Project Report for Information Retrieval

Maria Eleni Chochlidaki
Michail Samaritakis

Important Note: Detailed instructions on how to run and test the project can be found at the README.md file in the Github repository.

Outline:

Detailed Breakdown:

1. Introduction

- Project Overview:
The purpose of this project is to develop an "Information Retrieval System" utilising a web crawler and the Elasticsearch tool. This system will be capable of downloading textual data from a chosen website, indexing it, and then allowing efficient retrieval of information using vector space search with TF-IDF representation and cosine similarity. The project is divided into two main parts: the development of the web crawler for data collection and the implementation of the information retrieval system for indexing and searching the data.

In the first part, a custom web crawler is developed to scrape textual data from specified web pages. This data is then imported into an Elasticsearch database. The setup includes configuring Elasticsearch and Kibana for data storage, indexing, and visualisation. The crawler should be capable of extracting various types of textual data, such as articles, comments, discussions, and advertisements, from a target website.

In the second part, a simple Information Retrieval (IR) system is implemented. This involves the use of vector space search techniques, where documents and queries are represented using TF-IDF weights. Cosine similarity is then used to measure the relevance of documents to user queries. The objective is to create an efficient search mechanism that retrieves the most relevant documents based on the given queries. Additionally, the project includes visualising the retrieved data using Kibana to create insightful visualisations such as pie charts, line charts, and heat maps.

For our project we decided to work with the https://www.motorcycleforum.com/ website.

- Objectives:
The main objectives of this project are as follows:

1. Develop a Web Crawler:
   - Design and implement a custom web crawler to download textual data from a selected website.
   - Ensure the crawler can handle various types of data, including articles, tags, titles and in general information about each thread of the forum.
   - Store the scraped data in a format suitable for indexing in Elasticsearch.

2. Set Up Elasticsearch and Kibana:
   - Install and configure Elasticsearch for data indexing and storage.
   - Install and configure Kibana for data visualisation.
   - Create indices in Elasticsearch to store the data retrieved by the web crawler.

3. Implement Information Retrieval System:
   - Develop a simple IR system using vector space search with TF-IDF representation.
   - Implement cosine similarity to measure the relevance of documents to user queries.
   - Create a user-friendly interface for querying the database and retrieving relevant documents.

4. Data Visualization:
   - Utilise Kibana to create visualisations that provide insights into the indexed data.
   - Develop dashboards that display key information and trends derived from the data.

By achieving these objectives, the project aims to demonstrate the integration of web crawling, data indexing, and information retrieval technologies to create a functional and efficient search system.

2. Crawler Implementation

- Crawler Design and Development

The crawler was designed to efficiently extract data from the target website, focusing on user comments, metadata, and post information. The architecture consisted of a modular approach where each component handled a specific task. The main components included:
1. URL Fetcher: Responsible for sending HTTP requests and handling responses.
2. HTML Parser: Extracts relevant data from the fetched HTML content using selectors.
3. Data Cleaner: Processes raw data to remove unnecessary information and standardised formats.
4. Data Storage: Stores the cleaned data in a structured format, the preprocessing for Elasticsearch is done later.

The crawler was designed to be robust and scalable, capable of handling large volumes of data with a small amount manual intervention.

- Tools and Technologies Used

- Java: The primary programming language for implementing the crawler, chosen for its robustness and efficiency.
- Selenium: A Java library used for parsing HTML and Javascript loaded pages and extracting data.
- Elasticsearch Java Client: To index the collected data into Elasticsearch and generally to perform CRUD(Create, Read, Update, Delete) operation.
- JSON: For structuring the extracted data and interactions with Elasticsearch.

- Data Collection Process

Site Selection

The specific website was selected based on the following criteria:
- Relevance: Contains a rich set of user-generated content related to motorcycles.
- Accessibility: Publicly accessible content without requiring authentication.
- Data Variety: Offers a mix of posts, comments, and metadata useful for analysis.
- Website conformity: The website allows web crawling (robots.txt file from the website)

- Data Types Collected

- URL
- Title
- Breadcrumbs
- Content
- Author
- Date

- Tags

- Challenges and Solutions
  - Handling Dynamic Content: Some parts of the website contained dynamic content loaded via JavaScript. This was resolved by using a headless browser like Selenium to render the page and extract the complete HTML, compared to the initial version of the crawler that used Jsoup which is only compatible with static HTML web pages.
  - Rate Limiting and Captchas: Encountering rate limits and captchas was mitigated by implementing polite crawling practices such as respecting robots.txt, using delays between requests, and rotating IP addresses.
  - Data Cleaning: The raw HTML contained various unwanted tags and scripts. This was addressed by using Jsoup's parsing capabilities to extract only the necessary information and clean the data.
  - Scalability: Ensuring the crawler could handle large volumes of data involved optimising the code for performance and using multi-threading to parallelize the crawling process. Additionally, using batch processing for indexing data into Elasticsearch improved efficiency.

