CS6370: Natural Language Processing

Assignment 2

Shubham Biren Patel, ME19B170 Bagul Amit Sunil, MM19B023

1. **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).
   Answer. The inverted index representation for the above sentences (documents) is stored in a python `dictionary` with keys storing the *type* and the values storing a `list` of documents to which the *type* belongs. $S1 := 0$, $S2 := 1$, $S3 := 2$

Type	**Documents**
Herbivores	[0]
typically	[0, 1]
plant	[0, 1]
eaters	[0, 1]
meat	[0, 1]
Carnivores	[1]
Deers	[2]
eat	[2]
grass	[2]
leaves	[2]

2. **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.**
   Answer. *Term frequency* (TF) is calculated by simply storing the number of times a term (type) occurs in a document. *Inverse document frequency* (IDF) is calculated as: IDF $= \log_2(N/n)$, where N is the total number of documents and n is the number of documents containing a given term (type). The TF-IDF values allow us to represent the documents as vectors in a space spanned by terms (types). Down below for $S1$, i iterates over different types and k = Total number of types.

$$\boxed{S1 = \sum_{i=1}^{k} tf_{S1}(i) \cdot idf(i) \cdot Type(i)}$$

Type	**Documents**	tf_{S1}	tf_{S2}	tf_{S3}	idf	$S1$	$S2$	$S3$
Herbivores	[0]	1	0	0	1.585	1.585	0	0
typically	[0, 1]	1	1	0	0.585	0.585	0.585	0
plant	[0, 1]	1	1	0	0.585	0.585	0.585	0
eaters	[0, 1]	2	2	0	0.585	1.170	1.170	0
meat	[0, 1]	1	1	0	0.585	0.585	0.585	0
Carnivores	[1]	0	1	0	1.585	0	1.585	0
Deers	[2]	0	0	1	1.585	0	0	1.585
eat	[2]	0	0	1	1.585	0	0	1.585
grass	[2]	0	0	1	1.585	0	0	1.585
leaves	[2]	0	0	1	1.585	0	0	1.585

3. **Suppose the query is "plant eaters", which documents would be retrieved based on the inverted index constructed before?**
Answer. From the table, it is evident that the types "plant" and "eaters" are common to documents $S1$ and $S2$. Hence, these two documents would be retrieved given the inverted index representation.

4. **Find the cosine similarity between the query and each of the retrieved documents. Rank them in descending order.**
Answer. Similar to the vector representations of the documents, the query Q = "plant eaters" can be represented as $[0, 0, 0.585, 0.585, 0, 0, 0, 0, 0, 0]^T$. The cosine similarity can be calculated as (say for $S1$):

$$\boxed{\text{sim}(Q, S1) = \frac{Q \cdot S1}{\|Q\| \, \|S1\|}}$$

Based on this equation we obtain $\text{sim}(Q, S1) = 0.560$, $\text{sim}(Q, S2) = 0.560$, $\text{sim}(Q, S3) = 0$. Hence:

$$\boxed{\textbf{Rank 1}: S1, \textbf{Rank 2}: S2, \textbf{Rank 3}: S3}$$

5. **Is the ranking given above the best?**
Answer. No. The ranking given above is not best. As query is related to plant eater, $S1$ is best match. But ($S3$) deer eating grass and leaves is more similar (related) to given query than $S2$. Hence the best ranking would be :

$$\boxed{\textbf{Rank 1}: S1, \textbf{Rank 2}: S3, \textbf{Rank 3}: S2}$$

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.**
Answer. Code Implementation: refer to `informationRetrievel.py`

7. **(a) What is the IDF of a term that occurs in every document?**
Answer. Using the formula mentioned in question 2, if a term occurs in every document, $n = N$ and hence IDF $= \log_2(N/n) = \log_2(1) = \mathbf{0}$.
(b) Is the IDF of a term always finite? If not, how can the formula for IDF be modified to make it finite?
Answer. If some term from given query doesn't appear in any of the documents, then n will be zero and IDF will not be define for this term. We can modify IDF to make it finite by smoothing it off. For instance, using *One Smoothing* where, 1 is added to both the denominator and the numerator to make IDF finite. Besides this, 1 is added to the logarithm term keep IDF non-zero.

