

Programming Assignment 05 : Search Result Clustering

Authors : shashank.sinha@st.ovgu.de , vikas.kale@st.ovgu.de , hrushikesh.ahire@st.ovgu.de , rohit.rakesh@st.ovgu.de

Project Documentation: Search Result Clustering

Introduction

The "Search Result Clustering" project focuses on implementing a clustering algorithm to partition data points based on similarity. Clustering is an unsupervised machine-learning technique used to identify patterns in data without a predetermined target. This project specifically applies clustering to group similar documents, which is a common application in data analysis and machine learning.

Project Overview

The project's main objective is to create a cluster-based visualization of search results. Unlike traditional list-based result visualizations, this approach emphasizes the relationships and similarities between different search results. The project can be utilized in various scenarios where understanding the connection between documents is essential.

Technology Stack

- Programming Languages: Python, Java
- Frameworks and Libraries: Lucene (Pre-processing), Django (for backend), React JS (for frontend), Python's Sklearn library for machine learning algorithms.
- Algorithms: K-Means.

Step-by-Step Setup Instructions

This step involves preparing our computer to run the project by installing the necessary software and dependencies.

Install Python and Java:

- Python is essential for running the backend and various scripts.
- Java is required for using Lucene.

Install Django:

- Django is a high-level Python web framework that encourages rapid development and clean, pragmatic design.
- It will be used for the backend server.

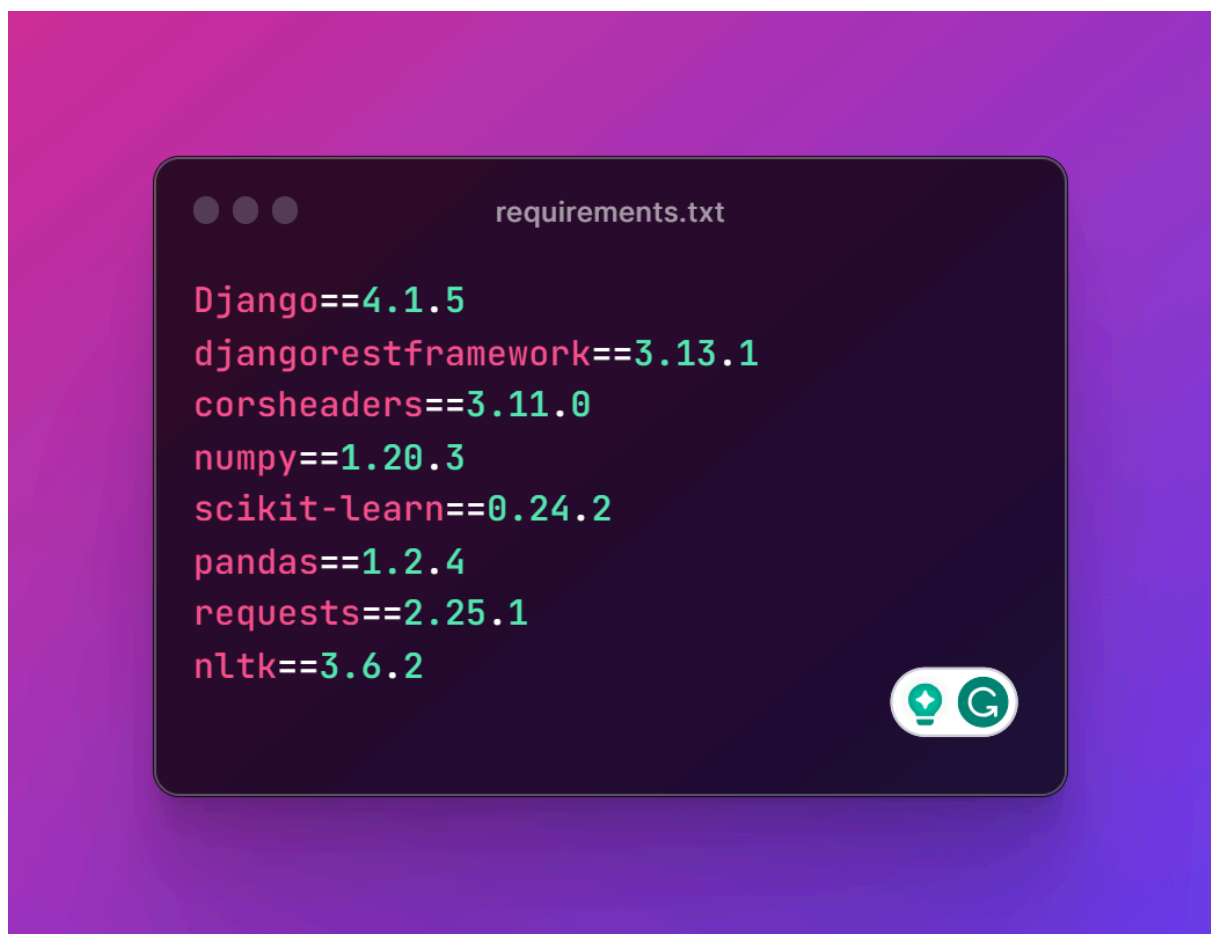
Install React JS:

- React JS is a JavaScript library for building user interfaces, primarily for single-page applications.
- It's used for the front-end development of the project.

Install Sklearn:

- Sklearn is a Python library for machine learning, providing simple and efficient tools for data analysis and modeling.

It can be a bit hectic and tedious to keep track of the same so the best way to deal with these installations and make sure that we do not miss out on any essential imports the lack of which could cripple the project is by creating a requirement.txt file and run the same when we use / setup the code for the first time, while not all but this could help make sure that most of the requirements to run the project are met.




Keep in mind the imports need to match what the device can support and this image is to provide an example

This solves most of our requirements for the backend, but since our frontend is react-based we need to install the dependencies with npm install as follows :

```
package.json

{
  "dependencies": {
    "@fortawesome/fontawesome-svg-core": "^6.5.1",
    "@fortawesome/free-brands-svg-icons": "^6.5.1",
    "@fortawesome/free-regular-svg-icons": "^6.5.1",
    "@fortawesome/free-solid-svg-icons": "^6.5.1",
    "@fortawesome/react-fontawesome": "^0.2.0",
    "@testing-library/jest-dom": "^5.17.0",
    "@testing-library/react": "^13.4.0",
    "@testing-library/user-event": "^13.5.0",
    "axios": "^1.6.5",
    "d3": "^7.8.5",
    "react": "^18.2.0",
    "react-dom": "^18.2.0",
    "react-router": "^6.21.1",
    "react-router-dom": "^6.21.1",
    "react-scripts": "5.0.1",
    "react-scroll": "^1.9.0",
    "web-vitals": "^2.1.4"
  }
}

# to install this you can use npm install for these imports
```



Also, we need to make sure Lucene is properly added (installed and imported) to the project as it is an integral part of the backend.

How does the project work?

I know I know, we have satisfied the requirements now let us move on to how the project works. We will begin where all user-based applications start, the frontend.

Our frontend is based heavily on React JS which is a JavaScript library for building user interfaces, it is user-friendly and quite easy to understand, we also set up our Django server to make sure that our query can be passed to our backend where the real magic happens

