RAJALAKSHMI ENGINEERING COLLEGE
RAJALAKSHMI NAGAR, THANDALAM – 602 105

RAJALAKSHMI
ENGINEERING COLLEGE
An AUTONOMOUS Institution
Affiliated to ANNA UNIVERSITY, Chennai

CP23211 ADVANCED SOFTWARE ENGINEERING LAB

LAB RECORD

Name:…………………… SHYAM V P ……..…………………………

Year / Branch / Section: ……1st YEAR/ ME-CSE…..………………….

University Register No.: …2116230711010…………..……………….

College Roll No.: ……230711010…………….………………………

Semester: ……………………2nd ……………..……………………………..

Academic Year: ……2023-2024…………………………………………

RAJALAKSHMI ENGINEERING COLLEGE

RAJALAKSHMI NAGAR, THANDALAM – 602 105

BONAFIDE CERTIFICATE

Name: SHYAM V P

Academic Year: 2023-2024 Semester: II Branch: ME-CSE

Register No: 2116230711010

Certified that this is the bonafide record of work done by the above student in the CP23211-Advanced Software Engineering Laboratory during the year 2023- 2024

Signature of Faculty-in-charge

Submitted for the Practical Examination held on 22/06/2024

Internal Examiner External Examiner

INDEX

Impact of Context on NLP-based Hate Speech Detection Accuracy

OVERVIEW OF THE PROJECT:

To keep the internet safe and welcoming for all users, it is essential to identify hate speech when it appears on various platforms. Despite the growing use of Natural Language Processing methods for this aim, it is still very difficult to comprehend how context affects hate speech identification. In this work, we compare the results of natural language processing models trained with and without context to find out how much of an impact context has on hate speech identification. We investigate the impact of adjacent words and larger context on hate speech recognition systems' accuracy using a varied dataset. Models that include context achieve improved recall rates and accuracy, as our results show that taking contextual information into account greatly enhances detection ability. We also find concrete examples where the surrounding context is crucial in determining whether a piece of words is hostile or not. Nevertheless, contextaware hate speech identification faces obstacles including dataset biases and contextual ambiguity, which restrict its usefulness. In order to create more reliable Natural Language Processing models that can detect hate speech in different online settings, it is essential to tackle these issues. This research adds to the current body of work on the topic of online hate speech by drawing attention to the need for context-aware methods in hate speech detection systems that rely on natural language processing.

SOFTWARE REQUIREMENTS SPECIFICATION (SRS)

EXP.NO: 1 **DATE: 05/03/2024**

CONTENTS

Automating Hate Speech Detection for Safer Online Spaces

1. Introduction

1.1 Purpose:

Hate speech detection aims to automatically identify content that attacks or discriminates against a person or group based on attributes like race, religion, ethnicity, sexual orientation, or disability. By flagging hate speech, platforms can discourage its spread and protect users from harassment and abuse. Automated detection tools can assist human moderators in reviewing large volumes of content for potential hate speech violations. Detection helps researchers track trends and understand the nature of hate speech online.

1.2 Scope:

Most current systems focus on written text in social media posts, comments, and online forums. Some systems are being developed to handle hate speech across different languages. Advanced systems aim to analyze images, audio, and video alongside text for a more comprehensive approach.

2. Overall Description

2.1 Product Perspective:

From a product perspective, hate speech detection tackles a major online challenge. It caters to social media platforms, online communities, and even educational institutions by offering real-time detection, customizable filters, and multilingual support. This empowers them to create safer spaces and reduce workload for human moderators. The value proposition lies in fostering inclusivity, improving brand trust, and generating data for research. However, the product must navigate the complexities of free speech, bias mitigation, and evolving language to achieve success. This can be measured through reduced hate speech content, user satisfaction, and detection accuracy.

2.2 Features

- **Text Analysis:** The core functionality involves analyzing textual content for signs of hate speech. This includes identifying offensive words, phrases, and discriminatory language.
- **Machine Learning:** Many systems leverage machine learning models trained on massive datasets of labeled hate speech and non-hate speech content.

- **Nuanced Detection:** Advanced systems go beyond simple keyword matching. They consider context, sarcasm, and cultural references to improve accuracy and avoid flagging protected forms of expression.
- **Multilingual capabilities:** Some projects incorporate multilingual support to detect hate speech across different languages. This is crucial for global platforms.
- **Customization:** Platforms can define the types of hate speech they want to target (racism, sexism etc.) and set sensitivity thresholds to avoid over-flagging.
- **Real-time Detection:** Ideally, the system should flag hate speech as it appears, allowing for immediate action by moderators or automated filtering.
- **Reporting and Analytics:** The project can provide valuable insights into hate speech trends and user behavior. This data can inform content moderation strategies and research efforts.
- **Explain ability:** Some systems can explain why a particular piece of content was flagged, which can be helpful for human reviewers and for improving the model's accuracy.

2.3 User Classes and Characteristics:

This hate speech detection project targets a range of users. Social media platforms need real-time, multilingual detection with customizable filters. Online communities require specific focus and integration with existing tools. Educational institutions and businesses value control and clear explanations for flagged content. Researchers seek access to anonymized data for analysis. Law enforcement has a potential future role, but ethical and legal considerations regarding bias and free speech must be addressed.

3. Specific Requirements

3.1 Functional Requirements:

3.1.1 Input & Output:

- **Input:** The system should accept textual content in various formats (posts, comments, messages) potentially across multiple languages.
- **Output:** The system should classify the input content as "hate speech" or "non-hate speech" with a confidence score.

3.1.2 Detection Capabilities::

- **Accuracy:** The system should achieve a high level of accuracy in hate speech detection, minimizing false positives (non-hate speech flagged) and false negatives (hate speech missed).

