CS 276 / LING 286 Spring 2015

Programming Assignment 3: Ranking

Due: May 14, 2015, 11:59 PM PDT

1 Overview

In this programming assignment, you will devise ranking functions. Given some queries and corresponding search results, your task is to rank the results. For each query-document pair, you are provided with several features that will help you rank the documents. You are also provided with a training set consisting of query-document pairs along with their relevance values. We will be implementing three different ranking functions and will use the NDCG metric for evaluating the effectiveness of the ranking function. The estimation of parameters for the ranking functions will be done manually (i.e, no machine learning).

2 Data

Note: The data set released along with the skeleton code comes from class 2014, and is located under the directory `2014.data/`. You can start coding and debugging with this old dataset. However, we will notify you on Piazza and release the new document rating data set that we have collected from you (class 2015) soon after we release this assignment. **Please make sure that you replace the old dataset with the new one, tune your parameters and submit the assignment with the new dataset.**

In this assignment, we have pre-partitioned the data into two sets for you: (a) training set (295 queries) and (b) development set (97 queries). The idea is while tuning and maximizing performance on the training set, you should also verify how well your tuned parameters are doing on the development set to make

sure you are not overfitting your training data. The data files are in the `data/` directory: (a) train – pa3.(signal|rel).train and (b) dev – pa3.(signal|rel).dev. **Make sure you report performances on both the training and development sets for all tasks**. There is a hidden test set (97 queries) which we have reserved to evaluate your final system. For each set, there are two types of files:

1. **Signal File** – pa3.signal.(train|dev): lists queries along with documents returned by a widely used search engine for each individual query (the list of documents is randomized and is not in the same order as returned by the search engine). Most queries will contain 10 documents, but it is possible for certain queries to be matched with less documents. The format for a pair of query/document (*qd*) is as follows.

```
query: 2015 math requirements stanford
  url: http://math.stanford.edu/
    title: department of mathematics stanford university
    header: Stanford Math Department
    header: Latest publications in math
    body_hits: stanford 23 44 92 159 165
    body_hits: 2015 97 118
    body_length: 251
    pagerank: 5
    anchor_text: http math stanford edu
      stanford_anchor_count: 44
    anchor_text: stanford math department
      stanford_anchor_count: 9
```

   This pattern repeats for the next url until all of the urls for this query are done and then the overall pattern repeats for the next query. There is only one title, pagerank and body_length for each url but there can be multiple header, body_hits and anchor_text (and corresponding stanford_anchor_count) lines.

   The *body_hits* line specifies the term followed by the positional postings list of that term in the document (sorted in increasing order). The *body_length* line states how many terms are present in the body of the document. The *stanford_anchor_count*, specified immediately after the *anchor_text line*, states how many anchors there are on the stanford.edu domain with that anchor text. For example, if the anchor text is "stanford math department" and the count is 9, that means there are nine links to the current page (from other pages) where the anchor text is "stanford math department". The *pagerank* is an integer from 0 to 9 that signifies a query-independent quality of the page (the higher the pagerank, the better the quality of the page). Each *header* line corresponds to a header (h1/h2/h3/h4 html tags) occurring on the page. Only headers that have query term hits are listed.

2. **Relevance File** – pa3.rel.(train|dev): lists the relevance judgments for each of the query-document pairs in the corresponding signal file. The relevance judgment is treated as a number ranging from −1 to 3 with a higher value indicating that the document is more relevant to that query. To work with the NDCG evaluation metric, we consider all relevance ratings less than zero to be equivalent to zero instead (Section 3). The format of this document is as follows:

   ```
   query: 2015 math ugrad requirements stanford
    url: https://www.stanford.edu/dept/pe/cgi-bin/ 2
    url: http://lksc.stanford.edu/students/fitness-center.html 1
    url: http://tusb.stanford.edu/tag/gym -0.5
   ```

   This pattern repeats for the next query until all of the queries in the file are done. The url line can be broken into the document url and the relevance judgment for the query-document pair.

The ranking functions also require certain collection-wide statistics (such as inverse document frequency) and we cannot infer this information just from the training set itself. As a result, you will also need to access the corpus from PA1 to derive the above statistics.

