Building a Search Engine: Final Assignment

Overview

This is the Final Assignment of the Course, and you will work on it in two phases.

In the first phase you will compute the PageRank of our collection of documents and you will use Support Vector Machines to classify them. You also will implement $k$-Means algorithm to cluster the documents. During the first phase you could work in teams, and the due date is Thursday May 2$^{nd}$ 2013 at 11:59am.

In the second phase, you will enhance the simplified search engine that you built in Part 1 and Part 2, improving the ranking of retrieved documents in response to a query. You will work on this second phase individually, and the due date is Thursday May 9$^{th}$ 2013 at 11:59am.

1 First Phase (Due 5/2 11:59am)

1.1 PageRank

You are required to calculate an approximation of the PageRank $\pi$ for the collection $\mathbb{D}$, that is, the eigenvector corresponding to the principal left eigenvalue 1 of the stochastic matrix $P$ representing the transition probabilities between pages in the collection.

In order to estimate $\pi$, you have to perform the following steps:

1. Extract the adjacency matrix $A$ of the collection $\mathbb{D}$. Use wiki links of the form:

   \[
   [\text{Target_Link}]
   \]

   The page title that the link is pointing to is the sequence of characters before the first occurrence of either # or | in $[\text{Target_Link}]$. (For example, $[[\text{Assembly language#Assembler|assembler}}]$,$[[\text{Assembly language#Assembler}}]$, $[[\text{Assembly language|assembler}}]$, and $[[\text{Assembly language}}]$ point to the same page with title Assembly language.) Among all the extracted page links consider only the links pointing to a page in $\mathbb{D}$ (i.e. links internal to $\mathbb{D}$ only).

2. Calculate the stochastic matrix $P$, as defined in Section 21.2.1 of the textbook$^1$. Use the following damping factor (probability of jumping to a random page): $\alpha = 0.1$.

3. Estimate $\pi$ by performing $t = 128$ iterations, each being a matrix-vector multiplication. (See Section 21.2.2 of the textbook.)

---

$^1$Note that the phrase "If a row of $A$ has no $1$s, then divide each element by $1/N$" at page 426 should be "If a row of $A$ has no $1$s, then each element of the row is $1/N$."
You should produce a text file, named `pagerank.dat`, containing \(|D|\) lines, where the \(i\)th line reports the \(i\)th component of \(\pi\).

For Step 1, we suggest you to code a Python program that will read the full collection file and will output the adjacency matrix \(A\) into a text file. Since this matrix is very sparse, you might want to generate a list representation of its entries: for example, the output file might have a line for each document, where line \(i\) contains the docIDs of the pages linked from page \(i\). Then, you can perform Step 2 and Step 3 through a MATLAB/Octave script, reading the list representation \(A\) and outputting `pagerank.dat`.

When constructing the matrices \(A, P\) and the vector \(\pi\), please be consistent with the order of the documents: the first row/column of the matrices \(A, P\) and the first element of \(\pi\) must correspond to the first document of the collection.

We will check your `pagerank.dat` file only. However, you are required to include in your submission all the code you developed to produce it. (If you used MATLAB or Octave, do not include a dump of your workspace. Please write a `.m` script instead.)

### 1.2 Support Vector Machines (SVM)

You will also use SVMs to classify the documents of the collection, similarly to what you did in the Classification Assignment with MNB/Rocchio classifiers. See Chapter 15 of the textbook for a discussion on SVMs.

We created the collection \(D\) as the union of subcollections of documents from \(k = 11\) different categories (see file `categories.dat`), and some other documents not belonging to any of these categories.

You might want to use and/or adapt your previous `vecrep` program to obtain the vector representation of the collection, say `vecrep.dat`. You can use the provided features list and stop words list or you can decide to build/use your own (for example, the advanced features list you built for the Classification Assignment).

You will train \(k\) 1-vs-all SVM classifiers, one for each category. To do so, first you will create \(k\) training sets (named `svmtraining0.dat`, `svmtraining1.dat`, etc.), using the provided `training.dat`, which contains 10,000 pairs of labeled documents, with training samples for each category.

You should use the package `SVM^light` to train/apply the SVM classifiers\(^2\). Please note that this package requires the feature IDs to start from 1 and be in increasing order, so you might need to adapt your previous `vecrep` program to comply with the `SVM^light` input format. In your `trainingX.dat` training set, you must assign the label +1 to the documents labeled as belonging to category \(X\), and label -1 to the other documents you chose to include in the training set for category \(X\) as negative examples. Run the `svm_learn` with a command similar to:

```
$ ./svm_learn [OPTIONS] svmtrainingX.dat modelX.dat
```

This command will produce the output model `modelX.dat`.