- **Nuance:** Go beyond simple keyword matching. Account for context, sarcasm, and cultural references to avoid flagging protected speech.
- **Customizability:** Allow users to define the types of hate speech to target (racism, sexism etc.) and adjust sensitivity thresholds.

3.1.3 Scalability and Performance:

- **Real-time processing:** Ideally, the system should analyze content in real-time to enable immediate action.
- **High volume handling:** The system should be able to efficiently process large amounts of content, especially for platforms with high user activity.

3.1.4 Reporting and Analytics:

- Generate reports on the volume and types of hate speech detected.
- Provide insights into trends and patterns over time.
- Offer data visualizations for easier comprehension.

3.1.5 Security and Privacy:

- Implement robust security measures to protect user data and prevent unauthorized access.
- Anonymize data used for training and analysis, ensuring user privacy.

3.2 Non-Functional Requirements:

3.2.1 Usability

The system interface should be intuitive and user-friendly for various technical backgrounds (social media platforms, researchers, etc.). Users should be able to configure the system to their specific needs (e.g., language, hate speech focus, sensitivity). Integrate seamlessly with existing content moderation platforms or learning management systems.

3.2.2 Performance

Minimize the time it takes to analyze content and provide results, especially for real-time applications. The system should be able to handle increasing volumes of content without performance degradation.

3.2.3 Security

Implement robust security measures to protect user data from unauthorized access, breaches, or leaks. An anonymize training and analysis data to safeguard user privacy. Ensure the system adheres to relevant data privacy regulations (e.g., GDPR, CCPA).

4. External Interface Requirements

4.1 Social Media Platforms & Online Communities :

- Provide a well-documented API for seamless integration with existing content moderation platforms.
- Allow configuration of the API to define targeted hate speech types (racism, sexism etc.) and adjust sensitivity thresholds.
- Offer flexible output formats like JSON or text reports with flagged content details and confidence scores.

4.2 Educational Institutions & Businesses:

- Similar to social media platforms, allow configuration of targeted hate speech and sensitivity levels.
- Integrations with Learning Management Systems (LMS) or internal communication platforms for streamlined content analysis.
- Enable exporting anonymized data on flagged content for internal analysis or reporting purposes.

4.3 Researchers:

- Allow filtering data by specific demographics, platforms, or hate speech categories for in-depth analysis.
- Enable export of anonymized data in research-friendly formats (CSV, JSON) for further analysis.
- Provide comprehensive documentation on the system's methodology, limitations, and potential biases to aid research efforts.

4.4 Law Enforcement:

- A secure and encrypted interface for authorized law enforcement personnel to access flagged content meeting specific criteria (e.g., threats of violence).
- Granular control over data access to ensure only relevant information is available to authorized users.
- Robust user authentication protocols to prevent unauthorized access to sensitive data.
- The interface should adhere to legal requirements for data handling and chain of

custody for potential use as evidence.

4.5 General Considerations:

- Comprehensive API documentation for developers integrating the system.
- All interfaces should implement strong security measures like user authentication, data encryption, and access controls.
- The interfaces should be designed to handle increasing user loads and data volumes.

5. Conclusion

Hate speech detection offers a powerful tool to combat online negativity. This project outlined the functionalities and considerations for building a robust system. By implementing features like real-time analysis, customizable filters, and multilingual support, the system empowers platforms to create safer spaces. However, addressing bias, ensuring explainability, and prioritizing user privacy are crucial for responsible development. Ultimately, this project has the potential to foster a more inclusive online environment where respectful dialogue can thrive.

SCRUM METHODOLOGY

EXP.NO: 2 **DATE: 14/03/2024**

1. Introduction

The Scrum methodology can be a great fit for developing a hate speech detection project due to its iterative and adaptable nature. Here's how Scrum can be applied to this project.

2. Objectives

- Develop a system to automatically identify hate speech online. This protects users from harassment and abuse.
- Empower platforms to moderate content effectively. Real-time detection and customizable filters help create safer online spaces.
- Support researchers in understanding hate speech trends. This data can inform strategies to combat online negativity.
- Promote a more inclusive online environment. By reducing hate speech, respectful dialogue can flourish.

3. Product Backlog Introduction

The product backlog is a dynamic list of features, enhancements, and fixes prioritized by the product owner. It serves as a roadmap for the development team.

4. Product Backlog

Start by creating a comprehensive product backlog. This backlog will list all user stories, features, and functionalities required for the hate speech detection system.

5. User Stories

- **As an Social Media Platform user:**
  - As a social media platform, I want the system to detect hate speech in real-time so I can take immediate action (e.g., remove content, flag for review).
  - As a social media platform, I want to define the types of hate speech I want to target (e.g., racism, sexism, etc.) to improve the accuracy of detection for my user base.
  - As a social media platform, I want the system to provide different output formats (e.g., text report, API response) for flagged content to integrate with my existing moderation workflows.

● **As a Online Community/Forum moderator:**

- As an online community moderator, I want the system to identify hate speech that is specific to my community (e.g., gaming lingo, hate symbols) to maintain a safe and inclusive space for my members.
- As an online community moderator, I want the system to integrate seamlessly with my existing moderation tools to streamline my workflow.

● **As a Educational Institution/Business:**

- As an educational administrator, I want the system to be configurable to different sensitivity levels to avoid flagging legitimate academic discourse.
- As an educational administrator, I want the system to provide clear explanations for why content is flagged as hate speech to guide users and improve their communication.

6. Sprint

- A time-boxed iteration during which a set of user stories is implemented and tested.

7. Sprint Backlog

The sprint backlog is a list of tasks selected from the product backlog for a specific sprint. In other terms Sprint Backlog could also be defined as the subset of Product backlog which is chosen for a specific sprint. In general a sprint backlog allows the development team to work on the tasks necessary to implement the User Stories with in the selected sprint.

8. Sprint Review

A meeting held at the end of each sprint to review and demonstrate the completed work.

