Dissertation on
A Comparative Study of Machine Learning Algorithms for Prediction of Students' Performance

Thesis submitted towards partial fulfilment
of the requirements for the degree of

Master in Multimedia Development

Submitted by
Satyabrata Mandal

EXAMINATION ROLL NO.: M4MMD23006B
UNIVERSITY REGISTRATION NO.: 160394 of 2021-22

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

2023

Master in Multimedia Development
Course affiliated to
Faculty of Engineering and Technology
Jadavpur University
Kolkata, India

CERTIFICATE OF RECOMMENDATION

This is to certify that the thesis entitled **“A Comparative Study of Machine Learning Algorithms for Prediction of Students’ Performance”** is a bonafide work carried out by Satyabrata Mandal under our supervision and guidance for partial fulfilment of the requirements for the degree of Master in Multimedia Development in the School of Education Technology, during the academic session 2022-2023.

SUPERVISOR
School of Education Technology
Jadavpur University,
Kolkata-700 032

DIRECTOR
School of Education Technology
Jadavpur University,
Kolkata-700 032

DEAN - FISLM
Jadavpur University,
Kolkata-700 032

Master in Multimedia Development
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 in a manner satisfactory to warrant its acceptance as a prerequisite to the degree for which it has been submitted. It is understood that by this approval, the undersigned do not endorse or approve any statement made, 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 Thesis ---

** Only in case the thesis is approved.

DECLARATION OF ORIGINALITY AND COMPLIANCE OF ACADEMIC ETHICS

I hereby declare that this thesis contains literature survey and original research work by the undersigned candidate, as part of his **Master in Multimedia Development** studies.

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

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

NAME: SATYABRATA MANDAL

EXAMINATION ROLL NUMBER: M4MMD23006B

THESIS TITLE: A COMPARATIVE STUDY OF MACHINE LEARNING ALGORITHMS FOR PREDICTION OF STUDENTS' PERFORMANCE

SIGNATURE: DATE:

Acknowledgement

I feel fortunate while presenting this dissertation at the **School of Education Technology, Jadavpur University**, Kolkata, in the partial fulfilment of the requirement for the degree of **Master in Multimedia Development.**

I hereby take this opportunity to show my gratitude towards 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 express my warm thanks to **Dr. Matangini Chattopadhyay**, **Director of 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 goes to my classmates from **Master in Multimedia Development and M.Tech in IT (Courseware Engineering).** I do wish to thank all the departmental support staff and everyone else who has made different contributions to this dissertation.

Finally, my special gratitude to my parents who have invariably sacrificed and supported me and made me achieve this height.

Date:

Place: Kolkata

Satyabrata Mandal

Examination Roll Number: M4MMD23006B

Master in Multimedia Development

School of Education Technology

Jadavpur University

Kolkata:700032

Contents

Executive Summary

In today's evolving educational landscape, predicting student performance is crucial for resource allocation and support. While universities have explored this area, middle and high schools remain under-researched. This study aims to fill this gap through a literature review, emphasizing data mining techniques. Analysis of factors like gender, parental education, lunch type, and test preparation enabled accurate predictions. These insights empower institutions to make informed decisions, offer targeted support, and enhance student outcomes.

Machine learning models can analyze large amounts of data, including data about students' past performance, demographic information, and other relevant factors. By applying a variety of algorithms, researchers and educators can create predictive models that outperform traditional methods. These models can configure educational content and make modifications to meet individual student needs. By continuously analyzing student performance and learning patterns, machine-learning models can recommend personalized learning plans, resources, and assignments. This approach enhances the learning experience and helps students understand and recollect information better. The study of comparing machine learning algorithms to predict student performance has a transformative impact on education. Various machine algorithms, such as Random Forest, SVM, Decision Tree, XGBoost, CatBoost, LightGBM, and Naive Bayes, were explored and assessed using R scores to predict the students' performance and compare the results of these models.

A comprehensive exploration of machine learning techniques applied to forecast student academic achievements is undertaken. The study rigorously evaluates various algorithms, such as Decision-Trees, support vector machines (SVM), and naive bayes, with meticulous attention to feature selection and hyperparameter tuning. Notably, Decision Tree and SVM models emerge as frontrunners, demonstrating impressive accuracies of 86.12% and 96.00%, respectively, while the Naive Bayes model also exhibits competitive performance, showcasing its potential in predicting student success. The research underscores the value of machine learning in educational contexts and offers actionable insights for institutions aiming to enhance student support and intervention strategies.

1. Introduction

1.1. Overview

Student performance is a critical concern in schooling. A student's success in school significantly impacts their likelihood of completing higher education and obtaining a degree. However, not all students perform equally well, and several factors influence their academic achievements. Innovative and effective teaching methods can improve comprehension and retention of knowledge. Addressing these factors comprehensively and holistically is crucial for improving overall student performance and ensuring that all students have an equitable opportunity to succeed in their educational journeys. It requires collaboration among educators, policymakers, families, and communities to create a supportive and inclusive learning environment for all students. The following academic and non-academic factors that affect students' performance are mentioned below:

- **Academic factors:** These encompass elements such as previous academic performance, standardized test scores, and course-related challenges.
- **Non-academic factors:** These include variables such as family background, health status, and participation in extracurricular activities.

In recent years, there has been a growing interest in harnessing machine learning to predict students' academic performance. Machine learning models allow for the analysis of student data collections, revealing valuable patterns that facilitate predictions regarding their academic achievements [1] [3].

1.2. Problem Statement

A Comparative Study of Machine Learning Algorithms for Prediction of Students' Performance

1.3. Objective

a) To develop classification models for categorizing students into two classes: pass and fail.

b) To achieve optimal performance of the classification model, hyperparameter adjustments are made during model training.

c) To provide valuable insights into which machine learning algorithm or combination of algorithms is most effective in predicting students' academic outcomes.

d) To explore the effectiveness of XGBoost in handling imbalanced datasets and understanding its boosting capabilities to enhance classification performance.

e) Assessing the impact of regularization techniques on controlling overfitting and enhancing the robustness of XGBoost models

g) To explore the ensemble learning capabilities of Random Forest for improved predictive accuracy.

h) To evaluate the interpretability of decision tree models for educational data and the transparency of rules used to classify students.

2. Background Concept

Machine learning provides a diverse range of algorithms capable of predicting student performance with remarkable accuracy and granularity. These algorithms leverage historical academic data, demographic information, and various other relevant factors to create predictive models. ML offers a variety of algorithms that can be used to predict student performance. Some common algorithms include Decision Trees, Random Forests, Support Vector Machines (SVM), Naive Bayes, and gradient-boosting methods like XGBoost, Cat Boost, and LightGBM. The choice of model depends on the nature of the data and the problem's complexity. Machine learning is a subfield of artificial intelligence that focuses on the development of algorithms and models that enable computers to learn and make predictions or decisions without being explicitly programmed. In the context of predicting students' academic performance, machine learning models leverage historical data and patterns to forecast how well a student is likely to perform in their studies.

Student achievement in education serves as a vital measure of their long-term success. This achievement is influenced by various factors, encompassing both academic and non-academic aspects. Academic factors significantly impact a student's school performance, encompassing elements such as standardized test scores and coursework difficulty. Non-academic aspects also play a crucial role in a student's achievements, including family background, health status, and extracurricular involvement. While machine learning models have been developed for predicting student performance, many existing models predominantly emphasize academic factors, often neglecting the influence of non-academic factors. Consequently, these models may provide inaccurate predictions for students with diverse backgrounds. The development of this model entails using a dataset that includes information about student records, encompassing achievements, socioeconomic backgrounds, and other relevant factors. To determine how well it works, the model will undergo an evaluation using a dataset. The information gathered from this assessment will be used to improve the model's performance.

Eventually, educators and policymakers will have access to this tool to help enhance student achievement. To predict student performance and compare results, various machine learning models with the following algorithms are applied, as mentioned below-

2.1. XGBOOST (eXtreme Gradient Boosting)

XGBoost, also known as eXtreme Gradient Boosting, is a widely used and powerful machine learning algorithm that can handle both classification and regression tasks effectively. It leverages the gradient-boosting framework. Has gained popularity for its efficiency, accuracy, and robustness. In machine learning competitions and real-world applications, XGBoost has proven its ability to generate high-quality predictions, as presented in Figure 1.

Belonging to the ensemble learning methods family, XGBoost specifically falls under the boosting category. Boosting is a technique that combines weak learners (usually decision trees) to create a strong learner. By adding decision trees, XGBoost enhances the model's capabilities by correcting any errors made by previous trees [1] [2].

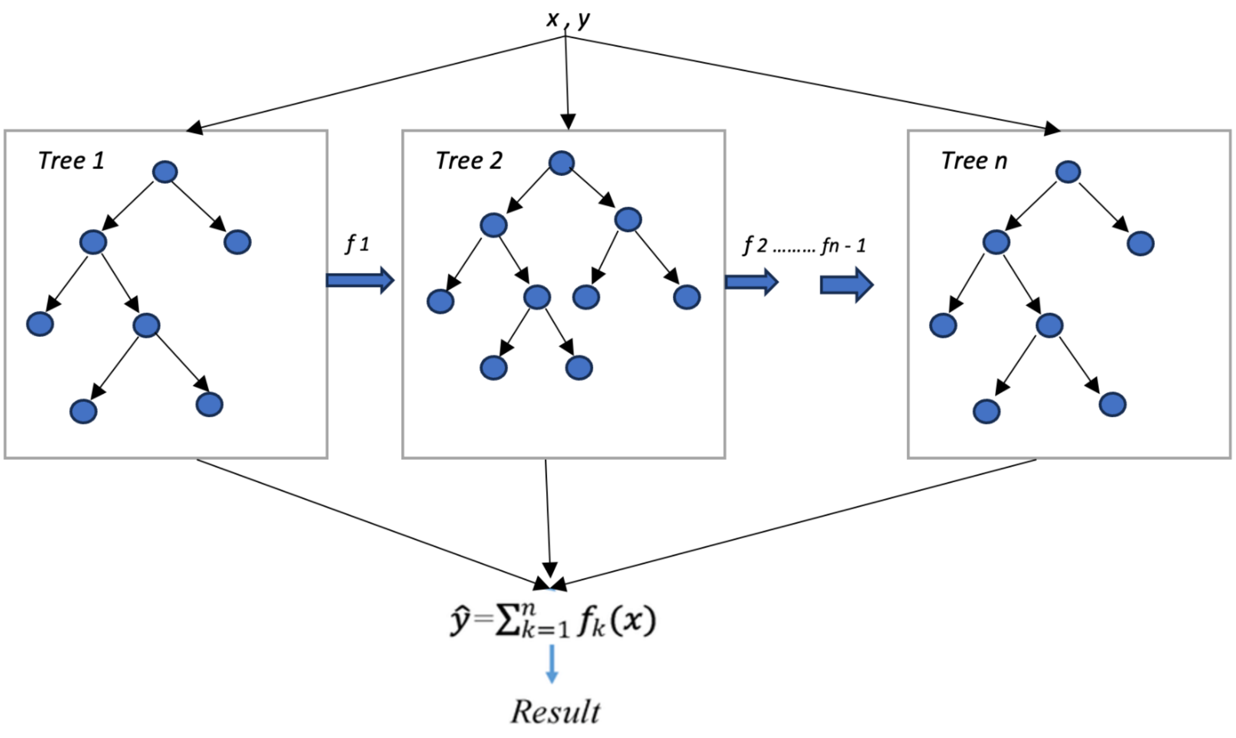


Figure 1: XG Boost Tree

2.2. Light GBM

LightGBM, or Light Gradient Boosting Machine, is a distinguished open-source machine learning framework originating from Microsoft. Its prominence in the realm of machine learning is attributed to its exceptional speed, efficiency, and scalability. This formidable gradient-boosting library is aptly equipped to grapple with intricate datasets and multifaceted machine-learning challenges, encompassing classification, regression, ranking, and recommendation systems [2].

What sets LightGBM apart is its innovative approach to histogram-based learning, its adeptness in handling features, and its harnessing of distributed computing resources. Of noteworthy significance is its capacity to seamlessly manage data without the need for explicit encoding, as well as its ability to capitalize on GPU acceleration for expedited model training whenever such hardware resources are available. In Figure 2, the hallmark of LightGBM lies in its capacity to strike an optimal equilibrium between model performance and computational efficiency, rendering it an esteemed choice within the arsenal of data scientists and machine learning practitioners. Its versatility extends to addressing an array of real-world problems with accuracy and precision [3].

