CS69001 Computing Laboratory – I

Assignment No: D1

Date: 27–August–2018

Python programming assignment

In this assignment you will implement two types of text-summarization algorithms. These algorithms take as input a text document that contains a number of sentences, each of which contains several words. The output is a summary of a given number of sentences. The summarization algorithms that you will implement, generate the summary by selecting some of the sentences in the document, that are deemed to be important (this process is known as 'extractive' summarization).

For this assignment, you will need to use the Python modules: **`NLTK`** (for text processing), **`NetworkX`** (for graph processing), and **`Numpy`** (for numerical computations, such as matrix operations). Before implementing the summarization algorithms, you should perform some initial tasks as described below.

Part 1: Preprocessing the sentences

1. Segment the document into sentences, and store the sequence of sentences in a list. The first sentence in the document should be the first element of the list, the second sentence should be the second element of the list, and so on.
2. Preprocess each sentence (list element).
   - Remove punctuation symbols (the method **`translate`** can be used).
   - Convert all English letters in the sentence to lowercase.
   - Tokenize the sentence into words.
   - Remove a set of very common words (called stopwords). You can use the inbuilt list of stopwords provided by NLTK, that can be accessed by:

     ```
     from nltk.corpus import stopwords
     stop_words = set(stopwords.words('english'))
     ```
   - Lemmatize, that is, reduce the different inflectional variants of a word to the dictionary form of the word (use **`WordNetLemmatizer`**)

At the end of this task, each element of the List should be a pre-processed sentence, in the same order as in the input document. For example, consider a toy document containing two sentences:

```
I was walking down the street.  I met my friends Rayan, George and Seni.
```

After the task, the contents of the list should be:

```
List[0] = i walk down street
List[1] = i meet my friend rayan george seni
```

Part 2: Sentence representation

Represent a sentence as a vector of the TF-IDF weights of its constituent words. TF denotes term frequency, that is, how frequently a word appears in a document. IDF denotes inverse document frequency, which measures how important a word is.

1. Compute TF-IDF of each word w in a given sentence s.
   - $\text{TF}(w,s) =$ Number of times the word w occurs in the given sentence s.
   - $\text{IDF}(w) = \log(\text{Total number of sentences}/\text{Number of sentences with } w \text{ in it})$.
   - TF-IDF of the word w in s is $\text{TF}(w,s) \times \text{IDF}(w)$.

2. Construct a TF-IDF matrix for the whole document, where the rows are the words, and the columns are the sentences, and the (i, j)-th entry in the matrix denotes the TF-IDF value of Word i in Sentence j.

Thus, each column of the matrix given the representation of a sentence. This representation is called the TF-IDF vector of the sentence.

Note: There are built-in methods for computing TF-IDF vectors in Python. However, you should implement the above steps yourself, without using any such built-in Python method.

Part 3: Implement matrix-based summarization algorithm

The TF-IDF matrix is very sparse. To reduce the dimensions of the matrix while preserving the semantics associated with it, we perform *Singular Value Decomposition* (SVD) and *Latent Semantic Analysis* (LSA).

SVD on the TF-IDF matrix results in 3 matrices—the left singular matrix u, the diagonal matrix of non-negative singular values s, and the right singular matrix (v) transposed v^{t}. LSA(K) on the matrix v^{t} essentially means choosing the top K rows of v^{t}. Each row $k \in K$ is supposed to represent a topic/concept in a lower-dimensional latent space. Extracting n highest valued columns (sentences) from each row $k \in K$ results in each concept being represented majorly by n sentences. Here, we will perform LSA$(K = 5)$ with $n = 3$, on the matrix v^{t}.

1. Perform Singular Value Decomposition on the TF-IDF matrix (you can use the `np.linalg.svd` method).
2. Perform LSA on the matrix v^{t} by:
   - First choosing the first 5 rows of the matrix ($K = 5$),
   - For each row $k \in K$, extract the top $n = 3$ largest valued columns (sentences).

Output the resulting 15 sentences as a summary of the document. Note that the in the final output summary, the sentences must be arranged in the same order in which they appear in the input document.

Part 4: Implement graph-based summarization algorithm

Create an undirected, weighted graph where:

- The vertices will be the sentences (represented by sentence ids / indices of the List you constructed in Part 1).
- There will be an edge between all vertex pairs (complete graph).
- The weight of the edge (u, v) between the nodes u and v will be the cosine similarity between the TF-IDF vectors of the sentences u and v (as computed in Part 2).

Compute PageRank of each node (sentence) in the graph. PageRank is a measure of the importance of a node in a graph. The underlying assumption is that more important nodes are likely to receive links from other important nodes.

Display the top 15 nodes (sentences) having the highest PageRank values as the summary. Note that the in the final output summary, the sentences must be arranged in the same order in which they appear in the input document.

Submit a single python source file implementing all the parts.