Predicting Student Performance with Machine Learning

Index Terms—K-Nearest Neighbours, Logistic Regression

1 Introduction and Summary

EDUCATION is a key part of society. A high education level of a country is well-known to indicate lower crime, faster economic growth, and greater civic engagement. It is also essential to individual lives, for both personal and economic reasons. Despite this, many societies grapple with high student drop-out and failure rates. One reason for this is a lack of resources and support for struggling students.

As such, it is vital that educators use these resources optimally, typically targeting the students needing the most help first. In this way, machine learning (ML) could be used to predict student performance. A typical method of doing this is using previous grades, but it would be much more useful to be able to predict based on personal information, since exam results may be unreliable. This gives us an insight into what factors have the greatest effect on student success, so limited resources can be targeted accordingly.

The dataset [1] contains information about students studying Portuguese Language from two Portuguese schools. It contains personal information as well as the previous two exam grades *G1* and *G2*, and the end-of-year grade *G3* where a score of 10 is required to pass [2]. This presents a binary classification task: using personal information as well as previous years' grades, predict whether a student will pass or fail their final year exam. Formally: given 32 input features and an unseen instance, predict whether it falls into category 0 (fail) or category 1 (pass). Using a k-nearest neighbours (KNN) model as a baseline, I propose using a logistic regression (LR) model for this task.

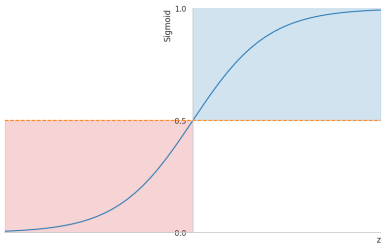

Fig. 1. Logistic regression model

LR uses the logit function, $logit(p) = log(\frac{1}{1-p})$, to predict the odds of a given instance being in class 0 or 1 based on the input features, meaning the parameters reflect how influential each feature is on the likelihood of a given instance being in class 1. The parameters are applied to the input features to give logit(p) where p is probability of being in class 1. The hypothesis function represents p and is given by the sigmoid function $\sigma = \frac{1}{1+\exp(-\theta^T X)}$. This function outputs the probability that an instance is in class 1. It is predicted to be in class 1 if it is more likely than a certain threshold (0.5 is used in this paper) that it is, and class 0 otherwise. This process of selecting parameters is called maximum likelihood estimation (MLE).

Fig. 1 shows the sigmoid function with probabilites from the model on the x-axis, where probabilities are classified as 1 within the blue-shaded area and 0 in the red-shaded area.

In summary, I show that LR performs better than KNN for this task, although using the right preprocessing and hyperparameters makes KNN perform almost as well. LR avoids overfitting where KNN does not, and so the LR model is more generalised. This is important for larger datasets that are less imbalanced. Overall, the dataset size and imbalance were the most limiting factors on the performance, so these must be tackled for future improvements.

2 Data and Experimental Setup

The dataset is small (649 instances, 32 features), with none missing. It was constructed using paper school reports and student questionnaires since a lot of school data is not digitised. The intensive construction partly explains the dataset's small size. The original paper [2] has more information on how this dataset was constructed as well as a full table of the dataset features.

