INFO-533
Assignment/Homework 4
Sanket Santosh Dalvi
B01032973

"I have done this assignment completely on my own. I have not copied it, nor have I given my solution to anyone else. I understand that if I am involved in plagiarism or cheating, I will have to sign an official form that I have cheated and that this form will be stored in my official university record. I also understand that I will receive a grade of 0 for the involved assignment and my grade will be reduced by one level (e.g., from A to A- or from B+ to B) for my first offense, and that I will receive a grade of "F" for the course for any additional offense of any kind."

- **Sanket Santosh Dalvi**

Problem 1:

Consider the two metrics Precision & Recall. For an IR system, which of the following is possible:

a. **Precision = 1**
b. **Recall = 1**
c. **Both = 1**

Justify your answer.

Answer:

Precision = 1:
It is possible for Precision to be 1 if the system retrieves only relevant items and doesn't retrieve any irrelevant items. This means that every item the system retrieved was relevant. However, achieving a Precision of 1 is often difficult in practical scenarios due to the presence of noise and ambiguity in data.

Recall = 1:
It is possible for Recall to be 1 if the system retrieves every relevant item and doesn't miss any. This means that the system retrieved all relevant items in the collection. However, achieving a Recall of 1 can be challenging in practice, especially with large datasets, as it requires the system to identify and retrieve every relevant item accurately.

Both = 1:
It is extremely challenging, if not impossible, to achieve both precision and recall of 1 simultaneously. This is because there is often a trade-off between precision and recall: increasing one typically leads to a decrease in the other. Achieving both metrics at 1 would require a system to perfectly distinguish between relevant and irrelevant items, retrieving all relevant items while excluding all irrelevant ones, which is highly unlikely in real-world scenarios. It's theoretically possible for both Precision and Recall being 1, but it's extremely rare or nearly impossible in real-world scenarios.

Problem 2:

Calculate Cosine Similarity between each document and query pair provided below:

Documents:

1. **fast car win more race**
2. **sport car and fast car win**
3. **car win major car race**
4. **formula race win major car bet**

Queries:

1. **formula**
2. **sport fast drive**

Answer:

Step 1: Calculate Term Frequency for each document using natural(raw) term frequency count and normalize the tf. (Using formulas from slide "Tf-Idf and Cosine similarity" from week 7-8)

Document 1					
Term	fast	car	win	more	race
TF (Count)	1	1	1	1	1
NTF (Normalized TF)	1/5 = 0.2	1/5 = 0.2	1/5 = 0.2	1/5 = 0.2	1/5 = 0.2

Document 2					
Term	sport	car	and	fast	win
TF (Count)	1	2	1	1	1
NTF (Normalized TF)	1/6 =0.1666	2/6 =0.3333	1/6 =0.1666	1/6 =0.1666	1/6 =0.1666

Document 3					
Term	car	win	major	race	
TF (Count)	2	1	1	1	
NTF (Normalized TF)	2/5 =0.4	1/5 =0.2	1/5 =0.2	1/5 =0.2	

Document 4						
Term	formula	race	win	major	car	bet
TF (Count)	1	1	1	1	1	1
NTF (Normalized TF)	1/6 =0.1666	1/6 =0.1666	1/6 =0.1666	1/6 =0.1666	1/6 =0.1666	1/6 =0.1666

Also, Compute term frequency (tf) and normalized tf for query terms

Query 1			
Term	formula		
TF (Count)	1		
NTF (Normalized TF)	1/1 =1		

Query 2			
Term	sport	fast	drive
TF (Count)	1	1	1
NTF (Normalized TF)	1/3 =0.3333	1/3 =0.3333	1/3 =0.3333

Step 2: Calculate the inverse document frequency. (Using formulas from slide "Tf-Idf and Cosine similarity" from week 7-8, that is idf = 1 + $\log_e$ (N/df_t))

Term	df	idf
fast	2	1.6931
car	4	1
win	4	1
more	1	2.3862
race	3	1.2876
sport	1	2.3862
and	1	2.3862
major	2	1.6931
formula	1	2.3862
bet	1	2.3862

Step 3: Calculate tf*idf for each document against each query.

