Information Retrieval: Assignment 3

Sawood Alam
salam@cs.odu.edu

December 11, 2011

Abstract

This document contains solutions and discussions on exercise questions 8.3, 8.4, 8.5 and 8.7 from the textbook, Search Engines Information Retrieval in Practice - W. Bruce Croft, Donald Metzler and Trevor Strohman.

1 Problem 8.3

For one query in CACM collection (provided at the book website), generate a ranking using Galago, and then calculate average precision, NDCG at 5 and 10, precision at 10, and the reciprocal rank by hand.

1.1 Solution

I have selected query 6 from the CACM query set that reads as follows.

> interested in articles on robotics motion planning particularly the geometric and combinatorial aspects we are not interested in the dynamics of arm motion

After running Galago batch query, I got the following relevant segment in the resultant output file.

```
------------------------------------------------------------
$ ./bin/galago batch-search -i cacm.index -c 10 cacm.query.xml
6 Q0 CACM-0695 1 -164.66490173 galago
6 Q0 CACM-2826 2 -166.82664490 galago
6 Q0 CACM-2828 3 -167.13014221 galago
6 Q0 CACM-1664 4 -167.29315186 galago
6 Q0 CACM-1543 5 -167.67100525 galago
6 Q0 CACM-2078 6 -168.12095642 galago
6 Q0 CACM-2176 7 -168.47441101 galago
6 Q0 CACM-1113 8 -169.13491821 galago
6 Q0 CACM-0605 9 -169.25828552 galago
6 Q0 CACM-1517 10 -169.68901062 galago
------------------------------------------------------------
```

Corresponding section of relevance judgement file looks like following.

```
------------------------------------------------------------
$ grep "^6 " cacm.rel
6 Q0 CACM-1543 1
6 Q0 CACM-2078 1
6 Q0 CACM-2828 1
------------------------------------------------------------
```

Running the result set against provided relevance judgement file gave me following statistical evaluations.

```
------------------------------------------------------------
$ ./bin/galago eval cacm.res cacm.rel
num_ret           6 10
num_rel           6 3
num_rel_ret       6 3
map               6 0.4111
ndcg              6 0.5833
ndcg15            6 0.5833
R-prec            6 0.3333
bpref             6 0.0000
recip_rank        6 0.3333
P5                6 0.4000
P10               6 0.3000
P15               6 0.2000
P20               6 0.1500
P30               6 0.1000
P100              6 0.0300
P200              6 0.0150
P500              6 0.0060
P1000             6 0.0030
------------------------------------------------------------
```

By manual comparison between result set and relevance judgement file, I found the relevant documents at 3rd, 5th and 6th places in the result set.

1.1.1 Average Precision

Having all above information, average precision can be calculated as following.

$$(1/3 + 2/5 + 3/6)/3 = 0.41$$

This result can be verified by the statistics provided by the eval utility of Galago.

1.1.2 NDCG

To calculate NDCG, I used binary ranking 1 for relevant documents and 0 for non-relevant documents.

To help calculating NDCG, I have written small Ruby script that can accep an array of resultant ranks and returns the NDCG values for each place in the result set. Script looks like the following.

```
---------------------------------------------------------------
def dcg(arr)
  r = []
  arr.each_with_index do |e, i|
    i.zero? ? r.push(e) : r.push(e/Math.log(i+1, 2) + r[i-1])
  end
  r
end

def ndcg(arr)
  r = []
  arr_dcg = dcg(arr)
  arr_ideal_dcg = dcg(arr.sort.reverse)
  arr_ideal_dcg.each_with_index do |d, i|
    d.zero? ? r.push(0) : r.push(arr_dcg[i] / d)
  end
  r
end
---------------------------------------------------------------
```

Running above code for this result set gave the following NDCG values for all the 10 places. (Result was manually rounded to 2 decimal places.)

```
---------------------------------------------------------------
ndcg([0, 0, 1, 0, 1, 1, 0, 0, 0, 0])
=> [0.0, 0.0, 0.24, 0.24, 0.40, 0.55, 0.55, 0.55, 0.55, 0.55]
---------------------------------------------------------------
```

Hence, we can easily find desired NDCG values from the above results.

$$NDCG(5) = 0.4$$
$$NDCG(10) = 0.55$$

1.1.3 Precision at 10

Since all the 3 relevant documents were retrieved in top 10 result set. Hence, precision at 10 can simply calculated as following.

$$3/10 = 0.3$$

This value may also be verified from the statistics provided by Galago eval utility.

