Eötvös Loránd University

Faculty of Informatics

Dept. of Software Technology and Methodology

Leveraging Machine Learning to Predict Student Performance in Computer Science Courses

Supervisor:
Dr. Máté Cserép
Associate Professor

Author:
László Csonka
Computer Science MSc

Budapest, 2025

Contents

Chapter 1

Introduction

1.1 Introduction

Introductory programming courses in higher education have consistently posed challenges for students and educators alike. These courses, essential for developing fundamental programming abilities, frequently see elevated failure rates due to the abstract concepts and logical reasoning involved. A multitude of pupils find it challenging to comprehend the essentials of programming logic, problem-solving, and algorithmic reasoning, resulting in pervasive academic underachievement. Early identification of at-risk students in the semester has emerged as a primary concern for educators, since prompt intervention can markedly enhance learning outcomes and retention rates. [1, 2, 3, 4].

With the increased availability of student performance data, educational institutions now have access to massive databases containing student behaviors like as attendance, quiz scores, assignment submissions, and engagement indicators. The use of machine learning (ML) algorithms to examine these datasets has been extremely beneficial in educational data mining. ML approaches such as Random Forest, Support Vector Machine (SVM), and Deep Neural Networks (DNN) allow for the analysis of historical and real-time student data to identify patterns that might help predict future performance. These predictive insights can then be used to adjust teaching tactics, give individualized support, and eventually improve student achievement. [5, 1, 6].

Predicting student performance in introductory programming courses offers a unique opportunity for educators to better understand the specific challenges stu-

dents face in learning to code. By employing ML algorithms, it becomes possible not only to forecast which students are at risk of failure but also to analyze the factors that contribute to their struggles. These insights enable educators to design more effective teaching interventions, such as offering additional tutoring, revising course content, or implementing targeted teaching techniques that address common areas of difficulty [5, 7, 2, 8, 4].

The goal of this thesis is to predict how well students will do in their first programming classes by using and comparing three different machine learning algorithms: Deep Neural Network, Support Vector Machine, and Random Forest. Using data from students provided by the university, this study tries to find the method that makes the most accurate predictions. The best was the Random Forest algorithm, which can be implemented in the university's TMS (Task Management System). This integration allows for real-time, continuous monitoring of student performance and progress throughout the semester. By systematically tracking key academic indicators such as quiz scores, assignment results, and other performance metrics, the TMS can automatically detect patterns and flag students who are at risk of falling behind. This enables early intervention, personalized support, and more targeted assistance from teachers, ultimately improving learning outcomes and retention.

At the end of the academic year, predicted outcomes will be compared with actual results, offering valuable feedback on student progress and highlighting discrepancies where predictions and performance diverge. This comparison will not only serve as an evaluation of the model's effectiveness but will also provide insight into how students' actual progress compares to their predicted trajectories. In cases where predictions suggest strong performance but actual results show underachievement, it may indicate external factors affecting student motivation or engagement, thereby guiding future research and intervention strategies.

The future scope of this work includes refining the prediction model by incorporating real-time data from the early stages of the semester. This would allow for even earlier detection of students who may struggle, enabling timely and more precise interventions. Additionally, integrating live performance data could support adaptive teaching strategies, giving educators immediate insights into the effectiveness of their methods and course materials. Such a feedback loop promotes a more responsive and personalized educational environment.[5, 3, 2, 7]

Chapter 2

Related Work

2.1 Related Work

A crucial area of research, educational data mining offers important insights into the academic performance and learning habits of students. Researchers assess educational data using machine learning techniques to predict student performance, identify learners who are at risk, and understand the factors that determine success or failure. The study emphasizes the trade-offs between computational complexity, interpretability, and simplicity as well as the predictive power of machine learning algorithms and their use in many educational contexts.

2.1.1 Predicting Student Performance in Higher Education

One of the primary goals of educational data mining is to predict students' academic performance. This is especially important as schools try to identify students who are at risk early and help them in specific ways [5, 3]. More and more big educational datasets are being created that show a wide range of student behaviors and academic outcomes [2, 4]. This makes studying this subject even more important.

A number of studies have shown that machine learning models can be used to guess how well students will do in school by looking at things like quiz scores, attendance, participation, and engagement measures [9, 8]. The main purpose of these studies is to predict how students will do in school and to learn more about the factors that affect success or failure in school [6, 7]. Predictions are useful, but many researchers also want to know how specific factors affect success so that teachers can make choices based on facts [3, 2].

A wide range of machine learning algorithms has been applied to educational datasets, from simple classifiers like Naive Bayes to more complex models like Support Vector Machine (SVM), Random Forest, and Deep Neural Networks (DNN) [6, 2, 7]. Each algorithm has its strengths and weaknesses depending on the complexity of the data and the specific educational context [9, 8]. Some studies focus on early warning systems to detect students at risk of failing [5, 3], while others emphasize predicting final grades or overall academic achievement [4, 6].

2.1.2 Overview of Machine Learning Algorithms in Educational Data Mining

Machine learning algorithms have been instrumental in predictive modeling within educational data mining. These algorithms allow researchers to analyze large datasets containing multiple features that interact in complex ways. The following section provides an overview of key algorithms commonly applied in educational data mining, discussing their strengths, weaknesses, and typical use cases.

Random Forest is a robust ensemble learning method that aggregates multiple decision trees to make predictions. It operates by creating multiple trees during training and averaging their outputs for regression tasks or using majority voting for classification tasks. Random Forest excels at handling large datasets with many features and is particularly effective when there are complex interactions between variables. One of its major strengths is its ability to reduce overfitting, making it suitable for predicting student performance based on numerous variables, such as quiz scores, attendance, and engagement metrics [5, 4, 1, 10]. Researchers can also leverage its feature importance measures to identify key predictors of academic success. However, it can be computationally intensive for very large datasets, though advancements in distributed computing have made it more scalable [10].

Decision Trees are intuitive and easy-to-interpret models, often used for both classification and regression tasks. They segment the dataset into smaller groups based on feature thresholds, which makes them highly interpretable and easy to visualize. However, standalone Decision Trees are prone to overfitting, which limits their generalizability. Combining multiple Decision Trees in ensemble methods, such as Random Forest, mitigates these issues and enhances predictive accuracy. Decision Trees are commonly applied in scenarios where simplicity and interpretability are

critical [6, 9].

Decision Tables are rule-based models that organize conditions and actions in tabular form, making them useful for structured decision-making tasks in educational data mining. They are particularly effective in applications where explicit rules need to be derived from data, such as determining student eligibility for certain programs or interventions. Decision Tables are easy to interpret and implement, but they may lack the flexibility of more advanced machine learning models when handling highly complex or non-linear data patterns [6].

J48 is an implementation of the C4.5 decision tree algorithm, used for classification tasks. It offers enhancements over standard Decision Trees by effectively handling missing data and accommodating both continuous and categorical attributes. This adaptability makes it suitable for diverse educational datasets [6, 11, 9]. J48 has been applied to predict student performance and identify key factors influencing success in programming courses. While J48 shares some limitations with traditional Decision Trees, such as overfitting smaller datasets, its ability to manage complex data types enhances its applicability in educational data mining.

Support Vector Machine (SVM) is a powerful supervised learning algorithm, particularly suited for classification tasks in high-dimensional datasets. It works by finding the hyperplane that best separates data points into classes, maximizing the margin between them. This makes SVM especially effective in predicting binary outcomes, such as whether a student will pass or fail a course [10, 6, 5]. Additionally, SVM can handle both linear and non-linear data through the use of kernel functions, making it versatile across different educational applications. However, SVM requires careful hyperparameter tuning, which can be computationally demanding, especially for large datasets.

Deep Neural Networks (DNN) are among the most powerful machine learning models, capable of capturing complex, non-linear relationships in large datasets. They use multiple layers of neurons to learn hierarchical representations of data, making them particularly effective for predicting student outcomes based on intricate patterns, such as engagement, quizzes, and participation records [8]. DNNs have demonstrated strong performance in early detection of at-risk students, often outperforming traditional models like Random Forest and SVM. However, their computational cost is high, making them less feasible for institutions with limited resources. Another challenge with DNNs is their lack of interpretability, which can

be a limitation in educational decision-making [12].

Naive Bayes is a probabilistic classifier that assumes independence between features. Despite this assumption, which may not always hold in educational data, Naive Bayes remains popular due to its simplicity and computational efficiency. It is particularly effective with small datasets where interpretability and ease of use are prioritized over absolute accuracy [5, 13]. In educational data mining, Naive Bayes has been used to predict student performance based on factors such as attendance, quiz scores, and demographic information. However, its performance can be inferior to more complex models like Random Forest and SVM in large-scale applications.

k-Nearest Neighbors (k-NN) is a simple yet effective algorithm that classifies data points based on their proximity to the nearest neighbors. It works well for small to medium-sized datasets but becomes computationally expensive with larger datasets due to the need to calculate distances for every query point. k-NN has been used in educational data mining to predict student performance by analyzing variables like grades and participation [10, 6]. However, as dataset sizes increase, k-NN's efficiency declines, making it less suitable for large-scale applications compared to models like Random Forest or SVM.

Overall, machine learning algorithms provide valuable insights into student performance and learning behaviors. While simpler models like Decision Trees, Naive Bayes, and k-NN offer ease of interpretability and low computational costs, more sophisticated models like Random Forest, SVM, and DNNs provide higher accuracy and robustness, particularly for large and complex educational datasets.

Figure 2.1 shows the overview of the predictive performance[1]of commonly used machine learning algorithms for student performance prediction, based on the [1, 5, 3, 11, 9, 6, 14, 7, 12].

[1]the accuracy of how well the model predicted student performance

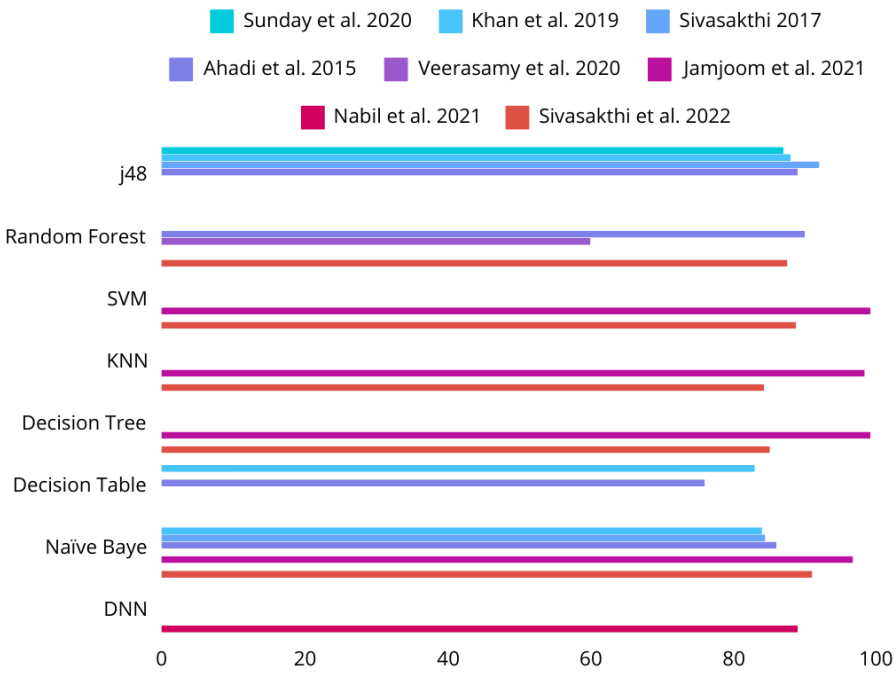

Figure 2.1: Overview of the predictive performance of commonly used machine learning algorithms for student performance prediction

2.1.3 Harnessing the Power of Random Forest, SVM, and DNN in Education

Among the various machine learning algorithms applied in educational data mining, **Random Forest, Support Vector Machine (SVM), and Deep Neural Networks (DNN)** stand out due to their strong predictive performance and versatility. These models have demonstrated success in prior research, excelling in handling complex educational datasets and offering practical applications for student performance prediction.

