MBT: Memory-Based Tagger

version 1.0

Reference Guide

ILK Technical Report – ILK 02-09

Walter Daelemans* Jakub Zavrel*† Antal van den Bosch
Ko van der Sloot

Induction of Linguistic Knowledge
Computational Linguistics
Tilburg University

(*) CNTS - Language Technology Group
University of Antwerp

(†) Textkernel B.V.

P.O. Box 90153, NL-5000 LE, Tilburg, The Netherlands
URL: http://ilk.uvt.nl[1]

November 6, 2002

[1]This document is available from http://ilk.uvt.nl/downloads/pub/papers/ilk.0209.ps. All rights reserved Induction of Linguistic Knowledge, Tilburg University and CNTS Research Group, University of Antwerp.

Contents

Preface

Part-of-Speech (POS) tagging is a process in which a morpho-syntactic class is assigned to each word in a text on the basis of the characteristics of the word and of the context in which it occurs. It is a first level of abstraction in text analysis, and is used in many language technology applications such as (shallow) parsing, information retrieval, spelling error correction, speech synthesis, and text mining.

Memory-Based Tagging is an approach to POS Tagging based on *Memory-Based Learning* (MBL). As an adaptation and extension of the classical k-Nearest Neighbor (k-NN) approach to statistical pattern classification, MBL has proven to be successful in a large number of tasks in Natural Language Processing (NLP). Since 1998, we have made available TiMBL, a flexible software tool incorporating an extensive family of memory-based learning algorithms. The most recent version of software and documentation are available free for research and teaching from `http://ilk.uvt.nl`. To use this software for POS Tagging is not straightforward, however. We want to be able to use previous tagger decisions as input for current decisions, we want to build separate case bases for known and unknown words, allow global sentence-level optimization, etc. The software you will find here implements this specific tagging functionality by wrapping software around TiMBL while keeping most of the flexibility of TiMBL intact.

Memory-Based Tagging was originally proposed in (Daelemans, 1995). The most complete description to date is contained in the union of (Daelemans et al., 1996) and (Zavrel and Daelemans, 1999). As is the case for TiMBL, the main effort in the development and maintenance of this software was invested by Ko van der Sloot. The system started as a rewrite of code developed by Peter Berck and adapted by Jakub Zavrel. The code has benefited substantially from trial, error and scrutiny by past and present members of the ILK and CNTS groups in Tilburg and Antwerp. This software was written in the context of projects funded by the Netherlands Organization for Scientific Research (NWO), Tilburg University Faculty of Arts, the Flemish National Science Foundation (FWO), and the University of Antwerp Research Council.

The current release (version 1.0) is the first public release, and uses TiMBL version 4.3. Note that you have to download and license your personal download of TiMBL 4.3 separately, if you have not already done so, to be able to compile the tagging software. See `http://ilk.uvt.nl`.

This reference guide is structured as follows. In Chapter 1 you can find the terms of the license according to which you are allowed to use MBT. The subsequent chapter gives instructions on how to install the software on your computer. Next, Chapter 3 offers a tutorial and information about the different parameters of the system. Readers who are interested in the theoretical and technical details of Memory-Based Tagging should consult (Daelemans et al., 1996; Zavrel and Daelemans, 1999). These papers are also included in this software distribution.

This document does *not* contain information about the TiMBL software package. In order to make the best use of this tagging software, it is strongly advised to get acquainted with the functionality of TiMBL, as explained in its reference guide (Daelemans et al., 2002).

