TnT — A Statistical Part-of-Speech Tagger

Thorsten Brants
Saarland University, Computational Linguistics
thorsten@coli.uni-sb.de

Contents

More information on TnT is available on the WWW at
`http://www.coli.uni-sb.de/~thorsten/tnt`

License

Permission to use, copy and modify this software and its documentation is granted to non-commercial entities without fee, provided that this license information and copyright notice appear in all copies of the software and the documentation, and provided that all publications of results produced with the help of TnT make a reference to this software.

A "non-commercial entity" is defined within the scope of this license as an educational institution (excluding a commercial training organisation), non-commercial research organisation, registered charity, registered non-profit organisation, or full-time student.

Use of this software by any other person or organisation for any purpose, or the commercial redistribution of this software (by itself or as part of another application) is allowed only under express written permission of the copyright holder.

Disclaimer

TnT and its documentation is provided on an "as is" basis, with no guarantee of its veracity or accuracy. No liability is accepted for any damage caused by its use.

1 What is TnT?

TnT, the short form of *Trigrams'n'Tags*, is a very efficient statistical part-of-speech tagger that is trainable on different languages and virtually any tagset. The component for parameter generation trains on tagged corpora. The system incorporates several methods of smoothing and of handling unknown words.

TnT is not optimized for a particular language. Instead, it is optimized for training on a large variety of corpora. Adapting the tagger to a new language, new domain, or new tagset is very easy. Additionally, TnT is optimized for speed.

The tagger is an implementation of the Viterbi algorithm for second order Markov models (cf. Rabiner, 1989). The main paradigm used for smoothing is linear interpolation, the respective weights are determined by deleted interpolation (Brown, Pietra, deSouza, Lai, & Mercer, 1992). Unknown words are handled by a suffix trie and successive abstraction (Samuelsson, 1993).

The programs are developed under Solaris in ANSI C using the GNU C compiler. All programs described here are command line oriented and require the user to be familar with Unix commands.

2 Installation

The installation on a Unix or Linux platform is easy. Other platforms are currently not supported, but transfer to any platform running the GNU C compiler should be straight forward, since no system specific functions are used.

The programs come in a compressed (`gzip`) archive file (`tar`), by default named `tnt-lin.tar.gz` for the Linux version and `tnt-sol.tar.gz` for the Solaris version. Move this file to a directory of your choice. Here, a sub-directory `tnt` will be created, which contains all the necessary files. This sub-directory is the installation directory and is named ***tnt*** in the rest of this document. Type

```
gzip -dc tnt-lin.tar.gz | tar xvf -
```

or

```
gzip -dc tnt-sol.tar.gz | tar xvf -
```

This uncompresses the archive and extracts all files to the installation directory.

Now, the installation directory contains four executables (which are described in section 4):

```
tnt tnt-diff tnt-para tnt-wc
```

as well as directories for the documentation and the language models.

Add the name of the installation directory (***tnt***) to the `PATH` environment variable to use these four programs from any place in your directory hierarchy, and set the environment variable `TNT_MODELS` to ***tnt***`/models`, so that `tnt` can find the language model files. That's it.

3 File Formats

Usually, there are two types of files the user has to deal with: the untagged input for the tagger and the tagged output of the tagger.

Additionally, we describe the format of the parameter files for lexical and contextual probabilities created by the parameter generation process. Optionally, the user can specify a mapping of tags used by the tagger to an output tagset.

3.1 Format of Untagged Files

Each line starting with two percentage signs (`%%`) marks a comment and is ignored by the programs.

Each token of the text occupies its own line, delimited by a linefeed character (`LF = 0x0a`). The token occupies all characters from the beginning of the line and must not contain white space (`TAB = 0x09, SPC = 0x20`). If the line contains white space, all characters after the first white space character are ignored (including this character). The tokens can be encoded using all characters with codes `0x21...0xFF`.

```
%% Brown Corpus
%% File N11, Sent 3
But
the
day
of
the
deadline
came
and
passed
,
and
the
men
who
had
scoffed
at
the
warnings
laughed
with
satisfaction
.
```

a) untagged format
(one column)

```
%% Brown Corpus
%% File N11, Sent 3
But           CC
the           DT
day           NN
of            IN
the           DT
deadline      NN
came          VBD
and           CC
passed        VBD
,             ,
and           CC
the           DT
men           NNS
who           WP
had           VBD
scoffed       VBN
at            IN
the           DT
warnings      NNS
laughed       VBD
with          IN
satisfaction  NN
.             .
```

b) tagged format
(two columns, separated by white space)

Figure 1: Format of untagged and tagged files

```
%% Lexicon created from the Brown corpus
%% ...
thaw          6   NN        3   VB   3
thawed        3   VBN       3
thawing       2   VBG       2
the       62597   DT    62589   IN   1   JJ   2   NNP   5
theaf         1   NN        1
%% ...
```

Figure 2: Part of a lexicon file

The file may contain empty lines. These can, e.g., be used to denote sentence or paragraph breaks. Figure 1a shows an example of untagged text from the Brown corpus (Francis & Kucera, 1982). The first two lines are comments, the rest is part of the corpus.

By convention, files containing a corpus in this format get `.t` as their filename extension. The file *tnt*`/models/sample.t` contains an untagged sample from the NEGRA corpus.