1.1.4 Reciprocal Rank

Since, the first relevant document appeared at rank 3, hence the Reciprocal Rank can be calculated as following.

$$1/3 = 0.33$$

This result can also be verified by the statistics provided by the eval utility of Galago.

1.2 Discussion

Biggest challenge I faced while solving this question was to identify a good query set that can illustrate varying possibilities still being complete and simple. Rest of the work was simple but little bit time consuming.

2 Problem 8.4

For two queries in the CACM collection, generate two uninterpolated recall-precision graphs, a table of interpolated precision values at standard recall levels, and the average interpolated recall-precision graph.

2.1 Solution

For this question, I am selecting query 6 that I selected for the previous question as well. And the other query is query 2 that says,

> i am interested in articles written either by prieve or udo pooch prieve b pooch u

Statistics given by the eval utility of Galago for query 2 is as follows.

```
------------------------------------------------------------
num_ret           2 10
num_rel           2 3
num_rel_ret       2 3
map               2 1.0000
ndcg              2 1.0000
ndcg15            2 1.0000
R-prec            2 1.0000
bpref             2 0.0000
recip_rank        2 1.0000
P5                2 0.6000
P10               2 0.3000
P15               2 0.2000
P20               2 0.1500
P30               2 0.1000
P100              2 0.0300
P200              2 0.0150
P500              2 0.0060
P1000             2 0.0030
```

For this second query all the three relevant documents appear at positions 1, 2 and 3.

Table 1 and Table 2 are showing the precision and recall values for the top ten positions for the two queries respectively.

Table 3 shows individual and average interpolated precision values at standard recall levels for the two queries.

Table 1: Recall Precision for Query 6

Position	1	2	3	4	5	6	7	8	9	10
Recall	0.0	0.0	0.33	0.33	0.67	1.0	1.0	1.0	1.0	1.0
Precision	0.0	0.0	0.33	0.25	0.4	0.5	0.43	0.38	0.33	0.3

Table 2: Recall Precision for Query 2

Position	1	2	3	4	5	6	7	8	9	10
Recall	0.33	0.67	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
Precision	1.0	1.0	1.0	0.75	0.6	0.5	0.43	0.38	0.33	0.3

Table 3: Interpolated Precision Values at Standard Recall Levels

Recall	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Query 6	0.33	0.33	0.33	0.33	0.4	0.4	0.4	0.5	0.5	0.5	0.5
Query 2	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
Average	0.67	0.67	0.67	0.67	0.7	0.7	0.7	0.75	0.75	0.75	0.75

2.2 Discussion

The set of queries I have chosen has some interesting properties that are reflected in the plots.

Usually Precision and Recall are inversely related that means one decreases with the increase of the other. But In these examples it is showing counter-intuitive nature. I was initially suspected my calculations but after careful inspection, I realised that the specific nature of the queries is causing this behaviour. One of the queries got the best placement with all the relevant document in the very beginning hence, resulted in precision value of one at every recall point. It also causes an overall uplift for the average interpolated values.

