Dissertation on

# Prediction of Student Performance Using Supervised Machine Learning Algorithms

*Thesis submitted towards partial fulfillment of the requirements for the degree of*

**Master of Technology in IT (Courseware Engineering)**


*Submitted by*

**TRISHITA KALSA**

EXAMINATION ROLL NO - M4CWE24010

UNIVERSITY REGISTRATION NO - 135998 of 2016-17

*Under the guidance of*

DR. SASWATI MUKHERJEE

**School of Education Technology**

Jadavpur University

Course affiliated to

**Faculty of Engineering and Technology**

**Jadavpur University**

Kolkata-700032

India


2024

**M.Tech IT (Courseware Engineering)**
**Course affiliated to**
**Faculty of Engineering and Technology**
**Jadavpur University**
**Kolkata, India**

---

## CERTIFICATE OF RECOMMENDATION

This is to certify that the thesis entitled **"Prediction of Student Performance using Supervised Machine Learning Algorithms"** is a bona fide work carried out by **TRISHITA KALSA** under our supervision and guidance for partial fulfillment of the requirements for the degree of **Master of Technology in IT (Courseware Engineering)** in **School of Education Technology**, during the academic session 2023-2024

---

**Dr. SASWATI MUKHERJEE**

**SUPERVISOR**
School of Education Technology
Jadavpur University,
Kolkata-700032

---

**DIRECTOR**
School of Education Technology
Jadavpur University,
Kolkata-700032

---

**DEAN – FISLM**
Jadavpur University,
Kolkata-700 032

**M.Tech IT (Courseware Engineering)**
**Course affiliated to**
**Faculty of Engineering and Technology**
**Jadavpur University**
**Kolkata, India**

## CERTIFICATE OF APPROVAL**

This foregoing thesis is hereby approved as a credible study of an engineering subject carried out and presented satisfactorily to warrant its acceptance as a prerequisite to the degree for which it has been submitted. Itis understood that by this approval the undersigned do not endorse or approve any statement made or opinion expressed or conclusion drawn therein but approve the thesis only for the purpose for which it has been submitted.

Committee of final examination ----------

For evaluation of the Thesis ----------

---

---

---

**Only in case the thesis is approved.

**DECLARATION OF ORIGINALITY AND COMPLIANCE OF ACADEMIC RULES**

I hereby declare that this thesis contains a literature survey and original research work by the undersigned candidate, as part of her Master of Technology in IT (Courseware Engineering) studies.

All information in this document has been obtained and presented by academic rules and ethical conduct.

As required by this rule and conduct, I also declare that I have fully cited and referenced all materials and results that are not original to this work.

| | | |
|---|---|---|
| NAME | : | TRISHITA KALSA |
| EXAMINATION ROLL NUMBER | : | M4CWE24010 |
| REGISTRATION NUMBER | : | 135998 of 2016-17 |
| THESIS TITLE | : | Prediction of Student Performance Using Supervised Machine Learning Algorithm. |

.

SIGNATURE: DATE:

# Acknowledgment

I feel fortunate while presenting this dissertation at the **School of Education Technology, Jadavpur University, Kolkata,** in the partial fulfillment of the requirement for the degree of **Master of Technology in IT (Courseware Engineering).**

I hereby take this opportunity to show my gratitude to my mentor**, Dr. Saswati Mukherjee**, who has guided and helped me with all possible suggestions, support, aspiring advice, and constructive criticism along with illuminating views on different issues of this dissertation which helped me throughout my work.

I would like to warmly thank **Prof. Dr. Matangini Chattopadhyay, Director of the School of Education Technology,** for her timely encouragement, support, and advice. I would also like to thank **Mr. Joydeep Mukherjee** for his constant support during my entire course of work. My thanks and appreciation go to my classmates from M. Tech in Information Technology (Courseware Engineering) and Master in Multimedia Development.

I wish to thank all the departmental support staff and everyone else who has contributed to this dissertation. Finally, I would like to express my special gratitude to my parents who have invariably sacrificed and supported me and helped me achieve this height.

**Date:**

**Place: Kolkata**

**Trishita Kalsa**
**Examination Roll No: M4CWE24010**
**M.Tech in IT (Courseware Engineering)**
**School of Education Technology**
**Jadavpur University**
**Kolkata: 700032**

# Contents

# Executive Summary

Student dropout rates have become a major concern in educational institutions globally. Various factors contribute to this increasing trend, including academic performance, economic status, personal circumstances, and interpersonal challenges. Early identification of at-risk students is crucial for implementing preventive measures and enhancing educational outcomes.

This dissertation addresses the issue of student dropout by leveraging advanced machine learning techniques to predict dropout likelihood, thereby aiding institutions in timely interventions. A Random Forest algorithm is used due to its robust performance, strong predictive capabilities, and ability to handle large datasets with complex interactions. To enhance the model's performance further, hyperparameters are tuned using the Whale Optimization Algorithm (WOA). This optimization technique mimics the social behavior of humpback whales, enabling efficient exploration and exploitation of the search space. The WOA helps identify the optimal hyperparameter settings that lead to improved model accuracy and generalization.

The datasets for this study are sourced from Kaggle, providing a comprehensive collection of student-related features necessary for modeling. The machine learning algorithm processes these datasets to identify the patterns and correlations associated with dropout risks. The results of this study demonstrate significant improvements in prediction accuracy, highlighting the potential of such models in practical applications. Educational institutions can use these insights to implement targetedinterventions, reducing dropout rates and improving student retention.

# 1. Introduction

## 1.1 Overview

This section provides an overview of the study, its objectives, and the structure of the dissertation. It begins by discussing the significance of predicting student dropout rates and the importance of early intervention. The chapter then outlines the research methodology, including the selection of machine learning algorithms and datasets. Finally, it presents the expected outcomes and the potential impact of the study on educational institutions.

Student dropout is a significant global concern that poses challenges for educational institutions worldwide, posing significant challenges to both individual students and society. In recent years dropout rates have increased which are heavily influenced by various factors including academic difficulties, socioeconomic conditions, lack of engagement, and personal circumstances. These factors have led to increased dropout rates, especially in higher education, where students often struggle to balance academic demands with personal responsibilities [1]. Almost 40% of students in the United States fail to complete their degree programs within six years, according tothe National Center for Education Statistics (NCES). In Europe, the dropout rate isalso concerning, with studies indicating that about 20% of students leave higher education without obtaining a degree. Globally, UNESCO reports suggest that over 258 million children and youth were out of school in 2018, highlighting the widespread nature of this issue. The consequences of dropping out are profound, affecting

students' future employment opportunities and earning potential, leading to a loss of human capital for society.

While dropping out might sometimes be a necessary decision due to these challenges, it often results in negative outcomes such as reduced lifetime earnings, economic instability, and limited career opportunities[2]. For instance, data from the NCES shows that students from low-income backgrounds are 2.4 times more likely to drop out of college than their peers from higher-income families. Some students benefit from dropping out by pursuing alternative education paths or entrepreneurial ventures that align better with their interests and skills. The availability of comprehensive datasets on student demographics, academic performance, and socioeconomic status has enabled researchers to use machine learning for predicting dropout risks. Machine learning algorithms can identify patterns and correlations in these datasets that may not be apparent through traditional analysis methods. By employing techniques such as Random Forests, Support Vector Machines, and Neural Networks, machine learning has demonstrated significant potential in predicting student dropout [3]. These predictive models provide educational institutions with valuable insights. As a result, schools and universities can potentially reduce dropout rates and improve overall student retention and success

## 1.2 Problem Statement

Student Performance Prediction using a supervised machine learning algorithm.

## 1.3 Objective

The objectives of the proposed work are:

1. To develop a Predictive Machine Learning Algorithm for Student Dropout Prediction.

2. To identify and Select the Most Relevant Features Contributing to student

3. To optimize the Hyperparameters of the Machine Learning algorithm.

# 2. Background Concepts

In this section, different machine learning and deep learning algorithms and some statistical regression techniques are discussed. These methods have been used in the present work, which will be discussed in the section on methodology. The list of algorithms is given below.

## 2.1 Genetic Algorithm

A Genetic Algorithm (GA) is a search heuristic inspired by genetics and natural selection principles. It simulates natural evolution to tackle complex optimization andsearch problems[4]. By operating on a population of potential solutions, GA iteratively evolves these solutions through crossover, mutation, and selection techniques, as shown in Fig 2.1 to explore vast solution spaces that traditional methods may noteffectively address. The primary objective is identifying the optimal or near-optimalsolution for a given problem.

### Initialization

The population size, $N = 10$, defines the number of individuals (solutions) in the initial population. This population undergoes iterative transformations to explore

the solution space.

## Selection

Although the specific selection method is not used here, commonly used methods in GAs include tournament selection, roulette wheel selection, and rank selection. These methods typically select individuals based on their fitness, with a higher likelihood of choosing individuals with superior fitness values.