Chapter 1

License terms

Downloading and using the MBT software implies that you accept the following license terms:

Tilburg University and University of Antwerp (henceforth "Licensers") grant you, the registered user (henceforth "User") the non-exclusive license to download a single copy of the MBT program code and related documentation (henceforth jointly referred to as "Software") and to use the copy of the code and documentation solely in accordance with the following terms and conditions:

- The license is only valid when you register as a user. If you have obtained a copy without registration, you must immediately register by sending an e-mail to `Mbt@uvt.nl`.
- User may only use the Software for educational or non-commercial research purposes.
- Users may make and use copies of the Software internally for their own use.
- Without executing an applicable commercial license with Licensers, no part of the code may be sold, offered for sale, or made accessible on a computer network external to your own or your organization's in any format; nor may commercial services utilizing the code be sold or offered for sale. No other licenses are granted or implied.
- Licensers have no obligation to support the Software it is providing under this license. To the extent permitted under the applicable law, Licensers are licensing the Software "AS IS", with no express or implied warranties of any kind, including, but not limited to, any implied warranties of merchantability or fitness for any particular purpose or warranties against infringement of any proprietary rights of a third party and will not be liable to you for any consequential, incidental, or special damages or for any claim by any third party.
- Under this license, the copyright for the Software remains the joint property of the ILK Research Group at Tilburg University, and the CNTS Research Group at the University of Antwerp. Except as specifically authorized by the above licensing agreement, User may not use, copy or transfer this code, in any form, in whole or in part.
- Licensers may at any time assign or transfer all or part of their interests in any rights to the Software, and to this license, to an affiliated or unaffiliated company or person.
- Licensers shall have the right to terminate this license at any time by written notice. User shall be liable for any infringement or damages resulting from User's failure to abide by the terms of this License.

Chapter 2

Installation

You can get the MBT package as a gzipped tar archive from:

`http://ilk.uvt.nl/software.html`

Following the links from that page, you will be required to fill in registration information and to accept the license agreement. You can proceed to download the file `Mbt.1.0.tar.gz`. This file contains the complete source code (C++) for the MBT program, a sample data set, the license and the documentation. The installation should be relatively straightforward on most UNIX systems.

To install the package on your computer, unzip the downloaded file:

```
> gunzip Mbt.1.0.tar.gz
```

and unpack the tar archive:

```
> tar -xvf Mbt.1.0.tar
```

This will make a directory `Mbt.1.0` under your current directory. Change directory to this:

```
> cd Mbt.1.0
```

and edit the variable `TIMBLPATH` in the `Makefile`. This variable should be set to the directory where a *compiled* Timbl 4.3 system resides.

Compile the executable by typing `make`[1], assuming that `make` is actually `gnumake` on your system. Solaris users should use `gnumake` explicitly, since their `make` defaults to SunOS `make`, which does not operate correctly.

If the process was completed successfully, you should now have executable files named `Mbtg`, `Mbt`, and a static library `libMbt.a`.

The e-mail address for problems with the installation, bug reports, comments and questions is `Mbt@uvt.nl`.

[1]We have tested this with gcc versions 2.95.2, 2.95.3, and 3.0.3

Chapter 3

Tutorial and Reference

Memory-based tagging is based on the idea that words occurring in similar contexts will have the same POS tag. The idea is implemented using the memory-based learning software package TiMBL (`http://ilk.uvt.nl/software.html`). The MBT software package makes use of TiMBL to implement a Part of Speech (POS) tagger–generator. The software consists of two executables: `Mbtg` to generate a tagger, and `Mbt` to use a generated tagger on text data. Given as input an annotated (tagged) corpus, `Mbtg` generates a lexicon, and case bases for known words (words in the corpus, hence also in the lexicon), and unknown words (in order to guess the POS tag of words not in the corpus). The lexicon associates words with their ambiguous tag, henceforth referred to as *ambitag*: a symbol representing all the POS tags a word can have according to the corpus. The `Mbt` executable takes a tagger constructed by `Mbtg` as input and can be used to tag text with it. For theoretical background, see (Daelemans et al., 1996; Zavrel and Daelemans, 1999).

