

# Slovene NLTK Tagger

Niko Colnerič  
Faculty of Computer and  
Information Science  
University of Ljubljana  
niko.colneric@gmail.com

Nejc Banič  
Faculty of Computer and  
Information Science  
University of Ljubljana  
banic.nejc@gmail.com

## ABSTRACT

This paper describes the process of building NLTK tagger for custom language. NLTK stands for Natural Language Toolkit, a library for use in Python (version 2.7). Tagger is a object, which processes a sequence of words and attaches a part of speech tag to each word. Our language of implemetation was Slovene. We constructed sequential **Trigram tagger**, **Brill tagger** and classifier based **Naive Bayes tagger**, which differ in tagging accuracy and time consumed for tagging. All this differences are also detaily compared. We recommend using the Brill tagger.

**Keywords:** Tagger, corpus, training, tagging accuracy, time consumed

## I. INTRODUCTION

Let us begin with the description of implemented taggers.

### A. Trigram tagger<sup>1</sup>

To understand how *trigram* works, we should first understand *unigram*. Unigram taggers are based on a simple statistical algorithm: *for each token, assign the tag that is most likely for that particular token*. In the case of tagging with unigram tagger, we only consider the current token, in isolation from any larger context. The best we can do is tag each word with its *a priori* most likely tag.

An n-gram tagger is a generalization of a unigram<sup>2</sup> tagger whose context is the current word together with the part-of-speech tags of the  $n - 1$  preceding tokens, as shown in figure 1.

The tag to be chosen,  $t_n$ , is circled, and the context is shaded in grey. In the example of an n-gram tagger shown in figure 1, we have  $n = 3$ ; that is, we consider

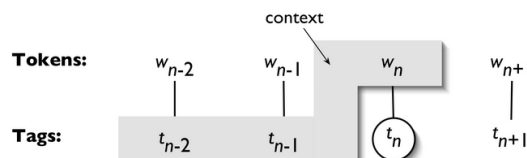


Fig. 1. Tagger Context

the tags of the two preceding words in addition to the current word. An n-gram tagger picks the tag that is most likely in the given context.

Although we use the name trigram all throughout this paper, we actually refer to sequential tagger, which consists of *affix*<sup>3</sup>, *unigram*, *bigram* and *trigram* taggers in this order respectively. First we build affix, which is used as ad *backoff-tagger* for unigram. Unigram is afterwards used as a backoff-tagger for bigram, which is finally used for trigram.

### B. Brill tagger

*TODO* - BANIČ Banič glej ta link : [nltk.googlecode.com/svn/trunk/doc/book/ch05.html](http://nltk.googlecode.com/svn/trunk/doc/book/ch05.html) poglavje 5.6 The Brill tagger is another part-of-speech tagger and was first described by Eric Brill in 1993. Before trying to explain it, we must address issues with all n-gram taggers (e.g. *Trigram tagger*).

First issue with n-gram tagger is size of their n-gram table. If we wan't to use the tagger on mobile computing device, compromise must be taken between performance and model size.

A second issue with n-gram tagger is that the only information it considers from prior context is tags. This is not practical because words themselves are useful information also. Therefore n-gram taggers are impractical models to be used to identify words in context.

<sup>1</sup>Definition and images summerized from [4]

<sup>2</sup>Unigram could also be reffered to as 1-gram

<sup>3</sup>The Affix tagger learns prefix and suffix patterns to determine the part of speech tag for word.

To bypass these problems we can use *Brill tagger*. It can be summarised as an transformation-based learning and performs very well using models that are just a fraction of the size of n-gram taggers.

The general idea is very simple: guess the tag of each word, then go back and fix the mistakes. In this way, a Brill tagger successively transforms a bad tagging of a text into a better one. As with n-gram tagging, this is a supervised learning method, since we need annotated training data to figure out whether the tagger's guess is a mistake or not. However, unlike n-gram tagging, it does not count observations but compiles a list of transformational correction rules.

