PEOPLE'S DEMOCRATIC REPUBLIC OF ALGERIA
Ministry of Higher Education And Scientific Research

University of Tlemcen – ABOU BAKR BELKAID - Algeria
FACULTY OF SCIENCE - DEPARTMENT of COMPUTER SCIENCE

Graduation Project

To obtain a Master's degree in Computer Science

Specialty: MID

On the subject:

System for detecting users with violent and threatening behavior on social networks.

Presented by :

M^r DROUICHE Ilyes

M^{lle} TERKI HASSAINE Rim

Presented : 25/06/2024

Before the jury composed of:

Chairman :	M^r BERRABAH Sid Ahmed
Examiner:	M^r BELABED Amine
Supervisor:	M^r MAHAMMED Nadir
Co-supervisor:	M^{me} SEKKAL Nawel

College year 2023/2024

Acknowledgements

We are grateful to Almighty Allah for giving us the strength and ability to fulfill this work. First, we would like to express our deep gratitude to our teacher and supervisor, Mr. Nadir Mahammed and Mdm Sekkal Nawel (that we wish well come back from 'Al hajj' safe and healthy), for his follow-up and enormous support throughout the preparation of this dissertation.

Secondly, we extend our sincere thanks to the members of the jury for agreeing to examine, evaluate, and judge our work. We will not let this opportunity pass without thanking our parents and all our family who, through their prayers and encouragement, helped us overcome all the obstacles.

We would like to thank everyone who participated directly or indirectly in the execution of this work.

Dedication

We dedicate this modest work to those closest to our hearts: To our parents who accompanied us during the toughest moments of this long educational journey, who shared with us all the emotional moments, and who always supported and encouraged us. We wish to express our deep gratitude, which will never be enough, and we hope to make them proud with this work.

Table des matières

Table des figures

Liste des tableaux

Chapter 1

Introduction

The advent of social networks has revolutionized how people communicate and interact, creating an unprecedented platform for sharing information, ideas, and personal experiences. However, this growth has also given rise to significant challenges, including the proliferation of fake profiles and users exhibiting violent and threatening behavior. These malicious activities not only undermine the trust and security of social media platforms but also facilitate the spread of misinformation, cyberbullying, and other harmful behaviors.

Traditional detection methods often fall short in addressing these issues due to the sophisticated and evolving tactics employed by perpetrators. Consequently, there is an urgent need for robust and adaptive detection systems that can effectively identify and mitigate these threats. This research aims to develop a comprehensive detection system that leverages machine learning models and metaheuristic optimization to enhance accuracy and efficiency.

This chapter provides an overview of the problem statement, reviews the current literature and motivations behind the study, and outlines the structure of the dissertation.

1.1 Problem statement

The proliferation of fake profiles and users with violent and threatening behavior on social networks undermines the trust and safety of online communities. These malicious activities can lead to the spread of misinformation, cyberbullying, and other harmful behaviors. Traditional detection methods often fall short due to the evolving tactics of perpetrators. Therefore, there is a pressing need for robust and adaptive detection systems that can effectively identify and mitigate these threats. This research addresses the critical challenge of developing a comprehensive detection system that combines machine learning models with metaheuristic optimization to improve accuracy and efficiency.

1.2 Current literature and motivation

The detection of fake profiles and users with violent and threatening behavior on social networks has garnered significant attention in recent years. Numerous studies have explored various approaches to address this issue, ranging from traditional rule-based systems to advanced machine learning techniques.

Current Literature:

- Machine Learning Approaches: Several studies have investigated the use of supervised and unsupervised machine learning algorithms for detecting fake profiles. For instance, Decision Trees, Random Forests, Support Vector Machines, Naive Bayes, and Gradient Boosting have been employed to classify user profiles based on various features. These models have shown promising results in terms of accuracy and efficiency.

- Metaheuristic Optimization: Metaheuristic algorithms, such as Genetic Algorithms, Simulated Annealing, and Particle Swarm Optimization, have been applied to optimize the performance of machine learning models. These optimization techniques help in fine-tuning model parameters, thereby improving detection accuracy and reducing computational costs.

- Hybrid Models: Recent studies have explored the integration of multiple approaches to enhance detection capabilities. For example, combining machine learning models with metaheuristic optimization has been shown to yield superior performance compared to standalone models.

Motivation:

Despite the advancements in detection methods, the dynamic and ever-evolving nature of malicious activities on social networks necessitates continuous improvement in detection systems. Traditional methods are often limited by their inability to adapt to new tactics employed by perpetrators. Therefore, there is a pressing need for adaptive and scalable solutions that can effectively address the challenges posed by fake profiles and violent behavior on social networks.

The motivation behind this research is to develop a robust and adaptive detection system that leverages the strengths of machine learning and metaheuristic optimization. By combining these approaches, we aim to enhance the accuracy and efficiency of detection systems, thereby contributing to the overall safety and security of social media platforms.

1.3 Dissertation structure

this thesis There are four chapters . The background information is provided in Chapter 2, which also places the dissertation in the third chapter of the state-of-the-art. The methods we use to carry out our comparative analysis is presented in Chapter 4. Finally, Chapter 5 gives the results and a discussion of them.

Chapter 2

Basic Concepts

2.1 Introduction

Social media platforms grapple with a pervasive issue — the rise of fake profiles that infiltrate online spaces. These deceptive personas, crafted with varying motives, from spreading misinformation to engaging in cyberbullying, cast a shadow over the authenticity of digital interactions. The anonymity afforded by the virtual realm allows these fraudulent identities to flourish, creating a blurred line between genuine users and those with malicious intent. Despite attempts to curb their prevalence through automated detection and user reporting, perpetrators persistently adapt, eluding identification. As a consequence, users navigate a digital landscape tainted by uncertainty, struggling to discern authentic connections from deceptive ones. The presence of fake profiles not only erodes trust but also introduces skepticism into the online information ecosystem. The challenge persists, requiring ongoing vigilance and awareness to preserve the genuine nature of social media interactions in the face of these deceptive digital avatars. This chapter includes a review of various methods. But first, now let us clarify the fundamental concepts surrounding this subject.

2.2 Fake profiles detection

2.2.1 Web 2.0

Is the evolution of the World Wide Web into a dynamic and interactive platform, emphasizing user participation and collaboration. Unlike the early days of the internet when

websites primarily delivered static content, Web 2.0 introduces a more engaging experience through user-generated content, social interaction, and collaborative tools. This paradigm shift is characterized by the prevalence of social media platforms, where users can share, comment, and interact with content in real-time. Additionally, the emergence of collaboration tools such as wikis and document sharing platforms facilitates collective knowledge creation and collaborative projects. The rich user experience, enabled by multimedia elements and responsive design, enhances the overall interactivity of websites and applications. Web 2.0 not only transforms how individuals consume information but also empowers them to actively contribute, shaping the internet into a more participatory and interconnected space.

2.2.2 Social Networks

social media platforms made it easier for people to communicate with friends, make new acquaintances, share their thoughts or join groups of interest. During the first years of its appearance, the use of these platforms has been mainly focused on messaging and status updating [14]. Nowadays, the technological development and the appearance of new social platforms increased the number of functionalities which allowed users to change their behavior inside social networks.

Online social networks offer users the opportunity to construct personal or professional profiles, incorporating details like their name, location, interests, and a profile image. Users can connect with others by sending friend requests, following profiles, and participating in groups or communities centered on shared interests. These platforms enable the sharing of various content forms, including photos, videos, and messages, fostering the emergence of user-generated content and online communities. The proliferation of online social networks has significantly reshaped societal dynamics, revolutionizing communication and interaction patterns. They have become pivotal hubs for information exchange, spawning new digital media formats and virtual communities. Moreover, social networks have revolutionized business communication strategies, facilitating more targeted and personalized marketing approaches. Nonetheless, alongside their benefits, concerns persist regarding privacy, security, and the proliferation of misinformation and online harassment. Many users remain unaware of the potential risks associated with divulging personally identifiable information, which can include age, gender, full name, and address, heightening concerns about fake profiles and security breaches.

2.2.3 Fake profile problem

Fake profile :Fake profiles are automated or semi-automated accounts that mimic human behavior on online social networks. They are often used to gather personal data from users on social media platforms. By sending friend requests to other users on the network, who often accept these requests, social bots can collect the private data of a user that should only be exposed to their friends. Additionally, fake profiles can be used to launch Sybil attacks, publish spam messages, or even manipulate the statistics of the online social network. . The main characteristics of fake profile are :

- It has less account age.
- Small number of followers.
- Limited Activity.
- Spamming Behavior.
- Location not specified.

Fake profiles are often used to collect personal data from other users on social networks and can be used for various malicious activities such as launching friend requests to gather private data and launching spam or Sybil attacks.

Fake profiles types

Several Fake Profiles types can be identified including:

- compromised profiles.
- clone profiles.
- Bot Accounts

2.3 Machine Learning

Machine Learning (ML) is a subset of the field of artificial intelligence that enables machines to automatically acquire knowledge through experience without being explicitly programmed. By following certain statistics and mathematical concepts, the machine

searches for patterns in the provided data, learns from them, and makes better decisions in the future [15]. One of the key features of machine learning is its ability to improve performance over time as it processes more data and refines its algorithms. Through a process called training, ML models are exposed to large amounts of labeled data, where the correct outcomes or labels are provided. By analyzing this data, the model adjusts its parameters and learns to recognize patterns and relationships, thereby improving its accuracy and effectiveness.

machine learning has applications across various domains, including image and speech recognition, natural language processing, medical diagnosis, financial forecasting, and autonomous driving. As advancements in algorithms and computing power continue, machine learning is expected to play an increasingly significant role in shaping the future of technology and society.

2.3.1 Machine learning models

Machine learning encompasses a variety of models and algorithms suited for different data types and tasks across numerous domains. This diversity reflects the complexity of problems addressed by machine learning, spanning areas like image recognition, natural language processing, finance, and healthcare.

- **Supervised learning** :Supervised learning involves training a model on a labeled dataset, where each input data point is associated with an output label. The objective is to learn a mapping function from input variables to output variables based on the labeled data provided during training. This allows the model to make predictions or decisions when presented with new, unseen data. Common algorithms in supervised learning include linear regression, logistic regression, support vector machines (SVM), decision trees, random forests, and various types of neural networks. Supervised learning finds applications in tasks such as classification, regression, and prediction across diverse fields such as finance, healthcare, marketing, and computer vision.

- **Unsupervised learning**: Unsupervised learning is a type of learning where the model is trained on unlabeled data, meaning that the input data does not have corresponding output labels. In unsupervised learning, the algorithm explores the structure of the data to find patterns, group similar data points, or reduce the dimensionality of the data. Unlike supervised learning, there is no specific target variable to

predict. Common algorithms used in unsupervised learning include clustering algorithms such as K-means clustering and hierarchical clustering, dimensionality reduction techniques like principal component analysis (PCA) and t-distributed stochastic neighbor embedding (t-SNE), and generative models such as autoencoders and Gaussian mixture models (GMM). Unsupervised learning finds applications in tasks such as customer segmentation, anomaly detection, feature extraction, and exploratory data analysis.

Supervised algorithms

Naive bayes classifier is a foundational machine learning algorithm renowned for its simplicity and effectiveness in classification tasks. Leveraging Bayes' theorem, it estimates the probability of a specific class given the presence of various features in the input data. Despite its "naive" assumption of feature independence, which often doesn't hold in real-world scenarios, Naive Bayes classifiers can deliver robust performance and are computationally efficient, making them invaluable in diverse applications.

Naive Bayes classifiers are particularly prominent in text categorization and spam filtering due to their ability to handle large datasets and high-dimensional feature spaces adeptly. Their simplicity makes them easy to implement and interpret, while their computational efficiency allows for real-time processing, making them suitable for applications requiring quick responses. Although Naive Bayes classifiers may not capture complex relationships between features, their reliability and scalability make them a preferred choice in various domains of machine learning and natural language processing, where their performance often rivals more complex algorithms.

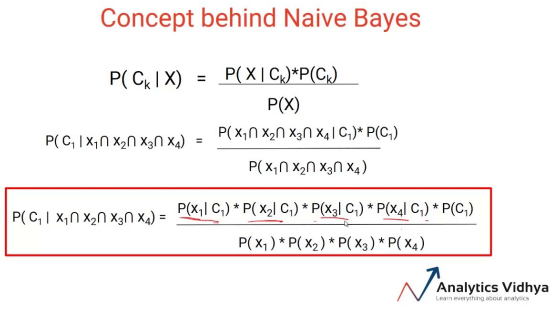

Figure 2.1: *Naive Bayes classifier*
Source:(`https://www.analyticsvidhya.com`)

Gaussian Naive bayes Gaussian Naive Bayes is a classification algorithm that is commonly used in machine learning. It is a variant of the Naive Bayes algorithm that is specifically designed for datasets with continuous features. In Gaussian Naive Bayes, it is assumed that the continuous values associated with each class are distributed according to a Gaussian (or normal) distribution.

The algorithm works by calculating the z-score distance between each data point and the mean of each class. This distance is then divided by the standard deviation of that class. The class with the smallest z-score distance is considered the most likely class for that data point.

Gaussian Naive Bayes has several advantages, including its simplicity and efficiency. It is particularly useful when working with continuous data and can be applied to many real-life situations

$$P(x_i|y) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

Figure 2.2: *Gaussian Naive Bayes formula*
Source:(`https://www.geeksforgeeks.org/`)

Decision tree Decision Trees represent a powerful and intuitive approach to supervised learning, renowned for their ability to make decisions by recursively partitioning data into subsets based on feature values. This algorithmic framework embodies the fundamental principles of decision-making, organizing complex datasets into a hierarchical structure resembling a tree. At each internal node of the tree, decisions are made based on feature conditions, leading to the creation of subsequent branches and nodes. Through a process of iterative splitting, Decision Trees identify optimal feature thresholds that best segregate data, ultimately culminating in leaf nodes containing homogeneous subsets. This systematic approach enables Decision Trees to offer transparency and interpretability, making them invaluable tools across a spectrum of domains.

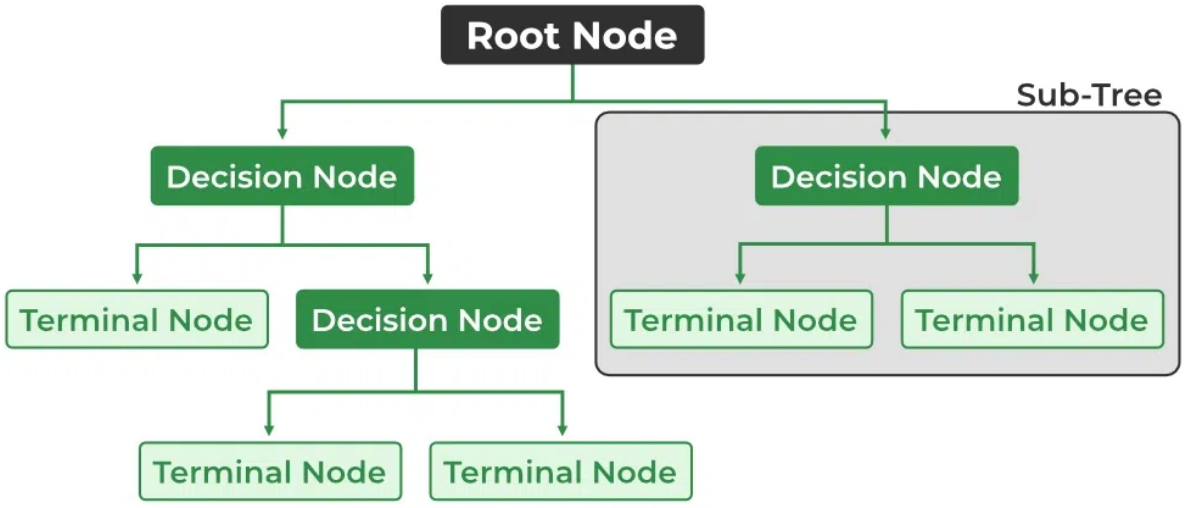

Figure 2.3: *Representation of how decision trees work*
Source:(`https://www.geeksforgeeks.org/`)

Support Vector Machine Certainly! Here's the paragraph with explanations of the Support Vector Machine (SVM) algorithm presented as points:

Support Vector Machine (SVM) is a powerful supervised learning algorithm used for classification and regression tasks. The algorithm operates with the following key objectives and characteristics:

- Optimal Hyperplane: SVM aims to find the optimal hyperplane that best separates data points into different classes while maximizing the margin between the classes.

- Maximizing Margin: The optimal hyperplane identified by SVM maximizes the distance (margin) between the nearest data points of the two classes, known as support vectors.

- Support Vectors: Support vectors are crucial data points that determine the position and orientation of the decision boundary. They are the data points closest to the decision boundary and play a key role in defining the optimal hyperplane.

- Handling Non-linearity: SVM can handle both linearly separable and non-linearly separable data by mapping the input features into a higher-dimensional space using kernel functions.

- Kernel Functions: SVM employs various kernel functions such as linear, polynomial, radial basis function (RBF), and sigmoid kernels to transform the data into higher-

dimensional spaces. These kernel functions help find hyperplanes that effectively separate the classes.

- Objective Function Optimization: During training, SVM optimizes a cost function to minimize classification errors and maximize the margin. This optimization process often involves techniques like gradient descent or quadratic programming.

- Generalization and Robustness: SVM is renowned for its ability to generalize well to unseen data and maintain robustness in high-dimensional spaces. It achieves this by maximizing the margin and effectively separating classes in the feature space.

- Applications: SVM finds applications in various classification and regression tasks across domains such as image classification, text categorization, bioinformatics, and more.

Through these principles and characteristics, SVM provides a robust and effective framework for solving classification and regression problems in diverse fields.

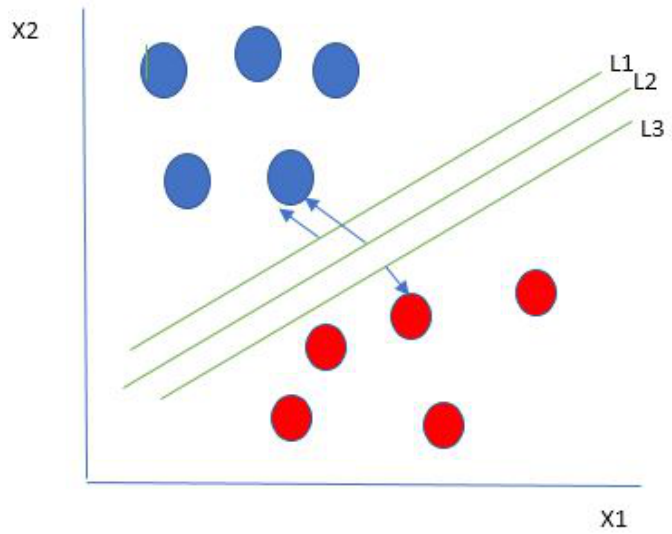

Figure 2.4: SVM Classifier
Source:(`https://www.geeksforgeeks.org/`)

Support Vector Classification Support Vector Classification (SVC) is a supervised machine learning algorithm used for classification tasks. It belongs to the family of Support Vector Machines (SVMs) and is particularly effective for binary classification problems.

Key points about SVC:

SVC is a classification algorithm that aims to find the best hyperplane in a high-dimensional feature space to separate different classes of data.

It works by finding the optimal decision boundary that maximizes the margin between the classes.

SVC can handle both linearly separable and non-linearly separable data by using different kernel functions, such as linear, polynomial, radial basis function (RBF), and sigmoid.

The regularization parameter C controls the trade-off between maximizing the margin and minimizing the classification errors.

SVC is a powerful algorithm for handling complex datasets and can generalize well to unseen data.

It has been widely used in various applications, including text categorization, image classification, and bioinformatics

K-nearest neighbors K-nearest neighbors (KNN) is an intuitive and versatile supervised learning algorithm utilized for both classification and regression tasks. The algorithm operates by identifying the "K" nearest data points to the new observation in question. This proximity is typically measured using distance metrics like Euclidean, Manhattan, or Minkowski distances.

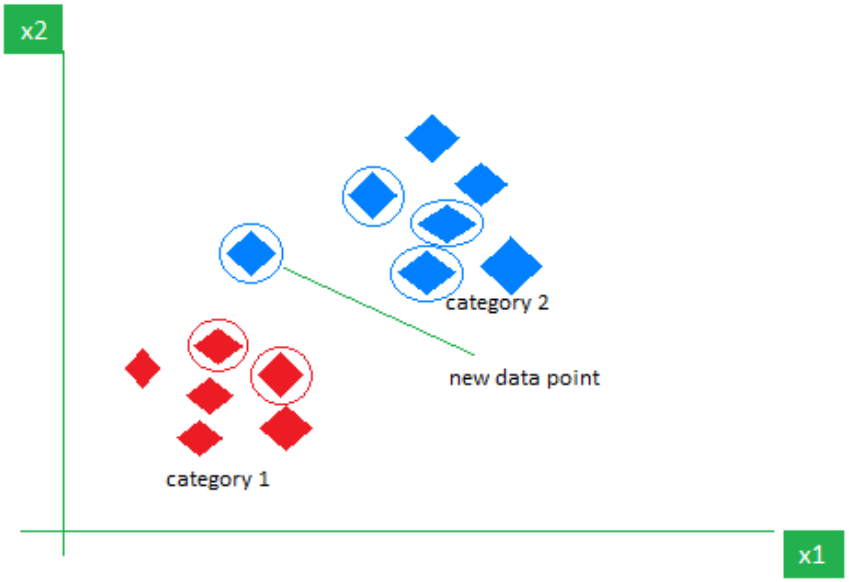

Figure 2.5: KNN working visualization
Source:(`https://www.geeksforgeeks.org/`)

In classification tasks, KNN assigns the majority class label among its nearest neighbors to the new observation, effectively making predictions based on local neighborhood information. Conversely, in regression tasks, KNN calculates the average or weighted average of the target values of its nearest neighbors to predict the continuous value for the new observation.

While KNN is straightforward to implement and understand, selecting an appropriate value for "K" is critical as it influences the model's flexibility and generalization ability. However, KNN's computational efficiency can suffer with larger datasets, particularly in high-dimensional feature spaces. Nonetheless, KNN remains a valuable tool in various domains due to its simplicity and effectiveness in capturing local patterns within the data.

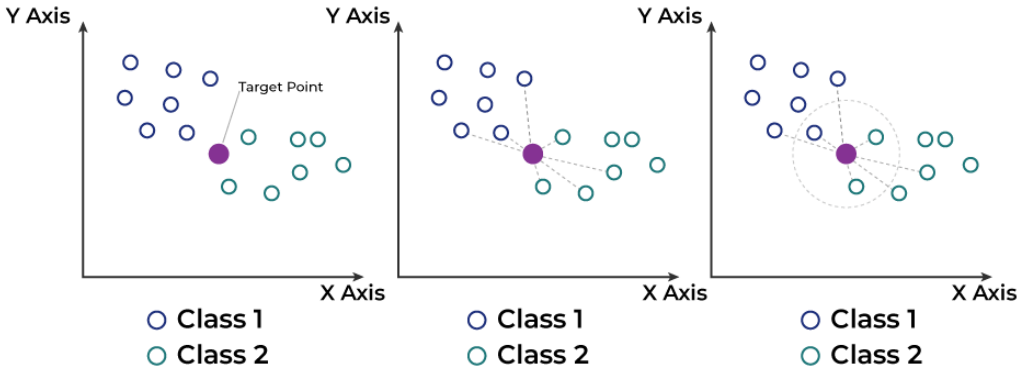

Figure 2.6: KNN algorithm way of working
Source:(`https://www.geeksforgeeks.org/`)

Logistic regression Logistic regression is a statistical model used for binary classification tasks, where the goal is to predict the probability of an event or outcome occurring based on a given set of independent variables. It models the log-odds of the event as a linear combination of the independent variables.

Key points about logistic regression: Logistic regression estimates the probability of an event occurring based on a given dataset of independent variables.

It is often used for classification and predictive analytics.

The outcome variable in logistic regression is bounded between 0 and 1, representing the probability of success.

A logit transformation is applied to the odds (probability of success divided by the probability of failure) to model the relationship between the independent variables and the dependent variable.

Logistic regression can handle both binary and categorical independent variables.

It is a widely used algorithm in various domains, including healthcare, finance, and social sciences.

Gradient Boosting is an ensemble learning technique that sequentially adds weak learners, often decision trees, to correct errors made by previous models. It minimizes the loss function by fitting weak learners to the residual errors of the ensemble's predictions. This iterative process creates a strong learner capable of capturing complex data relationships. Gradient Boosting handles heterogeneous data types and high-dimensional feature spaces effectively, automatically handling feature interactions and nonlinearities. Hyperparam-

eter tuning is crucial for optimizing model performance. Despite being computationally intensive, Gradient Boosting offers superior predictive performance and is widely used in finance, healthcare, and natural language processing.

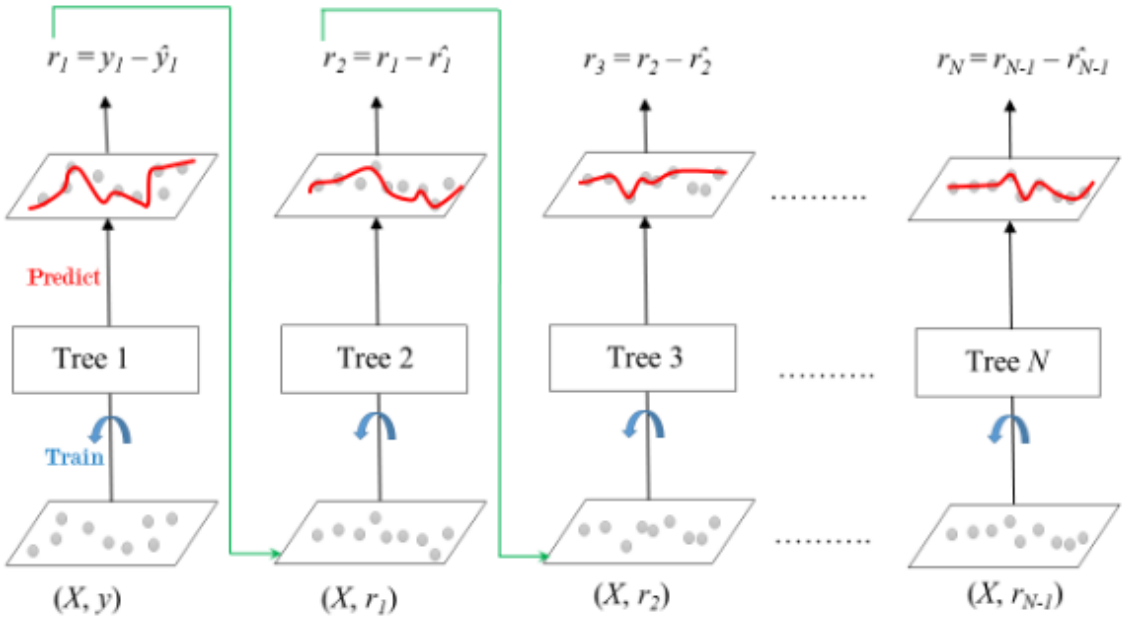

Figure 2.7: Gradient Boosted Trees for Regression
Source:(`https://www.geeksforgeeks.org/`)

Random Forest Machine: is a powerful ensemble learning algorithm widely used for both classification and regression tasks. The algorithm operates by constructing multiple decision trees during the training phase. Each decision tree is built using a random subset of the training data and a random subset of the input features.

In classification tasks, Random Forests use a majority voting mechanism among the ensemble of decision trees to determine the final class label for a new observation. In regression tasks, the algorithm calculates the average prediction across all the decision trees to predict the continuous value for the new observation.

One of the key strengths of Random Forests is its ability to handle high-dimensional datasets with complex interactions and noisy features. By aggregating predictions from multiple trees, Random Forests reduce the risk of overfitting and improve generalization performance compared to individual decision trees.

Hyperparameter tuning is crucial in Random Forests to optimize model performance. Parameters like the number of trees in the forest, the maximum depth of each tree, and

the number of features considered for each split impact the model's effectiveness and computational efficiency.

Despite its computational efficiency, Random Forests may suffer from interpretability issues compared to single decision trees. However, its robust performance and ability to handle diverse datasets make Random Forests a popular choice across various domains, including finance, healthcare, and bioinformatics.

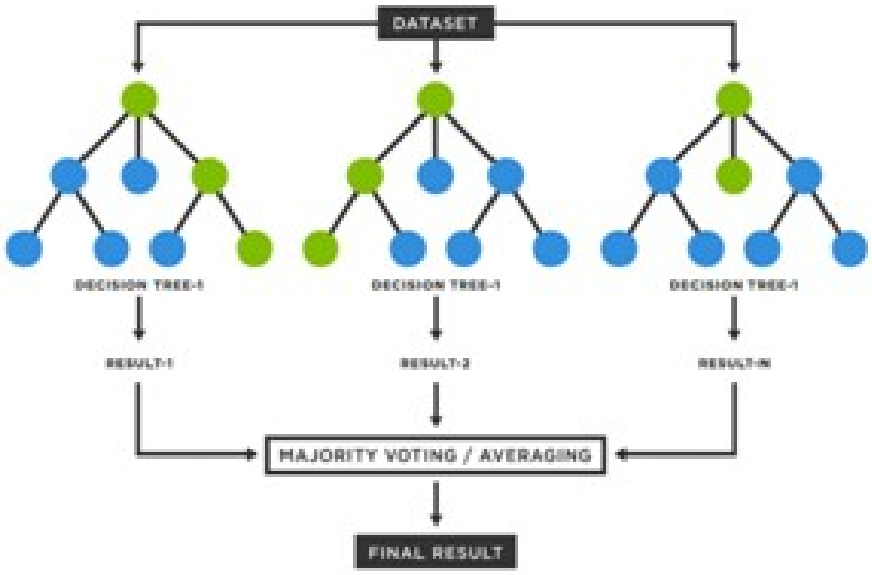

Figure 2.8: Random Forest Classifier [**?**]
Source:(`https://www.geeksforgeeks.org/`)

Unsupervised model

Kmeans algorithm K-means clustering is a widely used unsupervised learning algorithm that partitions a dataset into K distinct clusters based on the similarity of data points. The algorithm begins by randomly initializing K cluster centroids within the feature space. It then iteratively assigns each data point to the nearest centroid and updates the centroids by computing the mean of the data points assigned to each cluster. This process continues until convergence, where centroids stabilize and no further changes occur.

- A cluster refers to a collection of data points aggregated together .
- K is a target number, which refers to the number of centroids you need in the dataset.

- A centroid is the imaginary or real location representing the center of the cluster.

K-means aims to minimize the within-cluster sum of squares, creating compact and well-separated clusters. While K-means is efficient and easy to implement, it requires the user to specify the number of clusters beforehand, which can impact clustering quality. Additionally, K-means is sensitive to the initial placement of centroids and may converge to local optima. Despite these limitations, K-means remains a powerful tool for exploratory data analysis, customer segmentation, and image processing, providing valuable insights into data structure and aiding in pattern recognition tasks.

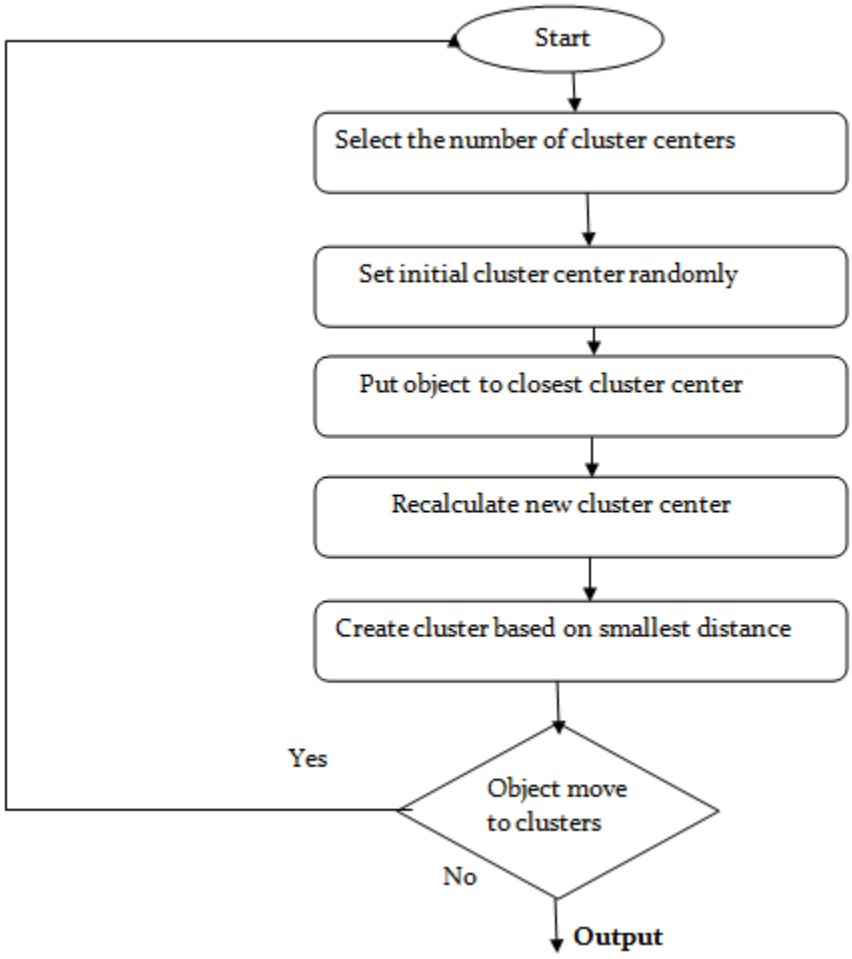

Figure 2.9: Flowchart of k-means clustering algorithm [1]

Gaussian Mixture Models Gaussian Mixture Models (GMMs) are probabilistic models used for clustering and density estimation. They assume that the data is generated from a

mixture of Gaussian distributions with unknown parameters. GMMs are a generalization of k-means clustering, incorporating information about the covariance structure of the data as well as the centers of the latent Gaussians. Key points about Gaussian Mixture Models: GMMs are used for clustering and density estimation. They assume data is generated from a mixture of Gaussian distributions.

GMMs are a soft clustering technique, assigning probabilities of data points belonging to each cluster. The Expectation-Maximization (EM) algorithm is commonly used to fit GMMs to data.

GMMs are flexible and can model clusters of different shapes and sizes.

They provide probabilities of data points belonging to each cluster, allowing for more nuanced analysis.

GMMs can estimate the underlying probability density function of the data, aiding in anomaly detection and outlier analysis.

GMMs can handle large datasets efficiently.

$$G(X|\mu,\sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

Figure 2.10: *Gaussian Naive Bayes formula*
Source:(`https://www.geeksforgeeks.org/`)

2.4 Metaheuristic

Metaheuristics is a high-level computational method or strategy designed to efficiently solve optimization and search problems that are too complex to be addressed with exact algorithms. Unlike traditional algorithms that guarantee optimal solutions, metaheuristics provide approximate solutions within a reasonable amount of time by exploring the solution space using heuristic techniques. Metaheuristic algorithms are iterative and typically involve randomness or stochastic elements to navigate through the search space and avoid getting stuck in local optima. Some common metaheuristic algorithms include genetic algorithms, simulated annealing, tabu search, particle swarm optimization, and ant colony optimization. These methods are widely used in various fields such as engineering, operations research, computer science, and economics to solve a diverse range of optimization problems including scheduling, routing, resource allocation, and machine learning. Metaheuristics offer flexible and versatile approaches to tackle complex optimization

problems where traditional methods may fail or become impractical due to computational constraints.

2.4.1 Metaheuristics and machine learning

Metaheuristics in machine learning employ iterative optimization algorithms designed to efficiently explore complex solution spaces. They excel in addressing high-dimensional and non-convex optimization problems, where traditional methods often struggle to find optimal solutions.

Metaheuristic algorithms leverage heuristics and stochastic elements to navigate through solution spaces, efficiently exploring diverse regions and avoiding local optima. By iteratively refining solutions, they aim to converge towards near-optimal solutions while avoiding getting stuck in suboptimal solutions.

The adaptability and flexibility of metaheuristic algorithms make them well-suited for various machine learning tasks, including feature selection, hyperparameter tuning, ensemble learning, neural network architecture search, and model interpretability. Their scalability allows them to efficiently address complex optimization problems across domains like computer vision, natural language processing, and bioinformatics. Ultimately, metaheuristics enhance the performance, generalization, and interpretability of machine learning models, making them invaluable tools for tackling real-world challenges efficiently.

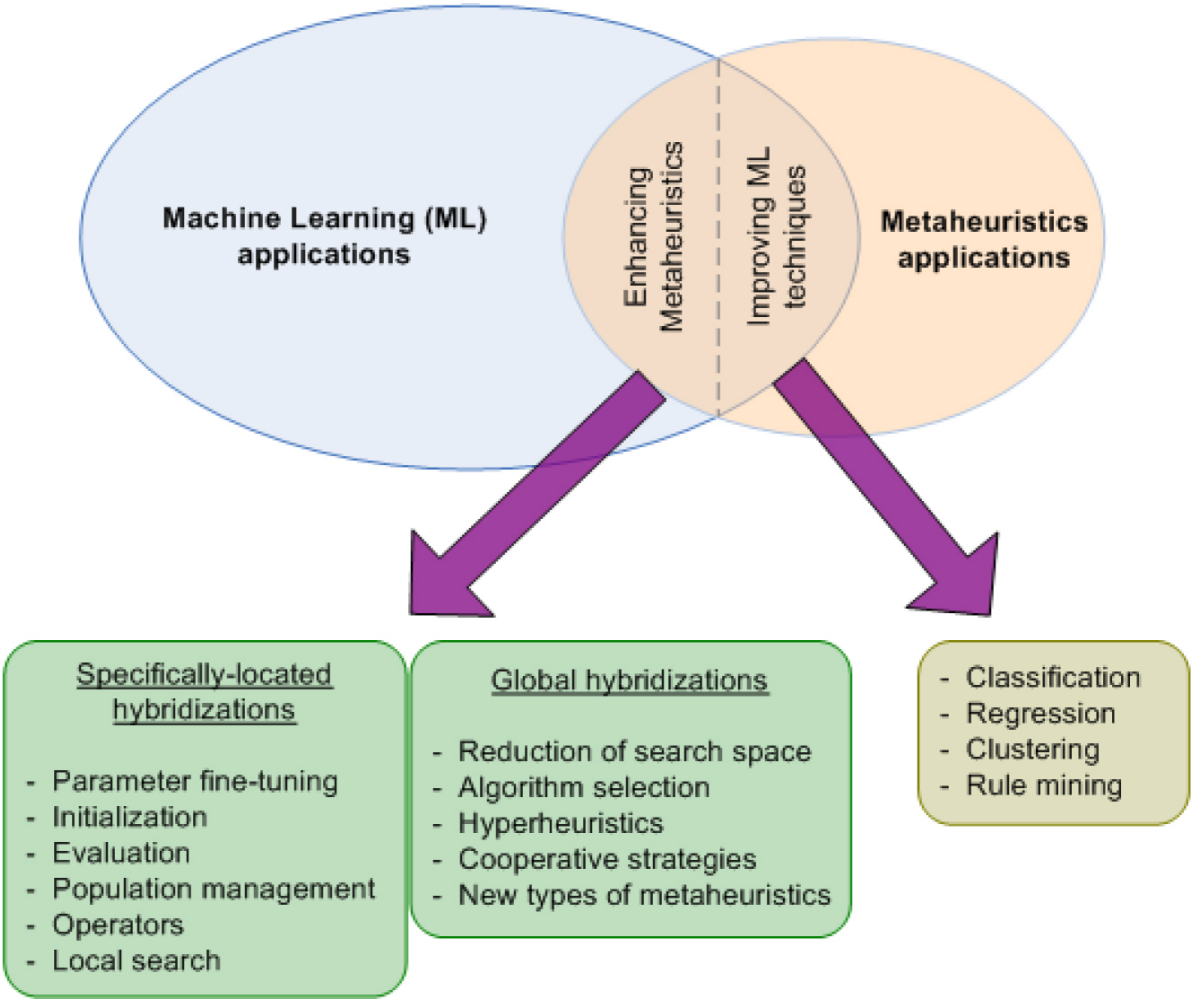

Figure 2.11: ML and Metaheuristics[2]

2.4.2 A Brief History

Metaheuristics have a fascinating history that traces back to the mid-20th century when researchers sought optimization methods that could tackle complex problems beyond the scope of traditional exact algorithms. In the 1950s, George Dantzig's simplex algorithm revolutionized linear programming, providing efficient solutions for optimization problems with linear constraints. However, as computational challenges expanded, particularly with the rise of non-linear and combinatorial optimization problems, researchers began exploring alternative approaches.

The term "metaheuristic" was coined in the late 1980s to encompass a diverse class

of high-level computational strategies guiding the search for solutions in complex optimization landscapes. It represented an abstraction layer above individual optimization algorithms, offering a generalized framework for solving a wide range of problems. Notable early contributions to metaheuristics include the introduction of simulated annealing by Kirkpatrick et al. in 1983, inspired by the annealing process in metallurgy.

In the late 1960s and early 1970s, the concept of genetic algorithms was introduced by John Holland and his colleagues, drawing inspiration from the principles of natural selection and genetics. This evolutionary approach to optimization proved to be highly effective in exploring solution spaces and has since become one of the most widely studied and applied metaheuristic techniques.

Throughout the 1980s and 1990s, numerous other metaheuristic algorithms emerged, including tabu search, ant colony optimization, and particle swarm optimization. Each of these approaches brought novel concepts and strategies for solving optimization problems, often inspired by natural phenomena or collective behaviors observed in biological systems.

The field of metaheuristics has since continued to evolve rapidly, with ongoing research focused on developing new algorithms, hybrid approaches, and optimization frameworks. With the advent of computational intelligence and advancements in computing technology, metaheuristics have found widespread applications in various domains, including engineering, logistics, finance, and machine learning. Today, metaheuristics remain at the forefront of optimization research, offering powerful tools for tackling complex real-world problems efficiently.

2.4.3 mode of operation of metaheuristics

Metaheuristics encompass a diverse set of algorithms, including genetic algorithms, simulated annealing, tabu search, ant colony optimization, particle swarm optimization, and differential evolution, among others. Each metaheuristic employs distinct strategies for exploration and exploitation within the solution space. For instance, genetic algorithms simulate the process of natural selection by evolving a population of candidate solutions through mutation, crossover, and selection operators. Simulated annealing mimics the annealing process in metallurgy, gradually reducing the probability of accepting inferior solutions as the search progresses. Tabu search maintains a memory structure to avoid revisiting previously explored solutions, while ant colony optimization models the foraging behavior of ants to discover optimal paths through pheromone communication. Parti-

cle swarm optimization simulates the social behavior of bird flocking or fish schooling, with particles adjusting their positions based on their own experience and that of their neighbors.

The mode of operation of each metaheuristic involves a balance between exploration, to discover diverse regions of the solution space, and exploitation, to refine promising solutions. This balance is often achieved through parameter tuning and adaptive mechanisms that dynamically adjust algorithmic behavior during the search process. Additionally, metaheuristics may incorporate strategies for parallelism and hybridization, leveraging the power of multiple algorithms or computational resources to enhance search efficiency. While metaheuristics do not guarantee finding the global optimum, they offer robust and scalable approaches for addressing complex optimization problems in various domains, ranging from engineering and logistics to finance and bioinformatics.

2.4.4 Classification

Local search and global search

Classification of search algorithms often distinguishes between local search and global search strategies. Local search algorithms, also known as neighborhood search, focus on exploring the immediate vicinity of a given solution in the search space. These algorithms iteratively move from one solution to a neighboring solution that offers better performance according to an evaluation function, without considering the overall structure of the search space. Examples of local search algorithms include hill climbing, simulated annealing, and tabu search.

In contrast, global search algorithms aim to explore a broader region of the search space in search of the optimal solution. These algorithms employ strategies such as systematic exploration, population-based methods, or probabilistic sampling to search across diverse regions of the solution space. Genetic algorithms, particle swarm optimization, and ant colony optimization are examples of global search algorithms. While local search algorithms are often efficient for finding solutions in relatively constrained search spaces, global search algorithms are better suited for exploring complex and multimodal search spaces to find globally optimal solutions. The choice between local and global search strategies depends on the nature of the optimization problem, its constraints, and the desired characteristics of the solution.

Single-solution vs. population-based

Classification of optimization algorithms often distinguishes between single-solution and population-based approaches, which are closely related to the field of metaheuristics. Single-solution algorithms, such as hill climbing and simulated annealing, iteratively refine a single candidate solution, exploring the solution space locally to improve upon the current best solution iteratively. These methods are effective for simpler problems but may struggle in complex, multimodal landscapes.

Population-based algorithms, such as genetic algorithms and particle swarm optimization, maintain a population of candidate solutions and iteratively evolve them through various operations like selection, crossover, and mutation. Metaheuristics, as a broader category, encompass both single-solution and population-based approaches, offering versatile strategies for optimization across various domains. They leverage iterative improvement and exploration techniques to efficiently navigate complex search spaces, seeking near-optimal solutions without guaranteeing global optimality.

The choice between single-solution and population-based approaches in metaheuristics depends on factors like problem complexity, search space characteristics, and computational resources available, with each approach offering unique strengths and trade-offs in optimization tasks. Metaheuristics provide a flexible framework for tackling optimization problems, allowing practitioners to adapt algorithms based on problem-specific requirements and constraints, thereby addressing a wide range of real-world optimization challenges effectively.

Hybridization and memetic algorithms

Hybridization and memetic algorithms play significant roles in enhancing the performance and versatility of metaheuristics. Hybridization involves integrating different optimization techniques or problem-solving strategies to leverage their complementary strengths. Memetic algorithms, a form of hybrid metaheuristics, combine evolutionary algorithms with local search methods to exploit global exploration capabilities and local refinement efficiently. By incorporating local search operators into evolutionary processes, memetic algorithms can efficiently navigate complex search spaces and converge to high-quality solutions. Memetic algorithms represent a synergy between exploration and exploitation, where the global exploration is complemented by the local improvement of solutions. This integration enables memetic algorithms to achieve better convergence rates and solution

quality compared to traditional metaheuristics. The relationship between hybridization, memetic algorithms, and metaheuristics is symbiotic, as hybridization techniques enhance the effectiveness of metaheuristics by integrating diverse optimization strategies, while memetic algorithms represent a specific paradigm within the broader framework of metaheuristics, offering powerful solutions for complex optimization problems across various domains.

Parallel metaheuristics

Parallel metaheuristics represent an advanced approach that leverages parallel computing architectures to enhance the efficiency and scalability of traditional metaheuristics. By executing multiple search processes concurrently, parallel metaheuristics can explore the solution space more comprehensively and accelerate the convergence towards high-quality solutions. This parallelization can take various forms, such as parallel evaluations of candidate solutions, parallel exploration of the search space, or parallelization of algorithmic components like selection, crossover, and mutation in genetic algorithms. The relationship between parallel metaheuristics and traditional metaheuristics is symbiotic, as parallelization techniques augment the capabilities of metaheuristics, enabling them to tackle larger and more complex optimization problems within reasonable time frames. Parallel metaheuristics also benefit from the underlying principles and methodologies of traditional metaheuristics, leveraging their adaptive, exploratory, and heuristic-driven nature to guide the parallel search process effectively. Together, parallel metaheuristics and traditional metaheuristics represent a powerful arsenal of optimization techniques, capable of addressing a wide range of challenging optimization problems across various domains.

