Mirna Arivalagan - 220142881

Database and Information Retrieval – SIT772

Assignment 2

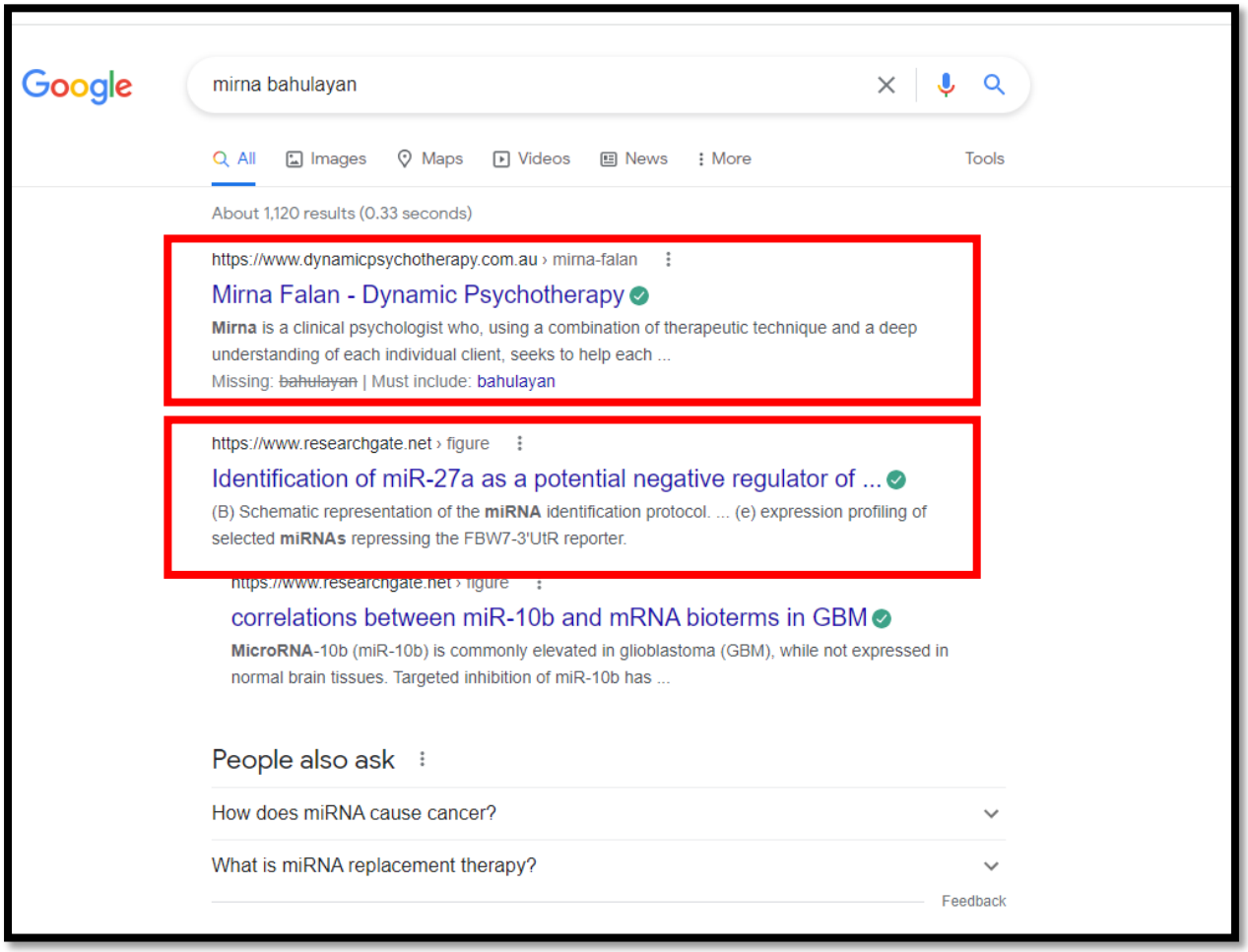
Trimester 3 2021

Table of Contents

Question 1

The search term is used for this question is **Mirna Bahulayan** and there was a total of 1120 results appeared in google. From the first page of the 9 results seen, 8, results had either 1 or a combination of both terms, and 1 result had no relevant search term. Below is the number of results with the combination of search terms:

Search Term Combination	Results on page 1	Colour of boxes on screenshot
Mirna Bahulayan	4	Orange
Mirna	2	Red
Bahulayan	2	Green



Based on the search results above, it does appear that google interprets all queries as a Boolean conjunction, as from the 9 results above, 4 of them had a combination of both terms, meaning the AND operator was factored in, and the remaining 4 had either one of the search terms, meaning google has factored in the OR operator. I also brought up 1 search results where it did not contain the term Mirna or Bahulayan, but it had a similar term of Mrna, so my assumption would be that it uses the NOT operator and bring up similar search terms to what we are searching for.