3.2 Format of Tagged Files

The format of tagged files is similar to that of untagged files. It extends the format by a second column per line. The columns are separated by any number of white space (`TAB = 0x09, SPC = 0x20`). The first column is the token, the second column is the tag. Everything after the second column is ignored.

Again, a line starting with two percentage signs (`%%`) is a comment and is ingored by the programs. A tagged file may contain empty lines, but must not contain a line with only one column. Figure 1b shows an example of tagged text.

By convention, files containing a corpus in this format get `.tt` as their filename extension. The file *tnt*`/models/sample.tt` contains a tagged sample from the NEGRA corpus.

3.3 Format of the Lexicon

The lexicon is created during the parameter generation step by the program `tnt-para` (see section 4.1). It contains the frequencies of words and their tags as they occured in the training corpus. These frequencies are used during tagging to determine lexical probabilities. Generally, the user does not need to change the content of a lexicon file.

Every line in the lexicon starting with two percentage signs (`%%`) is a comment. A line with a lexicon entry contains four or more columns (an even number), separated by white space. The first column is the token, the second number is the frequency of this token in the training corpus. The rest of the columns list the tags that occured with the tokens (odd numbered columns) together with their frequencies (even numbered columns). The sum of the frequencies of all tags for a word is equal to the number in the second column.

The lexicon may contain some special entries to inform the tagger how to process classes of tokens, or to set processing details. They start with an at sign (`@`). Currently, the tagger recognizes the following special lexicon entries:

```
%% n-grams, Brown corpus
%% ...
NNP     62028
NNP     CC      2885
NNP     CC      CD      28
NNP     CC      NN      51
%% ...
NNP     CD      912
NNP     CD      CC      23
NNP     CD      CD      7
%% ...
```

a) long form

```
%% n-grams, Brown corpus
%% ...
NNP     62028
        CC      2885
                CD      28
                NN      51
%% ...
        CD      912
                CC      23
                CD      7
%% ...
```

b) abbreviated form

Figure 3: Part of an n-gram file

entry	*description*	*example match*
`@CARD`	tokens consisting of a sequence of decimal digits	`42`
`@CARDPUNCT`	decimal digits followed by punctuation	`42.`
`@CARDSUFFIX`	decimal digits followed by any suffix	`42nd`
`@CARDSEPS`	decimal digits separated by dots, dashes, etc.	`4.2`
`@UNKNOWN`	tag frequencies to handle unknown words (if unknown word mode 1 is selected for `tnt`)	???
`@USECASE`	use upper/lower case of words (this entry is only followed by either `0` or `1`)	n.a.

As an example, the lexicon entry

```
@CARDPUNCT 527 ADJA 488 ADV 32 CARD 3 TRUNC 4
```

specifies that decimals followed by punctuation occured 527 times as a token in the training corpus. These were tagged as ADJA 488 times, as ADV 32 times, etc. (this entry is taken from the lexicon that was generated using the German NEGRA corpus). Now, if the tagger detects such a sequence and cannot find another entry for it in the lexicon, it takes this distribution to calculate lexical probabilities.

The special entry `@USECASE` indicates, if upper/lower case information should be used (`@USECASE 1`) or should not (`@USECASE 0`).

Figure 2 shows a part of a lexicon file. By convention, files containing a lexicon in this format get `.lex` as their filename extension. The files *tnt*`/models/negra.lex` and `susanne.lex` contain examples of lexicon files.

3.4 Format of the n-gram File

The n-gram file is, like the lexicon file, created during the parameter generation step by `tnt-para`. It contains the contextual frequencies for uni-, bi-, and trigrams. Generally, the user does not need to change the content of an n-gram file.

Every line starting with two percentage signs (%%) is a comment. A line for an n-gram entry contains two (unigrams), three (bigrams) or four (trigrams) columns. All but the last column contain tags, the last column contains the frequency of that particular n-gram in the training corpus.

Bigrams and trigrams can use an abbreviated form of an entry. If a line starts with one `TAB` character (`0x09`), the first tag of the previous line is repeated, if it starts with two `TAB` characters, the first two tags are repeated. Usually, n-gram files use the abbreviated form to use less space on disk.

Figure 3 shows an example of an n-gram file. By convention, files containing n-grams in this format get `.123` as their filename extension. The files *tnt*`/models/negra.123` and `susanne.123` contain examples of n-gram files.

3.5 Format of Map Files

The purpose of a map file is mapping the tags after tagging but before they are written to a file. This can be done either if some names of tags in subsequent processing are different from those in the language model, or if there is more information in the tagset used by the tagger than is needed by subsequent processing steps. In the latter case, it is better to provide the tagger with a larger tagset, and map tags after tagging to the smaller tagset, than training the tagger on the smaller tagset itself (cf. Brants, 1997).

The format of the map file is line oriented. The first column is the original tag as used by the tagger, the second column (separated by white space) is the corresponding tag that the tagger should write to the output. The file `tnt/models/susanne2.map` contains a mapping from a larger Susanne tagset (159 tags), which is described in (Sampson, 1995), to a smaller tagset consisting of 62 tags. The latter tagset makes broader distinctions and takes only the first two characters of the original tagset into account.

4 Usage

The application of TnT consists of two steps:

1. parameter generation
2. tagging

Step 1 creates the model parameters from a tagged training corpus. It is performed when you start to use the tagger, or when you want to modify the model parameters by using a different or larger corpus. Alternatively, you can use one of the pre-defined models for German or English (see section 5).

