CS6370: Natural Language Processing
Project

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

Name:	Roll No.:
Abani Singha	MA23M001
Manish Kumar Kumawat	MA23M010
Muhammed Dilshah U	MA23M014
Partha Sakha Paul	MA23M016
Sourav Majhi	MA23M022

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.

So, after removing the stop words the documents are:

d_1: Herbivores typically plant eaters meat eaters (6 terms)

d_2: Carnivores typically meat eaters plant eaters (6 terms)

d_3: Deers eat grass leaves (4 terms)

Hence the corresponding inverted index representation is:

Word	**Documents (d_1, d_2, d_3)**
Herbivores	d_1
typically	d_1, d_2
plant	d_1, d_2
eaters	d_1, d_2
meat	d_1, d_2
Carnivores	d_2
Deers	d_3
eat	d_3
grass	d_3
leaves	d_3

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

By the definition of **TF** we have the following matrix**(TF):**

Terms	**d_1 (6 terms)**	**d_2 (6 terms)**	**d_3 (4 terms)**
Herbivores	1/6	0	0
typically	1/6	1/6	0
plant	1/6	1/6	0
eaters	2/6	2/6	0
meat	1/6	1/6	0
Carnivores	0	1/6	0
Deers	0	0	1/4
eat	0	0	1/4
grass	0	0	1/4
leaves	0	0	1/4

And by the definition of **IDF** we have the following matrix (**IDF)**:

Terms	**DF (t)**	**IDF**
herbivores	1	$\log_{10}(3/1) \approx 0.4771$
typically	2	$\log_{10}(3/2) \approx 0.1761$
plant	2	$\log_{10}(3/2) \approx 0.1761$
eaters	2	$\log_{10}(3/2) \approx 0.1761$
meat	2	$\log_{10}(3/2) \approx 0.1761$
carnivores	1	$\log_{10}(3/1) \approx 0.4771$
deers	1	$\log_{10}(3/1) \approx 0.4771$
eat	1	$\log_{10}(3/1) \approx 0.4771$
grass	1	$\log_{10}(3/1) \approx 0.4771$
leaves	1	$\log_{10}(3/1) \approx 0.4771$

By multiplying with the corresponding IDF to each row of the TF matrix we get the final **TF-IDF matrix** as follows:

Terms	**d_1**	**d_2**	**d_3**
Herbivores	0.0795	0	0
typically	0.0294	0.0294	0
plant	0.0294	0.0294	0
eaters	0.0587	0.0587	0
meat	0.0294	0.0294	0
Carnivores	0	0.0795	0
Deers	0	0	0.1193

eat	0	0	0.1193
grass	0	0	0.1193
leaves	0	0	0.1193

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

Query: "plant eaters"
We will search for documents that contain both terms: plant & eaters

From the inverted index:
plant → {d1, d2}
eaters → {d1, d2}

Intersection:
Documents that contain both "plant" and "eaters":
plant ∩ eaters = {d1, d2}

Final Answer:
Documents retrieved = **d1 and d2**

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

We'll use the TF-IDF matrix we previously calculated.
Query terms: plant eaters
We need to build a TF-IDF vector for the query, and compare it with d1 and d2, since only those contain both words.

TF-IDF vector for Query:
The query has 2 terms: plant and eaters.

TF in query:
- TF(plant) = 1/2 = 0.5
- TF(eaters) = 1/2 = 0.5

We will multiply each by their IDF values (from before):

- IDF(plant) ≈ 0.1761
- IDF(eaters) ≈ 0.1761

So, the query vector is:
q = [0.5×0.1761,0.5×0.1761] = **[0.08805, 0.08805]**

Cosine Similarity calculations:
Query vs d1:

- q = [0.08805, 0.08805]
- d1 = [0.0294, 0.0587]

Dot product:
q · d1 = (0.08805×0.0294) + (0.08805×0.0587) ≈ 0.002589+0.005166 ≈ **0.007755**
Norms:

- **|q| =** √ (0.08805² + 0.08805²) ≈ √ (0.00776 + 0.00776) ≈ √0.0155 ≈ **0.1245**
- **|d1| =** √ (0.0294² + 0.0587²) ≈ √ (0.000864 + 0.00345) ≈ √0.00431 ≈ **0.0656**

CosSim(q, d1) = 0.007755 / (0.1245×0.0656) ≈ 0.007755 / (0.00817) ≈ **0.949**

Query vs d2
d2 vector is the same as d1 for these terms, so:
CosSim(q, d2) ≈ 0.949

Ranking documents:

Doc	Cos Sim.
d1	0.949
d2	0.949

Both documents are equally ranked based on cosine similarity.

Is the ordering desirable? If no, why not?
Even though **cosine similarity ranks both d1 and d2 equally**, but semantically:
d1 is a better match for the query "plant eaters" because it **supports** the concept.
But **d2 contradicts** it --- it says carnivores are *not* plant eaters.

