CS410 Project Proposal – Browser Extension for Intelligent Query Matching

Marcus Hwai Yik Tan (leader, htan8), Omid Afshar (oafshar2), Xue Ying Lin (xylin2)

When searching through the text contents of a webpage, users are currently limited to the browser-native functionality of using exact keyword match. In order to make the webpage search capabilities more "intelligent", our team is developing a Chrome extension that will allow users to query text in a document and receive ranked results. In this manner, our extension will leverage indexing, retrieval functions (BM25), and relevance judgments to enable a more intelligent search experience.

On the frontend, we will need to set up 3 data structures to maintain state about the positions of text within the webpage as well as the corresponding DOM nodes which contain the text. The data structures will work as follows: an inverted index to keep track of terms, term frequencies, document frequencies, etc.; a mapping to track text, absolute position of the DOM node the text belongs to, and absolute position of the text within the DOM node; a mapping in which the keys are the absolute position within the DOM and the values are the reference to the corresponding DOM nodes.

The logic of the extension consists of the following: When the extension starts up, it will iterate through the DOM nodes of the webpage's body. For each node, if it contains text, we add each piece of text as an entry into the inverted index and the text-to-DOM-node mapping. We also add each piece of text as an entry into the text-to-DOM-node mapping, and finally add the DOM node to the third mapping. When a user inputs query text, the extension will retrieve the relevant text information from the inverted index for each word (or phrase) in the query text. It will send this to the backend server using a POST request and wait for the response. When the backend server responds with the ranked results, the extension will display the results as a popup and highlight the text on the page.

As previously mentioned, the documents are pushed to a local backend server implemented using the Python Flask web framework (https://www.fullstackpython.com/flask.html). The Flask package provides functions that fetch data from the backend server to Python and push data vice versa.

A well-established Python package (Gensim package) is used for text analysis. As per standard practice, a vocabulary is established with the provided documents. Stop words are not included in the vocabulary. A bag of word representation is then created for the documents and the query. Ranking of the documents is done using the BM25 ranking function. If time permits, other ranking functions will be explored and provided as a choice to the user. The top N documents with the highest scores are then pushed to the backend server.

The performance of the ranking is evaluated with respect to a set of webpages and queries with user-determined relevance judgements. To obtain the relevance judgements, several users excluding team members will be chosen and provided with the extension together with a predetermined set of webpages and queries. The user can mark a highlighted group of words as relevant/irrelevant. Mean average precision is used as the performance metric.

Member	Task	Workload (hours)			
		Research	Implementation	Testing	Total
Marcus	Backend (Python and Python Flask), Integration (Chrome extension)	5	8	7	20
Omid	Frontend (javascript), user relevance judgements	5	8	7	20
Xue Ying	Integration, GUI (Chrome extension), user relevance judgements	5	8	7	20