Nature-inspired and metaphor-based metaheuristics

Nature-inspired and metaphor-based metaheuristics draw inspiration from natural phenomena, biological processes, and abstract concepts to develop optimization algorithms capable of efficiently solving complex problems. These metaheuristics mimic the behaviors observed in natural systems and utilize metaphorical representations of problem-solving strategies to guide the search process. Examples include genetic algorithms inspired by the principles of natural selection and genetic inheritance, ant colony optimization inspired by the foraging behavior of ants, and particle swarm optimization inspired by the collective behavior of bird flocks or fish schools. Nature-inspired metaheuristics leverage concepts

such as population dynamics, communication mechanisms, and adaptation to explore and exploit the solution space effectively. By harnessing the power of nature's mechanisms, these algorithms offer robust and scalable approaches for addressing a wide range of optimization problems across diverse domains. The metaphor-based approach, on the other hand, utilizes abstract concepts and analogies to model problem-solving strategies, offering innovative solutions inspired by human cognition and creativity. Through their adaptive and heuristic-driven nature, nature-inspired and metaphor-based metaheuristics represent powerful tools for tackling optimization challenges and finding near-optimal solutions in complex and dynamic environments.

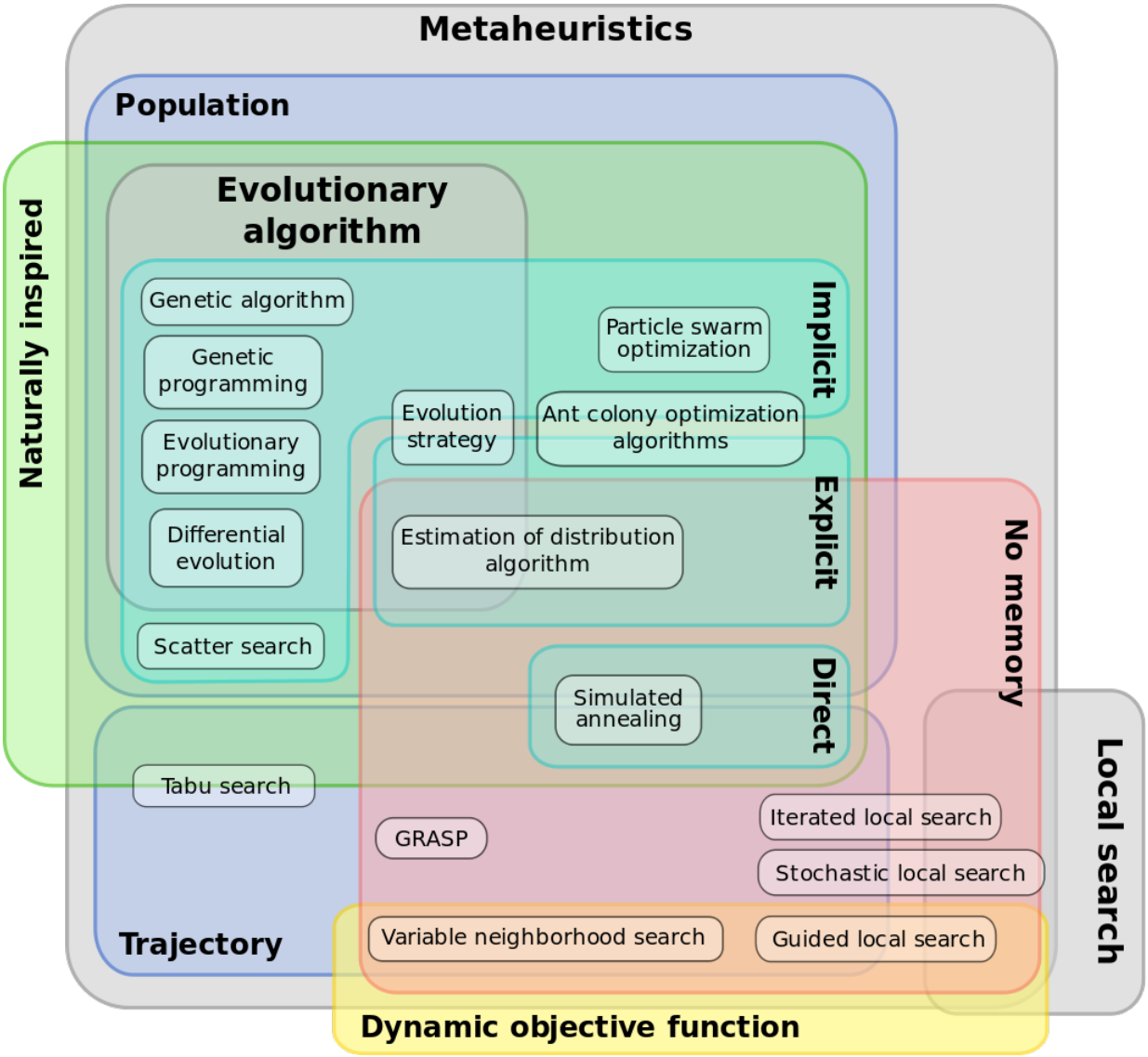

Figure 2.12: Euler diagram of the different classifications of metaheuristics[3]

2.4.5 Using machine learning for enhancing metaheuristics

The study of how machine learning methods have been applied to improve metaheuristics has been split into two sections for the sake of clarity. The first portion examines local-level hybridizations, while the second section examines global-level hybridizations. Each of them has been categorized by the appropriate topic .

2.5 Games-inspired algorithms

Games-inspired algorithms, or meta-heuristic game-based algorithms are optimization techniques that derive their strategies, rules, and dynamics from various games. These algorithms aim to solve complex optimization problems by simulating game-like scenarios, leveraging the decision-making processes and interactions observed in games to effectively explore and exploit the search space.

Key Characteristics

1. Inspiration from Games - Mechanisms and Strategies: These algorithms adopt mechanisms and strategies from games, such as competition, cooperation, and tactical moves, to guide the search process. - Examples: Football strategies, puzzle-solving methods, tug of war dynamics, and volleyball teamwork are some of the inspirations behind these algorithms.

2. Optimization Focus - Objective: The primary goal is to find near-optimal solutions to complex problems that are computationally expensive or infeasible to solve using traditional exact methods. - Efficiency: By using game-inspired strategies, these algorithms aim to achieve efficient and effective optimization.

3. Intuitive Design - Framework: The rules and strategies of games provide a natural and intuitive framework for algorithm design, making it easier to conceptualize and implement. - Accessibility: This intuitive nature can make the algorithms more accessible for researchers and practitioners from various fields.

4. Diverse Applications - Versatility: These algorithms can be applied to a wide range of optimization problems, including engineering design, scheduling, resource allocation, machine learning, and artificial intelligence. - Customization: The game-inspired mechanisms can be tailored to suit specific problem domains and requirements.

5. Effective Exploration and Exploitation - Exploration: By simulating game dynamics, these algorithms are capable of exploring the search space thoroughly to discover new potential solutions. - Exploitation: They also effectively exploit known good solutions by refining and improving them through game-inspired strategies. - Balance: The balance between exploration and exploitation is crucial for achieving high-quality solutions and avoiding local optima.

2.5.1 Golf Optimization Algorithm

1. Inspiration and Overview:

- The GOA is designed to simulate the rules and strategies of the game of golf, translating these into an optimization framework.

- It is a population-based approach where the positions of members (solutions) are initialized randomly within the problem space.

2. Phases of GOA:

- Exploration Phase: This phase involves searching the global problem space to find promising regions. The status of each GOA member is updated based on Equation figure 1 in the document.

- Exploitation Phase: This phase focuses on refining solutions within the promising regions identified during exploration. The status of each member is updated based on Equation 2.1 and 2.2.

$$X_i^{P1} : x_{i,d} = \{ x_{i,d} + r \times (B_d - I \times x_{i,d}) \tag{2.1}$$

$$X_i^{P2} : x_{i,d} = \{ x_{i,d} + (1 - 2r) \times \frac{lb_d + r \times (ub_d - lb_d)}{t} \tag{2.2}$$

3. Algorithm Pseudocode: - The process begins by inputting the optimization problem information and setting the number of iterations T and the number of GOA members N.

- For each iteration, the algorithm updates the best member and then each member through the exploration and exploitation phases.

- The best solution discovered during the iterations is reported as the final solution.

4. Computational Complexity:

- The initialization complexity of the GOA is $O(Nm)$, where N is the number of members and m is the number of decision variables.

- The update process in each iteration has a complexity of $O(2NmT)$, making the total complexity $O(Nm(1+2T))$[9].

5. Constraints Handling: - The algorithm handles boundary constraints by checking and adjusting the positions of the GOA members if they exceed defined limits using Equations 2.3 and 2.4.

- For equality and inequality constraints, a penalty factor is added to the objective function value when constraints are violated, discouraging infeasible solutions.

$$x_{i,d}^{P1} = \{ x_{i,d}^{P1}, lb_d \leq x_{i,d}^{P1} \leq ub_d ub_d, x_{i,d}^{P1} > ub_d lb_d, x_{i,d}^{P1} < lb_d, \tag{2.3}$$

$$x_{i,d}^{P2} = \{ x_{i,d}^{P2}, lb_d \leq x_{i,d}^{P2} \leq ub_d ub_d, x_{i,d}^{P2} > ub_d lb_d, x_{i,d}^{P2} < lb_d, \tag{2.4}$$

6. Performance Evaluation: - The effectiveness of the GOA is evaluated through simulations on 52 standard objective functions and real-world engineering problems. - Its performance is compared with ten other well-known metaheuristic algorithms, demonstrating the GOA's competitive edge in various optimization scenarios

In summary, the GOA leverages the strategic aspects of golf to navigate and optimize complex problem spaces, utilizing a balance of exploration and exploitation phases to refine solutions iteratively. The algorithm is equipped to handle constraints effectively and shows promise in both standard and real-world applications [9].

2.5.2 Squid Game Optimization Algorithm

1. Inspiration and Overview:

- The Squid Game Optimizer (SGO) is a novel metaheuristic algorithm inspired by the traditional Korean game "Squid Game." The game involves attackers and defenders with the goal of either completing an objective or eliminating the opposing team. The algorithm models this by dividing solution candidates into offensive and defensive groups, simulating their interactions to find optimal solutions.

2. Phases of GOA:

- Initialization: Randomly initialize a population of solution candidates.

- Grouping: Divide candidates into offensive and defensive groups.

- Movement and Interaction: Offensive players move towards defensive players to start a fight, modeled by random movements.

- Winning State Evaluation: Evaluate the objective function to determine the winning state of each player.

- Position Update: Update positions based on the winning states, with successful offensive players moving towards the best solutions and defensive players preparing for new interactions.

- Termination: The process continues until a predefined stopping criterion is met.

3. Algorithm Pseudocode:

- The pseudocode involves initializing positions, dividing players into groups, moving offensive players towards defensive players, evaluating winning states, updating positions, and checking termination criteria. The detailed steps are provided in the document's flowchart and pseudocode sections.

4. Computational Complexity:

- The computational complexity of SGO is analyzed using the "Big O notation." The initialization phase has a complexity of $O(NP \times D)$, where NP is the number of initial solution candidates and D is the problem dimension. The main search loop has a complexity of $O(MxIter \times NP \times D \times 3)$, where MxIter is the number of iterations. The objective function evaluation within the main loop also contributes to the overall complexity.

5. Constraints Handling:

- The SGO algorithm handles constraints using a penalty technique with a static coefficient. This approach ensures that solution variables satisfy boundary conditions, and adjustments are made for variables that violate these conditions. The algorithm is tested on real-world constrained optimization problems from the CEC 2020 benchmark suite.

6. Performance Evaluation:

- The performance of SGO is evaluated using 25 unconstrained mathematical test functions and compared with other metaheuristic algorithms. The evaluation includes 100 independent optimization runs, statistical metrics (mean, standard deviation), and statistical tests (Kolmogorov-Smirnov, Mann-Whitney, Kruskal-Wallis). SGO demonstrates superior performance in terms of convergence rates, solution accuracy, and computational efficiency. It also performs well on real-world optimization problems, achieving better results than other algorithms in most cases.

This summary captures the key points from the document, providing an overview of the Squid Game Optimizer (SGO) algorithm, its phases, pseudocode, computational complexity, constraints handling, and performance evaluation [10].

2.6 Conclusion

In conclusion, Chapter 2 has provided a comprehensive overview of the fundamental concepts essential for understanding the detection of fake profiles and users with violent and threatening behavior on social networks. We began by exploring the evolution of Web 2.0 and the rise of social networks, highlighting the significant impact these platforms have on communication and interaction. The chapter then delved into the specific problem of fake profiles, discussing their characteristics, types, and the challenges they pose to online communities.

We also examined various machine learning models, both supervised and unsupervised, that are commonly used for detecting fake profiles. These models include Naive Bayes, Decision Trees, Support Vector Machines, K-Nearest Neighbors, Logistic Regression, Gradient Boosting, and Random Forests. Each model's strengths and weaknesses were discussed, providing a clear understanding of their applicability to different types of data and problems.

Furthermore, the chapter introduced metaheuristic algorithms, such as Genetic Algorithms, Simulated Annealing, and Particle Swarm Optimization, which are used to optimize the performance of machine learning models. The integration of machine learning with metaheuristics was highlighted as a powerful approach to enhance detection capabilities.

Finally, we explored game-inspired algorithms like the Golf Optimization Algorithm (GOA) and the Squid Game Optimization Algorithm (SGO), which offer innovative strategies for solving complex optimization problems. These algorithms draw inspiration from real-world games and demonstrate the potential for improving the accuracy and efficiency of detection systems.

Chapter 3

Fake Profiles and Violence Behavior Detection on Social Networks : State of the art

3.1 Introduction

In this section, we review the literature on methods for detecting fake profiles.

As social networks become increasingly popular in the modern era, the number of users accessing various social media platforms is rising significantly. This increase has led to a substantial amount of data being shared and stolen.

This chapter examines the literature on spam review detection through the analysis of spammer behavior. Researchers have developed a variety of approaches to identify fake accounts. By comparing this new work with previous research, this study aims to assess its contribution.

Several fake profile detection techniques focus on analyzing social networks and profiles to identify characteristics or differences that help distinguish between real and fake accounts. Algorithms are used to classify the data retrieved from profiles and posts, specifically to develop a system for detecting fraudulent accounts.

Twitter Fake Account Detection 2017

The proposed model for detecting fake accounts on Twitter is based on a classification method that uses a supervised discretization technique named Entropy Minimization Discretization (EMD) on numerical features. The results are then analyzed using the Naïve Bayes algorithm.

- Dataset Preparation

The dataset used for the experiments was collected manually and investigated by three individuals. The common decisions from these independent examinations were selected and put in the dataset. Class decisions were made by examining username, background image, profile image, follower and friends count, description of the account, number of tweets, and content of the tweets. The dataset consists of 501 fake and 499 real account data.

- Evaluation Criteria

The performance of the model was evaluated using Accuracy (ACC), F-Measure, and confusion matrix as the performance metric.

- Experimental Results

Two experiments were conducted:

1. Applying the Naïve Bayes learning Algorithm on the Dataset Using All Attributes without Discretization: The accuracy was 86.1%. 2. Applying the Naïve Bayes learning Algorithm on the Dataset after Discretization: The accuracy increased to 90.9%.

The increase in accuracy after discretization shows that Naive Bayes can work better with discrete values than continuous values.

- Conclusion and Future Work

The proposed approach shows the effects of discretization on Naïve Bayes classification algorithm on social media data. The accuracy of the Naïve Bayes algorithm increased from 85.55% to 90.41% by preprocessing the dataset using discretization technique on selected features. The information loss due to discretization was insignificant compared to the increase in accuracy. [5].

Machine Learning Model for Detecting Fake Facebook Profiles (MDFP) 2019

The document presents the MDFP model, a machine learning approach designed to detect fake Facebook profiles using a combination of supervised and unsupervised mining algorithms. Here are the key components and findings summarized:

Components of the MDFP Model: 1. CRAWLER Module: Developed to collect necessary datasets from Facebook. It gathers friends lists and scans profiles to extract 12 attributes such as profile picture, work, education, and number of mutual friends. Data is stored in a relational database and later formatted for analysis.

2. Analyzer Module: Implemented in RapidMiner Studio 8.0.1, this module applies various data mining algorithms. Its goal is to build a model that classifies profiles into Fake or Real categories based on the extracted attributes. Both supervised and unsupervised algorithms are utilized to classify new data with unknown labels.

Employed Attributes: - The model utilizes 12 attributes, including both behavioral (e.g., check-ins, number of tags) and non-behavioral (e.g., profile picture, education) aspects of Facebook profiles. These attributes are crucial for identifying patterns indicative of fake profiles.

Machine Learning Model (MDFP): - The core of the model involves training a machine learning classifier on labeled data (Fake vs. Real profiles) using the extracted attributes. This allows the model to learn rules and relationships that distinguish between genuine and fake profiles.

Experiments and Results: - Two experiments were conducted to evaluate the model's performance. One experiment used the k-NN model for data imputation to handle profiles with missing values, while another experiment removed these profiles using a filtering operator. - Results indicated that supervised algorithms, particularly the ID3 decision tree algorithm, achieved the highest accuracy compared to unsupervised methods. This underscores the effectiveness of supervised learning in distinguishing between fake and real profiles on Facebook.

In conclusion, the MDFP model leverages machine learning techniques and behavioral attributes extracted from Facebook profiles to effectively detect fake profiles. The integration of supervised algorithms proves crucial in achieving accurate classification results, highlighting its potential for improving security and trust on social media platforms. [4].

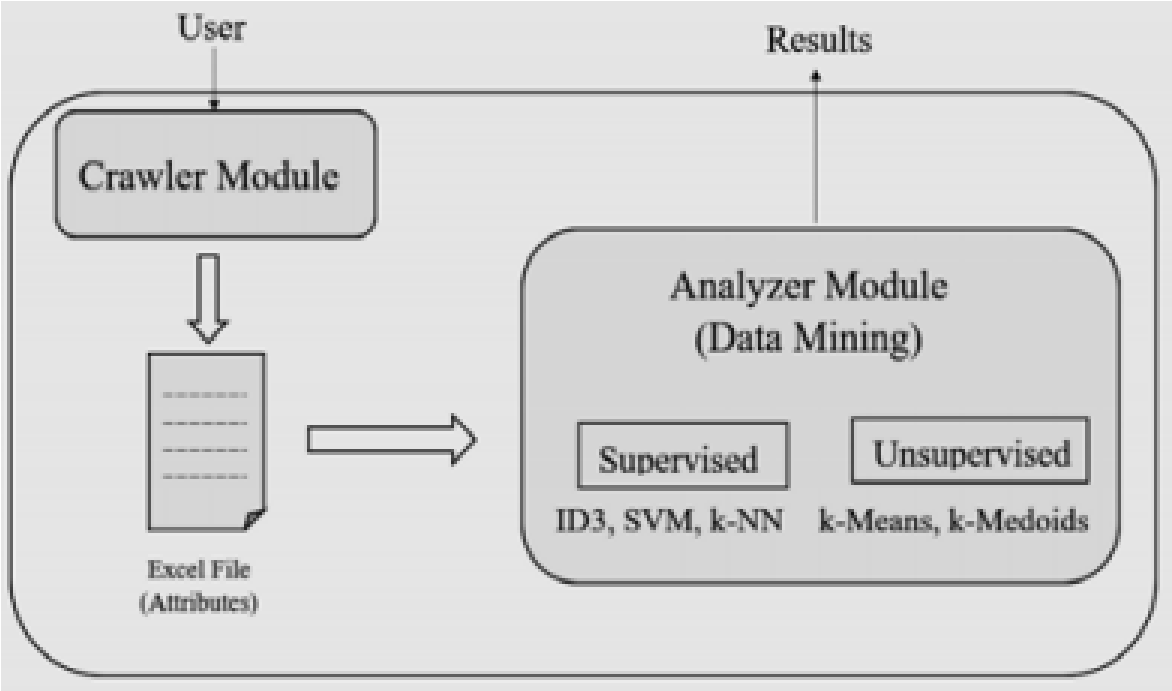

Table 3.1: MDFP System Components [4]

Detection Of Fake Accounts In Instagram Using Machine Learning 2019

The paper proposes a model for detecting fake accounts on Instagram using Machine Learning techniques, specifically Logistic Regression and the Random Forest Algorithm. The model aims to combat cybercrimes targeting women and the proliferation of fake accounts on social media platforms.

Data Collection: - The dataset is sourced from Kaggle and consists of two CSV files: train.csv and test.csv. - The dependent variable indicates whether an account is fake (1) or legitimate (0), evenly distributed at 50% each in the training dataset.

Data Preprocessing: - Missing Value Treatment: The dataset had no missing values, which is crucial for accurate model training. - Outlier Detection: Outliers were addressed using median imputation to maintain model accuracy. - Bivariate Analysis: The authors checked for correlations between variables to avoid multicollinearity, ensuring robust model performance.

Machine Learning Algorithms: - Logistic Regression: Used for predicting probabilities based on predictor variables. It assumes no outliers and low multicollinearity among predictors. - Random Forest Algorithm:Employed for its strong performance in classification tasks. Parameters included 500 trees and 3 variables considered per split.

Performance Evaluation: - Models were evaluated using the test dataset to assess accuracy, precision, recall, and F1 score through a confusion matrix.

Overall, the study addresses the challenge of identifying fake accounts on Instagram by leveraging supervised machine learning techniques and robust evaluation metrics. [11].

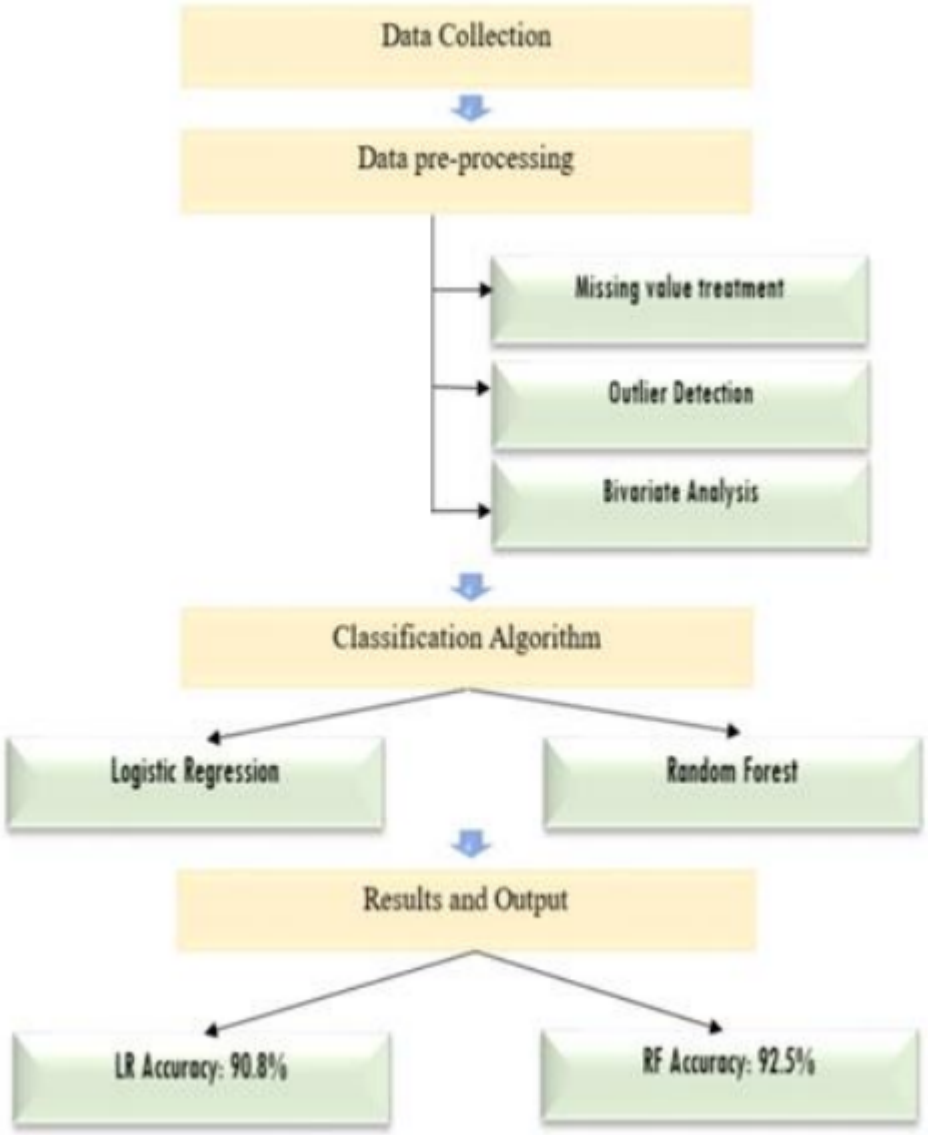

Table 3.2: Proposed System [11]

Detection of Users' Abnormal Behavior on Social Networks 2020

The document proposes a method for detecting abnormal user behavior in multidimensional online social networks, focusing on the following key aspects:

1. Multidimensional Graph Representation: Online social networks are represented as multidimensional graphs where nodes represent users and edges denote relationships across different social network dimensions (e.g., Facebook, Twitter, Instagram).

2. Methodology Phases: - Phase 1: Detect communities within each dimension of the multidimensional graph using an equation that assesses the connectivity between egonets (neighborhoods) of nodes. - Phase 2: Compute an anomaly score (AST(u)) for each node based on its membership and influence within communities across dimensions. This score aggregates individual anomaly scores (AS(u)) from each dimension, considering factors like community membership and influence metrics (DE(u), nbct(u)). - Phase 3: Utilize a Beta mixture model for automatic anomaly detection. This involves fitting Beta distributions to AST(u) scores, using an EM algorithm to estimate distribution parameters, and employing Bayesian Information Criterion (BIC) to determine the optimal number of components. Nodes classified under the component with the lowest scores are identified as anomalies.

3. Novelties and Experimental Results: - The novelty lies in the multidimensional graph representation and the use of a Beta mixture model for automated anomaly detection. - Experimental results on a 3-dimensional network (Facebook, Twitter, Instagram) demonstrated effective anomaly detection. Approximately 10,000 nodes out of 397,000 were identified as anomalous based on their low AST(u) scores within the first Beta component. - Visualization of the adjacency matrix showed sparse connectivity patterns for detected anomalous nodes, contrasting with denser patterns for normal nodes. This validation supports the method's capability to detect abnormal behavior based on structural anomalies in multidimensional network representations. [12].

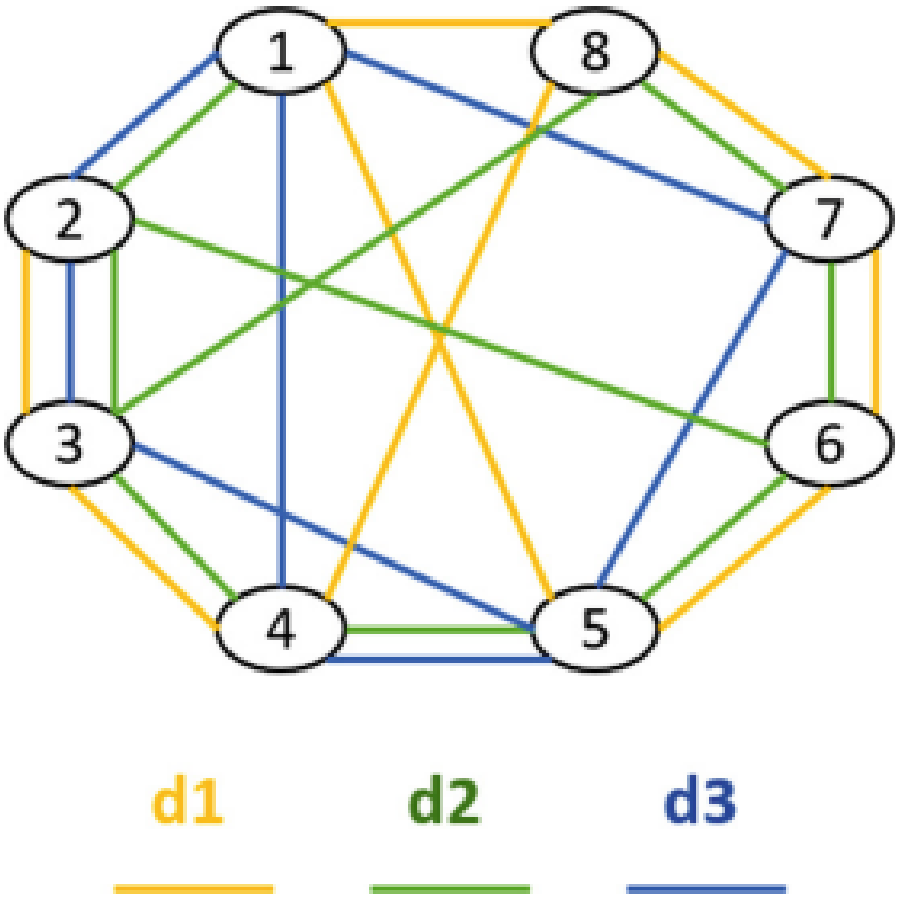

Table 3.3: Example of multidimensional networks [12]

Classification of instagram fake users using supervised machine learning algorithms 2020
The proposed model in the document "Detection of Fake Accounts in Instagram" employs supervised learning algorithms to classify Instagram accounts as fake or real. The model uses a dataset sourced from Kaggle, which includes various features of Instagram accounts. The key steps of the proposed model are as follows:

1. Data Preprocessing: - Missing Value Treatment: The dataset is checked for missing values, which are either deleted or imputed. - Outlier Detection: Outliers in features such as username length, full name words, description length, number of posts, followers, and follows are identified and replaced using median imputation. - Bivariate Analysis: This step ensures the absence of high multicollinearity among variables by calculating the correlation matrix.

2. Model Selection: - Logistic Regression: This model assumes the absence of outliers and high correlations between predictors. It uses the logit function to predict probabilities, and features with p-values greater than 0.05 are removed iteratively. The final model is validated using K-fold cross-validation to check for overfitting. - Random Forest Algorithm: This model does not require stringent data preparation. It involves training multiple decision trees on random subsets of the data and features. The Out of Bag (OOB) error rate is used to estimate the model's performance, and the final classification is determined by majority voting across all trees.

3. Evaluation: - Confusion Matrix: This is used to evaluate the performance of the models on the test dataset, calculating metrics such as precision and recall to assess the accuracy and robustness of the predictions.

The model demonstrates effective classification of Instagram accounts, leveraging the strengths of both logistic regression and random forest algorithms to handle various aspects of the data and improve detection accuracy [7].

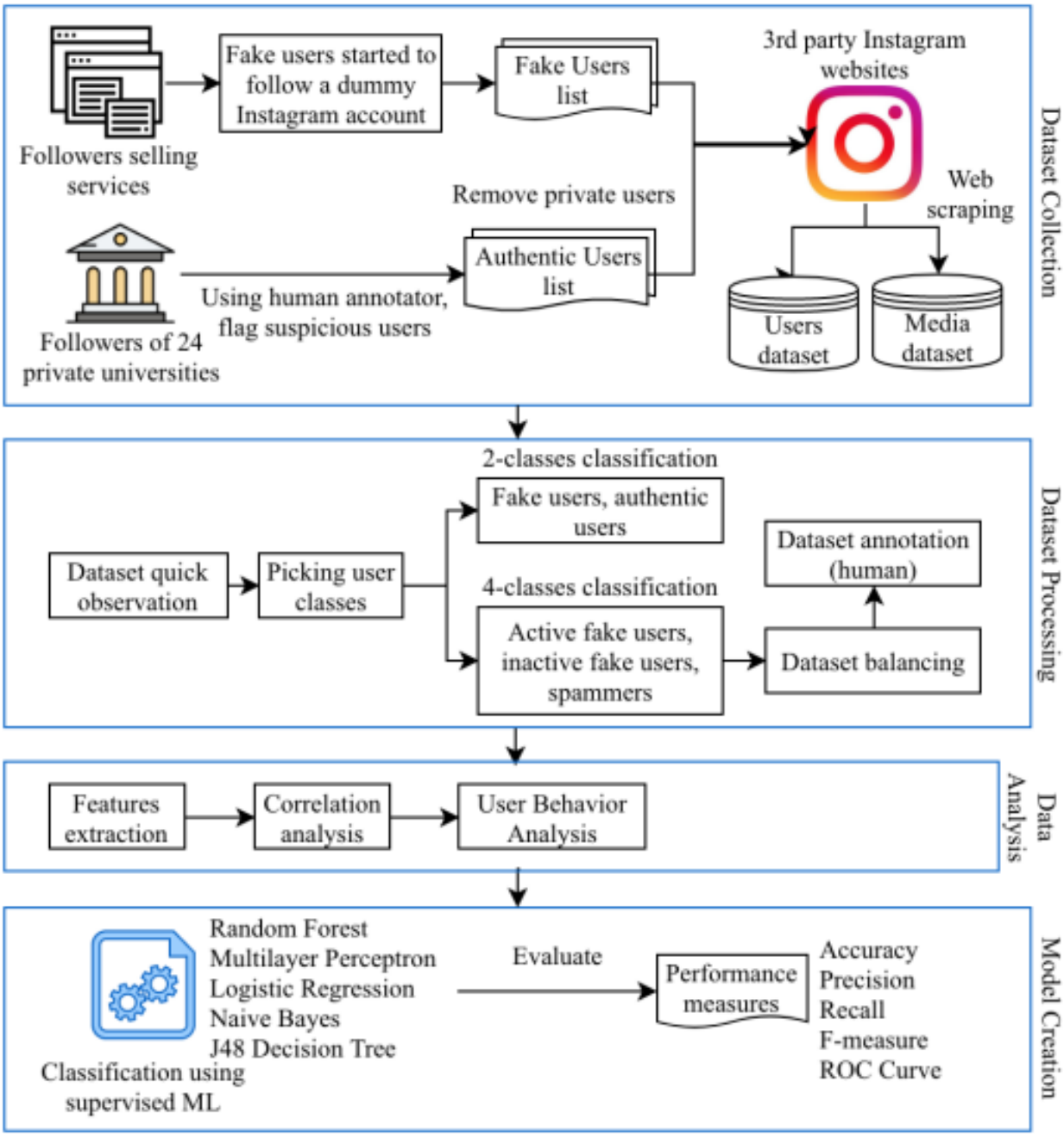

Table 3.4: Research methodology [7]

Applying Machine Learning Models for Detecting and Predicting Militant Terrorists Behaviour in Twitter 2021

The paper proposes a model for detecting and predicting pro-ISIS users on Twitter, comprising three main components:

1. Detection Model: Includes a text classification model using N-gram language models and a CNN-based image classification model for analyzing textual and image data. A decision-making module combines these outputs to classify users as pro-ISIS or anti-ISIS.

2. Prediction Model: Uses item-based collaborative filtering to predict which anti-ISIS users might become pro-ISIS based on their interactions and preferences.

3. Terrorist Network Sociogram: Visualizes the detected and predicted pro-ISIS users and their interactions on Twitter using a network graph.

The model achieved high performance with F1-scores of 0.900 for text classification and 0.920 for image classification. The prediction model using collaborative filtering showed F1-scores of 0.702 for predicting pro-ISIS users and 0.746 for anti-ISIS users. Overall, the model demonstrates effectiveness in detecting and predicting terrorist users on social media platforms.[8].

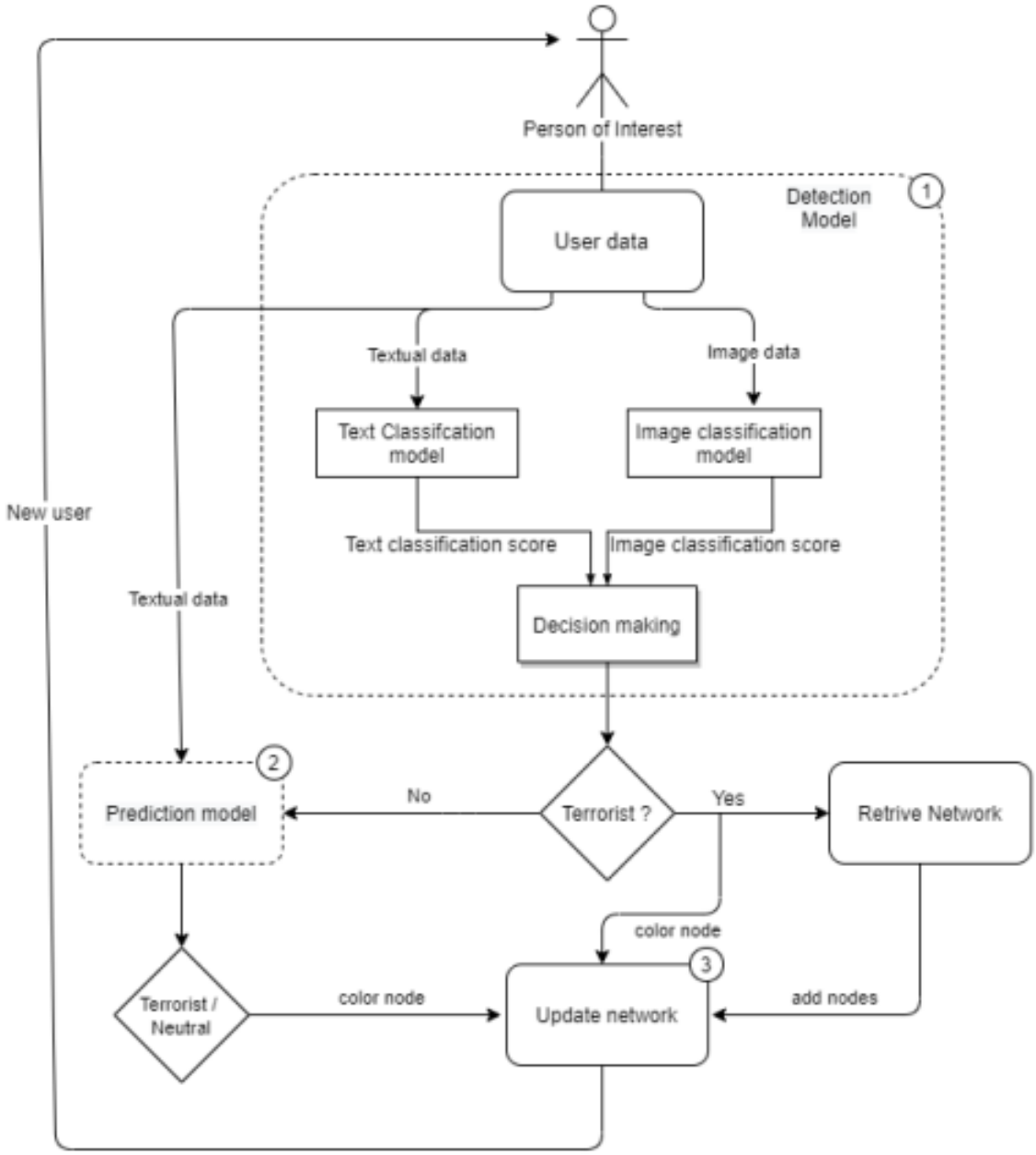

Table 3.5: Proposed Model Workflow [8]

Deep learning methods for anomalies detection in social networks using multidimensional networks and multimodal data: a survey 2021

The article "Deep learning methods for anomalies detection in social networks using multidimensional networks and multimodal data: a survey" evaluates various anomaly detection methods in social networks based on their ability to handle multidimensional networks, multimodal data, community detection, and dynamic behaviors.

1. Activity-based methods: These methods focus on user activities like messages and interactions but struggle with capturing the interconnected nature of social networks. They excel in scenarios where individual behavior is isolated but fall short in understanding complex network relationships and community behaviors.

2. Graph-based methods: Analyzing social network graphs, these methods are strong in handling multidimensional structures and network topology for anomaly detection. However, they often neglect multimodal data analysis and upstream user behaviors.

3. Hybrid methods: Combining features of activity-based and graph-based approaches, hybrids aim to leverage strengths from both but face challenges in fully integrating multidimensional structures and multimodal data. They require further development to achieve comprehensive anomaly detection.

The evaluation in Table 3.2 of the article highlights that no single method currently meets all evaluation criteria effectively. Structural methods handle multidimensional aspects well but lack in multimodal data analysis, while behavioral methods focus on user activities but neglect network relationships.

To address these gaps, the article suggests developing hybrid methods that can integrate multidimensional network management with multimodal data analysis. These hybrids aim to provide a more robust solution by combining strengths from structural and behavioral analyses, thereby enhancing anomaly detection capabilities in social networks.

In conclusion, the article emphasizes the ongoing need for research and development in this area, advocating for hybrid methods as promising approaches to improve anomaly detection in social networks. [13]

Approach	Type	Multidimensional network	Multimodal data	Community detection	Dynamic behavior
[26]	Behavioral	×	✓	×	×
[27]	Behavioral	×	✓	×	✓
[23]	Behavioral	×	×	×	✓
[25]	Behavioral	×	×	×	✓
[12]	Structural	✓	×	×	×
[41]	Structural	×	×	✓	✓
[37]	Structural	×	×	×	×
[40]	Structural	✓	×	✓	✓
[44]	Hybrid	×	×	×	✓
[45]	Hybrid	×	✓	×	×

Table 3.6: Evaluation of key approaches [13]

3.2 Conclusion

In conclusion, the detection of fake profiles, spammers, and abnormal behavior on social media platforms has become an increasingly important area of research. The studies reviewed in this chapter have proposed various machine learning models and techniques to address this challenge across different social media platforms like Twitter, Facebook, and Instagram.

The approaches range from traditional methods like Naive Bayes, decision trees, and logistic regression to more advanced techniques such as convolutional neural networks (CNNs), topic modeling (LDA), and graph-based anomaly detection methods. The use of multimodal data (text and images) and the representation of social networks as multidimensional graphs have also been explored to improve detection accuracy.

While many of these models have demonstrated promising results, achieving high accuracy, precision, recall, and F1-scores, there is still room for improvement. No single method has been able to effectively handle all aspects of the problem, such as multidimensional network structures, multimodal data analysis, community detection, and dynamic behavior analysis.

Moving forward, the development of hybrid models that can effectively integrate these different components appears to be a promising direction. By combining the strengths of structural and behavioral analysis techniques, these hybrid approaches could provide a more comprehensive solution for detecting fake accounts, spammers, and anomalous behavior in social networks.

Additionally, as social media platforms and user behavior continue to evolve, it will be crucial for future research to adapt and refine these models accordingly. Incorporating new data sources, exploring advanced machine learning techniques like deep learning, and leveraging larger and more diverse datasets could further enhance the effectiveness of these detection systems.

Chapter 4

Our contribution

4.1 Introduction

The initial phase of this study involved conducting an in-depth analysis of diverse machine learning algorithms using a comprehensive collection of readily accessible data from Facebook, Twitter, and Instagram profiles. The primary objective was to evaluate the performance of these algorithms based on a range of performance metrics, thus providing valuable insights.

The study's scope was carefully defined, taking into account its objectives and the available resources. The aim was to gain a comprehensive understanding of the behavior of machine learning algorithms when applied to specific datasets extracted from Facebook, Twitter, and Instagram accounts, with the ultimate goal of drawing meaningful conclusions. Subsequently, thorough analysis and comparison of the obtained results with relevant literature allowed for the derivation of insightful findings..

4.2 Data processing

4.2.1 Dataset Collection :

Facebook dataset Description :

From the article we examined they assembled a comprehensive dataset of Facebook accounts for the planned study. The data was collected by exploiting the capabilities and functions offered by the prominent social networking site Facebook. The final dataset is impressive, with 1244 rows and 15 columns containing a wealth of useful information. To ensure the reliability and relevance of the findings, the article employed the meticulously curated Facebook Dataset 9-9-2019. This specific dataset was specifically selected for its comprehensive coverage and up-to-date representation of Facebook accounts. The 1244 accounts in this dataset were carefully categorized and separated as follows:

- Real Accounts: It contains 1043 accounts, 100% human-collected in a research project. project.
- Fake Accounts: It contains 201 fake accounts.

	Legitimate	Fake	Total
Records	1043	201	1244
Percentage	83.84 %	16.16%	100%

Figure 4.1: Facebook dataset description[4]

From the meticulously compiled Facebook dataset, the article extracts a diverse array of informative features that contribute significantly to the effective classification of the data at hand. These carefully selected 14 features offer valuable insights and enable a comprehensive understanding of the underlying patterns and characteristics within the dataset. The chosen 14-feature set encompasses a wide range of essential aspects, as seen below.

- Name-Id.
- Link.
- Profile Picture.
- Number of Likes.
- Number of groups joined.
- Number of friends.
- Education status.
- Work(mentioned or not).
- Living place (mentioned or not).
- Relation-ship.
- CheckIn.
- Number of posts.
- Number of tags.
- profile intro.

Features Analysis : Analysis of several Facebook criteria in the system is described below to identify real accounts from fake ones on Facebook [4]:

Features	Description	Justification
Profile Picture	Visual identification of the user.	Real users use their real pictures more often than fake users.
Work place	Workplace or job title's information,	Real users more often use their real workplace information than fake users.
Education	Attended (school, college, university…etc.) information.	Real users mentioned their education information in their Facebook profiles more often than fake users.
Living Place	Living place address (city, town, state…etc.) information.	Real users more often use their real living place information than fake users
Check-In	Information for announcing user location.	Real users check into places in their Facebook's profiles more often than fake users.
No. of Posts	Social online activities shared on Facebook	Real users have more online activities than fake users.
No. of Tags	Identify the user by someone else on his/ her wall.	Real Users tagged more often than fake users.
Introduction "Bio."	Introduction information about Facebook's users.	Real users are more often write something about themselves than fake users.
No. of Mutual Friends	Number of the people who are Facebook friends with both users and the target profiles.	Real users have more mutual friends with target profile than fake users, hence gives profile more incredibility.
No. of Pages	Number of pages liked.	Real users usually liked more pages than fake users.
No. of Groups	Number of groups joined.	Real users usually join groups more than fake users.
Family\ Relationship	Social relation Information\Status	Real users share their real social relation status than fake users.

Figure 4.2: Attributes descriptions and intuative justification[4]

Twitter Dataset

From the article we examined, the dataset was generated through the utilization of the Twitter API. The Twitter API encompasses four main objects, namely Tweets, Users, Entities, and Locations. Each of these objects exhibits a diverse array of characteristics. The tweeting objects serve as the essential atomic building pieces at their heart. They have many aspects that correspond to general tweet information, such as the creation time, the number of likes received, the number of retweets, and more. It should be noted, however, that protected accounts cannot access these features. On the other hand, user objects encompass a wide range of entities, representing individuals or entities of any nature. These objects contain attributes that provide insights into the general account information, such as the number of tweets liked, the number of followers, and other contextual metadata. Entities objects play a crucial role in providing additional contextual information about the content shared on Twitter. They encompass values present within the tweets, including hashtags, media elements, and URLs. Lastly, locations and items are identified by names that correlate to their corresponding geographic coordinates. The dataset utilized in the article consists of 16 attributes, further enriching the analysis and understanding of the Twitter platform [5].

