CS6370 NLP ASSIGNMENT-2

Team No: 13

Team Members:

- **Mohammed Safi Ur Rahman Khan (CS21M035)**
- **Ganesh Jatavath (CS21M019)**
- **Abhishek Kumar (CS21M002)**

Q1. Now that the Cranfield documents are pre-processed, our search engine needs a data structure to facilitate the 'matching' process of a query to its relevant documents. Let's work out a simple example.

Consider the following three sentences:

S1 Herbivores are typically plant eaters and not meat eaters
S2 Carnivores are typically meat eaters and not plant eaters
S3 Deers eat grass and leaves

Assuming {are, and, not} as stop words, arrive at an inverted index representation for the above documents (treat each sentence as a separate document).

Ans: The Inverted index representation of the given 3 sentences or documents i.e, s1, s2, s3.

There are two types of inverted indexes:

1. Record-level inverted index 2. Word-level inverted index

Below table represents the frequency of word in each documents:

Words	**S1**	**S2**	**S3**
Carnivores	0	1	0
Deers	0	0	1
Eaters	2	2	0
Grass	0	0	1
Herbivores	1	0	0
Leaves	0	0	1
Meat	1	1	0
Plant	1	1	0
eat	0	0	1
Typically	1	1	0

Below table represents the words and its index with location within the text:

Words	(doc id, position of the word within the doc)
Carnivores	(2,1)
Deers	(3,1)
Eaters	(1,4)(1,6)(2,4)(2,6)
Grass	(3,3)
Herbivores	(1,1)
Leaves	(3,4)
Meat	(1,5)(2,3)
Plant	(1,3)(2,5)
eat	(3,2)
Typically	(1,2)(2,2)

According to sir's explanations, for an inverted index only doc id is enough. But we have also included positions by following the below reference.[1]

Q2. Next, we must proceed on to finding a representation for the text documents. In the class, we saw about the TF-IDF measure. What would be the TF-IDF vector representations for the documents in the above table? State the formula used.

Ans:

We have used the below formula for the TF-IDF vector representations for the documents.

TF-IDF score for term I in document j = TF(i,j) * IDF(i)

Where; IDF = Inverse Document Frequency

TF = Term Frequency

i = Term

j = Document

$$TF(i,j) = \frac{Term\ i\ frequency\ in\ document\ j}{Total\ word\ in\ document\ j}$$

$$IDF(i) = \log\left(\frac{Total\ Documents}{Documents\ with\ term\ i}\right)$$

Total Number of words in the document S1 = 6
Total Number of words in the document S2 = 6
Total Number of words in the document S3 = 4

D = Total number of documents = 3
dfi = document with term i

	TF	TF	TF				TF-IDF	TF-IDF	TF-IDF
Words	**S1**	**S2**	**S3**	**dfi**	**D/dfi**	**IDF**	**S1**	**S2**	**S3**
Carnivores	0	0.16	0	1	3/1=3	0.477	0	0.076	0
Deers	0	0	0.25	1	3/1=3	0.477	0	0	0.119
Eaters	0.33	0.33	0	2	3/2=1.5	0.176	0.058	0.058	0
Grass	0	0	0.25	1	3/1=3	0.477	0	0	0.119
Herbivores	0.16	0	0	1	3/1=3	0.477	0.076	0	0
Leaves	0	0	0.25	1	3/1=3	0.477	0	0	0.119
Meat	0.16	0.16	0	2	3/2=1.5	0.176	0.028	0.028	0
Plant	0.16	0.16	0	2	3/2=1.5	0.176	0.028	0.028	0
eat	0	0	0.25	1	3/1=3	0.477	0	0	0.119
Typically	0.16	0.16	0	2	3/2=1.5	0.176	0.028	0.028	0

3. Suppose the query is "plant eaters", which documents would be retrieved based on the inverted index constructed before?

Ans: Document 1 and 2 i.e, S1, S2 both will be retrieved with the same rank. Since the words of the query appear in only document 1 and 2 with equal frequency.