Query 1				
Term	Document 1	Document 2	Document 3	Document 4
formula	0 * 2.3862 =0	0 * 2.3862 =0	0 * 2.3862 =0	0.1666 * 2.3862 = 0.3975

Query 2				
Term	Document 1	Document 2	Document 3	Document 4
sport	0 * 2.3862 =0	0.1666 * 2.3862 =0.3975	0 * 2.3862 =0	0 * 2.3862 = 0
fast	0.2 * 1.6931 =0.33862	0.1666 * 1.6931 =0.2820	0 * 1.6931 =0	0 * 1.6931 =0
drive	0	0	0	0

Step 4: Calculate tf*idf for each query.

Query 1			
Term	TF	IDF	TF*IDF
formula	1	2.3862	2.3862

Query 2			
Term	TF	IDF	TF*IDF
sport	0.3333	2.3862	0.7953
fast	0.3333	1.6931	0.5643
drive	0.3333	0	0

Step 5: Calculate Cosine Similarity using below formula

Cosine Similarity(Query, Document) = Dot Product(Query, Document) / ||Query|| * ||Document||

1. **Calculate Cosine Similarity for Query1 and Document 1:**

   Cosine Similarity(Query1, Document1) = Dot Product(Query1, Document1) / ||Query1|| * ||Document1||
   Dot Product(Query1, Document1) = (2.3862 * 0) = 0
   Since dot product is zero the overall result will be zero, thus no need to calculate further.

2. **Calculate Cosine Similarity for Query1 and Document 2:**

   Cosine Similarity(Query1, Document2) = Dot Product(Query1, Document2) / ||Query1|| * ||Document2||
   Dot Product(Query1, Document2) = (2.3862 * 0) = 0
   Since dot product is zero the overall result will be zero, thus no need to calculate further.

3. **Calculate Cosine Similarity for Query1 and Document 3:**

   Cosine Similarity(Query1, Document3) = Dot Product(Query1, Document3) / ||Query1|| * ||Document3||
   Dot Product(Query1, Document3) = (2.3862 * 0) = 0
   Since dot product is zero the overall result will be zero, thus no need to calculate further.

4. **Calculate Cosine Similarity for Query1 and Document 4:**

   Cosine Similarity(Query1, Document4) = Dot Product(Query1, Document4) / ||Query1|| * ||Document4||
   Dot Product(Query1, Document4) = (2.3862 * 0.3975) = 0.9485

   ||Query1|| = sqrt($(2.3862)^2$) = 2.3862

   ||Document4|| = sqrt($(0.3975)^2$) = 0.3975

   Cosine Similarity(Query1, Document4) = 0.9485 / (2.3862 * 0.3975) = 0.9999 ~ 1

5. **Calculate Cosine Similarity for Query2 and Document 1**

   Cosine Similarity(Query2, Document1) = Dot Product(Query2, Document1) / ||Query2|| * ||Document1||
   Dot Product(Query2, Document1) = (0.7953 * 0) + (0.5643 * 0.33862) + (0 * 0) = 0.1910

   ||Query2|| = sqrt($(0.7953)^2 + (0.5643)^2 + (0)^2$) = 0.9751

   ||Document4|| = sqrt($(0)^2 + (0.33862)^2 + (0)^2$) = 0.33862

   Cosine Similarity(Query2, Document4) = 0.1910 / (0.9751 * 0.33862) = 0.5784

6. **Calculate Cosine Similarity for Query2 and Document 2**

   Cosine Similarity(Query2, Document2) = Dot Product(Query2, Document2) / ||Query2|| * ||Document2||
   Dot Product(Query2, Document2) = (0.7953 * 0.3975) + (0.5643 * 0.2820) + (0 * 0) = 0.4752

||Query2|| = sqrt($(0.7953)^2 + (0.5643)^2 + (0)^2$) = 0.9751

||Document2|| = sqrt($(0.3975)^2 + (0.2820)^2 + (0)^2$) = 0.4873

Cosine Similarity(Query2, Document2) = 0.4752 / (0.9751 * 0.4873) = 1

7. **Calculate Cosine Similarity for Query2 and Document 3**

Cosine Similarity(Query2, Document3) = Dot Product(Query2, Document3) / ||Query2|| * ||Document3||
Dot Product(Query2, Document3) = (0.7953 * 0) + (0.5643 * 0) + (0 * 0) = 0
Since dot product is zero the overall result will be zero, thus no need to calculate further.

