CS6370: Natural Language Processing
Project

Release Date: 21st March 2024 Deadline:

Name:	Roll No.:
Sanket Singh	EE21B118
Riya Mahesh	EE21B112
Ishaan Agarwal	EE20B046

General Instructions:

1. The template for the code (in Python) is provided in a separate zip file. You are expected to fill in the template wherever instructed. Note that any Python library, such as nltk, stanfordcorenlp, spacy, etc, can be used.
2. A folder named 'Roll_number.zip' that contains a zip of the code folder and your responses to the questions (a PDF of this document with the solutions written in the text boxes) must be uploaded on Moodle by the deadline.
3. Any submissions made after the deadline will not be graded.
4. Answer the theoretical questions concisely. All the codes should contain proper comments.
5. For questions involving coding components, paste a screenshot of the code.
6. The institute's academic code of conduct will be strictly enforced.

The first assignment in the NLP course involved building a basic text processing module that implements sentence segmentation, tokenization, stemming /lemmatization, stopword removal, and some aspects of spell check. This module involves implementing an Information Retrieval system using the Vector Space Model. The same dataset as in Part 1 (Cranfield dataset) will be used for this purpose. The project is split into two components - the first is a *warm-up* component comprising of Parts 1 through 4 that would act as a precursor for the second and main component, where you improve over the basic IR system.

[Warm up] Part 1: Working out a toy IR system [Numerical]

Consider the following three documents:

d_1: Herbivores are typically plant eaters and not meat eaters
d_2: Carnivores are typically meat eaters and not plant eaters
d_3: Deers eat grass and leaves

1. Assuming {are, and, not} as stop words, arrive at an inverted index representation for the above documents.

Assumption: Terms with multiple occurrences in the same document are not treated differently.

Ignoring the case. Inverted index representation is as follows:

Word	Document
herbivores	d1
typically	d1, d2
plant	d1, d2
eaters	d1, d2
meat	d1, d2
carnivores	d2
deers	d3
eat	d3
grass	d3
leaves	d3

2. Construct the TF-IDF term-document matrix for the corpus $\{d_1, d_2, d_3\}$.

TF-IDF term-document matrix:

Word	d1	d2	d3
herbivores	0.477	0	0
typically	0.176	0.176	0
plant	0.176	0.176	0
eaters	0.352	0.352	0
meat	0.176	0.176	0
carnivores	0	0.477	0
deers	0	0	0.477
eat	0	0	0.477
grass	0	0	0.477
leaves	0	0	0.477

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

Query - "plant eaters"

Based on the inverted indexing constructed in Q1, retrieved documents:

d_1: Herbivores are typically plant eaters and not meat eaters
d_2: Carnivores are typically meat eaters and not plant eaters

4. Find the cosine similarity between the query and each of the retrieved documents. Is the result desirable? Why?

Cosine Similarity calculations:
TF-IDF representation of query:

Word	TF-IDF
herbivores	0
typically	0
plant	0.176
eaters	0.176
meat	0
carnivores	0
deers	0
eat	0
grass	0
leaves	0

$$\cos\theta_1 = \frac{Q \cdot D1}{|Q|*|D1|} = \frac{0.176*0.176 + 0.176*0.352}{0.249*0.667} = \boldsymbol{0.559}$$

$$\cos\theta_2 = \frac{Q \cdot D2}{|Q|*|D2|} = \frac{0.176*0.176 + 0.176*0.352}{0.249*0.667} = \boldsymbol{0.559}$$

$$\cos\theta_3 = \frac{Q \cdot D3}{|Q|*|D3|} = \frac{0.176*0 + 0.176*0}{0.249*0.954} = \boldsymbol{0}$$

Ranking documents:

Cosine similarity order: $1 = 2 > 3$

Retrieval order: $1 \simeq 2 \succ 3$

Is the ordering desirable? If no, why not?:

Based on cosine similarity, documents d1 and d2 are retrieved as they contain both the terms ‘plant’ and ‘eaters’, while d3 has no matching terms. However on examination, we see that d3 conveys the concept of plant eaters though it does not contain the same terms. Document d2 on the other hand talks about carnivores (non plant eaters) which is not something that is relevant.

