Project Report

Automated Tagging of Articles

Nikita Jayakar (nmj303)

Ghanashyam V Tatti (gvt217)

Rochak Agrawal (ra3117)

Spring 2020

Introduction

Document tagging is a technique of adding extra information to documents. The additional information is usually a set of keywords that summarize the whole document. A reader can use those keywords to get an idea about what the document is about.

Document tagging can be further used to:

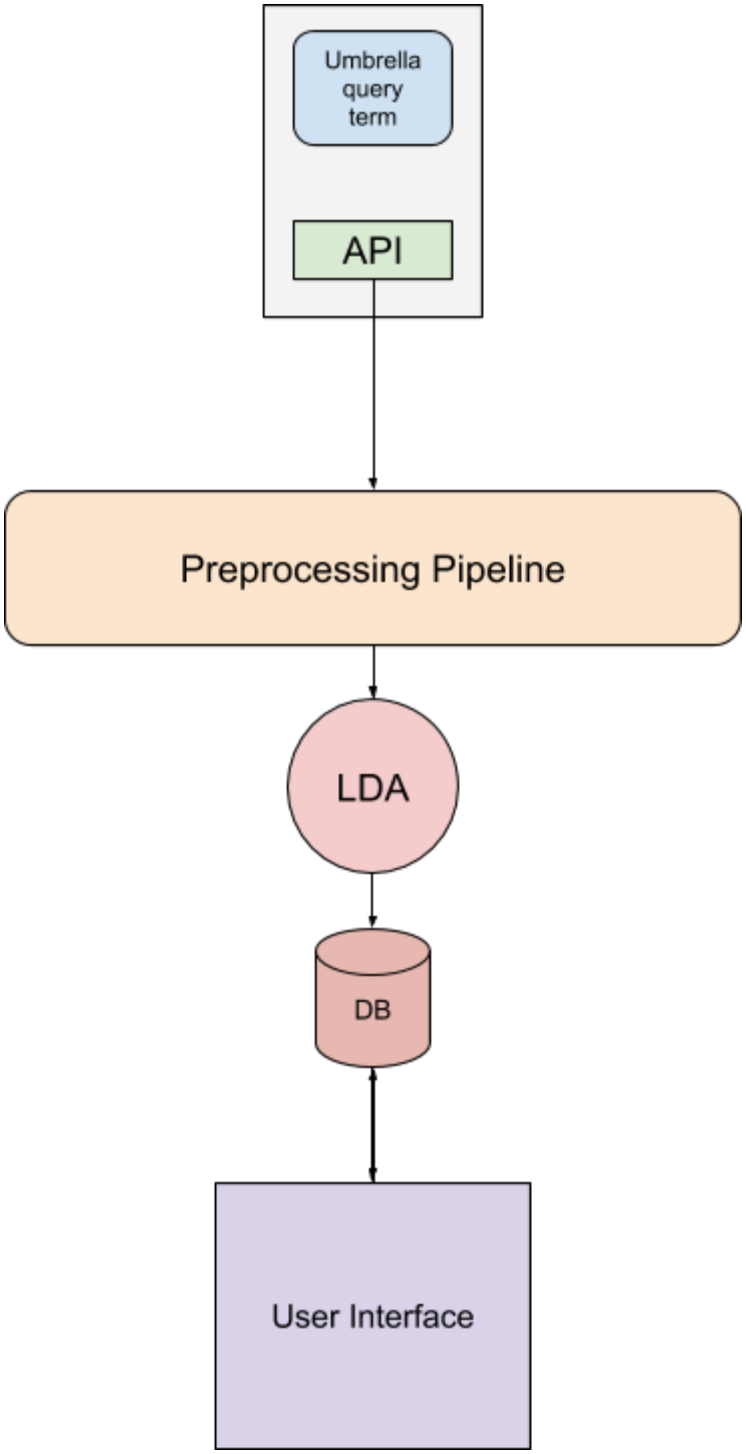
1. Store the result of indexing steps for later retrieval.
2. Build interactive applications on top of data stored in the Intelligent Data Operating Layer (IDOL).
3. Use the tag data to analyze the content and decide whether to index it at all.
4. Filter a large number of documents to obtain a small subset of relevant documents.

Over the years, many different approaches have been developed for automated tagging of documents on a small scale. However, there is a constant influx (of high velocity and veracity) of news articles and blog posts. For a blogger or a blog hosting service to manually tag it would be time-consuming and difficult. This project aims to generate tags for news articles and blog posts automatically.

