COMP4651 2024 Spring – Project report

Serverless Web Application for Large-Scale Text Crawling and Analysis

Group 29

CHAU, Wang Yik (20762496, wychauae), CHOI, Sheung Yin (20767020, sychoiaf),
LEUNG, Ka Wa (20770807, BossscoLeung), ORTEGA, Frederic Michel (20721351, Ferayddi)

1. Introduction/Background

Our group has developed a web application that aims to provide a way of dataset collection! Through the web application, the user can either upload their own raw data, which is a text file, or they can crawl reddit or google using a search query.

Then, they can access these raw datasets, view them, delete them, or select them to perform any of the three types of analysis:

- Word count analysis
- Sentiment analysis
- NER analysis

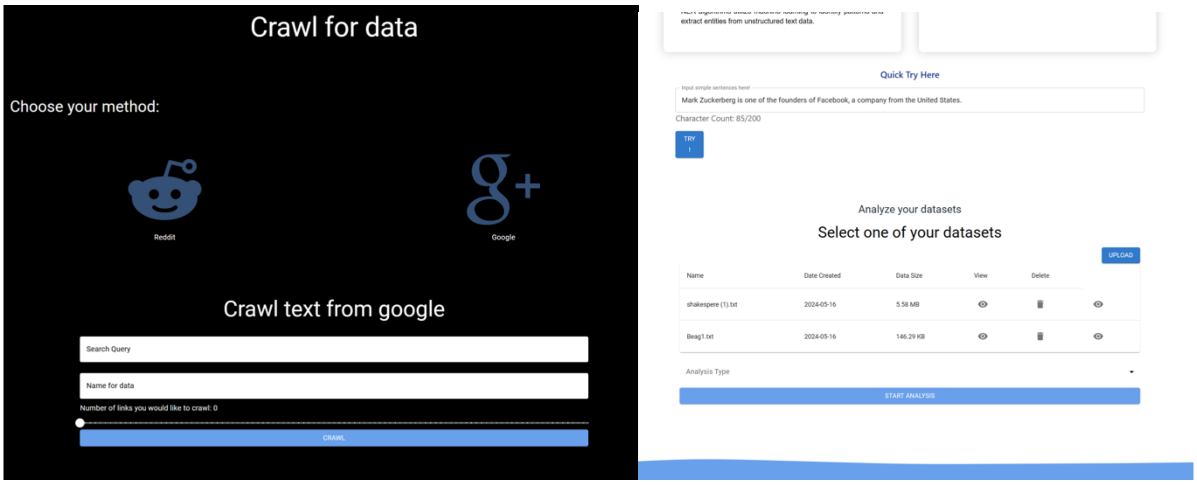
The result of those analysis is in text format, and is sent back in the server response, for the user to download. Spark has been used in the application's analysis step and uses techniques such as RDDs or data frames to parallelize tasks.

Our deliverable is in the form of docker images, which will leverage the power of cloud computing when deployed to Kubernetes, since scaling and management would be automated.

2. Implementation details

2.1. Front end

For the front end, we have used the React JS framework to develop the interface and have made use of the material UI library for certain components. The features of our web app are delivered mainly on two pages. The crawl page, where the user can crawl data from reddit or from google, using a query. And the data page, where the user can either select a data that has been crawled, or data that they upload.



Number of links you would like to crawl: 0

extract entities from unstructured text data.

Mark Zuckerberg is one of the founders of Facebook, a company from the United States.

Character Count: 85/200

Name	Date Created	Data Size	View	Delete	
shakespere (1).txt	2024-05-16	5.58 MB	👁	🗑	👁
Beag1.txt	2024-05-16	146.29 KB	👁	🗑	👁

For the front end to fetch or upload data to the backend, we created a services folder that takes care of sending axios requests and receiving the responses.

2.2 Back end

In our backend development, we rely on the express.js framework, which is built on top of node.js. It provides us with RESTful API endpoints that allow seamless communication between the frontend and backend. Through these endpoints, the frontend can interact with our database using various HTTP methods like POST and GET requests, enabling the retrieval, creation, updating, and deletion of data.

Moreover, the backend plays a crucial role in invoking Python programs dedicated to performing advanced analyses on user datasets. These programs encompass NER analysis, word count analysis, and sentiment analysis. Once the analyses are completed, the backend returns the results to the frontend. This seamless flow of data between the backend and frontend ensures efficient data processing and empowers users with valuable information derived from their datasets.

