**Objective**

* Implement ranking functions to rank documents against queries.
* You are given a training set: pairs of documents – query / relevance value
* Implement two different ranking functions + use NDCG metric for evaluating the effectiveness of the ranking function. Estimation of parameters (e.g. weights) will be done manual (no machine learning until next homework).

**Data description**

* Two folders with data: do the training using “Training” folder and test against “Test”.
* Two files inside each:
  + queries – gives pairs of queries and 10 sources found by a popular search engine. For each source, we also give several features that you can use for training.
  + relevance – for each query and document in “queries” gives you the rating of each document.

**Example from queries:**

query: dining hall food bad

url: http://news.stanford.edu/news/multi/features/food/eating.html

title: multidisciplinary teaching and research at stanford

body\_hits: food 79 110 204 220 233 284 351 420 510 522 560 575 610 639 717 736 758 803 892 906 988 1002 1063 1103 1151 1277 1335 1419

body\_hits: dining 99 288 378 401 875 913 1134 1385

body\_hits: bad 159 222 227 1060

body\_hits: hall 379

body\_length: 1464

pagerank: 3

url: http://tusb.stanford.edu/

title: the unofficial stanford blog

header: dinner hall

header: bad food in kitchen

body\_hits: bad 966 1029 2598 5905 6754

body\_hits: food 2148 3768 6576 6813

body\_hits: dining 3766

body\_hits: hall 3767

body\_length: 6912

pagerank: 4

anchor\_text: the unofficial stanford blog food

stanford\_anchor\_count: 1

**Example from relevance**

query: dining hall food bad

url: http://news.stanford.edu/news/multi/features/food/eating.html 1.0

url: http://tusb.stanford.edu/ 1.0

url: http://tusb.stanford.edu/2011/02/the-cool-cafe-beyond-the-gates-of-hell.html 3.0

url: http://tusb.stanford.edu/2011/04/disgust-its-whats-for-dinner.html 3.0

url: http://web.stanford.edu/group/ccr/blog/2009/10/what\_we\_really\_eat\_1.html 2.0

url: http://web.stanford.edu/group/religiouslife/cgi-bin/wordpress/wp-content/uploads/sermons/2010/sermon\_3-21-2009\_Neumann.doc 1.0

url: https://rde.stanford.edu/studenthousing/rf-letter-frosoco 0.0

url: https://tusb.stanford.edu/?cat=37 1.0

* Features description for queries:
* body\_hits – each line is a query term and specifies the positions in the document where that term was found
* title – title of the page
* pagerank – the quality metric for a page, independent of any query. Higher is better.
* body\_length – how many terms are in this document (not necessarily different)
* header: these are the H1,H2,H3,H4 headers in html tags. Each line is a header where one or more query term appears in.
* Ignore anchor\_text for now.
* Features description for relevance:
  + For each query you have the resources and a relevance rating between -1 and 3. Higher is better.
* If you need other statistics that are not present in the folder, you can use the data in Lab1.

**Output requirement:**

* Provide a program ( python script, c++ , java, whatever you like) that I can run with a command like like following: program TASK\_TYPE queriesFile relevanceFile. The output is the NDCG evaluation metric. Train first on training folder then test on the Test folder. Do not attempt to optimize for Test folder instead, the average of the two scores will be used to evaluate your homework.
* The TASK\_TYPE is one of the methods defined below in the TASK section
* Discussion about the utility of your homework

**Ranking**

* Build a term score ***tf*** for each field of a document: title, body, headers.

E.g. for above query **“**dining hall food bad” the raw term scores (rs):

Body tf: [8 1 28 4]. – counts how many times does each term appear in body

Title tf: [0 0 0 0]

Header tf: [ 1 1 1 1] – counts how many times does each term appear in all headers set.

* Give different weights for title, body and headers as defined in Chapter 6.
* For tokenization, consider only lower case terms.