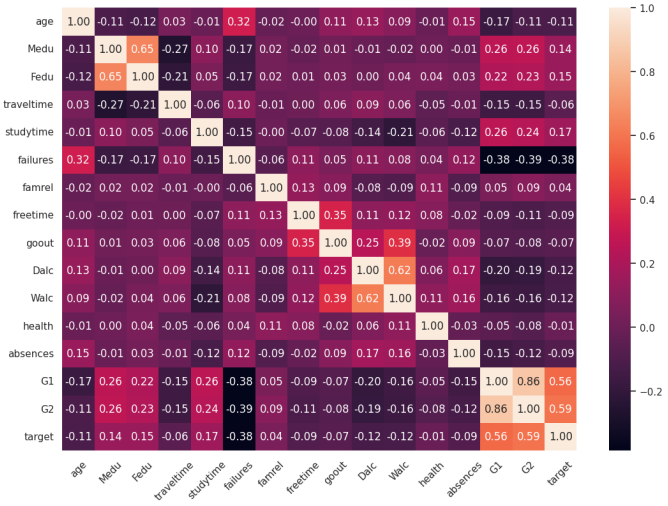

Fig. 2. Heatmap

Fig. 2 is a heatmap of the numeric dataset features. The darker areas represent negative correlation between features, purple means little to no correlation and lighter areas show positive correlation. Most of the heatmap is purple, meaning there is almost no correlation between most features which makes the models predictions more reliable [3]. Noticeably, a few are significantly positively correlated. *'Walc'* and *'Dalc'* are understandably strongly correlated since they are respectively weekend and weekday alcohol consumption. *'Fedu'* and *'Medu'* (Father/Mother's education) are also correlated, perhaps because they met during education or working in a job that required the same education level. Looking in the bottom-right corner we see *G1*, *G2* and the target are all strongly positively correlated since they are the three test grades from the school year. However, this suggests that *G1*, *G2* will be very useful to predict the target. So, the strongly correlated features are explainable and reflective of the real world which makes this dataset useful for ML.

The dataset is very imbalanced, with 85% passes and 15% fails. To counteract this, I attempted using the SMOTE technique [4] to fully rebalance the data, but it made the classifiers less accurate and useful since an equal number of passes and fails is unrealistic. Instead, I opted for class reweighting during the model training phase covered in Section 3.

The data is mostly linearly separable, meaning most of the data can be separated by a single hyperplane. I trained a linear support vector machine (SVM) on the data with a very small $\frac{1}{2^{15}}$ misclassification penalty which forces the optimiser to make close to zero error in classification to minimise the loss function, overfitting the data [5]. Because this model is a hyperplane separating the data into two, a very small error means most of the data is linearly separable. The SVM separated the data with 95% accuracy, so it was mostly linearly separable.

2.1 Dataset Split

The test set contained 20% of the data (130 instances), with the rest kept for training and validation (519 instances). There is a balance between having a large enough training set for high-quality predictions and having a large enough test set to get meaningful test feedback and evaluation. It is advised to set aside 20-30% of data for testing [6].This is evidently specific to the dataset, and since it is very small I chose 20% for testing because it strikes this balance. Since it is also imbalanced, a stratified split ensures both sets had the same proportions of pass/fails. Otherwise, the training set may contain only passes and the test set only fails, creating useless predictions.

Fig. 3 implies the differences between the two sets. Each column is a different set, and each row is either the distribution of feature means or standard deviations. Each point on the x-axis is a feature, but to keep the graphs simple I have removed the labels. Both sets have similar but not the same shapes for both means and standard deviations, meaning the trained models should be able to predict the test data but cannot get away with overfitting on the training data. Ideally, this reflects the deviations between the training data and the real world.

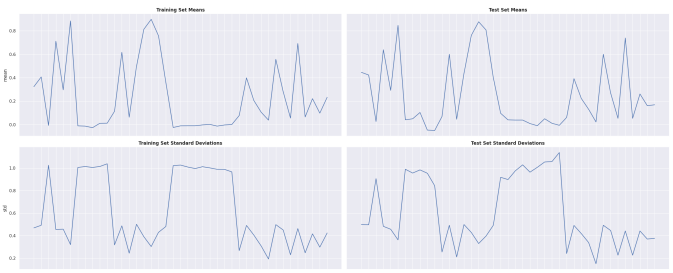

Fig. 3. Distribution of means and standard deviations across test and training sets

2.2 Data Transformation

For binary features like responses to yes/no questions, I transformed the labels to 0, 1. For nominal features, such as parent occupation ('teacher', 'healthcare related', 'civil services', 'at home' or 'other'), I tried encoding each label from 1-5, but the models appeared to learn an arbitrary order such that 'civil services', encoded to 3, ranked higher than 'teacher' encoded to 1. For this reason, I opted for dummy encoding instead and created a new feature column for each category. To avoid multicollinearity between these columns, I dropped one column per feature [3]. This means that for a given nominal feature, n categories became n-1 columns where the dropped category was represented by 0 in all the other category columns.

