CS 276 / LING 286 Spring 2016

Programming Assignment 3: Ranking

Due: May 12, 2016, 11:59 PM PDT

1 Overview

In this programming assignment, you will devise ranking functions to rank results given some queries and corresponding search 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 Starter Code and Data

2.1 Starter Code and Tutorial

Download the Java starter code from: `http://web.stanford.edu/class/cs276/pa/pa3-skeleton.zip`. You can use ant or Eclipse to build the code. An accompanying tutorial video that will help you walk through the code was put together by the teaching staff last year and is available here: `https://www.coursera.org/learn/cs276/lecture/eW7sj/pa3-tutorial`. Make sure you are signed up for the course on Coursera. Note we modified the code several places this year, but the principles will be same.

2.2 Data

The data for this assignment is available as a `.zip` file at the following url:

`http://web.stanford.edu/class/cs276/pa/pa3-data.zip`. We have partitioned the data into two sets for you:

1. Training set of 240 queries (`pa3.(signal|rel).train`)
2. Development set of 100 queries (`pa3.(signal|rel).dev`)

The idea is that while tuning and maximizing performance on the training set, you should also verify how well the tuned parameters are doing on the development set to ensure you are not overfitting your model. There is a hidden test set of 106 queries which we have reserved to grade your final model. 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 shuffled and is not in the same order as returned by the search engine). Each query has 10 or less documents. For example, 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 collected relevance data was an integer ranging from -1 to 3 with a higher value indicating that the document is more relevant to that query. We have averaged relevance scores for each query-url pair with -1 ignored. For example, the format of this document is as follows:

```
query: campus acreage
  url: http://engage.stanford.edu/2014/03/page/2/ 1.00
  url: http://engage.stanford.edu/tag/cultural/ 1.00
  url: http://facts.stanford.edu/about/lands 2.00
  ... ...
```

   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. You will find this corpus at `http://web.stanford.edu/class/cs276/pa/pa1-data.zip`.

3 NDCG

The evaluation metric 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 method to evaluate your ranking on grading data.

4 Ranking

4.1 Term Scores

In the signal files of the training data, each query-document pair provides term information from five different fields: `url`, `title`, `headers`, `body` and `anchors`. Additionally each pair provides `pagerank` 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` field 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. The 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[1] in the textbook 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 tf scaling is described in Section 6.4.1[2] of the textbook.

- **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

[1] Pg. 128 http://nlp.stanford.edu/IR-book/pdf/06vect.pdf
[2] Pg. 126 http://nlp.stanford.edu/IR-book/pdf/06vect.pdf

for most queries but not necessarily true). Again, you can use either the raw frequencies or sublinearly scale them.

- **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 [3].

- **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

$$qv_q \cdot (c_u \cdot tf_{d,u} + c_t \cdot tf_{d,t} + c_b \cdot tf_{d,b} + c_h \cdot tf_{d,h} + c_a \cdot tf_{d,a}) \tag{2}$$

Here, c_u, c_t, c_b, c_h and c_a are the weights given to `url`, `title`, `body`, `header` and `anchor` 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 an optimal value when run on the test set. You will use the training set given to derive the above parameters.

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 [4] 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 $f \in \{url, header, body, title, anchor\}$

[3] 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.

[4] `http://www.stanford.edu/class/cs276/handouts/lecture12-bm25etc.pdf`

in document d,

$$ftf_{d,f,t} = \frac{tf_{d,f,t}}{1 + B_f((len_{d,f}/avlen_f) - 1)} \tag{3}$$

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

$$w_{d,t} = \sum_f W_f \cdot ftf_{d,f,t} \tag{4}$$

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 the BM25 lecture regarding ranking with non-textual features.

Therefore, the overall score of document d for query q is then:

$$\sum_{t \in q} \frac{w_{d,t}}{K_1 + w_{d,t}} idf_t + \lambda V_j(f) \tag{5}$$

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
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.

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.

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 three scripts: `rank.sh`, `ndcg.sh` and `flow.sh`.

9.1.1 rank.sh script

`rank.sh` can be invoked as follows:

```
./rank.sh <sigPath> <taskOption> <idfPath> <buildFlag>
```

where the 4 arguments are as follows:

- `sigPath` - Signal file containing information for all query/url pairs to be ranked, located under the data directory.

- `taskOption` - 'baseline', 'cosine' (Task 1), 'bm25' (Task 2), 'window' (Task 3), or 'extra' (Extra Credit).
- `idfPath` - When args[3] (buildFlag) is 'true', set this as your PA1 corpus path. When args[3] (buildFlag) is 'false', set this as your existing idfs file.
- `buildFlag` - Set to 'true', will build idf from PA1 corpus. Set to 'false' to load from existing idfs file.

