CS6370 - ASSIGNMENT 2

K Jothir Sashank, Rishabh Adiga, Nihal John George
EE19B137, EE19B135, EE19B131

March 2022

1 Inverted Index for Given Documents

Inverted index is a representation for each word, which shows what all documents a given word is present in. This method ignores the stop words as they do not give meaningful information to us. In python, it can efficiently represented as a dictionary, whose keys are words, while values are lists containing document IDs.
The inverted index representation for each of the words are as follows:

Herbivores	S1
Typically	S1 , S2
Plant	S1 , S2
Eater	S1 , S2
Meat	S1 , S2
Carnivores	S2
Deers	S3
Eat	S3
Grass	S3
Leaf	S3

2 TF-IDF Vectors for Given Documents

Consider the order of the words taken from all the documents, ignoring the stop words to be written in vector form taken as follows:

Herbivores Typically Plant Eater Meat Carnivores Deers Eat Grass Leaf

With this given order of words, the term frequency vector can be constructed for each of the documents

$$\text{S1} = \begin{pmatrix}1\\1\\1\\2\\1\\0\\0\\0\\0\\0\end{pmatrix} \qquad \text{S2} = \begin{pmatrix}0\\1\\1\\2\\1\\1\\0\\0\\0\\0\end{pmatrix} \qquad \text{S3} = \begin{pmatrix}0\\0\\0\\0\\0\\0\\1\\1\\1\\1\end{pmatrix}$$

For each of the words, the IDF - Inverse Document Frequency is calculated using the formula

$$\textbf{IDF} = \log\left(\frac{N:\ Total\ number\ of\ documents}{n:\ Number\ of\ documents\ containing\ the\ given\ word} \right)$$

Using the above formula to find IDF for each word which is not a stopword, we get:

Word	**IDF**
Herbivores	log(3) = 0.477
Typically	log(1.5) = 0.176
Plant	log(1.5) = 0.176
Eater	log(1.5) = 0.176
Meat	log(1.5) = 0.176
Carnivores	log(3) = 0.477
Deers	log(3) = 0.477
Eat	log(3) = 0.477
Grass	log(3) = 0.477
Leaf	log(3) = 0.477

We now find the TF-IDF matrix by multiplying the TF of each word in the documents with the corresponding IDF for each word:

Word	**S1**	**S2**	**S3**
Herbivores	0.477 * 1	0.477 * 0	0.477 * 0
Typically	0.176 * 1	0.176 * 1	0.176 * 0
Plant	0.176 * 1	0.176 * 1	0.176 * 0
Eater	0.176 * 2	0.176 * 2	0.176 * 0
Meat	0.176 * 1	0.176 * 1	0.176 * 0
Carnivores	0.477 * 0	0.477 * 1	0.477 * 0
Deers	0.477 * 0	0.477 * 0	0.477 * 1
Eat	0.477 * 0	0.477 * 0	0.477 * 1
Grass	0.477 * 0	0.477 * 0	0.477 * 1
Leaf	0.477 * 0	0.477 * 0	0.477 * 1

TF-IDF vector representation:

$$\text{S1} = \begin{pmatrix} 0.477 \\ 0.176 \\ 0.176 \\ 0.352 \\ 0.176 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} \qquad \text{S2} = \begin{pmatrix} 0 \\ 0.176 \\ 0.176 \\ 0.352 \\ 0.176 \\ 0.477 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} \qquad \text{S3} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0.477 \\ 0.477 \\ 0.477 \\ 0.477 \end{pmatrix}$$

3 Documents Retrieved for "plant eaters" using Inverted Index

- The query "plant" retrieves documents S1 and S2.
- The query "eaters" retrieves documents S1 and S2 as well.

The document S3 will not be returned as it does not contain either of the words. Hence S1 and S2 will be retrieved for the query - "plant eaters".
There is further nuance in terms of the Boolean operator combining separate words in a query ("plant AND eaters" vs. "plant OR eaters"). For this set of documents, the confusion is avoided since both words return the same set of docs.

