Disaster Tweets Analyzer

Enhanced by NLP and LLMs

(As a part of ACTS Management System)

Functional Requirement Specification Document

Functional Requirements Document

Name of Document	Functional Requirement Specification Disaster Tweets Analyzer: Enhanced by NLP and LLMs
Author	Mansi, Ajay Singh, Dhananjay Singh, Damini Choudhary, Ritik Verma
Description of Content	Functional, Technical and Operational Requirements
Reference	

Contents

1.0 Introduction

Introduction to the project.

1.1 Purpose

The purpose of this document is to provide a detailed description of the requirements for the project "Disaster Tweets Analyzer Enhanced by NLP and LLMs". It aims to illustrate the purpose, scope, and functionality of the system, explaining the system constraints, interfaces, and interactions with other external applications. This document is intended for the customer for functional requirements approval and as a reference for the development team.

1.2 Scope

Intended Audience

This document is intended for the customer for functional requirements approval and as a reference for the development team.

1.3 References

- Functional Requirement Specifications (FRS)
- BERT research papers
- Twitter API documentation

2.0 Acronyms, terms, and definition

- FRS: Functional Requirements Specification
- NLP: Natural Language Processing
- BERT: Bidirectional Encoder Representations from Transformers
- API: Application Programming Interface.

3.0 Assumptions and constraints

- Twitter API keys must be obtained and configured
- Python libraries for text preprocessing must be installed
- Keywords and hashtags for filtering must be predefined
- Language and geolocation detection algorithms must be implemented
- Sentiment analysis tool must be integrated and configured

4.0 Basic Design approach

The design approach for the NLP with Disaster Tweets project using BERT involves several structured and systematic steps to ensure the efficient and accurate classification of tweets. The primary focus is on leveraging BERT's advanced language understanding capabilities to distinguish between disaster-related and non-disaster-related tweets. The approach includes data collection, preprocessing, model training, deployment, and continuous improvement.

I. Data Collection

- **Twitter API Integration**: Integrate with the Twitter API to collect real-time tweets. Use relevant hashtags and keywords related to disasters (e.g., #earthquake, #flood, #hurricane) to filter tweets.
- **Historical Data**: Collect historical data on past disasters to build a comprehensive dataset for training and testing the model.

II. Data Preprocessing

- **Text Cleaning:** Remove irrelevant information such as URLs, mentions, special characters, and emojis from the tweet text.
- **Tokenization**: Tokenize the text into words or sub words suitable for BERT input.
- **Lowercasing**: Convert all text to lowercase to ensure uniformity.
- **Stop Words Removal**: Optionally, remove common stop words that do not contribute to the tweet's context.

III. Model Training

- **Pre-trained BERT Model:** Use a pre-trained BERT model as a base. BERT (Bidirectional Encoder Representations from Transformers) is selected for its state-of-the-art performance in various NLP tasks.
- **Fine-tuning:** Fine-tune the BERT model on the labeled disaster tweet dataset. This involves:
    - Dataset Preparation: Split the collected data into training, validation, and test sets.
    - Model Fine-tuning: Adjust the pre-trained BERT model's weights using the training dataset. Implement early stopping and other regularization techniques to prevent overfitting.

    - Evaluation: Evaluate the model on the validation set to monitor its performance. Metrics such as accuracy, precision, recall, and F1-score are used for evaluation.

IV. Deployment

- **Model Serving:** Deploy the fine-tuned BERT model using a model-serving framework like TensorFlow Serving, PyTorch Serve, or Fast API. This enables the model to handle real-time tweet classification requests.
- **API Development:** Develop RESTful APIs to interact with the model. Key functionalities include:
    - /Classify Tweet: Endpoint to classify a single tweet.
    - /Batch Classify: Endpoint to classify a batch of tweets.
    - /fetchClassifiedTweets: Endpoint to retrieve classified tweets from the database.

V. Data Storage

- **Database Design**: Design a relational database to store raw tweets, preprocessed tweets, and classification results. Key tables include:
    - Raw Tweets: Stores raw tweets fetched from Twitter.
    - Preprocessed Tweets: Stores preprocessed tweets.
    - Classified Tweets: Stores classification results with tweet IDs, text, and labels.

VI. Continuous Improvement

- **Model Retraining**: Continuously collect new data and retrain the model to improve its accuracy and adapt to new disaster-related terminology.
- **Feedback Loop**: Implement a feedback loop where users can provide feedback on classification results. Use this feedback to correct misclassifications and improve the model.
- **Performance Monitoring**: Monitor the model's performance in production. Set up alerts for significant drops in performance and initiate retraining or other corrective measures as needed.

VII. Security and Privacy

- **Data Privacy**: Ensure that the data handling complies with relevant data privacy regulations.
- **Security Measures**: Implement security measures to protect the API endpoints and the data storage from unauthorized access.

