# INF5830, 2011, Obligatory assignment 1

*Deadline: Sept. 25, 18.00*
*To be delivered in devilry*

## Exercise 1 – Counting

FSNLP Chapter 1 counts some frequencies from Tom Sawyer. We will control whether they are correct. You may download Tom Sawyer from project Gutenberg. You should then

1. Calculate the frequencies of the most common words, corresponding to table 1.1 in FSNLP.
2. Calculate the frequencies of frequencies corresponding to table 1.2.

You have to prepare the file before you calculate the frequencies. You should remove the extra header added by Gutenberg. You should then tokenize. You may use nltk.word_tokenize(). You should also clean the text so you only have the words, e.g. get rid of punctuation and non-words and then lowercase.

You should hand in:

1. A description of which steps you did in the tokenization.
2. The tables corresponding to tables 1.1 and tables 1.2 and a short description of how you found them.

## Exercise 2 – Python

a) Write a Python function which takes one argument which is a list of numbers and returns the mean of the numbers.

b) Write a Python function which takes a list of numbers as an argument and returns the variance. We may sometimes be interested in the unbiased empirical variance (divide by (n-1)) and sometimes in the theoretical variance (divide by n). Include a second argument to the function which lets the user select between the two

c) Write a Python function which takes a list of numbers as an argument and returns the standard deviation. You should have an extra argument and do this two ways as with the variance. You may import sqrt from the math module or from NumPy.

You are not supposed to use the NumPy built-in functions in this exercise but implement the functions yourself. You may use the NumPy functions to check that you get the same results.

You should hand in the code.

## Exercise 3 – Estimation

a) We will look at sentence lengths in the Brown corpus and how it corresponds to genre. First select a random sample of 100 sentences from the 'news' genre. Use this to estimate the mean sentence length of the news genre in general.

b) Repeat with a different sample of 100 sentences form the same genre. How different are the two estimations? Are they compatible?

c) Then select a sample from the same genre of 900 sentences. What is the estimate this time?

d) Select a sample of 100 sentences from the 'fiction' category. Use this to estimate the average sentence length in fiction. Could this sample come from the 'news' genre, in other words are the two genres different with respect to sentence length?

Delivery:
Explain how you selected the random samples.
Answer the questions and explain how you found the answers.

## Exercise 4 – Collocations

To study collocations seriously one has to use large corpora. To simplify, we use a corpus where many phrases have a tendency to reappear, the Inaugural corpus supplied with NLTK. We consider the whole corpus.

We first convert the corpus to the class nltk.Text which gives us access to the method Text.collocation(). See the first pages in section 3.1 in the NLTK book.

We want more control and in particular to test out the effect of various association measures as described in chapter 5 in the FSNLP book. NLTK may also provide tools for this in the Collocation package. Unfortunately, when we move outside the NLTK book, the documentation isn't always as easy to follow, but we will try to take it stepwise. You will find some documentation at http://nltk.googlecode.com/svn/trunk/doc/howto/collocations.html

We will use two basic tools, and we start by giving them simpler names, so it becomes easier to refer to them

```
>>> Finder=nltk.collocations.BigramCollocationFinder
>>> AssocMeasures = nltk.collocations.BigramAssocMeasures
```

Finder is the basic tool we will use to find the collocations. It is a class with several useful methods. AssocMeasures will provide the various association measures.

Collocations are calculated from the frequencies of words and the frequencies of bigrams. We may build a bigram finder for the Inaugural corpus as follows.

```
>>> inaugural = nltk.corpus.inaugural
>>> tokens = inaugural.words()
>>> word_fd = nltk.FreqDist(tokens)
>>> bigram_fd = nltk.FreqDist(nltk.bigrams(tokens))
>>> inaug_finder = Finder(word_fd,bigram_fd)
```

We may then test our collocation finder and print out the top 20 results.

```
>>> scored = inaug_finder.score_ngrams(AssocMeasures.raw_freq)
>>> scored[:20]
```

As we see, many of the results are not good candidates. We will therefore do some filtering. We should remove non-words and possibly also stop words. This may done on the model of

```
      >>> inaug_finder.apply_word_filter(lambda w: not(w.isalpha()) )
```

You may check how the result has changed:

```
      >>> scored = inaug_finder.score_ngrams(AssocMeasures.raw_freq)
      >>> scored[:20]
```

You may apply apply_word_filter several times on different criteria to get the best result.

You may also consider removing low-frequent bigrams with the filter

```
      >>>finder.apply_freq_filter(<n>)
```

Where you provide a number for <n>

You are now ready to compare various association measures. In addition to the raw frequency (raw_freq) you should try

- Dice: dice
- T-score: student_t
- Chi squared: chi_sq
- Pointwise mutual information: pmi
- Log-likelihood ratio: likelihood_ratio

You should hand in

1. The 20 highest ranked collocations in order for each of the 6 tests, as well as for the Text.collocation method we started with.
2. Explain what you did to the corpus, filters and cut-off
3. Of the proposed collocations, give 5 examples you think should classify as collocations and 5 examples you think are not collocations and state short reasons for your choice.

The end