News Articles Sorting

Low Level Design (LLD)

Author: Tanjina Proma

Date: 1/08/2024

Contents

1 Introduction

1.1 What is Low-Level-Design document?

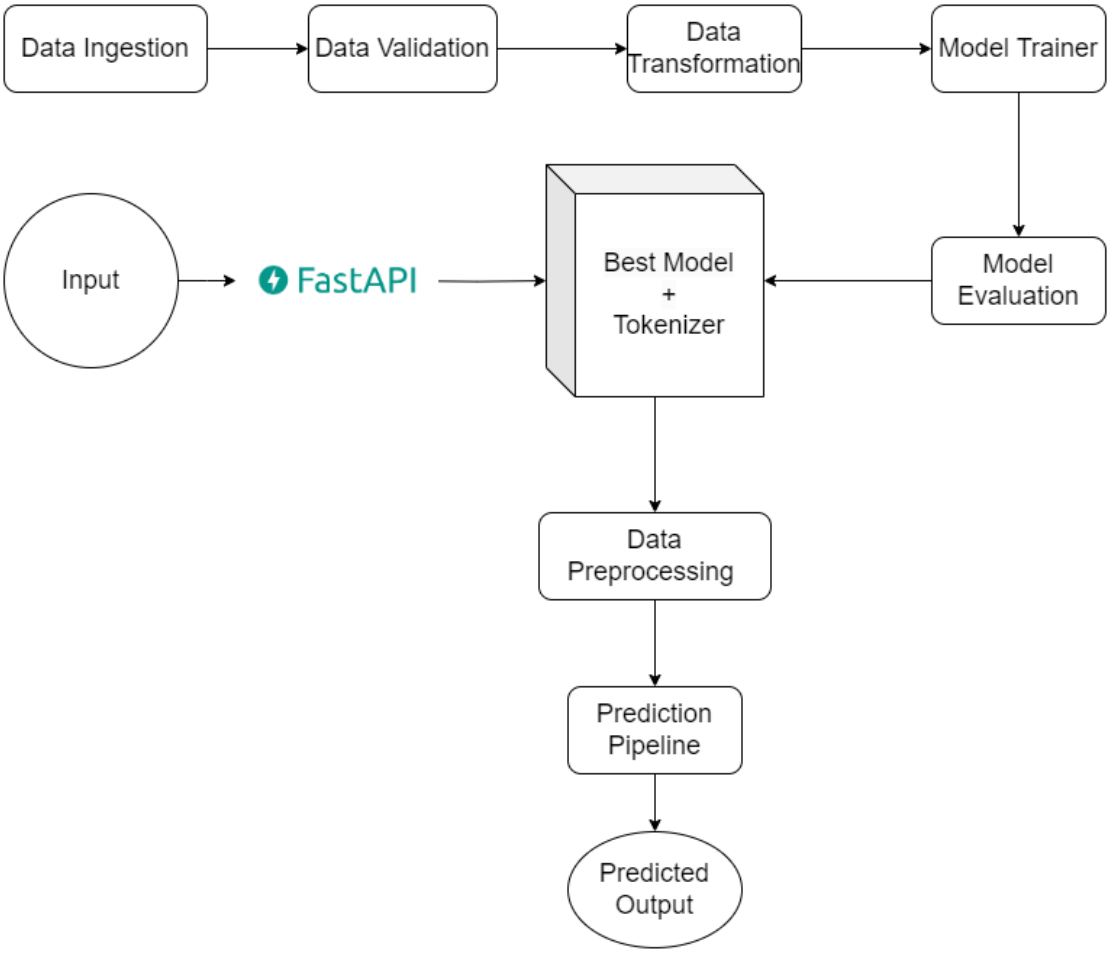
Low-Level Design (LLD) refers to the phase in software or system development where the actual implementation details of the system are planned out. This phase comes after the high-level design and involves creating detailed design specifications that describe how the system will be built.

The output of the LLD phase is often in the form of diagrams, charts, documents, and sometimes pseudocode that provides a blueprint for developers to start coding. It serves as a bridge between high-level architectural concepts and actual coding, making it easier for the development team to understand the implementation requirements.

1.2 Scope

The scope for news article classification is vast and pivotal in today's information age. By employing machine learning algorithms and natural language processing techniques, articles can be categorized into sections like politics, sports, finance, and more. This classification facilitates personalized content delivery, enhancing user experience on news platforms. Additionally, it aids in information retrieval, enabling users to access relevant articles swiftly. Moreover, news classification plays a crucial role in sentiment analysis, helping gauge public opinion on various topics. It also supports content moderation by identifying and filtering out inappropriate or misleading information. As technology advances, the scope broadens, encompassing multilingual classification, real-time updates, and the integration of emerging data sources. The evolution of news article classification continually reshapes how we consume and interact with information in our rapidly changing digital landscape.

2 Architecture



3 Architecture Description

3.1 Data Ingestion

Objective: Collect and gather news article data from the given source (APIs, databases, etc.) in the config.yaml file.

Actions:

- Define data sources (news websites, RSS feeds, databases).
- Extract data using web scraping, APIs, or other methods.
- Store data in a suitable format (e.g., CSV, JSON, and database).

3.2 Data Validation

Objective: Ensure data quality, consistency, and reliability.

Actions:

- Check for missing values, duplicates, and outliers.
- Validate data against predefined criteria or schemas.
- Handle data anomalies or errors appropriately.

3.3 Data Transformation and Preprocessing

Objective: Prepare data for model input by cleaning and formatting it.

Actions:

- Text cleaning: Remove HTML tags, special characters, and irrelevant symbols.
- Tokenization: Break text into words or subword units.
- Text normalization: Lowercasing, stemming, lemmatization.
- Feature engineering: Extract relevant features (TF-IDF, word embeddings).