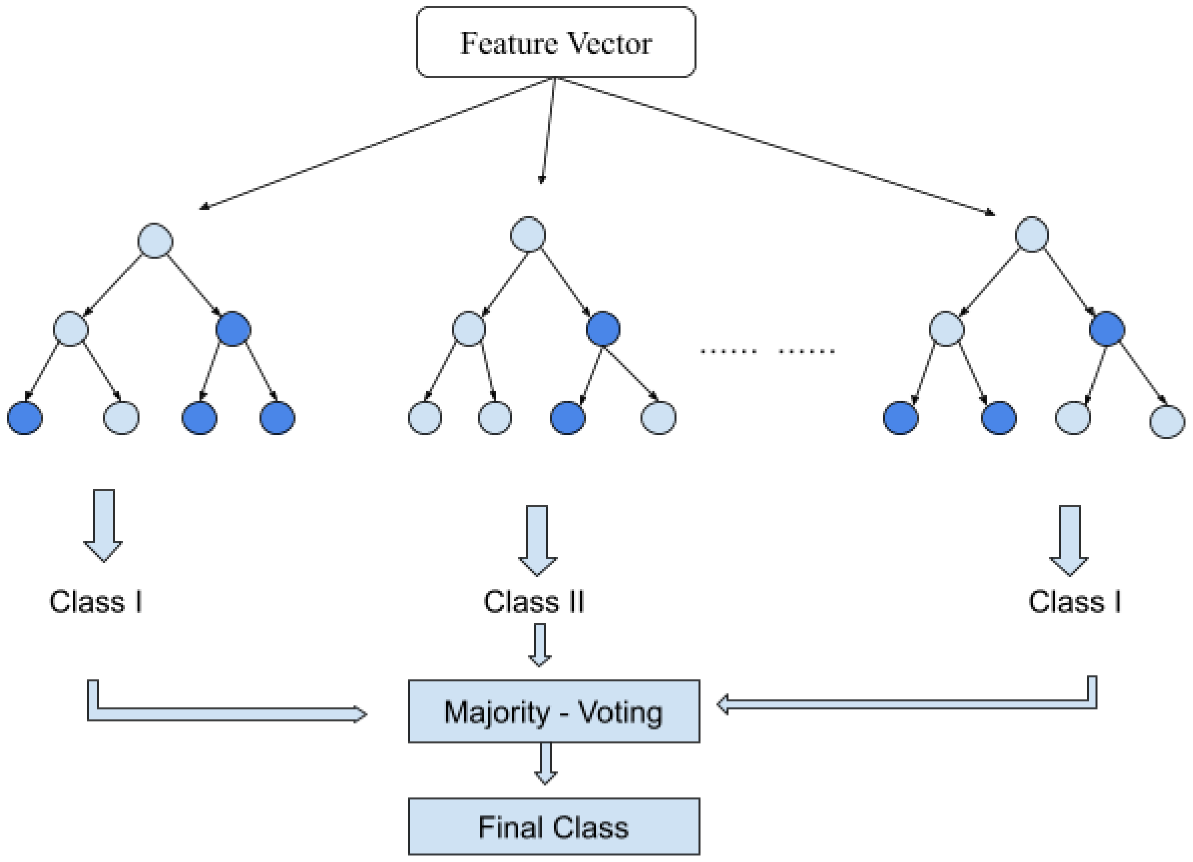


Figure 2: Light GBM Tree Algorithm

2.3. CatBoost

CatBoost is a machine-learning algorithm that leverages gradient-boosted decision trees. It excels at handling both numerical and qualitative features, eliminating the need for feature engineering. CatBoost is highly regarded for its speed, accuracy, and scalability. The way it operates is by adding trees to refine the model and rectify any errors made by previous trees, as shown in Figure 3. This iterative process, known as boosting, involves growing each tree using descent—a technique that minimizes the model's loss function. CatBoost stands out as a robust machine-learning algorithm capable of tackling diverse problem domains. Its efficiency, accuracy, and scalability make it a favourite among data scientists. One of its strengths lies in its ability to effortlessly handle both numerical features without requiring any feature engineering. Furthermore, thanks to parallelization and GPU acceleration techniques, it achieves efficiency. Additionally, its use of regularization techniques contributes to its accuracy. Lastly, CatBoost's scalability enables it to efficiently train models with large datasets [4].

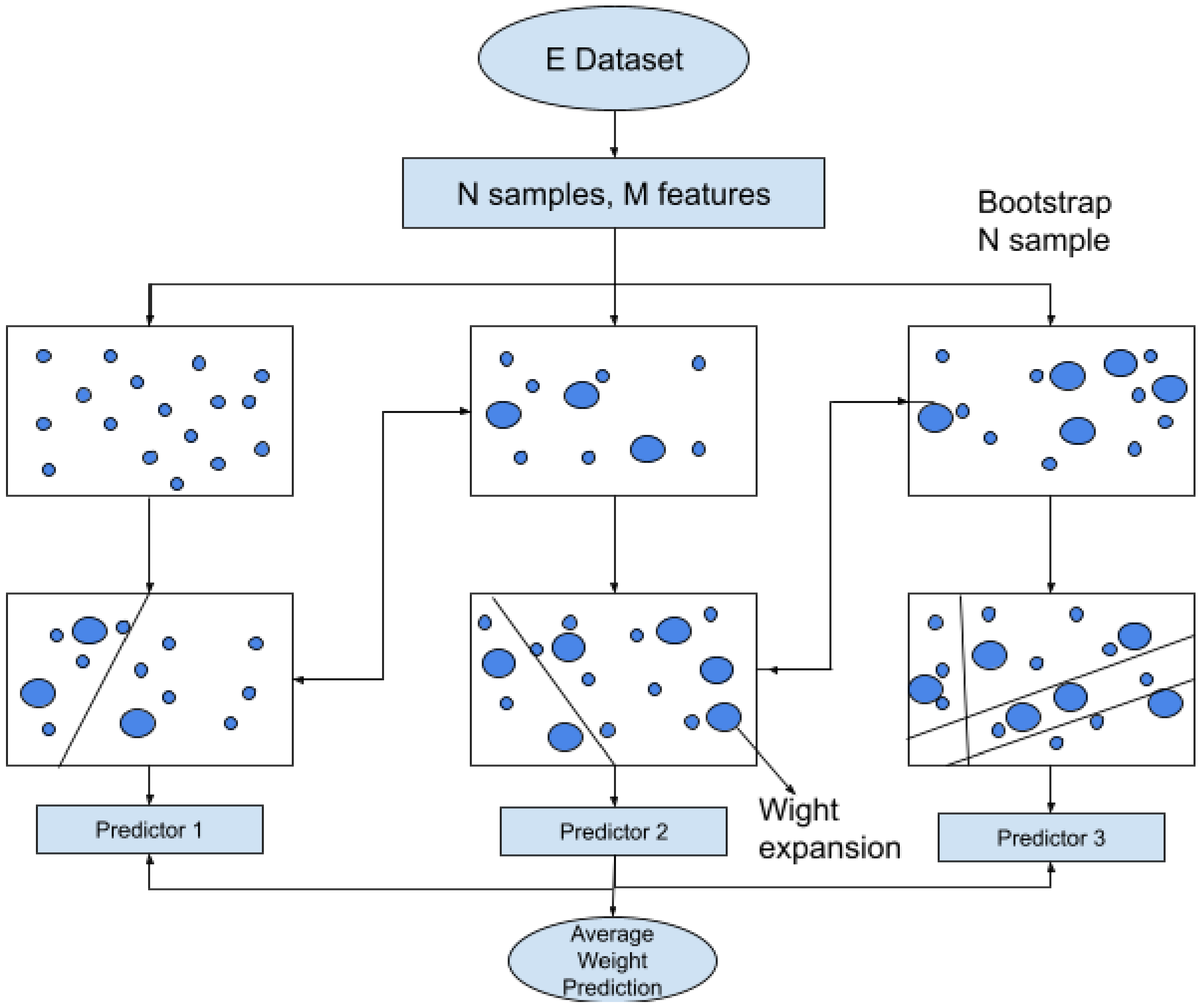


Figure 3: CatBoost Tree

2.4. Random Forest

Random forest is an algorithm in machine learning that forms a model by creating decision trees. Figure 4 shows Each tree is constructed using a subset of the data and features, which helps prevent the model from fitting closely to the training data. To put it simply, a random forest consists of a group of decision trees that are trained on subsets of the data. Each tree makes its own prediction, and all these predictions are then averaged to arrive at the final prediction. This approach enhances the model's accuracy. Reduces the risk of overfitting. Random forest is an algorithm for both regression and classification tasks. It's a good choice for beginners as well as experienced data scientists. It demonstrates accuracy, particularly when dealing with datasets. It handles noise and outliers effectively. Handle both numerical features [7].

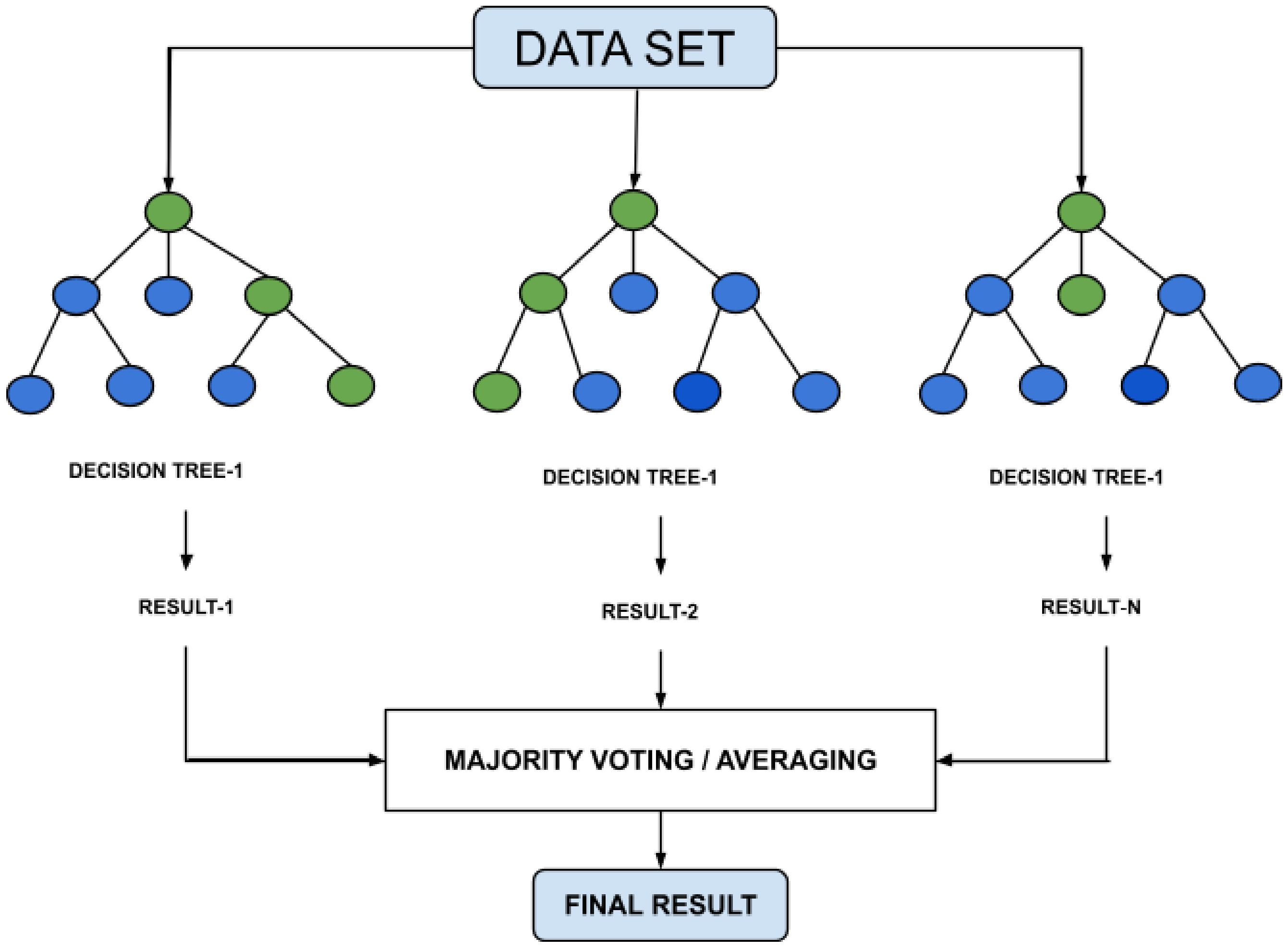


Figure 4: Random Forest Algorithm

2.6. Decision Tree

A decision tree serves as a supervised learning algorithm for handling both classification and regression tasks. Its functionality involves dividing the dataset into subsets until each subset can be classified or predicted with a high level of certainty. The structure of a decision tree resembles that of a flowchart consisting of nodes and branches, as shown in Figure 5. Nodes depict the decisions to be made, while branches represent the outcomes resulting from those decisions. The leaves of the tree signify classifications or predictions. Decision trees are widely used in machine learning due to their comprehensibility and interpretability. Moreover, they offer efficiency in training. Can handle both numerical and qualitative data effectively [9].