Each model provides distinct advantages:

- **Random Forest** offers high interpretability and robustness, making it ideal for identifying key factors influencing student success.
- **SVM** excels in structured datasets, providing precise classification for identifying at-risk students.
- **DNN** captures deep, non-linear relationships in student engagement and learning behaviors, making it effective for large-scale data analysis.

By integrating these three algorithms, institutions can build a **comprehensive and adaptive predictive system** that enhances early intervention strategies. This combination ensures **a broad, data-driven perspective on student performance**, enabling educators to make timely and informed decisions to support students more effectively.

2.1.4 Random Forest Methodology

Random Forest is an ensemble learning method that constructs multiple decision trees and aggregates their predictions to improve accuracy, reduce variance, and enhance robustness. The methodology is based on **Bootstrap Aggregation (Bagging)**, which helps minimize overfitting and increases model generalization [15].

The core idea involves training multiple decision trees on randomly sampled subsets of the dataset. Given a dataset D with n samples, a bootstrap sample D_t is drawn from D using sampling with replacement:

$$D_t \subset D, \quad |D_t| = n \tag{2.1}$$

Since some samples may appear multiple times in D_t while others may be omitted, each tree receives a slightly different training set, reducing correlation between models and improving generalization.

For regression tasks, the final prediction is obtained by averaging individual tree outputs:

$$\hat{y} = \frac{1}{T} \sum_{t=1}^{T} h_t(x) \tag{2.2}$$

where T represents the total number of trees, and $h_t(x)$ denotes the prediction from the t-th tree. For classification tasks, majority voting determines the final class prediction:

$$\hat{y} = \arg\max_{c} \sum_{t=1}^{T} \mathbb{1}(h_t(x) = c) \tag{2.3}$$

where c represents class labels, and $\mathbb{1}(\cdot)$ is an indicator function counting votes for each class.

To introduce additional randomness and reduce overfitting, Random Forest selects a random subset of features at each split instead of considering all available features. The number of features m' selected at each split is determined by:

$$m' = \sqrt{m} \quad \text{(for classification)}, \quad m' = \frac{m}{3} \quad \text{(for regression)} \tag{2.4}$$

This prevents dominant features from consistently being chosen, thereby increasing tree diversity.

Splitting Criteria

Each decision tree grows by recursively splitting nodes based on impurity reduction. The choice of impurity measure depends on the task:

- **Classification** Uses **Gini Impurity**, given by:

$$G = \sum_{i=1}^{C} p_i(1 - p_i) \tag{2.5}$$

  where p_i is the proportion of samples belonging to class i.

- **Regression:** Uses **Mean Squared Error (MSE)** to minimize variance in predicted values, defined as:

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^{n} (Y_i - \hat{Y}_i)^2 \tag{2.6}$$

  where Y_i represents the observed values, $\hat{Y}_i$ denotes the predicted values, and n is the number of observations [16].

Feature Importance

Random Forest provides an intrinsic measure of feature importance by analyzing how much each feature contributes to reducing impurity across all trees. The importance score for feature j is calculated as:

$$I_j = \sum_{t=1}^{T} \sum_{s \in \text{splits}(t)} \Delta\text{Impurity}(s) \cdot \mathbb{1}(\text{split feature} = j) \tag{2.7}$$

where $\Delta\text{Impurity}(s)$ represents the reduction in impurity at split s. Higher values indicate more influential features in decision-making [17].

2.1.5 Support Vector Machine Methodology

Support Vector Machines (SVM) are powerful supervised learning models used for classification and regression tasks. SVM is particularly effective in high-dimensional spaces and is widely applied in educational data mining to predict student performance and identify at-risk students [18].

SVM Decision Boundary and Hyperplane

SVM constructs an optimal hyperplane that maximizes the margin between different classes. Given a dataset D with n training samples, where each sample x_i has a corresponding label $y_i \in \{-1, 1\}$, the decision function for SVM is defined as:

$$f(x) = \mathbf{w}^T x + b \tag{2.8}$$

where $\mathbf{w}$ is the weight vector and b is the bias term. The optimal hyperplane maximizes the margin, which is the distance between the closest data points from each class (support vectors):

$$\max_{\mathbf{w},b} \frac{2}{\|\mathbf{w}\|} \tag{2.9}$$

subject to the constraint:

$$y_i(\mathbf{w}^T x_i + b) \geq 1, \quad \forall i = 1, \ldots, n \tag{2.10}$$

This optimization problem is solved using Lagrange multipliers, leading to the dual form:

$$\max_{\alpha} \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j x_i^T x_j \tag{2.11}$$

where α_i are the Lagrange multipliers.

Soft Margin SVM for Non-Separable Data

For real-world educational data, exact class separability is often impossible. Soft-margin SVM introduces slack variables ξ_i to allow for some misclassification:

$$y_i(\mathbf{w}^T x_i + b) \geq 1 - \xi_i, \quad \xi_i \geq 0 \tag{2.12}$$

The modified optimization problem is:

$$\min_{\mathbf{w},b,\xi} \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{n} \xi_i \tag{2.13}$$

where C is the regularization parameter balancing margin maximization and misclassification tolerance.

Kernel Trick for Non-Linear Decision Boundaries

SVM can efficiently handle non-linearly separable data using the kernel trick, which transforms data into a higher-dimensional space without explicitly computing the transformation:

$$K(x_i, x_j) = \phi(x_i)^T \phi(x_j) \tag{2.14}$$

where $K(x_i, x_j)$ is the kernel function, and $\phi(x)$ represents the transformation function. Common kernels include:

- Linear Kernel: $K(x_i, x_j) = x_i^T x_j$
- Polynomial Kernel: $K(x_i, x_j) = (x_i^T x_j + c)^d$
- Radial Basis Function (RBF) Kernel: $K(x_i, x_j) = \exp\left(-\gamma \|x_i - x_j\|^2\right)$

The choice of kernel significantly impacts model performance and must be tuned based on the dataset.

Support Vector Regression (SVR)

SVM can also be extended for **regression tasks**, known as **Support Vector Regression (SVR)**. Instead of finding a classification boundary, SVR fits a function within a margin ϵ while minimizing prediction errors.

To allow some flexibility for points outside the ϵ-margin, SVR introduces two types of slack variables:

- ξ_i: the amount by which the prediction exceeds the ϵ-tube when it is **above** the actual value,

- ξ_i^*: the amount by which the prediction falls below the ϵ-tube when it is **below** the actual value.

The objective function for SVR is:

$$\min_{\mathbf{w},b,\xi,\xi^*} \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{n}(\xi_i + \xi_i^*) \tag{2.15}$$

subject to:

$$\begin{aligned} y_i - (\mathbf{w}^T x_i + b) &\leq \epsilon + \xi_i, \\ (\mathbf{w}^T x_i + b) - y_i &\leq \epsilon + \xi_i^*, \\ \xi_i, \xi_i^* &\geq 0 \end{aligned} \tag{2.16}$$

where ϵ defines the margin of tolerance.

For regression tasks, SVR often optimizes predictions using the Mean Squared Error (MSE) loss function, which has been previously defined in Equation (2.6).

2.1.6 Deep Neural Network Methodology

Deep Neural Networks (DNNs) are powerful machine learning models capable of learning complex, non-linear patterns in data. DNNs are particularly useful in educational data mining, where they can model intricate relationships between student performance, engagement metrics, and learning behaviors [19].

Artificial Neurons and Network Structure

A DNN consists of multiple layers of artificial neurons, each performing weighted computations and applying an activation function to introduce non-linearity. Given an input feature vector x, the output of a neuron in layer l is computed as:

$$z_i^{(l)} = \sum_{j=1}^{n^{(l-1)}} w_{ij}^{(l)} a_j^{(l-1)} + b_i^{(l)} \tag{2.17}$$

$$a_i^{(l)} = \sigma(z_i^{(l)}) \tag{2.18}$$

where:

- $z_i^{(l)}$ is the weighted sum of the previous layer's activations,
- $w_{ij}^{(l)}$ are the weights between neurons,

- $b_i^{(l)}$ is the bias term, and
- $\sigma(\cdot)$ is an activation function.

Common activation functions include:

- **Sigmoid**: $\sigma(z) = \frac{1}{1+e^{-z}}$ (useful for probabilistic outputs)
- **ReLU (Rectified Linear Unit)**: $\sigma(z) = \max(0, z)$ (prevents vanishing gradients)
- **Softmax**: Used in classification tasks for multi-class probability distribution.

Forward Propagation and Loss Function

During forward propagation, the input data passes through the network layer by layer, producing an output. For classification, the output layer uses softmax activation, while regression tasks use a linear output.

The loss function quantifies the difference between predicted output $\hat{y}$ and the actual value y. Common loss functions include:

- Categorical Cross-Entropy for classification:

$$\mathcal{L}_{\text{CE}} = -\sum_{i=1}^{C} y_i \log(\hat{y}_i) \tag{2.19}$$

  where C is the number of classes.

- Mean Squared Error (MSE) for regression, which has been previously defined in Equation (2.6).

Backpropagation and Optimization

DNNs learn by updating weights using backpropagation, which computes the gradient of the loss function with respect to each weight via the chain rule:

$$\frac{\partial \mathcal{L}}{\partial w_{ij}^{(l)}} = \frac{\partial \mathcal{L}}{\partial a_i^{(l)}} \cdot \frac{\partial a_i^{(l)}}{\partial z_i^{(l)}} \cdot \frac{\partial z_i^{(l)}}{\partial w_{ij}^{(l)}} \tag{2.20}$$

The gradients are used to update weights via an optimization algorithm, such as:

- Stochastic Gradient Descent (SGD):

$$w_{ij}^{(l)} \leftarrow w_{ij}^{(l)} - \eta \frac{\partial \mathcal{L}}{\partial w_{ij}^{(l)}} \tag{2.21}$$

  where η is the learning rate

- Adam optimizer, which adapts the learning rate dynamically for each parameter [20].

Regularization Techniques

To prevent overfitting, various regularization methods are used:

- L2 Regularization (Weight Decay):

$$\mathcal{L} = \mathcal{L}_{\text{original}} + \lambda \sum_{l} \sum_{i} \sum_{j} (w_{ij}^{(l)})^2 \tag{2.22}$$

- Dropout, which randomly deactivates neurons during training to improve generalization.

- Batch Normalization, which normalizes activations across mini-batches.

Training Considerations

Training deep networks requires hyperparameter tuning, including:

- Learning rate η

- Number of hidden layers and neurons

- Batch size

- Number of epochs

Techniques like early stopping and learning rate scheduling are used to optimize convergence.

Chapter 3

Discussion

3.1 The dataset

3.1.1 Data Preprocessing and Standardization

The dataset used in this study comes from two courses: "Foundation of Programming" [1] and "Imperative Programming". For Foundation of Programming, data was collected from one fall semester, while for Imperative Programming, the dataset spans from two fall semesters. To ensure consistency across semesters, the data was normalizedand standardized. Unnecessary information was removed, columns were renamed for uniformity, and grading scales were adjusted where necessary.

For "Foundation of Programming", assignments and final exams were graded on a 0–100 scale, whereas in "Imperative Programming", assignments were graded on a 0–30 scale, and final exams on a 0–50 scale. To maintain consistency, the "Foundation of Programming" scores were rescaled converting them to match the "Imperative Programming" scale before merging the datasets.

After standardizing the grading system and merging the semester results, outliers were removed to enhance data quality. These steps are referenced as the main preprocessing activities. Specifically, students were removed if:

- Their total weekly exam scores were ≤ 5 while their final exam score was ≥ 40.
- Their weekly exam total was > 6 but their final exam score was ≤ 20.

[1] Now its called "Programming", the earlier name used for clarity

These cases were likely data inconsistencies or anomalies, suggesting either unexpectedly high final scores despite poor weekly performance or disproportionately low final scores despite reasonable weekly exam results. Removing such outliers improves model accuracy and reliability.

To further standardize the dataset, the following steps were applied:

- Rounded assignment, final exam, and weekly exam scores to two decimal places.
- Filled missing values with zeros.
- Normalized time slots into two columns: one representing the day of the week and the other representing the course start hour.
- Anonymized IDs for privacy protection.

These thorough preprocessing steps ensured the dataset was clean, structured, and ready for further analysis and modeling.

