EECS 485 Project 4: Information Retrieval

Due 9pm Monday, Mar 21, 2016

In this assignment you will learn how to build and maintain an inverted index, and use it to do simple search queries. You will do this by adding a new search endpoint to your website as well as implementing an inverted index that will enable efficient retrieval of documents relevant to the user's query.

Your search page will allow users to search for photos with a text query. Your indexer will then take this query and find all photos that are similar to this query (based on their captions). A majority of the work for this project will be creating the inverted index and using it to find all of the similar photos. Once you have these photos, you will present them to the user in the correct ranking (based on their similarity scores).

Your tool should run in two phases: "index-time" and "query-time". The index-time phase occurs after your search engine has acquired a text corpus (this will be a file with all of the photo's captions), prior to any query processing. Your code should read the collection of text, compute the index, and write it to disk. Then later, immediately prior to query-time query processing, your search application should load the index from disk into memory. Note that unlike many search engines, the corpus will be small enough that you can build the entire index in memory.

Part 1: Updating SQL Tables

For query-time processing, when a user enters a search query, you will use this loaded in-memory inverted index to rank photos for the user. You will create a new table in your database (continue using the db from project 1) to store all of the photo's information with the following schema for the **PhotoSearch** table.

PhotoSearch (sequencenum, url, filename, caption, datetaken).

Follow these guidelines when creating this table:

- sequencenum is an integer
- urls and captions are at most 255 characters
- filenames are at most 40 characters
- date should be of type SQL DATE

You should load the XML file, resources/ search.xml, of 200 photos with captions into your database. Please reference this link for a way to do this that will help you avoid writing a script

or entering the data manually, https://dev.mysql.com/doc/refman/5.5/en/load-xml.html. You should keep search.xml in the directory that the rest of your sql files are in. If you get something like: “The used command is not allowed with this MySQL version”, use the updated vagrant.sh on Google Drive, or add the following line to your vagrant.sh file:

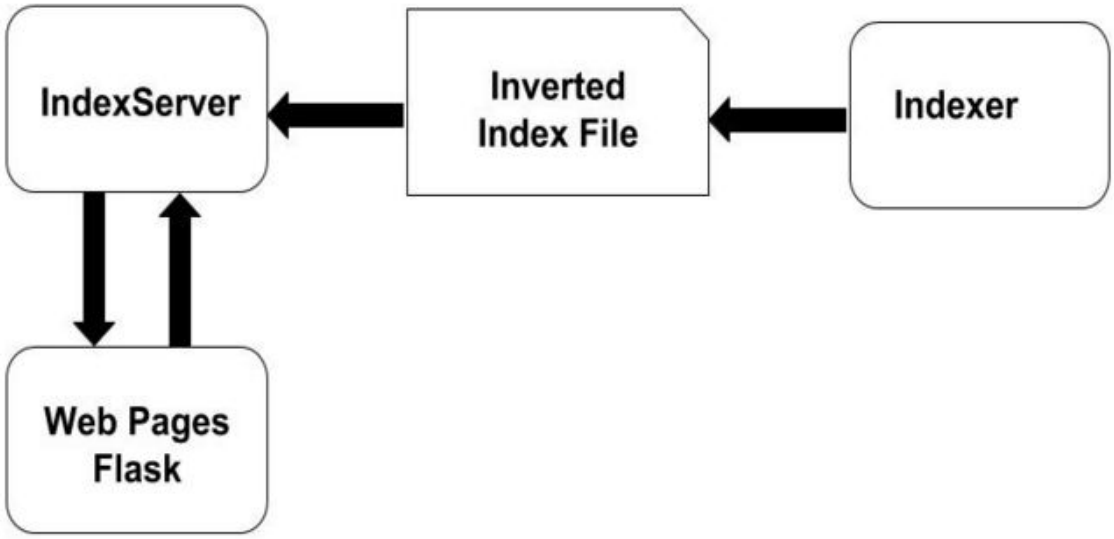
```
sudo printf "\n[mysqld]\nlocal-infile\n\n[mysql]\nlocal-infile\n" >> /etc/mysql/my.cnf
```

Part 2: Indexer & Index Server

System Design

Unlike earlier projects, only a small amount of your code for project 4 will be in Python; the rest will be written in C++.

