Checkpoint 2: Baseline Model and Results
CPSC 4300/6300 Applied Data Science

Michael Joseph Ellis

September 27, 2025

Model Choice and Rationale

We frame the task as multi-class classification of `GradeClass` (0=A, 1=B, 2=C, 3=D, 4=F). Guided by our EDA, we select gradient-boosted decision trees (XGBoost)[1] because they:

- handle mixed numeric/categorical inputs and non-linear feature interactions;
- are robust to monotone transformations and outliers in features like `Absences`;
- provide useful diagnostics (feature importance, confusion matrices) for interpretation.

We use `objective=multi:softprob`, `tree_method=hist`, and native categorical support (integer-coded categories). We drop `StudentID` and `GPA` to avoid identifier leakage and using the continuous target precursor.

Evaluation Protocol

We reserve 20% of the data as a held-out test set with stratification (random seed 42). On the remaining 80% training portion, we estimate baseline generalization via 5-fold *StratifiedKFold* cross-validation with shuffling, scored by macro-averaged F1 (macro-F1) to account for class imbalance (many F's). Importantly, cross-validation is performed strictly within the training split; the test set remains untouched until the very end.

For model training, we further carve out a small validation split from the training data (again stratified) that is used only to monitor training; we do not perform hyperparameter tuning in this baseline. Features require no scaling for tree models. Categorical predictors are provided as pandas `category` dtype to XGBoost's native categorical handling, while `StudentID` and `GPA` are dropped to prevent target leakage.

We report: (i) cross-validated macro-F1 on the training split; (ii) held-out test accuracy and macro-F1; (iii) per-class precision/recall/F1 via `classification_report`; and (iv) both raw-count and row-normalized confusion matrices for qualitative error analysis. Predicted labels are taken as `argmax` over `multi:softprob` outputs; the "confidence" used in examples is that maximum class probability.

[1] XGBoost documentation: `https://xgboost.readthedocs.io`. Chen & Guestrin (2016): `https://arxiv.org/abs/1603.02754`.

Results

Cross-Validation (train split): macro-F1 = 0.566 ± 0.057.

Held-out Test: Accuracy = 0.658, macro-F1 = 0.487.

Per-class precision/recall/F1 on the test set (labels map to A,B,C,D,F):

Class	Precision	Recall	F1	Support
A (0)	0.308	0.190	0.235	21.000
B (1)	0.471	0.444	0.457	54.000
C (2)	0.447	0.487	0.466	78.000
D (3)	0.379	0.398	0.388	83.000
F (4)	0.889	0.889	0.889	243.000
Macro avg	0.499	0.482	0.487	479.000
Weighted avg	0.656	0.658	0.656	479.000

Visualizations

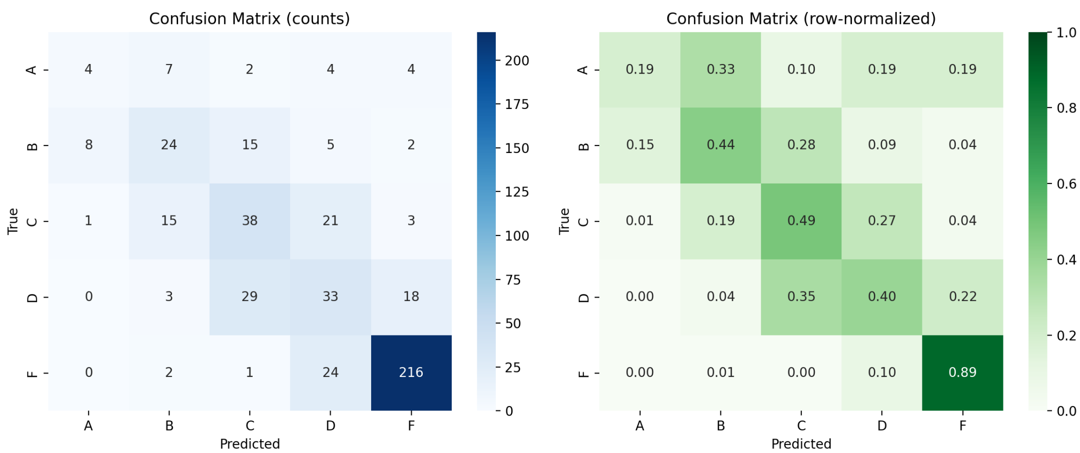

Figure 1: Confusion matrices (counts and row-normalized) for the held-out test set.

