COMS E6111 Advanced Database Systems
Fall 2012

Project 1

Due Date: Wednesday, October 3, 5 p.m. ET
(same deadline for all students, on-campus and CVN)

Teams

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

- Post a message in the **class discussion board on CourseWorks** asking for a team mate—the best way.
- Send **email to Pranay** right away (and definitely **before Friday, September 14, at 5 p.m.**) asking him to pair you up with another student without a team-mate. Pranay will do his best to find you a team-mate.

You do not need to notify us on your team composition. Instead, when you submit your project you should indicate in your submission your team composition.

Both students in a team will receive the same grade for Project 1. Team partners are expected to fully collaborate with each other on solving the project. However, communication about project details with somebody other than your partner is not permitted, and is considered cheating. If in doubt about what kinds of consultations are allowed, please check with the instructor.

Questions of a general nature that may be of interest to the whole class should be posted to the **class discussion board on CourseWorks**.

Important Notes:

- If you decide to drop the class, or are even remotely considering doing so, please be considerate and notify your team-mate 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 team-mate has dropped the class or moved to another planet. It is your responsibility to start working on the project and spot any problems with your team-mate 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.

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 Bing. The relevance feedback mechanism is described in Singhal: Modern Information Retrieval: A Brief Overview, 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* 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—hopefully—improve the relevance of the query results that are produced. Here's how your system, **which should be written in Java or Python (your choice)**, 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 Bing, using the Bing Search API (see below), using the default value for the various Bing parameters, **without modifying these default values**.
3. Present these results to the user, so that the user can mark all the web pages 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 text summary returned by Bing.

   IMPORTANT NOTE: You should display the **exact top-10 results returned by Bing for the query** (i.e., you cannot add or delete pages in the results that Bing returns). Also, the Bing 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 query; you can just add words, up to two 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.

5. Modify the current user query by adding to it the newly derived 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" from the main IR conference, SIGIR, at http://www.informatik.uni-trier.de/~ley/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.

You will use the Bing Search API in this project: this is Bing's open search web services platform. To use the Bing Search API, you will have to sign up for an account and create a ACCOUNT KEY, by going to https://datamarket.azure.com/dataset/8818F55E-2FE5-4CE3-A617-0B8BA8419F65 and signing up for the free service. You will use your ACCOUNT KEY and QUERY as parameters to encode a request URL. When requested from a web browser, or from inside a program, this URL will return an XML/JSON document with the query results. Please refer to the Bing API documentation at https://datamarket.azure.com/dataset/8818F55E-2FE5-4CE3-A617-0B8BA8419F65 for details on the URL syntax and the XML/JSON schema. You should parse the response XML/JSON document in your program to extract the title, link, and abstract (or summary) of each query result, so you can use this information in your algorithm.

IMPORTANT: Here are examples of use of the Bing Search API that should be helpful: Java version, Python version.

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 Mac OS X Snow Leopard operating system, starting with the query [*snow leopard*].

2. Look for information on Microsoft founder Bill Gates, starting with the query [*gates*].

3. Look for information on Microsoft founder Bill Gates, starting with the query [*bill*].

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

What You Should Submit

1. Your well-commented **Java or Python code**, which should follow the **format of our reference implementation** (see below);
2. A **Makefile** file explaining how we should compile/run your code on a CS machine running Linux (e.g., on the **clic.cs.columbia.edu** machines);
3. A **README** file including the following information:
    a) Your name and your partner'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 under Linux in your CS account);
    d) A clear description of the internal design of your project;
    e) A detailed description of your query-modification method (this is the core component of the project; see below);

f) **Your Bing Search Account Key** (so we can test your project);
g) Any additional information that you consider significant.

4. A **transcript** of the runs of your program on the 3 test cases above, with relevant results clearly marked, and with the re-phrased query and *precision@10* value for each run. The format of your transcript should closely follow the format of the transcript generated by our reference implementation.

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.

How to Submit

1. Create a directory named **<your-UNI>-proj1**, where you should replace **<your-UNI>** with the Columbia UNI of one teammate (for example, if the teammate's UNI is **abc123**, then the directory should be named **abc123-proj1**).
2. Copy the source code files into the **<your-UNI>-proj1** directory, and include all the other files that are necessary for your program to run.
3. Copy your **Makefile** and **README** files, as well as your query **transcript** (see above), into the **<your-UNI>-proj1** directory.
4. **Tar** and **gzip** the **<your-UNI>-proj1** directory, to generate **a single file <your-UNI>-proj1.tar.gz**, which is the file that you will submit.
5. Login to Courseworks at https://courseworks.columbia.edu/ and select the site for our class.
6. Select "Assignments."
7. Upload your **<your-UNI>-proj1.tar.gz** file under **"Project 1."**

Reference Implementation

We created a reference implementation for this project. To run the reference implementation, run the following from your CS account:

/home/gravano/6111/Html/Proj1/run.sh <bing account key> <precision> <query>

where:

- **<bing account key >** is your Bing Search Account Key (see above)
- **<query>** is your query, a list of words in single quotes (e.g., 'Milky Way')
- **<precision>** is the target value for *precision@10*, a real between 0 and 1

Please run the reference implementation from a directory where you have write permission (e.g., from your home directory), not from the **/home/gravano/6111/Html/Proj1** directory: the reference implementation needs to create some files in the directory from which it is run.

The reference implementation is interactive, and will return a transcript of your relevance feedback session. **Please adhere to this format for your submission.**

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 just focus your analysis on the html documents. However, the queries that you send to Bing 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 Bing 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.
- We will **not** grade your project in terms of efficiency.
- You are welcome to use external resources such as WordNet (see http://wordnet.princeton.edu/). However, the use of such resources is **not** encouraged, because they might introduce substantial "noise" into the process.
- You should **not** query Bing **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 Bing 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 Bing 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 will return many more than 10 documents.)

Luis Gravano