### C. Naive Bayes tagger

*TODO - BANIC*

## II. RELATED WORK

### A. Nltk-trainer

The code for building the tagger was taken from *Nltk-trainer* project [2], whose author is Jacob Perkins. Its an open-source project hosting on Github, with moto to *Train NLTK objects with zero code*. We almost exclusively used the script `train_tagger.py`, which has the ability to generate NLTK taggers. The

### B. JOS corpus

The basic component for building Slovene tagger is **JOS corpus** from JOS project [1]. It stands for: *Jezikoslovno Označevanje Slovenskega jezika* (Linguistic annotation of Slovenian language). It contains collections of various text in Slovenian language with annotation. For our project we used jos1M corpus that contain 1 million words with it's appropriate lemmas and morphosyntactic descriptions. The JOS is compatible with Slovene *MULTEXT-East morphosyntactic specifications Version 4*[3]. Its basic purpose is to define word classes. For each class there are various attributes and their appropriate values, which they can be mapped into morphosyntactic descriptions (i.e. MSD). The structure of JOS corpora is in **Extensible Markup Language (XML)**.

## III. IMPLEMENTATION

### A. Corpus transformation

*TODO - BANIC*

We can easily manipulate XML files in various programming languages (e.g. Python with `xml.dom.minidom`). This is mandatory because *nltk-trainer*[2] expects special form of input file. For further information see III-A.

### B. Training the tagger

As mentioned above, the script `train_tagger.py` was used for this pupose. Its mandatory argument is a corpus in .pos format, which generation is described in III-A. Beside this, there are many optional arguments. Here we will describe basics, for further description run script help with

```
train_tagger.py --help
```

One is wheather to generate a Sequential Tagger, a Brill Tagger or a Classifier Based Tagger. For Sequential we can then select sequential backoff algorithm. This can be any combination of the following letters:

- a: AffixTagger
- u: UnigramTagger
- b: BigramTagger
- t: TrigramTagger

The default is *aubt*, which was used for our Trigram tagger, but you can set this to the empty string to not train a sequential backoff tagger. The Brill Tagger is trained in front of the other tagger, which in our implementation was the Trigram tagger. When building classifier based algorithm, we must select the classifier or the sequence of classifier. Following are supported: Naive-Bayes, DecisionTree, Maxent, GIS, IIS, CG, BFGS, Powell, LBFGBS, Nelder-Mead, MEGAM and TADM. We should mention, that the time spent for training classifier based tagger, is much larger than, the time for a sequential or brill.

Script also has the possibility to set default tag. We used *-Neznan-*, which is Slovene word for *-unknown-*.

Last argument, that should be mentioned, is wheather to evaluate the tagger or not and the percentage of sentaces to be used. The default value is 1, which means that the *same* sentences are first used for training the tagger and then for evaluation. We reccomend this argument to be set below 1. If so, the right approach is used. Sentances are devided into two groups, one in used for training and second for evaluation. *Fraction* is the percentage of sentances assigned to training group. This was used for evaluating constructed taggers. For results of this evaluation see section IV).

### C. Usage with NLTK

*TODO - BANIC*

### D. Encoding problems

*TODO - BANIC*

#### IV. RESULTS

For all constructed taggers, we here provide their accuracy. The results were obtained by repeating the proces of building the tagger with various evaluation arguments. See table I and figure 2.

<i>fraction</i>	<i>Trigram</i>	<i>NaiveBayes</i>	<i>Brill</i>
0.75	0.8286	0.8451	0.8490
0.80	0.8304	0.8461	0.8514
0.82	0.8306	0.8465	0.8515
0.84	0.8299	0.8462	0.8508
0.86	0.8306	0.8472	0.8512
0.88	0.8299	0.8474	0.8517
0.90	0.8288	0.8467	0.8507
0.91	0.8278	0.8462	0.8489
0.92	0.8276	0.8464	0.8490
0.93	0.8281	0.8464	0.8491
0.94	0.8278	0.8461	0.8496
0.95	0.8289	0.8475	0.8510
0.96	0.8388	0.8538	0.8609
0.97	0.8401	0.8568	0.8612
0.98	0.8417	0.8588	0.8629
0.99	0.8379	0.8567	0.8578
<i>average</i>	<b>0.8317</b>	<b>0.8490</b>	<b>0.8529</b>

TABLE I  
TAGGERS ACCURACY FOR VARIOUS FRACTIONS.