4. Find the cosine similarity between the query and each of the retrieved documents. Rank them in descending order.

Ans:

Given Query: **plant eaters**

	TF	TF	TF	TF					TF-IDF	TF-IDF	TF-IDF
Words	**Q**	**S1**	**S2**	**S3**	**dfi**	**D/dfi**	**IDF**	**Q**	**S1**	**S2**	**S3**
Carnivores	0	0	0.16	0	1	3/1=3	0.477	0	0	0.076	0
Deers	0	0	0	0.25	1	3/1=3	0.477	0	0	0	0.119
Eaters	1/2=0.5	0.33	0.33	0	2	3/2=1.5	0.176	0.088	0.058	0.058	0
Grass	0	0	0	0.25	1	3/1=3	0.477	0	0	0	0.119
Herbivores	0	0.16	0	0	1	3/1=3	0.477	0	0.076	0	0
Leaves	0	0	0	0.25	1	3/1=3	0.477	0	0	0	0.119
Meat	0	0.16	0.16	0	2	3/2=1.5	0.176	0	0.028	0.028	0
Plant	1/2=0.5	0.16	0.16	0	2	3/2=1.5	0.176	0.088	0.028	0.028	0
eat	0	0	0	0.25	1	3/1=3	0.477	0	0	0	0.119
Typically	0	0.16	0.16	0	2	3/2=1.5	0.176	0	0.028	0.028	0

Cosine similarity is computed by using the following formula:

$$similarity(A,B) = \frac{A \cdot B}{\|A\| \times \|B\|} = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \times \sqrt{\sum_{i=1}^{n} B_i^2}}$$

|S1| = 0.107
|S2| = 0.107
|S3| = 0.238
|Q| = 0.124

Q.S1 = 0.007
Q.S2 = 0.007
Q.S3 = 0

sim(Q, S1) = Q.S1/|Q|.|S1| = (0.007)/(0.124* 0.107) = 0.527
sim(Q, S2) = Q.S2/|Q|.|S2| = (0.007)/(0.124*0.107) = 0.527
sim(Q, S3) = Q.S3/|Q|.|S3| = 0

Rank them in descending order:
2 possible order: S1, S2, S3 or S2, S1, S3

5. Is the ranking given above the best?
Ans:
Yes and No. If we follow the first ranking i.e., S1, then the ranking is good because, for the query "plant eaters", probably the user is looking for "Herbivores" as the answer. But if we follow the second ranking, i.e., S2, then the ranking is not that good as it returns "Carnivores" as the first answer.

This is because of blind matching of the words without looking for anything else. This makes both S1 and S2 nearly similar.

6. Now, you are set to build a real-world retrieval system. Implement an Information Retrieval System for the Cranfield Dataset using the Vector Space Model.

Ans: The Code has been submitted along with this report

7. (a) What is the IDF of a term that occurs in every document?
(b) Is the IDF of a term always finite? If not, how can the formula for IDF be modified to make it finite?

Ans:
(a). Inverse document frequency (IDF) of word 'w' is computed using below formula:

$$IDF(w) = \log\left(\frac{Total\ no.of\ Documents}{No.of\ Documents\ contain\ word\ w}\right)$$

If the term occurs in every document, then IDF(w) becomes log(1). And it results in value 0. Inverse document frequency is a measure of how much information the word provides, whether the term is common or rare across all documents.

IDF(w) =0 means, it removes the meaning from TF*IDF.

Examples:
1. The word **"The"** appear in every document and it adds no meaning to knowledge.
2. The word **"data"** appears in almost all the computer science related documents and it adds no extra information.

(b). No, The IDF of a term will not always be finite. For suppose there are a finite number of documents(n) and we have created a Tf-Idf model with respect to this corpus. Then say we get a new query which has a word that's not part of the corpus vocabulary. For this, the idf will become infinity.
This can be solved by using smoothing.

$$\text{idf}(t) = \log \frac{1+n}{1+\text{df}(t)} + 1,$$

The smoothing formula taken from the reference [2]:

8. Can you think of any other similarity/distance measure that can be used to compare vectors other than cosine similarity. Justify why it is a better or worse choice than cosine similarity for IR.

Ans:
Manhattan Distance:
Alternate similarity/distance measure that can be used to compare vectors other than cosine similarity is Manhattan Distance. Manhattan distance is unbounded so the relative comparison between the two distances for the similarity is a very tedious process.