I standardised the numeric features, including ordinal ones. Although LR is not as sensitive to the feature magnitude, KNN very much is and so this step is important for ensuring KNN provides a meaningful baseline.

Finally, I applied principal component analysis (PCA) to the training data. The dataset now had about 40 features, which is a red flag for the 'curse of dimensionality' since a lot of data would be required to generalise accurately in that many dimensions. This dataset is too small for that so PCA reveals how much data variance each dimension explains, and so which features can be removed while maintaining most of the data variance. This is especially important for KNN and LR which suffer from overfitting due to the 'curse'. After using PCA on just the training set, the same dimensionality reduction was applied to the test data which avoids leakage and unhelpful predictions.

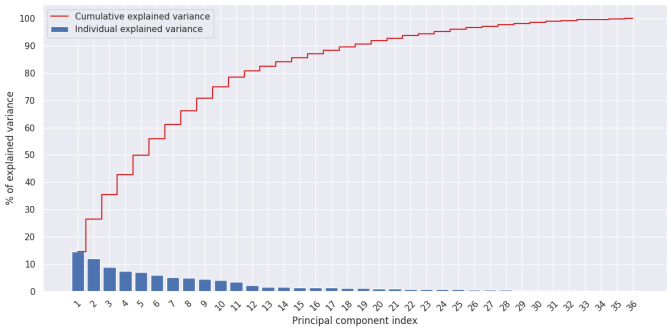

Fig. 4. Cumulative explained variance plot

In Fig. 4 the blue bars represent how much variance each component explains. The red line is the cumulative expected variance, showing that 90% of the variance is explained

by about half the number of dimensions and almost 80% by just 13. I chose to keep the first 16, keeping 85% of the variance, because it more than halved the number of features and maintaining a high percentage of the original variance makes the ML models more likely to generalise. Cross-validation could be useful for this choice in future.

2.3 Evaluation Metrics

Accuracy is a common ML evaluation metric, but since the dataset is very imbalanced, accuracy could be very misleading. If the model always predicts 1 (pass), the accuracy would be high since most passed but it would not be useful. Instead, a confusion matrix is better to evaluate each model since it shows how effective they are at correctly predicting passes and fails individually. In a context of limited resources false positives may be worse than false negatives, and the confusion matrix is an accessible visual guide that educators can use whatever the context, since it is not restricted to a single number.

However, a single number to score each model is very useful for comparison, so I use both AUC and F1-score. AUC is the area under the ROC curve, created when plotting recall, $\frac{true\ positives\ (TP)}{true\ positives\ +\ false\ negatives\ (FN)}$, on the y-axis against selectivity, $\frac{true\ negatives\ (TN)}{true\ negatives\ +\ false\ positives\ (FP)}$, on the x-axis. Note that the positive label is 1 (pass). This shows the correct positive predictions as a proportion of all positive predictions. A perfect classifier has a recall = 1 and selectivity = 0, so the ROC is in the top-left corner of the graph and the AUC = 1. The worst classifier would have recall = selectivity (ROC is y = x, AUC = 0.5) because for every correct positive prediction there is one incorrect - its predictions are statistically as good as a coin flip. In this way, an AUC closer to 1 means the classifier is better. When resources are limited, it is valuable to ensure as many of the positive predictions as possible are correct, so they are not wasted. In this context, AUC is a useful metric.

Otherwise, it might be more important to ensure everyone who needs support is found, even if the model overestimates the number needing support. In this case false negatives are more of a concern than false positives, so F1-score is a useful measure. It is the harmonic mean between recall and precision, where precision is the proportion of positive predictions that were correct and recall is the proportion of actual positives correctly predicted. This gives F1-score as

$$2\,\frac{Precision \times Recall}{Precision + Recall}$$

3 Model Evaluation

I propose LR for this task, using KNN as a baseline. Both are applicable to binary classification tasks. KNN is a good baseline since it is easy to understand, commonly used, and typically less intricate than LR though that depends on how hyperparameters are tuned. Data exploration showed that most data is linearly separable making LR a good candidate. Other suitable models are SVMs and decision trees, and a naïve Bayes model would also be an interesting baseline, but the dataset's original paper [2] used these but not KNN or LR, so these present a different approach to the dataset and a greater contribution to the literature.

