CS5100 Final Project Report
Student Success Prediction (Phase 1 & Phase 2)

Zehan Wang

Fall 2025

1 Introduction

This project builds a machine learning pipeline to predict whether a student is *at risk of failing* based on the UCI student performance dataset. The target label is a binary variable

$$\texttt{at\_risk} = \begin{cases} 1 & \text{if final grade } G3 < 10, \\ 0 & \text{otherwise.} \end{cases}$$

The project is divided into two phases:

- **Phase 1**: Implement a correct and measurable baseline pipeline, including preprocessing, a Gradient Boosting model, and a Random Forest implemented from scratch.
- **Phase 2**: Extend and improve the pipeline through additional scope items such as using the full dataset, feature selection, and ensemble methods.

The full code for this project is publicly available at:

https://github.com/ZehanWang810/CS5100-Final-Project

2 Data and Task

The dataset is derived from the UCI "Student Performance" dataset and includes demographic, socio-economic, and academic features. In Phase 1, I mainly worked with a small "mini" subset (about 10% of the full data), and in Phase 2 I extended the experiments to the full dataset.

Key properties of the data used in this project:

- **Mini dataset**: ≈ 39 rows, 34 columns after preprocessing.
- **Full dataset**: 395 rows, 56 raw columns before preprocessing.
- The feature types include both numeric (e.g., age, number of past failures) and categorical (e.g., school, address, family support).

3 Phase 1: Baseline Pipeline

Phase 1 focused on constructing a correct, testable pipeline and implementing the required models.

3.1 Preprocessing

The preprocessing function `preprocess_data` performs the following steps:

1. **Load** the raw CSV into a pandas DataFrame.
2. **Create target** `at_risk` from the final grade $G3$:

$$\texttt{at\_risk} = \mathbf{1}[G3 < 10].$$

3. **Drop leakage columns** G1, G2, G3 so that partial or final grades are not directly used as features.
4. **Encode categorical variables** using one-hot encoding.
5. **Impute** missing values (if any) with simple strategies such as mean or mode.
6. **Scale numeric columns** to the $[0, 1]$ range.
7. Return a clean DataFrame containing **only** numeric columns and the target.

This preprocessing passes the autograder checks for schema, data types, and absence of leakage.

3.2 Models Implemented

Two models were implemented:

Gradient Boosting (GB). A standard Gradient Boosting classifier from `scikit-learn` is used inside a `Pipeline` that chains preprocessing and the classifier. This serves as a strong baseline.

Random Forest from Scratch (RF). A custom `RandomForest` class was implemented without using `sklearn`'s RandomForest APIs. Key design points:

- Bagging with bootstrap samples for each tree.
- A simple decision tree (with max depth control) for each estimator.
- Majority voting across trees at prediction time.

3.3 Evaluation Metrics

The primary metrics are:

- **F1 score**: balances precision and recall for the positive class (`at_risk` $= 1$).
- **ROC-AUC**: area under the ROC curve, measuring ranking quality across thresholds.

On the mini dataset, the Gradient Boosting model achieved an F1 of about 0.29 and ROC-AUC of about 0.47, while the custom Random Forest achieved an F1 of about 0.33. These values satisfy the minimum requirements for Phase 1. All 18 public autograder tests passed successfully.

4 Phase 2: Scope Extensions

As a solo student, I was required to complete at least 4 points of additional scope in Phase 2. I implemented the following three items:

Scope 1: Use the **full dataset** instead of the mini subset (1 point).

Scope 2: Apply **feature selection** using mutual information (1 point).

Scope 3: Implement an additional form of **ensemble learning** via stacking (2 points).

Together these satisfy the required number of scope points.

4.1 Scope 1: Full Dataset vs Mini Dataset

The first extension evaluates the models on the full dataset of 395 students instead of the mini set. Table 1 compares performance on mini vs full data.

Table 1: Scope 1 — Performance on mini vs full dataset. Values are rounded to three decimals.

Dataset	Model	F1	ROC-AUC
Mini	Gradient Boosting	0.286	0.469
Mini	Random Forest (scratch)	0.333	–
Full	Gradient Boosting	**0.406**	**0.676**
Full	Random Forest (scratch)	**0.424**	–

Observation. Moving from the mini dataset to the full dataset improves F1 by roughly 40–50% for both models. ROC-AUC for Gradient Boosting also increases from ≈ 0.47 to ≈ 0.68. This confirms that the mini dataset is too small and noisy, and that most of the performance gains in Phase 2 come simply from using more data.

4.2 Scope 2: Feature Selection via Mutual Information

The second extension performs feature selection based on mutual information between each feature and the `at_risk` label. The top 20 features by mutual information were inspected; the top 15 were used for training.

The most informative features include `Walc` (weekend alcohol consumption), `address_R` (rural vs urban), `reason_home`, `Fjob_health`, `reason_reputation`, `failures`, `paid_no`, `higher_yes`, `activities_no`, `nursery_yes`, `school_GP`, and several parental and time-related variables.

Table 2 summarizes the effect of feature selection on the full dataset.

Table 2: Scope 2 — Effect of mutual-information feature selection on the full dataset.