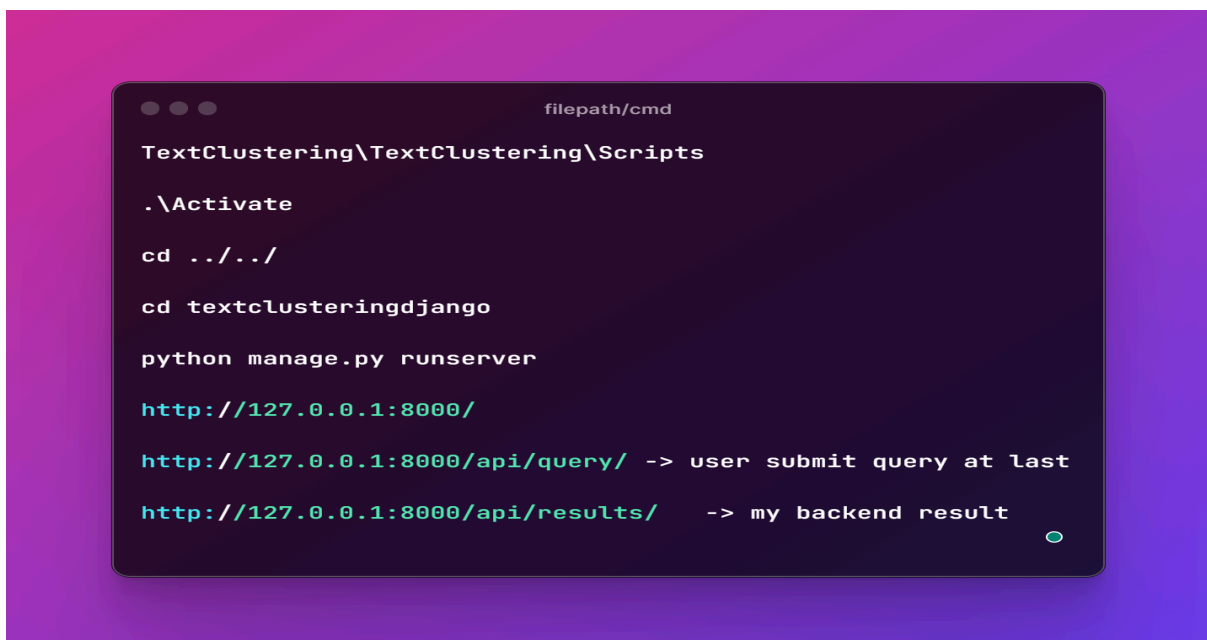
Launching the frontend

Navigate to the folder where we have cloned our file and go over to the **Text-Clustering** folder there we will find two folders named

- **textclusteringdjango**
- **textclusteringreact**

We set up and launched our servers using the following code.

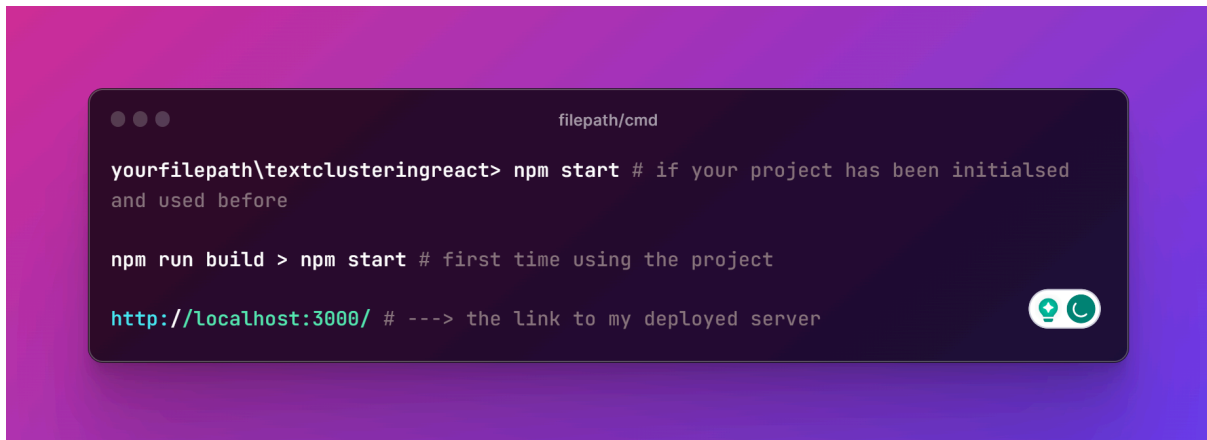
- **textclusteringdjango**

A terminal window with a dark background and light-colored text. The title bar at the top reads 'filepath/cmd'. The terminal content shows the following commands and output:

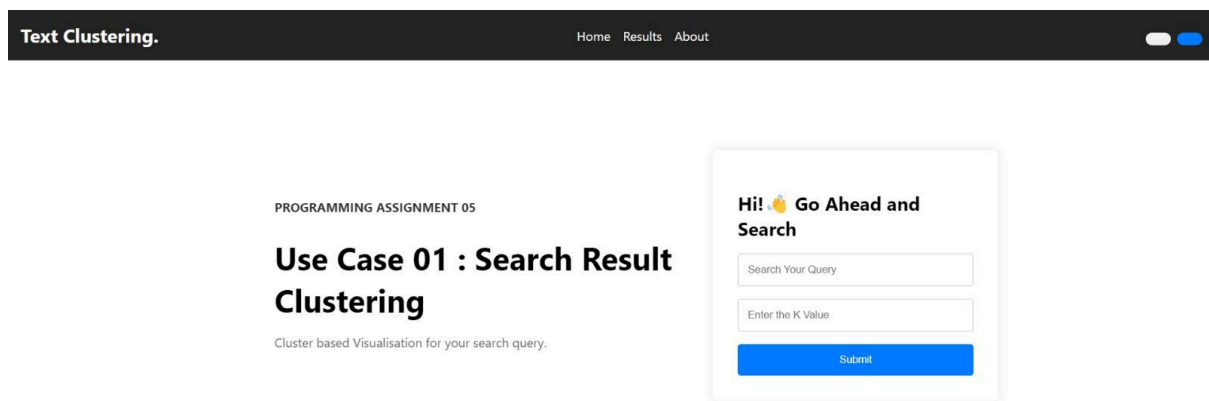
```
TextClustering\TextClustering\Scripts
.\Activate
cd ../../
cd textclusteringdjango
python manage.py runserver
http://127.0.0.1:8000/
http://127.0.0.1:8000/api/query/ -> user submit query at last
http://127.0.0.1:8000/api/results/ -> my backend result
```

A small green cursor is visible at the end of the last line.

- textclusteringreact



Setting up the server gives us these three screens



It gives us the home screen where we have a form where the user can input the query they want to search and provide the number of clusters they want

PROGRAMMING ASSIGNMENT 05

Use Case 01 : Search Result Clustering

Cluster based Visualisation for your search query.

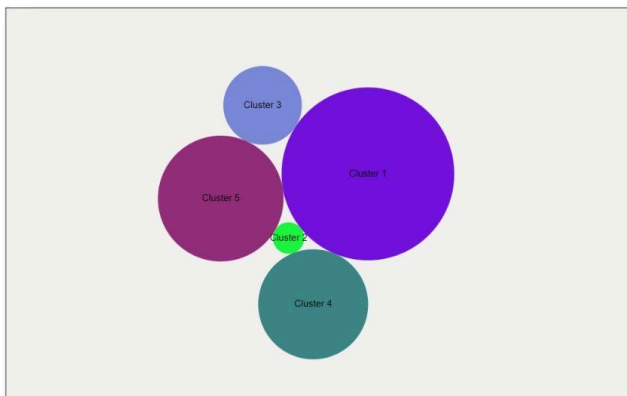
Hi! 🌟 Go Ahead and Search

Hello Germany

8

Submit

After the user enters the required values , they can then submit the values by using the submit button which then triggers the backend to search for instances of the query in the database.



