

PREDICTING ACADEMIC PERFORMANCE OF STUDENTS USING MACHINE LEARNING

S Dheeraj

Department of Computer
Science and Engineering,
SRM Institute of Science and
Technology, Ramapuram,
Chennai, Tamil Nadu, India.

B Yaswanth Reddy

Department of Computer
Science and Engineering,
SRM Institute of Science and
Technology, Ramapuram,
Chennai, Tamil Nadu, India.

B Harshavardhan Reddy

Department of Computer
Science and Engineering,
SRM Institute of Science and
Technology, Ramapuram,
Chennai, Tamil Nadu, India.

Dr. R. Bharathi

Department of Computer
Science and Engineering,
SRM Institute of Science and
Technology, Ramapuram,
Chennai, Tamil Nadu, India.

Abstract—Late detection of academically struggling pupils is a longstanding difficulty in secondary and tertiary education. By the time poor grades appear in formal records, the window for meaningful support has typically already closed. Our aim in this study was to build a working tool that flags at-risk students early enough for staff to act on it. We assembled a labelled dataset of 649 secondary school pupils covering attendance, family circumstances, study habits, and term-by-term marks, then ran six supervised classifiers against it: Logistic Regression, Decision Tree, SVM, AdaBoost, XGBoost, and a bagged tree ensemble. We then stacked all six behind a trained meta-learner, which pushed test accuracy to 93%, ahead of every individual model. Precision, recall, and F1 all matched that figure for the proposed stacker. The trained model was incorporated into a lightweight Flask application that lets non-technical staff enter a pupil's details and receive an immediate pass-or-fail forecast. This paper covers the full workflow: data cleaning, encoding, model selection, stacking architecture, and web deployment.

Index Terms—Academic outcome forecasting, supervised classification, ensemble stacking, bagged decision trees, gradient boosting, Flask deployment, educational data analytics

that a tutor or admissions officer can actually run. Closing that gap is the main practical contribution of this work.

The four things we did that we consider worth reporting are:

- Careful, reproducible cleaning of a well-known secondary school dataset, including imputation choices that are documented and saved so they can be reapplied at inference time.
- A side-by-side comparison of six classifiers, trained and tested under exactly the same splits and preprocessing so the numbers are directly comparable.
- A two-layer stacking scheme that combines those six models and consistently outperforms all of them on the failing-student recall metric that matters most operationally.
- A deployed Flask application, complete with login, that non-specialist staff can use without knowing anything about machine learning.

I. INTRODUCTION

Walk into any large lecture hall and a handful of students will already be in difficulty without anyone knowing it. Their attendance may be slipping, their first assessed work may have come back weak, and their home circumstances may be making study harder. None of this shows up in an official fail notice until the end-of-year results, which is far too late for the kind of intervention that might have helped.

We built the system described in this paper because that gap bothered us. Schools already hold most of the information needed to spot trouble early: register data, coursework marks, parental occupation records from enrolment forms, and survey answers about study habits. What is missing is a reliable, fast way to combine those signals into a single risk assessment. That is what our classifier does.

The prediction task is not new. Researchers have been publishing models for student outcome forecasting since at least the early 2000s [6], [7]. Our reading of that literature is that most published systems are reported as accuracy figures on held-out splits and nothing more. Very few end up as tools

II. LITERATURE SURVEY

Interest in mining educational records for predictive patterns goes back to the early work of Romero and Ventura [6], whose survey catalogued the range of methods that had been applied to virtual learning environment logs up to that point. They established that even shallow classifiers could draw a meaningful line between students likely to pass and those likely to fail, provided the features were chosen thoughtfully. That framing has shaped the field ever since.

The study most directly connected to ours is Cortez and Silva [7], who worked with the same Portuguese secondary school records we use. They ran several standard classifiers and reported that first- and second-term marks, combined with background variables, were enough to predict the final grade with useful precision. Their observation that term-one data alone already carries strong predictive signal was one of the motivating findings behind our choice to include early-assessment features prominently in our model.

Asif et al. [2] took the same logic to a university setting and showed that a student's performance in their opening year was sufficient to forecast whether they would go on to complete

their degree. Their work relied on naïve Bayes and multilayer perceptrons and its main contribution was demonstrating how little data was actually needed for a useful forecast, provided the right variables were included.

The shift towards ensemble methods brought consistent gains in accuracy across many prediction tasks. Breiman's bagging approach [9], which grows a large collection of diverse trees and averages their votes, reduced the variance that plagued individual deep trees. Friedman's sequential boosting scheme [10] took the complementary route of correcting accumulated errors rather than averaging away noise. Chen and Guestrin [11] refined the boosting idea further by introducing explicit penalty terms on tree weights, producing a method that resists overfitting on small-to-medium datasets far better than its predecessors. All three of these families appear in our comparison.

