CS6370: Natural Language Processing

Project

Release Date: 24th March 2024 Deadline: 8th May 2025

Name:	Roll No.:
Aadit Mahajan	BS21B001
Anirudh Rao	BE21B004
Shahista Afreen	NA21B050
Shreya Rajagopalan	BE21B038
Sidharthan SC	BE21B039

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.

carnivores	d1, d2
deers	d3
eat	d3
eaters	d1, d2
grass	d3
Herbivores	d1
leaves	d3
meat	d1, d2
plant	d1, d2
typically	d1, d2

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

Terms	D1	D2	D3	dfi	d/df_i	IDF_i	idf*D1	idf*D2	idf*D3
Carnivores	1	0	0	1	3	0.4771	0.4771	0	0
Deers	0	0	1	1	3	0.4771	0	0	0.4771
Eat	0	0	1	1	3	0.4771	0	0	0.4771
Eaters	2	2	0	4	0.75	-0.1249	-0.2488	-0.2488	0
Grass	0	0	1	1	3	0.4771	0	0	0.4771

Herbivores	1	0	0	1	3	0.4771	0.4771	0	0
Leaves	0	0	1	1	3	0.4771	0	0	0.4771
Meat	1	1	0	2	1.5	0.1761	0.1761	0.1761	0
Plant	1	1	0	2	1.5	0.1761	0.1761	0.1761	0
Typically	1	1	0	2	1.5	0.1761	0.1761	0.1761	0

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

d1 and d2

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

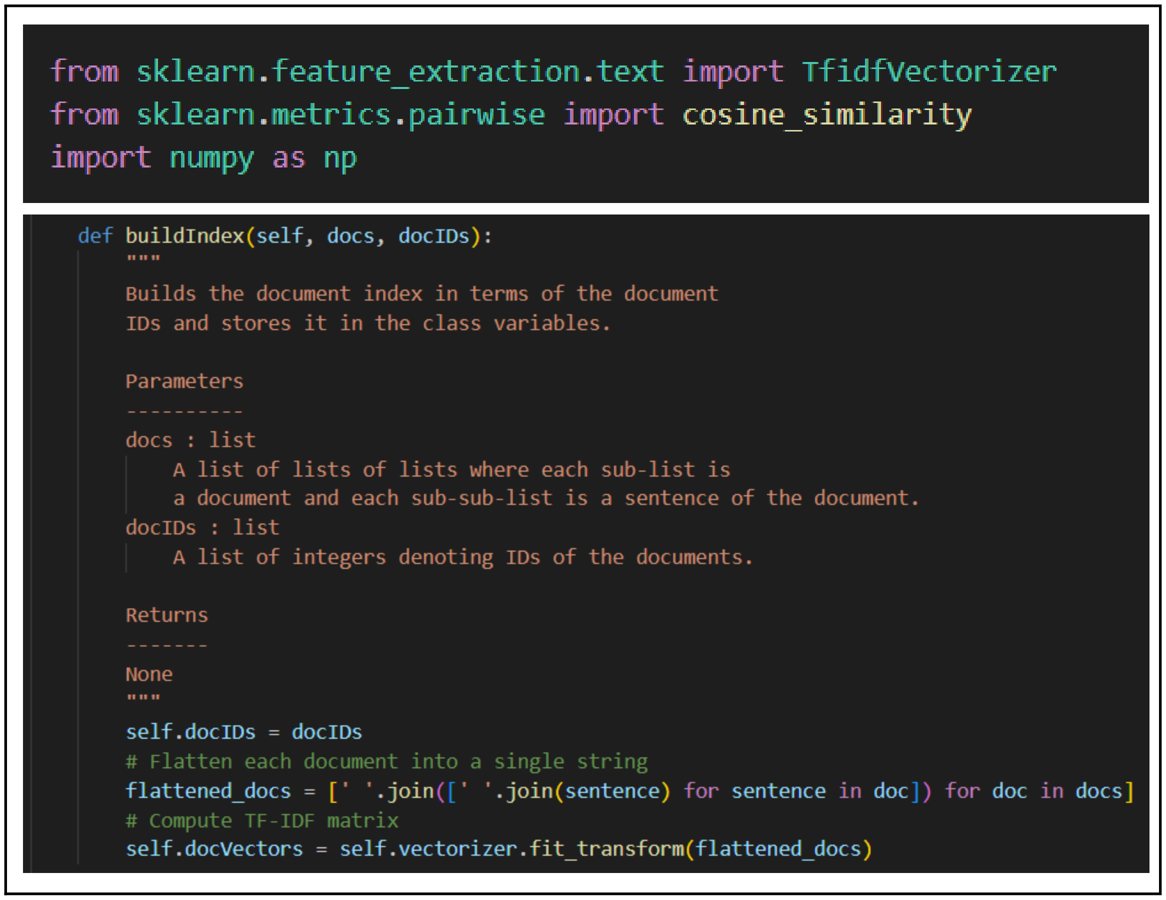
Cosine Similarity calculations:
sim(query, doc1) = 0.464916
sim(query, doc2) = 0.464916
sim(query, doc3) = 0