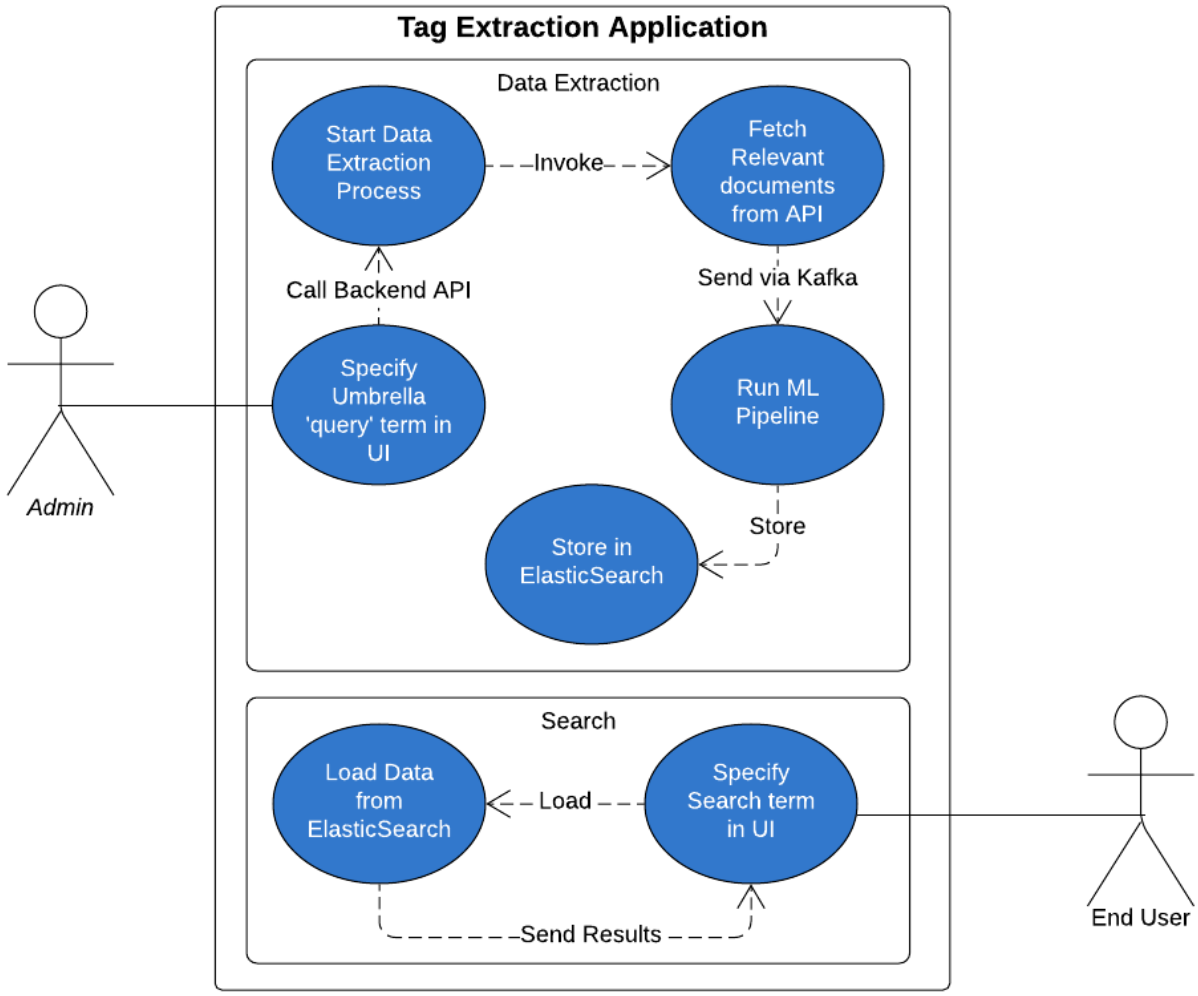
We present a simple UI to the user where they enter a query term. The system gathers articles that contain the query term and then performs automated tagging of those documents. The generated tags will then be accessible by the user in the UI, which also allows them to go to a specific article.

The system has been developed while keeping the large throughput in mind and is resilient to a massive amount of data input. It is achieved by using various big data technologies that will be explained later.

