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 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.

```
import subprocess
import argparse
import os
from math import log
def precision(res,rel):
  i = 0
  for doc0 in rel:
    for doc1 in res:
      if doc0 in doc1:
        i +=1
  return float(i)/len(res)

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
  return docs_avg/docs_found

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
def reciprocal(res,rel):
  doc_idx = 0
  for doc1 in res:
    doc_idx +=1
    for doc0 in rel:
      if doc0 in doc1:
        return float(1)/doc_idx

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/')
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')
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')
args = parser.parse_args();
g_bin = args.gpath + 'galago'
index_path = args.idxpath
input_path = args.inpath
query_path = args.qpath
query_idx = args.query
rel_path = args.rpath

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()

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

p = subprocess.Popen(g_bin+" batch-search --index="+index_path+" "+query_path, shell=
    True, stdout=subprocess.PIPE, stderr=subprocess.PIPE)
ranking =  p.communicate()[0]
ranking_per_query = {}
top_ten_per_query = {}
lines = []
lines = ranking.split('\n')
for line in lines:
  if line == '':
    continue
  qid ,q ,file ,seq ,rank ,g = line.split(" ")
  if int(qid) not in ranking_per_query:
    ranking_per_query[int(qid)]=[]
    top_ten_per_query[int(qid)]=[]
  ranking_per_query[int(qid)].append([file,rank])

for qid in ranking_per_query.keys():
  for i in range(0,10):
    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:
  for line in f:
93    qid ,q ,file ,isrel = line.split(" ")
    # print qid,q,file,isrel
95    if int(qid) not in relevant_docs_per_query :
      relevant_docs_per_query[int(qid)] = []
97    if int(isrel) == 1:
      relevant_docs_per_query[int(qid)].append(file)
99
print "Top ten documents from results:"
101 for doc in top_ten_per_query[int(query_idx)]:
  print doc[doc.rfind(input_path):].replace(input_path,""),
103 print
print "Relevant documents:"
105 for doc in relevant_docs_per_query[int(query_idx)]:
  print doc,
107 print
print "Precision@10:"
109 print precision(top_ten_per_query[int(query_idx)][:10],relevant_docs_per_query[int(
    query_idx)])
print "Average Precision:"
111 print average_precision(top_ten_per_query[int(query_idx)],relevant_docs_per_query[int
    (query_idx)])
print "NDCG@5"
113 print ndcg(top_ten_per_query[int(query_idx)],relevant_docs_per_query[int(query_idx)
    ],5)
print "NDCG@10"
115 print ndcg(top_ten_per_query[int(query_idx)],relevant_docs_per_query[int(query_idx)
    ],10)
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:
0.2
Average Precision:
0.416666666667
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.

```
import subprocess
import argparse
import os
from math import log

def recall_precision(res,rel,fp,fp2):
  recall = []
  precision = []
  rel_ret = 0.0
  i = 0
  for doc1 in res:
    i +=1
    for doc0 in rel:
      if doc0 in doc1:
        rel_ret +=1
    recall.append(rel_ret/len(rel))
    precision.append(rel_ret/i)
    fp.write(str(rel_ret/len(rel))+" "+str(rel_ret/i)+"\n")
  points = i-2
  while points>=0:
    if precision[points+1]>precision[points]:
      precision[points] = precision[points+1]
    points-=1

  for k in range(0,i):
    fp2.write(str(recall[k])+" "+str(precision[k])+"\n")

  return rel_ret/len(rel)

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/')
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')
parser.add_argument('--query1',help='1st Query id to graphs',required=True)
parser.add_argument('--query2',help='2nd Query id to graphs',required=True)
parser.add_argument('-r','--rpath',help='Path to the .rel relevance file of galago',
    required=False,default='cacm.rel')
args = parser.parse_args();
g_bin = args.gpath + 'galago'
index_path = args.idxpath
input_path = args.inpath
query_path = args.qpath
query1_idx = args.query1
query2_idx = args.query2
rel_path = args.rpath

