CS6370 : NLP

ASSIGNMENT 1 - PART 2

TEAM MEMBERS :
EE17B001 - AKASH REDDY
AE17B010 - NAYAN N

Answer 1

Building an inverted index representation -

1. **Tokenization -** Tokenizing lowercase sentences/documents using nltk's word_tokenize to split each sentence into a list of words.
2. **Lemmatization -** Performing lemmatization on the obtained words to arrive at the base form of each word.
3. **Cleaning -** Removing stop words from each document's list of lemmatized tokens.
4. **Matching -** Collecting distinct tokens from the entire corpus and matching each token to every document in which it occurs in an inverted index.

```
leaf ['S3']
meat ['S1', 'S2']
plant ['S1', 'S2']
eater ['S1', 'S2']
eat ['S3']
carnivore ['S2']
typically ['S1', 'S2']
grass ['S3']
herbivore ['S1']
deer ['S3']
```

Inverted Index Representation

A working sample and the code for the above process can be found here.
Source - Toy Search Engine slides from class

Answer 2

To construct a tf-idf vector representation of the given corpus of sentences we employ the tf-idf vectorizer of the sklearn package. By default the vectorizer uses the smoothed version of the idf formula which is,

$$IDF(t,d) = 1 + log\left(\frac{N}{1+|\{d \in D : t \in d\}|}\right)$$

Here, 1 is added to the denominator to prevent the case when a query contains a word which is not present in the corpus. Additionally 1 is added to the log term to ensure that the idf score is always positive, greater than 0. The tf formula used is -

$$tf(t,d) = f_{t,d}$$

Where, $f_{t,d}$ is simply the raw count of the term "t" in document "d". The product of TF and IDF is taken to obtain an unnormalized tf-idf vector across the terms in the document. Finally, the tf-idf vector for each document is normalized by dividing by the L2-norm of the obtained vector.
The final tf-idf vectorized representation is as below -

	Tokens	S1	S2	S3
0	carnivore	0.000000	0.445047	0.0
1	deer	0.000000	0.000000	0.5
2	eat	0.000000	0.000000	0.5
3	eater	0.676940	0.676940	0.0
4	grass	0.000000	0.000000	0.5
5	herbivore	0.445047	0.000000	0.0
6	leaf	0.000000	0.000000	0.5
7	meat	0.338470	0.338470	0.0
8	plant	0.338470	0.338470	0.0
9	typically	0.338470	0.338470	0.0

tf-idf representation

Sources - tf-idf ; sklearn vectorizer
Code - Working found here

Answer 3

Using the inverted index representation of the corpus constructed in Answer 1, we find all relevant documents corresponding to the tokens in the query. The query sentence is passed through the same preprocessing steps, i.e. tokens are obtained by tokenization, each token is lemmatized, stop words are removed and distinct tokens in the query are retrieved.
For each token obtained after preprocessing we retrieve documents which contain these tokens.

```
eater ['S1', 'S2']
plant ['S1', 'S2']
```

Retrieved Documents for Query

We observe that documents S1 and S2 contain both tokens in the query, hence these documents are retrieved.

Code - (Note, the same packages from Answer 1 are used here)

Answer 4

To find the cosine similarity between the query and the documents, we use the tf-idf vectorizer fit on the given corpus from Answer 2. The tf-idf vector representation of the preprocessed query from Answer 3 is taken and the cosine similarity is found with each of the documents using the sklearn pairwise cosine similarity metric. The formula to find cosine similarity is as below,

$$k(x, y) = \frac{x^T . y}{||x||\,||y||}$$

Here,
x and y are the tf-idf vector representations of the document and query respectively
||.|| is the L2 norm of the vector
k(x,y) is the cosine similarity

The cosine similarities between the documents are shown below,

```
Cosine similarity between query and document S1 : [[0.71800303]]
Cosine similarity between query and document S2 : [[0.71800303]]
Cosine similarity between query and document S3 : [[0.]]
```

Documents S1 and S2 have equal scores, while S3 has no similarity with the query.
Sources - cosine similarity ; usage of cosine similarity metric
Code - found here

Answer 5

The ranking given above suggests that both documents S1 and S2 are equally relevant to the query given. This is expected as both documents contain the tokens in the query.

