CS6200: Project 2

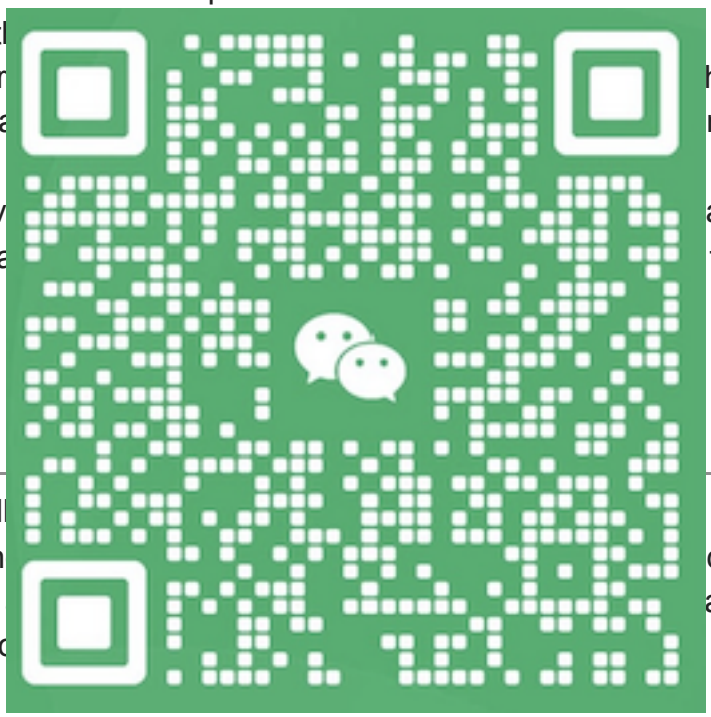
course.khoury.northeastern.edu/cs6200s14/pr2/pr2.html

CS6200 Project Two

important Submit by committing your files to svn. See the checklist below to make sure you turn everything in.

Corrections

- The BM25 formula has been updated: you should only take the log of the first term. This term should never be zero, so your program should never take log(0).
- The BM25 formula has been updated: the third term should have \(k_2\) in the denominator rat
- The Jelinek-Mer hould have separate summa my original note was correct.
- Note: it is irrelev a\) and the second by \(1-\la to set \(\lambda\) to (for instance)



Overview

In this project, you wil documents and create a search engin ctions and compare their results alysts. You will then write a short repo

Report

Write a 2-5 page report providing the following information.

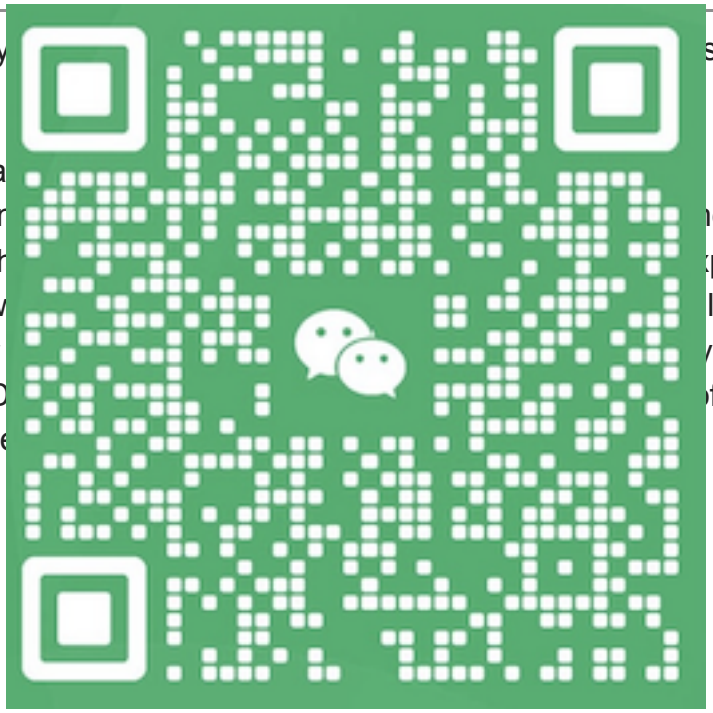
1. *Before you write any code,* read this entire assignment and think about the queries and the different scoring functions. Write a paragraph describing how you expect each scoring function will perform and why. It's OK if you're not sure, and it's OK if you don't get it right. This portion of the report will only be graded based on whether it's there or not and whether it has some meaningful content. The idea is just to start thinking about what's unique about each method and how that might affect their results.

