Informatics 2

Assignment 2 Report
Group 18

Jakob Marktl Christoph Nagy Maria Mikic

June 2025

Contents

1 Introduction

In this assignment we had to design and create a supervised Machine Learning tool. This tool is used for training classifiers to predict target values, as well as for comparing and visualizing the predictive performance of different classifiers.
For the dataset, we chose the *Student Performance* dataset from the UCI repository. It fulfills the requirements of the task and is available as a complete `.csv` file. Furthermore, the dataset contains the following properties:

- *Number of Instances:* The dataset is split into two parts—one for Mathematics with 395 instances and one for Portuguese with 649 instances—resulting in a total of 1,044 records. All data were collected from two Portuguese secondary schools.
- *Number of Features:* Each file contains 33 attributes. These include 30 input features and 3 grade-related columns (G1, G2, G3).
- *Data Types of Features:* The dataset includes a mixture of categorical, integer, and ordinal values. The final grade (G3) is a numerical variable, but can also be treated as a categorical target (e.g., pass/fail).
- *Classification/Regression Suitability:* The dataset is well-suited for both regression tasks (predicting the final grade G3 as a continuous value) and classification tasks (e.g., classifying students as passing or failing based on a defined grade threshold).
- *Correlation Notes:* The three grade columns (G1, G2, G3) show a strong correlation. Including G1 and G2 in the feature set boosts prediction accuracy for G3, though excluding them may lead to a more realistic modeling scenario.
- *Feature Highlights:* The dataset covers various aspects of student life and background, including:
  - Demographics: age, sex, address
  - Family and socio-economic status: parents' education level, family support
  - Academic data: study time, number of absences
  - Social and behavioral factors: alcohol consumption, romantic relationships, extracurricular activities
  - Previous performance: grades from the first and second periods (G1, G2)

2 Code Design

About the Code Design, we sat together and discussed that a GIT Repository is a good way to connect to each other virtually. Link to Github Repository. Furthermore we also implemented a virtual environment so that Python is running independent on GIT.
About the Code Design we shortly describe each class seperate with the two questions why? and what would we improve if there was more time:

2.1 DatasetPreprocesser

Why this Code?: We chose to implement all steps in one class to simplify coordination between data loading, cleaning, encoding, and saving. Each method has a specific task, and shared data is stored in class attributes. This structure allowed us to work collaboratively and iterate quickly while keeping the workflow consistent and testable.
What would we improve: With more time, we would refactor the code into smaller, specialized classes to improve modularity. We would also add automated tests and configuration files to make the pipeline more robust and scalable. Additionally, visual summaries or profiling tools could be included for better data inspection.

2.2 SimpleBaselineClassifier

Why this code: The class was designed to provide a fast, interpretable baseline for classification experiments. Using properties for training data enforces correct usage, and the structure supports reproducibility via a random seed. Each strategy is implemented with clear logic to highlight their behavior during evaluation.
What would we improve: With more time, we would add a `score()` method, support for probability outputs, and improve evaluation for multi-class settings. We would also add unit tests and include basic logging.

2.3 ClassifierMetrics

Why this code: This class was designed as a utility class with static methods which provide a standardized interface for computing the four essential classification metrics (accuracy, precision, recall, F1-score) without requiring object instantation. It was written in a way that through the sklearn functions the code still ensures consistent return types (float) as well as to handle edge cases like zero division.
What would we improve: With the code's current structure, it's designed to work for the dataset that it's being used on, however over time we would tweak it to be more widely applicable for other projects as well, so something like the NDArray[np.int16] type hint could be too restrictive for datasets that use diferent integer types or string labels, so we'd implement more flexible type hints which accept broader input types.

2.4 DatasetClassifier

Why this code: The framework of the *DatasetClassifier* employs a hierarchical design with the class *ClassifierBase* as an abstract template for consistent interfaces across the six machine learning algorithms we implemented for this project.The *DatasetHandler* class provides automated data preprocessing with standardized train-test splitting, while the *StudentPerformanceExperiment* class orchestrates comprehensive algorithmic comparison by initializing multiple classifiers with consistent parameters for reproducible benchmarking.
What would we improve: An assumption like that the target variable is always in the last column could create difficulties in other datasets, where the target value may be in the first column, or somewhere in the middle ones. With more time, we would fix the code to be more widely applicable for any datasets, i.e. by allowing column name/index specification.