Ranking documents:
d3<d1=d2

Is the ordering desirable? If no, why not?:
The ordering is not ideal as document 3 is also closely related to the query but it will not be retrieved according to the cosine similarity.

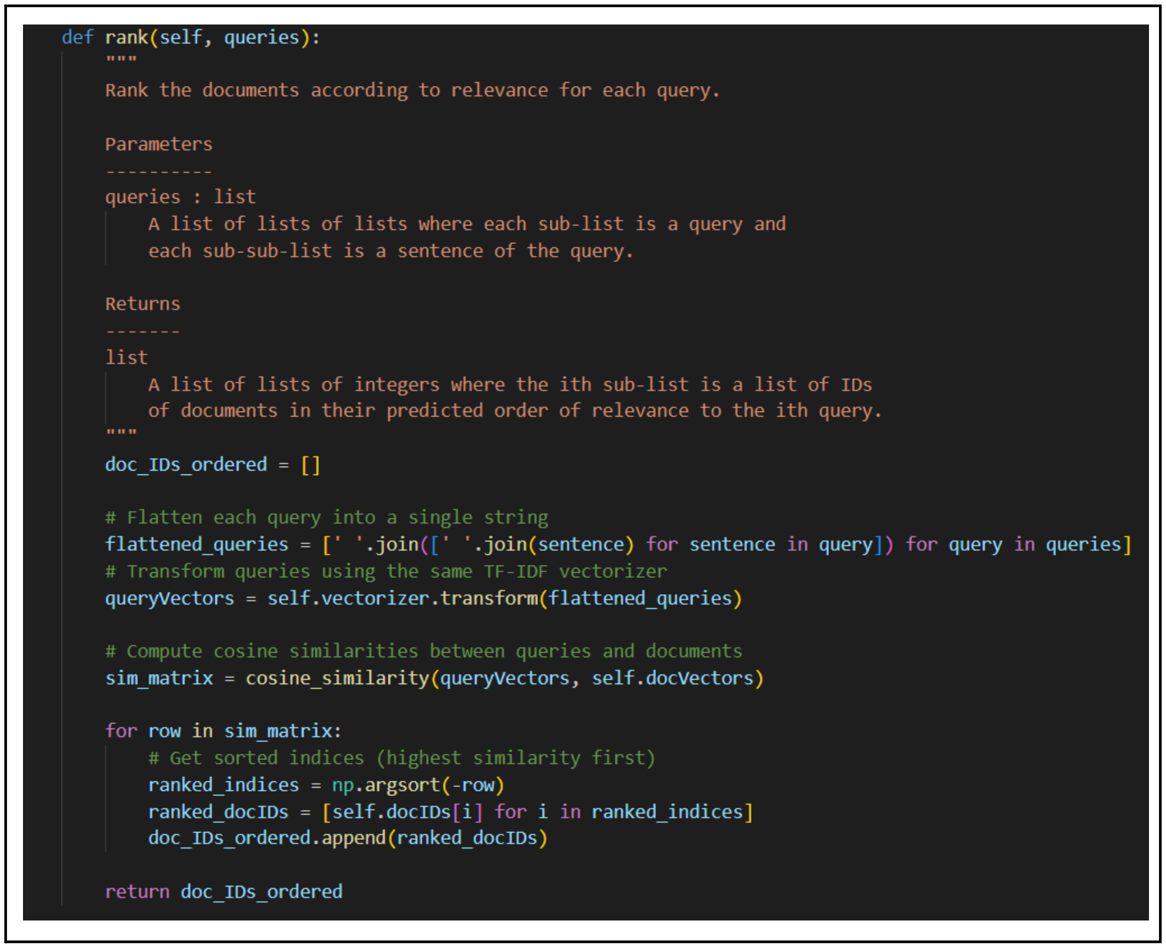
[Warm up] Part 2: Building an IR system [Implementation]