Hence this order is not desirable.

[Warm up] Part 2: Building an IR system [Implementation]

1. Implement the retrieval component of the IR system in the template provided. Use the TF-IDF vector representation for representing documents.

Implementation: *informationRetrieval.py*

```
11  class InformationRetrieval():
12
13  >   def __init__(self):⋯
25
26
27  >   def buildIndex(self, docs, docIDs, ⋯
90
91
92  >   def termDocumentMatrix(self, docs, docIDs):⋯
```

[Warm up] Part 3: Evaluating your IR system [Implementation]

1. Implement the following evaluation measures in the template provided (i). Precision@k, (ii). Recall@k, (iii). $F_{0.5}$ score@k, (iv). AP@k, and (v) nDCG@k.

Implementation: *evaluation.py*

Precision@k: Refer to function `queryPrecision` under `class Evaluation`

```
5       def queryPrecision(self, query_doc_IDs_ordered, query_id, qrels, k):
6           relevant_docs = [int(entry['id']) for entry in qrels if int(entry['query_num']) == int(query_id)]
7           retrieved_docs = query_doc_IDs_ordered[:k]
8           num_relevant_retrieved = len(set(relevant_docs).intersection(set(retrieved_docs)))
9           precision = num_relevant_retrieved / k if k > 0 else 0
10          return precision
```

Recall@k: Refer to function `queryRecall` under `class Evaluation`

```
42      def queryRecall(self, query_doc_IDs_ordered, query_id, qrels, k):
43          relevant_docs = [int(entry['id']) for entry in qrels if int(entry['query_num']) == int(query_id)]
44          retrieved_docs = query_doc_IDs_ordered[:k]
45          num_relevant_retrieved = len(set(relevant_docs).intersection(retrieved_docs))
46          recall = num_relevant_retrieved / len(relevant_docs) if len(relevant_docs) > 0 else 0
47          return recall
```

$F_{0.5}$ score@k: Refer to function `queryFscore` under `class Evaluation`

```
79      def queryFscore(self, query_doc_IDs_ordered, query_id, qrels, k):
80          precision = self.queryPrecision(query_doc_IDs_ordered, query_id, qrels, k)
81          recall = self.queryRecall(query_doc_IDs_ordered, query_id, qrels, k)
82          fscore = (2 * precision * recall) / (precision + recall) if (precision + recall) > 0 else 0
83          return fscore
```