3.4 BERT model Integration and Fine-tuning

Objective: Integrate BERT (Bidirectional Encoder Representations from Transformers) for text classification and fine-tune it on specific news article data.

Actions:

- Load pre-trained BERT model.
- Fine-tune BERT on the labeled news article dataset for classification tasks.
- Adjust hyper parameters for optimal performance.

3.7 Training pipeline

Objective: Train the text classification model using the prepared dataset.

Actions:

- Split dataset into training, validation, and possibly test sets.
- Train the BERT-based model using appropriate training algorithms (e.g., stochastic gradient descent).
- Monitor and record training metrics (loss, accuracy) for analysis.

3.8 Model Evaluation

Objective: Assess the model's performance and generalization.

Actions:

- Evaluate the model using metrics like accuracy, precision, recall, and F1-score.
- Conduct cross-validation or use a holdout dataset for robust evaluation.
- Identify areas of improvement and potential biases.

3.9 Inference Pipeline (Prediction)

Objective: Deploy the trained model for making predictions on new, unseen news articles.

Actions:

- Develop an inference pipeline to process new text inputs.
- Utilize the trained model to classify news articles into predefined categories.
- Generate predictions and probabilities for each class label.

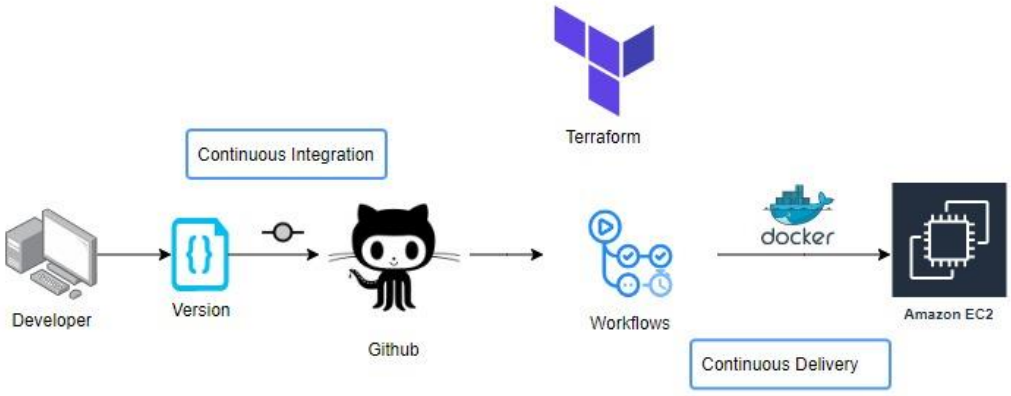
3.10 Integration and Deployment

Objective: Integrate the model into a production environment for real-time use.

Actions:

- Set up an API or service for model inference.
- Deploy the model using scalable infrastructure (cloud services, containers).
- Ensure compatibility with existing systems or applications.

Deployment Architecture



3.11 Logging and Monitoring:

Objective: Monitor model performance and system health in production.

Actions:

- Implement logging mechanisms to record predictions, errors, and model usage.
- Set up alerts for potential issues or deviations in performance.

- Monitor resource utilization and system behavior.

4. Unit Test Cases

1. Data Ingestion:

- **Test Case 1 - Data Source Availability**:
    - Ensure the data source (e.g., API endpoint, file path) is accessible.
    - Mock the data retrieval process and assert successful data retrieval.

2. Data Validation:

- **Test Case 2 - Missing Values Check**:
    - Create a mock dataset with missing values and validate the data validation function identifies and handles them correctly.
- **Test Case 3 - Duplicate Detection**:
    - Generate a dataset with duplicate entries and verify the duplication removal process.

3. Data Transformation and Preprocessing:

- **Test Case 4 - Text Cleaning**:
    - Input text with HTML tags, special characters, and symbols.
    - Ensure the cleaning function removes these elements effectively.
- **Test Case 5 - Tokenization Accuracy**:
    - Validate that the tokenizer correctly breaks text into tokens or words.
    - Compare the tokenized output against expected tokens for a given input.

4. BERT Model Integration and Fine-tuning:

- **Test Case 6 - BERT Model Loading**:
    - Check if the BERT model loads without errors and is ready for fine-tuning.
    - Ensure the loaded model matches the expected architecture and parameters.
- **Test Case 7 - Fine-tuning Accuracy**:
    - Train the BERT model on a small dataset with known labels.
    - Verify that the model's accuracy improves after fine-tuning.

5. Training Pipeline:

- **Test Case 8 - Training Data Split**:
    - Confirm that the dataset splits into training and validation sets as intended.
    - Check the ratio of samples in each split.
- **Test Case 9 - Training Convergence**:
    - Train the model on a small dataset for a limited number of epochs.
    - Verify that the loss decreases and accuracy increases or stabilizes.

6. Model Evaluation:

- **Test Case 10 - Evaluation Metrics Calculation**:
    - Use a test dataset with known labels to evaluate the model.
    - Calculate evaluation metrics (accuracy, precision, recall, F1-score) and compare against expected values.

7. Inference Pipeline (Prediction):

- **Test Case 11 - Inference Output**:
    - Input sample text and ensure the model produces the expected output class or label.

8. Integration and Deployment:

- **Test Case 12 - API Endpoint Test**:
    - Send a request to the deployed API with sample text for classification.
    - Validate that the response includes the predicted label or class.

9. Logging and Monitoring:

- **Test Case 13 - Logging Records**:
    - Simulate model usage and verify that logs are created accurately with relevant information (timestamps, predictions, input data).