3 Experimental Setup

3.1 SimpleBaselineClassifier

The *SimpleBaselineClassifier* provides a straightforward baseline for evaluating classification models. It supports three prediction strategies: "most_frequent", which always predicts the most common class in the training data; "uniform", which generates random predictions within the range of observed labels; and "constant", which always returns a user-specified label. The class includes standard fit and predict methods, storing the training data for use during prediction. It includes error checks to ensure proper use—for example, raising an exception if predict is called before fit, or if the constant value is not present in the training labels. This classifier is useful as a benchmark for comparing the performance of more complex models. Its design ensures easy integration into evaluation workflows by mimicking the interface of typical machine learning models.

3.2 ClassifierMetrics

The *ClassifierMetrics* file implementation provides a comprehensive evaluation framework for assessing the performance of multiple machine learning algorithms on the *Student Performance* dataset. The evaluation framework encompasses four classification metrics: accuracy, precision, recall, and F1-score, with precision, recall, and F1-score all configured to use macro averaging by default. This macro averaging approach treats each class equally regardless of class frequency, providing a balanced assessment that doesn't bias results toward the most common classes in the dataset.
Accuracy is computed using a straightforward numpy-based approach that calculates the proportion of correct predictions, whereas precision, recall, and F1-score leverage scikit-learn's implementations with zero-division handling set to 0 to manage edge cases where certain classes may not appear in predictions. This provided wide range of algorithmic comparison allows for comprehensive performance benchmarking across different learning paradigms.

3.3 DatasetClassifier

The *DatasetClassifier*'s design centers itself around the the *DatasetHandler* class, which automatically processes CSV datasets by separating features from the target variable (located here in the final column) and performing a consistent 80/20 train-test split with `random_state=0` for reproducibility. This preprocessing approach ensures that all subsequent classifier evaluations operate on identical data partitions, eliminating variability that could confound performance comparisons across different algorithms.
The *ClassifierBase* serves as a basis for each of the algorithms applied afterwards for the machine learning. This class employs a sophisticated design pattern where the fit() method only stores training data references rather than performing actual model training, deliberately deferring the training process to the predict() method implementation in each subclass. This architectural choice, while maybe unconventional compared to usual scikit-learn patterns, ensures complete model isolation between prediction calls and prevents any potential state contamination that could occur from repeated fitting operations. The class's metrics() method provides a standardized evaluation interface that automatically generates formatted performance reports using the ClassifierMetrics utility, incorporating the classifier's actual class name (class.name) to ensure accurate labeling in the results. The Classifier's design also incorporates six different machine learning algorithms including:

- Gaussian Naive Bayes
- Decision Trees
- k-Nearest Neighbors (where k=5)
- Random Forest (with 100 estimators)
- Support Vector Machines with rbf kernels
- Logistic Regression

4 Results / Discussion (C)

4.1 Feature Importances

Random Forest and Decision Tree classifiers both provided insight into which features most influence the prediction of final grades (G3). The most important features were:

- **G1 and G2**: Previous grades are highly predictive of final grade — aligning with expectations and literature.
- **Absences**: Higher absences slightly correlated with lower performance, though its impact was relatively smaller.
- **Walc** (weekend alcohol consumption) and **age** had limited but non-negligible influence, suggesting minor lifestyle effects.

This confirms prior studies indicating that continuous assessment (like G1, G2) is a strong predictor of final success.