4 Ranking of Documents using Cosine Similarity

Considering the query itself to be a document and finding the TF-IDF value of the given query - we have:

$$\text{TF vector} = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} \text{ IDF Vector} = \begin{pmatrix} 0.477 \\ 0.176 \\ 0.176 \\ 0.176 \\ 0.176 \\ 0.477 \\ 0.477 \\ 0.477 \\ 0.477 \\ 0.477 \end{pmatrix} \text{ TF-IDF Vector, query} = \begin{pmatrix} 0 \\ 0 \\ 0.176 \\ 0.176 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0. \\ 0 \end{pmatrix}$$

The formula to find the cosine similarity between a query and a given document is given by

$$\text{score}(\overrightarrow{document}, \overrightarrow{query}) = \frac{\overrightarrow{document} \;.\; \overrightarrow{query}}{|\overrightarrow{document}|\;|\overrightarrow{query}|}$$

Applying this formula to the above given TF-IDF form of the query with respect to

- Document S1 is $\dfrac{0.092928}{0.24890.6666} = 0.56$
- Document S2 is $\dfrac{0.092928}{0.24890.6666} = 0.56$

Hence Both the documents have the same cosine similarity of 0.56 when calculated - i.e taking the dot product of both the TF-IDF vectors and dividing by the modulus of the vectors. This gives the cosine similarity of the query and the documents.

5 Is the above ranking the best?

No,the ranking above is not the best.We get the same similarity value using cosine similarity for S1 and S2 , but we require S1 to have higher similarity than S2(and hence a better rank) to the query since S1 is talking about herbivores which are plant eaters whereas S2 is talking about carnivores and hence is logically less related to the query.

6 Building an IR System using Vector Space Model

The vector space model for the IR system has been implemented in the informationRetrieval.py file.

7 Inverse Document Frequency

a)What is the IDF of a term that occurs in every document?
The formula for IDF(Inverse document frequency) is:

$$IDF = \log(\frac{N}{n}) \tag{1}$$

Where N is total number of documents and n is number of documents in which the term/word occurs.
If the term occurs in every document, then n = N and hence:

$$IDF = \log(\frac{N}{N}) = \log(1) = 0 \tag{2}$$

Therefore, the IDF of a term that occurs in every document is 0.

b)Is the IDF of a term always finite? If not, how can the formula for IDF be modified to make it finite?
We know the formula for IDF is:

$$IDF = \log(\frac{N}{n}) \tag{3}$$

Consider a term/word that does not occur in any document. This implies n = 0 and hence:

$$IDF = \log(\frac{N}{0}) = \log(\infty) = \infty \tag{4}$$

Therefore, when the term does not occur in any document, the IDF is not finite.
This can be solved by adding 1 to the denominator. However this causes another problem – when a term is present in all documents, its IDF value becomes $log\frac{N}{N+1} < 0$. So we add 1 to the numerator also, to restrict the IDF to non-negative values. The final formula is

$$IDF = \log(\frac{N+1}{n+1}) \tag{5}$$

8 Other Similarity or Distance Measures – Compare with Cosine Similarity

Jaccard Similarity – Defined as Intersection-over-Union of the words present in the query and document. This measures what fraction of words are common between them.

$$J = \frac{|Q \cap D|}{|Q \cup D|} \tag{6}$$

The Jaccard similarity index does not consider repeated occurrences of words. Meanwhile, the union of words continues to increase for larger documents, while the intersection remains small due to the small query size. Hence, Jaccard index does accentuate the differences in word content between doc and query well enough, when considering number of occurrences of query words, and document size.

Minkowski Distance – For p = 1 and 2, it is known as Manhattan and Euclidean distance respectively.

$$D_p(\mathbf{q}, \mathbf{d}) = (\sum_{i=1}^{n} |q_i - d_i|^p)^{\frac{1}{p}} \tag{7}$$

Without normalising with vector lengths, Euclidean distance takes into account the frequency of words in the document. This can be problematic if the dataset contains documents with different sizes, as large documents will have long vectors, while queries will mostly be short. Since the search engine needs to return relevant documents regardless of size, Euclidean distance is not a good fit for IR. This argument holds for other Minkowski measures as well. **Reference:**

`https://en.wikipedia.org/wiki/Jaccard_index`,
`https://en.wikipedia.org/wiki/Minkowski_distance`

9 Why is accuracy not used as a metric to evaluate information retrieval systems?

Accuracy for an IR system is defined as follows:

$$Accuracy = \frac{A+D}{A+B+C+D} = \frac{True\ Positives + True\ Negatives}{Total\ Number\ of\ Documents} \tag{8}$$

where A,B,C,D are shown below:

	Relevant Documents	**Irrelevant Documents**
Retrieved Documents	A (True Positive)	B (False Positive)
Not Retrieved Documents	C (False Negative)	D (True Negative)

The value D which is True Negatives is very large in information retrieval systems for any given query because of the large size of the number of documents.Therefore, the number of True Negatives will be very close to the total number of documents(D is very large compared to A,B,C) and hence the numerator is almost the same value as denominator in the accuracy formula resulting in the accuracy value being very close to 1 for any IR system independent of how precise and relevant its retrieval method is.Basically, this metric is not sensitive to A,B,C values making it difficult to compare different systems.Hence it is not a good metric to evaluate IR systems.

