Assignment 4

Fall 2016
CS834 Introduction to Information Retrieval
Dr. Michael Nelson

Mathew Chaney

December 4, 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 q8.3.py -n 1000 -q 10
query 10
query: parallel languages  languages for parallel computation
precision: 0.027
recall: 0.771428571429
precision @10: 0.9
NDCG @5: 1.0
NDCG @10: 0.942709999032
avg precision: 0.697677898817
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	0.697677898817	1.0	0.942709999032	0.9	1.0

Table 1: Calculations for CACM query 10 from top 1000 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 6.

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

Listing 2: The run function

2.2.1 Generating the Uninterpolated Recall-Precision Graph

The generated graph is shown in Figure ??.

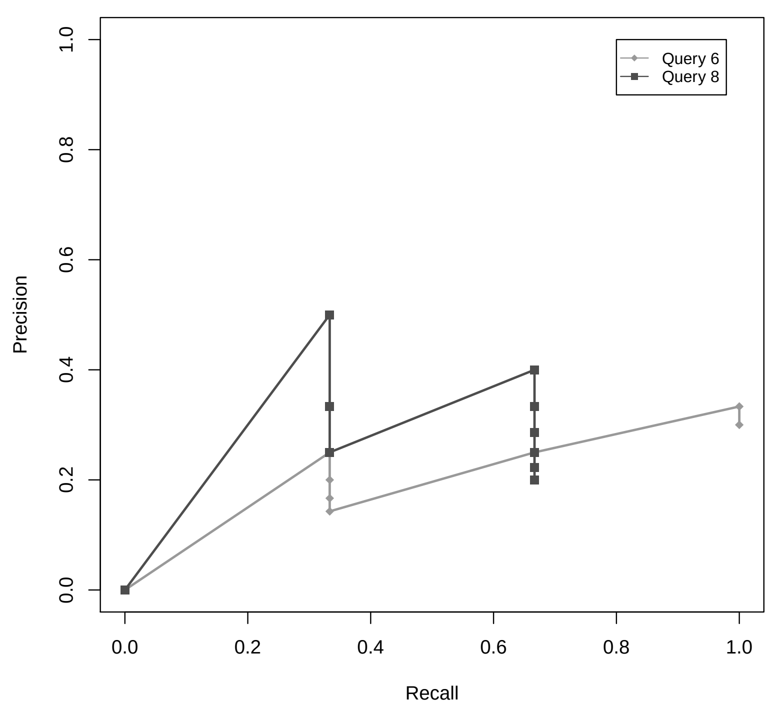

Figure 1: Uninterpolated Recall-Precision Graph for CACM Queries 6 and 8.

2.2.2 Generating the Table of Interpolated Precision Values

2.2.3 Generating the Average Interpolated Recall-Precision Graph

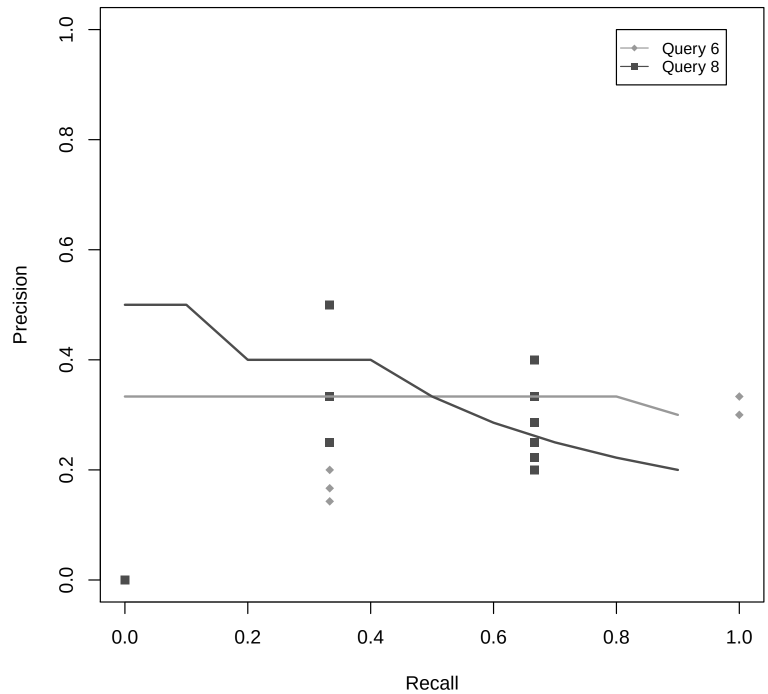

Figure 2: Average Interpolated Recall-Precision Graph for CACM Queries 6 and 8.

3 Appendix

3.1 Code listings

```python
import argparse
import re
import requests
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='the galago search server port')
    parser.add_argument('-q', '--qnum', type=int, default=10, help='the query number to use')
    parser.add_argument('-n', type=int, default=10, help='the number of results to retrieve')
    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, precision)
```

```
    res = []
    for i in range(len(retr)):
        if retr[i] in rel:
            res.append(prun[i])
    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 = []
    res.append(max(prun))
    for i in np.arange(0.1, 1, 0.1):
        res.append(max(prun[int(i*10):]))
    return np.arange(0, 1, 0.1).tolist(), res

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

def process(qnum):
    qstr = getquery(qnum)
    retr = query(qstr)
    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
    ):
    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)
    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)
```

Listing 3: getrel.py

```python
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 top {7} retrieved documents.}}
\\label{{tab:query20}}
\\end{{table}}
"""

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

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

Listing 4: q83.py

```python
from getrel import *

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]))

results = process(args.qnum)
printresults(*results)
qnum, qstr, retr, rel, prun, rrun, prec, rec, ndcg5, ndcg10, avgprec, recip = results
printdata(rrun, prun, 'urpg{0}.dat'.format(qnum))

irrun, iprun = ipr(rrun, prun)
printdata(irrun, iprun, 'ipr{0}.dat'.format(qnum))
```

Listing 5: q84.py

```r
plotone <- function(data, qnum) {
    pdf(paste('urpg', qnum, '.pdf', sep=''))
    plot(data, type='o', pch=15, ylim=c(0,1), xlim=c(0,1),
        main=paste("Recall-Precision Graph for CACM Query ", qnum, sep=""),
        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(d2, lwd=2, type="p", pch=15, col="gray30")
    lines(id1, lwd=2, type="l", col="gray60")
    lines(id2, lwd=2, type="l", 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()
}
args = commandArgs(trailingOnly=TRUE)

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

Listing 6: Script used to generate the recall-precision graphs

4 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.