3 NDCG

The evaluation metric that will be used is Normalized Discounted Cumulative Gain (NDCG) since we are using a non-binary relevance metric. Since each query has at most 10 results returned, we use NDCG for the first 10 search results.
Then, for a particular query q,

$$\text{NDCG}(q) = \frac{1}{Z} \sum_{m=1}^{p} \frac{2^{R(q,m)} - 1}{\log_2(1+m)} \tag{1}$$

Here, $R(q, m)$ is the relevance judgment given to document m for query q. Z is a normalization factor. It is the the ideal NDCG (iNDCG) value. The ideal NDCG value is calculated by ordering the documents in decreasing order of relevance and calculating the NDCG value with Z=1. If iNDCG is zero, NDCG(q) = 1. Finally, p is the number of documents that are possible matches for that query.
We can compute the NDCG for a set of queries $Q = \{q_1, \ldots, q_m\}$ by taking the average of the NDCGs for each of the individual queries. The starter code contains a Java implementation of NDCG which you can use directly to evaluate your ranking function on the training data. We will be using the same file to evaluate your ranking on test data.

4 Ranking

4.1 Term Scores

In signal files in the training data, each query-document pair provides term information from five different fields: url, title, headers, body and anchors (it provides pagerank also but we won't be using it in cosine similarity. Even for BM25F, we will consider it separately as explained in Section 6). Each of the required ranking functions will construct a term score (tf) vector for each query-document pair from hits in these different fields. All of our ranking functions only care about terms that occur in the query.

The raw term score vector, rs, counts how many times a query term occurs in a field. For the anchor field, we assume that there is one big document that contains all of the anchors with the anchor text multiplied by the anchor count. A similar approach can be followed for the header field as well. Thus, in the qd example whose term-vector is $\begin{bmatrix}2015 & math & requirements & stanford\end{bmatrix}^T$, the rs vector for the body field will be $\begin{bmatrix}2 & 0 & 0 & 5\end{bmatrix}^T$ as there are 2 hits for the term "2015" in the body field and 5 hits for the term "stanford". Similarly, the rs vector for the anchor field will be $\begin{bmatrix}0 & 53 & 0 & 53\end{bmatrix}^T$ as there are 53 total anchors that contain the terms "math" and "stanford". Finally, the rs vector for the title field is $\begin{bmatrix}0 & 0 & 0 & 1\end{bmatrix}^T$, for the url field is $\begin{bmatrix}0 & 1 & 0 & 1\end{bmatrix}^T$ and that for the header field is $\begin{bmatrix}0 & 2 & 0 & 1\end{bmatrix}^T$. Note that in order to extract url hits, you will have to tokenize the url on non-alphanumeric characters.

While calculating the raw term scores, we convert everything to lowercase and then calculate the counts. The body_hits given in the data do not perform any stemming. However, for the other fields, you are free to experiment with different techniques like stemming etc.

4.2 Output Requirements

In all three tasks, the goal is to derive specific types of ranking functions based on the training data and relevance values. Once the ranking function rf has been crafted, we will then pass in the test data set and your application must use rf to rank the query-document pairs and output the list of documents for each query in decreasing rank order. The NDCG evaluation metric will then be applied on these lists against the evaluation provided by you in the search ratings task earlier in the course. Higher the value, the better your ranking algorithm works.

5 Task 1 - Cosine Similarity

The first task is to implement a variant of cosine similarity (with the L1-Norm) as the ranking function. This essentially involves constructing the *document vector* and the *query vector* and then taking their dot product. Recall from Figure 6.15 in the text book that in order to construct the vectors, we need to

decide on how we compute a term frequency, a document frequency weighting, and a normalization strategy. Let's discuss these for both the vectors separately.

5.1 Document vector

- **Term frequency**
  We compute the raw term frequencies for each query term in the different fields using the method described in Section 4.1. For each of the fields, we can compute the tf vector, either using the raw scores themselves or by applying sublinear scaling on the raw scores (In sublinear scaling, we have $tf_i = 1 + log(rs_i)$ if $rs_i > 0$ and 0 otherwise. Thus, the tf vector for the body field for qd will be $\begin{bmatrix}1.6931 & 0 & 0 & 2.6094\end{bmatrix}^T$. More information about sublinear scaling is described in IIR 6.4.1). **Please describe the reasons for making that choice in the report.**

- **Document frequency**
  We will not use any document frequency in the document vector. Instead, it is incorporated in the query vector as described below.