10 Precision-Recall Weightage using α in the F_α Measure

The formula for F_α measure is given below:

$$F_\alpha = \frac{1}{\dfrac{\alpha}{precision} + \dfrac{1-\alpha}{recall}}; \quad \alpha \in [0,1] \tag{9}$$

- For $\alpha \in [0, 0.5)$, more weightage is given to recall
- For $\alpha \in (0.5, 1]$, more weightage is given to precision
- For $\alpha = 0.5$, both precision and recall are given equal weightage

Therefore for $\alpha \in [0, 0.5) F_\alpha$ measure gives more weightage to recall than to precision.
Note - For $\alpha = 0$, F_α is same as recall and for $\alpha = 1$, F_α is same as precision.

11 Shortcoming of Precision @ k metric, Addressed by Average Precision@k

Precision@k metric is calculated as the number of relevant documents out of the top k retrieved documents. It has a small flaw in evaluating a given system - that is if the relevant documents are retrieved at a higher rank, they get the same weightage as that of the relevant documents which are ranked low. Hence lower ranked relevant documents are counted the same way in P@k method.

$$\textbf{Precision@k} = \frac{Number\ of\ relevant\ documents\ in\ the\ k\ retrieved\ documents}{k}$$

The Average Precision@k method is used to resolve this issue. Average Precision@k is calculated by using a weighted mean of P@i's for i from 1 to k with respect to relevant documents. This ensures that the relevant document which is lowly ranked gets a high weightage and hence the system which retrieves a relevant document at a higher rank declared better with a higher AP@k value.

$$\textbf{Average Precision@k} = \frac{\sum precision@k * rel(i)}{Number\ of\ relevant\ documents}$$

Here rel(i) = 1 if i'th ranked document is relevant, else it is 0

For example, two systems A B retrieve 10 documents each where A's top 3 ranked documents are relevant and B's bottom 3 ranked documents are relevant - the P@k value is 3/10 = 0.3 for both, where as the AP@k value for A is (1/3)*(1/1+2/2+3/3) = 1 and AP@k value for B is (1/3)*(1/8+2/9+3/10) = 0.3639
It can be seen that the system which retrieves high ranked relevant documents will be at a lower average precision@k

12 What is Mean Average Precision (MAP) @ k? How is it different from Average Precision(AP)@k?

The Average precision@k is specific to a query, hence for different queries we get different AP@k values for the same system - hence for some queries the system can give high AP@k values where as for some other query it can give a much lower AP@k value. Hence this metric cannot be used to compare different systems.
The Mean Average Precision@k value takes a mean of AP@k over many queries for the given system and gives a universal value for a system. This particular value can be used to compare different systems directly to find out which ones are better, rather than comparing AP@k values for individual queries.

$$\text{MAP@k} = \frac{\sum AP@k\ (over\ all\ queries)}{Total\ number\ of\ queries}$$

This is better evaluation metric than AP@k

13 For Cranfield dataset, is AP or nDCG more appropriate and why?

The average precision@k is calculated by averaging out all the precision@i's for i from 1 to k. It measures the precision by taking the documents as either relevant or not relevant, but not as a percentage relevance. It gives us an estimate of the precision values overall based on the relevant document ranking for a given query. Another issue is that, it does not tell if one document is more relevant than the other, which is possible in a real life scenario.

$$\textbf{Average Precision@k} = \frac{\sum_{i=1}^{k} precision@i}{Number\ of\ relevant\ documents}$$

The nDCG metric considers this and evaluates the score while considering the relevance ranking as a criteria using the hierarchy of relevance, hence performs better in the Cranfield dataset.

$$\textbf{nDCG} = \frac{DCG}{iDCG}$$

where DCG (Discounted Cumulative Gain) $= \sum \frac{relevance_i}{log_2(i+1)}$
and iDCG is the DCG for the ideal ranking of documents.

14 Implement (a) Precision @ k (b) Recall @ k (c) F-Score @ k (d) Average Precision @ k (e) nDCG @ k

The evaluation metrics for the IR system has been implemented in the evaluation.py file.