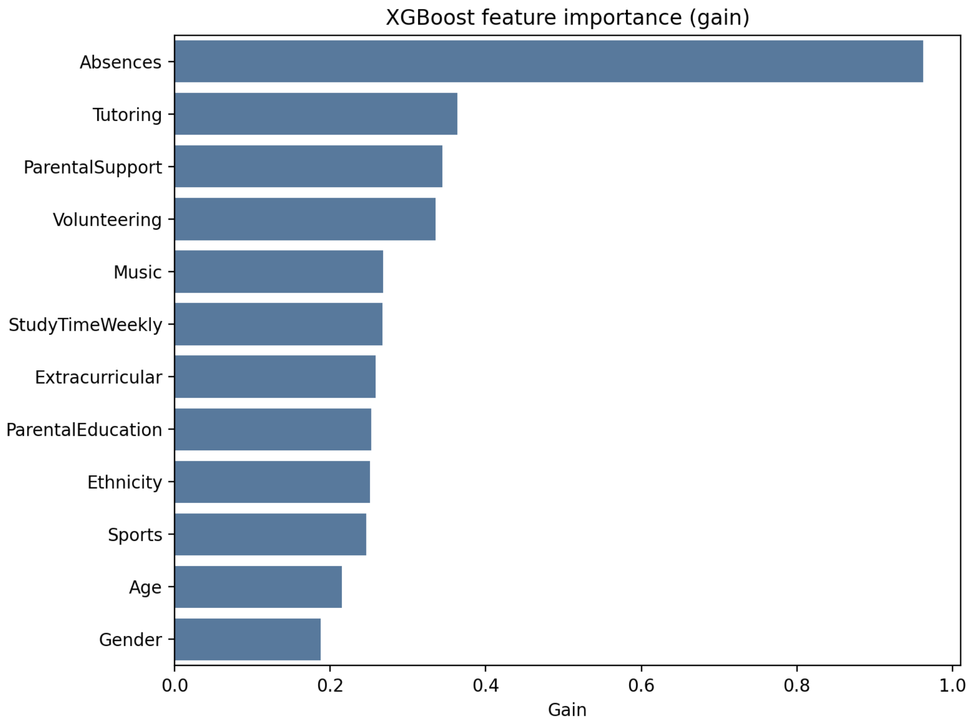

Figure 2: XGBoost feature importance (gain).

Discussion

Overall accuracy is driven by strong performance on the majority class F (precision/recall ≈ 0.89), while performance on minority classes (A/B) is notably weaker—a pattern consistent with the class imbalance seen in EDA. Macro-F1, which weights each class equally, therefore lands below accuracy. The confusion matrices indicate that most errors occur between "neighboring" grades (e.g., A vs. B, C vs. D), suggesting the features capture broad achievement bands but struggle to separate adjacent boundaries.

From an interpretability angle, the feature-importance plot provides directional insight into which signals the model finds most useful. While tree-based gain does not establish causality, it helps prioritize follow-up analysis and potential feature engineering (e.g., interaction or monotonic constraints if justified by domain knowledge). Because outputs are uncalibrated probabilities, users should be cautious interpreting the magnitude of predicted confidence; if calibrated decision support is desired, post-hoc calibration (isotonic or Platt scaling) on a validation set would be appropriate.

Clear, incremental next steps include:

- **Class sensitivity:** incorporate per-class sample weights (e.g., inverse-frequency) to reduce bias toward F; optionally explore focal loss or custom loss.
- **Hyperparameter tuning:** a small search over depth, learning rate, trees, subsampling, and regularization to improve macro-F1 while guarding against overfit.
- **Decision policy:** if the use case prioritizes early risk identification, optimize metrics aligned to that objective (e.g., recall for D/F) or set class-specific thresholds.

- **Calibration:** apply probability calibration if thresholds or risk scores will be acted on by advisors or automated systems.
- **Feature refinement:** engineer richer engagement signals (e.g., attendance trends, interaction terms) if available, and reassess importance and confusion patterns.

Example Predictions (Three Cases)

We include three high-confidence predictions from the test set, aiming to illustrate success and borderline cases. "Index" refers to the row index within the test split (not `StudentID`). "Confidence" is the model's maximum predicted probability for the predicted class. We deliberately select one predicted A, one predicted B, and one predicted F when available.

Index	True	Pred	Confidence
475	B	A	0.999
118	A	B	0.998
464	F	F	1.000

Brief interpretation: the B→A and A→B flips are typical boundary confusions where the available features do not cleanly separate adjacent top grades; despite very high confidence, such cases highlight the need for probability calibration if confidence will drive actions. By contrast, the F→F prediction reflects a pattern the model recognizes reliably on this dataset, aligning with strong test-set precision/recall for F. In practice, pairing these probabilities with decision thresholds tuned to institutional goals (e.g., flagging at-risk students) is recommended.

Reproducibility

Artifacts produced by `data analysis/models/xgb_baseline.py`:

- Figures: `Checkpoint 2/figures/xgb_confusion_mats.png`, `Checkpoint 2/figures/xgb_feature_importanc`
- Summary JSON: `Checkpoint 2/xgb_baseline_summary.json`
- Three cases: `Checkpoint 2/xgb_three_cases.csv`

Run with Python 3.11 virtual environment (Windows PowerShell):

```
.\.venv311\Scripts\python.exe ".\data analysis\models\xgb_baseline.py"
```

References:

- XGBoost Documentation: `https://xgboost.readthedocs.io/en/stable/`
- Chen, T., & Guestrin, C. (2016). XGBoost: A Scalable Tree Boosting System. *arXiv:1603.02754*.