Sprint 1 Review:

- The sprint review is a meeting that includes the demonstration of implemented features at that particular sprint that is conducted at the end of each sprint.
- In the Hate speech Detection: an approach to Hate speech detection and profiling the sprint backlog for the first week is the Use case Diagram and the sprint backlog for the second use case is Software Requirement Specification document.

9. Software Used

- **Development Platform:** Jupiter IDE, Python R Studio

10. Conclusion

The Agile Scrum framework for the Hate speech Detection model: an approach to cyber anomaly detection and profiling ensures a systematic approach to development, focusing on user needs and iterative improvements. By breaking down the project into manageable sprints, the framework allows for continuous feedback, resulting in a robust and user-friendly NLP Model for Speech detection Team and administrators alike.

USER STORIES

EXP.NO: 3 **DATE: 26/03/2024**

As a Social Media Platform user:

1. Detecting hate speech

User Story: I want the system to detect hate speech in real-time so I can take immediate action (e.g., remove content, flag for review).

Acceptance Criteria:

- The system analyzes text content submitted through an API within a maximum of 2 seconds.
- The system outputs a flag indicating "hate speech" or "not hate speech" with a confidence score above 80%.
- The flagged content includes the original text and specific phrases/words identified as hate speech.

2. Defining types of Hate speech

User Story: I want to define the types of hate speech I want to target (e.g., racism, sexism, etc.) to improve the accuracy of detection for my user base.

Acceptance Criteria:

- The system provides a configuration interface to define targeted hate speech categories (e.g., racism, sexism, etc.).
- The system prioritizes detection of user-defined hate speech categories in the analysis.
- The system accuracy for user-defined categories remains above 75% with a confidence score.

3. Providing different output formats

User Story: I want the system to provide different output formats (e.g., text report, API response) for flagged content to integrate with my existing moderation workflows.

Acceptance Criteria:

- The system offers options to receive flagged content data in JSON and text report formats.
- The output data includes details like flagged content, timestamps, confidence scores, and identified hate speech categories.
- The API response follows a documented format for seamless integration with existing moderation tools.

As an Online community/forum moderator:

4. Detecting hate speech towards community

User Story: I want the system to identify hate speech that is specific to my community (e.g., gaming lingo, hate symbols) to maintain a safe and inclusive space for my members.

Acceptance Criteria:

- The system allows uploading a glossary of community-specific hate speech terms and symbols.
- The system incorporates the uploaded glossary into the analysis process for improved detection accuracy.
- The system highlights flagged content containing community-specific hate speech terms/symbols for moderator review.

5. Seamless integration

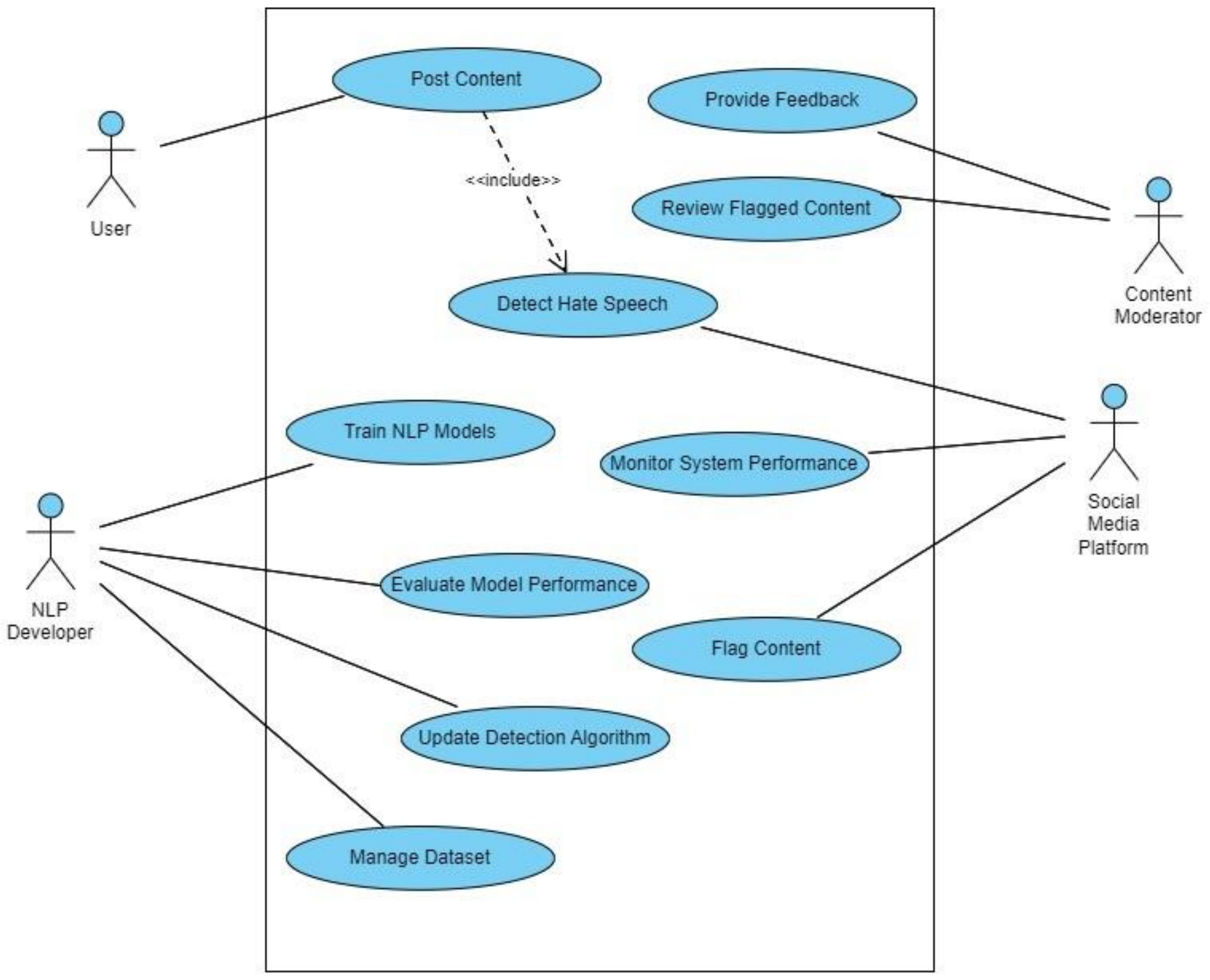
User Story: I want the system to integrate seamlessly with my existing moderation tools to streamline my workflow.

