Wikipedia

February 2, 2024

1 Structure and Text in Wikipedia

In this assignment, you will use regular expressions to process the source wiki of Wikipedia articles with the purpose of:

1. Extracting useful information from structures such as piped links and category links.
2. Extract the table of contents of the article.
3. Extract a clean version of the text that can be used for NLP.
    - This is done by removing references, infoboxes, pictures, and categories.
    - Piped links *[[string1|string2]]* would need to be replaced with the surface string *string2*.

1.1 Write Your Name Here:

2 Submission Instructions

1. Click the Save button at the top of the Jupyter Notebook.
2. Please make sure to have entered your name above.
3. Select Cell -> All Output -> Clear. This will clear all the outputs from all cells (but will keep the content of ll cells).
4. Select Cell -> Run All. This will run all the cells in order, and will take several minutes.
5. Once you've rerun everything, select File -> Download as -> PDF via LaTeX and download a PDF version *wikipedia.pdf* showing the code and the output of all cells, and save it in the same folder that contains the notebook file *wikipedia.ipynb*.
6. Look at the PDF file and make sure all your solutions are there, displayed correctly. The PDF is the only thing we will see when grading!
7. Submit **both** your PDF and notebook on Canvas.
8. Make sure your your Canvas submission contains the correct files by downloading it after posting it on Canvas.

2.1 Warmup exercises (45p)

Create regular expressions for the following languages. Implement each regular expression in Python and test it on the positive and negative examples provided.

1. [15p] Sequences (possibly empty) of **letters or digits**.
    - Positive: *40g3Hs5*, *fh39fjs745kHyGJ798*, …
    - Negative: *ds&4H*, *J4%DE#*, …
2. [15p] Sequences (possibly empty) of **letters and digits where a lower case letter is always folowed by an even digit and an uppercase letter is always followed by an odd digit**.

   - Positive: *R1G34*, *49F77f83g6h4J34k89H3*, *92*, …
   - Negative: *j*, *jJ*, *J6j*, *Y5F7J65g6*, …
3. [15p] Sequences of characters (any kind) that **start with a capital letter and end with the same capital letter**.
   - Positive: *H%sm34H*, *KfdhiHJHK3sj^hgK*, …
   - Negative: *3kdj4*, *h34h*, *H34h*, *#hfh#*, …

[ ]:
```
import re

# YOUR CODE HERE
p1 = re.compile("")




# YOUR CODE HERE
p2 = re.compile("")





# YOUR CODE HERE
p3 = re.compile("")
```

2.2 Wiki source processing

Read the source of the Wikipedia article. For debugging purposes, you may consider using a shorter article first.

[ ]:
```
# You can use this shorter article first.
# fname = '../data/FM-2030.txt'
fname = '../data/University_of_North_Carolina_at_Charlotte.txt'
source = open(fname, 'r', encoding = 'utf-8').read()
```

Task 1.a (20 points) Design a regular expression *piped* that matches piped strings of the type *[[string1|string2]]*. Use parantheses to group string1 and string2, such that *piped.findall(source)* returns them in a tuple *(string1, string2)*. For example, when run on the source of the UNCC article, the code below should result in a list that starts as *[('Public university', 'Public'), ('University of North Carolina', 'UNC System'), …]* and that contains 44 elements.

[ ]:
```
import re

piped_re = re.compile(r'YOUR RE GOES HERE')
mp = piped_re.findall(source)
print('Found', len(mp), 'piped links.')
print(mp)
```

Task 1.b (15 points)Design a regular expression *categ* that matches category strings of the type *[[Category:name]]*. Use parantheses to group the name part. When run on the source of the UNCC article, the code below should result in a list that starts as *['University of North Carolina* *at Charlotte|', 'Educational institutions established in 1946', ...]* and that contains 6 elements.

```
[ ]: categ_re = re.compile(r'YOUR RE GOES HERE')
     mc = categ_re.findall(source)
     print('Found', len(mc), 'categories.')
     print(mc)
```

Task 2 (15 points) Extract the table of contents of the article, i.e. a list of all the section titles in the article. When run on the UNCC article, it should find 33 section titles.

```
[ ]: title_re = re.compile(r'YOUR RE GOES HERE')
     mt = title_re.findall(source)
     print('found', len(mt), 'titles.')
     for title in mt:
         print(title)
```