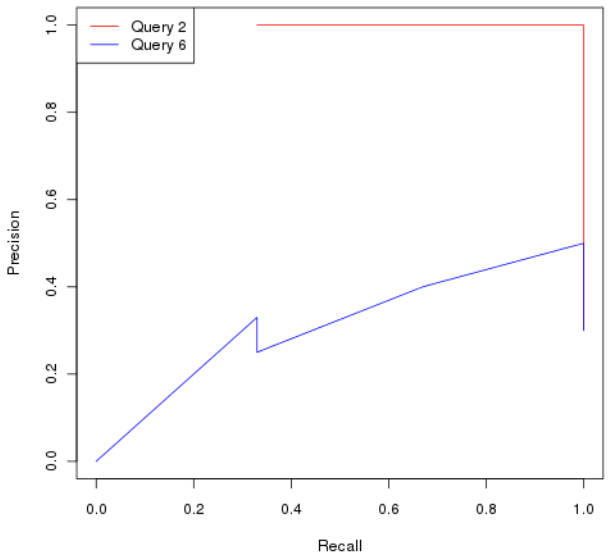

Figure 1: Uninterpolated Recall-Precision for Two Queries

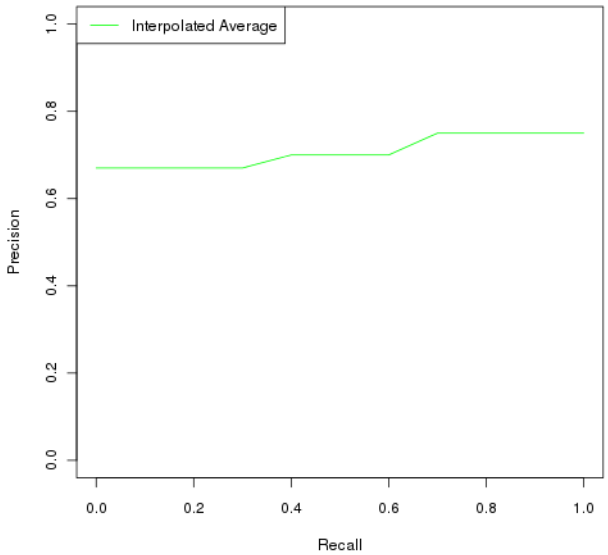

Figure 2: Interpolated Average Recall-Precision for Two Queries

3 Problem 8.5

Generate the mean average precision, recall-precision graph, average NDCG at 5 and 10, and precision at 10 for the entire CACM query set.

3.1 Solution

Figure 3 shows a comparison of various statistical measures over the set of 64 queries of the CACM corpus.

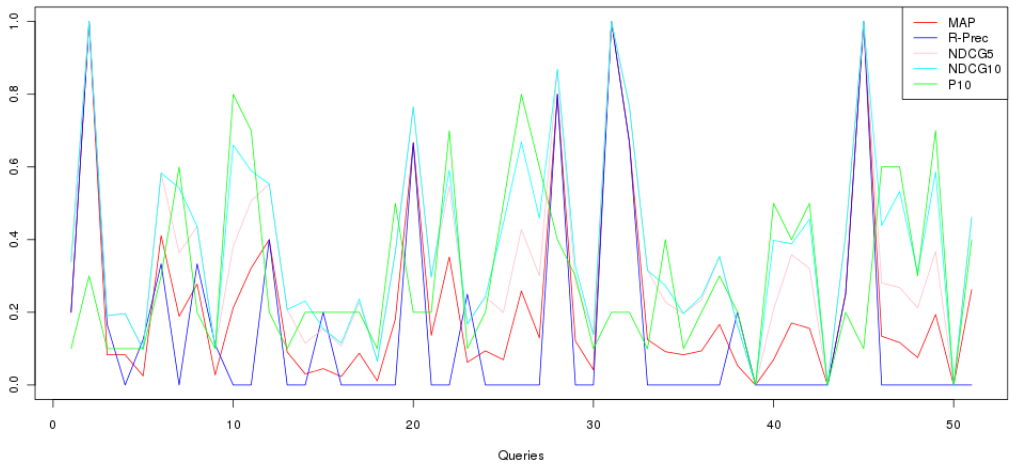

Figure 3: Comparision of Various Statistical Measures

3.2 Discussion

Looking at the combined comparison graph, it is clear that for intermediate values, most of the measures are showing similar patterns. But, some measures penalise or reward aggressively. Hence, Their values usually exist on the two extremes. Mean Average Precision and NDCG are good examples of such measures. While, Precision at 10 remains in the middle region for most of the queries.

R-Precision was not part of this question. Also, It is not calculated accurately because, number of results is limited to 10 for each query, while there are more than 10 relevant documents in the relevant judgement file for several queries. I have added it here to get a rough idea of how this measure co-relates to the others.

Also, In the Galago eval utility, it did not report NDCG at 10, rather it reports NDCG at 5 and NDCG 15. But I cleverly used the NDCG at 15 value in place of NDCG at 10. The justification of the validity of this use is simple as I had already limited the number of results per query, hence, there will be no relevant documents after 10 while calculating the NDCG value at 15. This will cause the NDCG at 10 value propagate till NDCG at 15 without any change.

4 Problem 8.7

Another measure that has been used in number of evaluations is R-precision. This is defined as the precision at R documents, where R is the number of relevant documents for a query. It is used in situations where there is a large variation in the number of relevant documents per query. Calculate the average R-precision for the CACM query set and compare it to the other measures.

4.1 Solution

Looking at the relevance judgement file, following is the list of number of relevant documents for each query in sorted order of the query number.