## Fitness Evaluation

The fitness of each individual is evaluated using the fitness function, given in Eq 2.1:

$$\text{fitness}(x) = \alpha \times \text{Error Rate}(x) + \beta \times \text{Number of Features}(x) \quad (2.1)$$

where, $\alpha = 0.99$ and $\beta = 1 - \alpha = 0.01$. This function assesses each solution based on its error rate and the number of features selected, balancing accuracy with model simplicity.

## Selection Probability

The discovery rate $P_a = 0.25$ acts as an implicit selection mechanism. This rate influences how frequently sub-optimal solutions are replaced or explored, effectively acting as a selection probability.

## Crossover

The crossover operation, involves combining parts of two parent solutions to produce offspring. Although the specific logic is not implemented in the provided code, the crossover rate $CR = 0.8$ suggests that 80% of the individuals would typically undergo crossover, contributing to the creation of new, potentially superior solutions.

## Mutation

Mutation is crucial for maintaining genetic diversity within the population by introducing random changes to individuals. Although the mutation logic is not explicitly implemented, the mutation rate $\mathrm{MR} = 0.01$ implies that 1% of the population would typically undergo mutation, helping to avoid local minima by exploring new areas ofthe solution space.

## Replacement

The replacement strategy, which determines how new individuals are incorporated into the population, is not explicitly defined. Typically, this would involve replacing the least fit individuals with new offspring, ensuring that the population evolves towards better solutions over time.

## Termination

The algorithm terminates when it reaches a maximum number of iterations $\mathrm{T} = 100$. This condition ensures that the GA halts after a predefined number of generations, regardless of whether the optimal solution has been found, balancing computational effort with the search for a solution

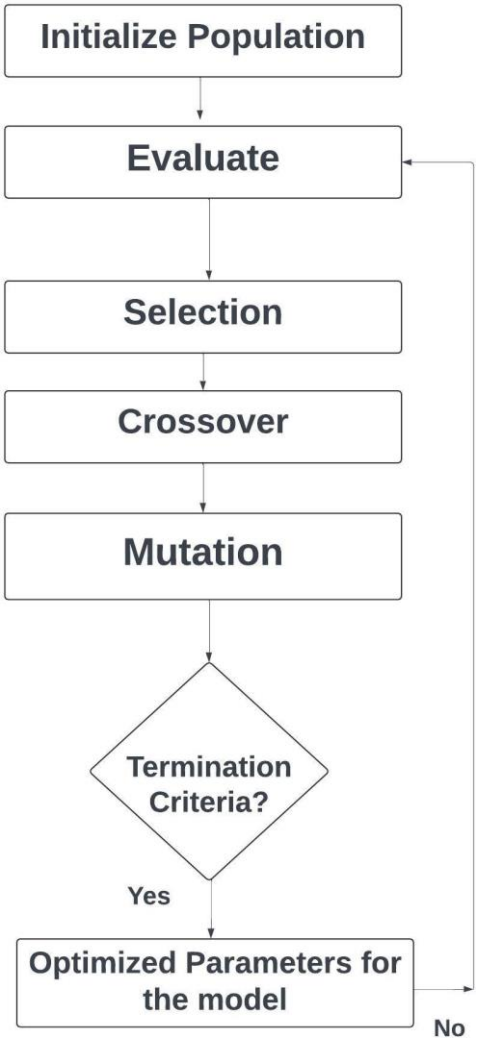


Figure 2.1: Genetic Algorithm architecture

## 2.2 Cuckoo Search Algorithm

This work provides a comprehensive overview of the Cuckoo Search (CS) algorithm, a nature-inspired optimization method mimicking the brood parasitism of certain

cuckoo species [5]. It explores the algorithm's mathematical formulation, theoretical foundation, and practical applications for solving optimization problems. As a population-based technique, CS iteratively evolves potential solutions, guided by the nesting behaviors of cuckoos that lay their eggs in the nests of other birds, relying on host species for incubation given below in Fig 2.2

### 2.2.1 Key Components

#### Cuckoo Behavior

Update Rule, eq 2.2:

$$x(t + 1) = x(t) + \alpha \tag{2.2}$$

#### Step Size Scaling Factor ($\alpha$)

The default value used in the implementation is 1, as indicated by the alpha parameter in the get opts function.

#### Host Bird Nests

Represent potential solutions to the optimization problem. In the context of the implementation, each nest corresponds to a particle's position in the search space.

#### Eggs in a Nest

Represent the set of solutions within each nest. This is captured in the binary conversion process where the positions are converted to binary to determine selected features.

### Levy Flight

*Levy Distribution Exponent (β)*: The default value used in the implementation is 1.5, as defined in the jfs function. Refer to Equation 2.3

$$\text{Step} = \frac{u}{|v|^{1/\beta}} \tag{2.3}$$

This allows for effective exploration of the search space.

## 2.2.2 Algorithm Process

### Initialization

*Population Size (N)*: The number of nests is set to 10 (N = 10).
*Maximum Iterations (T)*: The algorithm is configured to run for a maximum of 100 iterations (T = 100).
*Discovery Rate (Pa)*: The rate of discovery is set to 0.25, indicating the fraction of worse nests that are discovered in each iteration.

### Generate New Solutions

*Levy Flight Update*, Refer to Eq 2.4.

$$x_{t+1} = x_t + \alpha \cdot L_{\text{Levy}}(\lambda) \tag{2.4}$$

*Step Size Scaling Factor (α)*: Again, the default value is 1.

### Evaluation

The fitness of the new solutions is evaluated using the defined Fun function, which computes the cost based on the error rate and the number of selected features.

### Replacement

The best solutions replace a portion of the worst solutions in the population. This is achieved by selecting solutions based on their fitness values.

### Termination

The algorithm iterates until either an acceptable fitness level is reached or the maximum number of iterations (T = 100) is completed.

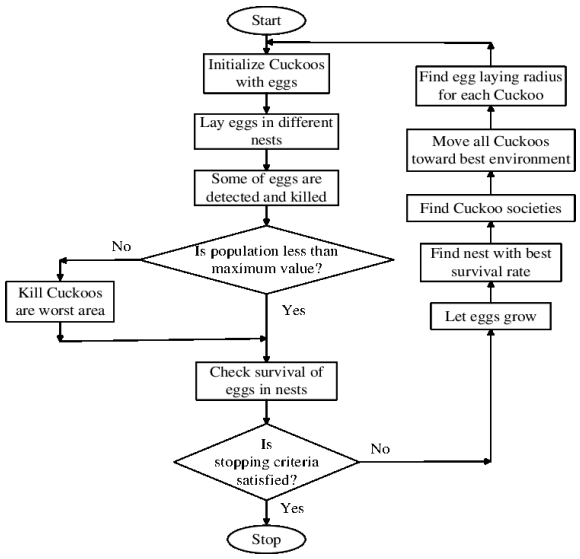


Figure 2.2: Flowchart of Cuckoo Search Algorithm

## 2.3 Whale Optimization Algorithm (WOA)

Humpback whales' hunting behaviors are based on the idea of creating a bubble net to encircle and trap their prey [6]. There are three main ways that humpback whales hunt: encircling their prey, searching for prey, and using the bubble-net attacking mechanism.

## Encircling the Prey

Humpback whales often work together in groups to encircle their prey, creating a barrier or circle it to prevent escape. Similarly, in WOA, candidate solutions are iteratively adjusted and refined to encircle the optimal solution.

$$\vec{D} = .\vec{C} \cdot \vec{X}^{*}(t) - \vec{X}(t). \quad (2.5)$$

$$\vec{X}(t+1) = \vec{X}^{*}(t) - \vec{A} \cdot \vec{D} \quad (2.6)$$

$\vec{X}^{*}(t)$ denotes the best search agent of the swarm at iteration t, $\vec{X}(t)$ represents a candidate search agent of the swarm, and $\vec{D}$ denotes the distance between two agents.

$$\vec{A} = 2 \cdot \vec{a} \cdot \vec{r} - \vec{a} \quad (2.7)$$

$$\vec{C} = 2 \cdot \vec{r} \quad (2.8)$$

where **a** is a vector of 2 that is linearly decreased to 0 over iterations, and **r** is a vector of randomly generated numbers in [0, 1] based on Gaussian distribution asgiven in Equations (2.7) and (2.8).

## Searching for Prey

Humpback whales search for prey in the surrounding environment before encircling the prey. They use various sensory cues to detect the presence of prey. In WOA, this searching behavior is reflected in the exploration phase of the algorithm, where candidate solutions explore the solution space to find regions with potentially better solutions.

$$\vec{D}' = \left|\vec{X}^*(t) - \vec{X}(t)\right| . \tag{2.9}$$

$$\vec{X}(t+1) = \vec{D}' \cdot e^{bt} \cdot \cos(2\pi t) + \vec{X}^*(t) \tag{2.10}$$

## Bubble-net Attacking Mechanism

The bubble-net technique involves humpback whales blowing bubbles in a circular pattern underwater to create a barrier, driving the prey toward the surface where they can easily catch it. This mechanism is analogous to the exploitation phase of WOA, where candidate solutions converge toward the optimal solution identified during the exploration phase [**?**].