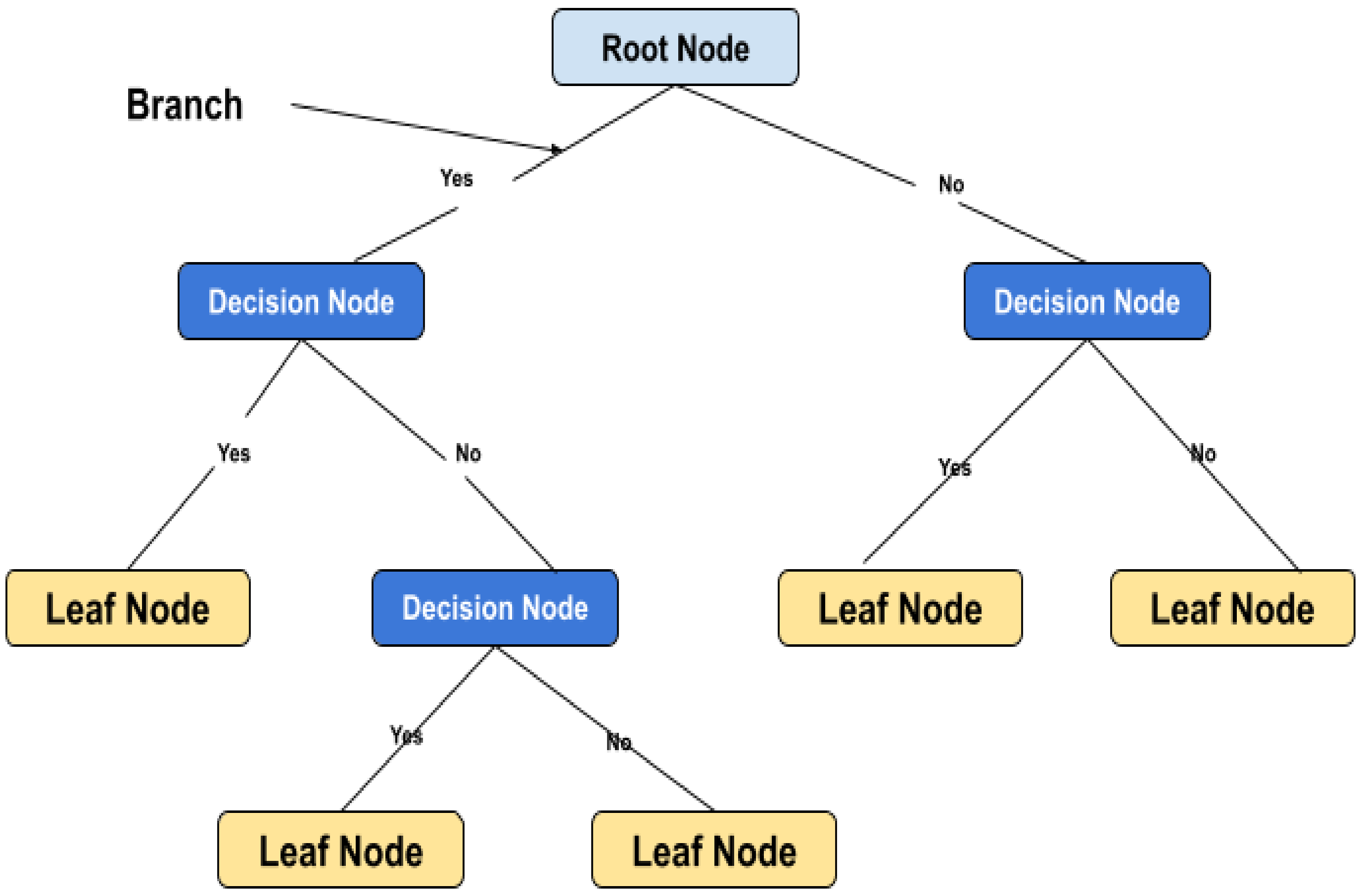


Figure 5: Decision Tree

2.7. Support Vector Machines (SVM)

Support vector machines (SVMs) are a type of machine learning algorithm that can be used for both classification and regression tasks. They operate by identifying a line or plane called the hyperplane, which effectively separates the data into two regions. In one area, all the data points belong to one class, while in another part, they belong to another class [5].

The SVM algorithm aims to find the hyperplane and the margin. This means that it looks for a line or plane that maximizes the distance between itself and the nearest data points on each side. This approach helps prevent overfitting of the model to the data.

Figure 6 depicts a support vector machine containing two sets of data, represented by red points. The hyperplane represents the dividing line between these two classes. The significance of the data points is determined by their proximity to the hyperplane, with those closest to it being referred to as support vectors [10] [11].

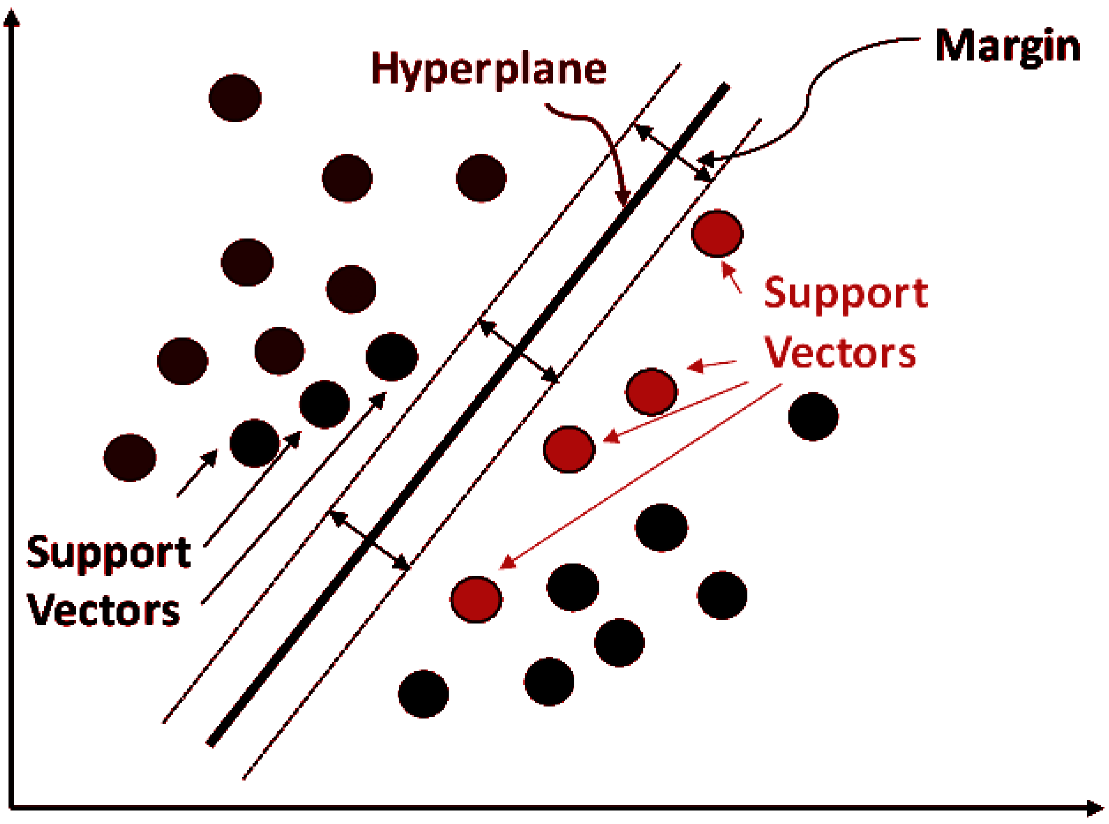


Figure 6: Support Vector Machines (SVM)

2.8. Naive Bayes

The Naive Bayes algorithm is a supervised learning method for classification issues that is based on the Bayes theorem. It is mostly employed in text categorization and has a large training set. The Naive Bayes Classifier is one of the most straightforward and efficient classification algorithms available today. It aids in the development of rapid machine-learning models capable of making accurate predictions. Being a probabilistic classifier, it makes predictions based on the likelihood that an object will occur [12].

Spam filtration, Sentiment analysis, and article classification are a few examples of Naive Bayes algorithms that are often used.

The Bayes theorem, commonly referred to as the Bayes' Rule, is used to calculate the likelihood of a hypothesis given certain previous information. The conditional probability determines this.

$$P(A|B)=\frac{P(B|A)P(A)}{P(B)}$$ Where

P(A|B) is the posterior probability: the probability of hypothesis A on the observed event B.

P(B|A) is the likelihood probability: the probability of the evidence given that the probability of a hypothesis is true.

P (A) is Priority Probability: the probability of a hypothesis before observing the evidence.

P(B) is Marginal Probability: the probability of Evidence.

2.9. Statistical-based Feature Selection

Statistical feature selection is a technique used in machine learning to pre-process data by selecting the most informative features from a dataset. It evaluates the relationship between each feature and the target variable using measures like correlation, squared tests, mutual information, or other significance tests. [11] By eliminating features that show insignificant associations with the target, this method reduces complexity, improves model efficiency, and potentially enhances performance by focusing on influential variables. It simplifies datasets, prevents overfitting, and makes machine-learning models more interpretable [12].

Univariate Feature Selection: In univariate feature selection, each feature is assessed separately in relation to the target variable, and a univariate statistical test is used to assess the degree of correlation between each feature and the goal. Based on this unique evaluation, the objective is to choose the qualities that are the most informative. When you want to quickly find a subset of characteristics that are most likely to have a major influence on your model's performance, univariate feature selection techniques are very helpful. They do not take into account feature interactions, and the dataset and particular machine learning method you want to employ will determine how successful they are.

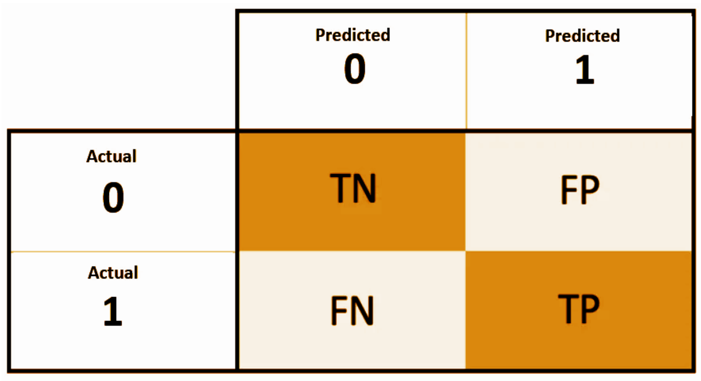
ANOVA F-statistic (f_classif): In classification problems, features are chosen using the ANOVA F-statistic. By contrasting the means of the feature values for each class, it evaluates the correlation between each feature and the target variable. A greater correlation between the characteristic and the target is indicated by a higher F-statistic value. The top k features (the number of features determines k to pick) are chosen using the SelectKBest scoring function in the code. This indicates that you are choosing the k characteristics that are considered to be the most pertinent for your classification assignment and have the highest F-statistic scores.

F score: The F score is commonly used in regression to determine the importance or significance of a feature in predicting the target variable. It measures how much of the variation in the target variable can be explained by a feature. A higher F score indicates that the feature holds importance in the regression model.

Chi-squared Test: It is a technique that uses the chi-squared (chi2) statistical test as the scoring function for selecting features, particularly when categorical features and classification tasks are involved. Each category feature's dependence or correlation with the target variable is evaluated using the chi2 test. For classification models, features that show a strong correlation with the goal are valuable.

Evaluation Metrics: Performance evaluation metrics are used to measure the classification performance of models. In classification tasks, there are applied evaluation metrics.

Confusion Matrix: One such metric is the Confusion Matrix, which provides a summary of predicted and actual class labels for a given set of data points. This matrix is particularly useful in classification problems with two classes. It can also be extended to handle multi-class scenarios. The Confusion Matrix consists of four cells that represent combinations of predicted and actual class labels, as illustrated in Figure 7. It is presented as a 2x2 matrix with four outcomes [13].



	Predicted 0	Predicted 1
Actual 0	TN	FP
Actual 1	FN	TP

Figure 8: Confusion matrix

True Positive (TP): The actual class is positive, and the model predicted the class correctly as positive.
False Positive (FP): The actual class is negative, but the model predicted the class as positive.
True Negative (TN): The actual class is negative, and the model predicted the class correctly as negative.
False Negative (FN): The actual class is positive, but the model predicted the class as negative.
Based on the TP, TN, FP, and FN, the following values are calculated:

Precision calculates the ratio of positives to the sum of positives and false positives. Precision plays a role in scenarios where false positives have consequences [13].

$$\text{Precision} = \frac{TP}{(TP+FP)}$$

Recall measures the proportion of positives compared to the sum of positives and false negatives [12] [1]. It becomes particularly important in situations where the consequences of missing a positive are significant.

$$\text{Recall} = \frac{TP}{(TP+FN)}$$

F1 Score provides an evaluation of how the model can accurately classify both positive and negative instances. It is especially useful when dealing with imbalanced datasets, where the number of instances in each class varies significantly [12] . The F1 score formula is

$$\text{F1 score} = 2 \times \frac{(precision \times recall)}{(precision + recall)}$$

Support represents how frequently samples are correctly classified as a class by a model, providing insights into pattern distribution and occurrence in the dataset. This information can be valuable for understanding data characteristics and identifying imbalances.

Accuracy indicates how well the model correctly predicts observations by comparing them to the number of observations [11]. Essentially, it measures how accurately both positive and negative instances are classified by the model.

$$\text{Accuracy} = \frac{(TP + TN)}{(TP + TN + FP + FN)}$$

Hyperparameter tuning involves the process of discovering the values for the hyperparameters of a machine-learning model. Unlike parameters, hyperparameters cannot be learned from training data. Must be set before training begins. Hyperparameter tuning is a procedure that requires experiments and evaluations to determine the best hyperparameter values that maximize the model's performance on a specific task. Grid search and randomized search are two techniques used for hyperparameter tuning in machine learning. Grid search works well when dealing with discrete hyperparameter spaces or when there is knowledge about specific values likely to yield good results. On the other hand, randomized search proves efficient when exploring large continuous hyperparameter spaces and aiming to test a wide range of values without exhaustively evaluating every possible combination [12] [13].

3. Literature Survey

A system called ProbSAP that aims to predict the performance of students [1]. This system utilizes machine learning algorithms to analyze student data from various sources, including transcripts, grades, and attendance records. By training algorithms, like XGBoost, LightGBM, and Catboost, with this data, ProbSAP generates predictions, which are then combined to provide a prediction. To validate its effectiveness, ProbSAP has undergone evaluation using datasets. The results demonstrate its ability to accurately forecast student performance, achieving an 85% accuracy rate in one study.

Kasumurthy et al. [2] conducted a study that employed machine learning algorithms to evaluate student performance. They compared two algorithms, regression and classification, for predicting student achievements. The results showed that the regression algorithm achieved an impressive accuracy rate of 93%, outperforming the classification algorithm, which reached 84%. The study also highlighted the significance of data processing techniques for extracting insights from learning experience repositories. Overall, the paper presents a methodology for analyzing student performance using machine learning algorithms and emphasizes the need for predictive models in various academic disciplines.

Analysis using data from an undergraduate course to understand student grades [3]. They used machine learning algorithms, including Logistic Regression, Linear Discriminant Analysis, K-Nearest Neighbors, Classification and Regression Trees, Gaussian Naive Bayes, and Support Vector Machines. The data was split into training and testing sets. The models were trained to predict students' final grades based on their performance in quizzes, exams, projects, and labs. The study's findings provided insights into the effectiveness of machine learning algorithms in predicting student performance. Notably, they found that Linear Discriminant Analysis was the best approach, with an impressive accuracy rate of 90.74%. This highlights the potential of LDA in early grade prediction, as it can identify students at risk and those who may need support proactively.

