A Major Project Report

On

SOCIAL MEDIA FORENSICS: CYBERBULLYING- RELATED HATE SPEECH DETECTION USING VOTING CLASSIFIER ENSEMBLE METHODS

Submitted to CMREC (UGC Autonomous)

In Partial Fulfilment of the requirements for the Award of Degree

of

BACHELOR OF TECHNOLOGY

IN

COMPUTER SCIENCE AND ENGINEERING (AI & ML)

Submitted

By

BH. SAI GOPAL VARMA	**(228R1A66D7)**
BUKKA ROHINI	**(228R1A66E0)**
KOLIPAKA SHASHANK	**(228R1A66F5)**
ADUMULLA MANOJ	**(228R1A66D0)**

Under the Esteemed guidance of

Mr. B. SAI KUMAR

Assistant Professor, Department of CSE(AI&ML)

Department of Computer Science and Engineering(AI&ML)

CMR ENGINEERING COLLEGE

(UGC AUTONOMOUS)

(Accredited by NAAC & NBA, Approved by AICTE, New Delhi, Affiliated to JNTU, Hyderabad)
(Kandlakoya, Medchal Road, Medchal-Malkajgiri Dist., Hyderabad-501 401)

(2025-2026)

CMR ENGINEERING COLLEGE

UGC AUTONOMOUS

(Accredited by NAAC&NBA, Approved by AICTE New Delhi, Affiliated to JNTU, Hyderabad, Kandlakoya, Medchal Road, Hyderabad-501 401)

Department of Computer Science & Engineering (AI & ML)

CERTIFICATE

This is to certify that the Major project entitled **"SOCIAL MEDIA FORENSICS: CYBERBULLYING-RELATED HATE SPEECH DETECTION USING VOTING CLASSIFIER ENSEMBLE METHODS"** is a bonafide work carried out by

BH. SAI GOPAL VARMA	**(228R1A66D7)**
BUKKA ROHINI	**(228R1A66E0)**
KOLIPAKA SHASHANK	**(228R1A66F5)**
ADUMULLA MANOJ	**(228R1A66D0)**

in partial fulfillment of the requirement for the award of the degree of BACHELOR OF TECHNOLOGY in COMPUTER SCIENCE AND ENGINEERING (AI&ML) from CMR Engineering College, under our guidance and supervision.

The results presented in this Major project have been verified and are found to be satisfactory. The results embodied in this Major project have not been submitted to any other university for the award of any other degree or diploma.

Internal Guide	**Major Project Coordinator**	**Head of the Department**
Mr. B. Sai Kumar	Mr. G. Venkateswarlu	Dr. Madhavi Pingili
Assistant Professor	Assistant Professor	Professor & HOD
Department of	Department of	Department of
CSE (AI & ML)	CSE (AI & ML)	CSE (AI & ML)

External Examiner: _____

DECLARATION

This is to certify that the work reported in the present Major project entitled "**SOCIAL MEDIA FORENSICS: CYBERBULYING-RELATED HATE SPEECH DETECTION USING VOTING CLASSIFIER ENSEMBLE METHODS**" is a record of bonafide work done by us in the Department of Computer Science and Engineering (AI & ML), CMR Engineering College. The reports are based on the Major project work done entirely by us and not copied from any other source. We submit our Major project for further development by any interested students who share similar interests to improve the Major project in the future.

The results embodied in this Major project report have not been submitted to any other University or Institute for the award of any degree or diploma to the best of our knowledge and belief.

BH. SAI GOPAL VARMA (228R1A66D7)
BUKKA ROHINI (228R1A66E0)
KOLIPAKA SHASHANK (228R1A66F5)
ADUMULLA MANOJ (228R1A66D0)

ACKNOWLEDGEMENT

We are extremely grateful to **Dr. A. Srinivasula Reddy**, Principal and **Dr. Madhavi Pingili**, Professor & HOD, Department of CSE(AI&ML), CMR Engineering College for their constant support.

We are extremely thankful to **Mr. B. Sai Kumar,** Assistant Professor, Internal Guide, Department of CSE(AI&ML), for his constant guidance, encouragement and moral support throughout the Major project.

We will be failing in duty if we do not acknowledge with grateful thanks to the authors of the references and other literatures referred in this Major project.

We thank **Mr. G. Venkateswarlu,** Major Project Coordinator for his constant support in carrying out the Major project activities and reviews.

We express our thanks to all staff members and friends for all the help and co-ordination extended in bringing out this Major project successfully in time.

Finally, We are very much thankful to our parents who guided us for every step.

BH. SAI GOPAL VARMA (228R1A66D7)
BUKKA ROHINI (228R1A66E0)
KOLIPAKA SHASHANK (228R1A66F5)
ADUMULLA MANOJ (228R1A66D0)

CONTENTS

ABSTRACT

Cyberbullying and hate speech continue to pose significant risks across digital platforms, necessitating reliable and scalable automated detection systems. This project presents a fully implemented ensemble learning framework for classifying harmful online content using Boosted Decision Trees (BoostDT) and Bagging Random Forest (BagRF), effectively leveraging their complementary strengths for improved predictive performance. The system incorporates a structured text preprocessing pipeline, including normalization, noise removal, tokenization, stop-word removal, lemmatization, and TF-IDF-based feature extraction, ensuring consistency, efficiency, and scalability across multiple datasets and varied language patterns. To address class imbalance, the framework is designed to support techniques such as SMOTE for improved class distribution and fairness. A soft-voting ensemble strategy combines probabilistic outputs of individual models, enhancing prediction robustness, stability, and overall classification accuracy. The trained models are deployed through a Django-based backend, where users can submit text inputs, and an admin interface enables effective monitoring of registered users and their prediction records. The overall architecture emphasizes modularity, extensibility, deployment readiness, and seamless integration with real-time content moderation systems.

Keywords: Cyberbullying Detection; Ensemble Framework; Boosted Decision Tree; Bagging Random Forest; Soft Voting; Text Preprocessing; TF-IDF;

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

1.1 Introduction

The rapid expansion of social media platforms has fundamentally reshaped digital communication, enabling users to interact, exchange opinions, and disseminate information at unprecedented scale. However, these platforms have also become major channels for cyberbullying and hate speech, leading to emotional distress, social exclusion, and long-term psychological harm. The increasing volume and complexity of such harmful content present a significant challenge for platform administrators, as manual moderation is neither scalable nor efficient. This has intensified the need for intelligent, automated systems capable of detecting abusive and aggressive language across diverse textual contexts **[17], [19]**.

Advances in machine learning and natural language processing (NLP) provide effective mechanisms for addressing this challenge through automated classification of harmful content **[10], [11]**. Recent studies demonstrate that ensemble-learning approaches where multiple algorithms are combined can deliver more accurate and stable predictions than individual models **[1], [4], [12]**. Building on these developments, this project implements an ensemble-learning framework to identify cyberbullying and hate speech within online communications **[2], [7]**. The proposed system leverages multiple datasets, handles class imbalance, and produces reliable classification outputs suitable for integration with content monitoring or moderation systems. The overall framework is designed to be scalable, extensible, and appropriate for real-world deployment scenarios **[3], [9]**.

1.2 Project Objectives

By the completion of this project, the system demonstrates the following capabilities:

1. A structured and scalable ensemble-learning framework that effectively detects cyberbullying and hate speech across diverse textual datasets.
2. A unified text preprocessing and feature-engineering pipeline that consistently transforms user-generated content into meaningful and machine-readable representations.
3. An integrated system architecture that combines data acquisition, preprocessing, feature extraction, model training, and decision-making modules, enabling seamless use within automated content- moderation environments.

1.3 Purpose of the Project

The purpose of this project is to develop an intelligent and structured framework capable of accurately identifying cyberbullying and hate speech in online text. The system implements a well-defined preprocessing and analysis pipeline that enables reliable categorization of harmful content using machine learning and natural language processing techniques. Such automated detection frameworks assist content moderation systems by identifying abusive language at scale and improving the safety of online communities **[3]**. The proposed approach supports integration with automated moderation solutions aimed at improving user safety across digital platforms and ensuring more responsible digital communication environments.

1.4 Problem Statement

The rapid growth of social media usage has resulted in rising incidents of cyberbullying and hate speech, posing significant challenges to user safety and the quality of online interactions **[17]**. The highly unstructured and dynamic nature of online text marked by slang, abbreviations, sarcasm, and continuously evolving offensive expressions makes manual moderation impractical. At the same time, many existing automated approaches that rely on rule-based filters or single-model classifiers struggle to handle linguistic complexity, imbalanced datasets, and overlapping categories of abusive behavior **[12]**. These shortcomings reduce detection accuracy, increase misclassification rates, and limit scalability in real-world environments. Consequently, there is a need for a structured framework that delivers reliable cyberbullying detection supported by unified preprocessing, effective feature representation, and ensemble-based decision making to overcome the constraints of current systems **[1]**.

1.5 Existing System

Existing approaches to cyberbullying and hate speech detection largely rely on rule-based filtering, traditional machine learning techniques, and standalone deep learning models **[17]**. Rule-based systems depend on predefined keyword lists and manually constructed patterns to identify offensive content. Conventional classifiers such as Support Vector Machines, Naïve Bayes, and Logistic Regression typically operate on Bag-of-Words or TF-IDF representations of text **[12]**.

More recent research has explored deep learning architectures, including LSTM, CNN, and transformer- based pretrained models such as BERT, to capture richer contextual meaning. While these approaches improve upon manual moderation, they continue to face limitations in handling diverse linguistic styles, rapidly evolving abusive expressions, and highly imbalanced datasets commonly observed on social media platforms **[8], [18].**

Disadvantages

- Limited capability to recognize sarcasm, implicit abuse, and context-dependent expressions.
- Keyword- and rule-based filters become ineffective when users mask offensive language through altered spellings, symbols, or creative phrasing.
- Single-model classifiers frequently deliver unstable performance when applied to noisy or imbalanced datasets.
- Conventional machine learning techniques depend heavily on manual feature engineering, which restricts scalability and adaptability.
- Deep learning approaches require large, well-annotated datasets that are often difficult and expensive to obtain.

1.6 Proposed System

The developed system presents a ensemble-learning framework designed to improve the detection of cyberbullying and hate speech across varied textual content. It integrates two complementary machine-learning models Boosted Decision Tree (BoostDT) and Bagging Random Forest (BagRF) within a soft-voting ensemble, producing more stable, consistent, and context-aware classification outcomes. A unified preprocessing pipeline is implemented to perform text normalization, noise removal, tokenization, stop-word elimination, lemmatization, and TF-IDF-based feature extraction. This ensures that raw text is transformed into clean and meaningful representations suitable for model learning. To address class imbalance commonly observed in cyberbullying datasets, the system applies synthetic data balancing techniques such as SMOTE **[8],** improving fairness and representation across categories.

Overall, the framework is designed to be scalable, modular, and reliable, enabling seamless integration with real-time content-moderation platforms and automated cyber-safety applications. The proposed system is illustrated in the block diagram below (Fig. 1.6.1).

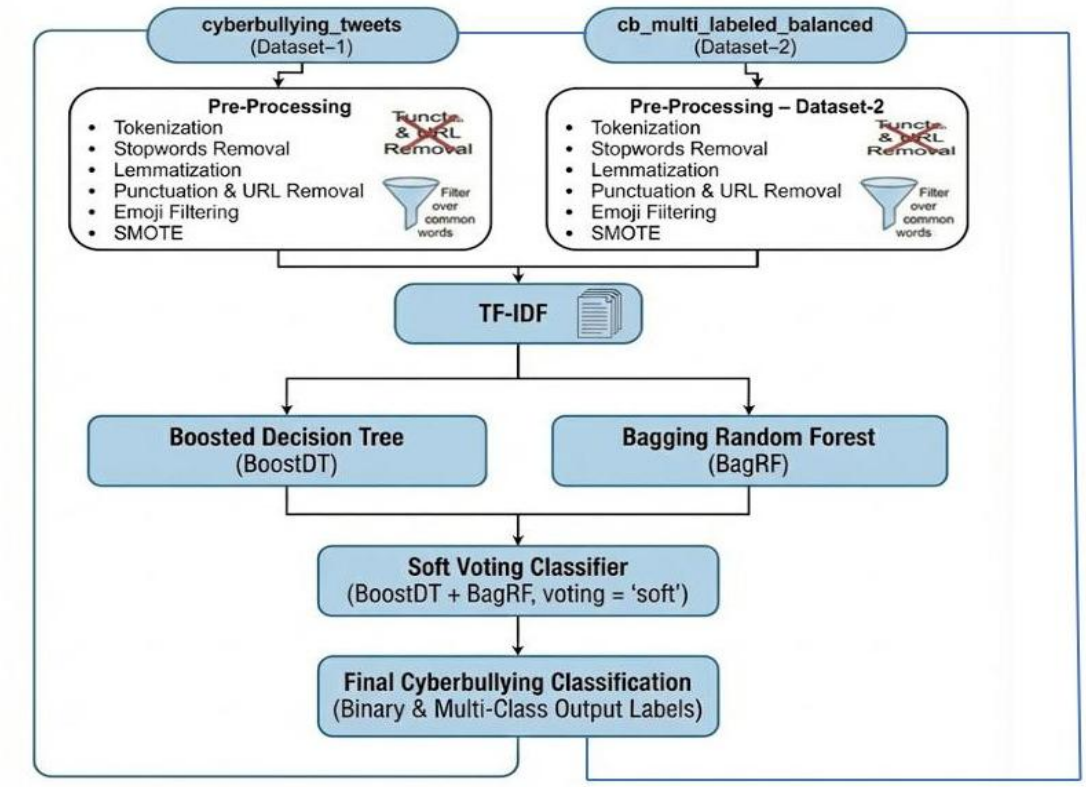


Fig. 1.6.1: Block diagram of proposed system.

Advantages

- Enhanced reliability through ensemble-based decision making rather than dependence on a single classifier.
- Improved handling of noisy, informal, and diverse linguistic patterns through a unified preprocessing pipeline.
- Stronger performance on imbalanced datasets as a result of integrated class-balancing strategies.
- A scalable architecture that can be extended to multi-class and multi-label abusive-content detection scenarios.

1.7 Input and Output Design

1.7.1 Input Design

The input module defines the structure, format, and flow of textual data processed by the cyberbullying and hate speech detection system. Because the system operates on user-generated content sourced from online platforms, the input layer is engineered to effectively handle diverse linguistic styles, informal expressions, and noisy text. The system accepts raw text sentences or posts as primary input, which may contain symbols, emojis, URLs, hashtags, abbreviations, and variations of abusive or aggressive language.

To ensure consistency and readiness for analysis, the input pipeline performs a series of transformations before data proceeds to feature extraction. These transformations include text normalization, removal of irrelevant characters, structured token handling, and preparation for TF-IDF representation. The system supports both multi-class and multi-label formats, allowing it to process labeled data during training and unlabeled text during prediction.

By defining clear formatting rules and a streamlined data flow, the input module enhances reliability, minimizes ambiguity, and enables efficient operation of subsequent preprocessing and analytical components.

Objectives

- A consistent and structured format has been established for capturing raw textual data, ensuring that all inputs are correctly received and prepared for preprocessing.
- The input pipeline effectively filters noise, ambiguity, and irrelevant elements, supporting a smooth transition into tokenization, feature extraction, and subsequent analytical stages.
- The system supports both multi-class and multi-label text inputs, enabling classification across diverse categories of cyberbullying and hate speech content.

1.7.2 Output Design

The output module defines the structure, format, and presentation of the results generated by the cyberbullying and hate speech detection system. Because the system performs text classification using an ensemble-learning framework, the outputs are designed to be clear, interpretable, and suitable for use by end users or downstream moderation tools. The primary output consists of the predicted cyberbullying or hate speech category associated with a given text input. In multi-label scenarios, the system can display more than one category when the content contains overlapping abusive expressions.

To improve usability, the outputs are presented in a structured and easily understandable format. Each prediction is mapped to a corresponding label and descriptive category name, and when required, can also include severity indicators aligned with moderation workflows. The system additionally supports confidence values or probability scores, allowing administrators and analysts to interpret results more effectively during evaluation and decision making.

By defining a consistent output structure, the system ensures clear communication of classification results, seamless integration with moderation dashboards, and reliable support for automated decision processes.

2. LITERATURE SURVEY

1. **J. Immanuel D., N. S. Nivetha, M. S. Sharmitha, T. K. Sharma, and M. Gowthamani, "Real-Time Cyberbullying Detection Using NLP and Ensemble Learning Techniques,"** ***in Proc. 7th Int. Conf. Mobile Computing and Sustainable Informatics (ICMCSI), IEEE*****, 2026.** This study proposes a real-time cyberbullying detection system using NLP and supervised machine learning models. It emphasizes preprocessing of noisy social media text and effective feature extraction. The Random Forest model achieves the best performance, demonstrating strong accuracy and generalization.

2. **R. Debnath and P. Banerjee, "Soft-Voting Ensemble Strategies for Toxic Comment Classification,"** ***IEEE Transactions on Computational Social Systems*****, 2025.** The authors introduce a soft-voting ensemble technique that combines probabilistic outputs of multiple classifiers. This method improves prediction reliability and balances precision–recall trade-offs. Experimental results show significant improvements in F1-score and classification performance.

3. **A. N. Kumar, S. Patel, and R. Mishra, "Boosting-Based Approaches for Online Abuse and Cyberbullying Detection,"** ***Applied Intelligence*****, 2025**. This work focuses on boosting algorithms such as AdaBoost and Gradient Boosting to enhance cyberbullying detection. By iteratively improving weak learners, the model captures subtle abusive language patterns. The approach achieves higher accuracy and reduced misclassification rates.

4. **I. Uddin, F. Rahman, and T. Kabir, "Voting Classifiers for Improving Toxic Speech Prediction Accuracy,"** ***Computers & Electrical Engineering*****, 2025.** The study evaluates hard and soft voting ensemble methods for toxic speech detection. It highlights the importance of classifier diversity in improving generalization and stability. The proposed ensemble outperforms individual classifiers across multiple metrics.

5. **H. Allwaibed, A. Alotaibi, and A. Alzahrani, "Cyberbullying Detection Approaches for Arabic Texts Using Ensemble Models,"** ***Frontiers in Artificial Intelligence*****, 2025.** This research addresses cyberbullying detection in Arabic text using ensemble learning techniques. It integrates language-specific preprocessing with multiple classifiers to improve performance. The study emphasizes handling linguistic diversity in NLP tasks.

6. **Edem Suresh Babu and G. S. Sravanthi, "Sarcasm Detection of Feature Augmentation Using Multiobjective Genetic Algorithm," *Evolution in Computational Intelligence*, SIST, vol. 436, pp. 457–466, 2025.** The paper proposes a sarcasm detection approach using feature augmentation optimized through a multiobjective genetic algorithm. The method selects optimal feature subsets to improve classification accuracy and reduce redundancy. The approach enhances the detection of implicit and context-aware text patterns.

7. **J. Morales, D. Santos, and P. Castillo, "Ensemble Modeling for Hate Speech Identification Across Platforms," *IEEE Access*, 2024.** This study presents an ensemble-based approach for detecting hate speech across multiple platforms. It leverages cross-platform data to improve generalization and adaptability. The results show consistent performance despite dataset variations.

8. **T. Saha and R. Kar, "Random-Forest-Driven Cyberbullying Detection on Microblog Data," *Journal of Information Security and Applications*, 2024.** The authors utilize Random Forest models to detect cyberbullying in microblog datasets. The method effectively handles high-dimensional features such as TF-IDF. Experimental results show strong precision, recall, and resistance to overfitting.

9. **L. Chen, Y. Wang, and Z. Zhao, "A Multi-Stage Ensemble Architecture for Detecting Abusive Language," *Information Sciences*, 2024.** This work introduces a multi-stage ensemble architecture where predictions are refined across sequential stages. Each stage enhances feature extraction and classification accuracy. The model significantly improves detection of complex abusive patterns.

10. **S. Sihab-Us-Sakib, M. Rahman, and M. Hasan, "Cyberbullying Detection of Resource-Constrained Language Using Transformer-Based Ensembles," *Natural Language Processing Journal*, 2024.** This study explores transformer-based ensemble models for low-resource languages. It leverages contextual embeddings and model aggregation to capture semantic nuances. The approach achieves superior performance compared to traditional machine learning methods.

Focused Area / Title	**Key Findings**	**Reference**
Real-Time Cyberbullying Detection using NLP and Ensemble Learning **[1]**	Proposes a real-time NLP-based cyberbullying detection system with effective preprocessing and feature extraction. Random Forest achieves the best performance with high accuracy and generalization.	J. Immanuel D., N. S. Nivetha, M. S. Sharmitha, T. K. Sharma, and M. Gowthamani, Proc. IEEE ICMCSI, 2026.
Soft-Voting Ensemble for Toxic Comment Classification **[2]**	Introduces a soft-voting ensemble approach that aggregates probabilistic outputs of classifiers. Improves prediction reliability and achieves higher F1-score and balanced precision–recall performance.	R. Debnath and P. Banerjee, "Soft-Voting Ensemble Strategies for Toxic Comment Classification," IEEE Transactions on Computational Social Systems, 2025.
Boosting-Based Cyberbullying Detection **[3]**	Utilizes boosting algorithms such as AdaBoost and Gradient Boosting to improve detection of abusive content. Enhances classification accuracy by capturing subtle linguistic patterns in text data.	A. N. Kumar, S. Patel, and R. Mishra, "Boosting-Based Approaches for Online Abuse and Cyberbullying Detection," Applied Intelligence, 2025.
Voting Classifiers for Toxic Speech Detection **[4]**	Evaluates hard and soft voting ensemble techniques for toxic speech classification. Demonstrates improved generalization and stability compared to individual models.	I. Uddin, F. Rahman, and T. Kabir, "Voting Classifiers for Improving Toxic Speech Prediction Accuracy," Computers & Electrical Engineering, 2025.
Ensemble Models for Arabic Cyberbullying Detection **[5]**	Focuses on detecting cyberbullying in Arabic text using ensemble learning. Highlights importance of language-specific preprocessing and feature engineering for improved performance.	H. Allwaibed, A. Alotaibi, and A. Alzahrani, "Cyberbullying Detection Approaches for Arabic Texts Using Ensemble Models," Frontiers in Artificial Intelligence, 2025.

