

Assignment 4

CS834 Introduction to Information Retrieval

Fall 2017

Polykarpos Thomadakis

November 24, 2017

Question 8.3

For one query in the 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.

Answer

For this assignment I used Galago in order to generate the rankings of the provided documents and then calculated the requested metrics based on the relevance file, also provided. The metrics were calculated based on the formulas that are presented on the book. The script that was written for the purposes of this assignment is presented in listing 1. The user provides parameters such as: the files on which the search will be performed, the location of galago binary, the query id to generate metrics, the path to the galago index that will be created only once, a path to the xml query file and the path to the relevance file. A sample of the output of the program is shown on figure 1 for the query number 8.

```
1 import subprocess
2 import argparse
3 import os
4 from math import log
5 def precision(res, rel):
6     i = 0
7     for doc0 in rel:
8         for doc1 in res:
9             if doc0 in doc1:
10                 i +=1
11     return float(i)/len(res)
12
13 def average_precision(res, rel):
14     docs_found = 0
15     docs_avg = 0.0
16     doc_idx = 0
17     for doc1 in res:
18         doc_idx +=1
19         for doc0 in rel:
20             if doc0 in doc1:
21                 docs_found+=1
22                 docs_avg += float(docs_found)/doc_idx
23     return docs_avg/docs_found
24
25 def ndcg(res, rel, p):
26     idcg = 1.0
27     dcg = 0.0
28     for i in range(2, p+1):
29         idcg += 1/log(i, 2)
```

```

31     for doc0 in rel:
32         if res[0] in doc0:
33             dcg = 1.0
34     doc_idx = 1
35     for doc1 in res[1:]:
36         doc_idx += 1
37         for doc0 in rel:
38             if doc0 in doc1:
39                 dcg += 1/log(doc_idx, 2)
40     return dcg/idcg
41 def reciprocal(res, rel):
42     doc_idx = 0
43     for doc1 in res:
44         doc_idx += 1
45         for doc0 in rel:
46             if doc0 in doc1:
47                 return float(1)/doc_idx
48
49 parser = argparse.ArgumentParser('gmetrics')
50 parser.add_argument('-g', '--gpath', help='Path to galago binary', required=False,
51                     default='')
52 parser.add_argument('-x', '--idxpath', help='Path to galago index', required=False,
53                     default='index/')
54 parser.add_argument('-i', '--inpath', help='Path to input', required=True)
55 parser.add_argument('-q', '--qpath', help='Path to query xml file', required=False,
56                     default='cacm.query.xml')
57 parser.add_argument('--query', help='Query id to extract relevance about', required=
58                     True)
59 parser.add_argument('-r', '--rpath', help='Path to the .rel relevance file of galago',
60                     required=False, default='cacm.rel')
61 args = parser.parse_args()
62 g_bin = args.gpath + 'galago'
63 index_path = args.idxpath
64 input_path = args.inpath
65 query_path = args.qpath
66 query_idx = args.query
67 rel_path = args.rpath
68
69 if not os.path.exists(index_path):
70     p = subprocess.Popen(g_bin+" build --indexPath "+index_path+" --inputPath "+
71                           input_path, shell=True, stdout=subprocess.PIPE, stderr=subprocess.PIPE)
72     p.wait()
73
74 if not os.path.exists(query_path):
75     print "Query file not found!"
76     exit()
77
78 p = subprocess.Popen(g_bin+" batch-search --index="+index_path+" "+query_path, shell=
79                       True, stdout=subprocess.PIPE, stderr=subprocess.PIPE)
80 ranking = p.communicate()[0]
81 ranking_per_query = {}
82 top_ten_per_query = {}
83 lines = []
84 lines = ranking.split('\n')
85 for line in lines:
86     if line == '':
87         continue
88     qid ,q ,file ,seq ,rank ,g = line.split(" ")
89     if int(qid) not in ranking_per_query:
90         ranking_per_query[int(qid)] = []
91         top_ten_per_query[int(qid)] = []
92     ranking_per_query[int(qid)].append([file ,rank])
93
94 for qid in ranking_per_query.keys():
95     for i in range(0,10):
96         top_ten_per_query[int(qid)].append(ranking_per_query[qid][i][0])

```

```

89 relevant_docs_per_query = {}
91 with open(rel_path, 'r') as f:
93     for line in f:
94         qid, q, file, isrel = line.split(" ")
95         # print qid, q, file, isrel
96         if int(qid) not in relevant_docs_per_query:
97             relevant_docs_per_query[int(qid)] = []
98         if int(isrel) == 1:
99             relevant_docs_per_query[int(qid)].append(file)
100
101 print "Top ten documents from results:"
102 for doc in top_ten_per_query[int(query_idx)]:
103     print doc[doc.rfind(input_path):].replace(input_path, "")
104
105 print "Relevant documents:"
106 for doc in relevant_docs_per_query[int(query_idx)]:
107     print doc
108
109 print "Precision@10:"
110 print precision(top_ten_per_query[int(query_idx)][:10], relevant_docs_per_query[int(
111     query_idx)])
112 print "Average Precision:"
113 print average_precision(top_ten_per_query[int(query_idx)], relevant_docs_per_query[int(
114     query_idx)])
115 print "NDCG@5"
116 print ndcg(top_ten_per_query[int(query_idx)], relevant_docs_per_query[int(query_idx)
117     ], 5)
118 print "NDCG@10"
119 print ndcg(top_ten_per_query[int(query_idx)], relevant_docs_per_query[int(query_idx)
120     ], 10)
121 print "Reciprocal"
122 print reciprocal(top_ten_per_query[int(query_idx)], relevant_docs_per_query[int(
123     query_idx)])

```

Listing 1: Script that generates the metrics of the assignment

```

poll@poll-Aspire-V5-591G:~/Desktop/CS834/A4$ python galago_metrics.py -l cacth -g /home/poll/Desktop/galagosearch-1.04/bin/ --query 8
Top ten documents from results:
/CACM-2951.html /CACM-2371.html /CACM-2625.html /CACM-3032.html /CACM-2949.html /CACM-2500.html /CACM-1685.html /CACM-2541.html /CACM-2776.html /CACM-2499.html
Relevant documents:
CACM-2625 CACM-2849 CACM-3032
Precision@10:
0.2
Average Precision:
0.418666666667
NDCG@5
0.317533622364
NDCG@10
0.215230932498
Reciprocal
0.333333333333

```

Figure 1: Sample output of the script of this assignment

Question 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.

