INFO 300 – Assignment 2 (10 points)

Weimao Ke, Drexel University October 28, 2020

1 Objectives

The assignment will allow each student to build a small text collection, index the text documents with a proper analyzer, and perform full-text searches. You will also examine related term statistics to compute TF*IDF weights and cosine similarities.

Please create a report the Markdown (.md) format, named **A2_YourDrexelIDs.md**, to document your workd in this assignment.

2 Tasks and Steps

2.1 Text Collection (2 points)

Please identify 20 research papers/articles. The articles may be from readings of other courses or elsewhere. For each article, please collect the following data (only):

- Author: the list of author names;
- Title: the title of the article;
- Abstract: the abstract of the article;
- Citations: the number of times the article has been cited[1]

Here is an example in the JSON format:

```
{
  "author":"Weimao Ke and Javed Mostafa",
  "title":"Scalability of findability: effective and efficient IR operations in
      large information networks",
  "abstract":"It is crucial to study basic principles that support adaptive and
      scalable retrieval functions in large networked environments such as the
     Web, where information is distributed among dynamic systems. We conducted
     experiments on decentralized IR operations on various scales of
     information networks and analyzed effectiveness, efficiency, and
     scalability of various search methods. Results showed ...",
  "citations":12
}
```

Prepare all your article abstracts in the above JSON format:

- Save all your data (article JSON) in an **A2_YourDrexelIDs_data.json** file;
- List one article JSON as an **example in Markdown documentation** as well.

[1] Repositories such as ACM Digital Library and Google Scholar do provide citation counts.

2.2 Index Mapping (2 points)

Identify an index name such as:

*yourdrexelid*_**info300_articles**

under which you will load and index the collected articles.

Create mappings for the above index so that:

- The *author*, *title*, and *abstract* fields will indexed as the **text** data type.
- **Stemming** should be used on *title* and *abstract*, but **NOT** on the *author* field.
- The *citations* field is **integer**.

Hint: Use the *english* analyzer if you want stemming by default; the *standard* analyzer, on the other hand, does not use stemming.

Here is an example of new mappings with analyzer specification for a field called *message*:

```
PUT /yourdrexelid_articles
{
  "mappings": {
    "properties": {
      "message": {
        "type": "text",
        "analyzer": "standard"
      }
    }
  }
}
```

To add new field properties to an existing mapping:

```
PUT /yourdrexelid_articles/_mapping
{
    "properties": {
      "field1": {
        "type": "text",
        "analyzer": "english"
      },
      "field2": {
        "type": "text",
        "analyzer": "english"
      }
    }
}
```

Include the following in your documentation:

- The **commands/requests** you used for mappings;
- The **responses** (acknowledgement) from the ElasticSearch server;

Make sure you have the proper mappings before the next step.

2.3 Data Indexing (1 point)

After you have properly configured the mappings for your index, load all your articles to the index and index them with *PUT*, for example:

```
PUT /yourdrexelid_articles/_doc/1
{
  "author": "...",
  "title": "...",
  "abstract": "...",
  "citations": ...
}
```

Make sure you have indexed all your article abstracts.

Include one **request/response** as an example in your documentation.

2.4 Search and Retrieval (1 point)

Form a search query with **three keywords** relevant to your collection of articles, for example *information science principles*, and search the *abstract* field on the index:

```
GET /yourdrexelid_articles/_search
{
    "query": {
        "match" : {
            "abstract" : {
                "query" : "information science principles"
            }
        }
    }
}
```

Please form **your own query** and make sure the query keywords do appear in (at least some of) your documents (abstracts). Identify the *top three* hits.

Include the following in documentation:

- The **search request** with the match query;
- The top **three articles** in the JSON format.

2.5 Manual Examination

We now will examine the query keywords, identify related statistics, and compute their TF*IDF weights for the top three hits (from the previous step), etc.

A. Term Stats (2 points) For each of the top hits, use the term vectors API to identify related statistics for terms in the document.

To quickly identify the DF (document frequency) for each query term in the index, you can use the query as a pseudo-document and analyze its term vectors:

```
GET /yourdrexelid_articles/_termvectors
{
    "doc": {
      "abstract": "information science principles"
    },
    "term_statistics" : true,
    "field_statistics" : false,
    "positions": false,
    "offsets": false
}
```

To identify TF (term frequency) of every term in document id 3, for example, you can use the following:

```
GET /yourdrexelid_articles/_termvectors/3
{
    "fields" : ["abstract"],
    "term_statistics" : true,
    "field_statistics" : false,
    "positions": false,
    "offsets": false
}
```

From the above requests, you should be able to collect DF and TF statistics. Compile the following table for the top three hits represented by query keywords:

	information		science		principles	
Doc ID	Term Freq.	Doc Freq.	Term Freq.	Doc Freq.	Term Freq.	Doc Freq.

B. TF*IDF (1 point) Based on the above table of TF and DF, compute the terms' TF*IDF weights for each document:

	TF*IDF weights			
Doc ID	information	science	principles	SUM

Use the *sum of TF*IDF weights* to rank the documents. Are they ranked in the same order as in the results from ElasticSearch?

C. Cosine Similarity (1 point) Treat the query as a vector in the dimensional space (with each of query keywords as one dimension). The query can be represented by the vector of the query terms' IDF weights:

Query Vector based on IDF weights		
information	science	principles

Compute the cosine similarity between the query vector and each of the document vector (using weights from the last two tables):

- Cosine(Q, doc#) =
- Cosine(Q, doc#) =
- Cosine(Q, doc#) =

Rank the document accordingly. Are they ranked in the same order?

3 Assignment Submission

Please document all your work and submit the **documentation** and **JSON data** to blackboard.

4 References

Please refer to the following references for specifics with ElasticSearch:

- https://www.elastic.co/guide/en/elasticsearch/reference/current/mapping.html
- https://www.elastic.co/guide/en/elasticsearch/reference/7.5/analysis.html
- https://www.elastic.co/guide/en/elasticsearch/reference/current/docs-termvectors.html

I've compiled a couple of presentations on related ElasticSearch operations on blackboard.

Please also refer to lecture slides on related concepts such as tokenization, normalization, TF*IDF, and cosine similarity, etc.