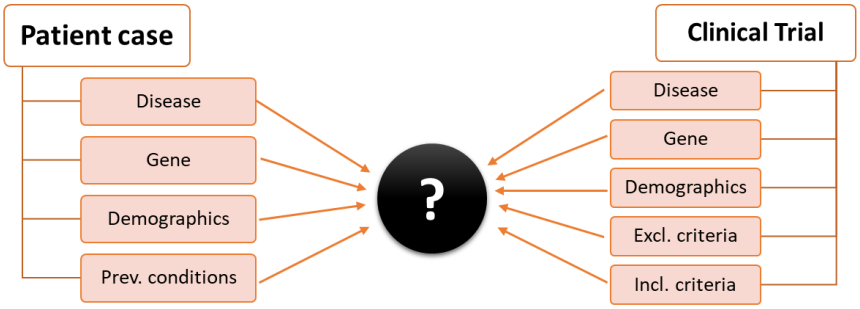
Matching
Patient Cases to Clinical Trials

Information Retrieval Course
2021/2022

Adapted from TREC Clinical Trials TRACK



The vast majority of clinical trials fail to meet their patient recruitment goal. NIH has estimated that 80% of clinical trials fail to meet their patient recruitment timeline and, more critically, many (or most) fail to recruit the minimum number of patients to power the study as originally anticipated. Efficient patient trial recruitment is thus one of the major barriers to medical research, both delaying trials and forcing others to terminate entirely.

In this project you will build a system to retrieve clinical trials from ClinicalTrials.gov, a required registry for clinical trials in the United States. The goal is to find clinical trials where patients can be enrolled.

Requirements

- You should have a computer where you can run experiments and implement the required algorithms. The Google Colab is a good substitute for this setup.
- It is advisable to use an Anaconda Environment with the libraries identified in the provided YAML file.
- It is advisable to have a working JupyterLab and PyCharm installation.
- You're free to use the programming environment of your choice.

Methodology

Protocol

- Read and parse the clinical trial documents.
- Create an index structure for the different fields of the clinical trials
- Read and parse the queries, i.e., the patients case description
- Select a random sample of 80% patient cases to be your training set. The remaining 20% cases will be your test set.
- With the training patient cases, train or calibrate your *retrieval algorithms*.
- With the test patient cases, compute the top 100 candidate clinical trials using a *retrieval algorithm*.
- Measure the success of the proposed algorithm with the proposed metrics.

Dataset

Documents. Clinical trial descriptions can be quite long, but the core aspect of the trial description are the inclusion/exclusion criteria. These are not all-inclusive statements about the trial to the point that other trial information can be ignored, but they are key aspects to defining trial eligibility.

Queries. The queries of this system will be a lengthy (5-10 sentence) patient case description that simulates an admission statement in an EHR. The queries are limited to just the free text description of a patient record, as the structured data in EHRs, while helpful, is outside the scope of this project.

Metrics

To assess the retrieved documents, you will use both system utility metrics and system stability metrics.

For system utility metrics, you should use:

- **Precision@10** to measure the percentage number of correct documents in the top 10 results.
- **nDCG@5** to measure the cumulative gain of retrieving multi-level relevance documents on the top 5 results.
- **Recall@100** to measure how many relevant results are not accessible to users.

For system stability metrics, you should use:

- **Mean Average Precision** to measure the robustness of the compute ranks,
- **Precision-recall curves** are an informative visualization of the performance of the system across the entire rank of results.

The evaluation will further be broken down into *eligible*, *excludes*, and *not relevant* to allow retrieval methods to distinguish between patients that do not have sufficient information to qualify for the trial (*not relevant*) and those that are explicitly excluded (*excludes*).

Phase 1: Base Pipeline

Deadline: 5 Nov

The code snippets and dataset are available here:

https://drive.google.com/drive/folders/17-h0XQyGKED7trvarR38c1OWh-9UypHl?usp=sharing

Reading the Clinical Trials Documents

Examine the code provided to read the documents. Inspect the different sections of a clinical trial document. **Clinical trials will be your corpus of documents.**

Due to the large size of the clinical trials database, you should only load the clinical trials that are in the `qrels-clinical_trials.txt` file. The documents outside that file were not judged by assessors, hence, we don't know if they are relevant or not.