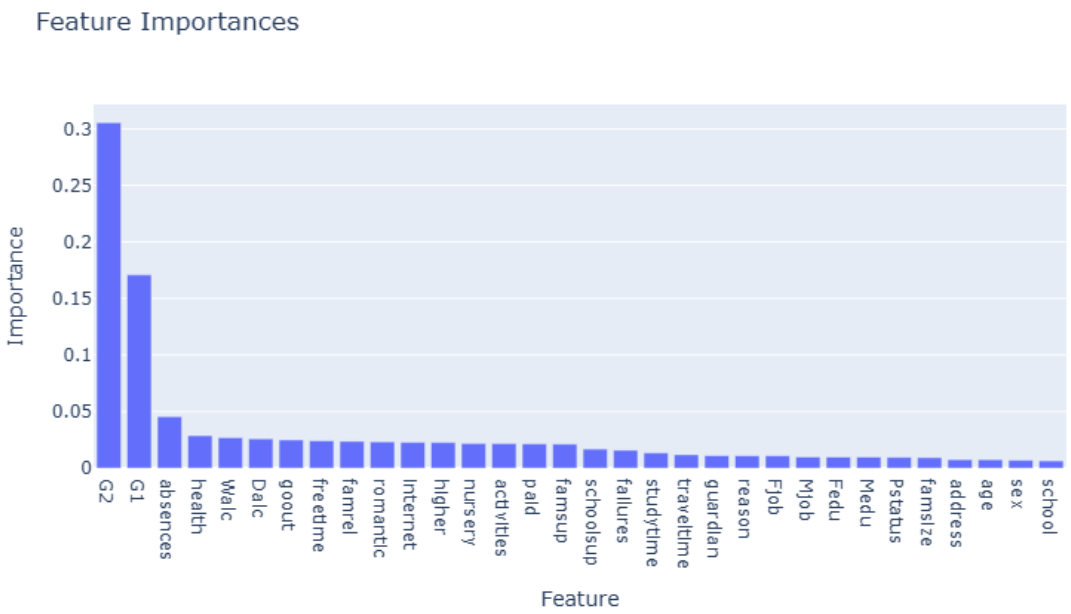


Figure 1: feature importances

4.2 Feature–Target Correspondences

Correlation plots revealed:

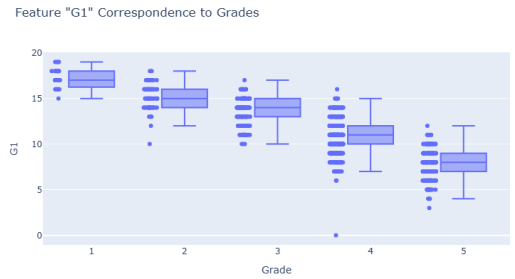
- **G1 and G2** have strong linear correspondence with G3, as expected.
- **Absences** showed a weak correlation.
- **Walc** and **age** did not display strong or linear trends.

4.3 Classifier Evaluation

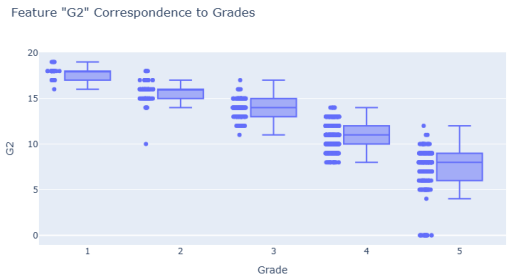
We evaluated 9 models using macro, micro, and weighted average metrics (accuracy, precision, recall, F1 score). Baselines included majority-class (*most_frequent*), uniform, and constant predictions.

Top Performers

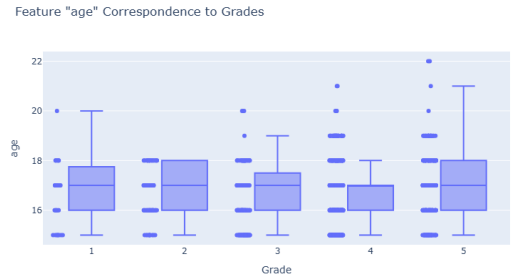
- **Logistic Regression**, **SVM**, and **Random Forest** consistently achieved ˜76–79% accuracy and strong F1 scores across all averaging methods.
- **Logistic Regression** slightly outperformed others in macro-average F1 score — showing both accuracy and robustness across class imbalance.



