Written Submission For

CSC 483/583: Homework 3

Spring 2017

By Mithun Paul

Today's Date: 14th Mar 2017

Due by 11:59 P.M., March 15 (upload to D2L or, if on paper, turn it in in class or in the instructor's office)

Because the credit for graduate students adds to more than 75 points, graduate students grades will be normalized at the end to be out of 75. For example, if a graduate student obtains 80 points on this assignment, her final grade will be $80 \times 75/85 = 70.6$. Under- graduate students do not have to solve the problems marked grad students only. If they do, their grades will not be normalized but will be capped at 75. For example, if an un- dergraduate student obtains 80 points on this project (by getting some credit on the grad students only problem), her final grade will be 75.

Note that only problem 1 (parts 1, 3, and 4) requires coding.

Problem 1 (30 points undergraduate students / 40 points graduate students)

For this problem you will learn how to use Lucene:

1. (15 points) Write code to index the following documents with Lucene 4:

Document #1: "information retrieval is the most awesome class I ever took."

Document #2: "the retrieval of private information from your emails is a job that the NSA loves."
Document #3: "at university of arizona you learn about data science."

Document #4: "the labrador retriever is a great dog."

Your indexed documents should have two fields: a tokenized and searchable text field containing the text of the document, and another non-tokenized field containing the document name, e.g., "Document #1".

For this task, you might want to read this tutorial first: http://www.lucenetutorial. com/lucene-in-5-minutes.html.

If you are programming in Python, this Python wrapper follows closely the original Lucene syntax: http://lucene.apache.org/ pylucene/

Qn 1.1) Search for the query "information retrieval". Print the list of documents in the search result, sorted in descending order of their relevance score. For each document, print its name and score.

Ans:

Found 2 hits for your query:information retrieval

1. Document #1 Score:1.1655893

2. Document #2 Score:1.1655893

Code uploaded to D2L.

Qn 1.2) (5 points) Describe the default scoring strategy in Lucene. Is it Boolean, vector- based, or a combination of both? To answer this question, you might want to start here: http://lucene.apache.org/core/6_4_1/index.html.

Update to qn 1.2: No changes at all. Yes, we now know that the similarity formula is BM25. But, as the query language for Lucene makes clear, Lucene supports Boolean and proximity queries as well. How are these combined into a single system, that is, a single scoring algorithm? (we actually discussed this in class...)

Ans:

I think Lucene first uses Boolean queries, if any, to narrow down the documents that need to be scored. For example if the query was "information AND retrieval" , it will pick all the documents from the index which has both the terms in it, and will ignore the rest.

Then if there is a proximity query specified by the user, within these documents it runs a proximity query. We must keep in mind that for proximity searches exact matches (words next to each other) are considered as proximity zero. But if a proximity distance is specified , which is more than zero, then the zero case becomes a subset of this proximity distance. So no need to calculate proximity=0 explicitly.

Also do note that an alternate way of substituting for boolean AND query is specifying a proximity query like this: "information retrieval"~10000000. Now essentially both the boolean query and the proximity query will return same results in this case. However, the proximity query will assign higher score to the documents in which the terms in query are closer together, thus affecting the final score. Though the trade off is that, proximity queries take more CPU cycles, and hence boolean queries, if possible are preferred.

Anyway, once all these preliminary steps are completed, and the number of documents reduced,a query vector and a document vector will be calculated for tf-idf scoring (Assuming that its ClassicSimilarity).

Qn 1.3). (10 points) Implement and run the following three queries: (a) "information AND retrieval", (b) "information AND NOT retrieval", and (c) "information AND retrieval WITHIN 1 WORD OF EACH OTHER. What documents and what scores are returned for each of these queries? For this task, you might want to read about the syntax of Lucene queries: http://lucene.apache.org/core/2_9_4/queryparsersyntax. Html.

Ans:

(a) "information AND retrieval",

Found 2 hits for your query:information AND retrieval

1. Document #1 Score:1.1655893

2. Document #2 Score:1.1655893

(b) "information AND NOT retrieval",

Ans: 0/no documents found

(c) information AND retrieval WITHIN 1 WORD OF EACH OTHER.

Ans:

Found 1 hits for your query:"information AND retrieval"~1

1. Document #1 Score:0.7204689

Qn 1.4. (10 points – graduate students only) Change the default similarity formula in Lucene to a different one. **For example, you might want to try a different formula, such as the classic cosine similarity over tf-idf weights.** Does the ordering of documents change using this new formula for the query "information retrieval"? What are the scores returned for each document now?

Ans: The ordering of the documents did not change. However the scores did change. Note that the classic cosine similarity after normalization has a maximum value of 1. Hence if you look at the results below, you can see that while BM25 returned scores more than 1, ClassicSimilarity scores are below 1.

Below are some results I got for both BM25 and ClassicSimilarity for the same query ("information retrieval") and document set.

