CS2580: Web Search Engines Fall 2014

Homework 3

Due November 24, 2014, 5pm EST Department of Computer Science, NYU

1 Overview

In this homework assignment, you have two main tasks: 1) implementing components for computing intrinsic document qualities such as PageRank and NumViews; 2) implementing and analyzing pseudo-relevance feedback for query expansion. For the first task, we will provide you with the skeleton code that you will augment with the desired functionalities. For the second task, we will not be providing skeleton code and you will need to design the class based on what we have.

Feel free to use energon machines in place of linserv machines.

2 Computing Document Qualities

We have introduced two new sets of classes and a new run mode for the document quality computation.

1. Mode **mining**: `congyu@linserv1[code]$ java -cp src edu.nyu.cs.cs2580.SearchServer \ --mode=mining --options=conf/engine.conf`

   The **mining** mode should be run before the **index** mode to compute the intrinsic document qualities, which can then be loaded during the index mode.

2. `CorpusAnalyzer` and `CorpusAnalyzerPagerank`: Those two classes provide the skeleton for PageRank computation. The `prepare` and `compute` functions are called during the **mining** mode and the `load` function should be called during the **index** mode (in your code).

3. `LogMiner` and `LogMinerNumviews`: Those two classes provide the skeleton for Numviews computation. The `compute` functions are called during the **mining** mode and the `load` function should be called during the **index** mode (in your code).

2.1 Computing PageRank

Using the same Wikipedia corpus as in HW2, you should implement your PageRank computation in two steps. In the first step, **prepare**, a Corpus Graph consisting of only the documents in our corpus should be generated (redirects should be properly handled). In the second step, **compute**, the PageRank value should be computed and stored. The last step, **load**, should load the stored PageRank values for the documents and index those values together with the tokens.

You can design your own format for storing the Corpus Graph. As in HW2, you should be aware of the memory consumption of your code.

2.1.1 Further Details

PageRank is an iterative process, compute the values using the following settings:

- one iteration with $\lambda = 0.10$
- two iterations with $\lambda = 0.10$

- one iteration with $\lambda = 0.90$
- two iterations with $\lambda = 0.90$

Choose one with the best estimated results for search queries and discuss your choices in a readme.txt file.

2.2 Computing NumViews

We further provided you with a log file at `data/log/20140601-160000.log`, which you should be using to compute NumViews for documents within the corpus. The format of the log is self-evident, where the document is identified by the base name (not the full path). Use your own judgement for documents not appearing in the log. Similar to PageRank computation, NumViews should be computed inside **compute** function and be loaded and indexed inside the **load** function.

2.3 Grading

We will run your homework with the three modes successively, **mining**, **index**, and **serve**. We will grade based on a test set of queries (see details in `Grader.java`). Similar to previous homeworks, in the **serve** mode, your code should load in the index on demand and return top results for the queries we issue against your search engine.

In addition to the ranked list of results based on your own ranking score (which takes into consideration topic relevance, PageRank, and NumViews), you search engine should also return:

1. For each result, its PageRank.
2. For each result, its NumViews.

3 Comparing PageRank and NumViews

As we mentioned in lecture, PageRank is an approximation to visitation information. In this part, we are interested in measuring the accuracy of this approximation. In order to do this, we will use *Spearman's rank correlation coefficient*. This metric is defined by converting the scores of a set of items into there rank values. You will perform this transformation once for documents sorted by PageRank and once for documents sorted by the number of views, each in *decreasing* order.

As an example, if document i had the top PageRank, then $x_i = 1$. If document j had the fourth best PageRank, then $x_j = 4$. If document i had the fifth highest number of views, then $y_i = 5$. If document j had the second highest number of views, then $y_j = 2$. If we compute these rank values for all documents, then the Spearman rank correlation is,

$$\rho = \frac{\sum_k (x_k - z)(y_k - z)}{\sum_k (x_k - z)^2 \sum_k (y_k - z)^2}$$

where $z = \frac{\sum_{k=1}^{n} x_k}{n}$ and n is the number of documents we are considering.

Compute the Spearman rank correlation coefficient between PageRank-based ranking and NumViews-based ranking for the documents in the collection. Because these values are query-independent, you only need to compute this value once. Please submit code to compute the Spearman rank correlation coefficient between each of the PageRank computations and the number of views. Your code should take as input the output from Sections 2.1 and 2.2. We should be able to call your code in the following way,

```
$ java edu.nyu.cs.cs2580.Spearman <PATH-TO-PAGERANKS> <PATH-TO-NUMVIEWS>
```

The output should be a single value measuring the correlation. Place the result in readme.txt.

3.1 Grading

You should submit your code for computing the correlation between PageRank and NumViews. Your code should run as described (i.e., you need to provide your own main class). You should also submit a readme.txt with the Spearman correlation in it.

4 Pseudo-Relevance Feedback

Pseudo-relevance feedback refers to the technique used to expand a query by using terms from the top k documents retrieved. In this part of the assignment, we will ask you to explore pseudo-relevance feedback for query representation and similarity.

4.1 Query Representations

The easiest way to compute a query representation is to take the most frequent m terms in the top k documents. You are going to modify your ranker so that we can retrieve an expanded query. Specifically, given a query, we would like you to compute the following,

$$p(w|D) = \frac{\sum_{d \in D} \#(w, d)}{\sum_{w'} \sum_{d \in D} \#(w', d)}$$

where D is the set of the top k documents. We will only need you to return the top m terms ranked by this probability.

That is, your request should have the form,

```
http://<HOST>:<PORT>/prf?query=<QUERY>&ranker=<RANKER-TYPE>&numdocs=<INTEGER>&numterms=<INTEGER>
```

where any arguments should be URI-encoded. The system should output tab-separated and follow the format,

```
<TERM-1><PROB-1>
<TERM-2><PROB-2>
...
<TERM-m><PROB-m>
```

where the probabilities are defined as $p(w|D)$. **Please renormalize these probabilities so that the values output sum to 1.**

For each query in `queries.tsv`, compute the expansion terms and make one file per query for use in the next part.

4.2 Query Similarity

One way to compute the similarity between queries is by using the similarity between the expanded queries. In this section, will be computing the similarity between the distributions computed in the first part. Specifically, you will be using the Bhattacharyya coefficient. This coefficient is defined as,

$$\beta(q_i, q_j) = \sum_{w \in \mathcal{V}} \sqrt{p(w|q_i) \times p(w|q_j)}$$

where $p(w|q)$ are the probabilities we output in the first part of this question.

Please submit code to compute the Bhattacharyya coefficient between all pairs of queries in an input file. Your code should take as input the path to directory with the output from the first part. We should be able to call your code in the following way,

```
$ java -cp src edu.nyu.cs.cs2580.Bhattacharyya <PATH-TO-PRF-OUTPUT> <PATH-TO-OUTPUT>
```

The output should be,

```
<QUERY-1><QUERY-2><COEFFICIENT>
<QUERY-1><QUERY-3><COEFFICIENT>
...
<QUERY-1><QUERY-m><COEFFICIENT>
...
<QUERY-m><QUERY-(m-1)><COEFFICIENT>
```

4.3 Grading

Your code *must* be able to run on a new set of queries using steps such as the following,

```
$ rm -f prf*.tsv
$ i=0
$ while read q ; do
  i=$((i + 1));
  prfout=prf-$i.tsv;
  http://localhost:12345/prf?query=$q&ranker=ql&numdocs=10&numterms=5 > $prfout;
  echo $q:$prfout > prf.tsv
  done < queries.tsv
$ java -cp src edu.nyu.cs.cs2580.Bhattacharyya prf.tsv qsim.tsv
```