However, if we were to evaluate the query subjectively it would seem more likely that the query is trying to retrieve information about Herbivores which are exclusively plant eaters.
The IR system does fulfill this requirement but also provides extra information about carnivores stating that they are not plant eaters. This can be viewed as an advantage in the sense that the system is giving a more comprehensive answer but also as a disadvantage as it is not being as succinct as one would desire, and not ranking the actual most relevant result at the highest position by ignoring synonymy and word relatedness in the documents.

Answer 6
-Code in informationRetrieval.py-

Answer 7

(a) Using the inverse document frequency formula,

$$IDF(t,d) = log(\frac{N}{|\{d \in D : t \in d\}|})$$

Where,
t is a term from the query
D is all the documents in the corpus
N is the total number of documents in the corpus, N = |D|
$|\{d \in D : t \in d\}|$ is the number of documents where term "t" appears

If there is a term which appears in every document, then the denominator of the IDF formula will be equal to N. Hence the IDF will be log(1) which is 0. This is representative of the function of the IDF part in the tf-idf score. Terms which appear in most documents do not provide any discriminating benefit across documents in information retrieval, and thus have a low score.

(b) The formula used above will not result in a finite IDF score if the term in a query does not appear in any document in our corpus. In this case, the denominator would be 0 leading to an infinite IDF score. To counter this, we make use of the smoothed inverse document frequency formula which is,

$$IDF(t,d) = 1 + log(\frac{N}{1 + |\{d \in D : t \in d\}|})$$

In this case the denominator will have a value of 1 even if the term does not appear in any document thus ensuring that the IDF score is always finite.
In the case that a term occurs in every document, the denominator will be N+1 giving us a negative log value. To ensure that we always get a positive idf score in order to smoothen out the tf-idf values from falling suddenly to 0 for some terms, we add 1 to the log term.

Answer 8

A possible method of measuring vector similarities is to calculate and compare euclidean distances. This implementation, however, would be subpar to cosine similarity in most scenarios.

Let us assume that we are using a bag of words vector representation where we create a vector space model with each perpendicular axis being one word from a vocabulary of distinct words. Also, instead of using TF-IDF as the vector component magnitude, we use simply the term frequency. Each document is represented in this vector space model by having a vector along each axis with a magnitude equal to the number of times a word appears in the document.

Now, let us assume we have a collection of documents which speak about cricket. In all of these documents it would be fair to assume that the frequency of the word cricket would be very high, meaning that the magnitude of the vector along the “cricket” axis in our vector space model would be high.
If we type a query like “cricket ball”, there could be documents which speak about cricket in a different context, say the insect which has a lower euclidean distance to the query than documents which speak about the sport.
However in the case of cosine similarity, since we are using a normalized metric effectively measuring the angle between the document and the query, we would retrieve documents which are more relevant.

Answer 9

In a general information retrieval system, which operates on a diverse corpus, most documents in the corpus will not be relevant to a given query. In most circumstances the data is extremely skewed (Class Imbalance) where a negligible percentage, of the order 0.1-0.01%, of the documents are relevant to the query.

A system training to maximize accuracy which tries to retrieve relevant documents would still inevitably try to reduce false positives and negatives, causing the accuracy to drop. However if the system results in a high number of **true negatives,** for example by simply providing all irrelevant or random documents for every query, it achieves an accuracy score of 99.9 - 99.99% thus giving us a system which does not perform information retrieval but exhibits a really high accuracy score by maximising true irrelevant documents (true negatives which are completely useless to a user looking for information). Hence accuracy is not a good metric to evaluate an information retrieval system.