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()

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

p = subprocess.Popen(g_bin+" batch-search --index="+index_path+" "+query_path, shell=
    True, stdout=subprocess.PIPE, stderr=subprocess.PIPE)
ranking =  p.communicate()[0]
ranking_per_query = {}
top_ten_per_query = {}
lines = []
```

```
lines = ranking.split('\n')
for line in lines:
  if line == '':
    continue
  qid ,q ,file ,seq ,rank ,g = line.split(" ")
  if int(qid) not in ranking_per_query:
    ranking_per_query[int(qid)]=[]
    top_ten_per_query[int(qid)]=[]
  ranking_per_query[int(qid)].append(file)
# print ranking_per_query.keys()
for qid in ranking_per_query.keys():
  for i in range(0,10):
    top_ten_per_query[int(qid)].append(ranking_per_query[int(qid)][i])

relevant_docs_per_query = {}
with open(rel_path,'r') as f:
  for line in f:
    qid ,q ,file,isrel = line.split(" ")
    # print qid,q,file,isrel
    if int(qid) not in relevant_docs_per_query :
      relevant_docs_per_query[int(qid)] = []
    if int(isrel) == 1:
      relevant_docs_per_query[int(qid)].append(file)

with open("graph1.txt",'w+') as fp:
  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)
with open("graph2.txt",'w+') as fp:
  with open("int_graph2.txt",'w+') as fp2:
    recall_precision(ranking_per_query[int(query2_idx)],relevant_docs_per_query[int(
    query2_idx)],fp,fp2)

int1x =[]
int1y =[]
int2x =[]
int2y =[]
int1x_final = []
int1y_final = []
int2x_final = []
int2y_final = []

points = [0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1.0]
with open("int_graph1.txt",'r+') as fp2:
  for line in fp2:
    x,y = line.split(" ")
    int1x.append(float(x))
    int1y.append(float(y))

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


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

```

```
  if len(int1x_final)>len(int2x_final):
    for i in range(0,len(int1x_final)-len(int2x_final)):
      int2y_final.apend(0)
  else:
    for i in range(0,len(int2x_final)-len(int1x_final)):
      int1y_final.apend(0)
print "Interpolated values for standard recall values"
for i in range(0,len(int1x_final)):
  print int1x_final[i],int1y_final[i],int2y_final[i]

with open("avg_int_graph.txt",'w+') as fp2:
  fp2.write(str(0)+" "+str((int1y[0]+int2y[0])/2)+"\n")

  for i in range(0,len(int1x_final)):
    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