Hence in conclusion:

- **Mathematical similarity** = both are equal (due to same term frequency and weights)
- **Semantic similarity** = d1 should be ranked higher

Therefore, **the result is not desirable** from a **meaning perspective**

[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.

Answer (Explanation + Reference to Code):
The retrieval component of the IR system using TF-IDF vector representation has been implemented in the InformationRetrieval class. Here is the working module:

TF-IDF Index Building – buildIndex()
self.vectorizer = TfidfVectorizer()
self.doc_vectors = self.vectorizer.fit_transform(flattened_docs)
Purpose: Converts each document into a TF-IDF vector.
flattened_docs: Each document is preprocessed into a flat string.
The resulting doc_vectors is a sparse matrix of shape (num_docs, vocab_size).

Query Ranking – rank()
query_vectors = self.vectorizer.transform(flattened_queries)
similarity_matrix = cosine_similarity(query_vectors, self.doc_vectors)
Queries are transformed into TF-IDF vectors using the trained vectorizer.
Cosine similarity is computed between each query vector and all document vectors.
Documents are then ranked in descending order of similarity for each query.

The TF-IDF model is chosen when method == "TF-IDF" (default).
The index is built using TfidfVectorizer.fit_transform.
Retrieval (ranking) is done using cosine_similarity.

Example Run:
Given this input:

```
docs = [
   [['this', 'is', 'the', 'first', 'document']],
   [['this', 'document', 'is', 'the', 'second', 'document']],
```

```
    [['and', 'this', 'is', 'the', 'third', 'one']],
    [['is', 'this', 'the', 'first', 'document']]]
queries = [
    [['this', 'is', 'the', 'first']],
    [['third', 'document']]]
```

The system builds the index and ranks documents according to the TF-IDF similarity with the queries.

[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.

To evaluate the performance of the Information Retrieval (IR) system, we implemented the following evaluation metrics at cutoff k:

Precision@k: This measures the proportion of top-k retrieved documents that are relevant.
Implemented using *queryPrecision()* for a single query and *meanPrecision()* to average over all queries.

Recall@k: This measures the proportion of all relevant documents that are retrieved in the top k.
Implemented using *queryRecall()* and *meanRecall()* functions.

F0.5 Score@k: This is the harmonic mean of Precision and Recall, with more emphasis on Precision (β=0.5).
Implemented using *queryFscore()* and *meanFscore().*

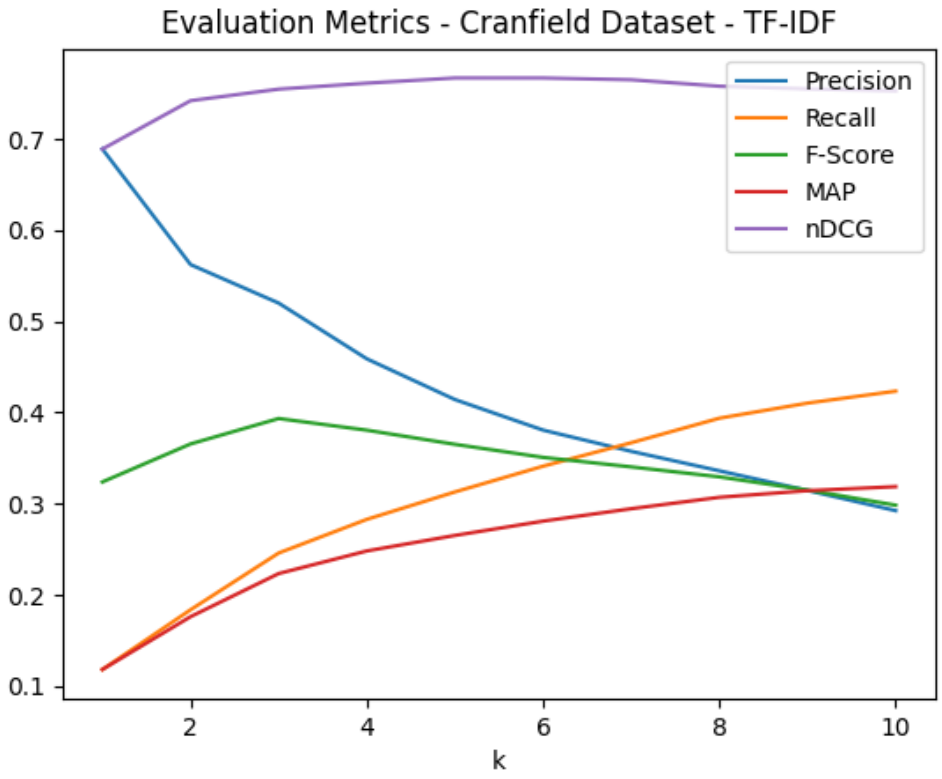
AP@k: This metric averages the precision values computed at the rank positions of relevant documents within the top k.
Implemented using *queryAveragePrecision().*
The *meanAveragePrecision()* function aggregates the *queryAveragePrecision()* scores across all queries to compute the overall Mean Average Precision at k **(MAP@k).**

nDCG@k: This measures the gain of a document based on its position in the result list, discounted logarithmically, and normalized using the ideal gain.
Implemented using *queryNDCG()* and *meanNDCG(),* considering graded relevance from the position field in qrels.