We provided you a parser to read the contents of a compressed file. However, if wish to extract the file, note that it will occupy ~3GB in your disk and because it will extract > 100k files it may freeze your computer when you access the folder with the clinical trials.

Reading the Patient Cases

Examine the code provided to read the patient cases. **Patient cases will be your queries.**

Vector Space Model

Using the scikit-learn implementation of the VSM with TF-IDF weights and cosine distance, build an index for the entire set of documents for which you have ground-truth. Consider the example below.

```
from sklearn.metrics.pairwise import pairwise_distances
from sklearn.feature_extraction.text import TfidfVectorizer

corpus = [
     'This is the first document.',
     'This document is the second document.',
     'And this is the third one.',
     'Is this the first document?',
 ]

# Learn a vocabulary of unigrams and bigrams
index = TfidfVectorizer(ngram_range=(1,2), analyzer='word', stop_words = None)
index.fit(corpus)

# Compute the corpus representation
X = index.transform(corpus)

# Compute the query representation
query = ['document']
query_tfidf = index.transform(query)

# Compute the query-corpus similarity for all documents
doc_scores = 1 - pairwise_distances(X, query_tfidf, metric='cosine')
print(doc_scores)
```

Use only the `brief_title` section of the clinical trial documents.

Language Models

Implement the Language Model with Jelinek-Mercer Smoothing. To access the corpus statistics you can use scikit learn's CountVectorizer class.

Evaluation

Examine the code provided to compute your experimental results. Compute all the mentioned measures. Organize your results into tables and graphs to support your discussion.

We suggest you to use wandb or matplotlib to visualize your data.

Report

The project report should have the following structure:

- Introduction
- Implemented methods
- Experimental setup
- Results discussion
- Conclusions

You will be writing your project report incrementally. On each phase you will be adding new portions to each section of your report.

Your report must be limited to 4 pages and must follow one these templates:

- Word: https://www.springer.com/gp/authors-editors/journal-author/word-template-zip-154-kb-/22044
- Latex: https://www.overleaf.com/latex/templates/springer-lecture-notes-in-computer-science/kzwwpvhwnvfj

Suggestions

1. **Pickle:** To save time you can use pickle to save and load variables to disk files:

   https://wiki.python.org/moin/UsingPickle

Phase 2: LETOR Models

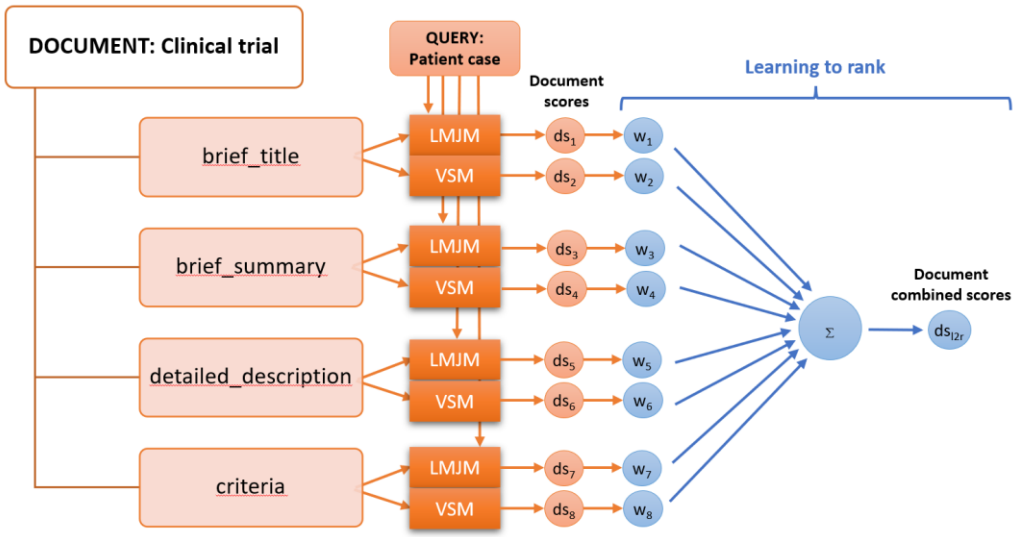
Deadline: 3 Dec

The code snippets and dataset are available here:

https://drive.google.com/drive/folders/17-h0XQyGKED7trvarR38c1OWh-9UypHl?usp=sharing

Learning to Rank Models

In this phase of the project, you will use the different sections of the clinical trial document as predictors of the clinical trial relevance for each patient case. The VSM and the LMD models must be used to compute such predictor signals. Learning to rank models can learn an ensemble of models to combine the strengths of different algorithms. Train **linear regressor**, e.g. the logistic regression model, to learn how to combine the VSM and LMD of the different document sections into a document relevance score.