Josep Figueroa Cañas et al. [4] present two methods for identifying students who may be at risk in a statistics course. The first method focuses on predicting student dropout, while the second method aims to predict course performance. Both approaches rely on tree-based classification models. To conduct their research, the authors collected data from a statistics course offered online at the Universitat Oberta de Catalunya. The dataset included information such as student scores on self-assessment quizzes, participation in virtual classroom activities, and final exam grades. The study revealed that students' performance on self-assessment quizzes proved to be reliable predictors of potential dropouts. Additionally, the authors discovered that it was possible to identify students at risk of dropping out by analyzing their performance during the course.

The XGBoost algorithm predicts student performance using data from a computer science course [5]. They preprocessed the data by removing outliers and filling in missing values. The features considered in the model are exam grades, study hours, and online participation. The method achieved an 85% accuracy rate when compared to actual student grades.

In [8], various machine learning algorithms are represented because they address challenges like the nature of data, feature-outcome relationships, and data scarcity. This work shows the efficacy of various machine learning methods on the present student performance. It also examines factors such as information, academic background, past achievements, learning behaviors, and socio-emotional aspects as indicators of student performance. (Reviews machine learning algorithms, regression, logistic regression, decision trees, random forests, support vector machines, and neural networks used in student performance prediction.)

A system named SPADES [6] aims to forecast student performance using data mining and classification techniques. The system utilizes a decision tree algorithm to make predictions based on factors including the student's exam scores, study hours, and involvement in online discussions. To assess its effectiveness, SPADES was tested on a dataset comprising students enrolled in a statistics course. The findings revealed that SPADES achieved an accuracy rate of 85% when predicting student performance.

Ali Selamat et al. [7] proposed a machine learning-based model for predicting student grades in subjects. To develop the model, they utilized a dataset consisting of students enrolled in a statistics course. The dataset contained factors like exam scores, study hours, and participation in online discussions. To evaluate the effectiveness of the model, they used another set of students from the course as a test group. The findings revealed that the model achieved an accuracy rate of 95% when it came to predicting student grades.

Ching Chieh Kiu et al. [8] used learning analytics to predict challenges and potential dropouts in a statistics course. They collected data from an online course at Universitat Oberta de Catalunya, examining self-assessment quiz performance, virtual classroom engagement, and final exam results. Their findings revealed that self-assessment quiz performance was a key predictor of dropout. Importantly, they could identify at-risk students by analyzing the first half of the course. To proactively support struggling students, the authors proposed two approaches using tree-based classification models: one for predicting dropouts and another for forecasting course performance. These insights offer valuable opportunities for instructors to provide targeted interventions, such as personalized tutoring or access to online resources, to help at-risk students succeed in the course.

Pallathadka et al. [9] proposed a survey of machine learning (ML) techniques applied to predict student performance, categorizing them into traditional ML algorithms and deep learning algorithms. Traditional ML algorithms, such as decision trees, support vector machines, and random forests, are simpler to train but tend to be less accurate. On the other hand, deep learning algorithms, including neural networks, convolutional neural networks, and recurrent neural networks, are more complex and challenging to train but can achieve higher accuracy. Recent studies in this field, as exemplified by a 2023 publication, have demonstrated impressive accuracy rates, such as 85% accuracy in predicting student performance using a modified XGBoost algorithm in a computer science course.

Feiyue Qiu et al. [10] propose a technique to forecast how well students will perform in e-learning by analyzing their learning process and behavior data. The researchers collected data from a computer programming e-learning course, including information on quiz scores, assignment completion, final exam results, and the student's interactions with the course materials, like the number of pages viewed and time spent on each page. To predict student performance, the authors utilized a machine learning algorithm that analyzed the learning process and behavioral data. Impressively, this algorithm achieved an 85% accuracy rate in predicting student performance.

A study that offers insights into the application of machine learning to predicting student performance [11]. The study utilized data comprising more than 10,000 students from a university. This dataset included information on the students' demographic characteristics, academic achievements, and attendance records. To predict the students' final year CGPA (Cumulative Grade Point Average), the researchers employed four machine learning techniques: regression, support vector regression, decision tree, and random forests. According to the study's findings, the CGPA in the previous year proved to be a reliable predictor of the final year's CGPA.

4. Proposed Approach

Several machine learning algorithms were considered to assess student performance in this study, which aimed to classify students into two categories: pass or fail. A method for predicting student performance using machine learning is proposed, examined, and evaluated. The main goal is to assign classes to students based on their academic standing and relevant demographic data, categorizing them as either pass or fail. A dataset containing student data, including academic records and demographic information, is considered. Several machine learning algorithms have been implemented and evaluated, including Support Vector Machine (SVM), Random Forest (RF), Decision Tree (DT), XGBoost, Catboost, LightGBM, and Naive Bayes (NB). The dataset has been preprocessed for feature scaling and handling missing values. The following are the steps in developing the machine learning-based predictive model:

4.1. Dataset description

The dataset was obtained from the UCI Machine Learning repository and contains a wide range of attributes related to students' academic performance and personal background. The dataset contains 1,044 sample records of students, comprising a total of 33 features. The features include the student's school, gender, age, family size, parents' educational backgrounds and employment, motivation for selecting a particular school, travel times, study habits, past failures, involvement in extracurricular activities, internet access, and health status, among others.

4.2. Data Pre processing

Data cleaning: Outliers are removed from the 'Grade 3' column, which typically ranges from 0 to 20 but contains an outlier value of 80. In addition, irrelevant columns such as 'Grade 1' and 'Grade 2,' which represent previous exam scores, have been removed. These columns are considered extraneous for the final prediction task and may introduce noise into the models. Following this preprocessing step, a binary "Pass/Fail" target variable is created based on an initial threshold.

Splitting data: The train_test_split function divides the dataset into training and testing sets. This step ensures that the models are tested for generalization on a separate dataset after being trained on a different one. The dataset is further split into training and test sets in an 80% to 20% ratio after identifying correlated features.

4.3. Feature Selection

Feature Encoding: Categorical columns are one-hot encoded while dropping the first category to prevent multicollinearity. Categorical columns within the dataset need to undergo one-hot encoding. This transformation converts categorical data into numerical data, which can be utilized by machine learning algorithms. The first category in each encoded feature is dropped to avoid multicollinearity.

SelectKBest with ANOVA F-statistic: SelectKBest with ANOVA F-statistic is used to select features for SVM, Decision Tree, XGBoost, CatBoost, LightGBM, and Random Forest. This statistical technique evaluates the relationship between each feature and the target variable ('Pass/Fail'). It selects the top 'K' features that have the most significant impact on the target variable.

Chi-squared Test: The chi-squared test is used to select features for the Nave Bayes classifier. This statistical test evaluates the independence of each feature with respect to the target variable ('Pass/Fail'). The classification task retains features that are highly dependent on the target variable.

4.4. Methodology

The workflow of the classification model is shown in Figure 8.

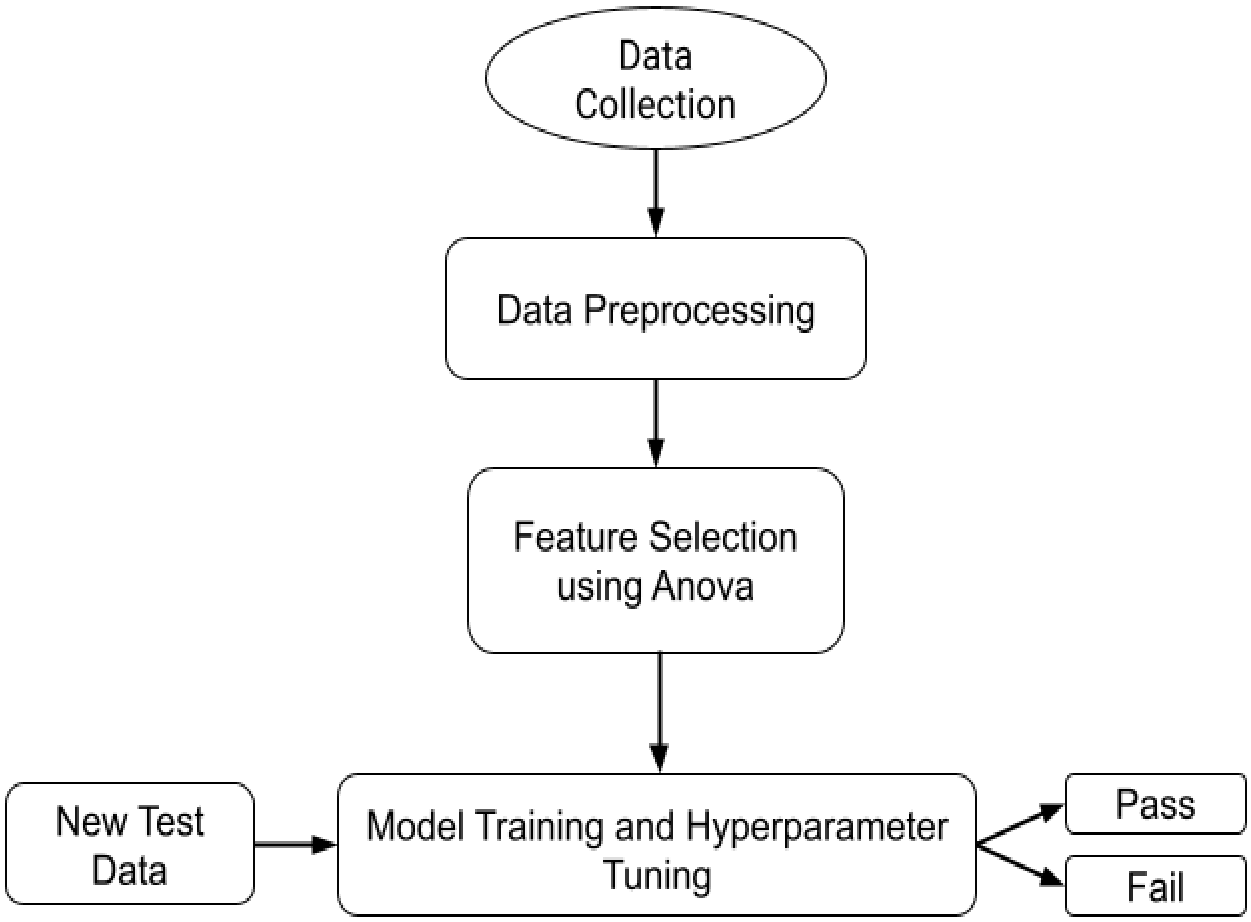

Data Collection
Data Preprocessing
Feature Selection using Anova
New Test Data
Model Training and Hyperparameter Tuning
Pass
Fail

Figure 8: Block Diagram of a Classification for the Student's Performance

4.5. Model Training and Hyperparameter Tuning: During the training and optimization phases of the model, various machine learning algorithms are utilized, including Support Vector Machine (SVM), Naive Bayes, Decision Tree, XGBoost, CatBoost, LightGBM, and Random Forest. The selected dataset undergoes hyperparameter tuning using techniques such as GridSearchCV for SVM, Decision Tree, XGBoost, CatBoost, LightGBM, and Random Forest. Additionally, Bayesian optimization is used in conjunction with LightGBM. The objective of this study is to identify the optimal configurations for each algorithm by testing various combinations of hyperparameters using cross-validation. Once the hyperparameters have been optimized, the models are trained on data to recognize patterns. Subsequently, a comprehensive evaluation of model performance is conducted using metrics such as accuracy, precision, recall, and F1 score to ensure that each algorithm has been fine-tuned for predicting student pass/fail outcomes. Here are tables 1, 2, 3, 4, 5, and 6 showing the models that use hyper-tuning.

Table 1: Different criteria are used to select the best tree in XGBoost and LightGBM.

`max_depth`	Maximum depth of a tree. It controls the complexity of the individual decision trees within the ensemble. Higher values can lead to overfitting.
`learning_rate`	The learning rate controls how much the model is updated at each iteration.
`n_estimators`	The number of trees in the model
`gamma`	A regularization parameter that controls how much the model is penalized for splitting on weak features
`subsample`	The fraction of the training data that is used to train each tree
`colsample_bytree`	a fraction of features to be randomly selected for building each tree. Similar to **'subsample,'** it introduces randomness to reduce overfitting.
`num_leaves`	This hyperparameter controls the maximum number of leaves in each tree. Increasing num_leaves can make the model more expressive but may also lead to overfitting.
`min_child_samples`	It specifies the minimum number of data points required in a leaf. A higher value can make the algorithm more robust but may underfit the data.

This grid represents a set of hyperparameters that can be used to fine-tune an Extreme Gradient Boosting (XGBoost) model. It includes several options for each hyperparameter:
max_depth: determines the maximum depth of each tree in the ensemble, offering choices of 3, 4, or 5 levels.
learning_rate: Controls the step size during optimization, with options of 0.1, 0.01, or 0.001.
n_estimators: specifies the number of boosting rounds, providing choices of 100, 200, or 300 iterations.
gamma: Regulates the minimum loss reduction to create a new tree branch, with options of 0, 0.1, or 0.2.
subsample: defines the fraction of samples used for training each tree, offering choices of 0.8, 0.9, or 1.0 (indicating a full training set).
colsample_bytree: Determines the fraction of features considered when growing each tree, with options of 0.8, 0.9, or 1.0 (indicating all features).

To optimize the performance of a light GBM classification model through grid tuning, consider the following ranges for hyperparameters:
depth: Experiment with values such as [4, 6, 8, 10] to determine the optimal tree depth. Deeper trees have the potential to capture complex data relationships.
learning_rate: Widen the range to include values such as [0.01, 0.1, 0.2, 0.3] to explore different learning rates, which impact the optimization process and model convergence.
Iterations: Extend the range to cover values such as [500, 1000, 1500, 2000] to investigate how the number of boosting rounds impacts model performance.
l2_leaf_reg: Investigate a wider range of values for L2 regularization, such as [1, 3, 5, 7], to evaluate the effect of regularization strength on the model's ability to generalize.
border_count: Test values such as [32, 64, 128, 256] to understand how varying the number of bins for numerical feature splitting influences both model accuracy and training speed.