3.1.2 Dataset Description

The datasets used in this study contain student performance data, including weekly exam scores, assignments, and final exams. Two versions of the dataset are available: one with group and time information, and another without these attributes. The number of records differs between the two versions due to the unavailability of group and time information for all semesters.

Dataset	Records
Dataset without Groups and Time	1382
Dataset with Groups and Time	1207

Table 3.1: Records in the dataset after preprocessing and standardization

Dataset with Groups and Time

This dataset includes:

- **Student Identifiers**: *AnonymId* (anonymized unique identifier)
- **Grouping Information**: *Group*, *Day*, *Start_Hour*

- **Weekly Exam Scores**: *First_ Weekly_ Exam*, *Second_ Weekly_ Exam*, ..., *Tenth_ Weekly_ Exam*
- **Assignment Score**: *Assignment*
- **Final Exam Score**: *Final Exam*
- **Bonus Weekly Retake Exam**: *Bonus_ Weekly_ Exam_ Retake*

This version of the dataset allows deeper analysis of contextual factors, like whether the time of day or the assigned group has an effect on student performance. These attributes—especially the day of the week (*Day*) and the start hour (*Hour*)—can be important, because students who attend late afternoon or evening lessons might be more tired or less focused than those who have classes earlier. Time can influence how alert or motivated a student is, and that can affect performance. The group number (*Group*) can also matter, because different groups might have different teachers, teaching assistants, or even smarter students are in a specified group, which can all affect how students learn. By including this kind of scheduling and group information, the dataset can help find patterns that wouldn't be visible in a version without this extra context.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
1	Id	AnonymId	Group	First_Weekly_Exam	Second_Weekly_Exam	Weekly_h	Weekly_	Weekly_	Weekly_	h_Weekly	Weekly_	Weekly_	Tenth_Weekly_Exam	Assignment	Final_Exam	Day	Start_Hour	Bonus_Weekly_Exam_Retake
2	1	bb69eb7a-8cc4-46c6-ad65-26df333a0b9e	1	2	2	2	1	2	2	1	0	2	1	29,5	50	2	13	0
3	2	a04124e2-faa1-4187-bf9c-b0b4d8bbf5fd	13	1	2	1	2	2	2	2	2	0	2	29,5	50	2	11	0
4	4	2aecb070-3d42-4007-9f85-9d34068cfcdc	20	2	1,8	2	1,7	0,6	2	2	2	1	2	30	49	3	8	0
5	5	16d4309d-1194-4479-a2ae-2cee40deb5da	7	2	2	2	2	1	2	2	2	0	2	30	50	2	11	0
6	6	7c238a62-ddf2-40f1-aafd-090f9736ee79	7	2	2	2	1	0	2	2	2	2	2	29,5	49	2	11	0

Figure 3.1: Overview of the dataset with groups and time.

Dataset without Groups and Time

This dataset includes:

- **Student Identifiers**: *AnonymId*
- **Weekly Exam Scores**: *First_ Weekly_ Exam*, *Second_ Weekly_ Exam*, ..., *Tenth_ Weekly_ Exam*
- **Assignment Score**: *Assignment*
- **Final Exam Score**: *Final Exam*
- **Bonus Weekly Retake Exam**: *Bonus_ Weekly_ Exam_ Retake*

This version excludes grouping and scheduling information, focusing only on the academic performance variables.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	Id	AnonymId	First_Weekly_Exam	Second_Weekly_Exam	_Weekly_	_Weekly	Weekly_	Weekly_	h_Weekly	_Weekly_	_Weekly	Tenth_Weekly_Exam	Assignment	Final_Exam	Bonus_Weekly_Exam_Retake
2	1	bb69eb7a-8cc4-46d6-ad65-26df333a0b9e	2	2	2	1	2	2	1	0	2	1	29,5	50	0
3	2	a04124e2-faa1-4187-bf9c-b0b4dbbbf5fd	1	2	1	2	2	2	2	2	0	2	29,5	50	0
4	4	2aecb070-3d42-4007-9f85-9d34068cfcdc	2	1,8	2	1,7	0,6	2	2	2	1	2	30	49	0
5	5	16d4309d-1194-4479-a2ae-2cee40deb5da	2	2	2	2	1	2	2	2	0	2	30	50	0
6	6	7c238a62-ddf2-40f1-aafd-090f9736ee79	2	2	2	1	0	2	2	2	2	2	29,5	49	0

Figure 3.2: Overview of the dataset without groups and time.

3.1.3 Dataset Preparation and Feature Engineering

The dataset was preprocessed to include relevant features such as weekly exam scores, assignments, and final exam scores. Every subset excludes retake exams and AnonymIds. Four main subsets were created to assess the effect of different feature combinations:

- **5 Weekly Exams Without Groups and Times**: Includes student scores from the first five weekly exams without additional metadata.
- **5 Weekly Exams With Groups and Times**: Incorporates grouping and course start time data along with the first five weekly exam scores.
- **6 Weekly Exams Without Groups and Times**: Includes student scores from the first six weekly exams without additional metadata.
- **6 Weekly Exams With Groups and Times**: Adds grouping and course start time information alongside six weekly exam scores.

The dependent variable (target) was the overall exam performance, and the features were the scores from the weekly exams and assignments. In case of SVM and RF the datasets were split into training and testing sets, with 80% used for training and 20% for testing. In case of DNN the datasets were split into training, testing and validation sets in 60%-20%-20% splitting.

3.1.4 Evaluation Metrics

To evaluate the performance of the models, the following metrics were calculated:

- **Mean Squared Error (MSE):** Measures the average squared difference between predictions and actual values. A lower MSE indicates better model performance. (2.6)

- **Root Mean Squared Error (RMSE):** The square root of the MSE, offering an interpretable measure in the same units as the target variable.

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \tag{3.1}$$

- **Mean Absolute Error (MAE):** Measures the average absolute difference between predictions and actual values, providing insight into the average prediction error.

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \tag{3.2}$$

- **R^2 Score:** Explains the proportion of variance in the target variable that can be predicted from the features. A higher R^2 score indicates better model performance.

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2} \tag{3.3}$$

- **Accuracy:** Accuracy was computed by comparing the predicted grades against the actual grades. The grade come from the predicted final exam points. The classification of grades was based on three distinct grading schemes:

  - **Subject-Based Grading:** This classification closely follows the grading scheme used in the actual subject evaluation. It applies a threshold-based approach where students are graded into five categories: *EXCELLENT, GOOD, AVERAGE, SATISFACTORY, and FAIL*. The intervals are determined based on percentage thresholds, ensuring consistency with the original grading system.

  - **Performance-Tiered Grading:** In this method, students are categorized into three broad performance tiers: *HIGH, AVERAGE, and LOW*. The classification is based on percentage thresholds:

    * Students scoring at least 60% of the maximum score are classified as *HIGH*.
    * Those scoring between 30% and 60% fall into the *AVERAGE* category.
    * Students scoring below 30% are classified as *LOW*.

This approach provides a simplified evaluation by grouping students into broader performance categories.

– **Buffer-Zone Grading:** This classification allows for a margin of error in grading by introducing a buffer mechanism. The grading thresholds remain similar to the first classifier, but with a tolerance adjustment:

  * A buffer zone of 6% of the total score is added to each threshold.
  * If a student's score is within this buffer zone, they may receive an upgraded classification. For example, a student who originally falls under *FAIL* but scores within 6% of the *SATISFACTORY* threshold will be upgraded to *SATISFACTORY*.

$$\text{buffer\_zone} = \text{max\_score} \times \text{buffer} \tag{3.4}$$

This adjustment accounts for minor variations in student performance and ensures a more lenient classification approach.

These three classifiers were used to assess accuracy by determining how well the model's predictions aligned with actual grades. The evaluation considered subject-based, performance-tiered and buffer-zone grading approaches to provide a comprehensive performance assessment.

3.2 Evaluation of Random Forest Variants

3.2.1 Simple Random Forest Model

The model is a simple model imported from `scikit-learn`[21] library without any adjustment.

Dataset	**MSE**	**RMSE**	**MAE**	$\mathbf{R^2}$
5 Weekly Exams Without Groups and Times	88.07	9.38	7.29	0.42
6 Weekly Exams Without Groups and Times	77.95	8.83	6.88	0.49
5 Weekly Exams With Groups and Times	73.98	8.60	6.56	0.63
6 Weekly Exams With Groups and Times	75.86	8.71	6.42	0.62

Table 3.2: Performance Metric for Simple Random Forest

Dataset	Accuracy
Performance-Tiered Grading	
5 Weekly Exams Without Groups and Times	81.59%
6 Weekly Exams Without Groups and Times	82.67%
5 Weekly Exams With Groups and Times	88.84%
6 Weekly Exams With Groups and Times	88.02%
Subject-Based Grading	
5 Weekly Exams Without Groups and Times	42.96%
6 Weekly Exams Without Groups and Times	46.57%
5 Weekly Exams With Groups and Times	51.24%
6 Weekly Exams With Groups and Times	53.31%
Buffer-Zone Grading	
5 Weekly Exams Without Groups and Times	46.93%
6 Weekly Exams Without Groups and Times	50.90%
5 Weekly Exams With Groups and Times	53.72%
6 Weekly Exams With Groups and Times	54.13%

Table 3.3: Accuracy Metric for Simple Random Forest

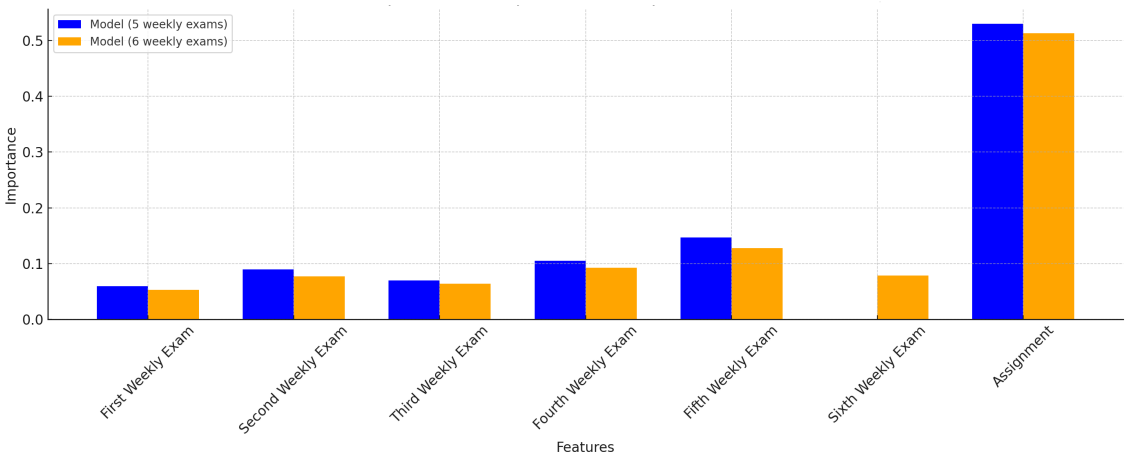

Figure 3.3: Feature importance of Simple Random Forest Model without groups and times

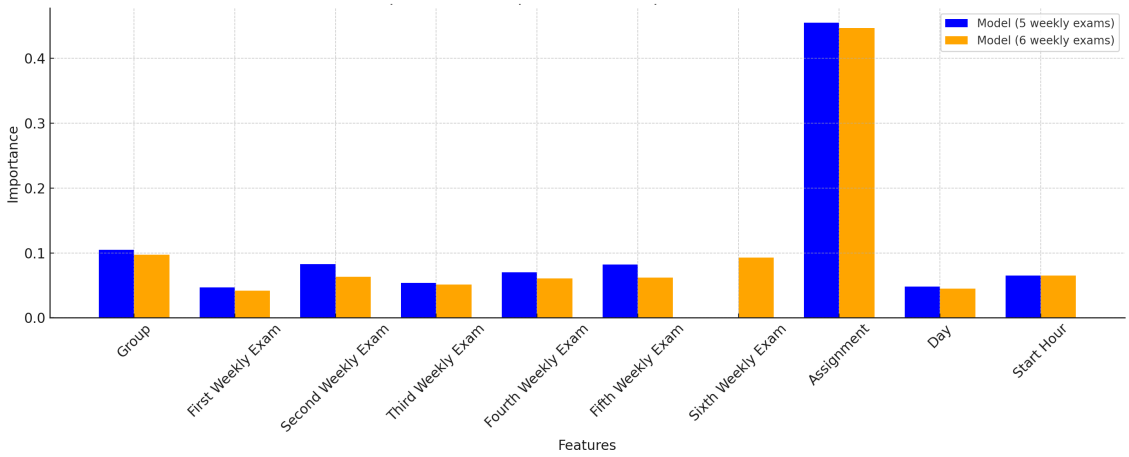

Figure 3.4: Feature importance of Simple Random Forest Model with groups and times

3.2.2 Weighted Random Forest Model

This approach tests whether strategic weighting improves prediction accuracy by better reflecting real-world grading structures, where assignments account for 50% of the total weight. The model adjusts sample weights to prioritize assignment predictions, ensuring that errors in this feature have a greater impact on optimization.

