Automated Tagger of Articles

Big Data Project

Ayushi Gupta(ag7335)
Moenuddeen Ahmad Shaik(ms10415)

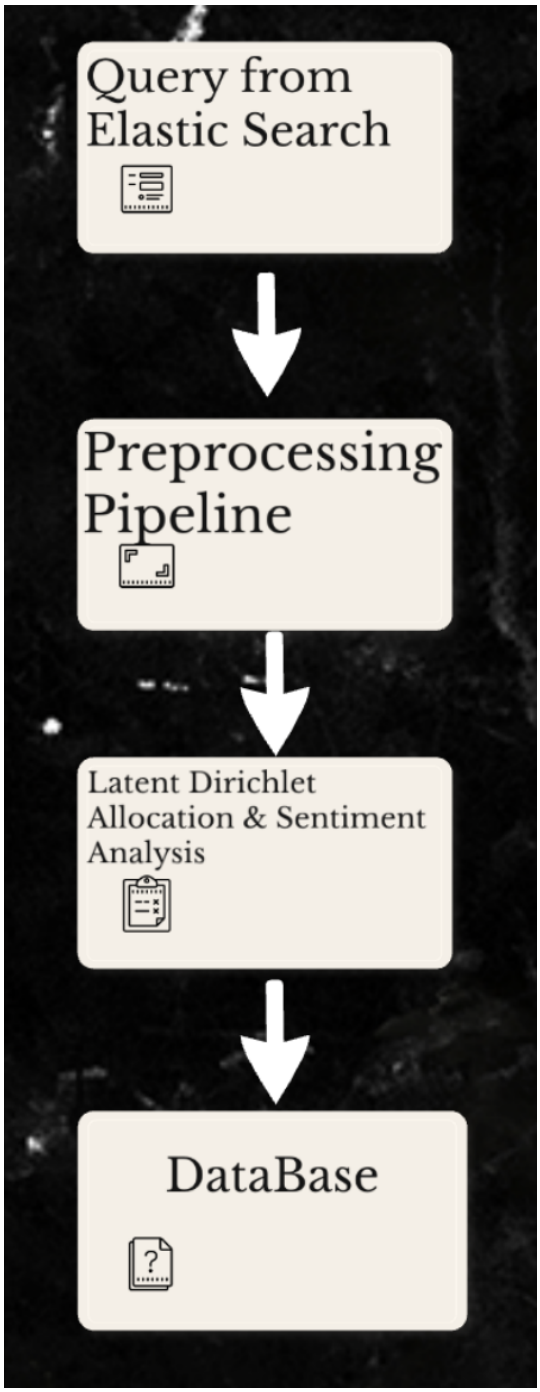
Introduction

The goal of this project is to generate the tags for the different kinds of blogs or articles. Users can specify the data and the query terms to search for the related blogs. All the related blogs will then be dumped into a Kafka queue which will process the streaming data and generate the associated tags with each blog Users can go to the specific blogs according to their choice. It will also include a pre-processing i.e, word2vec and TF-IDF. After the preprocessing the blogs will be tagged using topic modeling- Latent Dirichlet Allocation(LDA) in pySpark. To minimize the storage requirement for the whole system we can purge the resulting data after publishing the results to the user. We will be using pySpark, Apache Kafka and python which will be used to preprocess the massive data.