Table no. 2 Literature Review Summary

Focused Area / Title	**Key Findings**	**Reference**
Sarcasm Detection using Feature Augmentation with Multiobjective Genetic Algorithm **[6]**	Proposes a sarcasm detection approach using feature augmentation optimized through a multiobjective genetic algorithm. Improves classification accuracy by selecting optimal feature subsets and capturing implicit, context-aware text patterns.	Edem Suresh Babu and G. S. Sravanthi, "Sarcasm Detection of Feature Augmentation Using Multiobjective Genetic Algorithm," Evolution in Computational Intelligence, SIST, vol. 436, pp. 457–466, 2025.
Ensemble Modeling for Hate Speech Detection Across Platforms **[7]**	Develops an ensemble approach for cross-platform hate speech detection. Improves adaptability and maintains consistent performance across diverse datasets.	J. Morales, D. Santos, and P. Castillo, "Ensemble Modeling for Hate Speech Identification Across Platforms," IEEE Access, 2024.
Random Forest-Based Cyberbullying Detection **[8]**	Applies Random Forest algorithms for detecting cyberbullying in microblog data. Effectively handles high-dimensional TF-IDF features and reduces overfitting.	T. Saha and R. Kar, "Random-Forest-Driven Cyberbullying Detection on Microblog Data," Journal of Information Security and Applications, 2024.
Multi-Stage Ensemble Architecture for Abusive Language Detection **[9]**	Introduces a multi-stage ensemble system that refines predictions through sequential learning. Enhances feature extraction and improves detection accuracy for complex language patterns.	L. Chen, Y. Wang, and Z. Zhao, "A Multi-Stage Ensemble Architecture for Detecting Abusive Language," Information Sciences, 2024.
Transformer-Based Ensemble for Low-Resource Languages **[10]**	Explores transformer-based ensemble models for cyberbullying detection in resource-constrained languages. Improves contextual understanding and classification performance.	S. Sihab-Us-Sakib, M. Rahman, and M. Hasan, "Cyberbullying Detection of Resource-Constrained Language Using Transformer-Based Ensembles," Natural Language Processing Journal, 2024.

Table no. 2 Literature Review Summary

3. SOFTWARE REQUREIMENTS ANALYSIS

3.1 Modules and Their Functionalities

3.1.1 Data Analysis

Data analysis was conducted to examine the structure, composition, and characteristics of the datasets used for cyberbullying and hate speech detection. The datasets consist of user-generated text that varies considerably in length, vocabulary, and writing style, frequently containing informal language, abbreviations, emojis, and context-dependent expressions. Exploratory analysis identified multiple categories of abusive behavior, along with a pronounced class imbalance in which some categories occur far less frequently than others.

The data also contains noise in the form of misspellings, elongated words, hyperlinks, and special symbols, all of which require systematic preprocessing. These observations guided the development of the preprocessing pipeline, the selection of TF-IDF-based feature extraction, and the design of the ensemble classification framework. Overall, understanding dataset characteristics ensured that the implemented system could effectively manage linguistic variability and skewed class distributions.

3.1.2 Data Preprocessing

Data preprocessing is performed to transform raw and unstructured text into a clean, standardized format suitable for analysis and model training. Noise commonly present in social media content such as URLs,emojis, special characters, and repeated symbols is removed, and the text is normalized through lowercasing. Tokenization is then applied to divide sentences into meaningful units, followed by stop-word removal and lemmatization to eliminate redundancy and ensure consistent linguistic representation. The resulting processed text is subsequently prepared for feature extraction, establishing a reliable foundation for accurate classification in later stages.

3.1.3 Machine Learning Algorithm for Prediction

The system employs an ensemble-based machine learning strategy to improve the accuracy and reliability of cyberbullying detection. Two complementary classifiers Boosted Decision Tree (BoostDT) and Bagging Random Forest (BagRF) are implemented due to their ability to model complex textual relationships and reduce prediction variance.

Each model operates independently, and their probabilistic outputs are integrated using a soft-voting mechanism to produce the final classification outcome. This ensemble approach results in more stable decisions and improves the system's capability to process diverse linguistic inputs and overlapping abuse categories. In addition, the framework supports both multi- class and multi-label prediction scenarios, making it suitable for integration within different content- moderation environments.

3.2 Functional Requirements

The functional requirements describe the core operations that the cyberbullying detection system performs to fulfill its intended objectives. They specify how the system receives and validates user input, processes textual data through preprocessing and feature extraction, and produces meaningful classification results. These requirements ensure that the system operates consistently and reliably, while also supporting integration with external moderation tools and real-time applications. The following points summarize the primary functional capabilities implemented within the framework.

- The system shall accept raw textual input and route it to the preprocessing module.
- The system shall perform text cleaning, normalization, tokenization, and feature extraction to prepare data for classification.
- The system shall classify processed text into appropriate cyberbullying and/or hate speech categories using the ensemble model.
- The system shall present the classification results in a clear, structured, and interpretable output format.

3.3 Non-Functional Requirements

Non-functional requirements describe the quality attributes and performance expectations that govern how the system operates. Rather than defining specific features, they specify how the system should behave under various conditions and how efficiently it should deliver results. These requirements ensure reliability, usability, scalability, and overall consistency throughout the system's operation and future maintenance. The following points summarize the key non-functional characteristics implemented in the system.

- The system shall ensure high reliability and stability during all stages of text processing and classification.
- The system shall support scalable performance as dataset size and category complexity increase.

- The system shall maintain efficient processing with minimal latency in generating predictions.
- The system shall preserve data privacy and confidentiality for all user-generated inputs processed by the framework.

3.4 Feasibility Study

The feasibility study assesses whether the cyberbullying detection system can be realistically developed, implemented, and operated based on technical, operational, and economic considerations. Analysis confirms that the required preprocessing techniques, ensemble-learning models, and datasets are readily available and compatible with current computational environments. The system architecture is scalable, cost-effective, and capable of integrating with existing moderation workflows. Overall, the evaluation indicates that development and deployment of the system are practical and feasible.

1.Economic Feasibility

2.Technical Feasibility

3. Social Feasibility

3.4.1 Economic Feasibility

Economic feasibility evaluates whether the system can be developed and sustained within reasonable cost constraints. The implementation utilizes open-source machine learning libraries, publicly available datasets, and standard computing infrastructure, which significantly minimizes development and maintenance expenses. Because the system does not depend on specialized hardware or proprietary tools, the overall cost remains low. As a result, the solution is economically viable for academic use, research environments, and potential organizational adoption.

3.4.2 Technical Feasibility

Technical feasibility examines the availability of tools, technologies, and required expertise necessary to implement the system. The framework is developed using well-established preprocessing techniques, ensemble-learning algorithms, and widely supported platforms such as Python and scikit-learn. These technologies are mature, accessible, and compatible with standard hardware configurations. In addition, extensive documentation, strong community support, and a modular implementation structure further simplify development and maintenance. Overall, the evaluation confirms that the system is technically practical and achievable.

3.4.3 Social Feasibility

Social feasibility evaluates how the system is likely to be perceived and accepted by users, administrators, and other stakeholders in real-world environments. Because cyberbullying and hate speech pose serious social and psychological risks, an automated detection solution is expected to receive strong acceptance and support. Users benefit from safer digital interactions, while organizations gain a reliable mechanism for maintaining platform integrity and enforcing community standards. The system aligns with growing public expectations for responsible online behavior and proactive moderation, indicating a high likelihood of positive social adoption.

4. SYSTEM SPECIFICATIONS

4.1 Software Requirements

The software requirements specify the core tools and platforms necessary to develop and operate the cyberbullying detection system. The implementation relies on a stable programming environment, standard natural language processing libraries, and general-purpose utilities that support preprocessing, analysis, model training, and evaluation. These tools provide a consistent development workflow, ensure compatibility across environments, and enable straightforward scalability as system capabilities expand.

- Operating System: Windows / Linux / macOS
- Programming Language: Python 3.x
- Core Libraries: NLTK, Scikit-learn, NumPy, Pandas
- Development Environment: VS Code / PyCharm / Jupyter Notebook
- Documentation Tools: MS Word / LaTeX

4.2 Hardware Requirements

The hardware requirements define the minimum computational resources necessary for processing datasets, executing preprocessing tasks, and training machine-learning models. The system operates efficiently on standard computing hardware, including a multi-core processor, sufficient RAM for dataset handling, and moderate storage capacity for model files and datasets. The configuration remains scalable, allowing additional resources to be incorporated if the system is extended to larger datasets or real-time deployment scenarios.

- Processor: Intel Core i3/i5 or equivalent
- Memory (RAM): Minimum 8 GB
- Storage: 250 GB HDD/SSD
- Display: Standard 14" or higher
- Optional: Internet connectivity for dataset access and future cloud integration.

5. SOFTWARE DESIGN

5.1 System Architecture

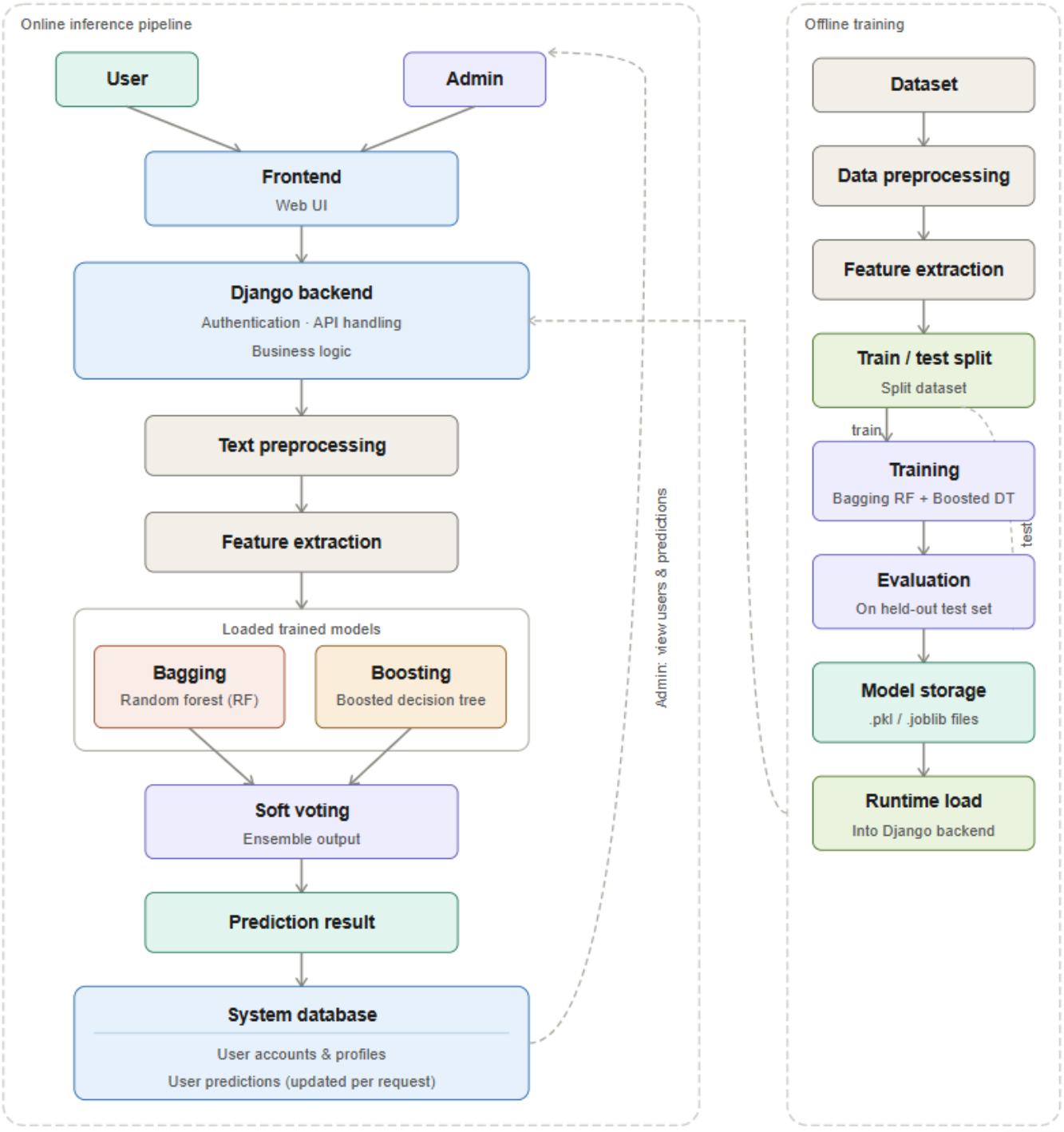


Fig. 5.1: System Architecture

The system architecture, depicted in Fig. 5.1, is structured into two major components: an online inference pipeline and an offline training pipeline. In the online layer, both users and administrators interact with the system through Django templates, eliminating the need for a separate frontend framework. The Django backend manages request handling, authentication, session control, and business logic within a unified framework. Users can submit textual inputs for analysis, while the admin has privileged access to monitor registered users and their prediction history. This tightly integrated architecture simplifies deployment and ensures efficient server-side rendering and control.

In the inference workflow, user input text is processed through a well-defined natural language processing pipeline. The preprocessing stage includes normalization, noise removal, tokenization, stop-word removal, and lemmatization to clean and standardize the input. This is followed by TF-IDF-based feature extraction to convert text into numerical representations suitable for machine learning models. The extracted features are then passed to preloaded models Bagging Random Forest and Boosted Decision Tree which generate prediction probabilities. These outputs are combined using a soft-voting ensemble technique to produce a final classification result, which is displayed to the user and stored in the system database.

The offline training pipeline is responsible for building and maintaining the predictive models. It begins with dataset collection, followed by the same preprocessing and feature extraction steps to ensure consistency with the inference stage. The dataset is split into training and testing subsets for proper evaluation. Both models are trained and validated using the test set to assess performance. Once trained, the models are serialized into .pkl or .joblib files and integrated into the Django backend for runtime use. This modular design ensures extensibility, allowing future updates, retraining, or integration of additional models without affecting the deployed system.

5.2 Dataflow Diagram

The Level-1 Data Flow Diagram represents the overall functioning of the ensemble classification system by illustrating interactions between external entities, processes, and the database. Fig 5.2 diagram highlights that the two primary entities are the User and the Admin, both of whom interact through the User Authentication process. The user provides login credentials and receives authentication status, while the admin validates requests and accesses the user list.

Once authenticated, the user submits raw text input, which flows through the core processing stages. The Text Preprocessing module cleans and standardizes the input, followed by Feature Extraction that converts the text into numerical feature vectors. These vectors are then passed to the Ensemble Classification module, where Bagging Random Forest and Boosted Decision Tree models generate predictions.

The prediction results are stored in the Application Database along with user information. The Result Display and Management module retrieves stored predictions and presents them to the user. Additionally, the admin can view stored predictions directly from the database, enabling monitoring and system oversight.

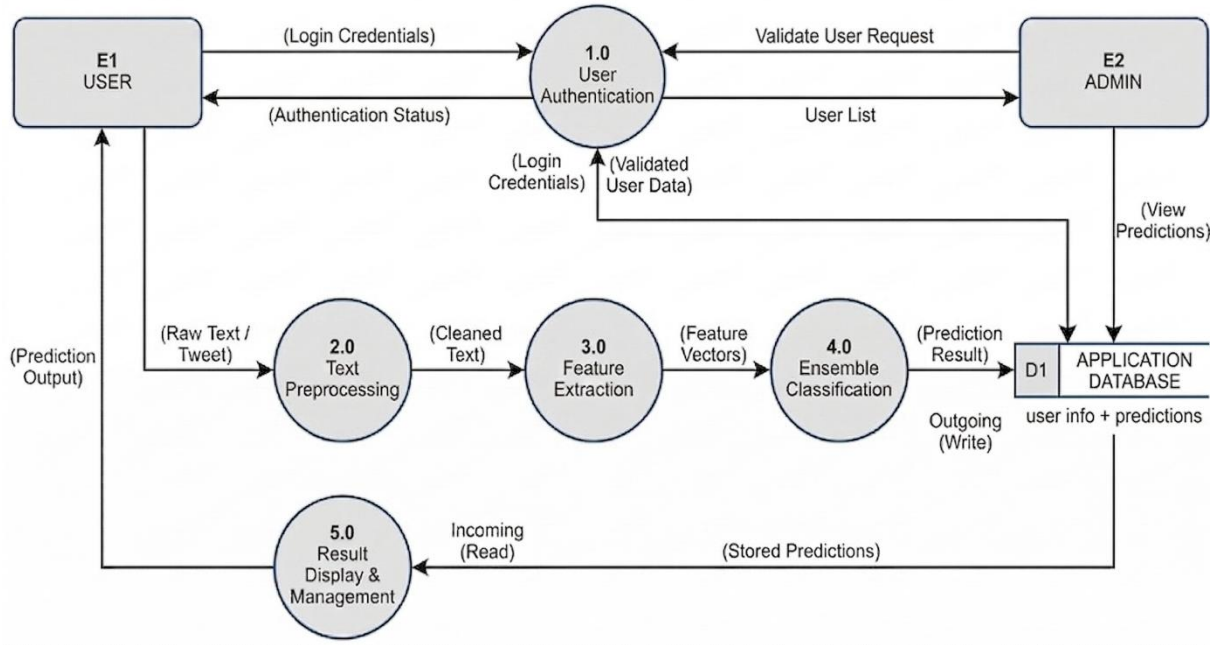


Fig 5.2 Dataflow Diagram

5.3 UML Diagrams

UML (Unified Modeling Language) is a standardized language used for specifying, visualizing, constructing, and documenting the artifacts of software systems. UML helps in representing the design of systems and understanding their components. Created by the Object Management Group (OMG), UML 1.0 was proposed in January 1997. UML is closely associated with object-oriented analysis and design. The two main categories of UML diagrams are Behavioral and Structural diagrams, each serving distinct purposes in the modeling process.

The Behavioral UML diagrams describe the behavior of the system, its actors, and the interaction between the components. On the other hand, Structural UML diagrams depict the static structure of the system, showing its components and relationships. UML has been integrated as a standard by OMG, and its primary goals are to provide a formal basis for understanding modeling languages, offer a ready-to- use expressive language for system developers, and encourage the growth of object-oriented tools.

Goals of UML:

- To provide a standard visual representation of the system design.
- To simplify understanding of system architecture for developers and reviewers.
- To improve communication among team members during development.
- To model both structural and behavioral aspects of the system.
- To support object-oriented design and development practices.
- To document the system clearly for future reference and maintenance.

Types of UML Diagrams:

1. Sequence Diagram:

2. Use Case Diagram:

3. Activity Diagram:

4. Class Diagram:

5.3.1. Sequence Diagram

The sequence diagram illustrates the end-to-end workflow of the cyberbullying detection system, covering both offline training and online inference processes. In the training phase, the system loads and preprocesses the dataset, applies the ensemble voting classifier combining Bagging Random Forest and Boosted Decision Tree, and evaluates performance using metrics such as accuracy, precision, recall, and F1-score. Once validated, the trained model is saved to disk as a persistent artifact. This phase is executed offline and ensures that the model is optimized and ready for deployment without affecting runtime performance.

In the online inference phase, the user interacts with the system by registering and logging in, with all credentials managed through the database. After authentication, the user provides input text, which triggers the system to load the saved model and perform prediction. The model analyzes the input and classifies it into the appropriate cyberbullying category. The prediction result is then returned to the user, completing the interaction cycle. This separation of training and inference enhances system efficiency, scalability, and real-time responsiveness. The overall process is represented in Fig. 5.3.1.

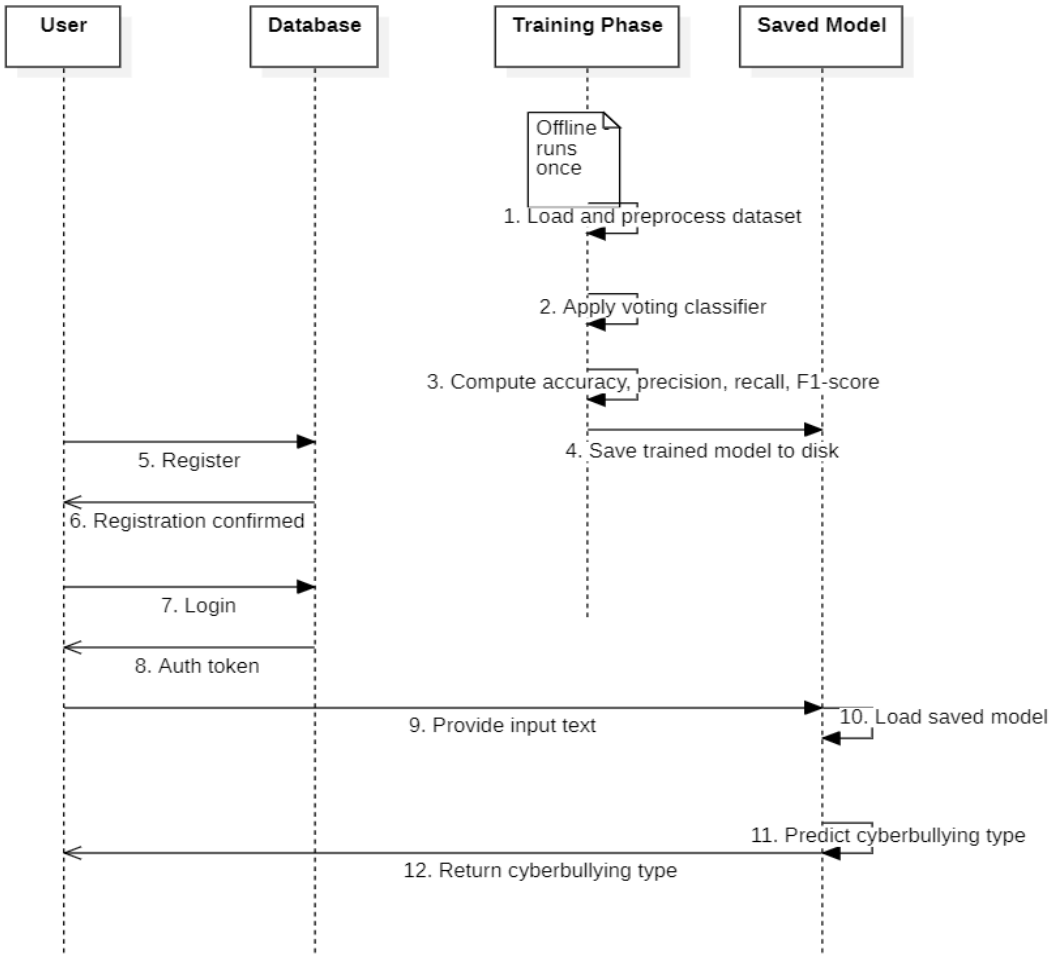


Fig. 5.3.1: Sequence Diagram

List of actions

• User:

The user interacts with the system by registering and logging in through the interface. After successful authentication, the user provides input text for cyberbullying detection. The user finally receives the predicted classification result generated by the system.