Task 3.a (20 points) Design a regular expression *ref_re* that matches reference strings enclosed between reference tags "<ref ...> ... </ref>" so that they can be eliminated from the document. Beware also of the alternative form "<ref .../>".

```
[ ]: ref_re = re.compile(r'YOUR RE GOES HERE')
     mref = ref_re.findall(source)
     print('found', len(mref), 'references.')
     for ref in mref:
         print(ref)
```

Remove all references from the source string.

```
[ ]: source = ref_re.sub('', source)
     print(source)
```

Task 3.b (20 points) Replace all piped links [[string1|string2]] and [[string2]] with the surface string string2.

```
[ ]: pip_re = re.compile(r'YOUR RE GOES HERE')
     mpip = pip_re.findall(source)
     print('found', len(mpip), 'piped links.')
     for pip in mpip:
         print(pip)

     source = pip_re.sub('', source)
     print(source)
```

Task 3.c (15 points) Design a regular expression file_re that matches file strings of the type *[[File: ...]]*. Use the regular expression to remove all file strings from the source.

```
[ ]: file_re = re.compile(r'YOUR RE GOES HERE')
     mfile = file_re.findall(source)
     print('found', len(mfile), 'file links.')
     for file in mfile:
```

```
    print(file)

source = file_re.sub('', source)
print(source)
```

Task 3.d (15 points) Use a regular expression to remove all category links from the source.

[ ]:
```
categ_re = re.compile(r'YOUR RE GOES HERE')
mc = categ_re.findall(source)
print('Found', len(mc), 'categories.')
print(mc)

source = categ_re.sub('', source)
print(source)
```

Task 3.e (20 points) *Mandatory for graduate students, optional (bonus points) for undergraduate students*

- Remove all templates and infoboxes from the source document.
  - These are any strings of the type '{{ … }}'
  - Beware that there can be multiple levels of nesting, e.g. '{{ … {{ .. {{ …. }} .. }} … }}'. This cannot be matched with regular expressions (explain why).

[ ]:
```
def remove_templates(s):
    # YOUR CODE GOES HERE
    return s

source = remove_templates(source)
print(source)
```

Task 3.f (20 points) *Mandatory for graduate students, optional (bonus points) for undergraduate students*

Design a regular expression that finds all occurences of integer numbers in an input strings and uses a substitution to replace them with the equivalent real numbers by appending '.0' to them. Use it to implement the function `realize(s)` below.

[ ]:
```
def realize(s):
    # YOUR CODE HERE


# This should print 'When we add 4.0 to 1.5 and 0.5 to -2.5 we end up with 5.5␣
↪and -2.0.'
print(realize('When we add 4 to 1.5 and 0.5 to -2.5 we end up with 5.5 and -2.
↪'))
```

Task 3.g [10 + 10 Bonus points]

- Design a search-and-replace function based on the `sub()` method for regular expressions that finds all occurences of integer numbers in an input strings and replaces them with an incremented version. Use it to implement the function `increment(s)` below.

- Design a search-and-replace function based on the `sub()` method for regular expressions that finds all whitespace separated tokens (like `str.split()`) and replaces them with a copy that has _ appended to it.

Hint: Read the documentation on the `sub()` function to see how to use it with a function argument.

[ ]:
```
def increment(s):
    # YOUR CODE HERE




# This should print 'Eve has 6 apples. She gives 4 to Adam and the remaining 3␣
 ↪to the snake.'
print(increment('Eve has 5 apples. She gives 3 to Adam and the remaining 2 to␣
 ↪the snake.'))

def underscore(s):
     # YOUR CODE HERE



# This should print 'Eve_ has_ 5_    apples._'
print(underscore('Eve has 5    apples.'))
```

Task 4 [Bonus points] Anything extra goes here. For example:

- Look at the output of the spaCy tokenizer on some text and find instances where it makes certain types of mistakes. Propose and implement regular expressions that fix one or two types of mistakes.
  - It might help to look at the code for the spaCy tokenizer.
- Design and test regular expressions for extracting other types of structures from Wiki text.

[ ]:
```

```