RETRIEVAL EVALUATION

Report submission as a requirement for the module of "Information Retrieval Systems"

MSc Degree in Data Analysis
at the school of computing
Robert Gordon University
Aberdeen, Scotland
November 2020

Table of Contents

Search Engine Search Process

For this example, I will be querying the following text "Where did the black cat sat" in the IR system given for this coursework.

Let us have a look at the **inverted index,** which will be used to support our query. A total of **4** documents built the index; The total of terms found in the documents is **7** of which each contains its own **posting list**. Each posting list includes the document frequency and term frequency of each word.

```
Inverted File
next (df/1) => [(0,1)]
mat (df/3) => [(0,1)  (1,1)  (2,1)]
cat (df/2) => [(0,2)  (3,1)]
sat (df/3) => [(0,1)  (1,1)  (3,1)]
black (df/2) => [(1,2)  (3,1)]
wa (df/1) => [(2,1)]
dog (df/3) => [(1,1)  (2,1)  (3,1)]
Statistics
total docs= 4
Total No Index terms from Docs = 17
Number of posting lists = 7
Total number of postings = 15
```

Figure 1: IRDocuments Inverted index

As an example, we can pick the word **black**; this word is mentioned twice **(df/2)** in the list of documents in our database. The term is mentioned twice in document-2 **(1,2)** and once in document-4 **(3,1)**.

User Interface

The keyword/ search query "Where did the black cat sat" gets handled by the interface as a string which is introduced as a parameter in a method which aims to process the query in contrast with the inverted index showed above. In the context of our IR system, this method is called scoreIt.

```
public List queryIt ( Query theQuery ) {
        return weighter.scoreIt (theQuery );
}
```

Figure 2: User query ready to be scored

Text Indexing Operations

The next step deals with the query similarly as we do for the creation of our inverted index. **Some** of the techniques applied to our text include:

- **Tokenizing**: Divide the phrase into tokens (e.g. [Where], [did], [the]). It also allows for each of the words or group of words to be treated individually, allowing them to be stored on an iterable list which in turn can be used for a long-range of purposes such as stop word removal, stemming or weighting.
- **Stop words**: Allows for the removal of some words (e.g. Where did the black cat sat → Where black cat sat). It helps to reduce the amount of noise generated by the constant appearance of these words, which influence the score given to a document.
- **Stemming**: Reduce terms to their roots (e.g. automation → automat) and allow us to find a match between words easily.

Searching

Once our phrase has been pre-processed, we then proceed to look up each of our terms in the dictionary entries (i.e. **inverted index**) during the lookup we also count the **document frequency** which counts in how many documents a term appears. Finally, we get ready the results of our lookup and iterate though the retrieved values of the dictionary, in other words, the **posting list** of each of our terms; this is the final step before the weightings.

```
we= inv.lookupEntry( queryWord );

if ( we == null ) continue;
wordCnt=we.count();

qWeight= queryWeight( we );
pi= (MyPostingsIterator) we.postingsIterator();
while ( pi.morePostings() ) {
        post= (Posting) pi.getPosting();
```

Figure 3: Term Lookup Inverted index and posting list iteration.

Ranking

The next step is to get the term frequency from each term posting list, followed by a mathematical function that allows us to give a score to each document. Implementation of this mathematical function can be seen below.

```
double idf= Math.log10((double)inv.nDocs()/wordCnt);
double normTF=(double)mp.freq()/(double)minv.getMaxtf(mp.id());
double tf=(double)mp.freq();
// Performs the last calculations for determining the weight of the word
switch (theWf){
    case 1: res=1.0; break; // default weighting coordination level matching
    case 2: res= tf; break;//tf
    case 3: res= idf;break; // idf $3
    case 4: res= tf*idf;break;    //tf* idf
    case 5: res= normTF;break; //normalised tf
    case 6: res= normTF*idf;break;    //normalised tf* idf
    default: res=1.0;
}
return res;
```

Figure 4: Scoring documents with weighting functions

We use the following parameters within the function to score each document

- **Term Frequency** counts how many times the term in our search query appears in a document and computes a score.
- **Document Frequency** Scores for IDF. For TF*IDF, it helps by reducing the score given to words that appear too often in single documents.
- **Total number of documents** is used in combination with document frequency to weight the change in score. Log() is used to avoid weighting it too strongly.

