Natural Language Processing: Programming Project II POS tagging

Reykjavik University – School of Computer Science

October 2009

1 Base tagger (60%)

In this part you develop a *base tagger*, i.e. a tagger which always selects the most frequent tag for each word (token). Start by downloading the file `http://www.ru.is/faculty/hrafn/Data/eng.zip`. The .zip file contains files for training (*eng.train*) and testing (*eng.tst*), as well as a gold standard for the test file (*eng.tst.gold*).

Use the following procedure to develop your tagger:

1. **Training**

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

```
7374 . .
7290 , ,
7243 the DT
3751 of IN
3382 to TO
3378 in IN
2993 a DT
2861 ) )
2861 ( (
2838 and CC
```

- Write a Perl program, *buildDictionary.pl* which reads a file in the format described above (i.e. the format of *eng.freq*). The output is a dictionary, *eng.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. If you find this to difficult to implement, then you are allowed to generate a dictionary in the following format instead:

  ```
  word tag
  ```

  i.e. a file which only includes information on a word and its most frequent tag[1].

2. **Tagging new text (testing)**. Write a Perl program, *baseTagger.pl*, accepting a dictionary (having either of the two formats described above) as input and tokenised text for testing, i.e. one token per line with an empty line between sentences. The program outputs one line for each word and its most frequent tag.

   For unknown words:

   - If the word starts with a lower case letter then use the tag *NN* (denoting a common noun).
   - If the word starts with an upper case letter then use the tag *NNP* (denoting a proper noun).
   - Moreover, output *<UNK>* at the end of the line to mark the word as unknown.

   An example of usage: *perl baseTagger.pl eng.dict eng.tst eng.tst.out*

   An example of lines from *eng.tst.out*:

[1] Note, however, that a dictionary with complete frequency information is, for example, needed to build a tagger based on a HMM.

```
Realizing NNP <UNK>
he PRP
was VBD
almost RB
running VBG
Jovah NNP <UNK>
slowed NN <UNK>
to TO
a DT
walk NN
, ,
then RB
stopped VBN
and CC
looked VBD
out RP
over IN
the DT
sea NN
. .
```

2 Accuracy (30%)

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

```
word correct_tag word tag_from_tagger <UNK>
```

(<UNK> 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>*

An example of output:

```
Number of tokens: 2144
Number of errors: 293
Overall tagging accuracy: 86.33%
Tagging accuracy for known words: 94.41%
Number of unknown words: 302
```

```
Unknown word ratio: 14.09%
Number of errors for unknown words: 190
Tagging accuracy for unknown words: 37.09%
```

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

3 Linguistic rules (10%)

Can you add simple linguistic rules to *baseTagger.pl* for the purpose of increasing the accuracy? You could, for example, add rules to improve the accuracy of unknown words and/or known words.

Incrementally develop rules and use *accuracy.pl* to compute your accuracy at each step.

4 What to hand in

Return the sequence of commands you used for training your base tagger (i.e. the commands you used to generate *eng.freq*), all program code (also scripts), along with the output from *accuracy.pl* using base tagging (and improved tagging if you implemented that part). Moreover, return instructions on how to train and test your tagger and how to compute the accuracy.