Search Engine Presentation

Group B:
Ahmad Al-Waili - 160354605
Bobby Dewan Akram Khaleque - 150033770
Abdul Gaffar Abdul Khadeer - 190687836
Saif Abrahim Saleem - 160437333

Software and Packages Used

- Front end:
  - Axios
  - Vue JS
- Back End:
  - Python
  - Node JS
  - Elasticsearch
  - HTTPS

Data set, techniques and methods used

Jokes data set:

- Contains about over 100 jokes
- Each joke has its average rating, calculated from 70,000 user reviews
- Keywords for each joke
  - Preprocessing
  - Indexing
  - Retrieval methods

Joke ID	Joke Text	Average Rating
0	A man visits the doctor. The doctor says "I have bad news for you.You have cancer and Alzheimer's disease". The man replies "Well,thank God I don't have	0.668959897

Figure 1: single sample of dataset entry (AverageJokeReviews.csv)

94	Just a thought	0.62780962
95	Two attorney	0.836661173
96	A teacher	0.901376164
97	Age and Womanh	0.531440911
98	A bus station is	0.048197522
99	Q: Whats	0.569435085

Figure 2: Sample of the last five joke entries (AverageJokeReviews.csv)

Preprocessing

- Stemming - Resolving each word presented in a sentence to its root (normalised) word through stripping the suffixes of the the word (-ed, -ise, -s, etc).
  - ‘Cats’ -> ‘Cat’
  - ‘Troubling’, ‘Troubles’, ‘Troubled’ -> ‘Troubl’
- Lemmatization - Resolving each word presented to its lemma (the correct root belonging to its language) word; The canonical/dictionary form of the word.
  - ‘The boy’s cars are different colours’ -> ‘The boy car be differ colour’
- Removing Stopwords - A crucial component when preprocessing natural language, effectively removing all redundant words (stopwords which hold little meaning).

Indexing methods

- TF-IDF using keyword search
- Inverted indexing - Elasticsearch
  - Inverted Index consists of a list of all the unique words that appear in any document
- Boolean model
  - Searches only for the query, not the keywords
    - It has to find the full joke/rating
  - Multi Match method where the rating and text can be searched for
    - Uses an array method in the query method which makes the JSON data: [‘title’, ‘authors’, ‘name’]
  - search_bool.js
- Fuzzyness method used - Elasticsearch
  - Creates a set of all possible variations or expansions of the search term within a specified edit distance

Retrieval methods

- Ranking uses scores of the search results.
  - The highest is placed at the top of the list.
- Keywords and categories are ranked based on the search itself.
- Joke ratings ranked as well
  - Again from highest to lowest
  - So if words like ‘Funny’ or ‘Good’ appear in the search we show the highest rated joke first
- Vector model
  - Elasticsearch makes use of the Lucene scoring formula, which represents the relevance score of each document with a positive floating-point number
- Aggregation
  - An aggregation is a unit of work that builds analytic info over a set of documents
  - The context of execution defines what this document set is.

Evaluation

- DCG Discounted Cumulative Gain
  - DCG measures the usefulness, or *gain*, of a document based on its position in the result list.
    - Relevant documents have higher ranking or *_score*
    - Relevant documents are more useful than not relevant documents.
  - Cumulative Gain
    - CG is the sum graded of all ‘relevant’ documents in the search
    - The value of CG is unaffected of the ‘order’ of results from the search
  - Cumulative Gain is sometimes called Graded Precision as it is identical to the Precision metric if the rating scale is binary.

$$\mathrm{CG_p} = \sum_{i=1}^{p} rel_i$$

Evaluation

$$\mathrm{DCG_p} = \sum_{i=1}^{p} \frac{rel_i}{\log_2(i+1)} = rel_1 + \sum_{i=2}^{p} \frac{rel_i}{\log_2(i+1)}$$

- DCG
  - The premise of DCG compared to CG is that relevant documents that are shown lower in the list are penalised.
  - The traditional formula of DCG accumulated at a particular rank position p is defined as

```
{
    "requests": [
    {
        "id": "query_1",
        "request": { "query": { "match": { "joke_text": "funny" }}},
        "ratings": [{ "_index": "my_index", "_id": "doc1", "rating": 1 }]
    }],
    "metric": {
       "dcg": {
            "k" : 20,
            "normalize": false
       }
    }
}
```

```
"unrated_docs": [
    {
        "_index": "jokes-index",
        "_id": "oxFt9HABMm_Y9u-2n305"
    },
    {
        "_index": "jokes-index",
        "_id": "phFt9HABMm_Y9u-2n305"
    },
    {
        "_index": "jokes-index",
        "_id": "3RFt9HABMm_Y9u-2n306"
    }
],
"hits": [
    {
        "hit": {
            "_index": "jokes-index",
            "_type": "jokes",
            "_id": "oxFt9HABMm_Y9u-2n305",
            "_score": 4.8278265
        },
```

Evaluation Rating - eval_search.js

```
Rated documents
        oxFt9HABMm_Y9u-2n305 - jokes - 4.8278265
        phFt9HABMm_Y9u-2n305 - jokes - 4.6858034
        3RFt9HABMm_Y9u-2n306 - jokes - 2.1722803
```

```
{
    "requests": [
    {
        "id": "query_1",
        "request": { "query": { "match": { "joke_text": "funny" }}},
        "ratings": [{ "_index": "jokes-index", "_id": "oxFt9HABMm_Y9u-2n305", "rating": 1 }]
    }],
    "metric": {
       "dcg": {
            "k" : 20,
            "normalize": false
       }
    }
}
```

```
"hits": [
    {
        "hit": {
            "_index": "jokes-index",
            "_type": "jokes",
            "_id": "oxFt9HABMm_Y9u-2n305",
            "_score": 4.8278265
        },
        "rating": 1
    },
```

Demonstration

- Frontend
- Backend
- Node JS
- Vue JS
- Elasticsearch

Project Evaluation

The main observations prevalent about the search engine's performance:

- ➔ The ranking of jokes by users, within the joke index proved to be useful when returning jokes from the search engine. This in turn allowed for quick and effective query processing (increasing performance).
- ➔ An increase in performance using pre-processing methods as important information is extracted from unstructured text queries.
- ➔ The search engine execution speed was due to the combination of the boolean and vector space models as well as the inverted index data structure. This is due to the fact that the boolean model evaluates the boolean similarity of documents which is complemented by the vector space model which measures the relevance of documents in a vector space. This is further complemented by the inverse index data structure makes indexing words using TF-IDF quicker as all unique words have been stored.