The crawler implementation was successfully designed to navigate, extract, clean, and index data from the target website, laying a solid foundation for further data analysis and visualisation in Elasticsearch and Kibana.

3. Elasticsearch Setup and Indexing
- Setting Up Elasticsearch and Kibana
Step-by-step Setup Process can be found in the README.md file in the Github repository.

- Data Parsing and Structuring
Methods Used to Parse and Index Data:
1. Data Parsing:
  - The data collected by the web crawler are then processed to be ready for indexing in Elasticsearch.

3. Index Creation:
  - An index was created in Elasticsearch using the following command in Kibana's Dev Tools:

```
PUT /motorcycle_forum
{
 "mappings": {
  "properties": {
   "content": {
    "type": "text",
    "term_vector": "with_positions_offsets_payloads"
   }
  }
 }
}
```

and the mapping we followed can be found on the 'mapping.json' file in the Github repository.

4. Batch Indexing:
   - Data was indexed in batches using the Bulk API to improve performance and efficiency.

- Document Editing (CRUD Operations)
Description of CRUD Operations:
1. Create:
   - Documents were created and indexed using the code found in the 'CRUD.java' file.

2. Read:
   - Documents were retrieved by ID using the code found in the 'CRUD.java' file.

3. Update:
   - Documents were updated using the code found in the 'CRUD.java' file.

4. Delete:
   - Documents were deleted using the code found in the 'CRUD.java' file.

*Instructions for all the above can be found in the README.md file in Github.

- Challenges and Solutions
Issues Faced and Solutions:
1. Elasticsearch Configuration Issues:
   - Issue: Misconfigurations in the `elasticsearch.yml` file leading to cluster formation issues.
   - Solution: Ensured proper configuration settings and reviewed Elasticsearch logs to identify and fix configuration errors.

2. Data Formatting Errors:
   - Issue: JSON parsing errors during indexing.
   - Solution: Implemented robust error handling and validation checks to ensure data is correctly formatted before indexing.

3. Bulk Indexing Performance:
   - Issue: Slow performance during bulk indexing.
   - Solution: Optimised the batch size and used multi-threading to parallelize the indexing process.

4. CRUD Operation Failures:
   - Issue: Inconsistent document states due to partial updates or network issues.
   - Solution: Implemented retries and checks to ensure operations are completed successfully.

5. Maven Dependencies:
   - Issue: A lot of the dependencies could not be found and we had a difficult time to resolve them and get everything running.
   - Solution: A lot of documentation reading and trial and error.