This document exemplifies how to use the MBT package. As an example data file, we have added a small part of the tagged "Eindhoven Corpus" of Dutch written text (Uit den Boogaard, 1975) to the distribution, a Dutch POS-tagged corpus. The data consists of only about 100,000 words, so the quality of taggers trained with this data will not be high. It is meant as a toy corpus. The tag set used consists only of 10 broad-category POS tags.

3.1 `Mbtg`, the tagger generator

The input file containing the material for generating a tagger must consist of two whitespace-separated columns. The first column contains a word or punctuation mark with in the corresponding position of the second column its POS tag. A line may also contain only the symbol <`utt`> to mark the end of a sentence. The following are the two first sentences of the file `eindh.data`, present in this distribution of MBT.

```
Dit Pron
in Prep
verband N
met Prep
de Art
gemiddeld Adj
langere Adj
levensduur N
van Prep
```

```
de Art
vrouw N
. Punc
<utt>
De Art
verzekeringsmaatschappijen N
verhelen V
niet Adv
dat Conj
ook Adv
de Art
rentegrondslag N
van Prep
vier Num
procent N
nog Adv
een Art
ruime Adj
marge N
laat V
ten Prep
opzichte N
van Prep
de Art
thans Adv
geldende V
rentestand N
. Punc
<utt>
```

3.1.1 Defining feature patterns

In generating the tagger, information has to be provided to the tagger generator about the context and the form of the words to be tagged. This is done by the parameters `-p` (feature pattern for known words), and `-P` (feature pattern for unknown words). Patterns are built up as combinations of the following symbols:

For -p and -P

`d`	Left context (tag)
`a`	Right context (ambitag)
`w`	Left or right context (word)

For -p only (known words)

`f`	Focus (ambitag for known words)
`W`	Focus (word)

For -P only (unknown words)

`F`	Focus (position of the unknown word)
`c`	The focus contains capitalized characters
`h`	The focus word contains a hyphen
`n`	The focus word contains numerical characters
`p`	Character at the start of the word
`s`	Character at the end of the word

The symbols `d`, `a`, `w`, `p`, and `s` can occur more than once to indicate the scope of the context. Symbols to the left of the focus symbols indicate left context, and symbols to the right of the focus symbols indicate right context.

For example, for known words, the following are a few possible patterns:

`dfa`	focus ambitag with one disambiguated tag on the left and one ambitag to the right
`ddfa`	focus ambitag with two disambiguated tags to the left and one ambitag to the right
`ddfWa`	as previous, plus the focus word (note that `W` can be declared only immediately after `f`)
`dwdwfWaw`	as previous, plus for each context tag the corresponding word (two left, one right)

For unknown words:

`dFa`	one disambiguated tag to the left and one ambitag to the right
`psdFa`	as previous, plus the first and last letter of the unknown word to be tagged
`psssdFa`	as previous, plus the three last letters of the word to be tagged
`psssdwFaw`	as previous, plus the left and right neighboring words

An example command line for tagger generation is the following (">" is the command line prompt):

```
 > Mbtg -T eindh.data -p ddfa -P psssdFa
```

This will generate a tagger based on information about the previous two predicted tags and the following ambitag for the known words, and about the first and three last letters of the word, the previous predicted tag, and the following ambitag for the unknown words. Supposing the annotated corpus you use to construct the tagger is in the two-column file `eindh.data`, Mbtg will generate the following output to standard output:

```
Memory Based Tagger Generator Version 1.0
  (c) ILK and CNTS 1998 - 2002.
  Induction of Linguistic Knowledge Research Group, Tilburg University
  Centre for Dutch Language and Speech, University of Antwerp

  Based on Timbl version 4.3

Constructing a tagger from: eindh.data
  Creating lexicon: eindh.data.lex of 17044 entries.
```

The first data structure created by the tagger is a frequency-sorted lexicon `eindh.data.lex` with for each word the different tags it was assigned, along with their frequency in the training corpus.

```
  Creating ambitag lexicon: eindh.data.lex.ambi.05
  Creating ambitag translation table: eindh.data.ambi.05
```