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



```python
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
import numpy as np
```

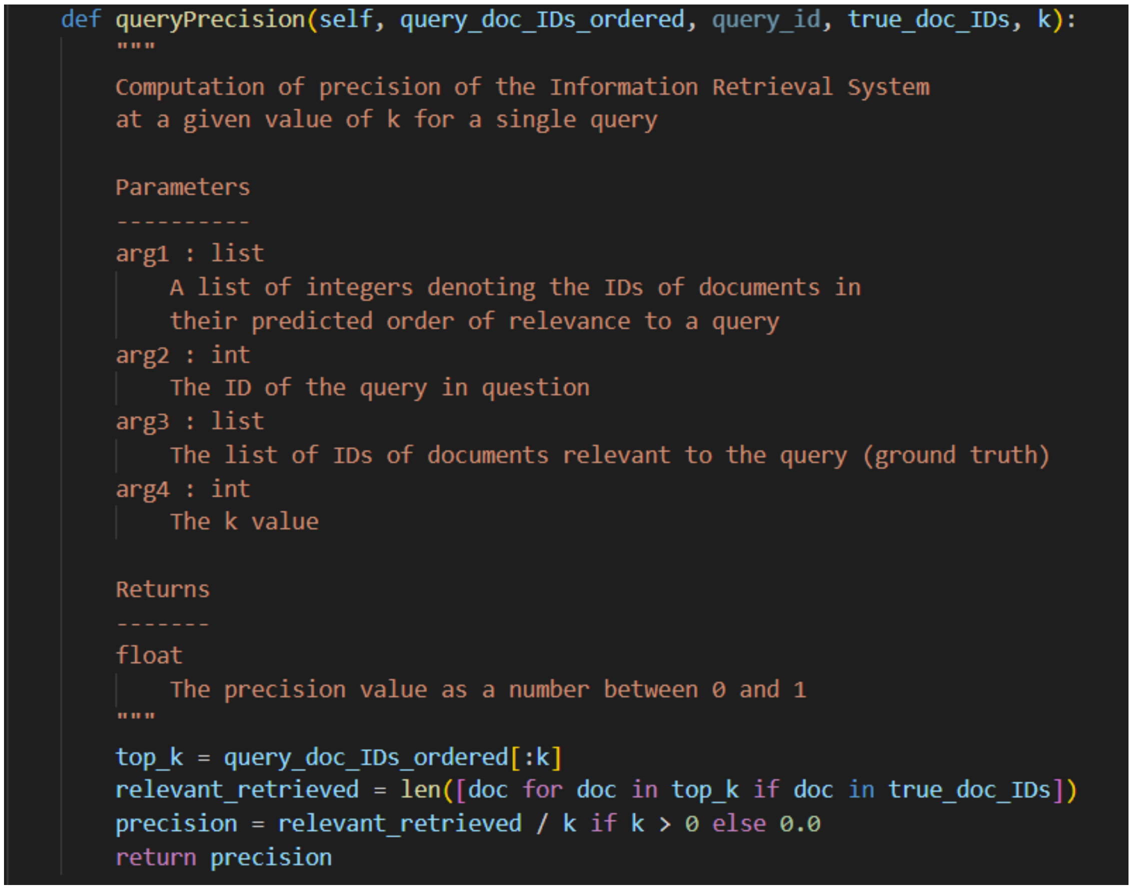
```python
    def buildIndex(self, docs, docIDs):
        """
        Builds the document index in terms of the document
        IDs and stores it in the class variables.

        Parameters
        ----------
        docs : list
            A list of lists of lists where each sub-list is
            a document and each sub-sub-list is a sentence of the document.
        docIDs : list
            A list of integers denoting IDs of the documents.

        Returns
        -------
        None
        """
        self.docIDs = docIDs
        # Flatten each document into a single string
        flattened_docs = [' '.join([' '.join(sentence) for sentence in doc]) for doc in docs]
        # Compute TF-IDF matrix
        self.docVectors = self.vectorizer.fit_transform(flattened_docs)
```


```python
def rank(self, queries):
    """
    Rank the documents according to relevance for each query.

    Parameters
    ----------
    queries : list
        A list of lists of lists where each sub-list is a query and
        each sub-sub-list is a sentence of the query.

    Returns
    -------
    list
        A list of lists of integers where the ith sub-list is a list of IDs
        of documents in their predicted order of relevance to the ith query.
    """
    doc_IDs_ordered = []

    # Flatten each query into a single string
    flattened_queries = [' '.join([' '.join(sentence) for sentence in query]) for query in queries]
    # Transform queries using the same TF-IDF vectorizer
    queryVectors = self.vectorizer.transform(flattened_queries)

    # Compute cosine similarities between queries and documents
    sim_matrix = cosine_similarity(queryVectors, self.docVectors)

    for row in sim_matrix:
        # Get sorted indices (highest similarity first)
        ranked_indices = np.argsort(-row)
        ranked_docIDs = [self.docIDs[i] for i in ranked_indices]
        doc_IDs_ordered.append(ranked_docIDs)

    return doc_IDs_ordered
```