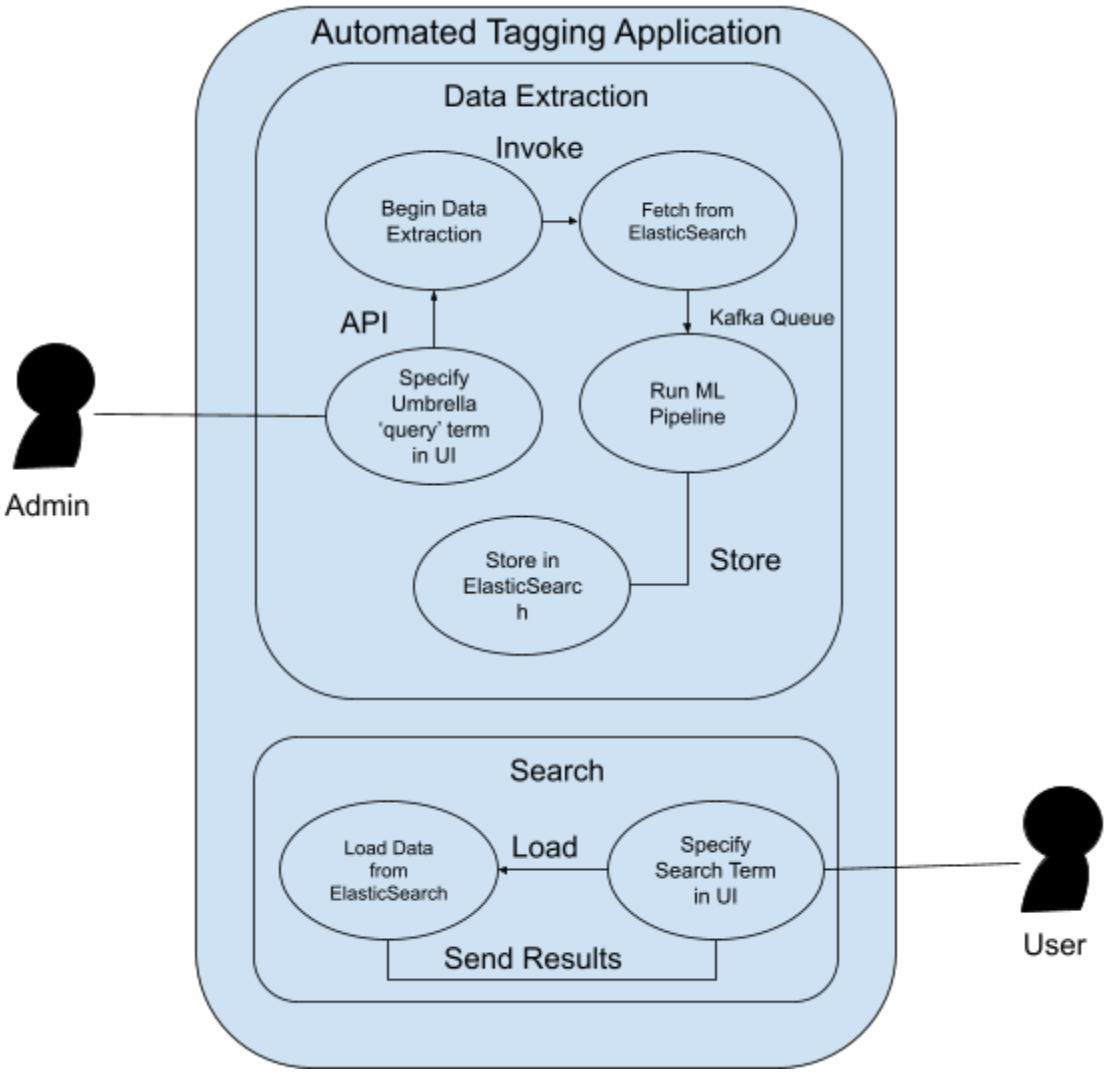
Presentation Link

https://prezi.com/view/TbrLxY0W4cQz8xbD7bUG/

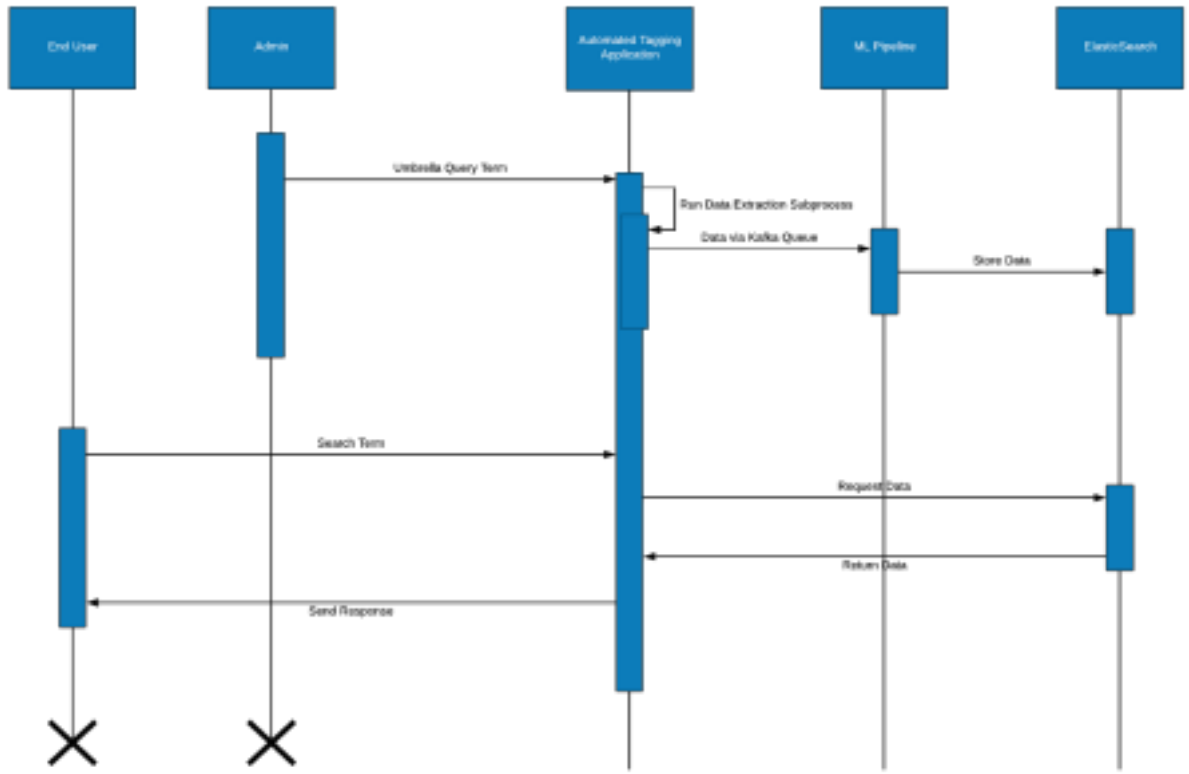
System Architecture Diagram



Use Case Diagram



Sequence Diagram



System Overview

The UI will allow the user to specify an umbrella query term. This query term will be used to gather the relevant documents by querying an elastic search index. All the data collected using elastic search will then be dumped into a Kafka queue, which helps in processing the streaming data and managing huge workloads due to the high amount of the relevant articles. The ML pipeline process consumes data from the Kafka queue. Incoming documents are subject to the cleaning of data for further analysis. It then goes through the core of the system for automated

tagging. This is achieved by using Latent Dirichlet Allocation (LDA) to generate the tags relevant to each article. We also perform sentiment analysis in each article using NLTK. The generated tags, along with the sentiment, are then dumped into another ElasticSearch index. The UI will fetch the results from the ElasticSearch index and publish them.

Now the user can perform searches from the search screen present in the UI and will be presented with the tags relevant to each article. The system also provides a link to every article in case the user wants to read the article.

Technical Components

Elastic Search

Elasticsearch is the distributed search and analytics engine at the heart of the Elastic Stack. Logstash and Beats facilitate collecting, aggregating, and enriching your data and storing it in Elasticsearch. Kibana enables you to interactively explore, visualize, and share insights into your data and manage and monitor the stack. Elasticsearch is where the indexing, search, and analysis magic happens.