Answer

The chosen queries for this assignment are 3 and 5. The code can be seen in listing 2. The same command line arguments are given from the user, plus one more query index that is requested to extract metrics about. I used gnuplot for the graphs presented in figures 2, 3 and 4. The table with the interpolated precision values on standard recall values is shown in table 1.

```

1 import subprocess
2 import argparse
3 import os
4 from math import log
5
6 def recall_precision(res, rel, fp, fp2):
7     recall = []
8     precision = []
9     rel_ret = 0.0
10    i = 0
11    for doc1 in res:
12        i += 1
13        for doc0 in rel:
14            if doc0 in doc1:
15                rel_ret += 1
16                recall.append(rel_ret/len(rel))
17                precision.append(rel_ret/i)
18                fp.write(str(rel_ret/len(rel))+ " "+str(rel_ret/i)+"\n")
19    points = i-2
20    while points >= 0:
21        if precision[points+1] > precision[points]:
22            precision[points] = precision[points+1]
23        points -= 1
24
25    for k in range(0, i):
26        fp2.write(str(recall[k])+ " "+str(precision[k])+ "\n")
27
28    return rel_ret/len(rel)
29
30 parser = argparse.ArgumentParser('gmetrics')
31 parser.add_argument('-g', '--gpath', help='Path to galago binary', required=False,
32                     default='')
33 parser.add_argument('-x', '--idxpath', help='Path to galago index', required=False,
34                     default='index/')
35 parser.add_argument('-i', '--inpath', help='Path to input', required=True)
36 parser.add_argument('-q', '--qpath', help='Path to query xml file', required=False,
37                     default='cacm.query.xml')
38 parser.add_argument('--query1', help='1st Query id to graphs', required=True)
39 parser.add_argument('--query2', help='2nd Query id to graphs', required=True)
40 parser.add_argument('-r', '--rpath', help='Path to the .rel relevance file of galago',
41                     required=False, default='cacm.rel')
42 args = parser.parse_args()
43 g_bin = args.gpath + 'galago'
44 index_path = args.idxpath
45 input_path = args.inpath
46 query_path = args.qpath
47 query1_idx = args.query1
48 query2_idx = args.query2
49 rel_path = args.rpath
50
51 if not os.path.exists(index_path):
52     p = subprocess.Popen(g_bin+" build --indexPath "+index_path+" --inputPath "+
53                          input_path, shell=True, stdout=subprocess.PIPE, stderr=subprocess.PIPE)
54     p.wait()
55
56 if not os.path.exists(query_path):
57     print "Query file not found!"
58     exit()
59
60 p = subprocess.Popen(g_bin+" batch-search --index="+index_path+" "+query_path, shell=
61                      True, stdout=subprocess.PIPE, stderr=subprocess.PIPE)
62 ranking = p.communicate()[0]
63 ranking_per_query = {}
64 top_ten_per_query = {}
65 lines = []

```

```

lines = ranking.split('\n')
61 for line in lines:
    if line == '':
63         continue
    qid ,q ,file ,seq ,rank ,g = line.split(" ")
65     if int(qid) not in ranking_per_query:
        ranking_per_query[int(qid)] = []
67         top_ten_per_query[int(qid)] = []
        ranking_per_query[int(qid)].append(file)
69     # print ranking_per_query.keys()
    for qid in ranking_per_query.keys():
71         for i in range(0,10):
            top_ten_per_query[int(qid)].append(ranking_per_query[int(qid)][i])
73
    relevant_docs_per_query = {}
75     with open(rel_path, 'r') as f:
        for line in f:
77             qid ,q ,file ,isrel = line.split(" ")
            # print qid,q,file,isrel
79             if int(qid) not in relevant_docs_per_query :
                relevant_docs_per_query[int(qid)] = []
81             if int(isrel) == 1:
                relevant_docs_per_query[int(qid)].append(file)
83
            with open("graph1.txt", 'w+') as fp:
85                 with open("int_graph1.txt", 'w+') as fp2:
                    recall_precision(ranking_per_query[int(query1_idx)], relevant_docs_per_query[int(
                        query1_idx)], fp, fp2)
87             with open("graph2.txt", 'w+') as fp:
                with open("int_graph2.txt", 'w+') as fp2:
89                     recall_precision(ranking_per_query[int(query2_idx)], relevant_docs_per_query[int(
                        query2_idx)], fp, fp2)

91     int1x = []
    int1y = []
93     int2x = []
    int2y = []
95     int1x_final = []
    int1y_final = []
97     int2x_final = []
    int2y_final = []
99
    points = [0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1.0]
101    with open("int_graph1.txt", 'r+') as fp2:
        for line in fp2:
103            x,y = line.split(" ")
            int1x.append(float(x))
105            int1y.append(float(y))

107    with open("int_graph2.txt", 'r+') as fp2:
        for line in fp2:
109            x,y = line.split(" ")
            int2x.append(float(x))
111            int2y.append(float(y))

113    # print int1x
115    for point in points:
        for i in range(0, len(int1x)-1):
117            if int1x[i] < point and int1x[i+1] > point:
                int1x_final.append(point)
                int1y_final.append(int1y[i])
119            if int2x[i] < point and int2x[i+1] > point:
                int2x_final.append(point)
                int2y_final.append(int2y[i])
121
123

