# ACOPOST: User manual

## Version 1.8.4

Ingo Schröder
ixs@users.sourceforge.net

## Contents

# 1 Introduction

This document describes how to use the ACOPOST program suite.

ACOPOST is a collection of part-of-speech tagging algorithms, each originating from a different machine learning paradigm.

- t3 is a trigram tagger based on Markov models.
- met is a maximum entropy inspired tagger.
- tbt is an error-driven learner of transformation rules.
- et is an example-based tagger.

An evaluation of the individual part-of-speech taggers and of novel combination techniques can be found in an accompanying technical report [Schröder, 2002].

# 2 Installation

ACOPOST is available under the GNU public license[1] from the project homepage hosted at `http://www.sourceforge.net`.

ACOPOST comes as a gzipped tar archive of the source code named acopost-$x.y.z$.tar.gz where $x.y.z$ is the version number. No pre-compiiled binaries are available but don't worry: Compiling is easy. You only need a C compiler (gcc is recommended) and the `make` program which are both most probably already installed on your machine if you're using UNIX.[2] Some scripts use the Perl programming language[3] which you want to have installed anyway.

[1] See `http://www.gnu.org/licenses/gpl.html`.
[2] I have not tried to compile ACOPOST on MS Windows but I am interested in reports from Windows users.
[3] See `http://www.perl.org/` and `http://www.perl.com/`.

Find a convenient place in your directory tree and unzip the archive which unpacks into a new directory acopost-$x.y.z$:

```
PROMPT> gunzip -c acopost-1.8.4.tar.gz | tar fxv -
acopost-1.8.4/
acopost-1.8.4/src/
acopost-1.8.4/src/Makefile
acopost-1.8.4/src/array.c
...
```

The fresh directory contains at least the following files and directories:

- Text file `README` with a short intro and latest changes.
- Directory `bin` which contains the Perl scripts and where the binaries are installed after compilation.
- Directory `src` which contains the C files.
- Directory `docs` which contains the documentation, this user guide and a technical report [Schröder, 2002].
- Directory `examples` which contains some example files.

To compile, change to the `src` directory and type `make`. If everything works out ok, issue the command `make install` which installs the binaries into the directory `../bin`. Congratulations! You're done.

If something goes wrong, try to fix it by adpating the `Makefile` or the source code. Don't forget to tell me about your problems so that I can provide a better solution with the next release.

You can now chose to add the `bin` directory as a full path to your `PATH` variable, to move/copy all binaries from the `bin` directory to a directory already in your `PATH` variable or simply decide to always use the full path to an ACOPOST program.

# 3 File formats

I tried to keep everything as simple as possible in order to be able to use other tools on the corpora, e.g., UNIX tools like `grep`, `sed`, `wc` etc. or Perl. Therefore, I chose line-based formats for the corpora, i.e., each line of texts (separated by the newline character `\n`) holds exactly one sentence. The items in a sentence are separated by one or more white space characters, i.e., tabular `\t` or space characters. Punctuation marks should be separated from preceding words.

ACOPOST uses two file formats for text: raw and cooked.

- Raw text follows the line-based format described above but doesn't contain any additional information. Here's an example from the Wall Street Journal corpus [Marcus et al., 1993]:

  ```
  The rest went to investors from France and Hong Kong .
  ```

- Cooked text contains the part-of-speech tags for the words. The tag immediately follows the word and the two are separated by one or more white space characters, i.e., in the same way ajacent words are separated. Of course, a line of cooked text must always contain an even number of items. Here's the same example as above as cooked text:

  ```
  The DT rest NN went VBD to TO investors NNS from IN France NNP and CC Hong NNP Kong NNP . .
  ```

  Note that the period functions as both a word and a tag symbol in the Wall Street Journal corpus.

The ACOPOST program suite contains Perl scripts which convert from and into different formats, e.g., `wsj2cooked.pl` (cf. Section 5.13), `tt2cooked.pl` (cf. Section 5.12) and `cooked2tt.pl` (cf. Section 5.4).