• Database:

The user interacts with the system by registering and logging in through the interface. After successful authentication, the user provides input text for cyberbullying detection. The user finally receives the predicted classification result generated by the system.

• Training Phase (Offline):

The training phase runs offline, where the dataset is loaded and preprocessed. The ensemble voting classifier (Bagging Random Forest and Boosted Decision Tree) is applied, and performance metrics such as accuracy, precision, recall, and F1-score are computed. The trained model is then saved to disk for future use.

• Saved Model:

During runtime, the saved model is loaded when the user provides input text. The model processes the input and predicts the cyberbullying category. The prediction result is then returned to the user, completing the inference workflow.

5.3.2 Use Case Diagram

The use case diagram illustrates the overall functionality of the soft-voting based cyberbullying detection framework by clearly distinguishing between the roles of the User and the Analyst. The user interacts with the system through registration and login, which are connected to the database for storing and validating credentials. After authentication, the user can directly access the prediction functionality, where input text is analyzed using a pretrained model to classify cyberbullying content. This represents the inference phase of the system, ensuring efficient and real-time prediction without involving the training process.

The Analyst is responsible for the training and evaluation pipeline of the system. This includes collecting social media data, preprocessing the text, and applying the voting classifier that combines Bagging Random Forest and Boosted Decision Tree models. The processed data is then used to evaluate model performance using standard metrics such as accuracy, precision, recall, and F1-score.

Finally, the analyst analyzes the results to assess model effectiveness and identify areas for improvement. This separation of training and inference ensures modularity, scalability, and clarity in system design. The overall process is represented in Fig. 5.3.2.

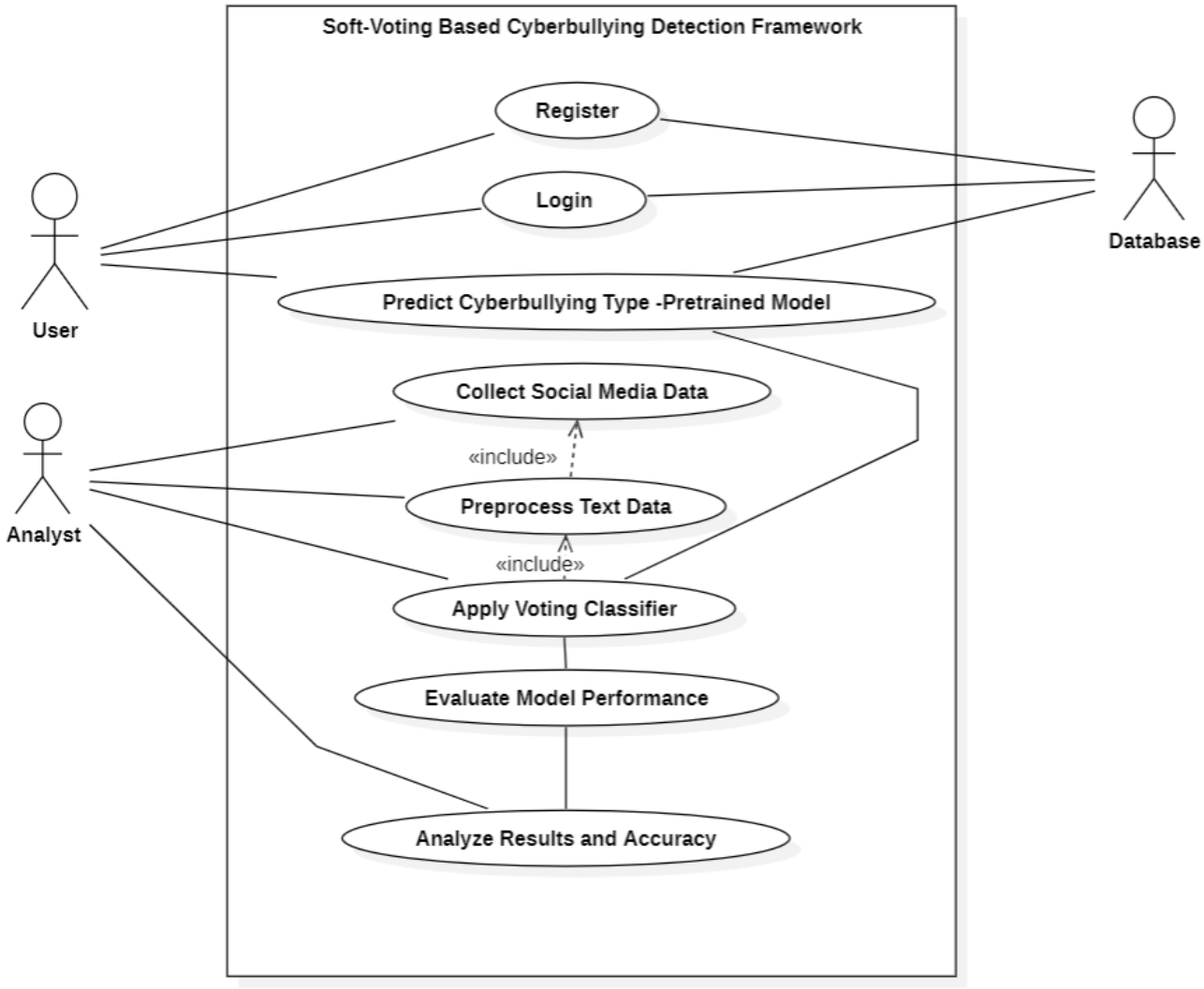


Fig. 5.3.2 Use Case Diagram

5.3.3 Activity Diagram

The activity diagram represents the step-by-step workflow of the soft-voting based cyberbullying detection system. It starts with collecting social media text, followed by preprocessing and feature extraction to transform raw input into structured features suitable for machine learning models. The flow then splits into two parallel activities where Boosted Decision Tree and Bagging Random Forest classifiers are applied simultaneously.

After parallel processing, the outputs from both models are merged and combined using a soft voting mechanism. This aggregation step enhances prediction reliability by considering the probabilities of both classifiers. Finally, the system classifies the type of cyberbullying and reaches the end state, completing the activity flow. The overall flow is represented in Fig. 5.3.3.

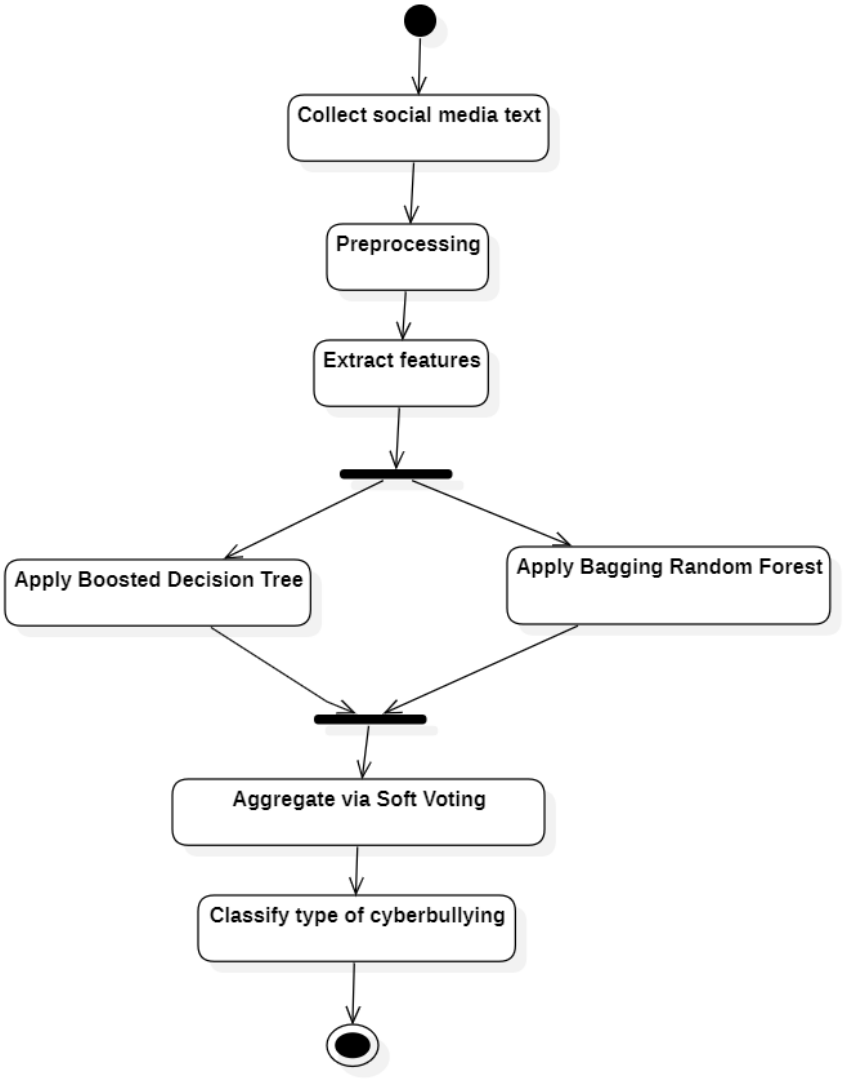


Fig. 5.3.3 Activity Diagram

5.3.4. Class Diagram

The class diagram represents the structural design of the cyberbullying detection system, showing the interaction between different components involved in training, prediction, and data management. The ModelTrainer trains individual models using the dataset, while the VotingClassifierBuilder aggregates these models and saves the final ensemble as a SavedModelFile. The Evaluation component assesses model performance using test data and generates reports, ensuring the model's effectiveness before deployment.

On the user side, the User interacts with the system by registering, logging in, and requesting classification. The CyberbullyingClassifier loads the pretrained model and processes user input to generate predictions, which are returned as a Result. The Database manages user credentials and stores prediction outputs, enabling authentication and persistence. Overall, the diagram clearly separates training, evaluation, and inference components, reflecting a modular and scalable system architecture. The overall process is represented in Fig. 5.3.4.

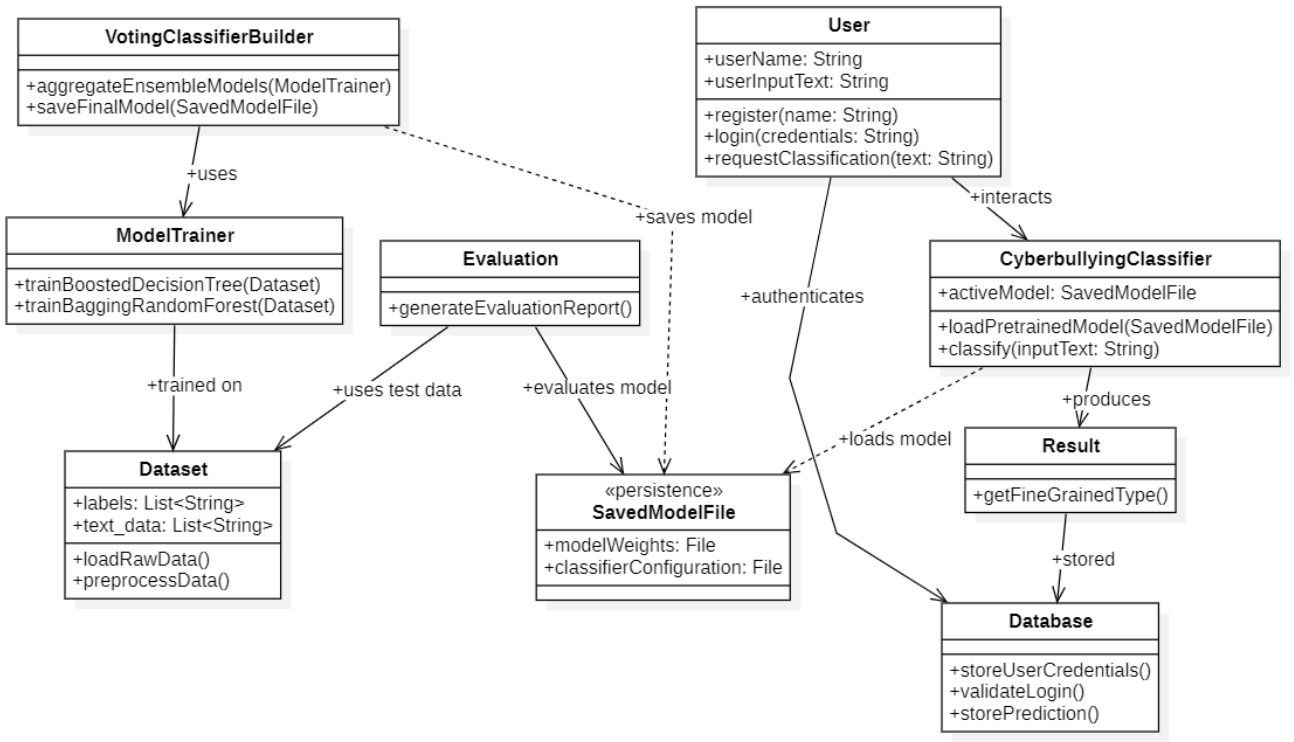


Fig. 5.3.4 Class Diagram

6. CODING AND IMPLEMENTATION

6.1 Source Code

Dataset1.ipynb:

```
import warnings
warnings.filterwarnings('ignore')
```

```
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
import seaborn as sns
```

```
data = pd.read_csv('dataset1/cyberbullying_tweets.csv')
```

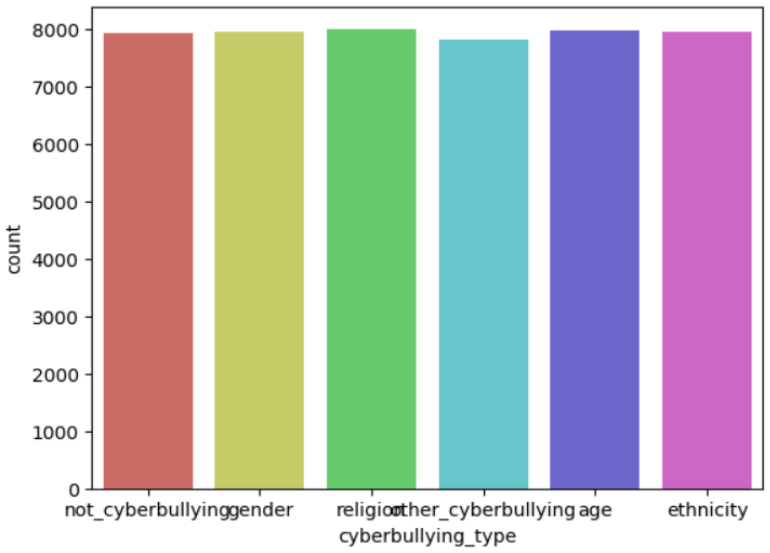
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 47692 entries, 0 to 47691
Data columns (total 2 columns):
 #   Column              Non-Null Count  Dtype 
---  ------              --------------  ----- 
 0   tweet_text          47692 non-null  object
 1   cyberbullying_type  47692 non-null  object
dtypes: object(2)
memory usage: 745.3+ KB
```

```
data.head()
```

	tweet_text	cyberbullying_type
0	In other words #katandandre, your food was cra...	not_cyberbullying
1	Why is #aussietv so white? #MKR #theblock #ImA...	not_cyberbullying
2	@XochitlSuckkks a classy whore? Or more red ve...	not_cyberbullying
3	@Jason_Gio meh. :P thanks for the heads up, b...	not_cyberbullying
4	@RudhoeEnglish This is an ISIS account pretend...	not_cyberbullying

```
sns.countplot(x='cyberbullying_type',data=data, palette='hls')
plt.show()
#plt.savefig('count_plot') mal: the nodule malignancy, 0: benign, 1: malignant
```



```
data['message'] = data['tweet_text']
```

```
from wordcloud import WordCloud, STOPWORDS
comment_words = ''
stopwords = set(STOPWORDS)
```

```
for val in data.message:

    # typecaste each val to string
    val = str(val)

    # split the value
    tokens = val.split()

    # Converts each token into lowercase
    for i in range(len(tokens)):
        tokens[i] = tokens[i].lower()

    comment_words += " ".join(tokens)+" "
```

```
wordcloud = WordCloud(width = 800, height = 800,
                      background_color ='white',
                      stopwords = stopwords,
                      min_font_size = 10).generate(comment_words)

# plot the WordCloud image
plt.figure(figsize = (12, 12), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)

plt.show()
```

```
from nltk.corpus import stopwords
from sklearn.model_selection import train_test_split
from nltk.tokenize import word_tokenize
import string
from nltk.stem import WordNetLemmatizer
```

```
data['cyberbullying_type'].value_counts()
```

```
religion               7998
age                    7992
gender                 7973
ethnicity              7961
not_cyberbullying      7945
other_cyberbullying    7823
Name: cyberbullying_type, dtype: int64
```

Binary

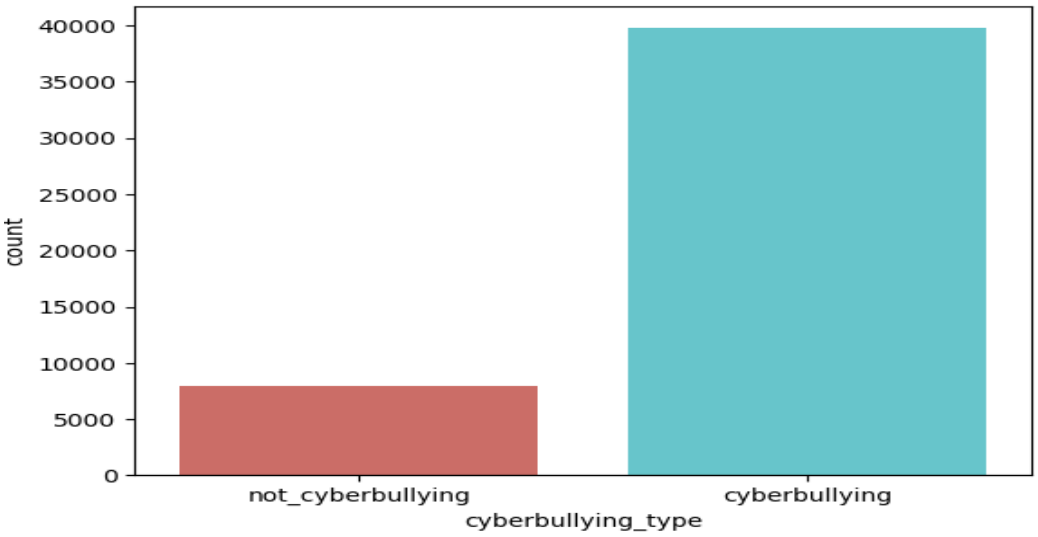
```
def change_label(df):
  df['cyberbullying_type'].replace(['religion','age','gender','ethnicity','other_cyberbullying'],'cyberbullying',inplace=True)
```

```
change_label(data)
```

```
data['cyberbullying_type'].value_counts()
```

```
cyberbullying        39747
not_cyberbullying     7945
Name: cyberbullying_type, dtype: int64
```

```
sns.countplot(x='cyberbullying_type',data=data, palette='hls')
plt.show()
```



```
Tweet = []
Labels = []

for row in data["message"]:
    #tokenize words
    words = word_tokenize(row)
    #remove punctuations
    clean_words = [word.lower() for word in words if word not in set(string.punctuation)]
    #remove stop words
    english_stops = set(stopwords.words('english'))
    characters_to_remove = ["'",'```',"rt","https","’","“","”","\u200b","--","n't","'s","...","//t.c" ]
    clean_words = [word for word in clean_words if word not in english_stops]
    clean_words = [word for word in clean_words if word not in set(characters_to_remove)]
    #Lematise words
    wordnet_lemmatizer = WordNetLemmatizer()
    lemma_list = [wordnet_lemmatizer.lemmatize(word) for word in clean_words]
    Tweet.append(lemma_list)
```

```
# Import label encoder
from sklearn import preprocessing

# label_encoder object knows
# how to understand word labels.
label_encoder = preprocessing.LabelEncoder()

# Encode labels in column 'species'.
data['cyberbullying_type']= label_encoder.fit_transform(data['cyberbullying_type'])

data['cyberbullying_type'].unique()
```

```
array([0, 1])
```

```
X = data['message']
y = data['cyberbullying_type']
```

```
# Extract Feature With CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
cv = TfidfVectorizer(stop_words='english')
X = cv.fit_transform(X) # Fit the Data
```

```
from imblearn.over_sampling import SMOTE
```

```
smt = SMOTE()
X, y = smt.fit_resample(X, y)
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=42)
```

```
ML_Model = []
accuracy = []
precision = []
recall = []
f1score = []

#function to call for storing the results
def storeResults(model, a,b,c,d):
    ML_Model.append(model)
    accuracy.append(round(a, 3))
    precision.append(round(d, 3))
    recall.append(round(c, 3))
    f1score.append(round(b, 3))
```

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
```

SVM

```
from sklearn import svm
svc = svm.LinearSVC()
svc.fit(X_train, y_train)

y_pred    = svc.predict(X_test)

svc_acc = accuracy_score(y_pred, y_test)
svc_prec = precision_score(y_pred, y_test,average='weighted')
svc_rec = recall_score(y_pred, y_test,average='weighted')
svc_f1 = f1_score(y_pred, y_test,average='weighted')
```

```
storeResults('SVM',svc_acc,svc_prec,svc_rec,svc_f1)
```

Random Forest

```
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier()

rf.fit(X_train, y_train)

y_pred    = rf.predict(X_test)

rf_acc = accuracy_score(y_pred, y_test)
rf_prec = precision_score(y_pred, y_test,average='weighted')
rf_rec = recall_score(y_pred, y_test,average='weighted')
rf_f1 = f1_score(y_pred, y_test,average='weighted')
```

```
storeResults('RandomForest',rf_acc,rf_prec,rf_rec,rf_f1)
```