```

```

125     if len(int1x_final)>len(int2x_final):
126         for i in range(0,len(int1x_final)-len(int2x_final)):
127             int2y_final.append(0)
128     else:
129         for i in range(0,len(int2x_final)-len(int1x_final)):
130             int1y_final.append(0)
131     print "Interpolated values for standard recall values"
132     for i in range(0,len(int1x_final)):
133         print int1x_final[i],int1y_final[i],int2y_final[i]
134
135     with open("avg_int_graph.txt",'w+') as fp2:
136         fp2.write(str(0)+" "+str((int1y[0]+int2y[0])/2)+"\n")
137
138     for i in range(0,len(int1x_final)):
139         fp2.write(str(int1x_final[i])+" "+str((int1y_final[i]+int2y_final[i])/2)+"\n")

```

Listing 2: Script that generates the graphs of the assignment

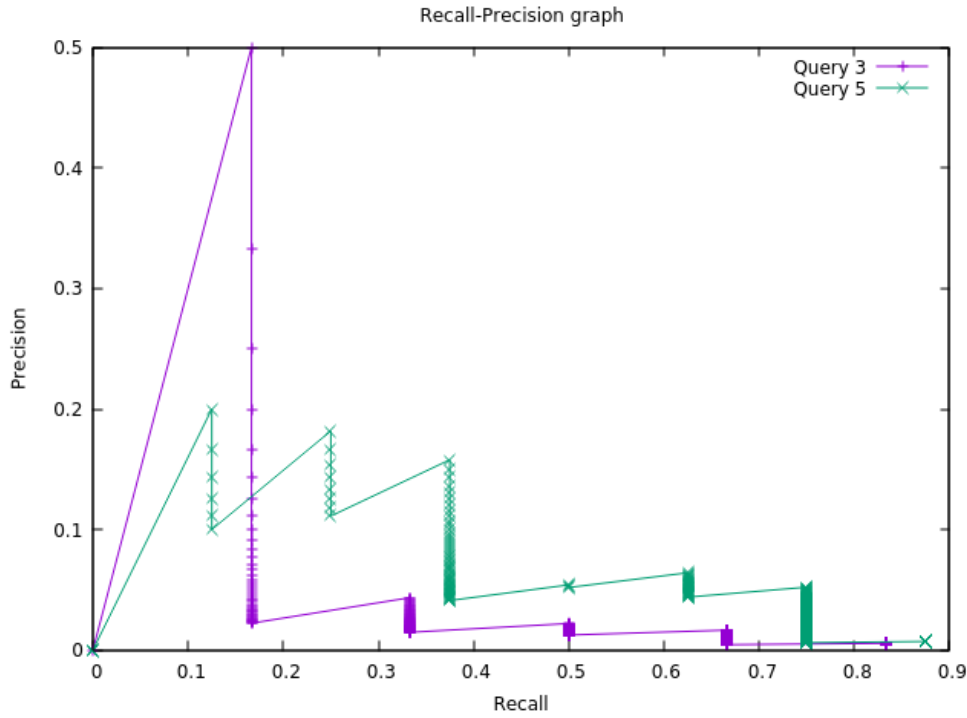


Figure 2: The uninterpolated recall-precision graphs for the two queries

Table 1: Table with the interpolated values in standard recall values

0.1	0.5	0.2
0.2	0.0434782608696	0.181818181818
0.3	0.0434782608696	0.157894736842
0.4	0.0220588235294	0.0641025641026
0.6	0.0165289256198	0.0641025641026
0.7	0.00563063063063	0.0521739130435
0.8	0.00563063063063	0.00713557594292