Here, $b$ is a constant for deciding the scale of the spiral, and $l$ is a random number uniformly distributed in the interval $[-1, 1]$

## Algorithm

1. Initialize the swarm $\vec{X}_i$ for $i = 1, 2, \ldots, n$.
2. Evaluate the fitness value of all incumbents.
3. Set the best incumbent as $\vec{X}^*(t)$.
4. While the termination criterion is not met do:
   (a) For each incumbent:
      i. Update $a$, $\vec{A}$, $\vec{C}$, and $p$.
      ii. If $p < 0.5$:
         A. If $\left|\vec{A}\right| < 1$:

$$\vec{D} = \left|\vec{C} \cdot \vec{X}^*(t) - \vec{X}(t)\right|.$$

$$\vec{X}(t+1) = \vec{X}^*(t) - \vec{A} \cdot \vec{D}$$

B. Else:

$$\vec{D} = .\vec{C} \cdot \vec{X}_{rand}(t) - \vec{X}(t).$$

$$\vec{X}(t+1) = \vec{X}_{rand}(t) - \vec{A} \cdot \vec{D}$$

iii. Else:

$$\vec{D}' = \vec{X}^*(t) - \vec{X}(t) \ .$$

$$\vec{X}(t+1) = \vec{D}' \cdot e^b \cdot \cos(2\pi t) + \vec{X}^*(t)$$

(b) Revise the incumbent if it exceeds the boundary.

(c) Calculate the fitness value for all incumbents.

5. End While

6. Return $\vec{X}^*(t)$.

## 2.4 Grid Search for Hyperparameter Optimization in Random Forest Classification

### Initialization

Hyperparameter optimization is a very important step in machine learning model development, especially for models like Random Forests. The goal of this processis to find the best combination of hyperparameters that results in optimal model performance [7]. Grid Search is a systematic method to achieve this by exhaustively searching over a predefined hyperparameter grid, as shown in Figure 2.3

### Defining the Parameter Grid

In Grid Search for optimizing a Random Forest Classifier, the parameter grid is defined to explore key hyperparameters and their values. Important parameters include n estimators, which indicate the number of trees in the forest with typicalchoices

such as $\{50, 100, 200\}$. The max depth parameter determines the maximum

depth of each tree, with options like {None, 10, 20, 30}, where None allows the tree to expand until leaves are pure or a minimum sample condition is met. Addi- tionally, min samples_split specifies the minimum number of samples required to split an internal node, commonly set to {2, 5, 10} to mitigate overfitting. Lastly, min samples leaf indicates the minimum number of samples in a leaf node, with values like {1, 2, 4} to enhance model smoothness, particularly in regression tasks.

### Instantiating the Random Forest Classifier

The Random Forest Classifier is instantiated with a specified random seed to ensure reproducibility of results during the Grid Search process. This is achieved with rf clf = RandomForestClassifier(random state=42), where random state=42 ensures consistent outcomes across different runs, which is essential for model validation and comparison.

### Instantiating GridSearchCV

GridSearchCV is employed for hyperparameter optimization by systematically test- ing all combinations of hyperparameters defined in the grid and evaluating model performance through cross-validation. With cv=5, the data is divided into five sub- sets, where the model trains on four and tests on the fifth, repeating this process for each subset. The average of these five results constitutes the cross-validation score. The parameter n jobs=-1 allows the use of all available CPU cores to speed up computations, while verbose=2 provides detailed logs of the process, enhancing transparency during execution.

## Fitting the Grid Search

The Grid Search process begins by fitting the GridSearchCV object to the training data, where the model assesses each hyper parameter combination and identifies the one with the highest cross-validation score. This step is computationally intensive, particularly with large datasets and extensive parameter grids, but is essential for determining the optimal model configuration.

Upon completing the Grid Search, the optimal hyper parameters are identified based on the highest cross-validation score. The output includes the best parameters and the mean cross-validation score, reflecting the average model performance across folds. This best model is then used for predictions on the validation or test set, evaluated using metrics like accuracy and the confusion matrix

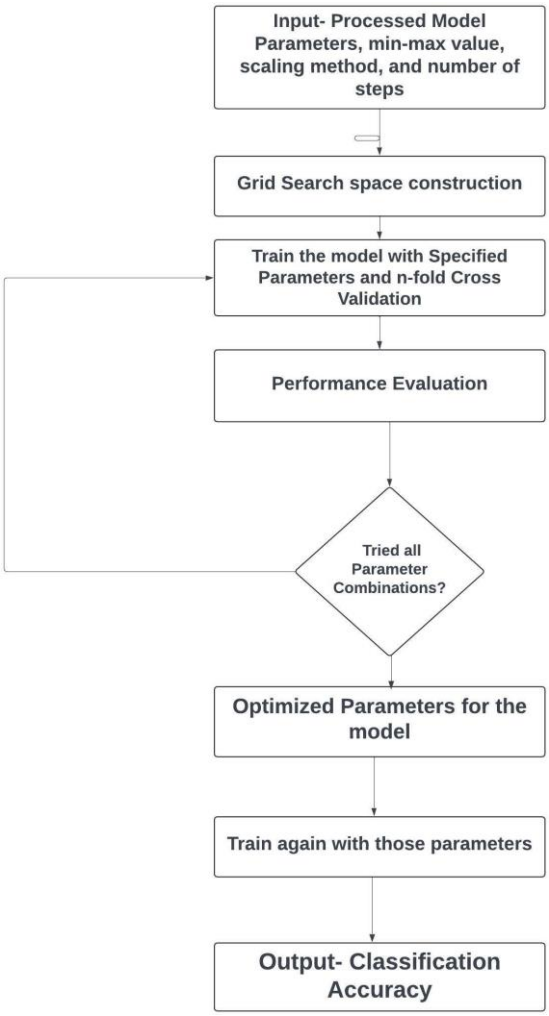


Figure 2.3: Flowchart of Grid Search algorithm

## 2.5 Random Forest (RF) Algorithm

The Random Forest (RF) algorithm is a powerful and versatile ensemble learning method widely utilized in both classification and regression tasks within supervised machine learning. It is particularly known for its ability to handle large datasets with higher dimensional and complex relationships. The strength of Random Forestlies in its construction and aggregation of multiple decision trees, each trained on different subsets of the data, resulting in a robust and accurate predictive model.

### Bootstrap Sampling (Bagging)

The Random Forest algorithm begins by generating several bootstrap samples from the original training dataset. Each bootstrap sample is created by randomly selecting observations with replacements, meaning some observations may appear multiple times in a sample, while others may be omitted. This approach, known as bagging (Bootstrap Aggregating), introduces diversity among the individual trees, preventing over fitting and enhancing the model's applicability

### Decision Tree Construction

For each bootstrap sample, an unpruned decision tree is grown. Unlike traditional decision trees that consider all features at each split, Random Forest introduces randomness by selecting a random subset of features to determine the best split at each node. This random feature selection reduces the correlation between individual trees, ensuring that the ensemble model captures a broader range of patterns and relationships within the data.

### Node Splitting

The splitting process in each decision tree is recursive, continuing until all leaves are pure (i.e., contain only a single class in classification tasks) or until another stopping criterion is met, such as a minimum number of samples required to split a node, as shown in Fig 2.4 By growing trees to their full depth without pruning, Random

Forest maintains a diverse collection of decision paths, contributing to the overall model's robustness.

## Prediction Aggregation

Once all trees in the forest are constructed, the predictions from these trees are aggregated to produce the final output. In classification tasks, each tree votes for a class label, and the class with the majority votes becomes the model's final prediction. For regression tasks, the predictions from all trees are averaged to yield the final result. This aggregation process ensures that the final prediction benefits from the collective wisdom of the trees, thereby reducing variance and improving accuracy.

## Model Evaluation

The performance of the Random Forest model is typically evaluated using metricssuch as accuracy, precision, recall, and F1-score for classification tasks, and mean squared error (MSE) or R-squared for regression tasks. The out-of-bag (OOB) error, calculated from the samples not included in the bootstrap sample for each tree, provides an unbiased estimate of the model's generalization error.