AP@k: Refer to function `queryAveragePrecision` under `class Evaluation`

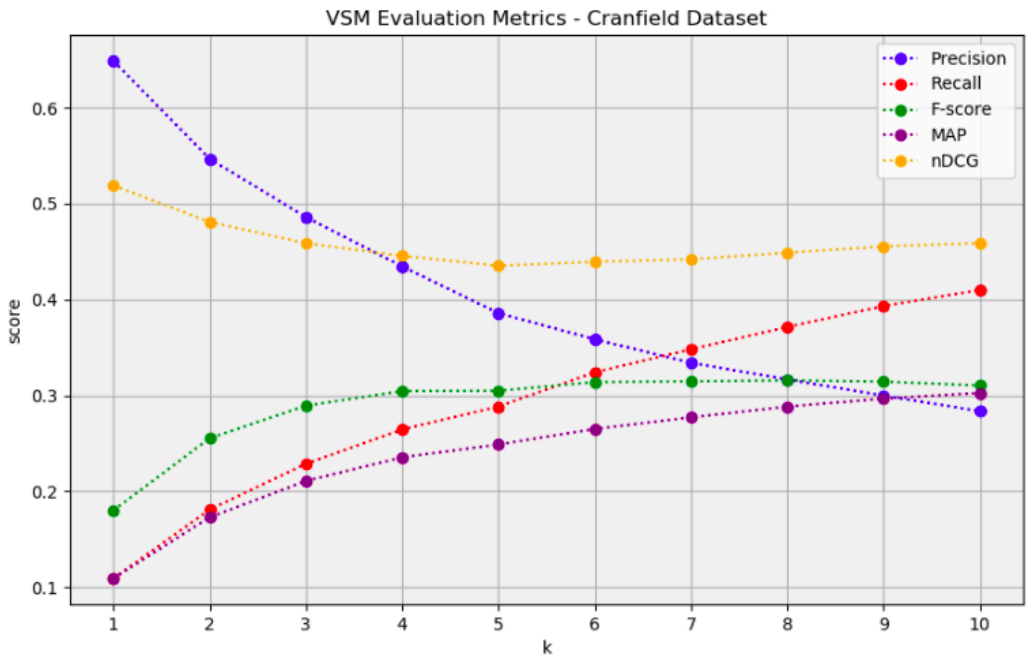
```
161     def queryAveragePrecision(self, query_doc_IDs_ordered, query_id, qrels, k):
162         relevant_docs = {int(entry['id']) for entry in qrels if int(entry['query_num']) == int(query_id)}
163         num_relevant_docs = len(relevant_docs)
164         cumulative_precision = 0
165         num_relevant_retrieved = 0
166
167         for i in range(k):
168             doc_id = query_doc_IDs_ordered[i]
169             if doc_id in relevant_docs:
170                 num_relevant_retrieved += 1
171                 cumulative_precision += num_relevant_retrieved / (i + 1)
172
173         avgPrecision = cumulative_precision / num_relevant_docs if num_relevant_docs > 0 else 0
174         return avgPrecision
```

nDCG@k: Refer to function `queryNDCG` under `class Evaluation`

```python
def queryNDCG(self, query_doc_IDs_ordered, query_id, qrels, k):
    relevant_docs = {int(entry['id']): 5 - int(entry['position']) for entry in qrels if int(entry['query_num']) == int(query_id)}

    DCG = 0

    relevant_scores= [] # Dictionary with keys as doc_id and value as relevance score
    for i in range(k):
        doc_id = int(query_doc_IDs_ordered[i])
        if doc_id in relevant_docs.keys():
            DCG += relevant_docs[doc_id]/math.log2(i+2)
            relevant_scores.append(relevant_docs[doc_id])

    ideal_relevance = sorted(relevant_scores, reverse=True)
    IDCG = sum(ideal_relevance[i] / math.log2(idx + 2) for idx, i in enumerate(range(len(ideal_relevance))))

    nDCG = DCG / IDCG if IDCG > 0 else 0
    return nDCG
```

2. Assume that for a given query, the set of relevant documents is as listed in incran_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 a function of k. The 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.

Graph:



Observation:

1. Precision decreases with increase in rank.
   Precision = (Retrieved ^ Relevant) / (Retrieved)
   With a step increase in rank, the denominator always increases by 1 while the numerator either stays the same (Document retrieved is not relevant) or increases by 1 (Relevant document retrieved at that step). Hence, it is clearly a decreasing function.

2. Recall increases with increase in rank.
   Recall = (Retrieved ^ Relevant) / (Relevant)
   The denominator here is constant (ground truth positives) and with a step increase in rank, either number of relevant documents increase or remain the same.

3. F-score is converging with increasing rank as it is the harmonic mean of precision (decreasing function) and recall (increasing function).

4. Mean Average Precision is also increasing with increasing rank.
   The formula for MAP involves a constant denominator (number of ground truth positives) while the numerator is a summation of precisions at ranks where a relevant document is retrieved at that step. Thus the numerator is increasing making MAP an increasing function.

5. nDCG decreases initially and remains nearly constant with increasing rank.
   For small k, IDCG is the DCG with all ground truths being retrieved. Hence, the nDCG decreases initially. As k becomes larger, the IDCG remains constant while DCG can only increase to a certain limit and hence increases slightly and remains constant thereforth.

