COMS E6111-Advanced Database Systems (index.html)
Spring 2022

Project 1

Summary of Deadlines

- By **Tuesday, February 1, at 5 p.m. ET:** Find a teammate or email Akhil (mailto:ak4590@columbia.edu) if you need help finding one.
- By **Thursday, February 24, at 5 p.m. ET:** Submit your project electronically on Gradescope. You can use project grace days (proj-policies.html) normally for the submission.

Teams

You will carry out this project in **teams of two**. If you can't find a teammate, please follow these steps:

- Post a message on the Ed Discussion board asking for a teammate—**the best way**.
- Send email to Akhil (mailto:ak4590@columbia.edu) right away (and definitely **before Tuesday, February 1, at 5 p.m. ET**) asking Akhil to pair you up with another student without a teammate. Akhil will make a best effort to find you a teammate.

You do not need to notify us of your team composition. Instead, you and your teammate will indicate your team composition when you submit your project on Gradescope (click on "Add Group Member" after one of you has submitted your project). You will upload your final electronic submission on Gradescope exactly once per team, rather than once per student.

Important notes:

- If you decide to drop the class, or are even remotely considering doing so, please be considerate and **notify your teammate immediately**.
- On a related note, do not wait until the day before the deadline to start working on the project, just to realize then that your teammate has dropped the class. It is your responsibility to start working on the project and spot any problems with your teammate early on.
- You can do this project by yourself if you so wish. Be aware, however, that you will have to do exactly the same project as two-student teams will.
- Please check the Class Policies webpage (proj-policies.html) for important information about what kinds of collaboration are allowed for projects.

Project Description

In this project, you will implement an information retrieval system that exploits user-provided relevance feedback to improve the search results returned by Google. The relevance feedback mechanism is described in Singhal: Modern Information Retrieval: A Brief Overview

(Readings/singhal.pdf), IEEE Data Engineering Bulletin, 2001, as well as in Chapter 9, "Relevance Feedback & Query Expansion," of the Manning, Raghavan, and Schütze Introduction to Information Retrieval (http://nlp.stanford.edu/IR-book/) textbook, available online.

User queries are often ambiguous. For example, a user who issues a query [jaguar] might be after documents about the car or the animal, and in fact search engines like Bing and Google return pages on both topics among their top 10 results for the query. In this project, you will design and implement a query-reformulation system to disambiguate queries and improve the relevance of the query results that are produced. Here's how your system, **which should be written in Python**, should work:

1. Receive as input a user query, which is simply a list of words, and a value—between 0 and 1—for the target "precision@10" (i.e., for the precision that is desired for the top-10 results for the query, which is the fraction of pages that are relevant out of the top-10 results).
2. Retrieve the top-10 results for the query from Google, using the Google Custom Search API (see below), using the default value for the various API parameters, without modifying these default values.
3. Present these results to the user, so that the user can mark all the webpages that are relevant to the intended meaning of the query among the top-10 results. For each page in the query result, you should display its title, URL, and description returned by Google.
   IMPORTANT NOTE: You should display the **exact top-10 results returned by Google for the query** (i.e., you cannot add or delete pages in the results that Google returns). Also, the Google Custom Search API has a number of search parameters. Please do not modify the default values for these search parameters.
4. If the precision@10 of the results from Step 2 for the relevance judgments of Step 3 is greater than or equal to the target value, then stop. If the precision@10 of the results is zero, then you should also stop. Otherwise, use the pages marked as relevant to **automatically** (i.e., with no further human input at this point) derive **new words** that are likely to identify more relevant pages. You may introduce at most 2 new words during each round.
   IMPORTANT NOTE 1: You **cannot delete any words** from the original query or from the query from the previous iteration; you can just add words, up to 2 new words in each round. Also, your queries must consist of just keywords, without any additional operators (e.g., you cannot use negation, quotes, or any other operator in your queries).
   IMPORTANT NOTE 2: The **order** of the words in the expanded query is important. Your program should automatically consider the alternate ways of ordering the words in a modified query, and pick the order that is estimated to be best. In each iteration, you can reorder all words—new and old—in the query, but you cannot delete any words, as explained in the note above.
5. Modify the current user query by adding to it the newly derived words and ordering all words **in the best possible order**, as determined in Step 4, and go to Step 2.

