600.466 - 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, you are very strongly encouraged to do a project on a topic other than Option 1 and you must obtain specific approval in advance to do options 1 or 2.

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 hopo. 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 ($DF_i$) 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 $DF_i$ 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

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 compari-
son 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

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 different 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: Multilingual Web Page Extraction and Language Identification
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: Information Extraction of Rolodex Data from Web Pages
Discussed in Class.

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

What to Turn in:

You are encouraged to submit an optional brief (1-2 page) project overview at some point before the project due date and/or meet with the instructor regarding the project in person to discuss any questions or concerns. Your optional project proposal should include the names of the group members working on the project, and a brief summary of the work to be done. You should address your basic design decisions, what you expect that your system will accomplish, and anything that you have questions about. You should state specifically how your project will be evaluated empirically, (e.g. using what measures – such as precision and recall of retrieved documents, and/or percentage accuracy in classification – and also using what standard of “truth”, such as posthoc correction or comparison with some preexisting data set). In some projects, such as building a document visualization tool, there is no natural quantitative evaluation metric, and you should simply state this and suggest another appropriate standard under which the functional success of your project can be evaluated.

If you propose to have 3 members working on the project, you should justify 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 receiving feedback first. 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”.

The final project should be submitted both electronically and in hard copy, as in previous assignments. The writeup should include whatever description you feel appropriate to communicate the strengths of your contribution. At a minimum it should include sufficient documentation to explain how to execute your code. In the spirit of all assignments in the class, some appropriate form of empirical evaluation should also be included. Demos may be scheduled with the instructor if you feel that your project is not self explanatory and could benefit from an interactive presentation.