 as threshold and whether they observed invalid sequences or not

[5] The full, original and bug fixed Perl source code is available at our github page `https://github.com/ppke-nlpg/SS05` with the permission of the authors.

resulted several bug fixes (in the Perl code), and a somewhat parallel (highly extended), refractored Python code that finally could reproduce their results, but with smaller F-scores.

3 Adaptation of lexicalisation (to agglutinative languages)

For English SS05 is a fast (but not freely available) method and has the clear advantage over other competing taggers. The question comes naturally: can it be used for agglutinative languages as well? The lexicalisation part of the method generates finer label classes by adding lexical information to the POS tags and labels of frequent words and POS tags to the labels of non-frequent words (see Table 2). Due to this procedure, there is an 18 times increase in the number of labels (from 23 to 422). Unfortunately, this high number of classes makes it impossible to use or reasonably slows down most of the taggers available for English, because of the exponential growth of learning time that can be observed for example in *Maximum Entropy-based (ME) learners*, which have been shown effective for Hungarian maximal NP chunking [3]. In ME training a multi-label learning problem is broken down into a number of independent binary learning problems (one per label) which makes the training process exponentially slower.

This problem with a high number of classes holds particularly true for agglutinative languages like Hungarian. If we want to apply the SS05 algorithm to agglutinative languages (to enjoy its benefits on speed and performance compared to other taggers), we can not use lexicalisation or we rule out most of the (state-of-the-art) taggers because of the *slow-down effect* due to the highly increased number of classes. For example the number of tags describing a token morphosyntactically in Hungarian in the Humor tagset [7] exceeds 200, which is one order of magnitude greater than the 36 Penn POS tags used for English.

Lexicalisation, combined with the large number of frequent words and the forms of IOB tags, makes the number of labels so high that it can not be handled easily with any tagger, e.g. Brant's TnT tagger has the upper limit of 2048 tags and ME taggers are also unsuitable because of the exponential slow-down mentioned above. This fact makes the method unsuitable for complex languages such as Hungarian.

In order to be able to tackle the challenge arousing from the high number of labels in agglutinative languages, we made experiments with both zero and mild lexicalisation. In the latter case just the frequent words (referenced as *just words*) and their labels are lexicalised[6], the other labels are left untouched (see table 2 for example) instead of the original *full lexicalisation* where the labels of the non-frequent words were augmented with their POS tag. (See Table 2 for details.)

[6] The label of those word and POS combinations, that were above some threshold were augmented with the word and POS tag. (See Table 2 for details.) The threshold was selected to be 50 by following the original SS05 paper.

4 Multiple taggers voted

Since the results of SS05 were published, many good taggers have been made available. The original authors wanted to show the impact of their method on an ordinary tagger, but left the evaluation of better taggers to the reader. As TnT is being a surprisingly fast POS tagger, we have no doubt that their method was extremely fast and efficient.

Our primary goal was to create a freely available, fast, and adaptable solution, that performs in pair or better than SS05 for English and can be evaluated on other (agglutinative) languages as well to measure the impact of voting on different representations with a wide spectrum of taggers. Therefore, we gathered freely available taggers with different inner-workings and compared them to TnT at the aforementioned levels of lexicalisation.

The following sections contain the brief introductions of the used taggers. We also wanted to evaluate the performance of some taggers, that represent the state-of-the-art on an agglutinative language to examine if they yield some additional improvement to our tagging task.

4.1 TnT

TnT is originally a POS tagger created by Thorsten Brants in 2000 [2]. The program is not freely available, closed source and written in C/C++. The underlying method uses second-order Hidden Markov Machine (HMM) with an extra check for boundary symbols at the end of sentences to prevent *'loose end'*. Additionally, the program uses many trickery solution disclosed partly in the original paper. The program also uses a *guesser*, that tries to model suffixes of rare unseen words, but this feature has no role in this experiment. We used a different version of this program from the one used originally for SS05, but with default settings replicating the original experiment.