Question 2(A)

Question	Overall Precision	Overall Recall	Precision – 5th retrieved doc
1	=6/15 = 0.40	=6/12 = 0.5	(33): 2/5 = 0.40
2	=7/13 = 0.54	=7/11 = 0.64	(27): 3/5 = 0.60
3	=6/12 = 0.50	=6/12 = 0.50	(21): 1/5 = 0.20
4	=5/10 = 0.50	=5/9 = 0.56	(27): 1/5 = 0.20

Question 2(B)

Couple of instances where, aiming for precision is crucial is in a health setting or legal setting. In a health scenario, the risks are quite high. For example, if a doctor is looking up side effects to a particular medication, or potential interaction effects between multiple medications, precision is really important here as the doctor needs accurate information on these medications, as miss prescribing medication can cause further complications to the patients. With regards to legal cases, again precision is important here to bring up cases that are relevant to the search terms that the lawyer is looking up, as bringing up irrelevant searches will cost the lawyer more time to research even further as lawyers usually have to research plenty of case laws when working on a case, so having to read through irrelevant documents will just overload their already busy schedules.

Couple of scenarios where recall may work better is when looking up music or travel destinations. For example, when someone is looking up for a certain genre of music, or certain artist, bringing up searches that are not relevant to that certain artist or genre is not a bad thing, as this may introduce some new music to the person searching and expand their music collection. Same goes for travel destinations, if for example, someone is searching for a resort with a pool in Cairns, and the search not only brings up results for resorts in Cairns, but also resorts in Palm Cove or Port Douglas, this will provide with the traveller with more options, and options to go off the beaten path a little. In both these scenarios, there is no risks here, so it's ok to bring up irrelevant results to the search terms.

Question 2(C)

I believe the way google ascertains if the searcher is looking for higher precision or higher recall is first by looking at the time the user spends on the web pages of the returned results. If the user spends more time on a page where it is relevant to the search terms put in, and when the user spends only a few seconds on a page where it is not relevant to the search terms put in, then google can tell that the person searching for something is looking for more precision. For example, when I search books on machine learning, and google brings up a combination of results on machine learning books and predictive analytics books, I then spend a few minutes looking on the machine learning pages, but only spend a few seconds on the predictive analytics pages, google will know that I am specifically looking for precision here and only want to look at machine learning books.

One way google can tell if a user is interested in recall and doesn't mind looking at other pages that may not be exactly relevant but could be of interest is when the user clicks on the related searches section at

the bottom of the google search page. By doing so, google can know to bring up items that may be similar to what the user is looking for.

Question 3(A)

Applying Heaps' law theory where the more texts are gathered, there will be diminishing return of new unique words to be discovered, I would say that the combination of C2 + C3 would see more new words identified compared to C2 + C3, because the total value of C1 + C3 is 300,500 documents and this is lower than the combination of C2 + C3 which will see 315,000 documents. Even though C2 + C3 has more documents, thus there will be diminishing returns of new unique words, when you look at the overall unique words, **C2 + C3** will have more than C1 + C3.

When the calculation of heaps law is plotted on a graph, you will observe a relationship where with lower number of documents, there is a significant increase of unique new words, but as the document size grows, the increase in unique new words reduces and reaches a plateau eventually. However, there will be instances where the number of unique words keep growing as the number of documents increases, this can be attributed to reasons such as spelling errors, email addresses and names such as individual, companies and product names where there can be many unique names.

Question 3(B)

To calculate the tf-idf for the documents specified, we first need to prepare a term frequency table. Below is the TF table from the words in the 3 different documents.

Term Frequency Table					
Terms	Sweet	Potatoes	Oranges	Sour	Apple
D1	2	1	0	0	0
D2	2	0	1	1	0
D3	3	1	1	0	1
Total	7	2	2	1	1

Next, we need to compute the IDF (Inverse Document Frequency) and this is the log base 2 value of N/n. N here represents the total number of documents and n is the number of documents the specified term appears in.