An ambitag is a symbolic label defining for a word which different tags it can have according to the corpus. In the ambitag lexicon, each word is associated with its corresponding ambitag, represented in two forms: a letter code generated by the tagger, and a string of tags separated by hyphens. Some frequency-based smoothing is implemented in this approach: whenever a word–tag combination occurs less than a given percentage (5% by default) of the word's total frequency, it is not included in the ambitag. The parameter `-%` < *percentage* > can be used to modify this threshold. The ambitag translation table is used for associating the generated ambitag letter codes with the more understandable hyphenated representation.

```
Creating list of most frequent words: eindh.data.top100
```

Next, the tagger generator creates a list with (by default) the 100 most frequent words in the corpus. Only words in this list will be used when the symbols *w*, *W* are used in the `-p`, `-P` patterns. The number of most frequent words can be modified with the parameter `-M` < *number* >. All words *not* in the most-frequent-words list are transformed into special symbols: `HAPAX-`< *code* >, where < *code* > is either `0`, or a combination of `H`, `C`, and `N`. `H` indicates that the word contains a hyphen, `C` denotes that the word is capitalised, and `N` indicates that the word contains a number. `HAPAX-HCN` would for example be the transformed code for the word "B-52".

```
Create known words case base
  Timbl options: ' -a IGTREE -vS   -H'
  Algorithm = IGTREE

  Processing data from the file eindh.data.................................
    ready: 95548 words processed.
  Creating case base: eindh.data.known.ddfa
  Deleted intermediate file: eindh.data.known.inst.ddfa
```

In this part of the tagger generation process, the case base for known words is generated. To do this, TiMBL is used, by default with the options shown above (IGTREE algorithm for known words). The process consists of two steps. First, instances are created using the specified information sources for known words (as indicated in `-p`), then the case base is generated from that (which may imply a significant storage reduction, depending on the TIMBL options used). Finally, the intermediate file with instances is deleted — this can be overruled with the option `-X`.

```
Create unknown words case base
  Timbl options: ' -a IB1 -vS  -H'
  Algorithm = IB1

  Processing data from the file eindh.data.................................
    ready: 95548 words processed.
  Creating case base: eindh.data.unknown.psssdFa
  Deleted intermediate file: eindh.data.unknown.inst.psssdFa
```

This part of the screen log describes the process of the creation of the case base for unknown words (for which no ambitag is available, and for which the inclusion of the word itself in the input is pointless). It is parallel to the procedure for known words, but it uses information sources specified in the `-P` pattern, and uses as default TIMBL settings the IB1-GR algorithm (the overlap metric with gain ratio feature weighting).

```
Created settings file 'eindh.data.settings'
```

```
Ready:
  Time used: 83
  Words/sec: 2302
```

The tagger generation process ends with some information about the real time needed to construct the tagger (total time used and number of words per second), and with the construction of a settings file, which will be used by the Mbt executable to use the tagger on new data. E.g. for our toy corpus, the settings file looks like this:

```
e <utt>
l eindh.data.lex.ambi.05
k eindh.data.known.ddfa
u eindh.data.unknown.psssdFa
r eindh.data.ambi.05
p ddfa
P psssdFa
O K: -a IGTREE -vS U: -a IB1 -vS
L eindh.data.top100
```

These settings can be modified by the user to a certain extent, which is especially useful for experimenting with different TiMBL options. However, this is not advised unless the user knows what s/he is doing (not all TiMBL options and Mbtg settings can be modified given the datastructures created).

Some Mbtg parameters have remained undiscussed until now.

