Predicting Student Academic Success in Universities
Data Analysis and Machine Learning
Group 14

Tomoki Nakane (5507642)
Guillermo Gil de Avalle Bellido (5787084)
Elisabeta Veliaj (5792010)
Arnau Domenech Romero (5786282)

October 12, 2024

Abstract

University student success and failure in higher education are critical issues, given their implications for individual futures and societal development. Utilizing data from the Polytechnic Institute of Portalegre, spanning from 2008 to 2019, this paper employs different predictive models utilizing academic, demographic, and socioeconomic variables, to predict students into three academic success risk levels. We predict students' success using decision trees and random forests, while mitigating imbalance in the data using undersampling and oversampling methods. Decision tree with bagging using oversampled data resulted in the best predictions, based on metrics such as the recall rate and F1 score. Additionally, the analysis reveals factors such as the number of courses passed, grades and paying tuition fees on time, in predicting student success. These insights suggest that specific policies could be drafted, implemented and assessed to improve educational attainments and reduce dropout rates.

1 Introduction

Ensuring successful completion of higher education is an important task for universities. However, a non-insignificant number of post-secondary students, averaging 9.5% across the EU, still abandon their programs before graduation (Eurostat, 2024). This rate contrasts with counterparts in similar countries like the US, at 6.5% (U.S. Department of Education, 2023), and Japan, with only 1.3% (Ministry of Education, Culture, Sports, Science and Technology, 2023). For this reason, the EU has set a target to reduce the share of "early leavers" across their universities to less than 9% by 2030 (Council of the European Union, 2021).

Machine Learning (ML) techniques can be leveraged to help with this task, by assisting in the prediction of which individuals are more at risk of dropping out of their studies. Such predictions could enable educators to tailor their programs and provide targeted support to these students, consequently decreasing the likelihood of academic churn.

In this regard, previous literature has already attempted similar predictions of academic success. A paper by Miguéis et al. (2018) utilized first-year grades from students at 'The European Engineering School' to predict academic success, aiming to provide universities with a leading resources' indicator. Moreover, another study by Martins et al. (2021) also adopted a similar approach to predict academic success, although it only considered pre-university indicators. In this case, the objective was to establish the likelihood of student support before enrolment to already implement targeted support at the start of their university journey. Other studies, such as those by Hoffait and Schyns (2017), Thammasiri et al. (2014), and Beaulac and Rosenthal (2019), have pursued similar approaches, focusing on either pre- or post-admission data. However, few papers have integrated both approaches, combining data collected before and after enrolment.

Therefore, this paper will analyse a sample of former undergraduates ($N = 4424$) who attended the Polytechnic Institute of Portalegre (Portugal) between the academic years 2008/09 and 2018/19, using indicators from before and after their enrolment. Supervised machine learning methods, such as Decision Trees, Bagged Decision Trees, and Random Forests, will be employed to predict three different possible outcomes related to academic churn, namely, students graduating on time, graduating with a delay, or dropping out. These classification methods are considered well-suited for this analysis due to their robustness to outliers and their ability to rank feature importance, which is particularly crucial given the large number of parameters involved (i.e., 240 features after dummy coding the categorical variables) and the high volatility of some of them, such as family financial situations.

Our results indicate that the studied methods predict with a very similar level of accuracy, about approximately 60-80%, with higher accuracy encountered in the Random Forests model. However, given our focus on preventing false negatives, which translate into preventing misclassifying some students as graduating on time, where they are in risk of dropping

out or graduating with a long delay. Therefore, we aim at reducing the F1 score and recall rates, which brings us to favour decision trees with bagging using oversampled data.

From another perspective, all the models prioritize the consideration of some features for their explanatory power, given that some of them are consistently selected as the most predictive across models. This signals the existence of dominant predictors, particularly in post-enrollment features such as academic grades and amount of courses approved. Overall, these results establish a basic framework for the early detection of students at risk of academic churn. With the appropriate use of these prediction models and on what predictors they should focus, EU educators and policymakers can better act in advance to prevent potential dropouts, which could have a high indirect contribution towards the EU's objectives.

The structure of this paper is as follows. In Section 2, we describe the data and in Section 3 we explain how we perform data rebalancing and present the methodology for prediction and cross-validation. Section 4 contains the results of the analysis, and Section 5 concludes.