[5, 3, 6, 12, 8, 3, 28, 3, 9, 35, 19, 5, 11, 44, 10, 17, 16, 11, 11, 3, 11, 17, 4, 13, 51, 30, 29, 5, 19, 4, 2, 3, 1, 0, 0, 20, 12, 16, 12, 10, 0, 21, 41, 17, 26, 0, 0, 12, 8, 0, 0, 0, 0, 0, 0, 0, 1, 30, 43, 27, 31, 8, 12, 1]

There is just one query that has 51 relevant documents in the corpus. rest of the queries have smaller number of relevant documents. Hence, I decided to regenerate results for 50 results per query. Once batch query was processed, I evaluated it against the relevance judgement file. As a result, average evaluation measures are shown in Table 4.

Table 4: Comparison of R-precision with Various Relevance Measures

Relevance Measure	Average Value
R-precision	0.3119
MAP	0.2826
NDCG 5	0.4701
NDCG 15	0.4419
Reciprocal Rank	0.7296
Precision 5	0.3922
Precision 10	0.2980
Precision 15	0.2614
Precision 20	0.2373
Precision 30	0.1843

4.2 Discussion

Looking at the R-precision graph (blue color) in the Figure 3 generated for the previous question, there were several places where R-precision was showing Zero while other measures were not. I investigated the cause of this behaviour and realized thee there were two types of such points. One was for the queries that have no relevant results and the other type of points were the queries that have no relevant documents in top 10 results. Because the result set for the previous question was limited to 10 results per query this was causing the inaccurate calculation. While in the new result set, this issue did not exist. I have manually investigated those queries in my result set. Related file named "cacm50.res" is included in the submitted assignment archive.

I had this information in advance that there is a query that has as many relevant results as 51. But, I decided not to increase the result set limit to 51 or any higher number, rather I decided to keep it 50. There is a very little chance that the 51st result (if included) will be a relevant document to that specific query. And if it is so, the effect of one position will not change the result significantly. The effect (if there is any) will be further vanished when the average over entire query set will be calculated.

5 Problem 6.7

Implement a simple algorithm that selects phrases from the top-ranked pages as the basis for the result clusters. Phrases should be considered as any two-word sequence. Your algorithm should take into account phrases frequency in the results, phrase frequency in the collection, and overlap in the clusters associated with the phrases.

5.1 Solution

I have created a simple vector space model using two-word sequence phrases as vectors. The value or weight for each vector is the phrase frequency in the document multiplied by inverse document phrase frequency. It is derived from the idea of TF*IDF. A matrix holds these values as phrases in column and documents in rows.

```
---------------------------------------------------------------
# Matrix storing phrase-frequency * inverse-document-frequency
document_phrase_vector = [documents * phrases]

procedure search(query)
  results = response_from_search_engine(query)
  top_ranked_pages = results.slice(0, N = 10)
  result_phrases = phrase_extractor(top_ranked_pages)
  results_vector = document_phrase_vector.filter_rows(results)
  results_vector = results_vector.filter_cols(result_phrases)
  K = top_result_phrase_count(results_vector)
  clustered_results = k_means_cluster(results, K, overlap = F)
  display(clustered_results)
end
---------------------------------------------------------------
```

A sub-routine that helps extracting pairs of adjacent words and phrases in the top-ranked pages (or any set of pages that are passed as parameter).

It is assumed that we already have some routine to clean the html pages and transform them in a plain text file with space separated words without any HTML tags, new-lines, punctuation marks or other symbols.

```
-------------------------------------------------------------
procedure phrase_extractor(top_ranked_pages)
  page = top_ranked_pages.concat
  page = page.clean // make simple space separated text file of words
  words = page.split(' ')
  temp = []
  i = 1
  while(i < words.length)
    temp.push(words[i-1] + ' ' + words[i])
  end
  return temp
end
-------------------------------------------------------------
```

This is a sub-routine that discovers top-ranked phrases in the result set based on some threshold. It then returns the number of such phrases that can be used to estimate the number of clusters in the clustering algorithm.

```
-------------------------------------------------------------
procedure top_result_phrase_count(results_vector)
  phrase_weights = column_sum(results_vector)
  phrase_weights = phrase_weights.sort(DESC)
  phrase_weights = phrase_weights.filter(value > threshold)
  return phrase_weights.count
end
-------------------------------------------------------------
```

5.1.1 Algorithm Explanation

When a query is being fired, results are extracted from underlying search engine. Assuming that results are sorted based on some ranking parameters by the underlying search engine itself. Top N (for example 10) pages from the result set are used for clustering the whole result set. A "phrase_extractor" routine extracts all two-word phrases from the N top ranked result pages. A sub-matrix is then created from global document phrase vector matrix by filtering only result pages and phrases in the top ranked result pages. Clustering is then performed on this sub matrix using K-means clustering algorithm without overlaps in the clusters. Number of clusters is not the

same as distinct phrases in the result set. That number is identified in another procedure called "top_result_phrase_count".