Elasticsearch provides near real-time search and analytics for all types of data. Whether you have structured or unstructured text, numerical data, or geospatial data, Elasticsearch can efficiently store and index it in a way that supports fast searches. You can go far beyond simple data retrieval and aggregate information to discover trends and patterns in your data. And as your data and query volume grows, the distributed nature of Elasticsearch enables your deployment to grow seamlessly right along with it.

Kafka

Apache Kafka is a community distributed event streaming platform capable of handling trillions of events a day. Initially conceived as a messaging queue, Kafka is based on an abstraction of a distributed commit log. Since being created and open sourced by LinkedIn in 2011, Kafka has quickly evolved from messaging queue to a full-fledged event streaming platform.

Founded by the original developers of Apache Kafka, Confluent delivers the most complete distribution of Kafka with Confluent Platform. Confluent Platform improves Kafka with additional community and commercial features designed to enhance the streaming experience of both operators and developers in production, at massive scale.

PySpark

PySpark is an interface for Apache Spark in Python. It not only allows you to write Spark applications using Python APIs, but also provides the PySpark shell for interactively analyzing your data in a distributed environment. PySpark supports most of Spark's features such as Spark SQL, DataFrame, Streaming, MLlib (Machine Learning) and Spark Core.

React

We wanted our system to be platform-independent and enable the user to access the system from a PC, Mac, or even a smartphone. Our frontend UI is, therefore, built using React. React helps in efficiently updating and rendering the right components of our UI when the data changes. It helps the system deliver beautiful visualizations of the data being processed, enabling the user to make more insightful decisions and ease his process of document discovery and understanding.

