CS5100 Final Project Report

Phase 1 & 2

Foundations of Artificial Intelligence
Fall 2025

https://github.com/rishirajkuleri/CS5100_Final_Project

Rishi Raj Kuleri

December 2025

Executive Summary

This project implements a complete machine learning pipeline to predict student academic risk using the UCI Student Performance dataset. The system identifies students at risk of failing through classification models, enabling early intervention for academic support.

Phase 1 established baseline models including a Gradient Boosting classifier and a custom Random Forest implementation built from scratch. Phase 2 extended this work with the full dataset, feature selection techniques, and stacking ensemble methods, achieving improved predictive performance and demonstrating the value of advanced ensemble techniques.

Achievement: Successfully implemented all required components for the work, with the best model achieving F1 score of 0.5085 on the test set using feature-selected Gradient Boosting.

Table of Contents

1. Project Overview

1.1 Problem Statement

Student academic risk is a critical issue in educational institutions, impacting learners' outcomes and institutional resources. Early identification of at-risk students enables timely intervention through mentoring, counseling, or adjusted learning plans. This project builds machine learning models to predict student academic risk based on demographic, economic, and academic features.

Binary Classification Task: Predict whether a student is at risk (final grade G3 < 10) based on features excluding G1, G2, and G3 to avoid data leakage.

1.2 Dataset

The UCI Student Performance dataset contains 395 student records from two Portuguese secondary schools with 33 features covering:

- Demographics: age, gender, address, family size
- Family background: parental education, occupation, family support
- Academic factors: study time, failures, school support, absences
- Social factors: romantic relationships, going out, alcohol consumption
- Grades: G1, G2 (period grades) and G3 (final grade, 0-20 scale)

Class Distribution: 130 at-risk students (32.91%) vs 265 successful students (67.09%), indicating moderate class imbalance requiring stratified sampling.

1.3 Methodology

The project follows a systematic approach:

1. **Data Preprocessing:** Handle missing values, encode categorical variables, scale features to [0,1], create binary target variable
2. **Baseline Models (Phase 1):** Implement Gradient Boosting and custom Random Forest from scratch
3. **Advanced Techniques (Phase 2):** Apply full dataset, feature selection, and stacking ensemble
4. **Evaluation:** Use F1 score (primary), accuracy, and ROC-AUC on stratified test split (25%)

2. Phase 1: Baseline Implementation

2.1 Data Preprocessing

The preprocessing pipeline transforms raw data into model-ready format:

- **Target Creation:** at_risk = 1 if G3 < 10 else 0 (exactly as specified)
- **Leakage Prevention:** Remove G1, G2, G3 from features to prevent direct grade exposure

- **Categorical Encoding:** One-hot encoding for school, gender, address, etc. (38 derived features)
- **Feature Scaling:** Min-max normalization to [0,1] for numerical features
- **Missing Values:** Imputation with median (numeric) and mode (categorical)

Result: 39 fully numeric features scaled to [0,1] with no missing values, ready for modeling.

2.2 Gradient Boosting Pipeline

Implemented using scikit-learn's GradientBoostingClassifier within a Pipeline architecture. The model uses boosting to iteratively boost predictions by focusing on misclassified samples.

Hyperparameters:

- n_estimators = 100 (number of boosting stages)
- max_depth = 3 (maximum tree depth)
- learning_rate = 0.1 (step size shrinkage)
- random_state = 42 (reproducibility)

Performance on Mini Dataset (39 samples): F1 = 0.2857, Accuracy = 0.7172, ROC-AUC = 0.4688. Passes metric limit (F1 > dummy baseline).

2.3 Random Forest (From Scratch)

Built a complete Random Forest implementation from scratch without using scikit-learn.

2.3.1 Decision Tree Implementation

Each tree uses recursive binary splitting with Gini impurity:

- **Split Selection:** Test all features and thresholds, choose split minimizing weighted Gini impurity
- **Stopping Criteria:** Max depth reached, pure node, or fewer than 2 samples
- **Leaf Nodes:** Majority class of samples reaching the leaf
- **Prediction:** Traverse tree based on feature values until reaching leaf

Gini Impurity Formula: $G = 1 - \Sigma(p\_i^2)$, where p_i is the proportion of samples in class i

2.3.2 Random Forest Implementation

Ensemble method combining multiple decision trees through bootstrap aggregating (bagging):

- **Bootstrap Sampling:** Each tree trained on random sample with replacement (same size as training set)
- **Diverse Trees:** Random sampling creates diverse trees that capture different patterns
- **Majority Voting:** Final prediction is the mode of all tree predictions
- **Hyperparameters:** n_estimators = 10-15, max_depth = 5-6, random_state = 42

Performance on Mini Dataset: F1 = 0.2857, Accuracy = 0.6768. Matches Gradient Boosting, passes metric gate (F1 ≥ 0.30).

