CS 410 Project Documentation

Christopher Dimitri Sastropranoto

Project Goal

The goal of the project was to build a google chrome extension that can summarize a CNN article.

Architecture

The initial plan was to host a python API that would accept GET requests from the extension. The extension would pass in the raw text to the API and get the summarized text back. I was unfortunately unable to successfully host the python server due to time constraints and a lack of experience in using either Django or Flask. In the end I worked around this issue by having all the code to summarize text live in the chrome extension.

Implementation

The project code is mainly split into two parts. I first did some exploratory work on how to do extractive text summarization using the NLTK python library in a Jupyter Notebook environment. The notebook can be found in the *exploration* folder. The next step was to actually build the chrome extension. The code for the extension lives in the *chrome-extension* folder.

Folder Structure

exploration

- test-article.txt —> Test article used for exploratory purposes
- text_summarization_exploration.ipynb —> Jupyter notebook

chrome-extension

- manifest.json —> Configuration file for chrome extension
- content.js —> Code for scraping the page content and summarizing it
- user-interface.js —> Code to hand UI interactions

user-interface.css —> Stylesheet for extension.

user-interface.html —> HTML file for extension UI.

icon.png —> Icon for extension.

Summarization Algorithm

The program relies on extractive summarization to summarize text. In extractive summarization, each sentence is given a score and the highest scoring sentences are used in the summary. This means that the summary is only limited to the content in the text. Sentences are scored based on the frequency counts of each of their non-stop word terms.

1. Tokenize words and sentences
2. Remove stop words from tokenized word list.
3. Calculate the frequency of each word and put into a dictionary. The dictionary keys are the words and the values are the counts.
4. Normalize the counts in the dictionary from step 2.
5. Calculate the score of each sentence. The score is calculated by adding up the scores of each word in a sentence.
6. Sort sentences by descending order of score.
7. Take the top N sentences by score but preserve their order.
8. Output step 7 as the result.

Improvements

- Incorporate inverse document frequency into the scoring function.
- Use abstractive summarization instead of extractive summarization. Abstractive summarization uses deep learning models to decipher the meaning of the passage and generate new sentences for the summary.

Running the program

Refer to the instructions provided in the README.MD section.