# Homework 1

In this exercise, you will train a tagger for the Estonian language. For this, I have built a largish (around five million tokens) tagged corpus of Estonian, using text from the *OpenSubtitles* corpus

[http://opus.lingfil.uu.se/OpenSubtitles2013.php (http://opus.lingfil.uu.se/OpenSubtitles2013.php)] , which is contained in the file `estonianSmall.txt` (available in GauchoSpace). This is a useful resource that contains "parallel corpora" aligned sentence by sentence for many languages (the texts themselves are movie subtitles, which are know to provide a rather good approximation of colloquial speech in terms of the frequencies of words). I have tagged this corpus myself using some techniques, therefore, in principle, it would not be impossible to attain 100% accuracy in tagging (but it will be in practice very difficult).

## Basic Task:

**Exercise 1:** Train a trigram tagger using this corpus, and estimate its tagging accuracy. The trigram model has to have backoff into a bigram (which itself backs off onto a unigram model, and finally into a default tagger).

**Exercise 2:** The second part concerns improving the accuracy testing using three-fold cross-validation.

## Advanced Task:

As above, but try getting a better performance other tagging models. You can do this either by tweaking the N-gram model, or by changing the model itself (e.g., Brill tagger, HMM, MaxEnt).

## Loading the Estonian Corpus

First, we need to load the Estonian tagged corpus. NLTK comes with default corpus readers for seveeal formats. In this case, the Estonian corpus I've provided you with comes in a format that contains one sentence per line, and the lines have the format of the example line below:

```
Aga/J midagi/P sa/P ikka/D mäletad/V ?/Z
```

Notice that in the example above, the words and the POS tags are separated by a /. The function `TaggedCorpusReader` ([https://nltk.googlecode.com/svn/trunk/doc/api/nltk.corpus.reader.tagged.TaggedCorpusReader-class.html (https://nltk.googlecode.com/svn/trunk/doc/api/nltk.corpus.reader.tagged.TaggedCorpusReader-class.html)]) from `nltk.corpus.reader` reads precisely this type of files. Hence we import the function and we read our file, placing it in the `EstonianCorpus` object variable:

In [1]:
```
from nltk.corpus.reader import TaggedCorpusReader

mypath = "/home/fermin/COURSES/"

EstonianCorpus = TaggedCorpusReader(mypath,"estonianSmall.txt",
                                    encoding="latin-1")
```

Notice that you need to change the `mypath` variable to the correct path in you computer, where you have downloaded the `estonianSample.txt` file. For instance, if you have saved the file onto your desktop, if you are using in Windows the path will look something like:

`C:\\\\Users\<your_user_name>\Desktop\\`

on a Mac it would need to look like

`/Users/<your_user_name>/Desktop/`

where is the user name you have in your machine. After reading the corpus with `TaggedCorpusReader`, the usual methods `.words`, `.tagged_words`, `.sents`, and `tagged_sents` are available to us (so we do not need to read directly from the files themselves, NLTK takes care of that for us).

# Breaking down the Corpus into Subcorpora

Firstly, we extract the tagged sentences from the corpus:

In [2]:
```
sentences = EstonianCorpus.tagged_sents()
```

Then you need to compute the approximate size of each of the three chunks that you will use for the cross-validation.

After this, you need to create in chunks a list with three lists, each of whose elements corresponds to approximately one third of the original corpus. In order to do the 3-fold cross-validation you need to repeat the process three times, each time selecting two parts of the corpus as a training set, and the remaining one as the testing set. Training the Trigram Tagger

This can be done exactly as we did in class, by sequentially training unigram, bigram and trigram taggers (but remember that you will have to do this three times, once for each training and testing corpus).

**Note:** This corpus is larger than the ones you've used before, therefore training & evaluating the models will take significantly more time.

After training, you can evaluate each model using the `.evaluate` method of the trigram tagger.

Finally, after evaluating each tagger, store the three results in a list, and compute the average, which will be the expected performance of your tagger.

**NOTE:** You can easily compute the average of a list of numbers using `numpy`'s `mean` ([http://docs.scipy.org/doc/numpy/reference/generated/numpy.mean.html (http://docs.scipy.org/doc/numpy/reference/generated/numpy.mean.html)]) method, for instance,

In [3]:
```
from numpy import mean

mean([4,3,2,4,5,7,8,10,29])
```

Out[3]:
```
8.0
```