Compute IDF		
Terms	N/n	log_base 2 value
IDF(Sweet):	1.00	0.000
IDF(Potatoes):	1.50	0.585
IDF(Oranges):	1.50	0.585
IDF(Sour):	3.00	1.585
IDF(Apple):	3.00	1.585

Once the logbase 2 values has been calculated, we can then compute the TF-IDF, which is the number of times the term appears in the document multiplied by the log base value. The formula to calculate TF-IDF is:

wi,j=TF(i,j)×IDF(i)

Compute TF-IDF					
Terms	Sweet	Potatoes	Oranges	Sour	Apple
D1	0.00	0.58	0.00	0.00	0.00
D2	0.00	0.00	0.58	1.58	0.00
D3	0.00	0.58	0.58	0.00	1.58

The final weights for the documents are as below:

- Doc 1: <0, 0.58, 0, 0, 0>
- Doc 2: <0, 0, 0.58, 1.58, 0>
- Doc 3: <0, 0.58, 0.58, 0, 1.58>

Question 4(i & ii)

To compute the document scores and ranking for the statement of Houses OR for OR Sale OR in OR Geelong OR Melbourne, we need to look at the number of times each term appears in each document. As the statement contains all OR statements, we will pick the maximum value as the final document score. And the document with the highest score will be ranked 1st and the document with the lowest score will be ranked last.

Below is the ranking and scores for each document for this query statement:

1st: Doc 1 (39)

2nd: Doc2 + Doc 3 (21)

3rd: Doc 4 (20)

Term	**Doc ID**			
	DOC1	**DOC2**	**DOC3**	**DOC4**
House	39	19	19	12
For	11	19	20	20
Sale	32	3	1	14
In	22	15	3	1
Geelong	22	16	21	13
Melbourne	4	21	9	13
Final Document Score	**39**	**21**	**21**	**20**
Ranking	**1st**	**2nd**		**3rd**

Question 4(iii & iv)

To compute the document scores and ranking for the statement Houses AND for AND Sale AND in AND Geelong OR Melbourne. First, we need to decide how we will place the bracket for this query as it contains a mix of AND and OR statements.

I have decided to use the following approach:

(Houses AND for AND Sale AND in AND Geelong) or Melbourne – meaning that I am looking for documents that needs to contain Houses + For + Sale + In + Geelong, or only bring up documents that contains Melbourne.

Next, we need to first calculate the document scores for the statement within the brackets: **(Houses AND for AND Sale AND in AND Geelong)** and as this contains all AND statements, we will pick the lowest term frequency value as the first pass document score to compare later with the OR part of the statement. For the AND statements, I have landed with the following document scores:

- Doc 1: 11
- Doc 2: 3
- Doc 3: 1
- Doc 4: 1

Now that we have the scores for the AND statement, we calculate the document scores for the OR statement by comparing the values with the AND statement that I have assigned as X.

As the last part of the statement is OR, we need to pick the maximum value. Below are the scores that I have landed as the final documents scores:

- Doc 1: 11
- Doc 2: 21
- Doc 3: 9
- Doc 4: 13

And here is the final ranking:

1st: Doc 2

2nd: Doc 4

3rd: Doc1

4th: Doc 3

Below is the table of my calculations:

Term	Doc ID			
	DOC1	DOC2	DOC3	DOC4
House	39	19	19	12
For	11	19	20	20
Sale	32	3	1	14
In	22	15	3	1
Geelong	22	16	21	13
Melbourne	4	21	9	13
X = (Houses AND for AND Sale AND in AND Geelong)	11	3	1	1
X or Melbourne	11	21	9	13
Final Document Score	11	21	9	13
Ranking	3rd	1st	4th	2nd

Question 4(v & vi)

To compute the document scores and ranking for the statement Houses AND NOT Geelong, we need to look at the frequency for the term Geelong in each document and reverse them to 0 where the term is present in the document, and where the term is not present in the document, we set them as 1. Once we have done the reversal, next we need to compare the frequency between the terms Houses and Not Geelong and select the lowest frequency as the final document score as we are dealing with an AND statement here.

With this statement and the documents, we have, the lowest frequency across all 4 documents is 0, thus this means all documents will score 0 and will not be retrieved as all documents contain the word Geelong.

