Keseyan_CS242_Assignment

Hovanes Keseyan

February 16, 2018

Exercise A

Consider the following document D, taken from a collection C.

"The University of California, Riverside is one of 10 universities within the prestigious University of California system, and the only UC located in Inland Southern California. Widely recognized as one of the most ethnically diverse research universities in the nation."

Consider the following two queries:

- Q1: university Riverside
- Q2: diverse university

Characteristics of collection C are as follows:

- docs in collection C: 1000
- docs in C that contain "Riverside": 100
- docs in C that contain "university/ies": 200
- docs in C that contain "diverse": 150

Compute the scores of Q1 and Q2 for D, using (a) BM25, and (b) Unigram Language Model (with smoothing method of your choice). Make and state any assumptions necessary, e.g., about the constants in BM25.

Given:

No relevancy information

r = 0

R = 0

Documents in the Corpus

N = 1000

Words in the Document

D = 40

Query frequency of each term is 1

qfi = 1

Frequency of each term in the document:

f_university = 4

f_Riverside = 1

f_diverse = 1

Assumptions:

Average words per document in the corpus:

mu <- 100

Constants:

k1 <- 1.2

b <- 0.75

k2 <- 100

Dirichlet Smoothing will be used for the Unigram Language Model score.

Code:

```
ri <- 0
R <- 0
N <- 1000
D <- 40
qfi <- 1
mu <- 100
k1 <- 1.2
b <- 0.75
k2 <- 100

K <- k1*((1-b)+b*D/mu)

# Q1: i=1 for university, i=2 for Riverside
ni <- c(200, 100)
fi <- c(4, 1)
# BM25
bm25q1 <- sum(log(((ri+0.5)/(R-ri+0.5))/((ni-ri+0.5)/
                                          (N-ni-R+ri+0.5)))*(k1+1)*fi/(K+fi)*(k2+1)*qfi/(k2+qfi))
# Unigram Language Model using Dirichlet Smoothing
unigq1 <- prod(log((fi+mu*(ni/(mu*N)))/(D+mu)))


# Q2: i=1 for diverse, i=2 for university
ni <- c(150, 200)
fi = c(1, 4)
# BM25
bm25q2 <- sum(log(((ri+0.5)/(R-ri+0.5))/((ni-ri+0.5)/
                                          (N-ni-R+ri+0.5)))*(k1+1)*fi/(K+fi)*(k2+1)*qfi/(k2+qfi))
# Unigram Language Model using Dirichlet Smoothing
unigq2 <- prod(log((fi+mu*(ni/(mu*N)))/(D+mu)))

library(knitr)
results <- matrix(c(bm25q1, bm25q2, unigq1, unigq2), nrow=2, ncol=2)
rownames(results) <- c("Query 1", "Query 2")
colnames(results) <- c("BM25", "Unigram")
kable(results, caption = "Scores for each query per scoring method")
```

Table 1: Scores for each query per scoring method

	BM25	Unigram
Query 1	5.520470	16.99394
Query 2	4.909598	16.83807

Conclusion:

The BM25 score for was ~5.52 for Query 1 and ~4.91 for Query 2. The Unigram Language Model score using Dirichlet Smoothing was ~16.99 foR Query 1 and ~16.84 for Query 2. Query 1 scores higher when using the BM25 score than Query 2 while the opposite is true for the alternative scoring method.

Exercise B

Compute the PageRank score of each node in the graph below. Show your work.

In how many iterations does the computation converge?

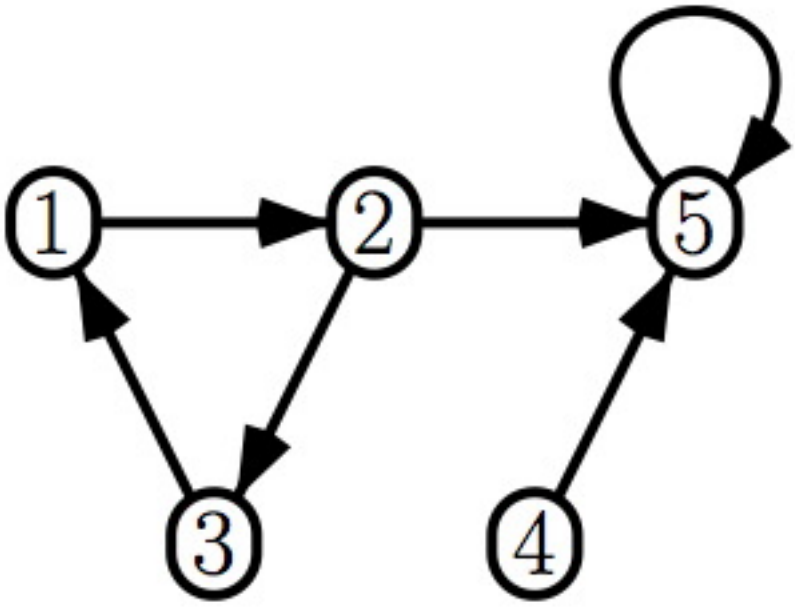

Figure 1: Graph for Exercise B

Setup

We set up the PageRank matrix (PR in code below) by counting the share of incoming link references for each node. We will treat the self-reference in node 5 as a third incoming link reference for node 5, where it receives one-third of the ranking weight.