2. Describe how each method actually performed on the queries. Was there a method which was significantly better than the others? Was there a query which was significantly harder than the others? Explain why you got the results you did, and how that compares to what you expected to happen before implementing the rankers. Include a table of the GAP score for each scoring function on each query, and the average GAP score of each scoring function across all queries.
3. (Extra credit) Formulate alternative versions of each query which produce better GAP scores for at least two scoring functions. Submit your alternative topics.xml file, include a table of the improved GAP scores in your report, and explain why you think these queries produce better scores. If they don't improve the GAP scores for all the scoring functions, try to explain why. In order to qualify, your replacement queries must be no more than ten words long. Extra credit will be assigned on a query-by-query basis, but your analysis need not explain all ten queries in detail.

Running Queries

For this assignment, y ssword is required):

- topics.xml conta the queries using the text stored ir ents are only there to clarify th press.
  Note that even v l somewhat ambiguous. For y 230? The tallest? The heaviest? D f what makes information retri



- corpus.qrel contains the relevance grades from expert assessors. While these grades are not necessarily entirely correct (and defining correctness unambiguously is quite difficult), they are fairly reliable and we will treat them as being correct here. The format here is:

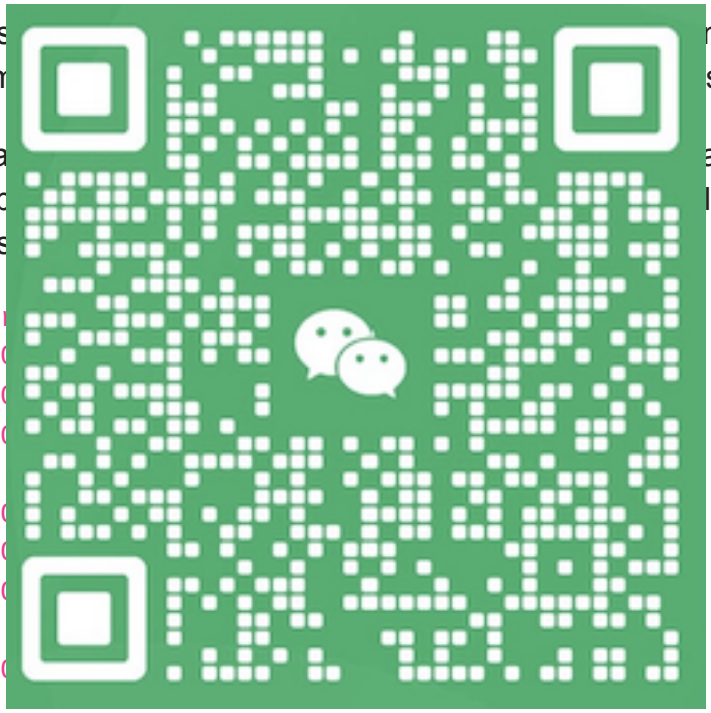
  ```
  <topic> 0 <docid> <grade>
  ```

  - `<topic>` is the ID of the query for which the document was assessed.
  - `0` is part of the format and can be ignored.
  - `<docid>` is the name of one of the documents which you have indexed.
  - `<grade>` is a value in the set {-2, 0, 1, 2, 3, 4}, where a higher value means that the document is more relevant to the query. The value -2 indicates a spam document, and 0 indicates a non-spam document which is completely non-relevant. Most queries do not have any document with a grade of 4, and many queries do not have any document with a grade of 3. This is a consequence of the specific meaning assigned to these grades here and the manner in which the documents were collected.