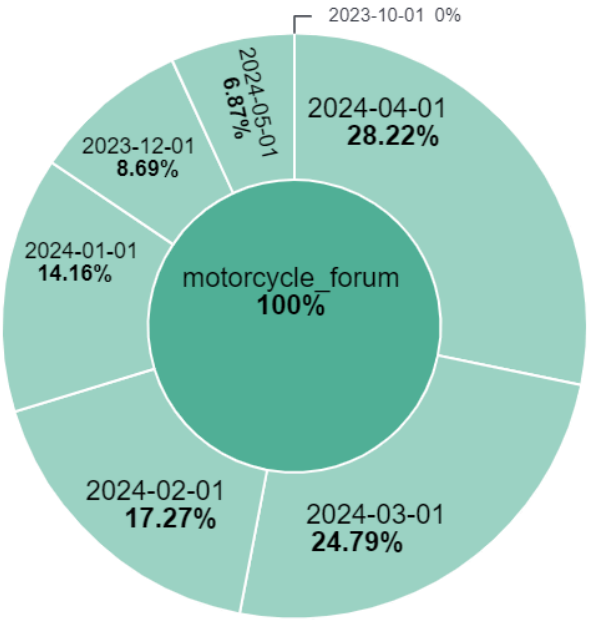
4. Data Visualisation in Kibana

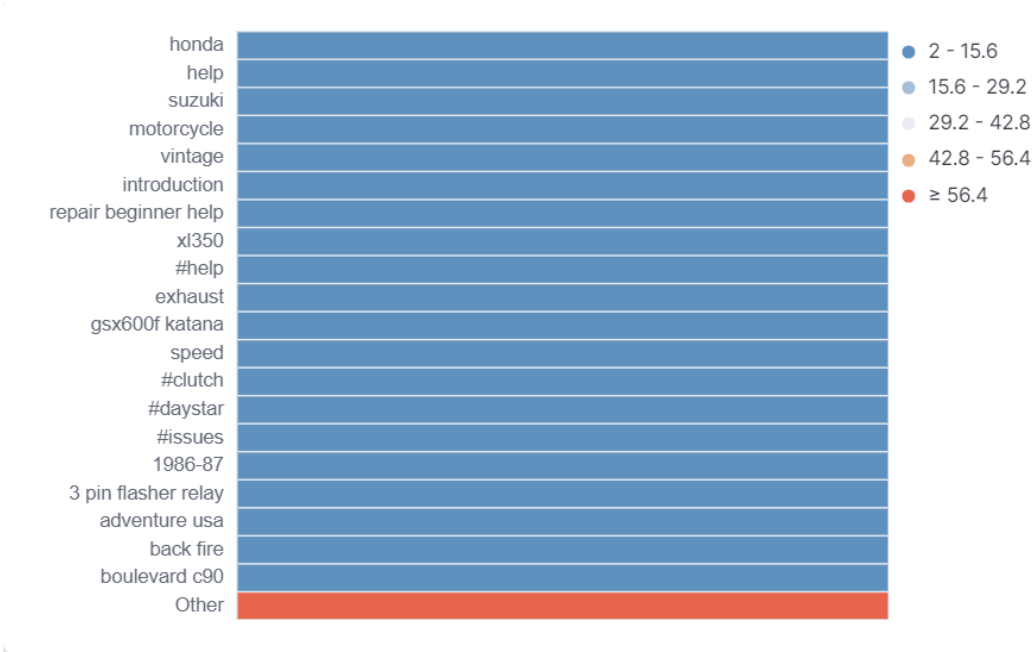
- Visualisation Goals

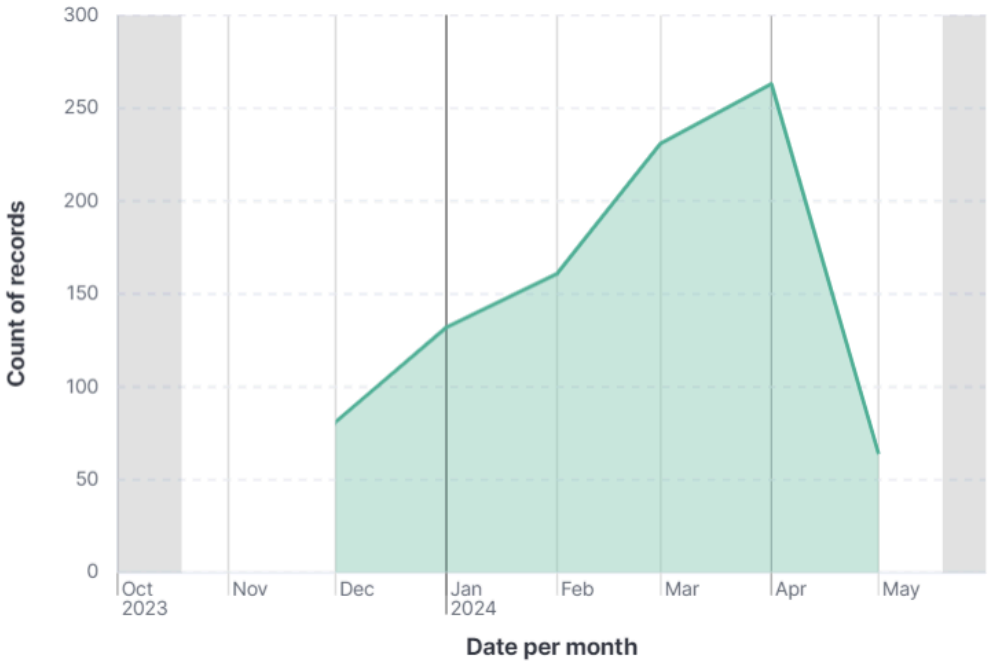
The primary goal of visualising the data in Kibana is to gain insights into the collected and indexed data from the motorcycle forum. By creating visualisations, we aim to:

- Identify trends and patterns in user discussions.
- Understand the distribution of posts across different categories.
- Analyse user activity over time.
- Track the most discussed topics and common keywords.