Flask

Flask is a lightweight WSGI web application framework. It is designed to make getting started quick and easy, with the ability to scale up to complex applications. It began as a simple wrapper around Werkzeug and Jinja and has become one of the most popular Python web application frameworks.

Flask offers suggestions, but doesn't enforce any dependencies or project layout. It is up to the developer to choose the tools and libraries they want to use. There are many extensions provided by the community that make adding new functionality easy.

Deployment

Prerequisites: Please run pip install requirements.txt

1. Python >= 3.6
2. Spark
3. Kafka
4. ElasticSearch
5. NPM or React for UI deployment

Setup:

0. Start the zookeeper, kafka and elastic search services using the following commands

zookeeper-server-start /usr/local/etc/kafka/zookeeper.properties

kafka-server-start /usr/local/etc/kafka/server.properties

elasticsearch

1. Extract the project zip
2. CD back into the project directory and run the following python scripts as shown below

python3 backend.controller.ApiController.py

python3 backend.ldapipeline.backendEngine.py

3. To load the dataset into elasticsearch, run `*python -m backend.dataHandler.initES*`
4. Finally, cd into the UI directory and Start the react app by running the command "*npm start*". Then access the UI on the browser at *http://localhost:3000/*

Usage

Data Extraction:

1. Click 'Extract' on the navbar.

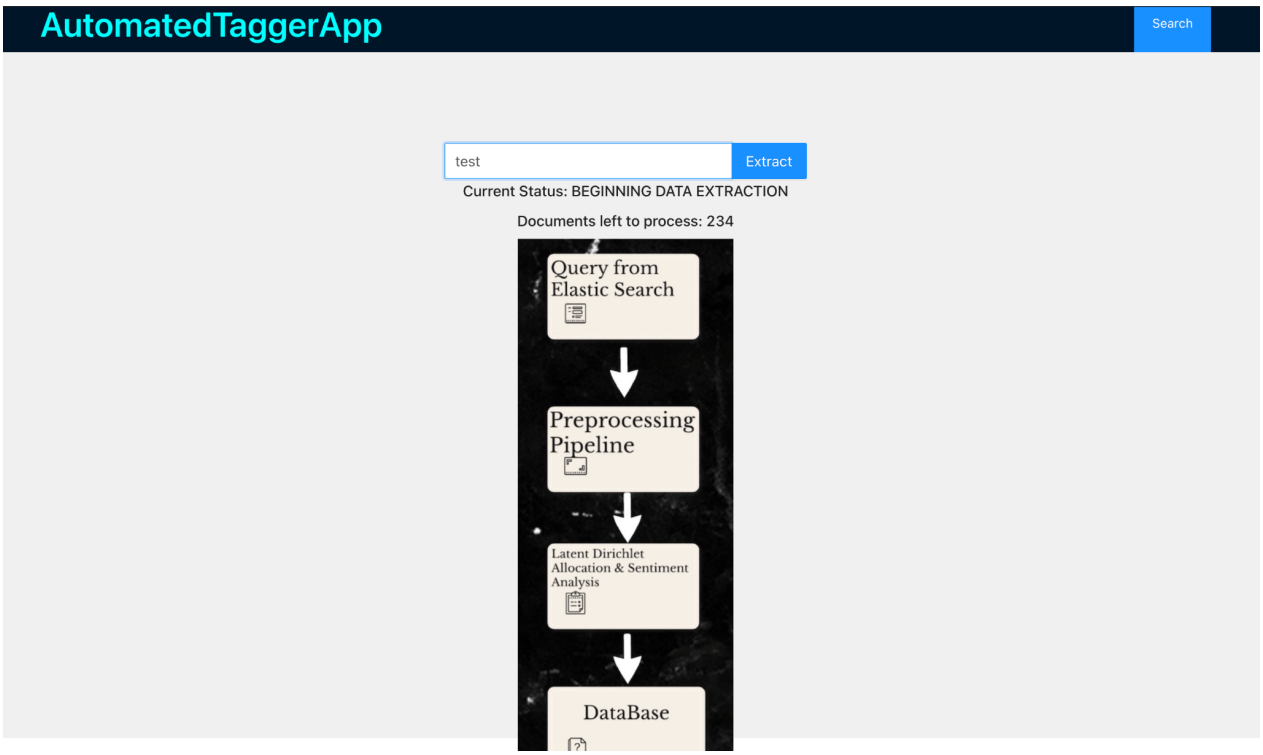
2. Enter the query term and click the ‘Extract’ button.
3. The API will fetch the raw data from the ElasticSearch database. Then run it through the LDA pipeline, and store the processed data back into another index on ElasticSearch.