The "top_result_phrase_count" procedure is simple. It sums up the values of each phrase in the results vector (column-wise sum). This sum gives an idea about the aggregated importance of certain phrases in the result set. Phrases are then sorted in descending order of aggregated values and counted up to a minimum threshold on values.

5.2 Discussion

One question arises, whether the top phrases (top K phrases identified in the "top_result_phrase_count" procedure) will fall in distinct clusters? At this point, I have no proof that the final clusters in the result set will have all those top-ranked phrases in distinct clusters. Intuitively, it is less likely to happen, as those top ranked phrases might cluster together. But, this is something to be explored and examined further. But, "top_result_phrase_count" procedure gives a good measure to estimate, how many clusters should be there (dynamically depending on the result set) in the final result?

6 Problem 7.2

Can you think of another measure of similarity that could be used in the vector space model? Compare your measure with the cosine correlation using some example documents and queries with made-up weights. Browse the IR literature on the Web and see whether your measure has been studied (start with van Rijsbergen's book).

6.1 Solution

Another similarity measure that I can think of is ratio between the mod of difference of the two vectors and mod of their sum. It is actually the distance measure. Hence, more the distance, less is the similarity.

$$D_{d,q} = \frac{|\vec{d} - \vec{q}|}{|\vec{d} + \vec{q}|}$$

Where d and q are the two vectors in the vector space (typically document and query vectors but, these can be any two documents as well.)

These vectors usually have positive coefficients in case of document vectors. But, negative queries can have negative coefficients as well. In normal cases this measure can work well but it may fail or stretch the measure in some cases.

6.1.1 Example

Consider a small document "D" and a query "Q" as follows.

```
------------------------------------------------------------
    D = (book pen book ink book)
    Q = (book pen)
------------------------------------------------------------
```

Now I will compute the cosine correlation as well as my distance measure for this document and query to illustrate the working. For the sake of simplicity, I am not putting the vector bar in the formulas.

$$d = 3 * book + 1 * pen + 1 * ink$$
$$q = 1 * book + 1 * pen$$

In this example, book, pen and ink are the only words in the vector space. These words are representing independent unit vectors (assuming that these words are independent in the language model) making a right angle between every couple of vectors.

By doing simple vector subtraction and addition, we get the following vectors.

$$d - q = 2 * book + 0 * pen + 1 * ink$$
$$d + q = 4 * book + 2 * pen + 1 * ink$$

Coefficients of the two vectors, their difference and their sum can be calculated easily using standard vector equation.

$$|d| = \sqrt{(3^2 + 1^2 + 1^2)} = 3.3167$$
$$|q| = \sqrt{(1^2 + 1^2)} = 1.4142$$

$$|d - q| = \sqrt{(2^2 + 0^2 + 1^2)} = 2.2361$$
$$|d + q| = \sqrt{(4^2 + 2^2 + 1^2)} = 4.5826$$

Now that we have intermediate steps and their numeric values, we can proceed to calculate the measures.

$$Cosine\, similarity = \frac{(1 * 3) + (1 * 1)}{(3.3167 * 1.4142)} = 0.8528$$
$$My\, distance\, measure = \frac{2.2361}{4.5826} = 0.4879$$

I can repeat this process for other vectors to illustrate more examples. But, I believe, It is a very simple distance measure and one example is sufficient to illustrate the working.

6.2 Discussion

To study the behaviour of this distance measure, consider one dimensional vector (for the sake of simplicity) for the following cases.

6.2.1 CASE 1: Short and Large Coefficients

On one dimensional vector space, suppose coefficients of d and q are 2 and 1 respectively. The distance measure will produce a value of 1/3. Now, change the coefficients to 100 and 99. In the space, the two are just one unit away as in previous case but the distance measure will be 1/199. Although, it has stretched/squeezed the linear distance, intuitively, this is the desired result. Because, large coefficients mean that there are several occurrences of the property represented by that dimension. This means, two documents having a term 100 times and 99 times respectively are more similar (less distant) than the two documents having a term twice and once respectively.

6.2.2 CASE 2: Coefficients with Opposite Signs

If the coefficient of d is positive, the coefficient of q is negative and its modulus is less than that of d, it will work fine. But, when the two coefficients are equal and opposite, the distance measure will be infinite (extremely non-relevant). Beyond that point, distance measure will start giving continuously decreasing finite values that is not valid (and counter-intuitive). This behaviour can be observed in Figure 4.

