Homework 11:
Web Crawling and POS Tagging

Florian Fink
Symbolische Programmiersprache

Due: Thursday February 02, 2021, 16:00

In this exercise you will:

- process text from the given URL
- find homographs within this text
- match sequences of POS-tags in sentences

Exercise 1: Text Processing and Homographs [6 points]

This homework will be graded using unit tests by running: `python3 -m unittest -v hw11_crawling/test_analysis.py`

Implement following methods that can process the text from the given URL:

- `get_text(html)` - creates the list of clean paragraphs (no HTML markup) from the given html string (use `BeautifulSoup`) and returns paragraphs as a string. `Hint`: join the list of paragraphs by newline.
- `get_headline(html)` - returns the headline from the given html string.
- `get_normalized_tokens(text)` - should tokenize the text with NLTK and return the list of lower case tokens without stopwords (use NLTK to remove stopwords).

Use NLTK to find all homographs within the text. We use the following definition: Distinct words that have the same written form are called homographs. In other words, homographs are words with the same spelling and different POS.

- Implement a function `get_pos_dict(tokens)` that stores mapping between words and their possible POS tags. `Hint:` use `defaultdict`, a subclass of the built-in `dict` class. Setting `defaultdict` to `set` makes the `defaultdict` useful for building a dictionary of sets.

- Implement a function `filter_dict_homographs(word_dict_h)` that deletes an entry from the dictionary, if this entry is not a homograph.
- Implement a function `find_homographs(tokens)` that returns a dictionary which holds homographs. Use already implemented methods.

Exercise 2: POS-Tags [10 Points]

This homework will be graded using unit tests by running: `python3 -m unittest -v hw11_crawling/test_pos_match.py`

In this exercise, you will write a program to match sequences of POS tags in sentences. Download the file `hydrogenics_report.txt` into the `data/` folder of your project. Take a look at the file `hw11_crawling/pos_match.py`. Implement the remaining unimplemented methods to make it work:

- `Sentences.from_file(cls, path)` – reads the file at path, tokenizes the sentences (use NLTK), pos-tags the sentences and returns a new instance of the `Sentences` class. `Hint:` The constructor of `Sentences` expects a list of tagged sentences (each sentence being a list of pos-tagged words). [2 points]
- `PosExpr.from_string(cls, expr)` – creates an instance of PosExpr from a string expression. `Hint:` The constructor of `PosExpr` expect a list of strings; take a look a the tests to see how the function is used. [0 points]
- `PosExpr.match_expr(expr, pos)` – returns True if `expr` matches `pos`. An expression *XX* matches the pos-tag *XX*, the expression * matches any pos-tag and an expression *XX** matches the pos-tags *XX*, *XXY*, ... For example *NN** should return True for the tags *NN*, *NNP* and *NNPS*. [2 points]
- `PosExpr.matches(sentence)` – returns a list of matches in the given sentence (list of (word,pos)-pairs). A match is a list of (word, pos)-pairs, where the tags in the sentence matched the expression mask provided by PosExpr for all possible positions. For example given `p=PosPattern.from_string("X Y")`, `p.matches([(a,X),(b,Y),(c,Z),(d,X),(e,Y)])` should return the list `[[(a,X),(b,Y)],[(d,X),(e,Y)]]`. [4 points]
- `find(sentences, expr)` – returns a list of strings (not the (word,pos)-pairs) that match the given expression in all sentences. For example `find_string(sentences, "JJ NN")` should return the flat list `[...,"prior year",...]`. [2 points]