- **Normalization**
  We cannot use cosine normalization as we do not have access to the contents of the document and, thus, do not know what other terms (and counts of those terms) occur in the body field. As a result, we use length normalization instead. Moreover, since there can be huge discrepancies between the lengths of the different fields, we divide all fields by the same normalization factor, the body_length (Note that some documents have a body_length of 0, so you will have to smooth them somehow. A good strategy is to add a value, say 500, to the body length of each document. You can experiment with this value or with other smoothing strategies and report them).

5.2 Query vector

- **Term frequency**
  The raw term frequencies can be computed using the query (should be 1 for most queries but not necessarily true). Again, you can use either the raw frequencies or sublinearly scale them. **Please mention your choice and the reasons behind it in the report.**

- **Document frequency**
  Each of the terms in qv should be weighted using the idf value for each of the terms in the query. Computing the idf requires going to the corpus from PA1 to determine how many documents contain the query terms. One issue is that it is possible for a query term t to not appear in the collection corpus and it is not possible to evaluate idf_t. In such a case, we will apply the Laplace add-one smoothing technique learned earlier in the

course (This essentially assumes the existence of a hypothetical dummy document that contains all possible terms, and therefore, adds 1 to each numerator and denominator with the idf_t formula).

- **Normalization**
  No normalization is needed for query length because any query length normalization applies to all docs and so is not relevant to ranking.

For a document d and query q, if qv_q is the query vector and $tf_{d,u}$, $tf_{d,t}$, $tf_{d,b}$, $tf_{d,h}$ and $tf_{d,a}$ are the term score vector for the url, title, body, header and anchor fields, respectively, then the net score is $qv_q \cdot (c_u \cdot tf_{d,u} + c_t \cdot tf_{d,t} + c_b \cdot tf_{d,b} + c_h \cdot tf_{d,h} + c_a \cdot tf_{d,a})$. Here, c_u, c_t, c_b, c_h and c_a are the weights given to url, title, body, header and anchors fields, respectively.
The goal is to determine the weights for all 5 fields (and, thus, the ranking function using cosine similarity) so that the NDCG function is of a optimal value when run on the test set. You will use the training set given to derive the above parameters. **In the report, you should mention parameter values and describe briefly the intuition and reasons behind why the weights were selected.**
Hint: Note that the absolute values of weights won't matter as they will be the same for all documents, only the relative weights for different fields is important; i.e. you can multiply each weight by a constant and the ranking will remain the same. In order to estimate the relative weights, try to reason the relative importance of the different fields.

6 Task 2 - BM25F

The second task is to implement the BM25F ranking algorithm. The algorithm is decribed in detail in the lecture slides. Specifically, you should have a look at slides 30-32 of the BM25F lecture `http://www.stanford.edu/class/cs276/handouts/lecture12-bm25etc.pdf` before reading further. Here, instead of using the term scores from Section 4.1, we use field-dependent normalized term frequency (ftf). Thus, for a given term t and field $f \in \{url, header, body, title, anchor\}$ in document d,

$$ftf_{d,f,t} = \frac{tf_{d,f,t}}{1 + B_f((len_{d,f}/avlen_f) - 1)} \tag{2}$$

where $tf_{d,f,t}$ is the raw term frequency of t in field f in document d, $len_{d,f}$ is the length of f in d and $avlen_f$ is the average field length for f. The variables $avlen_{body}$, $avlen_{url}$, $avlen_{title}$, $avlen_{header}$ and $avlen_{anchor}$ can be computed using the training set. B_f is a field-dependent parameter and must be tuned for this task. If $avlen_f$ is zero (should not happen in this dataset), then $ftf_{d,f,t} = 0$. Then, the overall weight for the term t in document d among all fields is

$$w_{d,t} = \sum_f W_f \cdot ftf_{d,f,t} \tag{3}$$

Here, W_f is also a field-dependent parameter that determines the relative weights given to each field. This value is similar in theory to the tuning parameters for Task 1.
Since, we also have a non-textual feature, in the form of *pagerank*, we incorporate it into our ranking function using the method described in slide 30 of the lecture. Therefore, the overall score of document d for query q is then:

$$\sum_{t \in q} \frac{w_{d,t}}{K_1 + w_{d,t}} idf_t + \lambda V_j(f) \tag{4}$$

where K_1 is also a free parameter and V_j can be a log/saturation/sigmoid function as mentioned in the slides (you will need to experiment with the other parameter λ' used by the V_j function).
Thus, for this task, there are a minimum of 13 parameters to optimize, namely $B_{url}, B_{title}, B_{header}$, B_{body}, B_{anchor}, $W_{url}, W_{title}, W_{header}$, W_{body}, W_{anchor}, λ, λ' and K_1. Additionaly, you also have to select the V_j function appropriately (**include the reasoning behind your choice in the report**). While in theory, BM25F should give a better NDCG value as it incorporates a lot of more information, this need not necessarily be the case. **In the report, you should mention all the parameter values and also describe the reasons as to why the weights work in getting a good NDCG value.**
Hint: The weight values obtained in Task1 may be a good starting point for this task. Again note that the weights will depend on the "importance" of the fields. Moreover, as mentioned in the slides, log(pagerank) works well in practice but you should try other functions as well and see how they work.

7 Task 3 - Smallest Window

The final task is to incorporate window sizes into the ranking algorithm from Task 1 (or Task 2 if you prefer). For a given query, the smallest window $w_{q,d}$ is defined to be the smallest sequence of tokens in document d such that all of the terms in the query q for are present in that sequence. A window can only be specific to a particular field and for anchor fields, all of the terms in q must be present within a particular anchor text (i.e, if one term occurs in one anchor text and another term in a different anchor text, then it cannot be considered for a window). If d does not contain any of the query terms or a window cannot be found, then $w_{q,d} = \infty$. Intuitively, the smaller $w_{q,d}$ is, the more relevant the document should be to the query. Thus, we can also multiply the document score (from Task 1 or Task 2) by a boost based on w such that:

If $w_{q,d} = \infty$, then the boost is 1.

If $w_{q,d} = |Q|$ where Q are the unique terms in q, then we multiply the score by some factor B.

For values of $w_{q,d}$ between the query length and infinite, we provide a boost between B and 1. The boost should decrease rapidly with the size of $w_{q,d}$ and can decrease exponentially or as $\frac{1}{x}$.

Thus, for this task, there are either 6 or 14 parameters to optimize, depending on whether you decide to modify cosine similarity or BM25F. The choice of function to use when the window size is not the same as the query length is another factor to also consider.
As with the previous tasks, you should describe the reasons for the weights chosen for this task.

8 Extra Credit

Extra credit will be given if additional ranking algorithms are derived that incorporate other signals indicating relevance of a document to a particular query. For example, like in task 3, we use the smallest window as a signal where a smaller window size indicates that a document is more likely going to be matched with a query. Credit will be given based both on the ideas of the signals used as well as the performance of ranking algorithm on the test set.

9 Deliverables

9.1 Input/Output format

The starter code contains a script named `rank.sh`. The script can be invoked as follows:

```
$ ./rank.sh <inputDataFile> <taskId>
```

where the 2 arguments are as follows:

- inputDataFile - File containing information for all query/url pairs to be ranked (in the same format as the signal files)
- taskType - 'baseline', 'cosine' (Task 1), 'bm25' (Task 2), 'window' (Task 3), or 'extra' (Extra Credit).

Feel free to change `rank.sh` according to your needs. Your program should build the ranking algorithm based on the task given. You can read in whatever training data you need to build your model but we will not pass that as inputs to the script (so in a nutshell, hardcode the relative path to such files either in the script or in the code). The script should output (to stdout) for each query, the query followed by the documents (in the form of urls) in decreasing rank order specified by your ranking. You can print anything you want to stderr.
For example, if query $q1$ has three documents and the file is listed as follows:

```
query: q1
  url: http://xyz.com
  ...
```

```
  url: http://def.edu
  ...
  url: http://ghi.org
```

And if your ranking algorithm gives a rank of 1 to ghi.org, 2 to xyz.com and 3 to def.edu, then you should output the order in the following format:

```
query: q1
 url: http://ghi.org
 url: http://xyz.com
 url: http://def.edu
query: q2
 url: ...
 .
 .
```

9.2 Report

A write-up of the design choices undertaken for the various tasks as well as the actual parameter values must be mentioned in `report.pdf`. Explanations as to why the parameters work as well as the overall effectiveness of the ranking functions must also be mentioned. It must be a maximum of 3 pages long. Reports that are too terse or those that do not contain enough analysis will not get full credit.
In particular, you need to address the following questions in the report:

1. What was the reasoning behind giving the weights to the url, title, body, header and anchor fields for the three tasks? Were there any particular properties about the documents that allowed a higher weight to be given to one field as opposed to another?

