MBT: Memory-Based Tagger

version 3.1

Reference Guide

ILK Technical Report – ILK 07-09

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

Induction of Linguistic Knowledge Research Group
Department of Communication and Information Sciences
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]

December 11, 2007

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.

POS tagging is a prototypical instance of the more general sequence tagging task, in which a sequence of words is mapped to any same-length sequence of tags, with a one-to-one mapping between words and tags. This setup can apply to diverse tasks such as syntactic base-phrase chunking (Tjong Kim Sang and Buchholz, 2000), named-entity recognition, and information extraction.

Memory-Based Tagging is an approach to sequence 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 (Daelemans and Van den Bosch, 2005). Since 1998, we have made available TiMBL, a flexible software tool incorporating an extensive and growing family of memory-based learning algorithms and associated metrics (Daelemans et al., 2007). The most recent version of software and documentation are available under the GNU General Public License from `http://ilk.uvt.nl/timbl`. To use this software for POS Tagging is not entirely trivial, 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 and Van den Bosch, 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's Faculty of Arts, the Flemish National Science Foundation (FWO), and the University of Antwerp Research Council.

The current release (version 3.1) uses TiMBL version 6.1, and will not work with older versions of TiMBL. Note that you have to download and install TiMBL 6.1 separately to be able to compile the tagging software. See `http://ilk.uvt.nl/timbl`.

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, copy, and modify 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 and Van den Bosch, 1996; Zavrel and Daelemans, 1999; Daelemans and Van den Bosch, 2005). The first two papers are also included in the 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 the reference guide that accompanies the software (Daelemans et al., 2007).

Chapter 1

GNU General Public License

MBT is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 3 of the License, or (at your option) any later version.

MBT is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with MBT. If not, see <http://www.gnu.org/licenses/>.

In publication of research that makes use of the Software, a citation should be given of: *"Walter Daelemans, Jakub Zavrel, Antal van den Bosch and Ko van der Sloot (2007). MBT: Memory-Based Tagger, Reference Guide. ILK Technical Report 07-09,*
Available from `http://ilk.uvt.nl/downloads/pub/papers/ilk.0709.pdf`"

For information about commercial licenses for the Software, contact `timbl@uvt.nl`, or send your request to:

Prof. dr. Walter Daelemans
CNTS - Language Technology Group
Department of Linguistics / University of Antwerp
Universiteitsplein 1
B-2610 Wilrijk (Antwerp)
Belgium
Email: walter.daelemans@ua.ac.be

Chapter 2

Installation

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

`http://ilk.uvt.nl/mbt`

Following the links from that page, you can download the file `mbt-3.1.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, unpack the downloaded file:

```
> tar zxf mbt-3.1.tar.gz
```

This will make a directory `mbt-3.1` under your current directory.

Alternatively you can do:

```
> gunzip mbt-3.1.tar.gz
```

and unpack the tar archive:

```
> tar -xvf mbt-3.1.tar
```

Change directory to the mbt-3.1 directory and configure the package by typing

```
> cd mbt-3.1
```

```
> ./configure --prefix=<timbl_location>
```

the prefix should be the directory where Timbl is installed previously. Mbt will be installed there also.

Note: It is possible to install Mbt in a different location, but there is no need, and it makes things more complicated. It involves using the `--with-timbl=` option in configure.

Anyway: after `configure` you can build Mbt:

```
> make
```

and (as recommended) install:

```
> make install
```

If the process was completed successfully, you should now have executable files named `Mbtg` and `Mbt` in the installation directory `<timbl_location>/bin`, and a static library `libMbt.a`

in the directory `<timbl_location>/lib`.

Within the `<timbl_location>` directory a subdirectory is also created: `share/doc/mbt` where the Mbt3.1 documentation can be found, and which in turn contains a subdirectory `examples` with example data files.