2 Data Description

This paper considers the dataset from Realinho and Baptista (2021) which coincides with that used by Martins et al. (2021). This data involves a cross-sectional sample of former undergraduates from the Polytechnic Institute of Portalegre (Portugal), enrolled between the academic years 2008/09 and 2018/19 and across different study programs. The data enables us to further expand Martins et al. (2021) by considering both pre-enrolment and post-enrollment data to predict academic success. Table 1 shows the sample summary statistics.

As a dependent variable, we consider three categories:

- **Success**: Individuals who have graduated on time (considered as low risk).
- **Relative Success**: Students who took up to three extra years to graduate (considered as moderate risk).
- **Failure**: Students who took more than three extra years to graduate or did not graduate (considered as high risk).

Here, risk is considered as the exposure to not finishing your studies or doing so with a long delay, building on the consideration by Martins et al. (2021). For our dataset, about 52% of the students graduate on time, 30% of the students drop out, and 18% of the students graduate with delay.

For the independent variables, our analysis incorporates three aspects of variables. First, we include academic variables such as admission grade, course preference order at enrolment, and first/second semester average grades (0-20). Second, we consider demographic variables such as age at

enrolment, gender, and marital status. Finally, we consider socio-economic variables such as parents' occupations and education, student debt and scholarships. To facilitate the analysis, all categorical variables were converted into dummy variables. We do not apply any other filters as there are no missing values.

Table 1: Sample Summary Statistics

Variable	**Mean**	**Min.**	**Max.**	**s.d.**	**obs.**
Previous Qualification (grade)	132.6	95.0	190.0	13.18833	4424
Admission Grade	127.0	95.0	190.0	14.482	4424
Age at Enrollment	23.27	17.0	70.0	7.587816	4424
Gender	0.3517	0	1	0.4775604	4424
Debtor Dummy	0.1137	0	1	0.31748	4424
Scholarship Holder Dummy	0.2484	0	1	0.4321442	4424

To evaluate the performance of our models, we split the datasets into training and test sets. The data was divided such that 80% was used for training the model and 20% was reserved for testing.

3 Methods and Validation

3.1 Data Balancing

In this section, we discuss the two data balancing methods we utilize. As mentioned in Section 2, we have an imbalanced data with about 52% of the students graduating on time, 30% of the students dropping out, and 18% of the students graduating with delay. Therefore, to account for this phenomenon, we take two different alternatives.

In the first approach, we balance the three populations, using random undersampling to ensure the representation of minority groups. The two larger groups (i.e. students graduating on time and students dropping out) are randomly sampled n_s times, where n_s denotes the number of observations of the smallest group (i.e. students graduating with delay). Although this reduces the number of observations, there are cases where random undersampling outperformed other oversampling methods such as SMOTE and random oversampling such as in Hasanin et al. (2019).

In the second approach, we oversample the minority groups using Synthetic Minority Oversampling Technique (SMOTE) proposed by Chawla et al. (2002). The two minority groups (i.e. students graduating with delay and students dropping out) are oversampled n_l times, where n_l denotes the number of observations of the largest group (i.e. students graduating on time), using synthetic samples. Such synthetic samples are generated by first calculating the difference between a sample's feature vector and that of a nearby neighbour, identified using k nearest neighbours. This difference is then multiplied by a random number between 0 and 1, and the resulting value is added to the original feature vector. The process creates more general decision regions, avoiding overfitting compared to just

resampling the minority group n_l times(Chawla et al., 2002).

However, SMOTE has a weakness that it introduces additional artificial noise and affects the classification(Grina et al., 2020). Therefore, we also compare the results with the original unbalanced data to evaluate its performance.

3.2 Decision Trees and Random Forests

In this paper, the goal is to predict and classify which of the three groups each student falls into. Many supervised machine learning methods could be used for this task, such as multinomial logistic regression, decision trees, and support vector machines. However, we opt for decision trees and random forests because of the following advantages:

1. **Decision Trees**
    - Simple and easy to interpret
2. **Random Forests**
    - Better accuracy than decision trees
    - Less sensitive to noise (less overfitting)
3. **Both Methods**
    - No need of rescaling variables
    - Drops unnecessary variables (parsimonious model)
    - Robust against outliers

Because our case contains 240 variables and differently scaled variables, the trees can provide effective predictions. In addition, random forest is less affected by the noise created by SMOTE for the oversampled data.