8. **Calculate Cosine Similarity for Query2 and Document 4**

Cosine Similarity(Query2, Document4) = Dot Product(Query2, Document4) / ||Query2|| * ||Document4||
Dot Product(Query2, Document4) = (0.7953 * 0) + (0.5643 * 0) + (0 * 0) = 0
Since dot product is zero the overall result will be zero, thus no need to calculate further.

Thus, Result will be as below:

Cosine Similarity				
Query	Document 1	Document 2	Document 3	Document 4
Query 1	0	0	0	1
Query 2	0.5	1	0	0

Thus,
For Query 1 : formula , In Result Document 4 will be retrieved as a relevant document.
For Query 2: sport fast drive, In Result Document 2 and Document 1 will be retrieved as a relevant document.

If All documents are retrieved in the result, then ranking based on the cosine similarity will be as follows:

Query 1			Query 2		
Rank	Document	Cosine Score	Rank	Document	Cosine Score
1	Document 4	1	1	Document 2	1
2	Document 1	0	2	Document 1	0.5
3	Document 2	0	3	Document 3	0
4	Document 3	0	4	Document 4	0

Problem 3:

How storing term indexes in sorted manner helps in searching the terms? Give an example where sorted index can be searched quickly compared to unsorted.

Answer:

Organizing term indexes in a sorted manner is beneficial for efficient term searches because it enables faster lookup methods such as binary search. With sorted indexes, finding a particular term is simpler and quicker, as you can pinpoint its location without needing to scan through the entire index one item at a time.

For example, consider a scenario where you have an index of terms in a book stored in a sorted manner:

Sorted Index:

Car: Pages 5, 15, 25
Race: Pages 10, 20, 30
Sport: Pages 3, 8, 18

If we want to find the pages where "Race" is mentioned, we can use binary search on the sorted index to quickly narrow down the search. Starting in the middle at "Race," we would compare it with the target term. Since the index is sorted, we can determine whether "Race" comes before or after the current position. This process continues, effectively halving the search space with each comparison until the term is found.

In contrast, if the index were unsorted, we would have to search through the entire index sequentially to find the term, which would be much slower, especially for large indexes.

Problem 4:

Is stemming good in every situation? Explain.

Give a stemmed example query where the document 4 from question 2 will be retrieved because of stemming but the query has no relation with the document 4.

Answer:

Stemming, the process of reducing words to their base or root form, can be valuable in various scenarios, particularly in information retrieval and text analysis tasks. By converting different forms of a word into a common base form, stemming helps in grouping together variations of the same word, which can enhance search accuracy and text analysis efficiency. However, while stemming offers several benefits, it's not without its drawbacks. One potential issue is the loss of meaning that can occur when words are reduced to their stems. For example, the words "better" and "best" both stem to "good," which could lead to ambiguity or misinterpretation in certain contexts. Additionally, stemming algorithms may sometimes overstem or understem, resulting in incorrect word reductions. This can lead to mismatches between stemmed query terms and the content of documents, potentially resulting in the retrieval of irrelevant information. Therefore, while stemming can be a useful tool, it's important to consider its limitations and potential impact on the accuracy of search and analysis results.

Query:

Original query: "sports car race"
Stemmed query: "sport car race"
Result Example:

Document 1: "The sports car raced down the track."
Document 2: "I enjoy watching sports and cars racing."

In this example, if the search engine stems the query terms to "sport," "car," and "race," it might retrieve both Document 1 and Document 2. However, while Document 1 is relevant (it mentions all three terms in the context of a sports car race), Document 2 is not relevant to the user's intent of finding information about a "sports car race." This is because stemming has reduced "sports" to "sport," which matches "racing" in Document 2, even though the document is not about a "sports car race."

Let's take stemmed example query where the document 4 from question 2 will be retrieved because of stemming but the query has no relation with the document 4.

Original query: "No one can bet on this car for winning the race."

Stemmed query: "No one can bet on thi car for win the race."

In the stemmed query, "winning" is stemmed to "win," which matches the word "win" in document 4 ("formula race win major car bet"). This match occurs because stemming reduces words to their base form without considering their original context. As a result, document 4 may be retrieved even though it is irrelevant to the original query, which is about not being able to bet on a car for winning a race.

