Goals and motivation of this lab, and what to submit

Tagging is one of the basic steps in developing many Natural Language Processing (NLP) tools, and is often also a first step in starting to annotate and analyze a corpus in a new language. In this lab, we will explore POS tagging and build a (very!) simple POS tagger using an already annotated corpus, just to get you thinking about some of the issues involved. In addition, this lab demonstrates some basic functions of the NLTK3 library. NLTK (Natural Language Toolkit) is a popular library for language processing tasks which is developed in Python. It has several useful packages for tasks like tokenization, tagging, parsing, etc. In this lab, we use only very basic functions like loading the data and reading sentences, but we hope you may be inspired to look into additional aspects of NLTK on your own. As before, we have written most of the code for the lab already and included a lot of explanatory comments, but we will ask you to add a few things here and there.

1.Understanding the tsents data structure

For this lab, we consider a small part of the Penn Treebank POS annotated data. This data consists of around 3900 sentences, where each word is annotated with its POS tag using the Penn POS tagset. To access the data, our code first imports the dependency_treebank from nltk.corpus package using the command from nltk.corpus import dependency_treebank. We then extract the tagged sentences using the following command (on line 95): tsents = dependency_treebank.tagged_sents() tsents contains a list of tagged sentences. A tagged sentence is a list of pairs, where each pair consists of a word and its POS tag. A pair is just a Tuple with two members, and a Tuple is a data structure that is similar to a list, except that you can't change its length or its contents. The Python Tuple documentation (for Python 2_ or Python 36) provides a useful summary introduction to tuples. Once you've loaded lab2.py, tsents[0] contains the first tagged sentence. tsents[0][0] gives the first tuple in the first sentence, which is a (word, tag) pair, and tsents[0][0][0] gives you the word from that pair, tsents[0][0][1] its tag

2.Computing the distribution of tags
Construct a frequency distribution of POS tags by completing the code in the tag_distribution function, which returns a dictionary with POS tags as keys and the number of word tokens with that tag as values. Hint: look at the sent_length_distribution function if you aren't sure what to do here. Using plot_histogram, plot a histogram of the tag distribution with tags on the x-axis and their counts on the y-axis, ordered by descending frequency. Hint: To sort the items (i.e., key-value pairs) in a dictionary by their values, you can use: sorted(mydict.items(), key=lambda x: x[1]) Providing reverse=True as the third argument gives the result in reverse order. help(sorted) is your friend.

3.Computing the conditional distribution of tags
For each word Construct a conditional frequency distribution (CFD) by completing the code in the word_tag_distribution function. A CFD is a dictionary whose values are themselves distributions, keyed by context or condition. In our case we want words as conditions == keys, with values a frequency distribution of tags for that word. For example, for the word book, the value of your CFD should be a frequency distribution of the POS tags that occur with book. Uncomment the code at the bottom of the file once you've implemented the function to see the tags for book.

4.Building a Unigram Tagger

We shall now build a simple POS tagger called a unigram tagger using the function unigram_tagger. This function takes three arguments: · The first argument is a conditional frequency distribution, which can be generated using the word_tag_distriution you completed above. · The second argument is the most frequent POS tag in the corpus. · The third argument is a sentence that needs to be tagged. The goal of this function is to tag the sentence using probabilities from the CFD and most frequent POS tag. In order to make it work, you must complete its helper function, called ut1, to process a single word. If the word is seen (present in the CFD), ut1 should assign the most frequent tag for that word. For unseen words (not present in the CFD), it should assign the overall most frequent POS tag, as passed in as the 2nd argument.

**This lab is adapted from the university of edinburgh school of informatics