Dataset	**MSE**	**RMSE**	**MAE**	**R^2**
5 Weekly Exams Without Groups and Times	88.03	9.38	7.29	0.42
6 Weekly Exams Without Groups and Times	78.04	8.83	6.87	0.49
5 Weekly Exams With Groups and Times	74.43	8.63	6.60	0.62
6 Weekly Exams With Groups and Times	75.55	8.69	6.43	0.62

Table 3.4: Performance Metric for Weighted Random Forest

Dataset	Accuracy
Performance-Tiered Grading	
5 Weekly Exams Without Groups and Times	81.59%
6 Weekly Exams Without Groups and Times	82.67%
5 Weekly Exams With Groups and Times	88.84%
6 Weekly Exams With Groups and Times	88.02%
Subject-Based Grading	
5 Weekly Exams Without Groups and Times	42.96%
6 Weekly Exams Without Groups and Times	46.93%
5 Weekly Exams With Groups and Times	52.07%
6 Weekly Exams With Groups and Times	52.89%
Buffer-Zone Grading	
5 Weekly Exams Without Groups and Times	46.93%
6 Weekly Exams Without Groups and Times	51.26%
5 Weekly Exams With Groups and Times	54.55%
6 Weekly Exams With Groups and Times	53.72%

Table 3.5: Accuracy Metric for Weighted Random Forest

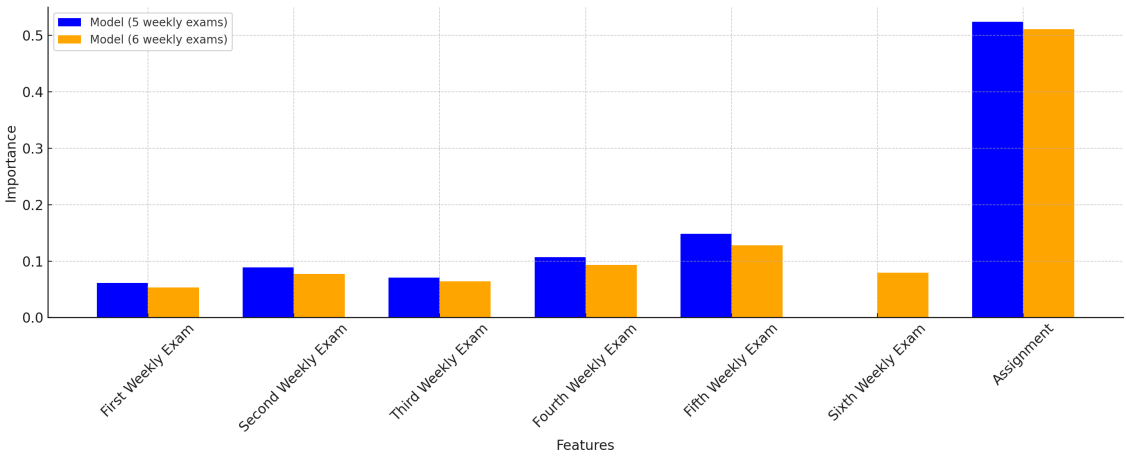

Figure 3.5: Feature importance of Weighted Random Forest Model without groups and times

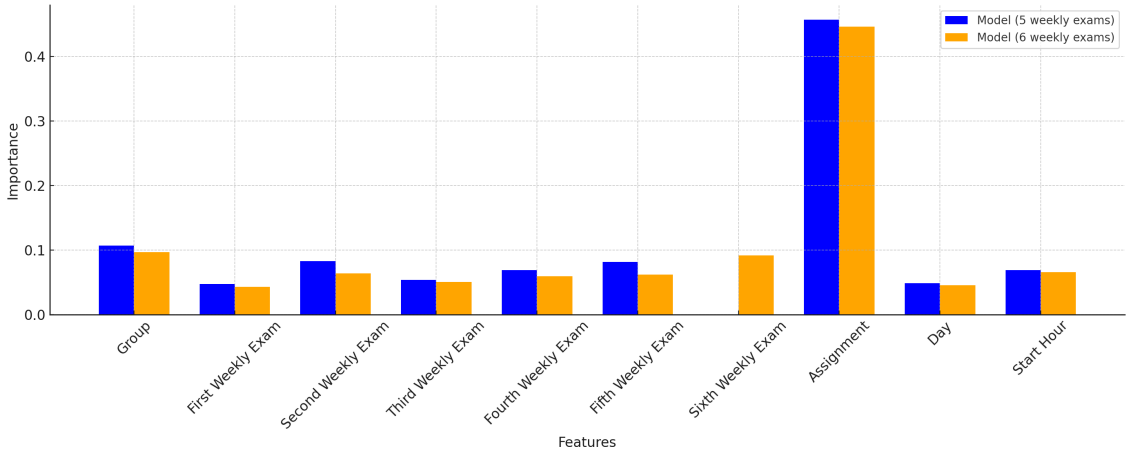

Figure 3.6: Feature importance of Weighted Random Forest Model with groups and times

3.2.3 Optimized Random Forest Model

The Optimized Random Forest Model is trained using `GridSearchCV`[22] to fine-tune key hyperparameters, such as the number of trees, maximum depth, and minimum samples per split. Unlike the default model, this approach systematically searches for the best configuration through cross-validation, ensuring improved generalization and reducing overfitting. By evaluating multiple parameter combinations, `GridSearchCV` identifies the optimal settings for enhanced predictive accuracy.

Dataset	**MSE**	**RMSE**	**MAE**	**R^2**
5 Weekly Exams Without Groups and Times	80.76	8.99	7.02	0.47
6 Weekly Exams Without Groups and Times	75.71	8.70	6.80	0.50
5 Weekly Exams With Groups and Times	72.85	8.54	6.56	0.63
6 Weekly Exams With Groups and Times	70.52	8.40	6.29	0.64

Table 3.6: Performance Metric for Optimized Random Forest

Dataset	Accuracy
Performance-Tiered Grading	
5 Weekly Exams Without Groups and Times	81.95%
6 Weekly Exams Without Groups and Times	82.31%
5 Weekly Exams With Groups and Times	88.02%
6 Weekly Exams With Groups and Times	88.43%
Subject-Based Grading	
5 Weekly Exams Without Groups and Times	44.04%
6 Weekly Exams Without Groups and Times	45.85%
5 Weekly Exams With Groups and Times	52.48%
6 Weekly Exams With Groups and Times	50.83%
Buffer-Zone Grading	
5 Weekly Exams Without Groups and Times	49.46%
6 Weekly Exams Without Groups and Times	50.18%
5 Weekly Exams With Groups and Times	54.96%
6 Weekly Exams With Groups and Times	53.72%

Table 3.7: Accuracy Metric for Optimized Random Forest

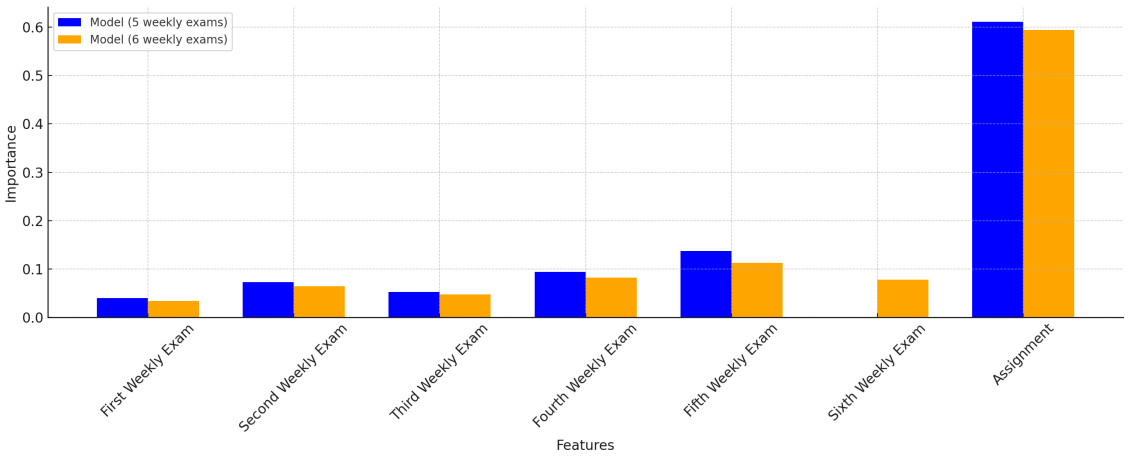

Figure 3.7: Feature importance of Optimized Random Forest Model without groups and times

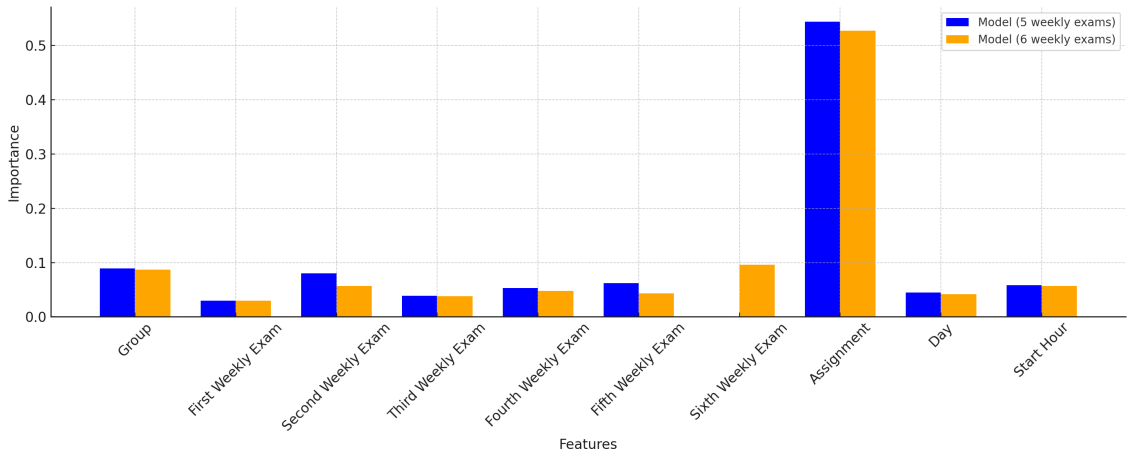

Figure 3.8: Feature importance Optimized Random Forest Model with groups and times

3.2.4 Bagged Random Forest Model

The Bagging Random Forest Model enhances stability by training 10 Random Forest Regressors on different bootstrap samples and averaging their predictions. This ensemble method reduces variance and mitigates overfitting compared to a single Random Forest trained on the full dataset.

For regression, bagging smooths out prediction errors and stabilizes results without significantly altering feature importance. While Random Forest already reduces variance, bagging further refines predictions, leading to slight improvements in MSE and RMSE.

This approach is particularly effective in high-variance datasets like student performance prediction, where individual test scores and assignments may fluctuate unpredictably.