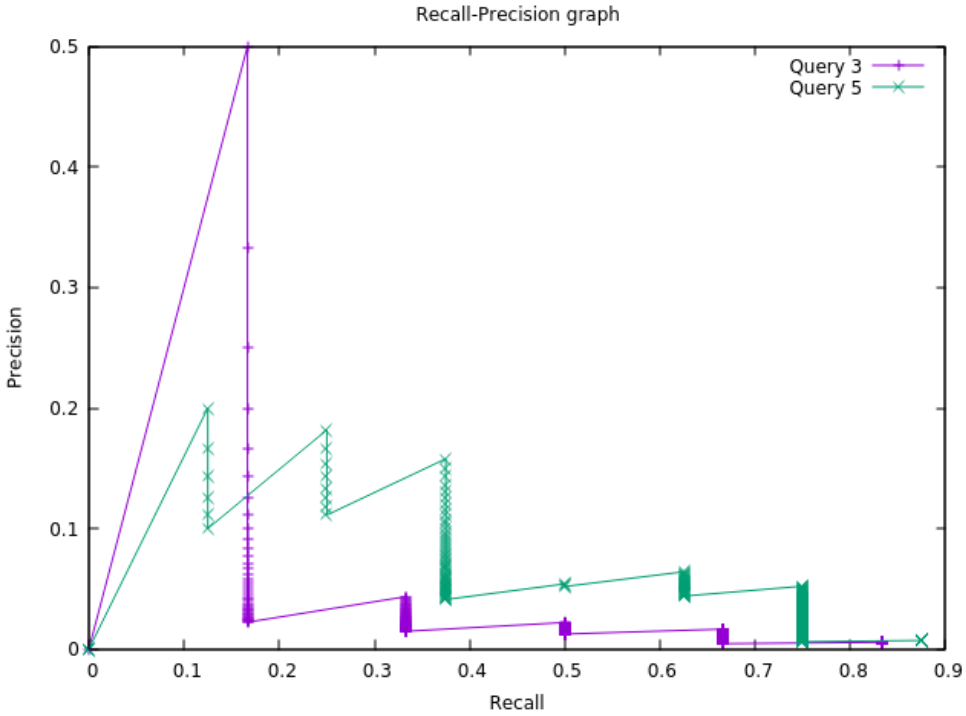

Figure 2: The uninterpolated recall-precision graphs for the two queries

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

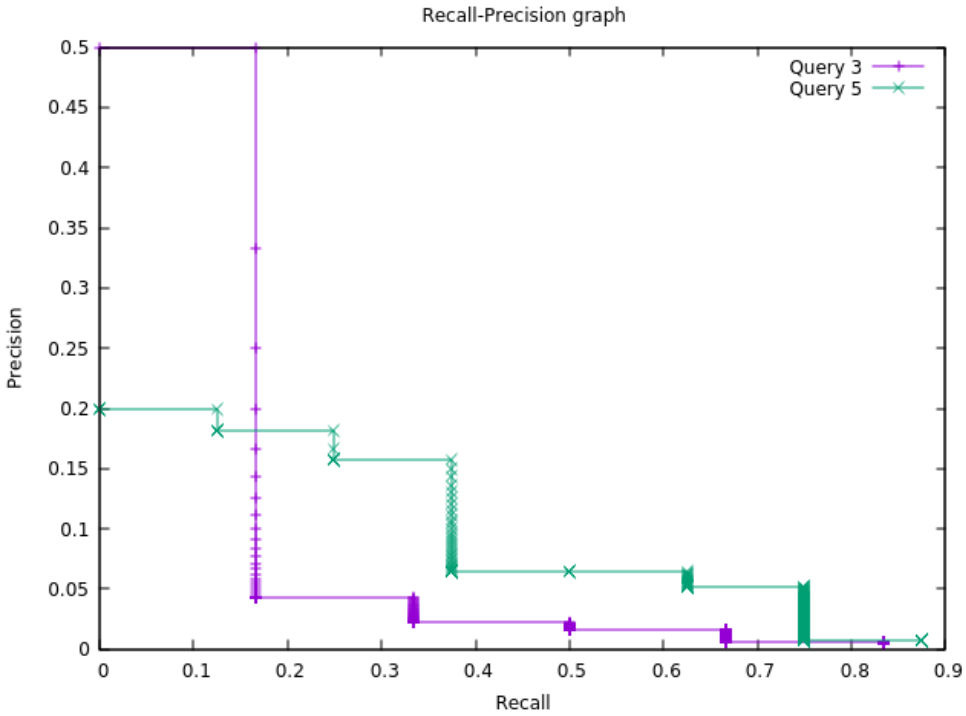

Figure 3: The interpolated recall-precision graphs for the two queries

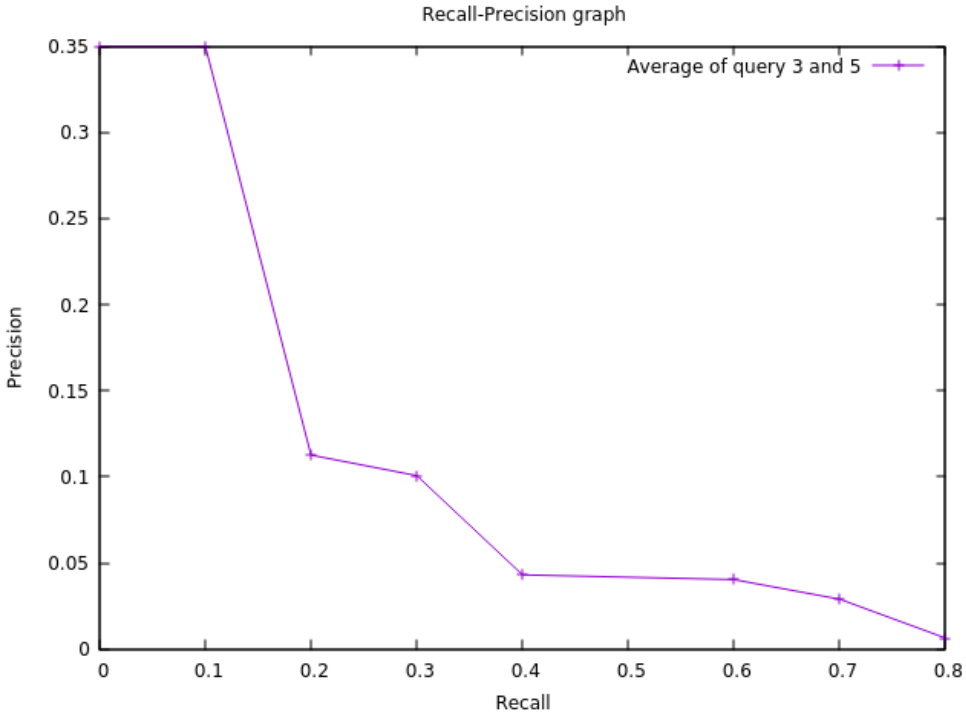

Figure 4: The interpolated average recall-precision graph for the two queries

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
import argparse
import os
from math import log
def recall_precision(res,rel):
  recall = []
  precision = []
  rel_ret = 0.0
  i = 0
  for doc1 in res:
    i +=1
    for doc0 in rel:
      if doc0 in doc1:
        rel_ret +=1
    if len(rel) ==0:
      recall.append(0.0)
      precision.append(0.0)
    else:
      recall.append(rel_ret/len(rel))
      precision.append(rel_ret/i)
  points = i-2
  while points>=0:
    if precision[points+1]>precision[points]:
      precision[points] = precision[points+1]
    points-=1
  return recall,precision

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

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

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

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

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/')
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')
# 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')
args = parser.parse_args();