$$\boxed{\text{IDF} = \log_2\left(\frac{N+1}{n+1}\right) + 1}$$

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.**
Answer. *Dot Product* seems like a good measure of similarity to compare vectors. If a query Q is at the same angle θ from documents $D1$ and $D2$ then both of them will be be equally likely candidates for retrieval based on *Cosine Similarity.* To break this tie and rank order such documents, dot product would work better for IR since it also accounts for the magnitude of the document vectors besides the angle. The *Distance* between vectors is yet another measure of similarity, but just like *Cosine Similarity*, it suffers from the same issue of equidistant documents.

9. **Why is accuracy not used as a metric to evaluate information retrieval systems?**
Answer. Accuracy is defined as the total number of correct predictions divided by total number of predictions made. Due to class imbalance, accuracy may give wrong evalution by getting overwhelm by one class (with more examples) over another (with less examples). It does not take into account varies different scenarios where minimising false negatives and false positives is of paramount importance. Because of these reasons accuracy is not used as a metric to evaluate information retrieval system.

10. **For what values of α does the F_α measure give more weightage to recall than to precision?**
Answer. We know,

$$\boxed{F_\alpha = \frac{1}{\alpha \cdot P^{-1} + (1-\alpha) \cdot R^{-1}}}$$

So, for $\alpha < 0.5$, recall is given more weightage than precision.

11. **What is a shortcoming of Precision @ k metric that is addressed by Average Precision @ k?**
Answer. Precision @ k metric uses first k retrieved documents and measures the proportion of relevant documents. The ranking is not involved in calculating Precision @ k. But for different applications, ranking of documents might have importance which is not captured by Precision @ k. Thus shortcoming of Precision @ k addressed by Average Precision @ k is giving importance to the rank of retrieved documents (more weight to earlier documents in rank) and calculating average for k retrieved documents.

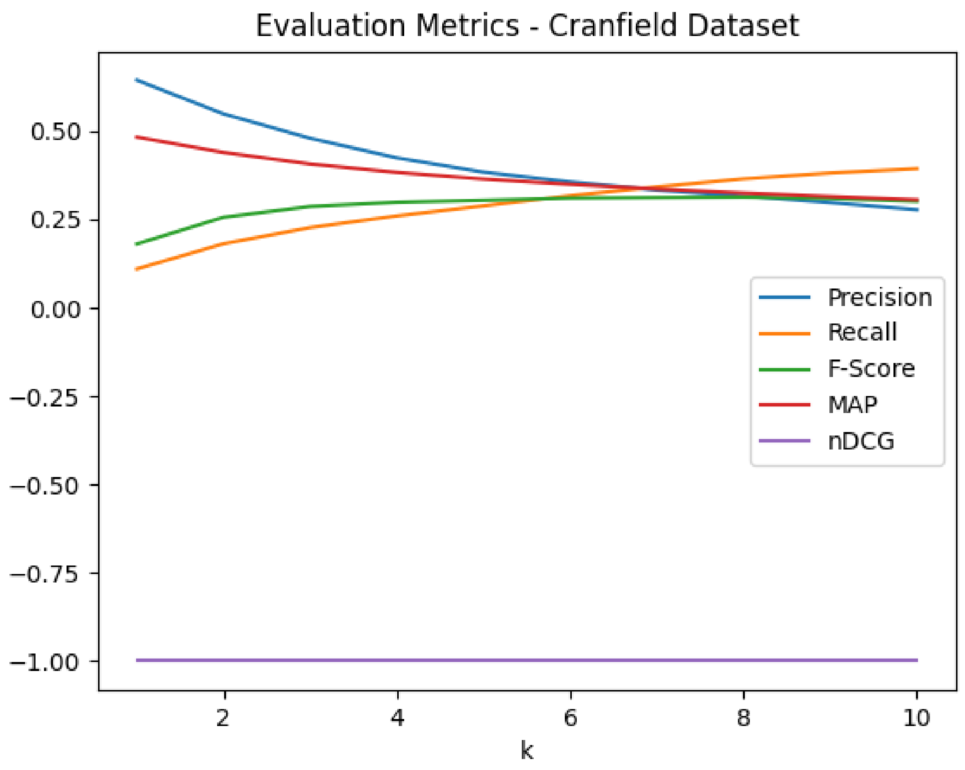
12. **What is Mean Average Precision (MAP) @ k? How is it different from Average Precision (AP) @ k?**
Answer. Mean Average Precision (MAP) @ k is a metric used to evaluate how good a ranking system is performing over different queries. For multiple queries, average of Average precision (AP) @ k values is MAP @ k. Thus the difference between them is that AP @ k is used to evaluate performance of system on one query whereas MAP @ k is used to evaluate performance of system across multiple queries by using (averaging) AP @ k on each of the query.