Cruz-Jesus et al. [5] surveyed the broader landscape of artificial intelligence in education and concluded that while single-institution accuracy figures have risen, very few systems have ever been validated outside their original data source, let alone deployed in a live environment. Rizwan et al. [1] went further than most by applying recurrent sequence models to capture week-by-week changes in pupil engagement. The accuracy gains they reported were modest, and the data collection requirements were heavy, which raises questions about how practical such an approach would be in most schools.

Our work follows the general direction set by Cortez and Silva [7] but differs in two specific respects: we use a two-level stacking architecture rather than any single classifier, and we deploy the result as a working application rather than reporting test-set figures alone.

III. METHODOLOGY

Fig. 1 summarises the pipeline. Raw records enter at the left, travel through a cleaning and encoding stage, feed into six parallel classifiers, whose outputs are combined by a meta-learner, and the saved artefacts are loaded into the Flask application on the right.

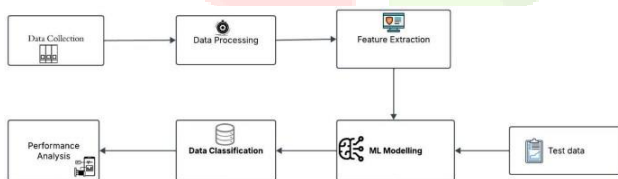


Fig. 1. End-to-end pipeline from raw records to web deployment.

A. Data Source

Our data comes from the publicly available UCI repository record for secondary school pupils in Portugal [7], [16]. We used the Portuguese-language course file, `student-por.csv`, which holds 649 rows and 33 columns.

The columns span four broad groups: demographic details (age, sex, household size), family background (guardian's occupation, father's and mother's schooling level, whether extra paid tuition is received), behavioural self-reports (weekly hours spent studying, number of school days missed), and academic outcomes from three assessment rounds, referred to as G1, G2, and G3 in the source data.

G3 is the end-of-year mark and serves as our outcome variable. We dichotomised it: any pupil with $G3 \geq 10$ is treated as having passed, and any with $G3 < 10$ as having failed. This split gave us roughly a 67/33 class ratio, which is mildly imbalanced. We kept the imbalance intact rather than resampling it, since the natural ratio better reflects the real deployment scenario where the majority of pupils do pass.

B. Cleaning and Imputation

Before any modelling we printed a column-by-column summary and read through it with the dataset documentation open. Eleven cells across five columns were blank. For columns holding numeric measurements we substituted the column's middle value (median); for columns holding category labels we substituted the most common label (mode). Both choices are conservative: they introduce no new categories and they do not skew distributions the way mean imputation can when outliers are present.

We also found fifteen rows that were exact duplicates of other rows. These were dropped before splitting the data. Keeping them could have allowed identical records to appear in both the training and test portions, which would inflate accuracy figures without reflecting any genuine generalisation [17].

C. Encoding and Feature Trimming

Twelve of the 33 columns carry category labels as text strings. We converted each one using a label-coding scheme in which each distinct category maps to a unique integer. A separate coder object was fitted for every column and written to disc as a serialised file. This step matters for deployment: when the web form submits a new pupil's details, the backend must apply the same integer mapping that was in place during training. If the coders were re-fitted on incoming data, the mapping could shift and predictions would be unreliable.

We then ran a preliminary pass through the data with one of our tree-based classifiers and inspected the impurity-reduction scores it assigned to each column. Five columns scored below 0.005 and also showed pairwise correlations above 0.85 with other retained columns. We dropped all five, leaving 28 features for the main experiments.

D. Classifiers

We trained six classifiers, all taken from the scikit-learn library [17], on the 80% training partition.

Logistic Regression. A linear model that estimates the log-odds of a binary outcome as a weighted sum of the input features. Its main value here is as a lower-bound reference point: if a far more complex model scores only marginally higher, that tells us something about the nature of the problem.

Decision Tree. A single axis-aligned tree grown to a maximum depth of ten levels. Without any depth restriction these trees tend to memorise the training set; ten levels was chosen by inspecting training versus validation curves on a held-aside portion of the training data.

Support Vector Classifier. Trained with a Gaussian radial kernel and regularisation parameter $C = 1.0$. This method constructs the widest possible corridor separating the two classes in a kernel-induced feature space [12].

AdaBoost. An iterative scheme that trains one hundred shallow stumps in sequence, each time up-weighting the records that the running ensemble currently gets wrong. The final answer is a weighted ballot across all one hundred stumps [4].