Table 2: Different criteria are used to select the best tree in CatBoost.

`depth`	This particular setting determines how deep the individual decision trees are in the ensemble. Increasing the depth allows for trees, but there is a possibility of overfitting occurring.
`learning_rat e`	It scales the step size at each iteration while moving toward a minimum of the loss function. A smaller learning rate makes the optimization process more robust but may require more iterations.

`iterations`	The number of boosting rounds (trees) is chosen for the ensemble. Adding iterations may improve performance. It will also lengthen the training time.
`l2_leaf_reg`	This parameter adds L2 regularization to the cost function. It helps prevent overfitting by penalizing large coefficients. The value determines the strength of the regularization.
`border_count`	The number of splits for numerical features Higher values can lead to more precise splits but may increase training time.

The provided hyperparameters represent the range for tuning a CatBoosting model.
learning_rate: The step size during optimization is set to 0.1.
max_depth: The maximum depth of each tree in the ensemble, limited to 3 levels.
n_estimators: The number of boosting rounds, capped at 100 iterations.
random_state: A seed value (42) for randomization to ensure reproducibility.
verbose: Set to 0, indicating no output during training.

Table 3: Different criteria for selecting the best tree in a Random-Forest

`max_depth`	The maximum depth of the individual trees
`n_estimators`	The number of trees in the forest
`min_samples_split`	The minimum number of samples required to split an internal node
`min_samples_leaf`	The minimum number of samples required to be at a leaf node
`max_features`	The number of features to consider when looking for the best split

For hyperparameter tuning of a Random Forest model using GridSearchCV, the following hyperparameter ranges can be defined:
n_estimators: Investigate values like [50, 100, 200, 300] to find the ideal number of decision trees in the forest. Increasing this number can enhance performance but may lead to higher computational costs.
max_depth: Widen the range to include values such as [None, 10, 20, 30] to control the maximum depth of individual decision trees. Deeper trees can capture complex patterns but may increase the risk of overfitting.
min_samples_split: Extend the range to values like [2, 5, 10, 15] to specify the minimum number of samples needed to split an internal node. Higher values promote simpler trees and help prevent overfitting.

min_samples_leaf: Test values like [1, 2, 4, 8] to determine the minimum number of samples required for a leaf node. Increasing this value can prevent overfitting by ensuring sufficient samples in each leaf.
max_features: Explore options like ['auto', 'sqrt', 'log2', None] to control how many features to consider when making splits. 'auto' uses all features, 'sqrt' uses the square root, 'log2' uses the base-2 logarithm, and None uses all features. This parameter affects feature selection during tree building.

Table 4: Different criteria for selecting the best tree in a Decision-Tree

`max_depth`	The maximum depth of the tree controls the maximum number of levels in the tree. A higher value can lead to overfitting
`min_samples_split`	The minimum number of samples required to split an internal node If a node has fewer samples than this value, it will not be split further
`min_samples_leaf`	The minimum number of samples required to be at a leaf node This parameter enforces a constraint on the size of leaf nodes
`max_features`	The number of features to consider when looking for the best split It can take different values like auto, sqrt, and log2

The provided hyperparameters represent a grid of values for tuning a decision tree model.
max_depth: It specifies the maximum depth of the decision tree. The grid includes options for unlimited depth (None) as well as limited depths of 10 and 20 levels.
min_samples_split: This parameter determines the minimum number of samples required to split a node. You have choices of 2, 5, and 10 samples.
min_samples_leaf: It sets the minimum number of samples required in a leaf node. The grid offers values for 1, 2, and 4 samples.
max_features: This parameter controls the number of features to consider when making a split. The grid includes options like 'auto' (all features),'sqrt' (square root of features), and 'log2' (log base 2 of features).

Table 5: Criteria for hyperplane maximization in the Support Vector Machine

`C`	Regularization parameter. Controls the trade-off between the training error and the complexity of the model.
`kernel`	Kernel function. The most common kernels are linear, polynomial, and Gaussian.

`degree`	Degree of the polynomial kernel. Only used for polynomial kernels.
`gamma`	Kernel coefficient. Controls the smoothness of the decision boundary.
`coef0`	Independent term in the kernel function.
`shrinking`	Whether to use the shrinking heuristic This heuristic helps to prevent overfitting.
`probability`	Whether to enable probability estimates This can be useful for calculating confidence intervals for predictions.
`class_weight`	Class weights. This can be used to give more weight to certain classes in the training data.

These hyperparameters control various aspects of the SVM model's behavior and are typically fine-tuned to optimize its performance for a specific task.

'C': The regularization parameter 'C,' which controls the trade-off between maximizing the margin and minimizing the classification error. It is set to the best value found during the model tuning process, defaulting to 1.0 if not specified.

'kernel': The choice of kernel function, which determines the transformation of the input data into a higher-dimensional space. The default kernel is 'rbf' (Radial Basis Function), but the best kernel is selected during tuning.

'degree': The degree of the polynomial kernel function if 'kernel' is set to 'poly.' It defaults to 3 but may take a different value if optimized during tuning.

'gamma': The kernel coefficient for 'rbf,' 'poly,' and ‘sigmoid’ kernels. It is set to 'scale' by default, but the best value is determined during tuning.

'coef0': An independent term in the kernel function. It is set to 0.0 by default but may vary if optimized during tuning.

'shrinking': A Boolean parameter that determines whether shrinking heuristics are used. It defaults to true.

'probability': A Boolean parameter indicating whether probability estimates should be calculated. It defaults to false but may be set to true if needed.

'class_weight': A parameter that specifies class weights for the SVM classes. It defaults to none but can be set to other values to address class imbalance issues.

Table 6: Criteria for hyperplane maximization in the Naïve - Bayes

'alpha'	Represents the Laplace smoothing parameter.

Hyperparameter tuning for a Naïve Bayes model is performed using GridSearchCV, with a focus on the **'alpha'** hyperparameter. The grid includes values of 0.1, 1.0, and 10.0, enabling the selection of the optimal smoothing parameter to enhance model performance and accuracy.

In order to determine the ideal set of hyperparameters for the selected classifier, a randomized search with cross-validation is used. Cross-validation is a technique that ensures the model is trained and tested on new data at each step by splitting the dataset into multiple portions. Some portions are used for training, while others are used for testing. Instead of dividing the dataset into two parts for training and testing, we use multiple subsets to train and evaluate our model. The fit() method is used to train the dataset. The fit() function typically uses the input features and their corresponding target values to enable the model to learn patterns and make predictions. It also adjusts the model to fit the training data.

5. Results and Analysis

In this work, Python (version 3.10) is used with the required built-in libraries. The dissertation uses an integrated strategy of machine learning techniques to evaluate the academic achievement of schoolchildren. The Kaggle repository is used to pull an example dataset. Scaling operations and data pre-processing follow the selection of pertinent characteristics from the dataset. After dividing the dataset into a train set and a test set, a number of machine learning classifiers are constructed, including Support Vector Machine (SVM), Random Forest (RF), Decision Tree (DT), XGBoost, CatBoost, Naive Bayes, and Light GBM models. For improved classification accuracy, hyperparameter modifications are also applied. Accuracy, precision, and recall are the performance indicators for categorization model performance.

Classification report of XG Boost model

The XG Boost algorithm achieves high accuracy (0.89), high precision (0.91), high recall (0.97), and a balanced F1-Score (0.94). These results show the algorithm's ability to make precise forecasts of positives and capture the majority of positive instances, indicating that it is well-suited for the task at hand. Here's an extended range for some of the hyperparameters.

Classification report of Cat Boost model

The Cat Boost algorithm has an accuracy of 0.89, indicating that it correctly predicts outcomes in approximately 89% of cases. It also has high precision (0.90), which means it accurately predicts positive cases, and high recall (0.98), which means it effectively identifies nearly all positive cases. The F1-Score is 0.94, indicating a harmonious balance between making accurate positive predictions and capturing the majority of actual positive instances.

Classification report of Light GBM model

The Light GBM algorithm achieves an accuracy of 0.88, indicating that it correctly predicts outcomes for approximately 88% of instances. It also demonstrates vital precision (0.90), which accurately predicts positive cases, and high recall (0.97), indicating that it effectively identifies a significant portion of actual positive instances. The F1-Score, which balances precision and recall, is 0.94, demonstrating a harmonious balance between accurately predicting and capturing the most actual positive instances.

Classification report of Decision Tree model

The decision tree model performed well, achieving an accuracy rate of 100%. The performance metrics for a Decision Tree algorithm in a specific task The decision tree model notably produces outstanding results, with an accuracy of 1.00 indicating that it accurately predicts outcomes in all cases. Furthermore, it has perfect precision (1.00), indicating that the model's positive predictions are accurate. While the recall is very high at 0.99, indicating that the model effectively captures the majority of actual positive instances, the F1-Score is perfect at 1.00, indicating an exceptional balance between precision and recall.

Classification report of Random Forest model

The Random Forest algorithm achieves an accuracy of 0.90, indicating that it correctly predicts outcomes for approximately 90% of instances. It also demonstrates strong precision (0.90), meaning it accurately predicts positive cases, and a very high recall (0.99), indicating it effectively identifies nearly all actual positive instances. The F1-Score, which balances precision and recall, is 0.95, demonstrating a harmonious balance between accurately predicting positive instances and capturing the majority of actual positive instances. Overall, the Random Forest model performs very well in this task, delivering reliable results.

Classification report of Support Vector Machine model

The Support Vector Machine (SVM) model exhibits outstanding performance in a specific task, with a high accuracy of 0.96, strong precision of 0.96, perfect recall of 1.00, and an impressive F1-Score of 0.98. This indicates its exceptional ability to accurately predict positive cases while effectively capturing all actual positive instances.

Classification report of Naïve Bayes Classification model

The Naïve Bayes algorithm achieves an impressive accuracy of 0.96, indicating that it correctly predicts outcomes for approximately 96% of instances. It also demonstrates strong precision (0.97), meaning it accurately predicts positive cases, and high recall (0.99), indicating that it effectively identifies the majority of actual positive instances. The F1-Score, which balances precision and recall, is impressive at 0.98, demonstrating a harmonious balance between accurately predicting positive instances and capturing most actual positive instances.

Table 14 provides a comprehensive overview of the evaluation metrics, including Accuracy, Precision, Recall, and F1-Score, for all the algorithms utilized in this analysis. Additionally, Figure 9 visually illustrates the accuracy and performance of these algorithms.

Table 14: Performance Score Table Classification Report of Various ML Algorithm

Algorithms	Accuracy	Precision	Recall	F1-Score
XG Boost	0.89	0.91	0.97	0.94
Cat Boost	0.89	0.90	0.98	0.94
Light GBM	0.88	0.90	0.97	0.94
Decision Tree	1.00	1.00	0.99	1.00
Random Forest	0.90	0.90	0.99	0.95
SVM	0.96	0.96	1.00	0.98
Naïve Bayes	0.96	0.97	0.99	0.98

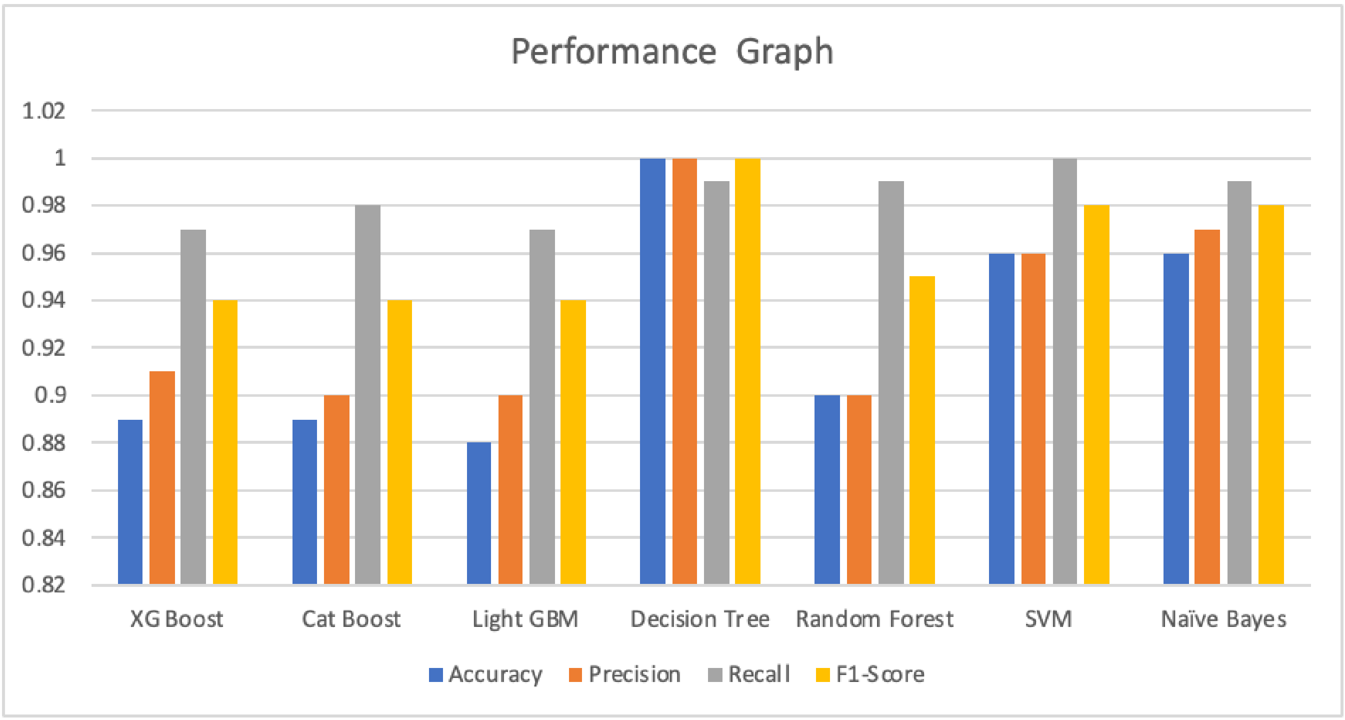


Figure 9: Performance Chart of Various Classification Models

6. Conclusion and Future Work

In conclusion, this work focused on enhancing the predictive accuracy of ensemble classification methods. This study employed a comparative analysis of several machine learning models, including XG Boost, Cat Boost, Naïve-Bayes, Light GBM, Random Forest, Decision Tree, and Support Vector Machine, for the critical task of predicting student academic performance. Feature selection via ANOVA plays a crucial role in improving model efficiency and identifying at-risk students' poor performance in school and university. These findings have significant implications for educational institutions seeking to enhance student outcomes through data-driven interventions. In evaluating the effectiveness of classification models, accuracy, precision, and recall are used as metrics. Based on experimental analysis results, it has been found that the XGBoost algorithm outperforms models in terms of efficiency.