4.2 NLTK-TNT

In search of a free substitute of TnT we found *Natural Language Toolkit for Python (NLTK)* [1] which implements a basic variant of TnT. This framework is written in Python and the TnT implementation lacks the handling of Unknown words (but has the API for a drop-in replacement, that was not necessary for this experiment) and using a simple HMM augmented with the checking of boundary symbols at the end of a sentence. The whole program is implemented in Python bearing in mind the simplicity instead of speed, therefore the tagging phase is very slow and unknown tags are substituted with the *Unk* symbol.

4.3 PurePOS

PurePOS [10] is a freely available, fast substitute of TnT, including most of the *advanced features* found in TnT. The program is implemented in Java and its speed is comparable with TnT. PurePOS is the state-of-the-art POS tagger for Hungarian [10], but to our knowledge it was never used for a task other than POS tagging. On its input the tagger needs word, stem and label tuples. As stem we used the word itself to make it impossible for the suffixguesser to have any impact on the experiments. We also experimented with the words instead of POS-tags and the combination of words and POS tags without success.

4.4 CRFsuite

We also tested *CRFsuite*[9], a first-order *Conditional Random Field* tagger. Conditional Random Fields (CRFs) is a popular method for general sequential tagging and CRFsuite is a fast, freely available implementation of first-order CRFs in C++. As input, one can define features for each token. The author has his own featureset for CoNLL-2000 task, which was evaluated as well (referred to as 'official CRFSuite'). There is also a slow-down with the increasing number of labels, but the running times are still feasible.

4.5 HunTag3

To evaluate different methods in our experiments we used *HunTag3* [3], a maximum entropy markov model (MEMM) tagger. The program consists of a simple Maximum Entropy (ME) unigram model combined with a first- or second-order Viterbi decoder. The user can define advanced features for tagging and the program can handle more than 2 million features. Its authors observed zero invalid tag sequence with it for Hungarian and English [3]. The program is written in *Python 3* and uses standard tools (SciPy [4], NumPy [16], Scikit-learn [11]) internally, therefore it is considerably fast without lexicalisation, but due to the maximum entropy approach many classes make the program exponentially slow. Therefore we did not use this tagger on lexicalised data.

The authors of *HunTag3* made it possible to use CRFSuite as an external tagger after the advanced featurization of *HunTag3*. We also included this method to our experiments. *HunTag3* (including CRFSuite as an external tagger) is successfully surveyed for *maximal Noun Phrase (NP) chunking*[7] making *HunTag3* the current state-of-the-art chunker for Hungarian [3]. In the same paper, the program is also used for Hungarian named-entity recognition (NER) and for English chunking on the CoNLL-2000 dataset too[8] [3].

5 The test bench

The original CoNLL-2000 data had to be prepared for processing: the development set of every tenth sentence was stripped and the set of frequent words

[7] where the top level NP in the parse tree is the sought

[8] *HunTag3* with CRFsuite in this combination outperformed *HunTag3*, but did not overcome the state-of-the-art.

was generated from the development set with the original threshold frequency of 50. We implemented an unified wrapper for each tagger and converter, which made it possible to use each tagger in conjunction with each converter for each representation on each lexicalisation level. The tagged data then were delexicalised and converted to the voting format, voted and then finally evaluated. We kept the evaluated data at each intermediate step, so we could get an insight of what is really happening and why. Furthermore, we also surveyed whether voting more independent taggers boosts F-scores or not. Voting was defined on different dimensions: for instance, voting between more taggers in one IOB format or between all the IOB formats with one tagger[9].

During the implementation we found that a good converter can be a crucial part of the system, and we found that not every bug could be squashed by testing every possible way of conversion on the original gold standard data.

6 Results

First we tried to reproduce the results of SS05 with the code we got. Since TnT was not the part of the package, we could not reproduce the exact same numbers of SS05 even with the original code. The numbers we got were significantly lower, because of the multiple bugs we found and fixed in the converting routines. The reproduced numbers, with the fixed programs (as the part of the full experiment) are in `[bracketed typewriter style]`. See table 3, 5 and table 6, 8.

6.1 Lexicalisation alone

To measure the gain on voting, we ran each tagger on each representation and lexicalisation level solely. We found that *CRFSuite with the official features* unanimously took the lead, and it turned out, that the best lexicalisation level was the mild lexicalisation (*just words*), where only the frequent words were lexicalised. See table 3, 4, 5.

