“Intelligent Browsing for News” Chrome Extension Documentation
Shreya Rao

Overview of Code Function

This chrome browser extension parses text from news articles on the Associated Press US News Feed and uses it to recommend relevant articles to the user for further reading. It does this by providing a pop-up button in the extension which can be clicked on to open a separate window containing links to relevant articles. The purpose of this extension is to allow people to gain extra information on a topic when they read about it, rather than getting all their information on a topic from one place.

Implementation Detail

There are two parts to this implementation: the chrome extension front end, which is built in HTML, CSS, and Javascript, and the Python backend, which is implemented using Flask.

The front end follows a similar pattern as most Chrome extensions. It includes the following files:

- Manifest.json: This file lays out the basic structure and requirements for the extension. It specifies which background script will be running while the extension is activated, the permissions the extension has, and the default browser action when the extension is activated. In my case, this is a popup window.
- Popup.html: This file shows the basic html layout of the popup that comes up when the extension is clicked on. It includes a reference to its corresponding javascript file, which is run when the button in the popup is clicked on.
- Popup.js: This file runs the main logic file when the popup button is clicked on.
- Background.js: This code includes the event listener which is always running in the background when the extension is activated. It is triggered when any sort of message (in our case, the api call) is sent.
- Content.js: The main logical file which connects the front and back end of the extension. This code uses HTML tags to scrape the text data of the article (getArticleDetails) and uses it as the body of a POST API call to our Flask backend. The results of that API call are passed back to this file as a JSON object (fetchResource). The JSON object is then formatted into a list of titles and corresponding urls in HTML format and fed into a new window which is opened when the script is called (addRelevantArticles).
- Icon.png and styles.css: Specify the styling of the extension’s html components

The back end is where the core of the Intelligent Browsing logic lies. It is in the form of a Flask Application in one file: app.py. The logic is as follows:

1. We build a list of documents to use for our TF-IDF calculations. To do this, we use BeautifulSoup to scrape the data from the AP US News Feed url. For each article found on the US News Feed, we extract the link to that article and use BeautifulSoup again to scrape the text data in each article. This is done in a for-loop: for every paragraph extracted from the content of the article, we add it to a string representation of the

contents of the entire article. Finally, the content string is appended to our documents array, so we have a list of documents.

2. We use TF-IDF calculations to extract keywords from our article. This article document is the body of the API POST request that we got from the front end, so it can be referenced in the flask app as request.json['body'].
    a. First, the documents in the documents array are pre-processed to remove special characters and stop words that are too common to be significant. The stopwords came from a stopwords.txt file that is commonly used on open source projects.
    b. Next, we use sklearn's CountVectorizer function to create a vocabulary from our documents. The resulting vector has a word and its count in the dataset of documents.
    c. After that, we use sklearn's TfidfTransformer to compute IDF for each of the words in our vocabulary. Because the vocabulary is derived from many documents, this calculation is useful to us. We then use the Tdidftransformer to generate tf-idf scores for the words in our current document with the .transform function.
    d. Finally, we sort and gather the top 3 words with the highest tf-idf scores from our word to Tf-idf score mapping. These are the keywords we will use in our query.
3. Once the keywords are extracted, we use the newsapi API (https://newsapi.org/) to search the top three keywords using a bitwise AND. This api searches articles from a wide variety of sources to return the ones most relevant to the keywords entered as parameters in the request string. The resulting array of articles is returned from the API call and gets sent back to the front end in content.js.

Software Usage

Video tutorial: https://mediaspace.illinois.edu/media/t/1_1w68n90n
Python packages required (use pip install...):

- Flask
- Sklearn
- Requests
- Beautiful Soup
- Pandas
- Numpy

Instructions to set up app:

- To run a chrome extension, navigate to chrome://extensions
- Click on "Load Unpacked" and select the project directory which has "manifest.json" in the root
- Activate the extension and you'll see it at the top of your browser
- Meanwhile, you must run the Flask App locally at 127.0.0.1:5000
- Run `python app.py` in a terminal in the root directory to see the app start up and log requests

Basic Use Case:

- Look at feed of Associated Press News articles: https://apnews.com/hub/us-news?utm_source=apnewsnav&utm_medium=navigation
- Click on any article, click on the browser extension ‘Relevant News Finder’, and select ‘Find Related Articles’
- After a few seconds, a window will pop up with links to related articles

Contributions

I worked on this project individually.