Once you have built the \(k\) 1-vs-all classifiers, create the file `test.dat`, by reading the vector representation of the collection (`vecrep.dat`) and assigning the label 0 ("no category") to each document. Then, classify the documents, calling

\(^2\) `SVM^light` was written by Thorsten Joachims, Department of Computer Science, Cornell University. You can download it from the following URL: [http://svmlight.joachims.org/](http://svmlight.joachims.org/). Note that there is also an interface to Python, in case you want to use the trained classifiers in your search engine.
$ ./svm_classify [OPTIONS] test.dat modelX.dat predictionX.dat

In the produced output predictionX.dat, you will see both positive and negative scores for each document. A document whose score is $\geq \epsilon$ should be considered as belonging to category $X$, while all the other documents should be considered as not belonging to category $X$. You can experiment with the different parameters of SVMlight and with $\epsilon \geq 0$ in order to obtain the best performance in terms of precision/recall.

Then, you will combine the labels produced by processing the prediction files predictionX.dat, to produce the file labelSVM.dat, consisting of $|D|$ lines, where each line contains none, one or more than one category labels, separated by one space, as produced by the SVM classifiers. For example, if labelSVM.dat contains:

```
2
0 1
2
```

it means that the first and the fourth documents were judged belonging to category 2, the second document was assigned to categories 0 and 1, and the third document was not assigned to any category.

Finally, please complete file resultSVM.txt with the required information, as in the provided sample file.

1.3 $k$-Means

In many practical applications, the training data is not classified, and by just looking\textsuperscript{3} at the data we have to group the data points into different groups/clusters/etc. In this case we say our training data is unsupervised.

$k$-means is a clustering algorithm that you will implement for this part. In $k$-means algorithm, you are given a set of points and a number of clusters, $k$. The goal is to partition these points into $k$ sets $M_1, \ldots, M_k$ with mean points $\mu_1, \ldots, \mu_k$ (each $\mu_i$ is the mean of the points in $M_i$) such that the following cost function is minimized:

$$\text{cost function} = \sum_{i=1}^{k} \sum_{x \in M_i} \|x - \mu_i\|^2. \quad (1)$$

The $k$-means problem is hard, and you will implement an approximate method for solving $k$-means problem, called Expectation-Maximization (EM). The algorithm you need to implement is provided in Algorithm 1.

Note that EM solutions converge to local minimums by as $\epsilon \to 0$. So you should run your algorithm with many different initial values and pick the one which minimizes the cost function in (1). Also note that choosing proper initial values for $\mu_i$'s can make your algorithm converge faster. (HINT: it might be better to use $k$ data points for initial values of $\mu_i$'s). You can experiment with different initial values of $\mu_i$'s and $\epsilon$ in order to obtain better performance.

You should run your $k$-means algorithm on training.dat file with $k = 11$ (after removing the category labels, since this is an unsupervised algorithm). Your output file, clusterKM.dat, is going to be a file with $|D|$ lines (as in the previous section) with the same format of training.dat: the $i^{th}$ line has two numbers;

\textsuperscript{3}Well, algorithm is supposed to do that!
Algorithm 1: EM algorithm for $k$-means problem.

Input: Number of clusters $k$, a set of data points $X$, and $\epsilon \geq 0$.

Output: $M_i$ clusters and their means.

begin
  1. Initialize $\mu_1, \ldots, \mu_k$;
  2. $\delta_i \leftarrow \|\mu_i\|$, for $1 \leq i \leq k$;
  3. $M_i \leftarrow \emptyset$, for $1 \leq i \leq k$;
  4. while $\exists i$ such that $\delta_i \geq \epsilon$ do
    5. For each point $x \in X$, assign $x$ to cluster $M_j$ if $\mu_i$ is the closest point in $\mu_1, \ldots, \mu_k$ to $x$;
    6. $\mu'_i \leftarrow$ the mean of the points in $M_i$ for $1 \leq i \leq k$;
    7. $\delta_i \leftarrow \|\mu'_i - \mu_i\|$, for $1 \leq i \leq k$;
    8. $\mu_i \leftarrow \mu'_i$, for $1 \leq i \leq k$;
  9. OUTPUT: $M_i$ and $\mu_i$, for $1 \leq i \leq k$;
end

the first number is the document id in the $i$th line of training.dat file and the second number is the ID of the cluster you assign the document, e.g., the second number is $j$ if the $i$th document belongs to cluster $M_j$. Note that the only important point of these numbers is that two documents must have the same number if and only if they belong to the same cluster.

Again, you might use or adapt your previous vecrep program to obtain the vector representation of the collection, say vecrep.dat.

We only check your clusterKM.dat file, but you should submit your source code as well.

1.4 Evaluation

Your grade for this part will depend upon the correctness of your pageRank.dat output, the way you chose to build the training sets for SVM and the produced labels. This part will count for 25% of the grade for the entire Final Assignment.

