Assignment 4

Fall 2016
CS834 Introduction to Information Retrieval
Dr. Michael Nelson

Mathew Chaney

December 5, 2016

Contents

Listings

List of Figures

List of Tables

1 Question 8.3

1.1 Question

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.

1.2 Approach

Galago version 3.10 was first downloaded from the Project Lemur Source Forge website, which can be found at the following URL: https://sourceforge.net/projects/lemur/files/lemur/galago-3.10/. The CACM document corpus was downloaded from the textbook's website, found here: http://www.search-engines-book.com/collections/. Galago was used to create an index of the CACM corpus and to run as a server to respond to queries on that index.

The `getrel.py` and `q83.py` scripts (found in Listings 3 and 4, respectively) was created to issue queries to the Galago search server using the Python Requests library [1]. The HTML responses were then parsed using the Python Beautiful Soup library [2], where the CACM document identifiers were extracted for use in calculating the different evaluation scores for the Galago ranking.

The query used was from the CACM query set, number 10, and only the first 1000 retrieved documents were considered when calculating all scores for this experiment.

1.2.1 Initial Precision and Recall Calculations

Precision and Recall were calculated with the following equations:

$$Recall = \frac{\mid A \cap B \mid}{\mid A \mid}$$

$$Precision = \frac{\mid A \cap B \mid}{\mid B \mid}$$

In these equations, A is the relevant set of documents for the query, and B is the set of retrieved documents.

1.2.2 Calculating Precision at Specific Rankings

A list of precision values was created by calculating the cumulative precision at each document ranking with the set of retrieved documents up to that ranking.

1.2.3 Calculating Average Precision

Average precision was calculated by adding the precision at each retrieval ranking position for documents which are part of $A \cap B$, or the set of retrieved documents that are relevant, and then dividing by the size of that set to obtain the average. This can also be described as the area under the precision-recall curve, which can be expressed as the following summation:

$$AveP = \sum_{k=1}^{n} P(k)\Delta r(k)$$

where k is the rank in the sequence of retrieved documents, n is the number of retrieved documents, $P(k)$ is the precision at cut-off k in the list, and $\Delta r(k)$ is the change in recall from items $k-1$ to k.

1.2.4 Calculating Normalized Discounted Cumulative Gain (NDCG)

First, discounted cumulative gain at rank p (DCG_p) was calculated with the following formula:

$$DCG_p = rel_1 + \sum_{i=2}^{p} \frac{rel_i}{log_2 i}$$

The ideal discounted cumulative gain at rank p ($IDCG_p$) is a simple series, expressed as:

$$IDCG_p = 1 + \sum_{i=2}^{p} \frac{1}{log_2 i}$$

Finally, normalized discounted cumulative gain at rank p ($NDCG_p$) is expressed as:

$$NDCG_p = \frac{DCG_p}{IDCG_p}$$

with rel_i being the relevancy for document i in the retrieval ranking. For this experiment, this value is either 0 or 1.

1.2.5 Calculating Reciprocal Rank

Reciprocal rank is defined as the reciprocal of the rank at which the first relevant document is found, so if the 3^{rd} document in the retrieval ranking list is the first relevant document, the reciprocal rank is $\frac{1}{3}$.

1.3 Results

After building the index, CACM query 10 was processed by the `getrel.py` script, the output of which can be found in Listing 1. This script calculates all the values shown in Table 1, which are all of the required values for the question.

```
[mchaney@mchaney-l getrel]$ python q83.py -q 10 -n 10000
query 10
query: parallel languages  languages for parallel computation
precision: 0.0190816935003
recall: 0.914285714286
precision @10: 0.9
NDCG @5: 1.0
NDCG @10: 0.942709999032
avg precision: 0.5922383982
reciprocal rank: 1.0
```

Listing 1: Output from running the getrel.py script for queries 1 and 10 from the CACM collection.

Query #	Avg. Prec.	NDCG @5	NDCG @10	Prec. @10	Recip. Rank
10	0.5922383982	1.0	0.942709999032	0.9	1.0

Table 1: Calculations for CACM query 10 from all retrieved documents.

2 Question 8.4

2.1 Question

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

2.2 Approach

Using the `getrel.py`, `q84.py` and `graphs.R` scripts, found in Listings 3, 5 and 9 were created to complete this task.

