Low Level Design (LLD)

Document Tagging Project

Author Name: Dibyendu Biswas.

Revision Number: 1.0
Last Date of revision: 24-November-2023

Document Version Control

Date issued	Version	Author	Comments
24-10-2023	1.0	Initial LLD – v1.0	Dibyendu Biswas

Reviews

Date issued	Version	Reviewer	Comments

Approval Status

Version	Date	Reviewed By	Approved By	Comments

Contents

Abstract

The Document Tagging Project is an innovative and scalable solution designed to address the challenge of organizing and categorizing vast amounts of unstructured textual data. In today's data-driven world, organizations accumulate massive repositories of documents, articles, reports, and other textual content. Efficiently tagging and classifying these documents based on their content is essential for improving searchability, knowledge management, and information retrieval.

The impact of the Document Tagging Project extends across various industries and applications. It empowers content creators, knowledge managers, and data analysts to streamline document organization and retrieval processes, thereby saving time and improving decision-making. Additionally, the project's scalability ensures adaptability to diverse data sources and domains, making it a valuable asset for organizations seeking to unlock the full potential of their textual data.

1. Introduction

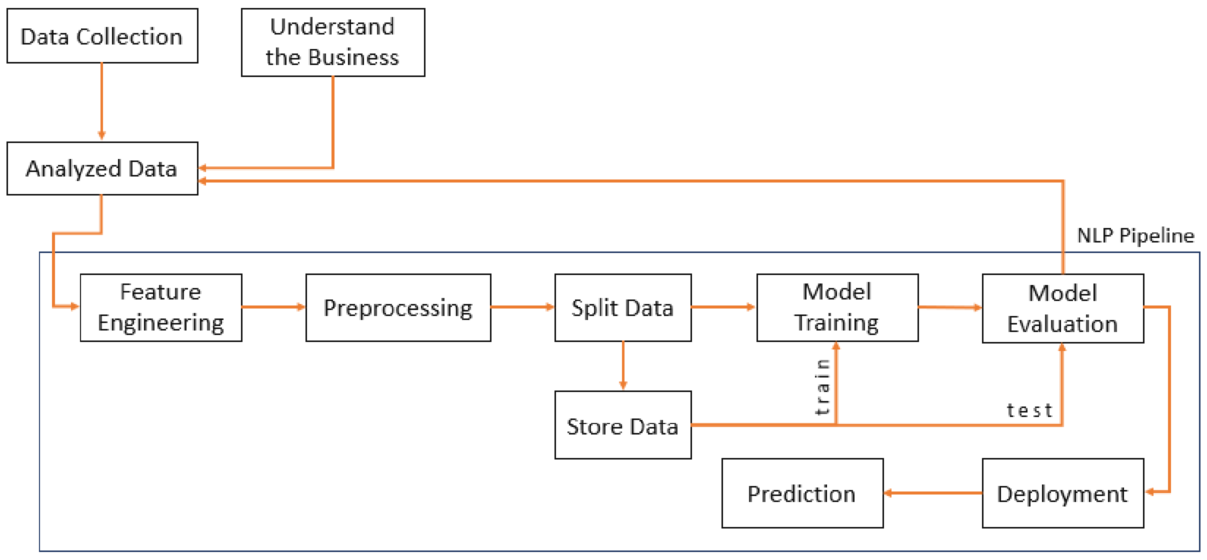
1.1 What is Low-Level Design Document

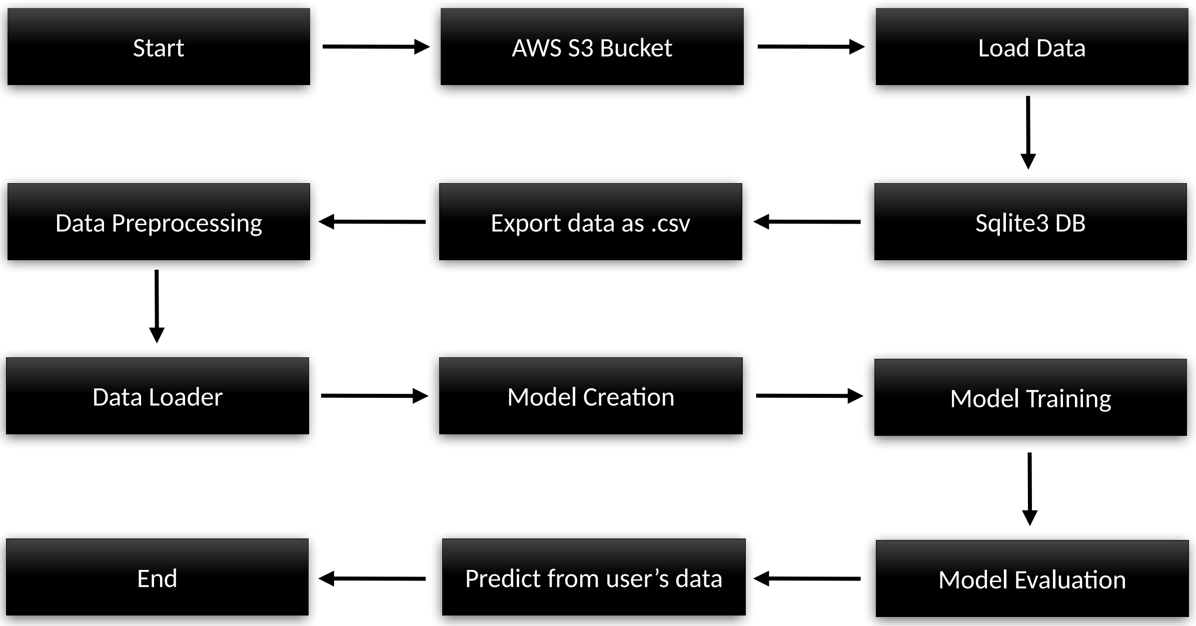
The goal of LLD or a low-level design document (LLDD) is to give the internal logical design of the actual program code for Food Recommendation System. LLD describes the class diagrams with the methods and relations between classes and program specs. It describes the modules so that the programmer can directly code the program from the document.