Answer 10

The formula for F_α is as follows -

$$F_\alpha = \frac{P*R}{\alpha*R + (1-\alpha)*P} \qquad \alpha \in [0, 1]$$

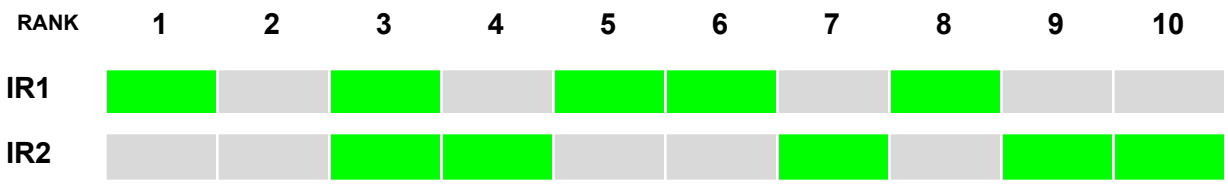
Here, when $\alpha \in [0,\ 0.5)$, the F_α measure gives more weightage to recall than to precision. For example, $F_0 = R$(completely weighted towards recall).

Answer 11

The Precision@K metric evaluates the performance on an IR system when it returns a fixed “K” number of documents. It is defined as:

$$\frac{Number\ of\ relevant\ documents\ out\ of\ K}{Number\ of\ total\ documents\ returned\ (K)}$$

This gives us an overall sense of how relevant all the results returned are. **However, Precision@K does not care for the ranking of the results.** For example, consider the following pages of length 10 (assume K=10) of returned results for a particular query from two different IR systems:



(Green documents are relevant, grey documents are irrelevant.)

Both IR1 and IR2 have a Precision@10 value of 5/10 = 0.5 and are considered equally good by the Precision@K metric. However, from an IR system perspective (such as a search engine), IR1 is better as it ranks relevant results higher.

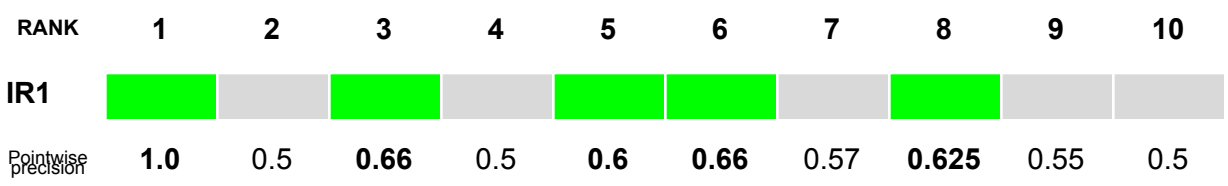
Average Precision@K addresses this shortcoming of Precision@K. Average Precision@K is calculated as follows:

$$\frac{\sum_{k=1,\, k \in X}^{K} \frac{Number\ of\ relevant\ documents\ out\ of\ k}{Number\ of\ total\ documents\ returned\ (k)}}{|X|}$$

where X is the set of all indices/ranks at which a relevant document occurs. Therefore, the precision value at each index where a relevant document occurs is calculated, and the average of these values is returned.

Therefore, an IR system that tries to maximise Average Precision(AP)@K will also try to maximise the Precision@k values for smaller values of k<K. As a result, IR systems that rank relevant documents higher will have a higher AP@K value for the same Precision@K value.

Evaluating the AP@K values of the same toy example used earlier:



IR2										
	0	0	**0.33**	**0.5**	0.4	0.33	**0.43**	0.375	**0.44**	**0.5**

AP@10 of IR1: $\frac{1.0 + 0.66 + 0.6 + 0.66 + 0.625}{5}$= 0.709

AP@10 of IR2: $\frac{0.33 + 0.5 + 0.43 + 0.44 + 0.5}{5}$= 0.44

thereby returning IR1 as the better IR system. Thus, AP@K allows us to take ranking of relevant results into consideration.

Answer 12

Mean Average Precision(MAP)@K is different from Average Precision(AP)@K in that, AP@K is the Average Precision value calculated as discussed above in Answer 11 **for a single query.** When AP@K is calculated for the IR system on each query in the dataset, and the mean across all of them is taken, we obtain MAP@K. Hence, MAP@K is an overall metric of evaluation of an IR system across all the queries at our disposal.

Answer 13

nDCG is more appropriate than MAP for evaluating an IR system built on the Cranfield Dataset, since the dataset has relevance scores of the documents **on a graded scale**. nDCG takes advantage of these relevance scores and ranks documents in a better manner from a user perspective.

