PythonAssignment2

February 12, 2020

1 Q1

Define three strings:
* `sent1 = "It's a beautiful day in the neighborhood, please won't you be my beautiful neightbor?"`
* `sent2 = "We are inside the world of beautiful people, but that's not who we are; we are not beautiful!"`
* `query`

The value of `query` string should be input by the user and be composed of one to three words of your choice. You can use the `input` function, e.g. `query = input("enter your query\n")`

Turn `sent1`, `sent2`, and `query` to sets of words and store them into variables `setA`, `setB`, and `setC`, respectively.

Hint: you need to split the string into a list of words and turn the result into a set. Use the `split()` method on the string and the `set` function on the resulting list. If you don't turn the string into a list first and apply the the `set` function to the string, the result will be a set of the caharacters rather than of the words. For this assignment, we're interested in sets of words.

Print the three sets

```
In [ ]:
```

2 Q2

Sets have methods defined on them for performing set operations. Assume we have two sets, `A` and `B`, we can find their intersection by calling the intersection method, `A.intersection(B)` or, equivalently, `B.intersection(A)`, since interection is commutative. The same applies to set union with a method `union()`.

Find the intersection and union between the set versions of `query` and `sent1` (i.e. `setC` and`setA`) and store the cardinalities of the resulting sets in variables `insct1` and `union1`, respectively. Note that set cardinality is the number of elements in the set. You can use the `len` function for this.

Find the intersection and union between the set versions of `query` and `sent2` (i.e. `setC` and `setB`) and store the cardinalities of the resulting sets in variables `insct2` and `union2`, respectively.

Now calculate how similar `query` is to `sent1` and how similar `query` is to `sent2` using a measure called the Jaccard index:

$$\text{Jaccard(A,B)} \quad = \quad \frac{|\, A \cap B \,|}{|\, A \cup B \,|}$$

Having calculated the values for this measure and stored them in `insct1`, `union1`, ..., apply this index measure to calculate the similarity of `query` to `sent1` and store the value in `jaccard1`. Calculate the similarity of `query` to `sent2` and store the value in `jaccard2`.

Print the sentence that has the higher jaccard value using the `if` statement.

```
In [ ]:
```

3 Q3

Write a function named `jaccard` that takes two **strings** and calculates the Jaccard index for them.

```
In [ ]: def jaccard(query, doc):
            return none
```

4 Q4

Retrieve a sentence in the `reuters` corpus (e.g. the first sentence. The example below retrieves the 67th sentence in the corpus). Call the `jaccard` function you wrote in **Q3** to calculate the Jaccard index for `query` with this sentences. Note that `sents()` method on a corpus gives you the sentences in the corpus, where each sentence is stored as a *list* of words. This means that when you call the function, you should convert the corpus sentence to a string, because the function you defined earlier takes two strings. you can use the `join()` method to perform this conversion from list to string. Make a tuple of this sentence and the Jaccard index and print it.

```
In [1]: import nltk
        nltk.download('reuters')
        nltk.download('punkt')
        from nltk.corpus import reuters

        sents = reuters.sents()
```

```
[nltk_data] Downloading package reuters to C:\nltk_data...
[nltk_data]   Package reuters is already up-to-date!
[nltk_data] Downloading package punkt to C:\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

```
In [3]: my_sent = sents[67]

        print(my_sent)
```

```
['The', 'fledgling', 'exchange', 'currently', 'trades', 'coffee', 'and', 'rubber',
```

```
In [ ]:
```