Below an example has been output on how document 2 is scored based on our query by using **IDF** only(i.e. scoring function). Keep in mind that only the words black, cat and sat were a match to the dictionary.

$$black \rightarrow 0.30 = \log\left(\frac{4}{2}\right)$$

$$cat \rightarrow 0.30 = \log\left(\frac{4}{2}\right)$$

$$sat \rightarrow 0.12 = \log\left(\frac{4}{3}\right)$$

We can now add the results and get a total of 0.72 as our document score. In figure ## have output the result of the other documents where our terms were found.

Results

The last step is about outputting the results to the user. In this step, documents were scored and stored in a list with a corresponding ID. The list is then sorted from highest to lowest and returned to the user. Below we can see how this step is implemented in code.

```
                dWeight= documentWeight( post );
                scores[ post.id() ] += qWeight*dWeight;
            }
        }
        ArrayList scorepairs = new ArrayList();
        for(int i=0; i<scores.length; i++) {
            if ( scores[i] != 0.0 ) {
                scorepairs.add( new DocumentScore( i, scores[i] ) );
            }
        }
        Collections.sort( scorepairs, Collections.reverseOrder() );
        System.out.println( scorepairs.toString() );
        return scorepairs;
```

Figure 5: Ranking results and returning them to the user

Our final result is shown below; (document id three, which is document 4) has the highest score, and so Is ranked highest and shown as the first option to the user.

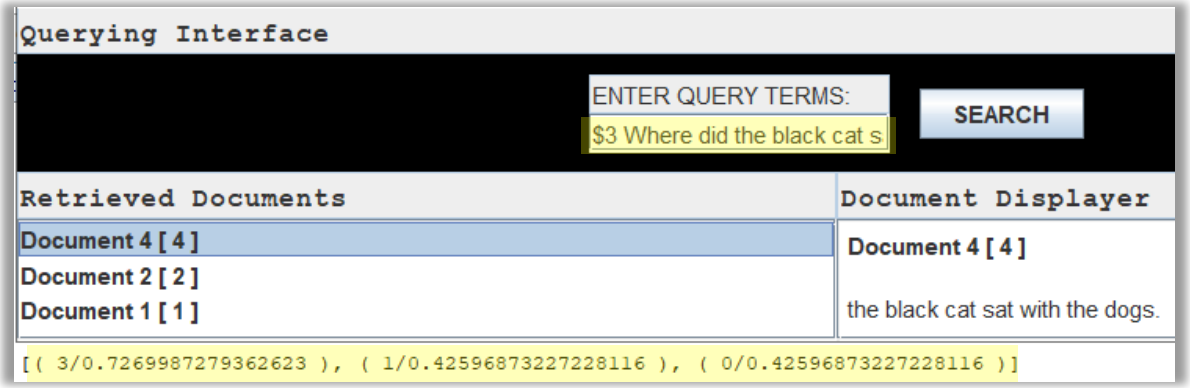


Figure 6: Final results / document scores

Inverted Index with and without Stemming

For this exercise, I have removed all documents from the IRDocuments folder and added a new file containing the following sentence: "dog dogs cat cats mat mats" autom automation. My goal is to prove how each group of words (e.g. cat, cats) is influenced by stemming and the construction of the inverted index.

I have added below two pictures representing both inverted indexes, **without stemming** in red and **with steaming** in green.