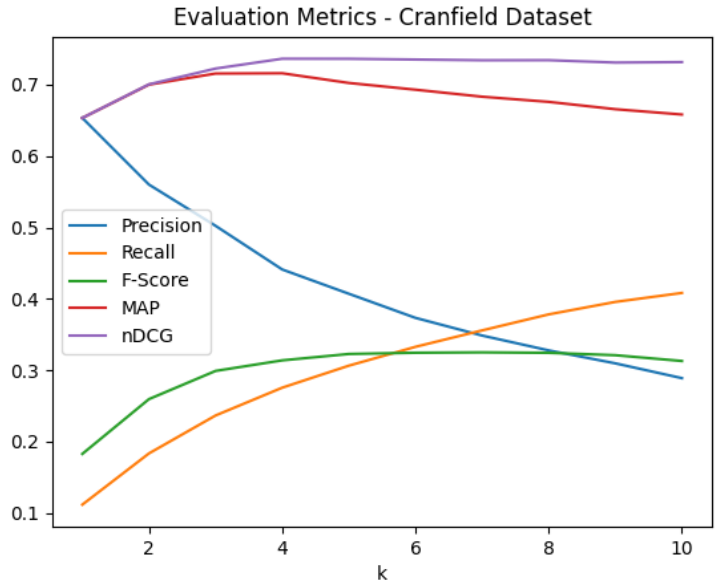
It does better than MAP which uses binary relevance scores (0 - not relevant, 1 - relevant) to do the ranking, without utilising the degree-of-relevance information which the Cranfield Dataset provides. We are choosing to assume that any document with a relevance score is relevant, which may not be the best threshold to demarcate relevance or lack of it. Even if we choose to assume that relevance score <=2 is not relevant, and >2 is relevant, we are still manually picking a threshold and introducing bias into the evaluation metric, while throwing away the finer-grained relevance information we have.

Answer 14

-IMPLEMENTED IN CODE : evaluation.py-

Answer 15

Below is the plot of Evaluation Metrics@k (averaged across all queries) vs k.



For the calculation of nDCG, we need relevance SCORES (4 being most relevant, 1 being least relevant). The IDCG is subsequently obtained by taking the descending order of scores and computing the DCG. However, in cran_qrels.json, the value of the key “position” is a relevance RANK (1 is most relevant, 4 least relevant) as gathered from the README.txt file provided in the Cranfield Dataset folder.

Therefore, we have calculated relevance scores as (5 - relevance rank) for the calculation of nDCG.

Some observations from the graph:

- With increasing k, the value of mean precision decreases and the value of mean recall increases.

  When we consider the plots for a single query, the precision@k value may increase or decrease when k is increased by 1, depending on whether the new considered document is relevant or not. Hence, the precision value can fluctuate upwards or downwards. However, in the usage of IR systems, a lot of the documents available in the database are unrelated. Therefore, as we evaluate precision on more and more number of top results, we tend to capture more irrelevant documents and the precision has a decreasing tendency overall when we increase k. When we average over many queries, the fluctuations that we observe in a single query are smoothened out, while the decreasing tendency of the precision@k curve which is common to all queries tends to stand out and remain.

  Similarly, for recall@k of a single query, when k is increased by 1, the recall either increases or stays the same (when the new document is relevant and irrelevant respectively). Therefore recall never decreases with increasing k, however the increases

happen as abrupt steps in the curve, leading to a staircase-like piecewise monotonically increasing function. But when we average over many queries, these sudden steps of increase of recall get averaged out, and the increasing tendency of the recall@k curve remains.