2.3 Results

2.3.1 Uninterpolated Recall-Precision Graph

The uninterpolated recall-precision graph is shown in Figure 1.

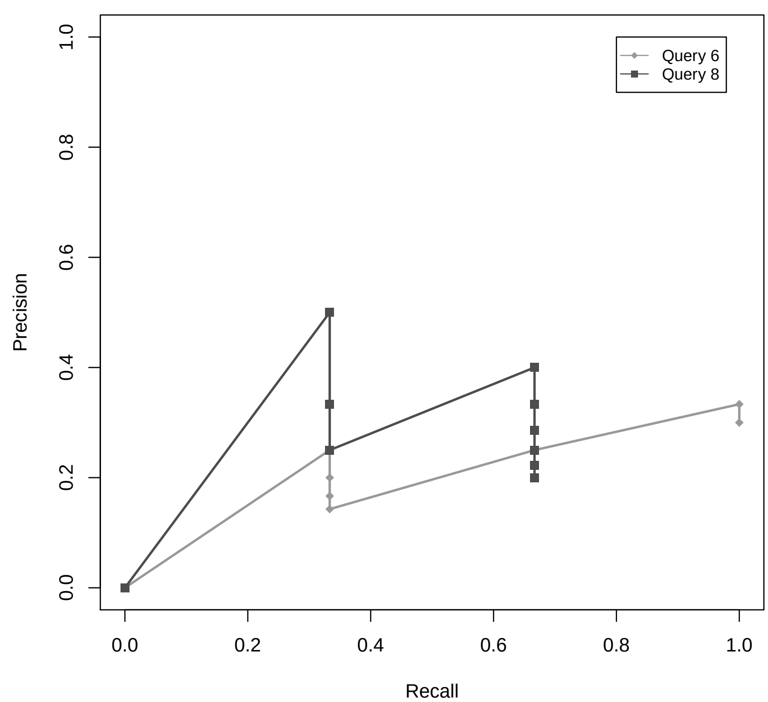

Figure 1: Uninterpolated Recall-Precision Graph for CACM Queries 6 and 8.

2.3.2 Interpolated Precision

The graph for the interpolated precision at standard recall values is shown in Figure 2 and the table of the values for each query, including the averages, is shown in Table 2.

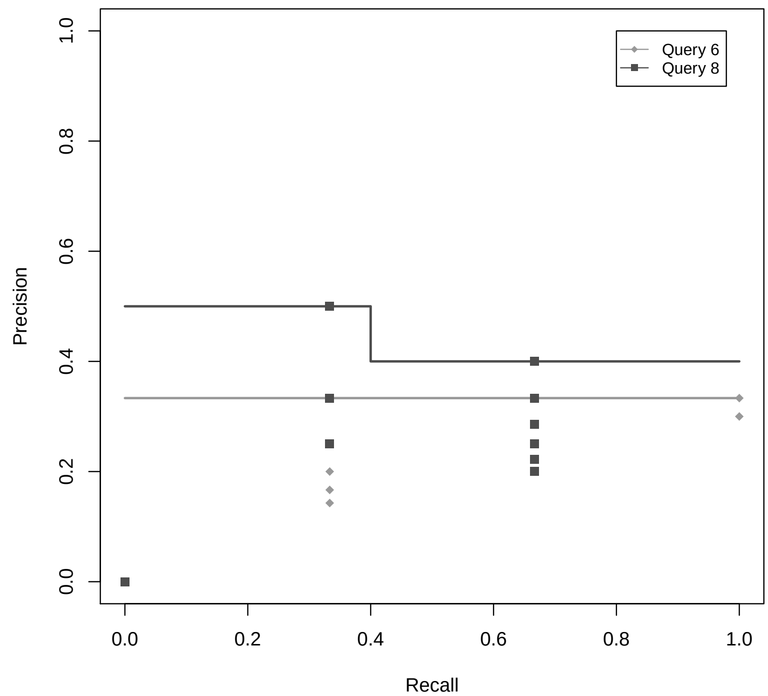

Figure 2: Graph of interpolated precision at standard recall values for CACM queries 6 and 8.