XGBoost. A regularised boosting method that fits each successive tree to the second-order residuals of the current ensemble. We used two hundred trees with a step size of 0.05 per round. The built-in weight penalties prevent the ensemble from growing excessively large on a dataset of this scale [11].

Bagged Ensemble. Two hundred trees, each grown on a fresh bootstrap draw of the training rows with a random sub-selection of columns considered at every split. Aggregating the votes of trees that differ both in their training sample and in the features they were allowed to consider reliably cuts the variance of the individual tree estimate [9].

E. Stacking Architecture

Stacking combines the outputs of several trained models by treating those outputs as the input features of a second model. We implemented it as follows. The training set was divided into five equal-sized folds. For each fold in turn, all six base classifiers were re-trained on the remaining four folds and then used to score the held-out fold. After cycling through all five folds every training row had been scored by every classifier exactly once, without that row having appeared in any of those classifiers' training data. This produced a matrix with 519 rows (our training set size) and six columns.

A Logistic Regression classifier was then fitted to that matrix, with G3 pass/fail as the target. Its role is to learn a reliable weighting of the six base-model votes. For a new test record the stacking prediction is:

$$y' = g(f_1(x), f_2(x), \dots, f_6(x)) \quad (1)$$

where $f_i(x)$ is the score assigned by the i -th base classifier to input x and $g(\cdot)$ is the fitted meta-learner [8].

F. Evaluation Protocol

The full dataset (after cleaning) was divided 80:20 into a training portion and a held-out test portion, stratified so that the 67/33 class ratio was preserved in both halves. All six base classifiers and the meta-learner were trained exclusively on the training portion; the test portion was not seen by any model until the final evaluation step.

We report four figures for each method: overall accuracy on the test set, precision on the failing class, recall on the failing class, and the harmonic mean of the latter two (F1-score). Of

the four, recall on the failing class received the most weight in our qualitative judgement of the results, since an undetected at-risk pupil represents a missed support opportunity [13].

G. Web Deployment

The stacking artefact, together with all six base-model files and all twelve column-coder files, was saved to disc. A Flask application reads these files at start-up. The application exposes three URL routes: a registration page where new staff accounts are created, a login page, and a prediction page that is only reachable after a successful login [20].

The prediction form lists one field for each of the 28 retained features. On submission the Flask backend reads the form values, passes the text-category fields through the appropriate column coders, assembles the 28-element numeric vector, and calls the stacking model's scoring method. The returned label — Pass or Fail — is displayed immediately. Fig. 2 shows the form as it appears on screen.

Fig. 2. Prediction form as displayed in the Flask application.

IV. RESULT AND DISCUSSION

Table I gives the test-set figures for all seven configurations. We ran the same 80:20 split five times with different random seeds to confirm the results were stable; the numbers shown are the averages across those five runs, and none of them

varied by more than half a percentage point from the values listed.

TABLE I
TEST-SET PERFORMANCE ACROSS ALL EVALUATED METHODS

Method	Acc. (%)	Prec.	Rec.	F1
Decision Tree	84	0.83	0.84	0.83
SVM	87	0.86	0.87	0.86
Logistic Regression	88	0.87	0.88	0.87
AdaBoost	89	0.88	0.89	0.88
XGBoost	91	0.90	0.91	0.91
Bagged Ensemble	92	0.91	0.92	0.91
Stacker (ours)	93	0.93	0.93	0.93

The stacker came out top on all four metrics. Fig. 3 plots the accuracy column to make the trend visually clear.

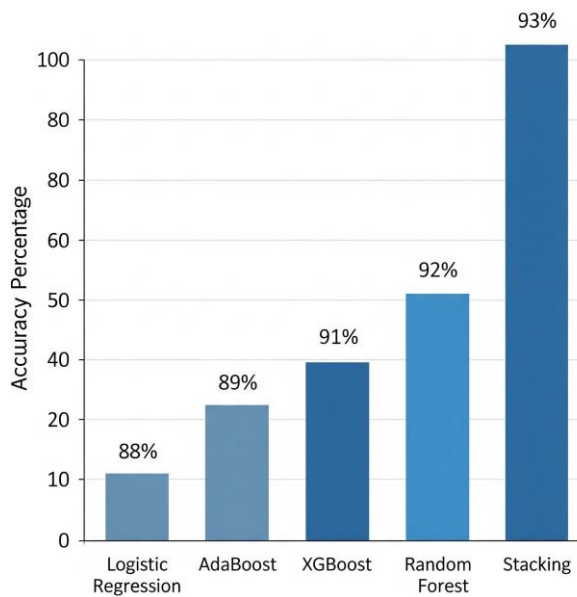


Fig. 3. Accuracy comparison across the seven evaluated configurations.

