

Assignment 4

CS834 Introduction to Information Retrieval

Fall 2017

Polykarpou 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 assingnment 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', '--idxxpath', 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.idxxpath
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:
92     for line in f:
93         qid, q, file, isrel = line.split(" ")
94         # print qid,q,file,isrel
95         if int(qid) not in relevant_docs_per_query:
96             relevant_docs_per_query[int(qid)] = []
97         if int(isrel) == 1:
98             relevant_docs_per_query[int(qid)].append(file)
99
100 print "Top ten documents from results:"
101 for doc in top_ten_per_query[int(query_idx)]:
102     print doc[doc.rfind(input_path):].replace(input_path, ""),
103 print
104 print "Relevant documents:"
105 for doc in relevant_docs_per_query[int(query_idx)]:
106     print doc,
107 print
108 print "Precision@10:"
109 print precision(top_ten_per_query[int(query_idx)][:10], relevant_docs_per_query[int(query_idx)])
110 print "Average Precision:"
111 print average_precision(top_ten_per_query[int(query_idx)], relevant_docs_per_query[int(query_idx)])
112 print "NDCG@5"
113 print ndcg(top_ten_per_query[int(query_idx)], relevant_docs_per_query[int(query_idx)], 5)
114 print "NDCG@10"
115 print ndcg(top_ten_per_query[int(query_idx)], relevant_docs_per_query[int(query_idx)], 10)
116 print "Reciprocal"
117 print reciprocal(top_ten_per_query[int(query_idx)], relevant_docs_per_query[int(query_idx)])

```

Listing 1: Script that generates the metrics of the assignment

```

poll@poll-Aspire-V5-591G:~/Desktop/CS834/A4$ python galago_metrics.py -i cacm -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:
0.0
Average Precision:
0.4166666666666667
NDCG@5
0.317533222364
NDCG@10
0.215230932498
Reciprocal
0.3333333333333333

```

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 = []