g_bin = args.gpath + 'galago'
index_path = args.idxpath
input_path = args.inpath
query_path = args.qpath
# query_idx = args.query
rel_path = args.rpath


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()

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

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

ranking_per_query = {}
lines = []
lines = ranking.split('\n')
```

```
top_ten_per_query = {}

for line in lines:
  if line == '':
    continue
  qid ,q ,file ,seq ,rank ,g = line.split(" ")
  if int(qid) not in ranking_per_query:
    ranking_per_query[int(qid)]=[]
    top_ten_per_query[int(qid)]=[]
  ranking_per_query[int(qid)].append(file)
for qid in ranking_per_query.keys():
  for i in range(0,10):
    top_ten_per_query[int(qid)].append(ranking_per_query[qid][i])
relevant_docs_per_query = {}
for i in range(1,63):
  relevant_docs_per_query[i]=[]
with open(rel_path,'r') as f:
  for line in f:
    qid ,q ,file,isrel = line.split(" ")
    if int(qid) not in relevant_docs_per_query :
      relevant_docs_per_query[int(qid)] = []
    if int(isrel) == 1:
      relevant_docs_per_query[int(qid)].append(file)
avg_prec = []
prec = {}
recall = {}
for i in relevant_docs_per_query.keys():
  avg_prec.append(average_precision(ranking_per_query[i],relevant_docs_per_query[i]))
  recall[i],prec[i] = recall_precision(ranking_per_query[i],relevant_docs_per_query[i
    ])