The individual taggers use additional data formats to store the model information. These formats have been chosen to be human-readable but completely understanding them requires deep insights into the tagging algorithms. The formats of model file might change between releases.

The format of the lexicon files is also line-based. Each line lists the word form and the possible tags including the tag counts.

```
WORDFORM TAG1 TAGCOUNT1 TAG2 TAGCOUNT2 ...
```

An older format allowed for an optional word count after the word form but since this information is redundant it is deprecated.

# 4 Tutorial

Nothing yet.

# 5 Program references

Note that not all programs in the `bin` directory are described here. This may be the case due to one of the following reasons.

- The program is considered to be of marginal importance.
- It hasn't reached a stable state.
- It's obsolete.

## 5.1 complementary-rate.pl

### 5.1.1 Purpose

Report the complementary error rate [Brill and Wu, 1998] of two versions of a tagged corpus.

### 5.1.2 Usage

```
complementary-rate.pl [-h] ref a b
```

| | |
|---|---|
| `-h` | display short help text and exit |
| `ref` | reference corpus in cooked format |
| `a` | first tagged corpus in cooked format |
| `b` | second tagged corpus in cooked format |

### 5.1.3 Example

```
PROMPT> ~/acopost/bin/complementary-rate.pl 0.ref 0.t3 0.tnt
accuracy A  96.221% 16651 654        0
accuracy B  96.689% 16732 573
comp(A,B)   22.783% 505 654
comp(B,A)   11.867% 505 573
PROMPT>
```

## 5.2 cooked2lex.pl

### 5.2.1 Purpose

Convert a corpus in cooked format to a lexicon.

### 5.2.2 Usage

```
cooked2lex.pl [-h] [-c] < in.cooked > out.lex
```

| | |
|---|---|
| `-h` | display a short help text and exit |
| `-c` | output deprecated word count after the word form (cf. Section 3) |

### 5.2.3 Example

```
PROMPT> cooked2lex.pl < negra.cooked > negra.lex
20602 sentences
55 tags 51272 types 355096 tokens
  1     49189  95.937%    238545  67.178%
  2      1884   3.675%     45586  12.838%
  3       164   0.320%     46789  13.176%
  4        32   0.062%     20090   5.658%
  5         1   0.002%      2715   0.765%
  6         1   0.002%      1363   0.384%
  7         1   0.002%         8   0.002%
Mean ambiguity A=1.611544
```

```
Entropy H(p)=4.273873
PROMPT>
```

## 5.3 cooked2ngram.pl

### 5.3.1 Purpose

Convert a corpus in cooked format to a file containing counts for tag $n$-grams.

### 5.3.2 Usage

```
cooked2ngram.pl [-h] < in.cooked > out.ngram
```

`-h` display a short help text and exit

### 5.3.3 Example

```
PROMPT> cooked2ngram.pl < corpus.cooked > corpus.ngram
```

## 5.4 cooked2tt.pl

### 5.4.1 Purpose

Convert a corpus in cooked format to a corpus in the format [Brants, 1997] used by the TnT tagger package [Brants, 2000].

### 5.4.2 Usage

```
cooked2tt.pl [-h] < in.cooked > out.tt
```

`-h` display a short help text and exit

### 5.4.3 Example

```
PROMPT> cooked2tt.pl < negra.cooked > negra.tt
20602 sentences
PROMPT>
```

## 5.5 cooked2wtree.pl

### 5.5.1 Purpose

Convert a corpus in cooked format to a weighted tree [Daelemans et al., 1997, Schröder, 2002] for use in example-based disambiguation.