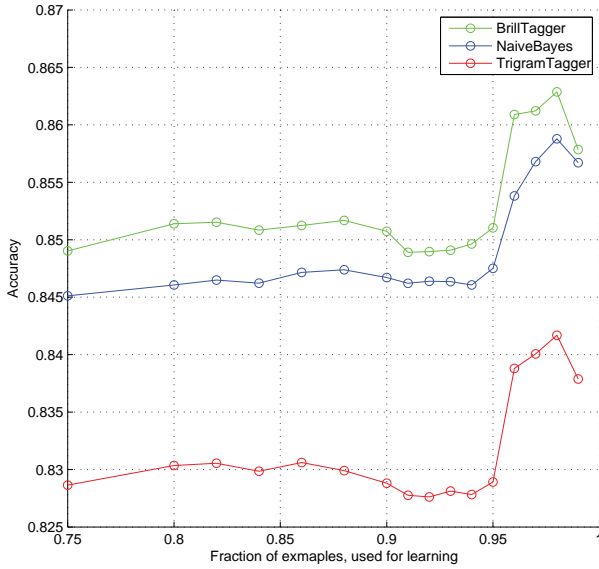


Fig. 2. Accuracy evaluation results.

To conclude, there are no major differences in tagging accuracy. The Brill and the Naive Bayes are about 2% better than Trigram.

However, during this evaluation we discover an other paramater, which should be taken into account. Under the assumption that most of the time was actually spent for tagging the sentences (besides just testing for equality and percentage calculation are needed), we noticed that the time for evaluations differ drastically. The results of resarching time consumption are listed in table II and graphically represented on figure 3.

<i>number of words</i>	<i>Trigram</i>	<i>NaiveBayes</i>	<i>Brill</i>
50	0.0003	4.1543	0.0011
75	0.0004	6.2464	0.0014
100	0.0006	8.5232	0.0018
125	0.0008	10.6370	0.0020
150	0.0009	12.4937	0.0024
175	0.0011	14.5484	0.0027
200	0.0011	16.3944	0.0029
225	0.0013	18.6332	0.0032
250	0.0015	20.7357	0.0035
275	0.0017	22.8280	0.0041
300	0.0018	24.8243	0.0042
325	0.0018	27.4270	0.0043
350	0.0022	29.2458	0.0051
400	0.0023	32.9599	0.0052
425	0.0024	36.0322	0.0056
450	0.0027	38.1471	0.0062
475	0.0027	39.2278	0.0064
500	0.0029	41.2688	0.0063

TABLE II  
TIME SPENT IN SECONDS FOR TAGGING DIFFERENT NUMBERS OF WORDS.

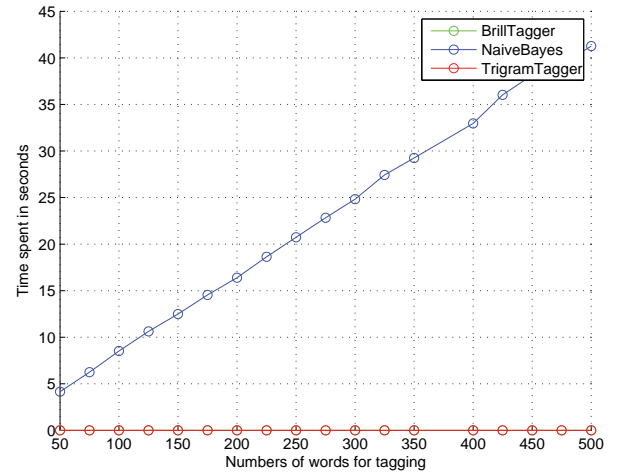


Fig. 3. Spent time measurements. NOTE: The green line is not visible because it is right under the red one.

The difference here is enormous. Brill and Trigram

run at almost the same speed, while Naive Bayes runs extremely slower.

## V. CONCLUSION

*TODO - BANIČ*

## ACKNOWLEDGEMENTS

We would like to express our our gratitude to all the contributor of *JOS* [1] and *nltk-trainer* [2] projects and out mentor at Faculty of Computer and Information Science.

## REFERENCES

- [1] <http://nl.ijs.si/jos/>
- [2] <https://github.com/japerk/nltk-trainer>
- [3] <http://nl.ijs.si/ME/V4/msd/html/msd-sl.html>
- [4] <http://www.nltk.org/book>