To get you started, we have provided a skeleton code framework in the pa4_CPP folder. It consists of some source code as well as compiled libraries. The following diagram shows the relevant software components.



Under pa4_CPP, the only files that you should add code to are Indexer.cpp and Index_server.cpp. You only need to add your code in Index_server::init, Index_server::process_query and Indexer::index. You may add other functions and data structures as necessary, however, you cannot delete any function from the original files and may not modify the header files.

Inverted Index

You will index all of the captions for the 200 photos we give you. Treat each caption as a separate (small) document, with its own doc-id.

Indexer.cpp is where you will be building your inverted index. The input of Indexer is the captions file, which can be found in resources/captions.txt, and the output is the inverted index file. Your inverted index should be written to index/inverted_index.txt. The indexer should only be run once to generate the inverted index.

A sample of your inverted index can be found in resources/sample.txt. In here we give you the format of the inverted index, which you should follow. If a word appears in more than one doc it does not matter which doc_id comes first in the inverted index. This sample is a subset of what you will be generating, and is a good reference to verify that your inverted index is correct. Note, that the log for idf is computed with base 10 and that some of your decimals may be slightly off due to rounding errors. In general you should be within .4 of these sample output decimals.

When building the inverted index file, you should use a list of predefined stopwords to remove words that are so common that they do not add anything to search quality ("and", "the", "etc", etc). We have given you a list of stopwords to use in ~~resources~~ stopwords.txt.

NOTE: When creating your index you should treat capital and lowercase letters as the same (case-insensitive). You should also only include alphanumeric words in your index and ignore any other symbols. If a word contains a symbol, simply remove it from the string.

Index Server

The index server loads an inverted index from disk and uses it to process search queries. You will create a vector of “hits” for each search query. A hit will be a photo with a caption that is similar to the user’s query.

IndexServer.cpp is where you will be writing the code for the index server. When the Index server is run, it will load the inverted index file into memory and wait for queries. Every time you make an appropriate call using the Python library, the Index server will invoke a virtual processQuery() method that you will implement. This processQuery() method is provided with the user's search string, and must return a list of all the "hit" docids, along with relevance scores (we have given you the skeleton code for this function). The results of the search() method are then returned to your web app via the network and the Python API.

When you get a user's query make sure you remove all stopwords. Since we removed them from our inverted index, we need to remove them from the query. Also rid the user's query of any non-alphanumeric characters as you did with the inverted index.

Note: make sure that stopwords.txt is in the same directory as your index server executable.

Relevance Scores

Your hits vector should be in order of the most relevant documents. A document is relevant if it is similar to the user's search query. We will use the normalized similarity formula from lecture to calculate the sim(Q, D) score. When finding similar documents, only include documents that have every word from the query in the document (boolean AND).

Running Your Code

- To build your code go to pa4_CPP folder and run: `make`
- To run the indexer: `./indexer input.txt output.txt`
- To run the index server: `./indexServer <portnum> <indexfile>`

The port number will be the port number that your index server listens on, so it must be different from the HTTP port numbers that your Flask web server opens. You currently have 2 HTTP ports (xxx0 and xxx1) so use xxx2 for your index server (xxx being the first 3 digits of your port number).

Part 3: Search web page

You will create a search front-end in Python, with the same port number as project 3. It contacts a back-end server that processes search queries and returns hits to your web app. Your code will then render the results on screen to the user. The only style that we are enforcing for the search page is that the picture results show up in the correct order with their corresponding captions. We have provided a Python library to you that hides the details of communicating with the back-end server (http://www.python-requests.org/en/latest/); you should simply pass in a target port number and a search string.