Step 2 applies the model parameters to new text and performs the actual tagging.

The rest of this section describes the program for parameter generation (`tnt-para`), the tagger (`tnt`), and two helper programs for comparing tagged files (`tnt-diff`) and for counting (`tnt-wc`). The programs give a short summary when started without parameters (except `tnt-wc`, which shows its parameters when started with `tnt-wc -h`).

4.1 Parameter Generation: `tnt-para`

The parameter generation requires a tagged training corpus in the format described in section 3.2. The training corpus should be large and the accuracy of assigned tags should be as high as possible. Generally, the larger the corpus and the higher the accuracy of the training corpus, the better the performance of the tagger.

Parameter generation is started as follows:

```
tnt-para [options] <corpusfile>
```

`<corpusfile>` is the file containing the tagged training corpus; if the special name `-` (a minus sign) is given as the file name, standard input (`stdin`) is read.

The file may be compress:ed, gzip:ed, or bzip2:ed, which is recognized by the suffixes `.Z`, `.gz`, and `.bz2`.

By default, the program generates lexical and contextual frequencies from the training corpus and stores them in two files in the current working directory. The names are the same as for the corpus file, but with the extensions .lex and .123, respectively.

`[options]` is one or more of the following:

`-h` : display a short command line summary; the same summary is shown when you run `tnt-para` without parameters.

`-i` : ignore case; all upper case characters are mapped to lower case characters, and the lexicon will contain lower case entries only.

`-l` : generate lexicon only; by default, lexicon and n-grams are generated, this option suppresses the n-grams.

`-n` : generate n-grams only; by default, lexicon and n-grams are generated, this option suppresses the lexicon.

`-o`*name* : use *name* as the base name for output files; by default, the name of the corpus is used as the base name for output files. This option allows you to specify a different name, the resulting output files will be *name*`.lex` and *name*`.123`.

`-v` : generate verbose n-grams; by default, the abbreviated form for n-gram files will be created. By using this option, the program generates the long form (cf. section 3.4).

Example: the current directory contains the file `mycorpus.tt`, and you want to generate model parameters with all default options (which is probably the right choice in most of the cases). Then type

```
tnt-para mycorpus.tt
```

This creates two files in the current directory, `mycorpus.lex` and `mycorpus.123`, containing the lexical and contextual frequencies.

Example: the current directory contains the file `mycorpus.tt`, and you want to generate a model that ignores upper/lower case and should be named `my2`. Then type:

```
tnt-para -i -o my2 mycorpus.tt
```

This creates two files in the current directory, `my2.lex` and `my2.123`, containing a lexicon with lower case characters only and the contextual frequencies.

4.2 Tagging: `tnt`

The tagging process requires two files containing the model parameters for lexical and contextual frequencies, and an input file in the format described in section 3.1. The tagger uses only the first column in each line of the input file, the rest is stripped. So the rest of the line can contain tags, comments, or any other material without having influence on the tagging process.

The tagger is started as follows:

```
tnt [options] model corpus
```

`model` is the name of the language model to be used. The tagger will look for three files: *`model`*`.lex` and *`model`*`.123` are obligatory and contain the lexical and contextual frequencies as generated by `tnt-para`. The third one, *`model`*`.map`, is optional. If it is found, it is used for mapping the tags before emitting them (see section 3.5).

The files of the language model are first searched for in the current directory, then in the directory pointed to by the environment variable `TNT_MODELS`. The files may be compress:ed, gzip:ed, or bzip2:ed, which is recognized by the suffixes `.Z`, `.gz`, and `.bz2`.

`corpus` is the file with the text to be tagged in the format described in section 3.1 (one token per line). Everything except the first column in each line is stripped and ignored.

The corpus may be compressed or gziped, which is recognized by the suffix `.Z` or `.gz`.

`[options]` is one or more of the following:

`-a`*length* : use a suffix trie of with maximum suffix length *length* to handle unknown words (default: `-a10`)

`-b`*file* **:** use *file* as a backup lexicon, i.e., if a word is not found in the lexicon, try to find it in the backup lexicon first, before applying the unknown word handler.

`-d`*mode* **:** use sparse data mode *mode*, which is one of the following (default: `-d4`).

> `1/`*c* **:** replace zero frequencies by the constant c. Example: `-d1/0.4` replaces 0 by 0.4. The slash and c can be omitted, in this case 0.5 is used as the constant.
>
> `2/`*c* **:** add constant c to all frequencies. Example: `-d2/0.3` adds 0.3 to all frequencies. The slash and c can be omitted, in this case 0.5 is used as the constant.
>
> `4/`λ_1`/`λ_2 **:** use linear interpolation for smoothing. λ_1 is used as the weight for unigrams, λ_2 is used as the weight for bigrams, $(1 - \lambda_1 - \lambda_2)$ is used as the weight for trigrams. Example: `-d4/0.2/0.3` sets λ_1 to 0.2, λ_2 to 0.3, and λ_3 implicitly to 0.5. The slashes and lambdas can be omitted, in this case the weights are determined automatically by deleted interpolation (which is the default mode for the program).

`-h` **:** displays a short command line summary; the same summary is displayed when running the program without parameters.