LR is potentially superior to KNN for this dataset. Since it is very imbalanced, overfitting is a serious concern. KNN is much more vulnerable to this than LR since it has no regularization option. For LR there are multiple, such as L1 or L2 norms and reweighting. In this way, LR is more flexible than KNN. KNN is also more sensitive to outliers or mislabelled data since it is distance-based, whereas LR uses MLE for predictions. This is important since student performance could understandably be prone to outliers (e.g., a good student failing their exams because a relative recently passed away).

3.1 Training Process

The models were trained and validated with 20 runs of 10-fold stratified cross-validation [2], where the training data is randomly divided into 10 equally sized subsets, each with roughly equal proportions of the target variables (pass and fail) since the dataset is imbalanced. The small training set size increases risk of the model overfitting and failing to generalise, despite data pre-processing to decrease it (such as PCA). K-fold cross-validation is very efficient because it reuses data and removes the need for a validation set, which would reduce the training set even further. Ridge regression was used for the LR model to reduce the chance of overfitting, in which a shrinkage penalty $\lambda \sum \theta_i$, where θ_i are model parameters and λ is a constant, is included in the cost function during training.

The target imbalance in the training data also increases risk of overfitting, so the misclassification costs for LR were reweighted using Sci-kit Learn's 'balanced' heuristic: $\frac{total\ samples}{2 \times number\ of\ class\ instances}$. This resulted in misclassifying a fail being about five times more costly than misclassifying a pass (actual weights are fail: 3.2, pass: 0.6).

The parameters of the LR model fit without much difficulty. Most were between -1 and 1, but two were much larger at -2.85 and 2.01 which suggests at least two input features were much better predictors of the final exam than any others. These two are likely the previous grades as initially suggested by Fig. 2.

To ensure the LR optimiser converged, the maximum number of iterations for estimating the model was raised from 100 to 1000.

3.2 Hyperparameter Selection and Tuning

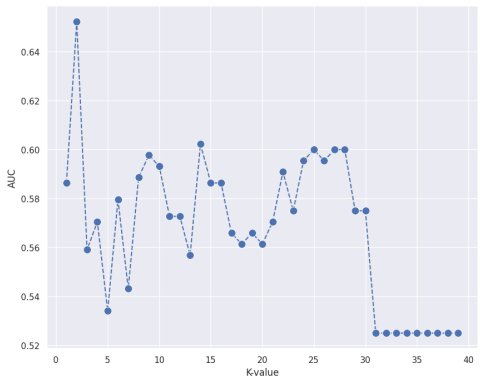

Fig. 5. AUC varying with value of k

The tuned hyperparameters were k and the distance metric (Euclidean or Manhattan) for KNN, and the type and strength of regularisation for LR (L1, L2 or none). These greatly affect model performance as shown by Fig 5, so an exhaustive grid search was used to tune them and select the best estimators. The dataset's small size was actually advantageous in this case, since a grid search is computationally expensive because it enumerates over all possible hyperparameter combinations.

The best estimators were chosen using a custom refit strategy [7]. Using the cross-validation described previously, the models with an AUC less than 0.8 were filtered out and those remaining were ranked by F1-score. Out of these, models greater than one standard deviation away from the best by F1-score were discarded. Finally, the model with the fastest prediction time is selected. In this way, hyperparameters that create high-AUC and F1-scoring models are found, giving the following models:

- KNN: k = 10 with Euclidean distance
- LR: L2 regularisation (ridge regression) with rate $\lambda = \frac{1}{1000}$

3.3 Model Comparison

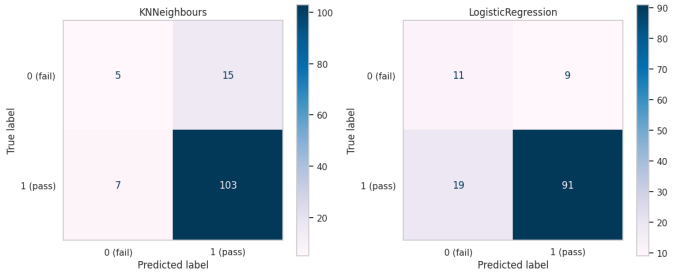

Fig. 6. Confusion matrices for KNN and LR

Fig. 6 shows the predictive results for both models. KNN had more total correct predictions, but it almost entirely predicted passes which suggests the model has overfit the training data. LR performed better at predicting fails, which suggests it has more successfully avoided overfitting.