print "Mean Average Precision "
print sum(avg_prec)/len(avg_prec)
int_graph = {}
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():
  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):
    point = points[j]
    for x in range(1,len(recall[i])):
      if recall[i][x-1]<point and recall[i][x]>point:
        int_graph[i][j] = prec[i][x]

average_int_graph = [0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0]
with open('cacm_graph','w+') as fp:
  for j in range(0,9):
    for i in ranking_per_query.keys():
      average_int_graph[j] += int_graph[i][j]
    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")

avg_ndcg5 = 0.0
avg_ndcg10 = 0.0
avg_prec10 = 0.0
for i in relevant_docs_per_query.keys():
  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)
  avg_prec10+=precision(top_ten_per_query[i],relevant_docs_per_query[i])
avg_ndcg5 = avg_ndcg5/ len(relevant_docs_per_query.keys())
avg_ndcg10 = avg_ndcg10/ len(relevant_docs_per_query.keys())
avg_prec10 = avg_prec10/len(relevant_docs_per_query.keys())
print "Average NDCG@5"
print avg_ndcg5
print "Average NDCG@10"
print avg_ndcg10
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

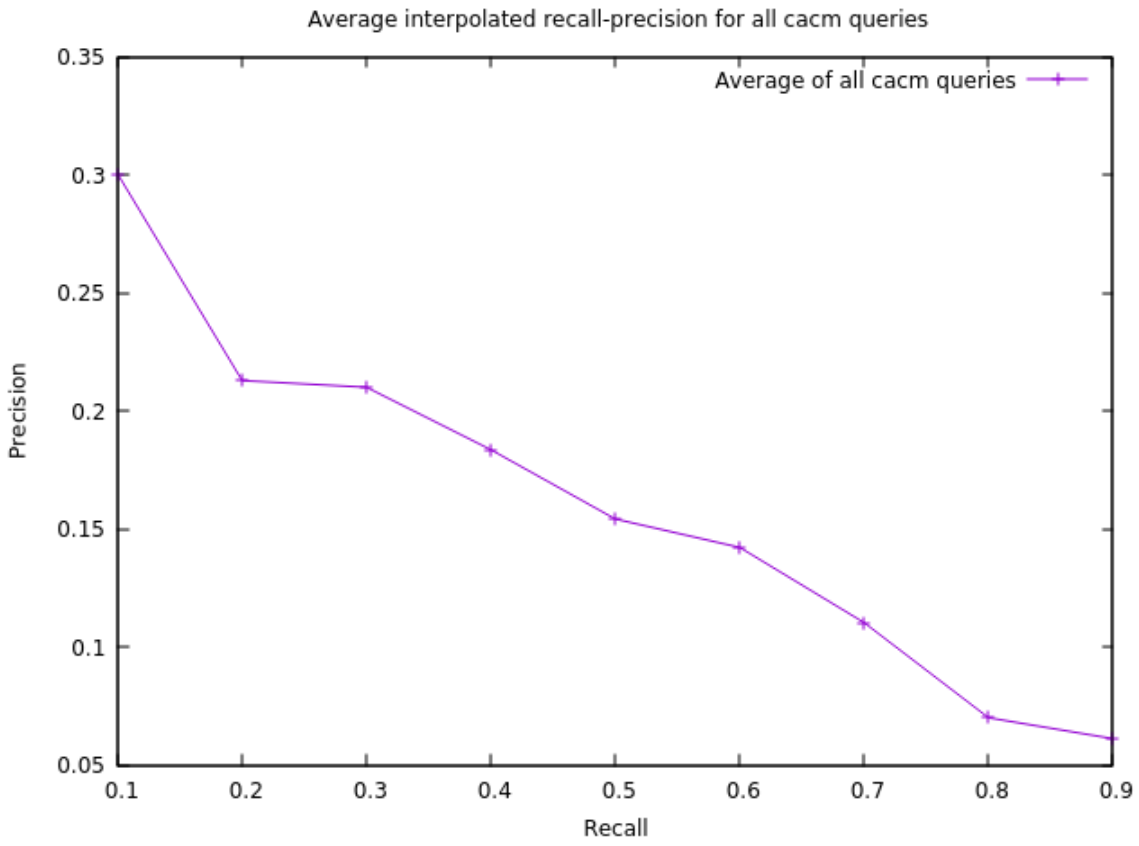

Figure 6: The average interpolated recall-precision graph for the entire query set

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.

```
import subprocess
import argparse
import os
from math import log
def recall_precision(res,rel):
  recall = []
  precision = []
  rel_ret = 0.0
  i = 0
  for doc1 in res:
    i +=1
    for doc0 in rel:
      if doc0 in doc1:
        rel_ret +=1
    if len(rel) ==0:
      recall.append(0.0)
      precision.append(0.0)
    else:
      recall.append(rel_ret/len(rel))
      precision.append(rel_ret/i)
  points = i-2
  while points>=0:
    if precision[points+1]>precision[points]:
      precision[points] = precision[points+1]
    points-=1
  return recall,precision

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

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

def r_precision(res,rel):
  i = 0
  rel_ret = 0
  for doc1 in res:
    if i == len(rel):
      break
    for doc0 in rel:
      if doc0 in doc1:
        rel_ret +=1
    i+=1
```

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

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

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

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/')