Problem 5:
Consider all the documents and 2nd query from question 4. Use formulas from the slides.

a. Calculate the TF-IDF scores of query with document 1 and 4 using natural (raw) term frequency count, no document frequency, and cosine similarity

b. Calculate the TF-IDF scores for all documents using logarithmic term frequency and inverse document frequency

c. Give the ranking for the query for all the documents. Show your calculations.

d. Explain with an example, the problems with using natural (raw) frequency

Answer:

a. Calculate the TF-IDF scores of query with document 1 and 4 using natural (raw) term frequency count, no document frequency, and cosine similarity

Calculate Term Frequency for document 1 and 4 using natural(raw) term frequency count

Document 1					
Term	fast	car	win	more	race
TF (Count)	1	1	1	1	1

Document 4						
Term	formula	race	win	major	car	bet
TF (Count)	1	1	1	1	1	1

Calculate tf of query with document 1 and 4

Term	Document 1	Document 4
sport	0	0
fast	1	0
drive	0	0

Calculate no document frequency using below formula
n(no) = 1

Term	df	idf
fast	2	1
car	4	1
win	4	1
more	1	1
race	3	1
sport	1	1
and	1	1
major	2	1
formula	1	1
bet	1	1

Calculate the TF-IDF scores
Since no document frequency is used tf-idf scores will be simply tf.

Term	Query	Document 1	Document 4
sport	1 * 1 = 1	0 * 1 = 0	0 * 1 = 0
fast	1 * 1 = 1	1 * 1 = 1	0 * 1 = 0
drive	1 * 1 = 1	0 * 1 = 0	0 * 1 = 0

Calculate cosine similarity

Cosine Similarity(Query, Document1) = Dot Product(Query, Document1)/ ||Query|| * ||Document1||

Dot Product(Query, Document1) = (1 * 0) + (1 * 1) + (1 * 0) = 1

||Query|| = sqrt($(1)^2 + (1)^2 + (1)^2$) = 1.7320

||Document1|| = sqrt($(0)^2 + (1)^2 + (0)^2$) = 1

Cosine Similarity(Query, Document1) = 1/(1.7320*1) = 0.5773

Cosine Similarity(Query, Document4) = Dot Product(Query, Document4)/ ||Query|| * ||Document4||

Dot Product(Query, Document4) = (1 * 0) + (1 * 0) + (1 * 0) = 0
Since Dot Product is 0,
Cosine Similarity(Query, Document4) will be 0.

b. Calculate the TF-IDF scores for all documents using logarithmic term frequency and inverse document frequency

Calculate logarithmic Term Frequency for each document using logarithmic term frequency
$Tf_w = 1 + \log (tf_{t,d})$

Document 1					
Term	fast	car	win	more	race
TF (Count)	1	1	1	1	1
TFW	1 + log(1) = 1	1 + log(1) = 1	1 + log(1) = 1	1 + log(1) = 1	1 + log(1) = 1

Document 2					
Term	sport	car	and	fast	win
TF (Count)	1	2	1	1	1
TFW	1 + log(1) = 1	1 + log(2) = 1.3	1 + log(1) = 1	1 + log(1) = 1	1 + log(1) = 1

Document 3					
Term	car	win	major	race	
TF (Count)	2	1	1	1	
TFW	1 + log(2) = 1.3	1 + log(1) = 1	1 + log(1) = 1	1 + log(1) = 1	

Document 4						
Term	formula	race	win	major	car	bet
TF (Count)	1	1	1	1	1	1
TFW	1 + log(1) = 1	1 + log(1) = 1	1 + log(1) = 1	1 + log(1) = 1	1 + log(1) = 1	1 + log(1) = 1

Calculate inverse document frequency using below formula

idf = log(N/df) where N = 4

Term	df	idf
fast	2	0.30
car	4	0
win	4	0
more	1	0.60
race	3	0.12
sport	1	0.60
and	1	0.60
major	2	0.30
formula	1	0.60
bet	1	0.60

Calculate the TF-IDF scores

Document 1					
Term	fast	car	win	more	race
TF	1	1	1	1	1
IDF	0.30	0	0	0.60	0.12
TF*IDF	0.30	0	0	0.60	0.12

Document 2					
Term	sport	car	and	fast	win
TF	1	1.3	1	1	1
IDF	0.60	0	0.60	0.30	0
TF*IDF	0.60	0	0.60	0.30	0

Document 3					
Term	car	win	major	race	
TFW	1.3	1	1	1	
IDF	0	0	0.30	0.12	
TF*IDF	0	1	0.30	0.12	

Document 4						
Term	formula	race	win	major	car	bet
TFW	1	1	1	1	1	1
IDF	0.60	0.12	0	0.30	0	0.60
TF*IDF	0.60	0.12	0	0.30	0	0.60

c. Give the ranking for the query for all the documents. Show your calculations.

