CS60092 - Information Retrieval
Midterm Evaluation Report
Group - 4

Group Members: **Ashwani Kumar Kamal (20CS10011), Hardik Pravin Soni (20CS30023), Sourabh Soumyakanta Das (20CS30051), Shiladitya De (20CS30061)**

Problem Statement: *Query-by-Example for Scientific Article Retrieval*

§1 Description Of the Problem Statement

Domain: Scientific Publications/En
Background: Its very common for users to note to come up with perfect queries (following proper syntax or method). Hence, the idea of query by example (QBE) came into being.
In this project, documents are retrieved based on the rhetorical structure of a paper like background, method, results etc.
Task: In this project the task is to come up with methods to improve the different metrics associated with extracting relevant documents per query.

§2 Progress Made

§2.1 Choosing Dataset

There are two datasets we could find related to our project. These are as follows:

1. **CSFCube Dataset:** The CSFCube dataset[1] contains an annotated test set for validation along with the dataset for training. The training dataset is a jsonl file containing multiple parts of a papers *viz.* paper_id, metadata, title, authors, doi, venue, pred_labels (background, method, results etc.)

2. **S2ORC Dataset:** The Semantic Scholar Open Research Corpus (S2ORC) dataset[2] also uses the pre-trained models of sequential_sentence_classification and SPECTER. This dataset contains 800,000 papers related to Computer Science. This will be considered for future study to cover a wider range of queries.

§2.2 Base Model

The basemodel has been made following the paper of Mysore et. al. [1]. The base model consists of various information retrieval models and language model based implementations. The various models implemented are as follows:

1. **tf-idf**: The standard tf-idf model (ltc.ltc) is applied here.

2. **bm25Okapi**: The Standard BM25 Okapi Model from BM25Okapi Library

3. **bert_nli**: The sentence bert_nli (from `'nli-roberta-base-v2'` of SentenceTransformer) is used.

4. **bert_pp**: The sentence bert_pp (from `'paraphrase-TinyBERT-L6-v2'` of SentenceTransformer) is used.

More models like `SPECTER, UnSimCSE, SuSimCSE` will be added to the set of base models.

§2.3 Paper Readings:

Various Papers have been read by the different members of the group. All these papers and topic division of the papers among the members has been listed out in the **Appendix - Work Division** section.

§3 Experiments Carried Out

We have made the base model as per the description given in the paper and the code snippets given in the github repo of CSFCube. The details of the experiments are given as follows:

§3.1 Training Phase

1. **tf-idf:** For tf-idf first the model is trained on input data and the tf-idf vectors are stored as json in the a file `all.json` in the `Results/tf-idf` directory. Then similar procedure is done for the queries and stored as `(background, method, results).json` files.

2. **bert_nli:** Similar to the previous approach the vectors are stored in the directory `Results/bert_nli` and the corresponding queries in their respective named directories.

For the rest models like `bert_pp` similar things are done. For `bm25Okapi` the model is handled in similar way *viz.* tokenizing the corpus beforehand and at the time of testing it with queries it is first tokenized then the scores are taken.

§3.2 Testing Phase

For testing the queries, we chose the query json files present in the respective directories. They are loaded and the results are calculated based on **cosine similarity** with the documents.
After this the various metrics (as mentioned in the paper [1]) are calculated. The findings are given in the next section.

§4 Analysis Of Results

In this section we talk about the analysis of the results computed in the previous step. The different metrics mentioned in the paper [1] are as follows:

1. Ranked Precision
2. Precision@20
3. Recall@20
4. $\text{NDCG}_{\%20}$

Some other metrics mentioned but not included in the paper are $\text{NDCG}_{\%100}$, NDCG@20, Mean Reciprocal Rank (MRR) etc.
The code snippets of the above mentioned metrics are given in the **Appendix Code snippets** section.
The results are of the analysis of the base model are as follows:

1. Background

Model	Ranked Precision	P@20	R@20	$\text{NDCG}_{\%20}$
TF-IDF	0.1777	0.2266	0.3789	0.4795
SENTBERT-NLI	0.2004	0.2750	0.4328	0.5781
SENTBERT-PP	0.2332	0.3109	0.5024	0.5974

2. Method

Model	Ranked Precision	P@20	R@20	NDCG$_{\%20}$
TF-IDF	0.0892	0.0748	0.2434	0.2440
SENTBERT-NLI	0.1656	0.1028	0.3265	0.3393
SENTBERT-PP	0.1826	0.0998	0.3388	0.3865

3. Result

Model	Ranked Precision	P@20	R@20	NDCG$_{\%20}$
TF-IDF	0.1083	0.1333	0.3067	0.3851
SENTBERT-NLI	0.1278	0.1826	0.4023	0.4072
SENTBERT-PP	0.1548	0.2273	0.5484	0.5183

4. Aggregated

Model	Ranked Precision	P@20	R@20	NDCG$_{\%20}$
TF-IDF	0.1247	0.1437	0.3084	0.3676
SENTBERT-NLI	0.1643	0.1859	0.3866	0.4404
SENTBERT-PP	0.1898	0.2119	0.4631	0.4995

§5 Inferences

After doing the above mentioned experiments we got to infer two main conclusions. These are as follows:

1. The sentence encoding models like `bert_nli`, `bert_pp` takes longer to get trained but the effective file size of the vectors is small as these models focus on the sentence encodings instead of words. For eg.: 32 MB file size of `all.json` of `bert_nli` against 430 MB of that of `tf-idf`.

