Comp 6751 # Project 1 **Fall 2021**

Due date: October 3rd, 2021

Goal: running text preprocessing pipeline in NLTK and proofreading results

Data: NLTK version of Reuter's text `training/267`

> INDONESIA UNLIKELY TO IMPORT PHILIPPINES COPRA
>
> Indonesia is unlikely to import copra from the Philippines in 1987 after importing 30,000 tonnes in 1986, the U.S. Embassy's annual agriculture report said. The report said the 31 pct devaluation of the Indonesian rupiah, an increase in import duties on copra and increases in the price of Philippines copra have reduced the margin between prices in the two countries. Indonesia's copra production is forecast at 1.32 mln tonnes in calendar 1987, up from 1.30 mln tonnes in 1986.

Overview: Do the following project in NLTK. Get the text using the NLTK corpus access commands, do not cut and paste from the assignment. Develop a single script called *PreProcess* that executes the following pipeline in NLTK, taking the file to be preprocessed as a parameter, *PreProcess(*`training/267`*)*

1. tokenization (NLTK)
2. sentence splitting (NLTK)
3. POS tagging (NLTK)
4. number normalization
5. date recognition
6. date parsing

Proofread every step in your pipeline. To save time and gain the anticipated insight, run your script on other texts as well. Solutions that only work on this text and not on others may not get full marks.

Description: Each step in your pipeline has specific additional requirements in order to be considered satisfactory. It is important that you do not limit yourself to the requirements, but think beyond the minimum requirements for your solution.

tokenization start with a regular expression based tokenizer from NLTK. Note that you are free (and possibly required to) improve on that tokenizer for best results. Copious in-line comments in the code for your changes are essential. The enhancements have to also be summarized in the report.

sentence splitting as for tokenization, start with a NLTK module. Enhance if necessary.

POS tagging inspect your options in NLTK. Pick the best module. Do not spend time to improve the POS tagger at this time.

number normalization numbers have their own regular grammars. Number normalization counteracts bad tokenization of numbers. Adapt the tokenizer if necessary and write a number--normalizer grammar to ensure that number segments that the tokenizer split are combined again (i.e. if the tokenizer creates three tokens '1', '.', '2', let the number-normalizer create a single number '1.2') Make sure it does not confuse commas and periods that are part of a number with those that are not. Include any other typographical conventions for numbers. Bonus: create a grammar for numbers written in words (e.g. 'three thousand and twelve').

date recognition Create a CFG *DateParseCFG*, that takes a date string as input, to parse dates. Use POS information and create your own gazetteers. Dates come in different formats, including 2020/9/30, September 3rd, the fifth of November. Do not limit your grammar to these formats. Your grammar should include the nonterminals DATE (as root), DAY, MONTH, YEAR. Aim for large coverage while minimizing the false positives and the false negatives.

For all modules, create your own test cases for a more general solution.

Deliverables: to be submitted in Moodle by October 3rd, 2020

- (2pts) tokenizer used (enhancements clearly identified in inline comments. This includes the number normalizer.)
- (2pts) sentence splitter used (any enhancements clearly identified in inline comments)
- (2pts) POS tagger used (any enhancements clearly identified in inline comments)
- (2pts) *DateParseCFG*
- (2pts) Report: a .pdf document that documents your work and submitted modules
- (2pts) Demo File: a .pdf document that shows examples of input and output for all modules and all interesting cases in a single "demo" file. Organize your demonstration of your demo to be readable. Do not include repetitive example runs. You must select the most helpful runs to show strengths of your modules/grammars. Errors or limitations must be addressed openly in the Report. Note that there are no solutions without errors or limitations.

Marking scheme: Grading is based on two components. The first component is adherence to the instructions. This is expected to be 100%. The second component is what you bring to the task. We can only appreciate this, if it is well documented in the report and in the code and illustrated with convincing examples in the Demo File.

Solutions that work on a wider variety of input data are preferred over narrow solutions (coverage). Solutions that produce fewer false positives are preferred (specificity). There is no optimum and you have to explain your choices.