Table 3. We tested each tagger on their own with **no lexicalisation**. The reproduced results of SS05 are in `[bracketed typewriter style]`, the best F-scores are in *italics*. In all cases *official CRFSuite* performed best.

	TnT	**NLTK TnT**	***HunTag3*** **Bigram**	***HunTag3*** **Trigram**	***HunTag3*** **CRFSuite**	**Official CRFSuite**	**PurePOS**
IOB1	`[82.23]`	82.10	91.73	92.00	92.41	*92.84*	84.28
IOB2	`[84.27]`	84.84	92.40	92.75	92.84	*93.40*	84.92
IOE1	`[78.83]`	78.75	91.85	92.05	92.10	*92.92*	84.39
IOE2	`[81.45]`	81.81	92.37	92.81	92.77	*93.25*	86.75
IOBES	`[86.95]`	87.58	93.26	93.47	93.41	*93.79*	87.85

[9] The full source code of the test bench is available freely at `https://github.com/ppke-nlpg/gut-besser-chunker`

Table 4. We tested each tagger on their own with **mild lexicalisation (just words)**. The best F-scores are in *italics* and all F-scores above 94% are **bold**. In all cases *official CRFSuite*, performed best.

	TnT	**NLTK**	***HunTag3***	***HunTag3***	***HunTag3***	**Official**	**PurePOS**
		TnT	**Bigram**	**Trigram**	**CRFSuite**	**CRFSuite**	
IOB1	87.39	87.33			93.20	***94.13***	88.33
IOB2	88.67	88.69			93.85	***94.70***	88.82
IOE1	87.06	87.00			93.35	***94.09***	88.50
IOE2	88.95	89.14			**94.13**	***94.61***	90.27
IOBES	90.23	90.66			**94.28**	***94.94***	91.04

Table 5. We tested each tagger on their own with **full lexicalisation**. The reproduced results of SS05 are in `[bracketed typewriter style]`, the best F-scores are in *italics* and all F-scores above 94% are **bold**. In all cases *official CRFSuite*, performed best.

	TnT	**NLTK**	***HunTag3***	***HunTag3***	***HunTag3***	**Official**	**PurePOS**
		TnT	**Bigram**	**Trigram**	**CRFSuite**	**CRFSuite**	
IOB1	`[91.12]`	91.00			92.64	***93.65***	91.42
IOB2	`[91.33]`	91.32			93.21	***94.03***	91.34
IOE1	`[91.17]`	91.04			92.94	*93.65*	91.35
IOE2	`[91.36]`	91.40			93.44	***94.12***	91.58
IOBES	`[91.43]`	91.61			93.35	***94.16***	91.65

6.2 Different representations voted against each other

Each training and test data were converted to all representations with all lexicalisation levels, tagged with each tagger and we selected each five representation separately and the remaining representations were converted by each converter and voted by simple majority voting (as in SS05). We also examined the performance of the converters, but apart from some bugs, there were no significant performance differences. We found that the mild (just words) lexicalisation performed best along with the official CRFSuite tagger and voting did not improved overall results compared to sole taggers. See tables 6, 7, 8 and 9.

Table 6. We tested each tagger with simple majority voting with **no lexicalisation**. The reproduced results of SS05 are in `[bracketed typewriter style]`, the best F-scores are in *italics*. In all cases *official CRFSuite* performed best.

	TnT	**NLTK**	***HunTag3***	***HunTag3***	***HunTag3***	**Official**	**PurePOS**
		TnT	**Bigram**	**Trigram**	**CRFSuite**	**CRFSuite**	
IOB1	`[84.40]`	84.64	92.60	92.83	93.11	*93.42*	85.47
IOB2	`[84.47]`	84.70	92.69	92.84	93.11	*93.45*	85.52
IOE1	`[84.46]`	84.70	92.62	92.84	93.09	*93.39*	85.50
IOE2	`[84.44]`	84.74	92.66	92.81	93.12	*93.42*	85.52
IOBES	`[85.50]`	85.64	93.03	93.17	93.32	*93.67*	86.11

Table 7. We tested each tagger with simple majority voting with **mild lexicalisation (just words)**. The best F-scores are in *italics* and all F-scores above 94% are **bold**. In all cases *official CRFSuite*, performed best.