Setting	Model	F1	ROC-AUC
Baseline (all features)	Gradient Boosting	0.406	0.676
Baseline (all features)	Random Forest (scratch)	0.424	–
Top-15 features	Gradient Boosting	**0.444**	0.604
Top-15 features	Random Forest (scratch)	0.291	–

Observation. Gradient Boosting *benefits* from feature selection: its F1 score increases from 0.406 to 0.444. However, the ROC-AUC slightly decreases, suggesting that the ranking quality is similar but the chosen threshold becomes more favorable for the positive class.

In contrast, the custom Random Forest performs worse after feature selection. This suggests that the ensemble of shallow trees relies on a wider set of weaker signals, whereas Gradient Boosting can exploit a smaller set of strong features.

4.3 Scope 3: Stacking Ensemble

The third extension explores a stacking ensemble. The base models are the Gradient Boosting and custom Random Forest from the earlier experiments. Their predicted probabilities are used as input features to a Logistic Regression meta-learner.

Table 3 compares the stacking model with the individual base models on the full dataset.

Table 3: Scope 3 — Stacking ensemble vs individual models on the full dataset.

Model	Description	F1	ROC-AUC
GB Baseline	Gradient Boosting	0.406	**0.676**
RF Baseline	Random Forest (scratch)	**0.424**	–
Stacking	GB + RF → Logistic Regression	0.340	0.665

Observation. The stacking model achieves ROC-AUC comparable to Gradient Boosting but a lower F1 score. One likely reason is that the two base models are not diverse enough: both rely on similar tree-based representations, so the meta-learner does not gain much additional information. Nonetheless, the implementation demonstrates a correct use of stacking and satisfies the "additional ensemble method" scope requirement.

5 Overall Discussion

Across all extensions, several patterns emerge:

- **Data quantity matters more than model complexity.** Moving from the mini dataset to the full dataset yields the largest improvements.
- **Feature selection can help gradient boosting.** Mutual-information feature selection improved GB's F1 and simplified the model, at the cost of a small drop in ROC-AUC.
- **Random forests prefer many weak features.** The custom Random Forest degraded when only the top 15 features were used.
- **Stacking is not guaranteed to help.** Without sufficient diversity between base models, stacking may not outperform the best individual model.

Overall, the final pipeline is correct, passes all tests, and supports quantitative comparisons between different modeling choices.

6 Reflection

This section addresses the reflection questions from the project rubric.

1. What was your favorite part of the project?

My favorite part was implementing the custom Random Forest and then seeing it pass the autograder tests. Writing the bagging logic, decision tree fitting, and majority voting from scratch helped me understand how ensemble methods really work under the hood, instead of treating them as black-box `sklearn` calls.

2. What was your least favorite part of the project?

The least favorite part was debugging small mismatches with the expected preprocessing schema. For example, making sure that all categorical columns were one-hot encoded, that no object dtypes remained, and that the target column was correctly excluded from the feature matrix. These bugs were sometimes subtle but they were also realistic problems that occur in applied ML work.

3. On what topic from the course did your perspective change the most?

My perspective changed the most on the importance of **data preprocessing and evaluation**. Before this project, I tended to focus mostly on model choice (e.g., which classifier to use). After working through the pipeline and the autograder constraints, I realized that cleaning the data, avoiding leakage, choosing good metrics, and building a reproducible evaluation loop are just as important as the algorithm itself.

4. If you had more time, which scope item or technique would you like to improve or extend?

If I had more time, I would like to extend the ensemble methods by: If I had more time, I would improve the stacking ensemble model. I would try a better hyperparameter search and test different meta-learners such as a calibrated logistic regression or a small neural network. This could make the ensemble more stable and improve performance.

5. If you had more time, what new scope items / techniques would you try?

If time allowed, I would explore probabilistic models to estimate uncertainty in predictions. These models could provide confidence estimates and make the "at-risk" classification more interpretable and trustworthy.

6. Roughly how many hours did you spend on Phase 1 and Phase 2?

Phase 1: about 12 hours.
Phase 2: about 9 hours.

Appendix A: Top-15 Features by Mutual Information

The following features were selected as the top 15 by mutual information with the `at_risk` label:

- `Walc`
- `address_R`
- `reason_home`
- `Fjob_health`
- `reason_reputation`
- `failures`
- `paid_no`
- `higher_yes`
- `activities_no`
- `nursery_yes`
- `school_GP`
- `romantic_yes`

- guardian_mother
- Fedu
- traveltime

These features capture aspects of student behavior, family background, and previous academic performance that are intuitively related to academic risk.

Appendix B: Repository and Reproducibility

The full project repository is available at:

https://github.com/ZehanWang810/CS5100-Final-Project

To reproduce the experiments:

1. Clone the repository and `cd` into the project directory.
2. Create and activate a Python environment with the required packages (e.g., `pandas`, `numpy`, `scikit-learn`, `pytest`).
3. Run the Phase 1 tests:

   ```
   pytest tests/test_phase_1.py -q
   ```

4. Run the Phase 2 scripts:

   ```
   python phase2_scope1_full_data.py
   python phase2_scope2_feature_selection.py
   python phase2_scope3_stacking.py
   ```

5. The printed metrics should match (up to small rounding differences) the results reported in this document.