New Endpoint: /search

After finishing part 2, you will have a small search engine. Now you will create a new web page `/search` to be your search homepage. You will add this search homepage to the rest of your controllers and template code from the past projects. Although you won't be using the other endpoints, please do not delete anything.

In this project, you can assume anyone who is in `/search` can use your search engine (you do not need to worry about access rights or sessions like past projects). Your /search endpoint should accept GET requests with a single argument, named 'query'

Your engine should receive a simple AND, non-phrase query (assuming the words in a query are independent), and return the ranked list of images. This should be a nice-looking list of hits, with photo thumbnails next to each with their captions. You can find the actual photos in resources/flickr-images. In addition to the photos, please show the number of results you get, as Google does when processing a search query. No in-doc position information is necessary for the inverted index. However, frequency information IS necessary inside the inverted index, so that tf-idf can be computed.

It is critical that your search feature use the inverted index for answering all queries. Because the set of photos is relatively small, you could probably iterate directly through all of the
caption strings without a huge computational burden. However, the purpose of this assignment is to learn about the inverted index. With an inverted index, we could scale your system up to very large document collections; that scaling would be impossible if you simply examine the caption strings directly. If your search feature simply examines the caption strings directly, you will not receive any credit.

Connecting Python Code to Index Server

Here are the steps you should follow:

- `import requests  // enable you to use the library`
- `result = requests.get('http://SERVERHOSTNAME:PORTNUMBER/search?query=QUERY')`
  `// send a query to the server`

`result` is a Response object. You can use `print result.text` to see the details

Finding Similar Photos

Your search page should also implement a feature for retrieving photos that are similar to a given "query photo". That is, after making a query and clicking a picture the user should be able to see a list of photos that are the most relevant to the clicked picture. Recall that to the search engine, a "query" and a "document" are basically the same thing; to implement this feature, simply take the clicked picture's caption string as a query, and return the results.

Ids

Make sure that all of these element IDs are present in your search HTML template:

- /search
    - the input for the search query should have an id searchBox
    - the submit button for the search query should have an id searchSubmit
    - you should have a <p> tag with id numResults where you display the number of hits returned
    - each <img> tag should be wrapped in an <a> tag
        - each <a> tag should have an id="pic_<pic_name>_id"
        - each <a> tag should have an href to the similar photos page
        - each <a> tag should have a class queryHit
    - each photo's caption should have a class caption

Part 4: Submitting

Submit the following files to the autograder:

- `Index_server.cpp`
- `Indexer.cpp`
- `tbl_create.sql`
- `load_data.sql`

We will post a link to the autograder when it is ready for this project.

In the `README.md` at the root of your repository please provide the following details:

- Group Name (if you have one)
- List the contribution for each team member:
  `User Name (uniqname): "agreed upon" contributions`
- Any need-to-know comments about your site design or implementation.

You can find useful information about this assignment from this book:
* [Information Retrieval](http://nlp.stanford.edu/IR-book/information-retrieval-book.html)

Spec updates

March 8th

- You should continue using your project 1 database for this project
- Each <img> tag should be wrapped in an <a> tag, see ids section
- Your /search endpoint should accept GET requests with a single argument, named 'query'

- When loading search.xml into your database, you should have it be in the directory with the rest of your sql files (this is where it will be on the autograder)
- stopwords.txt should be in the same directory as your index_server executable (this is where it will be on the autograder)
- You should keep search.xml in the directory that the rest of your sql files are in.
- If you get something like: “The used command is not allowed with this MySQL version”, use the updated vagrant.sh on Google Drive, or add the following line to your vagrant.sh file:

```
sudo printf "\n[mysqld]\nlocal-infile\n\n[mysql]\nlocal-infile\n" >> /etc/mysql/my.cnf
```