The key challenge in the project is in designing Step 4, for which you should be creative and use the ideas that we discussed in class—as well as the above bibliography and the course reading materials—as inspiration. You are welcome to borrow techniques from the research literature at large (either exactly as published or modified as much as you feel necessary to get good performance in our particular query setting), but **make sure that you cite the specific**

publications on which you based your solution. As a hint on how to search for relevant publications, you might want to check papers on “query expansion” in the main IR conference, SIGIR, at https://dblp.uni-trier.de/db/conf/sigir/index.html (https://dblp.uni-trier.de/db/conf/sigir/index.html). If you choose to implement a technique from the literature, you still need to make sure that you adapt the chosen technique as much as necessary so that it works well for our specific query setting and scenario, since you will be graded based on how well your technique works. If you want to do stopword elimination (this is of course optional), you can find a list of stopwords here (proj1-stop.txt).

You will use the Google Custom Search API (https://developers.google.com/custom-search/) (https://developers.google.com/custom-search/) in this project: this is Google’s web service to enable the creation of customized search engines. Furthermore, the code that you submit for **your project must run on the Google Cloud**, so it is a good idea to develop your code on a VM on the Google Cloud from the very beginning (see below), rather than writing it on a different platform and then adapting it to the Google Cloud for submission.

As a first step to develop your project, you should set up your Google Cloud account carefully following our instructions provided here (gc-setup.html). Our instructions also explain how you should set up a VM on the cloud, to develop and run your project. Please make sure that you do all this over **your Lionmail account**, not your personal Gmail account.

As a second step, you will have to sign up for the Programmable Search Engine service (https://programmablesearchengine.google.com/about/) (https://programmablesearchengine.google.com/about/):

1. Log off from all Gmail/Google accounts and then log on only your Lionmail account. (Google doesn't let you switch between accounts when you are setting up a Google Programmable Search Engine service.)
2. Press the "Get Started" button on the top right corner.
3. Create a new search engine by clicking the “New search engine” button on the top left corner.
4. Specify the following field values:
    - “Sites to search” should be "www.wikipedia.com" for now
    - “Language” should be "English"
    - “Name of search engine” should be "cs6111"
5. Press the “CREATE” button.
6. Select “Edit search engine” on the left, choose search engine “cs6111,” and click "Setup."
7. Under the top “Basics" button, turn on the "Search the entire web" button.
8. Under the top "Basics" button, under the "Sites to search" heading, select the "www.wikipedia.com" site and press the “Delete” button. This will enable the creation of a search engine to search the entire web but without an emphasis on any particular website (i.e., you will be using the general Google search engine).
9. Copy your “Search engine ID,” which you will need for querying.
10. Do not modify or change other settings.
11. Check the Google Custom Search JSON API documentation (https://developers.google.com/custom-search/v1/overview) and obtain a JSON API key by clicking on "Get a Key"; you will have to select the Google Cloud project that you have already created using our instructions (gc-setup.html) above.

You will use your search engine ID, your JSON API key, and a query as parameters to encode a search request URL. When requested from a web browser, or from inside a program, this URL will return a document with the query results. Please refer to the Google Custom Search JSON API documentation for details on the URL syntax and document schema. You should parse the response document in your program to extract the title, link, and description of each query result, so you can use this information in your algorithm. Here is a Python example of use of the Google Custom Search API that should be helpful: example (https://github.com/google/google-api-python-client/blob/master/samples/customsearch/main.py) (note that q refers to your query, developerKey refers to your Google Custom Search API key, and cx refers to your search engine ID).

By default, the Google Custom Search JSON API has a quota of 100 queries per day for free. Additional requests cost $5 per 1,000 queries, which will be deducted from the coupon credit that Columbia provided to you (see above). Please refer to the JSON API documentation (https://developers.google.com/custom-search/v1/overview) for additional details, which you should check carefully to avoid billing-related surprises.

Test Cases

Your submission (see below) should include a transcript of the runs of your program on the following queries, with a goal of achieving a value of 0.9 for precision@10:

1. Look for information on the Per Se restaurant in New York City, starting with the query [per se].
2. Look for information on Google cofounder Sergey Brin, starting with the query [brin].
3. Look for information on COVID-19 cases, starting with the query [cases].

We will check the execution of your program on these three cases, as well as on some other queries.