**Warning: the current implementation is far from efficient. Training on the Wall Street Journal corpus requires large amounts of main memory. Be careful!**

### 5.5.2 Usage

```
cooked2wtree.pl OPTIONS f-file < in.cooked > out.wsj
```

where `f-file` is a feature file (see below) and `OPTIONS` can be one or more of:

| | |
|---|---|
| `-a a` | is the minimal word count that a word must have to be considered (default: unlimited) |
| `-b b` | is the maximal word count that a word must have to be considered (default: unlimited) |
| `-d` | debug flag |
| `-e e` | file with tags to be excluded (default: exclude none) |
| `-i i` | file with tags to be explicitely included (default: include all) |
| `-h` | display a short help text and exit |
| `-r r` | rare word count threshold |
| `-w w` | word rank threshold (default: 100) |

### 5.5.3 Features

Features describe characteristics of tagging context that can be used for the tagging decision. The following features are allowed:

- `TAG[`*relpos*`]`: Include the tag at the relative position *relpos* as a criterion for the decision. For example, `TAG[-1]` means the tag of the word immediately to the left. Of course, *relpos* must be negative since the tags to the right are not yet known.
- `CLASS[`*relpos*`]`: Use the ambiguity class at the relative position *relpos* as a criterion. For example, `CLASS[1]` considers the ambiguity class of the word to the right of the current word.
- `WORD[`*relpos*`]`: Use the word form at the relative position *relpos* as a criterion. Note that only frequent words (see options `-r` and `-w`) are used. For rare words the artifical token `*RARE*` is substituted.
- `LETTER[`*relpos, index*`]`: Use the letter at position *index* of the word at the relative position *relpos* as a criterion. Negative values of *index* count from the end of the word backwards.
- `CAP[`*relpos*`]`: Use the binary answer whether the word at the relative position *relpos* is capitatized as a criterion.

- `HYPHEN[`*relpos*`]`: Use the binary answer whether the word at the relative position *relpos* contains a hyphen as a criterion.
- `NUMBER[`*relpos*`]`: Use the binary answer whether the word at the relative position *relpos* contains a digit as a criterion.
- `INTER[`*relpos*`]`: Use the binary answer whether the word at the relative position *relpos* contains an interpunctuation mark as a criterion.

The directory `examples/et` contains example feature files.

### 5.5.4 Example

```
PROMPT>
PROMPT>
```

## 5.6 et

### 5.6.1 Purpose

Assign tags to a natural language text in raw format using the example-based paradigm [Schröder, 2002, Section 5.4].

Note that the learning phase is done by the Perl script `cooked2wtree.pl` (cf. Section 5.5).

### 5.6.2 Usage

```
et OPTIONS knownwtree unknownwtree lexiconfile [in.raw] > out.cooked
```

where `knowntree` is a weighted tree file generated by `cooked2wtree.pl` (cf. Section 5.5) for known words, `unknowntree` is a weighted tree file for unknown words and `lexiconfile` is a lexicon file generated by `cooked2lex.pl` (cf. Section 5.2). If the input file `in.raw` is omitted standard input is used. `OPTIONS` can be:

`-v v` verbosity (default: 1)

### 5.6.3 Example

```
PROMPT> cooked2lex.pl < train.cooked > train.lex
...
PROMPT> cooked2wtree.pl -a 3 known.etf < train.cooked > known.wtree
...
PROMPT> cooked2wtree.pl -b 2 unknown.etf -e closed-class-tags < train.cooked > unknown.wtree
...
PROMPT> et known.wtree unknown.wtree train.lex < test.raw > test.et
[        0 ms::1]
```

```
[        0 ms::1] Example-based Tagger (c) Ingo Schröder, schroeder@informatik.uni-hamburg.de
[        0 ms::1]
[     2240 ms::1] read wtree with 156173 nodes from "known.wtree"
[     3580 ms::1] read wtree with 116334 nodes from "unknown.wtree"
[     3590 ms::1] done
PROMPT> evaluate.pl test.cooked test.et
2060 sentences
             test.et    33990     1434  95.952%
```

## 5.7 evaluate.pl

### 5.7.1 Purpose

Report tagging accuracy on sentence level, for unknown, known and all words.