Logistic Regression

```
from sklearn.linear_model import LogisticRegression
lr =  LogisticRegression(multi_class = 'multinomial', solver = 'lbfgs')
lr.fit(X_train, y_train)

y_pred    = lr.predict(X_test)

lr_acc = accuracy_score(y_pred, y_test)
lr_prec = precision_score(y_pred, y_test,average='weighted')
lr_rec = recall_score(y_pred, y_test,average='weighted')
lr_f1 = f1_score(y_pred, y_test,average='weighted')
```

```
storeResults('LogisticRegression',lr_acc,lr_prec,lr_rec,lr_f1)
```

MLP - OVO

```
from sklearn.multiclass import OneVsOneClassifier
from sklearn.neural_network import MLPClassifier

mlp = OneVsOneClassifier(MLPClassifier(random_state=1, max_iter=300))

mlp.fit(X_train, y_train)

y_pred    = mlp.predict(X_test)

mlp_acc = accuracy_score(y_pred, y_test)
mlp_prec = precision_score(y_pred, y_test,average='weighted')
mlp_rec = recall_score(y_pred, y_test,average='weighted')
mlp_f1 = f1_score(y_pred, y_test,average='weighted')
```

```
storeResults('MLP-OVO',mlp_acc,mlp_prec,mlp_rec,mlp_f1)
```

MLP - OVR

```
from sklearn.multiclass import OneVsRestClassifier
mlp = OneVsRestClassifier(MLPClassifier(random_state=1))

mlp.fit(X_train, y_train)

y_pred    = mlp.predict(X_test)

mlp1_acc = accuracy_score(y_pred, y_test)
mlp1_prec = precision_score(y_pred, y_test,average='weighted')
mlp1_rec = recall_score(y_pred, y_test,average='weighted')
mlp1_f1 = f1_score(y_pred, y_test,average='weighted')
```

```
storeResults('MLP-OVR',mlp1_acc,mlp1_prec,mlp1_rec,mlp1_f1)
```

Extension

```
from sklearn.ensemble import VotingClassifier, BaggingClassifier, ExtraTreesClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier

brf = BaggingClassifier(RandomForestClassifier())

bdt = AdaBoostClassifier(DecisionTreeClassifier(max_depth=1))

model = VotingClassifier(estimators= [('BoostDT', bdt),('BagRF', brf)], voting='soft')

model.fit(X_train, y_train)

y_pred = model.predict(X_test)

ext_acc = accuracy_score(y_pred, y_test)
ext_prec = precision_score(y_pred, y_test,average='weighted')
ext_rec = recall_score(y_pred, y_test,average='weighted')
ext_f1 = f1_score(y_pred, y_test,average='weighted')
```

```
storeResults('Extension',ext_acc,ext_prec,ext_rec,ext_f1)
```

Comparison

```
#creating dataframe
result = pd.DataFrame({ 'ML Model' : ML_Model,
                        'Accuracy' : accuracy,
                       'Precision': precision,
                       'Recall'   : recall,
                       'F1_score' : f1score
                      })
```

```
result
```

	ML Model	Accuracy	Precision	Recall	F1_score
0	SVM	0.963	0.963	0.963	0.965
1	RandomForest	0.954	0.954	0.954	0.955
2	LogisticRegression	0.943	0.943	0.943	0.946
3	MLP-OVO	0.950	0.950	0.950	0.952
4	MLP-OVR	0.950	0.950	0.950	0.952
5	Extension	0.994	0.994	0.994	0.994

Modelling

```
import pickle
pickle.dump(cv, open("data1_bin.pickle", "wb"))
```

```
import joblib
filename = 'model_data1_bin.sav'
joblib.dump(model, filename)
```

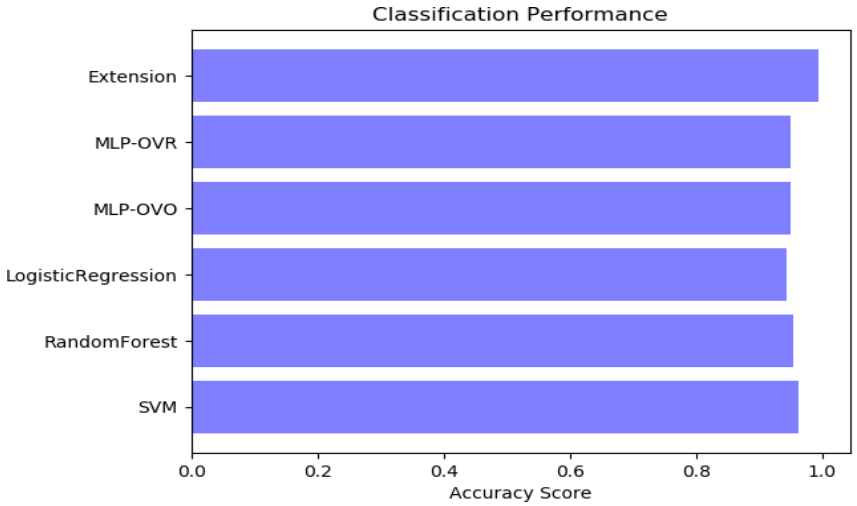
```
['model_data1_bin.sav']
```

Graph

```
classifier = ML_Model
y_pos = np.arange(len(classifier))
```

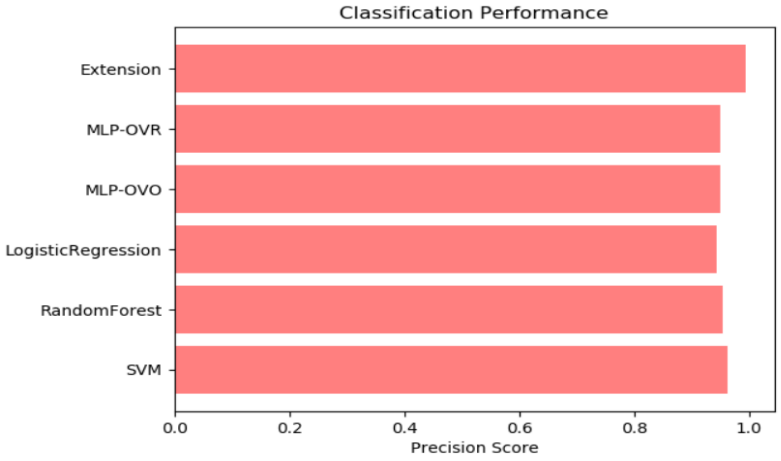
Accuracy

```
import matplotlib.pyplot as plt2
plt2.barh(y_pos, accuracy, align='center', alpha=0.5,color='blue')
plt2.yticks(y_pos, classifier)
plt2.xlabel('Accuracy Score')
plt2.title('Classification Performance')
plt2.show()
```



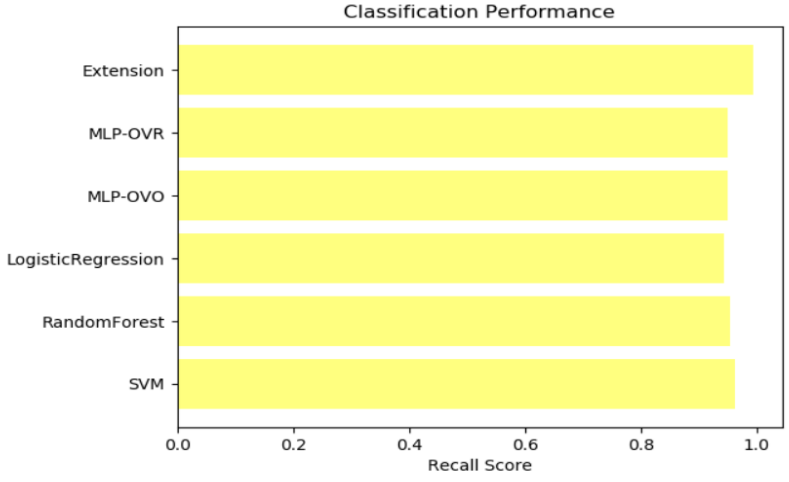
Precision

```
plt2.barh(y_pos, precision, align='center', alpha=0.5,color='red')
plt2.yticks(y_pos, classifier)
plt2.xlabel('Precision Score')
plt2.title('Classification Performance')
plt2.show()
```



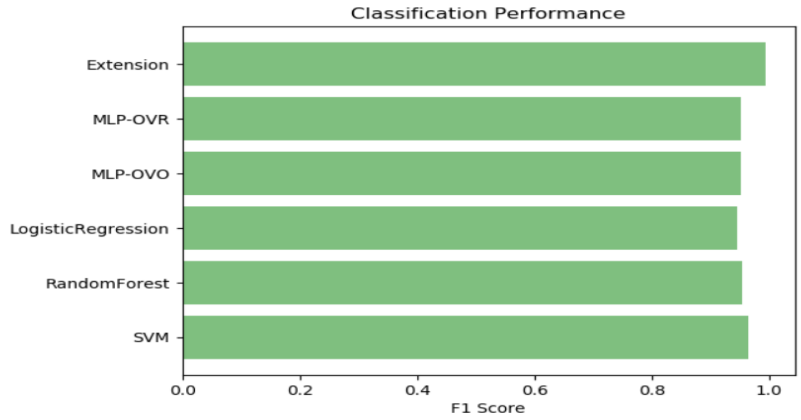
Recall

```
plt2.barh(y_pos, recall, align='center', alpha=0.5,color='yellow')
plt2.yticks(y_pos, classifier)
plt2.xlabel('Recall Score')
plt2.title('Classification Performance')
plt2.show()
```



F1 Score

```
plt2.barh(y_pos, f1score, align='center', alpha=0.5,color='green')
plt2.yticks(y_pos, classifier)
plt2.xlabel('F1 Score')
plt2.title('Classification Performance')
plt2.show()
```



Multi-Class

Collapse 9 child cells under Multi-Class (Press <Shift> to also collapse sibling sections)

```
data = pd.read_csv('dataset1/cyberbullying_tweets.csv')
```

```
data['message'] = data['tweet_text']
```

```
Tweet = []
Labels = []

for row in data["message"]:
    #tokenize words
    words = word_tokenize(row)
    #remove punctuations
    clean_words = [word.lower() for word in words if word not in set(string.punctuation)]
    #remove stop words
    english_stops = set(stopwords.words('english'))
    characters_to_remove = ["'",'``',"rt","https","’","“","”","\u200b","--","n't","'s","...","//t.c" ]
    clean_words = [word for word in clean_words if word not in english_stops]
    clean_words = [word for word in clean_words if word not in set(characters_to_remove)]
    #Lematise words
    wordnet_lemmatizer = WordNetLemmatizer()
    lemma_list = [wordnet_lemmatizer.lemmatize(word) for word in clean_words]
    Tweet.append(lemma_list)
```

```
# Import label encoder
from sklearn import preprocessing

# label_encoder object knows
# how to understand word labels.
label_encoder = preprocessing.LabelEncoder()

# Encode labels in column 'species'.
data['cyberbullying_type']= label_encoder.fit_transform(data['cyberbullying_type'])

data['cyberbullying_type'].unique()
```

```
array([5, 0, 2, 1, 3, 4])
```

```
X = data['message']
y = data['cyberbullying_type']
```

```
# Extract Feature With CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
cv = TfidfVectorizer(stop_words='english')
X = cv.fit_transform(X) # Fit the Data
```

```
smt = SMOTE()
X, y = smt.fit_resample(X, y)
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=42)
```

```
ML_Model = []
accuracy = []
precision = []
recall = []
f1score = []

#function to call for storing the results
def storeResults(model, a,b,c,d):
    ML_Model.append(model)
    accuracy.append(round(a, 3))
    precision.append(round(d, 3))
    recall.append(round(c, 3))
    f1score.append(round(b, 3))
```

SVM

```
svc = svm.LinearSVC()
svc.fit(X_train, y_train)

y_pred    = svc.predict(X_test)

svc_acc = accuracy_score(y_pred, y_test)
svc_prec = precision_score(y_pred, y_test,average='weighted')
svc_rec = recall_score(y_pred, y_test,average='weighted')
svc_f1 = f1_score(y_pred, y_test,average='weighted')
```

```
storeResults('SVM',svc_acc,svc_prec,svc_rec,svc_f1)
```

Random Forest

```
rf = RandomForestClassifier()

rf.fit(X_train, y_train)

y_pred    = rf.predict(X_test)

rf_acc = accuracy_score(y_pred, y_test)
rf_prec = precision_score(y_pred, y_test,average='weighted')
rf_rec = recall_score(y_pred, y_test,average='weighted')
rf_f1 = f1_score(y_pred, y_test,average='weighted')
```

```
storeResults('RandomForest',rf_acc,rf_prec,rf_rec,rf_f1)
```

LogisticRegression

```
lr =  LogisticRegression(multi_class = 'multinomial', solver = 'lbfgs')
lr.fit(X_train, y_train)

y_pred    = lr.predict(X_test)

lr_acc = accuracy_score(y_pred, y_test)
lr_prec = precision_score(y_pred, y_test,average='weighted')
lr_rec = recall_score(y_pred, y_test,average='weighted')
lr_f1 = f1_score(y_pred, y_test,average='weighted')
```

```
storeResults('LogisticRegression',lr_acc,lr_prec,lr_rec,lr_f1)
```

MLP – OVO

```
mlp = OneVsOneClassifier(MLPClassifier(random_state=1))

mlp.fit(X_train, y_train)

y_pred    = mlp.predict(X_test)

mlp_acc = accuracy_score(y_pred, y_test)
mlp_prec = precision_score(y_pred, y_test,average='weighted')
mlp_rec = recall_score(y_pred, y_test,average='weighted')
mlp_f1 = f1_score(y_pred, y_test,average='weighted')
```

```
storeResults('MLP-OVO',mlp_acc,mlp_prec,mlp_rec,mlp_f1)
```

MLP – OVR

```
mlp = OneVsRestClassifier(MLPClassifier(random_state=1))

mlp.fit(X_train, y_train)

y_pred    = mlp.predict(X_test)

mlp1_acc = accuracy_score(y_pred, y_test)
mlp1_prec = precision_score(y_pred, y_test,average='weighted')
mlp1_rec = recall_score(y_pred, y_test,average='weighted')
mlp1_f1 = f1_score(y_pred, y_test,average='weighted')
```

```
storeResults('MLP-OVR',mlp1_acc,mlp1_prec,mlp1_rec,mlp1_f1)
```

Extension

```
model = VotingClassifier(estimators= [('BoostDT', bdt),('BagRF', brf)], voting='soft')

model.fit(X_train, y_train)

y_pred = model.predict(X_test)

ext_acc = accuracy_score(y_pred, y_test)
ext_prec = precision_score(y_pred, y_test,average='weighted')
ext_rec = recall_score(y_pred, y_test,average='weighted')
ext_f1 = f1_score(y_pred, y_test,average='weighted')
```

```
storeResults('Extension',ext_acc,ext_prec,ext_rec,ext_f1)
```

Comparison

```
#creating dataframe
result = pd.DataFrame({ 'ML Model' : ML_Model,
                        'Accuracy' : accuracy,
                       'Precision': precision,
                       'Recall'   : recall,
                       'F1_score' : f1score
                      })
```

```
result
```

	ML Model	Accuracy	Precision	Recall	F1_score
0	SVM	0.832	0.830	0.832	0.830
1	RandomForest	0.825	0.827	0.825	0.847
2	LogisticRegression	0.820	0.815	0.820	0.818
3	MLP-OVO	0.322	0.458	0.322	0.957
4	MLP-OVR	0.634	0.631	0.634	0.705
5	Extension	0.985	0.985	0.985	0.985

Modelling

```
import pickle
pickle.dump(cv, open("data1_multi.pickle", "wb"))
```

```
import joblib
filename = 'model_data1_multi.sav'
joblib.dump(model, filename)
```

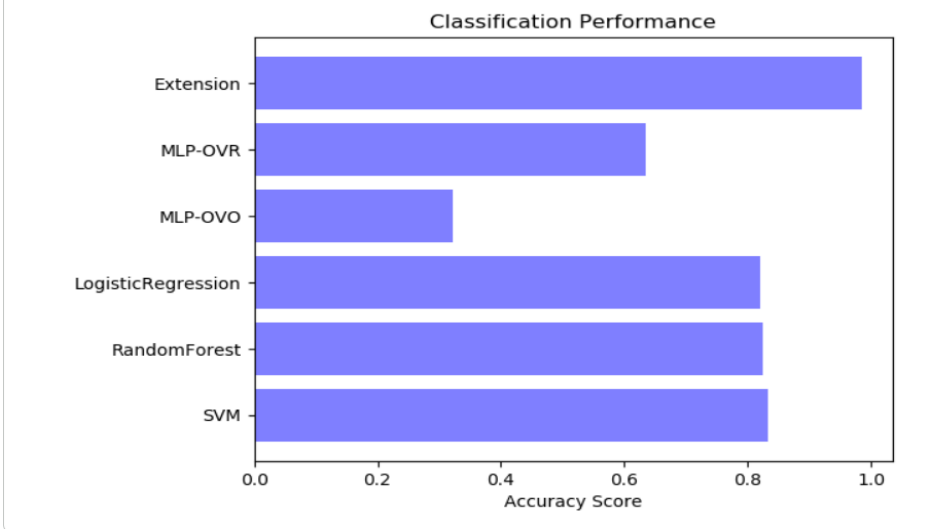
```
['model_data1_multi.sav']
```

Graph

```
classifier = ML_Model
y_pos = np.arange(len(classifier))
```

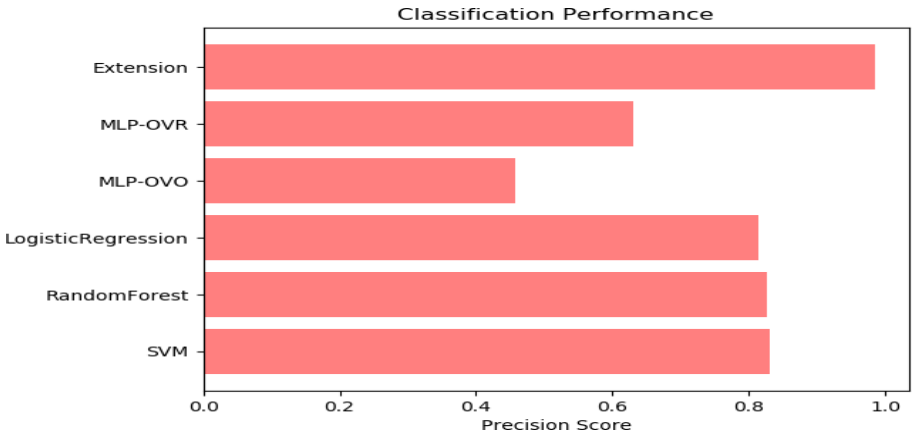
Accuracy

```
plt2.barh(y_pos, accuracy, align='center', alpha=0.5,color='blue')
plt2.yticks(y_pos, classifier)
plt2.xlabel('Accuracy Score')
plt2.title('Classification Performance')
plt2.show()
```



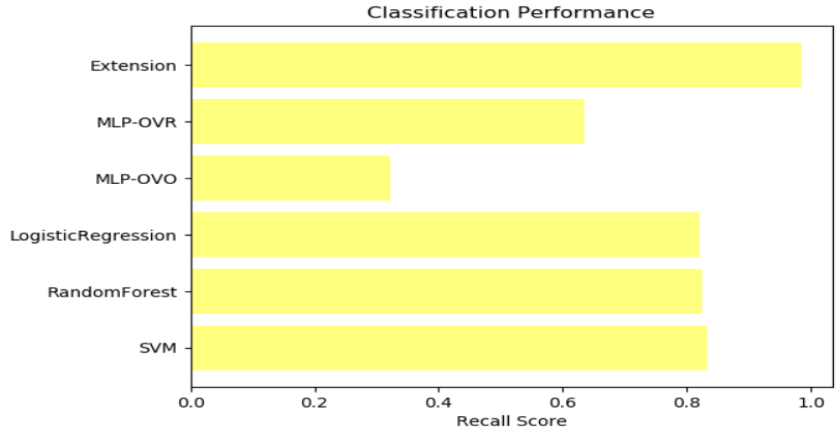
Preicision

```
plt2.barh(y_pos, precision, align='center', alpha=0.5,color='red')
plt2.yticks(y_pos, classifier)
plt2.xlabel('Precision Score')
plt2.title('Classification Performance')
plt2.show()
```



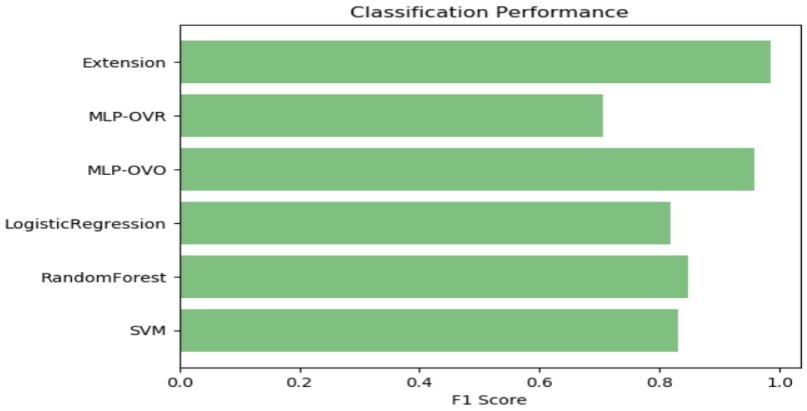
Recall

```
plt2.barh(y_pos, recall, align='center', alpha=0.5,color='yellow')
plt2.yticks(y_pos, classifier)
plt2.xlabel('Recall Score')
plt2.title('Classification Performance')
plt2.show()
```



F1 Score

```
plt2.barh(y_pos, f1score, align='center', alpha=0.5,color='green')
plt2.yticks(y_pos, classifier)
plt2.xlabel('F1 Score')
plt2.title('Classification Performance')
plt2.show()
```



Dataset2.ipynb:

```
import warnings
warnings.filterwarnings('ignore')
```

```
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
import seaborn as sns
```

```
data = pd.read_csv('dataset2/cb_multi_labeled_balanced.csv')
```