Recall	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Query 6	0.333	0.333	0.333	0.333	0.333	0.333	0.333	0.333	0.333	0.333	0.333
Query 8	0.5	0.5	0.5	0.5	0.4	0.4	0.4	0.4	0.4	0.4	0.4
Average	0.417	0.417	0.417	0.417	0.367	0.367	0.367	0.367	0.367	0.367	0.367

Table 2: Interpolated precision at standard recall values for CACM queries 6 and 8.

2.3.3 Average Interpolated Precision

The graph of the average interpolated precision at standard recall values for CACM queries 6 and 8 can be found in Figure 3.

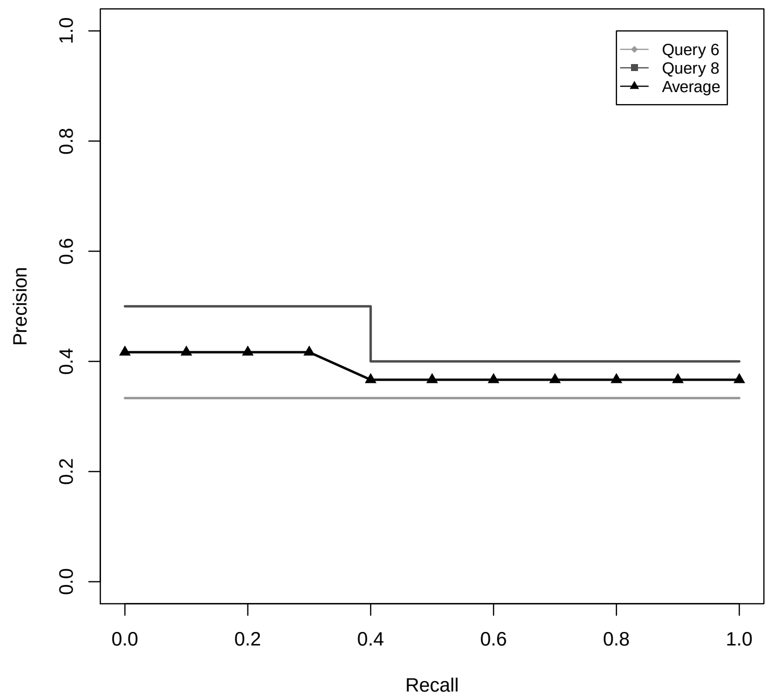

Figure 3: Average interpolated recall-precision graph for CACM Queries 6 and 8.

3 Question 8.5

3.1 Question

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

3.2 Approach

The `getrel.py` and `q85.py` scripts, found in Listings 3 and 6, were used to complete this question.

3.3 Results

The output from running the `q85.py` script can be found in Listing **??**.

Using only queries for which relevance judgments exist MAP, NDCG @5 and 10, and the precision @ 10 were calculated. The results can be found in Table 3. The generated recall-precision graph for the entire query set can be found in Figure 4.

MAP	NDCG @5	NDCG @10	Prec. @10
0.339552098123	0.461648777763	0.381724764912	0.317647058824

Table 3: Calculations for all CACM queries from all retrieved documents.

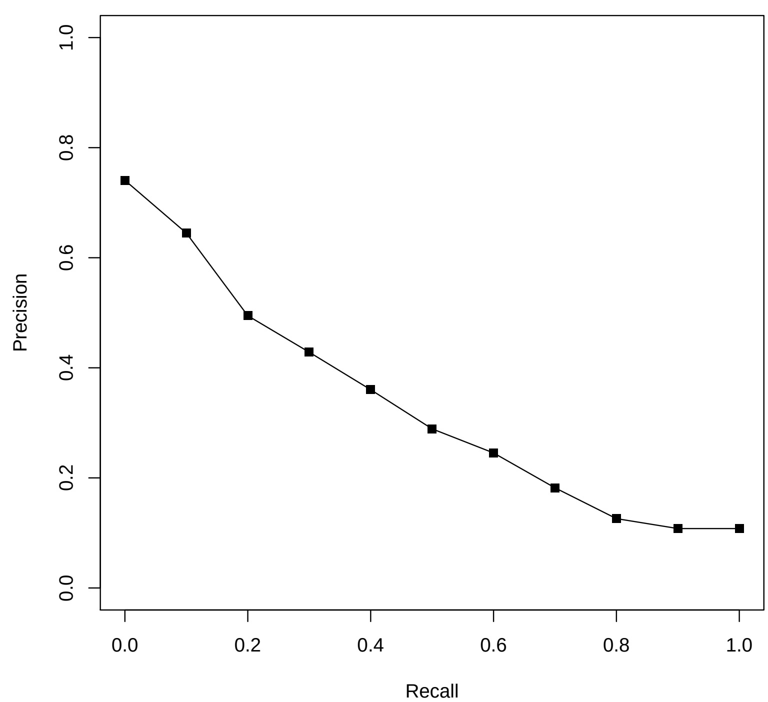

Figure 4: Recall-precision graph for all CACM Queries.