### 5.7.2 Usage

`evaluate.pl` [-h] [[-i] -l l] [-v] ref t1 ...

| | |
|---|---|
| `-h` | display short help text and exit |
| `-i` | use case-insensitive lexicon |
| `-l l` | use lexicon `l` |
| `-v` | be verbose |
| `ref` | reference corpus in cooked format |
| `t1` | tagged corpus in cooked format |

### 5.7.3 Example

```
PROMPT> evaluate.pl 0.ref 0.t3 0.tnt
1002 sentences
                0.t3    16651      654  96.221%
               0.tnt    16732      573  96.689%
PROMPT>
```

## 5.8 majority-voter.pl

### 5.8.1 Purpose

Report how often different numbers of different taggers have tagged words correctly. See Schröder [2002]. This immediately tells one how efficient a parallel combination of different taggers can be. Four numbers are given in each line: The number of taggers that were correct, the percentage of words, the accumulated percentage of words and the mean ambiguity of tags if all emitted tags are counted.

### 5.8.2 Usage

`majority-voter.pl` [-h] ref t1 t2 ...

| | |
|---|---|
| `-h` | display short help text and exit |
| `ref` | reference corpus in cooked format |
| `t1` | first tagged corpus in cooked format |
| `t2` | second tagged corpus in cooked format |

### 5.8.3 Example

```
PROMPT> majority-voter.pl 0.ref 0.t3 0.tbt 0.et 0.met
2061 sentences
35674 words
4:  92.928%  92.928% 0.937658
3:   3.493%  96.420% 0.983041
2:   1.343%  97.763% 1.010988
1:   1.090%  98.854% 1.068313
0:   1.146% 100.000%
PROMPT>
```

## 5.9 met

### 5.9.1 Purpose

Nothing yet.

### 5.9.2 Usage

`met OPTIONS modelfile [inputfile]`

where `modelfile` is a trained or a new model file and `inputfile` is either a corpus in cooked format (for training) or in raw format (for tagging). `OPTIONS` can be one or more of the following:

| | |
|---|---|
| `-b b` | beam factor (default: 1000) for viterbi search or n-best width (default: 5) for n-best search |
| `-c c` | command mode, "tag", "train" or "test" |
| `-d d` | dictionary file |
| `-f f` | threshold for feature count (default: 5) |
| `-h` | display short help and exit |
| `-i i` | maximum number of iterations (default: 100), training only |
| `-m m` | probability threshold (default: 1.0) |
| `-n` | use n-best instead of viterbi |
| `-p p` | UNIX priority class (default: 19) |
| `-r r` | rare word threshold (defualt: 5) |
| `-s` | case sensitive dictionary |
| `-t t` | minimum accuracy improvement per iteration (default: 0.0), training only |
| `-v v` | verbosity (default: 1) |

### 5.9.3 Example

```
PROMPT> met -c test -d train.lex train.model.met < test.cooked
[        0 ms::1] running as test
[        0 ms::1] using test.lex as dictionary file
[     1390 ms::1] read 54 tags, 40690 predicates and 83343 features
[     2090 ms::1] read 45779 lexicon entries, discarded 2237 entries
[    24620 ms::1] 35674 (35257 pos 417 neg) words tagged, accuracy  98.831%
PROMPT>
```

## 5.10 t3

### 5.10.1 Purpose

Assign tags to a natural language text in raw format using the Viterbi algorithm based on a hidden Markov model (HMM). The model information is extracted from a tag trigram file and a lexicon file.

Note that the learning phase is very easy for HMMs. For that reason, the training phase is done by the Perl script `cooked2ngramn.pl` (cf. Section 5.3).

