INF5830, 2015, Group 3 – 17.9

I expect that not everybody has completed the obligatory assignment before class. They may prefer to work on the assignment. But for they who have completed the assignment, or are near completion, or prefer to take a break from the oblig., we offer the following. And everybody should complete the following set – sooner or later (before next oblig.)

NumPy and SciPy

We have made a short description of some of the functionality of NumPy and SciPy. You are advised to work your way throught the paper, running the examples, up to T-distribution. You may then use SciPy to check some of your solutions to the obligatory assignment.

Tools for tokenization and normalization

Jurafsky and Martin, SLP, 3 ed. includes a subsection called "2.3.1 Unix tools for crude tokenization and normalization." This is based on Ken Church' famous *Unix for Poets* from 1994. It illustrates the efficiency of some Unix/Linux commands. It illustrates one possible approach to working with texts. Another approach, which is more in the line of the rest we are doing, is to do everything within (interactive) Python. Solve the same task in Python with the help of NLTK.

Conditional frequencies

When we have a tagged text, we may be interested in for each word which tags it may take and how frequent the different tags are. We will use the full Brown corpus and the 'universal' tagset. See the NLTK book, 2.ed, ch. 5 sec. 2.2 http://www.nltk.org/book/ch05.html.

a) Make a conditional frequency distribution which to each word form yields a frequency distribution of the tags it may take. Thus, for example

```
word_tag['run']['NOUN']
```

may return 52 (or some other number).

b) We will then consider probabilities instead of absolute numbers. Make a conditional frequency distribution which to a word and a tag returns the probability that the word gets this tag. The result now might be something like

```
>>> prob_word_tag['run']['NOUN']
>>> 0.2524271844660194
```

c) In HMM-tagging, even though our goal is to find the most probable tag given a word, what we actually use is the probability of the word given the tag. Now implement a conditional frequency distribution which to each tag yields a frequency distribution over words, e.g.,

```
>>> tag_word['NOUN'][‘run’]
>>> 52
```

d) Then turn this into relative frequencies, i.e., given a tag it should yield a probability distribution for the words that take this tag.

THE END