4 Question 8.7

4.1 Question

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.

4.2 Approach

The `getrel.py` and `q87.py` scripts were used to complete this exercise. They can be found in Listings 3 and 7. The script was run over the entire document set, which will minimize precision and maximize recall. This will be taken into consideration when comparing to the R-precision score.

4.3 Results

The output of running the `q87.py` script can be found in Listing 2.

```
[mchaney@mchaney-1 getrel]$ python q87.py -n 3204 -q 10
query 10
query: parallel languages  languages for parallel computation
relevant: 35
retrieved: 1677
precision: 0.0190816935003
recall: 0.914285714286
precision @10: 0.9
NDCG @5: 1.0
NDCG @10: 0.942709999032
avg precision: 0.5922383982
reciprocal rank: 1.0
|R|: 35
R-precision: 0.555555555556
```

Listing 2: Output of q87.py for query 10.

4.3.1 Comparison

For query 10 the R-precision score was 0.55, which is very close to the average precision over the entire document set. This rather simplistic comparison is one mark towards R-precision being a decently reliable measure of performance for a ranking. R-precision seems to be somewhat of an intuitive normalization of the precision. This is because the sample size of documents is bounded by the size of the relevant set of documents. This measure will favor rankings that push more relevant documents into higher ranks, which seems like a strong measure.

The only problem with this measure is if the relevant set is very small the results of the calculation may vary wildly. This may make the measure inappropriate for a targeted search because it will basically be 0 or 1, which isn't useful for comparing rankings.

5 Question

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

5.1 Approach

The first version of BPREF found in the text book is defined as follows:

$$BPREF = \frac{1}{R}\sum_{d_r}(1 - \frac{N_{d_r}}{R})$$

Where d_r is a relevant document, N_{d_r} gives the number of non-relevant documents that are ranked higher than document d_r and R is the size of the set of all relevant documents for the given query.

The second form is:

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

Where P is the number of preferences that agree and Q is the number of preferences that disagree.

The scripts `getrel.py` and `q89.py`, found in Listings 3 and 8 were used to complete this exercise.