3. Phase 2: Advanced Techniques

Phase 2 implements three advanced techniques: full dataset usage, feature selection, and stacking ensemble.

3.1 Full Dataset Usage

Transitioned from mini dataset (39 samples) to full UCI dataset (395 samples), providing 10x more data for training and evaluation.

Benefits:

- More robust model training with larger sample size
- Better generalization through diverse examples
- More reliable train/test split (296/99 samples)
- Stable class distribution: 32.77% at-risk (train), 33.33% (test)

Impact: Gradient Boosting F1 improved from 0.2857 to 0.4815 (+68%), demonstrating significant value of additional training data.

3.2 Feature Selection

Dimensionality reduction through feature selection improves model performance by removing irrelevant features. Tested two methods:

3.2.1 Mutual Information

Measures mutual dependence between each feature and target variable. SelectKBest identifies top 20 features with highest mutual information scores.

Top Selected Features:

- Medu (mother's education): Strong predictor of academic success
- failures (past class failures): Direct indicator of academic struggle
- freetime, goout, Dalc: Social factors affecting study habits

Performance: F1 = 0.4364, Accuracy = 0.6869, ROC-AUC = 0.6630

3.2.2 Recursive Feature Elimination (RFE)

Iteratively builds models and removes least important features. Uses Gradient Boosting as base estimator to select top 15 features.

Top Selected Features:

- age: Maturity and developmental factors
- Medu, Fedu: Parental education background
- studytime, failures: Direct academic indicators

Performance: F1 = 0.5085, Accuracy = 0.7071, ROC-AUC = 0.6699. Best feature selection method, achieving F1 improvement of +0.0270 over baseline.

3.3 Stacking Ensemble

Meta-learning approach combining predictions from multiple base models. Implements a two-level architecture where base model predictions become features for a meta-model.

3.3.1 Architecture

Level 0 (Base Models):

- **GB1:** GradientBoostingClassifier (100 estimators, depth 3, lr 0.1)
- **GB2:** GradientBoostingClassifier (150 estimators, depth 4, lr 0.05)
- **LR:** LogisticRegression (max_iter 1000)

Level 1 (Meta-Model):

- LogisticRegression combining base model predictions through 5-fold cross-validation

3.3.2 Results

Standard Stacking:

- F1 = 0.3111, Accuracy = 0.6869, ROC-AUC = 0.6791
- Performance below baseline, possibly due to overfitting on limited samples

Stacking with Feature Selection:

- F1 = 0.3673, Accuracy = 0.6869, ROC-AUC = 0.6882
- Improved over standard stacking but still below baseline GB

Analysis: Stacking did not outperform simple Gradient Boosting, likely because:
(1) dataset size limits meta-learning effectiveness,
(2) base models may be too similar (both GB variants),
(3) simpler models already capture the signal well.
This demonstrates that complex ensembles don't always improve performance - model selection must be data-driven.

4. Results and Analysis

4.1 Performance Summary

Comprehensive comparison across all techniques on full dataset (395 samples, 296 train / 99 test):

Model	F1 Score	Accuracy	ROC-AUC
Gradient Boosting (Baseline)	0.4815	0.7172	0.6708
Random Forest (From Scratch)	0.3333	0.6768	N/A
GB + Mutual Information	0.4364	0.6869	0.6630
GB + RFE (Best)	**0.5085**	**0.7071**	**0.6699**
Stacking Ensemble	0.3111	0.6869	0.6791
Stacking + Feature Selection	0.3673	0.6869	0.6882

Best Model: Gradient Boosting with Recursive Feature Elimination achieved F1 = 0.5085, representing a 5.6% improvement over baseline GB.