Grouped qrels were handled using:
group_qrels() – for binary relevance (for Precision, Recall, F-score, and AP).
group_qrels1() – for graded relevance used in 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:



Observations from the Evaluation Metrics Plot (Cranfield Dataset - TF-IDF):

The above plot illustrates how five evaluation metrics -- Precision, Recall, F-score, Mean Average Precision (MAP), and nDCG -- vary with different values of k (1 to 10) in a TF-IDF-based information retrieval system applied to the Cranfield dataset.

Precision

- **Trend:** Precision decreases as k increases.
- **Reason:** At lower k, only the most relevant documents are retrieved, leading to higher precision. As more documents are retrieved (larger k), less relevant documents begin to appear, reducing the precision.

Recall

- **Trend:** Recall increases with increasing k.
- **Reason:** Retrieving more documents increases the chance of including relevant ones, so the proportion of relevant documents found (recall) improves with larger k.

F-score

- **Trend:** Initially increases slightly, peaks around k=3, then slowly declines.
- **Reason:** F-score balances precision and recall. The peak indicates an optimal trade-off point, after which recall gain doesn't compensate for the precision drop.

MAP (Mean Average Precision)

- **Trend:** Gradually increases with k.
- **Reason:** As k increases, more relevant documents are found at various ranks, improving the average precision per query. However, the growth is slower as less relevant results are added at lower precision ranks.

nDCG (Normalized Discounted Cumulative Gain)

- **Trend:** High from the start and stabilizes quickly.
- **Reason:** nDCG accounts for the position of relevant documents, giving higher scores to relevant documents that appear early. Since top-ranked documents are often relevant, nDCG remains consistently high even as k increases.

Conclusion:

- Precision and recall show the expected trade-off behavior as k increases.
- F-score identifies an optimal k (around 3) balancing both metrics.
- MAP improves gradually, reflecting better ranking of relevant documents.
- nDCG remains high, suggesting that top-k results are generally well-ranked and informative.

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

Using the TF-IDF setup we came up with ---
Execution Time: 6.027541637420654 s

[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 Vector Space Model (VSM), while simple and effective in many cases, has several limitations that impact its performance in real-world retrieval scenarios. Based on the Cranfield dataset, here are some key shortcomings:

- **Lack of Semantic Understanding**

VSM relies on exact term matching and does not capture the meaning or context of words.
Example:
For a query numbered 107 in *cran_queries.json* file: **"aircraft structure in a noise environment"**,
Top five document IDs retrieved:
640
725
909
883
51

But one query which has the details below:
{"query_num": "107", "position": 2, "id": "728"}
"body" of doc_id 728: "free vibrations of continuous skin stringer panels. the determination of the natural frequencies and normal modes of vibration for continuous panels, representing more or less typical fuselage skin-panel construction for modern airplanes, is discussed in this paper are considered. a numerical example is presented, and analytical results for a particular structural configuration agree favourably with available experimental measurements.",
---- which has the semantic relation with the input query but not retrieved.

- **Polysemy**

Polysemy: Same word with multiple meanings is not disambiguated.
Example:
For a query ---
Query: “a fan is taking a picture with his role model in a nice place”
Top five document IDs :
1093
1329
1162
860
1164
These documents contain words like “fan”, “place”, “model” but all documents have different meanings(e.g. electronic device fan of the aircraft) than the query meaning (where fan suggests a person who likes a role model)

- **Term Independence Assumption**

Limitation: VSM treats each term independently, ignoring term order or proximity, which are often crucial for understanding document relevance.
The presence or absence of one term in a document is independent of the presence or absence of any other term.

Example:
Query: "boundary layer transition"
In VSM model using term independence, the model treats:

- "boundary"
- "layer"
- "transition"

as independent — so it does **not** consider that "boundary layer" is a common scientific phrase in aerodynamics.

Issue:

"boundary" and "layer" co-occur frequently and have **semantic dependency**, but the model ignores that relationship.

- **No Handling of Word Importance Across Contexts**

Limitation: TF-IDF weights are fixed across the corpus and do not adapt to different query contexts.

Example:

The term "flow" may be critical in a fluid dynamics context but irrelevant in a different aerospace context. VSM lacks this dynamic context awareness.

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.