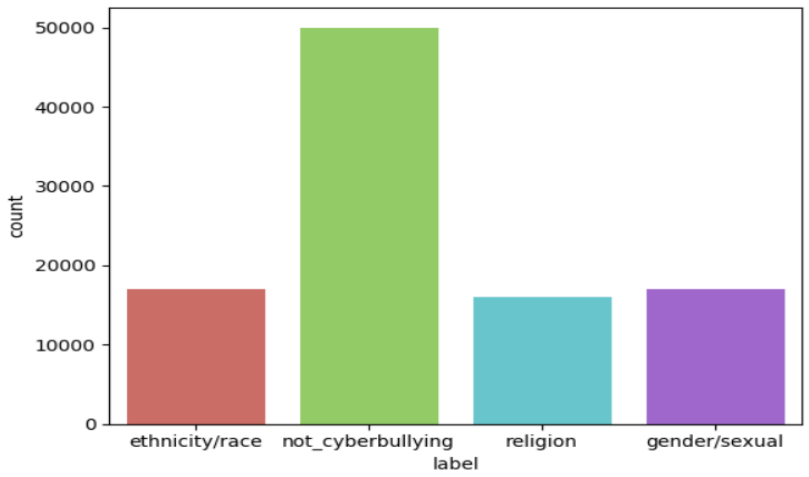
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 99990 entries, 0 to 99989
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype 
---  ------  --------------  ----- 
 0   text    99990 non-null  object
 1   label   99990 non-null  object
dtypes: object(2)
memory usage: 1.5+ MB
```

```
data.head()
```

	text	label
0	@ZubearSays Any real nigga isn't letting this ...	ethnicity/race
1	@MoradoSkittle @prolifejewess @DAConsult @Kell...	not_cyberbullying
2	the only thing i wish, i wish a nigga would	ethnicity/race
3	You saudias are not friends of Muslim idiots c...	religion
4	@JaydenT2399 @TractorLaw @holmes_gael @ercongc...	religion

```
sns.countplot(x='label',data=data, palette='hls')
plt.show()
#plt.savefig('count_plot') mal: the nodule malignancy, 0: benign, 1: malignant
```




```
data['message'] = data['text']
```

```
from wordcloud import WordCloud, STOPWORDS
comment_words = ''
stopwords = set(STOPWORDS)
```

```
for val in data.message:

    # typecaste each val to string
    val = str(val)

    # split the value
    tokens = val.split()

    # Converts each token into lowercase
    for i in range(len(tokens)):
        tokens[i] = tokens[i].lower()

    comment_words += " ".join(tokens)+" "
```

```
wordcloud = WordCloud(width = 800, height = 800,
                      background_color ='white',
                      stopwords = stopwords,
                      min_font_size = 10).generate(comment_words)

# plot the WordCloud image
plt.figure(figsize = (12, 12), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)

plt.show()
```


```
from nltk.corpus import stopwords
from sklearn.model_selection import train_test_split
from nltk.tokenize import word_tokenize
import string
from nltk.stem import WordNetLemmatizer
```

```
data['label'].value_counts()
```

```
not_cyberbullying    50000
ethnicity/race       17000
gender/sexual        17000
religion             15990
Name: label, dtype: int64
```

Binary

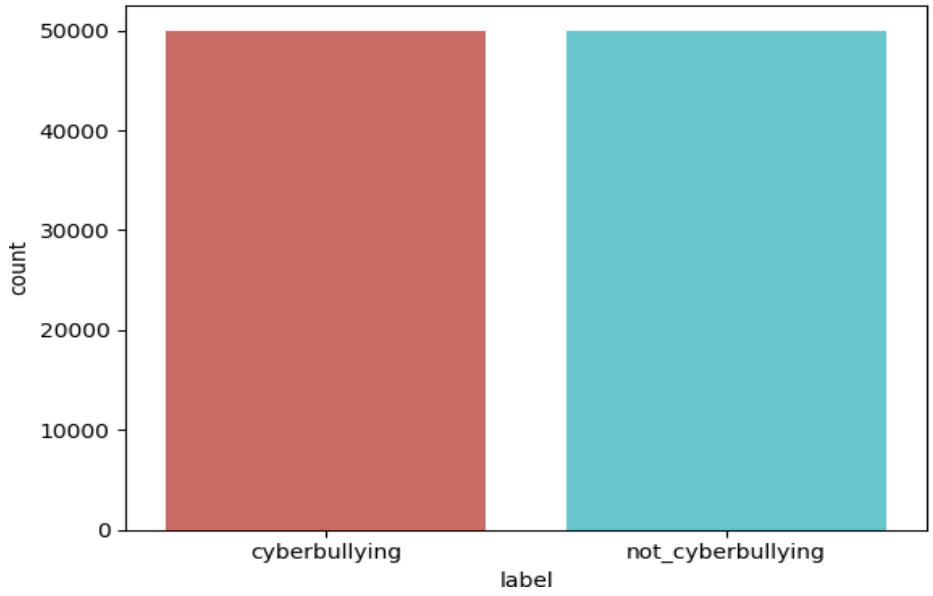
```
def change_label(df):
  df['label'].replace(['religion','ethnicity/race','gender/sexual'],'cyberbullying',inplace=True)
```

```
change_label(data)
```

```
data['label'].value_counts()
```

```
not_cyberbullying    50000
cyberbullying        49990
Name: label, dtype: int64
```

```
sns.countplot(x='label',data=data, palette='hls')
plt.show()
```




```
Tweet = []
Labels = []

for row in data["message"]:
    #tokenize words
    words = word_tokenize(row)
    #remove punctuations
    clean_words = [word.lower() for word in words if word not in set(string.punctuation)]
    #remove stop words
    english_stops = set(stopwords.words('english'))
    characters_to_remove = ["'",'``',"rt","https","’","“","”","\u200b","--","n't","'s","...","//t.c" ]
    clean_words = [word for word in clean_words if word not in english_stops]
    clean_words = [word for word in clean_words if word not in set(characters_to_remove)]
    #Lematise words
    wordnet_lemmatizer = WordNetLemmatizer()
    lemma_list = [wordnet_lemmatizer.lemmatize(word) for word in clean_words]
    Tweet.append(lemma_list)
```

```
# Import label encoder
from sklearn import preprocessing

# label_encoder object knows
# how to understand word labels.
label_encoder = preprocessing.LabelEncoder()

# Encode labels in column 'species'.
data['label']= label_encoder.fit_transform(data['label'])

data['label'].unique()
```

```
array([0, 1])
```

```
X = data['message']
y = data['label']
```

```
# Extract Feature With CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
cv = TfidfVectorizer(stop_words='english')
X = cv.fit_transform(X) # Fit the Data
```

```
from imblearn.over_sampling import SMOTE
```

```
smt = SMOTE()
X, y = smt.fit_resample(X, y)
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=42)
```

```
ML_Model = []
accuracy = []
precision = []
recall = []
f1score = []

#function to call for storing the results
def storeResults(model, a,b,c,d):
    ML_Model.append(model)
    accuracy.append(round(a, 3))
    precision.append(round(d, 3))
    recall.append(round(c, 3))
    f1score.append(round(b, 3))
```

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
```

SVM

```
from sklearn import svm
svc = svm.LinearSVC()
svc.fit(X_train, y_train)

y_pred    = svc.predict(X_test)

svc_acc = accuracy_score(y_pred, y_test)
svc_prec = precision_score(y_pred, y_test,average='weighted')
svc_rec = recall_score(y_pred, y_test,average='weighted')
svc_f1 = f1_score(y_pred, y_test,average='weighted')
```

```
storeResults('SVM',svc_acc,svc_prec,svc_rec,svc_f1)
```

Random Forest

```
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier()

rf.fit(X_train, y_train)

y_pred    = rf.predict(X_test)

rf_acc = accuracy_score(y_pred, y_test)
rf_prec = precision_score(y_pred, y_test,average='weighted')
rf_rec = recall_score(y_pred, y_test,average='weighted')
rf_f1 = f1_score(y_pred, y_test,average='weighted')
```

```
storeResults('RandomForest',rf_acc,rf_prec,rf_rec,rf_f1)
```

Logistic Regression

```
from sklearn.linear_model import LogisticRegression
lr =  LogisticRegression(multi_class = 'multinomial', solver = 'lbfgs')
lr.fit(X_train, y_train)

y_pred    = lr.predict(X_test)

lr_acc = accuracy_score(y_pred, y_test)
lr_prec = precision_score(y_pred, y_test,average='weighted')
lr_rec = recall_score(y_pred, y_test,average='weighted')
lr_f1 = f1_score(y_pred, y_test,average='weighted')
```

```
storeResults('LogisticRegression',lr_acc,lr_prec,lr_rec,lr_f1)
```

MLP – OVO

```
from sklearn.multiclass import OneVsOneClassifier
from sklearn.neural_network import MLPClassifier

mlp = OneVsOneClassifier(MLPClassifier(random_state=1))

mlp.fit(X_train, y_train)

y_pred    = mlp.predict(X_test)

mlp_acc = accuracy_score(y_pred, y_test)
mlp_prec = precision_score(y_pred, y_test,average='weighted')
mlp_rec = recall_score(y_pred, y_test,average='weighted')
mlp_f1 = f1_score(y_pred, y_test,average='weighted')
```

```
storeResults('MLP-OVO',mlp_acc,mlp_prec,mlp_rec,mlp_f1)
```

MLP – OVR

```
from sklearn.multiclass import OneVsRestClassifier
mlp = OneVsRestClassifier(MLPClassifier(random_state=1))

mlp.fit(X_train, y_train)

y_pred    = mlp.predict(X_test)

mlp1_acc = accuracy_score(y_pred, y_test)
mlp1_prec = precision_score(y_pred, y_test,average='weighted')
mlp1_rec = recall_score(y_pred, y_test,average='weighted')
mlp1_f1 = f1_score(y_pred, y_test,average='weighted')
```

```
storeResults('MLP-OVR',mlp1_acc,mlp1_prec,mlp1_rec,mlp1_f1)
```

Extension

```
from sklearn.ensemble import VotingClassifier, BaggingClassifier, ExtraTreesClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier

brf = BaggingClassifier(RandomForestClassifier())

bdt = AdaBoostClassifier(DecisionTreeClassifier(max_depth=1))

model = VotingClassifier(estimators= [('BoostDT', bdt),('BagRF', brf)], voting='soft')

model.fit(X_train, y_train)

y_pred = model.predict(X_test)

ext_acc = accuracy_score(y_pred, y_test)
ext_prec = precision_score(y_pred, y_test,average='weighted')
ext_rec = recall_score(y_pred, y_test,average='weighted')
ext_f1 = f1_score(y_pred, y_test,average='weighted')
```

```
storeResults('Extension',ext_acc,ext_prec,ext_rec,ext_f1)
```

Comparison

```
#creating dataframe
result = pd.DataFrame({ 'ML Model' : ML_Model,
                        'Accuracy' : accuracy,
                       'Precision': precision,
                       'Recall'   : recall,
                       'F1_score' : f1score
                      })
```

```
result
```

	ML Model	Accuracy	Precision	Recall	F1_score
0	SVM	0.988	0.987	0.988	0.988
1	RandomForest	0.993	0.993	0.993	0.993
2	LogisticRegression	0.978	0.978	0.978	0.979
3	MLP-OVO	0.489	0.657	0.489	1.000
4	MLP-OVR	0.968	0.968	0.968	0.970
5	Extension	1.000	1.000	1.000	1.000

Modelling

```
import pickle
pickle.dump(cv, open("data2_bin.pickle", "wb"))
```

```
import joblib
filename = 'model_data2_bin.sav'
joblib.dump(model, filename)
```

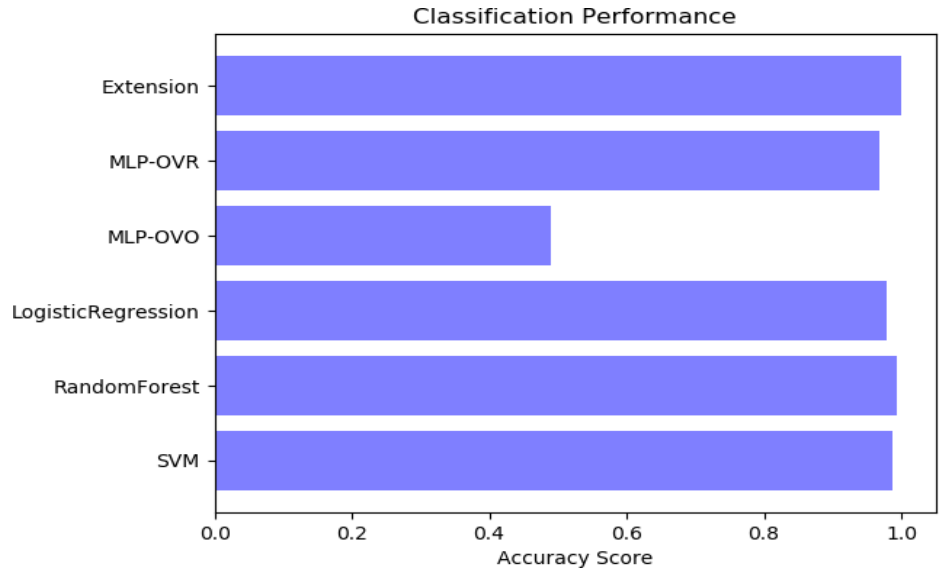
```
['model_data2_bin.sav']
```

Graph

```
classifier = ML_Model
y_pos = np.arange(len(classifier))
```

Accuracy

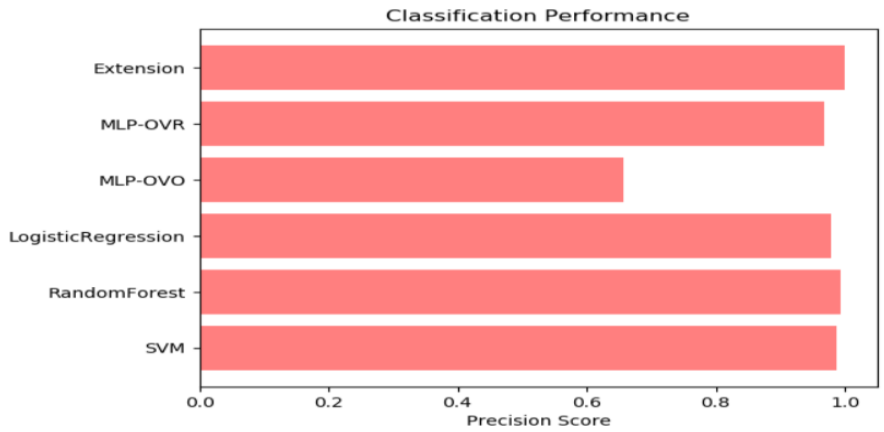
```
import matplotlib.pyplot as plt2
plt2.barh(y_pos, accuracy, align='center', alpha=0.5,color='blue')
plt2.yticks(y_pos, classifier)
plt2.xlabel('Accuracy Score')
plt2.title('Classification Performance')
plt2.show()
```



Precision

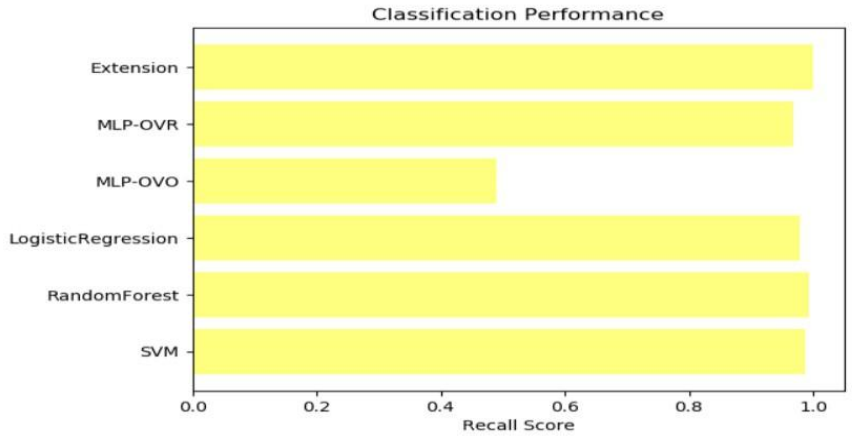
Collapse 1 child cell under Precision (Press <Shift> to also collapse sibling sections)

```
plt2.barh(y_pos, precision, align='center', alpha=0.5,color='red')
plt2.yticks(y_pos, classifier)
plt2.xlabel('Precision Score')
plt2.title('Classification Performance')
plt2.show()
```



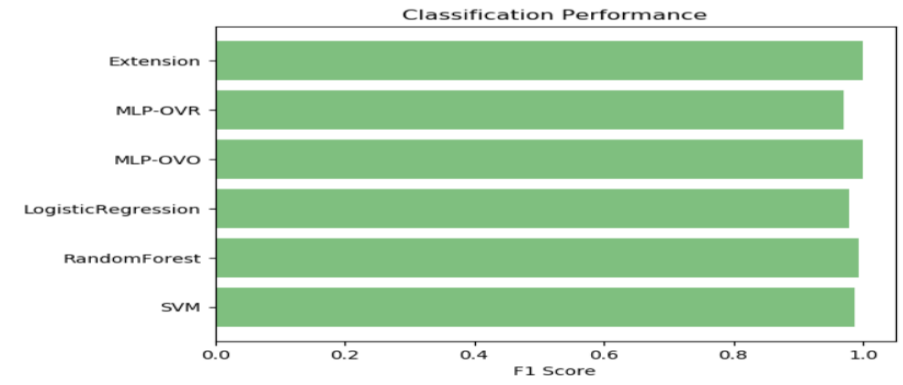
Recall

```
plt2.barh(y_pos, recall, align='center', alpha=0.5,color='yellow')
plt2.yticks(y_pos, classifier)
plt2.xlabel('Recall Score')
plt2.title('Classification Performance')
plt2.show()
```



F1 Score

```
plt2.barh(y_pos, f1score, align='center', alpha=0.5,color='green')
plt2.yticks(y_pos, classifier)
plt2.xlabel('F1 Score')
plt2.title('Classification Performance')
plt2.show()
```



Multi-Class

```
data = pd.read_csv('dataset2/cb_multi_labeled_balanced.csv')
```

```
data['message'] = data['text']
```

```
#print(data.info())
data = pd.concat([
    data[data.label == 'ethnicity/race'].sample(n=10_00),
    data[data.label == 'gender/sexual'].sample(n=10_00),
    data[data.label == 'religion'].sample(n=10_00),
    data[data.label == 'not_cyberbullying'].sample(n=10_00)
])
```

```
Tweet = []
Labels = []

for row in data["message"]:
    #tokenize words
    words = word_tokenize(row)
    #remove punctuations
    clean_words = [word.lower() for word in words if word not in set(string.punctuation)]
    #remove stop words
    english_stops = set(stopwords.words('english'))
    characters_to_remove = ["'",'``',"rt","https","’","“","”","\u200b","--","n't","'s","...","//t.c" ]
    clean_words = [word for word in clean_words if word not in english_stops]
    clean_words = [word for word in clean_words if word not in set(characters_to_remove)]
    #Lematise words
    wordnet_lemmatizer = WordNetLemmatizer()
    lemma_list = [wordnet_lemmatizer.lemmatize(word) for word in clean_words]
    Tweet.append(lemma_list)
```

```
# Import label encoder
from sklearn import preprocessing

# label_encoder object knows
# how to understand word labels.
label_encoder = preprocessing.LabelEncoder()

# Encode labels in column 'species'.
data['label']= label_encoder.fit_transform(data['label'])

data['label'].unique()
```

```
array([0, 1, 3, 2])
```

```
X = data['message']
y = data['label']
```

```
# Extract Feature With CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
cv = TfidfVectorizer(stop_words='english')
X = cv.fit_transform(X) # Fit the Data
```

```
smt = SMOTE()
X, y = smt.fit_resample(X, y)
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=42)
```

```
ML_Model = []
accuracy = []
precision = []
recall = []
f1score = []

#function to call for storing the results
def storeResults(model, a,b,c,d):
    ML_Model.append(model)
    accuracy.append(round(a, 3))
    precision.append(round(d, 3))
    recall.append(round(c, 3))
    f1score.append(round(b, 3))
```

SVM

```
svc = svm.LinearSVC()
svc.fit(X_train, y_train)

y_pred     = svc.predict(X_test)

svc_acc = accuracy_score(y_pred, y_test)
svc_prec = precision_score(y_pred, y_test,average='weighted')
svc_rec = recall_score(y_pred, y_test,average='weighted')
svc_f1 = f1_score(y_pred, y_test,average='weighted')
```

```
storeResults('SVM',svc_acc,svc_prec,svc_rec,svc_f1)
```

Random Forest

```
rf = RandomForestClassifier()

rf.fit(X_train, y_train)

y_pred     = rf.predict(X_test)

rf_acc = accuracy_score(y_pred, y_test)
rf_prec = precision_score(y_pred, y_test,average='weighted')
rf_rec = recall_score(y_pred, y_test,average='weighted')
rf_f1 = f1_score(y_pred, y_test,average='weighted')
```

```
storeResults('RandomForest',rf_acc,rf_prec,rf_rec,rf_f1)
```

LogisticRegression

```
lr =  LogisticRegression(multi_class = 'multinomial', solver = 'lbfgs')
lr.fit(X_train, y_train)

y_pred     = lr.predict(X_test)

lr_acc = accuracy_score(y_pred, y_test)
lr_prec = precision_score(y_pred, y_test,average='weighted')
lr_rec = recall_score(y_pred, y_test,average='weighted')
lr_f1 = f1_score(y_pred, y_test,average='weighted')
```

```
storeResults('LogisticRegression',lr_acc,lr_prec,lr_rec,lr_f1)
```

MLP – OVO

```
mlp = OneVsOneClassifier(MLPClassifier(random_state=1))

mlp.fit(X_train, y_train)

y_pred     = mlp.predict(X_test)

mlp_acc = accuracy_score(y_pred, y_test)
mlp_prec = precision_score(y_pred, y_test,average='weighted')
mlp_rec = recall_score(y_pred, y_test,average='weighted')
mlp_f1 = f1_score(y_pred, y_test,average='weighted')
```

```
storeResults('MLP-OVO',mlp_acc,mlp_prec,mlp_rec,mlp_f1)
```

MLP – OVR

```
mlp = OneVsRestClassifier(MLPClassifier(random_state=1))

mlp.fit(X_train, y_train)

y_pred     = mlp.predict(X_test)

mlp1_acc = accuracy_score(y_pred, y_test)
mlp1_prec = precision_score(y_pred, y_test,average='weighted')
mlp1_rec = recall_score(y_pred, y_test,average='weighted')
mlp1_f1 = f1_score(y_pred, y_test,average='weighted')
```

```
storeResults('MLP-OVR',mlp1_acc,mlp1_prec,mlp1_rec,mlp1_f1)
```