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

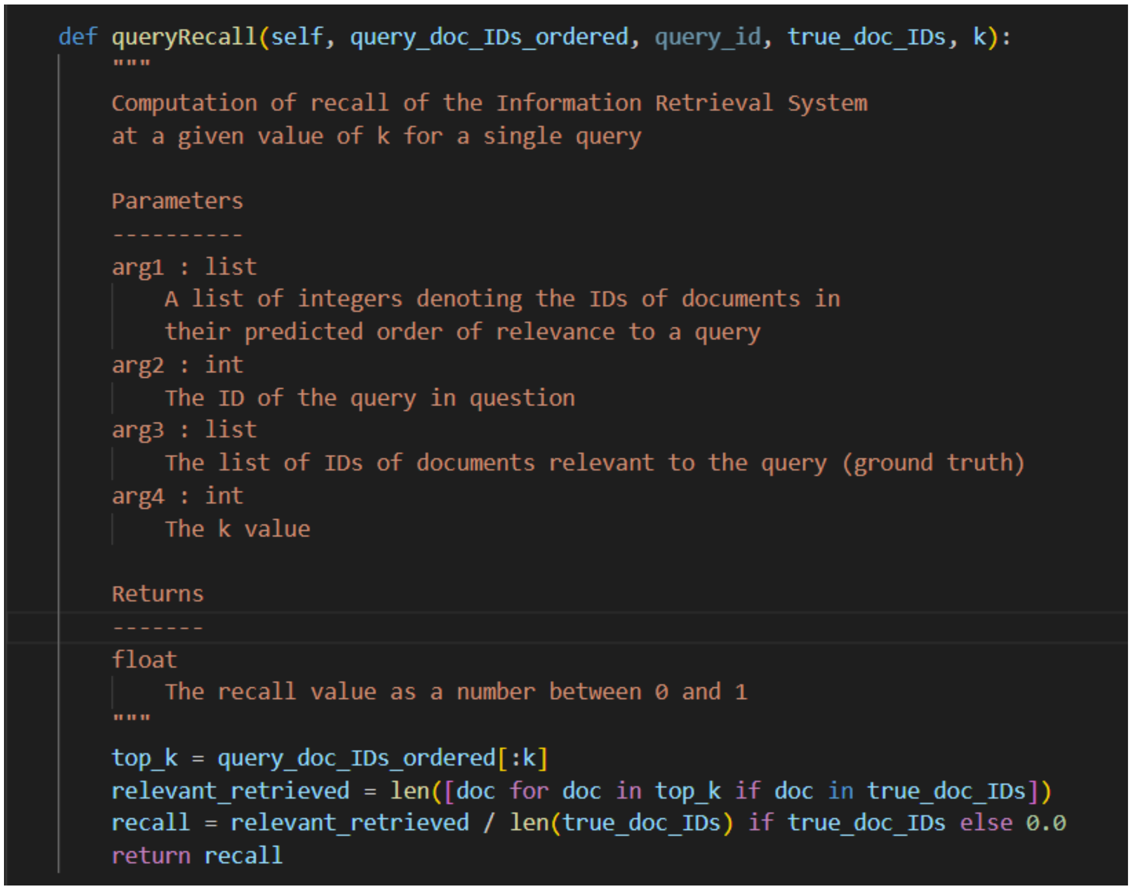
1. Implement the following evaluation measures in the template provided (i). Precision@k, (ii). Recall@k, (iii). $F_{0.5}$ score@k, (iv). AP@k, and (v) nDCG@k.