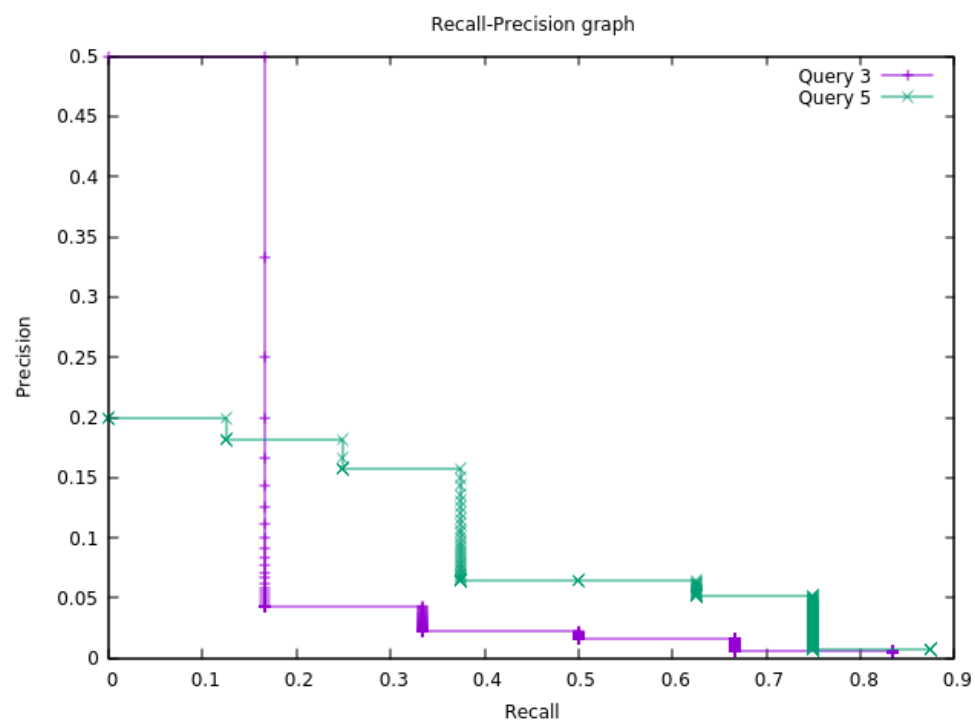


Figure 3: The interpolated recall-precision graphs for the two queries

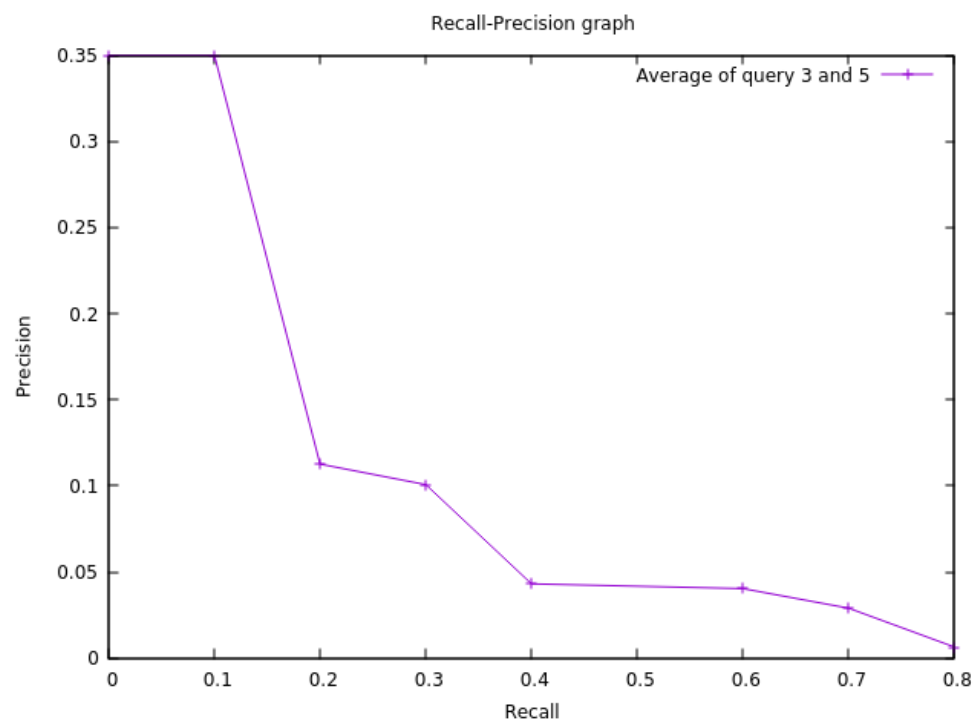


Figure 4: The interpolated average recall-precision graph for the two queries

Question 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.

Answer

Again, I used the formulas provided on the book to generate the metrics requested for the whole cacm query set. The script I wrote for this purpose is presented in listing 3. The values generated for this assignment can be seen in figure 5 and the average interpolated recall-precision graph for the entire query set in figure 6.

```
import subprocess
2 import argparse
import os
4 from math import log
def recall_precision(res, rel):
6     recall = []
    precision = []
8     rel_ret = 0.0
    i = 0
10    for doc1 in res:
        i +=1
12        for doc0 in rel:
            if doc0 in doc1:
14                rel_ret +=1
            if len(rel) ==0:
16                recall.append(0.0)
                precision.append(0.0)
18            else:
                recall.append(rel_ret/len(rel))
20                precision.append(rel_ret/i)
    points = i-2
22    while points >=0:
        if precision[points+1] > precision[points]:
24            precision[points] = precision[points+1]
            points -=1
26    return recall, precision

28 def ndcg(res, rel, p):
    idcg = 1.0
30    dcg = 0.0
    for i in range(2, p+1):
32        idcg += 1/log(i, 2)
    for doc0 in rel:
34        if res[0] in doc0:
            dcg = 1.0
36    doc_idx =1
    for doc1 in res[1:]:
38        doc_idx +=1
        for doc0 in rel:
40            if doc0 in doc1:
                dcg += 1/log(doc_idx, 2)
42    return dcg/idcg

44 def precision(res, rel):
    i = 0
46    for doc0 in rel:
        for doc1 in res:
48            if doc0 in doc1:
                i +=1
50    return float(i)/len(res)

52 def average_precision(res, rel):
```



```

docs_found = 0
54 docs_avg = 0.0
doc_idx = 0
56 for doc1 in res:
    doc_idx +=1
58     for doc0 in rel:
        if doc0 in doc1:
60             docs_found+=1
            docs_avg += float(docs_found)/doc_idx
62 if docs_found == 0:
    return 0.0
64 return docs_avg/docs_found

66 def ndcg(res,rel,p):
    idcg = 1.0
68 dcg = 0.0
    for i in range(2,p+1):
        idcg += 1/log(i,2)
70     for doc0 in rel:
        if res[0] in doc0:
72             dcg = 1.0
74 doc_idx =1
        for doc1 in res[1:]:
76             doc_idx +=1
            for doc0 in rel:
78                 if doc0 in doc1:
                    dcg += 1/log(doc_idx,2)
80 return dcg/idcg

82 parser = argparse.ArgumentParser('gmetrics')
parser.add_argument('-g','--gpath',help='Path to galago binary',required=False,
    default='')
84 parser.add_argument('-x','--idxpath',help='Path to galago index',required=False,
    default='index/')
parser.add_argument('-i','--inpath',help='Path to input',required=True)
86 parser.add_argument('-q','--qpath',help='Path to query xml file',required=False,
    default='cacm.query.xml')
    # parser.add_argument('--query',help='Query id to extract relevance about',required=
    True)
88 parser.add_argument('-r','--rpath',help='Path to the .rel relevance file of galago',
    required=False, default='cacm.rel')
args = parser.parse_args();
90 g_bin = args.gpath + 'galago'
index_path = args.idxpath
92 input_path = args.inpath
query_path = args.qpath
94 # query_idx = args.query
rel_path = args.rpath
96

98 if not os.path.exists(index_path):
    p = subprocess.Popen(g_bin+" build --indexPath "+index_path+" --inputPath "+
        input_path, shell=True, stdout=subprocess.PIPE, stderr=subprocess.PIPE)
100 p.wait()

102 if not os.path.exists(query_path):
    print "Query file not found!"
104 exit()

106 p = subprocess.Popen(g_bin+" batch-search --index="+index_path+" "+query_path, shell=
    True, stdout=subprocess.PIPE, stderr=subprocess.PIPE)
ranking = p.communicate()[0]
108
ranking_per_query = {}
110 lines = []
lines = ranking.split('\n')