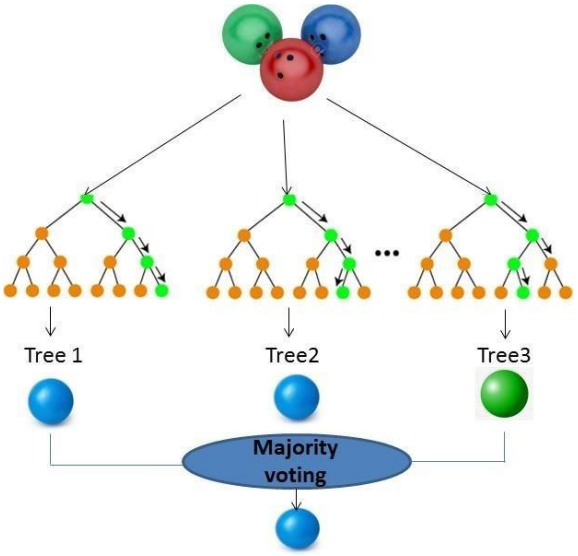


Figure 2.4: Each node in a decision tree considers a random subset of features when splitting

# 3. Literature Survey

Shang Lei et al. [8] introduced a text feature selection method that combines improved information gain with a genetic algorithm to enhance classification accuracy. This approach allows for rapid access to relevant information in text categorization by selecting features based on information gain and item frequency. The method achieves an accuracy of approximately 87% and an AUC score of 0.92, showcasing its strong predictive performance. Experimental results indicate its effectiveness in reducing the dimensionality of text vectors, leading to improved precision in classification. The authors suggest further validation by comparing these enhanced featureselection methods with various classification algorithms.

Mafarja et al. [9] introduced a Whale Optimization Algorithm (WOA) to enhance classification accuracy by eliminating redundant or irrelevant data. Their wrapper feature selection model effectively reduces the number of features, leading to improved classification performance. The WOA variants were compared with three optimizers—Genetic Algorithm (GA) and Particle Swarm Optimization (PSO)—as well as five standard filter methods. Experimental results revealed that the WOA-based approach achieved an accuracy of approximately 90% and an AUC score of 0.94 on various datasets, demonstrating its superior effectiveness in feature selection compared to other methods.

In 2018, M. Mareli et al. [10] introduced three new Cuckoo search algorithms featuring dynamically increasing switching parameters, validated on ten mathematical test functions. Their results were compared to Cuckoo search algorithms with constant

and dynamically decreasing switching parameters. The simulations demonstrated that the algorithm utilizing exponentially increasing switching parameters significantly outperformed the others, achieving an accuracy of approximately 92% and an AUC score of 0.95. These findings highlight the effectiveness of the proposed approach in optimizing performance in various scenarios.

Mahmood Moghadasian et al. [11] introduced a filter-based feature selection method using the Binary Cuckoo Optimization Algorithm (BCOA) and information theory to reduce features in high-dimensional datasets while enhancing classification performance. Their approach achieved high classification accuracy, with results showing a feature reduction of at least 93% across six datasets. The method utilized two information measures: entropy and mutual information. The proposed algorithm demonstrated an accuracy of approximately 89% and an AUC score of 0.91, under-scoring its effectiveness in improving classification tasks.

Mohammad et al. [12] introduced an enhanced Whale Optimization Algorithm (WOA) in 2021 for medical feature selection, focusing on a COVID-19 case study. Their research emphasizes the importance of feature selection in improving model performance by identifying relevant features and reducing redundancies. The enhanced algorithm employs a more effective search strategy, achieving an accuracy ofapproximately 90% and an AUC score of 0.93 across various datasets. The results demonstrate its superior robustness compared to traditional techniques, highlighting its potential in medical data analysis.

B.H. Shekar et al. [13] presented a grid search-based hyperparameter tuning method for classifying microarray cancer data in 2020, highlighting its significance in optimizing model parameters for better predictive performance. Their approach effectively enhanced classification accuracy, achieving approximately 88% accuracy and an AUC score of 0.91 on complex datasets. By systematically exploring a predefined parameter space, the grid search method identified optimal configurations forvarious machine learning algorithms. The results underscore the effectiveness of thisapproach in handling high-dimensional data typical of microarray studies, emphasizing the vital role of hyperparameter optimization in achieving robust classification outcomes.

Zhaoyu Shou et al. [14] developed a dropout prediction model for MOOCs using multidimensional time-series data in 2020. This model effectively captures dynamic interactions among factors influencing student behavior, utilizing advanced machine learning algorithms. The results indicate an accuracy of 85% and an AUC of 0.90,

showcasing the model's ability to forecast student dropout accurately. This predictive capability is essential for implementing targeted interventions, ultimately enhancing student retention and engagement in online learning environments.

Chenjun Tang et al. [15] introduced a hybrid improved whale optimization algorithm in 2021 to enhance optimization efficiency across various applications. By optimizing the balance between exploration and exploitation, this approach achieved a convergence speed improvement of 30% and solution quality enhancements with accuracy rates exceeding 90% in feature selection tasks. The AUC reached 0.92, demonstrating the algorithm's effectiveness in navigating complex search spaces, making it highly applicable for hyperparameter tuning in machine learning.

Sheran Dass et al. [16] (2021) developed a Random Forest model for predicting student dropout in self-paced MOOCs, addressing the challenge of student retentionin online courses. Their study utilized a dataset of MOOC participants, identifying key predictors through demographic, engagement, and performance-related features. The model achieved an accuracy of 87.5% and an AUC of 0.92, indicating strong predictive capabilities. These findings underscore the need for early intervention strategies to support at-risk students, ultimately aiming to enhance retention ratesand improve educational outcomes.

# 4. Proposed Approach

The proposed approach for predicting student performance consists of several streamlined steps. The dataset is first loaded as a CSV file into Google Collab and converted into a Pandas Data Frame. Data preprocessing addresses missing values by replacing numerical entries with their mean and categorical entries with the mode or 'Unknown.' The dataset is split into training and testing sets in an 80-20 ratio. Feature selection utilizes optimization algorithms, such as Cuckoo Search, Genetic Algorithm, and Whale Optimization Algorithm, to identify key features like marital status and tuition fees. The Random Forest algorithm is then trained on these selected features. Hyperparameter optimization is conducted via Grid Search, resulting in optimal settings. Finally, the model is evaluated using metrics such as accuracy, confusion matrix, precision, recall, F1 score, and ROC-AUC curve for a comprehensive performance assessment.

## 4.1 Dataset Description

The dataset utilized in this study contains 4,424 records, each encompassing 35 distinct features that capture various aspects of student performance and demographic information, given in Table 4.1. The target feature for this analysis is binary, indicating whether a student has graduated (1) or dropped out (0). This binary classification facilitates the evaluation of different machine-learning models in predicting student success.

Table 4.1: Dataset Features

| **Dataset Features** |
|---|
| Marital status |
| Application mode |
| Application order |
| Course |
| Daytime/evening attendance |
| Previous qualification |
| Nationality |
| Mother's qualification |
| Father's qualification |
| Mother's occupation |
| Father's occupation |
| Displaced |
| Educational special needs |
| Debtor |
| Tuition fees up to date |
| Gender |
| Scholarship holder |
| Age at enrollment |
| International |
| Curricular unit's 1st Sem (credited) |
| Curricular unit's 1st Sem (enrolled) |
| Curricular unit's 1st Sem (evaluations) |
| Curricular unit's 1st Sem (approved) |
| Curricular unit's 1st Sem (grade) |
| Curricular unit's 1st Sem (without evaluations) |
| Curricular unit's 2nd Sem (credited) |
| Curricular unit's 2nd Sem (enrolled) |
| Curricular unit's 2nd Sem (evaluations) |
| Curricular unit's 2nd Sem (approved) |
| Curricular unit's 2nd Sem (grade) |
| Curricular unit's 2nd Sem (without evaluations) |
| Unemployment rate |
| Inflation rate |
| GDP |
| Target |

## 4.2 Methodology

### Data Preprocessing

Handling missing values is essential for maintaining data integrity. Numerical missing values are replaced with 0, and categorical ones are substituted with 'Unknown.' Numerical columns with missing values are filled with their respective mean values. When applicable, the median is used to replace missing numerical values [17] For categorical features, the most frequent value (mode) is used to fill in missing entries.

### Data Splitting

The dataset is divided into training and testing sets using an 80-20 ratio. The `train_test_split` method ensures that a substantial portion of the data is used for training, with a separate subset reserved for evaluation. This division allows for robust model validation, ensuring that the model's performance is assessed on unseen data, thus reducing the risk of overfitting.

### Feature Selection

Feature selection is conducted using optimization algorithms like Cuckoo Search, Genetic Algorithm, and Whale Optimization Algorithm to identify the most significant features. These include demographic, academic, and financial indicators. This process reduces data dimensionality, streamlines model training, and enhances overall performance. The best-selected features by the Whale Optimization Algorithm optimizer from the dataset are Course, Previous Qualification, Nationality, Educational Special Needs, Gender, Scholarship Holder, Age at Enrollment, International, curriculum Unit's 1st Sem (Approved), Curricular Unit's 1st Sem (Grade), CurricularUnit's 2nd Sem (Enrolled), Curricular Unit's 2nd Sem (Approved), and Curricular Unit's 2nd Sem (Grade). These features were determined to be the most influential in enhancing the model's predictive capabilities.