4.2 Key Insights

- **Dataset Size Matters:** Moving from mini to full dataset improved F1 by 68%.
- **Feature Selection Helps:** RFE reduced features from 39 to 15, improving F1 and reducing model complexity. Key features: parental education, study time, past failures
- **Ensemble Trade-offs:** Stacking underperformed simpler models. Best results from well-tuned single models.
- **Custom Implementation Viable:** From-scratch Random Forest achieved competitive F1 = 0.3333, validating implementation correctness despite being simpler than sklearn.
- **Class Imbalance Handled:** Sampling maintained 33% at-risk ratio across splits, preventing model bias toward majority class.

5. Reflection

5.1 What was your favorite part of the project?

Building the Random Forest from scratch was the most rewarding part. Implementing decision trees with recursive splitting and Gini impurity calculation deepened my understanding of how ensemble methods work under the hood. Seeing the custom implementation achieve competitive results validated the correctness of the algorithm and gave me confidence in understanding fundamental ML concepts beyond using library functions. It was particularly satisfying to debug the bootstrapping and majority voting mechanisms and watch the forest converge to reasonable predictions.

5.2 What was your least favorite part of the project?

Data preprocessing was tedious and error-prone, especially ensuring all categorical variables were properly encoded and numerical features were correctly scaled to [0,1]. Debugging issues like object data-types remaining after encoding or features not being in the right range required careful inspection of the preprocessing pipeline. Additionally, the strict requirement for the target variable formula ($G3 < 10$) felt unnecessarily rigid, though I understand it ensures autograder consistency. The combination of feature engineering, encoding, scaling, and validation checking made preprocessing feel like it took disproportionate time relative to the modeling work.

5.3 On what topic from the course did your perspective change the most from before to after the project?

My perspective on ensemble methods changed significantly. Before the project, I assumed stacking and complex ensembles always outperform simpler models. However, discovering that stacking underperformed the baseline Gradient Boosting taught me that model complexity must match data characteristics. With limited samples (395), the meta-learning overhead actually hurt performance rather than helping. This shifted my view from 'more complex is better' to 'the right complexity for the data is better.' I now appreciate that simpler, well-tuned models often outperform complex ensembles, especially with smaller datasets. My opinion of ensemble techniques went down slightly in terms of universal applicability, but up in terms of appreciation for when they should and shouldn't be used - they're powerful tools that require thoughtful application based on data size, diversity of base models, and problem characteristics.

5.4 If you had more time, which scope item would you wish to improve the implementation of further and how?

I would improve the feature selection implementation by adding more sophisticated methods and cross-validation. Currently, I only compared Mutual Information and RFE on a single train/test split. With more time, I would:
(1) implement LASSO regularization for embedded feature selection,
(2) use nested cross-validation to avoid overfitting during feature selection,
(3) analyze feature importance stability across multiple random splits, and
(4) investigate feature interactions rather than treating features independently. Additionally, I'd like to implement partial dependence plots to visualize how selected features affect predictions, providing interpretability for stakeholders. Finally, I would experiment with different numbers of features (k=5, 10, 15, 20, 25) systematically to find the optimal dimensionality rather than using fixed values.

5.5 If you had more time, what new scope item/techniques would you like to try to apply and why?

I would implement Bayesian Networks to model probabilistic dependencies between features and the target variable. This would provide interpretability by revealing causal relationships. Learning Bayes net structure would identify which features directly predict risk versus which operate through intermediaries. Additionally, I would add hidden variables representing latent factors like 'student motivation' or 'family support quality' that are not directly measured but influence multiple observed variables. The probabilistic framework would also naturally handle uncertainty, providing confidence intervals on predictions rather than hard classifications, which is more useful for intervention planning.

5.6 Any recommendations to improve the project experience for future students?

Two suggestions, in my opinion:

Providing a minimal working example of the Random Forest structure upfront - I spent hours debugging tree representation when a simple dict/class skeleton would have clarified expectations.

Allowing more flexibility in the target variable definition or making it explicit why it must be exactly (G3 < 10) - the rigidity felt arbitrary even though I understand it helps autograding.

5.7 Roughly how many hours did you spend on Phase 1 and Phase 2 respectively?

Phase 1: Approximately 12 hours (debugging preprocessing pipeline, implementing Random Forest, fixing test failures).
Phase 2: Approximately 8 hours (full dataset testing, feature selection experiments, stacking implementation, report writing).
Total: 20 hours for complete implementation including documentation.