15 For each query, find the Prec@k, Rec@k, F-score@k, AP@k and nDCG@k for k = 1 to 10. Report the graph and your observations

K	Precision	Recall	F-score	mAP	nDCG
1	0.658	0.110	0.182	0.658	0.439
2	0.558	0.184	0.260	0.608	0.440
3	0.484	0.229	0.290	0.567	0.440
4	0.440	0.268	0.308	0.535	0.443
5	0.389	0.292	0.308	0.506	0.446
6	0.361	0.325	0.316	0.482	0.456
7	0.339	0.352	0.319	0.461	0.464
8	0.321	0.378	0.321	0.444	0.475
9	0.300	0.395	0.316	0.428	0.482
10	0.284	0.411	0.311	0.413	0.487

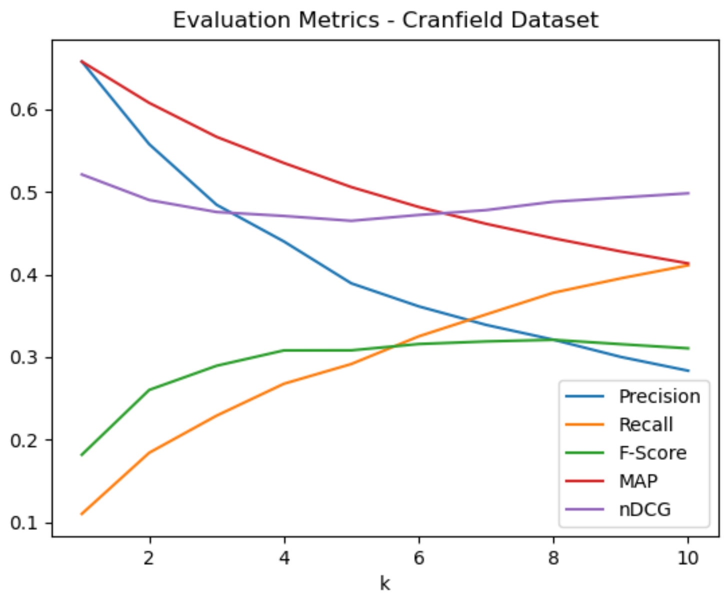

Figure 1: Plot of Metrics vs k

15.0.1 Observations

- **Precision** – Decreases with k, from 65% to 30%. So the search engine's fraction of relevant results among those retrieved decreases (in simpler words, it retrieves irrelevant docs). This happens since the cosine similarity could be close by for borderline relevant docs, while a clear decision can be made for the completely relevant/irrelevant ones. This allows noise words, synonyms and other effects to affect the decision. Note that precision can increase with k.
- **Recall** – Increases with k from 10% to 40%. Shows that more relevant documents are retrieved with increase in retrieval size. Monotonic increase of recall is always true since the denominator of recall does not change with k, while the number of relevant documents retrieved can only increase with k.
- **F-score** – Increases from 0.2 and becomes flat at k=4 around 0.4. Shows that the engine is able to improve its general performance upto 4 retrieved docs.
- **mAP** – Decreases with k. This happens since average precision is likely to decrease over k, since precision@k generally reduces with k.
- **nDCG** – Has a very slight increase. According to this metric, the engine is working well over each k.

16 Analyse results of search engine. Explain unexpected performance on certain queries

The search engine is able to give results that contain the same or similar stem words as the documents. The metrics obtained show the performance in this regard. But due to the absence of synonym resolution, and random sampling among equally ranked documents, the engine sometimes provides poor results. In general, writing abstract queries using words different from scientific literature (since Cranfield is based on aerospace engineering) will cause it to return docs 1400, 1399 etc Some unexpected results are given below –

- For queries whose words do not belong to the dataset (like "football"), the engine returns the last 5 documents (1400, 1399,...). This can be solved by randomly sampling documents instead.
- On searching for "aero", surprisingly, the engine returns only one document, that too it cites a journal whose name is "aero. sci. ". The engine is returning documents which have words starting with aero, like aeroplane, or aerodynamics. The engine can solve this using an adaptive stemming technique that checks for such prefixes apart from cosine similarity.
- Our engine does not handle synonyms. So on giving a query "a plane flies high", it returns documents having the word "plane", but referring to the flat surface. Proper synonym resolution is needed to prevent these.