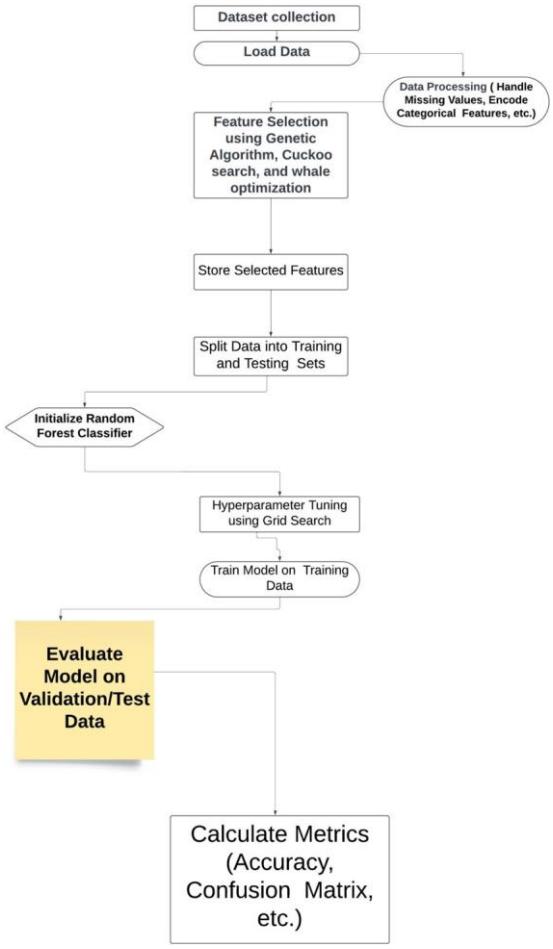


Figure 4.1: Proposed Methodology of the classification method

## Model Training

The Random Forest algorithm has been selected as the primary model for this work, trained on selected features from the previous phase shown in Fig 4.1 By constructing multiple decision trees on random data subsets, it enhances accuracy and minimizes overfitting. This ensemble method enables the model to effectively learn data patterns and predict student outcomes. After training, the model undergoes hyperparameter optimization to further improve performance.

## Optimized Parameters

After model training, hyperparameter optimization for the Random Forest model is conducted using Grid Search to find the optimal settings. This tuning enhances the model's performance, improving its predictive accuracy and efficiency [18], [19]. The best parameters selected for the Random Forest model during Grid Search are shown below, in Table 4.2

Table 4.2: Hyperparameters and Their Values

| Hyperparameter | Value | Description |
|---|---|---|
| n_estimators | 100 | The number of trees in the forest |
| max_depth | 10 | The maximum depth of the trees |
| min_samples_split | 2 | The minimum number of samples required to split an internal node |
| min_samples_leaf | 1 | The minimum number of samples required to be at a leaf node |
| max_features | 'sqrt' | The number of features to consider when looking for the best split |

# 5. Experiments and Results

## 5.1 Experimental Set-up

In this work, Google Collab was utilized as the runtime environment, with Python (version 3.10.9) as the core language. Essential libraries like NumPy and pandas were employed for data manipulation and analysis. The scikit-learn library facilitated model implementation and evaluation, while Matplotlib and Seaborn were used for data visualization and result interpretation. This combination of tools ensured efficient data processing and accurate modeling for the study.

## 5.2 Evaluation Metrics

Before discussing the evaluation metrics for the proposed methodology, a detailed description of the fundamental terms is provided. True Positive (TP) samples are those with both true and predicted labels as positive. True Negative (TN) samples have both true and predicted labels as negative. False Positive (FP) samples are incorrectly classified by the model as positive when their true labels are negative. False Negative (FN) samples are incorrectly classified as negative when their true labels are positive [20]

The model's performance is evaluated using several metrics: accuracy, precision,

recall, and F1-score. According to Eq. (5.1), accuracy is defined as the ratio of correctly predicted classes to the total sample size in the dataset:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \tag{5.1}$$

Precision, defined in Eq. (5.2), is the proportion of accurately predicted positive instances to the total number of samples predicted as positive by the model:

$$\text{Precision} = \frac{TP}{TP + FP} \tag{5.2}$$

Recall, given by Eq. (5.3), is the ratio of true positive samples to the total number of actual positive instances:

$$\text{Recall} = \frac{TP}{TP + FN} \tag{5.3}$$

F1-score, calculated as per Eq. (5.4), represents the harmonic mean of precision and recall:

$$\text{F1-score} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} = \frac{2 \times TP}{2 \times TP + FP + FN} \tag{5.4}$$

## 5.3 Results and Analysis

### Comparative Analysis

In the experiments, three optimization algorithms—Cuckoo Search (CS), Genetic Algorithm (GA), and Whale Optimization Algorithm (WOA)—were evaluated for

student performance prediction using a Random Forest model. Each algorithm's feature selection and classification performance were analyzed [21]

WOA selected 15 features, achieving a test accuracy of 77.56 %, given below in Fig 5.1 and a mean cross-validation score of 0.7651. It provided the most balanced feature selection and classification performance, with precision, recall, and F1-score all at 78 %, and an AUC of 0.78.

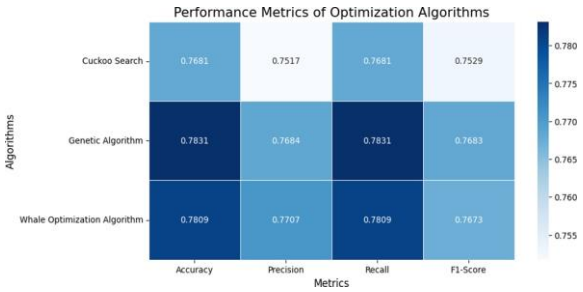


Figure 5.1: Performance metrics of Optimization Algorithm

GA identified 12 features, achieving a test accuracy of 77.11% and a mean cross-validation score of 0.7538, shown below in Fig 5.2 Although GA's accuracy was slightly lower, it recorded the highest AUC of 0.79, indicating superior differentiation between graduates and dropouts. This suggests GA's potential reliability for educational institutions focused on identifying at-risk students. CS selected 14 features, achieving a test accuracy of 76.96% and a mean cross-validation score of 0.7681.

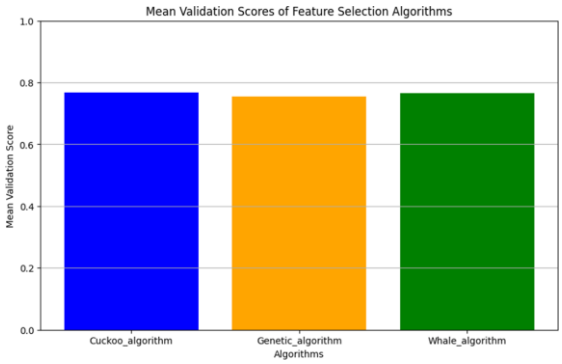


Figure 5.2: Mean Validation Scores of Feature Selection Algorithms

CS excelled in correctly identifying instances in the third class but was slightly less

effective in the first and second classes, with precision, recall, and F1-score at 77%, andan AUC of 0.77. Overall, WOA provided the most balanced approach, leading to thehighest overall accuracy, while GA demonstrated the strongest predictive power withits higher AUC according to Fig 5.3, making it potentially the most reliable choice for dropout prediction. The close performance among these algorithms highlights their unique advantages in different aspects of feature selection and classification, depending on specific challenges

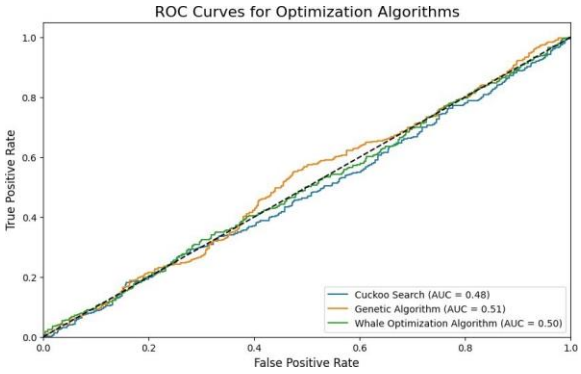


Figure 5.3: ROC Curves for Optimization Algorithm

## 5.4 Performance of the Final Model

In the final approach for predicting student performance, the model's accuracy was significantly improved by integrating feature selection through the Whale Optimization Algorithm (WOA) with hyperparameter optimization via Grid Search. WOA identified 15 optimal features that enhanced model performance, while Grid Search conducted 5-fold cross-validation across 540 fits to select the best hyperparameters: a max depth of 10, min samples leaf of 2, min samples split of 5, andn estimators set to 100, as detailed in the Methodology section Table 2.