- In the construction of the unknown words case base, instances are created only for these words which are relatively infrequent (the basic idea here is that unknown words will behave similarly to infrequent known words). The option `-n` $< n >$ determines the maximum frequency a word can have in the training corpus to use it (and its context) in the creation of the unknown words case base (default = 5).
- By default, the tagger generator expects the symbol `<utt>` on a line to indicate the boundary between two sentences. Using the option `-e` $<$ *string* $>$, this default behaviour can be modified. With `-e NL`, one or more empty lines between word-tag lines would be interpreted as an utterance marker.

3.2 `Mbt`, the tagger

Given that `Mbtg` was used to generate data files and a settings file defining a memory-based tagger, `Mbt` can be used to tag text. For example, continuing our example:

```
Mbt -s eindh.data.settings -t <testfile>
```

This starts the memory-based tagger, and sends the tagged output to standard output. The testfile should be tokenized (i.e. punctuation marks should be separated from the words). E.g. we included a short tokenized test file in the distribution (file `test.tok`).

```
> Mbt -s eindh.data.settings -t test.tok > test.out
```

```
Memory Based Tagger Version 1.0 (beta)
  (c) ILK and CNTS 1998 - 2002.
  Induction of Linguistic Knowledge Research Group, Tilburg University
  Centre for Dutch Language and Speech, University of Antwerp

  Based on Timbl version 4.3
  Sentence delimiter set to '<utt>'
  Beam size = 1
  Known Tree, Algorithm = IGTREE
  Unknown Tree, Algorithm = IB1

  Reading the ambitags from: eindh.data.ambi.05...ready, (78 tags).
  Reading the lexicon from: eindh.data.lex.ambi.05...ready, (17045 words).
  Reading frequent words list from: eindh.data.top100...ready, (100 words).
  Reading case-base for known words from: eindh.data.known.ddfa... ready.
  Reading case-base for unknown words from: eindh.data.unknown.psssdFa... ready.

Processing data from the file test.tok. ready.

Done:
  32 words processed.
  Known   words: 20
  Unknown words: 12 (37.5 %)
  Total        : 32
  Time used: 2
  Words/sec: 16
```

The file `test.out` will now contain the tagged text:

Het/Art Centrum//N voor/Prep Nederlandse/Adj Taal//Adv en/Conj Spraak//N en/Conj de/Art Inductie//N van/Prep Linguistische//Adj Kennis//N onderzoeksgroep//V werken/V aan/Adv geheugengebaseerd//V leren/V voor/Adv taaltechnologie//N ./Punc Al/Adv in/Prep 1994//Num werd/V er/Adv gewerkt/V aan/Prep de/Art memory-based//N tagger//N ./Punc

In producing output, the tagger concatenates tags to words, separated by either a single slash (/) when the word is in the lexicon, or a double slash (//) when the unknown words case base was used to predict the tag.

With `-T` < *testfile* >, files in the format of the tagger generation input (one word per line, two columns for word and corresponding tag) can be used. In that case, accuracy figures are computed, and the output, again sent to standard output, consists of the two input columns, separated by a / or //, and an additional column with the predicted tag. This way, a constructed tagger can be evaluated. E.g.,

```
Mbt -s eindh.data.settings -T eindh.test > eindh.test.out
```

will send the tagged text (four columns) to `eindh.test.out`, and compute total, known word, and unknown word accuracies for the test data, as well as tagging speed indications.

```
Memory Based Tagger Version 1.0
...
Done:
  4427 words processed.
  Known   words: 3635   correct from 3810 (95.4068 %)
  Unknown words: 448    correct from 617 (72.6094 %)
  Total        : 4083   correct from 4427 (92.2295 %)
  Time used: 3
  Words/sec: 1475
```

When neither `-t` nor `-T` is specified, Mbt processes data from standard input.

An additional parameter which can be used is the size of a beam search for the most probable sequence of tags for a complete sentence. In this case, the tagger produces a distribution of probabilities of tags for each word (rather than just the best one), and applies a beam search to look for the most probable sequence of tags at a global sentence level, as opposed to a deterministic decision for each word in the sentence independently. As a default, beam search is off (i.e. set to beam size 1).