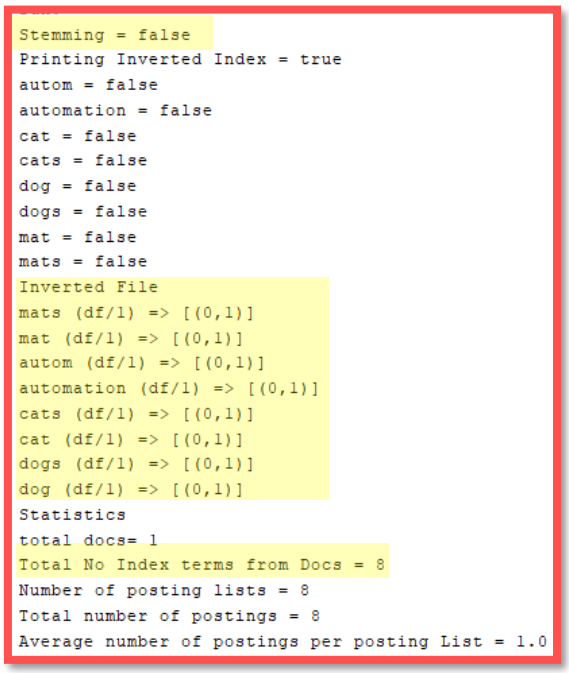


Figure 7: Inverted index without stemming

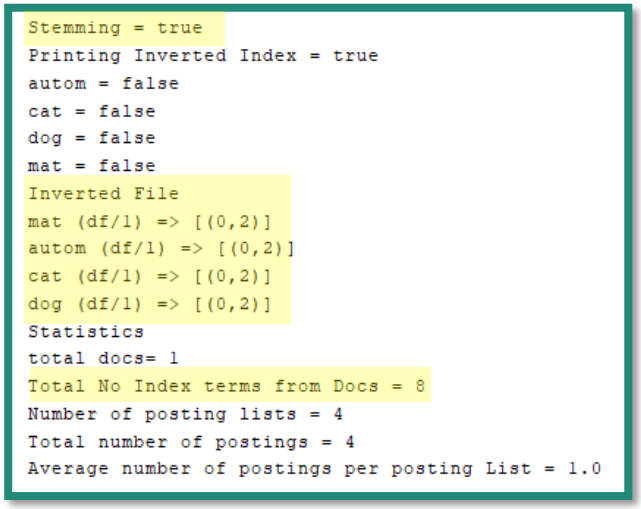


Figure 8: Inverted index with stemming

We can immediately see a difference in the size between both pictures; no stemming has increased its size, but more interestingly the statistics have doubled. The reason behind it is because stemming reduces each word to its roots allowing for dogs to be stored as dog and not as a separate entry. The disadvantages of turning off stemming is the impact we have on **storage space**; I have added only one file with a total of 8 terms, but we can already see that stemming reduces the size of the storage by half. The difference might not be that big in the example above, but when we are dealing with a gargantuan amount of data such as Google, we can easily find ourselves with an abysmal difference int terms of storage. Naturally, the **efficiency** of our search engine is improved when stemming is applied due to the decrease in

entries which means that our system will undergo fewer entries and posting list iterations when searching for the words found in our query allowing for this solution to be faster and computationally cheaper. In conclusion stemming reduces the storage required by the system and speeds up the search as well as **optimizes** the area of the main keyword search by including all similar but relevant terms.

Retrieval effectiveness for the specified queries

In this section, I will be comparing the **effectiveness** of the search engine with and without stemming. I will also discuss if stemming influences more the recall or precision of the system. There is a need for comparing recall and precision for this exercise, but because it is not known the number of false negatives that should be returned, I will set-up the evaluation in the following way.

