

**Due date:** September 30, 2020







**Goal:** running text preprocessing pipeline in NLTK and proofreading results

**Data:** 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. You should enhance the tokenizer so that it doesn't split numbers that include a comma or period (30,000, 1.32). Make sure that the "enhancements" do not confuse commas and periods that are part of a number with those that are not. Include any other typographical conventions for numbers.



**measured entity detection** units of measurements should be grouped with the numbers they usually follow. Develop your own *MeasuredEntityDetection* module that groups a number with its unit, if available (i.e. 30,000 tonnes). For this step, you should compile a unit gazetteer (best solution is to find a resource on the web and adapt its format to a simple word list). This is an open ended task.

**date recognition** 1987 is a date in text training/267. In another text, it may be a cardinal. Find a way to use POS information and a CFG *DateRecognizerCFG* to detect dates.

**date parser** dates come in different formats. Write a CFG for different date formats, including 2020/9/30, September 3rd, the fifth of November, but not limiting 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. The CFG *DateParseCFG* takes a date string as input.

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

**Deliverables:** to be submitted in Moodle before 29 September, 2020

- tokenizer used (enhancements clearly identified in inline comments. This includes the number normalizer.)
- sentence splitter used (any enhancements clearly identified in inline comments)
- POS tagger used (any enhancements clearly identified in inline comments)
- *MeasuredEntityDetection* module
- *DateRecognizerCFG*
- *DateParseCFG*
- **Report:** a .pdf document that documents your work and submitted modules
- **Demo:** 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. For in person demos, there is usually a 5min cutoff and the marker can ask for specifics. For the asynchronous, remote delivery, you must select the most helpful runs to show strengths of your modules/grammars. Errors or limitations should 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. 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.

**Resource:** for inspiration and clarification.

Using Numbers and Unit Symbols in Measurements: <http://www.ibiblio.org/units/measurements.html>

A Dictionary of Units of Measurement: <http://www.ibiblio.org/units/index.html>