```

```

1 lines = ranking.split('\n')
2 for line in lines:
3     if line == '':
4         continue
5     qid ,q ,file ,seq ,rank ,g = line.split(" ")
6     if int(qid) not in ranking_per_query:
7         ranking_per_query[int(qid)]=[]
8         top_ten_per_query[int(qid)]=[]
9         ranking_per_query[int(qid)].append(file)
10    # print ranking_per_query.keys()
11    for qid in ranking_per_query.keys():
12        for i in range(0,10):
13            top_ten_per_query[int(qid)].append(ranking_per_query[int(qid)][i])
14
15 relevant_docs_per_query = {}
16 with open(rel_path,'r') as f:
17     for line in f:
18         qid ,q ,file ,isrel = line.split(" ")
19         # print qid,q,file ,isrel
20         if int(qid) not in relevant_docs_per_query :
21             relevant_docs_per_query[int(qid)] = []
22         if int(isrel) == 1:
23             relevant_docs_per_query[int(qid)].append(file)
24
25 with open("graph1.txt",'w+') as fp:
26     with open("int_graph1.txt",'w+') as fp2:
27         recall_precision(ranking_per_query[int(query1_idx)],relevant_docs_per_query[int(query1_idx)],fp,fp2)
28 with open("graph2.txt",'w+') as fp:
29     with open("int_graph2.txt",'w+') as fp2:
30         recall_precision(ranking_per_query[int(query2_idx)],relevant_docs_per_query[int(query2_idx)],fp,fp2)
31
32 int1x = []
33 int1y = []
34 int2x = []
35 int2y = []
36 int1x_final = []
37 int1y_final = []
38 int2x_final = []
39 int2y_final = []
40
41 points = [0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1.0]
42 with open("int_graph1.txt",'r+') as fp2:
43     for line in fp2:
44         x,y = line.split(" ")
45         int1x.append(float(x))
46         int1y.append(float(y))
47
48 with open("int_graph2.txt",'r+') as fp2:
49     for line in fp2:
50         x,y = line.split(" ")
51         int2x.append(float(x))
52         int2y.append(float(y))
53
54
55 # print int1x
56 for point in points:
57     for i in range(0,len(int1x)-1):
58         if int1x[i]<point and int1x[i+1]>point:
59             int1x_final.append(point)
60             int1y_final.append(int1y[i])