### 5.10.2 Usage

```
t3 OPTIONS modelfile lexiconfile [in.raw] > out.cooked
```

where `modelfile` is a tag trigram file generated by `cooked2ngram.pl` (cf. Section 5.3) and `lexiconfile` is a lexicon file generated by `cooked2lex.pl` (cf. Section 5.2). If the input file `in.raw` is omitted standard input is used. `OPTIONS` can be:

- `-a a` smoothing parameters for transitional probabilities, see Schröder [2002, Section 5.1.1] and Brants [2000] for the default
- `-b b` beam factor (default: 1000), states that are worse by this factor or more than the best state at this time point are discarded
- `-d` debug mode
- `-h` display short help and exit
- `-l l` maximum suffix length for estimating output probability for unknown words (default: 10)
- `-m m` mode of operation (default: 0): 0 means tagging, 1 testing, ...
- `-q` quiet mode of operation
- `-r r` rare word count (default: 1) (for output probabilities)
- `-s s` theta for suffix backoff (default: SD of tag probabilities), see Schröder [2002, Section 5.1.1] and Brants [2000]
- `-t` test mode (reads cooked input)
- `-u` use line-buffered IO for input (default: block-buffered on files)
- `-v v` verbosity (default: 1)
- `-x` case-insensitive suffix tries (default: sensitive)
- `-y` case-insensitive when branching in suffix trie (default: sensitive)
- `-z` use zero probability for unseen transition probabilities (default: 1/#tags)

### 5.10.3 Example

```
PROMPT> cooked2lex.pl < train.cooked > train.lex
...
PROMPT> cooked2ngram.pl < train.cooked > train.ngram
...
PROMPT> t3 train.ngram train.lex < test.raw > test.t3
[        0 ms::1]
[        0 ms::1] Trigram POS Tagger (c) Ingo Schröder, schroeder@informatik.uni-hamburg.de
[        0 ms::1]
[       80 ms::1] model generated from 18541 sentences (thereof 491 one-word)
[       80 ms::1] found 55623 uni-, 74164 bi-, and 92214 trigram counts for the boundary tag
[      210 ms::1] computed smoothed transition probabilities
[     1940 ms::1] built suffix tries with 32602 lowercase and 74242 uppercase nodes
[     1970 ms::1] leaves/single/total LC: 8628 20073 32603
[     2040 ms::1] leaves/single/total UC: 18627 47180 74243
[     4420 ms::1] suffix probabilities smoothing done [theta 7.489e-02]
[    21690 ms::1] done
PROMPT> evaluate.pl test.cooked test.t3
2061 sentences
         test.t3    34547     1127  96.841%
```

## 5.11 tbt

### 5.11.1 Purpose

Nothing yet.

### 5.11.2 Usage

```
tbt OPTIONS rulefile [inputfile]
```

| | |
|---|---|
| `-i i` | maximum number of training iterations (default: unlimited), training only |
| `-l l` | lexicon file (default: none) |
| `-m m` | minimum improvement per training iteration (default: 1), training only |
| `-n n` | rare wore threshold (default: 0) |
| `-o o` | mode of operation (default: 0): 0 tagging, 1 testing, 2 training |
| `-p p` | preload file (default: lexically most probable tag), start from a different initial tagging |
| `-r` | assume raw format for input (default: cooked format), tagging only |
| `-t t` | template file (default: none), training only, see below |
| `-u u` | unknown word default tag (default: most probable tag from lexicon) |
| `-v v` | verbosity (defualt: 1) |