Later, I was exploring the Web to see if this distance measure was studied earlier or not, I found that an improved version (namely *"Bray-Curtis"*) of this measure has already been studied.[**?**][**?**]

They have an enhancement in the distance measure by not taking the mod of denominator (d + q). This eliminates the problem due to opposite coefficients described above. Because, it results in negative distance measures after critical point that can be safely considered as non-relevant. This behaviour can be observed in Figure 5.

Rscript used to generate Figure 4 and Figure 5 plots is follows.

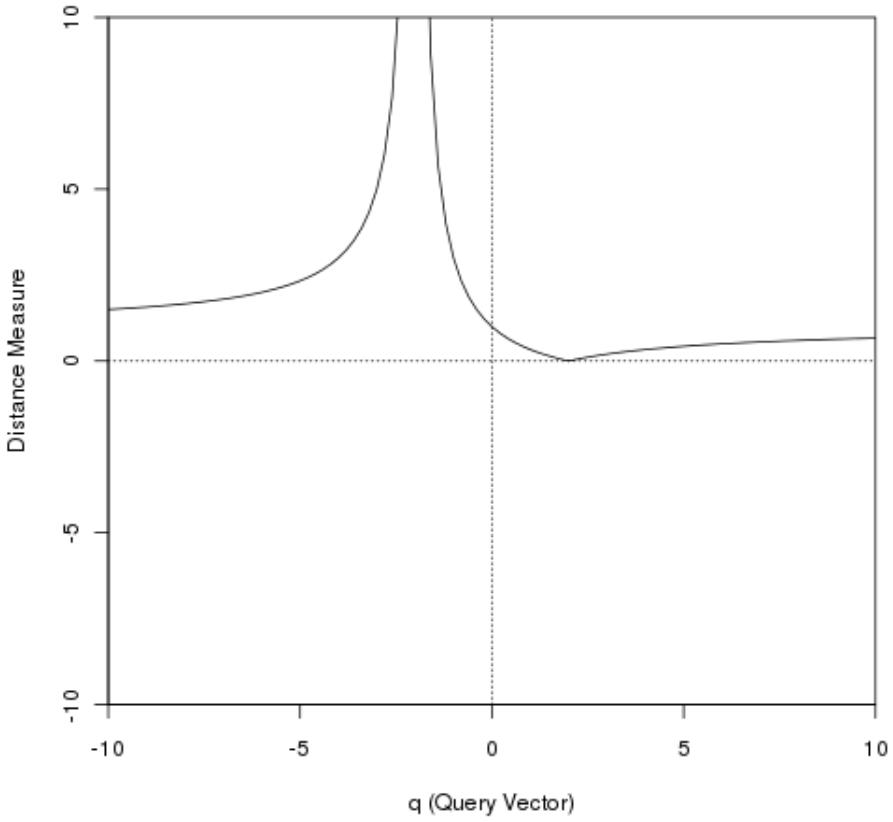

Figure 4: My distance measure for d=2

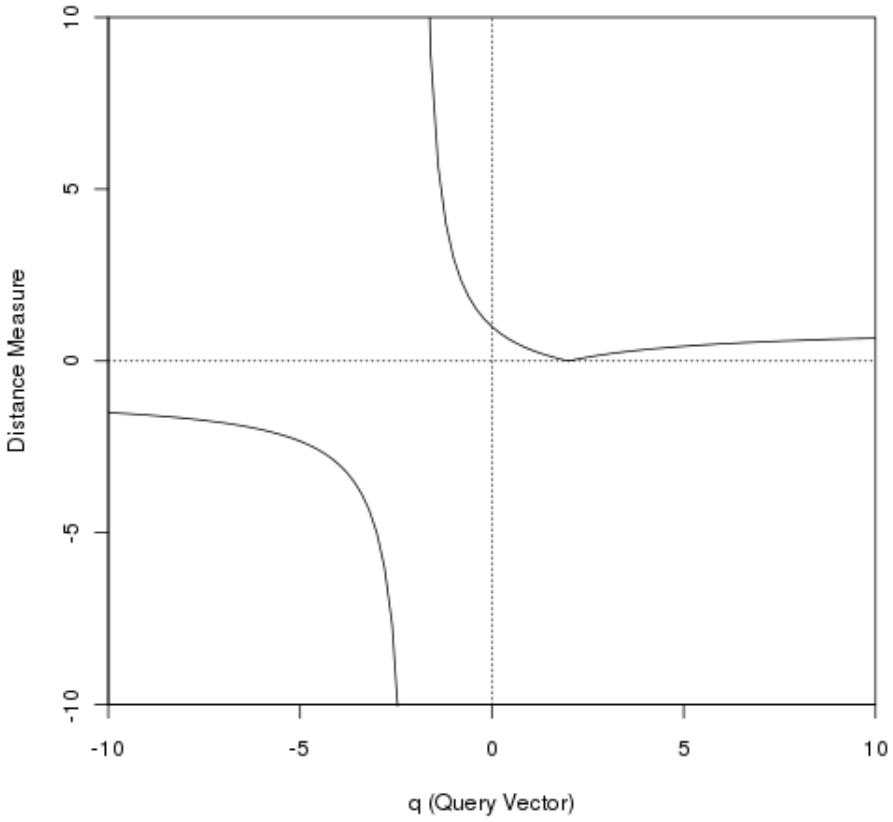

Figure 5: Bray-Curtis distance measure for d=2

```
-----------------------------------------------------------------
fn <- function(x) abs(2-x)/abs(2+x)
fn1 <- function(x) abs(2-x)/(2+x)

png("distmm.png", width=500, height=500)
plot(fn, -10, 10, type="l", xlab="q (Query Vector)",
     ylab="Distance Measure", xaxs="i", yaxs="i",
     ylim=c(-10, 10))
lines(c(-10, 10), c(0, 0), lty=3)
lines(c(0, 0), c(-10, 10), lty=3)
dev.off()

png("distbc.png", width=500, height=500)
plot(fn1, -10, 10, type="l", xlab="q (Query Vector)",
     ylab="Distance Measure", xaxs="i", yaxs="i",
     ylim=c(-10, 10))
lines(c(-10, 10), c(0, 0), lty=3)
lines(c(0, 0), c(-10, 10), lty=3)
dev.off()
-----------------------------------------------------------------
```

7 Problem 7.5

Implement a BM25 module for Galago. Show that it works and document it.

7.1 Solution

I tried writing the BM25 module in Java. But, it was not a successful effort (till the point of compilation and execution). Here is my code.

```
-------------------------------------------------------------
/*
 * This is a simple BM25 score calculator that is not
 * in running condition yet.
 * It has some parameters pre-defined and hard-coded in it.
 */
public class BM25 {
  /* Constructor method to initiate the parameters */
  public BM25() {
    b = 0.75;
    k = 1.2;
    cl = 10000.0; // Collection length
    dc = 100.0; // Document count