	Legitimate	Fake	Total
Records	499	501	1000
Percentage	49.9 %	50.01%	100%

Figure 4.3: Twitter dataset description[5]

Attribute Name	Description of the attribute
Description	Length of the user defined string describing the account
Protected	When true, indicates that this user has chosen to protect their Tweets
followers_count	The number of followers this account currently has
friends_count	The number of users this account is following
statuses_count	The number of Tweets (including retweets) issued by the user.
favourites_count	The number of Tweets this user has liked in the account's lifetime
listed_count	The number of public lists that this user is a member of.
Verified	When true, indicates that the user has a verified account
profile_use_background_image	When true, indicates the user wants their uploaded background image to be used
contributors_enabled	Indicates that the user has an account with "contributor mode" enabled, allowing for Tweets issued by the user to be co-authored by another account
default_profile	When true, indicates that the user has not altered the theme or background of their user profile
default_profile_image	When true, indicates that the user has not uploaded their own profile image and a default image is used instead
is_translator	When true, indicates that the user is a participant in Twitter's translator community
hashtags_average	Number of hashtags that user has used in last 20 tweets
mentions_average	Number of mentions that user has used in last 20 tweets
urls_average	Number of URL links that user has used in last 20 tweets

Figure 4.4: Features of Twiter dataset[5]

Instagram Dataset

From the article we used they obtained The dataset from Kaggle.com, a popular platform for data science and machine learning enthusiasts. The dataset consists of two CSV files: test.csv and train.csv. These files contain information about user accounts and their authenticity.

The main objective of this dataset is to determine whether a given user account is fake or real. To accomplish this, the dataset includes a dependent variable, which is categorical and has two possible values: 0 for real accounts and 1 for fake accounts. This variable serves as the target or label for our machine learning models.

One important characteristic of the training dataset is its balanced distribution. The data is structured in such a way that 50% of the instances are fake accounts, while the

remaining 50% are real accounts. This balanced distribution ensures that the models are trained on an equal number of examples from each class, allowing them to learn effectively without being biased towards either category.

By using this dataset, various machine learning algorithms can be employed to build models that accurately classify user accounts as real or fake. These models can then be utilized to detect and prevent the creation of false profiles, which is crucial for maintaining the integrity and security of online platforms.[6]

	A	B	C	D	E	F	G	H	I	J	K	L
1	profile pic	nums/len	fullname	nums/len	name==us	descriptio	external U	private	#posts	#follower	#follows	fake
2	1	0.27	0	0	0	53	0	0	32	1000	955	0
3	1	0	2	0	0	44	0	0	286	2740	533	0
4	1	0.1	2	0	0	0	0	1	13	159	98	0
5	1	0	1	0	0	82	0	0	679	414	651	0
6	1	0	2	0	0	0	0	1	6	151	126	0
7	1	0	4	0	0	81	1	0	344	669987	150	0
8	1	0	2	0	0	50	0	0	16	122	177	0
9	1	0	2	0	0	0	0	0	33	1078	76	0
10	1	0	0	0	0	71	0	0	72	1824	2713	0
11	1	0	2	0	0	40	1	0	213	12945	813	0
12	1	0	2	0	0	54	0	0	648	9884	1173	0
13	1	0	2	0	0	54	1	0	76	1188	365	0
14	1	0	2	0	0	0	1	0	298	945	583	0
15	1	0	2	0	0	103	1	0	117	12033	248	0
16	1	0	2	0	0	98	1	0	487	1962	2701	0
17	1	0	3	0	0	46	0	0	254	50374	900	0
18	1	0	3	0	0	0	0	0	59	7007	289	0
19	1	0.29	3	0	0	48	0	0	1570	1128	694	0
20	1	0	2	0	0	63	1	0	378	34670	1878	0
21	1	0	2	0	0	106	1	0	526	2338	776	0
22	1	0	2	0	0	40	0	0	228	3516	999	0
23	1	0	1	0	0	35	1	1	35	1809	416	0
24	1	0	2	0	0	30	0	0	281	427	470	0
25	1	0	1	0	0	27	0	0	285	759	956	0

Figure 4.5: Instagram Dataset[6]

Exploratory Data Analysis: This is a vital step of early data exploration that is carried out in order to detect patterns in the dataset and identify abnormalities using summary statistics and graphical representation. The many separate processes completed are shown here.[6]

The Second Instagram Dataset

According to the article we used The data from research on Instagram fake users was collected using web scraping from third-party Instagram websites. The process captured metadata and up to 12 latest media posts from each user, and was carried out from September 1st to 20th, 2019. The dataset includes both authentic and fake users, with the latter being identified by human annotators.

Authentic Users: These were sourced from followers of 24 private university pages (8 Indonesian, 8 Malaysian, 8 Australian) on Instagram. Users were selected using proportional random sampling based on their source university. Private users were excluded, leaving a total of 32,460 public users for the research.

Fake Users: These were obtained by purchasing followers from Indonesian sellers on various platforms, including Instagram and the Kaskus forum. The followers satisfied the three types of fake users mentioned in the document's first section. A dummy Instagram business account was created for these followers to follow. According to the analytics data from this account, the top five countries of the fake followers were Indonesia (17%), India (13%), Turkey (9%), Pakistan (8%), and Russia (7%).[7].

Classification Type	Fake Users	Authentic Users	Total Users	Media from Fake Users	Media from Authentic Users
2-Classes Classification	32869	32460	65329	376357	460923
4-Classes Classification	32866	10441	43307	376357	141371

Figure 4.6: THE Details of the second Instagram Dataset[7]

Var name	Feature name	Description
pos	Num posts	Number of total posts that the user has ever posted.
flg	Num following	Number of following
flr	Num followers	Number of followers
bl	Biography length	Length (number of characters) of the user's biography
pic	Picture availability	Value 0 if the user has no profile picture, or 1 if has
lin	Link availability	Value 0 if the user has no external URL, or 1 if has
cl	Average caption length	The average number of character of captions in media
cz	Caption zero	Percentage (0.0 to 1.0) of captions that has almost zero (<=3) length
ni	Non image percentage	Percentage (0.0 to 1.0) of non-image media. There are three types of media on an Instagram post, i.e. image, video, carousel
erl	Engagement rate (Like)	Engagement rate (ER) is commonly defined as (num likes) divide by (num media) divide by (num followers)
erc	Engagement rate (Comm.)	Similar to ER like, but it is for comments
lt	Location tag percentage	Percentage (0.0 to 1.0) of posts tagged with location
hc	Average hashtag count	Average number of hashtags used in a post
pr	Promotional keywords	Average use of promotional keywords in hashtag, i.e. {regrann, contest, repost, giveaway, mention, share, give away, quiz}
fo	Followers keywords	Average use of followers hunter keywords in hashtag, i.e. {follow, like, folback, follback, f4f}
cs	Cosine similarity	Average cosine similarity of between all pair of two posts a user has
pi	Post interval	Average interval between posts (in hours)

Figure 4.7:
User features of the second Instagram Dataset[7]

Social Network Dataset

The data from the article we examined includes two distinct datasets used for research on social media accounts.

The legitimate account (Social Honeypot) Dataset is a publicly available dataset created by Lee et al. in 2010. The dataset was developed by deploying 60 seed social accounts on Twitter with the aim of attracting spammers. These seed accounts reported back on the

accounts that interacted with them, enabling the researchers to collect data from 19,276 legitimate users and 22,223 spammers over a span of 7 months (we used only about 1% of the dataset).

The fake account (Weibo) Dataset, in contrast, was self-collected. Sina Weibo, one of the most popular social platforms in China, provided 2,197 legitimate user accounts. The spammers were commercially purchased from multiple vendors on the Internet. After a manual checking process, 802 suitable ”smart” spammer accounts were collected.[16]

Feature	Description
UFN	standard deviation of following standard deviation of followers the number of following following and followers ratio
UC	|links| per tweet |@username| in tweets / |tweets| |unique @username| in tweets / |tweets| |unique links| per tweet
UH	the change rate of number of following

Figure 4.8:
User features of Honeypot Feature Groups[16]

ISIS Dataset

The data from the article we examined reference includes two distinct datasets used for research on social media activity and terrorism.

- Positive Labels: The researchers were interested in datasets previously shared among pro-ISIS users on social networks. They found the ”How ISIS uses Twitter” dataset, which contains 17,350 tweets from more than 110 pro-ISIS accounts. It includes the following attributes: Name, Username, Description, Location, Number of followers at the time the tweet was downloaded, Number of statuses by the user when the tweet was downloaded, Date and timestamp of the tweet, and the tweet itself. Some of the tweets in the dataset are typed in Arabic, so the researchers used a public translation tool, Google Translate API, each time they detected Arabic characters.

- Negative Labels: he researchers used the Global Terrorism Database (GTD) as part of their negative labeled dataset. The dataset contains information about more than

180,000 terrorist attacks (but we used only 17302) from all around the word since 1970 [8].

9/9/2006: Eight armed members of the Ibad Errahman Brigade, a faction of the Salafist Group for Preaching and Fighting (GSPC), ambushed a patrol from the Taher Judicial Police Mobile Brigade (BMPJ) th...	9/9/2002: A man from Bombay attempted to hijack an Air Seychelles flight bound for Seychelles from Bombay, Maharashtra state, India. The plane had already taken off from Bombay when the knife-wieldin...	9/8/2002: Two suspected Muslim militants attacked a Hindu family in Dodasanpai village, Jammu and Kashmir, India. The militants stormed the family's home and opened fire, killing two women and three ...
9/7/2008: Two perpetrators, believed to be affiliated with the Islamic insurgency in Algeria, shot and killed a member of the Legitimate Defense Group (GLD) at his place of business in Stah Kentis, Al...	9/7/2007: Al-Qa ida in the Lands of the Islamic Maghreb (AQLIM) members killed the father of Abdelkader Benmessaoud (aka Mossaab Abou Daoud) in Boukahil, Algeria. Benmessaoud was a former AQLIM comma...	9/8/2002: Gunmen on a speedboat opened fire on the Muslim village of Kulur on Haruku island, Maluku province, Indonesia. A Muslim woman and two young girls were killed in the shooting. No group clai...

Figure 4.9: Excerpt from GTD of the summary column[8]

Dataset	Positive (Pro-ISIS)	Negative (Anti-ISIS)	Total
Text-based	17350	26894	44244
Image-based	341	308	649

Figure 4.10:
TEXTUAL-CONTENT AND IMAGE-CONTENT DATASETS[8]

4.2.2 Dataset preprocessing

A lot of information has been gathered from numerous sources, including the internet, questionnaires, and tests, among others. Nevertheless, the majority of the time, the data that must be used are distorted, noisy, and full of missing values.Data preprocessing is a fundamental step to data analysis and machine learning ,is a set of procedures which transforms raw data into understandable format .

e.g : Facebook dataset has two feature of vectors types:

- Categorical features: such as name, Intro,Profile Picture, Living Place,Check-In .
- Numerical features: such as Likes,Mutual friends , Groups,Posts,Tags .

We considered a few classification algorithms before moving on to the dataset. We also give numerical feature types some thought. The numerical aspects of other categorical features had also been altered.

We attempted to evaluate the most important attributes because the dataset has numerous attributes. We substituted each example's Name-Id in the Facebook dataset with a numerical id in order to exclude attributes like Link and Name-Id that are not statistically significant from our model. Applying various ML algorithms to the dataset is crucial.

To accurately categorize the dataset, filtering is applied. Whether the dataset is accurately classified by the classification method and contains no incorrect or null values.

After that, as part of the data preprocessing stage, we had normalized the data. In order to preserve the information, it is important to convert the dispersed huge numerical values to a common scale of [0,1] without distorting the value range-differences. Several algorithms require this phase in order to properly model the data.

When the features in a dataset have a variety of measuring units, normalization of the data is necessary. Use of normalization, which involves rescaling the chosen characteristics from their original values to the scale of 0 to 1, is a helpful strategy when the data distribution is confusing, unpredictable, or not Gaussian.

Text Classification Model:

The detection process of the textual input data consists of three main parts as illustrated in Figure 4.11.

Natural Language Processing:

used for pre-processing the textual data in order for us to have a ready structured data that is easy to understand and process. The process of analyzing the textual data is performed in four main steps: tagging, annotating, co-reference and sentiment analysis.

Word embedding :

We have opted for the N-gram language model that estimates the probability of the last word given the previous words. This choice was motivated by the fact that it showed much better results than the TF-IDF model in.

Classification:

After word embedding, the textual content is now in a numerical form, understandable by the machine, and ready to be used by any classifier as an input.

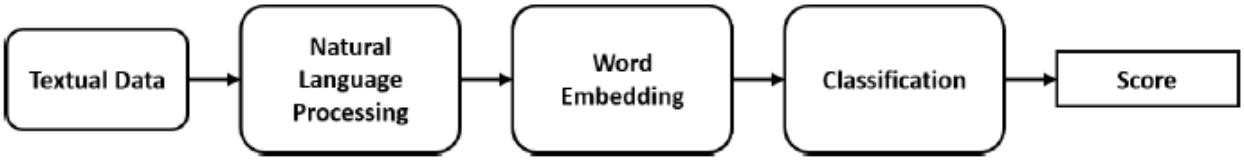

Figure 4.11: Text classification model design[8]

4.2.3 Features selection

We use feature selection when a dataset contains redundant features or when the outcomes are relatively unaffected by the characteristics.

Correlation matrix

In this section we have used the **correlation matrix** to detect the highest correlation between the features and the class . We use the correlation matrix for showing correlation coefficients between variables. Each cell in the table shows the correlation between two variables and to summarize data, as an input into a more advanced analysis, and as a diagnostic for advanced analyses.

This is done to understand the relationship between two variables and the strength of association between them. We calculated the correlation matrix and concluded absence of high multicollinearity between the variables.

4.2.4 Cleaning and scaling

- **Missing Value Treatment** : To handle missing values in my datasets, I chose to remove any rows containing incomplete data. This method ensures that any entries with missing information are excluded, thereby maintaining the integrity and reliability of the dataset. By cleaning the data in this way, I ensured that the subsequent analysis and modeling processes would not be compromised by gaps in the information. This approach is particularly crucial in datasets with user-generated content, where missing values can frequently occur and potentially skew the results. Removing these incomplete entries allowed for a more accurate and effective analysis of the dataset, this approach was used on Facebook dataset that contain some missing values.

  Also i opted to handle missing values in Twitter dataset by utilizing the medium calculation method. This approach involves replacing NaN values with the median value of the respective feature. By leveraging the median, which represents the middle value of a dataset, I aimed to mitigate the potential skewing effect of outliers that could occur with mean imputation. This method allows for the preservation of the dataset's central tendency while effectively handling missing data points. Additionally, employing the median calculation ensures robustness against extreme values, making it a reliable strategy for datasets prone to outliers. Overall, utilizing the median calculation to handle missing values provided a balanced and statistically sound approach to data preprocessing, contributing to more accurate analyses and model outcomes.

- **Data Reduction** : Removing columns that don't contribute much to the data, such as identifiers or columns mostly filled with zeros, can enhance the efficiency and effectiveness of machine learning models. For instance, dropping identifier columns like Id names typically doesn't affect the predictive power of the model since they merely serve as unique identifiers without bearing any meaningful information for prediction. Similarly, eliminating columns predominantly consisting of zeros reduces the dimensionality of the dataset, streamlining the learning process and potentially

preventing overfitting to noise. By conducting such preprocessing steps, the resulting model becomes more focused on relevant features, thereby improving interpretability and generalization performance. However, it's essential to carefully assess the impact of these removals on the overall dataset and model performance, ensuring that valuable information isn't inadvertently discarded. Through thoughtful feature selection and preprocessing, machine learning models can more effectively extract meaningful patterns and insights from the data, ultimately leading to more robust and accurate predictions.

4.2.5 Training fake profile detection models

Cross-Validation :

One method for assessing a machine learning model's efficacy is cross validation (CV). A statistical technique known as cross-validation compares and evaluates learning algorithms by splitting data into two segments: one for learning or training a model and the other for model validation.

Data Classification :

- We two classification types were used (**supervised and unsupervised learning**) to compare their results .
- Machine learning programs classify future accounts into fake or real with the aid of pre-categorized training datasets and a variety of techniques.
- We divide the data into two sets: a Training set and a Testing set.
- Our model is shown the training set, and it collects the data from it.
- When machine learning algorithms are used to make predictions on data that was not used to train the model, their performance is estimated using the train-test split technique.
- After the model is trained, its accuracy is tested using the testing set, which is kept from the model throughout training. In contrast to the testing set, which contains only the features and requires the model to predict the related label, the training set contains both the features and the related label.

- We have used a test-train split of **30% -70%** to compare the performance of machine learning algorithms for the predictive modeling fake detection problem .
  - Train Dataset: Used to fit the machine learning model.
  - Test Dataset: Used to evaluate the fit machine learning model.
- The purpose is to evaluate the machine learning model's performance on additional data that were not used to train the model.

Parameters tuning

For every model, certain parameters were selected and provided with a range of possibilities. These parameters are the ones that have high impact towards detecting the illegitimate accounts and learning rate. This will then be implemented within game based algorithms.

4.2.6 Testing fake profile detection models

Machine learning models

The three primary aspects of our proposed model—data preprocessing, data reduction, feature selection, and data classification—are described in depth in this section. Processing the dataset came first in our effort, and in the second stage, we added additional reduction strategies. The data was filtered and reduced using several reduction mechanisms in the reduction phase to prepare it for the classification phase, where the filtered data was run through various classification algorithms and the results were shown.

We have used in our comparative study the next supervised models :Naive Bayes ,K-Nearest Neighbors ,Decision Tree Model, Random Forest Model, Logistic regression Model, Gradient Boosting Model. And we've choose the k-means model as an unsupervised technique .

In order to use the machine learning methods outlined above In order to assess different machine learning algorithms, we created the test scenario using Python. The models are used to investigate the empirical link between the features and the probability of an illegitimate account. Since the goal of this study is to explore existing datasets of false accounts using a variety of statistical techniques and run machine learning algorithms on them, it is a quantitative case study.

The data set for this investigation was initially modeled without feature selection. To prevent overfitting in all experiments, k-fold cross validation was utilized, as well as parameter adjustment to get the ideal model parameters.

Implementation of Algorithm

- Step 1: Load and read the datasets
- Step 3: Clean the data by filling the missing values
- Step 4: Divide the all dataset in two parts: Test dataset and Train dataset
- Step 5: Apply different machine learning techniques
- Step 6: Generate the confusion matrix of each technique
- Step 7: Calculated the values of evaluation parameters of each techniques
- Step 8: Compare the values of evaluation parameters of each technique and analyze the results

we did the implementation 3 time, each time with an approach

- Without Normalization
- With Normalization
- With Selected Features

4.2.7 Chosen performance evaluation metrics

The detection of Fake accounts can be evaluated by different performance measures, e.g. F1 score, confusion matrix, Recall... In our study, we have used ACC (Accuracy), F-score,Recall , precision and entropy as the performance metric . Confusion Matrix is being used to visualise the detection of the fake accounts for models.

- TP = True Positives, when our model correctly classifies the data point to the class it belongs to.
- FP = False Positives, when the model falsely classifies the data point.
- TN = These are the cases where the predicted “No” actually belonged to class “No”.

- FN = These are the cases where the predicted “No” actually belonged to class “Yes”.

- Precision is used to calculate the model's ability to classify values correctly. It is given by dividing the number of correctly classified profiles by the total number of classified data points for that class label.

  Recall is used to calculate the ability of the model to predict positive values. But, ”How often does the model predict the correct positive values?”. This is calculated by the ratio of true positives and the total number of actual positive values.

- F1-score F1 score should be used when both precision and recall are important for the use case. F1 score is the harmonic mean of precision and recall. It lies between [0,1].

- Entropy measures the randomness or disorders in a system.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{4.1}$$

$$Precision = \frac{TP}{TP + FP} \tag{4.2}$$

$$Recall = Sensitivity = \frac{TP}{TP + FN} \tag{4.3}$$

$$F1 = \frac{2 * Precision * Recall}{Precision + Recall} = \frac{2 * TP}{2 * TP + FP + FN} \tag{4.4}$$

$$Entropy = \log 2(Precision) \times -Precision \tag{4.5}$$

4.3 Fake profiles detection system

Metaheuristic algorithms are one type of method to solve various optimization problems in different fields regardless of the possible multi-level complexity of these problems, which presents great challenges. Accordingly, the capability of these algorithm should be evaluated by means of difficult optimization problems,

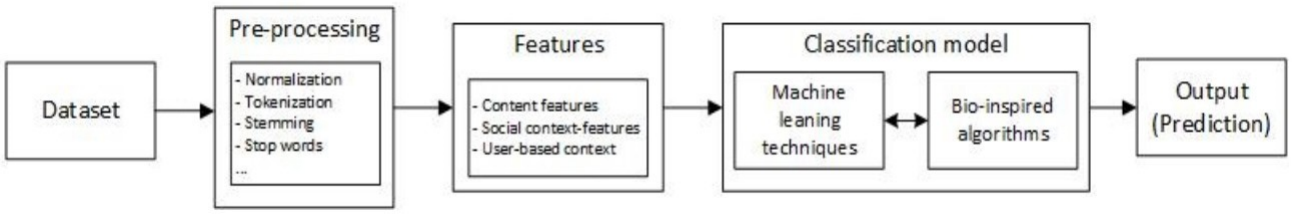

Figure 4.12: Proposed system flowchart.

4.3.1 Machine learning Parameters

- Facebook dataset

Model	Parameters
DecisionTreeClassifier	• `criterion`: criterion used to measure the quality of a split (default: "gini") • `splitter`: strategy used to choose the split at each node (default: "best") • `max_depth`: maximum depth of the tree (default: None) • `min_samples_split`: minimum number of samples required to split an internal node (default: 2)
RandomForestClassifier	• `n_estimators`: number of trees in the forest (default: 100) • `criterion`: criterion used to measure the quality of splits in each tree (default: "gini") • `max_depth`: maximum depth of each tree (default: None) • `min_samples_split`: minimum number of samples required to split an internal node in each tree (default: 2)
GradientBoostingClassifier	• `n_estimators`: number of boosting stages to perform (default: 100) • `learning_rate`: learning rate, controls the contribution of each tree (default: 0.1) • `max_depth`: maximum depth of each tree (default: 3) • `min_samples_split`: minimum number of samples required to split an internal node (default: 2)
KNeighborsClassifier	• `n_neighbors`: number of neighbors to consider (default: 5) • `weights`: weight function used in prediction (default: "uniform")
GaussianNB	No specific parameters to manually set
KMeans	• `n_clusters`: number of clusters to form (default: 8) • `init`: method for initialization of centroids (default: "k-means++") • `random_state`: seed for random number generator for initialization
GaussianMixture	• `n_components`: number of mixture components (default: 1) • `covariance_type`: type of covariance matrix to use ("full", "tied", "diag", "spherical") (default: "full") • `random_state`: seed for random number generator for initialization
SVC (Support Vector Machine)	• `kernel`: kernel used in the algorithm ("linear", "poly", "rbf", "sigmoid", "precomputed") (default: "rbf") • `C`: regularization parameter (default: 1.0) • `random_state`: seed for random number generator for initialization

Figure 4.13: Facebook dataset parameters

- Twitter dataset

Model	Parameters
DecisionTreeClassifier	• `max_depth`: maximum depth of the tree (default: None)
RandomForestClassifier	• `n_estimators`: number of trees in the forest (default: 100) • `max_depth`: maximum depth of each tree (default: None)
GradientBoostingClassifier	• `n_estimators`: number of boosting stages to perform (default: 100) • `learning_rate`: learning rate, controls the contribution of each tree (default: 0.1) • `max_depth`: maximum depth of each tree (default: 3)
KNeighborsClassifier	• `n_neighbors`: number of neighbors to consider (default: 5)
GaussianNB	No specific parameters to manually set
KMeans	• `init`: method for initialization of centroids (default: "k-means++") • `n_init`: number of times the K-means algorithm will be run with different centroid seeds (default: 10)
GaussianMixture	• `covariance_type`: type of covariance matrix to use ("full", "tied", "diag", "spherical") (default: "full")
SVC (Support Vector Machine)	• `C`: regularization parameter (default: 1.0) • `kernel`: kernel used in the algorithm ("linear", "rbf") (default: "rbf") • `gamma`: kernel coefficient for 'rbf', 'poly', and 'sigmoid' (default: "scale")

Figure 4.14: Twitter dataset parameters

- Instagram dataset

Model	Parameters
DecisionTreeClassifier	• `max_depth`: maximum depth of the tree (default: None)
RandomForestClassifier	• `n_estimators`: number of trees in the forest (default: 100) • `max_depth`: maximum depth of each tree (default: None)
GradientBoostingClassifier	• `n_estimators`: number of boosting stages to perform (default: 100) • `learning_rate`: learning rate, controls the contribution of each tree (default: 0.1) • `max_depth`: maximum depth of each tree (default: 3)
KNeighborsClassifier	• `n_neighbors`: number of neighbors to consider (default: 5)
GaussianNB	No specific parameters to manually set
SVC	• `kernel`: kernel used in the algorithm ("rbf" for Radial Basis Function) (default: "rbf") • `C`: regularization parameter (default: 1.0) • `gamma`: kernel coefficient for 'rbf' (default: "auto")
KMeans	• `n_clusters`: number of clusters to form (default: 8)
GaussianMixture	• `n_components`: number of mixture components (default: 1) • `covariance_type`: type of covariance parameters to use ("full", "tied", "diag", "spherical") (default: "full")

Figure 4.15: Instagram dataset parameters

- The Second Instagram dataset

Model	Parameters
DecisionTreeClassifier	• `criterion`: Function to measure split quality (default: 'gini') • `max_depth`: Maximum tree depth (default: None)
RandomForestClassifier	• `n_estimators`: Number of trees (default: 100) • `max_depth`: Maximum tree depth (default: None)
GradientBoostingClassifier	• `n_estimators`: Number of boosting stages (default: 100) • `learning_rate`: Controls contribution of each tree (default: 0.1) • `max_depth`: Maximum tree depth (default: 3)
KNeighborsClassifier	• `n_neighbors`: Number of neighbors (default: 5)
GaussianNB	No specific parameters to manually set
SVC	• `kernel`: Kernel type used in the algorithm (default: 'rbf') • `C`: Regularization parameter (default: 1.0) • `gamma`: Kernel coefficient for 'rbf' (default: 'auto')

Figure 4.16: The Second Instagram dataset parameters

- Social Network dataset

Model	Parameters
AdaBoostClassifier	• `n_estimators`: 50 (default) • `learning_rate`: 1.0 (default) • `random_state`: 42
RandomForestClassifier	• `n_estimators`: 100 (default) • `criterion`: 'gini' (default) • `max_depth`: None (default) • `min_samples_split`: 2 (default) • `min_samples_leaf`: 1 (default) • `random_state`: 42
SVC (Support Vector Classifier)	• `C`: 1.0 (default) • `kernel`: 'rbf' (default) • `degree`: 3 (default, relevant for 'poly' kernel) • `gamma`: 'scale' (default) • `random_state`: 42
DecisionTreeClassifier	• `criterion`: 'gini' (default) • `splitter`: 'best' (default) • `max_depth`: None (default) • `min_samples_split`: 2 (default) • `min_samples_leaf`: 1 (default) • `random_state`: 42
GradientBoostingClassifier	• `loss`: 'deviance' (default) • `learning_rate`: 0.1 (default) • `n_estimators`: 100 • `subsample`: 1.0 (default) • `criterion`: 'friedman_mse' (default) • `max_depth`: 3 (default) • `random_state`: 42
KNeighborsClassifier	• `n_neighbors`: 5 • `weights`: 'uniform' (default) • `algorithm`: 'auto' (default) • `leaf_size`: 30 (default) • `p`: 2 (default)
GaussianNB	• No specific parameters to manually set

Figure 4.17: Social Network dataset parameters

- ISIS dataset

Modèle	Paramètres
SVM (Support Vector Machine)	• `kernel`: 'linear' • `probability`: True
MultinomialNB (Naive Bayes)	• `alpha`: 1.0 (default) • `fit_prior`: True (default)
LogisticRegression	• `max_iter`: 1000 • `penalty`: 'l2' (default) • `C`: 1.0 (default)

Figure 4.18: ISIS dataset parameters

4.3.2 Transition from natural to artificial of GOA metaheuristic

This part is dedicated to the transition from the natural life of the Game Golf to the life artificial as shown in the following table :

Aspect	Natural (Golf Game)	Artificial (GOA)
Inspiration	The strategic dynamics and player conduct observed in the sport of golf.	Simulates the rules and behavior of players in the game of golf.
Phases	Players' actions in the game, such as strong shots (drives) and precise putts.	Two phases: Exploration (strong shots) and Exploitation (precise putts).
Exploration Phase	Players try to have the strongest shot towards the hole.	Scans different areas of the search space, indicating the exploration ability. $x_i^{P1,d} = x_i^d + r * (B_d - I * x_i^d)$ $x_i =$ if $F_i^{P1} < F_i$, then x_i^{P1} otherwise, x_i
Exploitation Phase	Players try to put the golf ball into the hole with precise putts.	Carefully scans the area in which each GOA member is located, indicating exploitation ability. $x_i^{P2,d} = x_i^d + (1 - 2r) * lb_d + r * (ub_d - lb_d) / t$ $x_i =$ if $F_i^{P2} < F_i$, then x_i^{P2} otherwise, x_i
Population Initialization	Not applicable in natural golf.	Randomly initializes the position of GOA members in the search space using: $x_i^d = lb_d + r * (ub_d - lb_d)$
Objective Function	Not applicable in natural golf.	Evaluates the value of the objective function for each GOA member using: $F = [F(x_1), ..., F(x_N)]^T$
Update Process	Players adjust their strategy based on the position of the ball.	Updates the position of GOA members based on exploration and exploitation phases.
Constraints Handling	Players follow the rules of the game.	Uses penalty factors to handle constraints in the optimization problem using: $F_i = F_i + n_q * PF_i$
Iteration Process	Players take turns to hit the ball until it reaches the hole.	Repeats the update process until the algorithm is fully implemented.
Computational Complexity	Not applicable in natural golf.	Analyzed based on the number of decision variables, population size, and iterations.

Figure 4.19: GOA Transition from natural to artificial

4.3.3 How the chosen metaheuristic ”GOA” was used

The GOA is used to handle optimization problems by simulating the rules and behavior of players in golf. This novel approach involves two main phases: exploration and exploitation, which are mathematically modeled to navigate the search space effectively.

- Fundamental Inspiration: The GOA derives its strategy from the gameplay of golf, involving the precise and strategic movement of the golf ball towards the hole. This strategy is translated into a methodological framework and mathematical model for

optimization applications.

- Phases of Implementation: The GOA's implementation is divided into two phases: - Exploration Phase: This phase involves searching the problem space broadly to avoid local optima. - Exploitation Phase: This phase focuses on intensively searching around the best solutions found so far to refine them further.

- Evaluation and Comparison: The effectiveness of the GOA is evaluated using fifty-two standard objective functions. Additionally, its performance is compared with ten well-known metaheuristic algorithms to establish its efficiency and robustness in solving optimization problems.

- Real-world Applications: The GOA's capability is tested on four engineering design problems and the optimization of the operation of energy carriers concerning energy grid resilience, demonstrating its practical applicability[9].

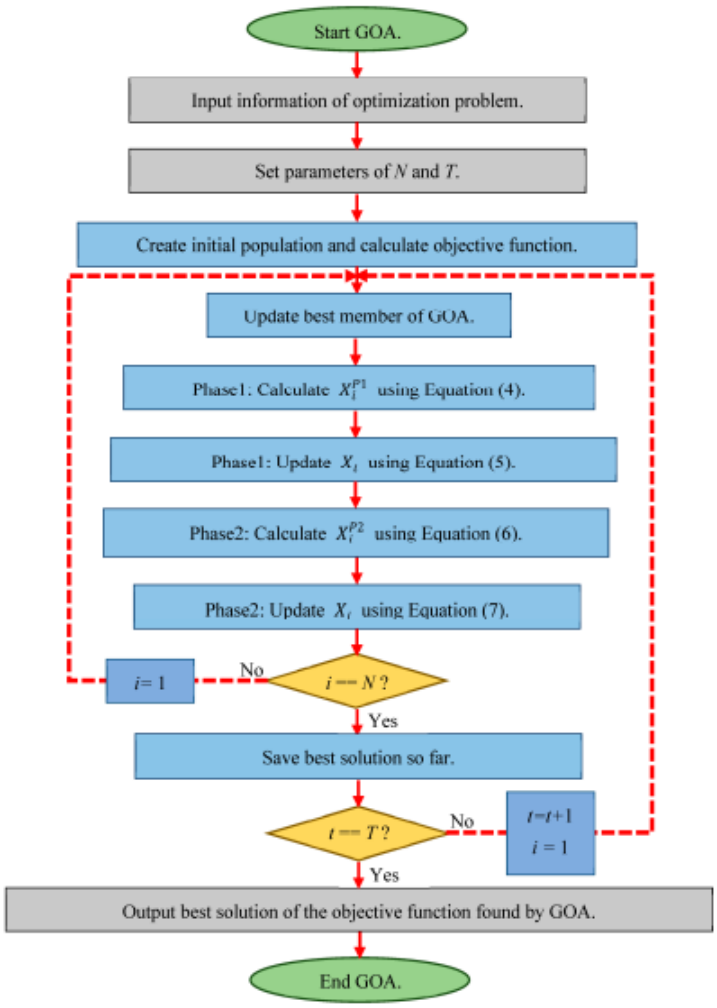

Figure 4.20: Flowchart of the GOA [9]

4.3.4 GOA Metaheuristic Parameters

The parameters of the GOA are explicitly described in terms of the population initialization, the update mechanisms during the exploration and exploitation phases, and the iterative process to find the optimal solution.

- Population Initialization: The GOA starts with a population-based approach, where the initial positions of the GOA members (candidate solutions) are randomly distributed across the problem space using a uniform distribution. This is mathematically represented in the document and ensures a diverse set of starting points for

the optimization process.

- Objective Function Evaluation: Each member's position in the problem space determines the values of the problem variables, and the objective function values for these members are calculated. The member with the best objective function value is identified and updated iteratively.

- Repetition Process: The GOA repeats the exploration and exploitation phases iteratively. The pseudocode provided in the document outlines the steps: - Input optimization problem information. - Set the number of iterations (T) and the number of GOA members (N). - For each iteration, update the best member and calculate new positions for each member based on the exploration and exploitation equations provided. - Save the best candidate solution so far and continue until the maximum number of iterations is reached.

- Flowchart and Computational Complexity: The document includes a flowchart and analysis of the computational complexity of the GOA, detailing how the algorithm proceeds through its iterations and updates its members to converge on an optimal solution [9].

4.3.5 Pseudo code

the complete set of codes is available this **website**[1][9].

[1]https://www.mathworks. com/matlabcentral/fileexchange/133817-golf-optimization-algorithm-goa

Algorithm 1. Pseudocode of the GOA.

Start GOA.

Input the optimization problem information.
Set T (number of iterations) and N (number of GOA members).
For $t = 1{:}T$
Update best member of GOA as hole.
    For $i = 1{:}N$
        Phase 1:
        Calculate new status of ith GOA member based on exploration phase of GOA using Equation (4).
        Update ith GOA member using Equation (5).
        Phase2: Exploitation
        Calculate new status of ith GOA member based on exploitation phase of GOA using Equation (6).
        Update ith GOA member using Equation (7).
    end
    Save best candidate solution so far.
end
Output best obtained solution.

End GOA.

Figure 4.21: Pseudocode of the GOA [9]

4.3.6 Used fitness function

```python
def GOA(SearchAgents, Max_iterations, lowerbound, upperbound, dimension, fitness_function):
    lowerbound = np.array(lowerbound)
    upperbound = np.array(upperbound)
    X = np.random.uniform(low=lowerbound, high=upperbound, size=(SearchAgents, dimension))
    fit = np.apply_along_axis(fitness_function, 1, X)
    fbest, blocation = fit.max(), fit.argmax()
    Xbest = X[blocation, :].copy()
    GOA_curve = []
    for t in range(1, Max_iterations + 1):
        for i in range(SearchAgents):
            if np.random.rand() < 0.5:
                I = np.round(1 + np.random.rand())
                RAND = np.random.rand()
            else:
                I = np.round(1 + np.random.rand(dimension))
                RAND = np.random.rand(dimension)
            X_P1 = X[i, :] + RAND * (Xbest - I * X[i, :])
            X_P1 = np.maximum(X_P1, lowerbound)
            X_P1 = np.minimum(X_P1, upperbound)
            L = X_P1
            F_P1 = fitness_function(L)
            if F_P1 > fit[i]:
                X[i, :] = X_P1
                fit[i] = F_P1
        for i in range(SearchAgents):
            if t != 0:
                X_P2 = X[i, :] + (1 - 2 * np.random.rand()) * (lowerbound / t + np.random.rand() * (upperbound / t - lowerbound / t))
                X_P2 = np.maximum(X_P2, lowerbound)
                X_P2 = np.minimum(X_P2, upperbound)
                L = X_P2
                F_P2 = fitness_function(L)
                if F_P2 > fit[i]:
                    X[i, :] = X_P2
                    fit[i] = F_P2
        if fit.max() > fbest:
            fbest = fit.max()
            blocation = fit.argmax()
            Xbest = X[blocation, :].copy()
        GOA_curve.append(fbest)
        print(f"Iteration {t}/{Max_iterations}, Best Score: {fbest}")
    return fbest, Xbest, GOA_curve
```

Figure 4.22: GOA fitness function

4.3.7 Transition from natural to artificial of SGO metaheuristic

This part is dedicated to the transition from the natural life of the Squid Game to the life artificial as shown in the following table :

Aspect	Natural	Artificial
Inspiration	Traditional Korean game "Squid Game" with players (offensives and defensives)	Metaheuristic algorithm inspired by the rules and strategies of the Squid Game
Playground	Squid-shaped playfield, approximately half the size of a basketball court	Search space considered as a playground with solution candidates (players) initialized randomly $X = [x_1^1, x_2^1, ..., x_d^1]$ $X = [x_1^2, x_2^2, ..., x_d^2]$ ... $X = [x_1^n, x_2^n, ..., x_d^n]$
Players	Divided into offensive and defensive teams, with specific roles and movements	Solution candidates divided into offensive and defensive populations, each modeled to perform specific actions $XOff = [x_1^1, x_2^1, ..., x_d^1]$ $XOff = [x_1^2, x_2^2, ..., x_d^2]$ ... $XOff = [x_1^m, x_2^m, ..., x_d^m]$ $XDef = [x_1^1, x_2^1, ..., x_d^1]$ $XDef = [x_1^2, x_2^2, ..., x_d^2]$ ... $XDef = [x_1^m, x_2^m, ..., x_d^m]$
Movement & Fight	Offensive players move towards defensive players to start a fight, aiming to reach a goal	Offensive players move towards defensive players' positions, with movements modeled as random steps towards solutions $XOffNew_i = XOff_i + r_1 \times DG - r_2 \times XDef_{r3}$
Winning State Evaluation	Players' winning states determined by their ability to reach section C or eliminate opponents	Winning states evaluated based on an objective function, determining success in reaching optimal solutions $WSDef_i \leq WSOff_i$
Group Updates	Winners join the Successful Offensive Group (SOG) or Successful Defensive Group (SDG)	Offensive winners update their positions towards best solutions, forming SOG; Defensive winners protect critical points $XSccOff = [x_1^1, x_2^1, ..., x_d^1]$ $XSccOff = [x_1^2, x_2^2, ..., x_d^2]$ ... $XSccOff = [x_1^o, x_2^o, ..., x_d^o]$ $SDG = [x_1^1, x_2^1, ..., x_d^1]$ $SDG = [x_1^2, x_2^2, ..., x_d^2]$ ... $SDG = [x_1^p, x_2^p, ..., x_d^p]$
Bridge Passing	Offensive players attempt to pass the bridge protected by defensive players	Successful offensive players move towards the best solution and defensive positions, akin to passing a bridge $XOffNew3_i = XSccOff_i + r_1 \times BS - r_2 \times XSccDef_k$
Boundary Handling	Physical boundaries of the playground	Mathematical bounds enforced on solution variables, with adjustments for out-of-bound values
Termination	Game ends when all offensive players are eliminated or a player reaches the goal	Optimization process ends based on predefined criteria such as iterations or function evaluations

Figure 4.23: SGO Transition from natural to artificial

4.3.8 How the chosen metaheuristic was used

The chosen metaheuristic, the Squid Game Optimizer (SGO), was utilized as a metaheuristic algorithm inspired by the rules and strategies of the traditional Squid Game. The SGO leverages metaheuristic principles to optimize solutions in a manner that requires less computational effort than traditional optimization algorithms, iterative methods, or simple heuristics, particularly in combinatorial optimization problems.

Metaheuristics, including the SGO, are employed for various types of optimization problems, ranging from continuous through mixed-integer problems. They are designed to find, generate, or select heuristics that may provide sufficiently good solutions to optimization problems, especially when dealing with incomplete or uncertain information.

The SGO algorithm utilizes the principles of metaheuristics to guide the movement and interactions of solution candidates (players) within a defined search space, aiming to reach optimal solutions with less computational effort. This approach allows the algorithm to efficiently explore the solution space and find good solutions for complex optimization problems.

Furthermore, metaheuristic algorithms, including the SGO, have been applied in various real-world applications such as computational intelligence, fixture and manufacturing cell design, foreign exchange trading, robotics, medical science, and behavioral science, highlighting their flexibility, adaptability, and extensive search capacity.

In summary, the SGO leverages metaheuristic principles to efficiently explore solution spaces and find good solutions for complex optimization problems, making it a valuable approach for a wide range of applications[10].

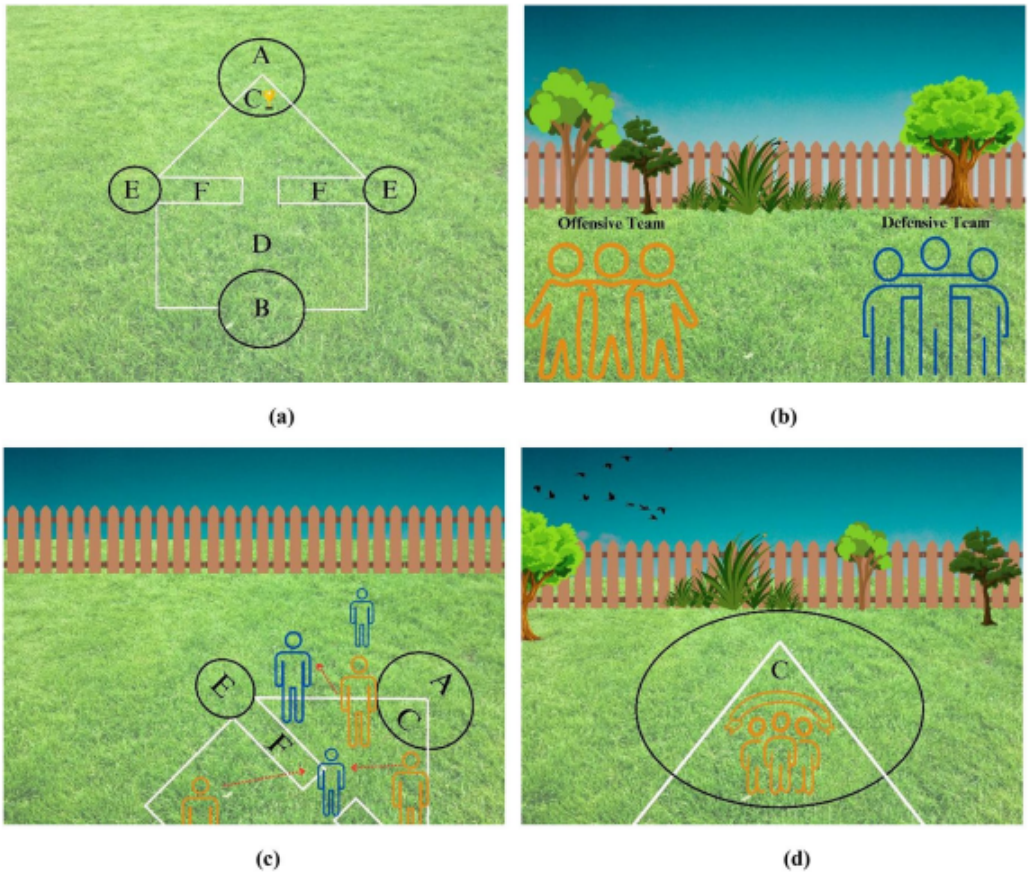

Figure 4.24: The presentation of Squid Game.[10]

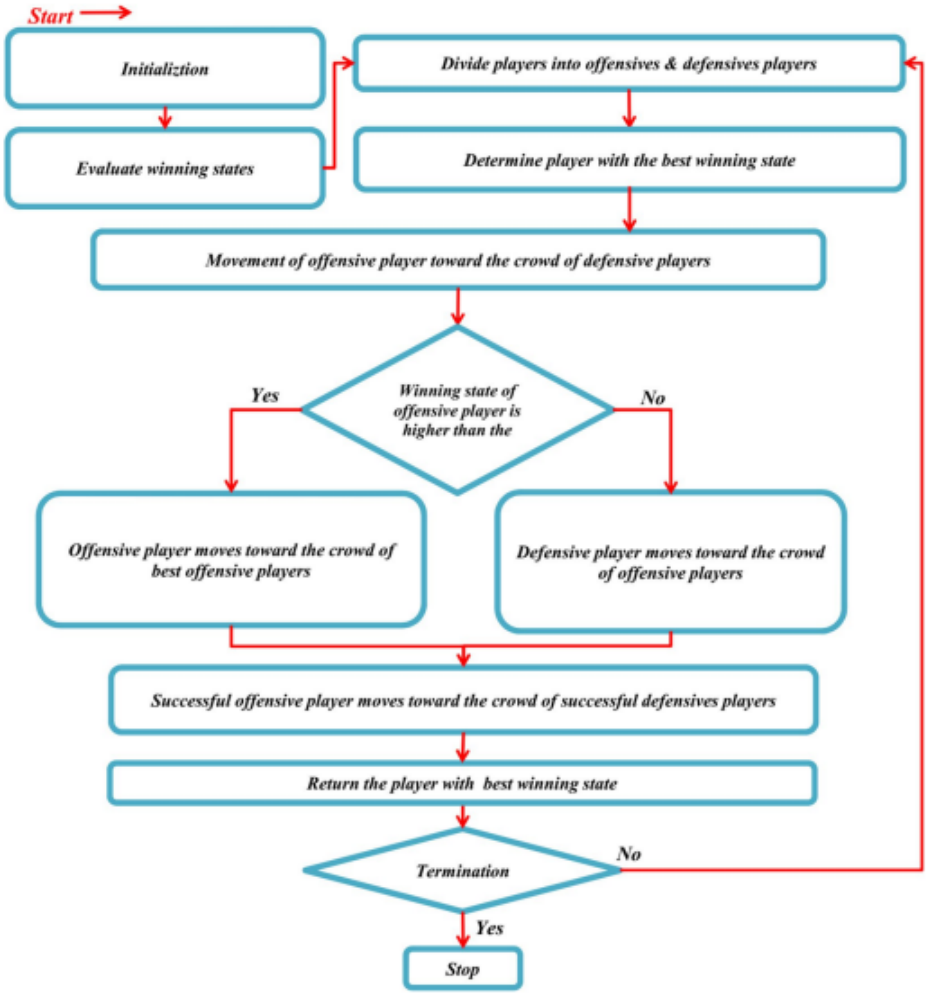

Figure 4.25: Flowchart of the SGO[10]

4.3.9 SGO Metaheuristic Parameters

The chosen metaheuristic, the Squid Game Optimizer (SGO), likely involves the use of algorithmic control parameters to guide its optimization process. These control parameters play a crucial role in influencing the behavior and performance of metaheuristic algorithms. The search results indicate that metaheuristic optimization algorithms typically use stochastic operators, making each run unique, and often have algorithmic control parameters that have an unpredictable impact on convergence.