Based on tf*idf scores calculated in question 5b, we can calculate cosine similarity between query and all documents for ranking as below:

Let's Calculate tf*idf scores for a query

Query					
Term	sport	fast	drive		
TFW	1	1	1		
IDF	0.60	0.30	0		
TF*IDF	0.60	0.30	0		

Query				
Term	Document 1	Document 2	Document 3	Document 4
sport	0 * 0.60 =0	1 * 0.60 =0.60	0 * 0.60 =0	0 * 0.60 = 0
fast	1 * 0.30 =0.30	1 * 0.30 =0.30	0 * 0.30 =0	0 * 0.30 =0
drive	0	0	0	0

Cosine Similarity(Query, Document1) = Dot Product(Query, Document1)/ ||Query|| * ||Document1||

Dot Product(Query, Document1) = (0.60 * 0) + (0.30 * 0.30) + (0 * 0) = 0.09

||Query|| = sqrt($(0.60)^2 + (0.30)^2 + (0)^2$) = 0.67

||Document1|| = sqrt($(0)^2 + (0.30)^2 + (0)^2$) = 0.30

Cosine Similarity(Query, Document1) = 0.09/(0.67*0.30) = 0.44

Cosine Similarity(Query, Document2) = Dot Product(Query, Document2)/ ||Query|| * ||Document2||

Dot Product(Query, Document2) = (0.60 * 0.60) + (0.30 * 0.30) + (0 * 0) = 0.45

||Query|| = sqrt($(0.60)^2 + (0.30)^2 + (0)^2$) = 0.67

||Document2|| = sqrt($(0.60)^2 + (0.30)^2 + (0)^2$) = 0.69

Cosine Similarity(Query, Document2) = 0.45/(0.67*0.69) = 0.97

Cosine Similarity(Query, Document3) = Dot Product(Query, Document3)/ ||Query|| * ||Document3||

Dot Product(Query, Document3) = (0.60 * 0) + (0.30 * 0) + (0 * 0) = 0
Since Dot Product is 0,
Cosine Similarity(Query, Document3) = 0

Cosine Similarity(Query, Document4) = Dot Product(Query, Document4)/ ||Query|| * ||Document4||

Dot Product(Query, Document4) = (0.60 * 0) + (0.30 * 0) + (0 * 0) = 0
Since Dot Product is 0,
Cosine Similarity(Query, Document4) = 0

Rank	Document	Cosine Similarity
1	Document 2	0.97
2	Document 1	0.44
3	Document 3	0
4	Document 4	0

d. Explain with an example, the problems with using natural (raw) frequency

A document with 10 occurrences of the term is more likely to be relevant than a document with 1 occurrence of the term. But not 10 times more likely. Relevance does not increase proportionally with term frequency. Using natural term frequency (TF) can cause problems in certain contexts, especially when it comes to information retrieval and text processing tasks.

Consider a sports news dataset with the following two documents:

Document 1: "The football match ended in a draw."
Document 2: "The cricket match ended in a tie."

Now, let's calculate the TF scores for the term "match" in each document using natural term frequency:

Document 1: "The football match ended in a draw."
Term frequency for "match": 1

Document 2: "The cricket match ended in a tie."
Term frequency for "match": 1

In this example, both documents have the same TF score for the term "match." However, the term "match" is more informative in Document 1 because it specifies the type of match (football), while Document 2 only mentions a generic "match" without specifying the sport. Using natural TF alone does not capture this difference in informativeness.

Another problem with natural TF arises when comparing documents of different lengths. Consider adding a third document:

Document 3: "The football match between the two rival teams ended in a draw."
Now, let's calculate the TF scores for the term "match" in each document:

Document 3: "The football match between the two rival teams ended in a draw."
Term frequency for "match": 1

In this case, Document 3 has the same TF score for "match" as Document 1, even though Document 3 provides more context and is potentially more informative due to the mention of "two rival teams." Natural TF does not account for the difference in document lengths, or the additional context provided in Document 3.

Overall, using natural TF alone can lead to misleading results, especially when comparing the importance of terms across documents of different lengths or when considering the specificity or informativeness of terms within a document.