Acceptance Criteria:

- The system provides plug-in or APIs for integration with popular community moderation platforms.
- Flagged content is automatically pushed to the existing moderation queue for review and action.

USE CASE DIAGRAM

EXP.NO: 4 **DATE: 04/04/2024**



NON-FUNCTIONAL REQUIREMENTS

EXP.NO: 5 **DATE: 16/04/2024**

1. **Performance**

Minimize the time it takes to analyze content and provide results, especially for real-time applications. The system should be able to handle increasing volumes of content without performance degradation.

2. **Security**

Implement robust security measures to protect user data from unauthorized access, breaches, or leaks. An anonymize training and analysis data to safeguard user privacy. Ensure the system adheres to relevant data privacy regulations (e.g., GDPR, CCPA).

3. **Usability**

The system interface should be intuitive and user-friendly for various technical backgrounds (social media platforms, researchers, etc.). Users should be able to configure the system to their specific needs (e.g., language, hate speech focus, sensitivity). Integrate seamlessly with existing content moderation platforms or learning management systems.

4. **Reliability**

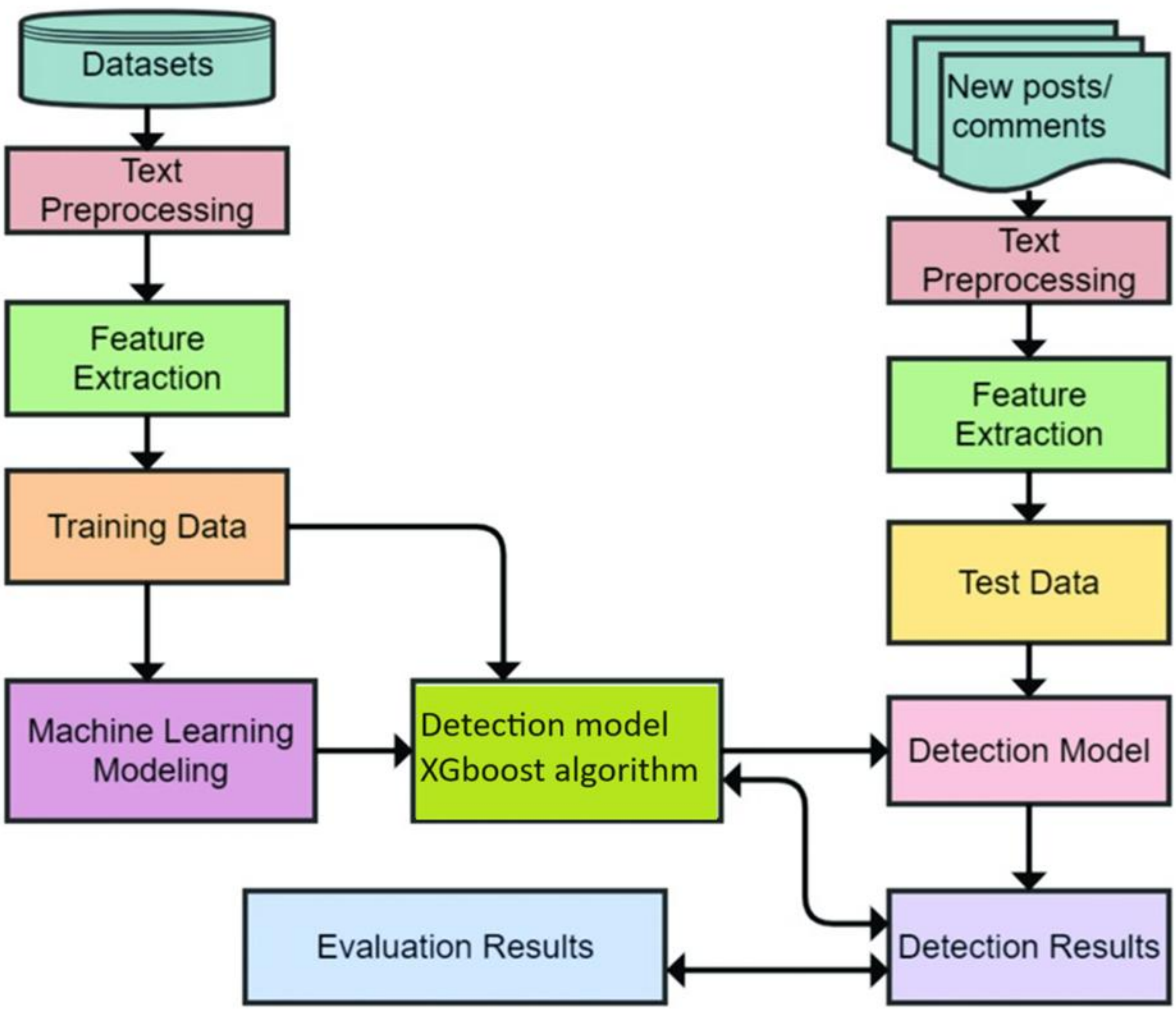
The system should be available and operational for extended periods with minimal downtime. Users rely on the system to maintain a safe online environment, so outages can disrupt critical operations. The system should gracefully handle errors and unexpected situations. This might involve error messages, automatic retries, or failover mechanisms to prevent complete service disruption. Implement regular backups to ensure data recovery in case of system failures. Backups should be secure and follow data privacy regulations.