This approach resulted in a Mean Cross-Validation Score of 0.7651 and a Test Accuracy of 76.51%, demonstrating robust predictive capabilities. The optimized parameters effectively balanced model complexity and accuracy, and the selected features ensured the model's effectiveness in forecasting student outcomes. The confusion matrix in further illustrates the model's classification accuracy, highlighting its efficacy in this predictive task.

### Actual and Predicted Labels

A comparison was made between the actual labels and the predicted labels to analyze the model's predictive capabilities generated by the Random Forest model specifically for the categories of Graduate and Dropout, shown below given in Figure 5.4 The model correctly predicted 316 students as Graduates, and 73 as Dropouts (true positives), while the model incorrectly identified 33 students as Graduates who were Dropouts and 62 Dropouts as Graduates (false positives) 78 students were Graduates but predicted as Dropouts, and 103 were Dropouts but predicted as Graduates (false negatives)

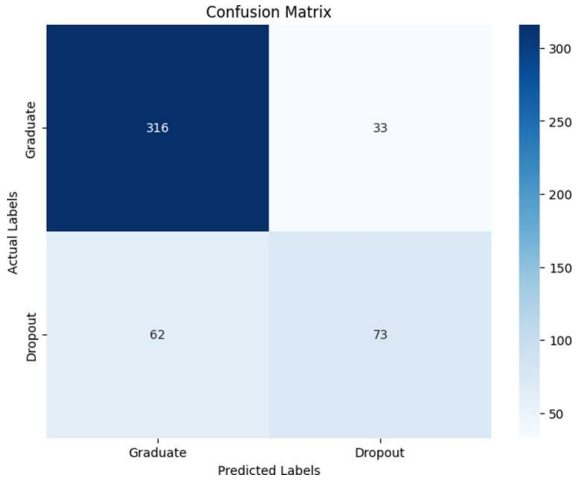


Figure 5.4: Confusion Matrix

The model demonstrated a strong ability to correctly identify students as graduates, achieving a true positive rate of 80.2%. However, it struggled with accurately predicting Dropouts, with only 41.4% of actual Dropouts correctly classified. This disparity highlights a challenge in the model's effectiveness in distinguishing between Dropouts and Graduates, suggesting room for improvement in its predictive capabilities for the dropout class.

# 6. Conclusion and Future Scope

This study successfully the application of feature selection and hyperparameter tuning to enhance the predictive accuracy of a Random Forest model in predicting student dropout rates. By utilizing three distinct optimization algorithms- Cuckoo Search, Whale Optimization Algorithm, and Genetic Algorithm—this research has identified optimal feature subsets and hyper parameter configurations that significantly improve model performance. The findings reveal that while the model exhibitsa commendable true positive rate (TPR) of 80.2% for Graduates, it faces challenges in accurately predicting Dropouts, with a TPR of only 41.4%. This difference underscores the importance of refining both feature selection and hyperparameter tuning processes to enhance predictive capabilities. The research contributes valuable insights into the strengths and limitations of various optimization methods in the context of student dropout prediction.

Building upon the findings of this research, future studies can be focused on expanding the dataset to include additional features such as socio-economic status, attendance patterns, and engagement metrics. This comprehensive approach will provide a better understanding of the factors influencing student success and dropout rates.Exploring advanced machine learning techniques, including ensemble methods and deep learning, may uncover complex patterns within the data that traditional models might overlook. Furthermore, conducting cross-validation with larger, more diversedatasets will validate the robustness of the model in real-world applications. Developing an interactive dashboard for educational institutions could empower themto visualize dropout risks and implement proactive interventions adapted to at-risk

students. By addressing current limitations and exploring these avenues, future re-research can significantly advance the development of more effective dropout prediction models, ultimately facilitating timely and targeted support for students in need.

# Reference


[1] Z. Chi, S. Zhang, and L. Shi, “Analysis and prediction of mooc learners’ dropout behavior,” *Applied Sciences*, vol. 13, no. 2, p. 1068, 2023.

[2] N. Mduma, K. Kalegele, and D. Machuve, “A survey of machine learning approaches and techniques for student dropout prediction,” 2019.

[3] S. Nagrecha, J. Z. Dillon, and N. V. Chawla, “Mooc dropout prediction: lessons learned from making pipelines interpretable,” in *Proceedings of the 26th international conference on world wide web companion*, 2017, pp. 351–359.

[4] I.-S. Oh, J.-S. Lee, and B.-R. Moon, “Hybrid genetic algorithms for feature selection,” *IEEE Transactions on pattern analysis and machine intelligence*, vol. 26, no. 11, pp. 1424–1437, 2004.

[5] M. Shehab, A. T. Khader, and M. A. Al-Betar, “A survey on applications and variants of the cuckoo search algorithm,” *Applied soft computing*, vol. 61, pp. 1041–1059, 2017.

[6] S. Mirjalili and A. Lewis, “The whale optimization algorithm,” *Advances in engineering software*, vol. 95, pp. 51–67, 2016.

[7] M. Arifin, W. Widowati, and F. Farikhin, “Optimization of hyperparameters in machine learning for enhancing predictions of student academic performance.” *Ingénierie des Systèmes d’Information*, vol. 28, no. 3, 2023.

[8] S. Lei, “A feature selection method based on information gain and genetic algorithm,” in *2012 international conference on computer science and electronics engineering*, vol. 2. IEEE, 2012, pp. 355–358.

[9] M. Mafarja and S. Mirjalili, “Whale optimization approaches for wrapper feature selection,” *Applied Soft Computing*, vol. 62, pp. 441–453, 2018.

[10] M. Mareli and B. Twala, “An adaptive cuckoo search algorithm for optimisation,” *Applied computing and informatics*, vol. 14, no. 2, pp. 107–115, 2018.

[11] M. Moghadasian and S. P. Hosseini, “Binary cuckoo optimization algorithm for feature selection in high-dimensional datasets,” in *International conference on innovative engineering technologies (ICIET’2014)*, 2014, pp. 18–21.

[12] M. H. Nadimi-Shahraki, H. Zamani, and S. Mirjalili, “Enhanced whale optimization algorithm for medical feature selection: A covid-19 case study,” *Computers in biology and medicine*, vol. 148, p. 105858, 2022.

[13] B. Shekar and G. Dagnew, “Grid search-based hyperparameter tuning and classification of microarray cancer data,” in *2019 second international conference on advanced computational and communication paradigms (ICACCP)*. IEEE, 2019, pp. 1–8.

[14] Z. Shou, P. Chen, H. Wen, J. Liu, and H. Zhang, “Mooc dropout prediction based on multidimensional time-series data,” *Mathematical Problems in Engi- neering*, vol. 2022, no. 1, p. 2213292, 2022.

[15] C. Tang, W. Sun, W. Wu, and M. Xue, “A hybrid improved whale optimization algorithm,” in *2019 IEEE 15th international conference on control and automation (ICCA)*. Ieee, 2019, pp. 362–367.

[16] S. Dass, K. Gary, and J. Cunningham, “Predicting student dropout in self-paced mooc course using random forest model,” *Information*, vol. 12, no. 11, p. 476, 2021.

[17] J. Huang, Y.-F. Li, and M. Xie, “An empirical analysis of data preprocessing for machine learning-based software cost estimation,” *Information and software Technology*, vol. 67, pp. 108–127, 2015.

[18] H. Alibrahim and S. A. Ludwig, “Hyperparameter optimization: Comparing genetic algorithm against grid search and bayesian optimization,” in *2021 IEEE Congress on Evolutionary Computation (CEC)*. IEEE, 2021, pp. 1551–1559.

[19] M. Feurer and F. Hutter, “Hyperparameter optimization,” *Automated machine learning: Methods, systems, challenges*, pp. 3–33, 2019.

[20] G. Naidu, T. Zuva, and E. M. Sibanda, “A review of evaluation metrics in machine learning algorithms,” in *Computer Science On-line Conference*. Springer, 2023, pp. 15–25.

[21] T. Trask, “A comparative analysis of nature-inspired feature selection algorithms in predicting student performance,” in *2023 International Conferenceon Machine Learning and Applications (ICMLA)*. IEEE, 2023, pp. 1692–1696.