1.2 Scope

Low-level design (LLD) is a component-level design process that follows a systematic refinement process. This process can be used for designing data structures, required software architecture, source code and ultimately, performance algorithms. Overall, the data organization may be defined during requirement analysis and then refined during data design work.

2. Architecture





3. Architecture Description

3.1 Data Description

Here Data is present in AWS S3 Bucket. Two datasets are present, that's:

- **Wiki Data:** this data is related to Wikipedia, where two feature are present, one is **"Title"** and **"Tags"** and 20,760 records are available.
- **Auto tagged data:** this data is related to Data Science, Statistics, Business. Here 76,365 records are available and two columns are there one is **"Title"** and **"Tags"**.

3.2 Data Loading

Here, data is present in AWS S3 Bucket. So we load the from S3 bucket.

3.3 Data Insertion into Database

After loading data from S3 bucket, we create database-using sqlite3, create database table and push data into particular database.

3.4 Export Data from Database

We export data from sqlite3 database as "data.csv" format for further process.

3.5 Data Pre-processing

In Data Pre-processing steps we perform some operations like:

- **Handle the data:** handle the missing values by applying "forward-fill" method and remove the duplicates.
- **Separate X and Y data:** separate the X and Y data as list.
  Text Pre-processing: apply the text pre-processing steps like, lowering the sentences, remove the punctuation, remove the special characters, remove the stop-of-words, etc.
- **Generate id based on tag data:** generate the unique id based on given tags.
- **Word representation:** convert the word into numeric values by using AutoTokenizer of pre-trained model.
- **Split the data:** split the data into (X_train, attn_mask_X_train, Y_train), (X_test, attn_mask_X_test, Y_test) and (X_valid, attn_mask_X_valid, Y_valid).

3.6 Data Loader

After splitting the data, then create train_dataset, test_dataset and valid_dataset by using torch.

3.7 Model Creation

Load the pre-trained model **"bert-based-uncased"** from HuggingFace and return the model and also create custom "data_collector" function for handling sequence data.

3.8 Model Training

After model creation, train the model by using some hyper-parameters.

3.9 Model Evaluation

Now, evaluate the model based on train_dataset, test_dataset and valid_dataset. In addition, store the performance score into **“performance_report.json”** file.

3.10 Training Pipeline

Create training pipeline for train the model just one go.

3.11 Prediction Pipeline

Create prediction pipeline for predict the output.

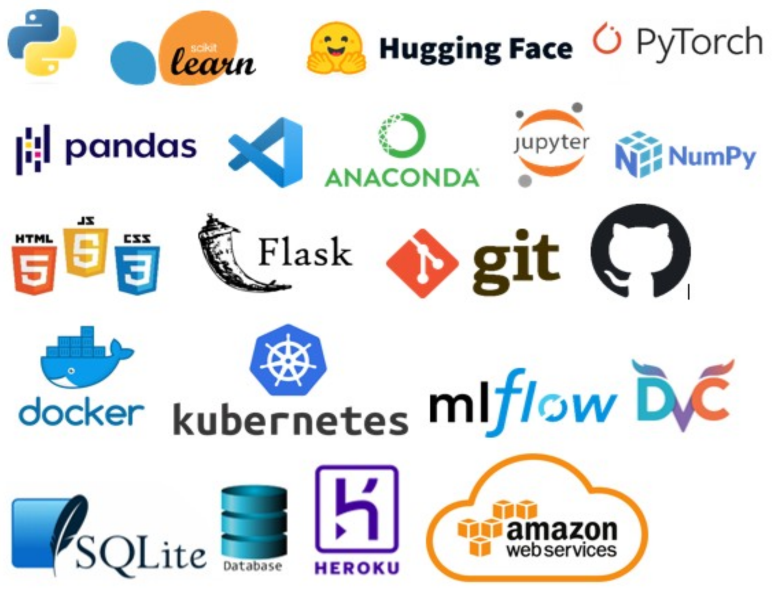
3.12 Predict from user’s data

Get the tags from user’s data by web interface.