- The F-score@k curve first increases and then decreases with increasing value of k. F-score is the harmonic mean of the precision and recall. Therefore, the maximum value of this metric should be achieved at the point when precision and recall are equal. At this point F-score = precision = recall, if we calculate F-score directly from the precision and recall values.
  However, we are calculating the mean F-score by calculating the mean of the individual query-wise Fscores (and not the harmonic mean of the mean precision and mean recall). Therefore, we will not get a perfect intersection point of the Fscore curve with the precision and recall curves. We can still see the increasing-decreasing tendency of the Fscore curve, and the maximum value occurring at the intersection point of precision and recall curves. The maximum Fscore achieved at this point is also very close to the precision value (or equivalently recall value).
- MAP@k and nDCG@k both give higher values than these 3 metrics. At k=1, they usually start at a value equal to the precision@1, but they grow as k increases from 1 (instead of decreasing).
- MAP@k reaches a maximum value for a certain value of k (around k=4), and then starts decreasing as relevant documents start showing up in the later indices when a higher k is considered, and they bring down the precision value calculated at those ranks, which are considered for the average precision calculation. This is a consequence of the fact that all relevant documents are not showing up relatively higher in the ranking.
- nDCG@k on the other hand, reaches a maximum value for a certain value of k and does not fluctuate much from there. It stays above the MAP curve. This is because the influence of each new document that is involved is discounted by a reciprocal logarithmic factor proportional to the rank of the result. Therefore, the influence of each new document on the nDCG metric decreases with increasing k. Also, nDCG considers the finer-grained relevance scores of the documents, instead of a binary relevant/irrelevant flag. Therefore, this metric places more emphasis on strongly relevant documents occurring in the topmost ranks, and is less easily influenced by later results than MAP is.

Answer 16

Upon printing individual query numbers and the corresponding top k documents returned, we observed that some queries are indeed eliciting particularly bad results from our IR system. For example, query ID 216 : "what investigations have been made of the wave system created by a static pressure distribution over a liquid surface .", returns all top 10 results of documents that are not relevant (they don't even have the smallest relevance score possible!). The returned document IDs are [904, 958, 764, 905, 1156, 39, 175, 407, 971, 367].

We see the same phenomenon with query ID 87: "what effect has the boundary layer in modifying the basic inviscid flow behind the shock, neglecting effects of leading edge and corner ." The returned document IDs are [1228, 334, 26, 439, 797, 655, 569, 1198, 707, 526].

We compare with a query that does well (query ID 88: "how does a satellite orbit contract under the action of air drag in an atmosphere in which the scale height varies with altitude ." which returns 7 out of 10 relevant results, in positions 1, 3, 4, 5, 6, 7, 8). We also look at the relevance information in cran_qrels.json, and we make the following observations for queries that don't do well:

- They either have very few documents that are relevant to them (such as query 216 which has only 2 relevant results). In this case, the probability of the IR system missing these few results and not returning anything relevant is high.
- Or, they have few relevant documents which are not highly relevant to the query. In the case of query ID 87, there are 8 relevant documents with relevance ranks 2,2,3,4,4,4,4,1 (lower average relevance, high number of 4s). The well-performing query ID 88 has ranks 2,2,2,3,3,4,1 (higher average relevance, high number of 2s and 3s). In such cases as query ID 87, it is easy for an IR system to mistake not-so-relevant documents as irrelevant due to sparser indication of relevance within the document. Vaguely relevant documents can even often carry similar words as the query with a somewhat related discourse, not so often the exact same words and phrases with a tight relation to the query. Since the Vector Space Model of the IR system does not take synonymy into account, it is easy to miss out on low-relevance documents.

Answer 17

Yes, there are shortcomings with the Vector Space Model.

- The order of the sequence of terms in the document and in the query are neglected when documents are retrieved. The Vector Space Model is a bag-of-words-based approach. Therefore, two documents with the same words would be represented by the same vector in the vector space, but could mean different things based on the ordering. The example described below in Answer 19 demonstrates such a situation.
- Word Relatedness is not considered in the vanilla Vector Space Model. Various words are treated as completely orthogonal (or statistically independent) to each other in the vector space, and the document is represented as a vector with components along these orthogonal axes. However, in the real world it is most often the case that sets of synonymous/related words exist in the corpus, whose axes would ideally have an angle less than 90 degrees between them - this synonymy is not modelled in the Vector Space Model. This means that the IR system misses out on very relevant articles that simply use words different from the query words. This reduces recall.
- On the other hand, polysemy is also not handled by the Vector Space Model. If the same term T is used in different senses in different documents D1, D2,..., Dn, all these documents D1 through Dn are returned as relevant to a query which contains the term T.