  This QREL does nt) pair. If an assessment is m s 0 (non-relevant).

You will write a progra a command line argument and which p ll queries found in topics.xml using that s

```
$ ./query.py --sco
202 0 clueweb12-000
202 0 clueweb12-000
202 0 clueweb12-000
...
214 0 clueweb12-000
214 0 clueweb12-000
214 0 clueweb12-000
...
250 0 clueweb12-000
```



The possible values for the `--score` parameter are defined below. The output should have one row for each document which your program ranks for each query it runs. These lines should have the format:

```
<topic> 0 <docid> <rank> <score> <run>
```

- `<topic>` is the ID of the query for which the document was ranked.
- `0` is part of the format and can be ignored.
- `<docid>` is the document identifier.
- `<rank>` is the order in which to present the document to the user. The document with the highest score will be assigned a rank of 1, the second highest a rank of 2, and so on.
- `<score>` is the actual score the document obtained for that query.

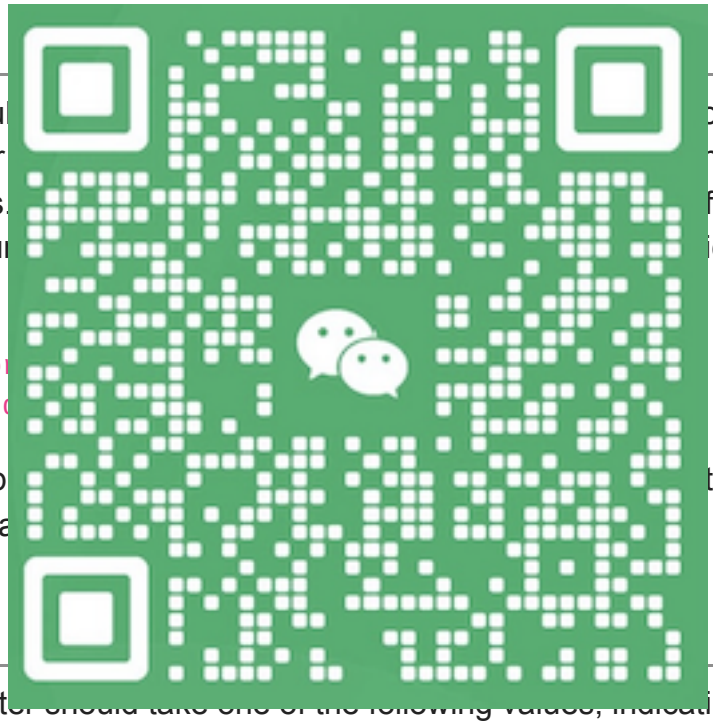
- `<run>` is the name of the run. You can use any value here. It is meant to allow research teams to submit multiple runs for evaluation in competitions such as TREC.

Query Processing

Before running any scoring function, you should process the text of the query in exactly the same way that you processed the text of a document. That is:

1. Split the query into tokens (it is most correct to use the regular expression, but for these queries it suffices to split on whitespace)
2. Convert all tokens to lowercase
3. Apply stop-wording to the query using the same list you used in project 1
4. Apply the same stemming algorithm to the query which you used in your indexer

Evaluation



To evaluate your resul on called Graded Average Precision, or ng the metric is here, if you're curious. for evaluation. In order to evaluate a ru ich I'll call run.txt) and run gap.py on it:

```
$ ./query.py --sco
$ ./gap.py corpus.
```

The program will outp txt, and also the mean GAP score for a

Scoring Functions

The `--score` parameter should take one of the following values, indicating how to assign a score to a document for a query.