Furthermore, we employ Sequelize as our Object-Relational Mapping (ORM) tool for handling database operations. Sequelize simplifies the interaction with our database by providing a higher-level abstraction layer, allowing us to work with JavaScript objects rather than writing raw SQL queries. It offers a wide range of features, including defining data models, executing queries, performing CRUD operations (Create, Read, Update, Delete), establishing associations between tables, and handling migrations. With Sequelize, we can efficiently manage and manipulate data in our database, ensuring consistency and providing a convenient interface for interacting with our database through the backend application.

2.3 Database

We have selected PostgreSQL as our database. PostgreSQL is highly scalable and capable of efficiently handling large volumes of data, making it an excellent choice for our application's data storage and retrieval requirements. However, when it comes to user datasets, we refrain from storing them directly as a TEXT type in the database, despite PostgreSQL's ability to support the storage of extremely long character strings (up to approximately 1 GB). Storing datasets as TEXT directly in the database would result in increased data throughput between the backend server and the database, especially with large text datasets, potentially leading to performance degradation.

Instead, we adopt an alternative approach. In the database, we store only the URL of the dataset. To handle the datasets, we use multer in express.js, which allows us to directly store the datasets on our backend server. This approach effectively addresses the challenges associated with high data throughput and potential performance degradation.

2.4 Spark Data Analysis

In our web application, users can upload files or initiate web crawling to gather data. Once the data is collected, users have the flexibility to select from a variety of functionalities to analyze their crawl data. These functionalities are implemented using PySpark due to its ability to handle large volumes of data efficiently.

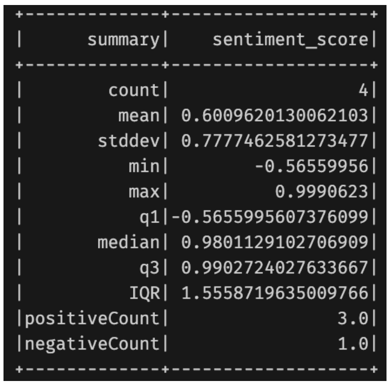
1. Word count analysis
   Word count analysis is a fundamental technique used to understand the frequency distribution of words within a corpus. In our application, users can perform word count analysis on their crawl data to gain insights into the most commonly occurring words. This analysis can be helpful for various purposes such as identifying key themes, topics, or trends within the data.

   Text preprocessing is a crucial step before conducting word count analysis. It involves cleaning and normalizing the text data to enhance the quality of analysis. In our approach, we employ *NLTK's PorterStemmer* to reduce words to their root form, thereby consolidating variations of the same word and improving the efficiency of subsequent analysis. Additionally, we utilize *PySpark's StopWordsRemover* to efficiently remove common stopwords from the tokenized text. This helps in reducing noise and focusing on the more meaningful content of the text. By integrating these preprocessing techniques, we ensure that the word count analysis accurately reflects the frequency distribution of meaningful words within the corpus, enabling users to gain valuable insights from their crawl data.

2. Sentiment analysis

   Sentiment analysis is the process of determining the sentiment or opinion expressed in a piece of text. In our application, users can perform sentiment analysis on their crawl data to understand the overall sentiment conveyed within the text. This analysis can be valuable for understanding public opinion, customer feedback, or social media sentiment, among other applications.

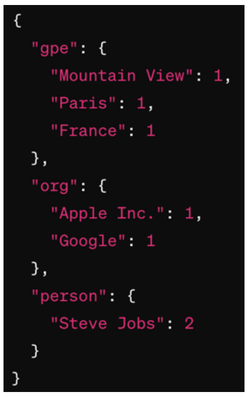
   We leverage a pre-trained Hugging Face model, *distilbert-base-uncased-finetuned-sst-2-english*, to assign sentiment scores to each crawl data, ranging from -1 (negative) to 1 (positive). After analyzing the sentiment of each crawl snippet, we compute various statistical measures to summarize the sentiment distribution across the crawl data. The final output includes statistics such as the count of analyzed texts, the mean, standard deviation, minimum, maximum, quartile ranges, and counts of positive and negative sentiments.



```
+-------------+-------------------+
|      summary|    sentiment_score|
+-------------+-------------------+
|        count|                  4|
|         mean| 0.6009620130062103|
|       stddev| 0.7777462581273477|
|          min|        -0.56559956|
|          max|          0.9990623|
|           q1|-0.5655995607376099|
|       median| 0.9801129102706909|
|           q3| 0.9902724027633667|
|          IQR| 1.5558719635009766|
|positiveCount|                3.0|
|negativeCount|                1.0|
+-------------+-------------------+
```