2. What other metrics, not used in this assignment, could be used to get a better scoring function from the document? The metrics could either be static (query-independent, e.g. document length) or dynamic (query-dependent, e.g. smallest window).

3. In BM25F, in addition to the weights given to the fields, there are 8 other parameters, $B_{url}, B_{title}, B_{header}, B_{body}, B_{anchor}$, λ, λ' and K_1. How do these parameters affect the ranking function?

4. In BM25F, why did you select a particular V_j function?

5. For a function that includes the smallest window as one component, how does varying B and the boost function change the performance of the ranking algorithm?

When including the parameter values (and the reasoning behind them in the report), please make sure they follow the following naming convention:

- For Task 1:

  `task1_W_<field name>`

  where `<field name>` can be url, title, body, header or anchor.

- For Task 2:

  `task2_W_<field name>`
  `task2_B_<field name>`

  where `W_<field name>` is equivalent to $W_{<fieldname>}$ from equation 3, `B_<field name>` is equivalent to $B_{<fieldname>}$ from equation 2 . For the other parameters, K_1, λ, λ' and V_j, use the same names as used in equation 4.

- For Task 3, just add a prefix

  `task3_`

  to the weight names cooresponding to the task that was modified Task 1 or Task 2 is modified. Additionally, please specify B.

- For extra credit, please name your weights in a similar way that is understandable. A brief description of what the weights depict in your extra credit model should also be added.

9.3 Partner

A list of people who worked together on the assignment, in *people.txt* One line per student:

`<sunet id1>`
`<sunet id2>`

10 Grading

We will be evaluating your performance on a different test dataset, which will have queries drawn from the same distribution as the training set. The format for the dataset is the same as the signal files we have provided you with.

Report: 45%. It should describe the various design choices used in determining the parameters for the various tasks. See Section 9.2 for the criteria required for the report.

Task 1: 15%. Your grade will be computed relative to the NDCG scores of others in the class (you will be penalized if you are "a lot" worse than others, where the definition of "a lot" will be decided post hoc based on the curve).

Task 2: 15%. Same as task 1.

Task 3: 10%. You get the full 10% if the NDCG score is able to exceed either Task 1 or Task 2 (depending on what you chose to modify) by a small amount.

Correctness/Code: 10%. A check to ensure that you are computing the tasks required such as cosine similarity and smallest window.

Queries + Relevance: 5%. Automatically added if you submitted queries (1%) and gave relevance ratings (4%) for query-document pairs. We will update this based on your submissions for the queries and search ratings quizzes.

Extra Credit: 10%. See Section 8.

An additional 10% is given if any of your ranking algorithms has the best overall NDCG value in the entire class, 8% for the runner up and 5% for coming in third place.

11 Submission Instructions

Make sure the shell script `rank.sh` runs as explained in Section 9.1. Then, just call the following command:

```
python submit.py
```

and select the appropriate option to submit Tasks 1, 2, 3, Extra credit or the report.
The submit script will execute your model on the test data on your machine, so it may take some time if you decide to build the model again. It is recommended to hard-code the weights for each of the tasks so that a rebuilding of the model is not needed. It will also check to see whether you have the right number of query-document pairs afterwards. Remember, it's an honor code violation to knowingly take a look at the test set.
Since `NdcgMain.java` will be used to evaluate the NDCG scores of your model on the test data, **the messages generated by stdout must conform to the output format mentioned earlier**. It is your responsibility to make sure that any debugging/logging output to stdout that you added for convenience has been removed or commented out before submission. In particular, you cannot have any blank lines in your standard output. Additionally we have also provided starter code in Java which includes a variety of functionality to hopefully make your life easier (at least in indicating roughly what needs to be done). Note that the starter code is for your convenience – feel free (and you are in fact encouraged) to modify anything as long as you conform to the specifications required by the submit script.

You can start by giving the *inputDataFile* and "baseline" arguments to `Rank.java`, and the code will produce a ranking. You can also score the result using `NdcgMain.java`.

As a last step, you will have to submit your code on Coursera. Zip your assignment directory using a zip archiver (without the PA1 corpus), name it as `SUNetid1_SUNetid2.zip` and upload it on the Coursera assignments page for the "Code" section using the simple uploader you have used before. Do not worry about making submissions for the other parts as we will populate those with scores using your reports and code.

Only one person in the group needs to submit the assignment (Please submit everything from the same member's SUNetid and with the same `people.txt` file)