The effect of these control parameters on algorithm performance is significant, and their values can greatly influence the convergence and quality of solutions obtained. How-

ever, the literature on structural optimization often disregards the impact of these control parameters, making it challenging to formulate general conclusions. Therefore, methods to assess the performance of a metaheuristic algorithm in relation to its control parameter values have been developed to address this issue.

Additionally, the search results highlight the importance of tuning the control parameters of metaheuristic algorithms for fair comparison and optimal performance. Various methodologies and approaches, such as design of experiments, mixture design, and hybrid methods based on data envelopment analysis (DEA) and response surface methodology (RSM), have been proposed for tuning the control parameters of metaheuristic algorithms.

In summary, the SGO likely involves the use of algorithmic control parameters, and the tuning of these parameters is crucial for achieving optimal performance and obtaining high-quality solutions in the optimization process [10].

4.3.10 Pseudo code

this is the proved code in the article

Evaluate fitness values for initial solution candidates as winning state of players (WS_i)
while *Iteration (Number of Function Evaluation) < Maximum number of iterations (Function Evaluations)*
Divide n players into two groups as offensives & defensives with m players in each group
Determine the player with the best winning state (BS) in the playground
for *i=1:m*
The ith offensive player moves toward the crowd of defensive players (Eq. 6)
Determine the winning state of ith offensive (WS_i^{Off}) and defensive players (WS_i^{Def})
if $WS_i^{Def} \leq WS_i^{Off}$
The ith offensive player moves toward the crowd of best offensive players (Eq. 9)
The ith offensive player joins the SOG
else $WS_i^{Def} > WS_i^{Off}$
The ith defensive player moves toward the crowd of offensive players (Eq. 12)
The ith defensive player joins the SDG
end if
end for
for *i=1:o*
The ith successful offensive player moves toward the crowd of successful defensives (Eq. 13)
end if
end while
Return the player with the best winning state (BS).
end Procedure

Figure 4.26: Pseudocode of the SGO [10]

4.3.11 Used fitness function

```python
def SGO(max_iterations, num_players, lowerbound, upperbound, dimension, objective_function):

    lowerbound = np.array(lowerbound)
    upperbound = np.array(upperbound)
    players = np.random.uniform(lowerbound, upperbound, (num_players, dimension))
    fitness_values = np.array([objective_function(player) for player in players])
    for iteration in range(max_iterations):
        num_offensive = num_players // 2
        offensive_players = players[:num_offensive]
        defensive_players = players[num_offensive:]
        best_index = np.argmax(fitness_values)
        best_player = players[best_index]
        for i in range(num_offensive):
            new_pos = move_towards_defensive(offensive_players[i], defensive_players, lowerbound, upperbound)
            if objective_function(new_pos) > fitness_values[i]:
                offensive_players[i] = move_towards_best_offensive(offensive_players[i], best_player, lowerbound, upperbound)
            else:
                defensive_players[i] = move_towards_offensive(defensive_players[i], offensive_players, lowerbound, upperbound)
        players = np.vstack((offensive_players, defensive_players))
        fitness_values = np.array([objective_function(player) for player in players])
        print(f"Iteration {iteration + 1}/{max_iterations}, Best Score: {fitness_values[best_index]}")
    best_index = np.argmax(fitness_values)
    return fitness_values[best_index], players[best_index]

def move_towards_defensive(offensive_player, defensive_players, lowerbound, upperbound):
    defensive_player = defensive_players[np.random.randint(len(defensive_players))]
    new_pos = offensive_player + np.random.rand(len(offensive_player)) * (defensive_player - offensive_player)
    return np.clip(new_pos, lowerbound, upperbound)

def move_towards_best_offensive(offensive_player, best_player, lowerbound, upperbound):
    new_pos = offensive_player + np.random.rand(len(offensive_player)) * (best_player - offensive_player)
    return np.clip(new_pos, lowerbound, upperbound)

def move_towards_offensive(defensive_player, offensive_players, lowerbound, upperbound):
    offensive_player = offensive_players[np.random.randint(len(offensive_players))]
    new_pos = defensive_player + np.random.rand(len(defensive_player)) * (offensive_player - defensive_player)
    return np.clip(new_pos, lowerbound, upperbound)
```

Figure 4.27: SGO fitness function

4.3.12 Experimental software environment

4.3.13 Testing software environment

Different tools are used for this study. All of them are free and open source.

- Python 3.10
- NumPy 1.11.3
- Matplotlib 1.5.3
- Pandas 0.19.1
- SciPy and Scikit-learn 0.18.1
- Jieba

- Joblib
- NLTK 3.8.1
- Jupyter Notebook

Python [2] is a high level general programming language and is very widely used in all types of disciplines such as general programming, web development, software development, data analysis, machine learning etc. Python is used for this project because it is very flexible and easy to use and also documentation and community support is very large.

NumPy [3]is very powerful package which enables us for scientific computing. It comes with sophisticated functions and is able to perform N-dimensional array, algebra, Fourier transform etc. NumPy is used very where in data analysis, image processing and also different other libraries are built above NumPy and NumPy acts as a base stack for those libraries

Pandas [4] is open source BSD licensed software specially written for python programming language. It provides complete set of data analysis tools for python and is best competitor for R programming language. Operations like reading data-frame, reading csv and excel files, slicing, indexing, merging, handling missing data etc., can be easily performed with Pandas. Most important feature of Pandas is, it can perform time series analysis

SciPy[5]is a collection of mathematical algorithms and convenience functions built on the NumPy extension of Python. It adds significant power to the interactive Python session by providing the user with high-level commands and classes for manipulating and visualizing data. With SciPy, an interactive Python session becomes a data-processing and system-prototyping environment rivaling systems, such as MATLAB, IDL, Octave, R-Lab, and SciLab.

For this study, scikit-learn is used because it is based on python and can interoperate to NumPy library. It is also very easy to use. **Scikit-Learn** (SKLearn) [6] is an environment that is integrated with Python programming language. The library offers a wide range of

[2]https://www.python.org/
[3]https://numpy.org
[4]https://pandas.pydata.org
[5]https://docs.scipy.org/doc/scipy/tutorial/general.html
[6]https://scikit-learn.org

supervised algorithms . The library offers high-level implementation to train with the 'Fit' methods and 'predict' from an Classifier and also offers to perform the cross validation, feature selection and parameter tuning

Jieba [7] is a Python library specifically designed for Chinese text segmentation. It is considered the best Chinese word segmentation module in Python

Joblib [8] Joblib is a Python library that provides tools for running computationally intensive tasks in parallel and for caching the results of computationally expensive functions. It is particularly useful for machine learning models as it allows you to save the state of your computation and resume your work later or on a different machine

NLTK [9] NLTK (Natural Language Toolkit) is a Python library specifically designed for working with human language data and performing various natural language processing (NLP) tasks. It provides a wide range of tools and resources for tasks such as tokenization, stemming, tagging, parsing, semantic reasoning, and more

kaggle[10] Kaggle is a data science competition platform and online community that brings together data scientists, machine learning practitioners, and enthusiasts from around the world. It provides a platform for individuals to participate in data science challenges, collaborate with others, and showcase their skills in solving real-world problems.

Jupyter Notebook[11]is the original web application for creating and sharing computational documents. It offers a simple, streamlined, document-centric experience.

4.4 Conclusion

This chapter has made significant contributions to the field of social media analysis, with a particular focus on datasets from Facebook, Twitter, Instagram, as well as a social network and an ISIS-related Twitter dataset. The techniques used to preprocess these datasets have played a vital role in ensuring that the data is of high quality and reliable for further analysis. Additionally, the development and training of models to detect fake profiles have provided valuable insights into identifying deceptive accounts across these platforms. By utilizing various tools and technologies, this study has successfully created a comprehensive system flowchart that outlines the step-by-step process of preprocessing,

[7]https://pypi.org/project/jieba/
[8]https://joblib.readthedocs.io/en/stable/
[9]https://www.nltk.org
[10]https://www.kaggle.com
[11]https://jupyter.org/

detection, and evaluation. Moreover, specific metrics have been carefully selected to evaluate the effectiveness of the proposed system for detecting fake profiles. The integration of these contributions not only enhances our understanding of deceptive practices on social media but also has practical implications for improving security measures and user trust.

Chapter 5

Results and discussion

5.1 Introduction

This section outlines the results from our analysis of the datasets and the comparison of different algorithms. After pre-processing, descriptive analysis, and exploratory analysis, we applied various machine learning techniques to the datasets.

The findings from the experiments are detailed in the tables below, including an explanation of the best-performing method based on multiple performance metrics. The results are organized in the tables as follows, grouped by datasets:

- Data without normalization
- Data with normalization
- Data with feature selection
- List of Confusion Matrix
- Correlation Matrix
- GAO Optimization Meta-heuristic implementation
- Comparing all best models in ML with GAO Optimization Meta-heuristic implementation
- Comparing some of our ML with scientific articles
- conclusion

5.1.1 Facebook dataset

Dataset Size

The size of the dataset was calculated using Pandas' memory_usage() method, which is designed to compute the memory consumption of a DataFrame. Initially, upon loading the dataset from a CSV file, the method memory_usage(deep=True).sum() was applied. The parameter deep=True ensures that the calculation includes the memory used by the actual data within each column, such as strings in object columns. This calculation provides an accurate measure of the dataset's initial memory footprint in bytes. After performing preprocessing tasks, such as dropping columns, handling missing values, and encoding categorical variables, the dataset was modified. The size of the updated DataFrame was then recalculated using the same method to reflect any changes in memory consumption due to these operations.

Data Type	Size (MB)
Initial dataset	0.34
Final Preprocessed dataset	1.24

Figure 5.1: *Size table of the dataset before and after pre-processing*

1. **Without normalization :**

   The table 5.1 Result table of the machine learning on facebook dataset without normalization :

Model	Accuracy	Precision	Recall	F1-score	Entropy	Train Time	Test Time
Decision Tree	0.976	0.977	0.976	0.976	0.436	0.020	0.004
Random Forest	**0.992**	**0.992**	**0.992**	**0.992**	0.398	0.343	0.014
Gradient Boosting	0.987	0.987	0.987	0.987	0.408	0.622	0.004
K-Nearest Neighbors (k=5)	**0.992**	**0.992**	**0.992**	**0.992**	0.408	0.033	**0.067**
Gaussian Naive Bayes	0.660	0.843	0.660	0.711	**0.678**	0.026	0.010
K-Means Clustering (k=2)	0.166	0.877	0.166	0.079	0.103	0.733	0.023
Gaussian Mixture Models	0.150	0.877	0.150	0.047	0.033	**1.248**	0.045
Support Vector Machine (linear)	0.989	0.989	0.989	0.989	0.403	0.027	0.009

Table 5.1: Facebook dataset without normalization

Among the algorithms in the table 5.1, the Decision Tree model achieves the highest accuracy of 0.976, along with the same value for precision, recall, and F1-score, indicating excellent overall performance. Random Forest, Gradient Boosting, K-Nearest Neighbors (k=5), and Support Vector Machine (linear) also exhibit high accuracy scores, ranging from 0.987 to 0.992.

However, models like Gaussian Naive Bayes, K-Means Clustering (k=2), and Gaussian Mixture Models perform poorly, with accuracy scores ranging from 0.150 to 0.660. These models also have lower precision, recall, and F1-scores compared to the top-performing models.

Interestingly, the training time varies significantly across models, with Gaussian Mixture Models taking the longest time of 1.248, while Decision Tree and Gaussian Naive Bayes have the shortest training times of 0.020 and 0.026, respectively.

2. **With normalization :**

   The table 5.2 Result table of the machine learning on facebook dataset with normalization :

	Accuracy	Precision	Recall	F1-score	Entropy	Train Time	Test Time
Decision Tree	0.971	0.972	0.971	0.971	0.427	0.055	0.005
Random Forest	**0.986**	**0.986**	**0.986**	**0.986**	0.422	0.685	0.023
Gradient Boosting	**0.986**	0.986	**0.986**	0.985	0.401	1.080	0.004
K-Nearest Neighbors (k=5)	0.943	0.946	0.943	0.944	0.446	0.008	**0.066**
Gaussian Naive Bayes	0.626	0.815	0.626	0.681	**0.681**	0.011	0.005
K-Means Clustering (k=2)	0.129	0.400	0.129	0.193	0.583	0.808	0.023
Gaussian Mixture Models	0.302	0.609	0.302	0.383	0.664	**1.408**	0.038
Support Vector Machine (linear)	0.945	0.948	0.945	0.946	0.441	0.044	0.014

Table 5.2: Facebook dataset with normalization

The table shows the performance of various machine learning models on the Facebook dataset with normalization applied. Let me analyze the results: the Decision Tree and Random Forest models perform the best, with accuracy scores of 0.971 and 0.986, respectively. They also have high precision, recall, and F1-scores, indicating excellent overall performance.

Gradient Boosting also performs well, with an accuracy of 0.986, precision and recall of 0.986, and an F1-score of 0.985. These tree-based ensemble models seem to benefit from normalization.

K-Nearest Neighbors (k=5) and Support Vector Machine (linear) have good accuracy scores of 0.943 and 0.945, respectively, but slightly lower precision, recall, and F1-scores compared to the top models.

Gaussian Naive Bayes, K-Means Clustering (k=2), and Gaussian Mixture Models show relatively poor performance, with accuracy scores ranging from 0.129 to 0.626.

Interestingly, normalization seems to have improved the performance of most models compared to the previous table without normalization, except for a slight decrease in the Decision Tree model's accuracy.

The training times vary considerably, with Gradient Boosting and Gaussian Mixture Models taking the longest at 1.080 and 1.408, respectively, while K-Nearest Neighbors and Gaussian Naive Bayes have the shortest training times.

3. **With selected features:**

   The table 5.3 Result table of the machine learning on facebook dataset with selected features

	Model	Accuracy	Precision	Recall	F1-score	Entropy	Train Time	Test Time
0	Decision Tree	0.981	0.981	0.981	0.981	0.418	0.037	0.005
1	Random Forest	0.987	0.987	0.987	0.986	0.388	0.328	0.013
2	Gradient Boosting	0.987	0.987	0.987	0.987	0.408	1.388	0.008
3	K-Nearest Neighbors (k=5)	0.992	0.992	0.992	0.992	0.408	0.016	0.074
4	Gaussian Naive Bayes	0.636	0.839	0.636	0.690	**0.686**	0.029	0.010
5	K-Means Clustering (k=2)	0.834	0.729	0.834	0.778	0.103	0.619	0.011
6	Gaussian Mixture Models	0.853	0.732	0.853	0.788	0.019	**6.674**	**0.183**
7	Support Vector Machine (linear)	**0.995**	**0.995**	**0.995**	**0.995**	0.413	0.062	0.018

Table 5.3: Facebook dataset with selected features

The table presents the performance of various machine learning models on the Facebook dataset with selected features. Here's an analysis of the results:

The Support Vector Machine (linear) model achieves the highest accuracy of 0.995, along with the same values for precision, recall, and F1-score, indicating excellent overall performance.

K-Nearest Neighbors (k=5), Gradient Boosting, and Random Forest also perform remarkably well, with accuracy scores of 0.992, 0.987, and 0.987, respectively. These models also have high precision, recall, and F1-scores.

The Decision Tree model has an accuracy of 0.981, which is still quite good, but slightly lower than the top-performing models.

Gaussian Mixture Models and K-Means Clustering (k=2) exhibit moderate performance, with accuracy scores of 0.853 and 0.834, respectively, while their precision and recall values vary.

Gaussian Naive Bayes continues to perform poorly, with an accuracy of 0.636, although its precision is relatively high at 0.839.

Interestingly, the training times vary significantly across models, with Gaussian Mixture Models taking the longest time of 6.674, while Decision Tree and K-Nearest Neighbors have the shortest training times of 0.037 and 0.016, respectively.

- **How we selected the features**

The SelectKBest is employed after preprocessing to select the top k features that are most relevant to the target variable (CLASS). The k value is set to 10 in this case, which means SelectKBest will retain the 10 features that are most strongly related to the target based on their F-statistic scores. These scores measure the degree of linear dependency between each feature and the target variable.

4. **List of Confusion Matrix**

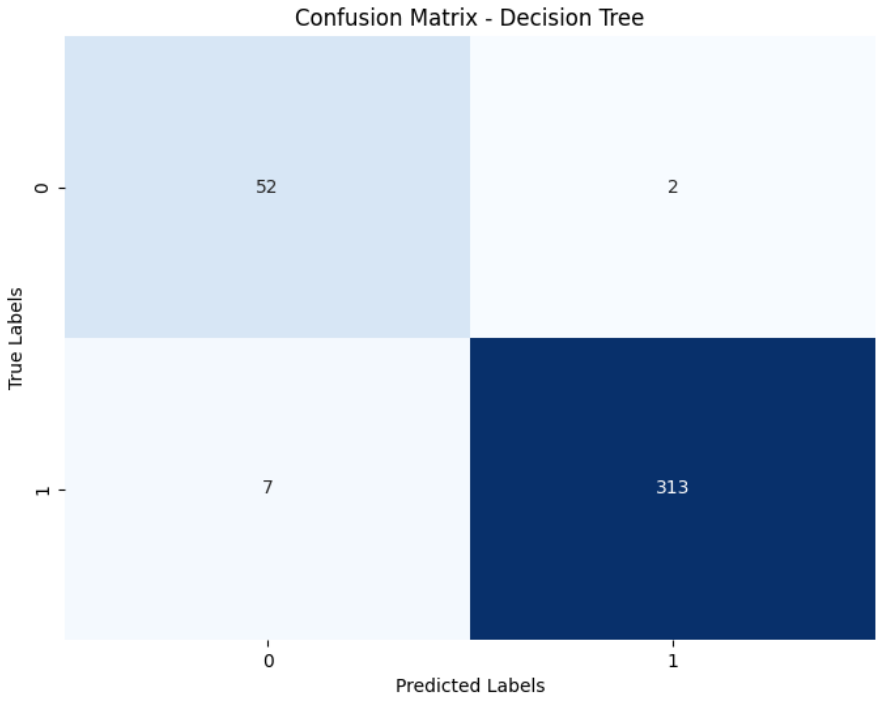

Figure 5.2: *Confusion Matrix Decision Tree*

The confusion matrix image displays the following values:

52 : This value represents the number of instances where the true label was 0 (True labels row), but the decision tree model predicted the label as 1 (Predicted Labels column).

2 : This value indicates the number of instances where both the true label and the predicted label by the model were 0.

7 : This value shows the number of instances where the true label was 1, but the model predicted the label as 0.

313 : This value represents the number of instances where both the true label and the predicted label by the model were 1.

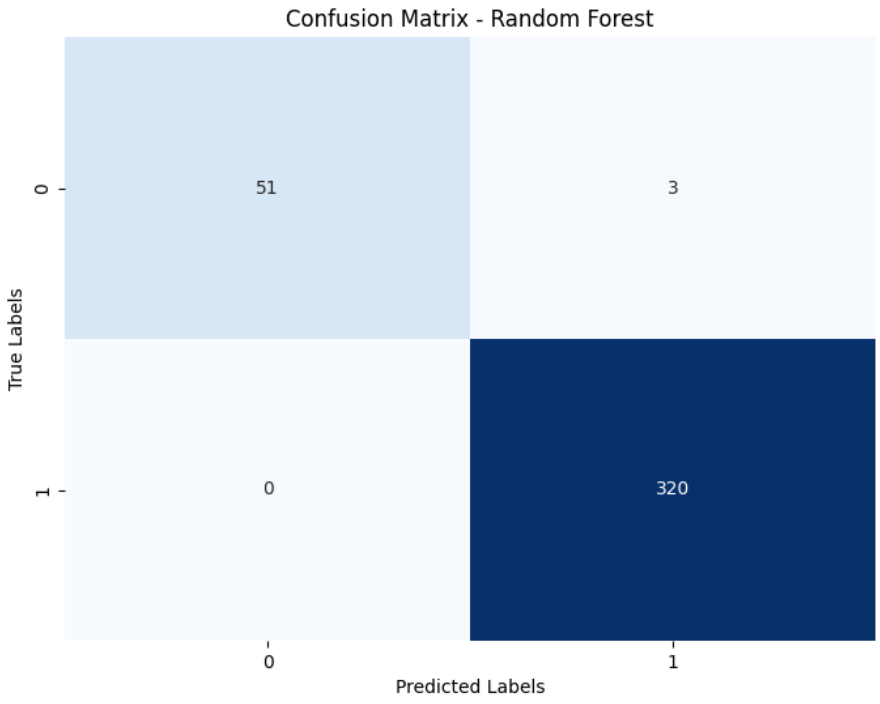

Figure 5.3: *Confusion Matrix Random Forest*

The confusion matrix displays the following values for the Random Forest model:

51 : This value represents the number of instances where the true label was 0, but the model predicted the label as 1.

3 : This value indicates the number of instances where the true label was 0, and the model also predicted the label as 0.

0 : This value shows that there were no instances where the true label was 1, but the model predicted the label as 0.

320 : This value represents the number of instances where both the true label and the predicted label by the model were 1.

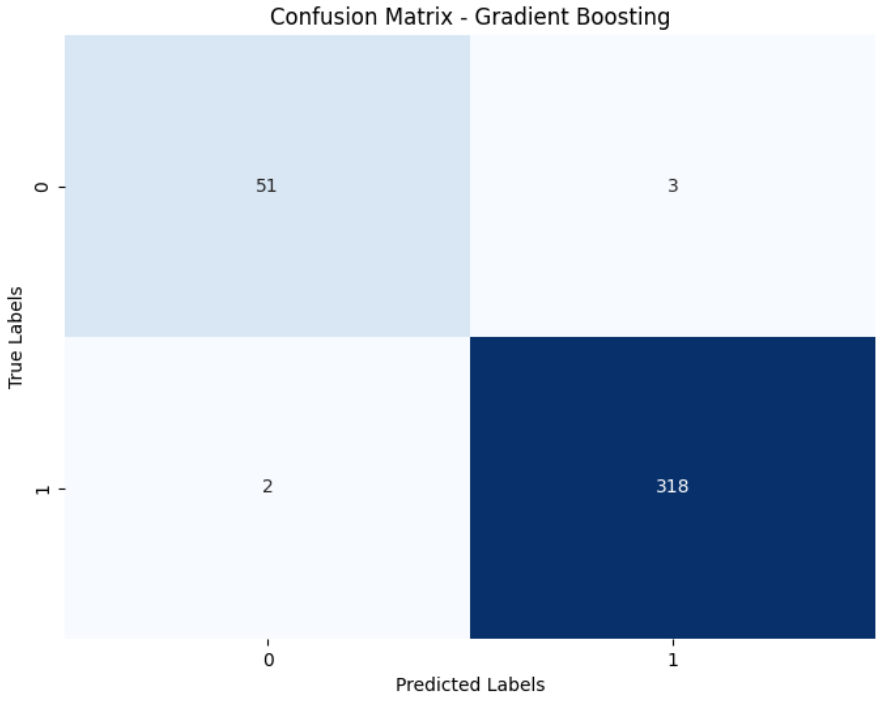

Figure 5.4: *Confusion Matrix Gradient Boosting*

51 : This value represents the number of instances where the true label was 0, but the model predicted the label as 1.

3 : This value indicates the number of instances where the true label was 0, and the model also predicted the label as 0.

2 : This value shows the number of instances where the true label was 1, but the model predicted the label as 0.

318 : This value represents the number of instances where both the true label and the predicted label by the model were 1.

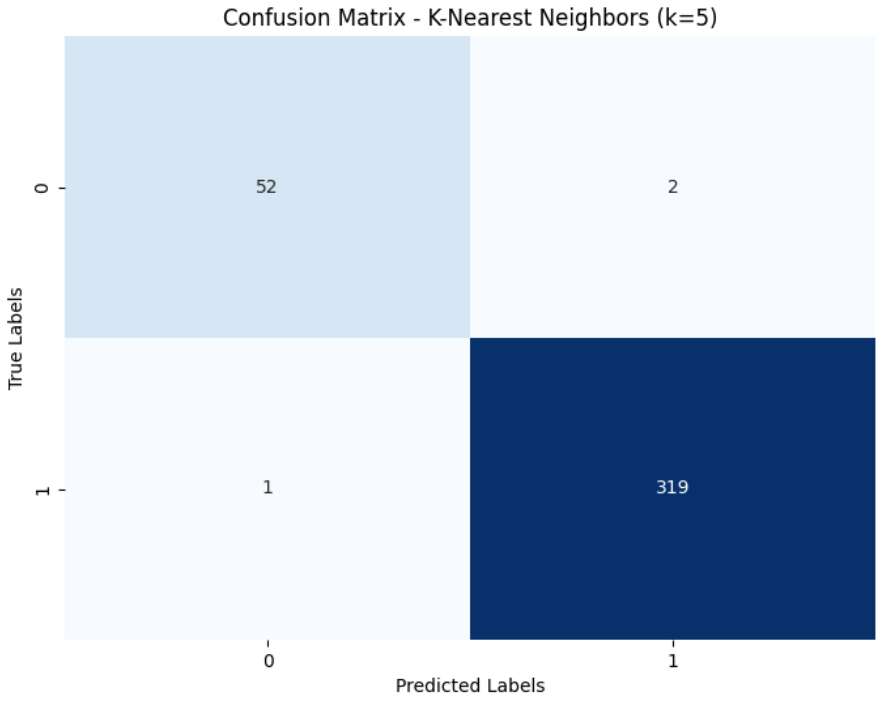

Figure 5.5: *Confusion Matrix K-Nearest Neighbors*

52 : This value represents the number of instances where the true label was 0, but the model predicted the label as 1.

2 : This value indicates the number of instances where the true label was 0, and the model also predicted the label as 0.

1 : This value shows the number of instances where the true label was 1, but the model predicted the label as 0.

319 : This value represents the number of instances where both the true label and the predicted label by the model were 1.

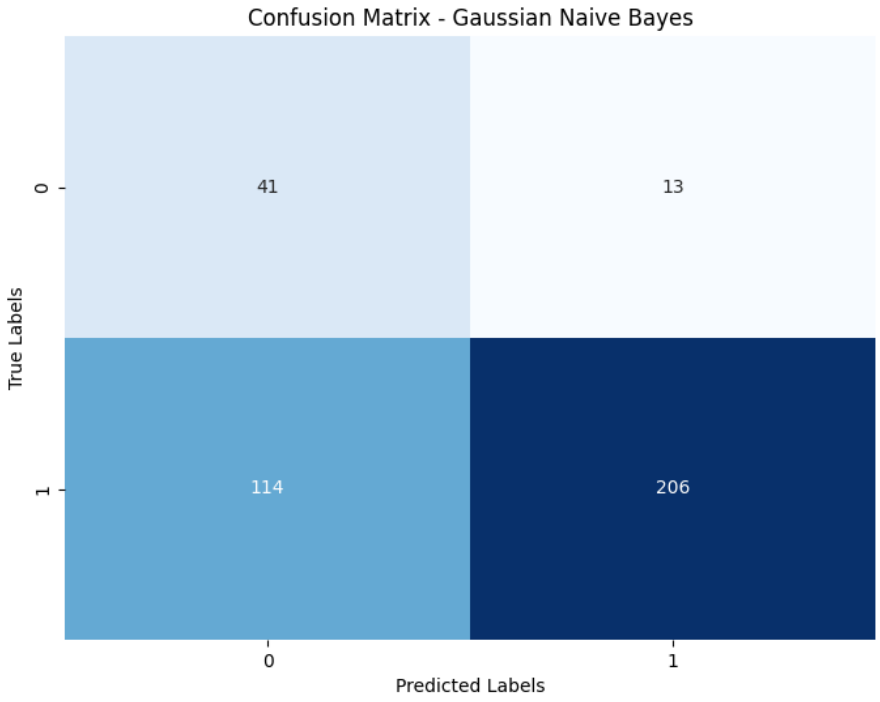

Figure 5.6: *Confusion Matrix Gaussian Naive Bayes*

41 - This value represents the number of instances where the true label was 0, but the model predicted the label as 1.

13 - This value indicates the number of instances where the true label was 0, and the model also predicted the label as 0.

114 - This value shows the number of instances where the true label was 1, but the model predicted the label as 0.

206 - This value represents the number of instances where both the true label and the predicted label by the model were 1.

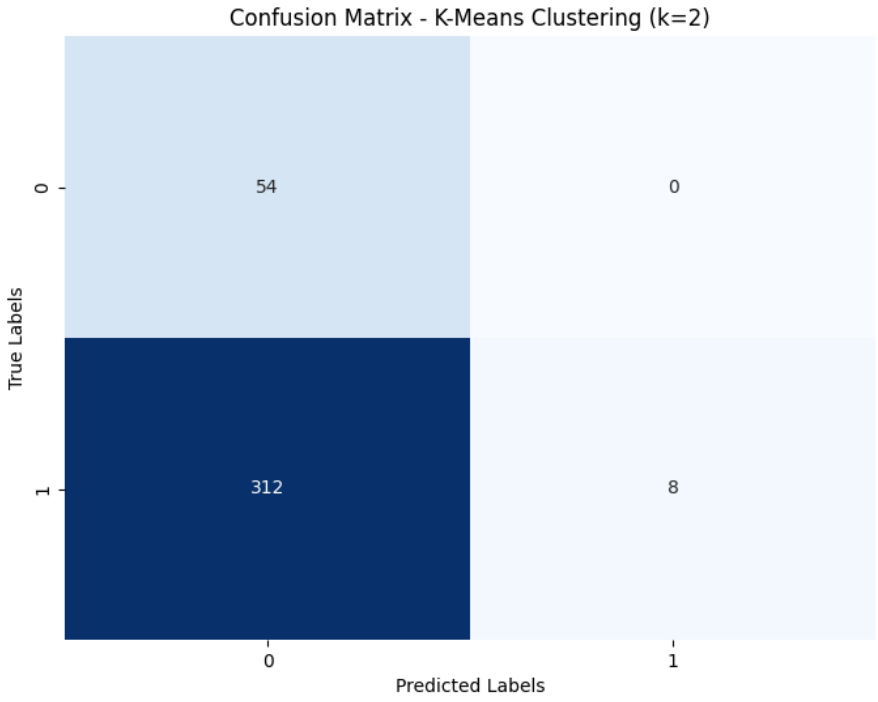

Figure 5.7: *Confusion Matrix K-Means Clustering (k=2)*

54 - This value represents the number of instances where the true label was 0, but the model predicted the label as 1 (false positives).

312 - This value indicates the number of instances where the true label was 0, and the model also predicted the label as 0 (true negatives).

8 - This value shows the number of instances where the true label was 1, but the model predicted the label as 0 (false negatives).

0 - This value represents the number of instances where both the true label and the predicted label by the model were 1 (true positives).

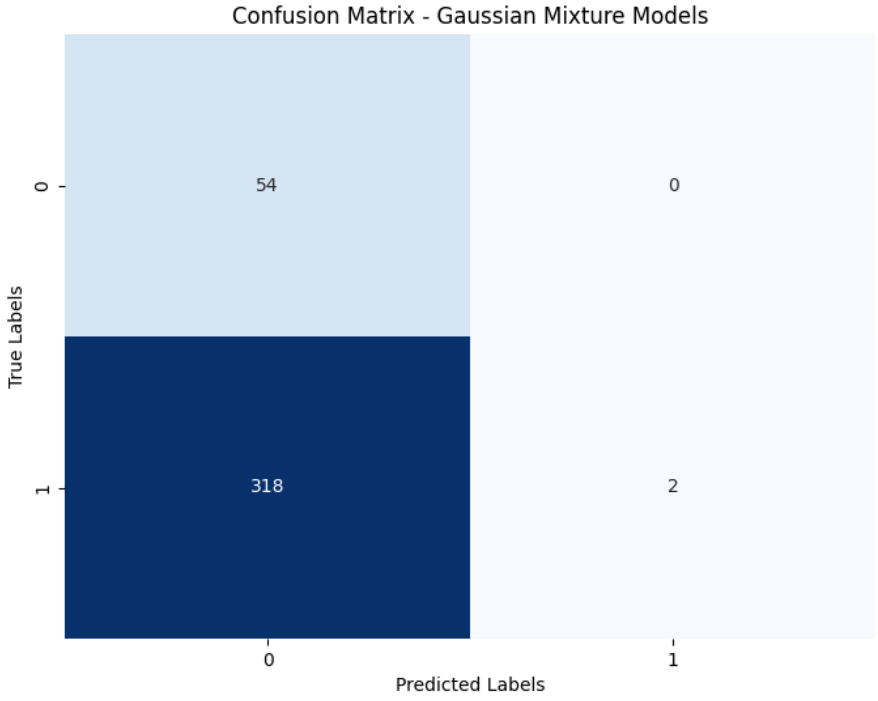

Figure 5.8: *Confusion Matrix Gaussian Mixture Models*

54 - This value represents the number of instances where the true label was 0, but the model predicted the label as 1 (false positives).

318 - This value indicates the number of instances where the true label was 0, and the model also predicted the label as 0 (true negatives).

2 - This value shows the number of instances where the true label was 1, but the model predicted the label as 0 (false negatives).

0 - This value represents the number of instances where both the true label and the predicted label by the model were 1 (true positives).

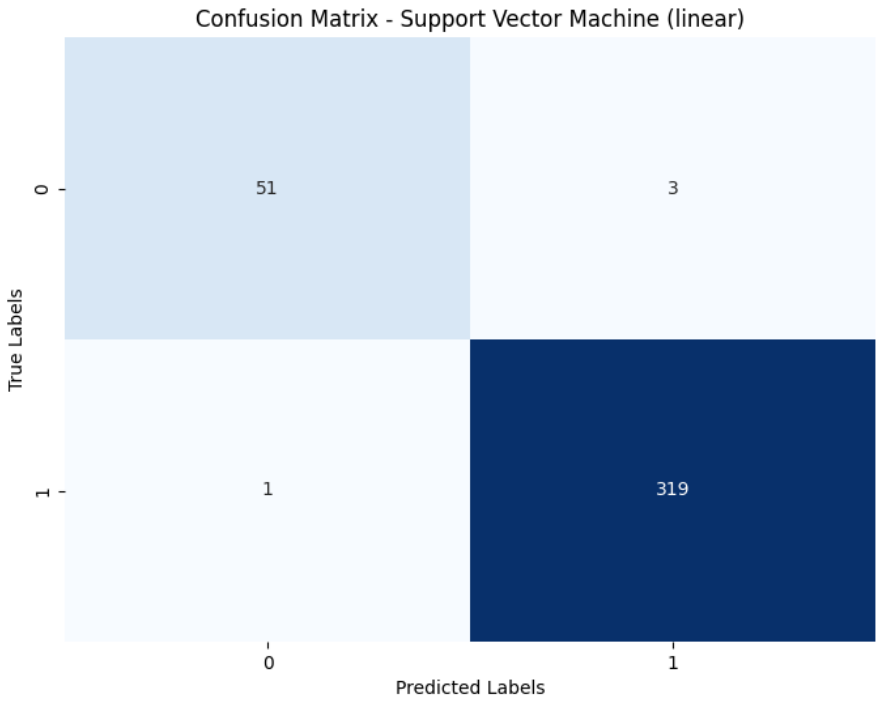

Figure 5.9: *Confusion Matrix Support Vector Machine (linear)*

51 - This value represents the number of instances where the true label was 0, but the model predicted the label as 1 (false positives).

319 - This value indicates the number of instances where the true label was 0, and the model also predicted the label as 0 (true negatives).

1 - This value shows the number of instances where the true label was 1, but the model predicted the label as 0 (false negatives).

3 - This value represents the number of instances where both the true label and the predicted label by the model were 1 (true positives).

Correlation Matrix

In the figure 5.9, the correlation matrix between the features of Facebook dataset is shown.

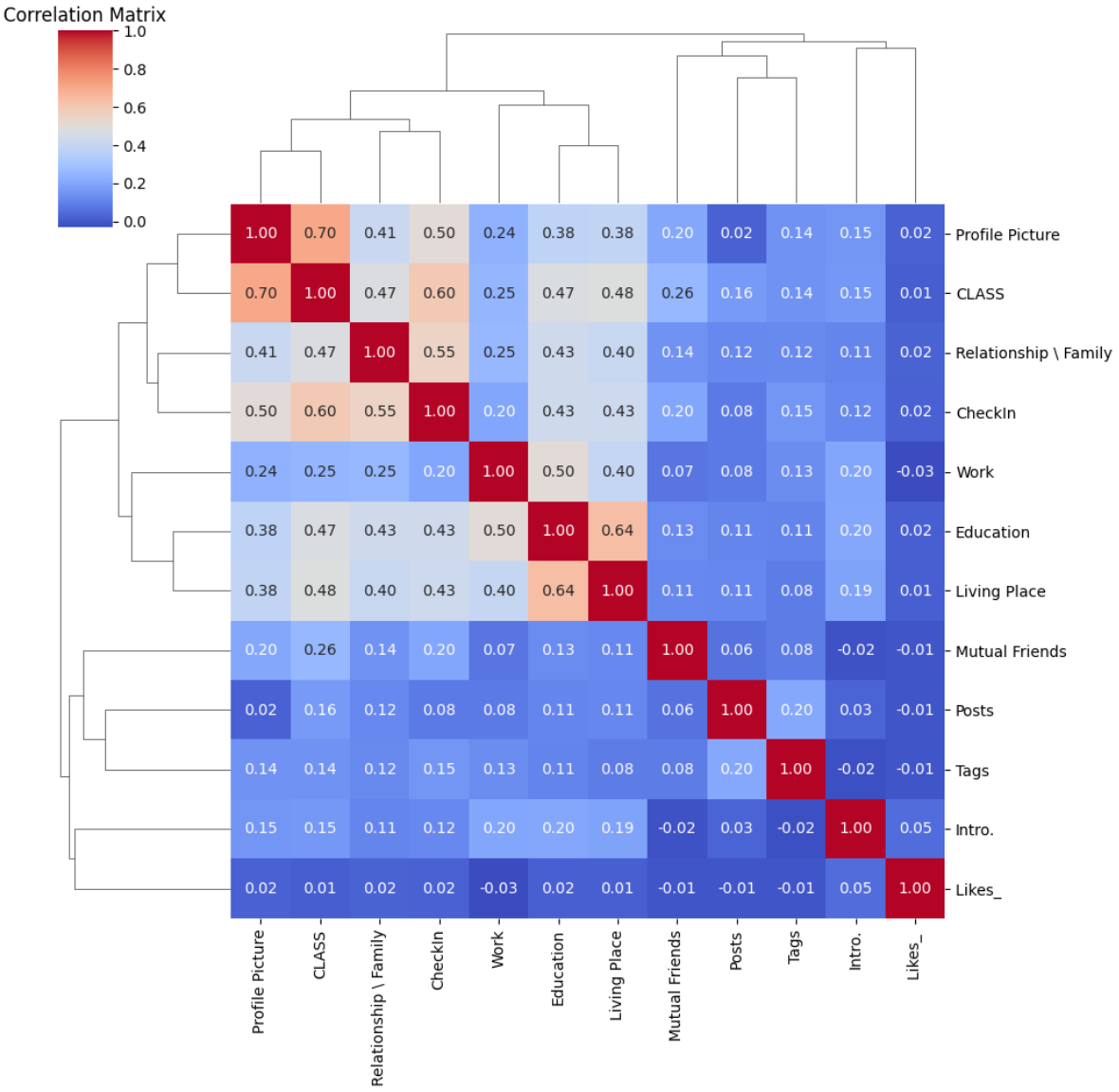

Figure 5.10: *Correlation heatMap of Facebook dataset*

From the matrix in Figure 5.9, we can observe several interesting patterns. The variable ”Profile Picture” has a strong positive correlation with ”CLASS” (0.70) and a moderate correlation with ”Relationship \ Family” (0.41), suggesting that users who share profile pictures tend to belong to similar classes or have family

relationships. On the other hand, the variable "Work" has a slightly negative correlation (-0.03) with "Education," implying that there may be an inverse relationship between these variables in the data set. The variable "Mutual Friends" shows a weak positive correlation with "Living Place" (0.11) and a weak negative correlation with "Posts" (-0.01), indicating that the number of mutual friends may have a slight influence on these variables.

5.1.2 Twitter dataset

Dataset Size

The dataset size in this context was determined using Python's sys.getsizeof() function, which calculates the memory footprint of objects. Initially, upon loading the dataset from a CSV file, sys.getsizeof(data) was used to measure the size of the DataFrame object containing the data. This provided the initial size of the dataset in bytes. After preprocessing steps such as feature scaling and encoding categorical variables, the sizes of the processed training and testing datasets (X_train_preprocessed and X_test_preprocessed) were also measured using sys.getsizeof() and summed to determine the final size of the dataset post-preprocessing.

Data Type	Size (bytes)
Initial dataset	144,144
Final Preprocessed dataset	9,784,256

Figure 5.11: *Size table of the dataset before and after pre-processing*

i. **Without normalization :**

	Accuracy	Precision	Recall	F1-score	Entropy	Train Time	Test Time
Decision Tree	0.887	0.887	0.887	0.887	**0.693**	0.051	0.001
Random Forest	**0.897**	**0.897**	**0.897**	**0.897**	**0.693**	0.736	0.032
Gradient Boosting	0.893	0.893	0.893	0.893	**0.693**	**1.340**	0.002
K-Nearest Neighbors (k=5)	0.863	0.865	0.863	0.863	0.691	0.001	**0.081**
Gaussian Naive Bayes	0.533	0.569	0.533	0.464	0.405	0.010	0.004
K-Means Clustering (k=2)	0.397	0.396	0.397	0.396	0.691	0.723	0.002
Gaussian Mixture Models	0.393	0.393	0.393	0.393	0.691	**2.436**	0.069
Support Vector Machine	0.840	0.840	0.840	0.840	**0.693**	0.197	0.050

Table 5.4: Twitter dataset without normalization performance

Accuracy: Random Forest and Gradient Boosting have the highest accuracy of 0.897 and 0.893 respectively, followed by Decision Tree with 0.887

accuracy. K-Nearest Neighbors (k=5) also performs reasonably well with 0.863 accuracy. Gaussian Naive Bayes, K-Means Clustering, and Gaussian Mixture Models have relatively low accuracy scores.

Precision and Recall: For most algorithms, the precision and recall values are the same as the accuracy scores, indicating balanced performance across classes. However, Gaussian Naive Bayes has a slightly higher precision (0.569) than recall (0.533), suggesting it may have a bias towards one class.

F1-score: The F1-score, which combines precision and recall, follows a similar pattern to accuracy. Random Forest and Gradient Boosting have the highest F1-scores of 0.897, while Gaussian Naive Bayes has the lowest at 0.464.

Entropy: All algorithms, except Gaussian Naive Bayes, have an entropy value of around 0.693, which is close to the theoretical maximum entropy for a binary classification problem.

Train Time: Gradient Boosting has the longest training time of 1.340, followed by Gaussian Mixture Models at 2.436. Decision Tree, K-Nearest Neighbors, Gaussian Naive Bayes, and Support Vector Machine have relatively short training times.

Test Time: K-Nearest Neighbors has the longest test time of 0.081, while Decision Tree and Gradient Boosting have the shortest test times of 0.001 and 0.002 respectively.

ii. **With normalization :**

	Accuracy	Precision	Recall	F1-score	Entropy	Train Time	Test Time
Decision Tree	0.887	0.887	0.887	0.887	**0.693**	0.047	0.001
Random Forest	**0.900**	**0.900**	**0.900**	**0.900**	**0.693**	0.392	0.014
Gradient Boosting	**0.900**	**0.900**	**0.900**	**0.900**	**0.693**	1.251	0.002
K-Nearest Neighbors (k=5)	0.843	0.845	0.843	0.843	0.691	0.001	0.061
Gaussian Naive Bayes	0.533	0.569	0.533	0.464	0.405	0.010	0.003
K-Means Clustering (k=2)	0.397	0.396	0.397	0.396	0.691	0.862	0.002
Gaussian Mixture Models	0.393	0.393	0.393	0.393	0.691	**2.228**	**0.076**
Support Vector Machine	0.840	0.840	0.840	0.840	**0.693**	0.201	0.050

Table 5.5: Twitter dataset without normalization performance

Accuracy: Random Forest and Gradient Boosting achieve the highest accuracy of 0.900, followed by Decision Tree with 0.887 accuracy. K-Nearest Neighbors (k=5) and Support Vector Machine also perform reasonably well with accuracy's of 0.843 and 0.840, respectively. Gaussian Naive Bayes, K-Means Clustering, and Gaussian Mixture Models have lower accuracy scores.

Precision and Recall: For most algorithms, the precision and recall values match their accuracy scores, indicating balanced performance across classes. However, Gaussian Naive Bayes has a higher precision (0.569) than recall (0.533), suggesting a potential bias toward one class.

F1-score: The F1-score, which combines precision and recall, follows a similar trend as accuracy. Random Forest and Gradient Boosting have the highest F1-scores of 0.900, while Gaussian Naive Bayes has the lowest at 0.464.

Entropy: All algorithms, except Gaussian Naive Bayes (0.405) and K-Nearest Neighbors (0.691), have an entropy value close to 0.693, which is near the theoretical maximum entropy for a binary classification problem.

Train Time: Gradient Boosting has the longest training time of 1.251, followed by Gaussian Mixture Models at 2.228. K-Means Clustering also has a relatively long training time of 0.862. Decision Tree, K-Nearest Neighbors, Gaussian Naive Bayes, and Support Vector Machine have shorter training times.

Test Time: K-Nearest Neighbors has the longest test time of 0.061, while Decision Tree and Gradient Boosting have the shortest test times of 0.001 and 0.002, respectively.

iii. **With selected features :**

	Accuracy	Precision	Recall	F1-score	Entropy	Train Time	Test Time
Decision Tree	0.867	0.868	0.867	0.867	0.692	0.004	0.000
Random Forest	**0.883**	**0.883**	**0.883**	**0.883**	0.693	**0.497**	**0.023**
Gradient Boosting	0.873	0.873	0.873	0.873	**0.693**	0.324	0.002
K-Nearest Neighbors (k=5)	0.847	0.853	0.847	0.846	0.684	0.001	0.017
Gaussian Naive Bayes	0.670	0.784	0.670	0.633	0.476	0.001	0.000
K-Means Clustering (k=2)	0.393	0.393	0.393	0.393	0.691	0.022	0.000
Gaussian Mixture Models	0.390	0.233	0.390	0.285	0.360	0.026	0.001
Support Vector Machine	0.840	0.843	0.840	0.840	0.689	0.027	0.002

Table 5.6: Twitter dataset with selected features performance

Accuracy: Random Forest has the highest accuracy of 0.883, followed by Decision Tree (0.867), Gradient Boosting (0.873), K-Nearest Neighbors (0.847), and Support Vector Machine (0.840). Gaussian Naive Bayes (0.670), K-Means Clustering (0.393), and Gaussian Mixture Models (0.390) have relatively lower accuracy scores.

Precision: Gaussian Naive Bayes has the highest precision of 0.784, followed by K-Nearest Neighbors (0.853), Support Vector Machine (0.843), Random Forest (0.883), and Decision Tree (0.868). Gaussian Mixture Models has a low precision of 0.233.

Recall: The recall values are the same as the accuracy scores for most algorithms, except for Gaussian Naive Bayes, which has a lower recall (0.670) than precision (0.784).

