CSE 4334/5334 Programming Assignment 1 (P1)

Fall 2018

Due: 11:59pm Central Time, Friday, November 2, 2018

The instructions on this assignment are written in an .ipynb file. You can use the following commands to install the Jupyter notebook viewer. "pip" is a command for installing Python packages. You are required to use "Python 3.6.x" (any version of Python equal to or greater than version Python 3.6.0.) in this project.

```
pip install jupyter
```

To run the Jupyter notebook viewer, use the following command:

```
jupyter notebook P1.ipynb
```

The above command will start a webservice at http://localhost:8888/ (http://localhost:8888/) and display the instructions in the '.ipynb' file.

Requirements

- This assignment must be done individually. You must implement the whole assignment by yourself. Academic dishonety will have serious consequences.
- You can discuss topics related to the assignment with your fellow students. But you are not allowed to discuss/share your solution and code.

Assignment Files

All the files for this assignment can be downloaded from Blackboard ("Course Materials" > "Programming Assignments" > "Programming Assignment 1 (P1)" > "Attached Files").

1. This instruction file itself "P1.ipynb"

2. Data file "debate.txt"

We use the transcript of the latest Texas Senate race debate between Senator Ted Cruz and Representative Beto O'Rourke, which took place on October 16, 2018. We pre-processed the transcript and provide you a text file debate.txt. In the file each paragraph is a segement of the debate from one of the candidates or moderators.

3. Sample results "sampleresults.txt"

4. Grading rubrics "rubrics.txt"

Programming Language

1. We will test your code under the particular version of Python 3.6.x. So make sure you develop your code using the same version.
2. You are free to use anything from the Python Standard Library that comes with Python 3.6.x (https://docs.python.org/3.6/library/ (https://docs.python.org/3.6/library/)).
3. You are expected to use several modules in NLTK--a natural language processing toolkit for Python. NLTK doesn't come with Python by default. You need to install it and "import" it in your .py file. NLTK's website (http://www.nltk.org/index.html (http://www.nltk.org/index.html)) provides a lot of useful information, including a book http://www.nltk.org/book/ (http://www.nltk.org/book/), as well as installation instructions (http://www.nltk.org/install.html (http://www.nltk.org/install.html)).
4. **You are NOT allowed to use any non-standard Python package, except NLTK.**

Task TF-IDF and Cosine Similarity

1. Description of Task

You code should accomplish the following tasks:

(1) **Read** the text file debate.txt. This is the transcript of the latest Texas Senate race debate between Ted Cruz and Beto O'Rourke. The following code does it.

In [7]:
```
import os
filename = './debate.txt'
file = open(filename, "r", encoding='UTF-8')
doc = file.read()
file.close()
```

(2) **Tokenize** the content of the file. For this, you need a tokenizer. For example, the following piece of code uses a regular expression tokenizer to return all course numbers in a string. Play with it and edit it. You can change the regular expression and the string to observe different output results.

For tokenizing the Texas Senate debate transcript, let's all use RegexpTokenizer(r'[a-zA-Z]+'). What tokens will it produce? What limitations does it have?

In [ ]:
```
from nltk.tokenize import RegexpTokenizer
tokenizer = RegexpTokenizer(r'[A-Z]{2,3}[1-9][0-9]{3,3}')
tokens = tokenizer.tokenize("CSE4334 and CSE5334 are taught together. IE
3013 is an undergraduate course.")
print(tokens)
```

```
['CSE4334', 'CSE5334', 'IE3013']
```

(3) Perform **stopword removal** on the obtained tokens. NLTK already comes with a stopword list, as a corpus in the "NLTK Data" (http://www.nltk.org/nltk_data/ (http://www.nltk.org/nltk_data/)). You need to install this corpus. Follow the instructions at http://www.nltk.org/data.html (http://www.nltk.org/data.html). You can also find the instruction in this book: http://www.nltk.org/book/ch01.html (http://www.nltk.org/book/ch01.html) (Section 1.2 Getting Started with NLTK). Basically, use the following statements in Python interpreter. A pop-up window will appear. Click "Corpora" and choose "stopwords" from the list.

In [ ]:
```
import nltk
nltk.download()
```

```
showing info https://raw.githubusercontent.com/nltk/nltk_data/gh-pages/
index.xml
```

After the stopword list is downloaded, you will find a file "english" in folder nltk_data/corpora/stopwords, where folder nltk_data is the download directory in the step above. The file contains 179 stopwords. nltk.corpus.stopwords will give you this list of stopwords. Try the following piece of code.

In [ ]:
```
from nltk.corpus import stopwords
print(stopwords.words('english'))
print(sorted(stopwords.words('english')))
```

(4) Also perform **stemming** on the obtained tokens. NLTK comes with a Porter stemmer. Try the following code and learn how to use the stemmer.

In [ ]:
```python
from nltk.stem.porter import PorterStemmer
stemmer = PorterStemmer()
print(stemmer.stem('studying'))
print(stemmer.stem('vector'))
print(stemmer.stem('entropy'))
print(stemmer.stem('hispanic'))
print(stemmer.stem('ambassador'))
```

(5) Using the tokens, compute the **TF-IDF vector** for each **paragraph. In this assignment, for calculating inverse document frequency, treat debate.txt as the whole corpus and the paragraphs as documents.** That is also why we ask you to compute the TF-IDF vectors separately for all the paragraphs, one vector per paragraph.

Use the following equation that we learned in the lectures to calculate the term weights, in which t is a token and d is a document (i.e., paragraph):

$$w_{t,d} = (1 + log_{10} tf_{t,d}) \times (log_{10} \frac{N}{df_t}).$$

Note that the TF-IDF vectors should be normalized (i.e., their lengths should be 1).

Represent a TF-IDF vector by a dictionary. The following is a sample TF-IDF vector.

In [ ]:
```
{'sanction': 0.014972337775895645, 'lack': 0.008576372825970286, 'regre
t': 0.009491784747267843, 'winter': 0.030424375278541155}
```

(6) Given a query string, calculate the query vector. Compute the **cosine similarity** between the query vector and the paragraphs in the transcript. Return the paragraph that attains the highest cosine similarity score. In calculating the query vector, the vector is also to be normalized.

2. What to Submit

Submit through Blackboard your source code in a single .py file. You can define as many functions as you want, but the file must define at least the following functions:

- **getidf(token)**: return the inverse document frequency of a token. If the token doesn't exist in the corpus, return -1. The parameter 'token' is already stemmed. (It means you should not perform stemming inside this function.) Note the differences between getidf("hispan") and getidf("hispanic").
- **getqvec(qstring)**: return the query vector for a query string.
- **query(qstring)**: return the paragraph in the transcript that has the highest cosine similarity score with respect to 'qstring'. Return its score too. The output format should be as follows.

the paragraph

the score

If all paragraphs have zero scores, return the following message.

```
No Match

0.0000
```

3. Sample Results

The file "sampleresults.txt" provides multiple sample results of calling the aforementioned three functions.

We use a script to automatically grade. Make sure your code produces identical results in idential format. Or else you won't get points. And apparently make sure your program runs.

4. Grading Rubrics

Your program will be evaluated on correctness, efficiency, and code quality.

Make sure to thoroughly understand the grading rubrics in file "rubrics.txt".