COMS E6111 Advanced Database Systems
Fall 2012

Project 2

New Due Date: Friday, November 2, 4 p.m. EST (sharp)
(same deadline for all students, on-campus and CVN)

Teams

You will carry out this project in **teams of two**. You do not need to notify us now of your team composition. Instead, when you submit your project you should indicate in your submission your team composition, which can be different from that of Project 1. Both students in a team will receive the same grade for Project 2. Please check the Collaboration Policy Web page for important information on what kinds of collaboration are allowed for projects, and how to compute the available number of project grace days for a team.

Important Notes:

- Please be considerate and notify your Project 1 team-mate immediately if you decide to change teams for Project 2; similarly, do not wait until the day before the project deadline to contact your former team-mate just to find out that s/he decided unilaterally to form another group.
- Please follow the same guidelines as in Project 1 to find a team-mate if you need to.
- 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.
- For this project, you have a choice: you can write your code in **either Java or Python**.

Description

This project has to do with Web "databases," and consists of two parts. In Part 1, you will implement the Web database classification algorithm that is described in the ACM TOIS '03 "QProber" paper by Gravano, Ipeirotis, and Sahami. In Part 2, you will implement a simplified version of the content-summary extraction algorithm that we discussed in class.

Part 1: Web Database Classification

As Part 1 of this project, you will implement the QProber database classification algorithm that we discussed in class, which is described in the ACM TOIS '03 "QProber: A System for Automatic Classification of Hidden-Web Databases" paper by Gravano, Ipeirotis, and Sahami. (This paper is part of the class reading list.) For your implementation, you should assume that the hierarchical categorization scheme that we will use consists of a root node with three children, "Computers," "Health," and "Sports," and that each of these three categories in turn has two leaf-level children, "Hardware" and "Programming" for "Computers," "Fitness" and "Diseases" for "Health,"

and "Basketball" and "Soccer" for "Sports." This 2-level categorization scheme is shown below:

- Root
  - Computers
    - Hardware
    - Programming
  - Health
    - Fitness
    - Diseases
  - Sports
    - Basketball
    - Soccer

Your program should implement the query-probing classification algorithm described in Figure 4 of the QProber paper. You should ignore (i.e., not use) the "confusion matrix adjustment" step. Also, you don't need to do any "training" of your system. Instead, we are providing you with the queries that you should use at each internal node of the categorization scheme. (Just for your information, these queries were derived automatically using the Support Vector Machines (SVM) version of QProber.) You can get the queries by clicking in each category above. For example, the line "Computers avi file" in the file associated with the "Root" category indicates that documents containing both the words "avi" and "file" should be classified under "Computers." Therefore, the number of documents matching query [avi file] in a database should be used towards the computation of the coverage of the database for the "Computers" category.

To derive the final classification of a database according to the Figure 4 algorithm, your program should receive as input:

1. The URL of the database to be classified (e.g., diabetes.org). You can assume that you will always get "http" URLs (e.g., not "https" or "ftp") and you should omit specifying "http" with the URL (i.e., your input should be diabetes.org and not http://diabetes.org).
2. The specificity and coverage thresholds. The specificity threshold t_{es} is a real number such that $0 <= t_{es} <= 1$, while the coverage threshold t_{ec} is an integer such that $t_{ec} >= 1$.

You will use the Bing Search API, as you did in Project 1. In your project, rather than querying each database directly, you will use the Bing API to avoid writing database-specific "wrappers." Specifically, you will use the "Site Restricted Search" feature of the Bing API, so that each query that you send to Bing matches Web pages only from the desired database (or Web site). For example, if you wanted to issue the QUERY [avi file] to the diabetes.org database, you should encode the request URL using your ACCOUNT_KEY and the QUERY, and add the string 'site:diabetes.org' to the query, where you replace ":" with "%3a" to comply with Bing's format. The number of matches reported by Bing for this query indicates the number of documents in the diabetes.org Web site that match both the words "avi" and "file".

IMPORTANT NOTE: If you query Bing using URL https://api.datamarket.azure.com/Data.ashx/Bing/SearchWeb/v1/Web?Query=%27site%3afifa.com%20premiership%27&$top=10&$format=Atom, Bing returns the results for the query [premiership] from site fifa.com. However, the results will not include the total number of pages matching the site-restricted query. To get this number, consider querying Bing with URL:

https://api.datamarket.azure.com/Data.ashx/Bing/SearchWeb/v1/Composite?Query=%27site%3afifa.com%20premiership%27&$top=10&$format=Atom, which returns metadata about the query results.

For testing, etc. you might want to save/cache locally the results that you get from Bing so that you only send a query to Bing if you haven't issued the exact same query before. This caching is not required, but it would help with the efficiency of your testing process.

Test Cases

So you can test your program, here are some Web sites and their associated classification in the above categories, **as of 10/8/2012** and for $\mathbf{t_{es}=0.6}$ and $\mathbf{t_{ec}=100}$. The first four cases are examples of good classification decisions by our algorithm, while the last one is not so good.

cancer.org: Root/Health/Diseases
yahoo.com: Root
fifa.com: Root/Sports/Soccer
hardwarecentral.com: Root/Computers/Hardware
diabetes.org: Root/Health

Note that the classification of a database will vary according to the threshold values that you choose, and that it is perfectly possible to classify a database under more than one category. Also, since these results were computed against the live version of Bing, which gets updated continuously, these results might of course change over time.

Part 2: Metasearching over Web Databases

As Part 2 of this project, you will write a program to build simple content summaries of Web databases, where a content summary of a database includes a list of words together with their (estimated) document frequency. As we discussed in class, content summaries are used during the database selection step of the metasearching process, to decide what databases to contact to execute a query, and what databases to ignore. You do not have to implement a database selection module, but rather just a program that will produce a content summary for a database, as described below. (If you are interested in reading some more about all this, you can refer to the ACM TOIS '08 "Classification-Aware Hidden-Web Text Database Selection" paper by Ipeirotis and Gravano. Reading this paper is strictly optional, and you should not follow the more sophisticated techniques described there to construct content summaries, but rather just follow the procedure below.)

Part 2a: Document Sampling

You will construct the content summary of a database based on a small "sample" of the pages or documents in the database. You will extract the document sample of a database **D** (e.g., diabetes.org) while you classify it with the procedure of Part 1, as follows:

```
For each category node C (e.g., "Root") that you "visit" while you classify database
D (using the Part 1 procedure):

     For each query q associated with this node (e.g., [avi file
     sites=diabetes.org])  retrieve the top-4 pages returned by Bing for the
     query. (These are all pages in database D.)
```

The document sample associated with a category node **C** and a database **D,** to which we refer as **sample-C-D,** is the set of documents retrieved from **D** as above by queries associated with **C**, plus the set of documents retrieved from **D** by queries associated with **subcategories of C that were visited during the classification of D.** You should *eliminate duplicate documents* from the document samples. For example, suppose that we classify the diabetes.org database under the "Health" category. During classification, we then visited the "Root" category node and the "Health" category node (and no other category). Therefore, the document sample associated with the "Root" node and the diabetes.org database (i.e., **sample-Root-diabetes.org**) consists of the top-4 documents returned by Bing for each of the queries [cpu sites=diabetes.org], [java sites=diabetes.org], [module sites=diabetes.org], ..., [pc windows sites=diabetes.org], [acupuncture sites=diabetes.org], ..., [game league sites=diabetes.org], which are associated with the "Root" category, plus the top-4 documents returned by Bing for each of the queries [aids sites=diabetes.org], ..., [aerobic sites=diabetes.org], ..., [exercise weight sites=diabetes.org], which are associated with the "Health" category.

Note: Please be nice to the Web sites when retrieving pages. For example, you might want to process (see below) each page that you retrieve before fetching the next page, so you space your requests. Alternatively, you can add a line to your program so that it waits for, say, 5 seconds in between page requests.