5. **Maintainability**

The system should be designed with maintainability in mind, allowing for future modifications, bug fixes, and upgrades.

OVERALL PROJECT ARCHITECTURE

EXP.NO: 6 **DATE: 25/06/2024**



STAGES:

STAGE 1: Data Collection and Data Pre-processing

- **Data Retrieval:** This layer is the first step where data is retrieved from a Twitter database. The data is retrieved for a specific period. This layer interacts with the data storage system (database) to retrieve, store, and manage data.
- **Data Pre-processing:** This section covers various techniques to prepare the raw data for further analysis.

STAGE 2: Feature Engineering

Features are extracted from the preprocessed text. These features could be things like:

- Word n-grams (sequences of n words)
- Part-of-speech tags
- Named entity recognition (identifying people, places, organizations)

STAGE 3: Machine Learning Modeling

- A machine learning model, here an XGBoost algorithm, is trained on the training data.
- During training, the model learns to identify patterns that differentiate hateful and non-hateful language.

STAGE 4: Evaluation and Deployment

Evaluation

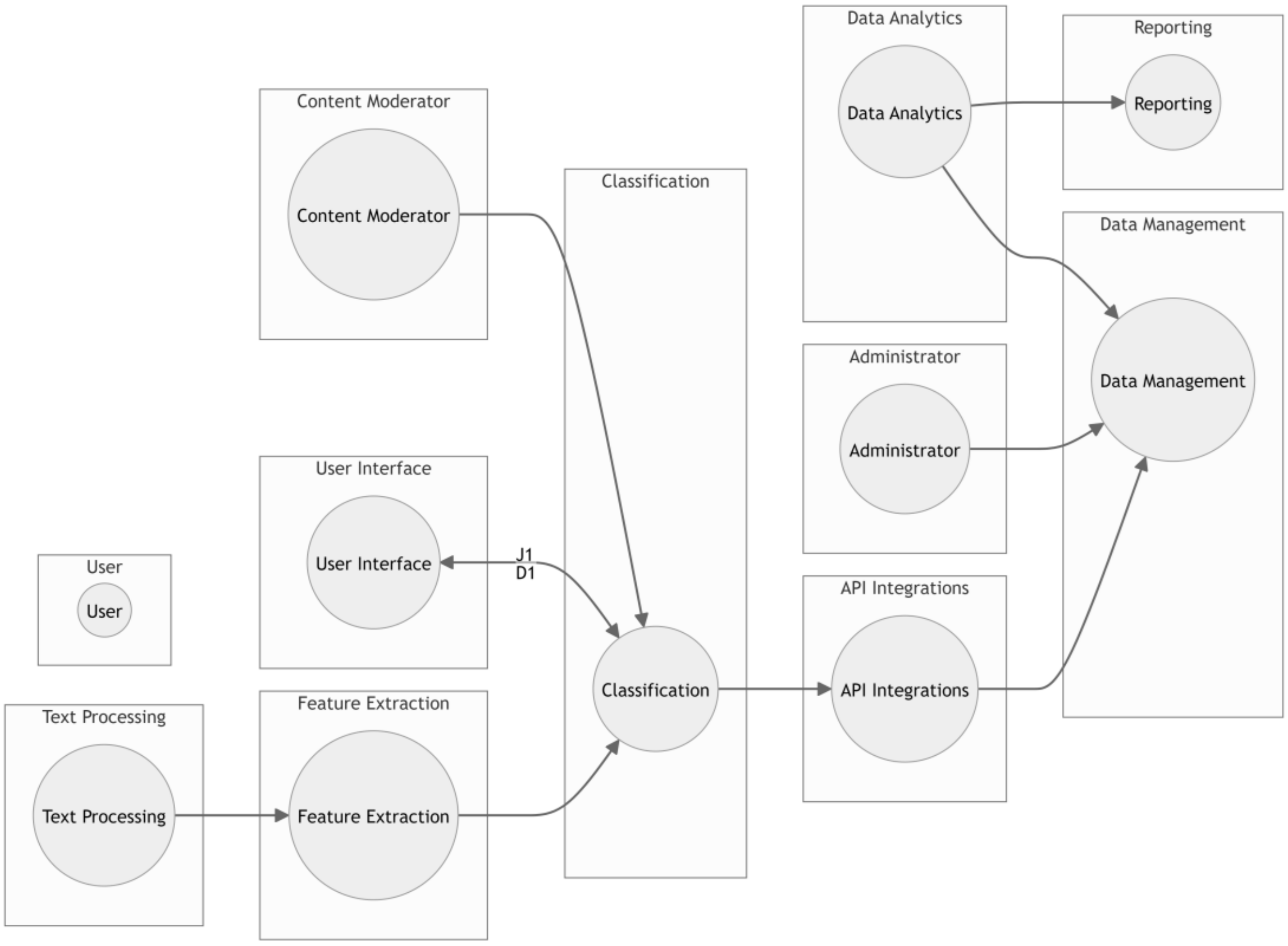
- The model is evaluated using the test data.
- This step helps assess the model’s accuracy in detecting hate speech.

Deployment

- Once a model is determined to be accurate, it can be deployed to real-world applications, such as filtering social media content.

BUSINESS ARCHITECTURE DIAGRAM

EXP.NO: 7 **DATE: 02/05/2024**



Actor:

- User: Represents the primary user who interacts with the XGBoost. This actor is involved in all frontend use cases.