As we know cosine similarity is bounded between 0 and 1. Because of this it is easy to interpret the values as less similar or more similar. Because of the above reasons, the Manhattan distance is not as good as the cosine similarity of the Information Retrieval system.

Following formula is used to compute manhattan distance between two vectors x and y:

$$\text{Mdist} = |x2 - x1| + |y2 - y1|$$

9. Why is accuracy not used as a metric to evaluate information retrieval systems?
Ans:
The formula for accuracy is

$$Accuracy = \frac{(A+D)}{(A+B+C+D)}$$

Where;
A = Number of relevant documents retrieved
B = Number of relevant documents not retrieved
C = Number of non-relevant documents retrieved
D = Number of non-relevant documents not retrieved.

Here, typically, the number of irrelevant documents are quite huge in number as compared to the number of relevant documents for the query.
Example: In a setup where we have 1 million documents (1,000,000), for a query typically hardly 10-15 documents will be relevant. (Say 15).
Now even if our IRS system is terrible and say it retrieves 15 irrelevant documents, then our accuracy will be as follows.
A = 0
B = 15
C = 15
D = 999970

$$Accuracy = \frac{(0 + 999970)}{(1000000)} = 0.99997$$

According to the accuracy metric, our IRS system is performing brilliantly but in reality, its terrible. This is the main reason why accuracy is not a good measure for an IRS system.
In general, Accuracy is not a good metric for any model which has a huge class imbalance as one class tends to dominate over the other.

10. For what values of α does the Fα -measure give more weightage to recall than to precision?
Ans:

$$F_{alpha} = \frac{PR}{(1 - \boldsymbol{\alpha})\boldsymbol{P} + \boldsymbol{\alpha R}}$$

Here if we observe, F_{alpha} is higher when the denominator is less and the numerator is greater. Ignore the numerator since it doesnt effect the weightage wrt **α**.
Observe the denominator. For this to be less for a given recall value, it must give less weightage to recall and more weightage to precision.

Hence, for the range of alpha in 0 to 0.5, more weightage is given to Recall.

11. What is a shortcoming of Precision @ k metric that is addressed by Average Precision @ k?
Ans:
Average Precision is given by

$$\text{AveP} = \frac{\sum_{k=1}^{n} P(k) \times \text{rel}(k)}{\text{number of relevant documents}}$$

Here, **rel(k)** is an indicator function equal to 1 if the item at rank k is a relevant document, zero otherwise.
I.e., we consider only those precision values at those positions that have a relevant document.

Search Engines generally produce a ranking of relevant documents and not just an unordered set.
A simple precision @ k metric doesn't capture this ranking of documents. It just calculates out of the k retrieved documents, how many are actually relevant. Ideally we want the relevant documents to occupy good ranks.

This problem is resolved in average precision @ k metric since it averages the precision @ k values. If the relevant documents are given low ranks (i.e., good ranks), then precision @ k value continually remains high thereby giving a good average precision @ k value. Opposite will reduce the average precision @ k value.

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

Ans:

Mean Average Precision @ k is given by the formula

$$\text{MAP} = \frac{\sum_{q=1}^{Q} \text{AveP(q)}}{Q}$$

MAP value is actually calculated over a set of queries. It essentially captures the overall performance of the IRS system. It is actually the mean of the Average Precision @ k values for a group of queries.

Whereas the Average Precision @ k value is for a single query and MAP is the average over a group of queries.

This MAP value is higher for the IRS system that puts relevant documents before irrelevant documents consistently for various queries.

13. For the Cranfield dataset, which of the following two evaluation measures is more appropriate and why? (a) AP (b) nDCG

Ans:

The Average Precision or the Mean Average Precision (MAP) value considers only whether the documents are relevant or not. I.e., it just considers binary relevance.

Whereas in nDCG, we can have real-valued relevance scores also. I.e., nDCG also captures how relevant the documents are. It is high if highly relevant documents are given good ranks.

Coming to the Cranfield dataset, we are given a file called **"cran_qrels.json"** document which contains relevant scores for the documents given a query.