13. **For Cranfield dataset, which of the following two evaluation measures is more appropriate and why? (a) AP (b) nDCG**
Answer. (a) Average Precision (AP) is more useful in case documents are either relevant or non relevant (binary relevance). Whereas nDCG uses graded relevance judgments. Cranfield dataset uses binary relevance judgment, so AP is a more appropriate evaluation measure in this case.

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**

Answer. Code Implementation: refer to `evaluation.py`

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.**
Answer.



- As k increases that is number of relevant document increases, recall increases as number of retrievals are more and more.
- But the precision suffers as k increases. Due to imperfect system as more and more of documents are retrieved, number of irrelevant documents are retrieved increases resulting in decrease in precision.
- Decrease in MAP with increasing k is slower than precision. This denotes that system performs better when several queries are considered than on particular query as number of retrieved documents increases.
- F-score increases with k for small values and then nearly gets constant. F-score can be used to break precision-recall trade-off. The flattening in figure indicates that good number of documents have been retrieved.

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.**
Answer. The recall of system is good and gets better as number of retrieved documents increases.

It seems to give best result around $k = 7$ if all measures are to be taken into consideration. There are some term of queries with special characters (like for example "stress-strain" with'-') and terms with no spacing ("stabilityproblems") for which matching won't occur and search engines performance will get affected .

17. **Do you find any shortcoming(s) in using a Vector Space Model for IR? If yes, report them.**
    Answer. There are few/many shortcomings in using Vector Space Model. Some of them are listed as follows:

    - Dimensionality curse. As the number of documents and queries increase, it takes more storage and time to compute. It becomes computationally expensive.
    - Need of modifying and calculating all vectors for addition of new terms or documents.
    - Structure neglected. Order in which term and sentences appear is lost.
    - Polysemy. It does not take into consideration meaning of particular word in the given context. Query with polysemous words may get high similarity with irrelevant documents (having same words but different meaning) and thus may suffer from precision.
    - Synonymy. It does not identify different words that have same meaning as related words in similarity calculation. Thus it may give less similarity even for relevant documents if words are synonyms.

18. **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?**
    Answer.

    We can include the title of documents in the calculation of TF-IDF values. But the contribution of these words will be very less for long documents, so proper weighing will be the key factor to get better results. Just like smoothing, some weight can be transferred between the terms of doc to make title contribution three times. But to avoid overemphasis, better techniques will be required.

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

    Advantages:

    - Vector space representations improves, word order is retained thus system will be better at modeling sequence using two words.
    - The Precision improves.

    Disadvantages:

    - The Recall is penalised.
    - Dimensions to work with increases. Thus computation becomes slow.
    - Orthogonality assumption is not suited for bigrams.

20. **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.

Answer. To get relevance feedback from user, we can track engagement of user with the system's result. When query is not matched correctly, user might rephrase it and not sped any time with prev query. So time spent on result by user can be used to get relevance judgements from users.