In the future, further research and improvements can be explored in several areas. Firstly, incorporating advanced learning techniques like neural networks, recurrent neural networks (RNNs), or convolutional neural networks (CNNs) can potentially enhance prediction accuracy by capturing intricate patterns and dependencies within the data. Secondly, exploring how natural language processing (NLP) models can analyze students' written assignments and comments to gain insights into the factors influencing their performance would be valuable. Additionally, integrating real-time data and adaptive learning systems could enable the provision of timely support to struggling students, potentially leading to higher retention rates.

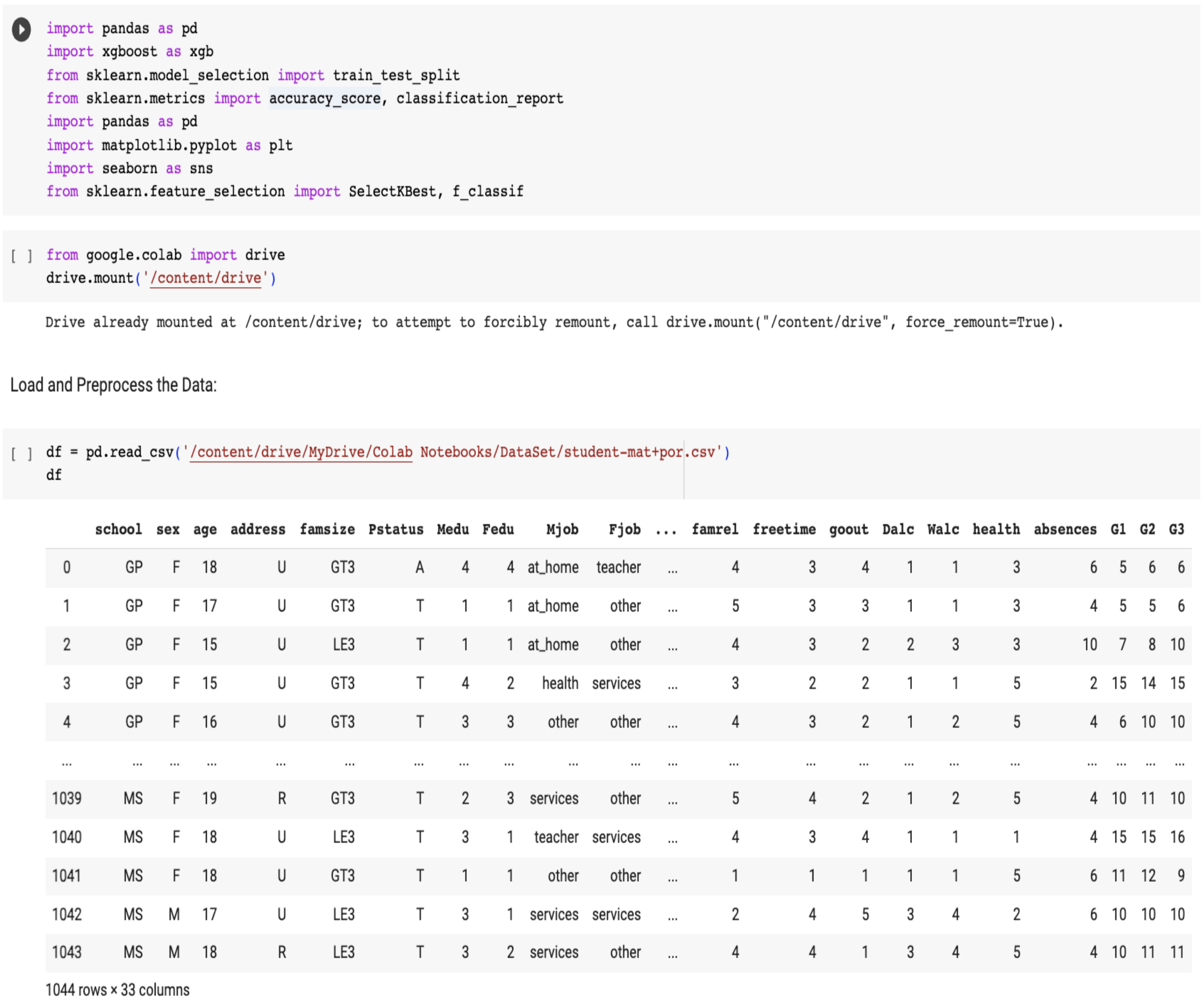
Lastly, investigating the impact of factors such as socioeconomic status and access to resources on student performance can provide a deeper understanding of the issue, facilitating the implementation of targeted interventions.

7. Reference

1. N. S. Kasumurthy Deepak, V. S. Senthil Kumar, and M. Nagaraj, "Analysis of student’s knowledge using innovative classification and comparison of prediction accuracy with regression algorithm," *International Journal of Emerging Technologies in Learning*, vol. 17, no. 1, pp. 1-12, 2022.
2. K. Harikumar Pallathadka, T. S. Anoop, and K. Manjunath, "A Survey of Machine Learning Techniques for Student Performance Prediction," *International Journal of Engineering Research & Technology,* vol. 10, no. 8, pp. 285-290, 2021.
3. F. Qiu, Y. Zhang, H. Li, and S. Lin, "Predicting students’ performance in e‐learning using learning process and behaviour data," Computers & Education, vol. 161, pp. 103851-103961, 2021.
4. H. Gull, M. Jamil, and M. A. Khan, "Improving Learning Experience of Students by Early Prediction of Student Performance Using Machine Learning," *Education and Information Technologies*, vol. 27, no. 1, pp. 357-379, 2022.
5. J. Figueroa-Cañas, J. M. Mas-Verdu, M. A. García-Peñalvo, and J. M. Martínez-Monés, "Early Prediction of Dropout and Final Exam Performance in an Online Statistics Course," *IEEE Transactions on Learning Technologies*, vol. 13, no. 1, pp. 58-72, 2020.
6. H. Farooq and T. Mehmood, "A review of machine learning techniques for student performance prediction in higher education," *International Journal of Engineering Education*, vol. 37, no. 1, pp. 1-17, 2021.
7. H. Farooq and T. Mehmood, "A review of machine learning techniques for student performance prediction in higher education," *International Journal of Engineering Education*, vol. 37, no. 1, pp. 1-17, 2021.
8. S. Gupta, S. Kumar, and V. Gupta, "Machine learning approaches for student performance prediction: A systematic review," IEEE Access, vol. 10, pp. 25491-25513, 2022.

9. A. Kumar, R. Singh, and Y. Singh, "Student performance prediction using machine learning techniques: A systematic review," *Education and Information Technologies*, vol. 26, no. 6, pp. 5301-5323, 2021.
10. R. Patel, V. P. Chaudhari, B. Patel, and A. Patil, "A review of machine learning techniques for student performance prediction," *International Journal of Engineering Research and Technology*, vol. 10, no. 11, pp. 663-667, 2021.
11. H. Gull, M. Saqib, S. Z. Iqbal, and S. Saeed, "Improving Learning Experience of Students by Early Prediction of Student Performance Using Machine Learning," Education and Information Technologies, pp. 357-379, 2022.doi:10.1007/s10639-022-10005-6.
12. R. Suleiman and R. Anane, "Institutional Data Analysis and Machine Learning Prediction of Student Performance," in 2022 IEEE 25th International Conference on Computer Supported Cooperative Work in Design (CSCWD), pp. 1480-1485, 2022, doi: 10.1109/CSCWD55351.2022.00275.
13. Y. Chen, J. Ma, X. Wang, and J. Gao, "A novel ensemble learning method for student academic performance prediction," Expert Systems with Applications, vol. 187, pp. 115827-115829, 2022.
14. S. D. Abdul Bujang, A. Selamat, R. Ibrahim, O. Krejcar, E. Herrera-Viedma, H. Fujita, and N. A. Md. Ghani, "Multiclass Prediction Model for Student Grade Prediction Using Machine Learning," IEEE Access, vol. 10, pp. 91338-91351, 2022.
15. C.C. Kiu, "Data mining analysis on student's academic performance through exploration of student's background and social activities," Open Access Library Journal, vol. 5, no. 1, pp. 4174-4191, 2018. doi:10.4236/oalib.1104174.

Appendix



```
import pandas as pd
import xgboost as xgb
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.feature_selection import SelectKBest, f_classif
```

```
[ ] from google.colab import drive
    drive.mount('/content/drive')
```

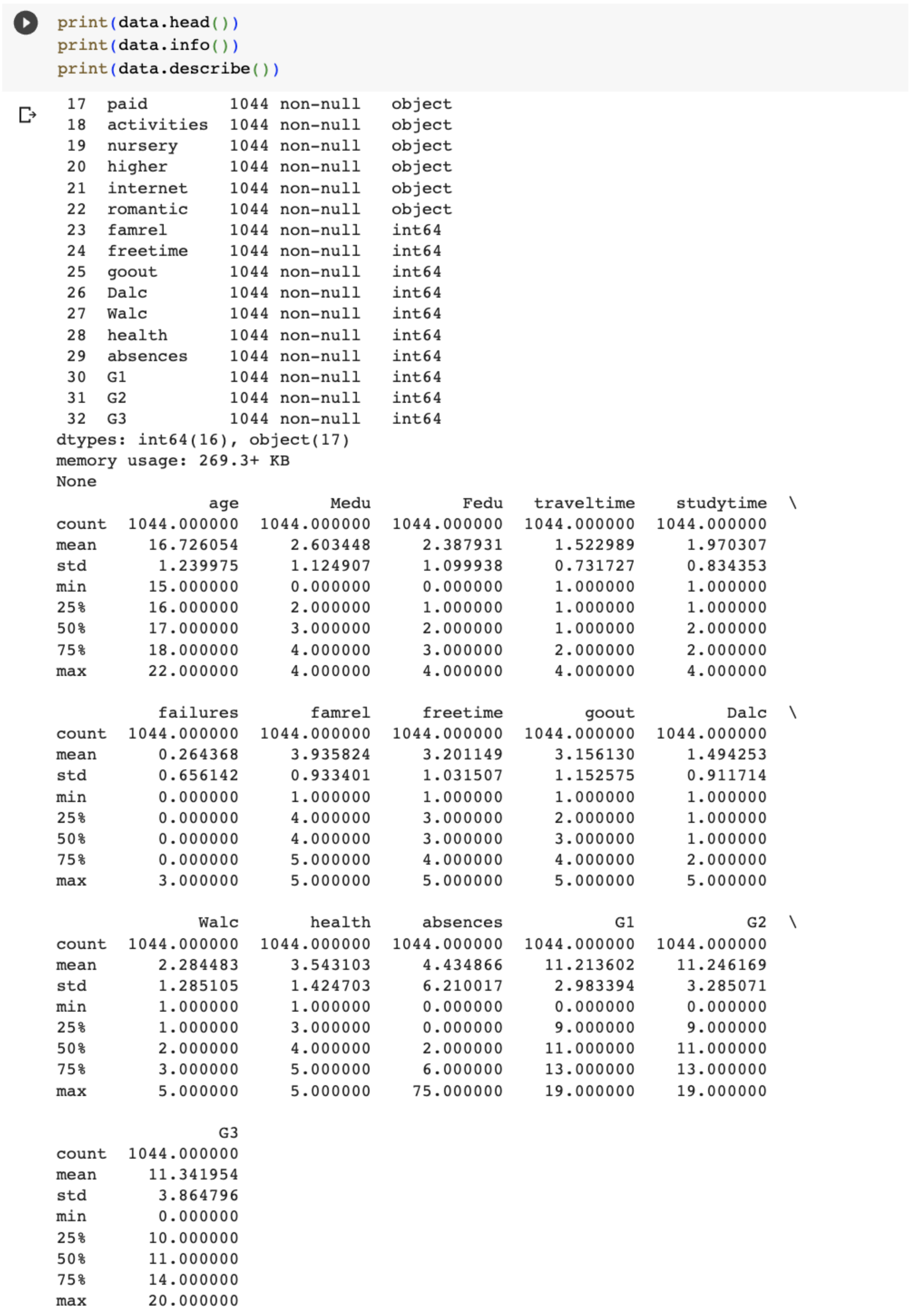
```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

Load and Preprocess the Data:

```
[ ] df = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/DataSet/student-mat+por.csv')
    df
```

	school	sex	age	address	famsize	Pstatus	Medu	Fedu	Mjob	Fjob	...	famrel	freetime	goout	Dalc	Walc	health	absences	G1	G2	G3
0	GP	F	18	U	GT3	A	4	4	at_home	teacher	...	4	3	4	1	1	3	6	5	6	6
1	GP	F	17	U	GT3	T	1	1	at_home	other	...	5	3	3	1	1	3	4	5	5	6
2	GP	F	15	U	LE3	T	1	1	at_home	other	...	4	3	2	2	3	3	10	7	8	10
3	GP	F	15	U	GT3	T	4	2	health	services	...	3	2	2	1	1	5	2	15	14	15
4	GP	F	16	U	GT3	T	3	3	other	other	...	4	3	2	1	2	5	4	6	10	10
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1039	MS	F	19	R	GT3	T	2	3	services	other	...	5	4	2	1	2	5	4	10	11	10
1040	MS	F	18	U	LE3	T	3	1	teacher	services	...	4	3	4	1	1	1	4	15	15	16
1041	MS	F	18	U	GT3	T	1	1	other	other	...	1	1	1	1	1	5	6	11	12	9
1042	MS	M	17	U	LE3	T	3	1	services	services	...	2	4	5	3	4	2	6	10	10	10
1043	MS	M	18	R	LE3	T	3	2	services	other	...	4	4	1	3	4	5	4	10	11	11

1044 rows × 33 columns



```
print(data.head())
print(data.info())
print(data.describe())
```

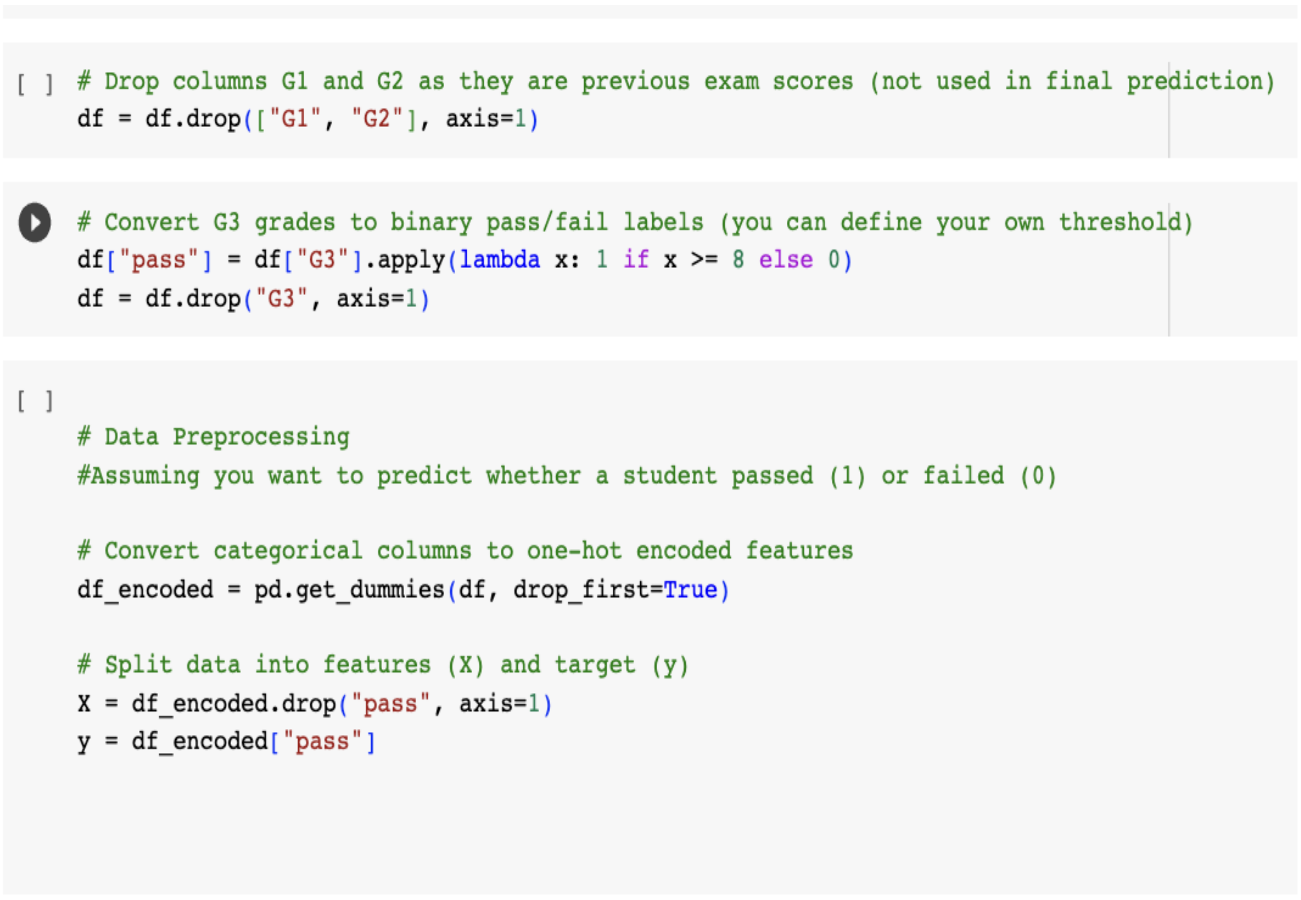
```
 17  paid        1044 non-null   object
 18  activities  1044 non-null   object
 19  nursery     1044 non-null   object
 20  higher      1044 non-null   object
 21  internet    1044 non-null   object
 22  romantic    1044 non-null   object
 23  famrel      1044 non-null   int64
 24  freetime    1044 non-null   int64
 25  goout       1044 non-null   int64
 26  Dalc        1044 non-null   int64
 27  Walc        1044 non-null   int64
 28  health      1044 non-null   int64
 29  absences    1044 non-null   int64
 30  G1          1044 non-null   int64
 31  G2          1044 non-null   int64
 32  G3          1044 non-null   int64
dtypes: int64(16), object(17)
memory usage: 269.3+ KB
None
               age         Medu         Fedu   traveltime    studytime  \
count  1044.000000  1044.000000  1044.000000  1044.000000  1044.000000
mean     16.726054     2.603448     2.387931     1.522989     1.970307
std       1.239975     1.124907     1.099938     0.731727     0.834353
min      15.000000     0.000000     0.000000     1.000000     1.000000
25%      16.000000     2.000000     1.000000     1.000000     1.000000
50%      17.000000     3.000000     2.000000     1.000000     2.000000
75%      18.000000     4.000000     3.000000     2.000000     2.000000
max      22.000000     4.000000     4.000000     4.000000     4.000000

          failures       famrel     freetime        goout         Dalc  \
count  1044.000000  1044.000000  1044.000000  1044.000000  1044.000000
mean      0.264368     3.935824     3.201149     3.156130     1.494253
std       0.656142     0.933401     1.031507     1.152575     0.911714
min       0.000000     1.000000     1.000000     1.000000     1.000000
25%       0.000000     4.000000     3.000000     2.000000     1.000000
50%       0.000000     4.000000     3.000000     3.000000     1.000000
75%       0.000000     5.000000     4.000000     4.000000     2.000000
max       3.000000     5.000000     5.000000     5.000000     5.000000

              Walc       health     absences           G1           G2  \
count  1044.000000  1044.000000  1044.000000  1044.000000  1044.000000
mean      2.284483     3.543103     4.434866    11.213602    11.246169
std       1.285105     1.424703     6.210017     2.983394     3.285071
min       1.000000     1.000000     0.000000     0.000000     0.000000
25%       1.000000     3.000000     0.000000     9.000000     9.000000
50%       2.000000     4.000000     2.000000    11.000000    11.000000
75%       3.000000     5.000000     6.000000    13.000000    13.000000
max       5.000000     5.000000    75.000000    19.000000    19.000000

                G3
count  1044.000000
mean     11.341954
std       3.864796
min       0.000000
25%      10.000000
50%      11.000000
75%      14.000000
max      20.000000
```

Data Pre-processing



```
[ ] # Drop columns G1 and G2 as they are previous exam scores (not used in final prediction)
    df = df.drop(["G1", "G2"], axis=1)
```

```
    # Convert G3 grades to binary pass/fail labels (you can define your own threshold)
    df["pass"] = df["G3"].apply(lambda x: 1 if x >= 8 else 0)
    df = df.drop("G3", axis=1)
```

```
[ ]
    # Data Preprocessing
    #Assuming you want to predict whether a student passed (1) or failed (0)

    # Convert categorical columns to one-hot encoded features
    df_encoded = pd.get_dummies(df, drop_first=True)

    # Split data into features (X) and target (y)
    X = df_encoded.drop("pass", axis=1)
    y = df_encoded["pass"]
```

XGBoost classification model, Statistical-based feature selection techniques, and hyperparameters

```
import pandas as pd
import numpy as np
import xgboost as xgb
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.feature_selection import SelectKBest, f_classif
from sklearn.metrics import accuracy_score, classification_report



# Perform feature selection using SelectKBest with ANOVA F-statistic
num_features_to_select = 10 # Adjust this number as needed
selector = SelectKBest(score_func=f_classif, k=num_features_to_select)
X_train_selected = selector.fit_transform(X_train, y_train)
X_test_selected = selector.transform(X_test)
```

```
# Define a parameter grid to search through
param_grid = {
'max_depth': [3, 4, 5],
'learning_rate': [0.1, 0.01, 0.001],
'n_estimators': [100, 200, 300],
'gamma': [0, 0.1, 0.2],
'subsample': [0.8, 0.9, 1.0],
'colsample_bytree': [0.8, 0.9, 1.0],
}

# Create an XGBoost classifier
xgb_model = xgb.XGBClassifier()
```

```
# Perform grid search with cross-validation
grid_search = GridSearchCV(estimator=xgb_model, param_grid=param_grid,
scoring='accuracy', cv=3)
grid_search.fit(X_train_selected, y_train)
```

```
                  GridSearchCV
             estimator: XGBClassifier
                  XGBClassifier
```

```
# Get the best hyperparameters
best_params = grid_search.best_params_

# Train an XGBoost classifier with the best hyperparameters
best_xgb_model = xgb.XGBClassifier(**best_params)
best_xgb_model.fit(X_train_selected, y_train)
```

```
                                         XGBClassifier
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=0.9, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              gamma=0.2, gpu_id=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=0.1, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=5, max_leaves=None,
              min_child_weight=None, missing=nan, monotone_constraints=None,
              n_estimators=100, n_jobs=None, num_parallel_tree=None,
              predictor=None, random_state=None, ...)
```

```
# Make predictions on the selected features of the test set
y_pred_xgb = best_xgb_model.predict(X_test_selected)

# Evaluate the model's performance
accuracy = accuracy_score(y_test, y_pred_xgb)
print("Best Hyperparameters:", best_params)
print("Accuracy:", accuracy)
```

```
Best Hyperparameters: {'colsample_bytree': 0.9, 'gamma': 0.2, 'learning_rate':
0.1, 'max_depth': 5, 'n_estimators': 100, 'subsample': 0.8} Accuracy:
0.8947368421052632
```

```
# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred_xgb)
print(f"Accuracy: {accuracy:.2f}")

# Print classification report
classification_rep = classification_report(y_test, y_pred_xgb)
print("Classification Report:\n", classification_rep)
```

```
Accuracy: 0.89
Classification Report:
               precision    recall  f1-score   support

           0       0.50      0.23      0.31        22
           1       0.91      0.97      0.94       187

    accuracy                           0.89       209
   macro avg       0.71      0.60      0.63       209
weighted avg       0.87      0.89      0.88       209
```

CatBoost Classifier model Statistical-based feature selection techniques and using hyper-parameters

```
pip install CatBoost
```

```
import pandas as pd
import numpy as np
from catboost import CatBoostClassifier, Pool
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.feature_selection import SelectKBest, f_classif
from sklearn.metrics import accuracy_score

# Perform feature selection using SelectKBest with ANOVA F-statistic
```

```
num_features_to_select = 10 # Adjust this number as needed
selector = SelectKBest(score_func=f_classif, k=num_features_to_select)
X_train_selected = selector.fit_transform(X_train, y_train)
X_test_selected = selector.transform(X_test)

# Define a parameter grid to search through
param_grid = {
'depth': [4, 6, 8],
'learning_rate': [0.01, 0.1, 0.2],
'iterations': [500, 1000, 1500],
'l2_leaf_reg': [1, 3, 5],
'border_count': [32, 64, 128],
}
# Create a CatBoostClassifier
catboost_model = CatBoostClassifier()
# Perform grid search with cross-validation
grid_search = GridSearchCV(estimator=catboost_model, param_grid=param_grid,
scoring='accuracy', cv=3)
grid_search.fit(X_train_selected, y_train)
# Get the best hyperparameters
best_params = grid_search.best_params_

# Train a CatBoostClassifier with the best hyperparameters
best_catboost_model = CatBoostClassifier(**best_params)
best_catboost_model.fit(X_train_selected, y_train)

# Make predictions on the selected features of the test set
y_pred_cb = best_catboost_model.predict(X_test_selected)

# Evaluate the model's performance
accuracy = accuracy_score(y_test, y_pred_cb)
print("Best Hyperparameters:", best_params)
print("Accuracy:", accuracy)
Best Hyperparameters: OrderedDict([('colsample_bytree', 0.8773887589868244),
('learning_rate', 0.012955998488203432), ('max_depth', 6),
('min_child_samples', 9), ('n_estimators', 195), ('num_leaves', 11),
('subsample', 0.8747060157801201)]) Accuracy: 0.8851674641148325
```

```
# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred_cb)
print(f"Accuracy: {accuracy:.2f}")

# Print classification report
classification_rep = classification_report(y_test, y_pred_cb)
print("Classification Report:\n", classification_rep)
```

```
Accuracy: 0.89
Classification Report:
               precision    recall  f1-score   support

           0       0.25      0.05      0.08        22
           1       0.90      0.98      0.94       187

    accuracy                           0.89       209
   macro avg       0.57      0.51      0.51       209
weighted avg       0.83      0.89      0.85       209
```

Light GBM model, Statistical-based feature selection techniques and using hyper-parameters

```
pip install lightgbm scikit-learn scikit-optimize
pip install scikit-optimize
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import accuracy_score
from skopt import BayesSearchCV
import lightgbm as lgb
from skopt.space import Real, Categorical, Integer