Term	Doc ID			
	DOC1	DOC2	DOC3	DOC4
Houses	39	19	19	12
Geelong	22	16	21	13
Not Geelong	0	0	0	0
Final Document Score	0	0	0	0
Ranking	None as all documents contain Geelong			

Question 5(i)

To compute the similarity scores between the 1st and 2nd query, first we need to generate a term frequency table. This is just the number of times the specific term appears in each document. The formula for this is: TF(i,j)=freq(i,j)

Step 1: Term Frequency											
Term	Book	Consider	Good	Make	Read	Feel	Better	Love	Once	Tom	Recent
Doc 1	2.00	1	1	1	1	1	1	0	0	0	0
Doc 2	1.00	0	1	0	1	1	1	1	0	0	0
Doc 3	1.00	0	0	0	1	1	1	0	1	1	1

Once we have generated the term frequency table, we can proceed to calculate the IDF. The formula for IDF is: IDF(i)=log2(Nni)

Step2: IDF Vector											
Term	Book	Consider	Good	Make	Read	Feel	Better	Love	Once	Tom	Recent
N/n	1.00	3	1.5	3	1	1	1	3	3	3	3
IDF	0.00	1.584963	0.584963	1.584963	0	0	0	1.584963	1.584963	1.584963	1.584963

Once we have the IDF values, we can then calculate the TF-IDF vector values. The formula for this is: wi,j=TF(i,j)×IDF(i).

Step3: TF-IDF											
Term	Book	Consider	Good	Make	Read	Feel	Better	Love	Once	Tom	Recent
Doc 1	0	1.584963	0.584963	1.584963	0	0	0	0	0	0	0
Doc 2	0	0	0.584963	0	0	0	0	1.584963	0	0	0
Doc 3	0	0	0	0	0	0	0	0	1.584963	1.584963	1.584963
Q1	1	0	1	0	0	0	0	1	0	0	0
Q2	1	0	1	1	1	1	1	0	0	0	0

The final TF-IDF weight for each document is:

Doc 1: < 0, 1.58, 0.58, 1.58, 0, 0, 0, 0, 0, 0, 0 >

Doc 2: < 0, 0, 0.58, 0, 0, 0, 0, 1.58, 0, 0, 0 >

Doc 3: < 0, 0, 0, 0, 0, 0, 0, 0, 1.58, 1.58, 1.58 >

Now that we have the TF-IDF Weights, we can calculate the cosine similarity for each query and document.

Document	Query	Formula	Score
Doc 1	Q1 <1,0,1,0,0,0,0,1,0,0,0>	Sim(Q1,D1) = ((1*0.58)/ Sqrt((1^2+1^2+1^2)*Sqrt(1.58^2+0.58^2+1.58^2))	0.15
	Q2 <1,0,1,1,1,1,1,0,0,0,0>	Sim(Q2,D1) = ((1*0.58)+(1*1.58)/ Sqrt((1^2+1^2+1^2+1^2+1^2+1^2+1^2)*Sqrt(1.58^2+0.58^2+1.58^2))	0.38
Doc 2	Q1 <1,0,1,0,0,0,0,1,0,0,0>	Sim(Q1,D2) = ((1*0.58)+(1*1.58)/ Sqrt((1^2+1^2+1^2)*Sqrt(0.58^2+1.58^2))	0.74
	Q2 <1,0,1,1,1,1,1,0,0,0,0>	Sim(Q2,D2) =((1*0.58)/ Sqrt((1^2+1^2+1^2+1^2+1^2+1^2+1^2)*Sqrt(0.58^2+1.58^2))	0.14
Doc 3	Q1 <1,0,1,0,0,0,0,1,0,0,0>	Sim(Q1, D3) = ((0 / Sqrt((1^2+1^2+1^2)*Sqrt(1.58^2+1.58^2+1.58^2))	0
	Q2 <1,0,1,1,1,1,1,0,0,0,0>	Sim (Q2, D3) = (0 / Sqrt((1^2+1^2+1^2+1^2+1^2+1^2+1^2)*Sqrt(1.58^2+1.58^2+1.58^2))	0

Based on the table above, the document that is most relevant to the Query 1 is document 2 with a similarity score of 0.74, followed by document 1 with a similarity score of 0.15.

For the 2nd query, the document most relevant to this query is document 1 with a similarity score of 0.38, followed by document 2 with a similarity score of 0.14.