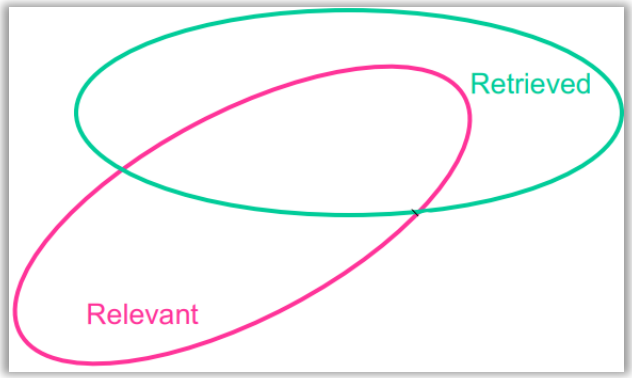


Figure 9: Retrieved and relevant document graph

For the number of relevant documents (i.e. the pink circle), I will be counting all the relevant documents that were retrieved, which relate to the search topic. In the case of "Understanding User Satisfaction" I have found documents (312, 261, 223, 206, 135, 726, 620, 6 and 52); **9 relevant documents**; "Visualization of Retrieval Results" found also **9 documents** (789, 728, 632, 620, 613, 583, 530, 528, 408). For the **green circle**, I will consider the top 10 documents retrieved.

Next, I will discuss the results of each table's precision at 5 and 10 documents as well as its overall recall. For referencing the **effectiveness** of each result in the table, I have evaluated them by choosing a threshold of 70% and based on it decided if the results are positive in green or below the threshold which would be negative in red. **Note**, I have judged the relevance of documents based on a majority vote bases.

After discussing the results in the tables and for comparison purposes, I decided to compute the average precision of P@ and recall with and without stemming. I will then compare the averaged results to define weather stemming affects precision and whether stemming influences recall or stemming the most.

Top 10	ID	R/N	Pre	Rec
1	312	R	100%	11%
2	261	R	100%	22%
3	223	R	100%	33%
4	206	R	100%	44%
5	**135**	**R**	**100%**	**55%**
6	761	N	83%	
7	726	R	85%	**66%**
8	71	N	75%	
9	634	N	66%	
10	**633**	**N**	**60%**	

Table 1: Understanding user satisfaction with stemming

Understanding user satisfaction with stemming

The total amount of documents retrieved was **305**. The query shows an outstanding result at P@5. The precision seems to greatly decrease after the 5th document showing a 60% result at P@10. Note that only 66% of the relevant documents have been returned.

Top 10	ID	R/N	Pre	Rec
1	312	R	100%	11%
2	261	R	100%	22%
3	206	R	100%	33%
4	135	R	100%	44%
5	**761**	**N**	**80%**	
6	726	R	83%	55%
7	71	N	71%	
8	633	N	62%	
9	620	R	66%	66%
10	**6**	**R**	**70%**	**77%**

Table 2: Understanding user satisfaction no stemming

Understanding User Satisfaction, no stemming

The total amount of retrieved document was **239**, lower than stemming which is to be expected. In contrast to **table 1** P@5 results show 20% decrease with no stemming. Also notice that when 50% of the documents were returned precision was at 83% which is 17% less than with stemming. At P@10 we saw an increase of 10% in both precision and recall.

Top 10	ID	R/N	Pre	Rec
1	223	R	100%	11%
2	206	R	100%	22%
3	634	N	66%	
4	620	R	75%	33%
5	**6**	**R**	**80%**	**44%**
6	556	N	66%	
7	535	N	57%	
8	528	N	50%	
9	52	R	55%	55%
10	**312**	**R**	**60%**	**66%**

Table 3: Understand user satisfaction no stemming

Understand User Satisfaction no stemming

A total amount of **233** documents were retrieved slightly lower than **table 2** this is due to stemming being off and finding less documents with the word understand and more with the word understanding. At P@5 we find similar results to **table 2** and at P@10 we find similar results to **table 1**.