With BM25

Found 2 hits for your query:information retrieval

1. Document #1 Score:1.1655893

2. Document #2 Score:1.1655893

With ClassicSimilarity

Found 2 hits for your query:information retrieval

1. Document #1 Score:0.6676969

2. Document #2 Score:0.6676969

More Proof:

To make sure that the ordering of the documents doesn't change based on the Similarity measure used, I did some more tests with some modifications to the documents. Results are as below.

Query: "Information Retrieval"

Documents:

addDoc(w, "**information retrieval** is the most awesome **information** class I ever took.","Document #1");

addDoc(w, "the **retrieval** of private **information** from your emails is a job that the NSA

loves.","Document #2");

addDoc(w, "at university of arizona you learn **information** about data science.","Document #3");

addDoc(w, "the labrador retriever is a great dog.","Document #4");

With BM25

Found 3 hits for your query:information retrieval

1. Document #1 Score:1.0477545
2. Document #2 Score:0.913322
3. Document #3 Score:0.3102993

With ClassicSimilarity

the query string is:information retrieval

Found 3 hits for your query:information retrieval

1. Document #1 Score:0.70708585
2. Document #2 Score:0.6074629
3. Document #3 Score:0.12025557

Problem 2 (10 points)

Compute variable byte codes and γ codes for the postings list < 777,17743,294068,31251336 >. Use gaps instead of docIDs where possible. Write binary codes in 8-bit blocks. You can use Google, or any other resource, to convert numbers to binary.

Ans:

< 777,17743,294068,31251336 >

Docid 1=777

Gap 1=16966

Gap 2= 294068 - 17743=276325

Gap 3=31251336-294068 = 30957268

Encoding Doc id 1

Ans:777

VB Encoding: **00000110 10001001**

Gamma encoding = **111 11111101 00001001**

Explanation:

777 in binary = 0000001100001001

VB Encoding (split into groups of 7, attach one to the last byte's continuation bit, zeroes otherwise):

00 00000110 10001001

Encoding using Gamma Code.

777 in binary = 0000001100001001

Offset: chop off first bit which is one = ~~0000001~~ 100001001

Length (number of left over bits after chopping) = 9

In unary 14= 1111111110

Gamma encoding (combine length +offset)= 111 11111101 00001001

Encoding Gap 1

Ans:16966

VB Encoding: **00000001 00000100 11000110**

Gamma encoding = **00011111 11111111 10000010 01000110**

Explanation:

16966 in binary = 01000010 01000110

VB Encoding (split into groups of 7, attach one to the last byte's continuation bit, zeroes otherwise):

0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	1	1	0	0	0	1	1	0
1	2	3	4	5	6	7	8	1	2	3	4	5	6	7	8	1	2	3	4	5	6	7	8

Encoding Gap 1 using Gamma Code.

16966 in binary = 0100001001000110

Offset: chop off first bit which is one = 00001001000110

Length (number of left over bits after chopping) = 14

In unary 14= 111111111111110

Gamma encoding (combine length +offset)= 01111111 1111111 00000100 1000110

Encoding Gap 2

Ans:276325

VB Encoding: **00010000 01101110 11100101**

Gamma encoding = **11111 11111111 11111000 00110111 01100101**

Explanation:

276325 in binary = 10000 1101110 1100101

VB Encoding (split into groups of 7, attach one to the last byte's continuation bit, zeroes otherwise)::

00010000 01101110 11100101

Encoding Gap 2 using Gamma Code.

276325 in binary = 10000011011101100101

To find offset: chop off first bit which is one = ~~1~~00001101110110010 1

Length(number of left over bits after chopping) = 18

In unary 18= 111111111111111110

Gamma encoding (combine length +offset)= 11111 11111111 11111000 00110111 01100101

<u>Encoding Gap 3</u>

Ans:30957268

VB Encoding:

00001110 01100001 00111101 11010100

Gamma encoding =

00000001 11111111 11111111 11111110 11011000 01011110 11010100

Explanation:

30957268 in binary = 1110110000101111011010100

VB Encoding (split into groups of 7, attach one to the last byte's continuation bit, zeroes otherwise):

00001110 01100001 00111101 11010100

Encoding Gap 3 using Gamma Code.

30957268 in binary = 1110110000101111011010100

To find offset: chop off first bit which is one = 1101100001011110110101000

Length(number of left over bits after chopping) = 24

In unary 24= 111111111111111111111110

Gamma encoding (combine length +offset)=

00000001 11111111 11111111 11111110 11011000 01011110 11010100

Problem 3 (10 points)

From the following sequence of γ-coded gaps, reconstruct first the gap sequence and then the postings sequence: 1110001110101011111101101111011.

Ans:

Gap sequence: <9,6,3,59,7 >

Postings Sequence: <9,15,18,77, 84>

Explanation:

Read till you hit first zero

1110

i.e length =3

Read 3 bits now.

001

Add a 1 to the beginning of what you read

I.e 1001

Convert to denary

i.e first doc id=9

Gap 1:

Length= 110 =2

offset= 110

denary= 6

Gap 2:

1110 001, 110 10, 1111110 110111, 10 11.