Dataset	**MSE**	**RMSE**	**MAE**	**R^2**
5 Weekly Exams Without Groups and Times	82.08	9.06	7.07	0.46
6 Weekly Exams Without Groups and Times	75.83	8.71	6.87	0.50
5 Weekly Exams With Groups and Times	69.40	8.33	6.52	0.65
6 Weekly Exams With Groups and Times	68.77	8.29	6.28	0.65

Table 3.8: Performance Metric for Bagged Random Forest

Dataset	Accuracy
Performance-Tiered Grading	
5 Weekly Exams Without Groups and Times	82.31%
6 Weekly Exams Without Groups and Times	83.03%
5 Weekly Exams With Groups and Times	89.26%
6 Weekly Exams With Groups and Times	88.84%
Subject-Based Grading	
5 Weekly Exams Without Groups and Times	44.77%
6 Weekly Exams Without Groups and Times	44.77%
5 Weekly Exams With Groups and Times	53.72%
6 Weekly Exams With Groups and Times	53.31%
Buffer-Zone Grading	
5 Weekly Exams Without Groups and Times	45.85%
6 Weekly Exams Without Groups and Times	48.01%
5 Weekly Exams With Groups and Times	56.61%
6 Weekly Exams With Groups and Times	56.20%

Table 3.9: Accuracy Metric for Bagged Random Forest

3.2.5 Stacked Random Forest Model

The Stacked Random Forest Model enhances prediction by integrating multiple learning algorithms rather than relying on a single Random Forest variant. It combines an Optimized Random Forest, AdaBoost, and Gradient Boosting, with a Gradient Boosting Regressor as the meta-model that refines the final predictions.

Stacking improves performance by leveraging each model's strengths: Optimized Random Forest ensures feature robustness, AdaBoost focuses on hard-to-predict cases, and Gradient Boosting iteratively corrects errors. Unlike Bagging, which reduces variance by averaging multiple models of the same type, stacking utilizes diverse models that complement each other, reducing bias while improving generalization.

Dataset	MSE	RMSE	MAE	R^2
5 Weekly Exams Without Groups and Times	84.96	9.22	7.05	0.44
6 Weekly Exams Without Groups and Times	78.51	8.86	7.08	0.48
5 Weekly Exams With Groups and Times	73.31	8.56	6.61	0.63
6 Weekly Exams With Groups and Times	73.01	8.54	6.32	0.63

Table 3.10: Performance Metric for Stacked Random Forest

Dataset	Accuracy
Performance-Tiered Grading	
5 Weekly Exams Without Groups and Times	81.23%
6 Weekly Exams Without Groups and Times	83.03%
5 Weekly Exams With Groups and Times	88.84%
6 Weekly Exams With Groups and Times	88.84%
Subject-Based Grading	
5 Weekly Exams Without Groups and Times	45.13%
6 Weekly Exams Without Groups and Times	39.71%
5 Weekly Exams With Groups and Times	54.55%
6 Weekly Exams With Groups and Times	50.83%
Buffer-Zone Grading	
5 Weekly Exams Without Groups and Times	51.26%
6 Weekly Exams Without Groups and Times	45.85%
5 Weekly Exams With Groups and Times	60.33%
6 Weekly Exams With Groups and Times	57.44%

Table 3.11: Accuracy Metric for Stacked Random Forest

3.3 Final Comparison and Selection of the Best Random Forest Variant

The evaluation of different Random Forest variants for student performance prediction reveals critical insights into the trade-offs between accuracy, generalization, and interpretability. The experiments explored five variations: Simple Random Forest, Weighted Random Forest, Optimized Random Forest, Bagged Random Forest, and Stacked Random Forest.

3.3.1 Analysis of Feature Importance

In all Random Forest models where feature importance could be extracted (Simple, Weighted, and Optimized), the assignment score consistently proved to be the most dominant predictor of final exam performance—contributing over 50% of the total importance without contextual data, and around 45% with group and time information included. Among the weekly exams, the fifth and sixth tests had the highest relevance, indicating that student progress in the latter half of the semester is more predictive of final outcomes.

Contextual features such as group number and course start time showed modest but non-trivial importance (approximately 10% and 6%, respectively), suggesting that scheduling and cohort dynamics may influence performance. Feature importance could not be extracted from the Bagged and Stacked models due to their ensemble structure. Figures 3.3, 3.4, 3.5, 3.6, 3.7 and 3.8 illustrate the feature importances across different model variations.

3.3.2 Comparison of Model Variants and Dataset Impact

One of the critical aspects of the study was evaluating how the dataset configuration, particularly the number of weekly exams and the inclusion of grouping and timing data, influenced model performance. While the addition of a sixth exam only slightly improved predictive accuracy in more advanced models (e.g., Optimized, Bagged, and Stacked Random Forests), the inclusion of group and timing information consistently enhanced model generalization across all variants. This structured data likely introduced context-aware patterns in student performance, making models more robust and reliable. Therefore, group and timing data had a more profound impact than the number of weekly exams, especially for high-variance learners. This is particularly evident in Bagged and Stacked Random Forest models, as shown in Tables 3.2, 3.4, 3.6, 3.8, 3.10 show the evaluation metrics for the different Random Forest models.

3.3.3 Best Grading Approach for Real-Time Application

Among the three grading methodologies, Buffer-Zone Grading proves to be the most accurate approach when applied to test-grade predictions. Since the models predict scores before the final grades are assigned, Buffer-Zone Grading introduces a margin of flexibility, helping to identify students at risk by allowing for small adjustments in classification. This ensures that students who are close to a threshold can be flagged for potential intervention earlier in the academic year. Tables 3.3, 3.5, 3.7, 3.9, 3.11 show the accuracy metrics for the different Random Forest models.

3.3.4 Conclusion

In summary, the **Bagged Random Forest Model** trained on the **6 Weekly Exams dataset with groups and times**, is the optimal choice due to its high

predictive accuracy, robust generalization, and reduced variance. When paired with **Buffer-Zone Grading**, it ensures both fairness and reliability, making it the best solution for real-time student performance prediction.

3.4 Evaluation of Support Vector Machine Variants

3.4.1 Simple Support Vector Machine Model

The model is a simple model imported from `scikit-learn` library without any adjustement.

Dataset	**MSE**	**RMSE**	**MAE**	$\mathbf{R^2}$
5 Weekly Exams Without Groups and Times	93.29	9.66	7.33	0.39
6 Weekly Exams Without Groups and Times	89.60	9.47	7.24	0.41
5 Weekly Exams With Groups and Times	162.75	12.76	8.58	0.18
6 Weekly Exams With Groups and Times	161.22	12.70	8.46	0.18

Table 3.12: Performance Metric for Simple SVM

.

Dataset	**Accuracy**
Performance-Tiered Grading	
5 Weekly Exams Without Groups and Times	81.59%
6 Weekly Exams Without Groups and Times	81.95%
5 Weekly Exams With Groups and Times	81.82%
6 Weekly Exams With Groups and Times	81.82%
Subject-Based Grading	
5 Weekly Exams Without Groups and Times	49.10%
6 Weekly Exams Without Groups and Times	49.10%
5 Weekly Exams With Groups and Times	47.93%
6 Weekly Exams With Groups and Times	49.59%
Buffer-Zone Grading	
5 Weekly Exams Without Groups and Times	55.23%
6 Weekly Exams Without Groups and Times	54.51%
5 Weekly Exams With Groups and Times	54.13%
6 Weekly Exams With Groups and Times	54.96%

Table 3.13: Accuracy Metric for Simple SVM

3.4.2 Weighted Support Vector Machine Model

This approach tests whether strategic feature weighting enhances prediction accuracy in Support Vector Regression (SVR) by assigning 50% weight to assignments and distributing the rest among other features. Unlike Random Forest, SVM optimizes based on support vectors, and incorporating sample weights influences the importance of different training points.

Dataset	**MSE**	**RMSE**	**MAE**	**R^2**
5 Weekly Exams Without Groups and Times	108.49	10.42	8.27	0.29
6 Weekly Exams Without Groups and Times	447.18	21.15	16.14	-1.94
5 Weekly Exams With Groups and Times	19671.66	140.26	70.77	-98.66
6 Weekly Exams With Groups and Times	135473.64	368.07	198.06	-685.31

Table 3.14: Performance Metric for Weighted SVM

Dataset	**Accuracy**
Performance-Tiered Grading	
5 Weekly Exams Without Groups and Times	81.59%
6 Weekly Exams Without Groups and Times	57.40%
5 Weekly Exams With Groups and Times	56.20%
6 Weekly Exams With Groups and Times	49.17%
Subject-Based Grading	
5 Weekly Exams Without Groups and Times	33.21%
6 Weekly Exams Without Groups and Times	27.44%
5 Weekly Exams With Groups and Times	41.74%
6 Weekly Exams With Groups and Times	33.06%
Buffer-Zone Grading	
5 Weekly Exams Without Groups and Times	40.43%
6 Weekly Exams Without Groups and Times	27.08%
5 Weekly Exams With Groups and Times	42.15%
6 Weekly Exams With Groups and Times	35.95%

Table 3.15: Accuracy Metric for Weighted SVM

3.4.3 Optimized Support Vector Machine Model

The Optimized SVM model leverages GridSearchCV to fine-tune hyperparameters, including kernel type, regularization parameter (C), and gamma scaling, through 5-fold cross-validation. By systematically evaluating parameter combinations, this approach enhances predictive accuracy while reducing overfitting.

Dataset	MSE	RMSE	MAE	R^2
5 Weekly Exams Without Groups and Times	94.54	9.72	7.18	0.38
6 Weekly Exams Without Groups and Times	91.57	9.57	7.14	0.40
5 Weekly Exams With Groups and Times	85.61	9.25	6.98	0.57
6 Weekly Exams With Groups and Times	79.90	8.94	6.53	0.60

Table 3.16: Performance Metric for Optimized SVM

Dataset	Accuracy
Performance-Tiered Grading	
5 Weekly Exams Without Groups and Times	80.51%
6 Weekly Exams Without Groups and Times	80.87%
5 Weekly Exams With Groups and Times	86.78%
6 Weekly Exams With Groups and Times	88.02%
Subject-Based Grading	
5 Weekly Exams Without Groups and Times	52.35%
6 Weekly Exams Without Groups and Times	49.10%
5 Weekly Exams With Groups and Times	55.37%
6 Weekly Exams With Groups and Times	57.85%
Buffer-Zone Grading	
5 Weekly Exams Without Groups and Times	56.68%
6 Weekly Exams Without Groups and Times	53.07%
5 Weekly Exams With Groups and Times	61.57%
6 Weekly Exams With Groups and Times	62.81%

Table 3.17: Accuracy Metric for Optimized SVM

3.4.4 Bagged Support Vector Machine Model

The Bagging SVM model enhances stability by training 50 Support Vector Regressors (SVR) on different bootstrap samples and averaging their predictions. This ensemble approach reduces variance and mitigates overfitting, improving generalization compared to a single SVM. Unlike Random Forest, where bagging naturally complements decision trees, SVM benefits from bagging by reducing sensitivity to individual support vectors, leading to more stable predictions.

Dataset	MSE	RMSE	MAE	R^2
5 Weekly Exams Without Groups and Times	108.57	10.42	7.82	0.29
6 Weekly Exams Without Groups and Times	107.59	10.37	7.78	0.29
5 Weekly Exams With Groups and Times	117.31	10.83	7.87	0.41
6 Weekly Exams With Groups and Times	116.95	10.81	7.85	0.41

Table 3.18: Performance Metric for Bagged SVM

Dataset	Accuracy
Performance-Tiered Grading	
5 Weekly Exams Without Groups and Times	80.87%
6 Weekly Exams Without Groups and Times	80.87%
5 Weekly Exams With Groups and Times	81.40%
6 Weekly Exams With Groups and Times	81.40%
Subject-Based Grading	
5 Weekly Exams Without Groups and Times	51.99%
6 Weekly Exams Without Groups and Times	52.35%
5 Weekly Exams With Groups and Times	53.31%
6 Weekly Exams With Groups and Times	53.31%
Buffer-Zone Grading	
5 Weekly Exams Without Groups and Times	57.40%
6 Weekly Exams Without Groups and Times	57.76%
5 Weekly Exams With Groups and Times	56.20%
6 Weekly Exams With Groups and Times	56.20%

Table 3.19: Accuracy Metric for Bagged SVM

3.4.5 Stacked Support Vector Machine Model

The Stacked SVM Model enhances prediction by integrating multiple learning algorithms rather than relying on a single model. It combines an optimized Support Vector Regressor (SVR), Random Forest, and Gradient Boosting, with a Ridge regression model as the final estimator to refine predictions.