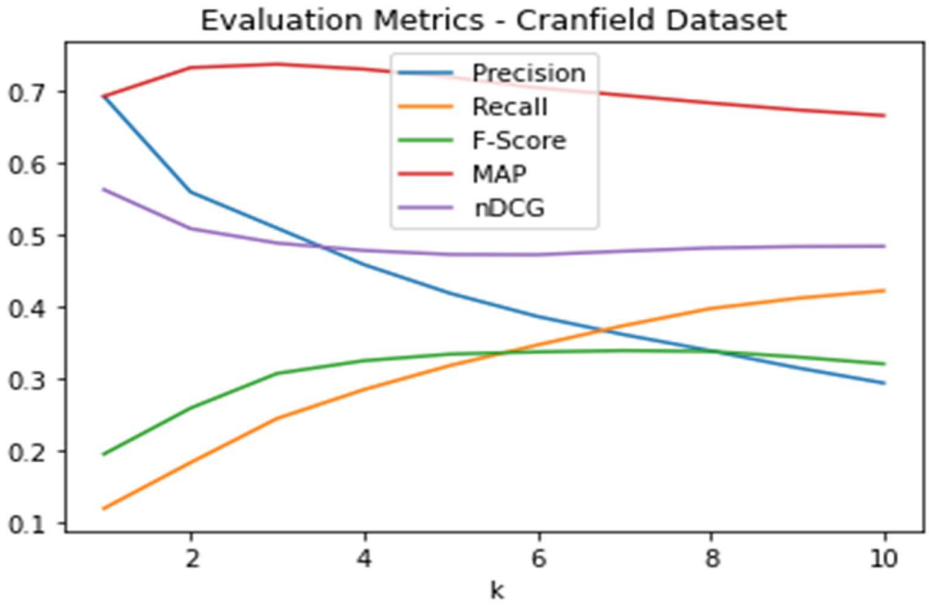
Hence nDCG will be a better metric for this since it can capture these relevant scores while evaluating the IRS system.
MAP can also be used but it can't exploit these relevance scores and just can capture whether relevant or not.

Therefore, nDCG is a better evaluation measure for the Cranfield dataset.

14. Implement the following evaluation metrics for the IR system:
(a) Precision @ k
(b) Recall @ k
(c) F-Score @ k
(d) Average Precision @ k
(e) nDCG @ k
Ans:
The code is submitted. We have coded the evaluation metrics from scratch without using any libraries.
For nDCG, since relevance order is required, we have modified the original relevance order to suit our need.

Originally, relevance order was 1 - Highly relevant, …, 4 - least relevant
Our new modified order is 4 - Highly relevant, …, 1 - least relevant, 0 - irrelevant.

15. Assume that for a given query, the set of relevant documents is as listed in "cran_qrels.json". Any document with a relevance score of 1 to 4 is considered as relevant. For each query in the Cranfield dataset, find the Precision, Recall, F-score, Average Precision and nDCG scores for k = 1 to 10. Average each measure over all queries and plot it as function of k. Code for plotting is part of the given template. You are expected to use the same. Report the graph with your observations based on it.
Ans:
The plot for different evaluation metrices as a function of k is as shown below:



Observations:

- The F-score is seen to increase initially and then stabilize attaining a more or less constant value (*around 0.3*). The F-score can be used to compare different models by observing the precision recall tradeoff. The maximum value observed for F-score is 0.318 at *k=8* and this information can be used to infer that the IR system with top 8 documents is the best for information retrieval.
- As expected, recall is monotonously increasing as k increases since more and more relevant documents are retrieved.[8]
- The Precision values are observed to decrease with an increase in the k values. Again, this is an expected outcome because as the number of retrieved documents increases, the density of relevant documents is bound to decrease.[8]
- The nDCG score is high initially but gradually smoothens near k=5-6
- If we observe, at around k = 8 the plots smoothen and don't tend to increase. So, this appears to be a good value for rank 'k'.

16. Analyse the results of your search engine. Are there some queries for which the search engine's performance is not as expected? Report your observations.

