CS 410 Software Documentation

1) An overview of the function of the code (i.e., what it does and what it can be used for).

The extractive article summarizer is largely divided into 3 steps.

1. Read an input article text file.
2. Summarize the article using the extractive method (word2vec, PageRank).
3. Output a text file with the result.

This python script can be used to summarize the article. Users can specify the number of sentences they want as output. Also, users can view the evaluation of the summarization. Rouge is used as an evaluation metric.

The ExtractiveSummarizer class consists of 6 functions.

1. read_textfile(): Read the article text file to summarize the specified by the user
2. write_textfile(): Write a summarized article to a specified file
3. get_similarity(): Calculate the similarity between two given sentences.
4. calculate_k(): Calculate the constant K
5. evaluate(): Evaluate the summarized article
6. generate_summary(): Generate summarized article

2) Documentation of how the software is implemented with sufficient detail so that others can have a basic understanding of your code for future extension or any further improvement.

The whole summarization works by calling a single function, generate_summary().

First, it reads the article text file to summarize. Then, it creates a similarity matrix by calculating the similarity between each sentence. To explain, it first pre-processes each word in a sentence. Then, it converts words in each sentence to a distinct vector. After word vectorization, it calculates the cosine similarity using the cosine distance between two vectors. We calculate the similarity for every sentence, which results in getting a similarity matrix.

Given the similarity matrix, it first generates a graph by using the NetworkX library. Each sentence in the matrix is treated as a single node in a graph and edges indicate the connection between sentences. Next, it gets the PageRank of the nodes in the graph using NetworkX pagerank function. This results in getting ranked sentences as an output. In other words, we know which sentence is more relatively important than other sentences. Using this information, we can select the top K sentence that users want to have as a result. If the user does not specify the number of summarized sentences they want, it calculates the K by the number of sentences of the input text.

Furthermore, if the user wants to evaluate the summarization result, it outputs the precision, recall, and f-score of the summarization. Furthermore, it uses the ROUGE metric with 3 different evaluation metrics. ROUGE-1 refers to the overlap of a unigram, ROUGE-2 refers to the overlap of a bigram, and lastly, ROUGE-L refers to the longest common subsequence-based statistics.

3) Documentation of the usage of the software including either documentation of usages of APIs or detailed instructions on how to install and run a software, whichever is applicable.

First, clone the GitHub repository: https://github.com/djeong20/CS410-Final-Project

Next, run the python script with the following format.
summarizer.py [-h] [--i INPUT_FILE] [--o OUTPUT_FILE] [--k K] [--eval EVAL]

The program receives 4 optional arguments.

1. Input file: A path to the article text file that the user wants to be summarized. If not specified, the sample article will be used.
2. Output file: A path to the output text file that summarized the output file could be stored. If not specified, a text file named "summarized_sample_article.txt" is created in the current directory.
3. Parameter K: the number of desired summarized sentences. If the user doesn't specify K, it selects the appropriate amount of sentences to output using the length of the document.
4. Evaluation: Boolean value to indicate whether to display the evaluation of the summarization or not.

System Requirements

1. Python3
2. Numpy
3. Nltk
4. NetworkX
5. Rouge

4) Brief description of contribution of each team member in case of a multi-person team.

Donghyeon Jeong (NetID: djeong20): Responsible for implementing the extractive summarization method.

1. Find text source
2. Implementation of Extractive Text Summarization
    a. Parsing and Preprocessing (2 hrs)
    b. Ranking (12+ hrs)
    c. Threshold Tuning (5 hrs)
3. Evaluation of Extractive Summarizer (3-4 hrs)
4. Project Proposal, Progress Report, and Software Documentation for Extractive Summarizer