# Perform label encoding for categorical features (if needed)
categorical_cols = X.select_dtypes(include=['object']).columns
for col in categorical_cols:
le = LabelEncoder()
X[col] = le.fit_transform(X[col])
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
```

Perform statistical-based feature selection using, for example, SelectKBest from scikit-learn

```
from sklearn.feature_selection import SelectKBest, f_classif
# Perform feature selection using SelectKBest with ANOVA F-statistic
num_features_to_select = 10 # Adjust this number as needed
selector = SelectKBest(score_func=f_classif, k=num_features_to_select)
X_train_selected = selector.fit_transform(X_train, y_train)
X_test_selected = selector.transform(X_test

# Define the hyperparameter search space
hyperparam_space = {
'num_leaves': Integer(10, 100), # Number of leaves in each tree
'learning_rate': Real(0.01, 0.1, 'log-uniform'), # Learning rate with
log-uniform prior
'n_estimators': Integer(50, 200), # Number of boosting rounds
'max_depth': Integer(3, 15), # Maximum depth of trees
'min_child_samples': Integer(2, 20), # Minimum number of data points in leaves
'subsample': Real(0.5, 1.0, 'uniform'), # Fraction of data used for training
each tree
'colsample_bytree': Real(0.5, 1.0, 'uniform'), # Fraction of features used for
each tree
}
```

Perform Bayesian optimisation for hyperparameter tuning

```
# Create the BayesSearchCV object for optimization
opt = BayesSearchCV(
lgb.LGBMClassifier(random_state=42),
hyperparam_space,
n_iter=50, # Adjust the number of iterations as needed
cv=3, # Number of cross-validation folds
n_jobs=-1, # Use all available CPU cores
scoring='accuracy', # Choose the appropriate scoring metric
random_state=42
)



opt.fit(X_train_selected, y_train)

best_params = opt.best_params_
```

Train the LightGBM model with the best hyperparameters

```
best_lgb_model = lgb.LGBMClassifier(**best_params)
best_lgb_model.fit(X_train_selected, y_train)

# Make predictions on the selected features of the test set
y_pred_lgb = best_lgb_model.predict(X_test_selected)

# Evaluate the model's performance
accuracy = accuracy_score(y_test, y_pred_lgb)
print("Best Hyperparameters:", best_params)
print("Accuracy:", accuracy)




# Get the best hyperparameters
best_params = opt.best_params_
best_score = opt.best_score_
print("Best Hyperparameters:", best_params)
print("Best Accuracy:", best_score)
# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred_lgb)
print(f"Accuracy: {accuracy:.2f}")
# Print classification report
classification_rep = classification_report(y_test, y_pred_lgb)
print("Classification Report:\n", classification_rep)
```

```
Best Hyperparameters: OrderedDict([('colsample_bytree',
0.8773887589868244), ('learning_rate', 0.012955998488203432),
('max_depth', 6), ('min_child_samples', 9), ('n_estimators', 195),
('num_leaves', 11), ('subsample', 0.8747060157801201)]) Best
Accuracy: 0.9077770042030892

Accuracy: 0.88 Classification Report:
 Accuracy: 0.88
 Classification Report:
               precision    recall  f1-score   support

           0       0.29      0.09      0.14        22
           1       0.90      0.97      0.94       187

    accuracy                           0.88       209
   macro avg       0.59      0.53      0.54       209
weighted avg       0.84      0.88      0.85       209
```

Random Forest model, Statistical-based feature selection techniques and using hyper-parameters

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.ensemble import RandomForestClassifier
from sklearn.feature_selection import SelectKBest, f_classif
from sklearn.metrics import accuracy_score
```

Feature Selection:

Perform feature selection using SelectKBest with ANOVA F-statistic. Adjust the value of num_features_to_select based on how many features you want to select.

```
num_features_to_select = 10
selector = SelectKBest(f_classif, k=num_features_to_select)
X_train_selected = selector.fit_transform(X_train, y_train)
X_test_selected = selector.transform(X_test)
```

Initialize the Random Forest Classifier

```
classifier = RandomForestClassifier()
```

Hyperparameter Tuning using GridSearchCV

hyperparameter grid (param_grid) for hyperparameter tuning using GridSearchCV.

```
param_grid = {
'n_estimators': [50, 100, 200],
'max_depth': [None, 10, 20],
'min_samples_split': [2, 5, 10],
'min_samples_leaf': [1, 2, 4],
'max_features': ['auto', 'sqrt', 'log2']
}

#Grid Search for Best Parameters:
grid_search = GridSearchCV(classifier, param_grid, cv=5, scoring='accuracy')
grid_search.fit(X_train_selected, y_train)
#Get the Best Hyperparameters:
best_params = grid_search.best_params_
```

```
#Train the Classifier with the Best Hyperparameters:
```

```
best_classifier = RandomForestClassifier(**best_params)
best_classifier.fit(X_train_selected, y_train)
```

```
                         RandomForestClassifier
RandomForestClassifier(max_depth=10, min_samples_leaf=2, min_samples_split=5,
                       n_estimators=50)
```

```
y_pred_rf = best_classifier.predict(X_test_selected)
accuracy = accuracy_score(y_test, y_pred_rf)
print(f"Accuracy: {accuracy:.2f}")
```

```
Accuracy: 0.90
```

```
 # Calculate accuracy
accuracy = accuracy_score(y_test, y_pred_rf)
print(f"Accuracy: {accuracy:.2f}")

# Print classification report
classification_rep = classification_report(y_test, y_pred_rf)
print("Classification Report:\n", classification_rep)
```

```
Accuracy: 0.90
Classification Report:
               precision    recall  f1-score   support

           0       0.67      0.09      0.16        22
           1       0.90      0.99      0.95       187

    accuracy                           0.90       209
   macro avg       0.78      0.54      0.55       209
weighted avg       0.88      0.90      0.86       209
```

Decision tree model, Statistical-based feature selection techniques and using hyper-parameters

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.tree import DecisionTreeClassifier
from sklearn.feature_selection import SelectKBest, f_classif
from sklearn.metrics import accuracy_score


num_features_to_select = 10
selector = SelectKBest(f_classif, k=num_features_to_select)
X_train_selected = selector.fit_transform(X_train, y_train)
X_test_selected = selector.transform(X_test)

classifier = DecisionTreeClassifier()

param_grid = {
'max_depth': [None, 10, 20],
'min_samples_split': [2, 5, 10],
'min_samples_leaf': [1, 2, 4],
'max_features': ['auto', 'sqrt', 'log2']
}

grid_search = GridSearchCV(classifier, param_grid, cv=5, scoring='accuracy')
grid_search.fit(X_train_selected, y_train)
```

```
GridSearchCV

estimator: DecisionTreeClassifier

DecisionTreeClassifier
```

```
best_params = grid_search.best_params_
best_classifier = DecisionTreeClassifier(**best_params)
best_classifier.fit(X_train_selected, y_train)
```

```
DecisionTreeClassifier
DecisionTreeClassifier(max_depth=10, max_features='sqrt', min_samples_leaf=2)
```

```
y_pred_dt = best_classifier.predict(X_test_selected)
```

```
# Calculate accuracy

accuracy = accuracy_score(y_test, y_pred_dt)
print(f"Accuracy: {accuracy:.2f}")

# Print classification report
classification_rep = classification_report(y_test, y_pred_dt)
print("Classification Report:\n", classification_rep)
```

```
Accuracy: 1.00
Classification Report:
               precision    recall  f1-score   support

        Fail       1.00      0.95      0.98        22
        Pass       0.99      1.00      1.00       187

    accuracy                           1.00       209
   macro avg       1.00      0.98      0.99       209
weighted avg       1.00      1.00      1.00       209
```

Support Vector Machine model, Statistical-based feature selection techniques and using hyper-parameters

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.svm import SVC
from sklearn.feature_selection import SelectKBest, f_classif
from sklearn.metrics import accuracy_score

num_features_to_select = 10
selector = SelectKBest(f_classif, k=num_features_to_select)
X_train_selected = selector.fit_transform(X_train, y_train)
X_test_selected = selector.transform(X_test)
#Initialize the Support Vector Machine Classifier:
classifier = SVC()
```

```
#Grid Search for Best Parameters:
param_grid = {
'C': [0.1, 1, 10],
'kernel': ['linear', 'rbf', 'poly'],
'gamma': ['scale', 'auto'] + [0.001, 0.01, 0.1, 1]
}
#Get the Best Hyperparameters:
best_params = grid_search.best_params_
valid_params = {
'C': best_params.get('C', 1.0), # Regularization parameter
'kernel': best_params.get('kernel', 'rbf'), # Kernel function ('linear', 'rbf',
etc.)
# Add other valid hyperparameters as needed

'degree': best_params.get('degree', 3), # Degree of the polynomial kernel
'gamma': best_params.get('gamma', 'scale'), # Kernel coefficient ('scale',
'auto', float)
'coef0': best_params.get('coef0', 0.0), # Independent term in the kernel
function
'shrinking': best_params.get('shrinking', True), # Whether to use the shrinking
heuristic
'probability': best_params.get('probability', False), # Whether to enable
probability estimates
'class_weight': best_params.get('class_weight', None), # Class weights (dict or
'balanced')
}

# Initialize the SVC classifier with the best hyperparameters
best_classifier = SVC(**valid_params)

best_classifier.fit(X_train_selected, y_train)
```

SVC

SVC()

```
y_pred_svc = best_classifier.predict(X_test_selected)

# Calculate accuracy

accuracy = accuracy_score(y_test, y_pred_svc)

print(f"Accuracy: {accuracy:.2f}")

# Print classification report

classification_rep = classification_report(y_test, y_pred_svc)

print("Classification Report:\n", classification_rep)
```

```
Accuracy: 0.96
Classification Report:
               precision    recall  f1-score   support

        Fail       1.00      0.64      0.78        22
        Pass       0.96      1.00      0.98       187

    accuracy                           0.96       209
   macro avg       0.98      0.82      0.88       209
weighted avg       0.96      0.96      0.96       209
```

Naïve Bayes Statistical-based feature selection techniques and using hyper-parameters

```
# Import necessary libraries
import numpy as np
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.feature_selection import SelectKBest, chi2
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score

# Feature selection using SelectKBest and Chi-squared test
num_features_to_select = 25
selector = SelectKBest(score_func=chi2, k=num_features_to_select)
X_train_selected = selector.fit_transform(X_train, y_train)
X_test_selected = selector.transform(X_test)

# Create a Naïve Bayes classifier
naive_bayes = MultinomialNB()

# Hyper-parameter tuning using GridSearchCV
param_grid = {'alpha': [0.1, 1.0, 10.0]}
grid_search = GridSearchCV(estimator=naive_bayes, param_grid=param_grid, cv=5)
grid_search.fit(X_train_selected, y_train)
best_alpha = grid_search.best_params_['alpha']

# Train the Naïve Bayes classifier with the best hyper-parameter
final_naive_bayes = MultinomialNB(alpha=best_alpha)
final_naive_bayes.fit(X_train_selected, y_train)

# Make predictions on the test set
y_pred_nb = final_naive_bayes.predict(X_test_selected)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred_nb)
print(f"Accuracy: {accuracy:.2f}")

# Print classification report
classification_rep = classification_report(y_test, y_pred_nb)
print("Classification Report:\n", classification_rep)
```

```
Accuracy: 0.96
Classification Report:
               precision    recall  f1-score   support

        Fail       0.89      0.73      0.80        22
        Pass       0.97      0.99      0.98       187

    accuracy                           0.96       209
   macro avg       0.93      0.86      0.89       209
weighted avg       0.96      0.96      0.96       209
```

K-Nearest Neighbors (KNN) model with statistical-based feature selection techniques and hyperparameter tuning

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.neighbors import KNeighborsClassifier
from sklearn.feature_selection import SelectKBest, f_classif
from sklearn.metrics import accuracy_score

num_features_to_select = 10
selector = SelectKBest(f_classif, k=num_features_to_select)
X_train_selected = selector.fit_transform(X_train, y_train)
X_test_selected = selector.transform(X_test)

classifier = KNeighborsClassifier()

# Define hyperparameter grid for tuning
param_grid = {
'n_neighbors': [3, 5, 7, 9], # Adjust the range as needed
'weights': ['uniform', 'distance'],
'p': [1, 2] # 1 for Manhattan distance, 2 for Euclidean distance
}
grid_search = GridSearchCV(classifier, param_grid, cv=5, scoring='accuracy')
grid_search.fit(X_train_selected, y_train)
```

```
GridSearchCV
estimator: KNeighborsClassifier
KNeighborsClassifier
```

```
best_params = {
'n_neighbors': 5, # Adjust the number of neighbors as needed
'metric': 'euclidean', # Adjust the metric as needed (e.g., 'euclidean',
'manhattan')
'weights': 'uniform', # Adjust the weight function as needed ('uniform' or
'distance')
}
best_classifier = KNeighborsClassifier(**best_params)
best_classifier.fit(X_train_selected, y_train)
y_pred_knn = best_classifier.predict(X_test_selected)
accuracy = accuracy_score(y_test, y_pred_knn)

# Calculate accuracy or other relevant metrics
from sklearn.metrics import accuracy_score, classification_report
accuracy = accuracy_score(y_test, y_pred_knn)
report = classification_report(y_test, y_pred_knn)
print(f"Best KNN Model Accuracy: {accuracy}")
print("Classification Report:\n", report)
```

```
Accuracy: 0.97
Best KNN Model Accuracy: 0.9712918660287081
Classification Report:
               precision    recall  f1-score   support

        Fail       0.94      0.77      0.85        22
        Pass       0.97      0.99      0.98       187

    accuracy                           0.97       209
   macro avg       0.96      0.88      0.92       209
weighted avg       0.97      0.97      0.97       209
```