1110 001, 110 10, 1011111101101111011.

1110 001, 110 10 , 10 1 , 111110 11011, 110 11.

9, 6,3

Gap 3

length= 10= 1

Offset = 11

decimal=3

Gap 4

length=111110=5

offset= 111011

decimal=59

Gap 5

Length =10=2

Offset =111

Decimal =7

Problem 4 (10 points)

Consider the table of term frequencies for 3 documents denoted Doc1, Doc2, Doc3 in the table below. Compute the tf-idf weights for the terms "car", "auto", "insurance", and "best", for each document, using the idf values from the second table.

	Doc 1	**Doc2**	**Doc 3**
car	27	4	24
auto	3	33	0
insurance	0	33	29
best	14	0	17

term	df_t	idf_t
car	18,165	1.65
auto	6,723	2.08
insurance	19,241	1.62
best	25,235	1.5

Ans:

TERM FREQUENCy= 1+ log tf

For car in doc 1: Tf = 1+log_10(27)=1+1.4313637641589871

	doc1	doc2	doc3	idf for doc	tf-idf doc1 and query	tf-idf doc2 and query	tf-idf doc3 and query
car	27	4	24				
after log frequency weighting	2.431363764	1.602059991	2.380211242	1.65	4.011750211	2.643398986	3.927348549
auto	3	33	0				
after log frequency weighting	1.477121255	2.518513994	0	2.08	3.07241221	5.238508995	0
insurance	0	33	29				
after log frequency weighting	0	2.518513994	2.462397998	1.62	0	4.079992583	3.989084757
best	14	0	17				
after log frequency weighting	2.146128036	0	2.230448921	1.5	3.219192054	0	3.345673382
				tf-idf=	**10.30335447**	**11.96190056**	**11.26210669**

Ans: as seen from the tf-idf calculation, DOC 2 is the closest/best result for the given query

Problem 5 (15 points)

Qn 5.1. What is the idf of a term that occurs in every document? Compare this with the use of stop word lists.

Ans:

System 1:

The tf-df of a term that occurs in every document is zero. So it has no contribution to the tf-idf calculation.

df= number of documents in which term appears=X
N= total number of documents =X

Therefore idf =Log_10(X/X)= log_10(1)=0

System 2: Removing stop words:

Removing the stop words first, even before calculating the tf-idf values is a smart move. That means, you will have so many less number of words whose tf-idf you have to calculate. After doing this, it will be smarter to remove the words which have tf-idf score of zero.

Caveat: If it was me, I wouldn't blindly trust a list of global stopwords. Instead I will take the most common stop words (Eg:and, of, are etc), make sure they occur in all the documents in my corpus. This will be time/cpu cycle consuming but precise. Because this is a corpus dependent thing. Some words which are stop words for a particular corpus (Eg: General search) might not be a stop word for another domain (Eg: Legal)

Qn 5.2). How does the base of the logarithm in the idf formula affect the score calculation of the following formula: Does a different logarithm base affect the relative scores of two documents on a given query? If so, how? To answer this question, you must know how to change the base of a logarithm. Google "how to change the base of a logarithm" :)

Ans: I don't think changing log base should affect a tf-idf calculation. As long as we are using the same log, say Log_2 , everywhere, it will be fine. After all, log is just a damping function. And changing base just changes a constant value equally. So it just becomes a scaling term.Further explanation is as below. You can ignore this part, if you are already satisfied with my answer:

Further Explanation:

The log conversion formula is:

$\text{Log}_b\ x = \text{Log}_a\ x / \text{Log}_a\ b$

For example to convert 100 from Log 10 to Log 2, all we have to do is,

Log_2 (100)= Log_10(100)/Log_10(2)

= 2/0.30102999566

= 6.64385618986

Further to prove the point that changing a log base wouldn't affect the tf-idf calculation, i did the same tf-idf calculation of qn 4, but with a log base of 2 for both tf and idf calculation. The results are as follows. As you can see, even though the values per se have changed, the result still remains the same, i.e Doc 2 has the highest tf-idf value for this query. Note, I have assumed N=50000, to match the idf values given in question 4.

	doc1	doc2	doc3	df	idf for doc-log_2 (N/df)	tf-idf doc1 and query	tf-idf doc2 and query	tf-idf doc3 and query
car	27	4	24	18165				
after log frequency weighting	5.754887502	3	5.584962501		1.598270253	9.197865503	4.794810759	8.926279428
auto	3	33	0	6723				
after log frequency weighting	2.584962501	6.044394119	0		3.032254564	7.83826434	18.32814165	0
insurance	0	33	29	19241				
after log frequency weighting	0	6.044394119	5.857980995		1.515247837	0	9.158755117	8.876293034
best	14	0	17	25235				
after log frequency weighting	4.807354922	0	5.087462841		1.124005532	5.403493527	0	5.718336377
					tf-idf=	**22.439623**	**32.281707**	**23.520908**