Homework 1 - Text Analytics - GitHub Link

Kristiyan Dimitrov - ktd5131

Problem 1 - Code Link, Notebook Link

For this homework, I experimented with 3 packages: re, nltk, and spacy. I explored tokenization, stemming, and part of speech tagging.

In terms of tokenization, regex proved to be fastest and most straightforward to use. `re.findall('\w+', text)` took on average 54ms to tokenize a text of 1.17 M characters. On the other hand, `nltk.word_tokenize(text)` took on average 1.27 s (approximately 20x slower).

There were also important differences in the types of tokens picked up. Nltk captured hyphenated tokens such as "well-to-do" or "Cough-cough", but also picked up tokens containing a full stop such as "yes.", "hours.", etc. Regex did not include tokens with punctuation, which might be the preferred behavior depending on the task at hand. Overall, regex and nltk produced ~212k and ~257k tokens respectively, which indicates significant differences.

I found SpaCy to be a powerful library with a flexible Object Oriented structure that is easy to use, but also affords a lot of customization. It provides built in language models, which execute the following pipeline: tokenization → tagging → parsing → ner (Named Entity Recognition). I disabled all but the first step and achieved an average tokenization time of 109ms. The tagging step, however, also performs Lemmatization, so the comparison with regex and nltk, in terms of performance time, is not fair. When I re-enabled the tagging part of the pipeline, each token object received additional attributes: POS and Tag. The downside is a considerable slowdown - an average of 5.78s i.e. 1 order of magnitude.

Problem 2

Part 2.1: `pattern = r'[a-zA-Z0-9_\.\#\!\?]+@\w+\.\w+'` Code Link

Part 2.2: `pattern1 = r'\d{1,2}/\d{1,2}/\d{4}' ; pattern2 = r'\w+ \d{2}\w{2} \d{4}' ;`

`pattern3 = r'\d{2}\.\d{2}\.\d{4}' ; pattern4 = r'\d{1,2}\s[JFMASOND]\w{2}\s\d{4}`Code Link