Ans:

The search engine's performance is not ideal but given the type of search engine we have built, it was expected to get such a performance.
Our search engine merely focuses on matching the words and not actually looking at the semantic closeness/similarity of the query with the documents.
This will hamper the performance considerably as user typically expects the search engine to fetch all the relevant documents (including those that may not have any common words but are considerably similar in semantic sense)

Some examples of the queries that have failed in our search engine are as below:

- For the query:
  "experimental techniques in shell vibration."
  The retrieved documents are having the doc IDs 848, 1036, 930, 908, 844.
  When we observe these documents, we find that they are nowhere related to the required information of the query. Instead they are just retrieved because of they have some common words.
- For the query:
  "basic dynamic characteristics of structures continuous over many spans."
  The retrieved documents are having the doc IDs 910, 660, 516, 1092, 998.
  The corresponding titles are: "natural frequencies of continuous beams of uniform span length. ", " the fundamental solution for small steady three-dimensional disturbances to a two-dimensional parallel shear flow. ", etc.
  Here also we observe some discrepancies in the output. The second document is not at all related to the query but is still retrieved.

- There are some cases where a word is immediately followed by special characters (like"/static/", "?couette-type?" etc.). These cases create unexpected results after tokenization, inflection reduction and affect the overall performance of the retrieval system since the query words often are not expected to be of the exact same form

(i.e. the tokenized and reduced word from document might be "/static/" but the query word would be simply "static").

Like these there are many discrepancies that make our IRS system not so good.

Q17. Do you find any shortcoming(s) in using a Vector Space Model for IR? If yes, report them.

Ans: Vector space model or term vector model is an algebraic model for representing text documents (and any objects, in general) as vectors of identifiers (such as index terms). It is used in information filtering, information retrieval, indexing and relevancy rankings. Here documents and queries are represented as vectors. Each dimension corresponds to a separate term. If a term occurs in the document, its value(term-weight) in the vector is non-zero. Weight values computed using the schemes called "tf-idf" weighting. Vector operations can be used to compare documents with queries.

Beside the vector space model has the advantages like "Simple model based on linear algebra", "Term weights are not binary", "Allows computing a continuous degree of similarity between queries and documents", "Allows ranking documents according to their possible relevance", "Allows partial matching".

It has many shortcomings(limitations) as follows:

1. Only some keywords which occur frequently are given importance over others.
2. Dependencies between words in both query and document are not taken care of as it is just a bag of words model.
3. Long documents are poorly represented because they have poor similarity values.
4. We are assuming words are independent dimensions which is not true since words also have relations between them.
5. Syntactical (grammar) contexts are not taken into account.
6. Suffers from synonym and polysemy.
7. Documents with similar context but different term vocabulary won't be associated, resulting in a false negative match.
8. The order in which the terms appear in the document is lost in the vector space representation that make it not suitable for modelling sequences of terms in documents.
9. The vector space model theoretically assumes that terms are statistically independent (orthogonality assumption) which is often not the case.
10. The vector space model is very computationally expensive involving a heavy number of intensive calculations. Also the vector space model is not very flexible in the sense that any new additions to the term vocabulary requires recalculation of all vectors thereby making it computationally expensive.

These are some shortcomings which we found using the refernces [9].

Q18. While working with the Cranfield dataset, we ignored the titles of the documents. But, titles can sometimes be extremely informative in information retrieval, sometimes even more than the body. State a way to include the title while representing the document as a vector. What if we want to weigh the contribution of the title three times that of the document?

Ans:

One way would be to just include the document title as a part of the document itself. The newly calculated TF-IDF would also include the information contained in the title. In case we

want to increase the weightage of the document title's contribution to three times that of the document, we can just include that many repetitions of the document title inside the document. (Although it is necessary that the document size is small enough, otherwise repeating the document title *x* number of times will not make much of a difference to a large sized document).

Q19. Suppose we use bigrams instead of unigrams to index the documents, what would be its advantage(s) and/or disadvantage(s)?

Ans:
An n-gram is a contiguous sequence of *n* items from a given sample of text or speech. an *n*-gram of **size 1 is referred to as a "unigram"**; **size 2 is a "bigram"**; **size 3 is a "trigram"**. When N>3 this is usually referred to as four grams or five grams and so on.
Formula to calculate number of N-grams in a sentence.
If X=Number of words in a given sentence K, the number of n-grams for sentence K would be:
N-grams = X - (N - 1)
Example: Sentence K= I want to learn Machine Learning

Unigram: here, X = 6 and N = 1 (for unigram)
N-gram = X - (N - 1) = 6 - (1–1)
= 6 (i.e. unigram is equal to number of words in a sentence)
[I], [want], [to], [learn], [Machine], [Learning]