1.5 Report (for the first phase)

Explain how one can compare the result of the SVM algorithm with the $k$-means algorithm. Briefly explain the pros and cons of using SVM and $k$-means algorithms.

1.6 First Phase: Submission

Please copy all the files mentioned below into a separate directory, cd there, and run the handin script from that directory. Be aware that since the handin script copies recursively all the files from the directory it is executed from, if you run the script from, say, your home directory, it will handin all your files and directories! The directory from where you run the handin script should contain only:

- Any source code you wrote for the PageRank part.
• The plain text file pageRank.dat.
• Any source code you wrote for the SVM part.
• The features list features.dat you used for the SVM part.
• The training sets svmtrainingX.dat (for $0 \leq X \leq 10$).
• The plain text files labelSVM.dat and resultSVM.txt.
• The plain text file readme.txt.

Please submit *only* the above files (do *not* submit the collection, your MATLAB/Octave dumps, etc.) using the following command:

$ /course/cs158/bin/cs158-handin part3

The due date for this part of the Final is Thursday May 2nd 2013 at 11:59am. Only one submission per team is required.

2 Second Phase (Due 5/9 11:59am)

2.1 General Requirements

You will each work individually on this part of the assignment. Please see Section 4 for more information.

You will work with a collection $D$ of roughly 40,000 documents, with the same format as in the previous parts. You will code two programs: createIndex to create the inverted index (and other support files, if needed) and queryIndex, using the inverted index to answer queries from the user.

2.1.1 Processing the Collection

When processing the document collection, you can adopt some of the processing rules of Part 1 and Part 2, but you might want to use different approaches, including:

• a different tokenization/stemming process;
• using a different stop words list;
• parsing Wikipedia tags;
• indexing bi-grams;
• ...

You are required to try at least one processing approach different from the one adopted in the previous assignments. Remember that you have to justify your choices in the report, so you might want to try several different variations, evaluating their impact on the quality of the results produced by your search engine.
2.1.2 Ranking Retrieved Documents

You are required to combine the scoring scheme from Part 2 (or a variation on it) with the measures of relevance that you previously developed, namely PageRank and MNB/Rocchio/SVM/k-means classifiers. Your ultimate task is figuring out an effective way of combining these different information sources in order to produce a good ranking of documents for a given query.

Some (non-exhaustive) examples include:

- obtaining the final ranking of documents as a weighted combination of the rankings obtained by term weighting and by sorting the pages using their PageRank value;
- applying a majority vote procedure instead of a weighted combination;
- trying to classify the query with the MNB/Rocchio/SVM/k-means classifier and then using the inferred information to pre-filter the documents to be returned;
- inferring whether the given query asks for a generic page or a specific one to underweight/overweight the contribution of term weights or PageRank;
- …

2.1.3 CreateIndex I/O Format

You will submit a bash script createIndex.sh, suitably invoking your createIndex program. Your bash script should accept two parameters, representing the file names of the collection of pages and of the inverted index to be created. We will invoke something like this:

```
$ ./createIndex.sh myCollection.dat myIndex.dat
```

Your createIndex program might generate other files as well, as you need them. In case, they must be created in the current directory (or in a subdirectory) and have extension .out. You might also want to include other immutable files (e.g., your stop word list) in your submission.

2.1.4 QueryIndex I/O Format

You will submit a bash script queryIndex.sh, suitably invoking your queryIndex program. Your bash script should accept one parameter, representing the file name of the inverted index. Your queryIndex program might need to read other files: please pass them as command line arguments of your queryIndex program, specifying them inside your bash script. Please do not hardcode file names in your program!

Your program will read queries from the standard input, one at a time, and output to the standard output the list of the docIDs of the 10 most relevant documents. After retrieving and outputting the relevant documents for a query, your program will wait for another query, until the user closes the standard input with a CTRL+D. Please note that your program might be run in an interactive way. Reading all the queries first and then producing all the results is not acceptable.

Your queryIndex should answer the following types of queries (see the definitions given in Part 1 and Part 2): one word (OWQ), boolean (BQ), phrase (PQ), free text (FTQ) and wild-card queries (WQ). You are not required to deal with wild-card phrase queries.
More precisely, you should output the docIDs of the 10 most relevant documents in decreasing order of relevance (highest first), separated by a blank space. If there are less than 10 relevant documents, output them all. If there is no relevant document, output a blank line. For example, the following might be a typical execution:

```plaintext
$ ./queryIndex.sh myIndex.dat
  Space
  10 12 14 0 1
  Space Odyssey
  10 42 12 34 56 14 89 0 13 1
  Space AND Odyssey
  10 12 14 0 1
  "2001: A Space Odyssey"
  10 12 14
  Titan*
  Full Metal Jacket
  42 45 54 12 88 89 67 81 9 6
```

Note that the I/O interface specified above allows us to use redirection, as in the following example:

```
$ ./queryIndex.sh myIndex.dat < myQueries.dat > myDocs.dat
```

The number of lines of `myQueries.dat` and `myDocs.dat` should be the same.