Scoring Function 1: Okapi TF

The parameter `--score TF` directs your program to use a vector space model with term frequency scores. We will use a slightly modified form of the basic term frequency scores known as Okapi TF, or Robertson's TF. In a vector space model, a query or a document is (conceptually, not literally) represented as a vector with a term score for each term in the vocabulary. Using term frequency scores, the component of the document vector for term i is: $$ d_i = oktf(d, i) $$ $$ oktf(d, i) = \frac{tf(d, i)}{tf(d, i) + 0.5 + 1.5 \cdot (len(d) / avg(len(d)))} $$ where $tf(d, i)$ is the number of occurrences of term i in document (or query) d, $len(d)$ is the number of terms in document d, and $avg(len(d))$ is the average document length taken over all documents. You should assign a ranking score to

documents by calculating their cosine similarity to the query's vector representation. That is, $$ score(d) = \frac{ \vec{d} \cdot \vec{q} }{ \|\vec{d}\| \cdot \|\vec{q}\| } $$ where \(\|\vec{x}\| = \sqrt{\sum_i x_i^2} \) is the norm of vector \(\vec{x}\).

Scoring Function 2: TF-IDF

The parameter `--score TF-IDF` directs your program to use a vector space model with TF-IDF scores. This should be very similar to the TF score, but use the following scoring function: $$ d_i = oktf(d, i) \cdot \log{ \frac{D}{df(i)} } $$ where \(D\) is the total number of documents, and \(df(i)\) is the number of documents which contain term \(i\).

Scoring Function 3: Okapi BM25

The parameter `--score BM25` directs your program to use BM25 scores. This should use the following scoring function for document \(d\) and query \(q\): $$ score(d, q) = \sum_{i \in q} \left[\log \left(\frac{ D + 0.5 }{ df(i) + 0.5 } \right) \cdot \frac{(1 + k_1) \cdot tf(d, i)}{ K + tf(d, i) } \cdot \frac{(1 + k_2) \cdot tf(q, i)}{ k_2 + tf(q, i) } \right] $$ $$ K = k_1 \cdot \left((1 - b) + b \cdot \frac{len(d)}{avg(len(d))} \right) $$ Where \(k_1, k_2, \) and \(b\) are constants. For starters on BM25. Feel free to experiment wit effect and try to improve performance.

Scoring Function 4:

The parameter `--sco` ge model with Laplace smoothing (a ve will use maximum likelihood e lel "trained" on the document. In this setu appear in the query. Here is the sco _d(i) $$ $$ p_d(i) = \frac{tf(d, i) + 1}{len(number of unique terms in the corpus. T guage model which generated the c) and \(+V\) smooth the probability ms which did not appear in the docume

Scoring Function 5: Language model with Jelinek-Mercer Smoothing

The parameter `--score JM` directs your program to use a language model with Jelinek-Mercer smoothing. This uses the same scoring function and the same language model, but a slightly different smoothing function. In this case we use $$ p_d(i) = \lambda \frac{tf(d, i)}{len(d)} + (1-\lambda) \frac{\sum_d tf(d, i)}{\sum_d len(d)} $$ The parameter \(\lambda \in [0, 1] \) allows us to mix the probability from a model trained on document \(d\) with the probability from a model trained on all documents. You can use \(\lambda=0.2\). You are encouraged, but not required, to play with the value of \(\lambda\) to see its effect. (What do you expect would happen with \(\lambda=0\) or \(\lambda=1\)?)

Note that you can calculate the second term using "the total number of occurrences of the term in the entire corpus" (the cumulative term frequency, or ctf) which you have stored in term_info.txt.

Submission Checklist

Submit your files in a folder named `pr2`.

- Your report
- Your source code
- The output ranking for each scoring function (zipped or gzipped)
- Your alternative topics.xml, if you attempted the extra credit

Rubric

- Scoring function implementation: 50 Points
  10 points:
  You correctly im

  10 points:
  You correctly im

  10 points:
  You correctly im

  10 points:
  You correctly im ning

  10 points:
  You correctly im smoothing

- Report: 50 Point
  10 points:
  You wrote your e

  10 points:
  You included the appropriate data tables

  20 points:
  You described your results adequately (4 points for each scoring function)

  10 points:
  Your analysis shows a good understanding of the scoring functions

  +10 points:
  You provided better-performing queries (+1 for each query)