**Tasks**

**1. Cosine similarity ranking (chapter 6 in the book).**

***How to create document vector***

1. Term frequency (tf)

Use either raw scores or apply sublinear scaling using tf = 1 + log(rs) if rs > 0, 0 otherwise. The body tf will be: [0.9 0 1.1447 0.60]. See chapter 6 for more information, formula 6.13

1. Document frequency.

Don’t use idf for this, it will be incorporated in Normalization (see below).

1. Normalization

Since in the currenct ***context*** you don’t have access to the entire corpus data, we don’t know the frequency of all other terms (i.e. other than ones in the queries).

To solve both B (idf in the formula) and normalization, only normalize the tf factor of each term in the document against length of the document (L = body\_length + title length + header length).

E.g. in formula 6.8 in Chapter 6, replace idf of a term with 1/ L.

If the score term t for the body of document d:

The full array for body will be denoted then by: , where each component is one of the terms in vocabulary.

Similar notations for title and header score vectors: : , .

***How to create query vector***

1. Term frequency (tf)

Same strategy as in document vector. Raw score is 1 for each term in the query.

1. Document frequency

Parse the data in Lab1, and cache a file on disk with df for each term. You can then easily compute idf. The N in the formula is the total number of files parsed.

1. Normalization

No other normalization is needed here:

If q is the query, then is the query score for term t.

The entire array containing all terms in the vocabulary is: .

***Final scoring***

The final formula scoring a document against a query is :

+ , where , are the weights of title, headers and body.

Your task is to find optimal values for weights, such that NDCG function is maximum.

***Efficient implementation***

Space and time complexity of your implementation is also very important for grading. Suggestions:

* Vocabulary can be very high (order of thousands of words)
* Number of documents can be millions.
* Don’t attempt to represent vectors as a plain array of dimension equal to vocabulary size
* Don’t attempt to compute cosine similarity in O(|Vocabulary size|), that is not a valid implementation !. Check Figure 6.14 in the book and try to adapt the pseuodocode there. Basically, the final pseudocode / structure should look like below:

1. Offline / preprocess step:

* Create a posting list for each term in any query present in training docs.
* The posting list for query term T should contain pairs of {(docId, raw term score, normalized score}). Basically, you should cache the scores done in the formulas above for each term and document.

1. Runtime step (new queries testing):

Compute the qv array for each term T in query. => qv has size # of words in the query.

For each term T in query:

PT = posting list of T

For each pair (docID, raw score, normalized\_score) in PT:

Scores[docId] += qv(T) \* normalized\_score

Return Top K components of Scores

1. Details

* Scores must a map that is dynamically created. If a document is new while iterating over lists, add the item, otherwise just add the new score.
* You have to run the two steps above for each zone: body, title, headers. Keep
* Top K must be computed with a Heap (see text below figure 6.14).

**2. BM25F**

For this task, you must implement the BM25F algorithm for ranking.

***Term score per document and zone – must be cached from training set (i.e. not computed at runtime).***

We’ll use field -dependent normalized term frequency, for each field {body, title, header}:

Where:

* raw\_score(d, t, f) = # of times term t appears in zone f of document d.
* = length of field f in document d (how many words)
* is a normalization parameter that must be tuned by you (for each of the three zones).

Then, the overall weight of a term t in document d among all fields is:

, where f iterates over the three fields and is a parameter that must be tuned by you for all fields. (you can also reuse the same weights you given to fields in Task 1).

***Final formula***

As you observed, we have a non-textual feature – pagerank. We incorporate it in rating using one of the : log / saturation / sigmoid functions (see BM25 slides in the Course folder).

*Score(d,q) =*

*Where:*

* *Idf is the inverse document frequency, can be computed as in Task 1 for query using the dataset in Lab 1.*
* *K1 is a parameter that you must tune.*
* *V are parameters that you must tune*

*Observation: Precompute as much as you can !*

*No need for disk I/O in these tasks, only RAM memory.*