`-H` **:** copy HTML tags from input directly to output without tagging. No modifications are made to lines containing HTML tags, they are treated as if they were not present in calculating context probabilities. The form of an HTML tag is: first non-space character in the line is '<' and last non-space character is '>'.

`-m` **:** unknown words are marked in the ouput with an asterisk (`*`) in the last column.

`-n`*n* **:** use n-grams for tagging, where $n = 1, 2, 3$. Default: `-n3`

`-u`*mode* **:** use mode *mode* to handle unknown words. *mode* is one of the following: (default: `-u3`)

> 0 No unknown words allowed. The tagger exits with an error when detecting an unknown word;
>
> 1 Take lexicon entry `@UNKNOWN` to determine lexical probabilities for unknown words;
>
> 2 combine statistics of all words to handle unknown words;
>
> 3 combine statistics of all singletons (words occuring once) to handle unknown words.

`-v`***num*** **:** set verbosity to *num*, where *num* is one of the following (default: `-v3`):

> 0 silent;
>
> 1 print progress info on stderr (startup, lambdas, dots...);
>
> 2 print header info in tagged output (source, model, date, ...);
>
> 3 print both progress and header info.

`-z`θ **:** output alternative tag if probability is at least $1/\theta$ of the best tag. Example: `-z100` ouputs the alternative tag if its probability is at least one hundredth of the best one.

`-Z`β **:** cut-off path if the probability is outside beam β (default: `-Z1000`). Finding the best path is not guaranteed when using this option. But setting $\beta = 1000$ causes almost no changes in the output in practical cases, and it has the advantage ofsignificantly increasing tagging speed. If necessary, this option can be turned off by setting `-Z0`.

The difference between `-z`θ and `-Z`β is that `-z`θ calculates more than one path per sequence in order to assign alternative tags (try `-z100`), and that `-Z`β restricts the search space for the best path. You may use both options simultaneously, but setting $\theta > \beta$ is senseless (do not use the options if you do not understand this).

Example: the current directory contains the parameter files `mycorpus.lex` and `mycorpus.123`, created previously. Additionally, it contains new text in file `newcorpus.tt` in the format described in section 3.1 (one token per line). Type

```
tnt mycorpus newcorpus.tt > newcorpus.tts
```

This uses the tagger with default options (trigram mode; smoothing is done by linear interpolation, weights are calculated by deleted interpolation; unknown words are handled with a suffix trie using suffixes of up to 10 characters), the lexical parameters `mycorpus.lex`, the contextual parameters `mycorpus.123`, and tags the words in `newcorpus.tt`. Output is written to standard output (`stdout`), which is redirected to `newcorpus.tts` by the last component of the command (by convention, statistically tagged files get the suffix `.tts`).

Example: ignore upper/lower case and use the previously generated parameter files `my2.lex` and `my2.123` (these were generated with the option `-i` of `tnt-para`). For smoothing use linear interpolation with the fixed weights $\lambda_1 = 0.1$, $\lambda_2 = 0.2$, and $\lambda_3 = 0.7$:

```
tnt -d4/0.1/0.2 my2 newcorpus.tt > newcorpus.tts
```

Again, the output is redirected to the file `newcorpus.tts`.

4.3 Counting Differences: `tnt-diff`

If you have a file containing the correct tags (e.g., created by manual editing), you can compare the correct version with the statistically tagged version.

The program for counting differences is started with

```
tnt-diff [options] <original file 1> <new file 2>
```

`<original file 1>` is the original file, containing the correct tag assignments.

`<new file 2>` is the statistically tagged file.

The files may be compress:ed, gzip:ed, or bzip2:ed, which is recognized by the suffixes `.Z`, `.gz`, and `.bz2`.

`[options]` is one or more of the following:

`-h` : display a short command line summary; the same summary is shown when running the program without parameters.

`-l`*lexiconfile* : use *lexiconfile* as a lexicon to determine known words and give separate counts for known and unknown words.

Example: compare the files A01.tt and A01.tts of the brown corpus. The first file is the original, the second file is generated by `tnt` using a model that is based on all other Brown corpus files (A02.tt ... R09.tt).

```
tnt-diff A01.tt A01.tts
```

The output looks like the following:

```
Comparing .. (2267 tokens)
Overall result:
Equal    :    2186 /   2267 ( 96.43%)
Different:      81 /   2267 (  3.57%)
```

The first line shows the total number of tokens in each of the files, the third line shows the number and percentage of tags identical in both files, and the fourth line shows the number and percentage of tags different in both files.

Example: compare the files A01.tt and A01.tts and show counts separately for known and unknown words. Known words are all those in brown-a02-r09.lex:

```
tnt-diff -l brown-a02-r09.lex A01.tt A01.tts
```

The output looks like the following:

```
Comparing .. (2267 tokens)
Overall result:
Equal    :    2186 /   2267 ( 96.43%)
Different:      81 /   2267 (  3.57%)

2219 tokens known (97.88%), 48 unknown (2.12%).

Counts for known tokens:
Equal    :    2142 /   2219 ( 96.53%)
Different:      77 /   2219 (  3.47%)

Counts for unknown tokens:
Equal    :      44 /     48 ( 91.67%)
Different:       4 /     48 (  8.33%)
```

The first part is identical to the previous example, showing the overall accuracy. Afterwards, the fractions of known and unknown tokens are shown, and their accuracies.