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')
# 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')
args = parser.parse_args();
g_bin = args.gpath + 'galago'
index_path = args.idxpath
input_path = args.inpath
query_path = args.qpath
# query_idx = args.query
rel_path = args.rpath


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()

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

p = subprocess.Popen(g_bin+" batch-search --index="+index_path+" "+query_path, shell=
    True, stdout=subprocess.PIPE, stderr=subprocess.PIPE)
```

```
ranking =  p.communicate()[0]

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

for line in lines:
  if line == '':
    continue
  qid ,q ,file ,seq ,rank ,g = line.split(" ")
  if int(qid) not in ranking_per_query:
    ranking_per_query[int(qid)]=[]
    top_ten_per_query[int(qid)]=[]
  ranking_per_query[int(qid)].append(file)
for qid in ranking_per_query.keys():
  for i in range(0,10):
    top_ten_per_query[int(qid)].append(ranking_per_query[qid][i])
relevant_docs_per_query = {}
for i in range(1,63):
  relevant_docs_per_query[i]=[]
with open(rel_path,'r') as f:
  for line in f:
    qid ,q ,file,isrel = line.split(" ")
    # print qid,q,file,isrel
    if int(qid) not in relevant_docs_per_query :
      relevant_docs_per_query[int(qid)] = []
    if int(isrel) == 1:
      relevant_docs_per_query[int(qid)].append(file)
avg_prec = []
prec = {}
recall = {}
for i in relevant_docs_per_query.keys():
  avg_prec.append(average_precision(ranking_per_query[i],relevant_docs_per_query[i]))
  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]))

print "Mean Average Precision"
print sum(avg_prec)/len(avg_prec)
int_graph = {}
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():
  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):
    point = points[j]
    for x in range(1,len(recall[i])):
      if recall[i][x-1]<point and recall[i][x]>point:
        int_graph[i][j] = prec[i][x]

average_int_graph = [0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0]
with open('cacm_graph','w+') as fp:
  for j in range(0,9):
    for i in ranking_per_query.keys():
      average_int_graph[j] += int_graph[i][j]
    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")

avg_ndcg5 = 0.0
avg_ndcg10 = 0.0
avg_prec10 = 0.0
avg_rprec = 0.0
for i in relevant_docs_per_query.keys():
  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)
```

