HW01 – ISIS 4221

Natural Language Processing 2021-I

Due date: 13-03-2021

Groups are allowed up to a maximum of 3 students or 4 only if they are the same project group. Individual work is also allowed.

Coding rules: Use jupyter notebooks and be sure that the notebook is executed and contain the results before submitting. All classes, methods, functions and free-code MUST contains docstrings with a detail explanation. Build a notebook for each point.

Report: Together with the notebooks, you must submit a written report (please use pdf format) with the answers to the questions and a short summary of the implementation.

Submission: Assignments are submitted via Brightspace. Do not email us your assignments. Please upload all files and documents.

[10p] Implement the following IR evaluation metrics using python+numpy (you must use numpy):

- [1p] Precision (relevance is binary)
  >>> relevance_query_1 = [0, 0, 0, 1]
  >>> precision(relevance_query_1)
  0.25
- [1p] Precision at K (relevance is binary)
  >>> relevance_query_1 = [0, 0, 0, 1]
  >>> k = 1
  >>> precision_at_k(relevance_query_1, k)
  0
- [1p] Recall at K (relevance is binary)
  >>> relevance_query_1 = [0, 0, 0, 1]
  >>> k = 1
  >>> number_relevant_docs = 4
  >>> recall_at_k(relevance_query_1, number_relevant_docs, k)
  0
- [1p] Average precision (relevance is binary)
  - Suppose that the input binary vector contains all relevant documents.

  >>> relevance_query_2 = [0,1,0,1,1,1,1]
  >>> average_precision(relevance_query_2)
  0.5961904
- [2p] Mean average precision -MAP- (relevance is binary)
  - Input: a list of binary vectors, each one represents a query result vector.
- [2p] DCG at K (relevance is a natural number)
  >>>relevance_query_3 = [4, 4, 3, 0, 0, 1, 3, 3, 3, 0]
  >>>k = 6

```
>>>dcg_at_k(relevance_query_3, k)
10.27964
```

- [2p] NDCG at K (relevance is a positive natural number)

```
>>>relevance_query_3  = [4, 4, 3, 0, 0, 1, 3, 3, 3, 0]
>>>k = 6
>>>ndcg_at_k(relevance_query_3, k)
0.7424
```

Search Engine Strategies Comparison

Next you are going to implement a search engine with four different strategies.

1. Binary Search (BS).

2. Binary Search using Inverted Index (BSII)

3. Basic Ranked Retrieval (RRI)

4. Ranked Retrieval and Document Vectorization (RRDV)

You should make your own implementation using numpy and pandas for handling arrays and arrays.

Note: There are extra points [15p] if your "inverted index" implementation is distributed (using for example MapReduce) or efficient disk sorting is done using BSBI. Both strategies are explained in chapter 4 of the book https://nlp.stanford.edu/IR-book/pdf/04const.pdf.

Dataset: There are three files that make up the dataset. "Docs raws texts" contains 331 documents in NAF format (XML - you must use the title and content to model each document). "Queries raw texts" contains 35 queries. "relevance-judgments.tsv" contains for each query the relevant documents for each one of the queries. These relevant documents were constructed manually by human judges and serve as an evaluation dataset.

Pre-processing steps: For the following points you must preprocessing documents and queries using word level tokenization, stop word removal, normalization, and stemming.

[15p] Binary Search Strategy (BS)

[5p] Build your own implementation of the Binary term-document incidence matrix using the 331 documents in dataset.

- What is the size of the vocabulary? What is the size of the matrix?
- The matrix must be store and read from disk. Look for a good strategy for store and read a large numpy matrix.

[5p] Build a function that read the binary matrix and compute conjunction binary queries using bitwise operation.

[5p] For each one of the 35 queries in the dataset retrieve the documents using conjunction binary queries. Write a file (BS-queries_results) with the results following the same format as “relevance-judgments” file:

q01 dXX,dYY,dZZ…

[15p] Binary Search using Inverted Index (BSII)

[5p] Build your own implementation of the inverted index using the 331 documents in dataset.

- The inverted-index must be store and read from disk. Look for a good strategy and explain it in detail. Is there a better way to implement inverted index in python? What would it be?

[5p] Build a function that read the inverted-index and compute Boolean queries using the merge algorithm. You must modify the merge algorithm to compute: AND, OR, and NOT. In class we discuss strategies to optimize the execution of the merge operation. How do you implement them?

[5p] a. For each one of the 35 queries in the dataset retrieve the documents using conjunction binary queries. Write a file (BSII-AND-queries_results) with the results following the same format as “relevance-judgments”:

q01 dXX,dYY,dZZ…

b. For each one of the 35 queries in the dataset retrieve the relevant documents using disjunctive binary queries. Write a file (BSII-OR-queries_results) with the results following the same format as “relevance-judgments” file.