Stacking improves performance by leveraging the strengths of each base model: SVR captures complex relationships, Random Forest ensures feature robustness, and Gradient Boosting iteratively corrects residual errors. Unlike bagging, which reduces variance by averaging multiple models of the same type, stacking utilizes diverse models that complement each other, improving generalization and reducing bias. By incorporating passthrough learning, the meta-model gains direct access to the original features, further refining its predictive accuracy.

Dataset	MSE	RMSE	MAE	R^2
5 Weekly Exams Without Groups and Times	76.75	8.76	6.96	0.50
6 Weekly Exams Without Groups and Times	75.05	8.66	6.84	0.51
5 Weekly Exams With Groups and Times	73.50	8.57	6.85	0.63
6 Weekly Exams With Groups and Times	69.38	8.33	6.36	0.65

Table 3.20: Performance Metric for Stacked SVM

Dataset	Accuracy
Performance-Tiered Grading	
5 Weekly Exams Without Groups and Times	81.95%
6 Weekly Exams Without Groups and Times	81.95%
5 Weekly Exams With Groups and Times	88.43%
6 Weekly Exams With Groups and Times	88.84%
Subject-Based Grading	
5 Weekly Exams Without Groups and Times	46.57%
6 Weekly Exams Without Groups and Times	46.21%
5 Weekly Exams With Groups and Times	54.13%
6 Weekly Exams With Groups and Times	52.89%
Buffer-Zone Grading	
5 Weekly Exams Without Groups and Times	49.46%
6 Weekly Exams Without Groups and Times	51.26%
5 Weekly Exams With Groups and Times	57.02%
6 Weekly Exams With Groups and Times	55.79%

Table 3.21: Accuracy Metric for Stacked SVM

3.5 Final Comparison and Selection of the Best Support Vector Machine Variant

The evaluation of different Support Vector Machine (SVM) variants for student performance prediction provides key insights into the model's ability to generalize, handle feature weighting, and leverage ensemble learning. This study explored multiple variations, including Simple SVM, Weighted SVM, Optimized SVM, Bagged SVM, and Stacked SVM models.

Comparison of Model Variants and Dataset Impact

An essential aspect of this study was assessing how the dataset structure influenced SVM performance. Results indicate that the inclusion of 6 Weekly Exams

tended to improve predictive accuracy, particularly in more advanced models such as Optimized and Stacked SVMs. Furthermore, the addition of group and timing information significantly enhanced generalization, most notably in the Optimized and Stacked models, where contextual features enabled better pattern recognition and reduced error. Tables 3.12, 3.14, 3.16, 3.18, 3.20 show the evaluation metrics for the different Support Vector Machine models.

Best Grading Approach for Real-Time Application

As discussed in the Random Forest section, Buffer-Zone Grading has emerged as the most practical grading strategy for real-time applications, thanks to its tolerance for near-boundary prediction errors. This finding remains valid in the context of Support Vector Machine models as well. Given that SVMs generate continuous output values, the buffer-based classification system naturally complements their behavior, minimizing misclassification in ambiguous cases and better supporting early risk detection. Accuracy results across all SVM variants using this grading scheme are summarized in Tables 3.13, 3.15, 3.17, 3.19, 3.21 show the accuracy metrics for the different Support Vector Machine models.

Conclusion

Despite the relatively stable performance of the Simple and Bagged models, their predictive power remained limited due to the lack of hyperparameter optimization or diversity in structure. The Weighted variant, while conceptually aligned with real-world grading, failed to generalize and significantly underperformed, likely due to poor sample weight distribution. For real-time student performance prediction using **Support Vector Machines**, the **Stacked SVM Model** trained on the **6 Weekly Exams dataset with groups and time**, combined with **Buffer-Zone Grading**, proves to be the most effective configuration. While the **Optimized SVM** achieved the highest accuracy under certain grading schemes, the Stacked model consistently demonstrated better generalization and robustness across datasets. This balance between predictive accuracy, stability, and adaptability makes it a more reliable choice for deployment in educational settings where early intervention and fairness are essential.

3.6 Evaluation of Deep Neural Network Variants

3.6.1 Tabnet Model

TabNetRegressor [23] is a deep learning model designed for regression tasks on tabular data. It utilizes a sequential attention mechanism to select relevant features dynamically, enabling interpretability and efficient learning directly from raw tabular datasets without extensive preprocessing.

Key Parameters

- `n_d`: Dimension of the decision prediction layer (48).
- `n_a`: Dimension of the attention embedding for each mask (48).
- `n_steps`: Number of sequential decision steps (8).
- `gamma`: Coefficient for feature reuse in the masks (1.8).
- `lambda_sparse`: Coefficient for the sparsity (1e-5) regularization to encourage minimal feature selection.
- `momentum`: Momentum parameter for batch normalization (0.5).
- `mask_type`: Type of mask function to use, either "sparsemax" or "entmax".

On the training it uses `Adam Optimizer` as a default optimizer.

Dataset	**MSE**	**RMSE**	**MAE**	$\mathbf{R^2}$
5 Weekly Exams Without Groups and Times	78.71	8.87	6.99	0.48
6 Weekly Exams Without Groups and Times	78.44	8.86	7.44	0.48
5 Weekly Exams With Groups and Times	82.08	9.06	7.68	0.58
6 Weekly Exams With Groups and Times	84.78	9.21	7.14	0.57

Table 3.22: Performance Metric for Tabnet Model

Dataset	Accuracy
Performance-Tiered Grading	
5 Weekly Exams Without Groups and Times	83.75%
6 Weekly Exams Without Groups and Times	83.03%
5 Weekly Exams With Groups and Times	87.60%
6 Weekly Exams With Groups and Times	86.36%
Subject-Based Grading	
5 Weekly Exams Without Groups and Times	46.21%
6 Weekly Exams Without Groups and Times	35.74%
5 Weekly Exams With Groups and Times	42.56%
6 Weekly Exams With Groups and Times	50.00%
Buffer-Zone Grading	
5 Weekly Exams Without Groups and Times	51.26%
6 Weekly Exams Without Groups and Times	41.52%
5 Weekly Exams With Groups and Times	44.63%
6 Weekly Exams With Groups and Times	52.07%

Table 3.23: Accuracy Metric for Tabnet Model

3.6.2 Optimized Tabnet Model

The Optuna framework [24] is used in this study to optimize the hyperparameters of the TabNet model. Optuna is a powerful, automatic hyperparameter tuning library that uses a Tree-structured Parzen Estimator (TPE) to efficiently explore the search space and find optimal configurations. In this implementation, Optuna minimizes the Mean Squared Error (MSE) by tuning parameters such as `n_d`, `n_a`, `n_steps`, `learning_rate`, and `momentum`.

Dataset	MSE	RMSE	MAE	R^2
5 Weekly Exams Without Groups and Times	76.63	8.75	7.09	0.50
6 Weekly Exams Without Groups and Times	83.12	9.12	7.42	0.45
5 Weekly Exams With Groups and Times	72.85	8.54	7.25	0.63
6 Weekly Exams With Groups and Times	79.10	8.89	6.99	0.60

Table 3.24: Performance Metric for Optimized Tabnet Model

Dataset	Accuracy
Performance-Tiered Grading	
5 Weekly Exams Without Groups and Times	81.59%
6 Weekly Exams Without Groups and Times	83.23%
5 Weekly Exams With Groups and Times	87.60%
6 Weekly Exams With Groups and Times	86.78%
Subject-Based Grading	
5 Weekly Exams Without Groups and Times	45.49%
6 Weekly Exams Without Groups and Times	41.52%
5 Weekly Exams With Groups and Times	47.52%
6 Weekly Exams With Groups and Times	53.31%
Buffer-Zone Grading	
5 Weekly Exams Without Groups and Times	49.10%
6 Weekly Exams Without Groups and Times	43.32%
5 Weekly Exams With Groups and Times	49.12%
6 Weekly Exams With Groups and Times	57.85%

Table 3.25: Accuracy Metric for Optimized Tabnet Model

3.6.3 FastAI Model

The `fastai`[25] library provides a powerful and flexible deep learning framework for tabular data processing, integrating data loading, preprocessing, and model training in a streamlined manner. In this study, `TabularDataLoaders` and `TabularLearner` are used to build an efficient learning pipeline.

The `TabularDataLoaders` class is used to preprocess the dataset and create optimized data batches for training. The data is first structured by combining input features (`X`) and target values (`Y`) into a single DataFrame. The `from_df()` method is then applied to handle feature normalization and automatic batch generation.

Once the data is prepared, the `TabularLearner` is used to train a deep learning model on the tabular dataset. The `TabularLearner` is a neural network-based model optimized for structured data, utilizing embedding layers for categorical variables, batch normalization, and dropout for regularization. Its key features include:

- **Loss Function:** MSE for regression tasks and Cross-Entropy Loss for classification.
- **Optimization:** fastai's adaptive `1cycle_policy` with the Adam optimizer enables efficient convergence.

- **Training Pipeline:** The model leverages fastai's built-in progressive learning rate finder and early stopping mechanisms.

The combination of `TabularDataLoaders` and `TabularLearner` provides a robust approach to handling tabular data efficiently while maintaining flexibility for hyperparameter tuning and optimization. (`TabularLearner(dls_five, y_names="target", layers=[200,100], metrics=[mae, rmse])`)

Dataset	MSE	RMSE	MAE	R^2
5 Weekly Exams Without Groups and Times	1647.23	40.59	39.28	-9.82
6 Weekly Exams Without Groups and Times	1643.78	40.54	39.21	-9.80
5 Weekly Exams With Groups and Times	1704.82	41.29	39.60	-7.64
6 Weekly Exams With Groups and Times	1714.30	41.40	39.60	-7.68

Table 3.26: Performance Metric for FastAI Model

Dataset	Accuracy
Performance-Tiered Grading	
5 Weekly Exams Without Groups and Times	4.69%
6 Weekly Exams Without Groups and Times	4.69%
5 Weekly Exams With Groups and Times	7.85%
6 Weekly Exams With Groups and Times	7.85%
Subject-Based Grading	
5 Weekly Exams Without Groups and Times	4.69%
6 Weekly Exams Without Groups and Times	4.69%
5 Weekly Exams With Groups and Times	7.85%
6 Weekly Exams With Groups and Times	7.85%
Buffer-Zone Grading	
5 Weekly Exams Without Groups and Times	4.69%
6 Weekly Exams Without Groups and Times	4.69%
5 Weekly Exams With Groups and Times	7.85%
6 Weekly Exams With Groups and Times	7.85%

Table 3.27: Accuracy Metric for FastAI Model

3.6.4 Optimized FastAI Model

Optimizing fastai's TabularLearner with Optuna involves fine-tuning key hyperparameters, such as the number of layers, neurons per layer, learning rate, dropout rate, weight decay, and batch size, to improve model performance.

Dataset	MSE	RMSE	MAE	R^2
5 Weekly Exams Without Groups and Times	81.55	9.03	7.18	0.46
6 Weekly Exams Without Groups and Times	77.73	8.82	7.03	0.49
5 Weekly Exams With Groups and Times	79.55	8.92	6.83	0.60
6 Weekly Exams With Groups and Times	77.63	8.81	6.97	0.61

Table 3.28: Performance Metric for Optimized FastAI Model

Dataset	Accuracy
Performance-Tiered Grading	
5 Weekly Exams Without Groups and Times	82.67%
6 Weekly Exams Without Groups and Times	82.67%
5 Weekly Exams With Groups and Times	87.19%
6 Weekly Exams With Groups and Times	86.36%
Subject-Based Grading	
5 Weekly Exams Without Groups and Times	44.04%
6 Weekly Exams Without Groups and Times	48.74%
5 Weekly Exams With Groups and Times	54.55%
6 Weekly Exams With Groups and Times	53.31%
Buffer-Zone Grading	
5 Weekly Exams Without Groups and Times	48.74%
6 Weekly Exams Without Groups and Times	51.62%
5 Weekly Exams With Groups and Times	58.68%
6 Weekly Exams With Groups and Times	58.68%

Table 3.29: Accuracy Metric for Optimized FastAI Model

3.7 Final Comparison and Selection of the Best Deep Neural Network Variant

The evaluation of Deep Neural Network (DNN) variants for student performance prediction highlights key trade-offs between accuracy, generalization, and interpretability. This study investigated multiple DNN models, including TabNet, Optimized TabNet, FastAI Model, and Optimized FastAI Model.