Frontend Use Cases:

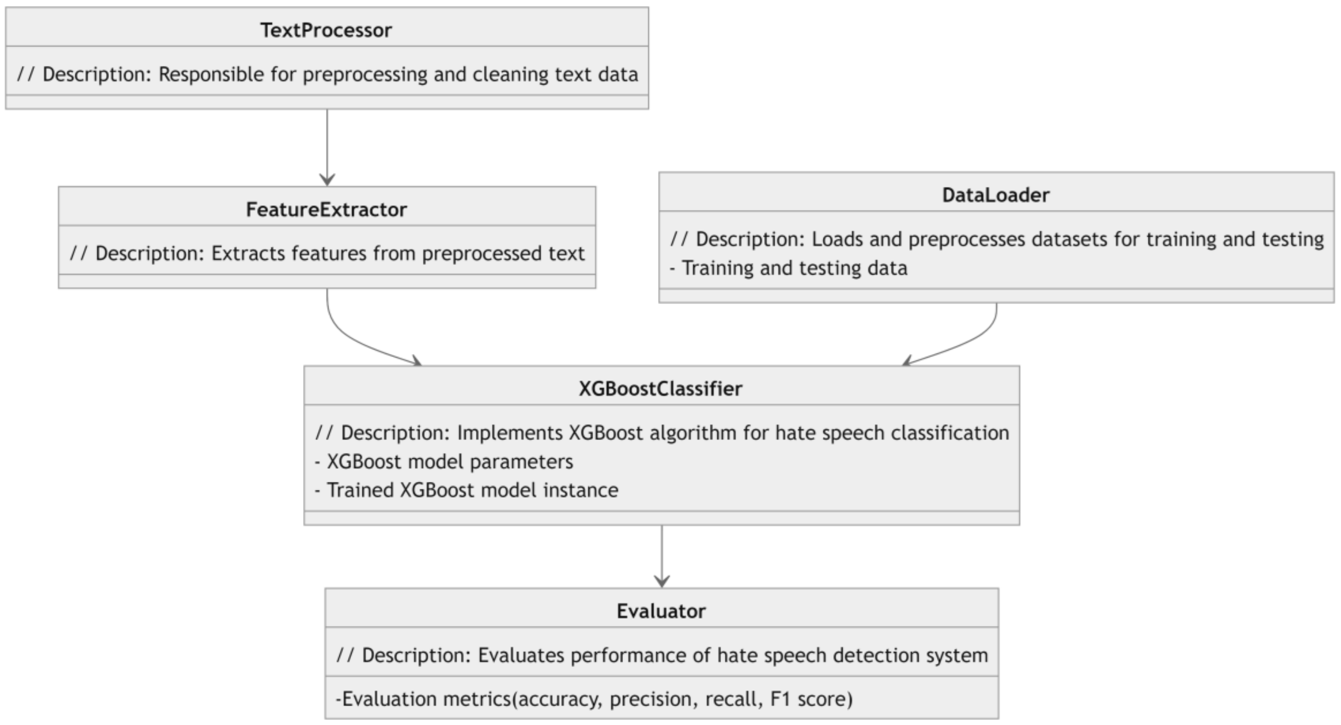
User Interface:
This is the part of the system that users interact with directly. In the diagram, it’s labeled “User Interface” and “User”. It could be a web application, mobile app, or any other interface that allows users to view information or input data.

Backend Use Cases:

- **Data Management:** This refers to the storage and management of data. In the diagram, it’s labeled “Data Management” and “Administrator”. This could include databases, data warehouses, and data lakes.
- **Data Analytics:** This refers to the processes and tools used to analyze data. In the diagram, it’s labeled “Data Analytics” and “Data Analyst”. Data analysts may use SQL queries, data visualization tools, and machine learning to uncover patterns and trends in the data.
- **Reporting:** This refers to the creation of reports that communicate the results of data analysis. In the diagram, it’s labeled “Reporting” . Reports can be presented in various formats, such as tables, charts, and graphs.
- **Content Moderation:** This refers to the process of reviewing and managing user-generated content. In the diagram, it’s labeled “Content Moderator”. Content moderators may use data and reporting tools to identify and remove inappropriate content.
- **API Integrations:** These are connections between the system and other applications. In the diagram, it’s labeled “API Integrations”. They allow data to be exchanged between the system and other software programs.
- **Data Processing:** This refers to the preparation of data for analysis. In the diagram, it’s labeled “Text Processing” and “Feature Extraction”. This may involve techniques such as removing punctuation or converting text to lowercase.
- **Classification:** This refers to the process of categorizing data points into predefined groups. In the diagram, it’s labeled “Classification”. For instance, it could be used to classify text data as positive, negative, or neutral sentiment.

CLASS DIAGRAM

EXP.NO: 8 **DATE: 07/05/2024**


TextProcessor
// Description: Responsible for preprocessing and cleaning text data
FeatureExtractor
// Description: Extracts features from preprocessed text
DataLoader
// Description: Loads and preprocesses datasets for training and testing
- Training and testing data
XGBoostClassifier
// Description: Implements XGBoost algorithm for hate speech classification
- XGBoost model parameters
- Trained XGBoost model instance
Evaluator
// Description: Evaluates performance of hate speech detection system
-Evaluation metrics(accuracy, precision, recall, F1 score)

Classes:

• **Text Processor:** This class is responsible for preprocessing and cleaning text data. Preprocessing may involve removing punctuation and special characters, converting text to lowercase, or stemming/lemmatization (reducing words to their base form).

• **Feature Extractor:** This class extracts features from the preprocessed text. Features are characteristics that can be used to distinguish between hateful and non-hateful language. Examples of features could be word n-grams (sequences of n words), part-of-speech tags, or named entity recognition (identifying people, places, organizations).

• **Data Loader:** This class is responsible for loading and preprocessing data for training and testing the XGBoostClassifier model.

• **XGBoostClassifier:** This class implements the XGBoost algorithm, a type of machine learning model used for classification tasks. The XGBoostClassifier is trained on labeled data (data that has already been classified as hateful or non-hateful) to learn how to identify hate speech.

