CS132a Information Retrieval spring 2021

# Assignment 5: Learning Elasticsearch

(Due April 21, 2021)

Update notes:
04-18-2021: Updated description of `evaluate.py`.

## Overview

This assignment is intended to help you get familiar with Elasticsearch (ES) while doing a little information retrieval "research" to compare alternative approaches to document indexing and querying. We will be using the TREC 2018 core corpus subset and five TREC topics with relevance judgments for evaluation.
For background, a high level overview of ES functionality with several case studies is found at https://apiumhub.com/tech-blog-barcelona/elastic-search-advantages-books/.

In this assignment, we will provide example code for:

- Populating and querying a corpus using ES
- implementing NDCG (normalized discounted cumulative gain) evaluation metric
- experimenting with "semantic" indexing and searching using fastText and BERT

You will:

- Index the corpus into ES with default standard analyzer and your customized one for the text fields.
- Integrate ES into your Flask service for interactive search. Beside the traditional lexical search as we did before, your system should also allow the user to select the text representation to use for search.
- Create a command line interface that runs a TREC query against an index and search options and shows the evaluation result using NDCG.
- Evaluate the performance of 5 provided TREC queries using NDCG. For each query, you should produce a result table along with a brief analysis.

On LATTE you will find the file `cosi132a_hw5.zip` that is structured as the following. Details are provided in the following sections.

```
├── embedding_service/
├── es_service/
├── evaluate.py
├── example_analyzer.py
├── example_embedding.py
├── example_query.py
├── hw5.py
├── load_es_index.py
├── metrics.py
├── pa5_data/
├── requirements.txt
```

```
├── scripts.sh
├── templates/
└── utils.py
```

Make sure you have installed all the packages specified in the `requirements.txt`. The latest distribution of the package elasticsearch-dsl-py does not support ES script score query yet which is essential to search documents based on cosine similarity. To make it work, after you install this package, add the following code at the end of `elasticsearch_dsl/query.py`:

```
class ScriptScore(Query):
    name = "script_score"
    _param_defs = {"query": {"type": "query"}}
```

# Data Resource

- For this assignment, you will use a larger subset of TREC 2018 core corpus that has already been processed. Specifically, each document has the following fields:

```
doc_id          # original document id from the jsonline file
title           # article title
author          # article authors
content         # main article content (HTML tags removed)
date            # publish date in the format "yyyy/MM/dd"
annotation      # annotation for its relevance to a topic
ft_vector       # fastText embedding of the content
sbert_vector    # Sentence BERT embedding of the content
```

  For the `annotation` field, the value is stored as the format of `topic_id-relevance`. The relevance can be either 0, 1 or 2, which represents irrelevant, relevant or very relevant. Topic id can be mapped to the topic title, description and narrative in the file `pa5_data/topics2018.xml`. In `utils.py`, the method `parse_wapo_topics` is provided to parse the topic xml file. If the annotation field is empty, it can be considered that this document is irrelevant to any topics. The subset can be downloaded from here: https://drive.google.com/file/d/1Y03Cgf-84lua5cmBEt8-4JQKbdgvu3vG/view?usp=sharing

# Elasticsearch Basics

- Download ES from https://www.elastic.co/downloads/past-releases#elasticsearch. Make sure you are choosing Elasticsearch 7.10.2. To start the ES engine:

```
$ cd elasticsearch-7.10.2/
$ ./bin/elasticsearch
```

- To test your ES is running, open http://localhost:9200/ in your browser. You should be able to see the health status of your ES instance with the version number, the name and more. Note that you should keep ES running in the backend while you are building and using your IR system.
- Elasticsearch is Java-based and API driven. In this assignment, we will use elasticsearch-py and elasticsearch-dsl to operate ES in Python. You will want to go over the

example code along with the package documentations to understand the underlying interaction mechanism.
- Read the official Elasticsearch Guide to know how it works from a high-level. You may want to focus on sections *Mappings*, *Text analysis* and *Query DSL*, and skim over the others. You can also ignore the examples from the tutorial as we will use Python ES wrappers instead of its Search API.