3.7.1 Comparison of Model Variants and Dataset Impact

A key focus was evaluating how the dataset structure impacted model performance. The results indicate that 6 Weekly Exams consistently improved predictive

accuracy compared to 5 Weekly Exams, suggesting that additional data provided more learning opportunities for the models. The inclusion of group and time information also led to increased accuracy, particularly in Optimized TabNet and Optimized FastAI models, which benefit from additional contextual features. Tables 3.22, 3.24, 3.26, 3.28 show the evaluation metrics for the different Deep Neural Network models.

3.7.2 Best Grading Approach for Real-Time Application

As observed in the Random Forest and Support Vector Machine sections, **Buffer-Zone Grading** consistently proved to be the most practical strategy for real-time applications. This finding remains valid for Deep Neural Network models as well. Since these models produce continuous output values, the buffer-based classification system naturally complements their behavior by reducing misclassification near decision boundaries and allowing for early identification of students at risk. Tables 3.23–3.29 present the accuracy results achieved by the DNN variants under different grading schemes.Tables 3.23, 3.25, 3.27, 3.29, show the accuracy metrics for the different Deep Neural Network models.

3.7.3 Conclusion

For real-time student performance prediction using Deep Neural Networks, the **Optimized FastAI Model** trained on the **6 Weekly Exams dataset with groups and times**, combined with **Buffer-Zone Grading**, is the optimal choice. While the **Optimized TabNet Model** achieved competitive results—especially in terms of accuracy—the Optimized FastAI model consistently demonstrated superior generalization and stability across all grading schemes. This balance between predictive performance, fairness, and adaptability makes it the most reliable choice for deployment in educational applications.

3.8 Further Evaluation of the Top Performing Models on the Combined Dataset

After evaluating all model families across various dataset configurations, the best-performing variant from each family was identified. However, those experiments were conducted on separate records for each subject, where the same student could appear

multiple times—once per course—without the model being aware of this duplication. As a result, models could not learn patterns that span across subjects for the same individual.

To address this limitation, a new experimental phase was introduced using a combined dataset in which each row corresponds to a single student and includes their performance data from both the "Foundation of Programming" and "Imperative Programming" courses. This consolidated representation allows the models to associate multiple course outcomes with the same student, supporting the discovery of more personalized and generalizable patterns.

In the following section, we revisit the top-performing models—one from each model family—and evaluate how well they generalize when student data is merged across subjects. The goal is to assess whether their earlier success translates to this more complex, holistic test environment.

3.8.1 Combined Dataset

Building upon these insights, a new dataset was constructed by merging student data from both the "Foundation of Programming" and "Imperative Programming" courses into a single, unified structure. In this representation, each row corresponds to a single student, capturing their performance in both courses simultaneously. This allows the models to associate multiple course outcomes with the same individual, providing a more personalized and student-centered perspective.

The preprocessing and standardization steps remained the same as described earlier, the only thing changed that the student identifier was used to match the corresponding rows. The resulting dataset contains 309 records, due to the variation in the amount of data held per subject. Based on the findings from previous experiments, the configuration using 6 Weekly Exams with group and time consistently yielded the best results across model families. This indicated that both the number of assessments and the inclusion of contextual metadata (group and time) significantly contributed to the improvement of the models, that is why we only experiment on the same subset of the dataset.

This combined dataset provides a more comprehensive testing setup, evaluating whether the top-performing models can generalize effectively when student results from multiple courses are integrated into a unified format.

3.8.2 Evaluation Results

Bagged Random Forest Results

Subject	MSE	RMSE	MAE	R^2
Programming Final Exam	29.23	5.41	3.72	0.50
Imperative Final Exam	65.71	8.11	6.63	0.36

Table 3.30: Performance Metrics for Bagged Random Forest

Subject	Grading Scheme	Accuracy
Programming Final Exam	Subject-Based	83.87%
Imperative Final Exam	Subject-Based	32.26%
Programming Final Exam	Buffer-Zone	85.48%
Imperative Final Exam	Buffer-Zone	40.32%
Programming Final Exam	Performance-Tiered	96.77%
Imperative Final Exam	Performance-Tiered	83.87%

Table 3.31: Accuracy Metrics for Bagged Random Forest

Stacked Support Vector Machine Results

Subject	MSE	RMSE	MAE	R^2
Programming Final Exam	28.81	5.37	3.67	0.51
Imperative Final Exam	75.55	8.69	7.15	0.26

Table 3.32: Performance Metrics for Stacked SVM

Subject	Grading Scheme	Accuracy
Programming Final Exam	Subject-Based	79.03%
Imperative Final Exam	Subject-Based	33.87%
Programming Final Exam	Buffer-Zone	80.65%
Imperative Final Exam	Buffer-Zone	38.71%
Programming Final Exam	Performance-Tiered	96.77%
Imperative Final Exam	Performance-Tiered	80.65%

Table 3.33: Accuracy Metrics for Stacked SVM

Optimized FastAI Results

Subject	MSE	RMSE	MAE	R^2
Programming Final Exam	35.79	5.98	4.63	0.39
Imperative Final Exam	77.75	8.82	7.17	0.24

Table 3.34: Performance Metrics for Optimized FastAI

Subject	Grading Scheme	Accuracy
Programming Final Exam	Subject-Based	80.65%
Imperative Final Exam	Subject-Based	43.55%
Programming Final Exam	Buffer-Zone	83.87%
Imperative Final Exam	Buffer-Zone	35.48%
Programming Final Exam	Performance-Tiered	96.77%
Imperative Final Exam	Performance-Tiered	75.81%

Table 3.35: Accuracy Metrics for Optimized FastAI

3.8.3 Comparison of Best Variants trained on combined dataset

While the selected models still achieve solid results on the combined dataset, a closer look at the regression and classification metrics reveals a clear disparity between the two subjects. For example, in the Bagged Random Forest model, the Programming Final Exam is predicted with relatively low error (MSE: 29.23, R^2: 0.50), whereas for the Imperative Programming Final Exam, the error is significantly higher (MSE: 65.70, R^2: 0.36). This trend is consistent across all three models. Classification accuracy under Buffer-Zone Grading further highlights this gap: while Programming consistently exceeds 80%, Imperative Programming remains in the 35–40% range. This performance gap is consistently observable across all three models, and is summarized in the regression and classification results shown in Tables 3.31, 3.32, 3.33, 3.34, and 3.35.

Several factors may contribute to this discrepancy. First and foremost, the combined dataset comprises only 309 complete records, a sharp decrease compared to the original datasets with over 1200 samples. This reduction increases the likelihood of overfitting, particularly for subjects like Programming that exhibit more predictable patterns. Smaller datasets also tend to exaggerate model confidence, producing artificially high accuracy scores that may not generalize well.

Another key aspect lies in the way data is structured. In this configuration, each row encapsulates a student's performance across two distinct courses, as opposed to the previous setup where each course was treated independently. This structural change introduces a new modeling challenge—capturing joint learning dynamics—which may not be equally strong across subjects. The models may find consistent signals in Programming due to its more systematic and gradual skill progression, whereas Imperative Programming likely involves more abstract reasoning and non-linear development, making its performance harder to forecast.

3.8.4 Correlation of the combined dataset features

To better understand the differences in model performance across the two subjects, we analyzed the correlation matrix of the combined dataset. This reveals how strongly individual features—such as weekly exams and assignments—are related to final exam performance, and helps explain why Programming appears more predictable than Imperative Programming.

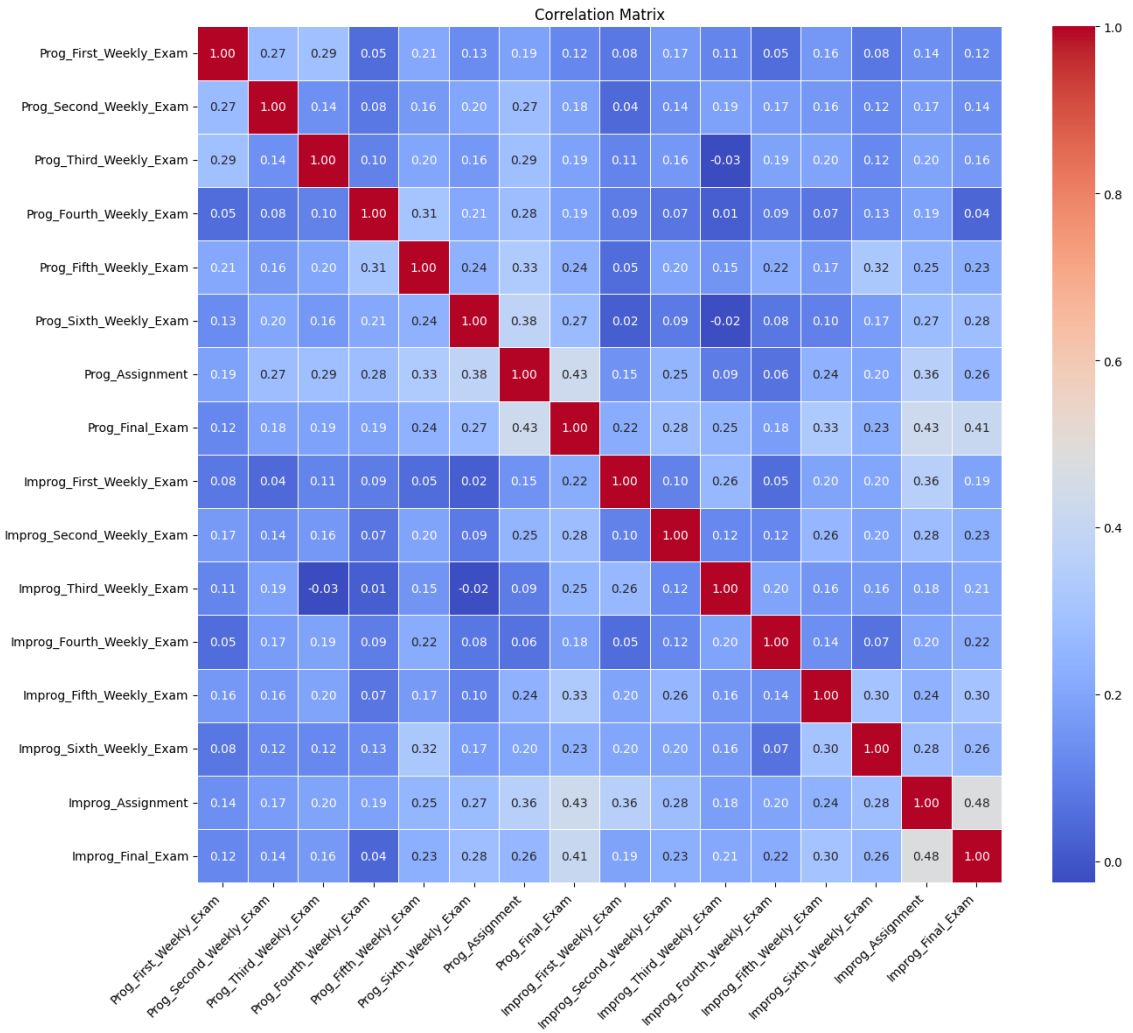

Figure 3.9: Correlation matrix of combined dataset features

Strong Correlations Between Assignments and Final Exam Scores

- **Programming:** The correlation between `Prog_Assignment` and `Prog_Final_Exam` is **0.60**, indicating a strong linear relationship. This suggests that students who perform well on assignments tend to achieve higher final exam scores. It also confirms that assignments are a reliable predictor in this course.

- **Imperative Programming:** The correlation between `Improg_Assignment` and `Improg_Final_Exam` is slightly lower at **0.53**, which still indicates a moderately strong relationship, but implies slightly more variability in how assignment performance translates to final exam outcomes.

- Additionally, the correlation between `Prog_Assignment` and `Improg_Assignment` is **0.46**, suggesting that students who are strong

in one course tend to perform reasonably well in the other, likely due to underlying academic ability or motivation.