5.2 Results

6 Appendix

6.1 Code listings

```python
import argparse
import re
import requests
import sys
import xmltodict
import numpy as np
from math import log
from bs4 import BeautifulSoup

def parseargs():
    parser = argparse.ArgumentParser()
    parser.add_argument('-p', '--port', type=int, default=42247, help='galago server port')
    parser.add_argument('-q', '--qnum', nargs='+', type=int, default=[6, 8], help='query number')
    parser.add_argument('-n', type=int, default=10, help='number of retrieval pages')
    return parser.parse_args()

args = parseargs()

def buildrel():
    rel = {}
    for line in open('cacm.rel').readlines():
        q, _, doc, _ = line.split()
        if q not in rel:
            rel[q] = []
        rel[q].append(int(doc.split('-')[1]))
    return rel

def buildqueries():
    with open('cacm.query.xml') as fd:
        return xmltodict.parse(fd.read())

REL = buildrel()
QUERIES = buildqueries()
RE = re.compile('/home/mchaney/workspace/edu/cs834-f16/assignments/assignment4/code/cacm/docs/CACM-([\d]+).html')
ID = {'id':'result'}
URL = 'http://0.0.0.0:{0}/search'
QUERY1 = 'what articles exist which deal with tss time sharing system an operating system for ibm computers'
PDICT = {'q': QUERY1, 'start': 0, 'n': args.n}

def query(qstr, port=args.port):
    PDICT['q'] = qstr
    PDICT['n'] = args.n
    res = requests.get(URL.format(port), params=PDICT)
    if not res.ok:
        return None
    soup = BeautifulSoup(res.text, 'html.parser')
    return [int(RE.match(href.text).groups()[0]) for href in soup.select("#result a")]

def recall(rel, retr):
    relset = set(rel)
    retrset = set(retr)
    return float(len(relset.intersection(retrset))) / len(relset)

def precision(rel, retr):
    relset = set(rel)
    retrset = set(retr)
    return float(len(relset.intersection(retrset))) / len(retrset)

def run(rel, retr, func):
    rr = []
    for i in range(1, len(retr)+1):
        rr.append(func(rel, retr[:i]))
    return rr

def avg(rel, retr, func):
    prun = run(rel, retr, func)
    res = []
    for i in range(len(retr)):
```

```
        if retr[i] in rel:
            res.append(prun[i])
    if len(res) == 0:
        return 0.0
    return float(sum(res))/len(res)

def getrel(rel, retr, i):
    return 1 if retr[i] in rel else 0

def DCG(rel, retr, p):
    sum = 0
    for i in range(2, p+1):
        sum += float(getrel(rel, retr, i-1)) / log(i, 2)
    return getrel(rel, retr, 0) + sum

def IDCG(p):
    sum = 0
    for i in range(2, p+1):
        sum += 1 / log(i, 2)
    return 1 + sum

def NDCG(rel, retr, p):
    dcg = DCG(rel, retr, p)
    idcg = IDCG(p)
    return dcg / idcg

def reciprank(rel, retr):
    for i in range(1, len(retr)+1):
        if retr[i-1] in rel:
            return 1.0 / i
    return 0.0

def ipr(rrun, prun):
    res = []
    for i in np.arange(0, 1.1, .1):
        for j in range(len(rrun)):
            if rrun[j] > i:
                idx = j
                break
        res.append(max(prun[idx:]))
    return np.arange(0, 1.1, 0.1), res

def rprecision(rel, prun):
    return prun[len(rel)]

def getquery(qnum):
    return QUERIES['parameters']['query'][qnum-1]['text']

def process(qnum):
    qstr = getquery(qnum)
    retr = query(qstr)
    if str(qnum) not in REL:
        return [None]*12
    rel = REL[str(qnum)]
    prun = run(rel, retr, precision)
    rrun = run(rel, retr, recall)
    prec = precision(rel, retr)
    rec = recall(rel, retr)
    avgprec = avg(rel, retr, precision)
    ndcg5 = NDCG(rel, retr, 5)
    ndcg10 = NDCG(rel, retr, 10)
    recip = reciprank(rel, retr)
    return qnum, qstr, retr, rel, prun, rrun, prec, rec, ndcg5, ndcg10, avgprec, recip

def printresults(qnum, qstr, retr, rel, prun, rrun, prec, rec, ndcg5, ndcg10, avgprec, recip):
    if not qnum:
        return
    print 'query {0}'.format(qnum)
    print 'query: {0}'.format(qstr)
    if args.n == 10:
        print 'relevant: {0}'.format(rel)
        print 'retrieved: {0}'.format(retr)
        print 'p-run: {0}'.format(prun)
        print 'r-run: {0}'.format(rrun)
    else:
        print 'relevant: {}'.format(len(rel))
```

```
        print 'retrieved: {}'.format(len(retr))
    print 'precision: {0}'.format(prec)
    print 'recall: {0}'.format(rec)
    print 'precision @10: {0}'.format(prun[9])
    print 'NDCG @5: {0}'.format(ndcg5)
    print 'NDCG @10: {0}'.format(ndcg10)
    print 'avg precision: {0}'.format(avgprec)
    print 'reciprocal rank: {0}'.format(recip)

def printdata(rrun, prun, fname):
    with open(fname, 'w') as fd:
        zipped = zip(rrun, prun)
        for z in zipped:
            fd.write('{0}\t{1}\n'.format(z[0], z[1]))

if __name__ == '__main__':
    for qnum in args.qnum:
        printresults(*process(qnum))
```