## Text Embedding Basics

- In addition to the traditional lexical search based on BM25 (default in ES), you will also explore a simple setting of semantic search based on the cosine similarity between the embeddings of query text and content text. Specifically, you will run a default BM25 query first to retrieve top K matched documents, and then run a second query filtering for those K docs and reranking by cosine similarity with the document embeddings.
- We will use two types of pretrained embeddings from fastText and Sentence Transformer. fastText can provide embeddings for tokens. Compared with word2vec, its training is more efficient and it considers subword information. When using fastText, to get a single embedding for a query or the document content, we simply average all the token embeddings from the text of the query or document. Sentence Transformer can directly encode a span of text into an embedding. Check this and this for a nice introduction to word embedding and transformers.
- You need to download the pretrained fastText embedding on wiki news and put it into `pa5_data/` folder. You can click this link to download. You don't need to download any pretrained model for sentence transformers, it will be loaded the first time it's called.

## Embedding Service

- In the starter code folder you downloaded from LATTE, `embedding_service/` module allows you to encode a query or document into an embedding. You can start the server by using the command line tool. Check `embedding_service/server.py` and `scripts.sh` for argument options and usage. After the server is running, you should be able to instantiate a client in your Python code to encode your text. Check `example_embedding.py` for how to use it. You don't need to implement anything for this part, but make sure you understand the basic code structure of this module and know how to use it.

## Elasticsearch Service

- In the starter code folder you downloaded from LATTE, `es_service/` module defines the document mapping and the index for loading the WAPO docs into ES. In the file `es_service/doc_template.py`, you need to build a custom analyzer that consists of a snowball stemmer, an asciifolding token filter, a lowercase token filter and a standard tokenizer. It will be used for the `custom_content` field from `BaseDoc` mapping.

- File `example_analyzer.py` provides examples that allow you to simulate the output from ES analyzers. You can test your customized analyzer in this file before it's been applied to the ES index.
- To build the index and load the data into ES, check `load_es_index.py` and `scripts.sh` for argument options and usage.
- File `example_query.py` provides some common types of query supported by ES. Play around with it and think about how it can be integrated into your system. Make sure you have started your ES server before you run the queries. Also start your embedding server if your query involves computing cosine similarity between embeddings.

# Programming Problems

- In the file `es_service/doc_template.py`, implement a custom analyzer that consists of a snowball stemmer, an asciifolding token filter, a lowercase token filter and a standard tokenizer. It will be used for the `custom_content` field from `BaseDoc` mapping.
- Create a command line interface in `evaluate.py` that runs 3 forms of each TREC query (title, description, narrative) against each of these retrieval options, returning the top 20 matches:

  BM25 + default analyzer
  BM25 + custom analyzer
  Reranking with fastText embeddings + default analyzer
  Reranking with sentence BERT embeddings + default analyzer

  You will need to implement everything needed for `evaluate.py` (search, evaluation, command line arguments, etc) all by yourself. The command in the `script.sh` is the expected form of your command line arguments, and you should implement the `evaluate.py` that can be executed using that command. Evaluate the performance of the first 5 TREC queries from `pa5_data/topics2018.xml` using NDCG@20 (implemented in `metrics.py`). For each query, you should produce a 4x3 table with 1 row per search type and 1 column per query type. The value of each cell is the NDCG@20 value. Include your results tables along with a paragraph of analysis of each table in the report.
- In the file `hw5.py`, Integrate elasticsearch into your Flask service for interactive search, allowing the user to select the text representation to use for search. For simplicity, your query should only match the text indexed in the `content` field or `custom_content` field. Provide an option (e.g. radio buttons) for the user to choose among the four retrieval options mentioned above.

# Notes

- A more popular (and more complicated) approach to integrate embeddings into document retrieval is to train a neural reranker model that leverages the relevance between query and each passage from the whole document. Check this paper Cross-Domain Modeling of Sentence-Level Evidence for Document Retrieval for more details if you are interested.

- ES is quite fast in terms of building the index. It took me ~2 minutes to index the TREC subset.

## Submission

(1) A zip archive named `hw5_<last_name_first_name>.zip` containing:
    - All the necessary files for running your ES-based Flask application.
    - All the necessary files for running the evaluation.
    - `README.pdf` with a clear running instruction and system design description; result tables and result analysis; a short write-up of how many hours you spent on this assignment, how difficult is this assignment and some general comments you have.

(2) Your `README.pdf` submitted separately. Do not include any embedding or date files in the submission.