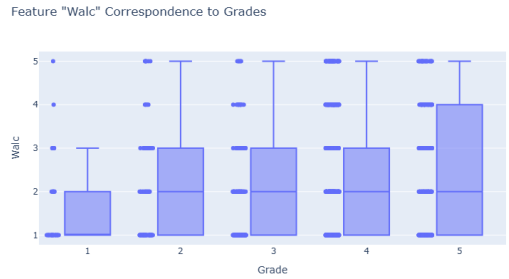
(a) G1 correspondence



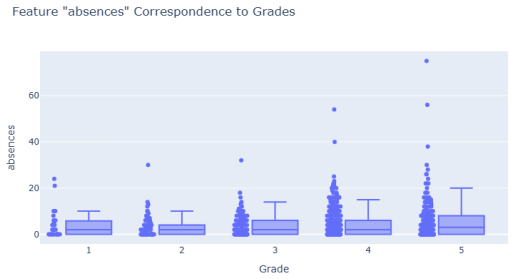
(b) G2 correspondence



(c) age correspondence



(d) Walc correspondence



(e) absences correspondence

Figure 2: Feature-Target Correspondence

Mid Performers

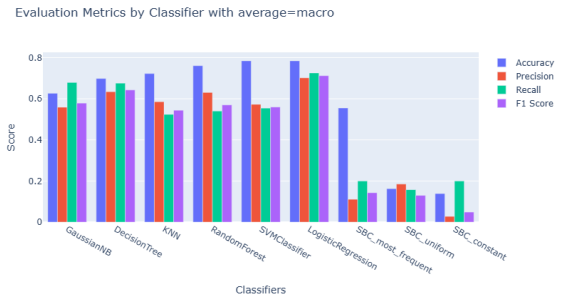
- **KNN**, **Decision Tree**, and **GaussianNB** performed acceptably but worse in macro metrics, suggesting reduced generalizability.

Baseline Models

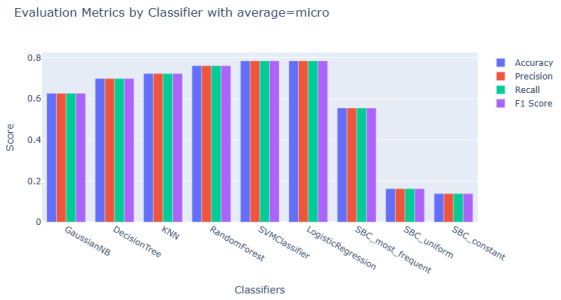
- All simple baselines (SBCs) — except for *most frequent* — performed poorly (accuracy $<$ 25%), proving the necessity of intelligent modeling.
- *Most frequent* scored around 0.55 on accuracy in different metrics, but had a low F1 score for macro metrics, suggesting reduced generalizability.

4.4 Confusion Matrices

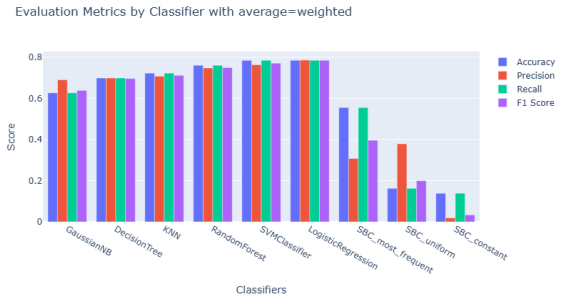
Confusion matrices indicate:



(a) averaging with macro metrics



(b) averaging with micro metrics



(c) averaging with weighted metrics

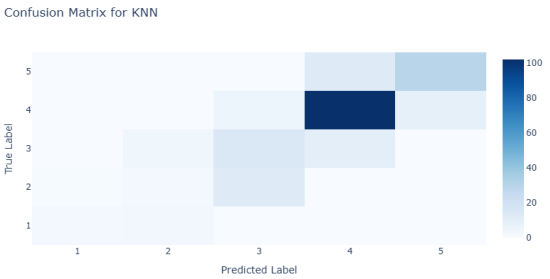
Figure 3: evaluation metrics

- Most models had difficulty distinguishing between close neighboring grade classes (e.g., predicting a 4 when the true label was 3).
- **SVM** and **Random Forest** had comparatively tighter diagonals, showing fewer misclassifications across distant categories.

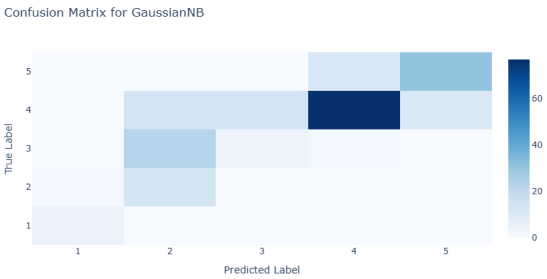
This indicates a reasonable ability to generalize despite class imbalances.