4.4 Counting Tokens and Types: `tnt-wc`

This program can be used to count tokens, types, and different tags in a corpus. It is started with

```
tnt-wc [options] <corpusfile> ...
```

`<corpusfile>` is the file containing the corpus in the format described above (one token per line; either tagged or untagged). You can give as many corpus files as you want, the counts are printed separately for each file, and a sum for the tokens. If no file is given, standard input (`stdin`) is read. Also, the minus sign (`-`) denotes `stdin` as input.

The files may be compress:ed, gzip:ed, or bzip2:ed, which is recognized by the suffixes `.Z`, `.gz`, and `.bz2`.

`[options]` is one or more of the following:

`-h` : Display a short command line summary.

`-i` : Ignore upper/lower case of tokens.

`-l` : Count word types.

`-t` : Count different tags.

`-w` : Count word tokens.

If none of `-l`, `-t`, and `-w` is given, the program prints all these counts.

Example: Count tokens, types and different tags of all files ending in `.tt.gz` in the current directory. The files are unzipped on the fly.

```
tnt-wc *.tt.gz
```

The output looks like the following:

```
ca01.tt.gz           2267 tokens,   840 types,  40 tags
ca02.tt.gz           2297 tokens,   901 types,  38 tags
ca03.tt.gz           2299 tokens,   804 types,  39 tags
...
cr07.tt.gz           2493 tokens,   709 types,  38 tags
cr08.tt.gz           2389 tokens,  1061 types,  40 tags
cr09.tt.gz           2367 tokens,  1064 types,  41 tags
sum               1170815 tokens
```

5 Pre-Defined Models

Two language models come together with the distribution of TnT: a German Model that is trained on the corpus of the NEGRA project (Skut, Krenn, Brants, & Uszkoreit, 1997), and one trained on the Susanne Corpus (Sampson, 1995). These are described in the following sections.

5.1 The German Model

Parameter files for the German model can be found in the directory *tnt*/models. The model is trained on the corpus of the NEGRA project, Saarbrücken, currently consisting of about 300.000 tokens of newspaper texts (Frankfurter Rundschau). It uses the Stuttgart-Tübingen-Tagset (STTS) (Thielen & Schiller, 1995). The corpus was partly tagged at the Institut für Maschinelle Sprachverarbeitung, Stuttgart, and partly at the Department of Computational Linguistics, Saarbrücken.

Lexical frequencies are stored in the file negra.lex, contextual frequencies are stored in negra.123. To use the model for tagging, be sure that TnT can find it, so either keep it in the current directory or in the directory pointed to by the environment variable TNT_MODELS. If you want to tag the file mycorpus.t, run the tagger with the commandline

```
tnt negra mycorpus.t > mycorpus.tts
```

and the results are stored in mycorpus.tts. Average accuracy on unseen German newspaper text is above 96%.

5.2 The English Model

Parameter files for the English model can be found in the directory *tnt*/models. The model is trained on the Susanne corpus, consisting of about 150.000 tokens. The model uses the base tagset, consisting of all uppercase characters and digits, but without the lower case extensions (cf. Sampson, 1995).

Lexical frequencies are stored in the file susanne.lex, contextual frequencies are stored in susanne.123. To use the model for tagging, be sure that TnT can find it, so either keep it in the current directory or in the directory pointed to by the environment variable TNT_MODELS. If you want to tag the file mycorpus.t, run the tagger with the commandline

```
tnt susanne mycorpus.t > mycorpus.tts
```

and the results are stored in mycorpus.tts. Average accuracy on unseen English text from the same domains as the Susanne corpus is around 96%.

The tagset of the previous model contains 159 tags. We defined a mapping to a second tagset, that makes less fine-grained distinctions and contains 62 tags only. The mapping is stored in the file susanne2.map. Otherwise, the model parameters are the same as in the previous example. To map the output of the tagger to the smaller tagset, use the model susanne2:

```
tnt susanne2 mycorpus.t > mycorpus.tts
```

Now, the output will be written using the smaller tagset.

6 Evaluation

We evaluate the tagger's performance under several aspects. First of all, we determine the tagging accuracy averaged over ten iterations. The overall accuracy, as well as separate accuracies for known and unknown words are measured.

Second, learning curves are presented, that indicate the performance when using training corpora of different sizes, starting with as few as 1,000 tokens and ranging to the size of the entire corpus (minus the test set).

An important characteristic of statistical taggers is that they not only assign tags to words but also probabilities in order to rank different assignments. The third set of experiments investigates alternative assignments that are "close to" the best assignment, with "close to" referring to the distance of the respective probabilities.

All tests are performed on partitions of the corpora that use 90% as training set and 10% as test set, so that the test data is guaranteed to be unseen during training. Each result is obtained by repeating the experiment 10 times with different partitions and averaging the single outcomes.

The tagging accuracy is the percentage of correctly assigned tags. We distinguish the overall accuracy, taking into account all tokens in the test corpus, and separate accuracies for known and unknown tokens. The latter are interesting, since usually unknown tokens are much more difficult to process than known tokens, for which a list of valid tags can be found in the lexicon.