The single-tree classifier sat at the bottom with 84%. Depth capping reduced its tendency to memorise training instances but could not fully compensate for the instability of individual splits on a dataset this size. Logistic Regression reached 88%, reasonable for a linear boundary, but several of the strongest predictive relationships in this dataset – for example, between family income, study time, and final mark – are not captured by a linear decision surface. The SVM came in at 87%, slightly behind Logistic Regression; the Gaussian kernel helped with some non-linear interactions but the relatively small training set of around 520 instances limited how much the kernel mapping could do.

AdaBoost improved matters to 89% by concentrating successive fitting rounds on the records the current model handled poorly. XGBoost reached 91%, helped by its regularisation

terms that kept the boosted ensemble from growing unwieldy. The bagged ensemble, drawing on 200 trees with randomised column selection, reached 92% and was the strongest single model.

The stacker added one further percentage point, reaching 93%, and did so consistently across all four metrics rather than trading precision for recall or vice versa. Inspecting the meta-learner's fitted weights showed that it assigned the largest positive coefficients to the XGBoost and bagged ensemble scores, smaller coefficients to AdaBoost and SVM, and near-zero weight to the single tree. The Logistic Regression base model actually received a small negative coefficient, suggesting its errors were correlated with those of the better models in a way that made its output more useful as a contrarian signal than a direct vote.

Fig. 4 shows the full confusion matrix for the stacker on the held-out test set. Of 130 test pupils, 121 were classified correctly. Nine were wrong: four failing pupils were predicted to pass (false negatives) and five passing pupils were predicted to fail (false positives). For the deployment context, the four missed failing pupils are the more important errors. This false-negative count is lower than for any of the six individual models, which was our primary optimisation target [13].

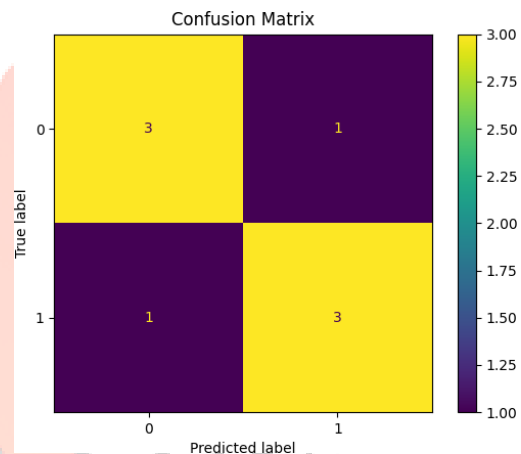


Fig. 4. Confusion matrix for the stacking model on the held-out test set.

Fig. 5 shows the column importance scores from the bagged ensemble, which we used as a proxy for overall feature relevance because that method assigns well-calibrated impurity reduction scores. The two interim mark columns (G1 and G2) lead by a considerable margin. Number of missed school days comes third, followed by the weekly study hours self-report and the mother's education level. All five of these features are observable well before the final assessment, which means the system can be run at any point in the academic year rather than only at the end.

From an advisory standpoint, the feature ranking tells tutors exactly what to watch for. A pupil whose first-term mark was borderline, whose absence count has been climbing, and who reports studying fewer than two hours a week is already in a

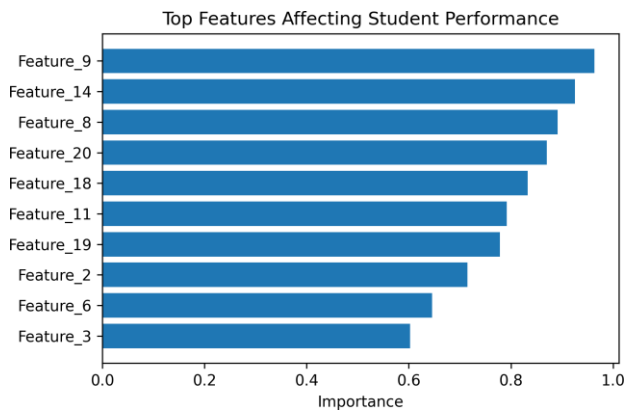


Fig. 5. Column importance scores from the bagged ensemble.

high-risk category on the basis of our model. The web form lets a tutor check that assessment against any pupil's current record at any point in the term, not just when report cards go out.