	TnT	**NLTK TnT**	***HunTag3* Bigram**	***HunTag3* Trigram**	***HunTag3* CRFSuite**	**Official CRFSuite**	**PurePOS**
IOB1	88.58	88.65			**94.15**	***94.68***	89.19
IOB2	88.65	88.72			**94.17**	***94.70***	89.23
IOE1	88.63	88.72			**94.14**	***94.68***	89.23
IOE2	88.59	88.68			**94.18**	***94.70***	89.26
IOBES	89.27	89.36			**94.51**	***95.06***	89.77

Table 8. We tested each tagger with simple majority voting with **full lexicalisation**. The reproduced results of SS05 are in `[bracketed typewriter style]`, the best F-scores are in *italics* and all F-scores above 94% are **bold**. In all cases *official CRFSuite*, performed best.

	TnT	**NLTK TnT**	***HunTag3* Bigram**	***HunTag3* Trigram**	***HunTag3* CRFSuite**	**Official CRFSuite**	**PurePOS**
IOB1	`[91.73]`	91.63			93.76	***94.33***	91.77
IOB2	`[91.74]`	91.64			93.75	***94.32***	91.77
IOE1	`[91.73]`	91.66			93.74	***94.31***	91.77
IOE2	`[91.74]`	91.67			93.75	***94.33***	91.78
IOBES	`[92.18]`	92.08			93.96	***94.65***	92.20

Table 9. Average gain on voting compared to sole tagging on each representation for each tagger. The table shows no real difference (<1%) with lexicalisation and for taggers other than TnT.

	TnT	**NLTK TnT**	***HunTag3* Bigram**	***HunTag3* Trigram**	***HunTag3* CRFSuite**	**CRFSuite Official**	**PurePOS**
No lex.	1.908	1.868	0.398	0.282	0.444	0.23	-0.014
just words	0.284	0.262			0.468	0.27	-0.056
Full lex.	0.542	0.462			0.676	0.466	0.39

In many cases during voting there was a draw between the taggers (2 x 3 or 2 x 2 identical votes and one tagger with a third opinion) and the number of votes were equal. The original paper and code of SS05 method has not clarified how these situations were handled. We used lexical ordering on tags by default because the code suggested us this was happened. Although we think not handling draws in voting by not setting any *tie breaking rule* is a generally bad idea as it makes the method unpredictable as this important detail is left to the implementer. Therefore we set the following tie breaking rule: in case of tie the first voter is right. In section 6.2 IOB2 was always chosen to be first as the least converted representation achieving no significant change in results. But in section 6.2 *CRFSuite tagger with its official features* was chosen to be first causing the results to unanimously improve with about 1% for each representation (which we consider to be an artefact).

6.3 Different taggers voted against each other

Table 10. Different taggers were voted against each other in each representation on each lexicalisation level. The best F-scores are in *italics* and all F-scores above 94% are **bold**. In all cases *mild lexicalisation (just words)*, performed best.

	No lexicalisation	**Just words lexicalised**	**Full lexicalisation**
IOB1	91.89	*93.55*	93.45
IOB2	92.61	***94.11***	93.81
IOE1	92.00	*93.56*	93.51
IOE2	92.75	***94.36***	**94.04**
IOBES	93.32	***94.60***	**94.03**

We voted different taggers against each other in each representation on each lexicalisation level to examine if using more taggers can add information to voting or not. None of the scores performed as good as *CRFSuite official* on its own. See table 10 for details.

6.4 Converters

At the time of writing this paper there is only one converter available (CoreNLP IOBUtils). We implemented our own version and used the original SS05 code too. After comparing the three converters we found that IOBUtils is a robust and error correcting converter. We fixed a lot of bugs in the other two implementations, reducing the differences between them. Our converter was able to count invalid tag sequences[10]. This made possible it to show how lexicalisation affect the number of invalid sequences on different levels of lexicalisation. See table 11 for details.