Code:

```
options(digits=3)
n1 <- c(0, 0, 1, 0, 0)
n2 <- c(1, 0, 0, 0, 0)
n3 <- c(0, 1, 0, 0, 0)
n4 <- c(0, 0, 0, 0, 0)
n5 <- c(0, 1/3, 0, 1/3, 1/3)
PR <- matrix(c(n1, n2, n3, n4, n5), nrow=5, ncol=5)
PR <- t(PR)
rownames(PR) <- c("1 ","2","3","4","5")
colnames(PR) <- c("1","2","3","4","5")
kable(PR, caption = "PageRank matrix for Figure 1")
```

Table 2: PageRank matrix for Figure 1

	1	2	3	4	5
1	0	0.000	1	0.000	0.000
2	1	0.000	0	0.000	0.000
3	0	1.000	0	0.000	0.000
4	0	0.000	0	0.000	0.000
5	0	0.333	0	0.333	0.333

```
v <- c(.2, .2, .2, .2, .2)

v1 <- PR%*%v
v2 <- PR%*%PR%*%v
v3 <- PR%*%PR%*%PR%*%v
v4 <- PR%*%PR%*%PR%*%PR%*%v
v5 <- PR%*%PR%*%PR%*%PR%*%PR%*%v
v6 <- PR%*%PR%*%PR%*%PR%*%PR%*%PR%*%v

m <- matrix(c(v1, v2, v3, v4, v5, v6), nrow=5, ncol=6)
colnames(m) <- c("1","2","3","4","5","6")
kable(m, caption = "PageRank vectors per iteration")
```

Table 3: PageRank vectors per iteration

1	2	3	4	5	6
0.2	0.200	0.200	0.200	0.200	0.2
0.2	0.200	0.200	0.200	0.200	0.2
0.2	0.200	0.200	0.200	0.200	0.2
0.0	0.000	0.000	0.000	0.000	0.0
0.2	0.133	0.111	0.104	0.101	0.1

Conclusion

The converged PageRank values for each node were as folllows:

- Node 1: 0.2
- Node 2: 0.2
- Node 3: 0.2
- Node 4: 0.0
- Node 5: 0.1

The computation seems to converge after 6 iterations.

Exercise C

Show how MapReduce can be used to efficiently solve the following problem:

Given a collection of input documents, output all pairs of keywords that co-occur in at least 1000 of the documents.

Write pseudocode for map and reduce functions.

Full points for most efficient implementation.

Hint: is multi-phase MapReduce useful here?

Setup

Create a 2-phase MapReduce job. The first Map function will take each document as an input, parse each word in each document, and send the document number and each word in it to a Reduce function. The first Reduce function will create a comprehensive running word list as it reads each word from each document and adds it to the list if it has not been added yet. When it adds a word to the word list, it will then add a word pair to the pair list by combining the new word with each existing word. The function then emits each document which holds the document number and the full list of word pairs found in the document. The next Map function then takes this list of word pairs and sends the next Reduce function a list of pairs and the documents they are found in. The last Reduce function then simply counts the number of documents per word pair and outputs the pair if it is found in at least 1000 documents.

Code

```
procedure Map1(input)
  while not input.done() do
    document <- input.next()
    number <- document.number
    tokens <- Parse(document)
    for each word w in tokens do
      Emit(number, w)
    end for
  end while
end procedure

procedure Reduce1(key, values)
  doc.num <- key
  wordlist <- values.next()
```

```
  pairlist <- 0
  while not values.done() do
    word2 <- values.next()
    if word is not in wordlist do
      wordlist <- wordlist + word
      for each word word1 in wordlist do
        pair <- word1 + word2
        doc.pairlist <- pairlist + pair
        Emit(doc)
      end for
    end if
  end while
end procedure

procedure Map2(input)
  while not input.done() do
    doc <- input.next()
    for each pair p in doc do
      Emit(pair, docnum)
    end for
  end while
end procedure

procedure Reduce2(key, values)
  total <- 0
  pair <- key
  while not values.done() do
    total <- total + 1
  end while
  if total >= 1000
    Emit(pair)
  end if
end procedure
```

Conclusion

MapReduce helps distribute this otherwise compute-intensive operation. Multi-phase MapReduce is useful here by distributing the computations in two separate phases: one to find each word pair, and one to count the number of documents each word pair appears (and perform the logical check for the problem condition).

Exercise D

For a specific query Q, suppose that a search engine can produce up to 3 results, where the i-th result has probability 1/(2i) of being relevant. That is, 1st result has probability 1/2, 2nd has 1/4, 3rd has 1/6, and so on. Also, assume Q has a total of 3 relevant results in the collection.

C1: What is the expected average precision (AP) if the engine outputs 2 results?

C2: How many results should the search engine output to maximize the expected AP? Show your calculations and results.

C3: How many results should the search engine output to maximize F (harmonic mean of precision and recall)? Show your calculations and results.

C1

```
results <- 2
relprob <- array(dim=results)
for (i in 1:results) {
  relprob[i] <- 1/(2*i)
}
P <- relprob/results
AP <- mean(P)
AP
```

```
## [1] 0.188
```

At 2 results, the expected AP is 18.8%.

C2

```
AP <- array(dim=3)
for (results in 1:3) {
  relprob <- array(dim=results)
  for (i in 1:results) {
    relprob[i] <- 1/(2*i)
  }
  P <- relprob/results
  AP[results] <- mean(P)
}
AP
```

```
## [1] 0.500 0.188 0.102
```

The search engine should output 1 result to maximize expected AP at 50%.

C3

```
c <- 3 # relevant results in the collection
FM <- array(dim=3)
for (results in 1:3) {
  relprob <- array(dim=results)
  for (i in 1:results) {
    relprob[i] <- 1/(2*i)
  }
  P <- relprob/results
  R <- relprob/c
  FM[results] <- (2*R*P)/(R+P)
}
```

```
## Warning in FM[results] <- (2 * R * P)/(R + P): number of items to replace
## is not a multiple of replacement length

## Warning in FM[results] <- (2 * R * P)/(R + P): number of items to replace
## is not a multiple of replacement length
```

```
FM
```

```
## [1] 0.250 0.200 0.167
```

The search engine should output 1 results to maximize F at 25%.