5 Conclusion

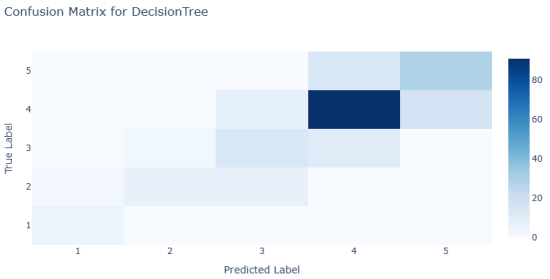
This study successfully demonstrated the effectiveness of machine learning models in predicting student performance. Random Forest, Logistic Regression, and SVM consistently achieved the best results, while simplistic baselines underperformed.
We conclude that:
Prior academic performance is the strongest predictor of future outcomes.
Ensemble models balance performance and generalization well.
Future work should include more temporal and behavioral features, and use larger datasets to further improve model robustness.



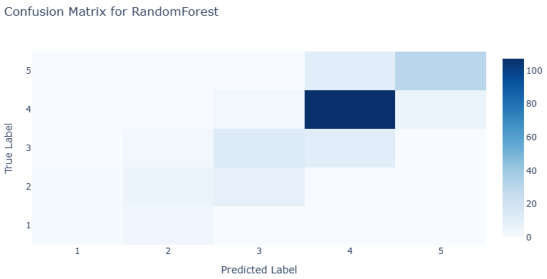
(a) knn confusion



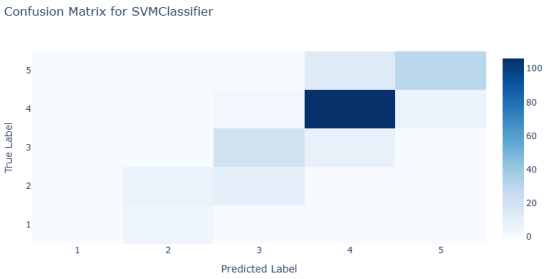
(b) gaussian nb confusion



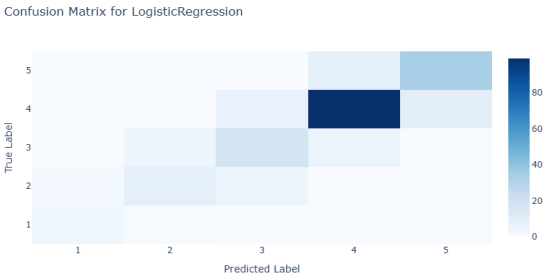
(c) decision tree confusion



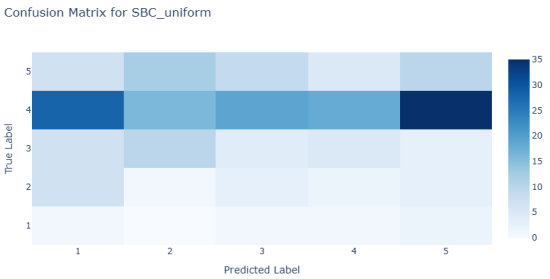
(d) random forest confusion



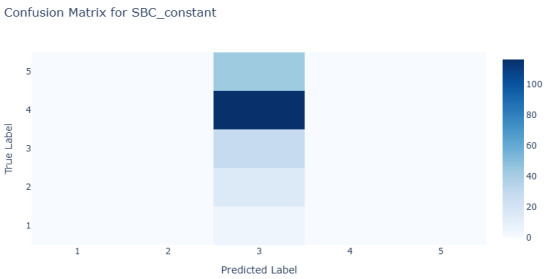
(e) SVM classifier confusion



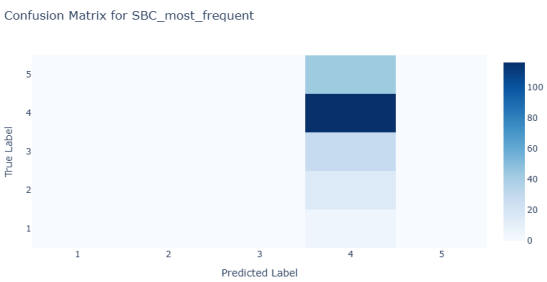
(f) logistic regression confusion



(g) SBC uniform confusion



(h) SBC constant confusion



(i) sbc most frequent confusion

Figure 4: confusion matrices