5.0 Risks

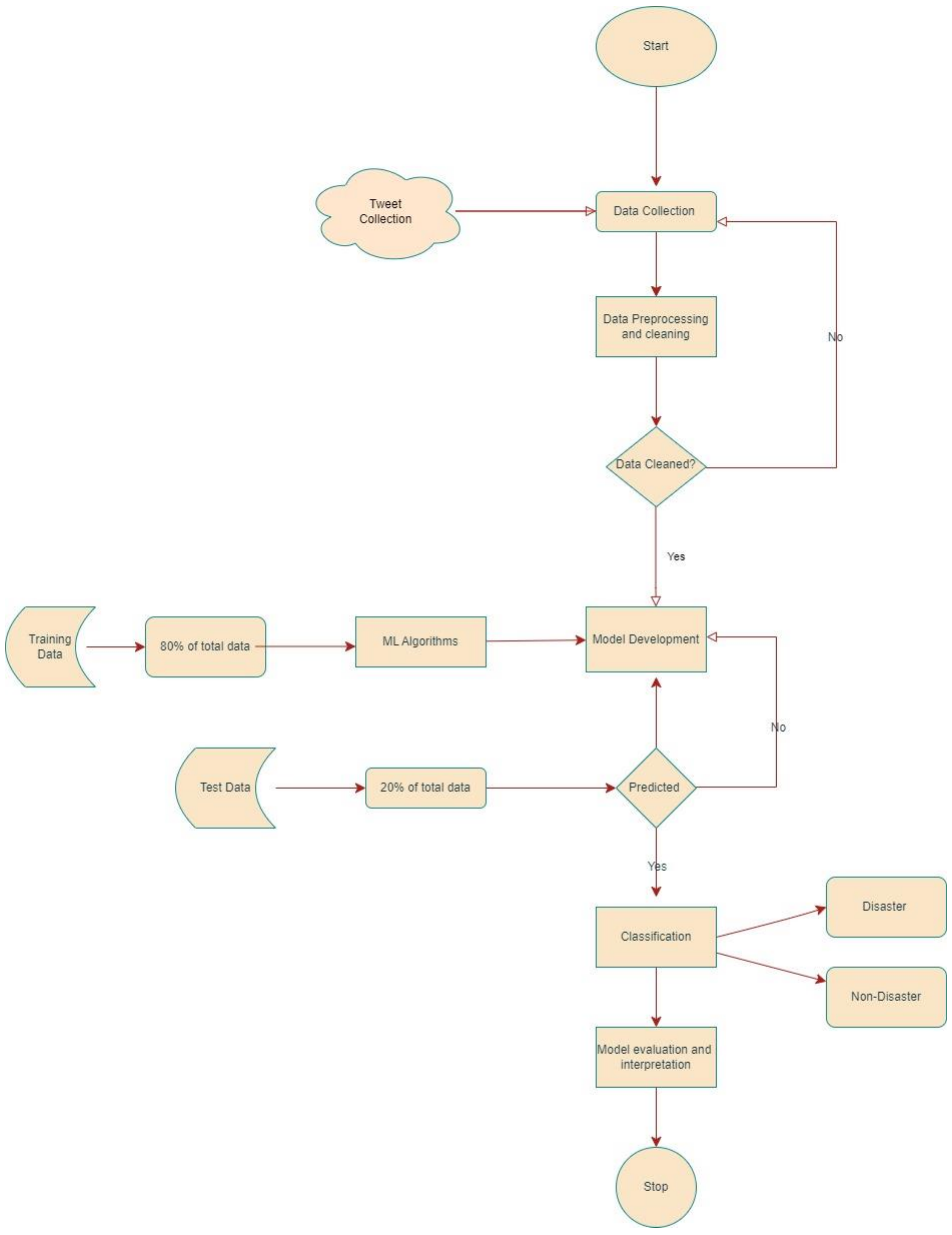
- Risks in the project include handling personal data and ensuring data privacy and security.

6.0 System overview

The system will classify tweets as disaster-related or not using a fine-tuned BERT model. The system will fetch tweets using the Twitter API, preprocess them, and classify them using the BERT model. The classified tweets will be stored in a database for further analysis and reporting.

7.0 Architecture Design

Architecture diagram is prepared to fulfill the requirements as per the FRS as shown in Figure 1.



8.0 Data Design

The data is stored in a relational database. The database tables are divided into master tables (prefixed by 'm'), logs (prefixed by 'l'), and transaction tables (no prefix).

Database Tables

- **Table: Tweets**
    - Idx (Primary Key)
    - TweetID: VARCHAR
    - TweetText: TEXT
    - Classification: VARCHAR
    - Timestamp: DATETIME
- **Indexes (Foreign Keys)**
    - ForeignKey: TweetID references Tweet Table.