61         if int2x[i]<point and int2x[i+1]>point:
62             int2x_final.append(point)
63             int2y_final.append(int2y[i])
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
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

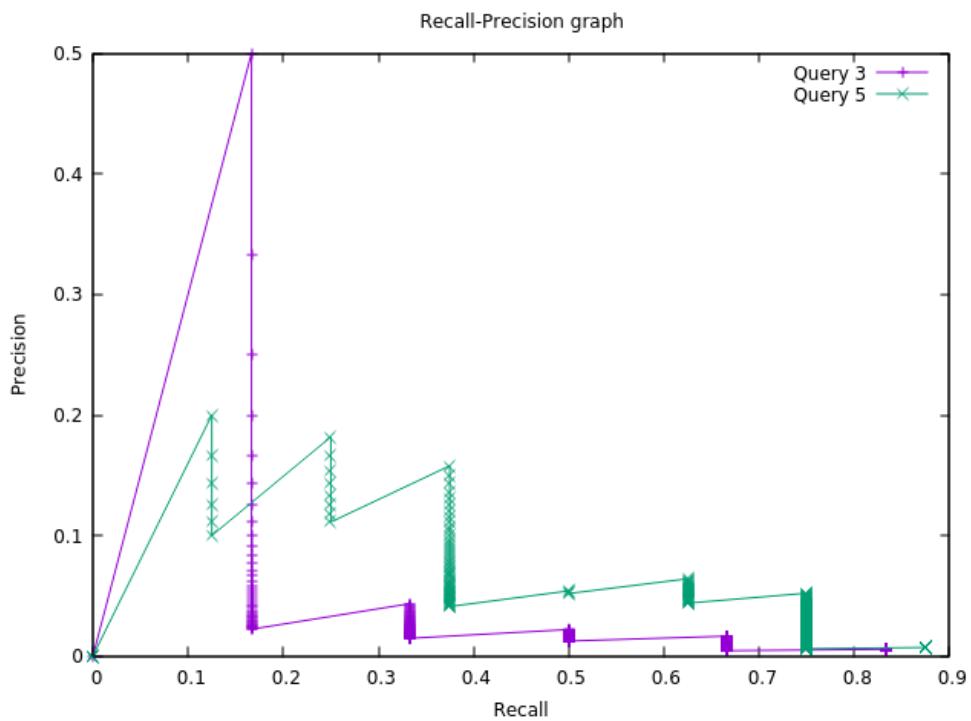


Figure 2: The uninterpolated recall-precision graphs for the two queries

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

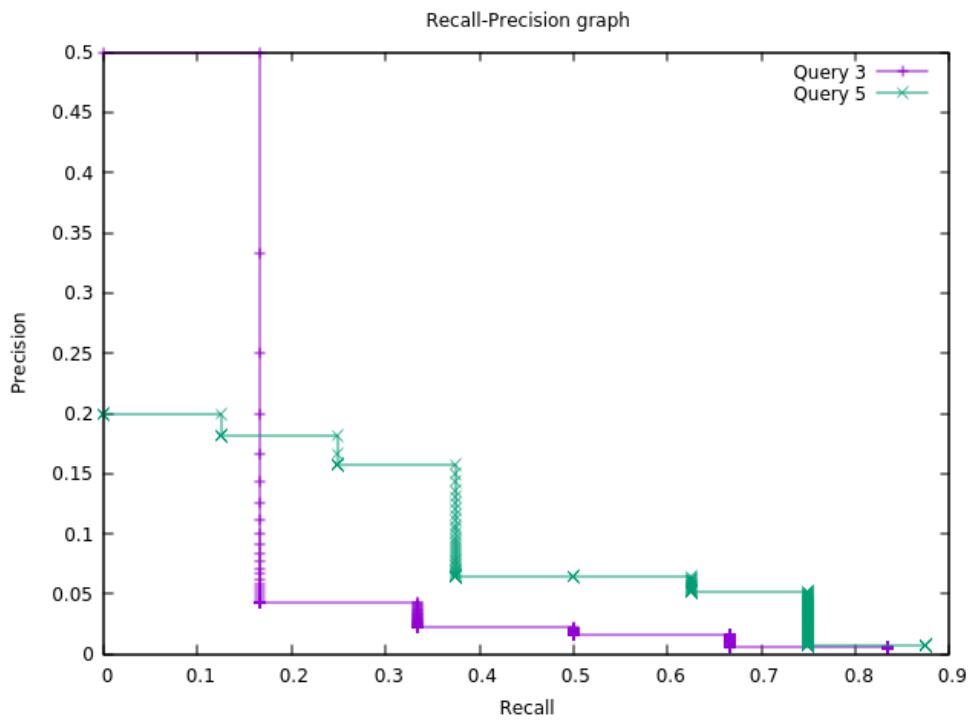


Figure 3: The interpolated recall-precision graphs for the two queries

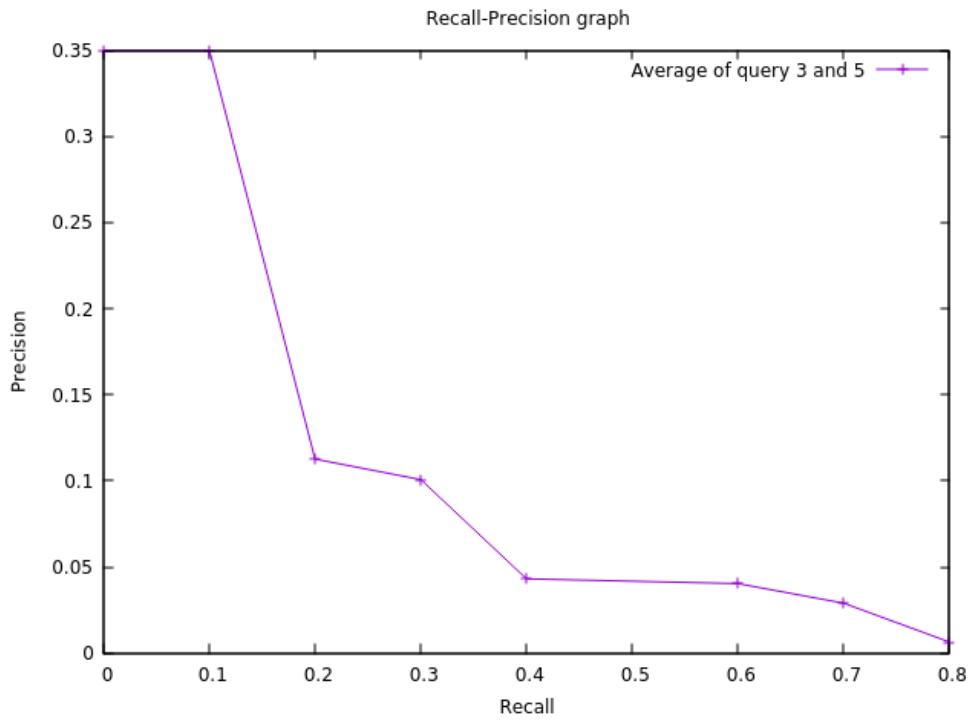


Figure 4: The interpolated average recall-precision graph for the two queries

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
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 average_precision(res, rel):
```

```

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

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

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

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

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

107 p = subprocess.Popen(g_bin+" batch-search --index="+index_path+" "+query_path , shell=
108                     True, stdout=subprocess.PIPE, stderr=subprocess.PIPE)
109 ranking = p.communicate()[0]

110 ranking_per_query = {}
111 lines = []
112 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])
141
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]
153
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")
161
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-V5-591G:~/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

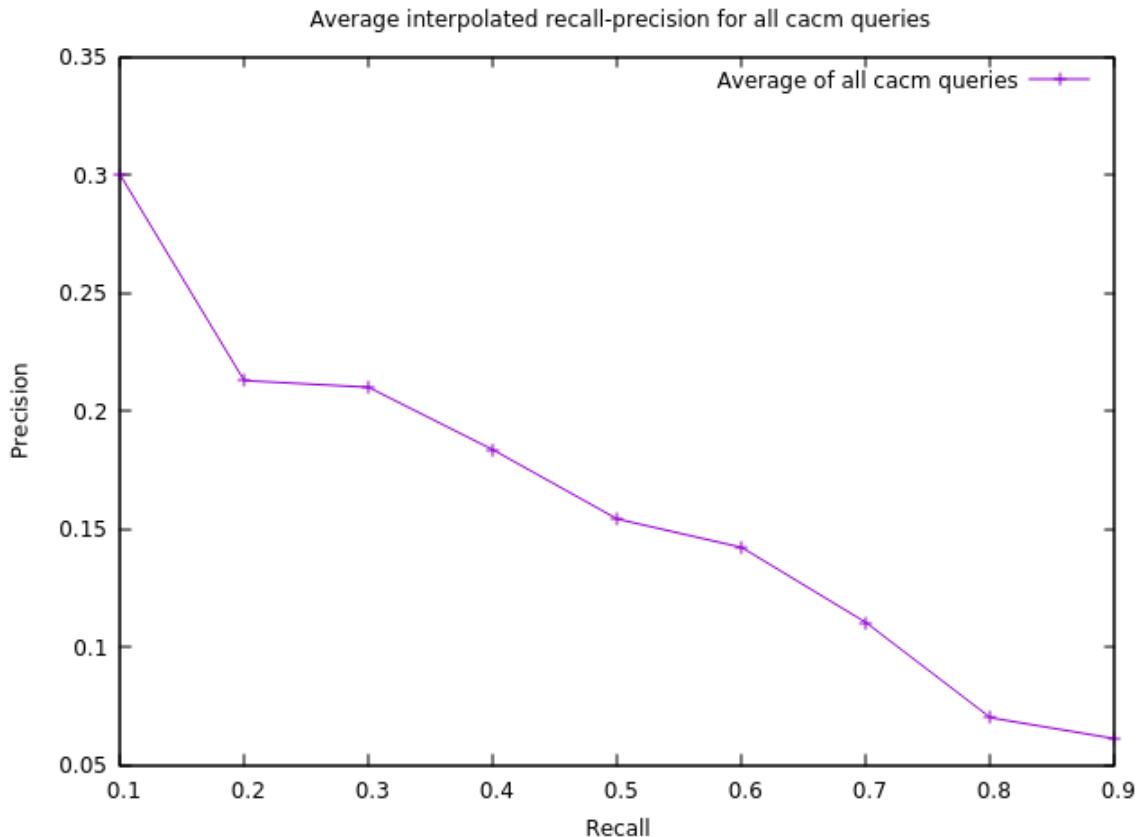


Figure 6: The average interpolated recall-precision graph for the entire query set

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
```

```

    if len(rel) == 0:
        return 0.0
    return rel_ret/len(rel)

65
def average_precision(res ,rel):
66    docs_found = 0
67    docs_avg = 0.0
68    doc_idx = 0
69    for doc1 in res:
70        doc_idx +=1
71        for doc0 in rel:
72            if doc0 in doc1:
73                docs_found+=1
74            docs_avg += float(docs_found)/doc_idx
75    if docs_found == 0:
76        return 0.0
77    return docs_avg/docs_found

79
def ndcg(res ,rel ,p):
80    idcg = 1.0
81    dcg = 0.0
82    for i in range(2,p+1):
83        idcg += 1/log(i ,2)
84    for doc0 in rel:
85        if res[0] in doc0:
86            dcg = 1.0
87    doc_idx =1
88    for doc1 in res[1:]:
89        doc_idx +=1
90        for doc0 in rel:
91            if doc0 in doc1:
92                dcg += 1/log(doc_idx ,2)
93    return dcg/idcg

95
parser = argparse.ArgumentParser('gmetrics')
96 parser.add_argument('-g','--gpath',help='Path to galago binary',required=False,
97                     default='.')
98 parser.add_argument('--xpath',help='Path to galago index',required=False,
99                     default='index/')
100 parser.add_argument('--inpath',help='Path to input',required=True)
101 parser.add_argument('--q','--qpath',help='Path to query xml file',required=False,
102                     default='cacm.query.xml')
103 # parser.add_argument('--query',help='Query id to extract relevance about',required=
104 #                     True)
105 parser.add_argument('--r','--rpath',help='Path to the .rel relevance file of galago',
106                     required=False,default='cacm.rel')
107 args = parser.parse_args();
108 g_bin = args.gpath + 'galago'
109 index_path = args.xpath
110 input_path = args.inpath
111 query_path = args.qpath
112 # query_idx = args.query
113 rel_path = args.rpath

114
115 if not os.path.exists(index_path):
116     p = subprocess.Popen(g_bin+" build --indexPath "+index_path +" --inputPath "+
117                          input_path , shell=True, stdout=subprocess.PIPE, stderr=subprocess.PIPE)
118     p.wait()
119
120 if not os.path.exists(query_path):
121     print "Query file not found!"
122     exit()
123
124 p = subprocess.Popen(g_bin+" batch-search --index="+index_path+" "+query_path , shell=
125                      True, stdout=subprocess.PIPE, stderr=subprocess.PIPE)