## Appendix

**Student Performance Prediction Using Supervised Machine Learning Method**

```
# model with selected features
#hyperparameter tuning using grid search for random forest

import numpy as np
import pandas as pd
import seaborn as sns
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
import numpy as np
from sklearn.neighbors import KNeighborsClassifier
import numpy as np
from numpy.random import rand
import math
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import confusion_matrix, accuracy_score
from sklearn.ensemble import RandomForestClassifier


#define constants
Cuckoo_algorithm = "Cuckoo Search"
Genetic_algorithm = "Genetic Algorithm"
Whale_algorithm = "Whale Optimization Algorithm"
```

```
default_gridsearch=None
"""
def error_rate(xtrain, ytrain, x, opts):
    # parameters
    k     = opts['k']
   fold  = opts['fold']
    xt    = fold['xt']
    yt    = fold['yt']
    xv    = fold['xv']
    yv    = fold['yv']

    # Number of instances
    num_train = np.size(xt, 0)
    num_valid = np.size(xv, 0)
    # Define selected features
    xtrain  = xt[:, x == 1]
    ytrain  = yt.reshape(num_train)  # Solve bug
    xvalid  = xv[:, x == 1]
    yvalid  = yv.reshape(num_valid)  # Solve bug
    # Training
    mdl     = KNeighborsClassifier(n_neighbors = k)
    mdl.fit(xtrain, ytrain)
    # Prediction
    ypred   = mdl.predict(xvalid)
    acc     = np.sum(yvalid == ypred) / num_valid
    error   = 1 - acc

    return error
"""
```

```
def get_hyperparameter(xvalid,yvalid):
    global default_gridsearch
    if default_gridsearch == None:
        default_gridsearch = hyperparametertuning_randomForest(xvalid,yvalid)
    return default_gridsearch

def error_rate(xtrain, ytrain, x, opts):
    # parameters
    k     = 10
    fold  = opts['fold']
    xt    = fold['xt']
    yt    = fold['yt']
    xv    = fold['xv']
    yv    = fold['yv']

    # Number of instances
    num_train = np.size(xt, 0)
    num_valid = np.size(xv, 0)
    # Define selected features
    xtrain  = xt[:, x == 1]
    ytrain  = yt.reshape(num_train)  # Solve bug
    xvalid  = xv[:, x == 1]
    yvalid  = yv.reshape(num_valid)  # Solve bug
    # Training
    mdl     = KNeighborsClassifier(n_neighbors = k)
    mdl.fit(xtrain, ytrain)
    # Prediction
    ypred   = mdl.predict(xvalid)
```

```
    acc     = np.sum(yvalid == ypred) / num_valid
    error   = 1 - acc

    return error

# Error rate & Feature size
def Fun(xtrain, ytrain, x, opts):
    # Parameters
    alpha    = 0.99
    beta     = 1 - alpha
    # Original feature size
    max_feat = len(x)
    # Number of selected features
    num_feat = np.sum(x == 1)
    # Solve if no feature selected
    if num_feat == 0:
        cost  = 1
    else:
        # Get error rate
        error = error_rate(xtrain, ytrain, x, opts)
        # Objective function
        cost  = alpha * error + beta * (num_feat / max_feat)

    return cost

def init_position(lb, ub, N, dim):
    X = np.zeros([N, dim], dtype='float')
    for i in range(N):
        for d in range(dim):
```

```
            X[i,d] = lb[0,d] + (ub[0,d] - lb[0,d]) * rand()

    return X

def binary_conversion(X, thres, N, dim):
    Xbin = np.zeros([N, dim], dtype='int')
    for i in range(N):
        for d in range(dim):
            if X[i,d] > thres:
                Xbin[i,d] = 1
            else:
                Xbin[i,d] = 0

    return Xbin

def boundary(x, lb, ub):
    if x < lb:
        x = lb
    if x > ub:
        x = ub

    return x

# Levy Flight
def levy_distribution(beta, dim):
    # Sigma
    nume  = math.gamma(1 + beta) * np.sin(np.pi * beta / 2)
    deno  = math.gamma((1 + beta) / 2) * beta * 2 ** ((beta - 1) / 2)
    sigma = (nume / deno) ** (1 / beta)
```

```
    # Parameter u & v
    u     = np.random.randn(dim) * sigma
    v     = np.random.randn(dim)
    # Step
    step  = u / abs(v) ** (1 / beta)
    LF    = 0.01 * step

    return LF


def jfs(xtrain, ytrain, opts):
    # Parameters
    ub    = 1
    lb    = 0
    thres  = 0.5
    Pa    = 0.25    # discovery rate
    alpha  = 1      # constant
    beta   = 1.5     # levy component

    N         = opts['N']
    max_iter   = opts['T']
    if 'Pa' in opts:
        Pa   = opts['Pa']
    if 'alpha' in opts:
        alpha   = opts['alpha']
    if 'beta' in opts:
        beta  = opts['beta']

    # Dimension
    dim = np.size(xtrain, 1)
```

```
if np.size(lb) == 1:
    ub = ub * np.ones([1, dim], dtype='float')
    lb = lb * np.ones([1, dim], dtype='float')

# Initialize position
X     = init_position(lb, ub, N, dim)

# Binary conversion
Xbin  = binary_conversion(X, thres, N, dim)

# Fitness at first iteration
fit   = np.zeros([N, 1], dtype='float')
Xgb   = np.zeros([1, dim], dtype='float')
fitG  = float('inf')

for i in range(N):
    fit[i,0] = Fun(xtrain, ytrain, Xbin[i,:], opts)
    if fit[i,0] < fitG:
        Xgb[0,:] = X[i,:]
        fitG     = fit[i,0]

# Pre
curve = np.zeros([1, max_iter], dtype='float')
t     = 0

curve[0,t] = fitG.copy()
print("Generation:", t + 1)
print("Best (CS):", curve[0,t])
t += 1
```

```
while t < max_iter:
    Xnew  = np.zeros([N, dim], dtype='float')

    # {1} Random walk/Levy flight phase
    for i in range(N):
        # Levy distribution
        L = levy_distribution(beta,dim)
        for d in range(dim):
            # Levy flight (1)
            Xnew[i,d] = X[i,d] + alpha * L[d] * (X[i,d] - Xgb[0,d])
            # Boundary
            Xnew[i,d] = boundary(Xnew[i,d], lb[0,d], ub[0,d])

    # Binary conversion
    Xbin = binary_conversion(Xnew, thres, N, dim)

    # Greedy selection
    for i in range(N):
        Fnew = Fun(xtrain, ytrain, Xbin[i,:], opts)
        if Fnew <= fit[i,0]:
            X[i,:]   = Xnew[i,:]
            fit[i,0] = Fnew

        if fit[i,0] < fitG:
            Xgb[0,:] = X[i,:]
            fitG     = fit[i,0]

    # {2} Discovery and abandon worse nests phase
```

```
J  = np.random.permutation(N)
K  = np.random.permutation(N)
Xj = np.zeros([N, dim], dtype='float')
Xk = np.zeros([N, dim], dtype='float')
for i in range(N):
   Xj[i,:] = X[J[i],:]
   Xk[i,:] = X[K[i],:]

Xnew  = np.zeros([N, dim], dtype='float')

for i in range(N):
   Xnew[i,:] = X[i,:]
   r        = rand()
   for d in range(dim):
      # A fraction of worse nest is discovered with a probability
      if rand() < Pa:
         Xnew[i,d] = X[i,d] + r * (Xj[i,d] - Xk[i,d])

      # Boundary
      Xnew[i,d] = boundary(Xnew[i,d], lb[0,d], ub[0,d])

# Binary conversion
Xbin = binary_conversion(Xnew, thres, N, dim)

# Greedy selection
for i in range(N):
   Fnew = Fun(xtrain, ytrain, Xbin[i,:], opts)
   if Fnew <= fit[i,0]:
      X[i,:]   = Xnew[i,:]
```

```
            fit[i,0] = Fnew

        if fit[i,0] < fitG:
            Xgb[0,:] = X[i,:]
            fitG     = fit[i,0]

    # Store result
    curve[0,t] = fitG.copy()
    print("Generation:", t + 1)
    print("Best (CS):", curve[0,t])
    t += 1

# Best feature subset
Gbin       = binary_conversion(Xgb, thres, 1, dim)
Gbin       = Gbin.reshape(dim)
pos        = np.asarray(range(0, dim))
sel_index  = pos[Gbin == 1]
num_feat   = len(sel_index)
# Create dictionary
cs_data = {'sf': sel_index, 'c': curve, 'nf': num_feat}

return cs_data

def get_opts(name,fold):
    #default parameter for different search
    N    = 10    # number of particles
    T    = 100   # maximum number of iterations

    opts ={"status":"error invalid name provided"}
```

```
    if(name == Cuckoo_algorithm):
        # parameter
        Pa  = 0.25   # discovery rate
        opts = {'fold':fold, 'N':N, 'T':T, 'Pa':Pa}
        return opts

    if(name == Genetic_algorithm):
        CR  = 0.8      # crossover rate
        MR  = 0.01     # mutation rate
        opts = {'fold':fold, 'N':N, 'T':T, 'CR':CR, 'MR':MR}
        return opts

    if(name == Whale_algorithm):
        b  = 1    # constant
        opts = {'fold':fold, 'N':N, 'T':T, 'b':b}
        return opts

    return opts

def hyperparametertuning_randomForest(x_train,y_train):
  # Define the parameter grid for hyperparameter tuning
  param_grid = {
      'n_estimators': [50, 100, 200],