```

```

112 top_ten_per_query = {}

114 for line in lines:
115     if line == '':
116         continue
117     qid ,q ,file ,seq ,rank ,g = line.split(" ")
118     if int(qid) not in ranking_per_query:
119         ranking_per_query[int(qid)]=[]
120         top_ten_per_query[int(qid)]=[]
121         ranking_per_query[int(qid)].append(file)
122 for qid in ranking_per_query.keys():
123     for i in range(0,10):
124         top_ten_per_query[int(qid)].append(ranking_per_query[qid][i])
125 relevant_docs_per_query = {}
126 for i in range(1,63):
127     relevant_docs_per_query[i]=[]
128 with open(rel_path , 'r') as f:
129     for line in f:
130         qid ,q ,file ,isrel = line.split(" ")
131         if int(qid) not in relevant_docs_per_query :
132             relevant_docs_per_query[int(qid)] = []
133         if int(isrel) == 1:
134             relevant_docs_per_query[int(qid)].append(file)
135 avg_prec = []
136 prec = {}
137 recall = {}
138 for i in relevant_docs_per_query.keys():
139     avg_prec.append(average_precision(ranking_per_query[i],relevant_docs_per_query[i]))
140     recall[i],prec[i] = recall_precision(ranking_per_query[i],relevant_docs_per_query[i])

142 print "Mean Average Precision "
143 print sum(avg_prec)/len(avg_prec)
144 int_graph = {}
145 points = [0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1.0]
146 for i in ranking_per_query.keys():
147     int_graph[i] = [0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0]
148     for j in range(0,10):
149         point = points[j]
150         for x in range(1,len(recall[i])):
151             if recall[i][x-1]<point and recall[i][x]>point:
152                 int_graph[i][j] = prec[i][x]

154 average_int_graph = [0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0]
155 with open('cacm_graph','w+') as fp:
156     for j in range(0,9):
157         for i in ranking_per_query.keys():
158             average_int_graph[j] += int_graph[i][j]
159             average_int_graph[j] = average_int_graph[j]/len(ranking_per_query.keys())
160     fp.write(str(((j+1)*0.1))+ " "+str(average_int_graph[j])+ "\n")

162 avg_ndcg5 = 0.0
163 avg_ndcg10 = 0.0
164 avg_prec10 = 0.0
165 for i in relevant_docs_per_query.keys():
166     avg_ndcg5+= ndcg(top_ten_per_query[i],relevant_docs_per_query[i],5)
167     avg_ndcg10+= ndcg(top_ten_per_query[i],relevant_docs_per_query[i],10)
168     avg_prec10+=precision(top_ten_per_query[i],relevant_docs_per_query[i])
169 avg_ndcg5 = avg_ndcg5/ len(relevant_docs_per_query.keys())
170 avg_ndcg10 = avg_ndcg10/ len(relevant_docs_per_query.keys())
171 avg_prec10 = avg_prec10/len(relevant_docs_per_query.keys())
172 print "Average NDCG@5"
173 print avg_ndcg5
174 print "Average NDCG@10"
175 print avg_ndcg10
176 print "Precision@10"

```

```
print avg_prec10
```

Listing 3: Script to generate the metrics of the assignment

```
poll@poll-Aspire-V5-S91G:~/Desktop/CS834/A4$ python cacm_metrics.py -i cacm -g /home/poll/Desktop/galagosearch-1.04/bin/
Mean Average Precision
0.287604220803
Average NDCG@5
0.282761380759
Average NDCG@10
0.191661579652
Precision@10
0.244444444444
```