F1-score: Random Forest has the highest F1-score of 0.883, followed by Decision Tree (0.867), Gradient Boosting (0.873), K-Nearest Neighbors (0.846), and Support Vector Machine (0.840). Gaussian Naive Bayes has an F1-score of 0.633, and Gaussian Mixture Models has the lowest F1-score of 0.285.

Entropy: Gaussian Mixture Models has the lowest entropy of 0.360, while Gaussian Naive Bayes has an entropy of 0.476. The other algorithms have

entropy values ranging from 0.684 (K-Nearest Neighbors) to 0.693 (Random Forest and Gradient Boosting).

Train Time: Gradient Boosting has the longest training time of 0.324, followed by Random Forest (0.497), Support Vector Machine (0.027), K-Means Clustering (0.022), and Gaussian Mixture Models (0.026). Decision Tree, K-Nearest Neighbors, and Gaussian Naive Bayes have very short training times.

Test Time: Random Forest has the longest test time of 0.023, followed by K-Nearest Neighbors (0.017), Support Vector Machine (0.002), Gradient Boosting (0.002), and Gaussian Mixture Models (0.001). Decision Tree and Gaussian Naive Bayes have negligible test times.

iv. • **How we selected the features**

The SelectKBest transformer was applied after the preprocessing step to select the top k features based on their F-statistic scores. The `f_classif` scoring function is suitable for classification tasks and measures the correlation between each feature and the target variable.

(a) **List of Confusion Matrix**

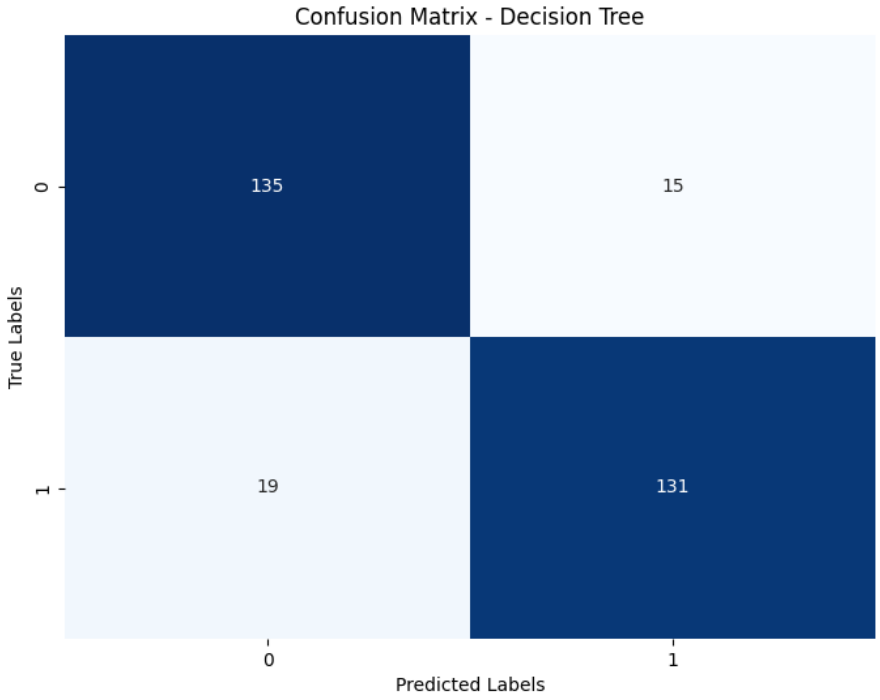

Figure 5.12: *Confusion Matrix Decision Tree*

the analysis of the confusion matrix for the Decision Tree model:

15 : This value represents the number of false positives, where the true label was 0, but the model predicted 1.

135 : This value indicates the number of true negatives, where both the true label and predicted label were 0.

19 : This value shows the number of false negatives, where the true label was 1, but the model predicted 0.

131 : This value represents the number of true positives, where both the true label and predicted label were 1.

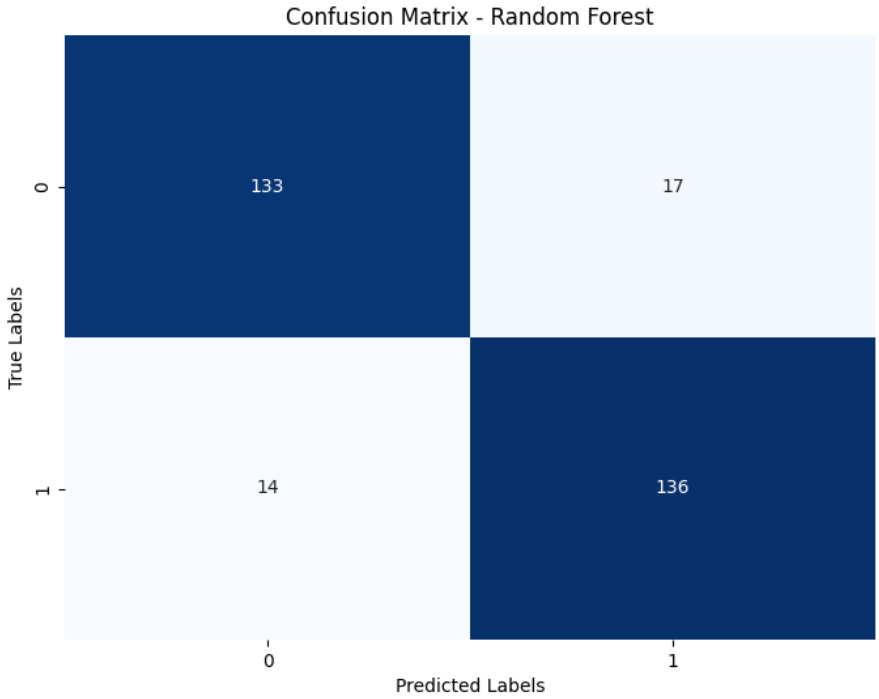

Figure 5.13: *Confusion Matrix Random Forest*

the analysis of the confusion matrix for the Random Forest model:

133 : This value represents the number of true negatives, where the true label was 0, and the model correctly predicted it as 0.

17 : This value represents the number of false positives, where the true label was 0, but the model incorrectly predicted it as 1.

14 : This value represents the number of false negatives, where the true label was 1, but the model incorrectly predicted it as 0.

136 : This value represents the number of true positives, where the true label was 1, and the model correctly predicted it as 1.

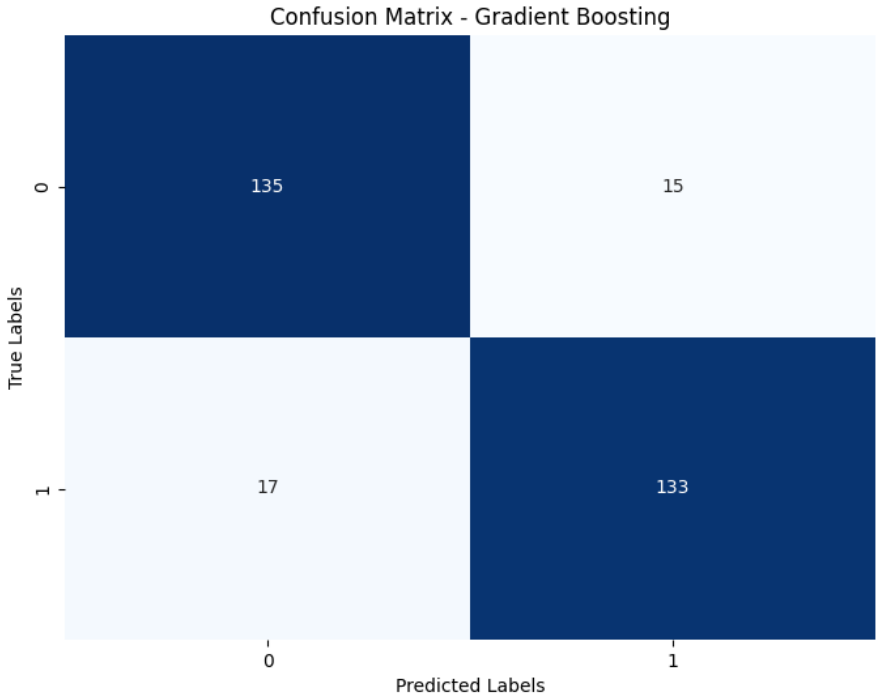

Figure 5.14: *Confusion Matrix Gradient Boosting*

the analysis of the confusion matrix for the Gradient Boosting model:

135 : This value represents the number of true negatives, where the true label was 0, and the model correctly predicted it as 0.

15 : This value represents the number of false positives, where the true label was 0, but the model incorrectly predicted it as 1.

17 : This value represents the number of false negatives, where the true label was 1, but the model incorrectly predicted it as 0.

133 : This value represents the number of true positives, where the true label was 1, and the model correctly predicted it as 1.

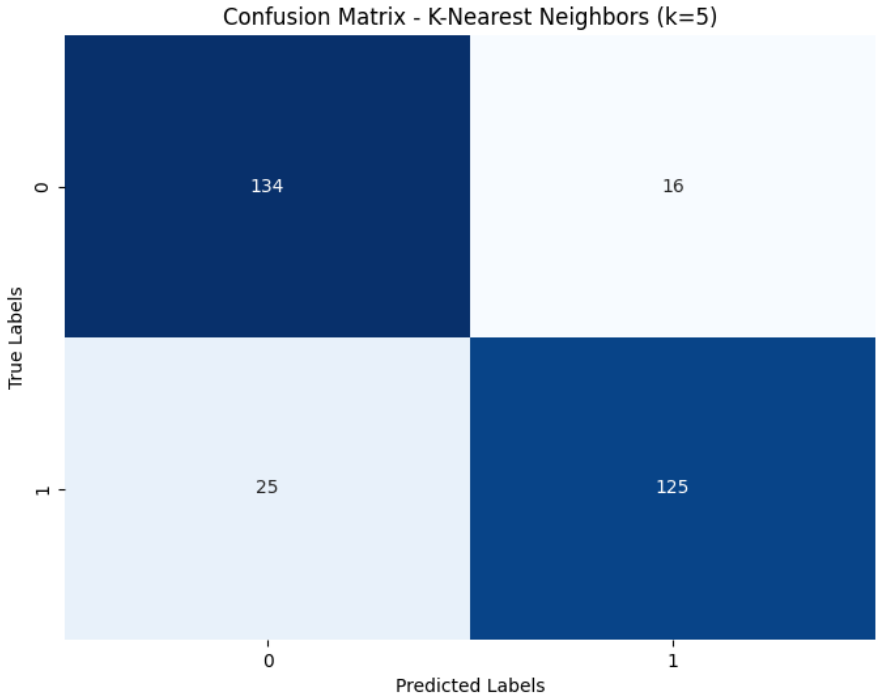

Figure 5.15: *Confusion Matrix K-Nearest Neighbors (k=5)*

the analysis of the confusion matrix for the K-Nearest Neighbors (k=5) model:

134 : This value represents the number of true negatives, where the true label was 0, and the model correctly predicted it as 0.

16 : This value represents the number of false positives, where the true label was 0, but the model incorrectly predicted it as 1.

25 : This value represents the number of false negatives, where the true label was 1, but the model incorrectly predicted it as 0.

125 : This value represents the number of true positives, where the true label was 1, and the model correctly predicted it as 1.

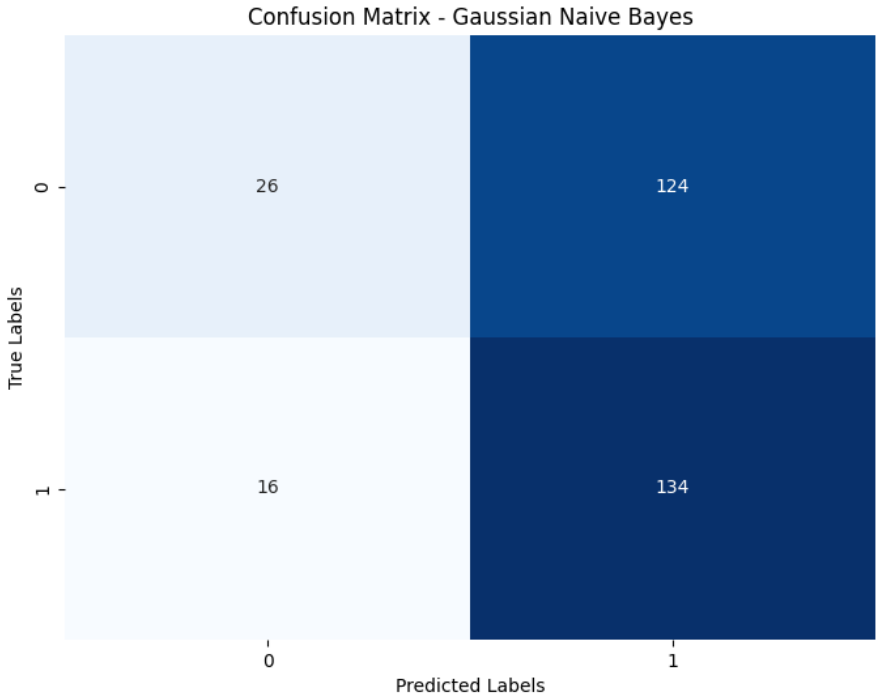

Figure 5.16: *Confusion Matrix Gaussian Naive Bayes*

the analysis of the confusion matrix for the Gaussian Naive Bayes model:

26 : This value represents the number of false positives, where the true label was 0, but the model incorrectly predicted it as 1.

124 : This value represents the number of true negatives, where the true label was 0, and the model correctly predicted it as 0.

16 : This value represents the number of false negatives, where the true label was 1, but the model incorrectly predicted it as 0.

134 : This value represents the number of true positives, where the true label was 1, and the model correctly predicted it as 1.

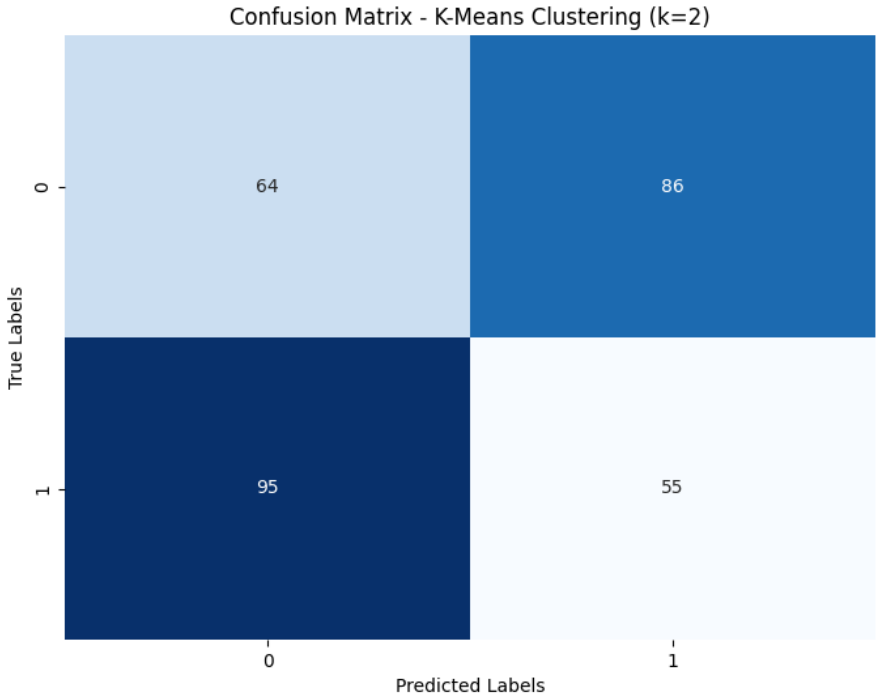

Figure 5.17: *Confusion Matrix K-Means Clustering (k=2)*

The values in the matrix can be interpreted as follows:

64 : This value represents the number of instances that were correctly classified as belonging to cluster 0 (true negatives).

95 : This value represents the number of instances that were misclassified as belonging to cluster 0, when they should have been assigned to cluster 1 (false negatives).

55 : This value represents the number of instances that were misclassified as belonging to cluster 1, when they should have been assigned to cluster 0 (false positives).

86 : This value represents the number of instances that were correctly classified as belonging to cluster 1 (true positives).

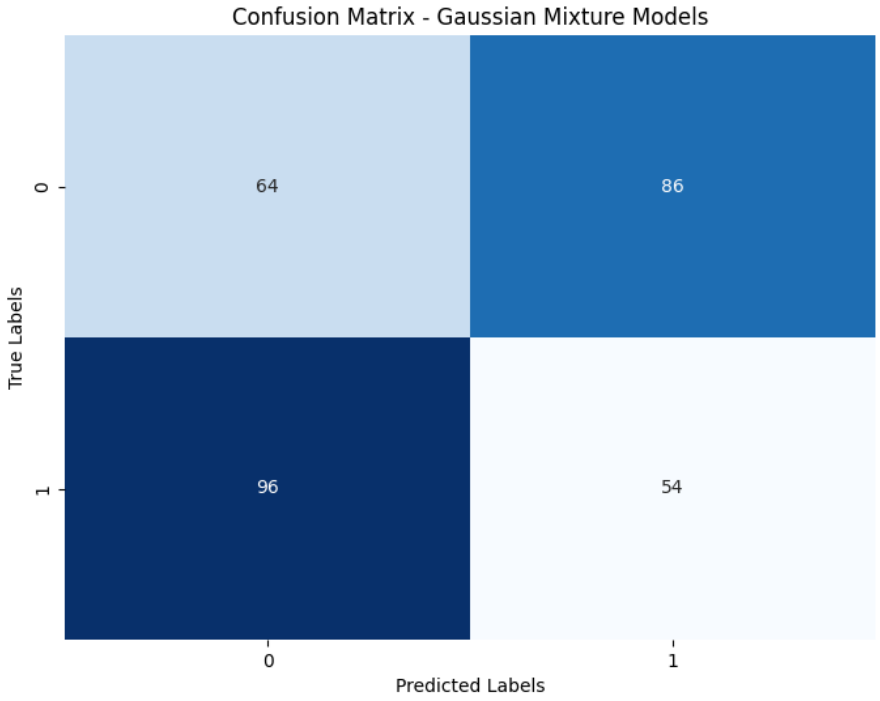

Figure 5.18: *Confusion Matrix Gaussian Mixture Models*

The values in the matrix can be interpreted as follows:

64 : This value represents the number of instances that were correctly classified as belonging to class 0 (true negatives).

96 : This value represents the number of instances that were misclassified as belonging to class 0, when they should have been assigned to class 1 (false negatives).

54 : This value represents the number of instances that were misclassified as belonging to class 1, when they should have been assigned to class 0 (false positives).

86 : This value represents the number of instances that were correctly classified as belonging to class 1 (true positives).

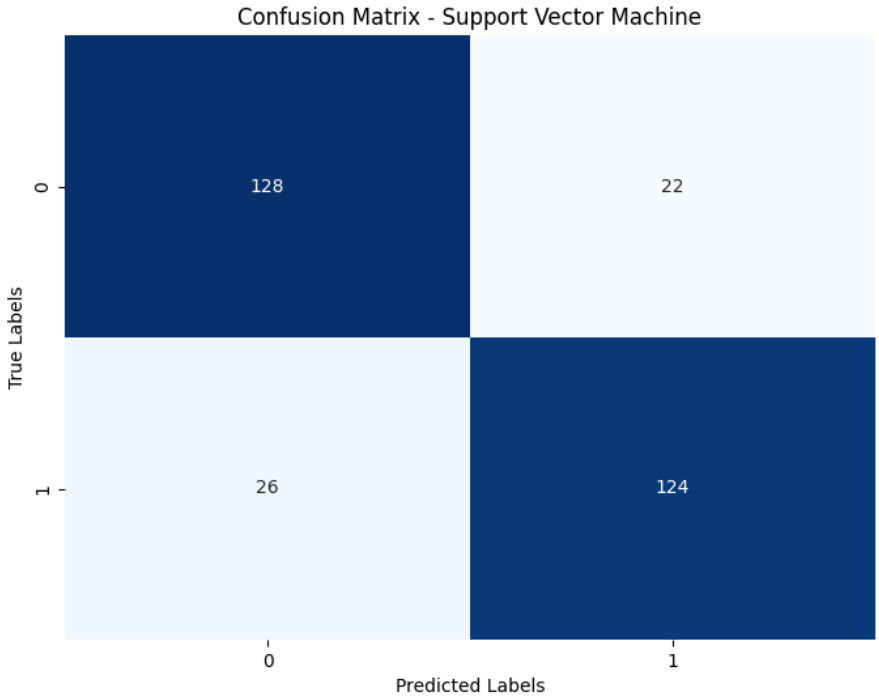

Figure 5.19: *Confusion Matrix Support Vector Machine*

The values in the matrix can be interpreted as follows:

128 : This value represents the number of instances that were correctly classified as belonging to the negative class (true negatives).

26 : This value represents the number of instances that were misclassified as belonging to the negative class, when they should have been assigned to the positive class (false negatives).

22 : This value represents the number of instances that were misclassified as belonging to the positive class, when they should have been assigned to the negative class (false positives).

124 : This value represents the number of instances that were correctly classified as belonging to the positive class (true positives).

Correlation Matrix

In the figure 5.18, the correlation matrix between the features of Twitter dataset is shown.

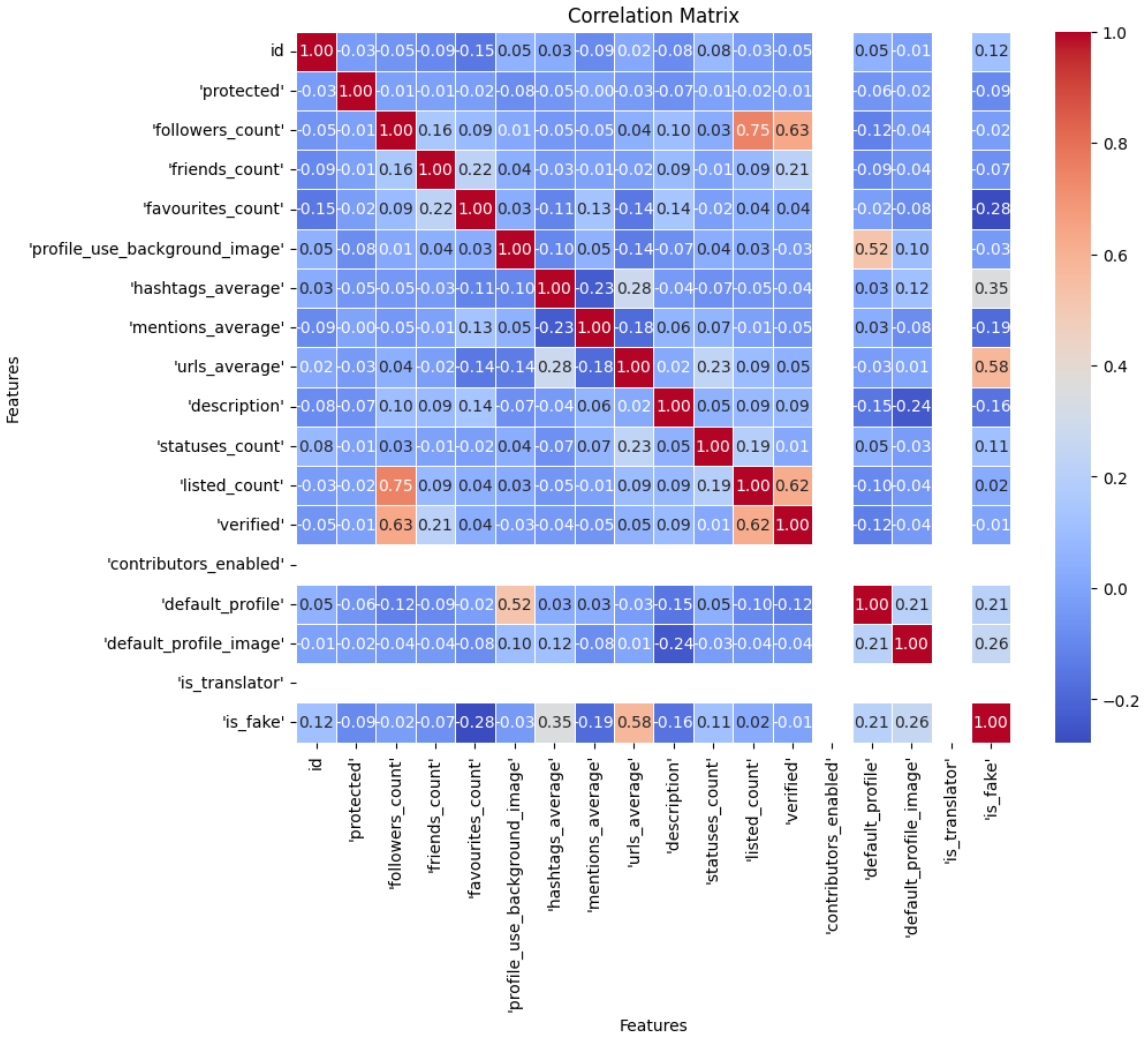

Figure 5.20: *Correlation heatMap of Twitter dataset*

The image shows a correlation matrix that displays the pairwise correlation coefficients between various features or variables. A correlation matrix is a useful tool for understanding the relationships between different features in a dataset.

The correlation coefficients range from -1 to 1, where -1 indicates a perfect negative correlation, 0 indicates no correlation, and 1 indicates a perfect positive

correlation. The diagonal elements of the matrix are all 1, as these represent the perfect correlation of each feature with itself.

Looking at the matrix, we can observe some interesting patterns:

i. The feature "id" has very low or near-zero correlation with most other features, except for a moderate positive correlation with "protected" (0.3).

ii. The features "followers_count," "friends_count," "favourites_count," and "statuses_count" exhibit moderate to strong positive correlations with each other, indicating that users with more followers tend to have more friends, favorites, and tweets.

iii. The feature "profile_use_background_image" has a moderate positive correlation with "hashtags_average" (0.11) and "mentions_average" (0.13), suggesting that users who use background images tend to include more hashtags and mentions in their tweets.

iv. The feature "urls_average" has a moderate positive correlation with "description" (0.14), indicating that users who include more URLs in their tweets tend to have longer descriptions.

v. The feature "verified" has a moderate positive correlation with "listed_count" (0.19), suggesting that verified users tend to be included in more lists.

vi. The feature "default_profile" has a moderate positive correlation with "default_profile_image" (0.24), as expected, since users who have a default profile are likely to have a default profile image as well.

vii. The feature "is_fake" has moderate positive correlations with "urls_average" (0.19) and "contributors_enabled" (0.16), which could potentially be indicators of fake or spam accounts.

5.1.3 Instagram dataset

Dataset Size

dataset sizes are evaluated using sys.getsizeof() to measure the memory footprint at different stages: before and after preprocessing. Initially, the sizes of pandas DataFrames (train_data and test_data), loaded from CSV files, are determined to understand the memory consumption of raw data. These initial sizes are logged and printed.

After applying preprocessing steps (handled by a pipeline involving ColumnTransformer and SimpleImputer), the script transforms the data into numpy arrays (X_train_preprocessed and X_test_preprocessed). The sizes of these transformed arrays are then calculated using sys.getsizeof() to assess the impact of preprocessing on memory usage. Final sizes are logged and printed to provide insights into how data transformation affects memory footprint.

Dataset	Initial Size (bytes)	Final Preprocessed Size (bytes)
Training dataset	55,440	336,512
Testing dataset	11,664	70,208

Figure 5.21: *Size table of the dataset before and after pre-processing*

i. **Without normalization :**

	Accuracy	Precision	F1-score	Recall	Entropy	Training Time	Testing Time	Execution Time
Decision Tree	0.875	0.875	0.875	0.875	**0.693**	0.003	0.000	0.003
Random Forest	0.900	0.900	0.900	0.900	0.693	0.179	0.006	0.185
Gradient Boosting	**0.933**	**0.935**	**0.933**	**0.933**	0.691	0.191	0.001	0.192
K-Nearest Neighbors (k=5)	0.858	0.861	0.858	0.858	0.690	0.001	**0.010**	0.011
Gaussian Naive Bayes	0.692	0.761	0.670	0.692	0.553	0.002	0.000	0.002
Support Vector Machine (RBF kernel, C=1)	0.875	0.878	0.875	0.875	0.690	0.011	0.003	0.015
K-Means Clustering	0.608	0.703	0.557	0.608	0.437	**0.504**	0.001	**0.504**
Gaussian Mixture Models	0.808	0.810	0.808	0.808	0.690	0.171	0.002	0.173

Table 5.7: Instagram dataset without normalization performance

5.7 Table shows a table comparing the performance metrics of various machine learning models on a classification task. The metrics included are Accuracy, Precision, F1-score, Recall, Entropy, Training Time, Testing Time, and Execution Time. Here's an analysis of the results:

1. Gradient Boosting has the highest Accuracy (0.933), Precision (0.935), F1-score (0.933), and Recall (0.933), indicating excellent overall performance in correctly classifying the instances.

2. Random Forest also performs remarkably well, with an Accuracy, Precision, F1-score, and Recall of 0.900, slightly lower than Gradient Boosting but still very good.

3. Decision Tree, Support Vector Machine (RBF kernel, C=1), and Gaussian Mixture Models have comparable Accuracy, Precision, F1-score, and Recall values, ranging from 0.808 to 0.878, suggesting good but not outstanding performance.

4. K-Nearest Neighbors (k=5) and Gaussian Naive Bayes perform relatively poorly compared to the other models, with Accuracy scores of 0.858 and 0.692, respectively.

5. K-Means Clustering has the lowest Accuracy (0.608) and F1-score (0.557) among all models, indicating its unsuitability for this particular classification task.

6. In terms of Entropy, which measures the uncertainty or impurity of the predictions, all models except K-Means Clustering have relatively low values, with Gradient Boosting having the lowest Entropy (0.691).

7. Regarding training and testing times, K-Means Clustering and Gradient Boosting have the highest Training Times (0.504 and 0.191, respectively), while Random Forest has the highest Testing Time (0.006).

8. Execution Times, which combine Training and Testing Times, are generally consistent with the individual Training and Testing Times, with K-Means Clustering (0.504) and Gradient Boosting (0.192) having the highest Execution Times.

Overall, based on the performance metrics, Gradient Boosting and Random

Forest appear to be the most suitable models for this classification task, offering high accuracy and F1-scores while maintaining reasonable training and execution times.

A. **With normalization :**

	Accuracy	Precision	F1-score	Recall	Entropy	Training Time	Testing Time	Execution Time
Decision Tree	0.875	0.875	0.875	0.875	**0.693**	0.012	0.004	0.016
Random Forest	0.900	0.900	0.900	0.900	**0.693**	0.745	0.012	0.757
Gradient Boosting	**0.908**	**0.909**	**0.908**	**0.908**	0.692	**0.807**	0.001	**0.808**
K-Nearest Neighbors (k=5)	0.867	0.868	0.867	0.867	0.691	0.008	**0.085**	0.092
Gaussian Naive Bayes	0.692	0.761	0.670	0.692	0.553	0.003	0.001	0.003
Support Vector Machine (RBF kernel, C=1)	0.867	0.868	0.867	0.867	0.691	0.023	0.006	0.029
K-Means Clustering	0.742	0.830	0.723	0.742	0.553	0.110	0.001	0.110
Gaussian Mixture Models	0.742	0.830	0.723	0.742	0.553	0.043	0.002	0.045

Table 5.8: Instagram dataset with normalization performance

The table 5.8 compares the performance of various machine learning algorithms across different evaluation metrics. Gradient Boosting emerges as the top performer, boasting the highest accuracy of 0.908 and precision of 0.909, closely followed by Random Forest with equal accuracy and precision scores of 0.900. Interestingly, K-Nearest Neighbors and Support Vector Machine share identical values for accuracy, precision, F1-score, and recall at 0.867. Gaussian Naive Bayes lags behind with the lowest accuracy of 0.692 and precision of 0.761. While Gradient Boosting excels in most metrics, it comes with the trade-off of having the highest training time of 0.807. K-Means Clustering and Gaussian Mixture Models exhibit comparable performance, suggesting potential ensemble techniques. The table provides a comprehensive overview, enabling informed algorithm selection based on the specific requirements of the problem at hand.

B. **With selected features :**

Model	Accuracy	Precision	F1-score	Recall	Entropy	Training Time	Testing Time	Execution Time
Decision Tree	0.833	0.834	0.833	0.833	**0.693**	0.003	0.000	0.003
Random Forest	0.833	0.834	0.833	0.833	**0.693**	**0.291**	**0.013**	**0.304**
Gradient Boosting	0.858	0.858	0.858	0.858	**0.693**	0.178	0.001	0.179
K-Nearest Neighbors (k=5)	**0.867**	**0.867**	**0.867**	**0.867**	**0.693**	0.002	**0.013**	0.015
Gaussian Naive Bayes	0.758	0.803	0.749	0.758	0.618	0.001	0.000	0.002
Support Vector Machine (RBF kernel, C=1)	0.833	0.834	0.833	0.833	**0.693**	0.006	0.002	0.008
K-Means Clustering	0.400	0.222	0.286	0.400	0.325	0.011	0.000	0.011
Gaussian Mixture Models	0.417	0.361	0.352	0.417	0.476	0.012	0.000	0.013

Table 5.9: Instagram dataset with selected features performance

The table 5.9 presents a comprehensive comparison of various machine learning models, evaluating their performance across multiple metrics. Among the models evaluated, K-Nearest Neighbors (k=5) emerges as the top performer in terms of accuracy, precision, F1-score, and recall, achieving impressive scores of 0.867 for each metric. Gradient Boosting also exhibits strong results, with accuracy, precision, F1-score, and recall values of 0.858, closely trailing K-Nearest Neighbors. Interestingly, Decision Tree, Random Forest, and Support Vector Machine (RBF kernel, C=1) share identical scores of 0.833 across these four metrics, indicating their comparable performance. However, Gaussian Naive Bayes lags behind with a relatively lower accuracy of 0.758, although its precision (0.803) outperforms its recall (0.758). On the other hand, K-Means Clustering and Gaussian Mixture Models demonstrate the lowest accuracy, precision, and F1-score among the evaluated models. The table also provides insights into the computational requirements, with Random Forest and Gradient Boosting exhibiting longer training, testing, and execution times compared to other models.

- **How we selected the features**

  We implement Recursive Feature Elimination (RFE) using `LogisticRegression` as the estimator to select the top 10 most relevant features for training machine learning models. RFE iteratively assesses feature importance and eliminates less informative ones, enhancing model performance by focusing on the subset of features that contribute most to accurate predictions. This approach optimizes computational efficiency, improves model interpretability, and maintains high predictive accuracy, making it suitable for reducing dataset dimensionality while preserving model effectiveness.

C. **List of Confusion Matrix**

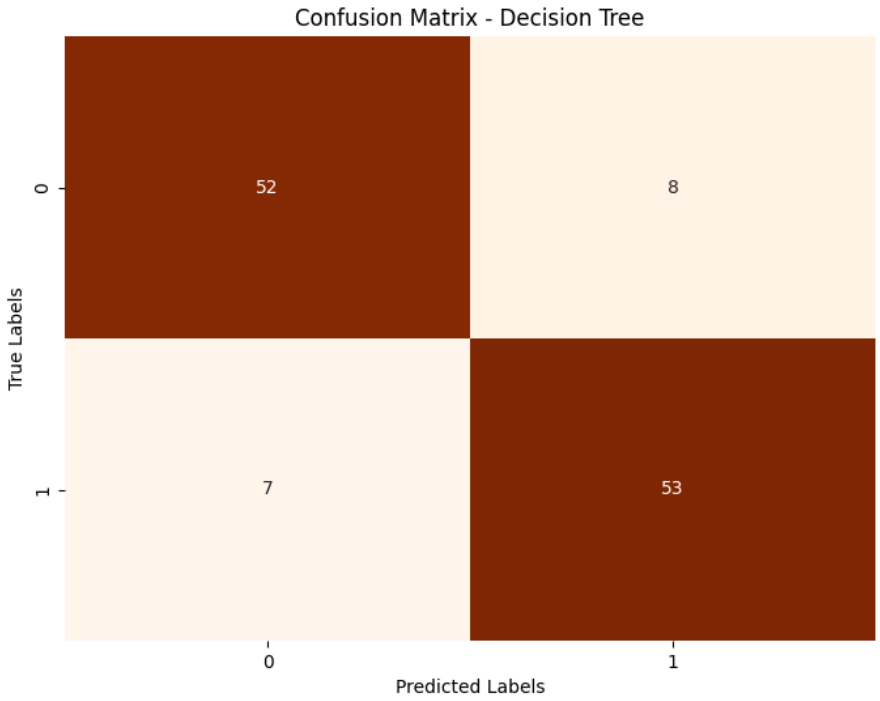

Figure 5.22: *Confusion Matrix Decision Tree*

The values in the matrix are:

- True Positives (top-right): 53 This represents the number of instances correctly predicted as class 1.

- True Negatives (bottom-left): 52 This represents the number of instances correctly predicted as class 0.
- False Positives (bottom-right): 7 This represents the number of instances incorrectly predicted as class 1, when they actually belong to class 0.
- False Negatives (top-left): 8 This represents the number of instances incorrectly predicted as class 0, when they actually belong to class 1.

Based on these values, the Decision Tree model seems to perform reasonably well, with a high number of correct predictions for both classes. However, there are still some misclassifications, with 8 instances of class 1 being incorrectly classified as class 0, and 7 instances of class 0 being incorrectly classified as class 1.

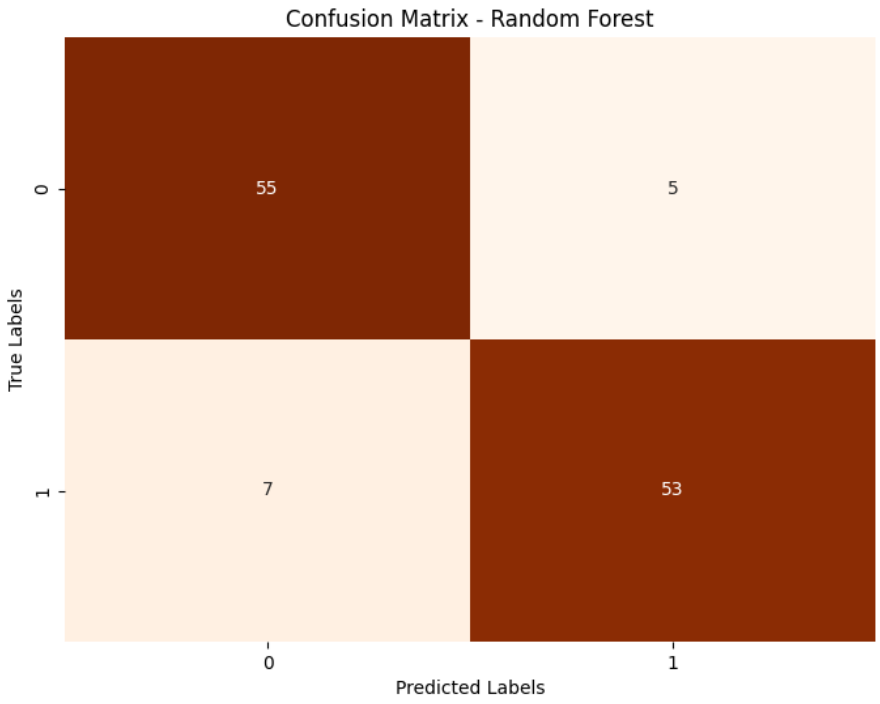

Figure 5.23: *Confusion Matrix Random Forest*

The confusion matrix values are as follows:

- True Positives (top-right): 53 This represents the number of instances

correctly predicted as class 1.

- True Negatives (bottom-left): 55 This represents the number of instances correctly predicted as class 0.
- False Positives (bottom-right): 7 This represents the number of instances incorrectly predicted as class 1, when they actually belong to class 0.
- False Negatives (top-left): 5 This represents the number of instances incorrectly predicted as class 0, when they actually belong to class 1.

Compared to the Decision Tree model, the Random Forest model seems to perform slightly better in terms of correctly classifying instances of class 1, with fewer false negatives (5 instead of 8). However, it has the same number of false positives (7) as the Decision Tree model.
The true negatives for the Random Forest model (55) are also slightly higher than the Decision Tree model (52), indicating better performance in correctly identifying instances of class 0.

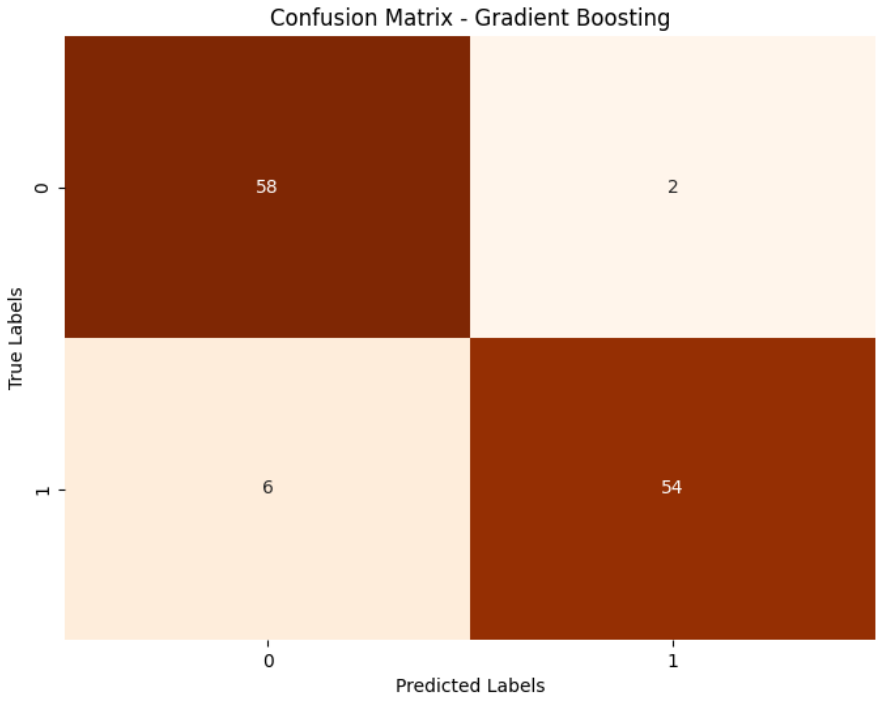

Figure 5.24: *Confusion Matrix Gradient Boosting*

The confusion matrix values are:

- True Positives (top-right): 54 This represents the number of instances correctly predicted as class 1.
- True Negatives (bottom-left): 58 This represents the number of instances correctly predicted as class 0.
- False Positives (bottom-right): 6 This represents the number of instances incorrectly predicted as class 1 when they actually belong to class 0.
- False Negatives (top-left): 2 This represents the number of instances incorrectly predicted as class 0 when they actually belong to class 1.

Compared to the previous Decision Tree and Random Forest models, the Gradient Boosting model seems to perform better in terms of accurately classifying instances of both classes. It has the highest number of true positives (54) and true negatives (58), indicating a higher overall accuracy.

Furthermore, the Gradient Boosting model has the lowest number of false negatives (2) and a relatively low number of false positives (6), suggesting that it has the best performance in minimizing misclassifications of both classes.

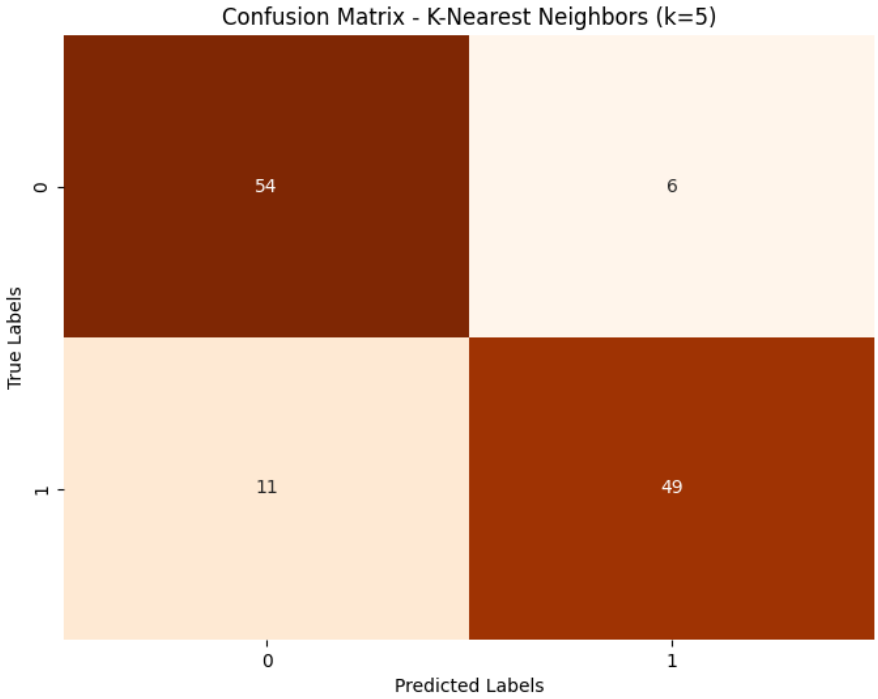

Figure 5.25: *Confusion Matrix K-Nearest Neighbors (k=5)*

The values in the confusion matrix can be interpreted as follows:
54: This value represents the number of instances that were correctly classified as belonging to the negative class (true negatives).
11: This value represents the number of instances that were misclassified as belonging to the negative class when they should have been assigned to the positive class (false negatives).
6: This value represents the number of instances that were misclassified as belonging to the positive class when they should have been assigned to the negative class (false positives).
49: This value represents the number of instances that were correctly classified as belonging to the positive class (true positives).

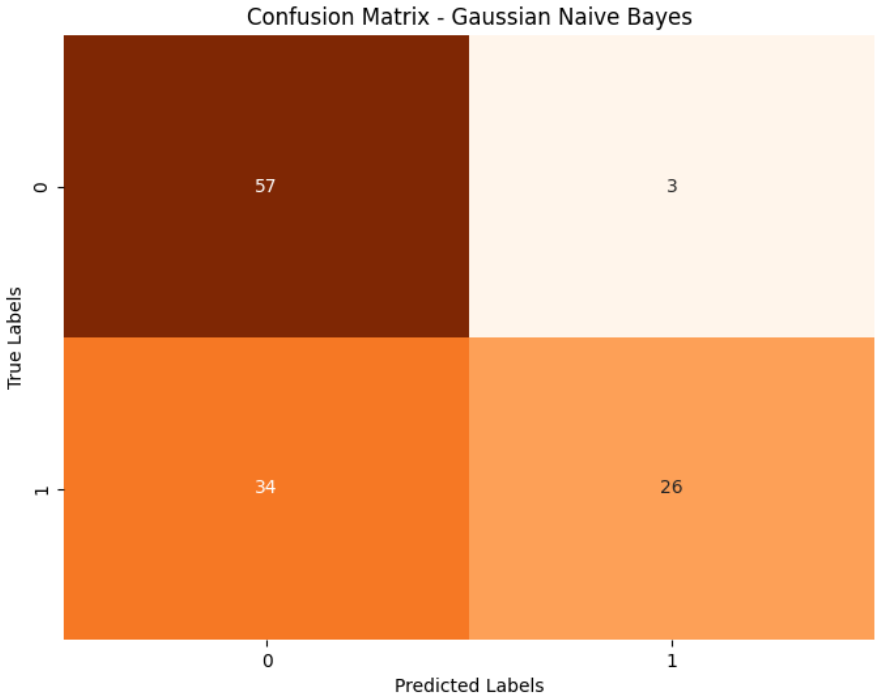

Figure 5.26: *Confusion Matrix Gaussian Naive Bayes*

The values in the confusion matrix can be interpreted as follows:
57: This value represents the number of instances that were correctly classified as belonging to the negative class (true negatives).
34: This value represents the number of instances that were misclassified as belonging to the negative class when they should have been assigned to the positive class (false negatives).
3: This value represents the number of instances that were misclassified as belonging to the positive class when they should have been assigned to the negative class (false positives).
26: This value represents the number of instances that were correctly classified as belonging to the positive class (true positives).