6.1 Tagging the NEGRA corpus

The German NEGRA corpus consists of newspaper texts (Frankfurter Rundschau) that are annotated with predicate-argument structures (Skut et al., 1997). It was developed in the project NEGRA (Nebenläufige grammatische Verarbeitung; Concurrent Grammar Processing) at the Saarland University, Saarbrücken. Part of it was part-of-speech tagged at the IMS Stuttgart. The annotation consists of four parts: 1) a non-projective predicate-argument structure, 2) phrasal categories (NP, PP, ...) that are annotated as node labels, 3) grammatical functions (subject, direct object, pre-nominal genitive, ...) that are annotated as edge labels, and 4) part-of-speech tags. This evaluation only uses the part-of-speech annotation.

By the time the experiments were performed, October 1998, it had a size of approx. 17,000 sentences (300,000 tokens).

Tagging accuracies for the NEGRA corpus are shown in table 4.

Figure 5 shows the learning curve of the tagger, i.e., the accuracy depending on the amount of training data. Training length is the number of tokens used for training. Each training length was tested ten times, training and test sets were disjoint, results were averaged. The training length is given on a logarithmic scale.

It is remarkable that tagging accuracy for known words is very high even for very small training corpora. This means that we have a good chance of getting the right tag if a word is seen at least once during training. Average percentages of unknown tokens are shown in the bottom line of each diagram.

We exploit the fact that the tagger not only determines tags, but also assigns probabilities. If there is an alternative that has a probability "close to" that of the best assignment, this alternative can be viewed as almost equally well suited. The notion of "close to" is expressed by the distance of probabilities, and this in turn is expressed by the quotient of probabilities. So, the distance of the probabilities of a best tag t_{best} and an alternative tag t_{alt} is expressed by $p(t_{best})/p(t_{alt})$, which is some value greater or equal to 1 since the best tag assignment has the highest probability (we use the `-z` option of TnT).

Figure 6 shows the *recall* when taking more and more alternatives into account. Here, an assignment is counted as correct if either of the alternatives is correct. The curves start as distance factor 1, i.e. only the best tag (or alternative tags with identical probabilities) is assigned. Note that this is the standard notion of accuracy, and the percentages at this point are the same as the averages in table 4.

6.2 Tagging the Penn Treebank

We use the Wall Street Journal as contained in the Penn Treebank for our experiments. The annotation consists of four parts: 1) a context-free structure augmented with traces to mark movement and non-contiguous constituents, 2) phrasal categories that are annotated as node labels, 3) a small set of grammatical functions that are annotated as extensions to the node labels, and 4) part-of-speech tags (Marcus, Santorini, & Marcinkiewicz, 1993). This evaluation only uses the part-of-speech annotation.

The Wall Street Journal part of the Penn Treebank consists of approx. 50,000 sentences (1.2 million tokens).

Tagging accuracies for the Penn Treebank are shown in table 7.

Table 4: Part-of-speech tagging accuracy for the NEGRA corpus, averaged over 10 test runs, training and test set are disjoint. The table shows the percentage of unknown tokens, separate accuracies and standard deviations for known and unknown tokens, as well as the overall accuracy.

	percentage	known		unknown		overall	
	unknowns	acc.	σ	acc.	σ	acc.	σ
NEGRA corpus	13.1%	97.7%	0.28	86.6%	1.01	96.3%	0.27

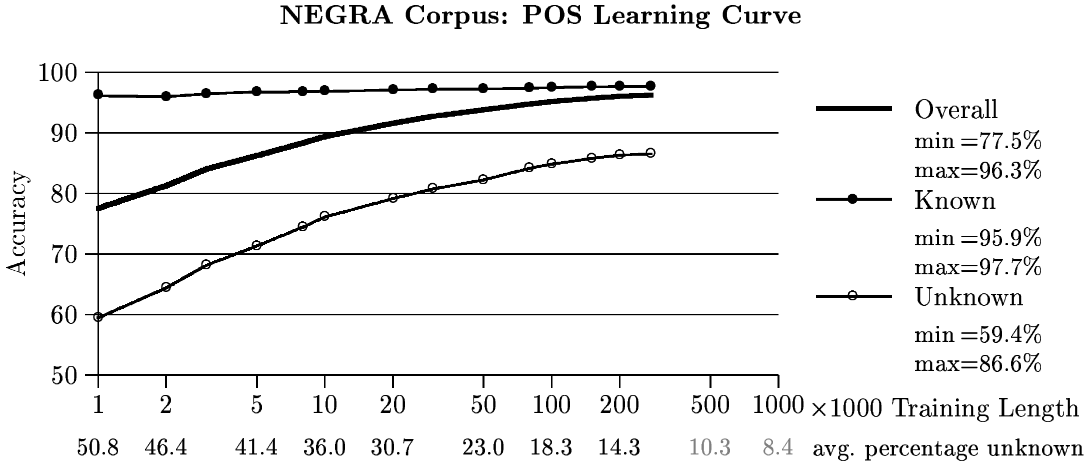

Figure 5: Learning curve for tagging the NEGRA corpus. The training sets of variable sizes as well as test sets of 10,000 tokens were randomly chosen. Training and test sets were disjoint, the procedure was repeated 10 times and results were averaged. Percentages of unknowns for 500k and 1000k training are determined from an untagged extension.