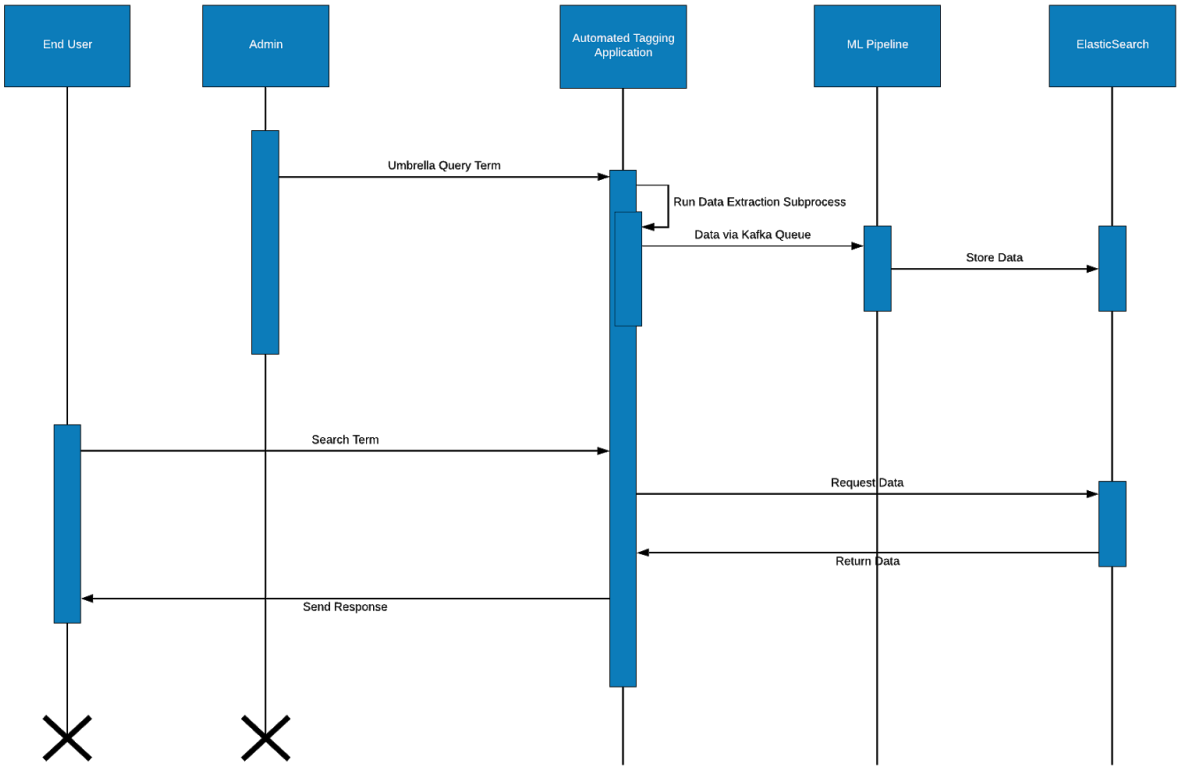
System Architecture Diagram



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 a search engine based on the Lucene library. It provides a distributed, multitenant-capable full-text search engine with an HTTP web interface and schema-free JSON documents. Our original motive was to use various news/article APIs to gather real-time data. However, due to their pricing, we looked into different avenues that will allow us to simulate the working of APIs, and we, therefore, selected elastic search. ElasticSearch exposes a simple REST API that enables the system to issue GET requests similar to what different APIs would offer. For a production-level system, the elastic search can be interchangeably used with various APIs or a combination of both to achieve maximum performance depending on the different use-cases.

Our current system has 87,000 articles that have been indexed in an elastic search. These articles belong to different categories and have varying lengths. This has helped us in simulating a real-world scenario where various articles might contain the query term.

Kafka

Apache Kafka is an open-source stream-processing software platform that aims to provide a unified, high-throughput, low-latency platform for handling real-time data feeds. Since the number of relevant documents matching the user search query can be highly varied, Kafka helps our system to maintain a streamlined flow of documents and provide high-throughput at the same time. It allows multiple producers and consumers, which help the system to scale horizontally as the load increases or if other text analysis functionalities are added at a later point in time. Since the raw input to all the other functionalities will still be the unprocessed documents, the idea of multiple producers and consumers fits best for our system.

We currently use it to dump relevant documents from elastic search to ML pipeline.

PySpark

Processing a large amount of textual data without employing parallelism or cluster computing would take hours, if not weeks. Apache Spark is an open-source distributed general-purpose cluster-computing framework with implicit fault tolerance and data parallelism. PySpark provides a python API to utilize Spark. Our system requires processing hundreds of documents for a single user query, and PySpark helps us achieve near real-time processing that enhances the user experience. Moreover, in a production environment, with thousands of users, Spark will be easy to scale to manage high workloads.

Our Machine Learning pipeline runs on Spark backend. The whole pipeline consists of data preprocessing, data cleaning, data processing, all of which run on Spark to provide high-performance analysis of the documents. Apart from that, the core of our system, Local Dirichlet Allocation (LDA), is also readily available in Spark.