    df = 50.0; // Document frequency
    // IDF calculation
    idf = Math.log((dc - df + 0.5) / (df + 0.5));
  }

  /* Publically exposed method to calculate the score */
  public double calculate(int c, int l) {
    // Actual BM25 formula
    result = idf * (c * (k + 1))
                  / (c + (k * (1 - b + (b * l / (cl / dc)))));
    return result;
  }
}
-------------------------------------------------------------
```

7.1.1 Code Available in Galago Source

The code for BM25 is available in the latest version of the Galago code repository. But, it is not compiled and not included in the JARs of binary package. Hence, when I tried to run a test query against "BM25" scoring module, it was giving me errors.

Here is how the code looks like.

```
-----------------------------------------------------------------
@RequiredStatistics(statistics = {"collectionLength",
                                  "documentCount"})
public class BM25Scorer implements ScoringFunction {
  double b;
  double k;
  double avgDocLength;
  double idf;

  public BM25Scorer(Parameters parameters,
                    CountValueIterator iterator)
                    throws IOException {
    b = parameters.get("b", 0.75D);
    k = parameters.get("k", 1.2D);

    long collectionLength = parameters.get("collectionLength",
                                           0L);
    long documentCount = parameters.get("documentCount", 0L);
    avgDocLength = (collectionLength + 0.0) / (documentCount
                                               + 0.0);

    long df = 0;
    if (parameters.containsKey("df")) {
      df = parameters.get("df", 0L);
    } else {
      df = iterator.totalEntries();
    }
    idf = Math.log((documentCount - df + 0.5) / (df + 0.5));
  }
```

```
  public double score(int count, int length) {
    double numerator = count * (k + 1);
    double denominator = count + (k * (1 - b + (b
                                    * length / avgDocLength)));
    return idf * numerator / denominator;
  }

  public String getParameterString(){
    return "bm25.b=" + b + ",bm25.k="+k;
  }
}
```