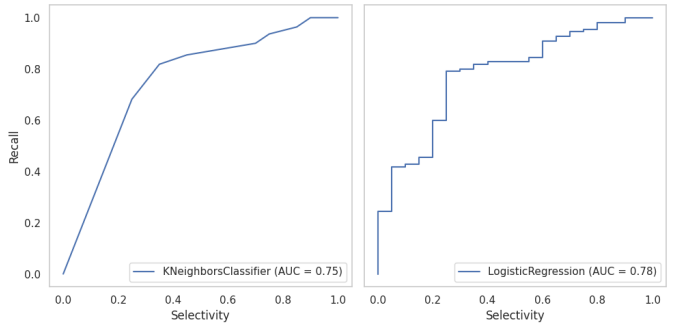

Fig. 7. ROC curves for KNN and LR

Fig. 7 shows that LR had a better AUC, implying it would be more useful under limited resources as previously discussed.

The F1-score was 0.81 for KNN and 0.80 for LR, suggesting that with the correct preprocessing and tuning KNN can perform about as well as KNN on this dataset. However, since LR was more resistant to overfitting it is more generalisable and so more useful.

4 Self-Evaluation Report

What have you learned from the lectures? Besides how to build a good ML model, I've most importantly learned how they fundamentally work so that I can use my own knowledge of maths to optimise and customise them, without always having to use library documentation. I also learned how many problems can be framed as ML tasks, and how much opportunity there is to contribute within the field.

What have you learned from the coursework? Collecting useful and dense data is hard, especially in the quantities required for a good model, so I learned techniques such as dimensionality reduction and cross validation are vital. I learned the importance of avoiding test data leakage. I was getting fantastic predictions and then I realised I was doing PCA on the whole dataset including the test data. After I remedied this, I got more realistic predictions.

What are your difficulties in the module? The hardest by far but perhaps most rewarding thing was attempting to design ML models without Sci-kit Learn. More generally, I found the maths (gradient descent, cost functions, etc.) most difficult but fascinating, so I am glad we got to investigate the underlying mathematics and reason behind the models.

What would you do differently if you were to do it again?

- Use cross validation to decide number of components to remove after PCA.
- Repeat without *G1* and *G2* to observe their effect on the predictions.
- Tune more hyperparameters - a lot more possibilities than I realised!
- Experiment using SVMs, since they are very powerful, but we did not cover them in lectures.

Are there unique contributions or novel ideas that make your project different from existing machine learning work? I used KNN and LR models which the dataset's original paper [2] did not. I also used PCA and reweighting which it did not.

References

[1] P. Cortez, "Student performance," UCI Machine Learning Repository, 2014, [Online]. DOI: https://doi.org/10.24432/C5TG7T.

[2] P. Cortez and A. M. G. Silva, "Using data mining to predict secondary school student performance," 2008.

[3] T. Kyriazos and M. Poga, "Dealing with multicollinearity in factor analysis: The problem, detections, and solutions," *Open Journal of Statistics*, vol. 13, 2023.

[4] Y. Zoralioğlu, M. F. Gül, F. Azizoğlu, G. Azizoglu, and A. N. Toprak, "Predicting academic performance of students using machine learning techniques," *2023 Innovations in Intelligent Systems and Applications Conference (ASYU)*, pp. 1–6, 2023.

[5] M. Mazuecos, "How to check for linear separability," https://maurygreen.medium.com, accessed: Dec 30, 2023.

[6] V. Gholamy, Afshin; Kreinovich and O. Kosheleva, "Why 70/30 or 80/20 relation between training and testing sets: A pedagogical explanation," *Departmental Technical Reports (CS)*, 2018.

[7] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.