• **Evaluator:** This class evaluates the performance of the XGBoostClassifier model. It likely calculates metrics such as accuracy, precision, recall, and F1 score to measure how well the model can correctly classify hate speech.

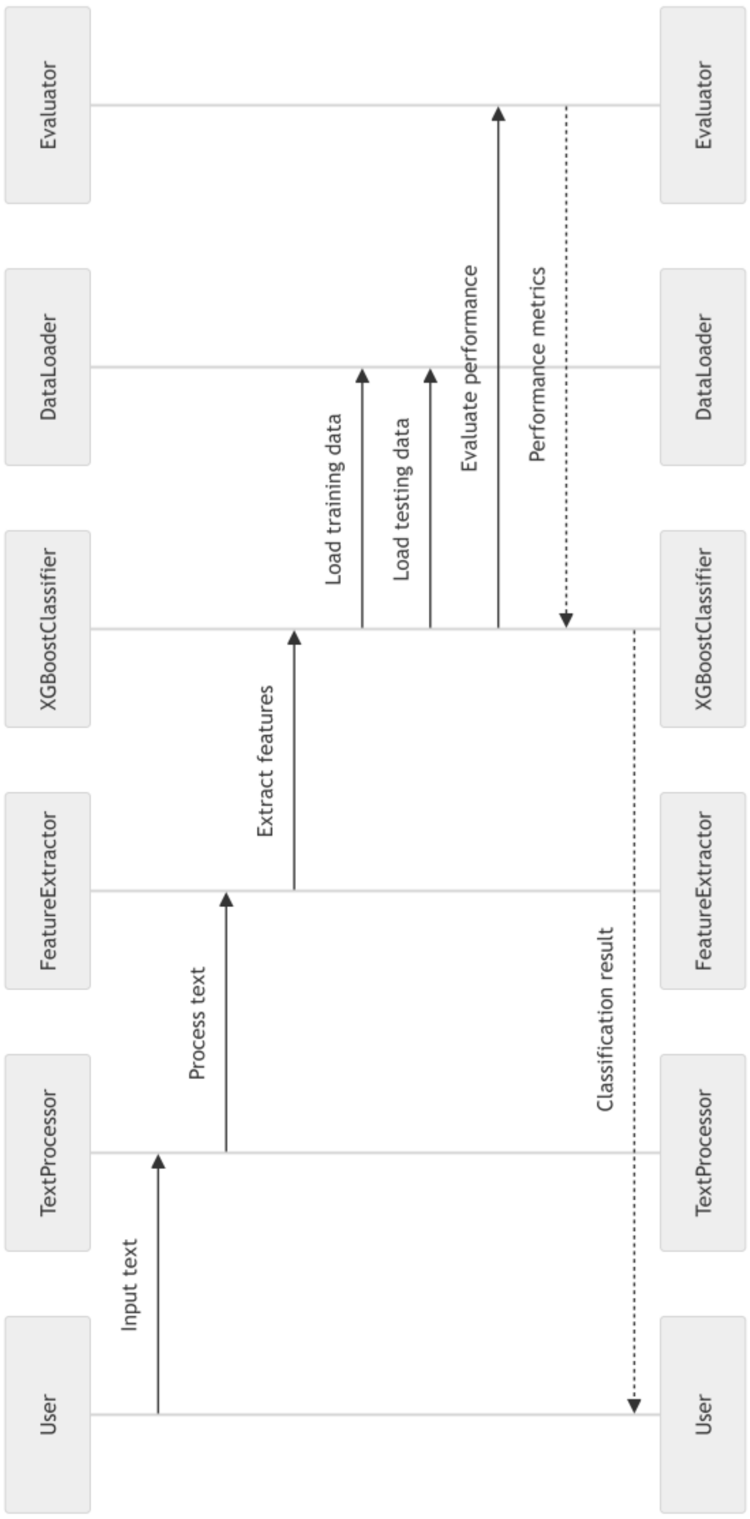
Relationships:

The Text Processor class likely provides preprocessed text data to the Feature Extractor class, which uses it to extract features. The Data Loader class likely provides training and testing data to the XGBoostClassifier class, which it uses to train the model and evaluate its performance. The XGBoostClassifier class likely provides predictions (classifications of text data as hateful or non-hateful) to the Evaluator class, which it uses to calculate performance metrics.

.

SEQUENCE DIAGRAM

EXP.NO: 9 **DATE: 16/05/2024**



Actor:

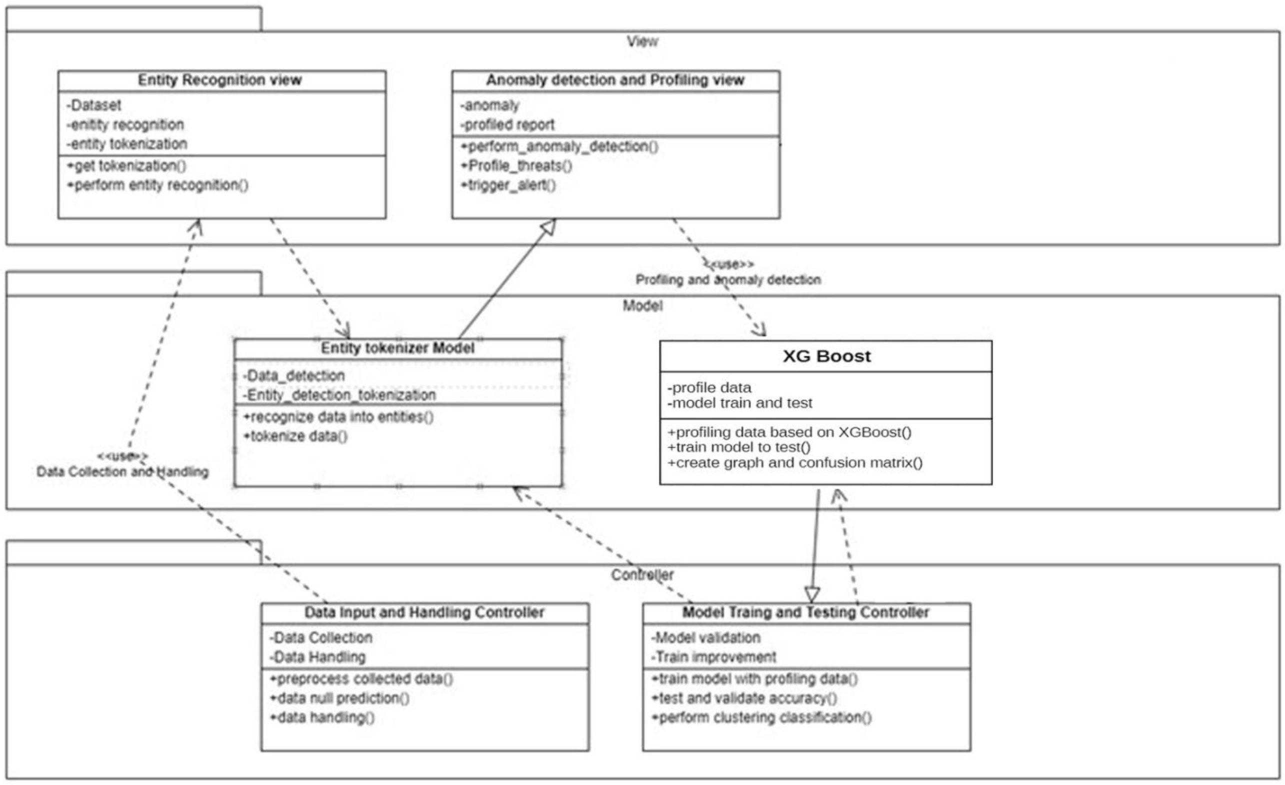
- **User:** User sending an Input Text to the Text Processor.

Sequence of Events:

- The **Text Processor** then performs the following actions:
  - **Processes** the text data. This likely involves preprocessing steps such as removing punctuation or converting text to lowercase.
  - **Extracts features** from the text. Features are characteristics that can be used to distinguish between categories of text. In sentiment analysis, for instance, features could be word n-grams (sequences of words) or part-of-speech tags.
- Once the text is processed and features are extracted, the Text Processor sends the **Processed Text** and **Extracted Features** to the **Feature Extractor** class.
- The **Feature Extractor** may perform additional feature extraction or transformation on the data.
- After processing the features, the Feature Extractor sends the **Extracted Features** to the **XGBoostClassifier** class.
- The **XGBoostClassifier** is a machine learning model, likely trained to classify text data. It uses the features to make a prediction about the text data.
- The prediction, likely a classification label (e.g., hate speech or not hate speech), is then sent back to the **Text Processor**.
- Finally, the **Text Processor** sends the **Classification Result** back to the **User**.

ARCHITECTURAL PATTERN (MVC)

EXP.NO: 10 **DATE: 28/05/2024**


View
Entity Recognition view
-Dataset
-enitity recognition
-entity tokenization
+get tokenization()
+perform entity recognition()
Anomaly detection and Profiling view
-anomaly
-profiled report
+perform_anomaly_detection()
+Profile_threats()
+trigger_alert()
<<use>>
Profiling and anomaly detection
Model
Entity tokenizer Model
-Data_detection
-Entity_detection_tokenization
+recognize data into entities()
+tokenize data()
XG Boost
-profile data
-model train and test
+profiling data based on XGBoost()
+train model to test()
+create graph and confusion matrix()
<<use>>
Data Collection and Handling
Controller
Data Input and Handling Controller
-Data Collection
-Data Handling
+preprocess collected data()
+data null prediction()
+data handling()
Model Traing and Testing Controller
-Model validation
-Train improvement
+train model with profiling data()
+test and validate accuracy()
+perform clustering classification()

Data Collection and Handling

- This section can be loosely considered the Controller in the MVC pattern. It is responsible for:
    - **Data Input and Handling Controller:** Handles data collection from various sources. This could involve user input from a web form or data from an API.
    - **Model Training and Testing Controller:** Handles training and testing of the machine learning model.

Entity Recognition view

- This section can be considered the View in the MVC pattern. It represents the presentation layer that displays the results of entity recognition tasks.

Profiling and anomaly detection

- This section can also be considered the View in the MVC pattern. It represents the presentation layer that displays the results of profiling and anomaly detection tasks.

Model

- This section represents the Model in the MVC pattern. It encapsulates the business logic of the application, including:

    - **Entity Tokenizer Model:** Likely responsible for tokenizing text data into individual words or phrases.

    - **XG Boost Model:** This is the machine learning model used for anomaly detection and profiling.

While the architecture leverages components similar to MVC (Model, View, Controller), they don't strictly adhere to the separation of concerns principles that are a core aspect of MVC. For instance, the Data Collection and Handling section (Controller) seems to have some responsibilities related to training and testing the model, which would typically be handled by the Model in MVC.

Here's a more traditional breakdown of an MVC application:

- **Model:** Represents the data and business logic of the application.
- **View:** Represents the user interface components that display information to the user.
- **Controller:** Handles user interaction, retrieves data from the model, and updates the view accordingly.

Relationships:

The Model, View, and Controller components interact as follows:

- The Model provides data and logic to the Controller.
- The Controller receives user input from the View and manipulates the Modelaccordingly (e.g., adding a new event).
- The Controller instructs the View to update itself based on changes in the Model(e.g., displaying a newly added event).
- The View interacts directly with the Model. It communicates with the Model through the Controller.