A test query against this module.

```
<parameters>
  <query>
    <number>book</number>
    <text>#combine(#feature:bm25(
                  #extents:test:part=stemmedPostings()))
    </text>
</query>
</parameters>
```

And the error I encountered.

```
Couldn't find a class for the feature named bm25.
```

7.2 Discussion

I am not comfortable with Java coding and specially packaging. Also, I am not very familiar with the Galago code-base (although, I tried my best to understand it).Hence, it was hard for me to build a successful module in a very modular fashion, along with proper parameters and exception handling

etc. Hence, I tried to assume that certain things will be available to me when I import some packages in my code. Then I did the coding mainly for the metamathematical portion of the algorithm. Later I discovered that there is an untested code in the code-base of Galago for BM25. I realized that the portion I did was almost correct as compared to the code available in the repository. Hence, I included my code along with the code available in the repository. I did some in-line documentation of my code as well. I did not make any further changes in my code after getting the code in the repository to avoid any honour code violation. For the sake of comparison, I have included both of them.

Probably, I would have not attempted this question but, there were not many choices left. This entire assignment was very challenging and long. I spend several days and couple off sleepless nights doing this assignment (in last two weeks) but, I am happy that I got some insight of Galago search engine (because there was no way to avoid that).

Often times I thought, I wont be able to do this assignment at all except couple of questions. But, I studied related material again and again and now I am satisfied with what I have done so far.

8 Problem 7.8

Using the Galago implementation of query likelihood, study the impact of short queries and long queries on effectiveness. Do the parameter settings make a difference?

8.1 Solution

Galago uses *"Dirichlet"* algorithm that is a derived form of pure *"query likelihood"* algorithm. Biggest drawback of pure *"query likelihood"* algorithm is its failure to handle query term(s) is the query phrase with zero relevance. Since, it uses multiplicative aggregation that makes aggregated similarity zero if a single term in the query phrase has zero similarity with the document (irrespective of the relevance of other terms in the query phrase).

"Dirichlet" algorithm uses smoothing function to overcome the effect of multiplication by zero. This way, a small fraction of relevance (from relevant terms) is distributed among all the terms that did not appear in the document.

Galago has two classes *"DirichletScorer"*[2] and *"DirichletSmoother"*[3] that work in conjunction to perform the operations.

Here is the code snippet from "DirichletScorer.java" file of Galago that shows the fact that it is using *query likelihood* formula.

```
-------------------------------------------------------------
public double score(int count, int length) {
  double numerator = count + (mu * background);
  double denominator = length + mu;
  return Math.log(numerator / denominator);
}
-------------------------------------------------------------
```

[2]http://www.galagosearch.org/galagosearch-core/xref/org/galagosearch/core/scoring/DirichletScorer.html

[3]http://www.galagosearch.org/galagosearch-core/xref/org/galagosearch/core/scoring/DirichletSmoother.html

Here is the code snippet from "DirichletSmoother.java" file of Galago that shows the formula used to smoothing the term and document relevance measure.

```
-------------------------------------------------------------
public double score(int count, int length) {
  double numerator = count + (mu * background);
  double denominator = length + mu;
  return Math.log(numerator / denominator);
}
-------------------------------------------------------------
```

8.1.1 Short and Long Queries

Average length queries are best performing queries because they gain enough topic knowledge encoded in them. While, short queries may be ambiguous and grab lots of noise in the result set. At the same time, very long queries may be penalized due to several non-matching terms. Although, smoothing operation normalize the effect of multiplication by zero to some extent but it does not protect overall relevance drop. For example if there is a query phrase with two highly related terms (against certain documents) but, it has five non-related terms as well. Smoothing will give very small fraction of relevance to those non-related terms but overall relevance will drop significantly. Because of the fact that multi plication of two big numbers with five very small numbers will effectively be very small number.

Let us consider a practical example. Suppose there is a single word query "apple", search result might be ambiguous because apple is being used in various different contexts. Now, we make it "apple computer" or "apple computer screen", it becomes more specific and it now has topic semantics encoded. because it is not going to be confused by apple fruit. Hence, it is expected to perform better in terms of relevance. Adding some more keywords might make the query so specific that a relevant document (with all or most of the query terms) might be hard to find. That will affect the recall a lot if the focus is on maintaining good precision.

8.2 Discussion

Parameter setting will have minimal effect on average size queries. But, it may play significant role in adjusting the trade-off between precision and recall for (specially for long queries with several non-matching terms). It usually does not affect the ordering of top-ranked results if the parameters are not altered significantly.