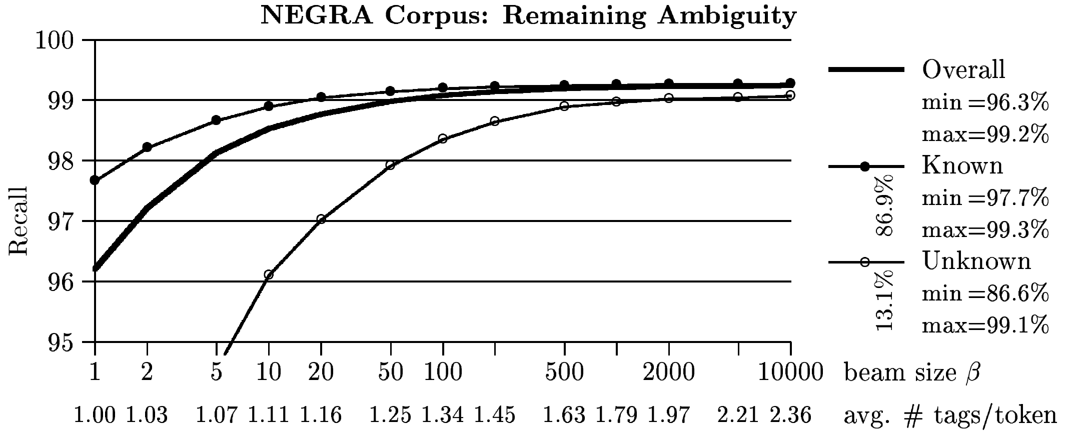

Figure 6: Tagging accuracy for the NEGRA corpus when some ambiguity remains after tagging. The best tag t_{best} and all tags t_{alt} with probabilities within the beam β (having $p(t_{best})/p(t_{alt}) \leq \beta$) are assigned. The numbers at the bottom line indicate the average number of assigned tags per token.

Table 7: Part-of-speech tagging accuracy for the Penn Treebank. The table shows the percentage of unknown tokens, separate accuracies and standard deviations for known and unknown tokens, as well as the overall accuracy.

	percentage	known		unknown		overall	
	unknowns	acc.	σ	acc.	σ	acc.	σ
Penn Treebank	2.8%	97.1%	0.12	84.2%	0.54	96.7%	0.13

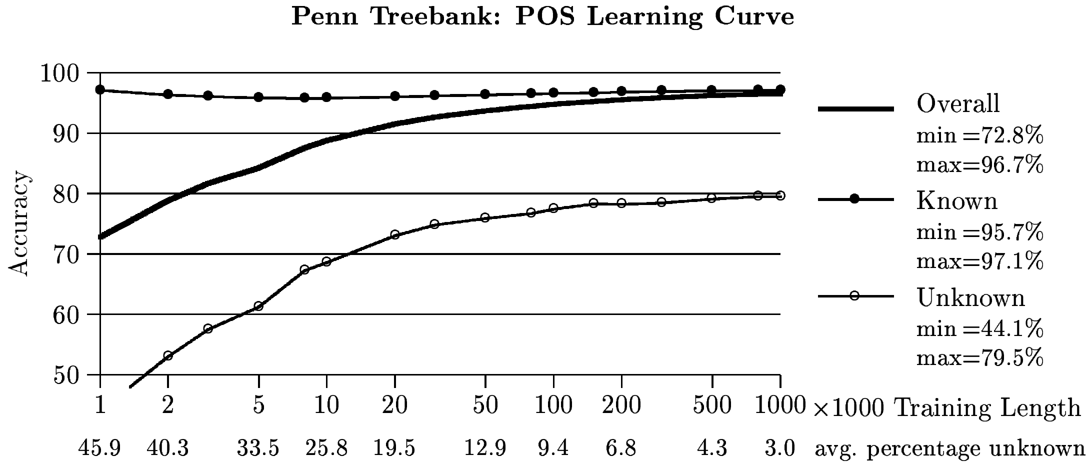

Figure 8: Learning curve for tagging the Penn Treebank. The training sets of variable sizes as well as test sets of 100,000 tokens were randomly chosen. Training and test sets were disjoint, the procedure was repeated 10 times and results were averaged.

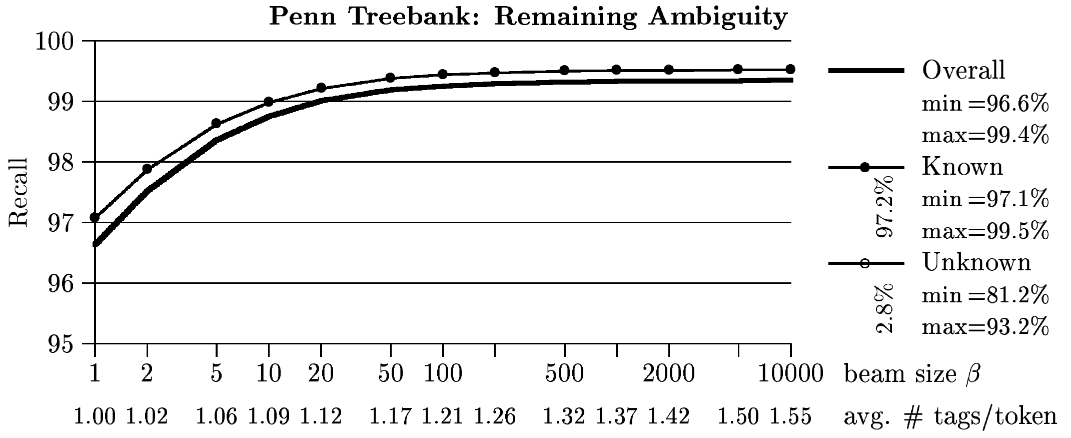

Figure 9: Tagging accuracy for the Penn Treebank when some ambiguity remains after tagging. The best tag t_{best} and all tags t_{alt} with probabilities within the beam β (having $p(t_{best})/p(t_{alt}) \leq \beta$) are assigned. The numbers at the bottom line indicate the average number of assigned tags per token. Results for unknowns are below 95% and therefore not shown in the diagram.