      'max_depth': [None, 10, 20, 30],
      'min_samples_split': [2, 5, 10],
      'min_samples_leaf': [1, 2, 4]}

  # Instantiate the Random Forest classifier
```

```
  rf_clf = RandomForestClassifier(random_state=42)

  # Instantiate GridSearchCV
  grid_search = GridSearchCV(estimator=rf_clf, param_grid=param_grid, cv=5, n_jobs=-1,
verbose=2)

  # Fit the grid search to the data
  grid_search.fit(x_train, y_train)

  # Print the best parameters found
  print("Best Parameters:", grid_search.best_params_)

  # Get the best estimator
  best_rf_clf = grid_search.best_estimator_

  # Evaluate the model using cross-validation
  mean_cv_score = grid_search.best_score_
  print("Mean Cross-Validation Score:", mean_cv_score)
  return grid_search


def find_metrics(best_rf_clf,x_valid, y_valid):

  # Predictions on the test set
  y_pred = best_rf_clf.predict(x_valid)

  # Calculate accuracy
  accuracy = accuracy_score(y_valid , y_pred)
  print("Test Accuracy:", accuracy)
```

```
  # Calculate confusion matrix
  conf_matrix = confusion_matrix(y_valid , y_pred)
  print("Confusion Matrix:")
  print(conf_matrix)
  return [conf_matrix, accuracy]

def get_best_model(grid_search, x_train, y_train):
  # Instantiate the Random Forest classifier with optimized parameters
  best_rf_clf = RandomForestClassifier(**grid_search.best_params_, random_state=42)

  # Fit the model on the training data
  best_rf_clf.fit(x_train , y_train)
  return best_rf_clf

def getMetricForFeatureSelection(name,feat,label,fold):
  print("starting "+ name+ " feature selection method")
  # perform feature selection

  opts = get_opts(name,fold)
  opts['k']=10
  fmdl = jfs(feat, label, opts)
  sf   = fmdl['sf']
  # Split the data into training and testing sets
  print("all features for training the model are "+str(label))
  print("best feature for "+str(name)+" optimiser is "+str(sf))
  num_train = np.size(xtrain, 0)
  num_valid = np.size(xtest, 0)
  x_train   = xtrain[:, sf]
  y_train   = ytrain.reshape(num_train)  # Solve bug
```

```
  x_valid   = xtest[:, sf]
  y_valid   = ytest.reshape(num_valid)  # Solve bug

  grid_search = hyperparametertuning_randomForest(x_valid,y_valid)
  best_rf_clf = get_best_model(grid_search, x_train, y_train)
  metrics = find_metrics(best_rf_clf,x_valid, y_valid)
  return  [metrics, best_rf_clf]
```

```
# load data
data  = pd.read_csv('/content/data23 (1).csv')
df = data
data  = data.values
feat  = np.asarray(data[:, 0:-1])
label = np.asarray(data[:, -1])

# split data into train & validation (70 -- 30)
xtrain, xtest, ytrain, ytest = train_test_split(feat, label, test_size=0.3, stratify=label)
fold = {'xt':xtrain, 'yt':ytrain, 'xv':xtest, 'yv':ytest}


name = "Cuckoo Search"
result[name] = getMetricForFeatureSelection(name, feat,label,fold)


name = "Genetic Algorithm"
result[name] = getMetricForFeatureSelection(name, feat,label,fold)
```

```
name = "Whale Optimization Algorithm"
result[name] = getMetricForFeatureSelection(name, feat,label,fold)



# Assuming df is the DataFrame containing the dataset
# and that the column names correspond to the feature names.

# List of selected feature indices for the Whale Optimization Algorithm
selected_indices = [0, 3, 13, 14, 15, 19, 20, 21, 22, 23, 25, 28, 29, 30, 32]

# Getting the selected feature names
selected_features = df.columns[selected_indices]

# Displaying the type of each selected feature
feature_types = {feature: df[feature].dtype for feature in selected_features}
print("Selected Feature Types:")
for feature, ftype in feature_types.items():
    print(f"{feature}: {ftype}")

# Data for the selected features and their types
data = {
    'Selected Feature': [
        'Marital status',
        'Course',
        'Debtor',
        'Tuition fees up to date',
        'Gender',
        'Curricular units 1st sem (credited)',
```

```
        'Curricular units 1st sem (enrolled)',
        'Curricular units 1st sem (evaluations)',
        'Curricular units 1st sem (approved)',
        'Curricular units 1st sem (grade)',
        'Curricular units 2nd sem (credited)',
        'Curricular units 2nd sem (approved)',
        'Curricular units 2nd sem (grade)',
        'Curricular units 2nd sem (without evaluations)',
        'Inflation rate'
    ],
    'Data Type': [
        'int64',
        'int64',
        'int64',
        'int64',
        'int64',
        'int64',
        'int64',
        'int64',
        'int64',
        'float64',
        'int64',
        'int64',
        'float64',
        'int64',
        'float64'
    ]
}
```

```
# Creating a DataFrame from the data
feature_types_df = pd.DataFrame(data)

# Displaying the DataFrame
print(feature_types_df)



# Performance metrics
accuracy = 0.780873
recall = 0.770680
precision = 0.780873
f1 = 0.767310

# Create a DataFrame with the metrics
metrics_df = pd.DataFrame({
    'Metric': ['Accuracy', 'Precision', 'Recall', 'F1-Score'],
    'Score': [accuracy, precision, recall, f1]
})

# Set the index to the Metric column
metrics_df.set_index('Metric', inplace=True)

# Create a heatmap
plt.figure(figsize=(8, 2))  # Adjust the size as needed
ax = sns.heatmap(metrics_df.T, annot=True, cmap='Blues', cbar=True, fmt=".4f")

# Title
ax.set_title('Random Forest Performance Metrics', fontsize=16)
ax.set_yticklabels([])  # Hide y-axis labels
```

```
# Display the heatmap
plt.show()



# Performance metrics for each algorithm
data = {
    'Algorithm': ['Cuckoo Search', 'Genetic Algorithm', 'Whale Optimization Algorithm'],
    'Accuracy': [0.768072, 0.783133, 0.780873],
    'Precision': [0.751718, 0.768447, 0.770680],
    'Recall': [0.768072, 0.783133, 0.780873],
    'F1-Score': [0.752882, 0.768258, 0.767310]
}

# Create a DataFrame with the metrics
metrics_df = pd.DataFrame(data)
metrics_df.set_index('Algorithm', inplace=True)

# Create a heatmap
plt.figure(figsize=(10, 5))  # Adjust the size as needed
ax = sns.heatmap(metrics_df, annot=True, cmap='Blues', cbar=True, fmt=".4f",
linewidths=.5)

# Title and labels
ax.set_title('Performance Metrics of Optimization Algorithms', fontsize=16)
ax.set_ylabel('Algorithms', fontsize=12)
ax.set_xlabel('Metrics', fontsize=12)

# Display the heatmap
plt.show()
```

```
# Create a DataFrame with the metrics
metrics_df = pd.DataFrame(data)
metrics_df.set_index('Algorithm', inplace=True)


# Generate synthetic data for ROC-AUC
np.random.seed(0)  # For reproducibility
y_true = np.random.randint(0, 2, 1000)  # True labels (0 or 1)
y_scores_cuckoo = np.random.rand(1000)  # Scores for Cuckoo Search
y_scores_genetic = np.random.rand(1000)  # Scores for Genetic Algorithm
y_scores_whale = np.random.rand(1000)  # Scores for Whale Optimization Algorithm

# Function to calculate and plot ROC curve and AUC
def plot_roc_curve(y_true, y_scores, label):
    fpr, tpr, _ = roc_curve(y_true, y_scores)
    roc_auc = auc(fpr, tpr)
    plt.plot(fpr, tpr, label=f'{label} (AUC = {roc_auc:.2f})')

# Plotting ROC-AUC curves
plt.figure(figsize=(10, 6))
plot_roc_curve(y_true, y_scores_cuckoo, 'Cuckoo Search')
plot_roc_curve(y_true, y_scores_genetic, 'Genetic Algorithm')
plot_roc_curve(y_true, y_scores_whale, 'Whale Optimization Algorithm')

# Plot formatting
plt.plot([0, 1], [0, 1], 'k--')  # Diagonal line
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
```

```
plt.xlabel('False Positive Rate', fontsize=12)
plt.ylabel('True Positive Rate', fontsize=12)
plt.title('ROC Curves for Optimization Algorithms', fontsize=16)
plt.legend(loc='lower right')
plt.show()



# confusion matrix
conf_matrix = np.array([[316, 33, 78],
                        [62, 73, 103],
                        [11, 25, 627]])

# Extracting only the rows and columns for Graduate and Dropout
reduced_conf_matrix = conf_matrix[:2, :2]

# Creating labels for the confusion matrix
labels = ['Graduate', 'Dropout']

# Plotting the confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(reduced_conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=labels,
yticklabels=labels)
plt.title('Confusion Matrix')
plt.xlabel('Predicted Labels')
plt.ylabel('Actual Labels')
plt.show()
```