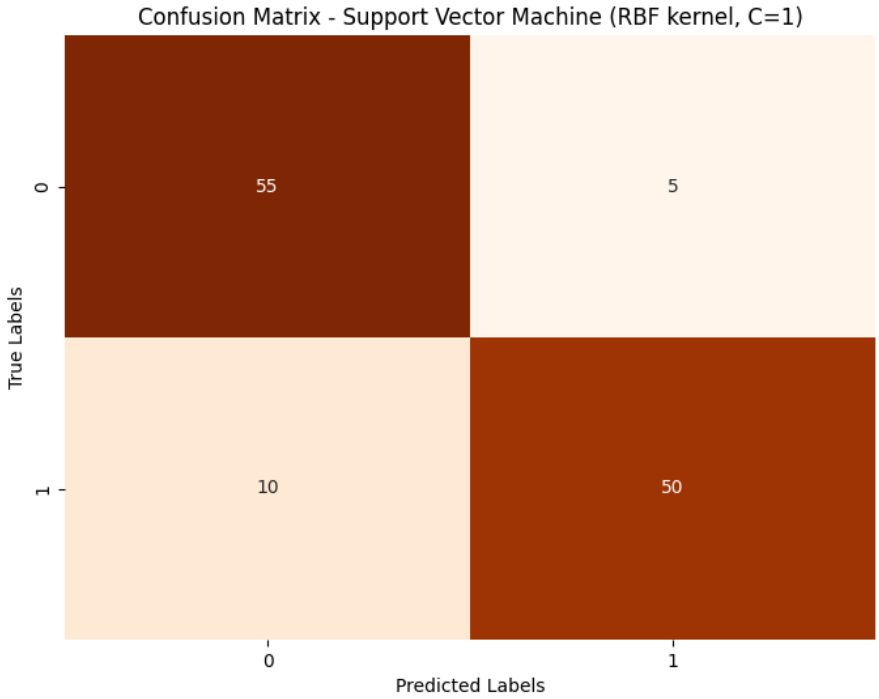

Figure 5.27: *Confusion Matrix Support Vector Machine (RBF kernel, C=1)*

The values in the confusion matrix can be interpreted as follows:
55: This value represents the number of instances that were correctly classified as belonging to the negative class (true negatives).
10: This value represents the number of instances that were misclassified as belonging to the negative class when they should have been assigned to the positive class (false negatives).
5: This value represents the number of instances that were misclassified as belonging to the positive class when they should have been assigned to the negative class (false positives).
50: This value represents the number of instances that were correctly classified as belonging to the positive class (true positives).

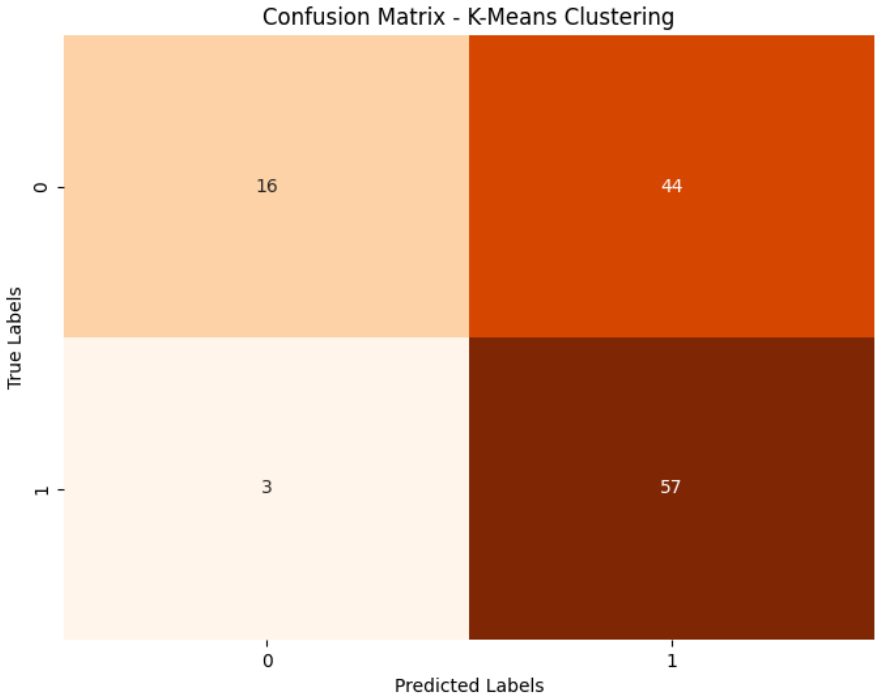

Figure 5.28: *Confusion Matrix K-Means Clustering*

The values can be interpreted as follows:
16: This value represents the number of instances that were assigned to cluster 0 when their true label was 0.
3: This value represents the number of instances that were assigned to cluster 1 when their true label was 0.
44: This value represents the number of instances that were assigned to cluster 1 when their true label was 1.
57: This value represents the number of instances that were assigned to cluster 0 when their true label was 1.

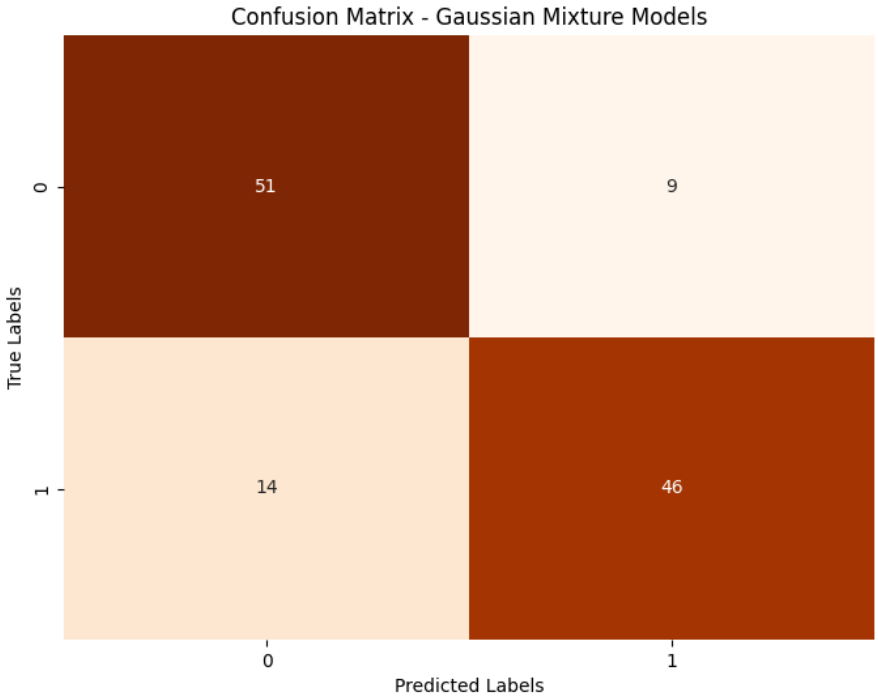

Figure 5.29: *Confusion Matrix Gaussian Mixture Model*

The values in the matrix are:
51: The number of instances correctly predicted as class 0 (true negatives). 9: The number of instances incorrectly predicted as class 1 when they actually belonged to class 0 (false positives). 14: The number of instances incorrectly predicted as class 0 when they actually belonged to class 1 (false negatives). 46: The number of instances correctly predicted as class 1 (true positives).

Correlation Matrix

In the figure 5.27, the correlation matrix between the features of Twitter dataset is shown.

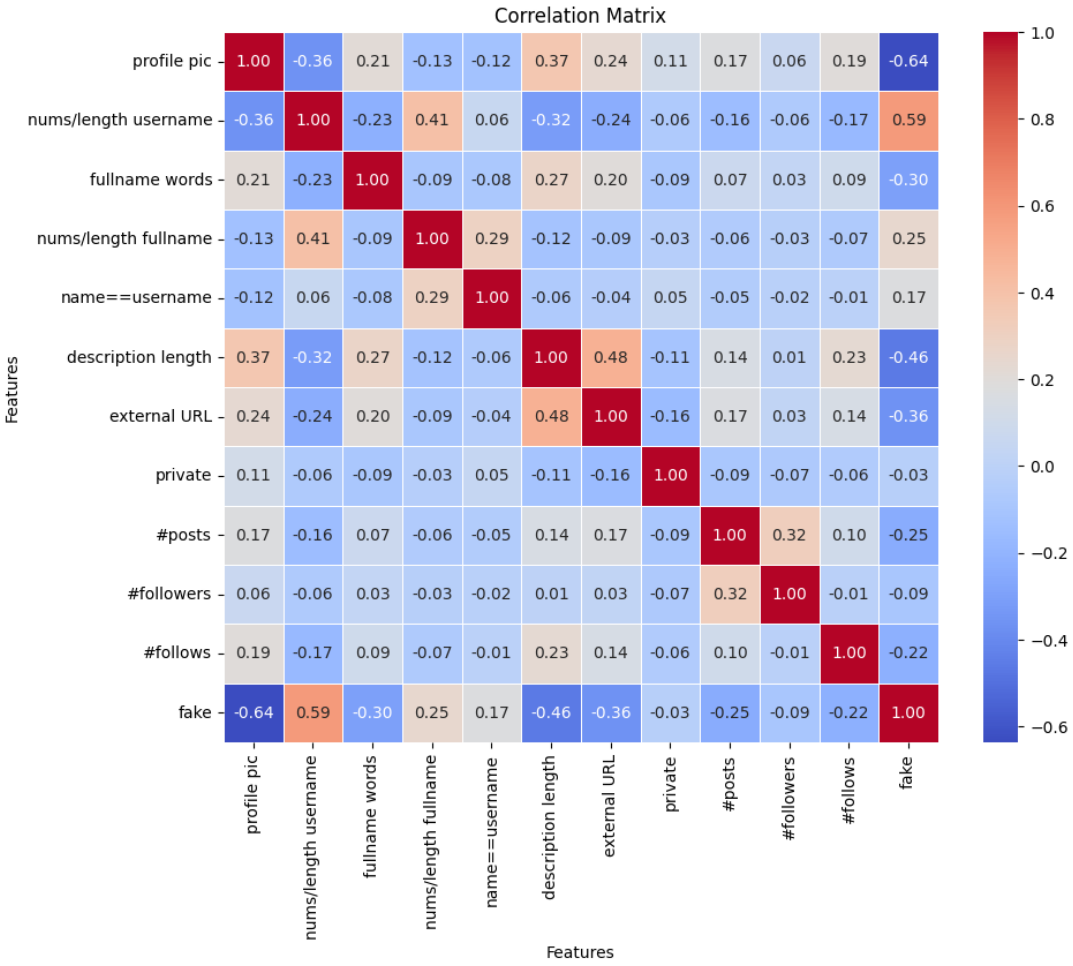

Figure 5.30: *Correlation heatMap of Train Instagram dataset*

Some key observations from from the matrix in Figure 5.27 :
-The feature ”profile_pic” has a strong positive correlation with itself (1.0) and moderate to weak positive correlations with ”description_length” (0.37), ”external_URL” (0.24), ”private” (0.11), ”#posts” (0.17), and ”#follows” (0.19).
-The feature ”nums/length_username” has a moderate negative correlation with ”profile_pic” (-0.36) and moderate positive correlations with ”nums/length_fullname” (0.41) and ”fake” (0.59).

-The feature ”fullname_words” has a weak negative correlation with ”nums/length_fullname” (-0.09) and weak positive correlations with ”description_length” (0.27) and ”external_URL” (0.20).
-The feature ”name==username” has a moderate positive correlation with ”nums/length_fullname” (0.29) and weak negative correlations with ”description_length” (-0.06) and ”external_URL” (-0.04).
-The feature ”fake” has a strong negative correlation with ”profile_pic” (-0.64) and moderate positive correlations with ”nums/length_username” (0.59) and ”nums/length_fullname” (0.25).

5.1.4 The second Instagram dataset

Dataset Size

The size of the dataset is determined by calculating the memory usage of the Pandas DataFrame containing the combined fake_data and real_data. Initially, each dataset is read from CSV files and then merged into a single DataFrame data. To prepare for model training, a binary indicator column is_real is derived based on the original 'class' labels. Afterward, the 'class' column is dropped to finalize the dataset structure for machine learning tasks. The memory footprint of this preprocessed DataFrame is computed using Python's sys.getsizeof() function, providing an approximate size in bytes. Subsequently, the dataset is split into training and testing subsets using train_test_split(), ensuring that both sets are sized appropriately for model evaluation. The memory usage of these split datasets (X_train, y_train, X_test, y_test) is individually calculated and summed to provide insights into the memory requirements at different stages of the data preparation pipeline

Data Type	Size (bytes)
Initial dataset	15,643,296
Preprocessed training set	12,166,912
Preprocessed testing set	5,214,432

Figure 5.31: *Size table of the dataset before and after pre-processing*

D. **Without normalization :**

	Model	Accuracy	Precision	Recall	F1-score	Train Time	Test Time
0	**RandomForest**	**0.958**	**0.943**	**0.953**	**0.948**	22.547	0.628
1	LogisticRegression	0.769	0.712	0.707	0.710	1.145	0.006
2	KNeighbors	0.783	0.744	0.700	0.721	0.021	15.060
3	SVM	0.729	0.752	0.483	0.588	**521.716**	**110.007**
4	DecisionTree	0.933	0.936	0.893	0.914	0.808	0.010
5	GaussianNB	0.765	0.738	0.639	0.685	0.037	0.012
6	KNNk5	0.783	0.744	0.700	0.721	0.019	20.342
7	GradientBoosting	0.889	0.826	0.914	0.868	23.074	0.075

Table 5.10: The second Instagram without normalization performance

The table provides detailed performance metrics and timing information for several machine learning models applied to a classification task. RandomForest emerges as the top performer with an impressive accuracy of 0.958, precision of 0.943, recall of 0.953, and F1-score of 0.948. Its train time of 22.547 is relatively high, but it has a reasonable test time of 0.628, making it a strong candidate for applications that prioritize overall classification performance.

DecisionTree also exhibits high accuracy (0.933), precision (0.936), and F1-score (0.914), but its recall is slightly lower at 0.893. Its train time of 0.808 and test time of 0.010 are among the fastest, making it an efficient option, especially for time-sensitive applications.

GradientBoosting achieves a good balance across metrics, with accuracy of 0.889, precision of 0.826, recall of 0.914, and F1-score of 0.868. Its train time of 23.074 and test time of 0.075 are moderate, suggesting it could be a suitable choice when balancing performance and efficiency is crucial.

LogisticRegression, KNeighbors, SVM, and GaussianNB lag behind in overall performance, with accuracy ranging from 0.729 to 0.783. However, LogisticRegression and GaussianNB have relatively fast train and test times, which could be advantageous in certain scenarios where speed is a priority.

Notably, SVM has the highest train time of 521.716 and a substantial test time of 110.007, indicating that it may be computationally expensive for this task. KNNk5 also has a relatively high test time of 20.342, which could be a concern in time-sensitive applications.

E. **With normalization :**

	Model	Accuracy	Precision	Recall	F1-score	Train Time	Test Time
0	**RandomForest**	**0.968**	**0.959**	**0.960**	**0.959**	26.096	0.416
1	LogisticRegression	0.802	0.763	0.724	0.743	0.211	0.002
2	KNeighbors	0.836	0.793	0.794	0.793	0.009	11.773
3	SVM	0.843	0.791	0.820	0.805	**283.777**	**50.469**
4	DecisionTree	0.943	0.953	0.901	0.926	0.918	0.004
5	GaussianNB	0.708	0.589	0.870	0.702	0.042	0.007
6	KNNk5	0.836	0.793	0.794	0.793	0.009	11.887
7	GradientBoosting	0.888	0.825	0.911	0.866	28.431	0.050

Table 5.11: The second Instagram without normalization performance

The table displays the performance metrics and training/testing times for various machine learning models on a classification task.
RandomForest stands out with the highest accuracy of 0.968, precision of 0.959, recall of 0.960, and F1-score of 0.959. Its train time of 26.096 and test time of 0.416 are reasonable, making it an excellent choice for this classification problem.
DecisionTree also performs well, with an accuracy of 0.943, precision of 0.953, recall of 0.901, and F1-score of 0.926. It has a fast train time of 0.918 and test time of 0.004, making it an efficient option.
GradientBoosting achieves an accuracy of 0.888, precision of 0.825, recall of 0.911, and F1-score of 0.866. Its train time of 28.431 and test time of 0.050 are moderate, suggesting a good balance between performance and efficiency.
LogisticRegression, KNeighbors, SVM, and GaussianNB lag behind in overall performance, with accuracy ranging from 0.708 to 0.843.
SVM has the highest train time of 283.777 and a substantial test time of 50.469, indicating potential computational inefficiency for this task.
KNeighbors and KNNk5 have relatively high test times of 11.773 and

11.887, respectively, which could be a concern in time-sensitive applications.

GaussianNB stands out with a low precision of 0.589, despite a reasonable recall of 0.870, suggesting it may struggle with classifying positive instances correctly.

F. **With selected features :**

	Model	Accuracy	Precision	Recall	F1-score	Train Time	Test Time
0	**RandomForest**	**0.906**	**0.920**	**0.836**	**0.876**	15.382	0.704
1	LogisticRegression	0.774	0.750	0.641	0.692	0.166	0.006
2	KNeighbors	0.799	0.760	0.717	0.738	0.176	2.098
3	SVM	0.785	0.742	0.702	0.721	**308.444**	**54.117**
4	DecisionTree	0.887	0.904	0.800	0.849	0.462	0.014
5	GaussianNB	0.753	0.701	0.654	0.677	0.095	0.006
6	KNNk5	0.799	0.760	0.717	0.738	0.180	1.620
7	GradientBoosting	0.795	0.735	0.752	0.743	10.378	0.044

Table 5.12: The second Instagram without normalization performance

The table presents a comparison of various machine learning models evaluated on a classification task using metrics such as accuracy, precision, recall, F1-score, and training/testing times.

RandomForest emerges as the top performer with an accuracy of 0.906, precision of 0.920, recall of 0.836, and F1-score of 0.876. Its train time of 15.382 and test time of 0.704 are reasonable, making it a strong choice for this classification problem.

DecisionTree also performs well, with an accuracy of 0.887, precision of 0.904, recall of 0.800, and F1-score of 0.849. It has a relatively fast train time of 0.462 and test time of 0.014, suggesting good efficiency.

LogisticRegression, KNeighbors, SVM, GaussianNB, KNNk5, and GradientBoosting lag behind in overall performance, with accuracy ranging from 0.753 to 0.799.

SVM stands out with the highest train time of 308.444 and a substantial test time of 54.117, indicating potentially high computational requirements for this task.

GradientBoosting achieves an accuracy of 0.795, precision of 0.735, recall of 0.752, and F1-score of 0.743, with a moderate train time of

10.378 and test time of 0.044, suggesting a balanced trade-off between performance and efficiency.

GaussianNB has the lowest accuracy of 0.753 and precision of 0.701, although its recall of 0.654 is not the lowest. Its train time of 0.095 and test time of 0.006 are relatively fast.

- **How we selected the features**

We used the `SelectKBest` with `f_classif` to identify the top 5 features by their ANOVA F-values from combined `fake_data` and `real_data`.

G. List of Confusion Matrix

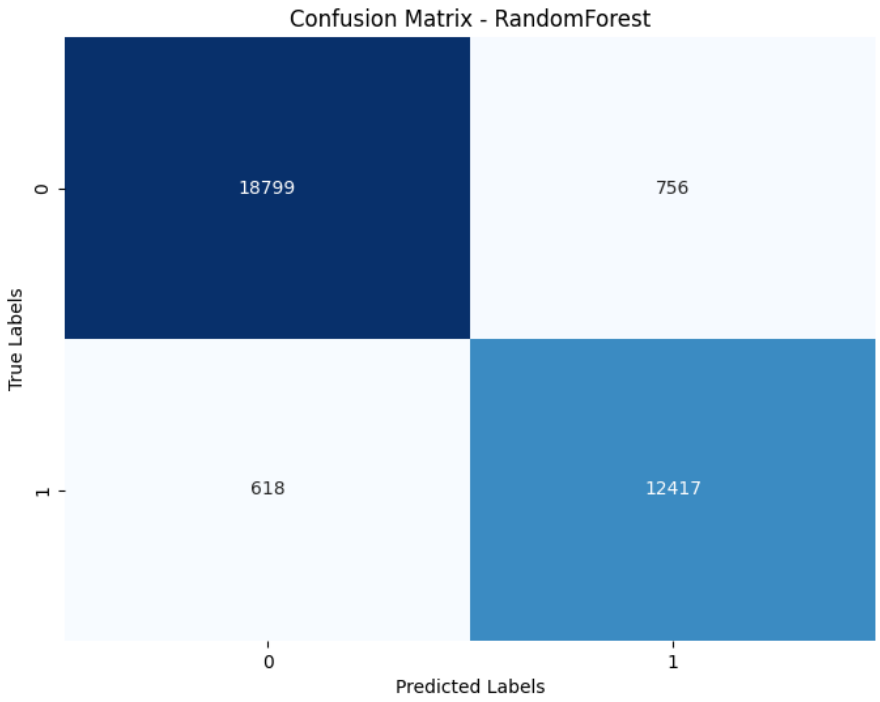

Figure 5.32: *Confusion Matrix Random Forest*

The matrix shows:

True Negatives (top-left): 18799 This represents the number of instances that were correctly predicted as the negative class (0).

False Positives (top-right): 756 These are instances that were incorrectly predicted as the positive class (1) when they actually belong to

the negative class (0).
False Negatives (bottom-left): 618 These are instances that were incorrectly predicted as the negative class (0) when they actually belong to the positive class (1).
True Positives (bottom-right): 12417 This represents the number of instances that were correctly predicted as the positive class (1).

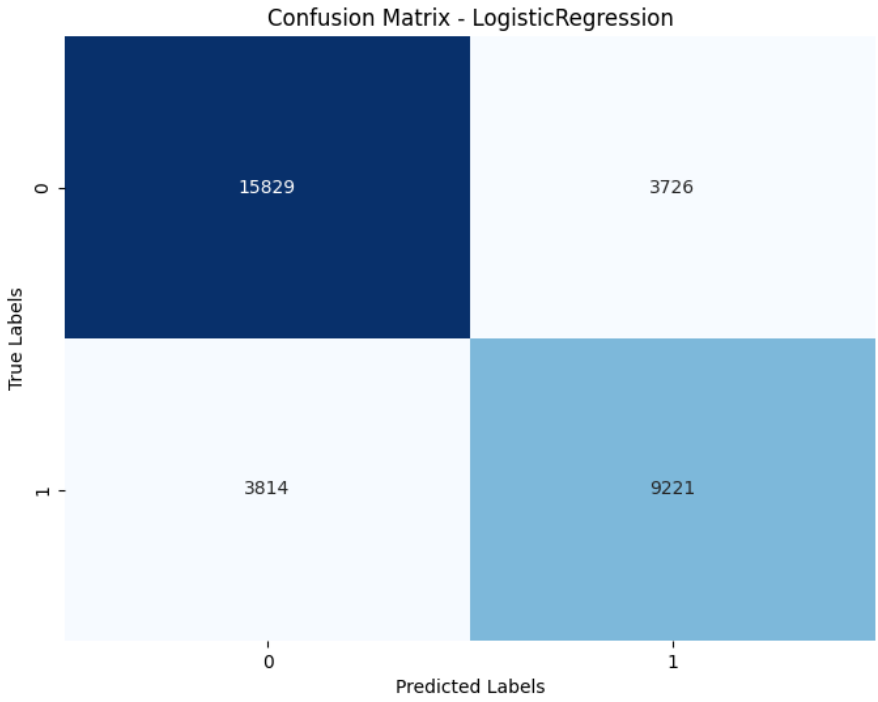

Figure 5.33: *Confusion Matrix Logistic Regression*

The matrix shows:
True Negatives (Top-Left): There are 15829 instances where the true label is 0 (negative class), and the model correctly predicted 0. These are the correct predictions for the negative class. False Positives (Top-Right): There are 3726 instances where the true label is 0 (negative class), but the model incorrectly predicted 1 (positive class). These are the incorrect predictions for the negative class, also known as Type I errors. False Negatives (Bottom-Left): There are 3814 instances where the true label is 1 (positive class), but the model incorrectly predicted

0 (negative class). These are the incorrect predictions for the positive class, also known as Type II errors. True Positives (Bottom-Right): There are 9221 instances where the true label is 1 (positive class), and the model correctly predicted 1. These are the correct predictions for the positive class.

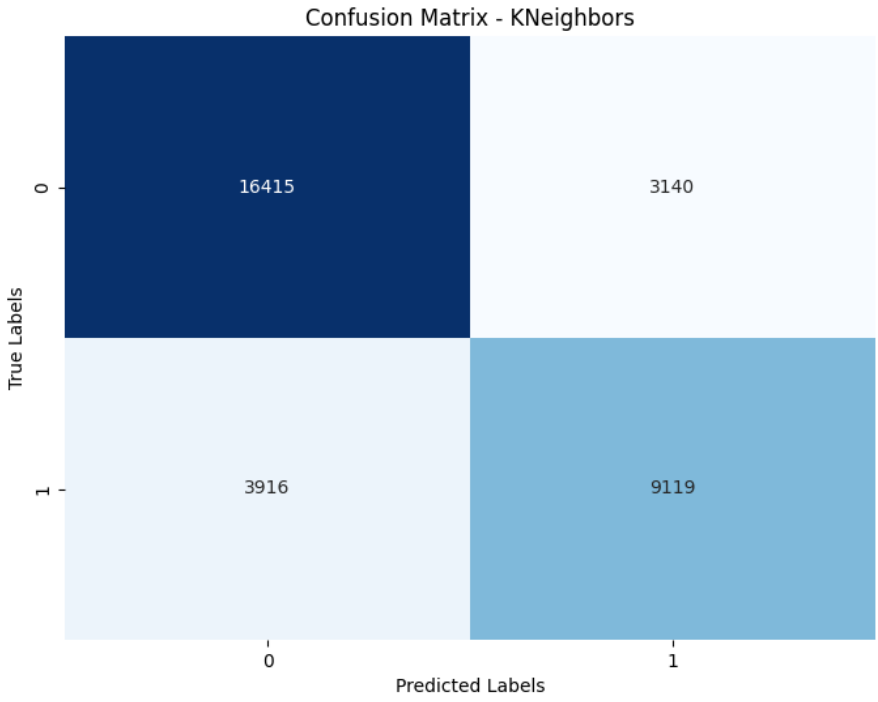

Figure 5.34: *Confusion Matrix KNeighbors Classifier*

The matrix shows:
True Negatives (top-left): 16415 This represents the number of instances that were correctly predicted as the negative class (0). False Positives (top-right): 3916 These are instances that were incorrectly predicted as the positive class (1) when they actually belong to the negative class (0). False Negatives (bottom-left): 3140 These are instances that were incorrectly predicted as the negative class (0) when they actually belong to the positive class (1). True Positives (bottom-right): 9119 This represents the number of instances that were correctly predicted as the positive class (1).

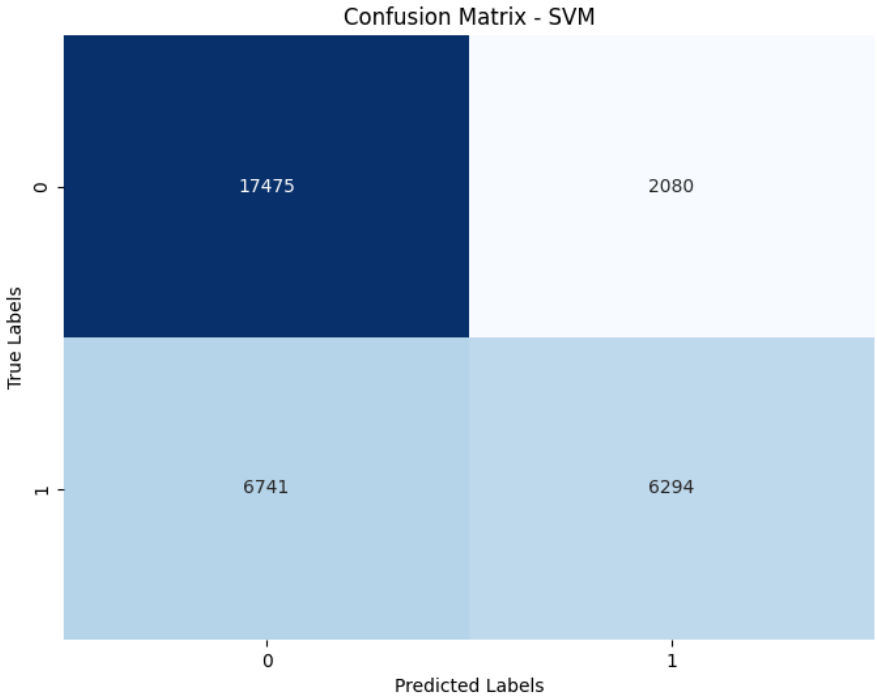

Figure 5.35: *Confusion Matrix SVM*

The values in the confusion matrix are: True Label 0:
True Positives (correctly predicted as 0): 17475 False Negatives (incorrectly predicted as 1): 2080
True Label 1:
False Positives (incorrectly predicted as 0): 6741 True Negatives (correctly predicted as 1): 6294

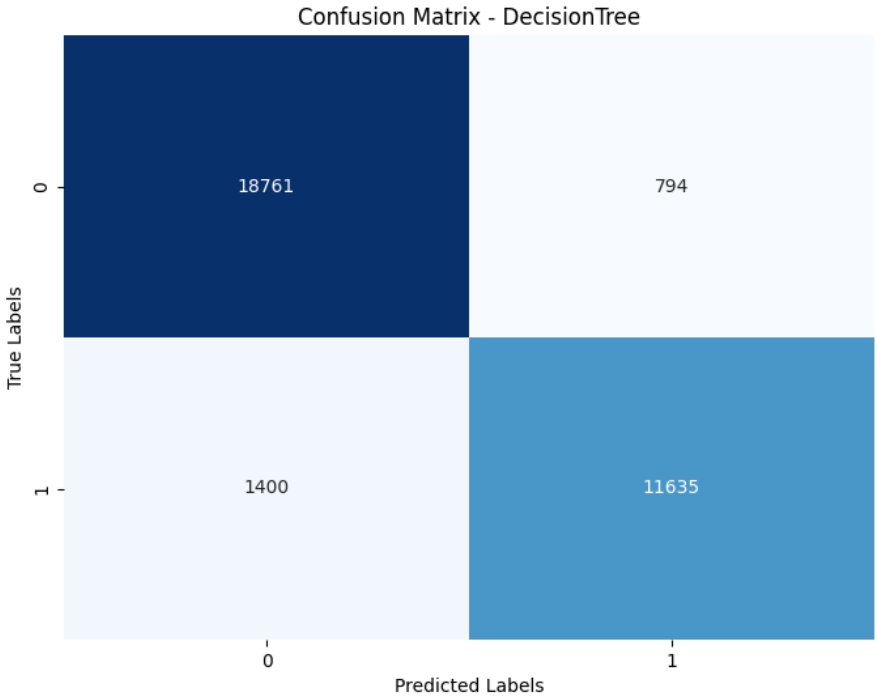

Figure 5.36: *Confusion Matrix Decision Tree*

The matrix shows:
True Negatives (top-left): 18761 This represents the number of instances that were correctly predicted as the negative class (0).
False Positives (top-right): 794 These are instances that were incorrectly predicted as the positive class (1) when they actually belong to the negative class (0).
False Negatives (bottom-left): 1400 These are instances that were incorrectly predicted as the negative class (0) when they actually belong to the positive class (1).
True Positives (bottom-right): 11635 This represents the number of instances that were correctly predicted as the positive class (1).
The confusion matrix provides insights into the model's performance by highlighting the types of errors it makes. A good model should have high values along the diagonal (True Negatives and True Positives) and low values in the off-diagonal entries (False Positives and False Negatives).

In this case, the Decision Tree model seems to perform reasonably well, with a high number of True Negatives and True Positives. However, there are still some misclassifications, with 794 False Positives and 1400 False Negatives.

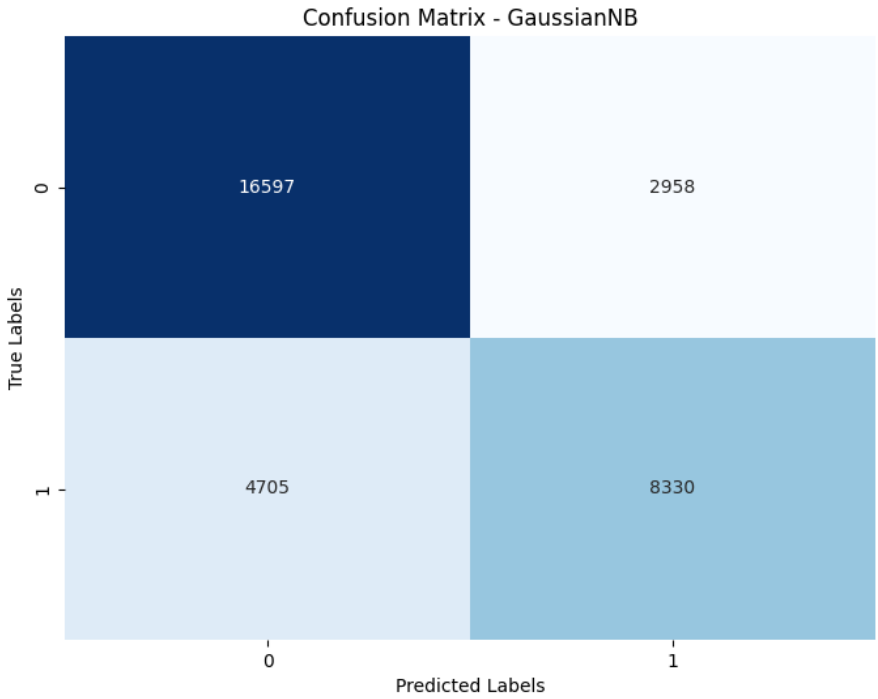

Figure 5.37: *Confusion Matrix GaussianNB*

The values in the confusion matrix are: True Label 0:
True Positives (correctly predicted as 0): 16597 False Negatives (incorrectly predicted as 1): 2958
True Label 1:
False Positives (incorrectly predicted as 0): 4705 True Negatives (correctly predicted as 1): 8330

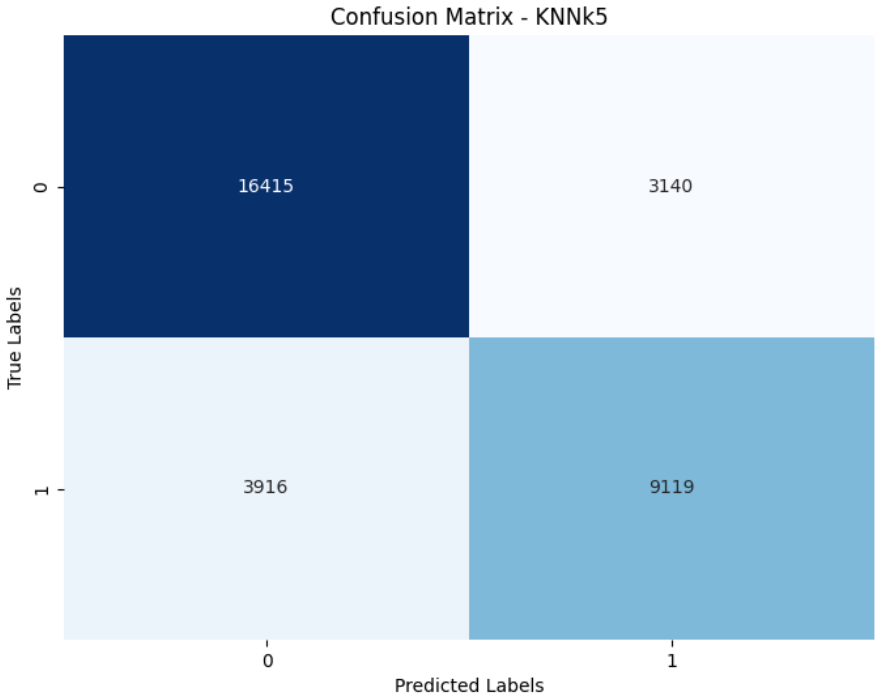

Figure 5.38: *Confusion Matrix KNN (K=5)*

The values in the confusion matrix are: True Label 0:
True Positives (correctly predicted as 0): 16415 False Negatives (incorrectly predicted as 1): 3140
True Label 1:
False Positives (incorrectly predicted as 0): 3916 True Negatives (correctly predicted as 1): 9119

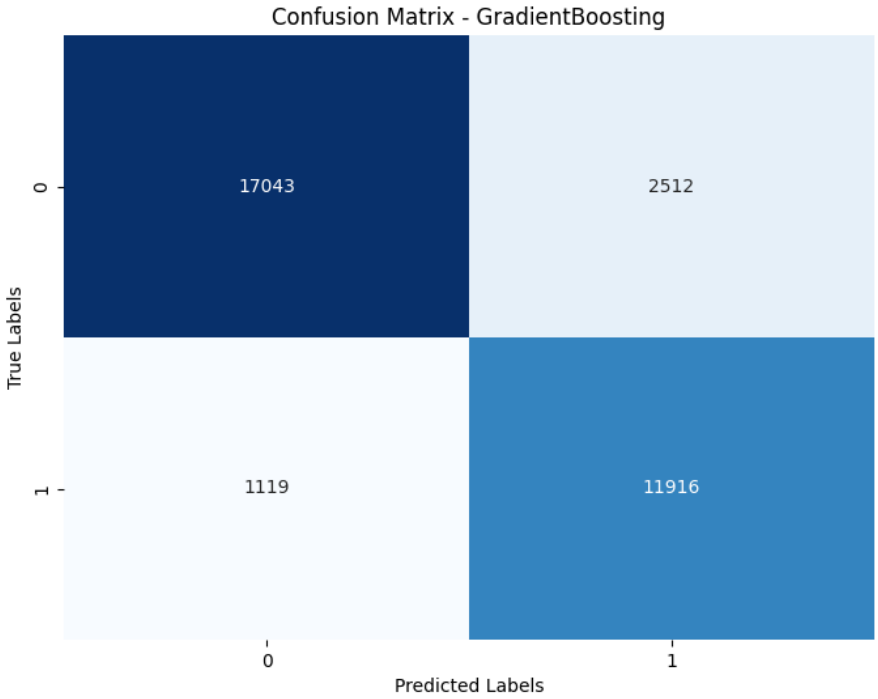

Figure 5.39: *Confusion Matrix Gradient Boosting*

The values in the confusion matrix are: True Label 0:
True Positives (correctly predicted as 0): 17043 False Negatives (incorrectly predicted as 1): 2512
True Label 1:
False Positives (incorrectly predicted as 0): 1119 True Negatives (correctly predicted as 1): 11916

Correlation Matrix

In the figure 5.36, the correlation matrix between the features of The second Instagram dataset is shown.

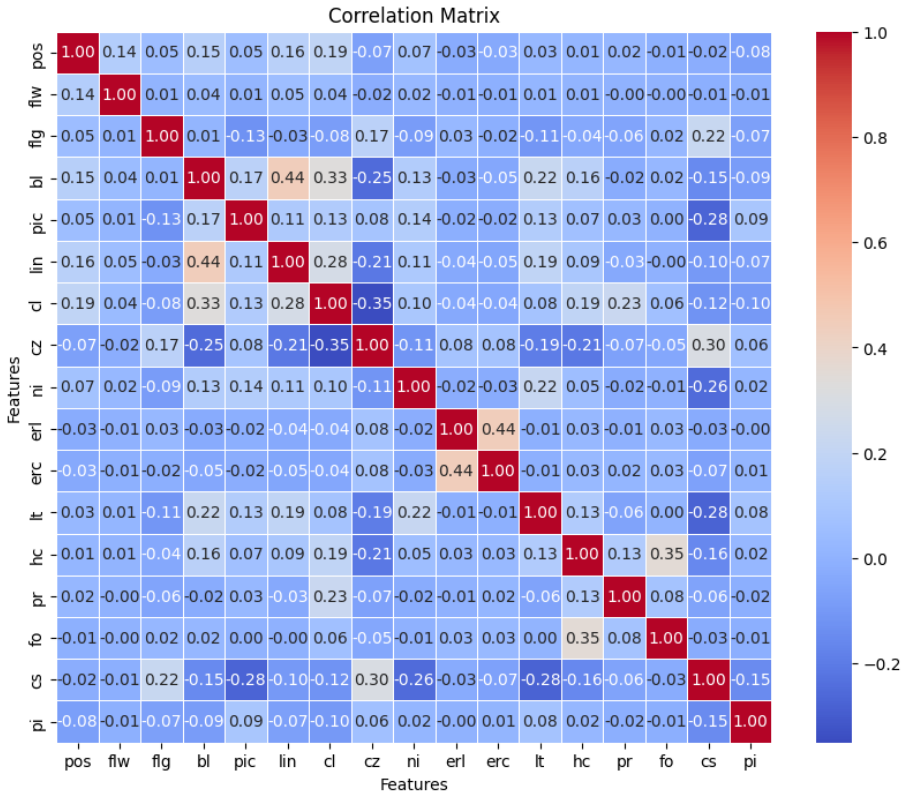

Figure 5.40: *Correlation heatMap of The second Instagram dataset*

Some observations from the matrix:

-There are strong positive correlations between variables like ”pos” and ”riw” (0.14), ”bl” and ”pic” (0.44), and ”d” and ”ln” (0.28).

-Strong negative correlations exist between ”pos” and ”z” (-0.07), ”bl” and ”ert” (-0.22), and ”cs” and ”ri” (-0.28).

-Many variables seem to have weak or negligible correlations with each other, indicated by values close to zero.

5.1.5 Social Network dataset

Dataset Size

sys.getsizeof() is used to measure and sum up the memory usage of Pandas DataFrames, providing a practical way to assess and manage memory resources when working with data in Python.

Data Type	Size (bytes)
Initial dataset	2,680,192
Preprocessed training set	39,960
Preprocessed testing set	17,224

Table 5.13: Size table of the dataset before and after pre-processing

H. **Without normalization :**

	Model	Accuracy	Precision	Recall	F1-score	Entropy	Train Time	Test Time
0	**AdaBoost Classifier**	0.679	0.637	0.946	0.761	**0.984**	70.535	7.853
1	RandomForest Classifier	0.829	0.812	0.890	0.849	0.703	50.336	0.755
2	SVC (Support Vector Classifier)	**0.851**	**0.876**	0.843	**0.859**	/	**741.220**	**612.655**
3	Decision Tree	0.775	0.769	0.835	0.800	0.136	51.628	0.248
4	Gradient Boosting	0.754	0.723	0.884	0.795	0.894	193.856	1.377
5	K-Nearest Neighbors (k=5)	0.558	0.550	**0.997**	0.709	0.912	4.150	32.127
6	Gaussian Naive Bayes	0.791	0.813	0.797	0.805	0.000	7.761	1.764
7	K-Means Clustering	0.510	0.524	0.952	0.676	0.692	173.486	0.000

Table 5.14: Social Network without normalization

Observations from the Table

1. **AdaBoost Classifier**: This model exhibits moderate accuracy and precision, but a high recall. The high recall indicates that the AdaBoost Classifier is proficient at identifying positive instances. However, it has a high entropy value, suggesting that the model might be

uncertain in its predictions. Additionally, it has a relatively high training time, which could be a drawback in time-sensitive applications.
2. **RandomForest Classifier**: The RandomForest Classifier shows high accuracy and precision, coupled with a good recall. It has a lower entropy value compared to the AdaBoost Classifier, suggesting more certainty in its predictions. In terms of time efficiency, it is faster in both training and testing compared to the AdaBoost Classifier.
3. **SVC (Support Vector Classifier)**: The SVC model outperforms all the other models in terms of accuracy and precision. It also has a high recall and the highest F1-score, indicating excellent overall performance. However, it has the longest training time and testing time, which might be a drawback for large datasets or real-time applications.
4. **Decision Tree**: The Decision Tree model has good accuracy and precision, with a recall slightly lower than that of the AdaBoost Classifier. It has a very low entropy value, suggesting high certainty in its predictions. In terms of time efficiency, it has a low training time and the shortest testing time, making it very efficient for predictions.
5. **Gradient Boosting**: The Gradient Boosting model has moderate accuracy and precision, with a high recall. It has a high entropy value, similar to the AdaBoost Classifier, suggesting that the model might be uncertain in its predictions. The training time is relatively high, and the testing time is moderate.
Conclusion
While the Support Vector Classifier (SVC) appears to be the best performing model in terms of accuracy, precision, recall, and F1-score, it also has the longest training and testing times. On the other hand, the Decision Tree model offers a good balance between performance and efficiency, with high accuracy and precision, and the shortest testing time. Therefore, the choice of model would depend on the specific requirements of the application, including the size of the dataset, the importance of prediction accuracy, and the time available for training and testing.

I. **With normalization :**

	Model	Accuracy	Precision	Recall	F1-score	Entropy	Train Time	Test Time
0	**AdaBoost Classifier**	0.679	0.637	0.946	0.761	**0.984**	175.693	7.395
1	RandomForest Classifier	**0.829**	**0.812**	0.890	**0.849**	0.703	121.973	1.081
2	SVC (Support Vector Classifier)	0.716	0.700	0.832	0.760	/	310.092	**597.465**
3	Decision Tree	0.775	0.769	0.835	0.800	0.136	148.057	0.244
4	Gradient Boosting	0.754	0.723	0.884	0.795	0.894	**718.148**	0.284
5	K-Nearest Neighbors (k=5)	0.561	0.552	**0.991**	0.709	0.939	0.349	31.000
6	Gaussian Naive Bayes	0.742	0.784	0.722	0.752	0.000	3.082	1.225
7	K-Means Clustering	0.510	0.524	0.952	0.676	0.692	173.486	0.000

Table 5.15: Social Network with normalization

Observations from the Table

1. **Accuracy**: The RandomForest Classifier has the highest accuracy (0.829), while K-Means Clustering has the lowest accuracy (0.510).
2. **Precision**: The RandomForest Classifier also has the highest precision (0.812), with K-Means Clustering having the lowest precision (0.524).
3. **Recall**: K-Nearest Neighbors has the highest recall (0.991). AdaBoost Classifier has the lowest recall among classifiers (0.946), but K-Means Clustering has an even lower recall (0.952) if considered.
4. **F1-score**: The RandomForest Classifier has the highest F1-score (0.849), and K-Means Clustering has the lowest F1-score (0.676).
5. **Entropy**: This column might be specific to the context of this analysis. It could possibly refer to some form of model complexity or uncertainty measure. AdaBoost Classifier has the highest entropy (0.984), and Gaussian Naive Bayes has the lowest entropy (0.000).
6. **Train Time**: Gradient Boosting takes the longest time to train (718.148), and Gaussian Naive Bayes takes the least time to train (3.082).