Figure 8 shows the learning curve of the tagger, i.e., the accuracy depending on the amount of training data. Training length is the number of tokens used for training. Each training length was tested ten times, training and test sets were disjoint, results are averaged. The training length is given on a logarithmic scale.

As for the NEGRA corpus, tagging accuracy is very high for known tokens even with small amounts of training data.

We exploit the fact that the tagger not only determines tags, but also assigns probabilities. If there is an alternative that has a probability "close to" that of the best assignment, this alternative can be viewed as almost equally well suited. The distance of the probabilities of a best tag t_{best} and an alternative tag t_{alt} is expressed by $p(t_{best})/p(t_{alt})$, which is some value greater or equal to 1 since the best tag assignment has the highest probability (we use the `-z` option of TnT).

Figure 9 shows the *recall* when taking more and more alternatives into account. Here, an assignment is counted as correct if either of the alternatives is correct. The curves start as distance factor 1, i.e. only the best tag (or alternative tags with identical probabilities) is assigned. Note that this is the standard notion of accuracy, and the percentages at this point are the same as the averages in table 7.

6.3 Summary of Part-of-Speech Tagging Results

Average part-of-speech tagging accuracy is between 96% and 97%, depending on language and tagset, which is at least en par with state-of-the-art results found in the literature.

Accuracy for known tokens is significantly higher than for unknown tokens. For the German newspaper data, results are 11% points better when the word was seen before and therefore is in the lexicon, than when it was not seen before (97.7% vs. 86.6%). Accuracy for known tokens is high even with very small amounts of training data. As few as 1000 tokens are sufficient to achieve 95%–96% accuracy for them. It is important for the tagger to have seen a word at least once.

Stochastic taggers assigns probabilities to tags. We exploit the probabilities to leave selected ambiguity after tagging if the probability of the second best assignment is close to that of the best assignment. This identifies reliable and unreliable assignments and we leave some ambiguity in the output of the tagger if the assignment of a unique tag would be unreliable. Allowing 103 tags in the output for 100 tokens increases accuracy by approx. 1% point for both the German and English data.

7 Restrictions

- Tokens and tags cannot contain white space (white space delimits columns).
- Tokens and tags cannot start with two percentage signs (%% at the beginning of a line starts a comment).
- Tokens cannot consist of an at sign (`@`) followed by any sequence of characters (these are special lexicon entries).
- There can be at most 32766 ($2^{15} - 2$) different tags (they are encoded using short integers).
- The maximum length of a line within any file is restricted to 16383 characters (this is the size of the input buffer).

Acknowledgements

Many thanks go to Hans Uszkoreit for his support during the development of TnT. Thanks go also to the Deutsche Forschungsgemeinschaft for financing the work by a grant in the Graduiertenkolleg Kognitionswissenschaft Saarbrücken. Large annotated corpora are the pre-requisite for developing and testing part-of-speech taggers, and they enable the generation of high-quality language models.

Therefore, I would like to thank all the people who were involved in building the Stuttgarter Referenzkorpus, the NEGRA Corpus, the Penn Treebank and the Susanne Corpus. And, last but not least, I would like to thank the users of TnT who provided me with bug reports and valuable suggestions for improvements.

References

Brants, T. (1997). Internal and external tagsets in part-of-speech tagging. In *Proc. of eurospeech.* Rhodes, Greece.

Brown, P. F., Pietra, V. J. D., deSouza, P. V., Lai, J. C., & Mercer, R. L. (1992). Class-based n-gram models of natural language. *Computational Linguistics*, *18(4)*, 467-479.

Francis, N. W., & Kucera, H. (1982). *Frequency analysis of English usage.* Boston: Houghton Mifflin.

Marcus, M., Santorini, B., & Marcinkiewicz, M. A. (1993). Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, *19*(2), 313-330.

Rabiner, L. R. (1989). A tutorial on Hidden Markov Models and selected applications in speech recognition. In *Proceedings of the IEEE* (Vol. 77, pp. 257–285).

Sampson, G. (1995). *English for the computer.* Oxford: Oxford University Press.

Samuelsson, C. (1993). Morphological tagging based entirely on Bayesian inference. In *9th nordic conference on computational linguistics NODALIDA-93.* Stockholm University, Stockholm, Sweden.

Skut, W., Krenn, B., Brants, T., & Uszkoreit, H. (1997). An annotation scheme for free word order languages. In *Proceedings of the fifth conference on applied natural language processing ANLP-97.* Washington, DC.

Thielen, C., & Schiller, A. (1995). Ein kleines und erweitertes Tagset fürs Deutsche. In *Tagungsberichte des Arbeitstreffens Lexikon + Text 17./18. Februar 1994, Schloß Hohentübingen. Lexicographica Series Maior.* Tübingen: Niemeyer.