Top 10	ID	R/N	Pre	Rec
1	789	R	100%	11%
2	728	R	100%	22%
3	632	R	100%	33%
4	620	R	100%	44%
5	**613**	**R**	**100%**	**55%**
6	583	R	83%	66%
7	530	R	85%	77%
8	528	R	87%	88%
9	408	R	**88%**	**100%**
10	**314**	**N**	**N/A**	**N/A**

Table 4: Visualization of retrieval results with stemming

Visualization of retrieval results with stemming

The total amount of retrieved document was **623.** At P@5 it returns a high score similar to the one we saw in **table 1**. I have decided not to include the results from P@10 and instead focus on P@9, this is because all relevant documents have already been retrieved with a precision of 88%.

Top 10	ID	R/N	Pre	Rec
1	789	R	100%	11%
2	632	R	100%	22%
3	613	R	100%	33%
4	530	R	100%	**44%**
5	**98**	**N**	**80%**	
6	97	N	66%	
7	88	N	57%	
8	82	N	50%	
9	80	N	44%	
10	**788**	**N**	**40%**	

Table 5: Visualization of retrieval results no stemming

Visualization of retrieval results no stemming

The total amount of retrieved document was **579**. P@5 shows the same score as in **table 2**. Unfortunately for this system we see a big decrease at P@10 of 30% in comparison o **table 2**. Only 44% of documents were found in the top 10 making it 33% less than in **table 2**.

Top 10	ID	R/N	Pre	Rec
1	632	R	100%	11%
2	408	R	100%	22%
3	91	N	75%	
4	88	N	50%	
5	**789**	**R**	**60%**	**33%**
6	774	N	50%	
7	748	N	42%	
8	713	N	37%	
9	709	N	33%	
10	**693**	**N**	**30%**	

Table 6: Visualization of retrieval result no stemming

Visualization of retrieval results no stemming

The total amount of retrieved document was **458**. P@5 shows a big decrease of 20% in precision compared to **table 3** and of 30% at P@10. Only 33% of relevant documents are found in the top 10 documents making it 33% less than in **table 3**.

Stemming results

The table below shows the average precision at document 5 and 10 as well as final recall of top 10 retrieved documents. Below is an example of the calculation of these values and the table showing the calculations.

- Final Recall No stemming = (77+66+44+33)/4
- P@10 No stemming = (70+60+40+30)/4

Measure	Stemming	No stemming	Conclusion on stemming
Average P@5	100%	75%	15% increase in precision with stemming
Average P@10	74%	50%	24% increase in precision with stemming
Average Recall	83%	55%	28% increase in recall with stemming

Table 7: Average results of both Systems

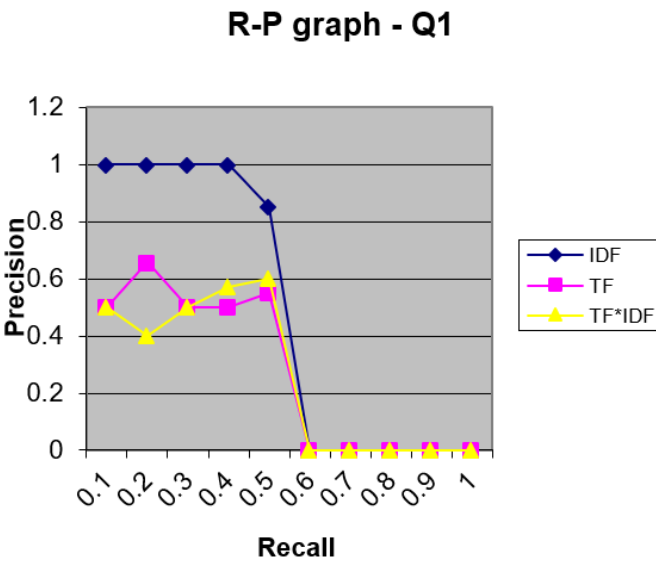
Note, The average of the table above is of P@ for both systems and not the average of the average precision of these systems.