Figure 5: Metrics requested for the entire cacm query set

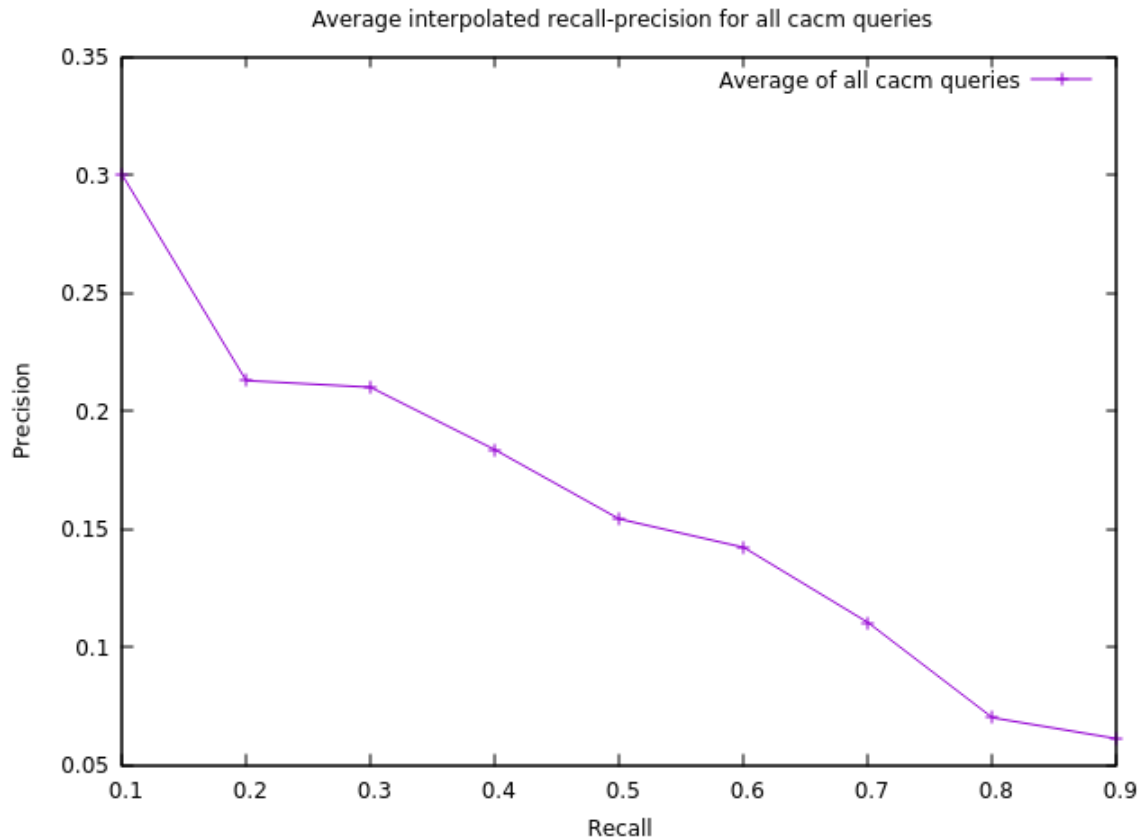


Figure 6: The average interpolated recall-precision graph for the entire query set

Question 8.7

Another measure that has been used in a 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.

Answer

The script to calculate the R-precision and compare it with other metrics can be seen in listing 4. Figure 7 shows the R-precision value as well as the other metrics of the CACM query set.

```
1 import subprocess
2 import argparse
3 import os
4 from math import log
5 def recall_precision(res, rel):
6     recall = []
7     precision = []
8     rel_ret = 0.0
9     i = 0
10    for doc1 in res:
11        i += 1
12        for doc0 in rel:
13            if doc0 in doc1:
14                rel_ret += 1
15            if len(rel) == 0:
16                recall.append(0.0)
17                precision.append(0.0)
18            else:
19                recall.append(rel_ret / len(rel))
20                precision.append(rel_ret / i)
21    points = i - 2
22    while points >= 0:
23        if precision[points + 1] > precision[points]:
24            precision[points] = precision[points + 1]
25        points -= 1
26    return recall, precision
27
28 def ndcg(res, rel, p):
29     idcg = 1.0
30     dcg = 0.0
31     for i in range(2, p + 1):
32         idcg += 1 / log(i, 2)
33     for doc0 in rel:
34         if res[0] in doc0:
35             dcg = 1.0
36     doc_idx = 1
37     for doc1 in res[1:]:
38         doc_idx += 1
39         for doc0 in rel:
40             if doc0 in doc1:
41                 dcg += 1 / log(doc_idx, 2)
42     return dcg / idcg
43
44 def precision(res, rel):
45     i = 0
46     for doc0 in rel:
47         for doc1 in res:
48             if doc0 in doc1:
49                 i += 1
50     return float(i) / len(res)
51
52 def r_precision(res, rel):
53     i = 0
54     rel_ret = 0
55     for doc1 in res:
56         if i == len(rel):
57             break
58         for doc0 in rel:
59             if doc0 in doc1:
60                 rel_ret += 1
61     i += 1
```

```

63     if len(rel) == 0:
        return 0.0
        return rel_ret/len(rel)
65
67 def average_precision(res, rel):
    docs_found = 0
    docs_avg = 0.0
    doc_idx = 0
    for doc1 in res:
        doc_idx +=1
        for doc0 in rel:
            if doc0 in doc1:
                docs_found+=1
                docs_avg += float(docs_found)/doc_idx
    if docs_found == 0:
        return 0.0
    return docs_avg/docs_found
79
81 def ndcg(res, rel, p):
    idcg = 1.0
    dcg = 0.0
    for i in range(2, p+1):
        idcg += 1/log(i, 2)
    for doc0 in rel:
        if res[0] in doc0:
            dcg = 1.0
    doc_idx =1
    for doc1 in res[1:]:
        doc_idx +=1
        for doc0 in rel:
            if doc0 in doc1:
                dcg += 1/log(doc_idx, 2)
    return dcg/idcg
95
97 parser = argparse.ArgumentParser('gmetrics')
parser.add_argument('-g', '--gpath', help='Path to galago binary', required=False,
                    default='')
parser.add_argument('-x', '--idxpath', help='Path to galago index', required=False,
                    default='index/')
99 parser.add_argument('-i', '--inpath', help='Path to input', required=True)
parser.add_argument('-q', '--qpath', help='Path to query xml file', required=False,
                    default='cacm.query.xml')
101 # parser.add_argument('--query', help='Query id to extract relevance about', required=
    True)
    parser.add_argument('-r', '--rpath', help='Path to the .rel relevance file of galago',
                        required=False, default='cacm.rel')
103 args = parser.parse_args();
g_bin = args.gpath + 'galago'
105 index_path = args.idxpath
input_path = args.inpath
107 query_path = args.qpath
# query_idx = args.query
109 rel_path = args.rpath

111
113 if not os.path.exists(index_path):
    p = subprocess.Popen(g_bin+" build --indexPath "+index_path+" --inputPath "+
        input_path, shell=True, stdout=subprocess.PIPE, stderr=subprocess.PIPE)
    p.wait()
115
117 if not os.path.exists(query_path):
    print "Query file not found!"
    exit()
119
p = subprocess.Popen(g_bin+" batch-search --index="+index_path+" "+query_path, shell=
    True, stdout=subprocess.PIPE, stderr=subprocess.PIPE)