Extension

```
model = VotingClassifier(estimators= [('BoostDT', bdt),('BagRF', brf)], voting='soft')

model.fit(X_train, y_train)

y_pred = model.predict(X_test)

ext_acc = accuracy_score(y_pred, y_test)
ext_prec = precision_score(y_pred, y_test,average='weighted')
ext_rec = recall_score(y_pred, y_test,average='weighted')
ext_f1 = f1_score(y_pred, y_test,average='weighted')
```

```
storeResults('Extension',ext_acc,ext_prec,ext_rec,ext_f1)
```

Comparison

```
#creating dataframe
result = pd.DataFrame({ 'ML Model' : ML_Model,
                        'Accuracy' : accuracy,
                       'Precision': precision,
                       'Recall'   : recall,
                       'F1_score' : f1score
                      })
```

```
result
```

	ML Model	Accuracy	Precision	Recall	F1_score
0	SVM	0.976	0.976	0.976	0.977
1	RandomForest	0.981	0.981	0.981	0.981
2	LogisticRegression	0.935	0.934	0.935	0.940
3	MLP-OVO	0.532	0.607	0.532	0.721
4	MLP-OVR	0.715	0.723	0.715	0.765
5	Extension	0.997	0.997	0.997	0.997

Modelling

```
import pickle
pickle.dump(cv, open("data2_multi.pickle", "wb"))
```

```
import joblib
filename = 'model_data2_multi.sav'
joblib.dump(model, filename)
```

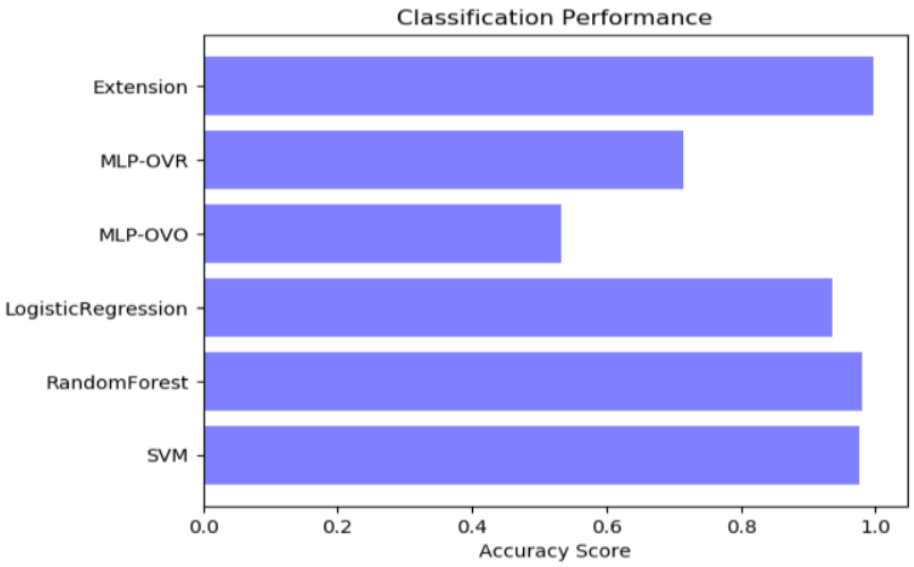
```
['model_data2_multi.sav']
```

Graph

```
classifier = ML_Model
y_pos = np.arange(len(classifier))
```

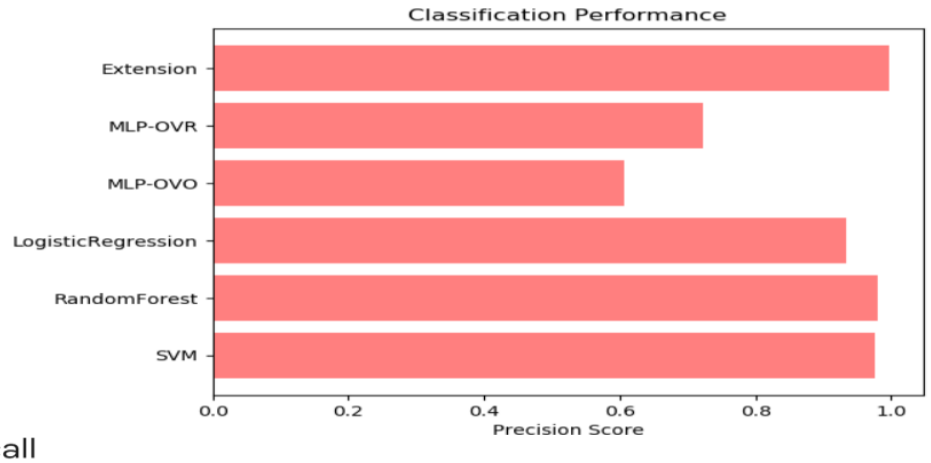
Accuracy

```
plt2.barh(y_pos, accuracy, align='center', alpha=0.5,color='blue')
plt2.yticks(y_pos, classifier)
plt2.xlabel('Accuracy Score')
plt2.title('Classification Performance')
plt2.show()
```



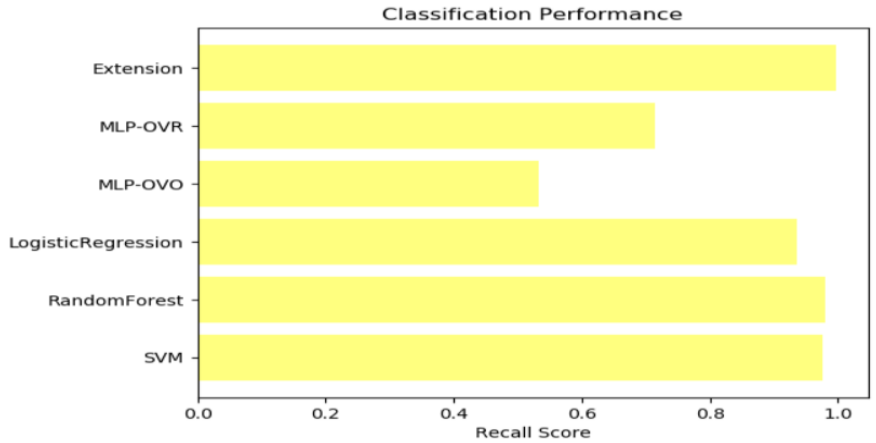
Preicision

```
plt2.barh(y_pos, precision, align='center', alpha=0.5,color='red')
plt2.yticks(y_pos, classifier)
plt2.xlabel('Precision Score')
plt2.title('Classification Performance')
plt2.show()
```



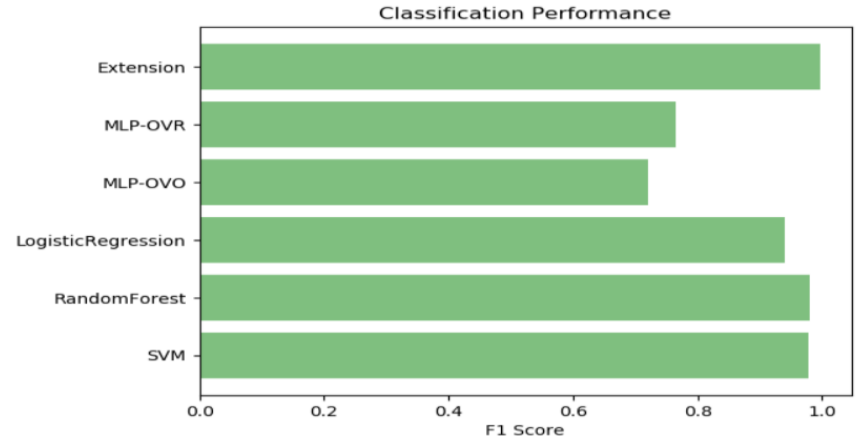
Recall

```
plt2.barh(y_pos, recall, align='center', alpha=0.5,color='yellow')
plt2.yticks(y_pos, classifier)
plt2.xlabel('Recall Score')
plt2.title('Classification Performance')
plt2.show()
```



F1 Score

```
plt2.barh(y_pos, f1score, align='center', alpha=0.5,color='green')
plt2.yticks(y_pos, classifier)
plt2.xlabel('F1 Score')
plt2.title('Classification Performance')
plt2.show()
```



Backend:

Views.py

Backend > views.py

```python
from django.shortcuts import render, redirect
from django.contrib.auth.models import User
from django.contrib.auth import authenticate, login, logout
from django.contrib import messages

def index(request):
    return render(request, "index.html")

def login_page(request):
    return render(request, "login.html")

def register_page(request):
    return render(request, "register.html")

# Define the login function
def user_login(request):
    if request.method == "POST":
        username = request.POST.get('username')
        password = request.POST.get('password')

        # Authenticate user
        user = authenticate(request, username=username, password=password)

        if user is not None:
            if not user.is_active:
                # User is inactive
                messages.error(request, "Your account is inactive. Please contact the admin.")
                return redirect('login_page')

            # Login the user
            login(request, user)

            if user.is_staff or user.is_superuser:
                # Redirect to admin home if user is staff
                return redirect('adminhome')
            else:
                # Redirect to user home if user is not staff
                return redirect('userhome')
```

```
        else:
            # Invalid username or password
            messages.error(request, "Invalid username or password.")
            return redirect('login_page')

    return render(request, 'login.html')

# Define the user registration function
def user_registration(request):
    if request.method == "POST":
        username = request.POST.get('username')
        email = request.POST.get('email')
        password = request.POST.get('password')
        confirm_password = request.POST.get('confirm_password')
        first_name = request.POST.get('first_name')
        last_name = request.POST.get('last_name')

        # Check if passwords match
        if password != confirm_password:
            messages.error(request, "Passwords do not match.")
            return redirect('register_page')

        # Check if username already exists
        if User.objects.filter(username=username).exists():
            messages.error(request, "Username already exists.")
            return redirect('register_page')

        # Check if email already exists
        if User.objects.filter(email=email).exists():
            messages.error(request, "Email already exists.")
            return redirect('register_page')

        # Create the user with is_active set to False
        user = User.objects.create_user(
            username=username,
            email=email,
            password=password,
            first_name=first_name,
            last_name=last_name
        )
        user.is_active = False  # Set is_active to False by default
        user.save()

        messages.success(request, "Registration successful! Please wait for admin approval.")
        return redirect('login_page')

    return render(request, 'register.html')

# Define the logout function
def user_logout(request):
    logout(request)
    messages.success(request, "You have been logged out successfully.")
    return redirect('login_page')
```

Views.py

User > views.py > user_prediction1

```python
from django.shortcuts import render
from .models import UserPrediction
import joblib
import pickle
import string
import nltk
import os
from django.conf import settings

from django.shortcuts import render
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer
import speech_recognition as sr

nltk.download('punkt')
nltk.download('wordnet')
nltk.download('stopwords')

model = joblib.load("model/model_data1_bin.sav")
with open("model/data1_bin.pickle", "rb") as f:
    vectorizer = pickle.load(f)

model2 = joblib.load("model/model_data2_bin.sav")
with open("model/data2_bin.pickle", "rb") as f:
    vectorizer2 = pickle.load(f)

def preprocess_text(text):
    words = word_tokenize(text)
    clean_words = [word.lower() for word in words if word not in string.punctuation]

    stop_words = set(stopwords.words('english'))
    extra_removals = ["'", '``', "rt", "https", "", "“", "”", "\u200b", "--", "n't", "'s", "...", "//t.c"]

    clean_words = [word for word in clean_words if word not in stop_words and word not in extra_removals]

    lemmatizer = WordNetLemmatizer()
    lemmatized = [lemmatizer.lemmatize(word) for word in clean_words]

    return " ".join(lemmatized)

def userhome(request):
    user = request.user
    return render(request, 'User/userhome.html', {'user':user})


def user_prediction1(request):
    prediction_label = None
```

```
    input_text = None

    if request.method == "POST":
        input_text = request.POST.get("user_input", "")
        processed_text = preprocess_text(input_text)
        vectorized_input = vectorizer.transform([processed_text])
        prediction = model.predict(vectorized_input)
        prediction_label = "Not Cyberbullying" if prediction[0] == 0 else "Cyberbullying"

        UserPrediction.objects.create(
            input_text=input_text,
            prediction_result=prediction_label
        )

    return render(request, 'User/userprediction1.html', {
        'input_text': input_text,
        'prediction_label': prediction_label
    })

def user_prediction2(request):
    prediction_label = None
    input_text = None

    if request.method == "POST":
        input_text = request.POST.get("user_input", "")
        processed_text = preprocess_text(input_text)
        vectorized_input = vectorizer2.transform([processed_text])
        prediction = model2.predict(vectorized_input)
        print(prediction)
        prediction_label = "Not Cyberbullying" if prediction[0] == 1 else "Cyberbullying"

        UserPrediction.objects.create(
            input_text=input_text,
            prediction_result=prediction_label
        )

    return render(request, 'User/userprediction2.html', {
        'input_text': input_text,
        'prediction_label': prediction_label
    })

def Graphs(request):
    graphs_folder = os.path.join(settings.BASE_DIR, 'static', 'graphs')
    static_url = settings.STATIC_URL + 'graphs/'

```

```
    comparison_images = []

    try:
        for i in range(1, 12):
            left = f'd1o{i}.png'
            right = f'd2o{i}.png'

            if left in os.listdir(graphs_folder) and right in os.listdir(graphs_folder):
                comparison_images.append({
                    'left': static_url + left,
                    'right': static_url + right,
                    'label': f'Comparison {i}'
                })
    except FileNotFoundError:
        comparison_images = []

    return render(request, 'User/graphs.html', {'comparisons': comparison_images})

def uservoice_prediction1(request):
    prediction_label = None
    input_text = None

    if request.method == "POST":
        input_text = request.POST.get("voice_input", "")
        processed_text = preprocess_text(input_text)
        vectorized_input = vectorizer2.transform([processed_text])
        prediction = model2.predict(vectorized_input)
        prediction_label = "Not Cyberbullying" if prediction[0] == 0 else "Cyberbullying"

    return render(request, 'User/uservoiceprediction1.html', {
        'input_text': input_text,
        'prediction_label': prediction_label
    })
```

Frontend:

Base.html:

templates › base.html › html › body.index-page › header#header.header.d-flex.align-items-center.light-background.sticky-top › div.container-fluid.position-relative.d-flex.align-items-center.justify-content-between › nav#navmenu.nav

```
<!DOCTYPE html>
{% load static %}
<html lang="en">

<head>
  <meta charset="utf-8">
  <meta content="width=device-width, initial-scale=1.0" name="viewport">
  <title>Social Media Forensics</title>
  <meta name="description" content="">
  <meta name="keywords" content="">

  <!-- Fonts -->
  <link href="https://fonts.googleapis.com" rel="preconnect">
  <link href="https://fonts.gstatic.com" rel="preconnect" crossorigin>
  <link href="https://fonts.googleapis.com/css2?family=Roboto:ital,wght@0,100;0,300;0,400;0,500;0,700;0,900;1,100;1,300;1,400;1,500;1,700;1,900&family=Poppins:ital,wght@0,100;0,200;0,300;0,400;0,500;0,600;0,700;0,800;0,900;1,100;1,200;1,300;1,400;1,500;1,600;1,700;1,800;1,900&family=Raleway:ital,wght@0,100;0,200;0,300;0,400;0,500;0,600;0,700;0,800;0,900;1,100;1,200;1,300;1,400;1,500;1,600;1,700;1,800;1,900&display=swap" rel="stylesheet">

  <!-- Vendor CSS Files -->
  <link href="{% static 'vendor/bootstrap/css/bootstrap.min.css' %}" rel="stylesheet">
  <link href="{% static 'vendor/bootstrap-icons/bootstrap-icons.css' %}" rel="stylesheet">
  <link href="{% static 'vendor/aos/aos.css' %}" rel="stylesheet">
  <link href="{% static 'vendor/swiper/swiper-bundle.min.css' %}" rel="stylesheet">
  <link href="{% static 'vendor/glightbox/css/glightbox.min.css' %}" rel="stylesheet">

  <!-- Main CSS File -->
  <link href="{% static 'css/main.css' %}" rel="stylesheet">


</head>

<body class="index-page">

  <header id="header" class="header d-flex align-items-center light-background sticky-top">
    <div class="container-fluid position-relative d-flex align-items-center justify-content-between">

      <a  class="logo d-flex align-items-center me-auto me-xl-0">
        <!-- Uncomment the line below if you also wish to use an image logo -->
        <!-- <img src="assets/img/logo.png" alt=""> -->
        <h1 class="sitename">Social Media Forensics: An Adaptive Cyberbullying Related Hate Speech Detection</h1>
      </a>

      <nav id="navmenu" class="navmenu">
        <ul>
          <li><a href="{% url 'home page' %}">Home</a></li>
          <li><a href="{% url 'login page' %}">Login</a></li>
        </ul>
        <i class="mobile-nav-toggle d-xl-none bi bi-list"></i>
```

```
      </nav>

      <div class="header-social-links">
      </div>

    </div>
  </header>

  <main class="main">

    {%block contents%}

    {%endblock%}

  </main>


  <!-- Scroll Top -->
  <a href="#" id="scroll-top" class="scroll-top d-flex align-items-center justify-content-center"><i class="bi bi-arrow-up-short"></i></a>

  <!-- Preloader -->
  <div id="preloader"></div>

  <!-- Vendor JS Files -->
  <script src="{% static 'vendor/bootstrap/js/bootstrap.bundle.min.js' %}"></script>
  <script src="{% static 'vendor/php-email-form/validate.js' %}"></script>
  <script src="{% static 'vendor/aos/aos.js' %}"></script>
  <script src="{% static 'vendor/waypoints/noframework.waypoints.js' %}"></script>
  <script src="{% static 'vendor/purecounter/purecounter_vanilla.js' %}"></script>
  <script src="{% static 'vendor/swiper/swiper-bundle.min.js' %}"></script>
  <script src="{% static 'vendor/glightbox/js/glightbox.min.js' %}"></script>
  <script src="{% static 'vendor/imagesloaded/imagesloaded.pkgd.min.js' %}"></script>
  <script src="{% static 'vendor/isotope-layout/isotope.pkgd.min.js' %}"></script>

  <!-- Main JS File -->
  <script src="{% static 'js/main.js' %}"></script>

</body>

</html>
```

Main.js

static > js > JS main.js > ...

```
(function() {
  "use strict";

  /**
   * Apply .scrolled class to the body as the page is scrolled down
   */
  function toggleScrolled() {
    const selectBody = document.querySelector('body');
    const selectHeader = document.querySelector('#header');
    if (!selectHeader.classList.contains('scroll-up-sticky') && !selectHeader.classList.contains('sticky-top') && !selectHeader.classList.contains('fixed-top')) return;
    window.scrollY > 100 ? selectBody.classList.add('scrolled') : selectBody.classList.remove('scrolled');
  }

  document.addEventListener('scroll', toggleScrolled);
  window.addEventListener('load', toggleScrolled);

  /**
   * Mobile nav toggle
   */
  const mobileNavToggleBtn = document.querySelector('.mobile-nav-toggle');

  function mobileNavToogle() {
    document.querySelector('body').classList.toggle('mobile-nav-active');
    mobileNavToggleBtn.classList.toggle('bi-list');
    mobileNavToggleBtn.classList.toggle('bi-x');
  }
  mobileNavToggleBtn.addEventListener('click', mobileNavToogle);

  /**
   * Hide mobile nav on same-page/hash links
   */
  document.querySelectorAll('#navmenu a').forEach(navmenu => {
    navmenu.addEventListener('click', () => {
      if (document.querySelector('.mobile-nav-active')) {
        mobileNavToogle();
      }
    });

  });

  /**
   * Toggle mobile nav dropdowns
   */
  document.querySelectorAll('.navmenu .toggle-dropdown').forEach(navmenu => {
    navmenu.addEventListener('click', function(e) {
      e.preventDefault();
      this.parentNode.classList.toggle('active');
      this.parentNode.nextElementSibling.classList.toggle('dropdown-active');
```

```
      e.stopImmediatePropagation();
    });
  });

  /**
   * Preloader
   */
  const preloader = document.querySelector('#preloader');
  if (preloader) {
    window.addEventListener('load', () => {
      preloader.remove();
    });
  }

  /**
   * Scroll top button
   */
  let scrollTop = document.querySelector('.scroll-top');

  function toggleScrollTop() {
    if (scrollTop) {
      window.scrollY > 100 ? scrollTop.classList.add('active') : scrollTop.classList.remove('active');
    }
  }
  scrollTop.addEventListener('click', (e) => {
    e.preventDefault();
    window.scrollTo({
      top: 0,
      behavior: 'smooth'
    });
  });

  window.addEventListener('load', toggleScrollTop);
  document.addEventListener('scroll', toggleScrollTop);

  /**
   * Animation on scroll function and init
   */
  function aosInit() {
    AOS.init({
      duration: 600,
      easing: 'ease-in-out',
      once: true,
      mirror: false
    });
```

```
Click to add a breakpoint
97  window.addEventListener('load', aosInit);
98
99  /**
100  * Animate the skills items on reveal
101  */
102 let skillsAnimation = document.querySelectorAll('.skills-animation');
103 skillsAnimation.forEach((item) => {
104   new Waypoint({
105     element: item,
106     offset: '80%',
107     handler: function(direction) {
108       let progress = item.querySelectorAll('.progress .progress-bar');
109       progress.forEach(el => {
110         el.style.width = el.getAttribute('aria-valuenow') + '%';
111       });
112     }
113   });
114 });
115
116 /**
117  * Initiate Pure Counter
118  */
119 new PureCounter();
120
121 /**
122  * Init swiper sliders
123  */
124 function initSwiper() {
125   document.querySelectorAll(".init-swiper").forEach(function(swiperElement) {
126     let config = JSON.parse(
127       swiperElement.querySelector(".swiper-config").innerHTML.trim()
128     );
129
130     if (swiperElement.classList.contains("swiper-tab")) {
131       initSwiperWithCustomPagination(swiperElement, config);
132     } else {
133       new Swiper(swiperElement, config);
134     }
135   });
136 }
137
138 window.addEventListener("load", initSwiper);
139
140 /**
141  * Initiate glightbox
```

```
Click to add a breakpoint
143     const glightbox = GLightbox({
144       selector: '.glightbox'
145     });
146
147     /**
148      * Init isotope layout and filters
149      */
150     document.querySelectorAll('.isotope-layout').forEach(function(isotopeItem) {
151       let layout = isotopeItem.getAttribute('data-layout') ?? 'masonry';
152       let filter = isotopeItem.getAttribute('data-default-filter') ?? '*';
153       let sort = isotopeItem.getAttribute('data-sort') ?? 'original-order';
154
155       let initIsotope;
156       imagesLoaded(isotopeItem.querySelector('.isotope-container'), function() {
157         initIsotope = new Isotope(isotopeItem.querySelector('.isotope-container'), {
158           itemSelector: '.isotope-item',
159           layoutMode: layout,
160           filter: filter,
161           sortBy: sort
162         });
163       });
164
165       isotopeItem.querySelectorAll('.isotope-filters li').forEach(function(filters) {
166         filters.addEventListener('click', function() {
167           isotopeItem.querySelector('.isotope-filters .filter-active').classList.remove('filter-active');
168           this.classList.add('filter-active');
169           initIsotope.arrange({
170             filter: this.getAttribute('data-filter')
171           });
172           if (typeof aosInit === 'function') {
173             aosInit();
174           }
175         }, false);
176       });
177
178     });
179
180   })();
```

Uservoiceprediction1.html

```
{% extends 'User/userbase.html' %}
{% block contents %}
<section class="container py-5">
  <div class="row justify-content-center">
    <div class="col-lg-8" data-aos="fade-up">
      <div class="card p-4 shadow rounded-4">
        <h2 class="mb-4 text-center">Voice Analysis</h2>
        <form method="POST">
          {% csrf_token %}
          <div class="mb-3">
            <label class="form-label">Click and speak:</label>
            <button type="button" class="btn btn-secondary mb-2" onclick="startRecognition()">🎤 Start Voice Input</button>
            <textarea id="voiceInput" name="voice_input" class="form-control" rows="6" readonly>{{ input_text }}</textarea>
          </div>
          <div class="text-center">
            <button type="submit" class="btn btn-primary px-4 py-2">Predict</button>
          </div>
        </form>

        {% if prediction_label %}
          <div class="alert alert-info mt-4 text-center" role="alert">
            <h5 class="mb-0">Prediction: {{ prediction_label }}</h5>
          </div>
        {% endif %}
      </div>
    </div>
  </div>
</section>

<script>
  function startRecognition() {
    const recognition = new (window.SpeechRecognition || window.webkitSpeechRecognition)();
    recognition.lang = 'en-US';
    recognition.interimResults = false;
    recognition.maxAlternatives = 1;

    recognition.start();

    recognition.onresult = function(event) {
      const transcript = event.results[0][0].transcript;
      document.getElementById('voiceInput').value = transcript;
    };

    recognition.onerror = function(event) {
      alert('Speech recognition error: ' + event.error);
    };
  }
</script>
{% endblock %}

```

Userprediction2.html

templates > User > userprediction2.html > ...

```
{% extends 'User/userbase.html' %}

{% block contents %}
<section class="container py-5">
  <div class="row justify-content-center">
    <div class="col-lg-8" data-aos="fade-up">
      <div class="card p-4 shadow rounded-4">
        <h2 class="mb-4 text-center">Text Analysis</h2>
        <form method="POST">
          {% csrf_token %}
          <div class="mb-3">
            <label for="user_input" class="form-label">Enter text:</label>
            <textarea name="user_input" class="form-control" rows="6">{{ input_text }}</textarea>
          </div>
          <div class="text-center">
            <button type="submit" class="btn btn-primary px-4 py-2">Predict</button>
          </div>
        </form>

        {% if prediction_label %}
          <div class="alert alert-info mt-4 text-center" role="alert">
            <h5 class="mb-0">Prediction: {{ prediction_label }}</h5>
          </div>
        {% endif %}
      </div>
    </div>
  </div>
</section>
{% endblock %}

```

6.2 Implementation:

6.2.1 Front-End Implementation:

The front-end of the cyberbullying detection system provides a simple, responsive, and role-based user interface. The main modules include User Registration, User Login, Admin Panel, Text-based Prediction, and Voice-based Prediction.

The Registration and Login modules enable secure user authentication and controlled system access. Input validation is applied to ensure correctness and prevent invalid submissions. The Admin Panel allows authorized administrators to review usage logs, monitor predictions, and manage users and datasets.

The Prediction module enables users to submit textual content for analysis. Once submitted, the input is sent to the backend through REST APIs. The resulting classification is returned and displayed as a clear cyberbullying category. The system intentionally presents only the final category output to keep the interface straightforward and easy to interpret.

The Voice-based Prediction feature allows users to speak instead of typing. Recorded audio is converted to text using a speech-to-text component and the extracted text is then processed through the same classification pipeline. Consistent layouts, clear navigation, and structured feedback enhance usability and accessibility.

6.2.2 Backend Implementation (Django):

The backend is implemented using Django, which provides a secure and scalable framework. The Model–View–Template architecture organizes system functionality into coherent layers:

- Models store user profiles, activity logs, and prediction records.
- Views manage requests, perform validation, and trigger prediction services.
- URLs / APIs define endpoints for authentication, text submission, voice-converted text submission, and result retrieval.

Security mechanisms such as authentication controls, middleware checks, and request validation ensure integrity of data processing. Prediction history is stored to support auditing and further analysis.

6.2.3 Model Integration and Processing Workflow

The machine-learning module is integrated as a backend service within Django. When a text request is received, it passes through the preprocessing pipeline that includes normalization, tokenization, stop- word removal, lemmatization, and TF-IDF feature generation.

Classification is performed using an ensemble approach that combines Boosted Decision Tree (BoostDT) and Bagging Random Forest (BagRF) through Soft Voting. Each model produces a decision, and the voting mechanism determines the final category label returned to the system. SMOTE-balanced datasets and stored model artifacts ensure stable behavior across input variations.

Responses are returned to the front-end in structured JSON format and rendered to the user as the predicted cyberbullying class.

6.2.4 Deployment and Reliability

The system is deployed on a standard server configuration using environment-based settings for security. Static assets are optimized, and REST endpoints are tested for consistent behavior under multiple usage scenarios. Unit and integration tests validate preprocessing, API interaction, and classification response handling.

6.2.5 Conclusion

The implementation successfully integrates a user-friendly interface, a secure Django backend, and an ensemble-based machine-learning framework into a unified cyberbullying detection platform. Both text and voice-based inputs are supported, predictions are generated in real time, and administrative monitoring features enable oversight and maintenance. The modular architecture allows future enhancements such as multilingual support, advanced deep-learning integration, and real-time moderation capabilities. Overall, the implemented system demonstrates a practical, scalable approach to improving online safety through automated cyberbullying detection.

7. SYSTEM TESTING

System testing is a critical activity that ensures the developed cyberbullying detection system performs accurately, consistently, and reliably under real operating conditions. The primary purpose of testing is to identify potential defects, validate system behavior, and confirm that all functional and non-functional requirements have been met.

In this project, system testing focuses on validating every major module, including front-end interfaces, Django backend services, preprocessing components, and the ensemble-based classification engine integrating Boosted Decision Tree (BoostDT) and Bagging Random Forest (BagRF) through Soft Voting. Testing verifies that user interactions, dataset handling, model prediction logic, and output presentation operate as expected without failures or inconsistencies.

System testing was performed across multiple scenarios and datasets to ensure correctness, robustness, and usability. Special emphasis was placed on classification behavior, handling of edge-case text inputs, and stability during repeated prediction requests.

7.1 Types of System Testing

7.1.1 Unit Testing

Unit testing was carried out to validate individual software components independently. Each Django view, function, preprocessing routine, and classifier interaction module was executed with controlled input values to verify expected outputs.

Key focus areas included:

- tokenization, normalization, and lemmatization behavior
- TF-IDF feature vector generation
- internal logic of ensemble combination
- database operations for login, registration, and predictions

Unit testing ensured that every internal logical path executed correctly and no unexpected conditions occurred. Failures during this stage would have been easier to isolate and correct before integration.

7.1.2 Integration Testing

Integration testing examined whether combined components interacted correctly once they were linked together.

Modules tested in combination included:

- front-end request submission with backend API responses
- preprocessing and feature extraction chained with classification
- ensemble model logic when receiving outputs from BoostDT and BagRF
- prediction logging and storage in the database

This testing stage confirmed that individually correct components functioned properly when executed together as a single workflow. Particular attention was paid to ensuring correct feature alignment between models and stability in Soft Voting combination.

7.1.3 Functional Testing

Functional testing validated that each feature performed according to specification and user expectations. Test cases simulated actual user scenarios such as registration, login, text submission, and voice-based prediction.

Major validation rules included:

- valid inputs are processed successfully
- invalid or empty inputs are rejected gracefully
- correct cyberbullying category is displayed for each prediction
- navigation links and page workflows operate correctly

Functional testing confirmed that system features were clearly available, usable, and responsive.

7.1.4 System Testing

System testing evaluated the project as a complete application. The focus was on overall reliability, consistency of behavior, and the accuracy of outcomes when used in real conditions.

Tests verified:

- coordinated execution across modules
- correct response to large datasets
- classification accuracy under varying abuse patterns

- stability during repeated sequential predictions

The testing demonstrated that the configuration yields predictable and correct results consistent with documented requirements.

7.1.5 White-Box Testing

White-box testing was applied to internal processing components including preprocessing pipelines and ensemble computation logic. Testers observed internal variable flows, code execution branches, and probability aggregation to ensure correct model contributions in Soft Voting.

7.1.6 Black-Box Testing

Black-box testing evaluated the system purely from the user's perspective, without examining internal code. Inputs were submitted through user forms and prediction outputs were observed, ensuring that visible system behavior aligned with expected outcomes.

This was particularly important in evaluating system usability and error-handling behavior.

7.1.7 Acceptance Testing

Acceptance testing ensured that the system satisfied all documented requirements and end-user expectations. Stakeholders reviewed usability, accuracy, output clarity, and workflow navigation.

Feedback confirmed that the system was intuitive, responsive, and aligned with real cyberbullying detection needs.

Test Result Summary:

All defined test cases passed successfully. No major defects were encountered.

7.2 Testing Strategies

A structured testing strategy was followed throughout the project lifecycle. Testing progressed systematically from component-level validation to full-system verification.

7.2.1 Test Strategy and Approach

Testing was performed both manually and programmatically. Detailed execution logs and dataset-based test scripts were used to validate consistency of classifier behavior.

Primary strategic objectives included:

- verifying correctness of text preprocessing and model inputsensuring ensemble behavior consistently outperformed single classifiers
- verifying error-free interactions between user interface and backend services
- validating prediction reliability under noisy, sarcastic, and ambiguous text

Field testing simulated real-world user activity, while controlled test cases verified logical correctness.

7.2.2 Test Objectives

The following objectives guided all testing activities:

- all fields and forms must operate correctly
- screens and interactions should respond without delay
- invalid or malformed inputs must be handled safely
- predictions should follow expected patterns in comparable research literature
- transitions between system pages must be correct and intuitive

7.2.3 Features Tested

The major system features examined included:

- data entry validation and prevention of duplicate accounts
- correct routing of each navigation link
- prediction accuracy across cyberbullying categories
- correct Soft Voting operation between BoostDT and BagRF
- adherence to expected model training and evaluation workflows

7.2.4 Integration Testing Strategy

Integration testing emphasized early detection of dependency conflicts and data mismatches.

Testing confirmed correct:

- alignment of TF-IDF features with both classifiers
- synchronization of probability outputs into Soft Voting
- interface communication flow with Django APIs

This strategy prevented error propagation into later development stages.

7.2.5 Acceptance Criteria

A prediction system instance was accepted only when it satisfied these conditions:

- accurate execution of classification pipeline
- stable runtime performance
- error-free navigation and submission
- meaningful categories shown without misinterpretation
- compliance with defined requirements

7.2.6 Overall Test Results

All planned test cases executed successfully. The system demonstrated stable performance, logical correctness, and consistent cyberbullying prediction accuracy. The Soft Voting ensemble consistently produced more reliable outcomes compared to evaluating either single classifier independently.

7.2.7 Conclusion

System testing confirmed that the developed ensemble-based cyberbullying detection system satisfies functional expectations, operates reliably under diverse input conditions, and integrates all modules effectively. Through rigorous testing strategies, the project achieved robustness, user readiness, and high classification dependability.

7.3 Sample Test Cases

S No.	Test Case	Expected Result	Result	Remarks (if any)
01.	User Login	User is authenticated and granted access to their dashboard	Pass	Verify incorrect credentials show appropriate error messages
02.	User Registration	New user account is created and stored in the database	Pass	Validate email format and duplicate-account handling
03.	User Account Activation	Registered user becomes active and can log in successfully	Pass	Ensure inactive users cannot access the system
04.	Text-Based Prediction	System processes submitted text and displays correct cyberbullying category	Pass	Test normal, abusive, sarcasm- hidden, and empty inputs
05.	Voice-Based Prediction	Voice input is converted to text and classified accurately	Pass	Check microphone permissions and noisy-audio handling
06.	Admin Panel	Admin can view users, predictions	Pass	Ensure only authorized admins can access this module

Table no 7.3 Test Cases

Test Case 1:

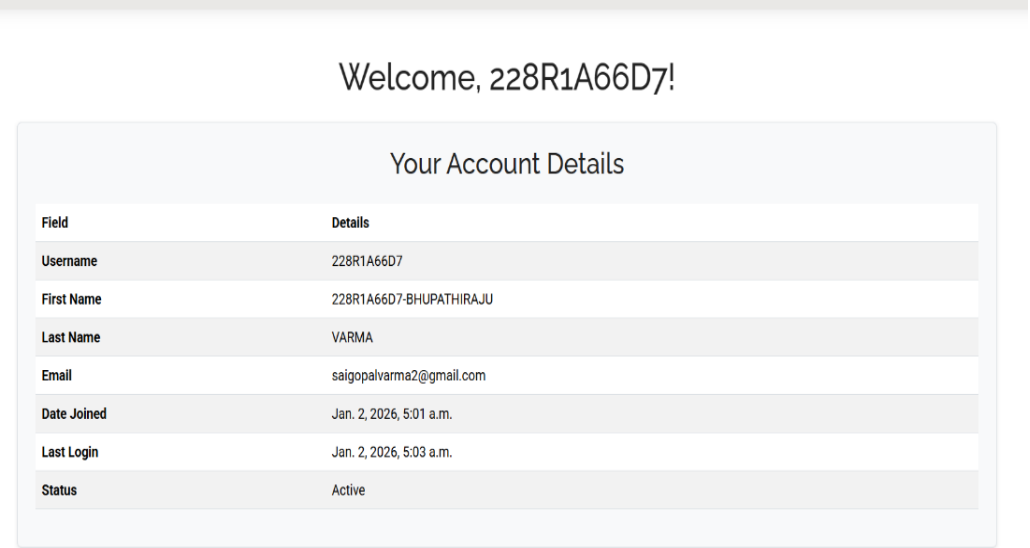


Fig. 7.3.1 User Login

Description: The user enters valid login credentials and submits the form. The system authenticates the credentials and redirects the user to the dashboard, displaying a personalized welcome message and complete account details such as username, email, status, and last login time. This seamless flow, illustrated in Fig. 7.3.1, confirms that both authentication and user-profile retrieval are functioning correctly.

Test Case 2:

Fig. 7.3.2 User Registration

Description: After submitting valid registration details, the system successfully creates the user account and displays a confirmation message indicating that registration is complete. The message also informs the user that their account will remain inactive until approved by the administrator. This confirms that the registration workflow and approval control mechanism are functioning correctly, as illustrated in Fig. 7.3.2.

Test Case 3:

Fig.7.3.3 User Account Activation

Description: When the administrator approves the newly registered user, the system changes the account status from Inactive to Active and displays a confirmation message indicating successful activation. The user is now allowed to log in and access the system. This verifies that the admin-controlled activation process works correctly and prevents unauthorized access before approval, as illustrated in Fig. 7.3.3.

Test Case 4:

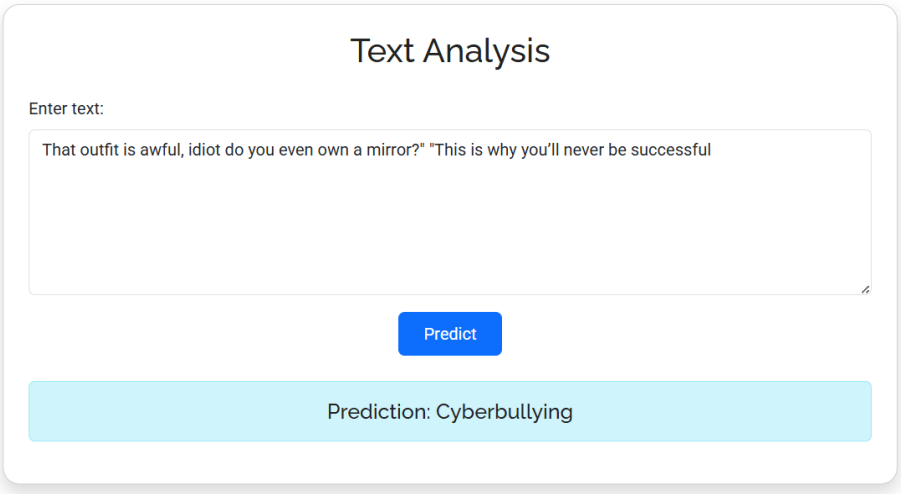


Fig. 7.3.4 Text-Based Prediction

Description: The user enters a piece of text into the input box and submits it for analysis. The system processes the text through the preprocessing pipeline and ensemble classifier, then displays the predicted result as "Cyberbullying." This confirms that the text-analysis workflow, model integration, and result display components are functioning correctly, as illustrated in Fig. 7.3.4**.**

Test Case 5:

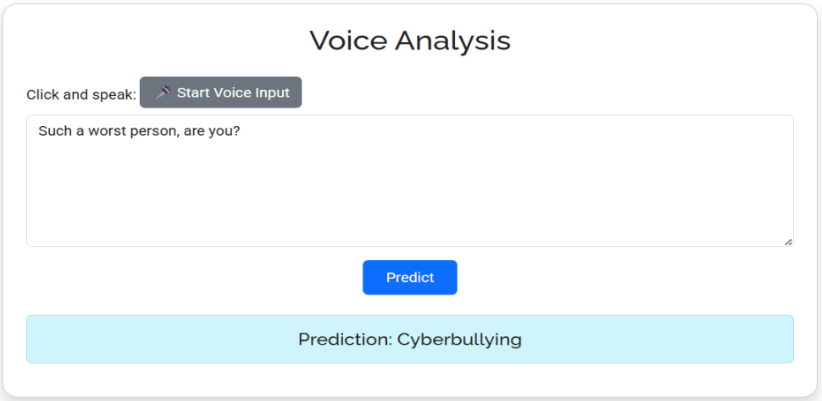


Fig. 7.3.5 Voice-Based Prediction

Description: The user activates the microphone and speaks a sentence, which is automatically converted to text and displayed in the input area. The system processes the transcribed text using the ensemble classifier and returns the predicted result as "Cyberbullying." This confirms that the speech- to-text integration, prediction pipeline, and result display are working together correctly, as illustrated in Fig. 7.3.5.

Test Case 6:

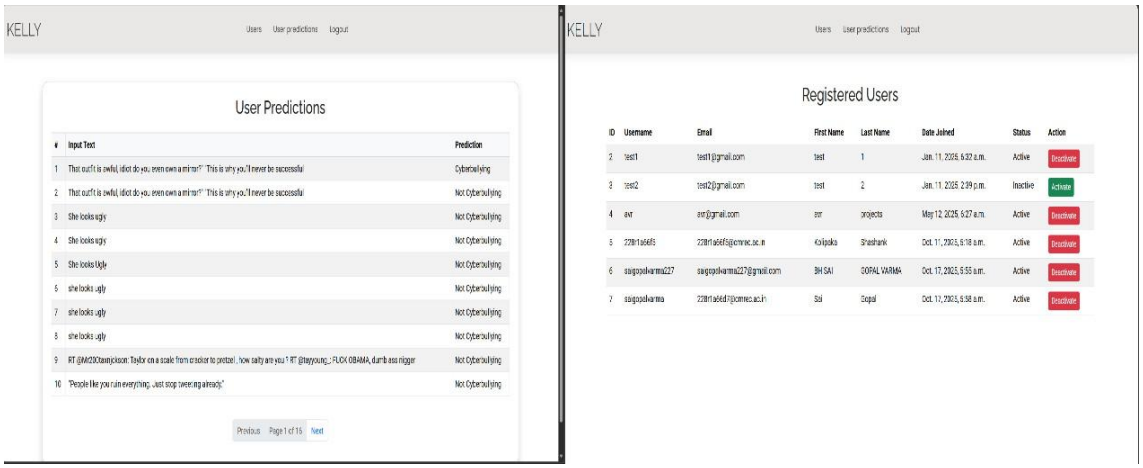


Fig. 7.3.6 Admin Panel

Description: The administrator logs into the system and navigates to the Admin Panel, where they can view both registered users and user prediction history. The system correctly displays user details, account status, and prediction logs, and allows the admin to activate or deactivate user accounts when required. This confirms that administrative monitoring and access-control functions are working correctly, as illustrated in Fig. 7.3.6.

8. RESULTS

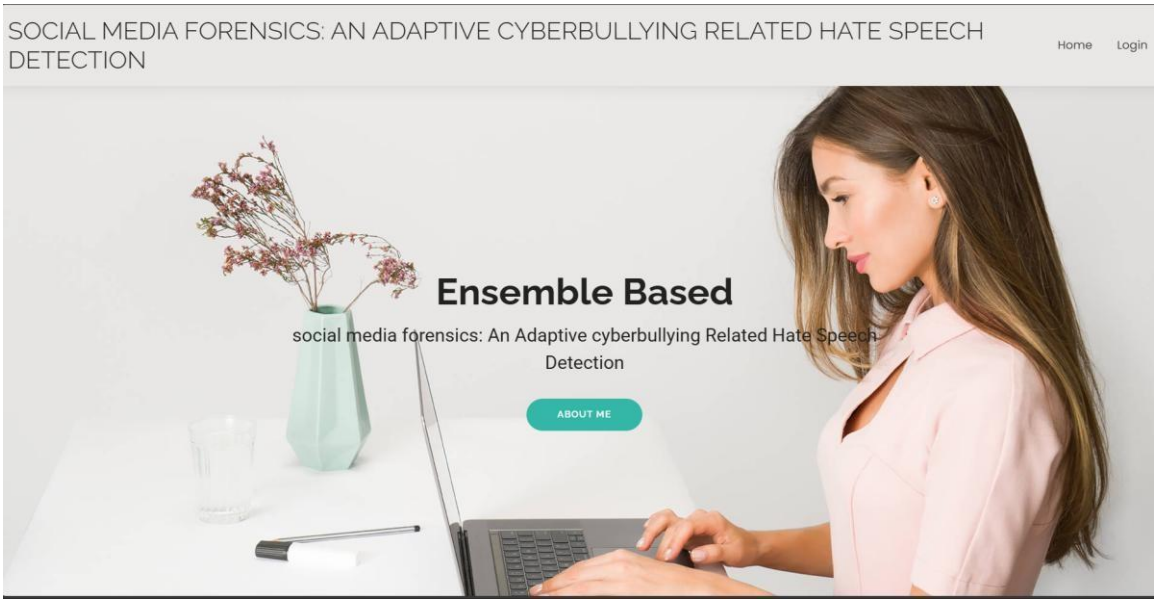


Fig. 8.1 Social Media Forensics Landing Page

Description: The primary interface of the application, titled "Cyberbullying Related Hate Speech Detection" system. It introduces the project as a forensic tool powered by advanced Ensemble Machine Learning methods for identifying offensive digital content, as illustrated in Fig. 8.1.

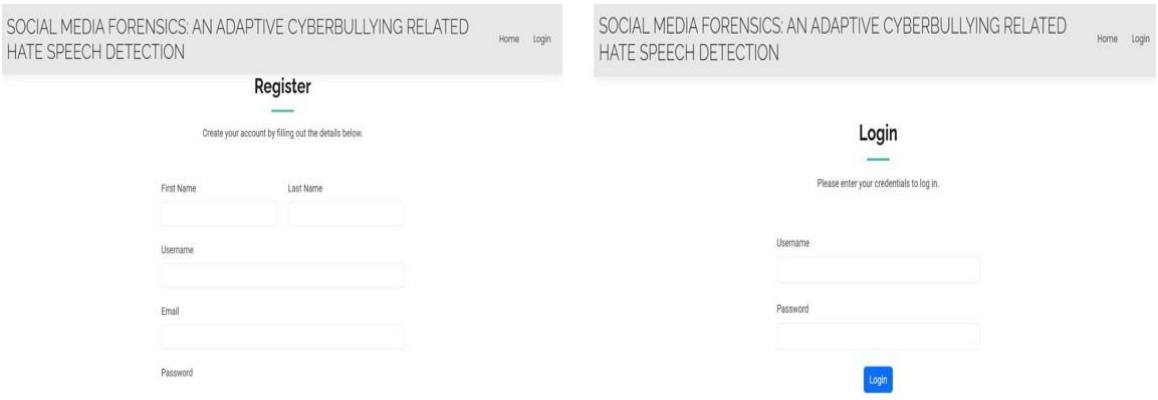


Fig. 8.2 User Authentication & System Access

Description: The entry point for the forensics platform, featuring Registration and Login modules. This interface ensures secure access to the ensemble-based detection tools for social media monitoring, as illustrated in Fig. 8.2.

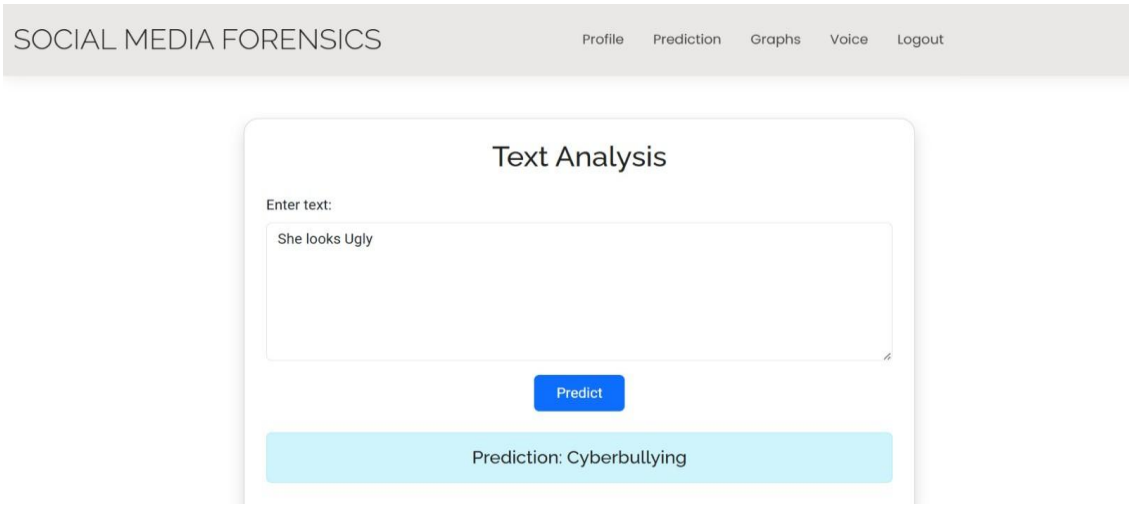


Fig. 8.3 Ensemble-Based Text Classification

Description: This screen shows the Text Analysis module in action. The input text is processed by a Soft-Voting Ensemble, which aggregates the probability scores from both the Boosted Decision Tree and Bagging Random Forest to accurately predict "Cyberbullying", as illustrated in Fig. 8.3.

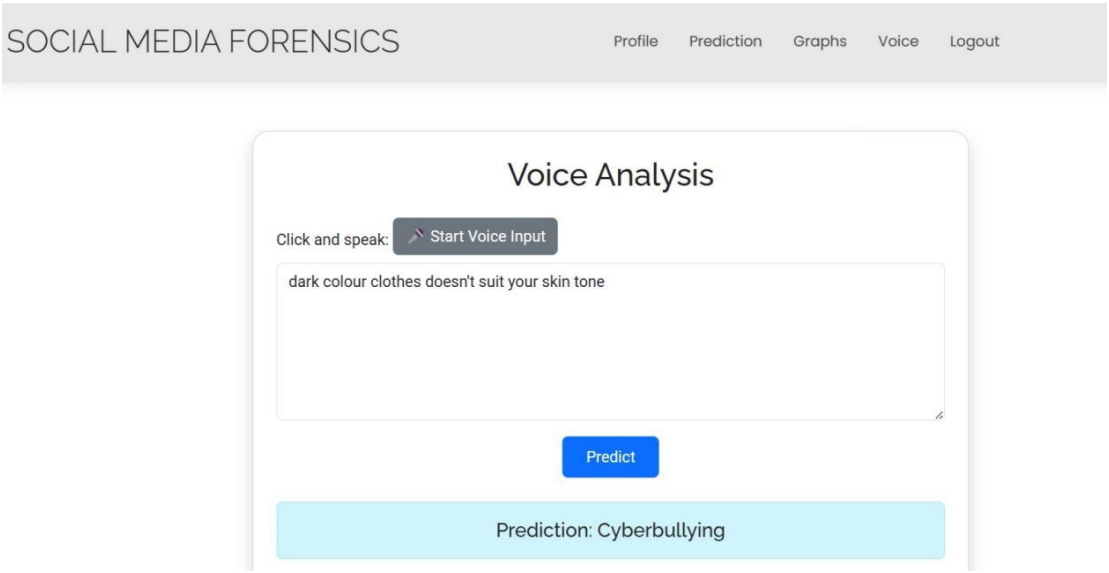


Fig. 8.4 Detection of Subtle Harassment via Voice Input

Description: This interface demonstrates the system flagging a discriminatory remark. By leveraging a combination of Boosting and Bagging techniques, the model is able to detect nuanced offensive speech that single-model architectures might overlook, as shown in Fig. 8.4.

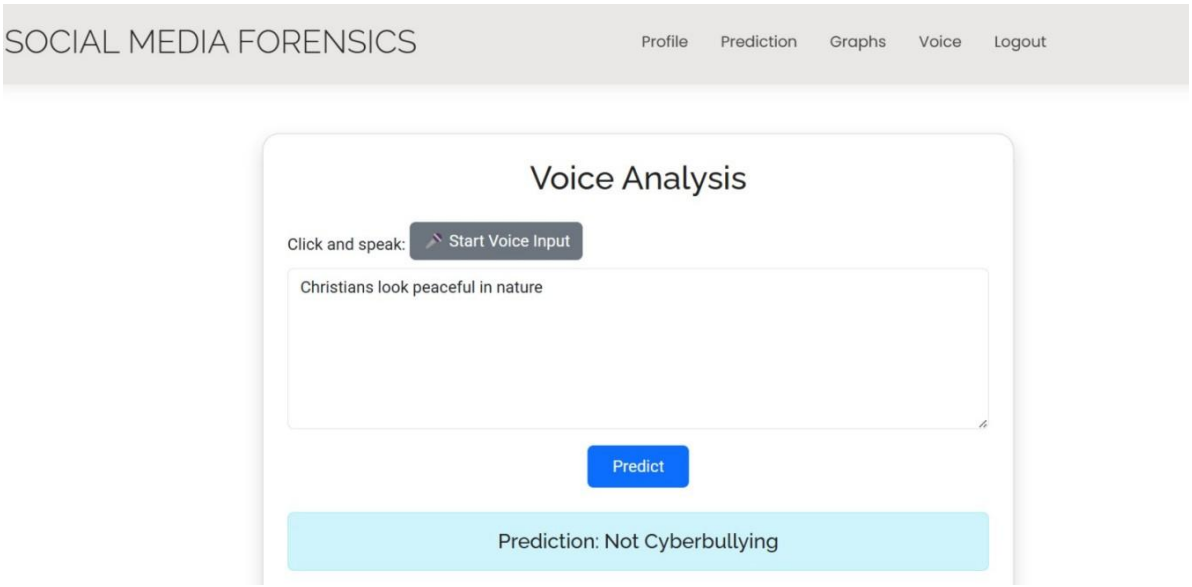


Fig. 8.5 Voice Transcription & Forensic Analysis

Description: This module utilizes speech-to-text conversion to analyze verbal harassment. After transcription, the integrated ensemble model evaluates the text, identifying "Not Cyberbullying" for neutral statements with high precision, as shown in Fig. 8.5.

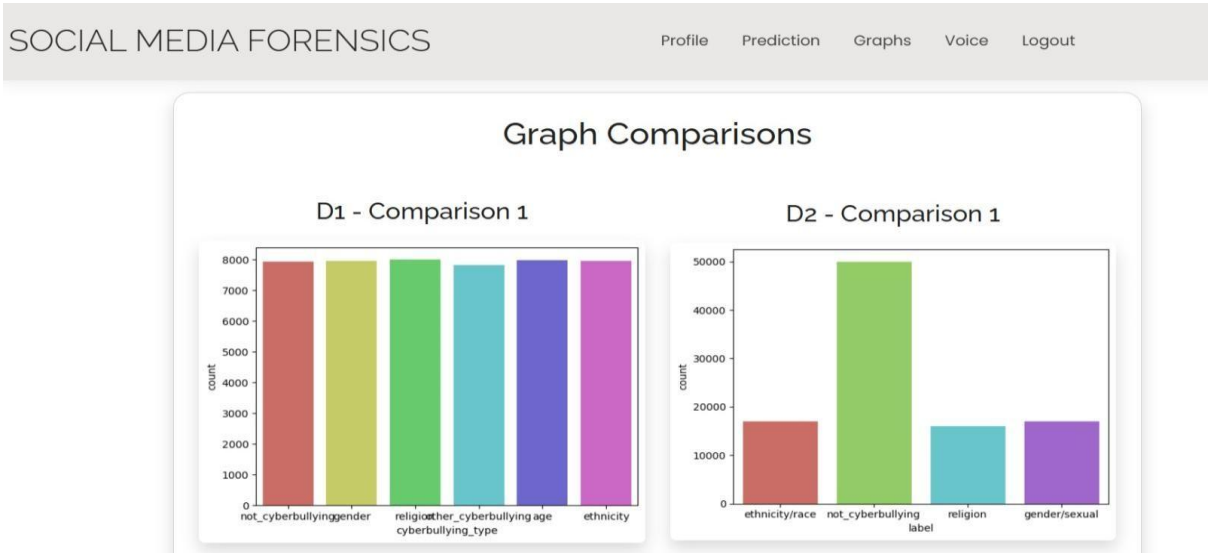


Fig. 8.6 Statistical Data - Class Distribution

Description: A dashboard displaying Graph Comparisons across two different datasets. It represents the count of each class across two different datasets, as shown in Fig. 8.6.

9. CONCLUSION

The present project introduces a comprehensive and intelligent cyberbullying detection framework that effectively leverages ensemble-based machine learning techniques to enhance the accuracy, robustness, and reliability of harmful content classification on social media platforms. The system was implemented using a structured pipeline that includes data collection, preprocessing, feature extraction, imbalance handling, model training, and evaluation, ensuring a systematic and end-to-end approach to solving the problem.

During the implementation phase, raw social media text data was preprocessed through multiple stages, including text cleaning, tokenization, stop-word removal, and normalization, to eliminate noise and improve data quality. The processed text was then transformed into numerical features using the TF-IDF (Term Frequency–Inverse Document Frequency) technique, enabling the machine learning models to capture the importance of words in different contexts. This feature engineering step significantly improved the effectiveness of the classification process.

To address the issue of class imbalance, which is common in cyberbullying datasets, the Synthetic Minority Over-sampling Technique (SMOTE) was applied. SMOTE generates synthetic samples for minority classes, thereby balancing the dataset and preventing the model from being biased toward the majority class. This step enhanced the model's ability to correctly identify minority class instances, particularly subtle or less frequent forms of abusive content.

The core of the system is the ensemble-based classification model, which combines Boosted Decision Tree and Bagging Random Forest algorithms using a soft-voting mechanism. This approach aggregates predictions from multiple models, reducing individual model bias and variance while improving overall generalization. The ensemble model demonstrates strong performance in handling noisy, ambiguous, and linguistically diverse social media text.

The framework supports both binary classification (bullying vs. non-bullying) and multi-class classification (different categories of abusive content), making it adaptable to a wide range of applications. The system was evaluated using standard performance metrics, and the results indicate improved accuracy, stability, and robustness compared to individual classifiers.

The experimental workflow confirms that ensemble learning improves prediction stability compared with individual classifiers, particularly in scenarios involving overlapping or subtle abusive expressions **[12]**. This validates the effectiveness of the proposed hybrid approach and highlights the advantage of combining multiple models for complex natural language processing tasks.

From a system design perspective, the project followed a structured software engineering methodology, including requirement analysis, feasibility study, system architecture design, UML modeling, and workflow visualization. These steps ensured the development of a modular, scalable, and maintainable system architecture. The modular design supports easy integration of additional components and facilitates future enhancements.

Beyond technical contributions, the project addresses a significant societal issue by promoting safer digital communication environments. Cyberbullying has become a major concern in online platforms, affecting individuals' mental health and well-being. The proposed system provides an automated and scalable solution to detect and mitigate harmful interactions. Additionally, the use of open-source tools and standard computational resources ensures cost-effectiveness and accessibility for academic and organizational use.

In conclusion, the project successfully demonstrates that a well-engineered ensemble-based machine learning framework, combined with effective preprocessing, TF-IDF feature extraction, and SMOTE-based imbalance handling, can significantly improve the performance of automated cyberbullying detection systems. The developed system not only achieves its intended objectives but also establishes a strong foundation for future research and real-world deployment.

10. FUTURE ENHANCEMENTS

Although the developed framework provides a strong and effective foundation for cyberbullying detection, several enhancements can be implemented to further improve its performance, scalability, adaptability, and real-world applicability. As cyberbullying continues to evolve in complexity across different platforms, continuous improvements in both modeling techniques and system architecture are essential to maintain high detection accuracy and robustness.

One of the primary directions for future enhancement is the integration of advanced deep learning models such as BERT (Bidirectional Encoder Representations from Transformers), RoBERTa, and LSTM-based architectures. These models are capable of capturing deeper contextual and semantic relationships within textual data, enabling more accurate identification of subtle, implicit, and context-dependent abusive expressions. Transformer-based architectures, in particular, can significantly improve the system's ability to understand sarcasm, slang, and evolving language patterns commonly observed in social media communication. A hybrid approach that combines these deep learning models with the existing ensemble framework can further enhance prediction performance and robustness.

Another important enhancement involves expanding the dataset to include multilingual and cross-domain content. Most current implementations are limited to English-language datasets, which restricts usability in diverse linguistic environments. Extending the system to support multiple languages using multilingual embeddings or transformer models would increase its applicability across different regions and user groups. Additionally, incorporating multimodal data such as images, videos, audio, and combined text-media inputs would allow the system to detect cyberbullying in more complex scenarios, such as memes and visual content, which are increasingly prevalent in online platforms.

The system can also be extended to support real-time monitoring and streaming capabilities. By integrating with live data sources such as social media APIs or message streams, the framework can continuously analyze incoming content and provide instant detection of harmful interactions. This would enable proactive moderation and immediate intervention, making the system suitable for deployment in real-world, large-scale environments such as social media platforms, online forums, and educational systems.

Furthermore, the existing Django-based RESTful backend can be enhanced to improve scalability and performance for production-level deployment. This includes optimizing API endpoints, implementing caching mechanisms, enabling pagination for large datasets, and introducing rate limiting to handle high volumes of requests efficiently. The incorporation of asynchronous processing techniques, such as background task queues, can further improve system responsiveness and reduce latency in real-time detection scenarios.

In addition, introducing adaptive learning mechanisms can significantly improve the system over time. By incorporating feedback loops where users or moderators validate predictions, the system can iteratively retrain and refine its models. This human-in-the-loop approach helps reduce misclassifications, improves generalization, and ensures that the system adapts to evolving patterns of cyberbullying behavior. Periodic retraining with updated datasets can also maintain long-term effectiveness.

From an analytical and usability perspective, the integration of interactive dashboards and visualization tools can enhance the practical utility of the system. These dashboards can provide insights such as frequency of cyberbullying incidents, category-wise distribution, severity analysis, and temporal trends. Such features would assist administrators, researchers, and policymakers in making informed decisions and developing effective prevention strategies.

Finally, strengthening privacy, security, and ethical considerations is essential for real-world deployment. Future improvements can include implementing data anonymization, secure data storage, and compliance with data protection standards. Incorporating fairness-aware machine learning techniques can help minimize bias in predictions, while explainable AI (XAI) methods can provide transparency into model decisions, thereby increasing user trust and system accountability.

Collectively, these enhancements will transform the current framework into a more intelligent, scalable, and user-centered solution. By integrating advanced modeling techniques, expanding data diversity, enabling real-time capabilities, optimizing backend performance, and ensuring ethical deployment, the system can evolve into a comprehensive platform for effective cyberbullying prevention and safer digital communication.

REFERENCES

1. J. Immanuel D., N. S. Nivetha, M. S. Sharmitha, T. K. Sharma, and M. Gowthamani, "Real-Time Cyberbullying Detection Using NLP and Ensemble Learning Techniques," in Proc. 7th Int. Conf. Mobile Computing and Sustainable Informatics (ICMCSI), IEEE, 2026.
2. R. Debnath and P. Banerjee, "Soft-Voting Ensemble Strategies for Toxic Comment Classification," IEEE Transactions on Computational Social Systems, 2025.
3. A. N. Kumar, S. Patel, and R. Mishra, "Boosting-Based Approaches for Online Abuse and Cyberbullying Detection," Applied Intelligence, 2025.
4. I. Uddin, F. Rahman, and T. Kabir, "Voting Classifiers for Improving Toxic Speech Prediction Accuracy," Computers & Electrical Engineering, 2025.
5. H. Allwaibed, A. Alotaibi, and A. Alzahrani, "Cyberbullying Detection Approaches for Arabic Texts Using Ensemble Models," Frontiers in Artificial Intelligence, 2025.
6. Edem Suresh Babu and G. S. Sravanthi, "Sarcasm Detection of Feature Augmentation Using Multiobjective Genetic Algorithm," Evolution in Computational Intelligence, SIST, vol. 436, pp. 457–466, 2025.
7. J. Morales, D. Santos, and P. Castillo, "Ensemble Modeling for Hate Speech Identification Across Platforms," IEEE Access, 2024.
8. T. Saha and R. Kar, "Random-Forest-Driven Cyberbullying Detection on Microblog Data," Journal of Information Security and Applications, 2024.
9. L. Chen, Y. Wang, and Z. Zhao, "A Multi-Stage Ensemble Architecture for Detecting Abusive Language," Information Sciences, 2024.
10. S. Sihab-Us-Sakib, M. Rahman, and M. Hasan, "Cyberbullying Detection of Resource-Constrained Language Using Transformer-Based Ensembles," Natural Language Processing Journal, 2024.
11. I. Hussain, A. Ullah, and M. N. Khan, "Differential Transfer Learning with Ensemble Optimization for Hate Speech Detection," Scientific Reports, 2024.
12. P. Shah and A. Shah, "Comparative Evaluation of Voting, Bagging, and Boosting for Hate Speech Detection," Procedia Computer Science, 2023.

13. M. Jain and S. Gupta, “An Ensemble-Driven Moderation Framework for Social Media Platforms,” IEEE International Conference on Data Science and Advanced Analytics, 2023.

14. R. Biswas, M. Paul, and A. Das, “Deep-Hybrid Ensembles for Detecting Cyberbullying Patterns,” arXiv Preprint, 2023.

15. S. W. Azumah, P. Oppong, and A. Mensah, “Adversarial-Aware Ensemble Approaches for Hate Speech Detection,” arXiv Preprint, 2023.

16. A. Toktarova, S. Khusainova, and M. Fayzullina, “Comparative Study of Ensemble and Single Models for Social Network Hate Speech,” International Journal of Advanced Computer Science and Applications, 2023.

17. M. S. Jahan and M. Oussalah, “A Systematic Review of Hate Speech Detection with Ensemble NLP Models,” Neurocomputing, 2023.

18. T. Mahmud, R. Hasan, and S. Rahman, “Ensemble-Based Detection for Low-Resource Cyberbullying Datasets,” Information Processing & Management, 2023.

19. A. Ahmed and F. Khan, “Machine Learning and Ensemble Techniques for Cyberbullying and Hate Speech,” CEUR Workshop Proceedings, 2023.

20. J. Sathya and F. M. H. Fernandez, “Ontology-Assisted Skip-Gram and Ensemble Models for Cyberbullying Detection,” International Conference on Communication and Electronics Systems (ICCES), 2023.