3.2.1 Server options

Sometimes it may take a long time to load all the data files necessary to start tagging, for instance when IB1 is used as the TIMBL classifier and a very large corpus was used for tagger generation. In applications where small files of text or even individual sentences have to be tagged at different points in time, e.g. on demand in a web demo, it is more efficient to use Mbt as a server. With the option `-S` $<$ *portnumber* $>$, a tagger server can be set up. E.g.

```
Mbt -s eindh.data.settings -S 1999 &
```

This sets up a tagger server on port 1999 of the local host. When running, the server can be accessed via telnet to this port through a special purpose client application. To prevent too many simultaneous clients from calling the server (e.g. in a demo environment), the option `-C` $<$ *number* $>$ can be used to restrict the number of clients that can communicate with the server at the same time (default 10).

3.2.2 Overwriting settings file options

Instead of using the settings file, it is also possible to specify each data file separately with the parameters `-l`, `-r`, `-k`, `-u`, and `-L`. These options can also be used to override the values given in a settings file.

3.3 Timbl options

To construct a memory-based tagger and use it, TiMBL is used. Various algorithms, parameters, and metrics are implemented in TiMBL. By default, the memory-based learning algorithm used to train and test a tagger is IGTREE for the known words, and IB1 with the overlap metric with gain ratio feature weighting, and k (the number of nearest neighbors) set to 1 for the unknown words.

This is most likely not the best (or even a good) setting for the particular corpus you want to build a tagger from. With the `-O` $<$ *string* $>$ parameter of Mbtg, a string can be provided with settings for the TiMBL algorithm to be used in tagger generation and tagging. The string has the following structure:

```
-O "K: <known words options> U: <unknown words options>"
```

or when the options are the same for known and unknown word tagging:

```
-O "<options>"
```

The following is an example:

```
-O "K: -a1 -w1 -vS U: -a0 -w1 -mM -k9 -vS"
```

This means that IGTREE (features sorted according to information gain, and classification through decision-tree traversal) will be used for the known words, and IB1 with gain ratio feature weighting, modified value difference metric, and 9 nearest neighbours, will be used for the unknown words. The `-Vs` setting mutes the output of the TIMBL classifiers. When this option is left out, the TiMBL output is also printed to screen. Using these particular example settings increase overall accuracy on the `eindh.test` file to 92.86%.

Please consult the reference guide provided with the TiMBL package to experiment with other parameter settings. Note that changing some TiMBL parameter must be accompanied by a complete regeneration of the tagger with `Mbtg`. Also, changing the verbosity settings of TiMBL may have the effect that more output is generated to the standard output streams than what is documented here.

References

Daelemans, W. 1995. Memory-based lexical acquisition and processing. In P. Steffens, editor, *Machine Translation and the Lexicon*, Lecture Notes in Artificial Intelligence. Springer-Verlag, Berlin, pages 85–98.

Daelemans, W., J. Zavrel, P. Berck, and S. Gillis. 1996. MBT: A memory-based part of speech tagger generator. In E. Ejerhed and I. Dagan, editors, *Proc. of Fourth Workshop on Very Large Corpora*, pages 14–27. ACL SIGDAT.

Daelemans, Walter, Jakub Zavrel, Ko van der Sloot, and Antal van den Bosch. 2002. TiMBL: Tilburg memory based learner, version 4.3, reference guide. ILK Technical Report 02-10, Tilburg University. available from http://ilk.uvt.nl/downloads/pub/papers/ilk.0210.ps.

Uit den Boogaard, P.C. 1975. *Woordfrequenties in geschreven en gesproken Nederlands*. Scheltema en Holkema, Utrecht, the Netherlands.

Zavrel, J. and W. Daelemans. 1999. Recent advances in memory-based part-of-speech tagging. In *VI Simposio Internacional de Comunicacion Social, Santiago de Cuba*, pages 590–597.