9.0 Component Design

Twitter Data Ingestion

- Description: Fetch tweets from Twitter API using specific keywords related to disasters.
- Process:
    - Connect to Twitter API.
    - Fetch tweets.
    - Store raw tweets in the database.

Data Preprocessing

- Description: Clean and preprocess tweets for model input.
- Process:
  - Remove URLs, mentions, and special characters.
  - Tokenize and lower-case text.
  - Store preprocessed tweets in the database.

BERT Model

- Description: Fine-tuned BERT model for classifying tweets.
- Process:
  - Load pre-trained BERT model.
  - Fine-tune on labeled disaster tweet dataset.
  - Classify new tweets and store results in the database.

10.0Any Specific Design Considerations

- Data Quality: Ensure the quality of the collected tweets by filtering out spam, advertisements, and irrelevant content. This improves the model's training and classification accuracy.
- Fine-tuning BERT: BERT is computationally intensive, and fine-tuning it requires substantial computing resources. Optimize the fine-tuning process by using techniques like mixed precision training and gradient accumulation.
- Microservices: Implement a microservices architecture where different components (data ingestion, preprocessing, classification, storage, API gateway) are decoupled and can be scaled independently.
- Performance Monitoring: Set up continuous monitoring of the model's performance in production. Track metrics such as accuracy, precision, recall, and F1-score, and set up alerts for performance degradation.

11.0Design Test

The design testing for the NLP with Disaster Tweets project using BERT involves multiple stages, each focusing on different aspects of the system. This ensures the overall functionality, performance, and reliability of the system. Below are the key components and strategies for design testing:

1. Unit Testing

Objective: Verify individual components of the system, such as data preprocessing, model training, and API endpoints, to ensure they function as expected.

Scope:

- Test text cleaning functions to ensure URLs, mentions, and special characters are correctly removed.
- Validate tokenization and text preprocessing steps.
- Check the accuracy of the BERT model's predictions on a small set of labeled tweets.
- Verify the API endpoints return the expected responses.

Tools: Use unit testing frameworks like PyTest (Python).

2. Integration Testing

Objective: Ensure different components of the system work together seamlessly.

Scope:

- Test the integration between the Twitter API, data preprocessing, BERT model, and database.
- Validate the end-to-end workflow from tweet ingestion to classification and storage.
- Ensure the API gateway correctly interfaces with the backend services.

Tools: Use integration testing frameworks like Postman for API testing and Selenium for end-to-end testing.

3. Performance Testing

Objective: Assess the system's performance under various conditions and ensure it meets the required standards.

Scope:

- Measure the response time of the API endpoints under different loads.
- Evaluate the model's inference time for classifying tweets.
- Test the system's scalability by simulating high tweet volumes during disaster events.

4. Load Testing

Objective: Determine the system's behavior under expected and peak load conditions.

Scope:

- Simulate high traffic to the API endpoints and monitor system performance.
- Ensure the database can handle large volumes of read and write operations.
- Assess the system's ability to maintain performance with increasing tweet ingestion rates.

5. Model Evaluation

Objective: Continuously evaluate the model's performance to ensure it remains accurate and reliable.

Scope:

- Periodically test the model on new data to assess its accuracy, precision, recall, and F1-score.
- Implement a feedback loop where misclassified tweets are reviewed and used for model retraining.

Tools: Use tools like Scikit-learn for evaluation metrics and TensorFlow or PyTorch for model retraining.

12.0 Cross Reference with System Requirement Specification

System Requirement Specification Features		
API's Developed	**Requirements**	**Adherence/Limitations**
Disaster Tweets Analyzer: Enhanced by NLP and LLMs		
/classifyTweet	Mapping with Functional requirement id FR1.0	Meets the functional requirement completely.

Appendix A/B/C – Appendix Name

Appendix A – Data Collection and Preprocessing

A.1 Twitter API Integration

Details: Instructions and code snippets for integrating with the Twitter API to collect tweets in real-time using relevant hashtags and keywords.

A.2 Text Cleaning

Details: Techniques and code snippets for cleaning tweet text by removing URLs, mentions, special characters, and converting text to lowercase.

A.3 Tokenization

Details: Steps for tokenizing tweet text using BERT tokenizer.

Appendix B – Model Training and Evaluation

B.1 Fine-Tuning BERT

Details: Instructions for fine-tuning a pre-trained BERT model on the disaster tweet dataset.

B.2 Evaluation Metrics

Details: Techniques for evaluating model performance using accuracy, precision, recall, and F1-score.

Appendix C – Deployment and Monitoring

C.1 Model Serving

Details: Instructions for deploying the fine-tuned BERT model using a model-serving framework.

C.2 Performance Monitoring

Details: Tools and methods for monitoring the performance and health of the d
deployed model.