Feel free to change `rank.sh` according to your needs, but be sure to include your `rank.sh` script in your submission directory at submission time. The `LoadHandler` class will be useful to generate idf information and save it to file. Instead of parsing the PA1 corpus every time, you may also load the idf file directly in your debugging for convenience. But we will delete existing idfs file and test your idf generation block by feeding in the PA1 corpus during grading.

The script should output to `stdout` with each query, following by the documents (urls) in decreasing order specified by your ranking. In your submission, keep `stdout` of `rank.java` as formatted in `printRankedResults` method, otherwise your grade may be negatively influenced.

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
 ... ...
```

9.1.2 ndcg.sh script

`ndcg.sh` can be invoked as follows:

```
./ndcg.sh <rankFile> <relFile>
```

where the 2 arguments are as follows:

- `rankFile` - File containing your ranking result from Section 9.1.1.
- `relFile` - File containing the relevance score, located under data folder.

This script will call `NdcgMain` class and print NDCG score for you. You do **not** need to modify this class. When grading, we will run your code with our own copy of `NdcgMain` class.

9.1.3 flow.sh script

This script `flow.sh` is mainly used to check if the `stdout` from `Rank.java` is valid. Make sure this script only outputs NDCG score, with no error/warning on your final submission.

```
./flow.sh <sigPath> <taskOption> <idfPath> <buildFlag> <relFile>
```

where the 5 arguments above are defined same from Sections 9.1.1 and 9.1.2.

9.2 Report

Please write up a 1-3 page report describing the design choices undertaken for the various tasks as well as the actual parameter values. Explanations as to why the parameters work as well as the overall effectiveness of the ranking functions must also be mentioned.
In particular, you need to address the following questions in the report:

1. Report NDCG on both training and development sets for all tasks.
2. For the three tasks, you should report all final model parameter values. Describe the intuition when tuning your models, and why those weights work in getting a good score. Were there any particular properties about the documents that allowed a higher weight to be given to one field as opposed to another?
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 task 1, you may either use raw frequencies or sublinearly scale them to compute term frequency. Please report your choice and the reasons behind them. For BM25F, why did you select a particular V_j function?
5. Briefly describe your design of smallest window. 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?
6. 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).

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 4, `B_<field name>` is equivalent to $B_{<fieldname>}$ from equation 3 . For the other parameters, K_1, λ, λ' and V_j, use the same names as used in equation 5.

- 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.

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 and dev set. The format for the dataset is the same as the signal files that we provided you.

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 your fellow students 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 BM25 properly. We will run our benchmark code with your tuned parameters.

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.

Submission

We will use corn and Gradescope for grading and collecting the assignment. Make sure that all `.sh` scripts are executable. There are two steps for submission:

1. Submitting your code on corn using our `submit` script
2. Uploading your report as a pdf file to Gradescope

Submitting Code

Before, submitting **please make sure that** your code follows the following directory structure to ensure auto-grading goes smoothly.

```
pa3-skeleton
├── build.xml
├── src
│   └── <any source files of your program>
├── ndcg.sh
├── rank.sh
├── flow.sh
└── <any additional files your program needs>
```

Note that we will compile and run your program on Farmshare using `ant` and our standard `build.xml`. If you did not complete the assignment on Farmshare, please verify your code compiles and runs on it before submission.

You will submit your program code using a Unix script that we've prepared. To submit your program, first put your files in a directory on Farmshare (e.g. corn.stanford.edu). Then, from your parent directory (this would be `pa3-skeleton` in the above directory structure), submit, i.e.:

```
cd pa3-skeleton
/afs/ir/class/cs276/bin/submit
```

If you are working in a team, only one team member needs to submit, but remember to indicate your partner's SUNetID when prompted by the submit script.

Uploading Report to Gradescope

Please upload your report to Gradescope under the Assignments section and make sure to select all the pages for the "Report" component. Again, if you are working in a team, only one team member needs to submit, but remember to add your partner as a team member for the assignment on Gradescope.

11 Final Notes

Since `NdcgMain.java` will be used to evaluate the NDCG scores of your model on the ranked result, **the messages generated by Rank.java `stdout` must conform to the output format mentioned earlier**. In the grading, we will run your code with our own copy of `NdcgMain` class. 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 or else your grade for the code portion may suffer. 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. But for whatever you modify, e.g. `rank.sh`, please be sure to include this in your submission directory when submitting your code.