601.466/666 - Final Project

The final project in this class provides an opportunity for you to apply the information retrieval and classification techniques you have learned in this class to an interesting application of your choice.

Students have wide flexibility in selecting a project description, and many of you have already chosen projects in consultation with the instructor.

For those of you still looking for project options, below are several possible project descriptions that will give you an idea of the focus, scope and complexity expected of final projects. However, because of their primarily sample nature, you are very strongly encouraged to do a project on a topic *other than* Option 1-3, and you must obtain specific approval *in advance* to do options 1-3.

Option 1: FriendFinder - a web Robot/Agent (Example only)

Johns Hopkins can be a lonely place. But there are potentially hundreds of students here and at other area universities that share many of your same interests. Build a web robot to find them by locating and analyzing all personal home pages at JHU.

The intuition behind this assignment is that if you take a collection of personal home pages of yourself, your friends and others who share your interests, convert them to a vector representation (as in HW2 and 3), and compute a centroid or profile v_F, and then take a collection of home pages of people who do not seem to share your interests and compute their vector centroid v_{NF}, new home pages can be classified simply by finding the maximum cosine similarity with the two centroids/profiles. You can also use a Bayesian classification model if you prefer.

(a) Begin with an existing web robot of your choice from the *robots* subdirectory of the class directory on hops. The advantages and disadvantages of each will be discussed in class.

(b) Add a function to the robot's code to determine if the WWW page currently being visited by the robot is a personal home page. There is no one definitive way of doing this, but options including searching for simple Perl string patterns in the URL's or titles. For example, *students.html*, *users.html* and *grad_students.html* are titles of the directories under which JHU CS and ECE student home pages are stored. Many people's home pages have *Homepage* or *Home Page* in their <TITLE>, and/or have URL's of the form `http://.../.../~name`. You may also consider building a simple vector model of homepages and non-homepages as in part (e) of this assignment option, but this is not necessary.

(c) Collect a set of homepages of your friends and other people with whom you share interests (HP_F). Do the same for people you dislike or people who seem have different interests from you (HP_{NF}). For the purposes of this assignment, you may wish to consider a *homepage* to also include all of its `.html` subdirectories, where things like interest, hobbies, activities and pet peeves may be listed (just concatenate all of these together in a single long text file).

(d) Create vectors from these collections of home pages (and compute their centroids) as in HW 2 and 3.

(e) To classify new home pages detected in Part (b) as more likely to share your interests or not, create a term vector from the home page, compare with the two centroids created in Part (c) and use that with minimum distance (sorting by a relative distance measure like $cos\_sim(v_i, v_F) - cos\_sim(v_i, v_{NF})$ for a total ordering of all home pages encountered). Ideally, this should be done as a Perl function embedded in the robot, but if you are having trouble doing this you can use the robot to write the detected home pages to disk, and then treat these pages as static documents to be classified just like in HW 2 and 3.

Issues:

One additional issue to consider is the stopword list - it may be larger than that used in ACM abstract retrieval, and should probably include HTML markup. The second is what to use for IDF in term weighting. Unless you pursue the second option in Part (e) and write all the detected home pages to disk before classifying them, you won't have an initial collection on which to compute IDF (to downplay terms that permeate all or most home pages). Thus if you pursue the first option in Part (e) - embedded real time classification of home pages, you should run a pass first where you download a large set of home pages to disk, compute document frequency (DF_t) for all terms in this set using the `make_hist` program from HW2, and then use this histogram in computing IDF for terms in all home pages, even those not used to originally compute the DF_t histogram. Also, if you use the embedded code approach you do not need to do stemming (as this code isn't in Perl and may be hard to call from within the robot).

Finally, you are not restricted to the vector classification model. If you would prefer to use a decision tree or Bayesian approach, please feel free to do so.

Evaluation:

You should provide a simple quantitative evaluation of your system. To do this, collect a set webpages that were *not* used in your training data and label these as *not_homepage*, *homepage_friend* and *homepage_notfriend*. Run your classifier on these web pages and compute the % accuracy of your system in distinguishing between homepages/not_homepages and likely friend/not_friend (based on your own subjective judgements, of course). You should do this tagging *before* you apply your robot to the page; *don't* do a posthoc grading of its performance - this can lead to bias in favor of your system.

