Natural Language Processing – Assignment I

Reykjavik University – School of Computer Science

September 2015

1 grep and sed – 16%

In this part, you process the corpus *eng.sent*[1] using the Linux regular expression tools *grep* and *sed*. This corpus contains one English sentence per line where each word is tagged with a part-of-speech (PoS) tag. The tagset used is the Penn Treebank tagset (see `http://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html`).

For each item below, show the exact command that you use.

1. (10%) Use *grep* or *egrep* to display information from *eng.sent* that match certain patterns. Note: The web page `http://www.uccs.edu/~ahitchco/grep/` should help.
   (a) Display all sentences that include the exact word forms "German" and "car" (not necessarily adjacent, but in that order).
   (b) Display the number of sentences that include the exact word form "tennis".
   (c) Display plural English nouns (not proper nouns) of length 12. Here we don't want the whole line, only the nouns in question.
   (d) Display all sentences that start with a Wh-pronoun.
   (e) Display all sentences that include the exact word form "offer", either as a verb or as a noun, and either preceded by a word tagged as "DT" or a word tagged as "TO".

2. (6%) Use *sed* to perform the following tasks on *eng.sent* and write the output to *eng.out*. Note: The web page `http://www.grymoire.com/Unix/Sed.html` should help.
   (a) Substitute a number at **the beginning** of a line with the number itself sourrounded by brackets (e.g. 123 becomes [123]).
   (b) Make each <token,tag> pair appear on a separate line. For example, the first three lines in eng.out should appear as:

   ```
   EU NNP
   rejects VBZ
   German JJ
   ```

   (c) Remove the PoS-tags such that only the tokens remain. For example, the first line in eng.out should look like this:

   ```
   EU rejects German call to boycott British lamb .
   ```

[1] This is a Reuters corpus, available from `http://www.ru.is/~hrafn/Data/eng.zip`. Due to copyright reasons, please make sure you do not distribute this corpus.

2 Python – 34%

This part consists of two Python programs.

1. (16%) Write a Python program, *corpusAnalysis.py*, which prints out various statistics/information about a given text file, which is a part of the gutenberg corpus. The name of the text file is given as a parameter to the program, i.e.

   ```
   python corpusAnalysis.py edgeworth-parents.txt
   ```

   The program should print out the following information for the given text file:

   ```
   Text:  edgeworth-parents.txt
   Tokens:  210663
   Types:  9593
   Types excluding stop words:  9466
   10 most common tokens:  [(',', 15219), ('the', 7149), ('.', 6945), ('to', 5150), ('and', 4769),
     ('"', 3880), ('of', 3730), ('I', 3656), ("'", 3293), ('a', 3017)]
   Long types:  ['incomprehensible', 'indiscriminately', 'contradistinction', 'misunderstanding']
   Adjectives ending in 'uous'  ['conspicuous', 'ingenuous', 'assiduous', 'contemptuous',
     'presumptuous']
   ```

   Note: Long types are those with more than 15 characters.

   Return your program code (.py file) along with the output of your program when running against the file *austen-emma.txt*.

2. (18%) Write a Python program, *findLongest.py*, accepting a text file as an argument. The program finds the longest lower case word in the file and prints it out along with its length. Note that the program should work for any text file, regardless of its format (i.e. whether the file has one word per line or multiple words per line). To invoke the program the user should type in:

   ```
    python findLongest.py <filename>
   ```

   Note: Use NLTK for tokenizing the input. The amount of code that you have to write is about 15 lines (or less).

   Test your program on the corpus *eng.sent* and return your program code (.py file) along with the output of your program when running against this corpus.

3 JFlex/Tokenisation – 22%

In this part, you develop a *tokeniser* for a natural language of your choice. Indeed, it would be best if your tokenizer could handle a family of languages (instead of a single language), for example some of the Germanic languages like English, German, Icelandic, Danish and Swedish. With language independence in mind, regular expressions for matching abbreviations are preferred to an implementation using a list of known abbreviations.

You need to use **JFlex** as the implementation languages and name your .flex file as *tokeniser.flex*. Implement a shell script called tokenise.sh (or tokenise.bat) for the user to start when executing your tokeniser (the shell script then calls the resulting Java program). The shell script should accept two parameters: the name of the input file and the name of the output file:

```
tokenise.sh input.txt output.txt
```

The input file is a text file with a "free format", i.e. the file can contain one sentence per line, many sentences per line, one token per line, several tokens per line, etc. The output file should contain one token per line.

3.1 Testing and what to return

Your goal should be to develop a tokeniser which is reasonably accurate, but it does not have to be perfect (which indeed is a difficult task!).

For testing you should gather text in your own language (and related languages) from the web, for example, text from newspapers, personal pages, university pages, etc. The text should contain at least 10,000 tokens and make sure that it contains **different types of tokens**, e.g. words, personal names, numbers and abbreviations. To start with you can use the attached test file *smallTestDataForTokeniser.txt*

Briefly discuss what problems you see with your tokeniser.

You need to return your program code (the .flex file), your test file (.txt file) and the output generated by your tokeniser (.txt file) when processing the test file.

4 Language Modeling – 28%

In this part, you develop an English trigram language model based on *eng.sent* by only using Linux tools: `sed, awk, head, tail, paste, sort, uniq, wc`. Note that here we are only interested in word (token) trigrams (including punctuations), not PoS trigrams.

(a) (6%) *eng.sent* is pre-tokenised even though the <token,tag> pairs do not appear on a separate line. However, in order to construct the language model you need a file with one token (word) per line without any empty lines. Show the sequence of command that you use for constructing this file, *eng.tok*.

(b) (8%) Show the sequence of commands you use to construct a trigram frequency file *engTri.freq*[2] (from *eng.tok*), sorted in descended order of frequency.

(c) (2%) Show the command you use for displaying the 10 most frequent trigrams from *engTri.freq*, as well as the output from this command.

(d) (6%) How many trigrams and distinct trigrams exists in eng.sent? Use *awk* and *wc* and *engTri.freq* to figure this out (show your commands and the output).

(e) (6%) Use the data from *engTri.freq* to estimate (using Maximum Likelihood Estimation):

P(said on Monday | said on)

Show which lines from *engTri.freq* you use to estimate this probability and your calculations.

[2]Containing four columns: frequency, $word_1$, $word_2$, $word_3$.