3. Using the `time` module in Python, report the run time of your IR system.

Runtime for 225 queries ~ 157 seconds
Average runtime per query = 0.698 seconds

[Warm up] Part 4: Analysis [Theory]

1. What are the limitations of such a Vector space model? Provide examples from the cranfield dataset that illustrate these shortcomings in your IR system.

Limitations:

1. Relevant documents that do not have many matching terms are not retrieved.
2. VSM assumes that the words are independent and orthogonal to each other.
3. Vocabulary mismatch problems due to polysemy and synonymy of words.
4. It ranks the documents based on query terms without considering their proximity within the document.
5. High latency & computational cost:- The TF-IDF vector is long (size of the vocabulary) with most terms having small values as they are present frequently or are absent. The process for computing similarity over such long vectors is expensive.

Examples from your results:

1. <u>Query-28</u>:

Ground truths:


```
{"query_num": "28", "position": 3, "id": "224"},
{"query_num": "28", "position": 3, "id": "279"},
{"query_num": "28", "position": 1, "id": "512"},
```


Results by VSM: None of the above are retrieved in the top-10.

2. <u>Query-36</u>:

Ground truths:


```
{"query_num": "36", "position": 2, "id": "169"},
{"query_num": "36", "position": 3, "id": "168"},
{"query_num": "36", "position": 1, "id": "518"},
```


Results by VSM: Only least relevant (168) is being retrieved in the top-10.

3. Query-167:

Ground truths:


```
{"query_num": "167", "position": 2, "id": "274"},
{"query_num": "167", "position": 3, "id": "82"},
{"query_num": "167", "position": 1, "id": "509"},
```


Results by VSM: Only document 274 is being retrieved at 10th position

Part 4: Improving the IR system

Based on the factual record of actual retrieval failures you reported in the assignment, you can develop hypotheses that could address these retrieval failures. You may have to identify the implicit assumptions made by your approach that may have resulted in undesirable results. To realize the improvements, you can use any method(s), including hybrid methods that combine knowledge from linguistic, background, and introspective sources to represent documents. Some examples taught in class are Latent Semantic Analysis (LSA) and Explicit Semantic Analysis (ESA).

You can also explore ways in which a search engine could be improved in aspects such as its efficiency of retrieval, robustness to spelling errors, ability to auto-complete queries, etc.

You are also expected to test these hypotheses rigorously using appropriate hypothesis testing methods. As an outcome of your work, you should be able to make a statement of structure similar to what was presented in the class:

> An algorithm $\boldsymbol{A_1}$ is better than $\boldsymbol{A_2}$ with respect to the evaluation measure $\boldsymbol{E}$ in task $\boldsymbol{T}$ on a specific domain $\boldsymbol{D}$ under certain assumptions $\boldsymbol{A}$.

Note that, unlike the assignment, the scope of this component is open-ended and not restricted to the ideas mentioned here. For each method, the final report must include a critical analysis of results; methods can be combined to come up with improvisations. It is advised that such hybrid methods are well founded on principles and not just ad hoc combinations (an example of an ad hoc approach is a simple convex combination of three methods with parameters tuned to give desired improvements).

You could either build on the template code given earlier for the assignment or develop from scratch as demanded by your approach. Note that while you are free to use any datasets to experiment with, the Cranfield dataset will be used for evaluation. The project will be evaluated based on the rigor in methodology and depth of understanding, in addition to the quality of the report and your performance in Viva.

Your project report (for Part 4) should be well structured and should include the following components.

1. An introduction to the problem setting,
2. The limitations of the basic VSM with appropriate examples from the dataset(s),
3. Your proposed approach(es) to address these issues,
4. A description of the dataset(s) used for experimentations,
5. The results obtained with a comparative study of your approach has improved the IR system, both qualitatively and quantitatively.

The latex template for the final report will be uploaded on Moodle. You are instructed to follow the template strictly.