Option 2: A mail/news filter/classifier (Example Only)

Overview: The major goal of Option 2 is to write a program that filters and routes/classifies your email into one of several topic areas or "bins" that you have defined in advance. One of these bins should be the class *junk mail* or *discard*. If you do not have a saved archive of

your email (or if you do not have enough incoming mail to save everything you receive in the next few weeks), you could also apply this program to filtering/routing netnews (which is stored on `/var/spool/mail`. If you do this, you may wish to restrict yourself to a subset of the newsgroups. The core approach could proceed in a manner similar to the following:

(a) Select K = 5 to 10 bins or topic areas that you would like your email (or news) sorted into. One should be labelled *junk*. Collect examples of messages that belong in each (as many as you can).

(b) Process the training messages (and those incoming messages in Part (d)) through the segment labeller you wrote in HW 1, discarding some regions (such as graphics) and making note of others (with embedded region tags like in the CACM collection) that you can use for later region weighting. Also run the messages through the tokenizer in HW 1 and create a vector corpus. You should use stopwords and TF-IDF weighting as appropriate, with optional region weighting given to certain fields in the header, section header segments, and other more or less significant regions of the text.

(c) Compute vector centroids of the messages in each of your training classes.

(d) Take new messages (not in your training set) and segment label and tokenize them as in part (b), creating a (weighted) term vector from each.

(e) For each new email message vector, compute cosine similarity with each of the topic centroids and return that classification/bin with the minimum distance. It would useful to print out a list of the top 2-3 bin labels and the cosine similarity with each (along with the sender/subject of the email or news story) so that the user can have more feedback.

(f) It is not adequate to simply have the project be a simple bag-of-words vector comparison with centroids. You should extend this basic model as much as possible. Consider a Bayesian model or decision tree as a second classification strategy. Consider ways to improve on the region weighting and identification of salient terms and classification evidence, such as sender, time-of-day, length, and other such predictive features for certain types of classification (e.g. work project vs. an extra-curricular activity).

Evaluation:

You should provide a quantitative evaluation of your system. To do this, collect a set of mail messages/news postings that were *not* used in your training data and label them with your target classes or bins. Run your classifier/router on these messages and compute the % accuracy of your system. You should do tagging of messages *before* you apply your classifier; *don't* do a posthoc grading of its performance - this can lead to bias in favor of your system.

Option 3: A ResumeFinder Robot (Example Only)

As discussed in class, build a robot that finds resumes on the web (restricted to the JHU domain), and then uses text classification techniques to both segment and classify the resume sections (e.g. education vs. computer languages), and also extract key information from them into a structured database format (e.g. known computer languages, degrees earned, address, etc.). You are encouraged to use the text classification techniques explored in homeworks 1 and 3 to aid in this.

Option 4a: Shopping Agent

Web robots can be used to do "intelligent" things for us both extracting information from the internet and in providing a value added service. One useful thing we could do with a web agent would be to create a Shopping Agent.

Imagine that you want a new book. You could give the shopping agent the name of the book, and it would search through the major online book stores (amazon.com, barnesandnoble.com, etc.), find the best deal, and then order it for you. By understanding the forms interface for each shopping service and sending a POST command with the appropriate fields to each store, we can create a single interface that would work for any of the book stores. This way, you only need to enter your name, phone number, shipping address, etc., one time to the Web Agent, and it takes care of the rest of the work, filling in this information whenever necessary.

Starting with the basic web robot, add in:

(a) the ability to read key fields out of a data file in the format of your choice, including name, phone number, shipping address, etc.

(b) visit the book sellers that you want to support, and download their ordering pages. You can look at the HTML code to find out what the field names you will need to post to are, and what the results will be. The Web Agents book shows you how to make a HTTP::Request that supports sending information as a POST to the server to get back dynamic content.

Evaluation:

Test your robot by having go partway through the process of locating and ordering differnt books. **Under no circumstances** should you give credit card numbers of any type or proceed to the payment phase. Please test your system very sparingly to avoid complaints from vendors. Compare your results to a manual use of the system.

Option 4b: MetaShopper

Another useful thing that Web Agents can do is extract information from multiple sources and condense it into a useful format. Examples of this would be MetaCrawler (http://www.metacrawler.com that searches a number of search engines, re-weights the results and returns you the answer,

BottomDollar (http://www.bottomdollar.com) that searches through many shopping sites and returns you the best prices for the product(s) that you searched for.

Implementing something like BottomDollar isn't very difficult. First visit all of the shopping sites that you wish to support, download their pages, and look at the HTML forms. Then you can find the fields that you will need to add to the HTTP::Request when doing a POST to get the search results back.

Take the basic web robot and add:

(a) A configuration file with the URLs and fields that you need to post to for each store.