Visualising the data helps in making informed decisions and deriving actionable insights from the large volume of text data.







*The json export of the charts can be found at the Github repository.

5. TF-IDF Calculation

- Explanation of TF-IDF and Cosine Similarity

Theoretical Background and Importance:

- TF-IDF (Term Frequency-Inverse Document Frequency): TF-IDF is a statistical measure used to evaluate the importance of a word in a document relative to a collection of documents (corpus). The importance increases proportionally to the number of times a word appears in the document but is offset by the frequency of the word in the corpus.

- Term Frequency (TF): Measures how frequently a term appears in a document. It is calculated as:

$$\text{TF}(t, d) = \frac{\text{Number of times term } t \text{ appears in document } d}{\text{Total number of terms in document } d}$$

- Inverse Document Frequency (IDF): Measures the importance of a term in the corpus. It is calculated as:

$$\text{IDF}(t) = \log\left(\frac{N}{\text{DF}(t)}\right)$$

where N is the total number of documents and $\text{DF}(t)$ is the number of documents containing the term t.

- TF-IDF: The product of TF and IDF:

$$\text{TF-IDF}(t, d) = \text{TF}(t, d) \times \text{IDF}(t)$$

- Cosine Similarity: Measures the cosine of the angle between two non-zero vectors in a multi-dimensional space, often used to measure document similarity in information retrieval. It is defined as:

$$\text{cosine\_similarity}(A, B) = \frac{\vec{A} \cdot \vec{B}}{\|\vec{A}\| \|\vec{B}\|}$$

where $\vec{A} \cdot \vec{B}$ is the dot product of the vectors, and $\|\vec{A}\|$ and $\|\vec{B}\|$ are the magnitudes of the vectors.

- Manual Calculation Examples
Step-by-Step Manual Calculation:
Consider a small corpus with three documents:
- Document 1: "the quick brown fox"
- Document 2: "the quick"
- Document 3: "the brown"

Step 1: Calculate Term Frequency (TF)
- TF("the", Document 1) = 1/4 = 0.25
- TF("quick", Document 1) = 1/4 = 0.25
- TF("brown", Document 1) = 1/4 = 0.25
- TF("fox", Document 1) = 1/4 = 0.25

Step 2: Calculate Document Frequency (DF)
- DF("the") = 3
- DF("quick") = 2
- DF("brown") = 2
- DF("fox") = 1

Step 3: Calculate Inverse Document Frequency (IDF)
- IDF("the") = \(\log \left(\frac{3}{3} \right) = 0\)
- IDF("quick") = \(\log \left(\frac{3}{2} \right) = 0.176\)
- IDF("brown") = \(\log \left(\frac{3}{2} \right) = 0.176\)
- IDF("fox") = \(\log \left(\frac{3}{1} \right) = 0.477\)

Step 4: Calculate TF-IDF
- TF-IDF("the", Document 1) = 0.25 * 0 = 0
- TF-IDF("quick", Document 1) = 0.25 * 0.176 = 0.044
- TF-IDF("brown", Document 1) = 0.25 * 0.176 = 0.044
- TF-IDF("fox", Document 1) = 0.25 * 0.477 = 0.119

- Programmatic Calculation
Code and Methods:
The following Java code snippet demonstrates how TF-IDF could be calculated programmatically:

```java
public class TfIdfCalculator {
    private Map<String, Integer> docFreq = new HashMap<>();
    private int totalDocs;
```

```
    public TfIdfCalculator(int totalDocs) {
        this.totalDocs = totalDocs;
    }

    public void calculateDocFreq(List<String[]> documents) {
        for (String[] document : documents) {
            Set<String> uniqueTerms = new HashSet<>(Arrays.asList(document));
            for (String term : uniqueTerms) {
                docFreq.put(term, docFreq.getOrDefault(term, 0) + 1);
            }
        }
    }

    public double tf(String term, String[] document) {
        long count = Arrays.stream(document).filter(t -> t.equals(term)).count();
        return (double) count / document.length;
    }

    public double idf(String term) {
        return Math.log((double) totalDocs / (1 + docFreq.getOrDefault(term, 0)));
    }

    public double tfIdf(String term, String[] document) {
        return tf(term, document) * idf(term);
    }
}
```

- Verification of Results
Comparison of Manual and Programmatic Results:
To ensure the accuracy of the programmatic TF-IDF calculations, the results from the program were compared with the manual calculations.