Weak or Negative Correlations in Early Weekly Exams

- For Programming, `Prog_First_Weekly_Exam` shows a surprising **negative correlation** with later exams (e.g., `Prog_Second_Weekly_Exam`: **-0.20**). This may reflect a recovery pattern: students who initially struggle might improve significantly by later weeks.
- In Imperative Programming, early weekly exams such as `Improg_First_Weekly_Exam` show low positive correlations with final performance (**0.08**–**0.16**). This indicates that early scores are poor predictors of overall success, likely due to a steeper learning curve or evolving skill requirements.

Moderate Correlations Between Later Weekly Exams and Final Performance

- `Prog_Sixth_Weekly_Exam` has a correlation of **0.36** with `Prog_Assignment`, suggesting that later weekly exams better reflect the skills assessed in the assignment and final exam.
- In contrast, the correlation between `Improg_Sixth_Weekly_Exam` and `Improg_Final_Exam` is only **0.29**, indicating a weaker alignment between weekly evaluations and overall performance.

Differences in Assessment Structure Between Subjects

The Programming final exam shows a stronger overall dependency on structured, incremental work (i.e., weekly tests and assignments). In contrast, Imperative Programming seems to demand more holistic understanding, possibly involving abstract thinking or problem-solving skills that are harder to quantify and model.

Implications for Predictive Modeling

- Features in Programming are more linearly related to the final exam, resulting in better model performance and more reliable generalization.

- The weaker and less consistent correlations in Imperative Programming explain the lower R^2 scores and buffer-zone accuracy observed earlier.

3.9 Conclusion

This study explored multiple machine learning and deep learning model families to predict student performance based on weekly assessments, assignments, and contextual course metadata. After extensive experimentation, three top-performing variants were selected for further testing on a combined dataset: the **Bagged Random Forest**, the **Stacked SVM**, and the **Optimized FastAI model**.

Among these, the **Bagged Random Forest** trained on the **6 Weekly Exams dataset with group and time information**, combined with the **Buffer-Zone Grading** method, proved to be the most reliable configuration. This model achieved the most balanced performance across both regression and classification metrics, demonstrating strong generalization capabilities. The Buffer-Zone Grading system was essential in this context, as it provides a flexible boundary that helps identify at-risk students near grade thresholds—making it the most appropriate method for real-time educational applications.

However, when evaluating these models on the combined dataset, which integrates students' results across two different subjects into a single row, performance became notably less consistent—especially for *Imperative Programming*. Correlation analysis confirmed that Programming exhibits stronger and more linear relationships between assignments and final exam scores, while *Imperative Programming* shows weaker and less predictable patterns, which likely contributed to the observed performance gap. This shift was attributed to a significant reduction in dataset size (only 309 rows), the structural complexity of cross-subject prediction, and weaker feature-to-outcome correlations in the more abstract course. As such, while the models show high potential, the combined dataset in its current form is not yet suitable for robust real-time prediction and early intervention purposes.

3.9.1 Future Work

To bridge the gap toward real-time application, future work should focus on targeted data collection strategies that ensure higher coverage and consistency across

courses. More specifically:

- **Continuous Data Expansion:** Increase the number of students included in the dataset by systematically integrating data from additional semesters and related courses.
- **Incremental and Continuous Learning:** Fine-tune the model incrementally using new data collected each semester, ideally retraining at the end of each academic year using student outcomes.
- **Real-Time Feedback Integration:** Develop pipelines that allow real-time updating of student records, enabling timely predictions and flagging at-risk students early on.
- **Collection of Additional Influential Features:** Future efforts could also include gathering data on other factors that may influence student performance. Examples include:
  - Class attendance or engagement metrics (e.g., participation in labs, lectures)
  - Previous academic background (e.g., entrance scores, GPA, advanced-level math or informatics)
  - Demographic and psychological variables (e.g., age, motivation level, or study habits)
  - Time spent on assignments or quizzes, or other behavioral logs from learning management systems

These efforts would not only improve model accuracy and fairness ,but also ensure that the system remains adaptive, transparent, and effective in real-world educational settings.

Chapter 4

Implementation

This document describes the API implementation [26] for real-time student performance prediction using a Bagged Random Forest model. The model is trained on 6 Weekly Exams dataset with group and time features and supports multiple grading methodologies:

- Performance-Tiered Grading
- Subject-Based Grading
- Buffer-Zone Grading

.

4.1 API Endpoints

4.1.1 Home Endpoint

URL: / Method: GET **Description:** Returns a simple status message.

Response:

```
{
"message": "API is running"
}
```

4.1.2 Student Grade Prediction

URL: `/predict` **Method:** POST **Description:** Receives student exam data and predicts final grades.

Request Body Example:

```
{
  "students": [
    {
      "Group": 1,
      "Day": 2,
      "Start_Hour": 10,
      "First_Weekly_Exam": 1,
      "Second_Weekly_Exam": 1.5,
      "Third_Weekly_Exam": 0.5,
      "Fourth_Weekly_Exam": 2,
      "Fifth_Weekly_Exam": 1.5,
      "Sixth_Weekly_Exam": 2,
      "Assignment": 25.0
    }
  ]
}
```

Response Example:

```
{
    "predictions": [
        {
            "bagged_rf_prediction": 40.63,
            "bagged_rf_classification_performance_tiered": "HIGH",
            "bagged_rf_classification_subject_based": "GOOD",
            "bagged_rf_classification_buffer_zone": "EXCELLENT"
        }
    ]
}
```

4.2 Building and Running the Docker Container

1. **Build the Docker image:**

```
docker build -t student-performance-api .
```

2. **Run the container:**

```
docker run -p 8000:8000 student-performance-api
```

3. **Access the API:**

   - `http://localhost:8000/` - Check API status.
   - `http://localhost:8000/predict` - Make student predictions.

Acknowledgements

I would like to express my sincere gratitude to Dr. Győző Horváth and Dr. Norbert Pataki for providing the datasets used in this study. Dr. Győző Horváth contributed data from the Foundation of Programming course, while Dr. Norbert Pataki supported this work with data from the Imperative Programming course. I would also like to thank Dr. Máté Cserép for facilitating access to these datasets and making them available for my use.

Bibliography

[1] João Pires, Jorge Bernardino, et al. "Predicting Student Performance in Introductory Programming Courses". In: *Computers* (2024). DOI: 10.3390/computers13090219.

[2] Yutong Liu et al. "Predicting Student Performance Using Clickstream Data and Machine Learning". In: *Education Sciences* 13.1 (2023). DOI: 10.3390/educsci13010017. URL: https://www.mdpi.com/2227-7102/13/1/17.

[3] Alireza Ahadi et al. "Exploring Machine Learning Methods to Automatically Identify Students in Need of Assistance". In: *Proceedings of the Eleventh Annual International Conference on International Computing Education Research*. ICER '15. New York, NY, USA: Association for Computing Machinery, 2015, pp. 121–130. ISBN: 9781450336307. DOI: 10.1145/2787622.2787717. URL: https://doi.org/10.1145/2787622.2787717.

[4] Cameron Cooper. "Using Machine Learning to Identify At-Risk Students in an Introductory Programming Course at a Two-Year Public College". In: *Advances in Artificial Intelligence and Machine Learning* 2.2 (2022), pp. 407–421.

[5] Mona M. Jamjoom et al. "Early Prediction for At-Risk Students in an Introductory Programming Course Based on Student Self-Efficacy". In: *Informatica* 45 (2021), pp. 1–9. DOI: 10.31449/inf.v45i6.3528.

[6] M. Sivasakthi and M. Pandiyan. "Machine Learning Algorithms to Predict Students' Programming Performance: A Comparative Study". In: *Journal of University of Shanghai for Science and Technology* 24 (2022), pp. 1–8.

[7] Ashok Kumar Veerasamy et al. "Using Early Assessment Performance as Early Warning Signs to Identify At-Risk Students in Programming Courses". In: *2020*

IEEE Frontiers in Education Conference (FIE). 2020, pp. 1–9. DOI: 10.1109/FIE44824.2020.9274277.

[8] Guohua Shen et al. "The Prediction of Programming Performance Using Student Profiles". In: *Education and Information Technologies* 28.1 (2023), pp. 725–740. ISSN: 1573-7608. DOI: 10.1007/s10639-022-11146-w. URL: https://doi.org/10.1007/s10639-022-11146-w.

[9] Kissinger Sunday et al. "Analyzing Student Performance in Programming Education Using Classification Techniques". In: *International Journal of Emerging Technologies in Learning (iJET)* 15.02 (2020), pp. 127–144. DOI: 10.3991/ijet.v15i02.11527. URL: https://online-journals.org/index.php/i-jet/article/view/11527.

[10] Afra K. Shahiri, W.M. Husain, and Nur H. Rashid. "A Review on Predicting Student's Performance Using Data Mining Techniques". In: *Procedia Computer Science* 72 (2015), pp. 414–422. DOI: 10.1016/j.procs.2015.12.157.

[11] Ijaz Khan et al. "Tracking Student Performance in Introductory Programming by Means of Machine Learning". In: *2019 4th MEC International Conference on Big Data and Smart City (ICBDSC)*. 2019, pp. 1–6. DOI: 10.1109/ICBDSC.2019.8645608.

[12] Aya Nabil, Mohammed Seyam, and Ahmed Abou-Elfetouh. "Prediction of Students' Academic Performance Based on Courses' Grades Using Deep Neural Networks". In: *IEEE Access* 9 (2021), pp. 140731–140746. DOI: 10.1109/ACCESS.2021.3119596.

[13] Ihsan A. Abu Amra and Ashraf Y. A. Maghari. "Students performance prediction using KNN and Naïve Bayesian". In: *2017 8th International Conference on Information Technology (ICIT)*. 2017, pp. 909–913. DOI: 10.1109/ICITECH.2017.8079967.

[14] M. Sivasakthi. "Classification and Prediction Based Data Mining Algorithms to Predict Students' Introductory Programming Performance". In: *2017 International Conference on Inventive Computing and Informatics (ICICI)*. 2017, pp. 346–350. DOI: 10.1109/ICICI.2017.8365371.

[15] L. Breiman. "Random Forests". In: *Machine Learning* 45.1 (2001), pp. 5–32. URL: https://www.stat.berkeley.edu/~breiman/randomforest2001.pdf.

[16] G. Biau. "Analysis of a Random Forests Model". In: *Journal of Machine Learning Research* 13 (2012), pp. 1063–1095. URL: https://www.jmlr.org/papers/volume13/biau12a/biau12a.pdf.

[17] G. Louppe. "Understanding Random Forests: From Theory to Practice". In: *arXiv preprint arXiv:1407.7502* (2014). URL: https://arxiv.org/abs/1407.7502.

[18] C. Cortes and V. Vapnik. "Support-vector networks". In: *Machine Learning* 20.3 (1995), pp. 273–297. URL: https://doi.org/10.1007/BF00994018.

[19] Y. LeCun, Y. Bengio, and G. Hinton. "Deep learning". In: *Nature* 521.7553 (2015), pp. 436–444. URL: https://doi.org/10.1038/nature14539.

[20] D. P. Kingma and J. Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014). URL: https://arxiv.org/abs/1412.6980.

[21] Scikit-learn Developers. *Scikit-learn Documentation*. 2024. URL: https://scikit-learn.org/stable/.

[22] Scikit-learn Developers. *GridSearchCV – Scikit-learn documentation*. 2025. URL: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html.

[23] DreamQuark-AI. *PyTorch TabNet: Attentive Interpretable Tabular Learning*. Accessed: 2025-02-26. 2023. URL: https://github.com/dreamquark-ai/tabnet.

[24] Optuna Contributors. *Optuna: An Automatic Hyperparameter Optimization Framework*. Accessed: 2025-02-26. 2023. URL: https://github.com/optuna/optuna.

[25] Fastai Team. *fastai: A Deep Learning Library for Practical Applications*. Accessed: 2025-02-26. 2023. URL: https://github.com/fastai/fastai.

[26] Csonka László. *Student Predictions API*. https://github.com/LackosS/msc-thesis/tree/main/final_work/student_predictions_api. Accessed: 2025-04-03. 2025.

List of Figures

List of Tables