### 5.11.3 Templates

Templates are patterns for rules. The file format is line-based, i. e., one rule per line, empty lines and everything after a hash sign `#` is ignored. The format for a rule or template is as follows:

```
TARGETTAG CONDITION1 CONDITION2 ...
```

where `TARGETTAG` is the new tag for the word under consideration and the conditions are prerequisites for the application of the rule. All conditions must be fulfilled for a rule to trigger. The following types of conditions are allowed:

| | |
|---|---|
| `tag[`*relpos*`]=`*tag* | The current tag of the word at relative position *relpos* is *tag*. |
| `bos[`*relpos*`]` | Begin of sentence marker at relative position *relpos*. |
| `eos[`*relpos*`]` | End of sentence marker at relative position *relpos*. |
| `word[`*relpos*`]=`*word* | The word at relative position *relpos* is *word*. |
| `rare[`*relpos*`]` | The word at relative position *relpos* is rare. |
| `prefix[`*length*`]=`*prefix* | The prefix of length *length* of the current word is *prefix*. |
| `suffix[`*length*`]=`*suffix* | The suffix of length *length* of the current word is *suffix*. |
| `cap[`*relpos*`]=`*mode* | The capitilization of the word at relative position *relpos* is as *mode* which can be:<br>`no` No character is capitilized.<br>`some` Some characters are capitilized.<br>`all` All characters are capitilized. |
| `digit[`*relpos*`]=`*mode* | The word at relative position *relpos* contains digits according to *mode* which can be `no`, `some` or `all` (see `cap`) |

The placeholders *tag*, *word*, *prefix*, *suffix* and *mode* can also be the wildcard symbol `*` in templates. A typical rule template which takes the two preceding tags into account would then be:

```
* tag[-2]=* tag[-1]=*
```

The `examples/tbt` directory contains example template files.

### 5.11.4 Example

```
PROMPT> cat train.rules
$. rare[0]
NN rare[0] digit[0]=no
ADJA rare[0] tag[0]=NN cap[0]=no
VVPP rare[0] tag[0]=ADJA suffix[0]=t
...
PROMPT> tbt -r -l train.lex train.rules < test.raw > test.tbt
Transformation-based Tagger
(c) Ingo Schröder, ingo@nats.informatik.uni-hamburg.de
done
PROMPT> evaluate.pl test.cooked test.tbt
2061 sentences
        test.tbt    34430     1244  96.513%
```

## 5.12 tt2cooked.pl

### 5.12.1 Purpose

Convert a corpus in a format [Brants, 1997] used by the TnT tagger package [Brants, 2000] to a corpus in cooked format.

### 5.12.2 Usage

```
tt2cooked.pl [-h] < in.tt > out.cooked
```

`-h` display a short help text and exit

### 5.12.3 Example

```
PROMPT> tt2cooked.pl < negra.tt > negra.cooked
396309 lines read
PROMPT>
```

## 5.13 wsj2cooked.pl

### 5.13.1 Purpose

Convert a corpus in Wall Street Journal format to cooked format.

### 5.13.2 Usage

```
wsj2cooked.pl < in.wsj > out.cooked
```

### 5.13.3 Example

```
PROMPT> wsj2cooked.pl < corpus.wsj > negra.cooked
PROMPT>
```

# References

Thorsten Brants. The negra export format for annotated corpora (version 3), 1997.

Thorsten Brants. TnT - a statistical part-of-speech tagger. In *Proceedings of the Sixth Applied Natural Language Processing Conference (ANLP-2000)*, Seattle, WA, USA, 2000.

Eric Brill and Jun Wu. Classifier combination for improved lexical disambiguation. In *Proc. Joint Conference COLING/ACL '98*, pages 191–195, Montréal, Canada, 1998.

Walter Daelemans, Antal van den Bosch, and Ton Weijters. Igtree: Using trees for compression and classification in lazy learning algorithms. *Artificial Intelligence Review*, 11:407–423, 1997.

M. Marcus, B. Santorini, and M. Marcinkiewicz. Building a large annotated corpus of English: the Penn Treebank. *Computational Linguistics*, 19(2), 1993. URL `ftp://ftp.cis.upenn.edu/pub/treebank/doc/cl93.ps.gz`.

Ingo Schröder. A case study in part-of-speech tagging using the ICOPOST toolkit. Technical report, Computer Science, University of Hamburg, 2002. URL `http://nats-www.informatik.uni-hamburg.de/~ingo/icopost/`.