3.11 Assumption

If any requiremts is needed then we can change it accordingly.

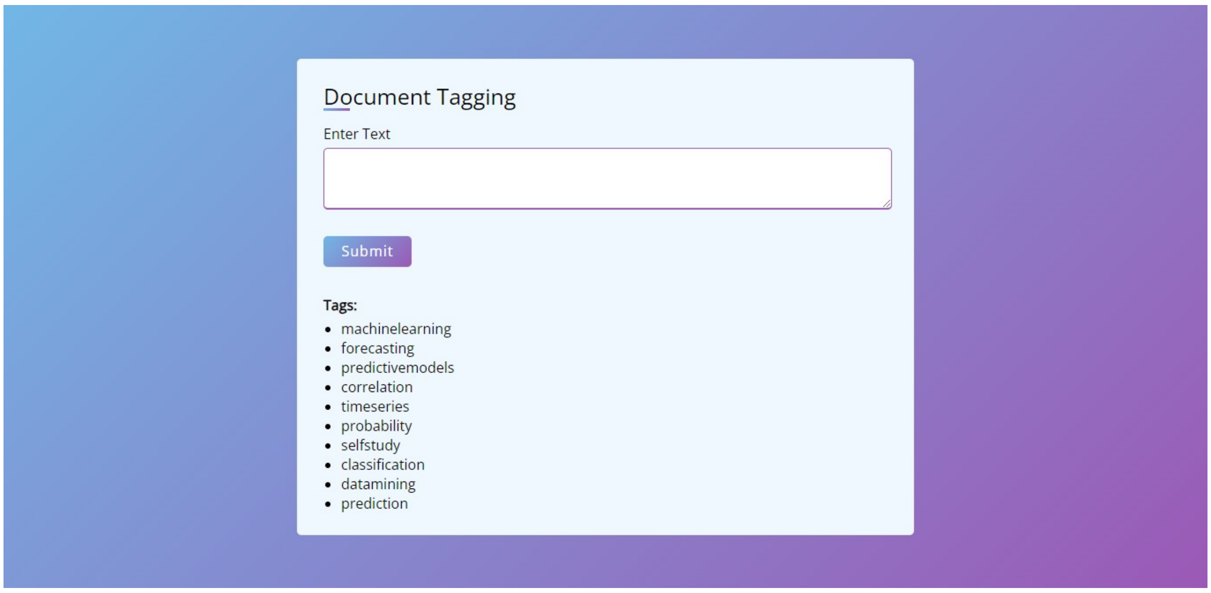
4. Technical Stack



5. Unit Test

SL No.	Test Case Description	Pre-Requisite	Expected Result
1.	Verify whether the Application URL is accessible to the user	1. Application URL should be defined	Application URL should be accessible to the user
2.	Verify whether the Application loads completely for the user when the URL is accessed	1. Application URL is accessible 2. Application is deployed	The Application should load completely for the user when the URL is accessed
3.	Verify whether user is able to edit all input fields	1. Application is accessible	User should be able to edit all input fields
4.	Verify whether user gets Submit button to submit the inputs	1. Application is accessible	User should get Submit button to submit the inputs
5.	Verify whether user gets the Output	1. Application is accessible	User should get the output
6.	Verify whether data is load from the S3 Bucket	1. CI/CD Pipeline	Data is load from S3 bucket and get the data
7.	Verify whether Database Operation is performed or not	1. CI/CD Pipeline	Database operation is successfully done and export the data as .csv format.
8.	Verify whether Data pre-processing is performed or not	1. CI/CD Pipeline	All the data pre-processing operation occurred successfully.
9.	Verify whether model creating and model training operation is performed or not	1. CI/CD Pipeline	Model creation and model training operation are performed.
10.	Verify whether you Dockerize your entire application	1. CI/CD Pipeline 1. Dockerfile	Dockerize the application successfully
11.	Verify whether CI/CD pipeline work or not	1. CI/CD Pipeline	CI/CD pipeline is working successfully

6. Web Interface



Document Tagging

Enter Text

Submit

Tags:

- machinelearning
- forecasting
- predictivemodels
- correlation
- timeseries
- probability
- selfstudy
- classification
- datamining
- prediction

7. Conclusion

Document Tagging project represents a significant step forward in automating and optimizing the categorization and organization of documents within organizations. By leveraging machine learning and natural language processing techniques, this project offers a powerful solution for improving document management workflows.

Through the development of a robust and scalable system, users can efficiently assign relevant tags to documents, enabling easier search, retrieval, and organization. The multi-label classification capability ensures that documents can be associated with multiple categories or topics, reflecting the complex nature of real-world document collections.

The project's user-friendly interface and customization options empower users to tailor the tagging system to their specific needs, whether it's in a corporate setting, academic institution, or any knowledge-intensive organization. Integrating the system with existing document management tools further enhances its utility and impact.

With a focus on accuracy, performance optimization, and ongoing maintenance, the Document Tagging project aims to provide a reliable and efficient solution that streamlines document management processes, saves time, and boosts productivity.

-- Thank You