Table 11. Number of invalid sequences (no lex./just words/full lex.): we can observe, that by tagger, representation and lexicalisation the numbers differ in wide ranges, but it is clear that lexicalising makes the taggers harder to produce valid sequences.

	TnT	**NLTK TnT**	***HunTag3* CRFSuite**	**CRFSuite Official**	**PurePOS**
IOB1	168/234/319	148/230/313	286/260/266	306/294/304	197/274/317
IOB2	423/662/634	490/658/633	0/19/168	0/14/111	0/4/49
IOE1	0/1/1	0/2/2	4/13/11	0/2/2	0/0/0
IOE2	174/107/205	187/84/215	0/44/254	0/12/158	3/16/76
IOBES	862/805/985	647/702/898	2/95/865	2/51/521	2/22/210

[10] By counting we mean that we counted the tags with invalid neighbours.

6.5 Summary

Albeit voting does not add much to the F-scores, official CRFSuite tagger and mild lexicalisation (just words) voted on IOBES achived the best results both in arbitrary phrase chunking and NP chunking. See table 12 for details. However, if we consider the training times we can not ignore the fact that on one hand, the lexicalisation is the best performing factor of the SS05 method and can not be omitted, but on the other hand, it is also the factor that slows down the whole algorithm.

The cornerstone of the lexicalisation is the right frequency threshold and lexicalisation rules. In our experiments we used the ones given by the SS05 algorithm as we only wanted to select the best tagger and also to eliminate the unneccessary part of the lexicalisation to be able to adapt easier for agglutinative languages. The current threshold is not a well-founded, language agnostic number. We think that, the deep investigation of the lexicalisation threshold could improve performance and also would optimise the training times. Unfortunately, searching for the right lexicalisation constant in conjunction with the best tagger would made the training times infeasible.

Given the above fact, the overhead caused by the voting is insignificant. Especially if one just want to train the tagger only once, because the tagging and voting not demand considerably more time.

Table 12. Summary of final F-scores, which outperformed the previous state-of-the-art results (+1% improvement)

method	**arbitrary phrase chunking**	**NP chunking**
SS05 [12]	94.01	95.23
SS05 (as we could reproduce)	91.74	93.99
Official CRFSuite + mild lex. (just words)	**95.06**	**96.49**

7 Conclusion

We reimplemented the state-of-the-art SS05 chunking method and asked for the authors for their source code to determine the parts of the algorithm that were undisclosed before in the original paper. On the one hand, we were able to understood their workflow. On the other hand, we could fix numerous bugs in their program and rerun all their tests. We tested a mild level of lexicalisation (just words) in conjunction with more taggers with different inner-workings. We tested multiple IOB label converters and fixed their bugs.

Our mild lexicalisation in conjunction with CRFSuite performed best across the taggers and lexicalisations. With voting these scores were further improved, however we found that simple majority voting in general does not add much to a combination of a good tagger with the appropriate level of lexicalisation[11].

We found that voting between all the examined taggers with simple majority voting can neither outperform the results of the best tagger alone in any representation nor on any level of lexicalisation. We examined the adaptation of the system to agglutinative languages with far more possible labels at lexicalisation and checked the number of invalid sequences created by taggers which has an influence on converters and therefore on the final result. We made our pipeline including the highly extended version of the SS05 algorithm fully available, to ease later reproduction and fine-tuning of our method.

8 Future work

In English one must distinguish between *arbitrary phrase identification* where most of the tokens belongs to one of the many chunk types and the *base Noun Phrase (NP) chunking task* where the only sought type is the lowest level NP-s. In Hungarian the most sought chunk types are the *maximal Noun Phrases (NP)* where the top level NP in the parse tree is needed. In this paper we only could measure our method on the first two, but we think that in the future all three of the task could be examined adding the named-entity recognition (NER) task for both English and Hungarian as well. All of the aforementioned task can be investigated in the same way by the same programs evaluated here in terms of voting[12], but this task spans beyond this paper.

We think there is much room for improvement on the fine-tuning of the lexicalisation such as using word classes generated by word embedding, a nowadays popular method, for lexical categorisation [13] and use the same technique to improve the corpus quality [8].