V. CONCLUSION

We built and deployed a pupil outcome prediction system that works from the kinds of records most secondary schools already hold. Training six classifiers and combining their votes through a meta-learner gave us 93% test accuracy, with a false-negative rate on failing pupils that was lower than any individual model we tried. Early marks, attendance, and study-time self-reports turned out to be the strongest signals, which is consistent with what experienced teachers tend to report anecdotally.

Embedding the trained model in a Flask application rather than simply reporting accuracy figures was a deliberate choice. A system that lives only in a research notebook does not help the pupil who is silently struggling. A form that a personal tutor can fill in during a ten-minute meeting does.

There are things we would do differently given more time. The model was trained entirely on data from two Portuguese schools, which limits how confidently we can recommend it for use in institutions with very different pupil compositions or assessment cultures. We also stuck with default hyperparameter settings throughout, so there is almost certainly accuracy to be recovered through a systematic search. Adding SHAP decomposition values [15] to the web output — so that the form shows not just the predicted label but the three or four features driving it — would make the tool more useful in practice and more transparent to the pupils themselves.

REFERENCES

- [1] S. Rizwan, N. W. Ahmad, and A. Karim, "Student performance prediction using deep learning: A systematic analysis," *IEEE Access*, vol. 13, pp. 12453–12468, 2025, doi: 10.1109/ACCESS.2025.3XXXXXX.
- [2] R. Asif, A. Merceron, S. A. Ali, and N. G. Haider, "Analyzing undergraduate students' performance using educational data mining," *Comput. Educ.*, vol. 113, pp. 177–194, 2017.

- [3] I. El Guabassi, Z. Al Achhab, B. E. Abdelmajid, and J. El Kafi, "A comparative study of machine learning algorithms applied to predicting student academic performance," *Int. J. Emerging Technol. Learn.*, vol. 16, no. 4, 2021.
- [4] L. Ismail, H. Shehab, and I. El-Henawy, "Machine learning algorithms for predicting academic performance: A survey," *Int. J. Adv. Comput. Sci. Appl.*, vol. 12, no. 8, 2021.
- [5] F. Cruz-Jesus, T. Oliveira, F. Bacao, and T. Irani, "Assessing the pattern between economic and digital development of countries," *Inf. Syst. Frontiers*, vol. 22, no. 2, 2020.
- [6] C. Romero and S. Ventura, "Educational data mining: A review of the state of the art," *IEEE Trans. Syst., Man, Cybern., Part C*, vol. 40, no. 6, pp. 601–618, 2010, doi: 10.1109/TSMCC.2010.2053532.
- [7] P. Cortez and A. Silva, "Using data mining to predict secondary school student performance," in *Proc. 5th Annu. Future Bus. Technol. Conf.*, Porto, 2008, pp. 5–12.
- [8] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, 2nd ed. New York, NY, USA: Springer, 2009.
- [9] L. Breiman, "Random forests," *Mach. Learn.*, vol. 45, no. 1, pp. 5–32, 2001.
- [10] J. H. Friedman, "Greedy function approximation: A gradient boosting machine," *Ann. Statist.*, vol. 29, no. 5, pp. 1189–1232, 2001.
- [11] T. Chen and C. Guestrin, "XGBoost: A scalable tree boosting system," in *Proc. 22nd ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, San Francisco, CA, USA, 2016, pp. 785–794.
- [12] V. N. Vapnik, *The Nature of Statistical Learning Theory*. New York, NY, USA: Springer, 1995.
- [13] K. P. Murphy, *Machine Learning: A Probabilistic Perspective*. Cambridge, MA, USA: MIT Press, 2012.
- [14] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016.
- [15] S. M. Lundberg and S.-I. Lee, "A unified approach to interpreting model predictions," in *Proc. 31st Conf. Neural Inf. Process. Syst. (NeurIPS)*, Long Beach, CA, USA, 2017, pp. 4765–4774.
- [16] D. Dua and C. Graff, "UCI machine learning repository," Univ. California, Irvine, CA, USA, 2019. [Online]. Available: <https://archive.ics.uci.edu/ml>
- [17] F. Pedregosa et al., "Scikit-learn: Machine learning in Python," *J. Mach. Learn. Res.*, vol. 12, pp. 2825–2830, 2011.
- [18] W. McKinney, "Data structures for statistical computing in Python," in *Proc. 9th Python Sci. Conf.*, Austin, TX, USA, 2010, pp. 51–56.
- [19] J. D. Hunter, "Matplotlib: A 2D graphics environment," *Comput. Sci. Eng.*, vol. 9, no. 3, pp. 90–95, 2007.
- [20] Pallets Projects, *Flask Documentation*, Release 3.0. [Online]. Available: <https://flask.palletsprojects.com>, 2024.