3. Named-entity recognition (NER)

   NER is a natural language processing (NLP) technique used to identify and classify named entities within a text. In our application, users can utilize NER to extract important entities from their crawl data, facilitating tasks such as identifying important entity in this batch of data or extract out the recent trend entity. We leverage the *SpaCy* library to extract named-entities from the text corpus obtained from the crawl data. After extracting named entities, we generate a count of each entity type. The count represents the frequency of occurrence of each entity type within the crawl data.



```
{
  "gpe": {
    "Mountain View": 1,
    "Paris": 1,
    "France": 1
  },
  "org": {
    "Apple Inc.": 1,
    "Google": 1
  },
  "person": {
    "Steve Jobs": 2
  }
}
```

3. Docker image building

As our web application consists of two main components: the frontend, and backend (Server + Database). To encapsulate the entire project, we have created 2 Docker files and used 3 images. For the front end, which is developed using React.js, we use the official ***node:20.12.2-buster*** image. Similarly, for the backend, we use the ***node:20.12.2-buster*** image. However, since our backend also requires running Python programs, we install the necessary Python development environment using the official ***python:3.9-buster image*** as well. In addition to Python, we incorporate Java development into our backend. For the database, we leverage the official PostgreSQL image named "***postgres***". To orchestrate the entire containerized environment, we have written a ***docker-compose.yml*** file. This file enables us to build and configure the complete container setup for our application, ensuring seamless integration of the frontend, backend, and database components.

4. Results

Overall, our team was able to achieve the goals that were set, and produced a convenient web application that makes use of Artificial intelligence and cloud computing. Our application provides two main sets of features. The first set of features involve crawling. The crawling can either be done on Reddit or Google and uses a search query to obtain data. Once the crawling data has been collected, the application cleans the data and stores it inside a text file, with a corresponding DB entry that has a reference to the text file path.

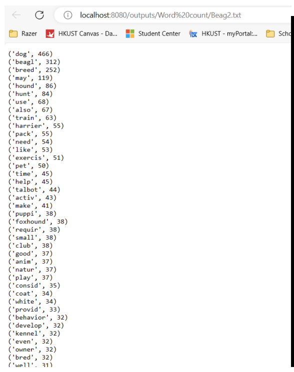
The user can then view this generated raw data file in a table. This table provides functional features to either view the full text file or delete the text file. If the user wants to, he can also upload his own text file with already collected text data.

The second set of features involves analyzing these files, which use spark and data frames. The user can choose a text file from the table, and execute one of three analysis functions, namely:

- Sentiment analysis
- NER analysis
- Word count analysis

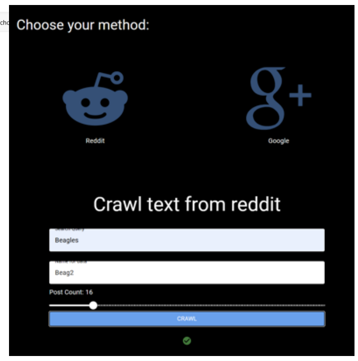
Note that the first two analyses also use AI models.

Once the analysis is launched, the website will display a loading icon and will wait for the server to send back a response. The server will execute the corresponding python scripts, based on the analysis desired. Once completed, the server will send back a successful response that will contain the result text file in the attachment. The user can therefore download the result text file upon completion.



```
('dog', 466)
('beagl', 312)
('breed', 252)
('may', 119)
('hound', 86)
('hunt', 84)
('use', 68)
('also', 67)
('train', 63)
('harrier', 55)
('pack', 55)
('need', 54)
('like', 53)
('exercis', 51)
('pet', 50)
('time', 45)
('help', 45)
('talbot', 44)
('activ', 43)
('make', 41)
('puppi', 38)
('foxhound', 38)
('requir', 38)
('small', 38)
('club', 38)
('good', 37)
('anim', 37)
('natur', 37)
('play', 37)
('consid', 35)
('coat', 34)
('white', 34)
('provid', 33)
('behavior', 32)
('develop', 32)
('kennel', 32)
('even', 32)
('owner', 32)
('bred', 32)
('well', 31)
```

Word count result txt

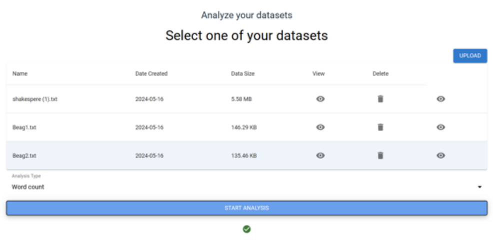


Beagles

Beag2

Post Count: 16

Crawling from



Analyze your datasets

Select one of your datasets

Name	Date Created	Data Size	View	Delete	
shakespere (1).txt	2024-05-16	5.58 MB	👁	🗑	👁
Beag1.txt	2024-05-16	146.29 KB	👁	🗑	👁
Beag2.txt	2024-05-16	135.46 KB	👁	🗑	👁

Analysis Type
Word count

Selecting dataset and launching analysis