1 Assignment

This assignment of today consists of two parts. First, you will familiarize yourself with the parser Bitpar, by playing around with a small toy-grammar that you will create yourself. Section 2 contains instructions on how to install Bitpar as well as basic information on its functionality. In the second part of the assignment you will use Bitpar to train a grammar with inside outside, and investigate the effect of the number of parameters on the maximum likelihood estimation. To prevent you from getting stuck on the dirty work, some pointers are provided in Section 3 through 4.

Go through the following steps:

1. Install bitpar on your machine
2. Create a toy grammar that can parse the following sentences (and use bitpar to parse them)

   The man sees the woman with the telescope

   The woman sees the man with the beard

   The man sees the bird with the telescope
3. Extract a grammar from a treebank (for instance a section of wsj) and see if you can train its parameters with Bitpar
4. Create a different version of the treebank in which all the non-terminals (except TOP) are unique, extract a grammar from this corpus, merge this grammar with the grammar from the previous step and train the resulting grammar with Bitpar. What happens with the parameters?

2 Bitpar

Installing Bitpar

```
mkdir $workdir                                                     # create a working directory
cd $workdir                                                        # change to working directory
wget ftp://ftp.ims.uni-stuttgart.de/pub/corpora/BitPar/New-BitPar.tar.gz   # download BitPar
tar xvf New-BitPar.tar                                             # extract files
cd BitPar/src                                                      # change to source directory
make                                                               # compile the parser
cd ../../                                                          # go back to $workdir
ln -s $workdir/BitPar/src/bitpar bitpar                            # create a symbolic link to bitpar
```

Usage

Information on the usage and different options of BitPar can be found by running it without any parameters:

```
./path-to-src/bitpar
```

More elaborate information is available in the file bitpar.1, that can be found in the subdirectory man1 in the BitPar folder. Flags you will probably use quite frequently are:

-s TOP	use S as a start symbol
-vp	print the parse forest with Viterbi probabilities
-b n	print the best n parse trees

Grammar, lexicon and corpus

Bitpar requires a specific format for the grammar, the lexicon and the corpus.

Grammar A grammar file should contain one grammar rule per line and should look like this:

frequency S NP VP

Lexicon The lexicon file should contain one entry per line. This entry starts with the word (that may contain blanks) and is followed by a tab and a sequence of part of speech tags followed by a whitespace and a frequency value. For example:

saw NN 3 VB 1

Corpus The corpus you want to parse should contain one entry per line. Sentences are separated by blank lines.

Examples

Print the parse forest with viterbi probabilities for all sentences in the corpus:

```
./ bitpar -vp -s TOP grammar lexicon corpus
```

Print the parse with the highest probability for all sentences in the corpus and write it to a file named 'out':

```
./ bitpar -vp -s TOP -b 1 grammar lexicon corpus > out
```

Do one pass of io and write the new grammar and lexicon to em.gram and em.lex:

```
.././bitpar -s TOP grammar lexicon corpus -em em
```

3 Creating a Bitpar grammar from a treebank

You can use the program PCFG_extractor.jar to extract a PCFG grammar from a corpus:

```
java -jar PCFG_extractor.jar treebank1 grammar
```

If you look at the output file, you will notice that the format is not equal to the format required by Bitpar for parsing. You can follow the steps below to create a Bitpar grammar and lexicon from your grammar file.

First, reorder the information such that lines start with their frequency count:

```
less grammar | sed 's/%/%%/g' | awk '{printf ($NF " ");
for (i=1; i<NF; i++) printf ($i " "); printf ("\n");}' > grammar.bitpar
```

Then create a lexicon file by coping the lines containing quotes:

```
less grammar.bitpar | grep -E '"[^"]+"' | sed 's/\"//g' | awk
'{print($3"\t"$2"\t"$1)}' > bitpar.lexicon
```

Finally, remove the lexical rules from the grammar file

```
sed -i 's/"//g' bitpar.grammar
grep -v '"' treebanks.grammar.bitpar
```

Create treebank with unique terminals

For the second part of this assignment you will have to create a treebank with unique non-terminals. On way of doing this is by assigning all non-terminals (except for TOP) in the treebank a unique number:

```
cat treebank | sed 's/\([A-Z]\) /\1-NR- /g' | awk '{ while($0~/-NR-/)
sub(/-NR-/, ++cnt); print}' > treebank_unique
```

4 Io with Bitpar

First create the corpus to parse from your treebank:

```
cat treebank | sed 's/$/""\n/' | grep -o '"\([^"]*\)"' | sed 's/"//g' > corpus
```

Then do a single pass of inside-outside with bitpar:

```
./bitpar -s TOP grammar lexicon corpus -em em1
```

Bitpar then generates a new lexicon and grammar file. Unfortunately, it is not possible to directly use these files for a next pass, as we need to remove probabilities that have gone to 0:

```
less em.gram | grep -v '0\.00' > em.grammar
less em.lex | grep -v '0\.000*' | awk '{print $1"\t"$2" "$3 }' > em.lexicon
```