You should consider the following text fields (or a combination of):

- brief_title
- detailed_description
- brief_summary
- criteria

Other non-text fields, that should be applied on the final rank, are the categorical and numerical ones:

- gender
- max age
- min age

In this phase of the project, you should plan your work carefully.

Implementation guidelines

Individual fields and retrieval models **12 November**

1. Generalize your LMJM and VSM implementation.
2. Compute the retrieval performance that you achieve with each clinical trial field.
3. For documents with missing fields, you must find a reasonable solution for handling that missing data.

   You should also rely on the analysis tools and skills that you developed in the first phase of the project.

LETOR model **19 November**

4. Use your training queries to select all pairs of (query, document) to train a relevance predictor.
5. For each (query, document) pair, the document score of each field/model combination should be used as an input feature to the LETOR model (ds_1 in the previous figure). Note that you will have relevant (query, document) pairs and non-relevant (query, document) pairs, i.e., positive and negative pairs.

   For the sake of efficiency, it is critical to write intermediate results into pickle files.
6. Train a logistic regression classifier to discriminate between relevant from non-relevant (query, document) pairs.

Final ranks **26 November**

7. To compute a rank, you should use the logistic regression coefficients (**coef_**)
8. All filters based on age and gender should be applied at this stage.

Suggestions

2. **Imbalanced data:** your training data is highly imbalanced. The class_weight parameter of the logistic regression allows you to mitigate this effect.
3. **Overfitting:** The regularization parameter, **C**, allows you to control the overfitting of your model.
4. **Explainability**: Understanding and explaining the decisions of ranking models is an important task that will give the user extra information to make his decision. You can use the coef_ variable to inspect the importance of each individual field and retrieval model.
5. **Explainability**: More thorough analysis can be done with Shap to inspect the feature importance of your models.https://github.com/slundberg/shap
6. **Results**: (i) analyze the per field+model retrieval performance, (ii) analyse the LETOR retrieval performance; (iii) analyse how the training parameters impact the overall results; (iv) try to explain the model decisions using the suggested tools.

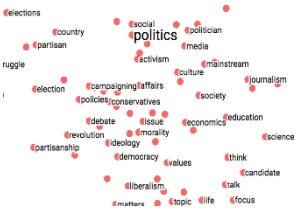
Phase 3: Contextual Embeddings

Deadline: 10 Jan

(15%) Input formatting and Tokenization

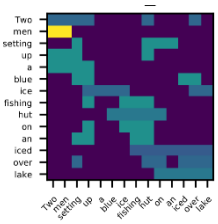
1. Study the code provided for phase 3: https://wiki.novasearch.org/wiki/IR_NLP_2021
2. Select a BERT model and load it in your project.
3. Select (1) a query, (2) a relevant document and (3) a non-relevant document.
    a. Use only the detailed description field.
4. Format the BERT input according to the next sentence prediction task.
5. Run the tokenizer and examine how it split the text into tokens.

(25%) Layer embeddings visualization



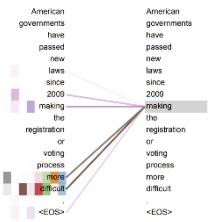
6. Extract 10 token embeddings of the first and last layer.
7. Plot the token embeddings of the first-layer and the token embeddings of the last-layer (different colors).
8. Examine and do a critical analysis of the embedding's visualizations.

(25%) Layer embeddings similarity visualization



9. Compute the similarity between the input and output embeddings of all layers.
10. Visualize the similarities matrix for different pairs of queries vs documents and do a critical analysis.

(25%) Self-attention head visualization



11. Using the provided implementation, visualize the attention weights of each head (use your preferred visualization).
12. Examine and do a critical analysis of the embedding's visualizations.

(10%) Downstream task: Learning to rank

13. Extract the embedding of the CLS token of the last layer.
14. Using the same architecture of the second phase, train a ranking model based on the CLS output embedding.
15. Run the same evaluation as in phase 2.