Precision@k:



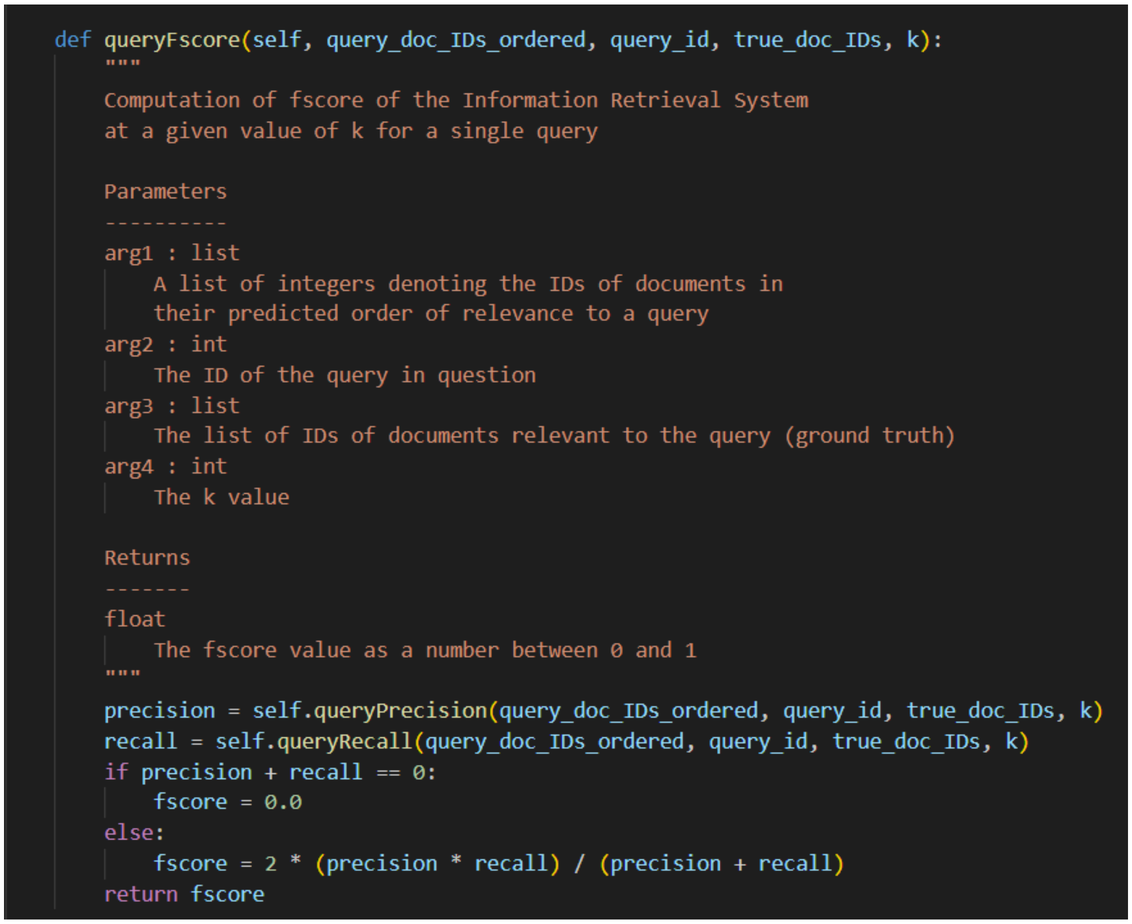
```
    def queryPrecision(self, query_doc_IDs_ordered, query_id, true_doc_IDs, k):
        """
        Computation of precision of the Information Retrieval System
        at a given value of k for a single query

        Parameters
        ----------
        arg1 : list
            A list of integers denoting the IDs of documents in
            their predicted order of relevance to a query
        arg2 : int
            The ID of the query in question
        arg3 : list
            The list of IDs of documents relevant to the query (ground truth)
        arg4 : int
            The k value

        Returns
        -------
        float
            The precision value as a number between 0 and 1
        """
        top_k = query_doc_IDs_ordered[:k]
        relevant_retrieved = len([doc for doc in top_k if doc in true_doc_IDs])
        precision = relevant_retrieved / k if k > 0 else 0.0
        return precision
```