Listing 3: getrel.py

```
from getrel import *

TABLE = """\\begin{{table}}[h!]
\\centering
\\begin{{tabular}}{{ | c | c | c | c | c | c | }}
\\hline
Query \# & Avg. Prec. & NDCG @5 & NDCG @10 & Prec. @10 & Recip. Rank \\\\
\\hline
{0} & {1} & {2} & {3} & {4} & {5} \\\\
\\hline
\\end{{tabular}}
\\caption{{Calculations for CACM query {6} from all retrieved documents.}}
\\label{{tab:q83}}
\\end{{table}}
"""

def printtab(qnum, qstr, retr, rel, prun, rrun, prec, rec, ndcg5, ndcg10, avgprec, recip,
    rprec):
    fname = 'query{0}.tab'.format(qnum)
    with open(fname, 'w') as fd:
        fd.write(TABLE.format(qnum, avgprec, ndcg5, ndcg10, prun[9], recip, qnum))

for qnum in args.qnum:
    results = process(qnum)
    printresults(*results)
    printtab(*results)
```

Listing 4: q83.py

```
from getrel import *

HEAD = """\\begin{table}[H]
\\centering
\\begin{tabular}{ l l l l l l l l l l l l }
Recall & 0.0 & 0.1 & 0.2 & 0.3 & 0.4 & 0.5 & 0.6 & 0.7 & 0.8 & 0.9 & 1.0 \\\\
\\cline{2-12}
"""

ROW = """{0} & {1:.3g} & {2:.3g} & {3:.3g} & {4:.3g} & {5:.3g} & {6:.3g} & {7:.3g} & {8:.3g}
    & {9:.3g} & {10:.3g} & {11:.3g} \\\\
\\cline{{2-12}}
"""

TAIL = """\\end{tabular}
\\caption{.}
\\label{tab:ipr68}
\\end{table}
"""

def printtable(iprl):
    with open('iptab.tex', 'w') as fd:
        fd.write(HEAD)
        for iprun, qnum in iprl:
```

```
            fd.write(ROW.format('Query {0}'.format(qnum), *iprun))
        avg = [float(sum(col))/len(col) for col in zip(*[col[0] for col in iprl])]
        printdata(np.arange(0, 1.1, .1), avg, 'avg.dat')
        fd.write(ROW.format('Average', *avg))
        fd.write(TAIL)

iprl = []
for qnum in args.qnum:
    results = process(qnum)
    printresults(*results)
    qnum, qstr, retr, rel, prun, rrun, prec, rec, ndcg5, ndcg10, avgprec, recip, rprec =
        results
    printdata(rrun, prun, 'urpg{0}.dat'.format(qnum))
    irrun, iprun = ipr(rrun, prun)
    printdata(irrun, iprun, 'ipr{0}.dat'.format(qnum))
    iprl.append((iprun, qnum))
printtable(iprl)
```

Listing 5: q84.py

```
from getrel import *

TABLE = """\\begin{{table}}[h!]
\\centering
\\begin{{tabular}}{{ | c | c | c | c | }}
\\hline
MAP & NDCG @5 & NDCG @10 & Prec. @10 \\\\
\\hline
{0} & {1} & {2} & {3} \\\\
\\hline
\\end{{tabular}}
\\caption{{Calculations for all CACM queries from all retrieved documents.}}
\\label{{tab:q85}}
\\end{{table}}
"""

def printtab(fname, cacmmap, avgndcg5, avgndcg10, avgprec10):
    with open(fname, 'w') as fd:
        fd.write(TABLE.format(cacmmap, avgndcg5, avgndcg10, avgprec10))

netavg = []
iprl = []
ndcg5lst = []
ndcg10lst = []
prunlst = []
for i in range(1, 64):
    qnum, qstr, retr, rel, prun, rrun, prec, rec, ndcg5, ndcg10, avgprec, recip = process(i)
    if avgprec:
        netavg.append(avgprec)
        prunlst.append(prun[9])
        irrun, iprun = ipr(rrun, prun)
        iprl.append((iprun, qnum))
        ndcg5lst.append(ndcg5)
        ndcg10lst.append(ndcg10)

# MAP
cacmmap = float(sum(netavg)) / len(netavg)
print 'average precision: {0}'.format(cacmmap)

# Recall-Precision
netavgrpg = [float(sum(col))/len(col) for col in zip(*[col[0] for col in iprl])]
printdata(np.arange(0, 1.1, .1), netavgrpg, 'avgq85.dat')

# NDCG @ 5 and 10
avgndcg5 = float(sum(ndcg5lst))/len(ndcg5lst)
avgndcg10 = float(sum(ndcg10lst))/len(ndcg10lst)
print 'NDCG @5 : {0}'.format(avgndcg5)
print 'NDCG @10: {0}'.format(avgndcg10)

# precision at 10
avgprec10 = float(sum(prunlst))/len(prunlst)
print 'Precision @10: {0}'.format(avgprec10)

printtab('q85.tab', cacmmap, avgndcg5, avgndcg10, avgprec10)
```