Across both queries, document 3 is not relevant as all as both scored 0 for the similarity scores.

Question 5(ii)

To calculate the similarity scores using TF instead of TF IDF, we follow the same steps above and work out the term frequency, but we do not need to compute the IDF and TF-IDF scores.

Term Frequency											
Term	Book	Consider	Good	Make	Read	Feel	Better	Love	Once	Tom	Recent
Doc 1	2	1	1	1	1	1	1	0	0	0	0
Doc 2	1	0	1	0	1	1	1	1	0	0	0
Doc 3	1	0	0	0	1	1	1	0	1	1	1
Q1	1	0	1	0	0	0	0	1	0	0	0
Q2	1	0	1	1	1	1	1	0	0	0	0

Below are the TF weights for each document:

Doc 1: < 2, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0 >

Doc 2: < 1, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0 >

Doc 3: < 1, 0, 0, 0, 1, 1, 1, 0, 1, 1, 1 >

Now that we have the weights for each document, we can calculate the cosine similarity scores:

Document	Query	Formula	Score
Doc 1	Q1 <1,0,1,0,0,0,0,1,0,0,0>	Sim(Q1,D1) = ((1*2)+(1*1)/ Sqrt((1^2+1^2+1^2)*Sqrt(2^2+1^2+1^2+1^2+1^2+1^2+1^2))	0.55
	Q2 <1,0,1,1,1,1,1,0,0,0,0>	Sim(Q2,D1) = ((1*2)+(1*1)+(1*1)+(1*1)+(1*1)+(1*1) / Sqrt((1^2+1^2+1^2+1^2+1^2+1^2+1^2)*Sqrt(2^2+1^2+1^2+1^2+1^2+1^2+1^2))	0.90
Doc 2	Q1 <1,0,1,0,0,0,0,1,0,0,0>	Sim(Q1,D2) = ((1*1)+(1*1)+(1*1) / Sqrt((1^2+1^2+1^2)*Sqrt(1^2+1^2+1^2+1^2+1^2+1^2+1^2))	0.71
	Q2 <1,0,1,1,1,1,1,0,0,0,0>	Sim(Q2,D2) =((1*1)+(1*1)+(1*1)+(1*1)+(1*1)/ Sqrt((1^2+1^2+1^2+1^2+1^2+1^2+1^2)*Sqrt(1^2+1^2+1^2+1^2+1^2+1^2+1^2))	0.83
Doc 3	Q1 <1,0,1,0,0,0,0,1,0,0,0>	Sim(Q1, D3) = ((1*1)/ Sqrt((1^2+1^2+1^2)*Sqrt(1^2+1^2+1^2+1^2+1^2+1^2+1^2))	0.22
	Q2 <1,0,1,1,1,1,1,0,0,0,0>	Sim (Q2, D3) = ((1*1)+(1*1)+(1*1)+(1*1) / Sqrt((1^2+1^2+1^2+1^2+1^2+1^2+1^2)*Sqrt(1^2+1^2+1^2+1^2+1^2+1^2+1^2))	0.62

Based on the table above, the document that is most relevant to query 1 is document 2, with a similarity score of 0.71, followed by document 1 with a similarity score of 0.55, and the least similar document to query 1 is document 3, with a score of 0.22.

For query 2, the most relevant document is document 1, with a similarity score of 0.9, followed by document 2, with a similarity score of 0.83 and the least similar is document 3, with a score of 0.62.

The biggest difference when calculating similarity scores using TF rather than TF-IDF for both these queries are:

- For query 2, the TF-IDF method had ranked doc 1 as the most similar, but the TF method had ranked doc 2 as the most similar.
- For query 2, the TF- IDF method had scored doc 3 with a similarity score of 0, however with the TF method, doc3, was the 2nd most similar document with a score of 0.62. I think in this instance, when looking at the query and document terms, they are quite similar, so the TF method is producing better results in this instance.
- For query 1, interestingly the TF method produced some scores for document 3 (0.22), and the TF-IDF method scored 0. In this instance, I think the TF-IDF method is more accurate, as the query only contains 1 term that are in document 3.

	Doc 1	Doc 2	Doc 3
Q1 (TF-IDF)	0.15	0.74	0
Q2(TF-IDF)	0.38	0.14	0
Q1 (TF)	0.55	0.71	0.22
Q2(TF)	0.9	0.83	0.62