Part 2b: Content Summary Construction

After you obtained one document sample for each category node under which you classified a given database, you will build a "**topic content summary**" associated with each such sample. The topic content summary will contain a list of all the words that appear in the associated document sample, and their document frequency in the sample (i.e., the number of documents in the sample that contain each word). Your program should output each topic content summary to a text file, containing all the words in the sample -in dictionary order- together with their respective document frequencies. (Note that you do not have to implement the absolute-frequency computation strategy that we briefly discussed in class, but rather just compute the document frequency of a word simply as the number of documents in the sample that contain that word.)

For example, if the site diabetes.org was classified under category "Health" then your program should output two files: **Root-diabetes.org.txt**, with the topic content summary **sample-Root-diabetes.org**, and **Health-diabetes.org.txt**, with the topic content summary **sample-Health-diabetes.org**. The **sample-Root-diabetes.org** content summary is based on all the documents retrieved by "Root"-level queries **plus** all the documents retrieved by "Health"-level queries (see Part 2a), because "Health" is a subcategory of "Root." In contrast, the **sample-Health-diabetes.org** content summary is based only on all the documents retrieved by "Health"-level queries.

To extract the text of a page, you can use the command "`lynx --dump`", available on the CS machines. Any part of the text after the "References" line should be ignored. Also any text within brackets "[....]" should be ignored. Any character not in the English alphabet should be treated as a word separator, and the words are case-insensitive. Alternatively, you can directly use or adapt our Java script, available here. This script converts an HTML document to lowercase, treats any character not in the English alphabet as a word separator, and then returns the set of words that appear in the document.

Note: You do not have to output content summaries for level 2 categories (i.e., for leaf categories), as there are no additional queries performed at this level.

What You Should Submit

1. **Your well commented Java or Python code** that implements Parts 1 and 2;
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 compile/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, for each part of the project;
    e. Your Bing account key (so we can test your project);
    f. Any additional information that you consider significant.

The **README** and **Makefile** files will determine part of your grade for the project. The rest of the grade will be determined by the accuracy of your implementation of both parts of the project. We will test your project for the test databases above as well as for other (undisclosed) databases.

How to Submit

1. Create a directory named **<your-UNI>-proj2**, 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-proj2**).
2. Copy the source code files into the **<your-UNI>-proj2** directory, and include all the other files that are necessary for your program to run.
3. Copy your **Makefile** and **README** files into the **<your-UNI>-proj2** directory.
4. **Tar** and **gzip** the **<your-UNI>-proj2** directory, to generate **a single file <your-UNI>-proj2.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>-proj2.tar.gz** file under **"Project 2."**

Reference Implementation

We have created a reference implementation for this project. To run the reference implementation **from a directory where you have write permission**, do:

$sh /home/gravano/6111/Html/Proj2/bin/run.sh <BING_ACCOUNT_KEY> <t_es> <t_ec> <host>

where:

- **<BING_ACCOUNT_KEY>** is your Bing account key
- **<t_es>** is the specificity threshold (between 0 and 1)
- **<t_ec>** is the coverage threshold
- **<host>** is the URL of the database to be classified

The reference implementation will return the classification of the **<host>** Web site and will create the required content summaries. Each line in the content summary has the following format:

- **<word>#<frequency in the document sample>#<number of matches>**

Your program does not have to output the **<number of matches>** field.

Also, note that our reference implementation includes **multiple-word** entries, corresponding to multiple-word query probes. The line for a multiple-word query in a database content summary has zero as the frequency in the document sample (it is tricky to define "frequency" for multiple words that should not necessarily be considered as a phrase), followed by the number of matches for the query in the database. For example, a content summary line *cables drive floppy#0.0#1.0* corresponds to query probe [cables drive floppy] and indicates that this query had exactly one match in the database in question. For your implementation, you have a choice: (a) you can include such multiple-word entries, just as we do in our reference implementation; alternatively, (b) you can decide **not** to include such multiple-word information in the content summaries. Either choice is fine, but you should specify your choice in your README file.

Luis Gravano