The results on **table 7** show that stemming saw an increase of 15% in precision at P@5 compared with non-stemming. We can also see a constant trend since P@10 also saw an increase of 24% with stemming. It appears that the extra documents returned by stemming were very relevant to the query and ranked in the top 10. Based on these results, I conclude that stemming plays a role in the increment of precision for this system hence making it more **effective** than without stemming.

I was curious to understand which system loses precision faster than the other; for this, I have measured the differences of both systems between P@5 and P@10. Stemming shows a 26% decrease in precision, whereas no-stemming shows a 25%. I can conclude that non-stemming includes less irrelevant documents as our scope increases; this could be due to fewer documents being retrieved when stemming is not used.

Is stemming a recall or a precision enhancing device? Based on the results of the experiment, we can see that not only the **precision** effectiveness is increased by stemming but also **recall** by 28%. Stemming returns more documents allowing for recall to see an increase. Due to the increase on recall, I was expecting a decrease on precision, but if the judgment of relevant documents in this evaluation is correct, I would have to conclude that stemming enhances both recall and precision.

Weighting function calculations



All of the rank methods seem to have included the same number of relevant documents; this fact makes it clear for IDF to be the most **effective** weighting function. Our goal is to obtain a result which resembles a straight line while at a high precision, IDF achieve this behaviour almost flawlessly for the first top 5 documents.

Figure 10: Deep learning text retrieval

TF and TF*IDF did not follow a straight pattern line; inconsistency is a big problem for these systems because we want to provide our users with at least five relevant consecutive results. Another problem that we can see is that both of these systems have a low precision which makes them highly **undesirable** in comparison with IDF result.

IDF will score as high the words in our query, which appear less throughout the collection; This means that having five relevant documents ranked as top 5 is due to these documents containing the same terms as in our query which does not often appear in the collection.

I decided to go more in-depth and return the posting lists found in **table 8** from the IDF documents that were found by our query to back up my conclusions.

Terms	IDF term frequency									
Doc Ids →	311	242	0	136	7	290	231	302	21	269
deep			3							
learning			3							
text	1	3	2				3	2		
retrieval	3		1			1		2		1

Table 8: IDF search, term presence in documents

Term	Document frequency – Number of documents = 792	
deep	20	1.59
Learning / learn	151	0.71
Text	189	0.62
Retrieval / retrieve	434	0.26

Table 9: Document frequency and IDF result

Table 9 holds the IDF score for the words which appear in our query. I was planning on using this information together with **table 8** to compute why this weighting function affected so **positively** our query. I also included **table 10** to explain why TF affected our query **negatively** and in turn discuss TF*IDF

Terms	TF term frequency									
Doc Ids →	205	19	113	256	569	445	213	7	414	398
deep							4			
learning	3						5			
text		3		1		1				
retrieval				1			1			1

Table 10: TF term frequency in documents

Unfortunately, the inverted index in the statistics which are printed for this project is not mapped correctly to the document IDs. In other words, document 7 has the words (retrieval, learning and deep) appear eight times, but the posting list for this document shows 0. From here and on, I will have to address my conclusions based on assumptions backed by theory.

I want to conclude with the first query by stating that IDF is the most accurate result for this query not only as seen in **Figure 10** but also in the **average precision** found in **table 11**.

Weighting measure	Average Precision	Recall at 10
TF	0.24	0.45
IDF	0.53	0.54
TF*IDF	0.29	0.54