What to Submit and When

Your Project 1 submission will consist of the following three components, which you should submit on Gradescope by **Thursday, February 24, at 5 p.m. ET**:

- Your well-commented **Python code**, which should follow the format of our reference implementation (see below) and run on your Google Cloud VM, set up as detailed here (gc-setup.html)
- A **README** file including the following information:
    a. Your name and Columbia UNI, and your teammate's name and Columbia UNI
    b. A list of all the files that you are submitting
    c. A clear description of how to run your program. Note that your project must compile/run in a Google Cloud VM that you set up exactly following our instructions (gc-setup.html). Provide all commands necessary to install the required software and dependencies for your program.
    d. A clear description of the internal design of your project, explaining the general structure of your code (i.e., what its main high-level components are and what they do), as well as acknowledging and describing all external libraries that you use in your code

   e. A detailed description of your query-modification method (this is the core component of the project); this description should cover all important details of how you select the new keywords to add in each round, as well as of how you determine the query word order in each round
   f. Your **Google Custom Search Engine JSON API Key and Engine ID** (so we can test your project)
   g. Any additional information that you consider significant
- A **transcript** of the runs of your program on the 3 test cases above, with relevant results clearly marked, and with the rephrased query and precision@10 value for each run. The format of your transcript should closely follow the format of the interactive session of our reference implementation (see below).

To submit your project, please follow these steps:

1. Create a directory named proj1.
2. Copy the source code files into the proj1 directory, and include all the other files that are necessary for your program to run.
3. Tar and gzip the proj1 directory, to generate a single file proj1.tar.gz.
4. **Submit on Gradescope exactly three files**:
   - Your proj1.tar.gz file with your code,
   - Your uncompressed README file, and
   - Your uncompressed query transcript file.

Reference Implementation for Project 1

We have created a reference implementation for this project. To run the reference implementation, ssh as "guest-user" to the VM running at 35.243.171.94 (i.e., open a terminal and type "ssh guest-user@35.243.171.94"). Use the password for this VM that Vaibhav included in the email to you with your Google coupon code. After you have logged into the guest-user account, run the following command:

```
/home/gkaraman/run <google api key> <google engine id> <precision> <query>
```

where:

- <google api key> is your Google Custom Search JSON API Key (see above)
- <google engine id> is your Google Custom Search Engine ID (see above)
- <precision> is the target value for precision@10, a real number between 0 and 1
- <query> is your query, a list of words in double quotes (e.g., “Milky Way”)

The reference implementation is interactive. Please adhere to the format of the relevance feedback session for your submission and your transcript file.

Also, you can use this reference implementation **to give you an idea of how good your algorithm should be**. Ideally, the performance of your own algorithm, in terms of the number of iterations that the algorithm takes to achieve a given precision value for a query, should be at least as good as that of our reference implementation.

Hints and Additional Important Notes

- Your implementation should not have any graphical user interface. Instead, please include a plain-text terminal interface just as that of the reference implementation that we have provided (see above).
- You are welcome to ignore non-html files when you decide on what keywords to add to your query in each iteration. (Most likely there will not be many non-html files among the top-10 results for a query.) In other words, you will get the top-10 results, including perhaps non-html files, and you can optionally just focus your analysis on the html documents. Your calculation of precision for the results may then focus on just the html files (so if, say, the query returns only 8 html files, you may calculate precision over only these 8 files; in this case, if, say, 4 results are relevant out of the 8 html files, then precision@10 will be 0.5). Please **explain in your README file how you are handling non-html files**. However, the queries that you send to Google should **not** limit the document types that you receive (you should just include keywords in the query).
- In each iteration, you can either just use and analyze the short document "snippets" that Google returns in the query results or, as an alternative, you can download and analyze the full pages from the web. This is completely up to you.
- You are welcome to use external resources such as WordNet (see http://wordnet.princeton.edu/ (http://wordnet.princeton.edu/)) or even word embeddings such as word2vec (https://en.wikipedia.org/wiki/Word2vec). However, the use of such resources for this particular project is not encouraged or required, given that we haven't discussed them in class yet.
- You should **not** query Google inside an iteration. In other words, you should decide on the query expansion for the next iteration based on the results from the previous iteration and their relevance judgments, but without querying Google again. So the order of the words should be determined based on the contents of the query results from the previous iteration. (For one thing, issuing extra queries would be unlikely to be helpful without new relevance judgments, since you are likely to get very different query results—for which you would not have judgments—even with small modifications of the queries.)
- If in the first iteration there are no relevant results among the top-10 pages that Google returns (i.e., precision@10 is zero), then your program should simply terminate, just as the reference implementation behaves.
- If in the first iteration there are fewer than 10 results overall, then your program should simply terminate; there is no need for your program to handle this case gracefully. (Keep in mind that this project is about "broad," ambiguous queries, which typically return well over 10 documents.)

Grading for Project 1

Your grade will be based on the effectiveness of your query modification method—which, in turn, will be reflected in the number of iterations that your system takes to achieve the target precision both for the test cases as well as for other unseen queries that we will use for grading —, the quality of your code, and the quality of the README file. We will not grade your project in terms of efficiency.