Bigram: here, X = 6 and N = 2 (for bigram)
Ngram = X - (N - 1) = 6 - (2–1)
= 5
[I want], [want to], [to learn], [learn Machine], [Machine Learning]

Trigram: here, X = 6 and N = 3 (for trigram)
Ngram = X - (N - 1) = 6 - (3–1) = 4
[I want to], [want to learn], [to learn Machine], [learn Machine Learning]

Adavatage of Bigram over unigrams:

1) Unigram doesn't take the ordering of the words into account, so the order doesn't make a difference in how words are tagged or split up. While on the other hand Bigram care about the order of the words, so it considers the context of each word by analyzing it by pairs.
   In many instances the order of the words might not matter at all.

   Consider a sentence : "**This boat is going to sink!**"
   Here, the word "sink" can either be a verb or noun, depending on its context. In order to figure out which one it is, you have to check the words before or after. In the case above, the word "to" let's us know that it is, in fact, a verb. In a unigram model, that context would *not* be considered and it might be tagged incorrectly.

2) unigram features are not sufficient to discriminate the two classes of sentiments i.e positive sentiment and negative sentiment.

   Consider these two sentences.

No, this movie is good. **[Positive sentiment]**
This movie is no good. **[Negative sentiment]**

If you consider unigrams or the words as features, both of the sentences will have the same feature set as they contain the same set of words.
Here features are: no, this, movie, is, good.
In unigrams with bag of words, the ordering of the words is lost.
Hence unigram features are not sufficient to discriminate the two classes of sentiments.

But If you consider bigrams, the features would be "no this", "this movie", "movie is", "is good", "is no", "no good".
With this bigram approach, both the sentences will generate different features as there is some kind of ordering which is captured in the features.

3) One advantage of using bigrams over unigrams is some of the dependencies between words are captured over unigrams which may increase precision.

Disadvantage of Bigrams:

1) Disadvantage of using bigrams is number of dimensions will be very and most of them will be zero which might result reduced recall.
2) The recall obtained with bigrams would lower i.e. it may not retrieve documents that have similar words but different neighbour words.
3) As in N-gram, the bigger N, the bigger complexity to calculate the probabilities and establish the word vector space.
4) if tokens are independent then 1-grams work as good as n-grams.

Q20. In the Cranfield dataset, we have relevance judgements given by the domain experts. In the absence of such relevance judgements, can you think of a way in which we can get relevance feedback from the user himself/herself? Ideally, we would like to keep the feedback process to be non-intrusive to the user. Hence, think of an 'implicit' way of recording feedback from the users.

Ans:
The following could be potential ways to implicitly record feedback from the users:

1. One way for implicit feedback is recording which documents user is opening from our search results which can be considered as relevant documents in calculating AP, precision, recall, F-score.
2. Using the user's past search history like if user searched the topic "aerodynamics" would most likely be looking for topics related to the "aerodynamics" and "its related concepts".
3. Using the visits of a user to a document and time spent on that document i.e when a user makes a query, an initial set of relevant documents can be retrieved. Thereafter, depending on the number of times the user is visiting a subset of the retrieved documents, the results can be fine-tuned. As we collect more and more information about the number of visits of a user to documents, or the amount of time a user is spending on a document we will progressively have better results in terms of retrieved documents.

References:

1. https://www.geeksforgeeks.org/invertedindex/#:~:text=An%20inverted%20index%20is%20an,document%20or%20a%20web%20page.
2. https://scikit-learn.org/stable/modules/feature_extraction.html#text-feature-extraction/
3. https://mungingdata.wordpress.com/2017/11/25/episode-1-using-tf-idf-to-identify-the-signal-from-the-noise/
4. https://www.analyticsvidhya.com/blog/2021/11/how-sklearns-tfidfvectorizer-calculates-tf-idf-values/
5. https://stackoverflow.com/questions/35109424/how-to-make-tf-idf-matrix-dense/
6. https://www.delftstack.com/howto/python/cosine-similarity-between-lists-python/
7. https://blog.paperspace.com/mean-average-precision/
8. https://en.wikipedia.org/wiki/Evaluation_measures_(information_retrieval)/
9. https://en.wikipedia.org/wiki/Vector_space_model/