Table 11:Q1 Average precision and recall

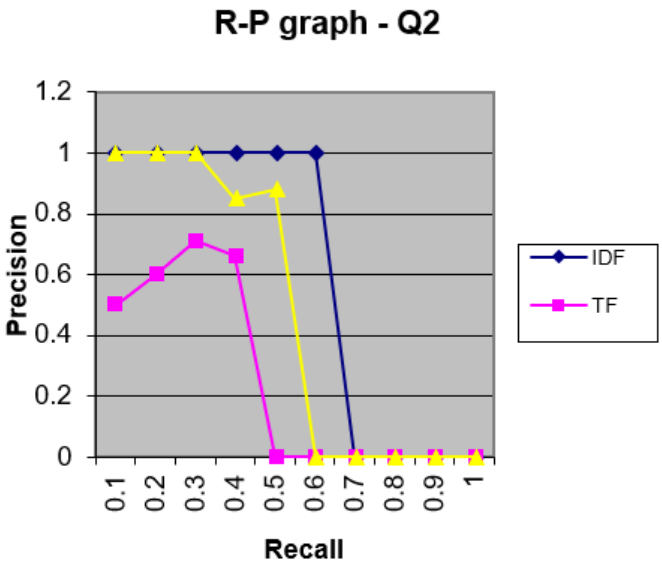


Figure 11: collaborative filtering recommender

For the second query we see IDF has included more relevant documents hence a greater recall but also the most effective for precision. TF is found to be far below the scale, this could be due to the terms mentioned in the query not appearing frequently enough in each of the documents in the collection. For TF*IDF we can see that it has benefit the most from the IDF section of the formula allowing it to obtain similar results as IDF but less due to the influence of TF

I would conclude that for the second query, we have found IDF to be more effective not only in **figure 11** but also in **table 13**. IDF also obtained a higher recall. When comparing both results (Q1, Q2) we can begin to see a pattern, IDF is usually at the top, tf*IDF and TF have less effectiveness and are located at the bottom.

Weighting measure	Average Precision	Recall at 10
TF	0.21	0.35
IDF	0.58	0.58
TF*IDF	0.50	0.52

Table 12: Q2 Average precision and recall

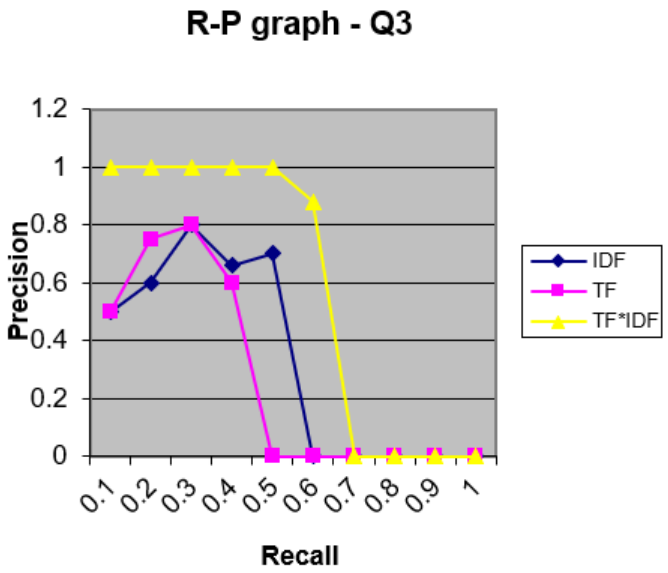


Figure 12: low dimensional feature space

The third graph shows TF*IDF as the most effective weighting function, I wasn't expecting to see it succeed this much because previous pattern shows that is usually found in between TF and IDF. Remember that most of the relevant documents could be poorly judge and I have found this to be the case in most of the material given for evaluating the systems in this project.

I can speculate that for this query in specific that the search engine found documents which did not have much term frequency nor unique words, this would score low on both TF and IDF and perhaps the documents were instead well balanced hence why were favoured by TF*IDF.