Clustering Result

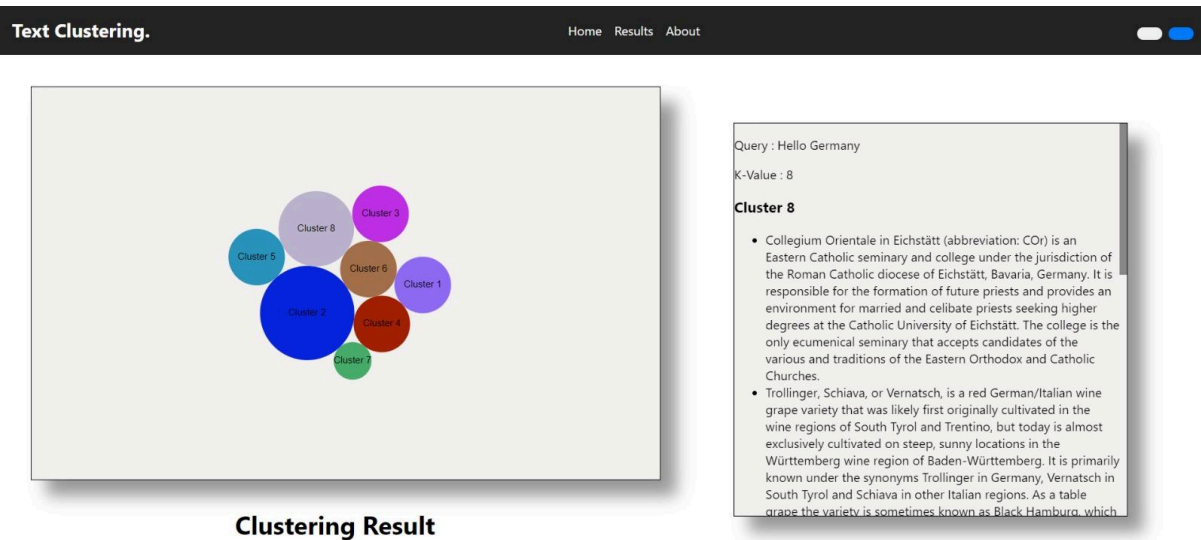
Query : How are you

K-Value : 5

Click on the Clusters to find more

This is the result page, it is divided into 2 parts

- One part shows a visualization where we can see the clustering while the other is a small window to provide more information. It also shows us the query and k value entered by the user




On clicking on a cluster , we can see the content inside that cluster in the window on the right. Also the size of the cluster depends on the amount of content inside it i.e the cluster with the most documents will be the largest while the one with the least documents will be the smallest.


Text Clustering. Home Results About

OUR TEAM


An incredible team of amazing individuals




Vikas Gajanan Kale
Frontend Development And Integration
[Twitter](#) [LinkedIn](#)



Shashank Sinha
Backend Development and Architecture
[Twitter](#) [LinkedIn](#)



Rohit Rakesh
Frontend Development and Design
[Twitter](#) [LinkedIn](#)



Hrushikesh Ahire
Backend Development and Architecture
[Twitter](#) [LinkedIn](#)

Just a page about our awesome team.

Backend: The success of any project is dependent on it having a sturdy backend, Here is how we go about it

1. Fetching Documents

In this step, we use the MediaWiki API to fetch approximately 2000 Wikipedia documents. The MediaWiki API is a web service that allows access to Wikipedia data, such as articles, in a programmable way.

2. Preprocessing (using Lucene)

For preprocessing, we are utilizing a custom analyzer, within a Lucene environment. Lucene is a high-performance, full-featured text search engine library. Key steps include:

- Standard Tokenization: Breaking down text into individual terms or words.
- Lowercase Filtering: Converting all tokens to lowercase to ensure uniformity, reducing the complexity of the search.
- Stopwords Removal: Eliminating common words (like 'and', 'the', etc.) that do not add significant meaning to the text.
- Stemming using Porter Stemmer: Reducing words to their root form to improve the search process. Porter Stemmer is a widely used stemming algorithm.

3. Fetching Relevant Documents (using Lucene)

This step involves indexing and searching documents using Lucene:

- Indexing with IndexWriter and BM25Similarity: Each document is indexed using Lucene's **IndexWriter**, and we are applying the BM25Similarity algorithm for indexing. BM25 is a bag-of-words retrieval function that ranks a set of documents based on the query terms appearing in each document.
- Query Parsing and Searching: Using **QueryParser** and **IndexSearcher** to search the indexed documents. The **QueryParser** parses the user's search query into a format suitable for searching within the Lucene index, and the **IndexSearcher** conducts the search.

4. Clustering (scikit-learn)

After retrieving relevant documents, we are using clustering for organization:

- Vectorization with Tf-IdfVectorizer: This step involves converting the text data of the documents into numerical vectors using TF-IDF (Term Frequency-Inverse Document Frequency). This is crucial for the clustering algorithm to process the textual data.

- Clustering with KMeans: The numerical vectors are then clustered using the KMeans algorithm from the **scikit-learn** library. KMeans is an unsupervised learning algorithm that groups data into k number of clusters.
- Assigning Cluster Labels: Each document is assigned a label corresponding to the cluster it belongs to.
- Result Preparation: The final output includes a mapping of cluster labels to the list of relevant documents in each cluster. This mapping is converted to JSON format to be sent to the frontend.