Listing 6: q85.py

```
from getrel import *

for qnum in args.qnum:
    qnum, qstr, retr, rel, prun, rrun, prec, rec, ndcg5, ndcg10, avgprec, recip = process(
        qnum)
    printresults(qnum, qstr, retr, rel, prun, rrun, prec, rec, ndcg5, ndcg10, avgprec, recip
        )
    rprec = rprecision(rel, prun)
    print '|R|: {}'.format(len(rel))
    print 'R-precision: {}'.format(rprec)
```

Listing 7: q87.py

```
from getrel import *
```

Listing 8: q89.py

```
plotone <- function(data, fname) {
    pdf(fname)
    plot(data, type='o', pch=15, ylim=c(0,1), xlim=c(0,1),
        ylab="Precision", xlab="Recall")
    dev.off()
}
urpgraph <- function(d1, d2, fname) {
    pdf(fname)
    plot(d1, lwd=2, type='o', pch=18, ylim=c(0,1), xlim=c(0,1), col="gray60",
        ylab="Precision", xlab="Recall")
    lines(d2, lwd=2, type="o", pch=15, col="gray30")
    legend(0.8, 1, c('Query 6', 'Query 8'), cex=0.8,
        col=c('gray60', 'gray30'), lty=c(1,1), pch=c(18,15))
    dev.off()
}
iprgraph <- function(d1, d2, id1, id2, fname) {
    pdf(fname)
    plot(d1, lwd=2, type="p", pch=18, ylim=c(0,1), xlim=c(0,1), col="gray60",
        ylab="Precision", xlab="Recall")
    lines(id1, lwd=2, type="s", col="gray60")
    lines(d2, lwd=2, type="p", pch=15, col="gray30")
    lines(id2, lwd=2, type="s", col="gray30")
    legend(0.8, 1, c('Query 6', 'Query 8'), cex=0.8,
        col=c('gray60', 'gray30'), lty=c(1,1), pch=c(18,15))
    dev.off()
}
aipgraph <- function(avg, id1, id2, fname) {
    pdf(fname)
    plot(avg, lwd=2, type="l", ylim=c(0,1), xlim=c(0,1), col="black",
        ylab="Precision", xlab="Recall")
    lines(avg, lwd=2, type="p", pch=17, col="black")
    lines(id1, lwd=2, type="s", col="gray60")
    lines(id2, lwd=2, type="s", col="gray30")
    legend(0.8, 1, c('Query 6', 'Query 8', 'Average'), cex=0.8,
        col=c('gray60', 'gray30', 'black'), lty=c(1,1,1), pch=c(18,15,17))
    dev.off()
}

args = commandArgs(trailingOnly=TRUE)

d1 <- read.table(paste('urpg', args[1], '.dat', sep=''))
d2 <- read.table(paste('urpg', args[2], '.dat', sep=''))

plotone(d1, paste('urpg', args[1], '.pdf', sep=''))
plotone(d2, paste('urpg', args[2], '.pdf', sep=''))
urpgraph(d1, d2, paste('urpg', args[1], '', args[2], '.pdf', sep=''))

id1 <- read.table(paste('ipr', args[1], '.dat', sep=''))
id2 <- read.table(paste('ipr', args[2], '.dat', sep=''))
iprgraph(d1, d2, id1, id2, paste('ipr', args[1], '', args[2], '.pdf', sep=''))

```

```
avg <- read.table('avg.dat')
aipgraph(avg, id1, id2, paste('aipr', args[1], args[2], '.pdf', sep=''))

overallavg <- read.table('avgq85.dat')
plotone(overallavg, 'avgq85.pdf')
```

Listing 9: Script used to generate the recall-precision graphs

7 References

[1] Kenneth Reitz. Requests: HTTP for Humans. Available at http://docs.python-requests.org/en/master/. Accessed: 2016/09/20.

[2] Leonard Richardson. Beautiful Soup. Available at: https://www.crummy.com/software/beautifulsoup/. Accessed: 2016/09/20.