However, some documents may be using the term T in a polysemous sense, and they may not be truly relevant to the query. For example, the word “bank” may mean a financial bank or a river bank. The Vector Space Model can thus result in low precision when it returns financial-bank-related documents when a query is looking for river-bank-related documents.

Answer 18

A way to include the title while representing documents as vectors, giving equal weightage to both, is by appending each title to its document and treating this as the “new effective” document for which to calculate the TF-IDF vectors, in the new vector space based on all terms in the “new effective” corpus with title terms included.

In order to weigh the contribution of the title three times that of the document, we can append three copies of the title to the document instead of one. This increases the term frequency (TF) of the terms in the title, therefore the vector representing the document grows along these specific term-axes due to the three-fold increased contribution of the title. However, the IDF values remain the same as the case where we append the title only once, since the IDF is calculated simply based on the **presence** of a term in a document, and not the intra-document frequency. Therefore, this is an effective way to weigh the contribution of the title three times that of the document.

Answer 19

The approach of using bigrams in building the index model would involve collecting sequential pairs of words and using these pairs as indices for our vector space model.

Advantages
This approach tackles one very evident problem in the unigram indexing model which is that it is agnostic to the ordering of words in a document. This is best explained with an example -
Let us consider two documents D1, D2 -

D1 : “Jack eats with a fork”
D2 : “Fork eats with a Jack”

Vector Representation in unigram model :

D1 : {“Jack”, “eats”, “with”, “a”, “fork” }
D2 : {“Fork”, “eats”, “with”, “a”, “Jack”}

Both of these documents have the exact same vector representation in the unigram indexing model, where it is evident that D2 would be less preferred in an IR system.

However, if we were to consider a bigram indexing model, the two documents would be decomposed as such -

D1 : {“Jack eats”, “eats with”, “with a”, “a fork”}

D2 : {“Fork eats”, “eats with”, “with a”, “a Jack”}

Here, the IR system sees a clear distinction between the two documents, which would improve its performance by reducing the bag-of-words basis of the IR system, and increasing the likelihood of returning more relevant documents based on sequence of words.

Disadvantages

If the query typed by the user has only one word or if the bigram representation of the query does not exist in the indexing model, then no relevant document would be found. There are higher chances, in the bigram model, of the query not having the right bigram match even though the individual words in the query might be available in the document set. A possible fix to this would be to take a weighted score of both the scores in the unigram index representation and the bigram index representation and retrieve the documents with the highest weighted score.

Answer 20

Considering a situation where we do not have relevance scores of documents given by domain experts we can implicitly obtain these scores from a user as follows -

A user, in a search engine setting, upon typing a query would be given a list of documents which our Information Retrieval system deemed most relevant. Glancing through the titles of the list of documents and their contents, the user would click on documents which they think are most relevant. We can have a system which records the documents that were clicked on by the user and the order in which they were clicked. For a certain query, we can store the frequency of a document being clicked, as a measure of user-deemed relevance. We can also combine into this metric whether the document was clicked first or second or later (order of click), from the search results page.

This user-based relevance metric can be combined with the rank of the document in the IR system’s results. The IR system rank of a document is a measure of word-based similarity that we obtain from methods such as vector space modelling and word-relatedness.

Therefore, a document that is frequently clicked by users that search a certain query, **and** also has high word-based similarity with the query, has a very high relevance due to the combination of both factors. To compare, a document with frequent clicks but not that much word-based similarity, or a document with high word-based similarity but not that many clicks by users, are possibly not as relevant. (The former would still likely be more relevant than the latter here. Since users are the best judges of relevance of a document, we can weigh the contribution due to the user-deemed relevance higher.)

Another factor of consideration could be the time that a user spends on a document. If the user exits the document quickly, it could mean that the document does not contain relevant information whereas if the user spends more time in a retrieved document it could mean that the

document has the information that they were looking for. The time spent could be recorded and used as a scoring metric along with the order in which documents were clicked.

More the number of users that are exposed to the search engine, stronger will be the user-based relevance scores. Ultimately we can use a weighted average of the vector-space-model-based relevance scores and user-based relevance scores while retrieving documents.

****