```
    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])
avg_ndcg5 = avg_ndcg5/ len(relevant_docs_per_query.keys())
avg_ndcg10 = avg_ndcg10/ len(relevant_docs_per_query.keys())
avg_prec10 = avg_prec10/len(relevant_docs_per_query.keys())
avg_rprec = avg_rprec/len(relevant_docs_per_query.keys())
print "Average NDCG@5"
print avg_ndcg5
print "Average NDCG@10"
print avg_ndcg10
print "Precision@10"
print avg_prec10
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 -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:

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

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

```
  precision = []
  rel_ret = 0.0
  i = 0
  for doc1 in res:
    i +=1
    for doc0 in rel:
      if doc0 in doc1:
        rel_ret +=1
    if len(rel) ==0:
      recall.append(0.0)
      precision.append(0.0)
    else:
      recall.append(rel_ret/len(rel))
      precision.append(rel_ret/i)
  points = i-2
  while points>=0:
    if precision[points+1]>precision[points]:
      precision[points] = precision[points+1]
    points-=1
  return recall,precision

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

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

def bpref1(res,rel):
  result = 0.0
  found = False
  R = len(rel)
  R_non_relevant = []
  i = 0
  for doc1 in res:
    found = False
    for doc0 in rel:
      if doc0 in doc1:
        found = True
    if not found:
      i+=1
    if i> R:
      break
    R_non_relevant.append(doc1)
  # print len(R_non_relevant)
  for doc0 in rel:
    i =0
    Ndr = R
    # print "AGAIn"
```

```
    for doc1 in R_non_relevant:
      if doc0 in doc1:
        Ndr = i
      i +=1
    result += (1.0-float(Ndr)/R)
  return 1.00/R*result

def bpref2(res,rel):
  result = 0.0
  found = False
  R = len(rel)
  R_non_relevant = []
  i = 0
  for doc1 in res:
    found = False
    for doc0 in rel:
      if doc0 in doc1:
        found = True
    if not found:
      i+=1
    if i> R:
      break
    R_non_relevant.append(doc1)
  # print len(R_non_relevant)
  p = 0.0
  q = 0.0
  for doc0 in rel:
    i =0
    found = False
    # print "AGAIn"
    for doc1 in R_non_relevant:
      if doc0 in doc1:
        found = True
        q += i
        p += R-i
      i+=1
    if not found:
      q+=R

  return float(p)/(p+q)


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/')
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')
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')
args = parser.parse_args();
g_bin = args.gpath + 'galago'
index_path = args.idxpath
input_path = args.inpath
query_path = args.qpath
query_idx = int(args.query)
rel_path = args.rpath


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()

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

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

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

for line in lines:
  if line == '':
    continue
  qid ,q ,file ,seq ,rank ,g = line.split(" ")
  if int(qid) not in ranking_per_query:
    ranking_per_query[int(qid)]=[]
    top_ten_per_query[int(qid)]=[]
  ranking_per_query[int(qid)].append(file)
for qid in ranking_per_query.keys():
  for i in range(0,10):
    top_ten_per_query[int(qid)].append(ranking_per_query[qid][i])
relevant_docs_per_query = {}
for i in range(1,63):
  relevant_docs_per_query[i]=[]
with open(rel_path,'r') as f:
  for line in f:
    qid ,q ,file,isrel = line.split(" ")
    # print qid,q,file,isrel
    if int(qid) not in relevant_docs_per_query :
      relevant_docs_per_query[int(qid)] = []
    if int(isrel) == 1:
      relevant_docs_per_query[int(qid)].append(file)

bp1=bpref1(ranking_per_query[query_idx],relevant_docs_per_query[query_idx])
bp2=bpref2(ranking_per_query[query_idx],relevant_docs_per_query[query_idx])
print "BPREF1"
print bp1
print "BPREF2"
print bp2
```

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