Recall@k:



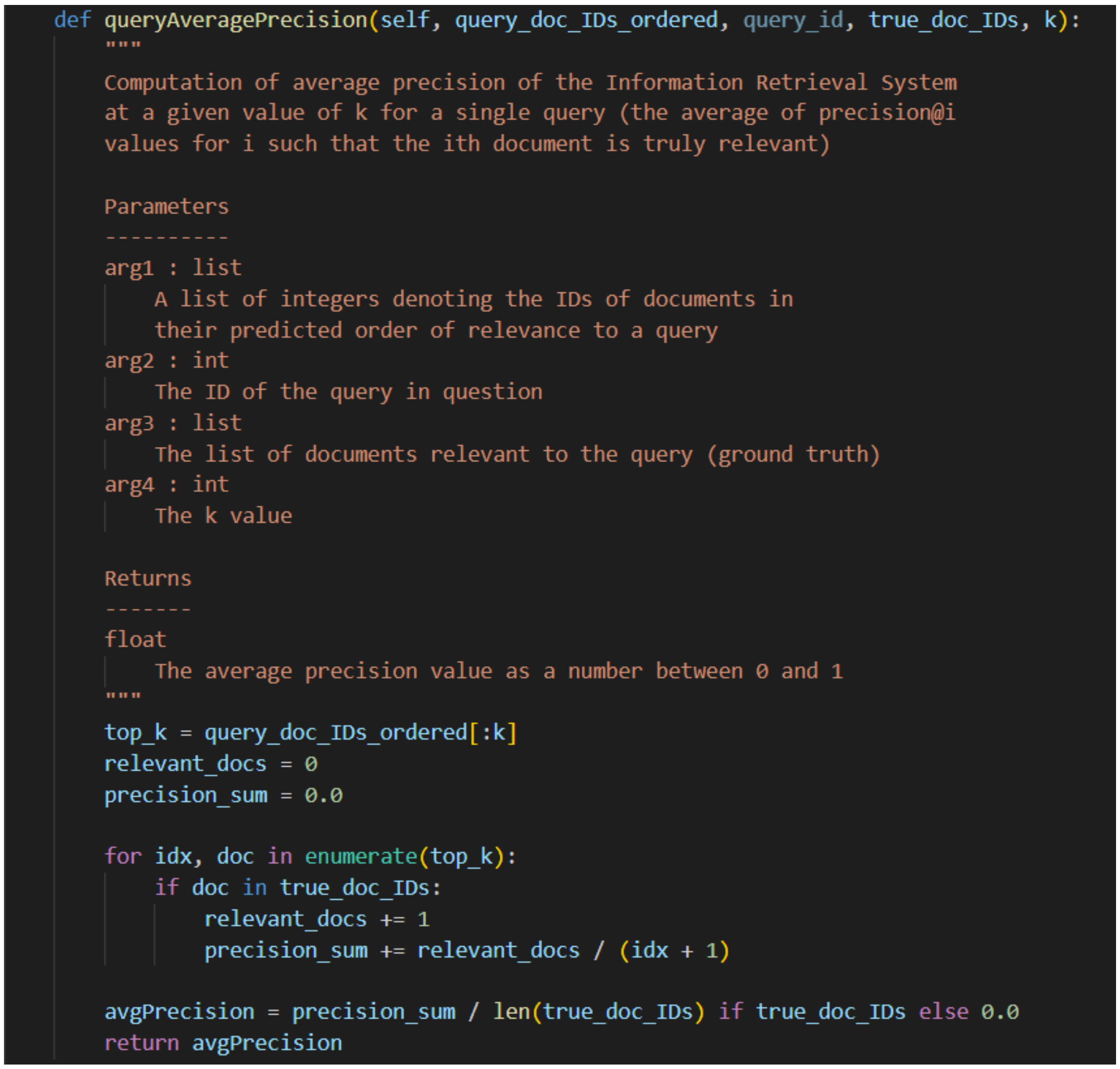
```
def queryRecall(self, query_doc_IDs_ordered, query_id, true_doc_IDs, k):
    """
    Computation of recall of the Information Retrieval System
    at a given value of k for a single query

    Parameters
    ----------
    arg1 : list
        A list of integers denoting the IDs of documents in
        their predicted order of relevance to a query
    arg2 : int
        The ID of the query in question
    arg3 : list
        The list of IDs of documents relevant to the query (ground truth)
    arg4 : int
        The k value

    Returns
    -------
    float
        The recall value as a number between 0 and 1
    """
    top_k = query_doc_IDs_ordered[:k]
    relevant_retrieved = len([doc for doc in top_k if doc in true_doc_IDs])
    recall = relevant_retrieved / len(true_doc_IDs) if true_doc_IDs else 0.0
    return recall
```