We also think that the possibility of using decision trees or other meta-learners to the voting task should be considered as all the needed tools are available including the problem of the large number of draws during the voting. Additionally, there is also room for experimenting with other available taggers as with our mild lexicalisation less tagger is ruled out by the large number of labels.

Acknowledgments We would like to thank professor Anoop Sarkar [12] for his cooperation and providing the original Perl code and letting us to use and distribute it with our own. This helped a lot to understand their algorithm better.

References

1. Bird, S., Klein, E., Loper, E.: Natural language processing with Python. O'Reilly Media, Inc. (2009)

[11] Which makes the whole method more adaptable to agglutinative languages.
[12] as the full pipeline is publicly available

2. Brants, T.: TnT: a statistical part-of-speech tagger. In: Proceedings of the sixth conference on Applied natural language processing. pp. 224–231. Association for Computational Linguistics (2000)
3. Endrédy, I., Indig, B.: HunTag3: a general-purpose, modular sequential tagger – chunking phrases in English and maximal NPs and NER for Hungarian. In: 7th Language & Technology Conference, Human Language Technologies as a Challenge for Computer Science and Linguistics. pp. 213–218. Poznań: Uniwersytet im. Adama Mickiewicza w Poznaniu (2015)
4. Jones, E., Oliphant, T., Peterson, P., et al.: SciPy: Open source scientific tools for Python (2001), `http://www.scipy.org/`
5. Manning, C.D., Surdeanu, M., Bauer, J., Finkel, J., Bethard, S.J., McClosky, D.: The Stanford CoreNLP natural language processing toolkit. In: Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations. pp. 55–60 (2014), `http://www.aclweb.org/anthology/P/P14/P14-5010`
6. Molina, A., Pla, F.: Shallow parsing using specialized hmms. The Journal of Machine Learning Research 2, 595–613 (2002)
7. Novák, A.: A new form of humor – mapping constraint-based computational morphologies to a finite-state representation. In: Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC'14). ELRA, Reykjavik, Iceland (2014)
8. Novák, A.: Improving corpus annotation quality using word embedding models. Polibits (2016), Accepted for publication
9. Okazaki, N.: CRFsuite: a fast implementation of Conditional Random Fields (CRFs) (2007), `http://www.chokkan.org/software/crfsuite/`
10. Orosz, G., Novák, A.: Purepos 2.0: a hybrid tool for morphological disambiguation. In: RANLP. pp. 539–545 (2013)
11. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., et al.: Scikit-learn: Machine learning in Python. The J. of Mach. Learn. Res. 12 (2011)
12. Shen, H., Sarkar, A.: Voting between multiple data representations for text chunking. In: Kégl, B., Lapalme, G. (eds.) Advances in Artificial Intelligence, 18th Conference of the Canadian Society for Computational Studies of Intelligence, Canadian AI 2005, Victoria, Canada, May 9-11, 2005, Proceedings. Lecture Notes in Computer Science, vol. 3501, pp. 389–400. Springer (2005), `\url{http://dx.doi.org/10.1007/11424918_40}`
13. Siklósi, B.: Using embedding models for lexical categorization in morphologically rich languages. In: Gelbukh, A. (ed.) Computational Linguistics and Intelligent Text Processing: 17th International Conference, CICLing 2016, Konya, Turkey, April 3-9, 2016. Springer International Publishing, Cham (2016)
14. Tjong Kim Sang, E.F., Buchholz, S.: Introduction to the CoNLL-2000 shared task: Chunking. In: Proceedings of the 2Nd Workshop on Learning Language in Logic and the 4th Conference on Computational Natural Language Learning - Volume 7. pp. 127–132. ConLL '00, Association for Computational Linguistics, Stroudsburg, PA, USA (2000), `\url{http://dx.doi.org/10.3115/1117601.1117631}`
15. Tjong Kim Sang, E.F., Veenstra, J.: Representing text chunks. In: Proceedings of the ninth conference on European chapter of the Association for Computational Linguistics. pp. 173–179. Association for Computational Linguistics (1999)
16. Van Der Walt, S., Colbert, S., Varoquaux, G.: The NumPy array: a structure for efficient numerical computation. Comp. in Sci. & Eng. 13(2) (2011)