Weighting measure	Average Precision	Average Recall at 10
TF	0.28	0.45
IDF	0.36	0.54
TF*IDF	0.62	0.54

Table 13: Q3 Average precision and recall

Weighting Function Retrieval results

We have learned how each of the functions behaves, and based on this behaviour, we can make sense of the final graph. We know TF will rank as higher documents which contain terms in our query which are repeated the most in a single document. We learned IDF will rank top the documents which contain the terms in our query, which are unique throughout our collection. TF*IDF is a combination of these two functions and rises the argument that the result will be in between the previously mentioned functions. This conclusion is backed up by our average results showing TF*IDF (i.e. yellow line) in between the other two; This concludes that TF*IDF is a middle ground, between the two, for this project at least. We can see IDF has been a bigger influence for TF*IDF than TF was as its line is found closer to the blue line.

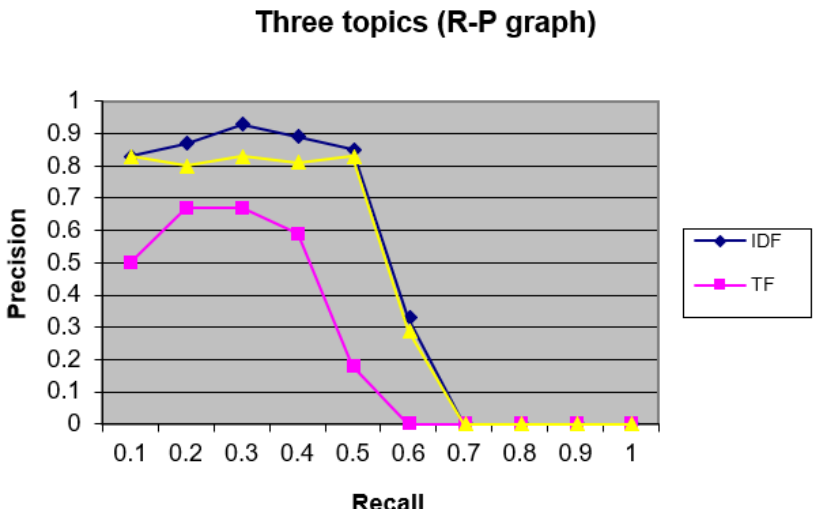


Figure 13: Q1, Q2, Q3 average result graph

I will also conclude that the way in which documents are written is a significant factor in how queries using different weighting functions retrieve them. The documents provided for this project are more suited for an IDF search as it yields not only the best precision but also the best recall, as seen in our final results in table 15. TF, on the other hand, will be the less effective function for these documents due to the papers not containing enough term frequency for the terms mentioned on our query.

Weighting measure	MAP	Average Recall at 10
TF	0.24	0.41
IDF	0.49	0.55
TF*IDF	0.47	0.53

Table 14: Q1, Q2, Q3 Median average precision

Beyond the scope of the project

After understanding that the format of a document determines how these functions respond, I was able to reach a deeper understanding of the topic and the project overall. For example, most people writing articles on the web try to eliminate redundancy; they do not want to mention the same word too often not to bore the reader. If the article talks about computers, then the author will try to use other terms such as machines, systems or use pronouns, whoever this is mostly true for short documents. Does this mean that short documents are less suited for a TF search? And more suited for an IDF search? This conclusion got me thinking about **research documents** which are hundreds of pages long and authors will not care so much about using alternate terms or pronouns for the topic of discussion, if the research paper is about neural networks then this phrase will be mentioned tens if not hundreds of times. This type of documents will have a high term frequency which means TF is going to see a benefit in long documents and will score this higher.

This insightful breakthrough pulled my attention to the project being discussed in this paper. I had a look at the **length** of the documents provided for the search queries and came across two incredible and exciting facts. First of all, the documents on the top 10 for TF are in most cases **longer** if not double in size to the ones found in IDF. Second and most insightful is to see that all of the 792 documents are **short** texts which is the reason why IDF scored higher than the other functions. This conclusion not only makes sense in theory but is also backed up by the results of the final graph, **figure 14** on this project.

My final thoughts on this project; It really depends what the user is searching for, if he is looking for research papers use TF, if the user is searching for short text, then use IDF and if the user is looking for something in between then use TF*IDF.