Mbt should now be ready for use. If you want to run the examples and demos from this manual, you should act as follows:

- Be sure to add `<timbl_location>/bin` to your PATH. (you probably already did that when installing Timbl.)
- copy all the files from `<timbl_location>/share/doc/mbt/examples` to some working location.
- and test:

  `cd` to the working location, and then

  Generate a tagger for the eindhoven corpus.

  ```
  Mbtg -T eindh.data
  ```

  This will create some datafiles and a settingsfile, eindh.data.settings

  And then tag the testset:

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

That's all!

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

Chapter 3

Tutorial and Reference

Memory-based tagging is based on the idea that words occurring in similar contexts will have the same 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 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 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 and Van den Bosch, 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 part-of-speech 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 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

The default values for `-p` and `-P` are `ddfa` and `dFapsss` respectively.

3.1.2 Running the tagger-generator

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

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

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 3.1
  (c) ILK and CNTS 1998 - 2007.
  Induction of Linguistic Knowledge Research Group, Tilburg University
  Centre for Dutch Language and Speech, University of Antwerp

  Based on Timbl version 6.1 (release)

Constructing a tagger from: eindh.data
  Creating lexicon: eindh.data.lex of 17040 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 its frequency in the training corpus.

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

An ambitag is a symbolic label defining for a word the 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.

```
  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  -FColumns +vS '
    Algorithm = IGTREE

    Processing data from the file eindh.data.............................
      ready: 95566 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 -FColumns +vS '
    Algorithm = IB1

    Processing data from the file eindh.data.............................
      ready: 95566 words processed.
    Creating case base: eindh.data.unknown.dFapsss
    Deleted intermediate file: eindh.data.unknown.inst.dFapsss
```

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-IG algorithm (the overlap metric with gain ratio feature weighting).

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

Ready:
  Time used: 10
  Words/sec: 9556
```

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.dFapsss
p ddfa
P dFapsss
O K: -a IGTREE U: -a IB1
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 relatively infrequent words (the basic idea being 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 in its context in the creation of the unknown words case base (the default value is 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. Special predefined values are `EL` and `NL`: with `-e EL`, one or more empty lines between word-tag lines are interpreted as an utterance marker, and with `NL` each newline is interpreted as an utterance marker.

3.1.3 Using data with user-provided features

Although `Mbtg` offers various feature construction methods, there are cases in which a sequence tagging task involves features that are not computable from the literal wordforms or from a local contextual window. For example, in named-entity recognition, binary gazetteer features may mark the membership of a word of a typed gazetteer list of personal names. To allow externally provided features (and thus some more flexibility), it is possible to have `Mbtg` include such user-provided features in the generation of the tagger. If `Mbtg` is invoked with `-E <enriched training file>` instead of `-T <tagged training file>`, then the user is free to include extra features after the first column (i.e. the word) and the second column (i.e. the tag) in the tagged training data.

For example, the `eindh.data` file for training a POS tagger may be enriched with gazetteer information that signifies the potential membership of a word in a typed gazetteer list. A line from `eindh.data` may be expanded as follows:

```
<utt>
Engeland  LOC  N
hoopt     ---  V
```

```
nog         ---  Adv
voor        ---  Prep
1973        TIM  Num
lid         ---  N
te          ---  Prep
zijn        ---  V
van         ---  Prep
de          ---  Art
Euromarkt   ORG  N
.           ---  Punc
```

In this example, the classification task thus remains the POS tagging task, but now in addition to all the local context and wordform features (determined by the `-p` and `-P` settings), this extra feature is added to the input feature vector of all cases for both the known-word and the unknown-word sub-taggers. The number of user-provided features is unbounded.

A consequence of using `-E` is that when the generated tagger is used, new texts that are to be tagged need to include the same user-provided features. The next section describes how `Mbt`, the tagger, operates in general; in Subsection 3.2.3 it is mentioned how data with user-provided features can be tagged.

3.2 `Mbt`, the tagger

After `Mbtg` is used to generate data files and a settings file defining a memory-based tagger, `Mbt` can be used to actually tag text.

3.2.1 Running the tagger

The following commandline invokes `Mbt`, the tagger:

```
Mbt -s eindh.data.settings -t <tokenized text file>
```

This commandline makes `Mbt` send its result, which consists of the text in the tokenized text file argument augmented with tags attached to each token in the file, to standard output. The testfile should be tokenized (i.e. punctuation marks should be separated from the words). Testfiles do not need to have all words on separate lines. By default, `Mbt` expects to find `<utt>` as sentence boundary marker, but as with `Mbtg`, the user can specify other sentence boundary markers with `-e` $< string >$. In the case of `Mbt` $< string >$ can also be EL, for empty line, or NL, for newline, to process text in which each sentence is printed on one line, ending with a newline. E.g. we included a short tokenized test file in the distribution (file `test.tok`). This file contains an empty line between the two sentences:

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

Memory Based Tagger Version 3.1
  (c) ILK and CNTS 1998 - 2007.
  Induction of Linguistic Knowledge Research Group, Tilburg University
  Centre for Dutch Language and Speech, University of Antwerp

  Based on Timbl version 6.1 (release)
```

```
  Reading the lexicon from: eindh.data.lex.ambi.05...ready, (17040 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.dFapsss... ready.
  Sentence delimiter set to 'EL'
  Beam size = 1
  Known Tree, Algorithm = IGTREE
  Unknown Tree, Algorithm = IB1

Processing data from the file test.tok:


Done:
  32 words processed.
  Known   words: 20
  Unknown words: 12 (37.5 %)
  Total        : 32
  Time used: 1
  Words/sec: 32
```

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 3.1
...
Done:
  4424 words processed.
  Known   words: 3730   correct from 3843 (97.0596 %)
  Unknown words: 466    correct from 581 (80.2065 %)
  Total        : 4196   correct from 4424 (94.8463 %)
  Time used: 1
  Words/sec: 4424
```

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 nearest-neighbor-based tag distribution for each word (rather than just the best one), and applies a beam search to look for the maximally likely 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).

Not that in general, 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.

3.2.2 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).

3.2.3 Tagging texts with user-provided features

When `Mbtg` is invoked to include user-provided features with the `-E` option (cf. subsection 3.1.3, then necessarily `Mbt`, when run on that generated tagger, will count on the user-provided features to be present in the test data to be tagged. For this purpose `Mbt` uses the same `-E` option. When invoked with `Mbt -E <enriched tagged test file>`, the test file is supposed to contain the usual word and class columns, and in between these two columns `Mbt` expects to find the same number of user-provided features as given in `Mbtg`'s original training data. A line from an example training file is shown in Subsection 3.1.3; the test file will need to have the same format. If the tags in the final column of the enriched test file are yet unknown, then the final column could simply be filled with dummy class values, such as question marks.

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 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 U: -a0 -w1 -mM -k9 -dIL"
```

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, the modified value difference metric, 9 nearest neighbours, and inverse-linear distance weighting will be used for the unknown words. Using these particular example settings increase overall accuracy on the `eindh.test` file from 94.8% to 95.5%.

Please consult the reference guide provided with the TiMBL package (Daelemans et al., 2007) 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*, volume 898 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag, Berlin, pages 85–98.

Daelemans, W. and A. Van den Bosch. 1996. Language-independent data-oriented grapheme-to-phoneme conversion. In J. P. H. Van Santen, R. W. Sproat, J. P. Olive, and J. Hirschberg, editors, *Progress in Speech Processing*. Springer-Verlag, Berlin, pages 77–89.

Daelemans, W. and A. Van den Bosch. 2005. *Memory-based language processing*. Cambridge University Press, Cambridge, UK.

Daelemans, W., J. Zavrel, K. Van der Sloot, and A. Van den Bosch. 2007. TiMBL: Tilburg Memory Based Learner, version 6.1, reference guide. Technical Report ILK 07-07, ILK Research Group, Tilburg University.

Tjong Kim Sang, E. and S. Buchholz. 2000. Introduction to the CoNLL-2000 shared task: Chunking. In *Proceedings of CoNLL-2000 and LLL-2000*, pages 127–132.

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*, pages 590–597.