$F_{0.5}$score@k:



```
def queryFscore(self, query_doc_IDs_ordered, query_id, true_doc_IDs, k):
    """
    Computation of fscore of the Information Retrieval System
    at a given value of k for a single query

    Parameters
    ----------
    arg1 : list
        A list of integers denoting the IDs of documents in
        their predicted order of relevance to a query
    arg2 : int
        The ID of the query in question
    arg3 : list
        The list of IDs of documents relevant to the query (ground truth)
    arg4 : int
        The k value

    Returns
    -------
    float
        The fscore value as a number between 0 and 1
    """
    precision = self.queryPrecision(query_doc_IDs_ordered, query_id, true_doc_IDs, k)
    recall = self.queryRecall(query_doc_IDs_ordered, query_id, true_doc_IDs, k)
    if precision + recall == 0:
        fscore = 0.0
    else:
        fscore = 2 * (precision * recall) / (precision + recall)
    return fscore
```

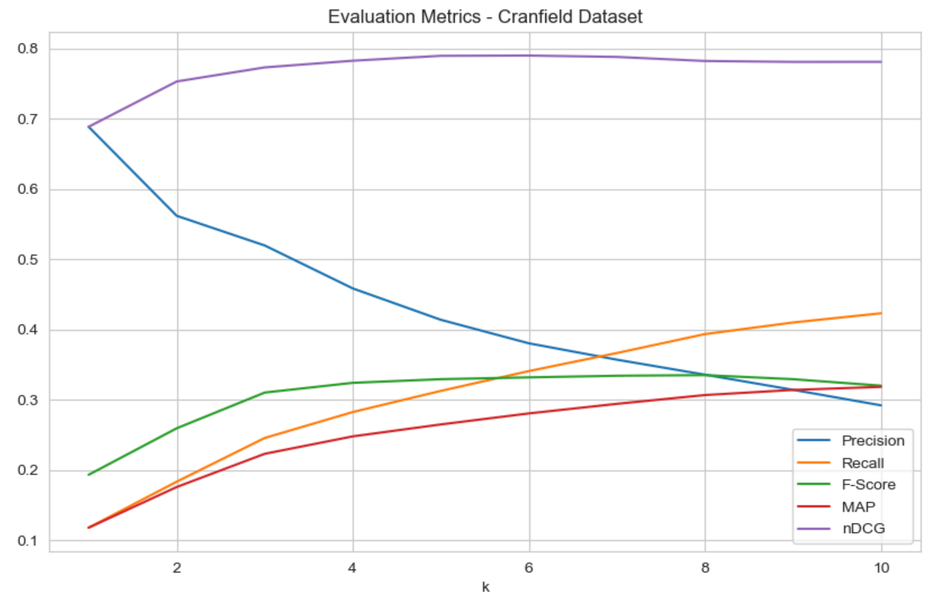
AP@k:



```python
    def queryAveragePrecision(self, query_doc_IDs_ordered, query_id, true_doc_IDs, k):
        """
        Computation of average precision of the Information Retrieval System
        at a given value of k for a single query (the average of precision@i
        values for i such that the ith document is truly relevant)

        Parameters
        ----------
        arg1 : list
            A list of integers denoting the IDs of documents in
            their predicted order of relevance to a query
        arg2 : int
            The ID of the query in question
        arg3 : list
            The list of documents relevant to the query (ground truth)
        arg4 : int
            The k value

        Returns
        -------
        float
            The average precision value as a number between 0 and 1
        """
        top_k = query_doc_IDs_ordered[:k]
        relevant_docs = 0
        precision_sum = 0.0

        for idx, doc in enumerate(top_k):
            if doc in true_doc_IDs:
                relevant_docs += 1
                precision_sum += relevant_docs / (idx + 1)

        avgPrecision = precision_sum / len(true_doc_IDs) if true_doc_IDs else 0.0
        return avgPrecision
```