Example comparison:
- Manual Calculation: TF-IDF("fox", Document 1) = 0.119
- Programmatic Calculation: The same result should be obtained using the `TfIdfCalculator` class in Java.

6. System Testing and Evaluation
- Test Cases and Scenarios
Different Test Cases Used to Evaluate the System:
1. Basic Functionality Tests:
  - Query: "Honda 400 engine"
  - Expected Result: Retrieve documents containing information about Honda 400 engine maintenance, performance, or issues.

2. Edge Case Tests:
  - Query: ""

- Expected Result:** Handle empty queries gracefully without errors.
- Query: "nonexistentword"
- Expected Result: Return zero results indicating no matching documents.

3. Performance Tests:
- Query: "engine"
- Expected Result: Retrieve a large number of documents to test the system's performance under heavy load.

4. Relevance Tests:
- Query: "Honda maintenance tips"
- Expected Result:** Retrieve documents specifically related to maintenance tips for Honda motorcycles.

- Performance Evaluation
Metrics and Methods Used to Evaluate the System's Performance:
1. Response Time:
- Measure the time taken to retrieve and display search results.
- Tools: Apache JMeter, custom logging within the Java code.

2. Throughput:
- Measure the number of queries the system can handle per second.
- Tools: Load testing using Apache JMeter.

3. Resource Utilisation:
- Monitor CPU, memory, and disk usage during query processing.
- Tools: System monitoring tools.

Methodology:
- Execute a series of predefined queries.
- Record response times and throughput.
- Analyse resource utilisation during peak loads.

- Accuracy and Relevance of Results
Discussion on the Accuracy and Relevance of the Retrieved Documents:
Accuracy and relevance were evaluated using precision and recall metrics:
1. Precision:
- The ratio of relevant documents retrieved to the total documents retrieved.
- Formula:

$$\text{Precision} = \frac{\text{Number of Relevant Documents Retrieved}}{\text{Total Number of Documents Retrieved}}$$

2. Recall:
- The ratio of relevant documents retrieved to the total relevant documents in the dataset.
- Formula:

$$\text{Recall} = \frac{\text{Number of Relevant Documents Retrieved}}{\text{Total Number of Relevant Documents}}$$

Evaluation Process:

- Select a set of benchmark queries.
- Manually identify relevant documents for each query.
- Compare the system's results against the manually identified relevant documents.
- Calculate precision and recall.

Example Evaluation:
- Query: "Honda 400 engine"
  - Total Documents Retrieved: 50
  - Relevant Documents Retrieved: 45
  - Total Relevant Documents: 50
  - Precision: 45/50 = 0.90
  - Recall: 45/50 = 0.90

The results indicated high precision and recall, demonstrating the system's ability to accurately retrieve relevant documents.

7. Conclusion
- Summary of Work Done
    - This project involved the design, development, and evaluation of an information retrieval system for motorcycle forum data. Key tasks included:
        - Developing a web crawler to collect data from the target website.
        - Setting up Elasticsearch and Kibana for data indexing and visualisation.
        - Implementing a vector space model with TF-IDF representation and cosine similarity for document retrieval.
        - Creating visualisations in Kibana to analyse the indexed data.
        - Testing and evaluating the system's functionality, performance, accuracy, and relevance.

- Achievements
    - Successfully collected and indexed a large dataset from a motorcycle forum.
    - Implemented an information retrieval system using TF-IDF and cosine similarity.
    - Achieved high precision and recall in retrieving relevant documents.
    - Created insightful visualisations in Kibana to facilitate data analysis.
    - Developed and executed a comprehensive testing framework to evaluate system performance.

- Potential Enhancements
Suggestions for Enhancing the System in the Future:
1. Enhanced User Experience:
   - Develop a more interactive and user-friendly search interface with features like auto-complete, suggestions, and advanced filtering options.

2. Integration with Other Data Sources:
   - Expand the system to include data from other relevant sources, such as additional forums, social media, and news websites.

3. Scalability Improvements:
   - Optimise the system architecture to handle larger datasets and higher query volumes.

4. Machine Learning for Ranking:
   - Implement machine learning algorithms to improve document ranking based on user feedback and interaction data.