React

We wanted that our system should 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 an easy to use web framework. We use it to serve our backend services such as ElasticSearch and the ML Pipeline to the UI through RESTFul APIs.

Deployment

Prerequisites:

1. Python >= 3.6
2. Spark
3. Docker

Setup:

1. Extract the project zip
2. CD into scripts and run `docker-compose up -d`
3. CD back into the project directory and run start.sh/start.cmd
4. To load the dataset into elasticsearch, run `python -m dataextraction.initES`
5. Finally, open http://localhost:5000/index.html on the browser to access the UI.

Usage

Data Extraction:

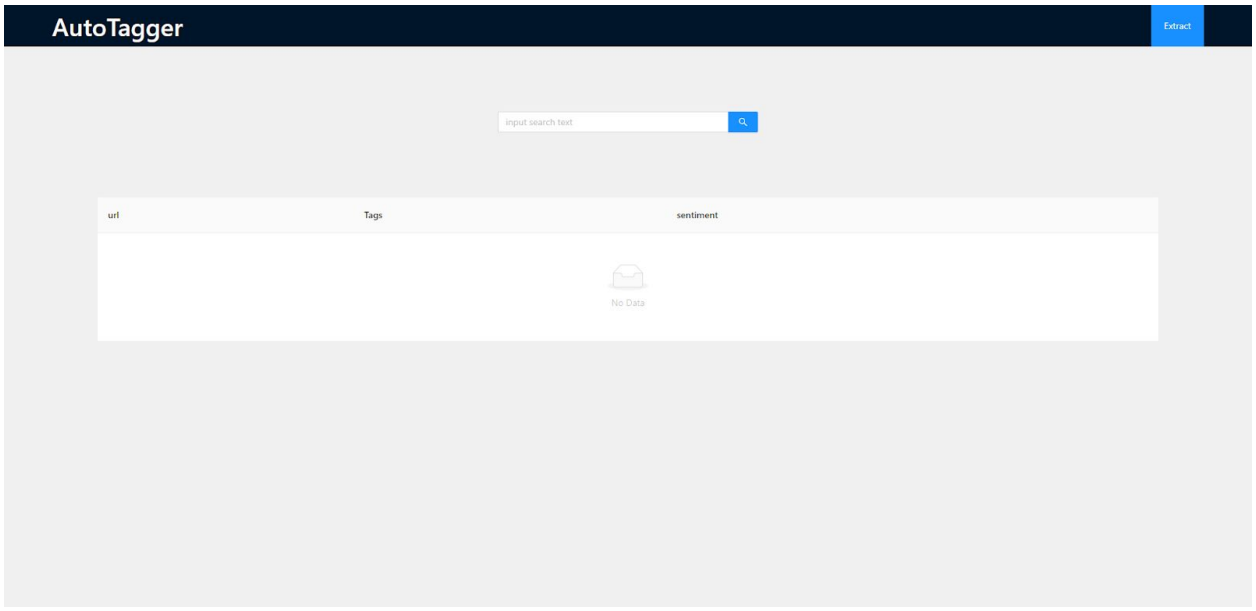
1. Click ‘Extract’ on the navbar.
2. Enter the umbrella term and click the ‘Extract’ button.
3. The process will fetch the raw data from ElasticSearch, run it through the ML pipeline, and store the processed data back into another index on ElasticSearch.

Search:

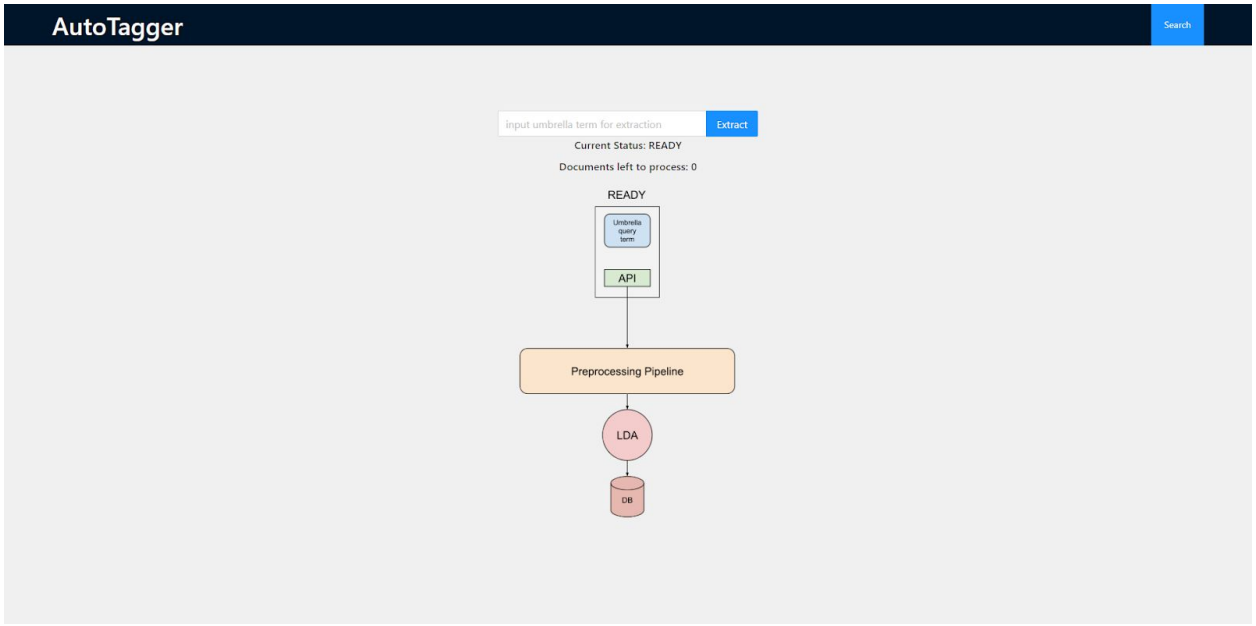
1. Go to the ‘Search’ page.
2. Enter the search query and click the search button.
3. The results will appear in the table.

Sample Screenshots

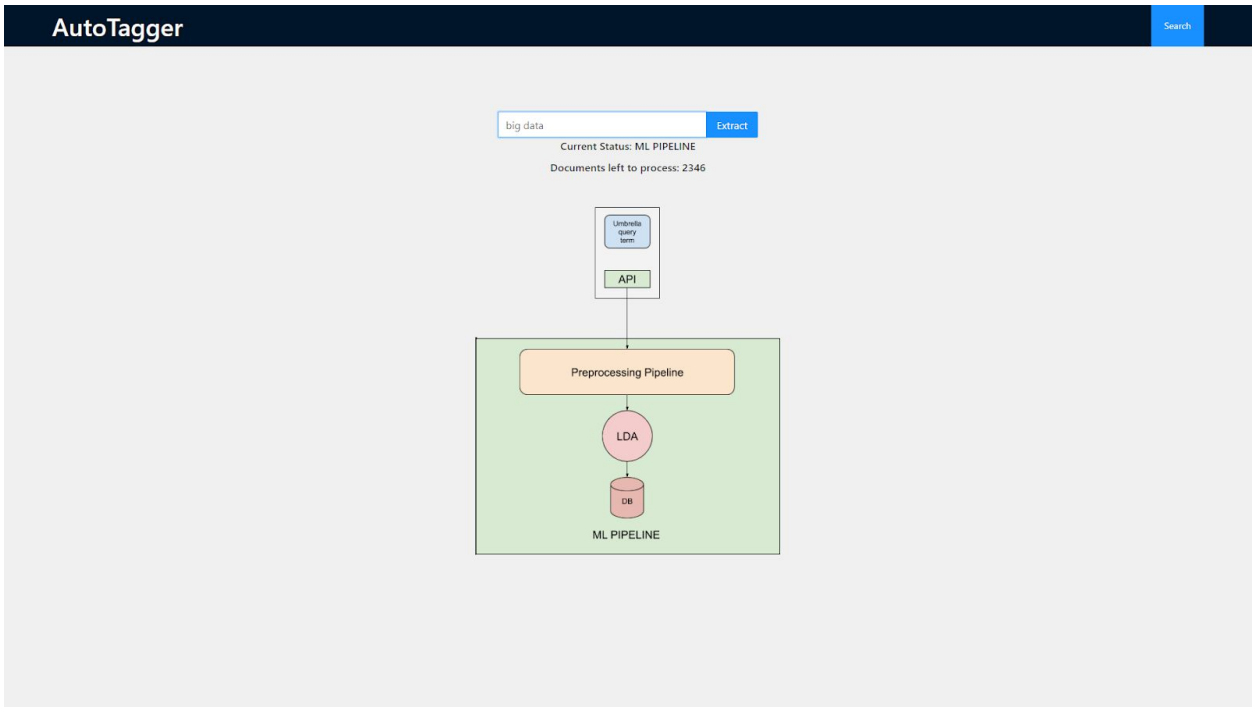
Search Page



Extract Page



Sample Query being Processed on Extract Page



Sample Query output on Search Page