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:
From the plot, we can observe that precision decreases monotonically as a function of rank, while recall increases monotonically as a function of rank. These trends abide by the definitions and suggest that the basic IR model works without obvious errors. The F-score is a weighted harmonic mean of precision and recall. It rises initially and peaks around k=5 and then plateaus, indicating that optimal balance between precision and recall at moderate rank. MAP also increases monotonically with k but the slope becomes gentler at higher rank, which implies decreasing returns as more results are considered. nDCG is consistently high (above 0.7) and increases slightly with k but then remains constant. It is close to 1, indicating the VSM model already performs well, however it can be improved further.

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

22.22 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:

- The model assumes that terms are orthogonal to each other meaning they are not related to each other, which ignores contextual relationships
- VSM does not take into account semantic relationships. It relies on exact term matching and cannot handle cases of synonymy or polysemy effectively.
- Exact term matching also means that if a query contains terms that are not present in relevant documents, those documents will not be retrieved even if they are relevant. And similarly, even irrelevant documents may be retrieved if the query contains some matching terms.
- It does not take word order into consideration.
- The model produces high-dimensional, sparse vector representation of documents, which can be computationally expensive.

Examples from your results:

The following examples from the implementation on Cranfield dataset display the limitations of VSM

Example 1:
Query 28: "what application has the linear theory design of curved wings."

- Metrics: Precision=0.0000, Recall=0.0000, F-score=0.0000, nDCG=0.0000
- Top 10 documents: [1051, 1075, 680, 923, 921, 920, 762, 247, 674, 470]
- Relevant documents: [224, 279, 512]
- Relevant docs in top 10: []
- Observation: Model takes only direct words like “curved wings” and “linear theory” into consideration. It does not learn the hidden

meaning to answer the relevant query.

Example 2:
Query 63: "where can i find pressure data on surfaces of swept cylinders ."

- Metrics: Precision=0.0000, Recall=0.0000, F-score=0.0000, nDCG=0.0000
- Top 10 documents: [738, 1045, 839, 843, 1046, 678, 891, 491, 1051, 1121]
- Relevant documents: [567, 564, 566, 539]
- Relevant docs in top 10: []
- Observation: Some are completely unrelated (about some polynomial equation). Others contain keywords present in the query like “pressure” and “cylinder”

Example 3:
Query 216: "what investigations have been made of the wave system created by a static pressure distribution over a liquid surface ."

- Metrics: Precision=0.0000, Recall=0.0000, F-score=0.0000, nDCG=0.0000
- Top 10 documents: [958, 175, 764, 1156, 407, 971, 367, 64, 1225, 1220]
- Relevant documents: [156, 506]
- Relevant docs in top 10: []
- Observation: The relevant documents are about “shallow-water wave resistance”. The query does not mention the specific term. The model is unable to learn that wave resistance is related to the query.

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.