```

```

121 ranking = p.communicate()[0]

123 ranking_per_query = {}
lines = []
125 lines = ranking.split('\n')
top_ten_per_query = {}

127 for line in lines:
129     if line == '':
        continue
131     qid ,q ,file ,seq ,rank ,g = line.split(" ")
        if int(qid) not in ranking_per_query:
133             ranking_per_query[int(qid)]=[]
            top_ten_per_query[int(qid)]=[]
135             ranking_per_query[int(qid)].append(file)
        for qid in ranking_per_query.keys():
137             for i in range(0,10):
                top_ten_per_query[int(qid)].append(ranking_per_query[qid][i])
139 relevant_docs_per_query = {}
        for i in range(1,63):
141             relevant_docs_per_query[i]=[]
        with open(rel_path , 'r') as f:
143             for line in f:
                qid ,q ,file ,isrel = line.split(" ")
145                 # print qid,q,file ,isrel
                if int(qid) not in relevant_docs_per_query :
147                     relevant_docs_per_query[int(qid)] = []
                    if int(isrel) == 1:
149                         relevant_docs_per_query[int(qid)].append(file)
        avg_prec = []
151 prec = {}
        recall = {}
153 for i in relevant_docs_per_query.keys():
        avg_prec.append(average_precision(ranking_per_query[i],relevant_docs_per_query[i]))
155 recall[i],prec[i] = recall_precision(ranking_per_query[i],relevant_docs_per_query[i])
        # print "Query "+str(i)+": Average Precision —> "+str(average_precision(
            ranking_per_query[i],relevant_docs_per_query[i]))

157 print "Mean Average Precision"
159 print sum(avg_prec)/len(avg_prec)
        int_graph = {}
161 points = [0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1.0]
        for i in ranking_per_query.keys():
163             int_graph[i] = [0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0]
                for j in range(0,10):
165                     point = points[j]
                        for x in range(1,len(recall[i])):
167                             if recall[i][x-1]<point and recall[i][x]>point:
                                    int_graph[i][j] = prec[i][x]

169 average_int_graph = [0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0]
171 with open('cacm-graph','w+') as fp:
        for j in range(0,9):
173             for i in ranking_per_query.keys():
                average_int_graph[j] += int_graph[i][j]
175             average_int_graph[j] = average_int_graph[j]/len(ranking_per_query.keys())
                fp.write(str(((j+1)*0.1))+ " "+str(average_int_graph[j])+ "\n")

177 avg_ndcg5 = 0.0
179 avg_ndcg10 = 0.0
        avg_prec10 = 0.0
181 avg_rprec = 0.0
        for i in relevant_docs_per_query.keys():
183             avg_ndcg5+= ndcg(top_ten_per_query[i],relevant_docs_per_query[i],5)
                avg_ndcg10+= ndcg(top_ten_per_query[i],relevant_docs_per_query[i],10)

```

```

185     avg_prec10+=precision(top_ten_per_query[i],relevant_docs_per_query[i])
    avg_rprec +=r_precision(ranking_per_query[i],relevant_docs_per_query[i])
187 avg_ndcg5 = avg_ndcg5/ len(relevant_docs_per_query.keys())
    avg_ndcg10 = avg_ndcg10/ len(relevant_docs_per_query.keys())
189 avg_prec10 = avg_prec10/len(relevant_docs_per_query.keys())
    avg_rprec = avg_rprec/len(relevant_docs_per_query.keys())
191 print "Average NDCG@5"
    print avg_ndcg5
193 print "Average NDCG@10"
    print avg_ndcg10
195 print "Precision@10"
    print avg_prec10
197 print "Average R-Precision"
    print avg_rprec

```

Listing 4: Script to generate the R-precision metric and compare it with others

```

poll@poll-Aspire-V5-591G:~/Desktop/CS834/A4$ python r_precision.py -t cacm -g /home/poll/Desktop/galagosearch-1.04/bin/
Mean Average Precision
0.287604220803
Average NDCG@5
0.282761380759
Average NDCG@10
0.191661579652
Precision@10
0.244444444444
Average R-Precision
0.047619047619

```

Figure 7: The comparison between R-precision, MAP, NDCG@5, NDCG@10 and Precision@10

Question 8.9

For one query in the CACM collection, generate a ranking and calculate BPREF. Show that the two formulations of BPREF give the same value.