17 Shortcomings in using a Vector Space Model for IR

The shortcomings of vector space model are listed below:

- The model is based on the assumption that all terms are orthogonal to each other.This is often not true as terms are related to each other in a lot of cases(for example all synonyms of a term are extremely similar and all antonyms are anti-similar).
- Since the documents are usually very large in terms of number of words, the similarity measures yield poor values since these values will be small because of small dot product values and high dimensionality and hence makes it harder to find the most relevant documents.
- This method of information retrieval has high wait times as it is calculation intensive and hence is not very time efficient.
- We have used unigram representation in the vector space here.Hence, the order in which words occur in the query or documents is not captured by the model.
- Even if we were to use bigrams(or ngrams for n>1), this method will not yield a good model because the number of dimensions increases drastically and this leads to drastically increasing need for computational power and hence results in increased processing times too.
- The vector space model is not modular as every time we add a new unseen term to the vector space, we need to recalculate all vectors which is a wastage of time and resource.

18 State a way to include the title while representing the document as a vector. How to weigh contribution of title three times that of the document?

We have used TF-IDF values in our vector space model to measure similarity of queries to documents. TF-IDF is a product of term frequency in a given document and the discriminative

power of the term which is calculated using all documents. This basically means that we are calculating weighted frequencies for each term in each document. Hence, the easiest method to include the title while representing the document as a vector is including the title as a sentence in the document and calculating the TF-IDF vectors for each document with this new corpus. For the Cranfield data set, we observe that the document sizes are not long and hence if we want to weigh the contribution of the title three times that of the document, we can just add 3 repetitions of the title in the documents which essentially corresponds to multiplying the TF-IDF representation of the title by 3 and adding to the documents TF-IDF.
Note - The above method will work only for the data set(document sizes are small) provided to us for this assignment because in reality, document sizes are very large and adding the title a few times will not increase the weightage of the title significantly. We need to either repeat the title a large number of times by looking at the number of sentences in the documents or try alternate methods to incorporate this.

19 Bigrams vs. Unigrams to index documents

The advantages and disadvantages of using a bigram instead of unigram to index documents are as follows.

Advantages:

- Word order or sequence is captured by bigrams but not unigrams which is the main limitation of the latter.
- Since bigrams consider sequence, it also captures contextual information.
- Due to the above points(sequence and context), bigrams have more precision in document retrieval compared to unigram.

Disadvantages:

- Using bigrams results in a drastic increase in dimensions of the vector space and results in more calculations and hence is computationally much more expensive.This will also result in increase in time taken for retrieval.
- We assume orthogonality in vector space model and this is even less likely to be true in the case of bigrams than for unigrams since we capture context and sequence in bigrams and hence bigrams with common words will be dependent on each other.
- The recall reduces when using bigrams compared to unigrams as it may not retrieve documents with similar words in the query due to surrounding words which do not match. The query may not contain the same bigrams because queries are usually just typed with a bunch of relevant words which need not make sense when read as a sentence(syntactically or grammatically incorrect queries which are still valid queries)

20 Getting Relevance Feedback from the User (passively)

Actively asking the users for feedback is a painful process for the users, so rather recording how the user behaves when the IR system gives the result and analysing the behaviour,relevant feedback can be collected. The following can be used to analyse the data:

- Post retrieving the results and displaying them to the user, the relevance can be sorted according to the fact - if the user has clicked on a given document or not. So the documents which users tend to click more will be more relevant in general and the ones which are not being viewed as much will be less relevant

- User’s browser/search history can be used to track and give relevant document retrieval to sort out polysemy. For example, the word ”park” when searched - the vehicle parking can be retrieved if the user’s search history is more of vehicles in general, else if the user’s search history is closer to nature then the document retrieval can be more towards walking/playing park.
- Users can be asked questions to find out how their mood is after the user spends a certain amount of time in various documents. If the user is happy then the documents retrieved can be classified as more relevant where as if the user is unhappy in general, it can be accounted for the documents being less relevant. This has to be done over a large quantity of users to get a better sense of data. Also, the weightage of this should be very less as the user’s mood depends on various other factors as well.
- Viewtime on documents can be used for relevance as well - the documents which have a high view time by the user will have a high relevance score, where as the documents which have a very low view time will have a low relevance score. Similarly, user searches, logins, purchases, feedbacks, forms filled etc can all be used to rank documents based on relevance.
- The areas of the screen where the user hovers his cursor on or the areas where the user spends his time can be categorised as more relevant.