(b) Add an interface where the user can type in the queries that they want.

(c) For each query, visit all the services in your configuration file, and get the resulting page.

(d) For each resulting page, use the Text Extraction methods from assignment #1 and assignment #3 to pull out the information that you need to display the user (product name, link to see the product, price) so that you have a common format for all services.

(e) Display the information to the user ranked by price.

Evaluation:

You can test the program by entering queries and then visiting the services that your web robot is visiting and compare the results. Make sure that your Text Extraction methods aren't missing anything, and that your ranking of results is working properly. If you want, you can also add some more advanced ranking by the search term vs. what was returned, instead of just by price.

Option 5: Language-Model based Information Retrieval

Discussed in Class.

Option 6: Cross-Lingual Information Retrieval

Discussed in Class.

Option 7: Topic Detection and Tracking; New Event Detection

Discussed in Class.

Option 8: Music IR

Discussed in Class.

Option 9: IR on Different Source Modalities (e.g. Speech, Image, Video)

Discussed in Class.

Option 10: One of the other potential projects discussed in class

Option 11: Another project of your choice

You are not restricted to the options listed above for your final project. You are free to pursue any application that is both of interest to you and relevant to the material covered in this course. Broad areas of relevance include core information retrieval and search engine implementation, routing and filtering, information extraction and web robot design, or ideally some combination of these areas. You will be graded both on the creativity and the challenge of the ideas you propose, as well as in the quality with which you accomplish these goals.

If you propose to have 3 members working on the project, by April 20 you must receive permission of the instructor **in advance**, justifying why the scope of the project requires this additional person, and what the approximate breakdown of responsibility will be.

Finally, it is *imperative* that no web robot used for this class be unleashed outside the Johns Hopkins domain without stating this intent explicitly and with an assessment of risk and the willingness of the websites you crawl to have robot visitors. You should be very clear on precisely which segments of the web will be visited, and what steps will be taken to ensure good robot "citizenship".

What to Turn in:

The final submission of the final project should be submitted electronically on a timetable discussed in class, using gradescope as in previous assignments.

Your electronic submission should include a PDF project writeup, as detailed below, and submission of code and data that you used/crawled. The PDF writeup should include whatever description you feel is useful or appropriate to communicate the strengths of your accomplishments on the project. In particular, in your writeup:

(1) On your **cover page**, please include (a) the names of your team members, (b) the section each is enrolled in (466/666), (c) your email addresses, (d) a 1-3 line summary of your project focus, (e) a URL where an on-line project interface is located *if* you have one (which is not necessary), and (f) a URL link to any external repositories of project components or data too large to submit via gradescope.

(2) Please provide a brief guide describing in sufficient detail how a user could run your code and/or use your provided web interface, especially if using a non-standard platform or interface.

(3) Give an itemized list of your project's particular achievements, strengths and selling points, especially things that may not be obvious upon inspection of your code or output, as well your personal assessment of their complexity, scope and success.

(4) Give a brief itemization of your project's limitations and list suggested possibilities for improvement or worthwhile extension with additional time.

(5) To both document your projects achievements, as well as the methods of use and expected output(s), please also include a range of samples/screenshots of your project output on a range of project functionalities and input conditions or search queries.

(6) In the spirit of all assignments in the class, empirical evaluation appropriate for the nature of the project should also be included.

(7) If your code submission includes components that were (a) written by people not on your team or (b) written by yourself for another course, for prior research and/or employment, please very clearly explain/document which components were done elsewhere. Such outside borrowing is permitted if clearly documented, and you are permitted to build upon prior accomplishment, just like you may utilize software libraries from elsewhere, but your project will be evaluated on the original work done for this course.

Please submit your writeup as a single *separate* PDF file including all team member names in the filename format specified below (e.g. `john_doe_adam_smith.pdf`).

You are also required to submit a complete archive of all your code developed for the project, and data files generated or used in the project, ideally in a single compressed nested directory (zip or tgz). Your directory should be named as follows to make it easier for me to identify when I uploaded it:
yourgivenname_yoursurname if done individually (e.g. `john_doe.zip`)
or
partner1givenname_partner1surname_partner2givenname_partner2surname
(e.g. `john_doe_adam_smith.zip`)

In most cases data/code submission can be accomplished using the normal homework submission procedures via gradescope, but if the combined project directory size is very large it may be necessary to upload from your jhu public_html directory or other hosting service (please provide the URL on the cover page).

Demos may be scheduled with the instructor if you feel that your project is not self explanatory and could benefit from an interactive presentation.