```

```

121 ranking = p.communicate()[0]
123 ranking_per_query = {}
124 lines = []
125 lines = ranking.split('\n')
126 top_ten_per_query = {}
127
128 for line in lines:
129     if line == '':
130         continue
131     qid, q, file, seq, rank, g = line.split(" ")
132     if int(qid) not in ranking_per_query:
133         ranking_per_query[int(qid)] = []
134         top_ten_per_query[int(qid)] = []
135     ranking_per_query[int(qid)].append(file)
136 for qid in ranking_per_query.keys():
137     for i in range(0, 10):
138         top_ten_per_query[int(qid)].append(ranking_per_query[qid][i])
139 relevant_docs_per_query = {}
140 for i in range(1, 63):
141     relevant_docs_per_query[i] = []
142 with open(rel_path, 'r') as f:
143     for line in f:
144         qid, q, file, isrel = line.split(" ")
145         # print qid, q, file, isrel
146         if int(qid) not in relevant_docs_per_query:
147             relevant_docs_per_query[int(qid)] = []
148         if int(isrel) == 1:
149             relevant_docs_per_query[int(qid)].append(file)
150 avg_prec = []
151 prec = {}
152 recall = {}
153 for i in relevant_docs_per_query.keys():
154     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])
156     # print "Query "+str(i)+": Average Precision --> "+str(average_precision(
157         ranking_per_query[i], relevant_docs_per_query[i]))
158
159 print "Mean Average Precision"
160 print sum(avg_prec)/len(avg_prec)
161 int_graph = {}
162 points = [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0]
163 for i in ranking_per_query.keys():
164     int_graph[i] = [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
165     for j in range(0, 10):
166         point = points[j]
167         for x in range(1, len(recall[i])):
168             if recall[i][x-1] < point and recall[i][x] > point:
169                 int_graph[i][j] = prec[i][x]
170
171 average_int_graph = [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
172 with open('cacm-graph', 'w+') as fp:
173     for j in range(0, 9):
174         for i in ranking_per_query.keys():
175             average_int_graph[j] += int_graph[i][j]
176         average_int_graph[j] = average_int_graph[j]/len(ranking_per_query.keys())
177         fp.write(str(((j+1)*0.1))+" "+str(average_int_graph[j])+"\n")
178
179 avg_ndcg5 = 0.0
180 avg_ndcg10 = 0.0
181 avg_prec10 = 0.0
182 avg_rprec = 0.0
183 for i in relevant_docs_per_query.keys():
184     avg_ndcg5+= ndcg(top_ten_per_query[i], relevant_docs_per_query[i], 5)
185     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])
186     avg_rprec +=r_precision(ranking_per_query[i],relevant_docs_per_query[i])
187 avg_ndcg5 = avg_ndcg5/ len(relevant_docs_per_query.keys())
188 avg_ndcg10 = avg_ndcg10/ len(relevant_docs_per_query.keys())
189 avg_prec10 = avg_prec10/len(relevant_docs_per_query.keys())
190 avg_rprec = avg_rprec/len(relevant_docs_per_query.keys())
191 print "Average NDCG@5"
192 print avg_ndcg5
193 print "Average NDCG@10"
194 print avg_ndcg10
195 print "Precision@10"
196 print avg_prec10
197 print "Average R-Precision"
198 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 -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
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:

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

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 -i cacm -g /home/poll/Desktop/galagosearch-1.04/bin/ --query 2
BPREF1
0.66666666666667
BPREF2
0.66666666666667

```

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

```

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

```

```

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

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

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)

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

```

```

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 "AGAIn"
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     return float(p)/(p+q)

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

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

```

```

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

```

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