Answer

I use the two formulas of the book to calculate BPREF for the query 2. They both give the same results as the book states and can be seen in figure 8. BPREF1 refers to the formula:

$$BPREF1 = \frac{1}{R} \sum_{dr} (1 - \frac{N_{dr}}{R})$$

and BPREF2 to the formula:

$$BPREF2 = \frac{P}{P+Q}$$

The code for this assignment is shown in listing 5.

```

poll@poll-Aspire-V5-591G:~/Desktop/CS834/A4$ python bpref.py -t cacm -g /home/poll/Desktop/galagosearch-1.04/bin/ --query 2
BPREF1
0.666666666667
BPREF2
0.666666666667

```

Figure 8: BPREF as generate by the two formulas in the book

```

import subprocess
2 import argparse
import os
4 from math import log
def recall_precision(res,rel):
6     recall = []

```

```

8 precision = []
rel_ret = 0.0
i = 0
10 for doc1 in res:
    i +=1
12     for doc0 in rel:
14         if doc0 in doc1:
            rel_ret +=1
16     if len(rel) ==0:
        recall.append(0.0)
        precision.append(0.0)
18     else:
        recall.append(rel_ret/len(rel))
        precision.append(rel_ret/i)
20 points = i-2
22 while points >=0:
    if precision[points+1]>precision[points]:
24         precision[points] = precision[points+1]
        points-=1
26 return recall,precision

28 def ndcg(res,rel,p):
    idcg = 1.0
    dcg = 0.0
30     for i in range(2,p+1):
32         idcg += 1/log(i,2)
    for doc0 in rel:
34         if res[0] in doc0:
            dcg = 1.0
36     doc_idx =1
    for doc1 in res[1:]:
38         doc_idx +=1
        for doc0 in rel:
40             if doc0 in doc1:
                dcg += 1/log(doc_idx,2)
42     return dcg/idcg

44 def precision(res,rel):
    i = 0
46     for doc0 in rel:
48         for doc1 in res:
            if doc0 in doc1:
                i +=1
50     return float(i)/len(res)

52 def bpref1(res,rel):
    result = 0.0
54     found = False
    R = len(rel)
56     R_non_relevant = []
    i = 0
58     for doc1 in res:
        found = False
60         for doc0 in rel:
62             if doc0 in doc1:
                found = True
            if not found:
64                 i+=1
            if i> R:
66                 break
        R_non_relevant.append(doc1)
68     # print len(R_non_relevant)
    for doc0 in rel:
70         i =0
        Ndr = R
72         # print "AGAI"

```



```

74     for doc1 in R_non_relevant:
75         if doc0 in doc1:
76             Ndr = i
77             i +=1
78             result += (1.0 - float(Ndr)/R)
79     return 1.00/R*result

80 def bpref2(res, rel):
81     result = 0.0
82     found = False
83     R = len(rel)
84     R_non_relevant = []
85     i = 0
86     for doc1 in res:
87         found = False
88         for doc0 in rel:
89             if doc0 in doc1:
90                 found = True
91             if not found:
92                 i +=1
93             if i > R:
94                 break
95         R_non_relevant.append(doc1)
96     # print len(R_non_relevant)
97     p = 0.0
98     q = 0.0
99     for doc0 in rel:
100         i =0
101         found = False
102         # print "AGAIIn"
103         for doc1 in R_non_relevant:
104             if doc0 in doc1:
105                 found = True
106                 q += i
107                 p += R-i
108                 i +=1
109             if not found:
110                 q +=R
111
112     return float(p)/(p+q)

114 parser = argparse.ArgumentParser('gmetrics')
115 parser.add_argument('-g', '--gpath', help='Path to galago binary', required=False,
116                     default='')
117 parser.add_argument('-x', '--idxpath', help='Path to galago index', required=False,
118                     default='index/')
119 parser.add_argument('-i', '--inpath', help='Path to input', required=True)
120 parser.add_argument('-q', '--qpath', help='Path to query xml file', required=False,
121                     default='cacm.query.xml')
122 parser.add_argument('--query', help='Query id to extract relevance about', required=
123                     True)
124 parser.add_argument('-r', '--rpath', help='Path to the .rel relevance file of galago',
125                     required=False, default='cacm.rel')
126 args = parser.parse_args();
127 g_bin = args.gpath + 'galago'
128 index_path = args.idxpath
129 input_path = args.inpath
130 query_path = args.qpath
131 query_idx = int(args.query)
132 rel_path = args.rpath

133 if not os.path.exists(index_path):
134     p = subprocess.Popen(g_bin+" build --indexPath "+index_path+" --inputPath "+
135                           input_path, shell=True, stdout=subprocess.PIPE, stderr=subprocess.PIPE)

```

```

134     p.wait()
135 if not os.path.exists(query_path):
136     print "Query file not found!"
137     exit()
138
139 p = subprocess.Popen(g_bin+" batch-search --index="+index_path+" "+query_path, shell=
140     True, stdout=subprocess.PIPE, stderr=subprocess.PIPE)
141 ranking = p.communicate()[0]
142
143 ranking_per_query = {}
144 lines = []
145 lines = ranking.split('\n')
146 top_ten_per_query = {}
147
148 for line in lines:
149     if line == '':
150         continue
151     qid ,q ,file ,seq ,rank ,g = line.split(" ")
152     if int(qid) not in ranking_per_query:
153         ranking_per_query[int(qid)] = []
154         top_ten_per_query[int(qid)] = []
155     ranking_per_query[int(qid)].append(file)
156 for qid in ranking_per_query.keys():
157     for i in range(0,10):
158         top_ten_per_query[int(qid)].append(ranking_per_query[qid][i])
159 relevant_docs_per_query = {}
160 for i in range(1,63):
161     relevant_docs_per_query[i] = []
162 with open(rel_path, 'r') as f:
163     for line in f:
164         qid ,q ,file ,isrel = line.split(" ")
165         # print qid,q,file,isrel
166         if int(qid) not in relevant_docs_per_query :
167             relevant_docs_per_query[int(qid)] = []
168         if int(isrel) == 1:
169             relevant_docs_per_query[int(qid)].append(file)
170
171 bp1=bpref1(ranking_per_query[query_idx],relevant_docs_per_query[query_idx])
172 bp2=bpref2(ranking_per_query[query_idx],relevant_docs_per_query[query_idx])
173 print "BPREF1"
174 print bp1
175 print "BPREF2"
176 print bp2

```

Listing 5: Script to generate the BPREF metric with both formulas