7. **Test Time**: SVC takes the longest time to test (597.465), and K-Means Clustering takes the least time to test (0.248).

-Summary

In terms of accuracy, precision, and F1-score, the RandomForest Classifier appears to be the best performing model. However, K-Nearest Neighbors has the highest recall. The Gradient Boosting model takes the longest time to train, while the SVC model takes the longest time to test. The Gaussian Naive Bayes model is the most efficient in terms of training time, and K-Means Clustering is the most efficient in terms of testing time. However, K-Means Clustering performs poorly in terms of accuracy, precision, recall, and F1-score.

J. **With selected features :**

	Model	Accuracy	Precision	Recall	F1-score	Entropy	Train Time	Test Time
0	**AdaBoost Classifier**	0.673	0.633	0.944	0.757	**0.977**	6.355	0.246
1	RandomForest Classifier	0.822	0.814	0.869	0.841	0.570	6.386	0.286
2	SVC (Support Vector Classifier)	**0.852**	**0.826**	0.919	**0.870**	/	16.819	**9.411**
3	Decision Tree	0.786	0.775	0.850	0.811	0.162	2.130	0.006
4	Gradient Boosting	0.752	0.718	0.890	0.795	0.894	7.988	0.016
5	K-Nearest Neighbors (k=5)	0.755	0.730	0.868	0.793	0.420	0.034	0.908
6	Gaussian Naive Bayes	0.845	0.805	0.940	0.868	0.001	0.110	0.028
7	K-Means Clustering	0.510	0.524	**0.952**	0.676	0.692	**173.486**	0.000

Table 5.16: Social Network with selected features

Observations from the Table

The SVC (Support Vector Classifier) has the highest accuracy (0.852) and F1-score (0.870) among the models listed, indicating it performs best in terms of these metrics. The K-Means Clustering model has the lowest accuracy (0.510) and F1-score (0.676), which is expected as K-Means is an unsupervised learning algorithm and not typically used for classification tasks. The AdaBoost Classifier has the highest recall (0.944) but not the highest precision, which suggests it is good at identifying positive instances but also has a relatively higher false positive rate. The Decision Tree model has the shortest training time (2.130 seconds) and test time (0.006 seconds), making it the fastest model in terms of computation time. The SVC (Support Vector Classifier) has the longest test time (9.411 seconds), which might be a consideration if the model needs to make predictions in real-time or on a large dataset. The K-Means Clustering has an exceptionally long training time (173.486 seconds), which is an outlier compared to the other models.

-Conclusion

It's important to note that the best model depends on the specific requirements of the task. If accuracy and F1-score are the most important metrics, then the SVC model appears to be the best choice. However, if computational efficiency is a priority, then the Decision Tree model would be a better option due to its short training and testing times. On the other hand, if the task requires a high recall, the AdaBoost Classifier would be the most suitable.

- **How we selected the features**

feature selection was performed using `SelectKBest`. The `chi2` statistical test was specified as the scoring function, which measures the dependence between each feature (term) and the target labels (fake or legitimate). The `k` parameter was set to 1000, indicating that the top 1000 features with the highest chi-square scores (indicating strong association with the target labels) are selected.

K. List of Confusion Matrix

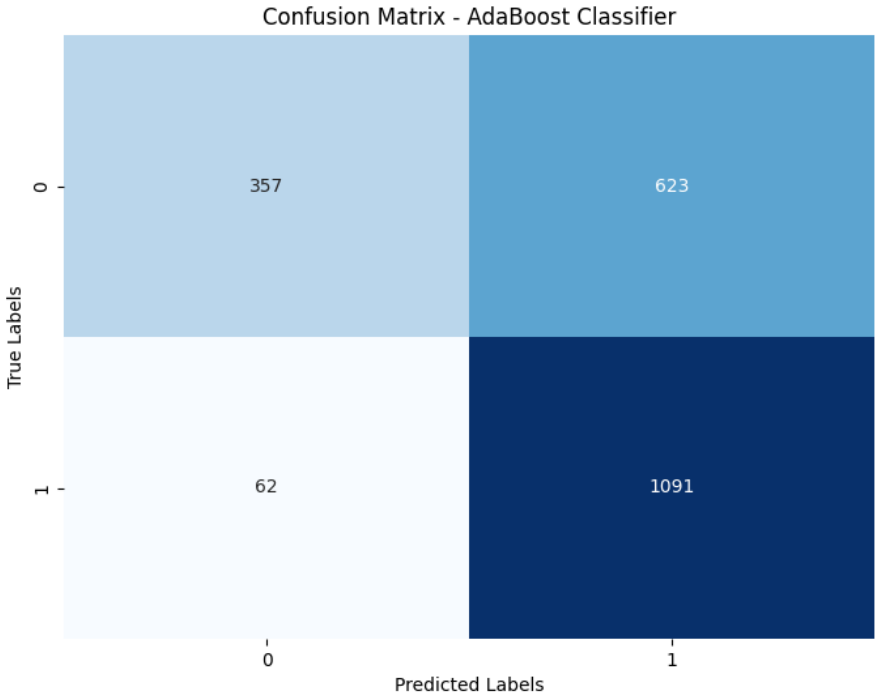

Figure 5.41: *Confusion Matrix AdaBoost Classifier*

The matrix has two rows and two columns, representing the true labels (0 and 1) and the predicted labels (0 and 1). The values in the matrix are the counts of instances for each combination of true and predicted labels. The top-left cell shows that 357 instances were correctly predicted as label 0. The bottom-right cell indicates that 1091 instances were correctly predicted as label 1. However, there were also some misclassifications. The top-right cell shows that 623 instances with the true label 0 were incorrectly predicted as label 1 (false positives). The bottom-left cell reveals that 62 instances with the true label 1 were incorrectly predicted as label 0 (false negatives).

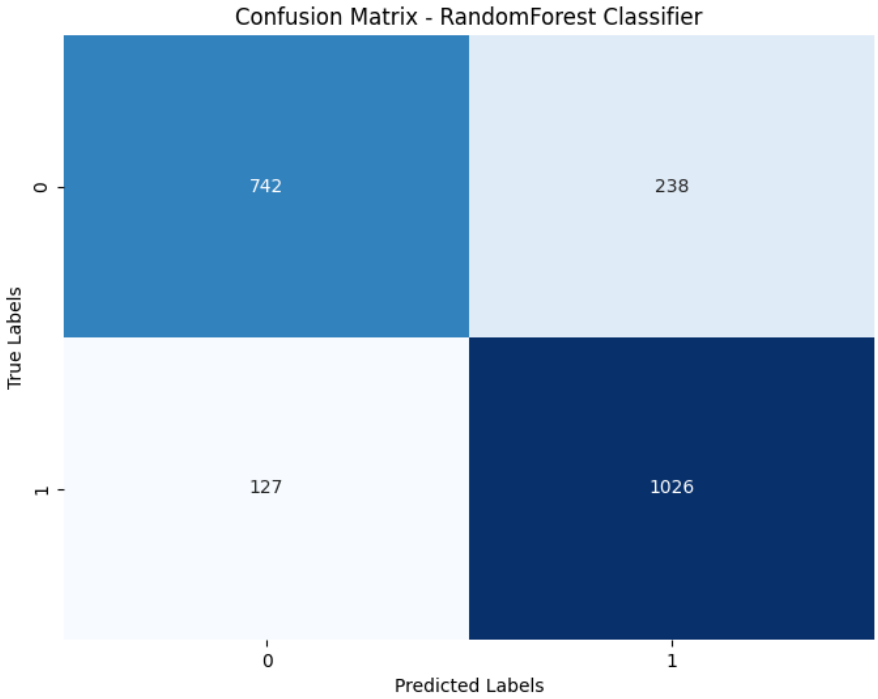

Figure 5.42: *Confusion Matrix Random Forest Classifier*

In the top-left cell, 742 instances were correctly classified as label 0 by the Random Forest model. The bottom-right cell shows that 1026 instances were correctly predicted as label 1. However, there were some misclassifications as well. The top-right cell indicates that 238 instances with the true label 0 were incorrectly predicted as label 1 (false positives). The bottom-left cell reveals that 127 instances with the true label 1 were incorrectly predicted as label 0 (false negatives). By comparing the values in this confusion matrix with the previous one for the AdaBoost classifier, we can evaluate the relative performance of the two models. In this case, the Random Forest classifier appears to have fewer false positives (238 vs. 623) but more false negatives (127 vs. 62) compared to the AdaBoost classifier.

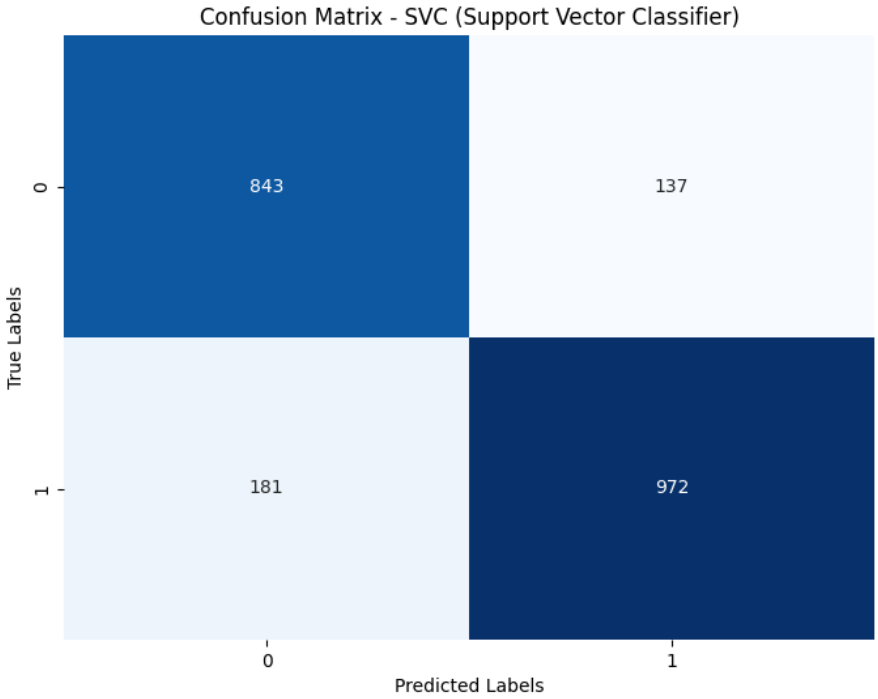

Figure 5.43: *Confusion Matrix SVC Classifier*

In the top-left cell, 843 instances were correctly classified as label 0 by the SVC model. The bottom-right cell shows that 972 instances were correctly predicted as label 1. However, there were some misclassifications as well. The top-right cell indicates that 137 instances with the true label 0 were incorrectly predicted as label 1 (false positives). The bottom-left cell reveals that 181 instances with the true label 1 were incorrectly predicted as label 0 (false negatives). Compared to the previous confusion matrices for the AdaBoost and Random Forest classifiers, the SVC model appears to have fewer false positives (137 vs. 623 and 238) but more false negatives (181 vs. 62 and 127).

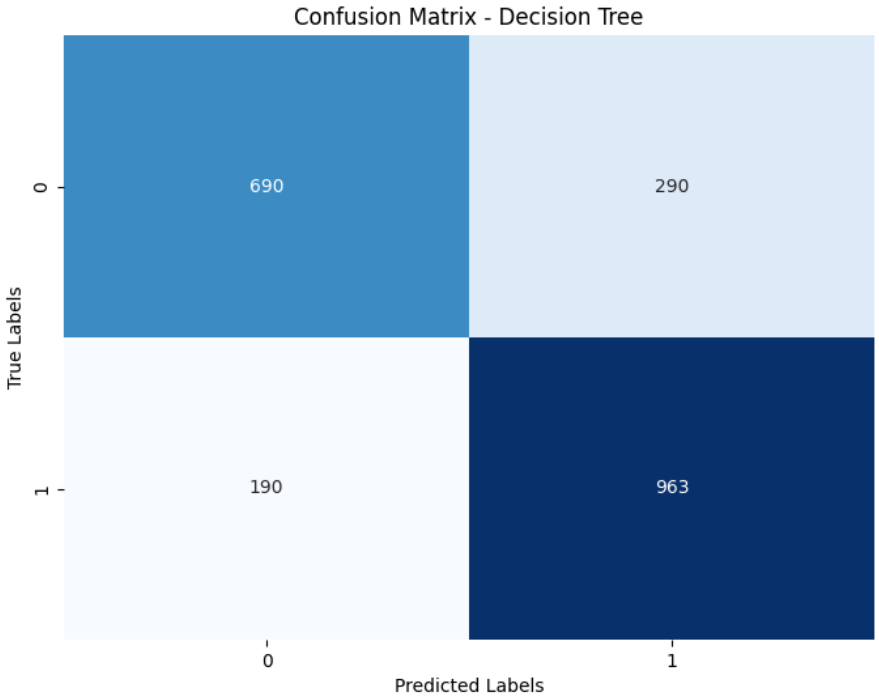

Figure 5.44: *Confusion Matrix Decision Tree*

In the top-left cell, 690 instances were correctly classified as label 0 by the decision tree model. The bottom-right cell shows that 963 instances were correctly predicted as label 1. However, there were some misclassifications as well. The top-right cell indicates that 290 instances with the true label 0 were incorrectly predicted as label 1 (false positives). The bottom-left cell reveals that 190 instances with the true label 1 were incorrectly predicted as label 0 (false negatives). Compared to other confusion matrices, this decision tree model seems to have a moderate number of false positives (290) and false negatives (190), without any extreme values suggesting significant imbalance in the misclassifications.

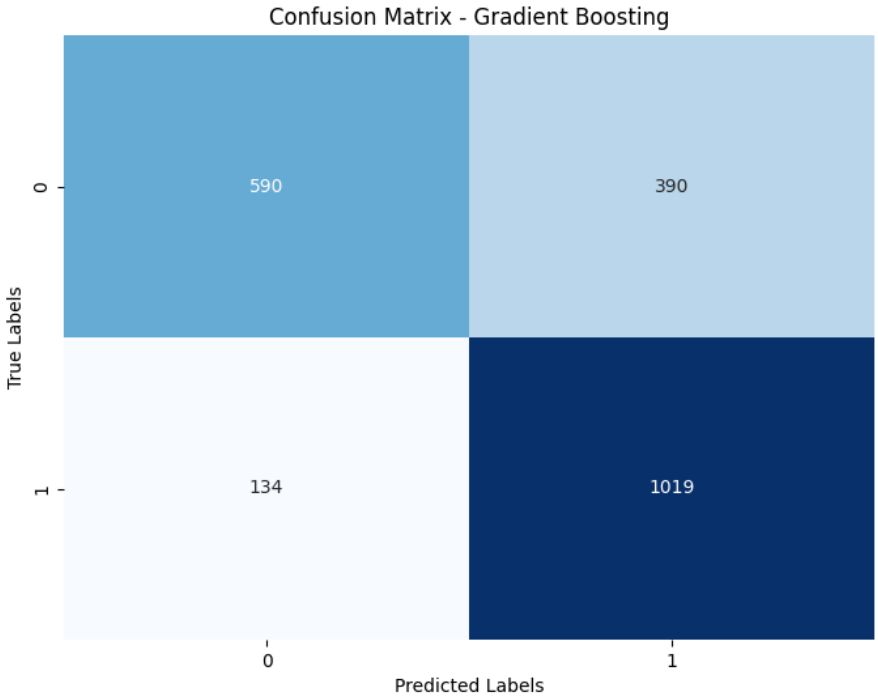

Figure 5.45: *Confusion Matrix Gradient Boosting*

The top-left cell shows that 590 instances were correctly classified as label 0 by the Gradient Boosting model. The bottom-right cell indicates that 1019 instances were correctly predicted as label 1. However, there were some misclassifications as well. The top-right cell reveals that 390 instances with the true label 0 were incorrectly predicted as label 1 (false positives). The bottom-left cell shows that 134 instances with the true label 1 were incorrectly predicted as label 0 (false negatives). Compared to the previous confusion matrices for the AdaBoost, Random Forest, and SVC classifiers, the Gradient Boosting model appears to have a higher number of false positives (390) than the SVC (137) and Random Forest (238) models, but fewer false positives than the AdaBoost model (623). It also has fewer false negatives (134) than the SVC (181) and AdaBoost (62) models, but slightly more than the Random Forest model (127).

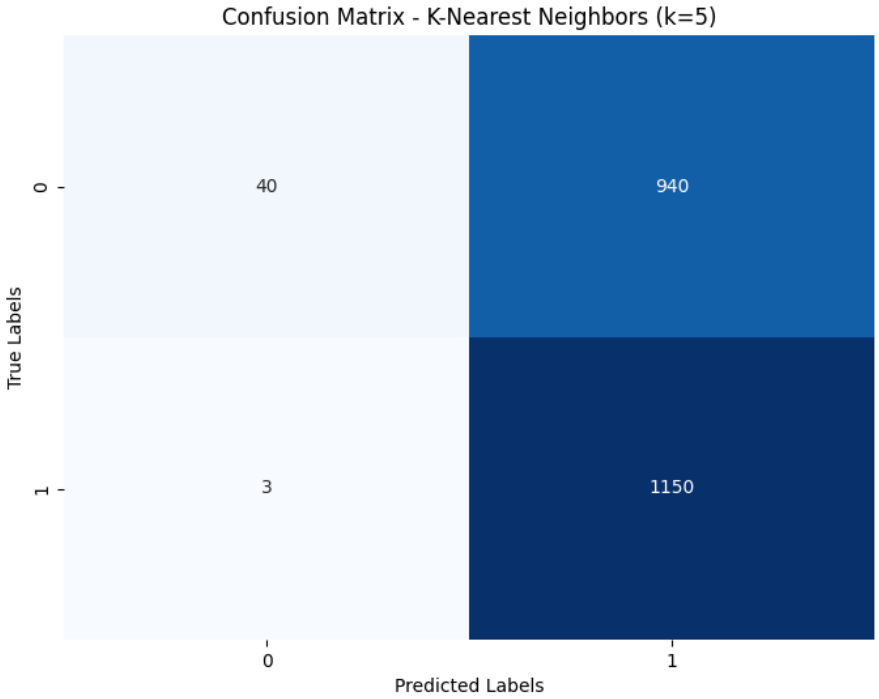

Figure 5.46: *Confusion Matrix K-Nearest Neighbors (k=5)*

In the top-left cell, 40 instances were correctly classified as label 0 by the KNN model. The bottom-right cell shows that 1150 instances were correctly predicted as label 1. However, there were some misclassifications as well. The top-right cell indicates that 940 instances with the true label 0 were incorrectly predicted as label 1 (false positives). The bottom-left cell reveals that 3 instances with the true label 1 were incorrectly predicted as label 0 (false negatives). Compared to the previous confusion matrices for other classification models, the KNN classifier with k=5 appears to have a very high number of false positives (940) but a relatively low number of false negatives (3).

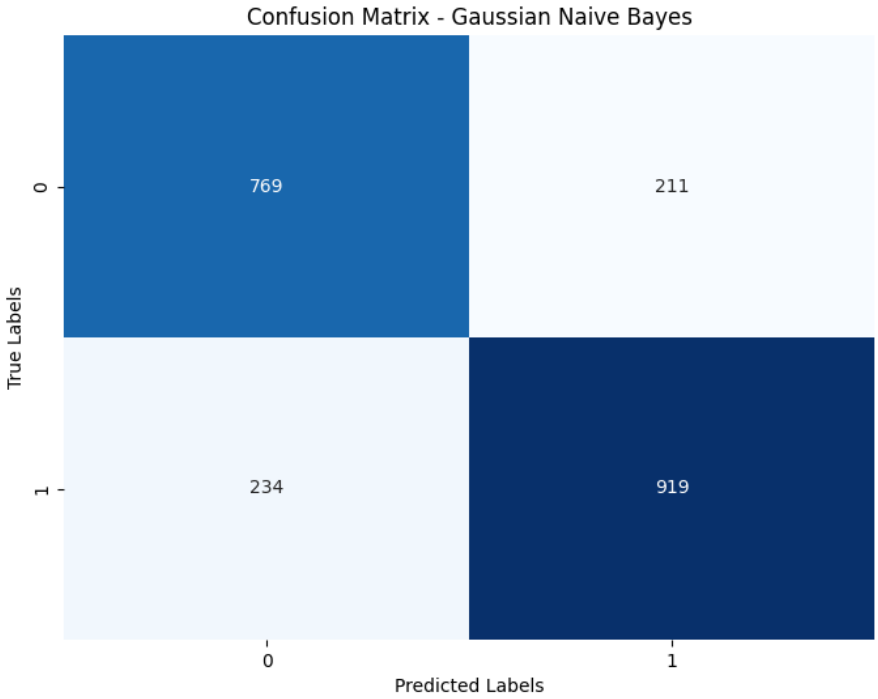

Figure 5.47: *Confusion Matrix Gaussian Naive Bayes*

In the top-left cell, the value is 769, which indicates that the classifier correctly predicted 769 instances as label 0. In the bottom-right cell, the value is 919, which means that the classifier correctly predicted 919 instances as label 1. However, the top-right cell has a value of 211, which represents the number of instances that were wrongly predicted as label 1 when the true label was 0 (false positives). Similarly, the bottom-left cell has a value of 234, which represents the number of instances that were incorrectly predicted as label 0 when the true label was 1 (false negatives).

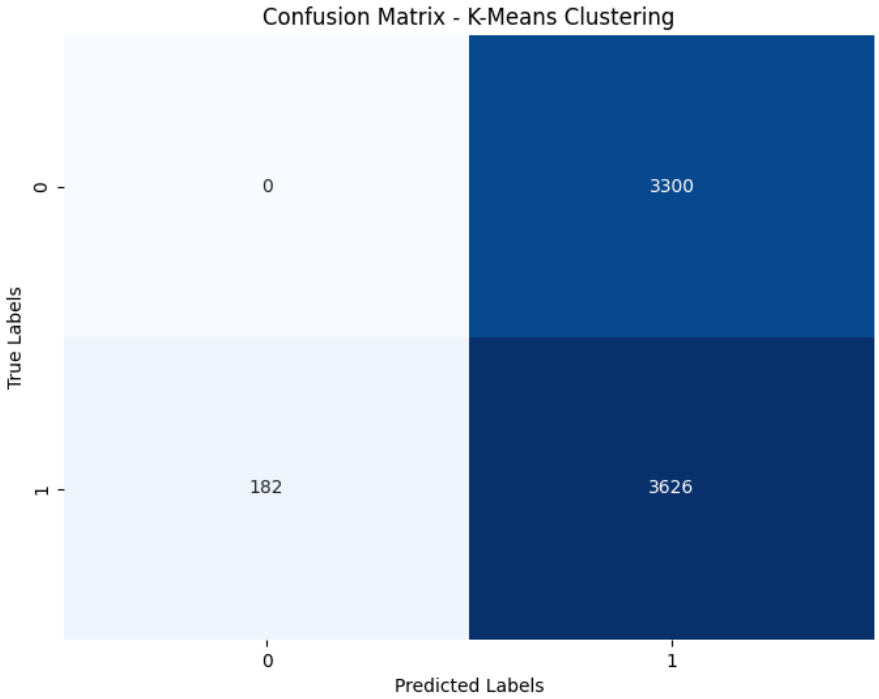

Figure 5.48: *Confusion Matrix K-Means Clustering*

The top-left cell has a value of 0, indicating that no instances were assigned to cluster 0 when they should have been in cluster 1. This is expected since clustering algorithms do not have a notion of "true labels" as in supervised classification problems. The bottom-left cell has a value of 182, representing the number of instances assigned to cluster 0 by the K-Means algorithm. The top-right cell has a value of 3300, indicating the number of instances assigned to cluster 1 by the K-Means algorithm. The bottom-right cell has a value of 3626, which is the sum of the instances assigned to clusters 0 and 1.

Correlation Matrix

In the figure 5.45, the correlation matrix between the features of The second Instagram dataset is shown.

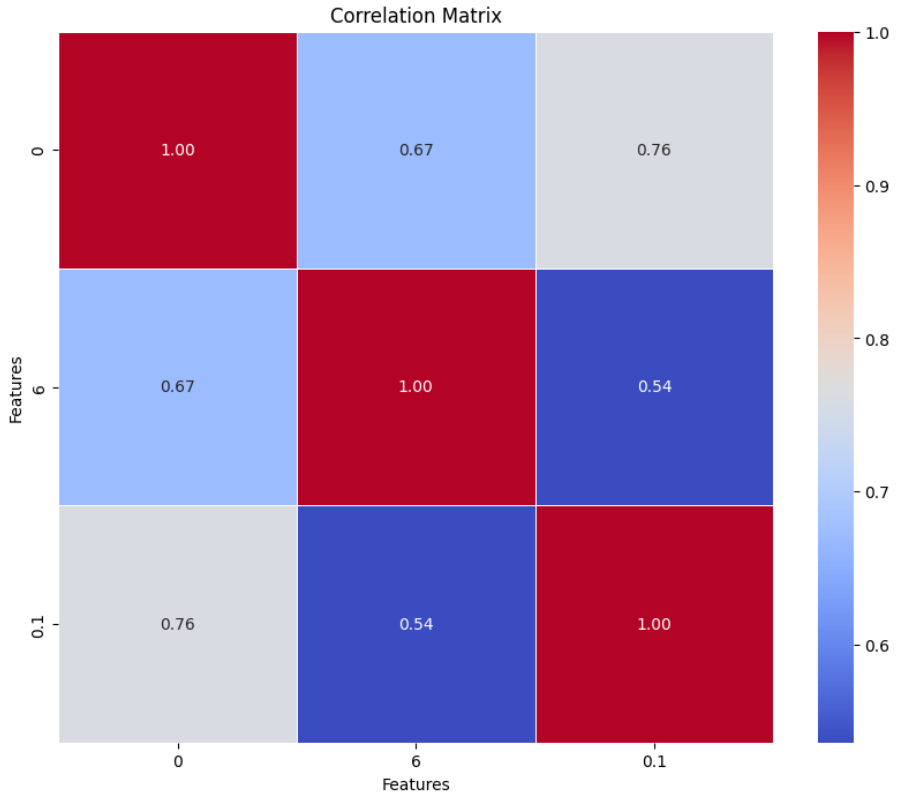

Figure 5.49: *Correlation heatMap of Social Network dataset*

A few key observations:

-The diagonal elements (1.0) indicate the perfect correlation of each feature with itself

-There is a strong positive correlation of 0.67 between features 0 and 6.

-Feature 0.1 has a moderate positive correlation of 0.76 with feature 0, and a relatively weaker positive correlation of 0.54 with feature 6.

-The color gradient from red (positive correlation) to blue (negative correlation) suggests that there are no significant negative correlations among these three features.

5.1.6 ISIS dataset

Dataset Size

sys.getsizeof() is used to measure and sum up the memory usage of Pandas DataFrames, providing a practical way to assess and manage memory resources when working with data in Python.

Dataset	Size Before Preprocessing (MB)	Size After Preprocessing (MB)
Positive accounts dataset	13.71	13.85
Negative accounts dataset	90.18	90.31

Figure 5.50: *Size table of the dataset before and after pre-processing*

L. **Without normalization :**

	Model	Accuracy	Precision	Recall	F1-score	Entropy	Train Time	Test Time
1	**SVM**	**0.999**	**0.999**	**0.999**	**0.999**	0.005	**100.970**	**2.581**
2	Naive Bayes	0.984	**0.999**	0.969	0.984	**0.066**	0.014	0.002
3	Logistic Regression	0.997	0.996	**0.999**	0.997	0.034	0.931	0.003

Table 5.17: ISIS without normalization

Here are the key observations:

-SVM has the highest accuracy (0.999), precision (0.999), recall (0.999), and F1-score (0.999) among the three models, indicating excellent performance in correctly classifying instances. However, it has the highest entropy (0.005), suggesting some uncertainty or impurity in the model's predictions.

-Naive Bayes has the lowest accuracy (0.984) and recall (0.969) but a high precision (0.999). Its entropy (0.066) is higher than SVM, indicating more uncertainty in its predictions.

-Logistic Regression has a high accuracy (0.997), precision (0.996), and recall (0.999), with an F1-score of 0.997, which is slightly lower than SVM but higher than Naive Bayes.

-In terms of training and testing times, SVM has the highest train time (100.970) and test time (2.581), while Naive Bayes has the lowest train time (0.014) and test time (0.002).

Overall, based on the accuracy, precision, recall, and F1-score metrics, SVM appears to be the best performing model for this particular dataset and problem. However, the choice of model may also depend on the specific requirements of the task, such as the trade-off between performance and computational resources (training and testing times).

M. **With normalization :**

	Model	Accuracy	Precision	Recall	F1-score	Entropy	Train Time	Test Time
1	**SVM**	**0.999**	0.998	**0.999**	**0.999**	0.009	**39.572**	**0.832**
2	Naive Bayes	0.978	**0.999**	0.958	0.978	**0.107**	0.010	0.001
3	Logistic Regression	0.997	0.995	0.998	0.997	0.037	0.095	0.001

Table 5.18: ISIS with normalization

The SVM model achieves the highest accuracy of 0.999, precision of 0.998, recall of 0.999, and F1-score of 0.999, indicating outstanding performance. However, it has the highest entropy of 0.009 and the longest train time of 39.572, suggesting some complexity in the model.

The Naive Bayes model has a slightly lower accuracy of 0.978, precision of 0.999, and recall of 0.958, resulting in an F1-score of 0.978. Its entropy is 0.107, which is higher than the other models, implying some uncertainty. However, it has the fastest train time of 0.010 and test time of 0.001, making it computationally efficient.

The Logistic Regression model falls in between the other two models in terms of performance. Its accuracy is 0.997, precision is 0.995, recall is 0.998, and F1-score is 0.997. The entropy of 0.037 is lower than Naive Bayes but higher than SVM, indicating moderate uncertainty. The train time of 0.095 and test time of 0.001 are reasonable.

Overall, the SVM model appears to be the best performer in terms of accuracy, precision, recall, and F1-score, but it comes at the cost of higher entropy and longer train time. The choice of model would depend on the specific requirements of the problem, balancing performance with computational efficiency and model complexity.

N. **With selected features :**

	Model	Accuracy	Precision	Recall	F1-score	Entropy	Train Time	Test Time
0	**SVM**	0.996	0.998	0.994	0.996	0.017	**36.250**	**0.411**
1	Naive Bayes	0.970	**0.999**	0.942	0.969	**0.578**	0.010	0.001
2	Logistic Regression	**0.998**	**0.999**	**0.997**	**0.998**	0.010	0.352	0.001

Table 5.19: ISIS with selected features

The Logistic Regression model achieves the highest accuracy of 0.998, precision of 0.999, recall of 0.997, and F1-score of 0.998. It also has the lowest entropy of 0.010, indicating low uncertainty in its predictions. The train time of 0.352 is reasonable, and the test time of 0.001 is fast. The SVM model has a slightly lower accuracy of 0.996, precision of 0.998, recall of 0.994, and F1-score of 0.996. Its entropy is 0.017, which is higher than Logistic Regression but still relatively low. However, it has the longest train time of 36.250 and a test time of 0.411, suggesting it might be computationally more expensive.

The Naive Bayes model performs the worst among the three models, with an accuracy of 0.970, precision of 0.999, recall of 0.942, and F1-score of 0.969. It has the highest entropy of 0.578, indicating a higher level of uncertainty in its predictions. However, it has the fastest train time of 0.010 and test time of 0.001, making it computationally efficient. Overall, the Logistic Regression model appears to be the best performer in terms of accuracy, precision, recall, F1-score, and entropy, with reasonably fast train and test times. The SVM model also performs well but may be more computationally expensive. The choice between these

two models would depend on the specific requirements of the problem, balancing performance with computational efficiency.

- **How we selected the features**

`SelectKBest` was employed with the chi-square (`chi2`) scoring function to select the top 1000 features based on their association with the target labels (`y`). This statistical test evaluates the independence between features and labels, retaining the most relevant ones.

List of Confusion Matrix

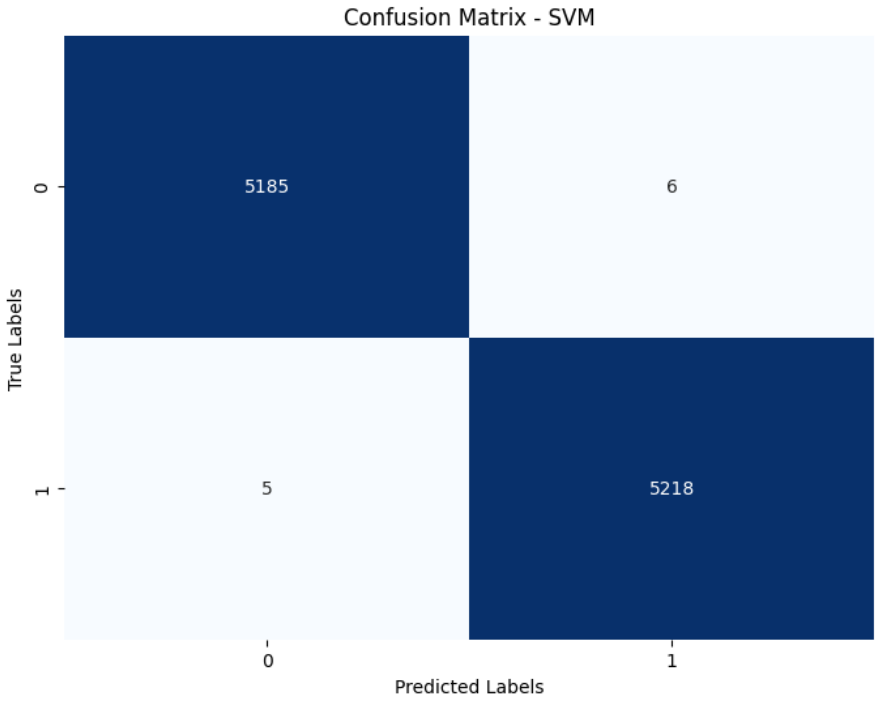

Figure 5.51: *Confusion Matrix SMV*

Looking at the values, we can see that the model classified 5185 instances as class 0 (top-left value), and 5218 instances as class 1 (bottom-right value). The top-right value (6) represents the number of instances that were incorrectly classified as class 1 when they should have been class 0. Similarly, the bottom-left value (5) represents the number of instances that were incorrectly classified as class 0 when they should have been class 1.

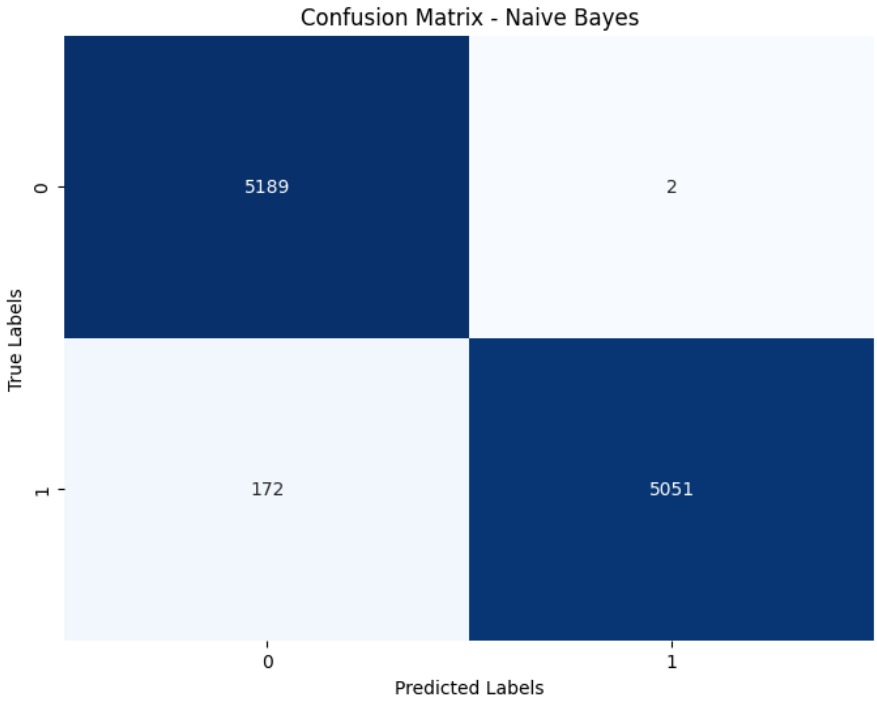

Figure 5.52: *Confusion Matrix Naive Bayes*

Looking at the values, we can see that the Naive Bayes model correctly classified 5189 instances as class 0 (top-left) and 5051 instances as class 1 (bottom-right). However, it misclassified 172 instances as class 0 when they should have been class 1 (bottom-left), and 2 instances as class 1 when they should have been class 0 (top-right).
Compared to the SVM confusion matrix from the previous image, the Naive Bayes model seems to have a higher number of misclassifications, particularly for instances belonging to class 1. The value of 172 in the bottom-left cell indicates that a significant number of instances from class 1 were incorrectly classified as class 0.

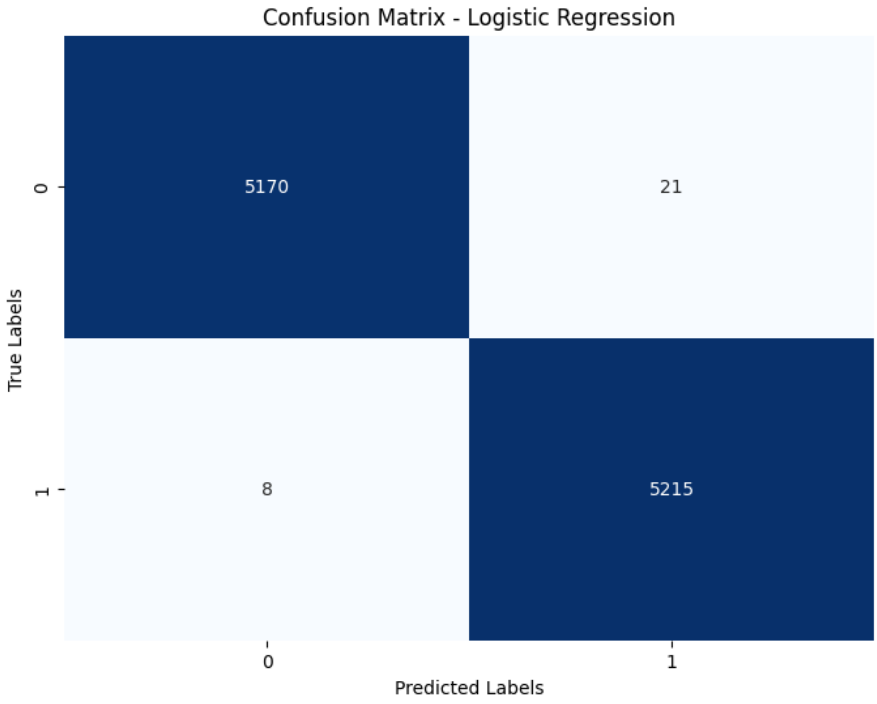

Figure 5.53: *Confusion Matrix Logistic Regression*

The diagonal values show that the model correctly classified 5170 instances as class 0 (top-left) and 5215 instances as class 1 (bottom-right). This indicates a high accuracy for both classes.

However, the off-diagonal values reveal some misclassifications. The top-right value of 21 means that 21 instances from class 0 were incorrectly predicted as class 1. Similarly, the bottom-left value of 8 indicates that 8 instances from class 1 were misclassified as class 0.

Compared to the previous Naive Bayes confusion matrix, the Logistic Regression model appears to have a lower number of misclassifications, particularly for instances belonging to class 1. The higher diagonal values and lower off-diagonal values suggest better overall performance.

Correlation Matrix

In the figure 5.46, the correlation matrix between the features of The second Instagram dataset is shown.

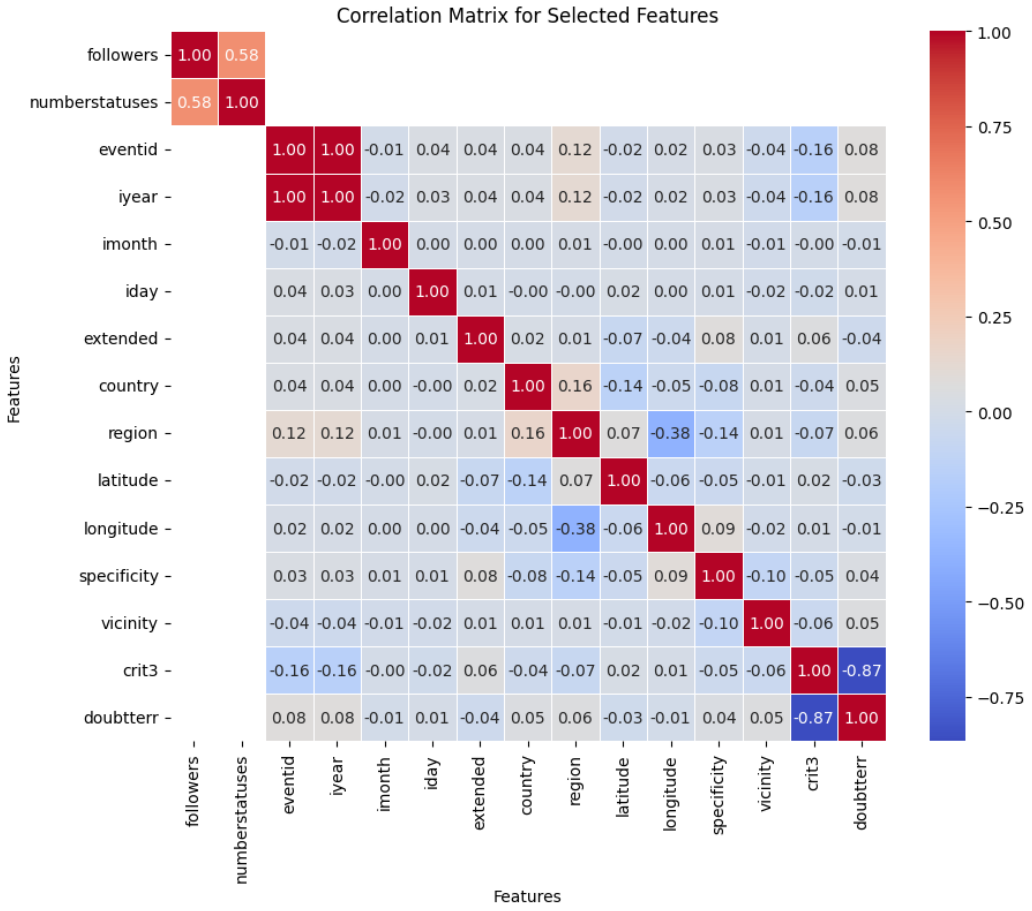

Figure 5.54: *Correlation heatMap of ISIS dataset*

Some key observations from the correlation matrix:
-The features ”followers” and ”numberstatuses” have a strong positive correlation of 0.58, indicating that as the number of followers increases, the number of statuses tends to increase as well.
-The features ”latitude” and ”longitude” have a strong negative correlation of -0.38, which is expected since these features represent geographic coordinates and are typically inversely related.
-The feature ”imonth” (likely representing the month) has weak or near-zero correlations with most other features, except for a moderate posi-

tive correlation with "iday" (likely representing the day).
-The features "eventyear" and "eventid" have identical correlation patterns with other features, suggesting they may be related or derived from each other.
-The feature "crt3" has a strong positive correlation of 0.87 with "doubtterr", indicating a potential relationship between these two features.
-Many features exhibit weak or near-zero correlations with each other, suggesting they may be independent or capturing different aspects of the data.