2. Secondly, the tf-idf vectors are extremely sparese. For eg. if a vector is chosen at random then it has only around 50 non-zero entries and rest all zeroes which is responsible for the increased size of the encodings file.

§6 References

1. Sheshera Mysore and Tim O'Gorman and Andrew McCallum and Hamed Zamani. 2021. CSFCube - A Test Collection of Computer Science Research Articles for Faceted Query by Example. In Proceedings of the Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2). `https://openreview.net/forum?id=8Y50dBbmGU`

2. Kyle Lo, Lucy Lu Wang, Mark Neumann, Rodney Kinney, and Daniel Weld. 2020. S2ORC: The Semantic Scholar Open Research Corpus. In Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics. Association for Computational Linguistics, Online, 4969–4983. `https://doi.org/10.18653/v1/2020.acl-main.447`

3. Chong Wang and David M. Blei. Collaborative Topic Modeling for Recommending Scientific Articles. `https://dl.acm.org/doi/pdf/10.1145/2020408.2020480`

4. M.A. Angrosh, Stephen Cranefield and Nigel Stanger. Contextual Information Retrieval in Research Articles: Semantic Publishing Tools for the Research Community. `https://www.semantic-web-journal.net/system/files/swj169_1.pdf`

5. Yuna Jeong and Eunhui Kim. SciDeBERTa: Learning DeBERTa for Science Technology Documents and Fine-Tuning Information Extraction Tasks. `https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=9791256tag=1`

§7 Ideas to Work Upon

The following list are the different ideas that we have decided to work upon after to improve both accuracy of the models as well as user experience.

1. Applying Neural Networks and related losses to improve the classification models. The losses are to be explored from the hugging-face library.
2. Applying the page ranking algorithm to provide a ranked set of other papers as suggestive readings to improve the user experience. This is similar to the Google's Page Ranking Algorithm. But the only problem is that this cannot applied in the CSFCube dataset as it does not contain the references, rather it can be applied in the S2ORC dataset.
3. Scraping the H-Index and citations of the authors to provide an additional feature in the search engine to get the results sorted by H-Index or citations and also by year of publishing.
4. Applying Semantic Similarity.
5. Collaborative filtering and applying a probabilistic model to this can also help cater similar queries with similar articles (future study as it requires user data to know which queries are frequent and similar)

§8 Appendix

§8.1 Work Division

As such we all have put our heads into the discussions and each and every step that we took but we did a specific allocation of work to make things easier. The work division is as follows:

1. Ashwani Kumar Kamal: Writing the code to generate the embeddings of tf-idf and bert-pp and reading the papers.
2. Hardik Pravin Soni: Writing the code to generate the embeddings of bert_nli and bert_pp and reading about S2ORC dataset and reading the papers.
3. Sourabh Soumyakanta Das: Writing the code for computing the metrics for different models and analysing the result and preparing report.
4. Shiladitya De: Writing the code for computing the metrics and analysing the results as well as preparing presentation and reading papers.

§8.2 Code Snippets

Few code snippets are attached for the different metrics used:

Ranked Precision:

```
def r_precision(r):
    r = np.asarray(r) != 0
    z = r.nonzero()[0]
    if not z.size:
        return 0.
    return np.mean(r[:z[-1] + 1])
```

Precision@k:

```
def precision_at_k(r, k):
    assert k >= 1
    r = np.asarray(r)[:k] != 0
    return np.mean(r)
```

Average Precision:

```
def average_precision(r):
    r = np.asarray(r) != 0
    out = [precision_at_k(r, k + 1) for k in range(r.size) if r[k]]
    if not out:
        return 0.
    return np.mean(out)
```

Mean Average Precision:

```
def mean_average_precision(rs):
    return np.mean([average_precision(r) for r in rs])
```

NDCG@k:

```
def dcg_at_k(r, k, method=0):
    r = np.asfarray(r)[:k]
    if r.size:
        if method == 0:
            return r[0] + np.sum(r[1:] / np.log2(np.arange(2, r.size + 1)))
        elif method == 1:
            return np.sum(r / np.log2(np.arange(2, r.size + 2)))
        else:
            raise ValueError('method must be 0 or 1.')
    return 0.


def ndcg_at_k(r, k, method=0):
    dcg_max = dcg_at_k(sorted(r, reverse=True), k, method)
    if not dcg_max:
        return 0.
    return dcg_at_k(r, k, method) / dcg_max
```

Recall@k:

```
    def recall_at_k(ranked_rel, atk, max_total_relevant):
    total_relevant = sum(ranked_rel)
    total_relevant = min(max_total_relevant, total_relevant)
    relatk = sum(ranked_rel[:atk])
    if total_relevant > 0:
        recall_atk = float(relatk)/total_relevant
    else:
        recall_atk = 0.0
    return recall_atk
```

Cosine Similarity:

```
def cosine_similarity(doc1, doc2):
  a = np.asarray(doc1)
  b = np.asarray(doc2)
  cos_sim = np.dot(a, b)/(np.linalg.norm(a)*np.linalg.norm(b))

  return cos_sim
```

Jupyter Notebook Link:

https://colab.research.google.com/drive/1PdvyNnlwA4eUyS6pTumNFI6OTbFY-GYc?usp=sharing