Search:

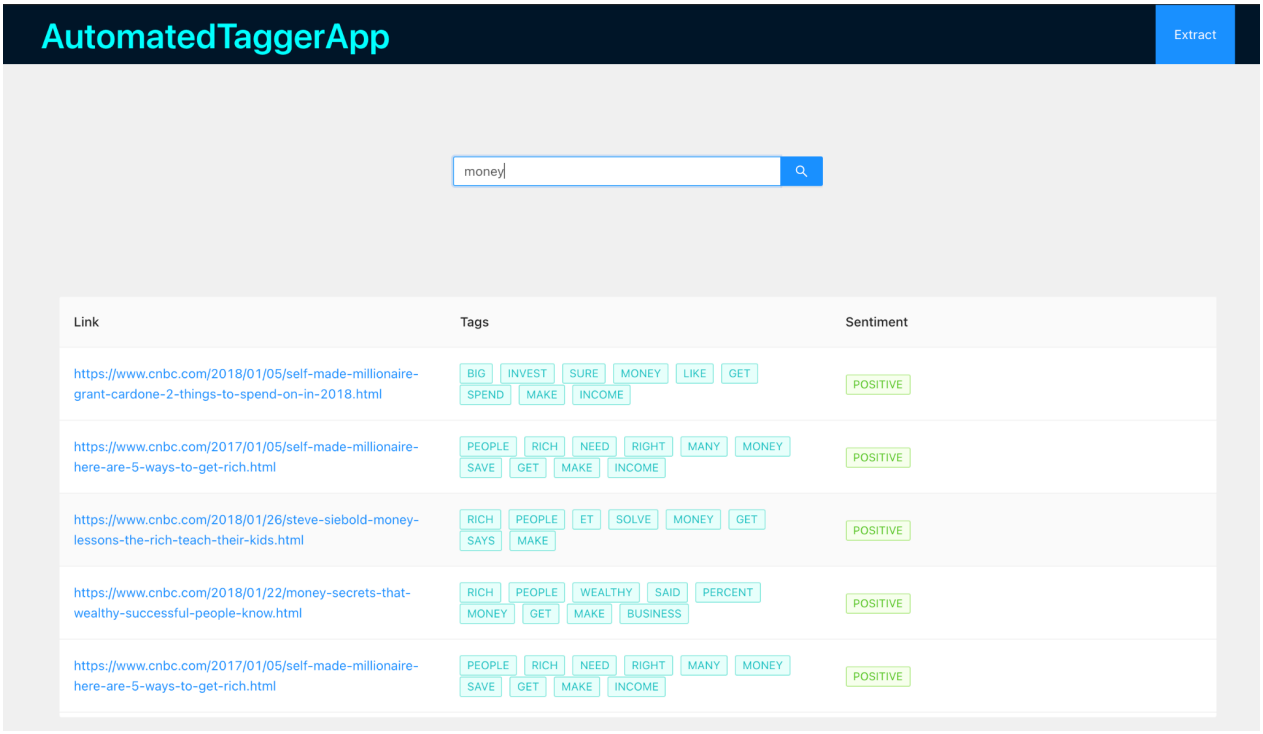
1. Go to the ‘Search’ page.
2. Enter the search query and click the search button.
3. The results will appear in the table.

Sample Screenshots

Extracting Data Screen



Search results screen



Reference

1. What is Elasticsearch? | Elasticsearch Guide [7.14]
2. https://www.confluent.io/what-is-apache-kafka/?utm_medium=sem&utm_source=google&utm_campaign=ch.sem_br.nonbrand_tp.prs_tgt.kafka_mt.xct_rgn.namer_lng.eng_dv.all_con.kafka-general&utm_term=kafka&creative=&device=c&placement=&gclid=EAIaIQobChMIx8XWm8XS8gIV4SCtBh0OoAhfEAAYASAAEgKWm_D_BwE
3. https://www.kaggle.com/jeet2016/us-financial-news-articles/version/1
4. PySpark Documentation — PySpark 3.1.2 documentation