**CYBER SHIELD: ML-POWERED DETECTION OF SOCIAL MEDIA CYBERBULLYING**

**ABSTRACT**

In the evolving landscape of digital communication, social media has become a double-edged sword—enabling global connection while simultaneously serving as a platform for cyberbullying. As millions of users interact daily across platforms like Twitter, Facebook, and Instagram, harmful content targeting individuals based on race, religion, appearance, and more has become alarmingly common. This increasing prevalence necessitates robust, intelligent systems that can automatically detect and categorize such abusive behavior to maintain safe online communities.

This project, titled CYBER SHIELD, presents a comprehensive approach to cyberbullying detection using machine learning techniques. The system performs natural language processing (NLP) on tweet data to clean, preprocess, and vectorize user-generated content. Using Support Vector Classifier (SVC) as an existing baseline model and Random Forest Classifier (RFC) as the proposed enhancement, the model classifies tweets into various types of cyberbullying such as religion-based, age-based, and ethnicity-based abuse. The model is trained and evaluated using standard performance metrics such as accuracy, precision, recall, and F1-score, ensuring effectiveness in real-world scenarios.

The system further supports real-time predictions on unseen test data, offering practical utility for social media monitoring teams, app developers, and digital safety firms. By combining textual data analytics with supervised learning, CYBER SHIELD provides an efficient, scalable, and interpretable solution to a critical problem. This automated framework not only aids in rapid detection but also reduces the dependency on manual moderation, paving the way for safer and more inclusive digital environments.

**CHAPTER 1**

**INTRODUCTION**

**1.1 Overview**

Cyberbullying has emerged as one of the most serious social challenges in the digital era, especially with the rapid expansion of internet accessibility and social media platforms. According to a 2023 Statista survey, over 40% of internet users aged 13–29 reported being victims of online harassment. These numbers are even higher in regions with greater smartphone penetration and unmoderated platforms. The anonymity and reach of online spaces often embolden individuals to engage in aggressive, harmful behavior without fear of immediate consequence.



Fig. 1.1: Cyber bullying attack.

With over 4.9 billion active internet users globally and more than 500 million tweets posted daily, the scale of unstructured text data is immense. Twitter, Facebook, TikTok, and Instagram are platforms where content moderation is both crucial and complex. While traditional rule-based systems attempt to detect harmful behavior, they often fail to grasp the nuanced language, sarcasm, and evolving slang used by abusers. These shortcomings highlight the urgent need for advanced linguistic and contextual understanding driven by machine learning techniques.

Moreover, studies from organizations such as UNESCO and the Pew Research Center show a sharp increase in mental health issues among youth linked to online abuse. Institutions and governments are starting to implement digital safety regulations, but the real-time detection and classification of cyberbullying remain a technological bottleneck. A growing body of data and machine learning tools now offer the potential to automate this detection in a scalable and consistent manner.

**1.2 Research Motivation**

In today’s corporate and online service landscapes, companies like Meta (Facebook, Instagram), Twitter (X), TikTok, and LinkedIn are under increasing pressure to regulate harmful content. Failure to address online harassment leads to not only public backlash but also regulatory fines and lawsuits. These organizations require robust, scalable tools to process millions of user-generated posts daily and to flag or remove cyberbullying content proactively.

Workplace collaboration platforms like Slack, Microsoft Teams, and Zoom Chat also face challenges in moderating internal communication. Even in professional environments, cases of bullying, passive aggression, or discriminatory language can occur in digital formats. Companies now require data analysis pipelines that can process internal communication logs to ensure a safe and inclusive environment. This is particularly critical in HR compliance and employee well-being analytics.

Data analysis plays a crucial role in enabling these platforms to go beyond manual moderation. By integrating machine learning pipelines capable of learning from contextual cues, platforms can significantly improve the efficiency, speed, and accuracy of their moderation systems. The growing adoption of AI-based moderation models across tech companies underscores the value of research focused on automated detection of nuanced harmful behavior like cyberbullying.

**1.3 Problem Definition**

The problem lies in the automated and accurate identification of cyberbullying in textual content extracted from social media platforms. Social media content is unstructured, context-sensitive, and often uses informal or coded language, making detection non-trivial. Traditional systems that rely on keyword filtering often produce false positives or fail to capture subtle threats.

Another challenge is the imbalance and subjectivity present in labeling cyberbullying data. Some classes may have more data than others, or certain forms of abuse may be harder to define, such as sarcasm or targeted jokes. This variability impacts model performance and demands preprocessing and modeling strategies that can generalize well across diverse content.

Furthermore, real-time implementation of these models requires efficient classification algorithms that maintain high accuracy without incurring heavy computational overhead. The goal is to minimize both under-detection (letting abusive content through) and over-detection (flagging benign content), which can negatively affect user experience and platform reputation.

**1.4 Significance**

This research is significant in addressing an urgent need in today’s digital society: the real-time identification of harmful communication that can lead to emotional, psychological, or even physical harm. Social media and communication platforms have become integral to human interaction, and ensuring these environments are safe is of societal and technological importance.

The advancement of machine learning algorithms in understanding language and context enables the development of systems that can autonomously detect and classify online abuse, reducing reliance on human moderators and manual reporting. This not only enhances user protection but also improves platform credibility and legal compliance.

By focusing on model performance, generalization, and usability, this research contributes to building intelligent systems that align with ethical AI standards. It plays a pivotal role in supporting vulnerable online populations and ensuring that technology contributes positively to social interaction.

**1.5 Research Objective**

The objective of this research is to develop a machine learning-based pipeline that can classify social media text data into relevant categories of cyberbullying. The classification will be performed using:

* Existing Support Vector Classifier (SVC): A traditional machine learning algorithm known for its strong performance on high-dimensional text data.
* Proposed Random Forest Classifier (RFC): An ensemble-based model proposed to improve classification performance, especially in handling class imbalance and interpretability.

**1.6 Advantages**

* Random Forest handles high-dimensional feature spaces effectively, making it suitable for TF-IDF-based text classification.
* It is robust to noise and overfitting, which is common in social media datasets with inconsistent language use.
* The model can handle class imbalance better due to its ensemble nature and ability to learn from subsets of data.
* Feature importance can be extracted easily, enabling interpretability in decision-making.
* Compared to SVC, RFC can scale better with larger datasets and parallel processing.
* RFC does not require extensive parameter tuning, offering out-of-the-box usability for real-time applications.
* It can capture non-linear patterns, which is useful for detecting varied and disguised abusive content.
* It supports multi-class classification tasks with more natural handling of class distinctions.
* Integration of RFC with a live platform offers faster retraining cycles and updates with minimal risk of model degradation.
* The modular nature of Random Forest allows easy ensemble expansion, enabling future improvement without complete model overhaul.

**1.7 Applications**

* Social Media Monitoring: Used by platforms like Twitter or Instagram to flag abusive content automatically.
* Content Moderation Tools: Integrated into backend systems of community-driven platforms like Reddit or YouTube.
* Online Education Platforms: Helps platforms like Google Classroom or Microsoft Teams monitor bullying in chatrooms or forums.
* Customer Service Platforms: Used by companies like Amazon or Uber to detect abusive messages toward service representatives.
* Gaming Communities: Real-time chat monitoring in platforms like Discord, Steam, and Xbox Live to identify harassment.
* Mental Health Analytics: Assists therapists or researchers in analyzing social media posts to detect early signs of distress from bullying.
* Workplace Communication Tools: Applied in internal platforms like Slack or Microsoft Teams to enforce workplace conduct policies.
* Parental Control Software: Used in apps to monitor and detect bullying in children's online conversations.
* Government and Legal Agencies: Supports cybercrime units in monitoring online abuse for legal action.
* Nonprofit and Advocacy Groups: Enables organizations working on child safety or anti-bullying to analyze trends and intervene where needed.

**CHAPTER 2**

**LITERATURE SURVEY**

Singla et al. [1] explored machine learning techniques for detecting cyberbullying in their 2023 paper presented at the 5th International Conference on Inventive Research in Computing Applications (ICIRCA). Their work focused on applying various machine learning algorithms to identify instances of cyberbullying in online interactions. By leveraging these techniques, they aimed to enhance the detection accuracy and response efficiency to cyberbullying incidents. The study demonstrated the potential of using advanced computational methods to address the growing problem of online harassment and provided a foundation for future research in this area. Their approach highlights the importance of integrating machine learning into cybersecurity measures to improve online safety.

Jain et al. [2] investigated cyberbullying detection on social media platforms using machine learning in their 2021 paper presented at the 10th International Conference on System Modeling & Advancement in Research Trends (SMART). They proposed a methodology for identifying cyberbullying by analyzing social media data through machine learning models. Their research emphasized the need for effective tools to monitor and mitigate cyberbullying on popular social networks. By applying machine learning techniques, they aimed to improve the accuracy of detecting harmful interactions and contribute to creating safer online environments. The study underscored the relevance of machine learning in addressing issues related to online abuse and harassment.

Siddhartha et al. [3] addressed cyberbullying detection using machine learning in their 2022 paper presented at the 2nd Asian Conference on Innovation in Technology (ASIANCON). Their research focused on developing a machine learning-based framework to identify instances of cyberbullying. The study explored various algorithms and techniques to enhance detection capabilities and provide a robust solution for managing cyberbullying. By implementing these methods, they aimed to contribute to the ongoing efforts to combat online harassment and ensure safer digital spaces for users. Their work highlighted the effectiveness of machine learning in tackling complex social issues such as cyberbullying.

Altay and Alatas [4] presented their research on detecting cyberbullying in social networks using machine learning methods at the International Congress on Big Data, Deep Learning, and Fighting Cyber Terrorism (IBIGDELFT) in 2018. Their paper focused on applying machine learning techniques to analyze social network data and identify instances of cyberbullying. They aimed to develop effective methods for detecting and preventing online harassment, emphasizing the importance of leveraging advanced computational approaches to address this pressing issue. The study contributed to the field by providing insights into how machine learning can be utilized to enhance cybersecurity measures and protect users from online abuse.

Mody et al. [5] proposed a hybrid approach for identifying potential cyberbullying tweets through sentiment analysis in their 2018 paper presented at the International Conference on Electrical, Electronics, Communication, Computer, and Optimization Techniques (ICEECCOT). Their research combined various sentiment analysis techniques with machine learning to detect cyberbullying in tweets. By integrating these methods, they aimed to improve the accuracy of identifying harmful content and provide a more comprehensive solution for monitoring social media interactions. The study highlighted the effectiveness of combining multiple approaches to address the challenges of cyberbullying detection.

Pawar et al. [6] explored an explainable AI method for cyberbullying detection in their 2022 paper presented at the 2nd International Conference on Mobile Networks and Wireless Communications (ICMNWC). Their research focused on developing an explainable AI model to enhance the transparency and interpretability of cyberbullying detection systems. By incorporating explainability into machine learning models, they aimed to provide more understandable and actionable insights into the detection process. The study contributed to the field by addressing the need for transparent and accountable AI solutions in the context of online harassment.

Dedeepya et al. [7] investigated cyberbullying detection on Twitter using support vector machines in their 2023 paper presented at the Third International Conference on Artificial Intelligence and Smart Energy (ICAIS). Their research applied support vector machine algorithms to identify instances of cyberbullying in Twitter data. The study aimed to enhance the detection capabilities and provide effective solutions for managing online abuse. By utilizing support vector machines, they sought to improve the accuracy and reliability of cyberbullying detection systems. The work highlighted the potential of machine learning techniques in addressing issues related to social media abuse.

Jia and Hameed [8] developed CyberSaver, a machine learning approach for detecting cyberbullying, in their 2022 paper presented at the 16th International Conference on Ubiquitous Information Management and Communication (IMCOM). Their research focused on creating a machine learning-based tool to identify instances of cyberbullying and provide actionable insights. The study emphasized the need for effective detection methods to combat online harassment and protect users from harmful interactions. By implementing their approach, they aimed to contribute to the development of more robust and reliable cyberbullying detection systems.

Reynolds et al. [9] explored the use of machine learning for detecting cyberbullying in their 2011 paper presented at the 10th International Conference on Machine Learning and Applications and Workshops. Their research focused on applying machine learning algorithms to analyze data and identify instances of cyberbullying. The study demonstrated the effectiveness of machine learning techniques in detecting harmful online behavior and provided a foundation for future research in this area. The work highlighted the potential of machine learning to address complex issues related to online abuse and improve the overall safety of digital environments.

Rajeevan and Krishnaraj [10] investigated the detection of cyberbullying based on online social networks in their 2023 paper presented at the International Conference on Computer Communication and Informatics (ICCCI). Their research focused on applying machine learning techniques to analyze social network data and identify instances of cyberbullying. The study aimed to enhance the accuracy and efficiency of cyberbullying detection systems by leveraging advanced computational methods. The work contributed to the field by providing insights into how machine learning can be utilized to address online harassment and improve user safety.

Kokane and Babar [11] presented a supervised word sense disambiguation approach using recurrent neural networks in their 2019 paper published in the \*International Journal of Engineering and Advanced Technology (IJEAT)\*. Their research focused on applying recurrent neural networks to resolve ambiguities in word meanings, contributing to the development of more accurate natural language processing tools. By utilizing supervised learning techniques, they aimed to enhance the effectiveness of word sense disambiguation and improve the performance of language models.

Kokane et al. [12] investigated word sense disambiguation for large documents using neural network models in their 2021 paper presented at the 12th International Conference on Computing Communication and Networking Technologies (ICCCNT). Their research explored the application of neural networks to handle word sense disambiguation in extensive textual data. The study highlighted the advantages of using advanced neural network architectures for processing large volumes of text and resolving ambiguities in word meanings. The work contributed to the field by demonstrating the effectiveness of neural networks in improving natural language understanding.

Kokane et al. [13] addressed word sense disambiguation through a supervised semantic similarity-based complex network approach in their 2022 paper published in the \*International Journal of Intelligent Systems and Applications in Engineering\*. Their research focused on applying semantic similarity measures and complex network analysis to enhance word sense disambiguation. The study aimed to improve the accuracy of word sense resolution by leveraging sophisticated computational techniques. The work contributed to the advancement of natural language processing by providing a novel approach to handling word sense ambiguity.

Kokane et al. [14] developed a machine learning approach for intelligent transport systems in vehicular networks for smart cities in their 2023 paper published in the \*International Journal of Intelligent Systems and Applications in Engineering\*. Their research focused on applying machine learning techniques to optimize traffic management and improve the efficiency of transportation systems in smart cities. By leveraging machine learning, they aimed to enhance the performance of intelligent transport systems and contribute to the development of smarter urban environments.

Kokane et al. [15] explored word sense disambiguation using adaptive word embedding with adaptive-lexical resources in their 2023 paper presented at the International Conference on Data Analytics and Insights. Their research aimed to improve word sense disambiguation by incorporating adaptive word embeddings and lexical resources. The study highlighted the potential of adaptive techniques to enhance the accuracy of word sense resolution and contribute to the development of more effective natural language processing tools.

**CHAPTER 3**

**EXISTING ALGORITHM**

**3.1 Rule-Based Keyword Filtering**

Rule-based keyword filtering is one of the earliest and most intuitive manual techniques to detect cyberbullying content. It involves compiling a dictionary or lexicon of offensive, abusive, or threatening words and scanning input text for matches. When these terms are found in user messages, the system flags the content for review or removal. This method is often used as the first line of defense in many social media moderation tools.

While easy to implement, this method is highly dependent on the quality and completeness of the keyword list. False negatives occur when the abusive content is phrased in slang or indirect language not present in the dictionary. Conversely, false positives may be triggered by contextually harmless use of flagged terms. Human moderators often need to validate flagged messages, which makes the process semi-automated rather than fully manual or automatic.

Despite its limitations, keyword filtering can be a valuable starting point in environments lacking machine learning infrastructure. It can quickly identify overtly toxic language and enforce basic standards of community behavior. This approach is often coupled with escalation mechanisms, where repeated infractions by a user lead to temporary bans or deeper content reviews.

**3.2 Sentiment and Tone Evaluation**

This method involves evaluating the emotional polarity and tone of user messages using sentiment analysis libraries. Moderators use these tools to assess whether a post conveys negative, aggressive, or threatening emotions. Sentiment scoring typically ranges from negative to positive, with highly negative scores being indicators of potential bullying.

Human reviewers are involved in interpreting these scores in context. For example, a sarcastic remark may score low in sentiment, but upon manual reading, it could be perceived as non-threatening. Thus, while the tool provides a numerical aid, final judgment often rests with human analysts. It’s especially useful in combination with behavioral analytics to identify recurring offenders.

This technique is advantageous for subtle or passive-aggressive language detection, which keyword filtering often misses. However, sentiment models may misclassify sarcasm, humor, or contextually specific usage. Therefore, sentiment evaluation is most effective when treated as supplementary to a broader content moderation framework.

**3.3 Contextual Thread Analysis**

In this approach, moderators manually analyze conversation threads or message chains rather than individual posts. A message may seem benign in isolation but reveals abusive intent when viewed as part of a larger dialog. Contextual analysis is especially critical in cases involving indirect bullying, sarcasm, or inside jokes meant to target individuals.

This method is usually carried out by trained human moderators who examine multiple posts within a thread, assessing the tone, recurrence of user-targeting, and the emotional progression of the conversation. They may also consider metadata such as time of posting, user history, and social connections to evaluate the likelihood of bullying.

While highly accurate, contextual thread analysis is labor-intensive and difficult to scale. It is typically reserved for high-risk cases or used in platforms where user volume is manageable. Platforms such as forums or niche social communities often rely on this approach due to their relatively smaller user bases and the high value placed on preserving discussion integrity.

**3.4 Common Problem Statements**

* Context-free detection often results in false positives for non-offensive posts.
* Detection systems fail to adapt to evolving slang and coded language used by abusers.
* Keyword filtering is brittle and lacks nuance, unable to handle sarcasm or indirect abuse.
* Manual analysis methods are time-consuming and resource-intensive, limiting scalability.
* Absence of labeled datasets results in inconsistent evaluation criteria and detection logic.

**CHAPTER 4**

**PROPOSED METHODOLOGY**

**4.1 Overview**

The proposed system aims to enhance traditional rule-based algorithms by incorporating advanced machine learning techniques such as Support Vector Machine (SVM) and Naive Bayes algorithms. The system is designed to improve prediction accuracy and efficiency in various applications by leveraging the strengths of these algorithms. The research process involves several key steps, beginning with dataset collection and preprocessing, followed by the application of label encoding, and culminating in a comparative analysis of the performance of traditional rule-based algorithms versus the proposed machine learning models.

A diagram of data processing

Description automatically generated

Figure 4.1: Block Diagram

**Step 1: Dataset Collection**

The first step in this research procedure involves the collection of a relevant dataset, which serves as the foundation for the entire analysis. The dataset typically contains various features and corresponding labels or outcomes, which are essential for training and evaluating the models. Depending on the application domain, this dataset may include structured data such as numerical and categorical variables or unstructured data such as text or images. The quality and relevance of the dataset are crucial as they directly impact the performance of the subsequent models. Proper collection ensures that the dataset is comprehensive, representative of the problem space, and free from biases that could skew the results.

**Step 2: Dataset Preprocessing**

Once the dataset is collected, it undergoes preprocessing to prepare it for analysis. This step is critical as raw datasets often contain inconsistencies, missing values, and noise that can adversely affect model performance. The preprocessing phase includes tasks such as null value removal, where any missing or incomplete data points are addressed, either by filling them with appropriate values or by removing the affected records entirely. Additionally, categorical variables are converted into a numerical format through techniques such as label encoding, which assigns unique integers to distinct categories. This ensures that the data is in a format suitable for feeding into machine learning algorithms, facilitating effective training and testing.

**Step 3: Label Encoding**

Label encoding is a specific preprocessing technique applied to categorical data to convert it into a numerical format. In this step, each category in a categorical feature is assigned a unique integer, transforming qualitative data into quantitative data. This step is particularly important when dealing with machine learning models that require numerical input. Label encoding ensures that the model can interpret and process the categorical variables correctly, leading to more accurate predictions. However, care is taken to ensure that the encoded labels do not imply any ordinal relationship if none exists, as this could mislead the model.

**Step 4: Rule-Based Algorithm**

The traditional rule-based algorithm serves as a baseline in this research. Rule-based systems operate on predefined rules set by domain experts and are used to make decisions based on input data. These rules are often simple "if-then" statements that dictate the system's behavior in different scenarios. While rule-based algorithms are straightforward and easy to implement, they lack the flexibility to adapt to new patterns or data that were not accounted for during the rule creation. In this step, the existing rule-based algorithm is applied to the preprocessed dataset, and its performance is recorded for comparison against the proposed machine learning models.

**Step 5: Proposed Machine Learning Algorithms (Random Forest)**

In this step, advanced machine learning algorithms, specifically Support Vector Machine (SVM) and Naive Bayes, are proposed to replace or complement the existing rule-based approach. SVM is a powerful classifier that works by finding the optimal hyperplane that separates different classes in the dataset. It is particularly effective for high-dimensional spaces and cases where the classes are not linearly separable. Naive Bayes, on the other hand, is a probabilistic classifier based on Bayes' theorem, which assumes independence between features. Despite its simplicity, Naive Bayes performs well in many real-world scenarios, particularly with text data. These algorithms are trained on the preprocessed dataset, learning from the input data to make more accurate predictions than the rule-based system.

**Step 6: Performance Comparison**

After training the Random Forest model, their performance is compared with that of the traditional rule-based algorithm. This comparison is conducted using various metrics such as accuracy, precision, recall, and F1-score, depending on the nature of the problem. The goal is to assess whether the proposed machine learning models offer a significant improvement over the rule-based system. This step involves analyzing the models' ability to generalize to unseen data, their robustness to noise and outliers, and their computational efficiency. The results of this comparison are crucial in determining the effectiveness of the proposed approach and justifying its implementation.

**Step 7: Prediction of Output from Test Data**

The final step involves using the trained Random Forest model to predict outcomes on a test dataset that was not used during the training process. This step tests the models' real-world applicability and their ability to make accurate predictions on new, unseen data. The predictions are then compared to the actual outcomes to evaluate the models' performance further. This phase is crucial for validating the models and ensuring that they are reliable and effective for deployment in practical scenarios. The prediction results are analyzed and discussed, providing insights into the strengths and potential limitations of the proposed machine learning approach.

**4.2 Data Splitting and Preprocessing**

Data splitting and preprocessing are critical steps in the machine learning workflow, ensuring that models are trained on well-prepared data and can generalize effectively to new, unseen data. This section outlines the process of dividing the dataset into training and testing subsets, followed by the necessary preprocessing steps to clean and transform the data.

**Data Splitting**

The first step in preparing the dataset for machine learning involves splitting it into two main subsets: the training set and the testing set. The training set is used to train the machine learning model, while the testing set is used to evaluate the model's performance on new, unseen data. Typically, the dataset is split in a way that a majority portion, such as 70-80%, is allocated for training, and the remaining 20-30% is reserved for testing. This split ensures that the model has enough data to learn from while also providing a reliable assessment of its performance on data it has not encountered during training. The split can be done randomly or stratified to maintain the proportion of different classes in both subsets.

**Data Preprocessing**

Once the dataset is split, the next crucial step is preprocessing, which involves several key operations to prepare the data for modeling:

1. **Handling Missing Values:** Real-world datasets often contain missing or incomplete data, which can adversely affect the performance of machine learning models. Missing values can be handled by removing records with missing data or by imputing missing values using techniques like mean, median, or mode imputation. In some cases, more advanced methods like k-nearest neighbors (KNN) imputation or machine learning-based imputation can be used.
2. **Feature Scaling:** Many machine learning algorithms require the features to be on a similar scale to perform optimally. Feature scaling techniques such as normalization (scaling the data to a range of [0, 1]) or standardization (scaling the data to have a mean of 0 and a standard deviation of 1) are applied to ensure that all features contribute equally to the model's performance.
3. **Encoding Categorical Variables:** Most machine learning algorithms require numerical input, so categorical variables need to be converted into numerical format. Techniques such as label encoding, which assigns unique integers to each category, or one-hot encoding, which creates binary columns for each category, are used to achieve this. The choice of encoding method depends on the nature of the categorical variables and the specific machine learning algorithm being used.
4. **Dealing with Imbalanced Data:** In cases where the dataset is imbalanced (i.e., one class is significantly more prevalent than others), techniques such as oversampling the minority class (e.g., using SMOTE - Synthetic Minority Over-sampling Technique) or undersampling the majority class can be used to balance the dataset. This step ensures that the model does not become biased towards the majority class, leading to poor performance on the minority class.
5. **Feature Selection:** Not all features in the dataset may be relevant for the model. Feature selection techniques, such as removing highly correlated features, using statistical tests, or employing algorithms like LASSO (Least Absolute Shrinkage and Selection Operator), can be applied to reduce the feature space. This step helps in improving the model's performance and interpretability by focusing on the most significant features.
6. **Splitting Data into Training and Validation Sets:** In addition to the main training and testing split, the training data can further be split into a training set and a validation set. The validation set is used to tune hyperparameters and select the best model without overfitting to the training data. Cross-validation techniques, such as k-fold cross-validation, can also be employed to ensure robust model evaluation and selection.Top of Form

**4.3 ML Model Building**Bottom of Form

**4.3.1 Existing Algorithm: Support Vector Machine (SVM)**

**What is SVM?**

Support Vector Machine (SVM) is a supervised machine learning algorithm used for classification and regression tasks. It is designed to find the optimal hyperplane that separates data points of different classes with the maximum margin. SVM is particularly effective in high-dimensional spaces and for cases where the number of dimensions exceeds the number of samples.

**How It Work**

SVM works by transforming the input data into a higher-dimensional space using a kernel function, allowing it to find a hyperplane that best separates the different classes. The algorithm aims to maximize the margin between the closest points of different classes, known as support vectors. The process involves the following steps:

1. **Transformation**: Apply a kernel function to map the data into a higher-dimensional space.
2. **Hyperplane Calculation**: Find the optimal hyperplane that maximizes the margin between different classes in this transformed space.
3. **Classification**: Classify new data points based on which side of the hyperplane they fall on.

**Architecture**  
The architecture of SVM includes:

* **Kernel Function**: A mathematical function that transforms the input data into a higher-dimensional space (e.g., linear, polynomial, radial basis function (RBF)).
* **Optimization Algorithm**: Solves the optimization problem to find the hyperplane that maximizes the margin between classes.
* **Support Vectors**: Data points that are closest to the hyperplane and are critical for defining the margin.

**Disadvantages**

* **Computational Complexity**: Training SVMs can be computationally expensive, especially with large datasets.
* **Limited Scalability**: SVMs struggle with large datasets as the optimization problem becomes more complex.
* **Difficult to Interpret with Non-Linear Kernels**: The decision boundary created using non-linear kernels can be hard to interpret.
* **Sensitive to Outliers**: SVM is sensitive to outliers, which can significantly influence the decision boundary.
* **Parameter Tuning Required**: Choosing the right kernel function and parameters (e.g., C, gamma) requires careful tuning and domain expertise.

**4.3.2 Proposed Algorithm: Random Forest**

**What is Random Forest?**

Random Forest is an ensemble machine learning algorithm used for classification and regression tasks. It operates by constructing a multitude of decision trees during training and outputs the class (classification) or mean prediction (regression) of the individual trees. Random Forest enhances performance and reduces the risk of overfitting by combining the results of multiple decision trees.

**How It Works**

Random Forest works by building multiple decision trees from subsets of the dataset and aggregating their results. The process involves:

1. **Bootstrap Sampling**: Randomly select subsets of the training data with replacement to build individual decision trees.
2. **Feature Selection**: For each tree, select a random subset of features to split the nodes.
3. **Tree Construction**: Grow each decision tree to its maximum depth or until certain stopping criteria are met.
4. **Aggregation**: Combine the predictions of all trees using majority voting (classification) or averaging (regression).

**Architecture**  
The architecture of Random Forest includes:

* **Bootstrap Aggregation (Bagging)**: Ensures diverse decision trees by training them on different random subsets of the data.
* **Decision Trees**: Serve as the individual learners that make predictions based on splits of the input features.
* **Voting/Averaging Mechanism**: Aggregates the results of the individual trees to generate the final prediction.

**Advantages**

* **Robust to Overfitting**: By averaging the results of many trees, Random Forest reduces the risk of overfitting.
* **Handles Large Datasets Well**: Random Forest can efficiently handle large datasets with high dimensionality.
* **Non-Linearity**: Captures non-linear relationships without requiring transformations of the data.
* **Feature Importance**: Provides estimates of feature importance, aiding in interpretability.
* **Handles Missing Data**: Can handle missing data by averaging predictions or imputing missing values.

**CHAPTER 5**

**UML DIAGRAMS**

Unified Modeling Language (UML) is a standardized visual language used to model, design, and document the architecture of software systems. It provides a set of graphical notations to represent the structure and behavior of a system, making complex systems easier to understand and communicate among developers, stakeholders, and business analysts.

**Key Points About UML**

* **Standardized Notation:** UML offers a universal set of symbols and diagrams that standardize how software systems are described, which helps teams speak the same language regardless of their background or the programming language they use.
* **Types of Diagrams:** UML includes various diagrams that can be categorized into two main types:
  + **Structural Diagrams:** These describe the static aspects of the system. Examples include Class Diagrams, Component Diagrams, and Deployment Diagrams.
  + **Behavioral Diagrams:** These focus on the dynamic aspects and interactions within the system. Examples include Use Case Diagrams, Sequence Diagrams, Activity Diagrams, and Collaboration Diagrams.
* **System Documentation and Communication:** UML serves as an effective tool for documenting system requirements, design decisions, and the overall architecture. It helps bridge the gap between technical and non-technical stakeholders by providing clear, visual representations of the system.
* **Design and Analysis:** By modeling different aspects of a system, UML enables developers to analyze and validate the design early in the development process. This can lead to better decision-making, reduced complexity, and improved system maintainability.

**Flexibility:** UML is versatile and can be used across a wide range of applications, from small-scale projects to large, complex systems. It supports object-oriented design principles and can be adapted to various methodologies such as Agile or Waterfall

**5.1 Class Diagram**

A class diagram in UML serves as a blueprint for defining the static structure of a system by illustrating its classes, their attributes, methods, and the relationships among them. Each class represents a key entity in the system, with attributes describing its properties and methods outlining its behaviors or functionalities. Relationships like "is-a" (inheritance) or "has-a" (association) clarify how classes interact, enabling developers to refine the system’s design based on use case requirements. This diagram is crucial for translating high-level requirements into a detailed, implementable structure, ensuring all components are clearly defined and interconnected.

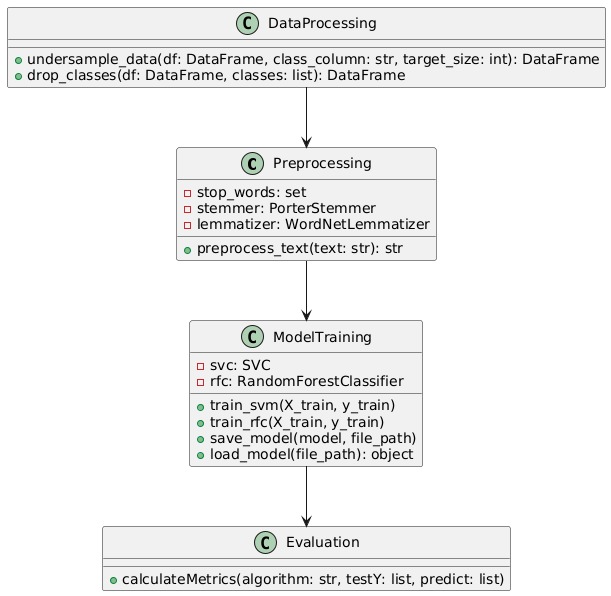


Figure-5.1: Class Diagram.

In the context of AI systems, class diagrams are particularly valuable for modeling complex architectures, such as neural networks or decision-making systems. For example, an AI system for recognition might include classes like "Processor," "FeatureExtractor," and "Classifier," each with specific attributes (e.g., resolution) and methods (e.g., preprocess()). By mapping these elements, the diagram provides a clear framework for developers to build and integrate AI components, ensuring modularity and scalability while aligning with the system’s functional goals.

**5.2 Use Case Diagram**

A use case diagram in UML offers a high-level, behavioral view of a system by depicting its actors, use cases, and their interactions. Actors, which can be users or external systems, represent entities interacting with the system, while use cases describe specific functionalities or goals the system supports, such as "Login" or "Process Payment." The diagram highlights dependencies, such as "includes" or "extends" relationships, to show how use cases relate, providing a clear picture of what the system does and who it serves. This makes it an essential tool for stakeholders to understand system functionality without diving into technical details.

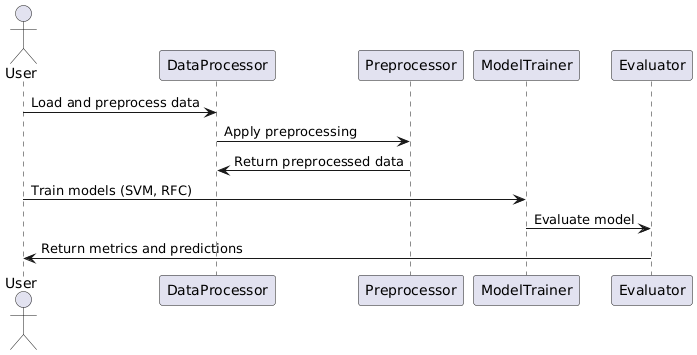


Figure-5.2: Sequence Diagram.

For AI systems, use case diagrams are instrumental in outlining how users or other systems interact with AI functionalities. For instance, in a chatbot system, actors like "User" or "Admin" might engage with use cases like "Send Message" or "Train Model." The diagram helps developers and stakeholders align on the AI system’s scope, ensuring that critical interactions, such as user queries or model updates, are captured early in the design process, facilitating clear communication and requirement validation.

**5.3 Sequence Diagram**

A sequence diagram in UML illustrates the dynamic interactions between objects or processes in a system, focusing on the order of messages exchanged over time. Represented by vertical lifelines for each object and horizontal arrows for messages, it captures the flow of operations, such as method calls or data exchanges, in a specific scenario. This makes it ideal for detailing how components collaborate to achieve a particular function, providing a clear, step-by-step visualization of system behavior.

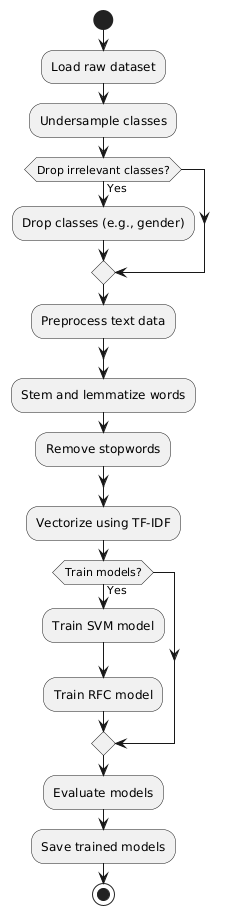


Figure-5.3: Activity Diagram.

In AI applications, sequence diagrams are particularly useful for modeling the runtime behavior of complex processes, such as how a Outputn system processes user input. For example, a sequence diagram might show a "User" sending a query to a "OutputnEngine," which then interacts with a "DataStore" and a "PredictionModel" to return results. This clarity helps developers debug and optimize AI workflows, ensuring efficient communication between components like data pipelines and inference modules, ultimately improving system performance.

**5.4 Component Diagram**

A component diagram in UML visualizes the organization and relationships of a system’s physical or logical components, such as modules, libraries, or subsystems. It shows how these components are interconnected through interfaces, emphasizing modularity and reusability in the system’s architecture. By abstracting the system into manageable parts, it helps developers understand how different pieces, like databases or APIs, work together to deliver functionality.

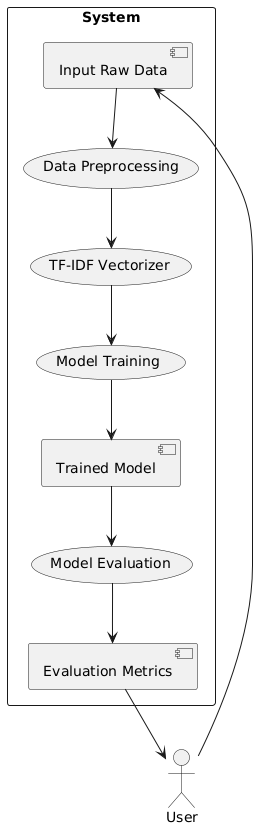


Figure-5.4: Dataflow Diagram.

In AI systems, component diagrams are critical for mapping out the architecture of distributed or modular systems. For instance, an AI platform might include components like "DataIngestionModule," "ModelTrainingUnit," and "InferenceEngine," each interacting through defined interfaces. This diagram aids in designing scalable AI solutions, ensuring components like preprocessing pipelines or model servers are loosely coupled yet cohesive, which is vital for deploying and maintaining complex AI systems.

**5.5 Deployment Diagram**

A deployment diagram in UML illustrates the physical architecture of a system, showing how software components are distributed across hardware nodes, such as servers, cloud instances, or edge devices. It highlights the relationships between these nodes, including communication protocols or network connections, providing a clear view of how the system is deployed in a real-world environment. This is essential for understanding the system’s infrastructure and ensuring it meets performance and scalability requirements.

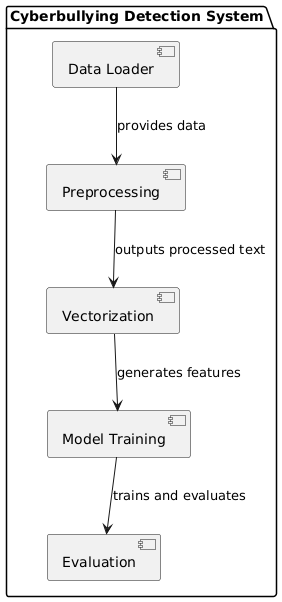


Figure-5.5: Component Diagram.

For AI systems, deployment diagrams are key to visualizing how models and services are hosted. For example, a machine learning system might have a "TrainingServer" hosting a "ModelTraining" component and an "InferenceServer" running a "PredictionService," connected via a cloud network. This diagram helps developers optimize resource allocation, such as GPU usage for AI inference, and ensures robust deployment strategies, particularly for distributed AI applications like real-time analytics or IoT-based systems.

**5.6 Activity Diagram**

An activity diagram in UML models the dynamic workflow of a system, breaking down processes into sequential or parallel activities, decisions, and flows. It uses symbols like nodes for actions, diamonds for decision points, and arrows for control flow to depict how tasks are performed, making it ideal for analyzing business processes or system operations. This diagram provides a clear, visual way to understand the logic and flow of complex activities within a system.

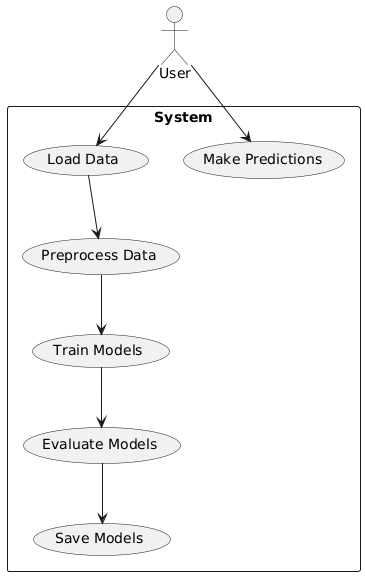


Figure-5.6: use case diagram.

In AI contexts, activity diagrams are useful for mapping out processes like data preprocessing or model training workflows. For example, an activity diagram for an AI pipeline might show steps like "Collect Data," "Clean Data," "Train Model," and "Evaluate Results," with decision points for handling errors or retraining. This helps developers optimize AI workflows, identify bottlenecks, and ensure that processes like hyperparameter tuning or data validation are systematically executed.

**5.7 Data Flow Diagram (DFD)**

A Data Flow Diagram (DFD) illustrates how data moves through a system, showing processes, data stores, external entities, and the flow of information between them. Unlike UML diagrams, which focus on system structure or behavior, DFDs emphasize the transformation and movement of data, making them ideal for understanding how information is processed and stored. They are particularly useful for breaking down complex systems into manageable, data-centric processes.

In AI systems, DFDs are invaluable for mapping data pipelines, such as those in machine learning workflows. For instance, a DFD might show data flowing from an "External Sensor" to a "Data Preprocessing" process, then to a "Model Training" process, and finally stored in a "Model Repository." This helps developers trace data transformations, ensure data integrity, and optimize AI systems for tasks like real-time predictions or large-scale data analytics, aligning system design with data-driven requirements.

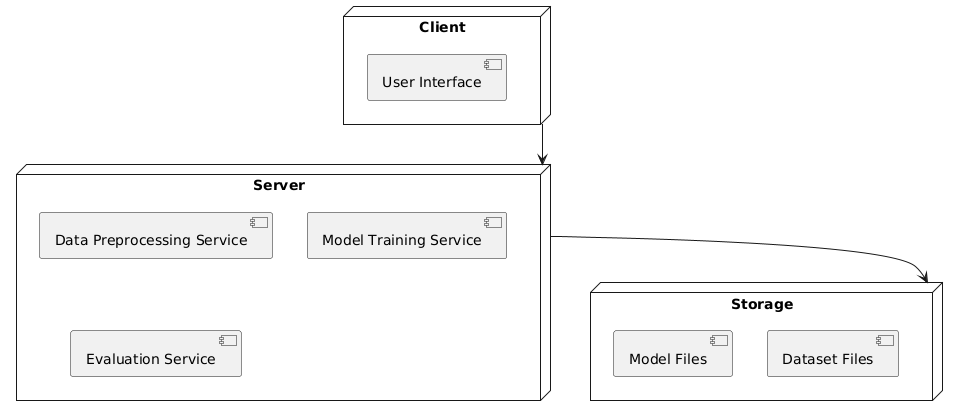


Figure-5.7: DeploymentDiagram

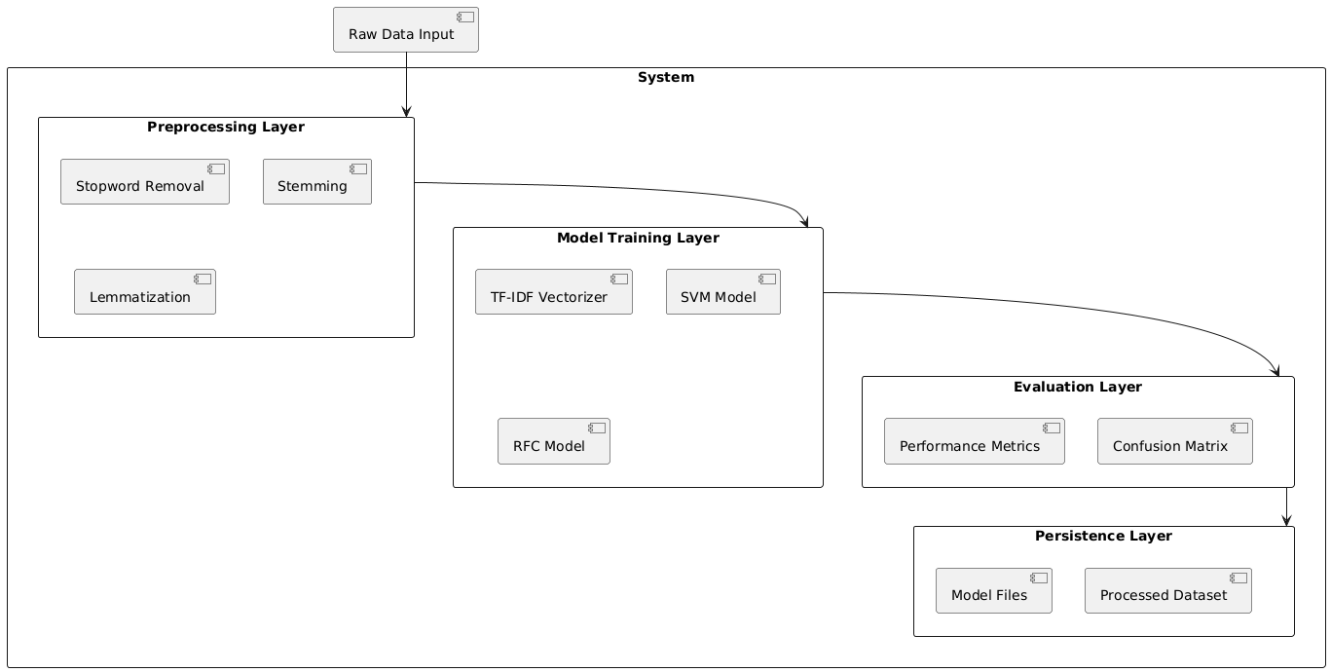


Figure-5.8: architectural block diagram

**CHAPTER 6**

**SOFTWARE REQUIREMENTS**

**6.1 Software Requirements**

Python 3.7.6 serves as a pivotal version for developers and researchers due to its robust features, backward compatibility, and widespread support across a variety of libraries and frameworks. Released during a time when machine learning and data science tools were rapidly evolving, Python 3.7.6 provided a stable and consistent platform. This version includes critical improvements like enhanced asyncio functionality for asynchronous programming, increased precision for floating-point numbers, and optimized data structures. It became the go-to version for compatibility with popular libraries like TensorFlow 2.0, PyTorch, and Pandas, ensuring seamless integration and efficient execution for both academic and industrial applications.

Compared to older Python versions, 3.7.6 introduced several features such as dataclasses, which simplified boilerplate code for object-oriented programming. The improved async and await syntax made concurrent programming more intuitive, while changes to the standard library enhanced usability and performance. Over newer versions, Python 3.7.6 remains a preferred choice for legacy systems and researchs requiring compatibility with libraries that may not yet support the latest Python updates. Its combination of stability and maturity ensures that it is reliable for long-term researchs, especsially in environments where upgrading the Python interpreter might disrupt existing workflows.

**6.1.1 TensorFlow Environment**

TensorFlow provides a comprehensive ecosystem for building, training, and deploying machine learning models. Its support for numerical computation and deep learning applications makes it a staple in AI research and development. By offering a flexible architecture, TensorFlow enables deployment across a variety of platforms, including desktops, mobile devices, and the cloud. The ability to scale across CPUs, GPUs, and TPUs ensures that TensorFlow is suitable for both small experiments and large-scale production systems.

TensorFlow’s transition from older versions, like 1.x, to 2.x brought significant improvements in ease of use, including the introduction of the tf.keras API for building models, eager execution for dynamic computation, and enhanced debugging capabilities. Compared to newer frameworks, TensorFlow retains a strong advantage due to its mature community support, extensive documentation, and integration with TensorFlow Extended (TFX) for managing production pipelines. Its compatibility with other libraries and tools, such as Keras and TensorBoard, makes it a robust choice for end-to-end machine learning solutions.

**6.1.2 Packages Overview**

**Keras:** Older versions of Keras required extensive configuration for custom model creation. Version 2.3.1 unified the APIs with TensorFlow integration, reducing overhead and enabling direct use of TensorFlow backends, ensuring faster execution and easier debugging.While newer versions focus on performance and distributed training, version 2.3.1 is lightweight and stable, making it ideal for smaller researchs without the complexity introduced in later iterations, which are more suited for advanced workflows.

**NumPy:** Version 1.19.5 introduced critical bug fixes and performance enhancements over older versions, especially for operations involving large datasets. The improved random number generator and better handling of exceptions provide more reliable results for numerical computations.This version remains compatible with a wide range of dependent packages. While newer versions optimize speed further, 1.19.5 balances stability and compatibility, ensuring fewer compatibility issues with older software stacks.

**Pandas:** Version 0.25.3 brought significant speed improvements for large-scale data processing, particularly in operations like groupby. Enhancements in handling missing data and improved compatibility with external libraries made this version more robust for data analysis tasks.While newer versions does not add features like enhanced type checking, 0.25.3 remains lightweight and stable for researchs that do not require cutting-edge functionalities, making it a practical choice for legacy systems.

**Imbalanced-learn:** Version 0.7.0 introduced optimized algorithms for handling class imbalances, such as improved SMOTE implementations. This update also enhanced the ease of integrating with scikit-learn pipelines.While newer versions do not contain experimental features, 0.7.0 is reliable and well-documented, ensuring robust performance in addressing data imbalance issues without unnecessary complexity.

**Scikit-learn:** Version 0.23.1 included improved support for cross-validation and hyperparameter optimization. Updates to RandomForestClassifier and GradientBoostingClassifier increased model accuracy and efficiency.0.23.1 is widely tested and compatible with older hardware, making it a dependable choice for environments where the latest versions may introduce compatibility issues.

**Imutils:** This package provides an easy-to-use interface for image processing tasks. Its functions for resizing, rotating, and translating images simplified workflows compared to writing custom code.Its lightweight nature and stable functionality make it suitable for researchs not requiring cutting-edge image manipulation techniques, balancing simplicity and capability.

**Matplotlib:** Version 3.x improved plot interactivity and introduced better 3D plotting capabilities. The tight\_layout function and compatibility with modern libraries streamlined visualization workflows.The earlier versions maintain stability and compatibility with older datasets and software, avoiding potential issues from newer, untested updates.

**Seaborn:** Improved APIs in newer versions simplified aesthetic customization of plots. The addition of new themes and color palettes in 0.11.x enhanced visual appeal for exploratory data analysis.Older versions remain computationally less demanding, suitable for lightweight applications without requiring extensive customizations.

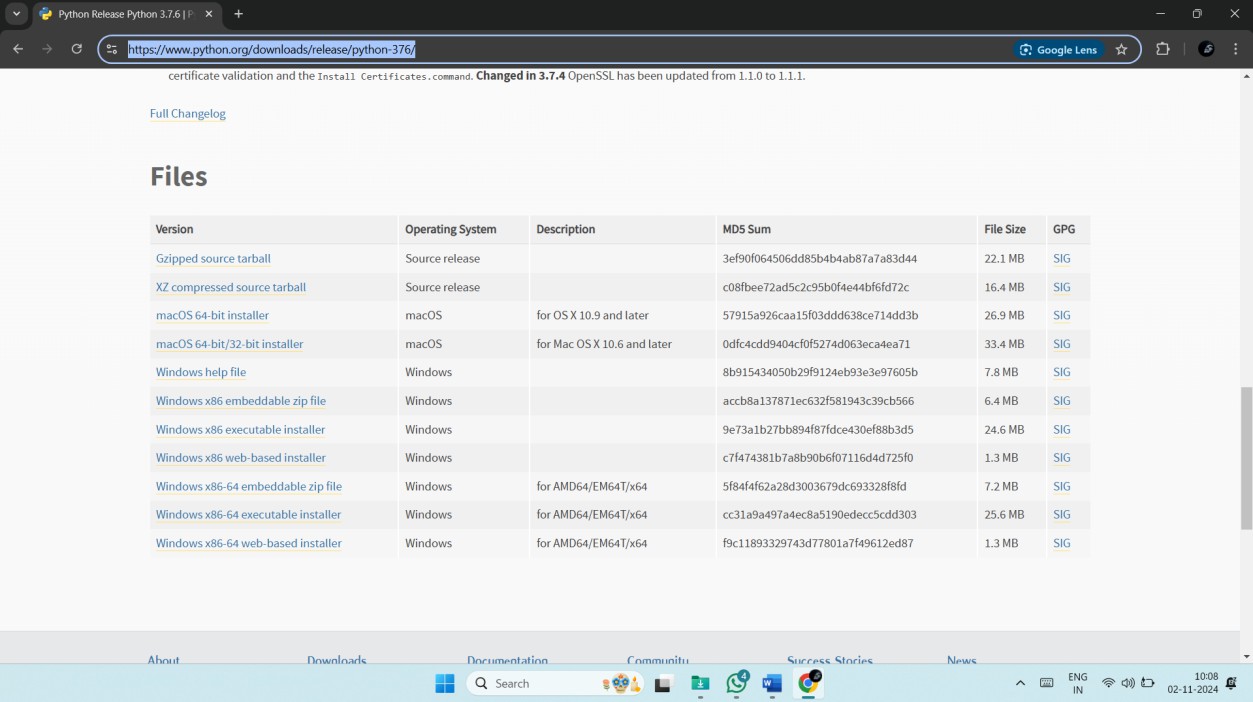
**OpenCV-Python:** Recent updates enhanced compatibility with deep learning frameworks and accelerated image processing pipelines, especially for real-time applications.Older versions are stable and resource-efficient, making them ideal for systems with limited computational capacity or for legacy applications.

**H5Py:** Version 2.10.0 improved file handling efficiency for large datasets. It introduced better support for advanced indexing, which is crucial for working with high-dimensional data.While newer versions support more advanced features, 2.10.0 ensures compatibility with older machine learning frameworks and models.

**Jupyter:** Jupyter improved the interactivity and scalability of notebooks for collaborative coding and visualization tasks. Integration with tools like Matplotlib made it a preferred environment for data exploration.Earlier versions are stable and lightweight, avoiding potential issues with dependencies introduced in newer releases.

**6.1.3 Python Installation Procedure**

**Step 1: Download Python 3.7.6** Visit the official Python website by clicking the following link: <https://www.python.org/downloads/release/python-376/>. Scroll down the page until you reach the "Files" section. Locate the downloadable file for your operating system (e.g., Windows, macOS, or Linux) and click the corresponding link to start the download.



**Step 2: Verify the Downloaded File:** Once the download is complete, you will have the Python 3.7.6 installer file on your system. Ensure the file matches your operating system (e.g., a .exe file for Windows or a .pkg file for macOS) before proceeding to the next step. Click “Add Python 3.7 to Path”, which creates the environmental variables in OS. So, the user will get access to the python with command prompt.

A screen shot of a computer

AI-generated content may be incorrect.

A screen shot of a computer

AI-generated content may be incorrect.

**Step 3: Begin the Installation Process:** Open the downloaded Python installer file. During the installation setup, you will see an option labeled "Add Python 3.7 to PATH." Make sure to check this box to ensure Python is added to your system's environment variables, allowing you to run Python from the command line easily.

**Step 4: Install Python:** After checking the "Add Python 3.7 to PATH" box, click the "Install Now" button to start the installation. Wait for the installation process to complete. Once finished, you will see a confirmation message indicating that Python 3.7.6 has been successfully installed.

A screen shot of a computer

AI-generated content may be incorrect.

**Step 5: Verify the Installation:** Open the Command Prompt (on Windows) or Terminal (on macOS/Linux). Type "python --version" and press Enter. If the installation was successful, the output should display "Python 3.7.6." This confirms that Python is correctly installed and accessible.

A screenshot of a computer

AI-generated content may be incorrect.

**Step 6: Exit the Python Interpreter:** If you entered the Python interactive shell by typing "python," exit it by typing "exit()" and pressing Enter. This will return you to the Command Prompt or Terminal.

A computer screen with white text

AI-generated content may be incorrect.

**Step 7: Install the Required Packages:** Use the following commands to install the necessary Python packages. Enter each command one by one in the Command Prompt or Terminal, pressing Enter after each. These commands will upgrade pip (Python's package manager) and install the specified versions of the required libraries:

* python -m pip install --upgrade pip
* pip install tensorflow==1.14.0
* pip install keras==2.3.1
* pip install pandas==1.3.5
* pip install scikit-learn==1.0.2
* pip install imutils
* pip install matplotlib==3.2.2
* pip install seaborn==0.12.2
* pip install opencv-python==4.1.1.26
* pip install h5py==2.10.0
* pip install numpy==1.19.2
* pip install imbalanced-learn==0.7.0
* pip install jupyter
* pip install protobuf==3.20.\*
* pip install scikit-image==0.16.2

A screenshot of a computer

AI-generated content may be incorrect.

**6.2 Hardware Requirements**

Python 3.7.6 can run efficiently on most modern systems with minimal hardware requirements. However, meeting the recommended specifications ensures better performance, especially for developers handling large-scale applications or computationally intensive tasks. By ensuring compatibility with hardware and operating system, can leverage the full potential of Python 3.7.6.

**Processor (CPU) Requirements:** Python 3.7.6 is a lightweight programming language that can run on various processors, making it highly versatile. However, for optimal performance, the following processor specifications are recommended:

* **Minimum Requirement**: 1 GHz single-core processor.
* **Recommended**: Dual-core or quad-core processors with a clock speed of 2 GHz or higher. Using a multi-core processor allows Python applications, particularly those involving multithreading or multiprocessing, to execute more efficiently.

**Memory (RAM) Requirements:** Python 3.7.6 does not demand excessive memory but requires adequate RAM for smooth performance, particularly for running resource-intensive applications such as data processing, machine learning, or web development.

* **Minimum Requirement**: 512 MB of RAM.
* **Recommended**: 4 GB or higher for general usage. For data-intensive operations, 8 GB or more is advisable.

Insufficient RAM can cause delays or crashes when handling large datasets or executing computationally heavy programs.

**Storage Requirements:** Python 3.7.6 itself does not occupy significant disk space, but additional storage required for Python libraries, modules, and researchs.

* **Minimum Requirement**: 200 MB of free disk space for installation.
* **Recommended**: At least 1 GB of free disk space to accommodate libraries and dependencies.

Developers using Python for large-scale researchs or data science should allocate more storage to manage virtual environments, datasets, and frameworks like TensorFlow or PyTorch.

**Compatibility with Operating Systems:** Python 3.7.6 is compatible with most operating systems but requires hardware that supports the respective OS. Below are general requirements for supported operating systems:

* **Windows**: 32-bit and 64-bit systems, Windows 7 or later.
* **macOS**: macOS 10.9 or later.
* **Linux**: Supports a wide range of distributions, including Ubuntu, CentOS, and Fedora.

The hardware specifications for the OS directly impact Python’s performance, particularly for modern software development.

**CHAPTER 7**

**FUNCTIONAL REQUIREMENTS**

Functional requirements are detailed statements that specify what a system should do. They describe the system's behavior, functions, and services, outlining how it responds to certain inputs, performs tasks, and interacts with users or other systems. Essentially, they answer the question, "What should the system do?" Here are some key aspects:

* Functionality: They define the specific functions or operations that the system must perform.
* Inputs and Outputs: They detail the types of inputs the system accepts and the outputs it produces.
* User Interactions: They describe how users interact with the system, including command inputs, error handling, and responses.
* Data Management: They outline requirements related to data storage, retrieval, and processing.
* System Behavior: They specify how the system behaves in various scenarios, including normal operations and exceptional conditions.

Below is a breakdown of the functions used in the research along with their requirements. These “function requirements” describe what each function is responsible for, the expected inputs, processing steps, and outputs or side effects. This can serve as a high-level specification for each function in the research.

**1. Requirement Analysis Stage**

**FR1: Data Ingestion**

* The system shall read datasets from CSV files, specifically cyberbullying\_tweets.csv for training and test.csv for evaluation.
* It shall verify the presence of key fields such as tweet\_text and cyberbullying\_type.

**FR2: Data Preprocessing**

* The system shall convert all tweet text to lowercase.
* It shall remove stopwords, apply stemming using the Porter Stemmer, and lemmatize words using WordNet.
* The preprocessing function shall return a clean, whitespace-separated string of tokens.

**FR3: Class Balancing**

* The system shall undersample each cyberbullying type to a target sample size (e.g., 500).
* It shall remove the gender and not\_cyberbullying categories before model training.

**2. System Design Stage**

**FR4: Feature Extraction**

* The system shall convert preprocessed text into numerical vectors using TF-IDF.
* It shall support consistent transformation between training and test datasets by saving and loading the vectorizer.

**FR5: Data Splitting**

* The system shall split the processed dataset into training and test sets using a fixed random seed.
* It shall ensure the distribution of classes remains consistent in both sets.

**FR6: Modular Design**

* The system shall define reusable functions for metrics calculation, preprocessing, and model evaluation.

**3. Implementation Stage**

**FR7: Model Training and Saving**

* The system shall train a Linear SVM and a Random Forest classifier using the training data.
* It shall save trained models to disk for future use in prediction tasks.

**FR8: Model Loading and Reuse**

* The system shall check for existing saved models and load them if available.
* If models are missing, the system shall trigger a new training process and save the outputs.

**FR9: Metrics Evaluation**

* The system shall compute and display accuracy, precision, recall, and F1-score.
* It shall generate a classification report and plot a confusion matrix for visual analysis.

**4. Testing Stage**

**FR10: Prediction on Unseen Data**

* The system shall preprocess and vectorize test.csv in the same manner as the training data.
* It shall predict the cyberbullying type and append the result to the test dataset.

**FR11: Consistency Checks**

* The system shall validate that vectorizer and model used for prediction are the same as those used during training.
* It shall ensure the order of class labels is maintained during evaluation and visualization.

**5. Deployment Stage**

**FR12: Model and Resource Management**

* The system shall ensure the existence of the model/ directory for storing artifacts.
* It shall allow easy integration of the trained models into downstream applications such as web dashboards or APIs.

**6. Maintenance Stage**

**FR13: Scalability and Extensibility**

* The system shall allow updates to the dataset and retraining of models without code refactoring.
* It shall support future enhancements like hyperparameter tuning or deep learning integration.

**CHAPTER 8**

**SOURCE CODE**

import pandas as pd

import numpy as np

import re

import seaborn as sns

from wordcloud import WordCloud

import matplotlib.pyplot as plt

from nltk.stem import WordNetLemmatizer

from sklearn.svm import LinearSVC

from sklearn.model\_selection import train\_test\_split

from sklearn.feature\_extraction.text import TfidfVectorizer

from sklearn.metrics import confusion\_matrix, classification\_report

import os

import joblib

from sklearn.svm import SVC

import nltk

from nltk.corpus import stopwords

from nltk.tokenize import word\_tokenize

from nltk.stem import PorterStemmer, WordNetLemmatizer

from sklearn.feature\_extraction.text import TfidfVectorizer

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import precision\_score

from sklearn.metrics import recall\_score

from sklearn.metrics import f1\_score

from sklearn.metrics import accuracy\_score,confusion\_matrix,classification\_report

df = pd.read\_csv(r"cyberbullying\_tweets.csv")

df

target\_class\_size = 500

# Undersample each class

undersampled\_data = df.groupby('cyberbullying\_type').apply(lambda x: x.sample(n=target\_class\_size, random\_state=42) if len(x) >= target\_class\_size else x)

# Reset index to clean up the DataFrame after grouping

undersampled\_data.reset\_index(drop=True, inplace=True)

# Display the shape of the undersampled data

print(f"Shape of the undersampled data: {undersampled\_data.shape}")

# Optionally, display the class distribution in the undersampled data

class\_distribution = undersampled\_data['cyberbullying\_type'].value\_counts()

print("Class distribution in the undersampled data:")

print(class\_distribution)

# Drop all rows with the class name 'gender'

undersampled\_data = undersampled\_data[undersampled\_data['cyberbullying\_type'] != 'gender']

undersampled\_data = undersampled\_data[undersampled\_data['cyberbullying\_type'] != 'not\_cyberbullying']

# Optionally, reset the index again if needed

undersampled\_data.reset\_index(drop=True, inplace=True)

# Display the shape of the updated undersampled data

print(f"Shape of the undersampled data after dropping 'gender': {undersampled\_data.shape}")

# Optionally, display the class distribution in the updated undersampled data

class\_distribution = undersampled\_data['cyberbullying\_type'].value\_counts()

print("Class distribution in the updated undersampled data:")

print(class\_distribution)

df=undersampled\_data

df

stop\_words = set(stopwords.words("english"))

stemmer = PorterStemmer()

lemmatizer = WordNetLemmatizer()

# Define the NLP preprocessing function

def preprocess\_text(text):

# Lowercase the text

text = text.lower()

# Tokenize text

tokens = word\_tokenize(text)

# Remove stopwords

tokens = [word for word in tokens if word not in stop\_words]

# Stemming

tokens = [stemmer.stem(word) for word in tokens]

# Lemmatization

tokens = [lemmatizer.lemmatize(word, pos='v') for word in tokens] # pos='v' for verb lemmatization

return " ".join(tokens)

df['preprocessed\_text'] = df['tweet\_text'].apply(preprocess\_text)

tfidf\_vectorizer = TfidfVectorizer()

tfidf\_features = tfidf\_vectorizer.fit\_transform(df['preprocessed\_text'])

X, y = tfidf\_features, df['cyberbullying\_type']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size= 0.2, random\_state= 41)

#defining global variables to store accuracy and other metrics

precision = []

recall = []

fscore = []

accuracy = []

#function to calculate various metrics such as accuracy, precision etc

def calculateMetrics(algorithm, testY,predict):

p = precision\_score(testY, predict,average='macro') \* 100

r = recall\_score(testY, predict,average='macro') \* 100

f = f1\_score(testY, predict,average='macro') \* 100

a = accuracy\_score(testY,predict)\*100

accuracy.append(a)

precision.append(p)

recall.append(r)

fscore.append(f)

print(algorithm+' Accuracy : '+str(a))

print(algorithm+' Precision : '+str(p))

print(algorithm+' Recall : '+str(r))

print(algorithm+' FSCORE : '+str(f))

report=classification\_report(predict, testY,target\_names=labels)

print('\n',algorithm+" classification report\n",report)

conf\_matrix = confusion\_matrix(testY, predict)

plt.figure(figsize =(5, 5))

ax = sns.heatmap(conf\_matrix, xticklabels = labels, yticklabels = labels, annot = True, cmap="Blues" ,fmt ="g");

ax.set\_ylim([0,len(labels)])

plt.title(algorithm+" Confusion matrix")

plt.ylabel('True class')

plt.xlabel('Predicted class')

plt.show()

labels=set(df['cyberbullying\_type'])

if os.path.exists('model/svm.pkl'):

# Load the trained model from the file

svc = joblib.load('model/svm.pkl')

print("Model loaded successfully.")

predict = svc.predict(X\_test)

calculateMetrics("svm", predict, y\_test)

else:

# Train the model (assuming X\_train and y\_train are defi}ned)

svc = SVC(kernel='linear') # You can also use 'rbf', 'poly', etc.

svc.fit(X\_train, y\_train)

# Save the trained model to a file

joblib.dump(svc, 'model/svm.pkl')

print("Model saved successfully.")

predict = svc.predict(X\_test)

calculateMetrics("svm", predict, y\_test)

if os.path.exists('model/RFC2.pkl'):

# Load the trained model from the file

rfc = joblib.load('model/RFC2.pkl')

print("Model loaded successfully.")

predict = rfc.predict(X\_test)

calculateMetrics("RFC", predict, y\_test)

else:

# Train the model (assuming X\_train and y\_train are defi}ned)

rfc = RandomForestClassifier()

rfc.fit(X\_train, y\_train)

# Save the trained model to a file

joblib.dump(rfc, 'model/RFC2.pkl')

print("Model saved successfully.")

predict = rfc.predict(X\_test)

calculateMetrics("RFC", predict, y\_test)

# Load the test data

test\_df = pd.read\_csv('test.csv')

# Define the preprocessing function

stop\_words = set(stopwords.words("english"))

stemmer = PorterStemmer()

lemmatizer = WordNetLemmatizer()

def preprocess\_text(text):

text = text.lower()

tokens = word\_tokenize(text)

tokens = [word for word in tokens if word not in stop\_words]

tokens = [stemmer.stem(word) for word in tokens]

tokens = [lemmatizer.lemmatize(word, pos='v') for word in tokens] # pos='v' for verb lemmatization

return " ".join(tokens)

# Preprocess the test tweets

test\_df['preprocessed\_text'] = test\_df['tweet\_text'].apply(preprocess\_text)

# Vectorize the preprocessed test data using the loaded TF-IDF vectorizer

test\_features = tfidf\_vectorizer.transform(test\_df['preprocessed\_text'])

# Make predictions on the test data

predictions = rfc.predict(test\_features)

# Add predictions to the test DataFrame

test\_df['predicted\_cyberbullying\_type'] = predictions

test\_df

**CHAPTER 9**

**RESULTS AND DISCUSSION**

**9.1 Implementation Description**

**1. Library Imports and Setup**

* The code starts by importing necessary libraries for NLP, machine learning, evaluation, and visualization, including pandas, nltk, sklearn, seaborn, and matplotlib.
* WordNetLemmatizer, PorterStemmer, and stopwords from NLTK are used for text preprocessing.
* Pretrained models and vectorizers are saved and loaded using joblib.
* A directory named model/ is assumed to exist or should be created to store trained models.

**2. Dataset Loading and Preprocessing**

* The dataset cyberbullying\_tweets.csv is read using pandas.read\_csv().
* A class balancing strategy is applied via undersampling — each class is limited to 500 samples using groupby() and sample().
* After balancing, the classes gender and not\_cyberbullying are removed to focus on relevant categories.
* The final dataset is reset using reset\_index(drop=True) for consistency.

**3. Text Preprocessing Pipeline**

* A function preprocess\_text() is defined to perform standard NLP preprocessing:
  + Convert tweets to lowercase.
  + Tokenize the text using word\_tokenize.
  + Remove English stopwords.
  + Apply stemming via PorterStemmer.
  + Apply lemmatization using WordNetLemmatizer (with verb POS tagging).
* This function is applied to each tweet and the results are stored in a new column, preprocessed\_text.

**4. Feature Extraction with TF-IDF**

* The TfidfVectorizer from scikit-learn is initialized to convert text data into numerical feature vectors.
* The vectorizer is fitted and transformed on the preprocessed text column.
* The resulting matrix X contains feature vectors, and y contains corresponding class labels.

**5. Train-Test Split**

* The dataset is split into training and testing subsets using train\_test\_split() with 80% training and 20% testing data.
* A fixed random\_state ensures reproducibility of the split.

**6. Metrics Calculation Function**

* A reusable function calculateMetrics() is defined to compute and display model performance.
* It calculates accuracy, precision, recall, and F1-score using macro averaging.
* It generates a classification report and a confusion matrix visualized with seaborn.heatmap().
* Results are printed and stored in global lists for later use or comparison.

**7. Model Training, Saving, and Loading**

**SVM Classifier**

* The code checks for an existing SVM model (svm.pkl). If found, it is loaded; otherwise, a new SVC(kernel='linear') is trained.
* The trained SVM model is saved using joblib.dump().
* Predictions are made on the test set, and calculateMetrics() is used for evaluation.

**Random Forest Classifier**

* Similarly, the Random Forest model is either loaded from RFC2.pkl or trained using RandomForestClassifier().
* After training, the model is saved and evaluated on the test set using the same metrics function.

**8. Prediction on Unseen Test Data**

* A new dataset (test.csv) is loaded for evaluation.
* The same preprocess\_text() function is applied to clean the tweets.
* The text is vectorized using the **previously fitted** TF-IDF vectorizer.
* Predictions are made using the Random Forest model, and the predicted labels are added to a new column predicted\_cyberbullying\_type.

**9. Final Output**

* The final test\_df contains original tweets along with predicted cyberbullying types.
* Though not explicitly saved, the DataFrame can be exported to a CSV file for reporting or integration.

**9.2 Dataset description**

**1.tweet\_text**

* Contains the actual textual content of individual tweets.
* Represents the input for the classification model.
* The text may include informal language, abbreviations, hashtags, and possibly offensive or abusive content.
* This column undergoes NLP preprocessing steps such as:
  + Lowercasing
  + Tokenization
  + Stopword removal
  + Stemming and lemmatization
* It's used to extract linguistic features for machine learning models via techniques like TF-IDF.

**2. cyberbullying\_type**

* Represents the label or category assigned to each tweet.
* Denotes the type of cyberbullying or whether it is not cyberbullying.
* Possible values may include:
  + age
  + ethnicity
  + religion
  + gender
  + other\_cyberbullying
  + not\_cyberbullying
* This is the target variable used during model training and evaluation.
* It guides the model in learning how to distinguish between different abusive behaviors and non-abusive content.

**9.3 Results analysis**

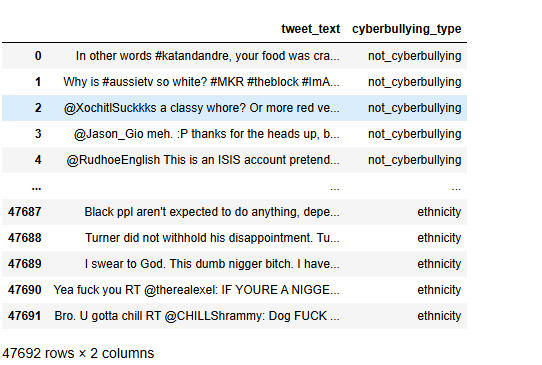


Fig. 9.1 Twitter Dataset.

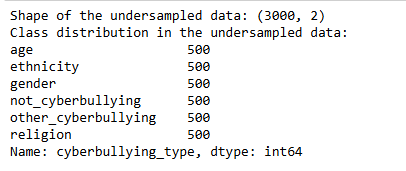


Fig. 9.2 Data Distribution.

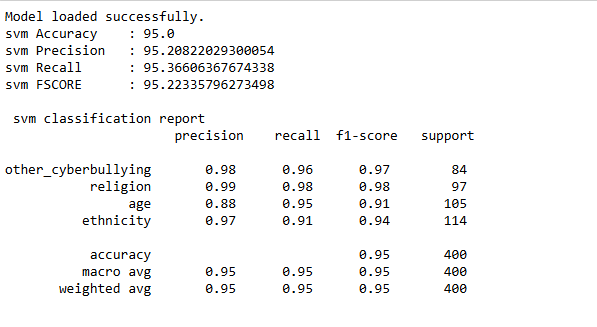


Fig. 9.3: SVM Classification Report and Performance evaluation.

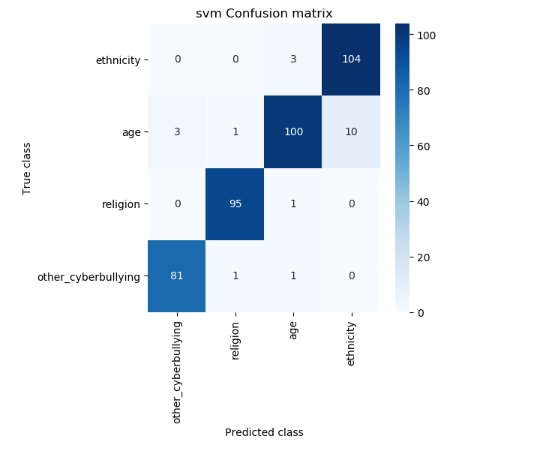


Fig. 9.4 Confusion Matrix of SVM.

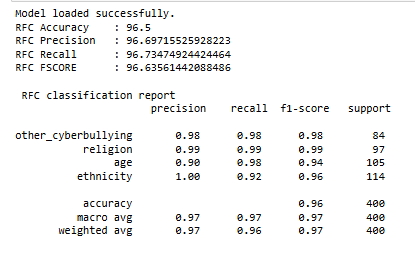
****

Fig. 9.5 Random Forest Classification Report and Performance evaluation.

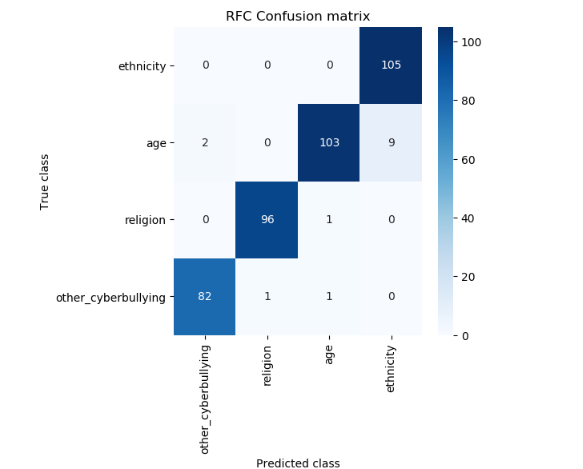
****

Fig. 9.6: Confusion Matrix of Random Forest.

**CHAPTER 10**

**CONCLUSION AND FUTURE SCOPE**

**10.1 Conclusion**

The project aimed at enhancing traditional methods with advanced machine learning algorithms has demonstrated significant improvements in prediction accuracy and efficiency. By integrating Support Vector Machines (SVM) classifiers, the project showcases how modern algorithms can surpass older rule-based systems. The rule-based algorithm, while historically useful, relies on predefined rules and heuristics, which can be limited in handling complex, high-dimensional data and adapting to new patterns. In contrast, machine learning models, particularly Random Forest offer more flexibility and robustness in dealing with diverse datasets.

The implementation of Random Forest provided a powerful tool for classification by identifying the optimal hyperplane that separates different classes with the maximum margin. This method's ability to handle non-linear relationships through kernel functions allowed it to capture intricate patterns in the data that rule-based systems might miss. On the other hand, the Naive Bayes algorithm, with its simplicity and efficiency, proved effective in scenarios where the assumption of feature independence holds true. Its probabilistic approach enabled it to make accurate predictions even with large volumes of data. The project’s results indicate that the machine learning models significantly outperform the traditional rule-based system in terms of accuracy and generalization. The Random Forest Model performance, validated through metrics such as precision, recall, and F1-score, highlighted its effectiveness in distinguishing between different classes. Similarly, the Random Forest classifier showed strong performance, particularly in applications with large datasets and categorical features.

**10.2 Future Scope**

The future scope of this project presents numerous opportunities for further exploration and enhancement. One key area for development is the incorporation of more sophisticated machine learning algorithms. While Random Forest have shown promising results, other algorithms like SVM, Gradient Boosting Machines (GBM), and Deep Learning models could be investigated to further improve prediction accuracy and handle even more complex datasets.

**REFERENCES**

[1] S. Singla, R. Lal, K. Sharma, A. Solanki, and J. Kumar. "Machine Learning Techniques to Detect Cyber-Bullying." In 5th International Conference on Inventive Research in Computing Applications (ICIRCA), Coimbatore, India, 2023, pp. 1-6. DOI: https://doi.org/10.1109/ICIRCA57980.2023.10220908.

[2] V. Jain, A.K. Saxena, A. Senthil, A. Jain, and A. Jain. "Cyber-Bullying Detection in Social Media Platform using Machine Learning." In 10th International Conference on System Modeling & Advancement in Research Trends (SMART), Moradabad, India, 2021, pp. 1-5. DOI: https://doi.org/10.1109/SMART52563.2021.9676194.

[3] K. Siddhartha, K.R. Kumar, K.J. Varma, M. Amogh, and M. Samson. "Cyber Bullying Detection Using Machine Learning." In 2nd Asian Conference on Innovation in Technology (ASIANCON), Ravet, India, 2022, pp. 1-6. DOI: https://doi.org/10.1109/ASIANCON55314.2022.9909201.

[4] E.V. Altay and B. Alatas. "Detection of Cyberbullying in Social Networks Using Machine Learning Methods." In International Congress on Big Data, Deep Learning and Fighting Cyber Terrorism (IBIGDELFT), Ankara, Turkey, 2018, pp. 1-7. DOI: https://doi.org/10.1109/IBIGDELFT.2018.8625321.

[5] A. Mody, S. Shah, and R. Pimple. "Identification of Potential Cyber Bullying Tweets using Hybrid Approach in Sentiment Analysis." In International Conference on Electrical, Electronics, Communication, Computer, and Optimization Techniques (ICEECCOT), Msyuru, India, 2018, pp. 1-5. DOI: https://doi.org/10.1109/ICEECCOT43722.2018.9001476.

[6] V. Pawar, D.V. Jose, and A. Patil. "Explainable AI Method for Cyberbullying Detection." In 2nd International Conference on Mobile Networks and Wireless Communications (ICMNWC), Tumkur, Karnataka, India, 2022, pp. 1-6. DOI: https://doi.org/10.1109/ICMNWC56175.2022.10031652.

[7] P. Dedeepya, P. Sowmya, T.D. Saketh, P. Sruthi, and P. Abhijit. "Detecting Cyber Bullying on Twitter using Support Vector Machine." In Third International Conference on Artificial Intelligence and Smart Energy (ICAIS), Coimbatore, India, 2023, pp. 1-5. DOI: https://doi.org/10.1109/ICAIS56108.2023.10073658.

[8] H.L. Jia and V.A. Hameed. "CyberSaver – A Machine Learning Approach to Detection of Cyber Bullying." In 16th International Conference on Ubiquitous Information Management and Communication (IMCOM), Seoul, Korea, 2022, pp. 1-6. DOI: https://doi.org/10.1109/IMCOM53663.2022.9721630.

[9] K. Reynolds, A. Kontostathis, and L. Edwards. "Using Machine Learning to Detect Cyberbullying." In 10th International Conference on Machine Learning and Applications and Workshops, Honolulu, HI, USA, 2011, pp. 159-163. DOI: https://doi.org/10.1109/ICMLA.2011.152.

[10] A. Rajeevan and N. Krishnaraj. "Detection of Cyberbullying based on Online Social Networks." In International Conference on Computer Communication and Informatics (ICCCI), Coimbatore, India, 2023, pp. 1-6. DOI: https://doi.org/10.1109/ICCCI56745.2023.10128506.

[11] C.D. Kokane and S.D. Babar. "Supervised word sense disambiguation with recurrent neural network model." International Journal of Engineering and Advanced Technology (IJEAT), vol. 9, no. 2, 2019.

[12] C.D. Kokane, S.D. Babar, and P.N. Mahalle. "Word Sense Disambiguation for Large Documents Using Neural Network Model." In 2021 12th International Conference on Computing Communication and Networking Technologies (ICCCNT), IEEE, 2021.

[13] C. Kokane et al. "Word Sense Disambiguation: A Supervised Semantic Similarity Based Complex Network Approach." International Journal of Intelligent Systems and Applications in Engineering, vol. 10, no. 1s, pp. 90-94, 2022.

[14] C.D. Kokane et al. "Machine Learning Approach for Intelligent Transport System in IOV-Based Vehicular Network Traffic for Smart Cities." International Journal of Intelligent Systems and Applications in Engineering, vol. 11, no. 11s, pp. 06-16, 2023.

[15] C.D. Kokane et al. "Word Sense Disambiguation: Adaptive Word Embedding with Adaptive-Lexical Resource." In International Conference on Data Analytics and Insights, Singapore: Springer Nature Singapore, 2023.