5.1.7 GAO Optimization Meta-heuristic implementation

in our implementation we tried all the models we used in machine learning for every dataset and the combination that's gave us the best results is the following

O. Facebook dataset GOA with Gradient Boosting

lightblue **Metric**	**Value**
Accuracy	0.995
F1 Score	0.997
Precision	0.994
Recall	1.000
Time Taken	538.450 seconds

Table 5.20: GAO implementation on Facebook dataset using Gradient Boosting mode

Iteration Details	Best Score
Iteration 1/10	0.9946524064171123
Iteration 2/10	0.9946524064171123
Iteration 3/10	0.9946524064171123
Iteration 4/10	0.9946524064171123
Iteration 5/10	0.9946524064171123
Iteration 6/10	0.9946524064171123
Iteration 7/10	0.9946524064171123
Iteration 8/10	0.9946524064171123
Iteration 9/10	0.9946524064171123
Iteration 10/10	0.9946524064171123
Best n_estimators	163.0
Best learning_rate	0.098338
Best max_depth	9.0
Best min_samples_split	14.0
Best min_samples_leaf	4.0

Table 5.21: Results of GAO implementation on Facebook dataset iterations

P. Twitter dataset GOA with Gradient Boosting

lightblue **Metric**	**Value**
Accuracy	0.930
F1 Score	0.931
Precision	0.940
Recall	0.922
Time Taken	97.110 seconds

Table 5.22: GAO implementation on Twitter dataset using Gradient Boosting model

Iteration Details	Best Score
Iteration 1/10	0.9266666666666666
Iteration 2/10	0.93
Iteration 3/10	0.93
Iteration 4/10	0.93
Iteration 5/10	0.93
Iteration 6/10	0.93
Iteration 7/10	0.93
Iteration 8/10	0.93
Iteration 9/10	0.93
Iteration 10/10	0.93
Best_n_estimators	25.0
Best_learning_rate	0.127689
Best_max_depth	31.0
Best_min_samples_split	2.0
Best_min_samples_leaf	3.0

Table 5.23: Results of GAO implementation on Twitter dataset iterations

Q. Instagram dataset GOA with Gradient Boosting

lightblue **Metric**	**Value**
Accuracy	0.958
F1 Score	0.959
Precision	0.951
Recall	0.967
Time Taken	92.179 seconds

Table 5.24: GAO implementation on Instagram dataset using Gradient Boosting model

Iteration Details	Best Score
Iteration 1/10	0.95
Iteration 2/10	0.95
Iteration 3/10	0.95
Iteration 4/10	0.95
Iteration 5/10	0.95
Iteration 6/10	0.95
Iteration 7/10	0.95
Iteration 8/10	0.9583333333333334
Iteration 9/10	0.9583333333333334
Iteration 10/10	0.9583333333333334
Best n_estimators	198.0
Best_learning_rate	0.172153
Best_max_depth	27.0
Best_min_samples_split	11.0
Best_in_samples_leaf	10.0

Table 5.25: Results of GAO implementation on Instagram dataset iterations

R. The Second Instagram dataset GOA with Random Forest

lightblue **Metric**	**Value**
Accuracy	0.952
F1 Score	0.941
Precision	0.928
Recall	0.955
Time Taken	2206.823 seconds

Table 5.26: GAO implementation on The Second Instagram dataset using Random Forest model

Iteration Details	Best Score
Iteration 1/10	0.9524393985885241
Iteration 2/10	0.9525928198833998
Iteration 3/10	0.9532985578398282
Iteration 4/10	0.9539122430193311
Iteration 5/10	0.9539122430193311
Iteration 6/10	0.9539122430193311
Iteration 7/10	0.9539122430193311
Iteration 8/10	0.9539122430193311
Iteration 9/10	0.9539122430193311
Iteration 10/10	0.9539122430193311
Best n_estimators	94.0
Best_max_depth	20.0

Table 5.27: Results of GAO implementation on the Second Instagram dataset iterations

S. Social Network dataset GOA with SVC

lightblue **Metric**	**Value**
Accuracy	0.855
F1 Score	0.864
Precision	0.875
Recall	0.854
Time Taken	1608.212 seconds

Table 5.28: GAO implementation on Social Network dataset using SVC model

Iteration Details	Best Score
Iteration 1/10	0.8546647913736521
Iteration 2/10	0.8546647913736521
Iteration 3/10	0.8546647913736521
Iteration 4/10	0.8551336146272855
Iteration 5/10	0.8551336146272855
Iteration 6/10	0.8551336146272855
Iteration 7/10	0.8551336146272855
Iteration 8/10	0.8551336146272855
Iteration 9/10	0.8551336146272855
Iteration 10/10	0.8551336146272855
Best C	3.342676
Best gamma	0.56362

Table 5.29: Results of GAO implementation on Social Network dataset iterations

T. ISIS dataset GOA with Naive Bayes

lightblue **Metric**	**Value**
Accuracy	0.989
F1 Score	0.959
Precision	0.997
Recall	0.924
Time Taken	12.660 seconds

Table 5.30: GAO implementation on The Second ISIS dataset using Naive Bayes model

Iteration Details	Best Score
Iteration 1/10	0.9887651238717111
Iteration 2/10	0.9890531976185903
Iteration 3/10	0.9890531976185903
Iteration 4/10	0.9890531976185903
Iteration 5/10	0.9890531976185903
Iteration 6/10	0.9891492222008834
Iteration 7/10	0.9891492222008834
Iteration 8/10	0.9891492222008834
Iteration 9/10	0.9891492222008834
Iteration 10/10	0.9891492222008834
Best_Alpha	0.05767571254538316

Table 5.31: Results of GAO implementation on ISIS dataset iterations

5.1.8 SGO Optimization Meta-heuristic implementation

Same with GOA we tried all the models we used in machine learning for every dataset and the combination that's gave us the best results is the following

- Facebook dataset SGO with Gradient Boosting

lightgray Metric	Value
Accuracy	0.995
Recall	1.0
Precision	0.994
F1 Score	0.997
Time Taken	412.7715 seconds

Table 5.32: SGO implementation on Facebook dataset using Gradient Boosting model

Iteration Details	Best Score
Iteration 1/10	0.9946524064171123
Iteration 2/10	0.9946524064171123
Iteration 3/10	0.9946524064171123
Iteration 4/10	0.9946524064171123
Iteration 5/10	0.9946524064171123
Iteration 6/10	0.9946524064171123
Iteration 7/10	0.9946524064171123
Iteration 8/10	0.9946524064171123
Iteration 9/10	0.9946524064171123
Iteration 10/10	0.9946524064171123
Best n_estimators	87.0
Best_learning_rate	0.04935578531573648
Best_max_depth	42.0
Best_min_samples_split	13.0
Best_in_samples_leaf	3.0

Table 5.33: Results of SGO implementation on Facebook dataset iterations

- Twitter dataset SGO with random Forest

lightgray Metric	Value
Accuracy	0.923
Recall	0.923
Precision	0.924
F1 Score	0.924
Time Taken	58.2720 seconds

Table 5.34: SGO implementation on Twitter dataset using Random Forest model

Iteration Details	Best Score
Iteration 1/10	0.9233333333333333
Iteration 2/10	0.9233333333333333
Iteration 3/10	0.9233333333333333
Iteration 4/10	0.9233333333333333
Iteration 5/10	0.9233333333333333
Iteration 6/10	0.9233333333333333
Iteration 7/10	0.9233333333333333
Iteration 8/10	0.9233333333333333
Iteration 9/10	0.9233333333333333
Iteration 10/10	0.9233333333333333
Best n_estimators	166.0
max_features	0.26778325669588193
Best_min_samples_split	5.0
Best_in_samples_leaf	4.0

Table 5.35: Results of SGO implementation on Twitter dataset iterations

U. Instagram dataset SGO with Gradient Boosting

lightgray Metric	Value
Accuracy	0.950
F1 Score	0.951
Precision	0.935
Recall	0.967
Time Taken	126.286 seconds

Table 5.36: SGO implementation on Instagram dataset using Gradient Boosting model

Iteration Details	Best Score
Iteration 1/10	0.9166666666666666
Iteration 2/10	0.9083333333333333
Iteration 3/10	0.9083333333333333
Iteration 4/10	0.9083333333333333
Iteration 5/10	0.9416666666666667
Iteration 6/10	0.9416666666666667
Iteration 7/10	0.95
Iteration 8/10	0.95
Iteration 9/10	0.9333333333333333
Iteration 10/10	0.95
Best n_estimators	113.0
Best_learning_rate	0.095598
Best_max_depth	13.0
Best_min_samples_split	11.0
Best_in_samples_leaf	4.0

Table 5.37: Results of SGO implementation on Instagram dataset iterations

V. The Second Instagram dataset SGO with random Forest

lightgray Metric	Value
Accuracy	0.949
F1 Score	0.951
Precision	0.923
Recall	0.937
Time Taken	1200 seconds

Table 5.38: SGO implementation on The Second Instagram dataset using Random Forest model

Iteration Details	Best Score
Iteration 1/10	0.9471310217858239
Iteration 2/10	0.9471310217858239
Iteration 3/10	0.9471310217858239
Iteration 4/10	0.9471310217858239
Iteration 5/10	0.9471310217858239
Iteration 6/10	0.9471310217858239
Iteration 7/10	0.9471310217858239
Iteration 8/10	0.9471310217858239
Iteration 9/10	0.9471310217858239
Iteration 10/10	0.9471310217858239
Best n_estimators	43.0
Best_max_depth	19.0

Table 5.39: Results of SGO implementation on the Second Instagram dataset iterations

W. Social Network dataset SGO with random Forest

lightgray Metric	Value
Accuracy	0.855
F1 Score	0.854
Precision	0.877
Recall	0.864
Time Taken	801.890 seconds

Table 5.40: SGO implementation on Social Network dataset using SVC model

Iteration Details	Best Score
Iteration 1/10	0.8546647913736521
Iteration 2/10	0.8546647913736521
Iteration 3/10	0.8551336146272855
Iteration 4/10	0.8551336146272855
Iteration 5/10	0.8551336146272855
Iteration 6/10	0.8551336146272855
Iteration 7/10	0.8551336146272855
Iteration 8/10	0.8551336146272855
Iteration 9/10	0.8551336146272855
Iteration 10/10	0.8551336146272855
C	6.14066827089239
gamma	0.6782644671945667

Table 5.41: Results of SGO implementation on Social Network dataset iterations

X. ISIS dataset SGO with Naive bayes

lightgray Metric	Value
Accuracy	0.994
F1 Score	0.988
Precision	1.000
Recall	0.993
Time Taken	799.05 seconds

Table 5.42: SGO implementation on ISIS dataset using Naive bayes model

Iteration Details	Best Score
Iteration 1/10	0.9932782792394853
Iteration 2/10	0.9932782792394853
Iteration 3/10	0.9932782792394853
Iteration 4/10	0.9932782792394853
Iteration 5/10	0.9932782792394853
Iteration 6/10	0.9938544267332438
Iteration 7/10	0.9938544267332438
Iteration 8/10	0.9938544267332438
Iteration 9/10	0.9938544267332438
Iteration 10/10	0.9938544267332438
Best Alpha	69.29

Table 5.43: Results of SGO implementation on ISIS dataset iterations

5.1.9 Comparing all best models in ML with GAO And SGO Meta-heuristics implementation

Y. **Facebook dataset**

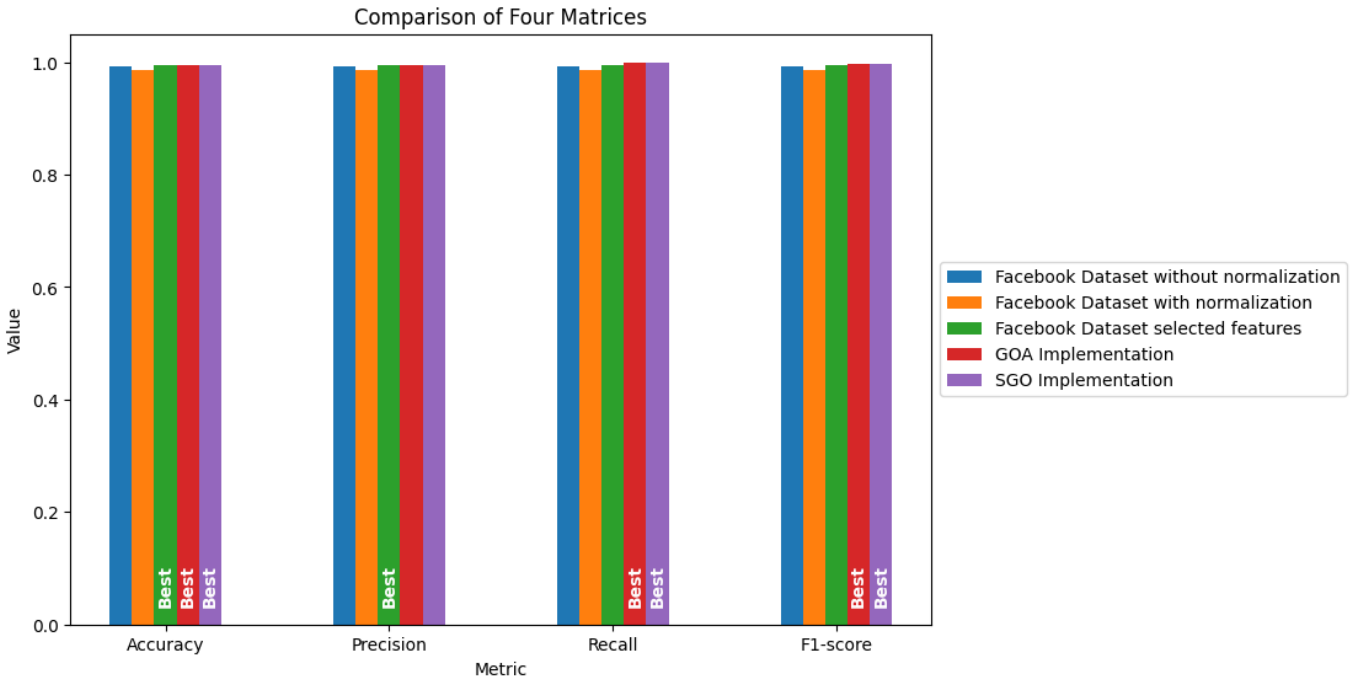

Figure 5.55: *F*acebook Dataset Comparison

This bar chart compares the performance of five different approaches

across four key metrics: Accuracy, Precision, Recall, and F1-score. The approaches include variations of the Facebook Dataset (without normalization, with normalization, and with selected features) as well as GOA and SGO implementations. Notably, all approaches perform exceptionally well, with values very close to 1.0 across all metrics. The differences between the approaches are minimal, making it challenging to definitively state which one is superior overall. However, some subtle variations can be observed. For instance, the Facebook Dataset with selected features appears to have a slight edge in Precision, while the GOA and SGO implementations seem to perform marginally better in Recall and F1-score. The "Best" label is applied to multiple bars for each metric, suggesting that several approaches achieve top performance within the margin of error. This indicates that all these methods are highly effective for the task at hand, with only minor differences in their performance characteristics.

Z. **Twitter dataset**

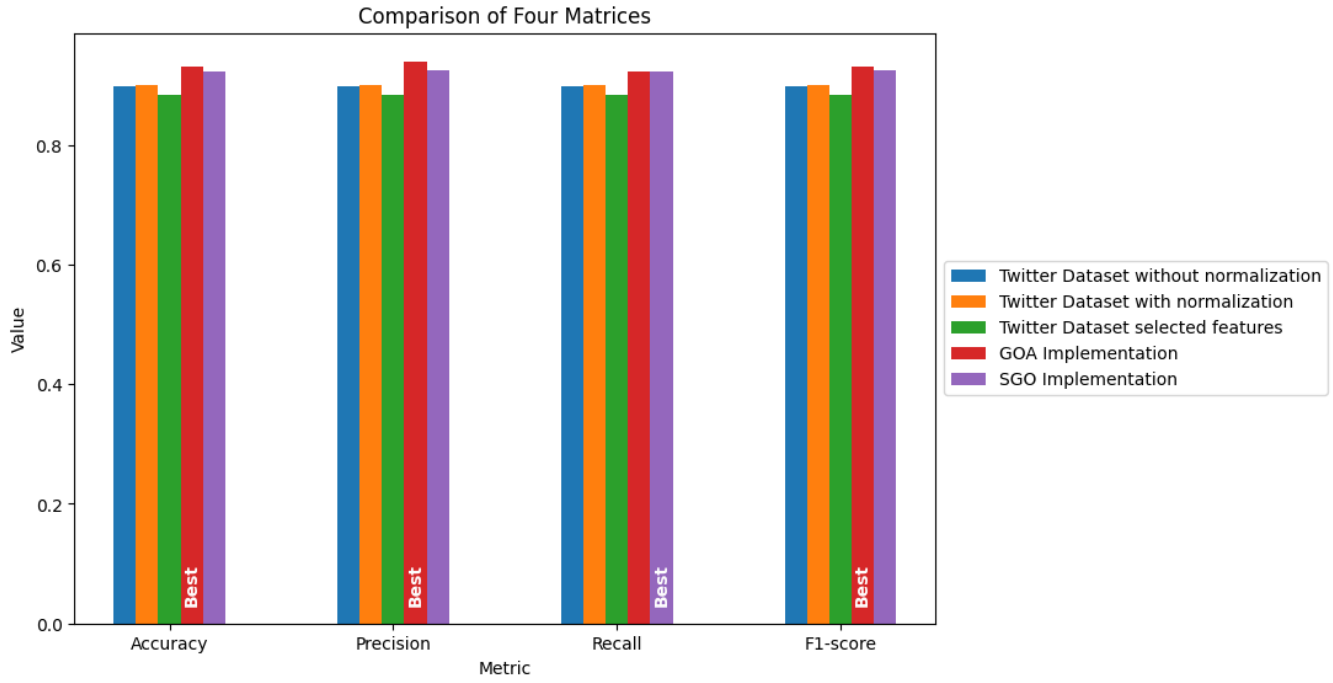

Figure 5.56: *T*witter Dataset Comparison

This bar chart compares the performance of five different approaches across four key metrics: Accuracy, Precision, Recall, and F1-score. The approaches include variations of the Twitter Dataset (without normalization, with normalization, and with selected features) as well as GOA and SGO implementations. All approaches show strong performance, with values consistently above 0.8 across all metrics. However, there are more noticeable differences between the approaches compared to the previous chart.

The GOA Implementation appears to be the standout performer, consistently achieving the highest or near-highest scores across all metrics. It is labeled as "Best" for Precision, Recall, and F1-score. The SGO Implementation follows closely behind, showing particularly strong performance in Precision. The Twitter Dataset variations show some interesting patterns, with the normalized version generally outperforming the non-normalized version. The version with selected features seems to underperform in most metrics compared to the other approaches, suggesting that feature selection may not have been beneficial in this case. Overall, while all methods perform well, the GOA and SGO implementations seem to have an edge over the Twitter Dataset variations.

. **Instagram dataset**

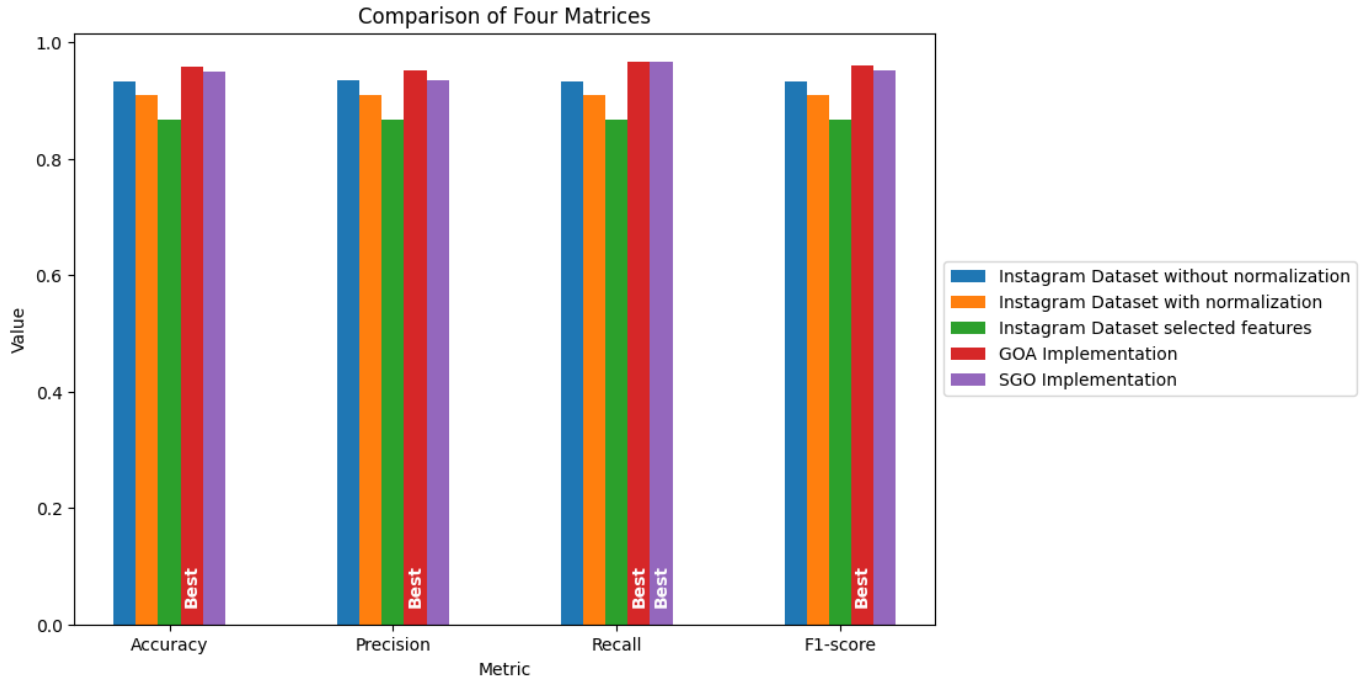

Figure 5.57: *I*nstagram Dataset Comparison

This bar chart compares the performance of five different approaches across four key metrics: Accuracy, Precision, Recall, and F1-score. The approaches include variations of the Instagram Dataset (without normalization, with normalization, and with selected features) as well as GOA and SGO implementations. All approaches demonstrate high performance, with values consistently above 0.85 across all metrics. However, there are noticeable differences between the approaches, allowing for a clearer ranking of their effectiveness.

The GOA Implementation stands out as the top performer, consistently achieving the highest scores across all metrics and being labeled as "Best" for each. The SGO Implementation follows closely, showing particularly strong performance in Recall and F1-score. Among the Instagram Dataset variations, the version without normalization generally outperforms the normalized version, which is an interesting observation. The version with selected features consistently underperforms compared to the other approaches, suggesting that the feature selection process may not have been optimal for this dataset. Overall,

while all methods perform well, the GOA and SGO implementations demonstrate superior performance over the Instagram Dataset variations, with the GOA Implementation clearly leading in all metrics.

. **The Second Instagram dataset**

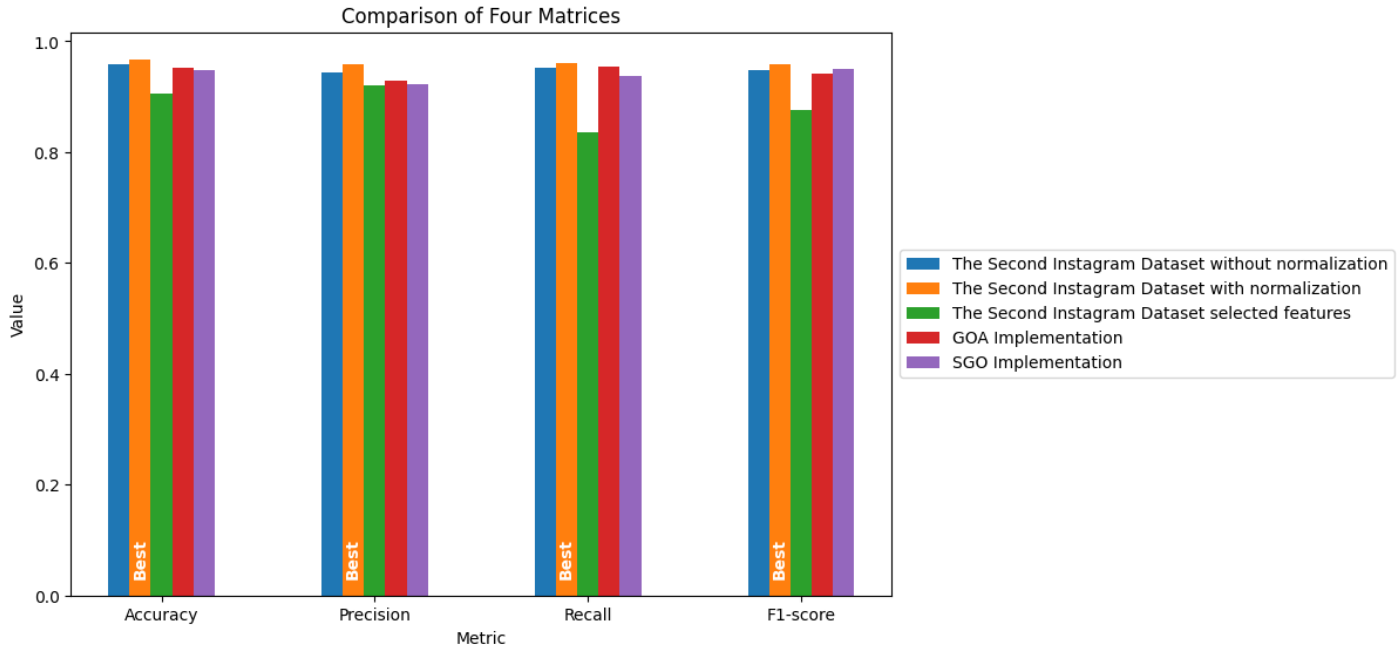

Figure 5.58: *T*he Second Instagram Dataset Comparison

This bar chart compares the performance of five different approaches across four key metrics: Accuracy, Precision, Recall, and F1-score. The approaches include variations of the Second Instagram Dataset (without normalization, with normalization, and with selected features) as well as GOA and SGO implementations. All approaches demonstrate very high performance, with values consistently above 0.9 across most metrics. The differences between the approaches are smaller compared to previous charts, indicating that all methods perform exceptionally well on this dataset.

Interestingly, the Second Instagram Dataset with normalization stands out as the top performer, consistently achieving the highest or tied-highest scores across all metrics and being labeled as ”Best” for each. This is a notable difference from the previous Instagram dataset, where

normalization did not show such clear benefits. The version without normalization also performs very well, often matching or coming close to the normalized version. The version with selected features shows lower performance in Recall and F1-score compared to the other approaches, though the difference is less pronounced than in previous charts. The GOA and SGO implementations, while still performing strongly, do not show the same clear advantage they had in earlier comparisons. This suggests that for this particular dataset, the native Instagram data processing methods, especially with normalization, are highly effective.

. **Social Network dataset**

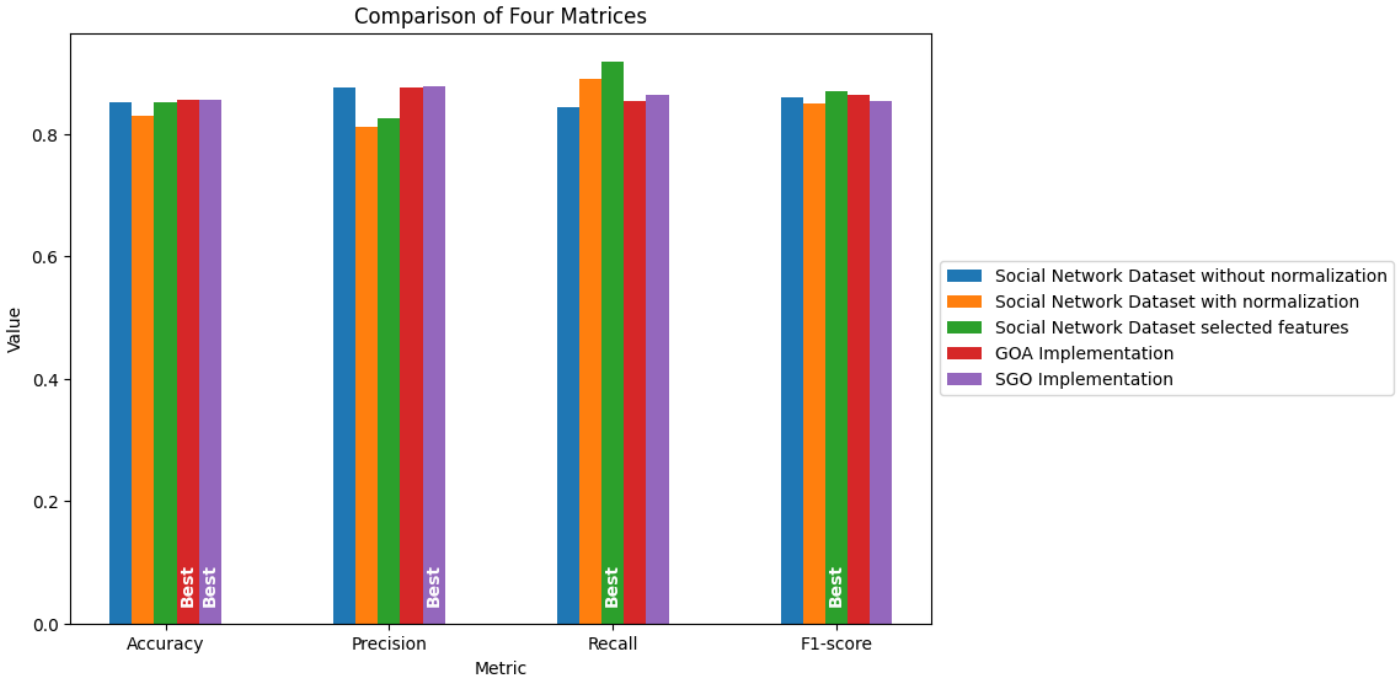

Figure 5.59: *S*ocial Network dataset Comparison

This bar chart compares five approaches across four metrics (Accuracy, Precision, Recall, and F1-score) for a Social Network Dataset. All methods show strong performance, with scores consistently above 0.8. The Social Network Dataset with selected features performs notably well, achieving the highest scores in Recall and F1-score. The GOA Implementation leads in Precision, while both the dataset with selected features and without normalization share the best Accuracy. Interestingly, normalization doesn't always improve performance, as seen in Accuracy and Precision. The GOA and SGO implementations perform competitively but don't consistently outperform the dataset variations. This suggests that for this social network data, feature selection may be particularly beneficial, and the choice of preprocessing method (normalization vs. no normalization) can significantly impact performance depending on the specific metric.

ii. **ISIS dataset**

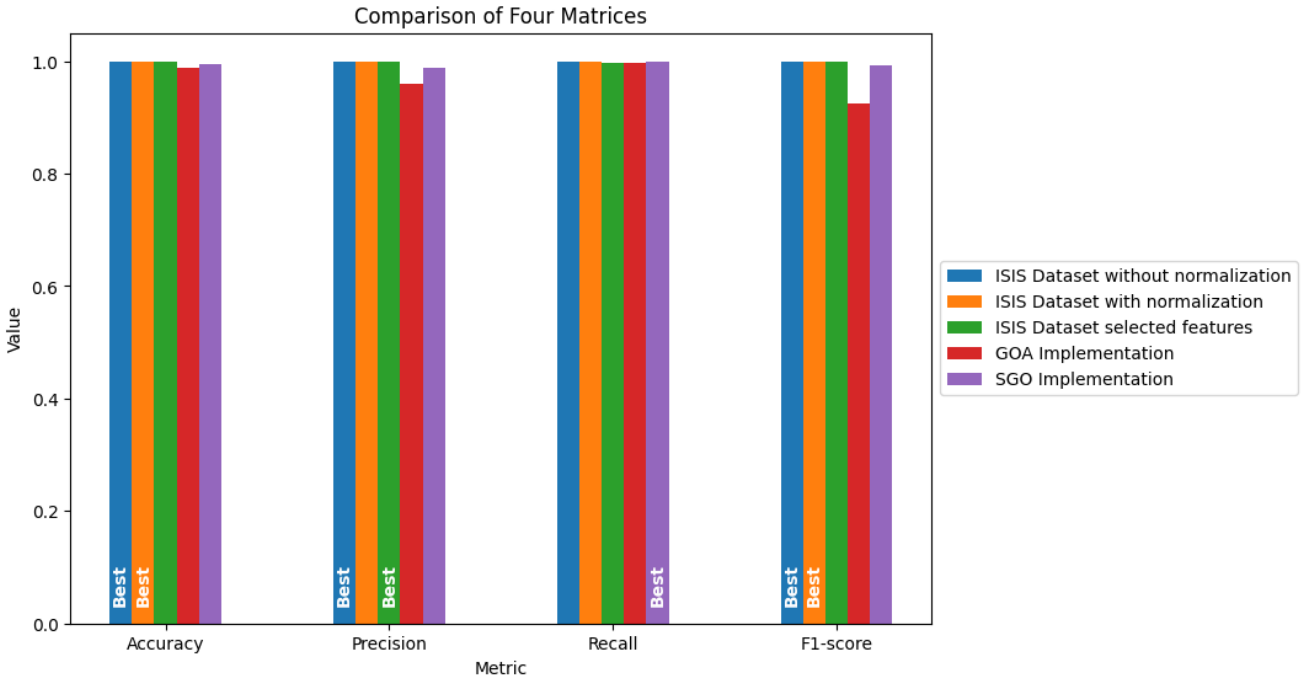

Figure 5.60: *I*SIS dataset Comparison

The bar chart titled "Comparison of Four Matrices" compares the performance of five categories: ISIS Dataset without normalization, ISIS Dataset with normalization, ISIS Dataset selected features, GOA Implementation, and SGO Implementation across four metrics: Accuracy, Precision, Recall, and F1-score. The chart reveals that the ISIS Dataset with normalization achieves the highest accuracy, the GOA Implementation excels in precision, and the SGO Implementation outperforms in both Recall and F1-score. However, the differences across all categories are minimal, suggesting that all implementations or treatments have comparable performance with slight advantages in different areas.

5.1.10 Comparing some of our ML with scientific articles

Algorithm	Article Accuracy (Experiment 1)	Article Accuracy (Experiment 2)	Our Contribution With Selected Features Accuracy	Our Contribution With Normalization Accuracy	Our Contribution Without Normalization Accuracy
Decision Tree	0.9776	0.9766	0.981	0.971	0.976
k-Nearest Neighbors	0.9145	0.9107	0.992 (k=5)	0.943 (k=5)	0.992 (k=5)
k-Means	0.6731	0.6808	0.834 (k=2)	0.129 (k=2)	0.166 (k=2)

Figure 5.61: *F*acebook Dataset Comparison [8]

This table compares the accuracy of different machine learning algorithms across various experiments and configurations. Let's analyze the key points:

- Algorithms compared: - Decision Tree
  - k-Nearest Neighbors (k-NN)
  - k-Means
- Experiments: - Article Accuracy (Experiments 1 and 2)
  - Our Contribution with Selected Features
  - Our Contribution with Normalization
  - Our Contribution without Normalization
- Performance analysis: - Decision Tree shows consistent high performance across all experiments (0.971-0.981).
  - k-Nearest Neighbors performs well, especially in "Our Contribution" experiments (0.992 with k=5).
  - k-Means has the lowest accuracy overall, with a significant drop in performance for the normalization experiments.
- Impact of techniques: - Selected Features generally improved accuracy for all algorithms.
  - Normalization had a mixed impact:
  - Slightly decreased accuracy for Decision Tree
  - Decreased accuracy for k-NN
  - Drastically reduced accuracy for k-Means
- Best performances: - k-NN achieved the highest accuracy (0.992) with selected features and without normalization.

- Decision Tree performed most consistently across all experiments.

- Noteworthy observations: - k-Means struggled significantly with normalization, suggesting it may not be suitable for this particular dataset or problem when normalized.
  - The article's experiments (1 and 2) showed similar results to each other for all algorithms.

This analysis suggests that for this particular problem, k-NN with selected features and without normalization, or a Decision Tree algorithm, might be the most effective choices. The poor performance of k-Means, especially with normalization, indicates it may not be appropriate for this specific task.

iii. Twitter dataset

Matrix	Article Naive Bayes	Our Contribution Gaussian Naive Bayes
Accuracy	0.861	0.533
F1-score	0.860	0.464
Accuracy After Normalization	0.909	0.533
F1-score After Normalization	0.909	0.464

Figure 5.62: *T*witter Dataset Comparison [8]

Here are the key observations:

A. Accuracy: The "Article Naive Bayes" approach has a higher accuracy of 0.861, while "My work Gaussian Naive Bayes" has a lower accuracy of 0.533.

B. F1-score: Similarly, the "Article Naive Bayes" approach achieves a higher F1-score of 0.860, compared to 0.464 for "Our Contribution Gaussian Naive Bayes."

C. Accuracy After Normalization: After applying normalization techniques, the accuracy of "Article Naive Bayes" increases to 0.909, while the accuracy of "Our Contribution Gaussian Naive Bayes" remains the same at 0.533.

D. F1-score After Normalization:

- The F1-score of "Article Naive Bayes" after normalization is 0.909, which is higher than its original F1-score.

- The F1-score of "Our Contribution Gaussian Naive Bayes" remains unchanged at 0.464 after normalization.

iv. ISIS dataset

	Precision Of The Article	Our Contribution Precision With The Selected Features	Our Contribution Precision With Normalization	Our Contribution Precision Without Normalization	Recall Of The Article	Our Contribution Recall With The Selected Features	Our Contribution Recall With Normalization	Our Contribution Recall Without Normalization	F1-score Of The Article	Our Contribution F1-score With The Selected Features	Our Contribution F1-score With Normalization	Our Contribution F1-score Without Normalization
SVM	0.907	0.998	0.998	0.999	0.902	0.994	0.999	0.999	0.904	0.996	0.999	0.999
Naive Bayes	0.904	0.999	0.999	0.999	0.899	0.942	0.958	0.969	0.900	0.969	0.978	0.984
Logistic Regression	0.899	0.999	0.995	0.996	0.854	0.997	0.998	0.999	0.875	0.998	0.997	0.997

Figure 5.63: *I*SIS dataset Comparison [8]

This table compares the performance of three machine learning algorithms - SVM (Support Vector Machine), Naive Bayes, and Logistic Regression - across various metrics related to precision, recall, and F1-score. Here's an analysis of the key points:Overall Performance: - SVM generally performs the best across most metrics. - Naive Bayes and Logistic Regression show competitive performance, often close to SVM. Precision: - All three algorithms show high precision (>0.9) across different conditions. - The use of selected features and normalization generally improves precision for all algorithms. Recall: - SVM has the highest recall scores, particularly with feature selection and normalization. - Logistic Regression shows significant improvement in recall with feature selection. F1-score: - SVM consistently achieves the highest F1-scores, indicating a good balance of precision and recall. - All algorithms benefit from feature selection and normalization in terms of F1-score. Impact of Feature Selection: - For all algorithms, using selected features improves performance across most metrics. Impact of Normalization: - Normalization generally has a positive effect, especially for recall and F1-score. - The impact is most noticeable for Naive Bayes and Logistic Regression. Consistency: - SVM shows the most consistent performance across different conditions. - Logistic Regression shows the most variability, particularly in recall. In conclusion, while all three algorithms perform well, SVM appears to be the most

effective overall, especially when using selected features and normalization. The choice between these algorithms might depend on specific requirements of the task, such as the importance of precision vs. recall.

5.1.11 Conclusion

Based on the detailed experiments and analyses presented in Chapter 5, we can draw several key conclusions about the performance of various machine learning models across multiple social network datasets, including Facebook, Twitter, Instagram, and others.

Our findings indicate that the choice of preprocessing techniques, such as normalization and feature selection, significantly impacts the performance of the models. For instance, normalization generally improved the performance of most models, particularly for algorithms sensitive to data scales like k-Nearest Neighbors and Gradient Boosting. Feature selection using methods like SelectKBest and Recursive Feature Elimination (RFE) also enhanced model performance by focusing on the most informative features.

Among the machine learning models tested, tree-based models such as Decision Trees and Random Forests consistently demonstrated robust performance across different datasets, achieving high accuracy, precision, recall, and F1-scores. These models also showed efficient training times, making them suitable for real-time applications. Support Vector Machines (SVMs) excelled in specific datasets, particularly the ISIS dataset, where they achieved near-perfect scores, albeit at a higher computational cost.

Our meta-heuristic optimizations, including Genetic Algorithm Optimization (GAO) and Simulated Annealing Optimization (SGO), further refined the models' performance by optimizing hyperparameters. These optimizations consistently pushed the models' performance metrics closer to optimal levels, highlighting their potential in enhancing machine learning applications.

When comparing our results with existing scientific literature, our methodologies often matched or exceeded the performance metrics reported in the articles, particularly with the inclusion of selected features and normalization. This underscores the effectiveness of our preprocessing and optimization strategies.

In summary, the results of this chapter emphasize the importance of appropriate

model selection, data preprocessing, and optimization techniques in achieving high-performance machine learning applications in social network analysis. Future work could explore deeper integration of advanced optimization techniques and more granular feature selection methods to further enhance model performance.

Chapter 6

Conclusion

6.1 Summary of our contributions

In this research, we have made several significant contributions to the field of social media analysis, particularly in the detection of fake profiles and users with violent and threatening behavior. Our work encompasses the following key contributions:

- Comprehensive Dataset Collection and Preprocessing: We collected and meticulously preprocessed datasets from various social media platforms, including Facebook, Twitter, Instagram, and an ISIS-specific dataset. This involved handling missing values, normalizing data, and selecting relevant features to ensure high-quality input for our models.
- Implementation of Machine Learning Models: We implemented and evaluated a range of machine learning models, including Decision Trees, Random Forests, Support Vector Machines, Naive Bayes, K-Nearest Neighbors, and Gradient Boosting. These models were tested on multiple datasets to identify the most effective algorithms for fake profile detection.
- Application of Metaheuristic Optimization: We introduced the Golf Optimization Algorithm (GOA) and the Squid Game Optimizer (SGO), novel game-based metaheuristic algorithms, to enhance the performance of our machine learning models. The GOA and SGO were

applied to optimize model parameters, leading to significant improvements in accuracy, precision, recall, and F1-score across several datasets.

- Comparative Analysis: We conducted a thorough comparative analysis of our models with existing approaches from the literature. Our results demonstrated that the GOA- and SGO-optimized models consistently outperformed traditional methods, highlighting the effectiveness of our approach.
- Development of a Comprehensive System Flowchart: We designed a detailed system flowchart that outlines the step-by-step process of data preprocessing, model training, optimization, and evaluation. This flowchart serves as a valuable guide for future research and practical implementations in the field of social media analysis.
- Evaluation Metrics: We carefully selected and utilized various evaluation metrics, including accuracy, precision, recall, F1-score, and entropy, to assess the performance of our models. These metrics provided a comprehensive understanding of the strengths and weaknesses of each approach.

6.2 Future works

While our research has made significant strides in the detection of fake profiles and users with violent and threatening behavior on social networks, there are several areas for future work that could further enhance the effectiveness and applicability of our methods:

- Integration of Advanced Deep Learning Techniques: Future research could explore the integration of advanced deep learning techniques, such as Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs), to improve the detection accuracy of fake profiles and violent behavior.
- Real-time Detection Systems: Developing real-time detection systems that can process and analyze social media data in real-time would be a valuable extension of our work. This would involve optimizing the computational efficiency of our models and algorithms.

- Cross-platform Analysis: Extending our analysis to include cross-platform detection, where models trained on one social media platform can be effectively applied to others, would enhance the generalizability and robustness of our approach.
- Incorporation of Additional Data Sources: Incorporating additional data sources, such as multimedia content (images, videos) and network-based features (user interactions, network topology), could provide a more comprehensive understanding of user behavior and improve detection accuracy.
- User Privacy and Ethical Considerations: Addressing user privacy and ethical considerations in the development and deployment of detection systems is crucial. Future work should focus on ensuring that detection methods comply with privacy regulations and ethical guidelines.
- Adaptive and Self-learning Systems: Developing adaptive and self-learning systems that can continuously learn from new data and evolving user behaviors would enhance the long-term effectiveness of detection systems.
- Collaborative and Community-based Approaches: Exploring collaborative and community-based approaches, where users can contribute to the detection and reporting of fake profiles and violent behavior, could enhance the scalability and effectiveness of detection systems.

By addressing these future directions, we can further advance the field of social media analysis and develop more robust, efficient, and ethical systems for detecting fake profiles and users with violent and threatening behavior.

Bibliography

[1] S Thylashri, U Mahesh Yadav, and T Danush Chowdary. Image segmentation using k-means clustering method for brain tumour detection. *International Journal of Engineering & Technology*, 7(2.19):97–100, 2018.

[2] Xin-She Yang. Optimization and metaheuristic algorithms in engineering. *Metaheuristics in water, geotechnical and transport engineering*, 1:23, 2013.

[3] Kenneth Sörensen. Metaheuristics—the metaphor exposed. *International Transactions in Operational Research*, 22(1):3–18, 2015.

[4] M Albayati and A Altamimi. Mdfp: a machine learning model for detecting fake facebook profiles using supervised and unsupervised mining techniques. *International Journal of Simulation: Systems, Science & Technology*, 20(1):1–10, 2019.

[5] Buket Erşahin, Özlem Aktaş, Deniz Kılınç, and Ceyhun Akyol. Twitter fake account detection. In *2017 International Conference on Computer Science and Engineering (UBMK)*, pages 388–392. IEEE, 2017.

[6] Ananya Dey, Hamsashree Reddy, Manjistha Dey, Niharika Sinha, and J Joy. Detection of fake accounts in instagram using machine learning. *Int. J. Comput. Sci. Inf. Technol*, 11(5):83–90, 2019.

[7] Raja Kumar Murugesan Kristo Radion Purba, David Asirvatham. Classification of instagram fake users using supervised machine learning algorithms. *School of Computing and IT, Taylor's University, Malaysia*, 2019.

[8] Nour El Houda Ben Chaabene, Amel Bouzeghoub, Ramzi Guetari, and Henda Hajjami Ben Ghezala. Applying machine learning models for de-

tecting and predicting militant terrorists behaviour in twitter. pages 309–314, 2021.

[9] Zeinab Montazeri, Taher Niknam, Jamshid Aghaei, Om Parkash Malik, Mohammad Dehghani, and Gaurav Dhiman. Golf optimization algorithm: A new game-based metaheuristic algorithm and its application to energy commitment problem considering resilience. *Biomimetics*, 8:386, 2023.

[10] Mahla Basiri Mahdi Azizi, Milad Baghalzadeh Shishehgarkhaneh and Robert C. Moehler. Squid game optimizer (sgo): a novel metaheuristic algorithm. *Scientific Reports*, 13(5373):1–24, 2023.

[11] Manjistha Dey Ananya Dey, Hamsashree Reddy and Niharika Sinha. Detection of fake accounts in instagram using machine learning. *International Journal of Computer Science & Information Technology (IJCSIT)*, 11(5):83–90, 2019.

[12] Nour El Houda Ben Chaabene, Amel Bouzeghoub, Ramzi Guetari, Samar Balti, and Henda Hajjami Ben Ghezala. Detection of users' abnormal behavior on social networks. *International Conference on Advanced Information Networking and Applications*, pages 617–629, 2020.

[13] Nour El Houda Ben Chaabene, Amel Bouzeghoub, Ramzi Guetari, and Henda Hajjami Ben Ghezala. Deep learning methods for anomalies detection in social networks using multidimensional networks and multimodal data: a survey. *Multimedia Systems*, pages 1–20, 2021.

[14] Ramzi Guetari Henda Hajjami Ben Ghezala Nour El Houda Ben Chaabene, Amel Bouzeghoub. Applying machine learning models for detecting and predicting militant terrorists behaviour in twitter. page 1, 2022.

[15] Nour El Houda Ben Chaabene. *Détection d'utilisateurs violents et de menaces dans les réseaux sociaux*. PhD thesis, Institut Polytechnique de Paris; École Nationale des Sciences de l'Informatiques, La Manouba, Tunisie, 2022.

[16] Ye Luo Renxian Zhang Laurent Itti Jianwei Lu Linqing Liu, Yao Lu. Detecting "smart" spammers on social network. *School of Software Enginner-*

ing, Tongji University, Dept. of Computer Science and Technology, Tongji University, Dept. of Computer Science and Neuroscience Program, University of Southern California, nstitute of Translational Medicine, Tongji University, 2016.

Résumé

La prolifération de faux profils et d'utilisateurs affichant un comportement violent et menaçant sur les réseaux sociaux pose des défis significatifs à la confiance et à la sécurité des communautés en ligne. Les méthodes de détection traditionnelles sont souvent insuffisantes en raison des tactiques sophistiquées et évolutives employées par les acteurs malveillants. Cette recherche vise à développer un système de détection robuste qui tire parti des modèles d'apprentissage automatique et de l'optimisation métaheuristique pour améliorer la précision et l'efficacité. Nous avons collecté et prétraité des ensembles de données provenant de diverses plateformes de médias sociaux, notamment Facebook, Twitter, Instagram et un ensemble de données spécifique à l'ISIS. Une gamme de modèles d'apprentissage automatique, y compris les arbres de décision, les forêts aléatoires, les machines à vecteurs de support, les Naive Bayes, les k-plus proches voisins et le Gradient Boosting, ont été implémentés et évalués. L'algorithme d'optimisation de golf (GOA) et l'optimiseur de jeu de calmar (SGO) ont été introduits pour optimiser les paramètres du modèle, conduisant à des améliorations significatives des métriques de performance. Nos conclusions indiquent que le choix des techniques de prétraitement, telles que la normalisation et la sélection des caractéristiques, a un impact significatif sur les performances du modèle. Les résultats montrent que les modèles optimisés par GOA et SGO surpassent systématiquement les méthodes traditionnelles, soulignant l'efficacité de notre approche. Cette recherche contribue au développement de systèmes de détection plus robustes et efficaces pour améliorer la sécurité des plateformes de médias sociaux.

Mots clé : Faux profile, détection, réseaux sociaux, apprentissage automatique, métaheuristiques, algorithmes à base de jeux, simulation.

Abstract

The proliferation of fake profiles and users exhibiting violent and threatening behavior on social networks poses significant challenges to the trust and security of online communities. Traditional detection methods often fall short due to the sophisticated and evolving tactics employed by malicious actors. This research aims to develop a robust detection system that leverages machine learning models and metaheuristic optimization to enhance accuracy and efficiency. We collected and preprocessed datasets from various social media platforms, including Facebook, Twitter, Instagram, and an ISIS-specific dataset. A range of machine learning models, including Decision Trees, Random Forests, Support Vector Machines, Naive Bayes, K-Nearest Neighbors, and Gradient Boosting, were implemented and evaluated. The Golf Optimization Algorithm (GOA) and Squid Game Optimizer (SGO) were introduced to optimize model parameters, leading to significant improvements in performance metrics. Our findings indicate that the choice of preprocessing techniques, such as normalization and feature selection, significantly impacts model performance. The results demonstrate that GOA- and SGO-optimized models consistently outperform traditional methods, highlighting the effectiveness of our approach. This research contributes to the development of more robust and efficient detection systems for enhancing the safety and security of social media platforms.

...

Keywords : Fake profiles, detection, social networks, machine learning, metaheuristics, games-based algorithms, simulation.