[15p] Basic Ranked Retrieval (RRI)

[5p] Modified inverted index to store the tf. Describe how you made the modification.

[5p] Build a function that read the modified inverted-index and compute the document score for a given query using:

$$score(q,d) = \sum_{t \epsilon q \cap d} tf.idf_{t,d}$$

[5p] For each one of the 35 queries in the dataset, retrieve the ranked documents -ordering by the score- (include only the documents with a score higher than 0 for a given query). Write a file (RRI-queries_results) with the results following the same format as “relevance-judgments” file:

q01 dXX: score(q01,dXX),dYY: score(q01,dYY),dZZ: score(q01,dZZ)…

Results evaluation. Calculate P@M, R@M, NDCG@M per query. M is the number of relevant documents found in relevance-judgments file per query. Then compute MAP as a general metric.

NOTE I: For P@M and R@M suppose a binary relevance scale. Documents not found in relevance-judgments file are NOT relevant for a given query.

NOTE II: For NDCG@M use the non-binary relevance scale found in the relevance-judgments file.

[15p] Ranked Retrieval and Document Vectorization (RRDV)

[5p] Build a function that from the inverted-index creates the tf.idf weighted vector representation of a document or query. Describe in detail your strategy, is it efficient? why, why not?

[5p] Build a function that receive two documents vectors and compute the cosine similarity.

[5p] For each one of the 35 queries in the dataset, retrieve the ranked documents -ordering by the cosine similarity score- (include only the documents with a score higher than 0 for a given query). Write a file (RRDV-queries_results) with the results following the same format as "relevance-judgments" file:

q01 dXX: cos_simi(q01,dXX),dYY: cos_simi(q01, dYY),dZZ: cos_simi(q01,dZZ)…

Results evaluation. Calculate P@M, R@M, NDCG@M per query. M is the number of relevant documents found in relevance-judgments file per query. Then compute MAP as a general metric.

NOTE I: For P@M and R@M suppose a binary relevance scale. Documents not found in relevance-judgments file are NOT relevant for a given query.

NOTE II: For NDCG@M use the non-binary relevance scale found in the relevance-judgments file.

[15p] GENSIM Corpus and Tf.Idf Model

[10p] a. Implement the BOW model using gensim and the 331 documents in dataset. Save the dictionary and the corpus in Matrix Market format.

b. Load the corpus and transform it to a TfidfModel (i.e. documents as tf.idf vectors). Serialize the resulting model to disk.

c. Prepare the model to perform some similarity queries. Create a MatrixSimilarity index and save it to disk.

[5p] a. For each one of the 35 queries in the dataset, retrieve the ranked documents -ordering by the cosine similarity score- (include only the documents with a score higher than 0 for a given query). Write a file (GENSIM-queries_results) with the results following the same format as "relevance-judgments" file:

q01 dXX: cos_simi(q01,dXX),dYY: cos_simi(q01, dYY),dZZ: cos_simi(q01,dZZ)…

b. Results evaluation. Calculate P@M, R@M, NDCG@M per query. M is the number of relevant documents found in relevance-judgments file per query. Then compute MAP as a general metric.

NOTE I: For P@M and R@M suppose a binary relevance scale. Documents not found in relevance-judgments file are NOT relevant for a given query.

NOTE II: For NDCG@M use the non-binary relevance scale found in the relevance-judgments file.

[15p] Results Analysis

[8p] Compare the results obtained by BS and BSII (AND/OR queries).

- Compare execution times for the 35 queries, what strategy performs better?
- What is the more expensive query? justify the answer. (I am looking for an answer like: q03 BSII disjunctive query is the most expensive because….). Think carefully about the answer and discuss with your partners.
- Based on your experiments and the knowledge about the computational cost of the merge operation, estimate the execution time of a query if the collection of documents is increased x100, x1000, and x10000. Justify your estimations.
- Suppose 100 new documents must be incorporated in the collection, what changes/steps would have to be made to include them in your implementation.

[7p] Compare RRI, RRDV and Gensim results.

- In terms of P@M, R@M, NDCG@M, and MAP which strategy obtained the best results? The results obtained by RRDV and Gensim are different or the same? justify your answer.
- Discuss with your partners the advantages and disadvantages of RRI and RRDV:
    - Considering all the implementation steps, which strategy has a lower computational cost?, Could you experimentally prove it?
- Suppose 100 new documents must be incorporated in the collection, what changes/steps would have to be made to include them in your implementation of RRI and RRDV.