PythonAssignment3

March 12, 2020

```
In [ ]: import nltk
        from nltk.corpus import reuters
        import re
```

1 Q1

Retrieve all the sentences in the reuters corpus of *three* genres of your choice and store them in a variable named `docs`. Note that `sents()` method on a corpus gives you the sentences in the corpus, where each sentence is stored as a *list* of words. Convert the list of lists that you retrieve into a list of strings. This step will prove handy for the next question. Remember that you can convert a sentence that is represented as a list of words into a string using `join` method.

You can find out about the genres in a corpus using the `categories()` method on it. Here's a list of some categories in reuters

`acq, alum, barley, bop, carcass, castor-oil, cocoa, coconut, coconut-oil, coffee, copper, copra-cake, corn, cotton, cotton-oil, money-supply, naphtha, nat-gas, nickel, nkr, nzdlr, oat, oilseed, orange, palladium, palm-oil, palmkernel, pet-chem, platinum, potato, propane, rand, rape-oil, rapeseed, trade, veg-oil, wheat, wpi, yen, zinc`

```
In [ ]:
```

2 Q2

Prompt the user to enter a query that will be matched against all the sentences in `docs` in order calculate how similar each sentence is to the query. Store the user's query as a string in a variable `query`.

To find similarity, calculate the Jaccard index for all the sentences in `docs` with `query` and store each sentence with its jaccard value in a tuple. Store the tuples in a list.

A suggested version of the `jaccard` function is provided below. Feel free to use this one or the function you developed last time.

```
In [ ]: def jaccard(query, doc):
            query = set(query.split())
            doc = set(doc.split())
            # to avoid division by zero check that the union does not yield the emp
```

```
    if len(query.union(doc)) > 0:
        return len(query.intersection(doc))/len(query.union(doc))
    else:
        return 0
```

3 Q3

Display the sentences with the top ten jaccard scores along with their scores.

In [ ]:

4 Q4

Repeat what you did in Q3 but this time the top ten sentences should have the query terms highlighted in them. For example, for a query `'paper storage'` and a sentence `'The paper blamed the waste on inadequate storage and bad preservation methods .'`, the sentence should be displayed as

```
The <paper> blamed the waste on inadequate <storage> and bad
preservation methods .
```

One way to accomplish this is using the `sub` method on regular expressions. Here's an example:

```
s = 'The paper blamed the waste on inadequate storage and bad
preservation methods .'
sm = re.sub(r'(paper|storage)', r'<\1>', s)
```

the `\1` allows you to reference the group that was matched and it is surrounded by two angle brackets because this is how we chose to highlight our query terms. If we wanted to hghlight our query terms in the sentence with say asterisks, here's how the `sub` call would look:

```
sm = re.sub(r'(paper|storage)', r'*\1*', s)
```

Accordingly, the string will be dispalyed as follows:

```
The *paper* blamed the waste on inadequate *storage* and bad
preservation methods .
```

For this question, you will need to construct the first regular expression from `query`. Hint: you can use the join method and string concatenation.

In [ ]: