Indiana Reed (ijreed)

INFO-I427

Final Project – Search Engine

Report

Project URL: http://cgi.soic.indiana.edu/~ijreed/ijreed_final_project/project/index.html

(I also wrote a short html page that redirects from the cgi-pub home directory: http://cgi.soic.indiana.edu/~ijreed/ index.html will redirect to the link above.

I chose to crawl 2000 pages starting with http://www.ign.com/, so my search engine is themed around video games. When crawling larger amounts of web pages, I ran into an error with my crawler. It parsed a link that was just a JPEG image and was unable to find links on it, so it threw an error and quit. I fixed this by using a try except to avoid these links. The only other thing I had to change from my original crawler was adding a webgraph (page:links) dictionary to be used for pagerank. I added it and then saved it to a file 'webgraph' using pickle.dump(). [example run: python crawl.py http://www.ign.com/ 2000 pages/ bfs (same as assignment)]

I also ran into an error with saving the 'invindex.dat' file. I tried fixing it by using pickle, but the size of the file was too large, so I switched back to just writing a file. The problem was I opened all my files with 'with open' statements, meaning large amounts of data got put into RAM. This caused an error. I changed my code to just open the files, then save the data so I can index it, then close the file. This fixed the problem. I also put everything into a function to make my code modular. [ex run: python index.py pages/ index.dat]

Next, I implemented pagerank. The first step was to construct an adjacency matrix from the webgraph (loaded in via pickle). I wrote a function for this. It's pretty simple and uses nested for loops to check for links between pages. If there is a link, the value of the cell is one. If not, the value is zero by default. My page rank function first calls the adjacency matrix function to construct the matrix. Then, it calculates the variable m using the adjacency matrix and the Google matrix (with appropriate weights for the dampening factor). Then, it multiplies the matrix m by the vector v until the values are small enough (this version of pagerank has all ranks less than 1). It does an extra multiplication to handle edge cases, just to be sure. Then, it loops through the ranks, inputting them into the answer dictionary (with the key being the page). This dictionary is then written to a file 'pageranks' using pickle. [python pagerank.py]

For my retrieve2.py file, I only had to make a few changes. I continued fixing my file openings to close them right after instead of using 'with open'. I also put everything into a function retrieve(search_terms) to modularize it (where as before it took command line input, it now just has a function that takes a list of search terms). Lastly, I had to save the TFIDF scores to a dictionary + pickle file 'TFIDF', instead of printing them out like before. [retrieve2.py is run as part of the website query, the code that runs it in command line is commented out]

Finally, to put it all together I wrote a search.py file. The main search(search_terms) function first does a couple input checks (also to make sure pageranks have been calculated). Then, it calls retrieve on the search terms to calculate TFIDF scores. Then it loads in the TFIDF file via pickle. To

calculate final scores I simply add the pagerank and TFIDF score for each page. These final scores are saved in the 'final' dictionary. This dictionary is sorted by score and saved to a list (taking only the top 20 hits). Then, the titles for each of the top 20 hits (urls) are retrieved from docs.dat. The function returns a list of urls and a list of titles for those urls. [search.py also runs via the website, but you can test it by running python search.py (test search: "nintendo")]

My demo.cgi file then displays these titles as hyperlinks (as in click the title and it takes you to the page). I spiced up my html code by using a CSS template found here: http://codepen.io/jackrugile/pen/ABeIi. I edited the template to suit my needs (adding h1, h2, p tags for example). This way everything followed the style including my title 'IGNoogle'. This is what I did for extra credit. [demo.cgi runs from the website, as well]

USER EVALUATIONS:

User 1: The search engine worked fairly well. Most of the top 10 results were usually relevant to what I entered. For example, I searched Nintendo and 7 of the top 10 titles had Nintendo in them. The others will still pretty relevant, but just more general (like video game reviews, not Nintendo specific). If you search more than one word, the results get far more accurate. I searched Nintendo switch and got a bunch of articles about the Nintendo switch. This is pretty much what you would expect from a search engine. Searching specific games like final fantasy or call of duty yields really good results, too. Mainly reviews of games from those series.

User 2: Usually at least half of the results were directly pertaining to the search. The rest were sometimes just a few words or a degree away from what I searched. If you looked up Tomb Raider or Overwatch it would come up with related games you might like (ie Best Playstation games of 2015, etc.). Some came up with gift ideas for 2016, too. Overall, it worked decently for what I search.

NOTE: I collaborated with Joel Park for high level ideas.