Natural Language Processing: Programming Project II PoS tagging

Reykjavik University – School of Computer Science

November 2009

1 Pre-processing (20%)

In this project you experiment with part-of-speech (PoS) tagging using the German *Tiger* corpus. Go to `http://www.ims.uni-stuttgart.de/projekte/TIGER/TIGERCorpus/` and download the corpus by selecting the "License" link.

The *tiger_release_aug07.export* file is the one you need to pre-process. This file has various data (see the link "Negra Export Format" on the Tiger web page), but in this project we are only interested in <*word, PoS tag*> pairs (starting from line no. 2356).

1.1 Data extraction

Here you need to generate one file from *tiger_release_aug07.export*. The file should contain column 1, as well as columns 3 (an STTS (Stuttgart-Tübingen Tagset) tag) and 4 (morphological tags) merged together. Put a dot (".") in between the data from columns 3 and 4. Thus, the resulting file contains a token and its morphosyntactic tag. Name this file *data.txt*. For example, the third and the forth lines in *data.txt* should look like this:

```
Perot   NE.Nom.Sg.Masc
wäre    VAFIN.3.Sg.Past.Subj
```

Make sure that there is an **empty line** between sentences in *data.txt*. If you do the above described extraction correctly, the file contains 939,052 lines.

1.2 Split into train and test corpora

Assuming you have the file *data.txt*, you now need to generate a <train, test> pair for this file. The last 49,982 lines (starting with "Die Folge ist ...") in the file constitutes the test data, everything above it consists of the training data.

Therefore, the output of this phase consists of two corpora: *train.txt* and *test.txt*.

1.3 Use a shell script!

You should write a shell script which uses some Linux text processing utilities like *head*, *tail*, *grep*, *sed*, and *awk* to extract all the relevant data described above. An alternative is to use Perl to perform this task.

2 Base tagger (30%)

In this part you develop a *base tagger*, i.e. a tagger which always selects the most frequent tag for each word (token). Use the following procedure to develop your tagger:

1. **Training**

   - For the training corpus *train.txt*, use the Linux tools *sed*, *uniq* and *sort* to construct the file *train.freq* showing the frequency of each <word, tag> pair appearing in *train.txt*.
     Here are the first ten entries from *train.freq*:

     ```
     41190 ,       $,.--
     38880 .       $..--
     15341 und     KON.--
     12988 in      APPR.--
      8331 ''      $(.--
      8329 ``      $(.--
      7942 der     ART.Dat.Sg.Fem
      7771 von     APPR.--
      6807 die     ART.Acc.Sg.Fem
      6426 der     ART.Gen.Sg.Fem
     ```

   - Write a Perl program, *buildDictionary.pl* which reads a file in the format described above (i.e. the format of *train.freq*). The output is a dictionary, *train.dict*. Each line has the following form:

```
w wfreq t1 t1freq t2 t2freq ...
```

Each line shows how often ($wfreq$) a specific word w appears in the input, followed by <tag, frequency> pairs for the given word, sorted by descending frequency. For example <t1, t1freq> denotes that w was tagged $t1freq$ times with tag $t1$ and that $t1$ was the most frequent tag for w.

Note that this dictionary contains more information than is needed for your base tagger, since it only needs information about the most freqent tag, but all this would be needed if you were developing an HMM tagger.

2. **Tagging new text (testing)**. Write a Perl program, *baseTagger.pl*, accepting a dictionary (having the format described above) as input and tokenised text for testing, i.e. one token per line with an empty line between sentences. I suggest that you supply the file *test.txt* as the second parameter even though it contains both the token and its correct tag – the program will just simply ignore the tag. The program outputs one line for each word and its most frequent tag.

   For unknown words:

   - Use the the tag **NN.Nom.Sg.Masc** as a default (this tag is the most frequent noun tag in the corpus).
   - Moreover, output *<UNKNOWN>* at the end of the line to mark the word as unknown.

   An example of usage: *perl baseTagger.pl train.dict test.txt test.out*

   An example of lines (no. 12-34) from *test.out*:

```
`` $(.--
Man PIS.Nom.Sg.*
hat VAFIN.3.Sg.Pres.Ind
einen ART.Acc.Sg.Masc
Teil NN.Nom.Sg.Masc
der ART.Dat.Sg.Fem
Landsleute NN.Gen.Pl.*
aus APPR.--
dem ART.Dat.Sg.Masc
demokratischen ADJA.Pos.Dat.Sg.Fem
Konsens NN.Acc.Sg.Masc
```

```
verloren VVPP.Psp
, $,.--
weil KOUS.--
sie PPER.3.Nom.Sg.Fem
sich PRF.3.Acc.Sg
solcher PIAT.Gen.Pl.Neut
Anbiederung NN.Nom.Sg.Masc <UNKNOWN>
nicht PTKNEG.--
anschließen VVINF.Inf
wollten VMFIN.3.Pl.Past.Ind
. $..--
```

3 Accuracy (30%)

Write a Perl program, *accuracy.pl*, accepting an input file in the following form:

```
word correct_tag word tag_from_tagger <UNKNOWN>
```

(<UNKNOWN> only appears if the word is unknown).

The input file is thus a combination of a gold standard file and the output from a tagger. The output is written to *standard out* and shows the accuracy of the tagger for known words, unknown words and all words.

An example of usage: *perl accuracy.pl <combined_file>*

Example output:

```
Number of tokens: 47372
Number of errors: 16384
Overall tagging accuracy: 65.41%
Tagging accuracy for known words: 71.19%
Number of unknown words: 4108
Unknown word ratio: 8.67%
Number of errors for unknown words: 3918
Tagging accuracy for unknown words: 4.63%
```

For this part, I recommend that you write a script which starts by combining *test.txt* and *test.out* into one file and then executes *accuracy.pl*.

4 Training and testing an HMM tagger (20%)

In this part you need to train *TriTagger*, the trigram tagger which is part of the *IceNLP* toolkit, and then use it to tag the test corpus.

Important: *IceNLP* assumes all files are **UTF-8** encoded. Therefore, you have to convert your training and test corpora to UTF-8 (if you have not done that already) before using *TriTagger*.

1. Download *IceNLP* from `http://www.ru.is/faculty/hrafn/Software/IceNLP-1.3.zip`, and extract to a directory of your choice.

2. Read the section on *TriTagger* in the user manual *IceNLP.pdf* (available in the /doc directory) to become familiar with how to train using a training corpus, and run the tagger using a test corpus.

3. Make a version of your test corpus which **only includes the words**, but not the PoS tags.

4. Build a training model using the *train.txt* corpus. **Show the command you use for training**.

5. Now use *TriTagger* to tag the test corpus[1] and make it write the output to the file *test.tri.out*. **Show the command you use for tagging (testing)**.

6. Finally, use your accuracy.pl program to compute the accuracy of *TriTagger* for the test corpus. **Show the output of accuracy.pl**.

5 What to hand in

Return a report describing each step you took to solve this project. In particular, describe all commands used and programs and scripts implemented. Moreover, handin all scripts and programs, along with instructions on how to use them. Also make sure that you discuss the results.

[1] Make sure that you start TriTagger with the *-cs* flag. Information about the flags is in the user manual and you can also see command line flags by typing *./tritagger.sh -help*.