Your `queryIndex` program is *not* allowed to read all the inverted index in memory. Instead, you can only load the dictionary before starting answering the queries, and then you must fetch the relevant postings lists directly from disk. Loading the dictionary should take no more than 5 minutes (for the provided collection). Your program should answer each “reasonable” query in less than 10 seconds, including fetching the relevant postings lists from disk.

### 2.2 Report

Please include a report in PDF format answering the questions below. When analyzing your data structures and algorithms, please adopt the big-O notation instead of generic “very long” or “very big”, parameterizing your analysis according to the relevant quantities (number of documents/terms/etc.).

*Please make sure that your answer is within the sentence limit, we do not expect more than that, less is fine as long as it is complete. Be concrete, precise and concise. Please do not answer questions that we do not ask! If you are not sure, ask the TAs.*

1. Describe how you modified processing of the collection in Section 2.1.1 compared to the processing from Parts 1 and 2. Justify each modification and explain how you implemented it. Does it add space or time overhead? Why? If it incurs the overhead, quantify it. (10 sentences)

2. Describe your ranking algorithm for `queryIndex` program. In particular, explain how you combine term weighting and PageRank/MNB/Rocchio/SVM/k-means, justifying your choices. (15 sentences)
3. To obtain the final ranking you should have experimented with several ranking algorithms. Please describe your experimental set up to determine whether one ranking scheme is better than the others. (5 sentences)

4. Please analyze your final ranking by explaining what are its strengths and weaknesses, that is, what type of queries it performs well/bad on, if it has a bias towards short/long documents, frequent terms, etc. Give a couple of negative and positive examples for each query type, arguing why the example is negative or positive. (10 sentences + examples with explanations)

5. Compare the ranking from Part 2 with your advanced ranking, by providing the same queries to both search engines, and comparing the obtained results. On what type of queries/terms/documents do you improve over ranking in Part 2? Provide 3 examples for each query type which clearly show the advantages of your ranking over the basic tf-idf ranking. Explain each of your examples.

2.3 Evaluation

We will establish a set of test queries in order to evaluate your search engine, in addition to those published before the due date. Your grade for this part (75% of the grade for the Final Assignment), will depend on:

- The number of queries your program answers in a satisfactory way.
- The algorithms/heuristics you developed for Section 2.1.2.
- The quality of your report.

2.4 Second Phase: Submission

Please copy all the files mentioned below into a separate directory, cd there, and run the handin script from that directory. Be aware that since the handin script copies recursively all the files from the directory it is executed from, if you run the script from, say, your home directory, it will handin all your files and directories! The directory from where you run the handin script should contain only:

- Any source code you wrote.
- Any external module (e.g., .py/.pyc) or file (e.g., stop word list) you need to run your code.
- The two Bash scripts `createIndex.sh` and `queryIndex.sh`, suitably modified to call your programs.
- The plain text file `submitter.txt`.
- Your report, in PDF format, named `report.pdf`.

Please submit only the above files (do not submit your index, the collection, or your output!) using the following command:

```bash
$ /course/cs158/bin/cs158-handin final
```

The due date for this part of the Final is Thursday May 9nd 2013 at 11:59am.
3 Data

For this assignment you will need the following files located in /course/cs158/data/final/:

- fullCollection.dat: the full collection (warning: 325 MB!);
- features.dat: a set of term features;
- categories.dat: list of the categories;
- resultSVM.txt: a plain text file you should fill in with the SVM classification results;
- queries.dat: a set of sample queries on the full collection;
- createIndex.sh: sample Bash script to invoke your createIndex program;
- queryIndex.sh: sample Bash script to invoke your queryIndex program;
- readme.txt, submitter.txt: plain text files you should fill in with your information.

4 Collaboration and Help

If you worked with another student during the previous assignments, you should continue working with this same student for the first phase.

The work for the second phase should be carried out individually, in particular experimenting with the advanced ranking scheme and writing the report. During the second phase you should not collaborate with neither your team-mate nor anyone else, that is, all experiments, comparisons between various methods and example queries should be your own work.

Email cs158tas at cs dot brown dot edu if you have any questions.

5 200-Level Credit

We will publish the specification for 200-level credit on Fri 22 Apr 2011, as a separate PDF. We will send out an email announcing its publication. The due date will be on Sun 8 May 2011 at 11:59am.
...And that, in simple terms, is how you increase your ranking on search engines.”

Have fun!