Decision trees are a method that divides the decision spaces into disjoint rectangular subspaces. Consider a classification case and suppose we have n observations, $0,\ldots,C-1$ outcomes, M different subspaces, and define the m-th subspace as $R_m; m = 1,\ldots,M$. Then the prediction function $g(x)$ is equal to,

$$g(x) = \sum_{m=1}^{M} \mathbf{1}(x \in R_m) c_m$$

where c_m is the predictor for subspace m. Then, we estimate $\hat{c}_m$ as

$$\hat{c}_m = \arg \max_{j \in \{0,\ldots,C-1\}} \hat{p}_{mj}$$

where $\hat{p}_{mj}$, which is the proportion of class j observations in subspace m, equals

$$\hat{p}_{mj} = \frac{\sum_{i=1}^{n} \mathbf{1}(Y_i = j)\mathbf{1}(X_i \in R_m)}{\sum_{i=1}^{n} \mathbf{1}(X_i \in R_m)}$$

Decision trees further split the node by either conducting a horizontal or vertical split, which reduces the training loss the most. However, decision trees, by themselves, have a high variance due to the above nature. If

there is a strong feature that creates a split high up in the tree, it affects the whole tree. If the split is determined by few observations, the trees may change drastically if those observations drop out.

Bagging is a method that can mitigate the high variance of decision trees. To reduce the variance, we ideally have B independent training samples. Then, the variance would reduce by $\frac{1}{B}$. However, we usually do not have B training samples. Instead, Bagging creates the B training samples by bootstrapping with resampling from the original training sample. The decrease in the variance depends on the correlation in between the bootstrapped samples. Thus, if all trees have similar splits, the reduction in the variance is marginal.

In contrast, the Random Forests method addresses one of the main limitations of bagged trees, namely, the issue of dominant features that generate identical bootstrapped trees. In addition to sampling observations, Random Forests mitigate the high correlation among the bagged samples by also sampling the features without replacement, thus neutralizing the effect of the aforementioned dominant features.

3.3 Training Cross-Validation and Pruning

Decision trees tend to overfit the training sample, meaning that they learn from the noise (also referred to as 'irreducible risk') of this sample, which reduces their ability to generalize to new, unseen data. To avoid this overfitting apart from considering different models, we also conduct refinements of current models, by pruning (for decision trees) and cross-validation (for all the models).

Regarding cross-validation, we employ 10-fold cross-validation in the training set. This method involves splitting the dataset into k distinct folds where $k = 10$. The models are then trained on $k-1$ folds and validated on the remaining fold. This process is repeated k times, with each fold being used exactly once as the validation set (Ramezan et al., 2019).

The selection of 10-fold cross validation is motivated by its ability to balance the trade-off between bias and variance. Specifically, using ten folds reduces bias since each training iteration utilizes 90% of the data, providing a more extensive learning set (Blockeel and Struyf, 2002). At the same time, it mitigates variance by testing the model on multiple different subsets, leading to a more reliable and stable performance estimate. This is particularly advantageous for our analysis, as it maximizes the efficient use of the training dataset for both training and validation purposes, ensuring robust model assessment and minimizing the risk of overfitting (Ramezan et al., 2019).

All the models underwent a process of hyperparameter optimization through cross validation. Similarly, as done in Martins et al. (2021) a Randomized Grid Search was used for this purpose, where a set of parameter values and combinations is randomly chosen, allowing control over the

number of parameter combinations that are attempted without computational expense.

To add to the Martins et al. (2021) study, we incorporate a pruning technique to address overfitting in decision trees and improve their performance on new data by removing the least reliable branches (Mingers, 1989). Same as above, we optimized hyperparameters using Randomised Grid Search. This process, combined with 10-fold cross-validation, balances bias and variance, leading to a more robust and reliable model. Lastly, by preventing overfitting and enhancing performance metrics such as accuracy and Cohen's kappa score, pruning helps our decision tree models generalize well to unseen data (Bradford et al., 1998).

3.4 Model Comparison

In terms of model cross-validation, various statistics such as accuracy, recall, F1 score, Cohen's Kappa, and OOB (Out-of-Bag) score were employed to compare the performance of the different models. Given our focus on classification, measures such as RMSE, MSE or R^2 were disregarded.

Accuracy offers a general measure of correctness but can be misleading in imbalanced datasets like ours, where many students graduate on time and less fail or graduate late. Recall, defined as the percentage of true positives out of all positives in the prediction, is crucial for ensuring that student interventions are not missed, as it ensures that most at-risk students are flagged. Here, a positive outcome is considered as you being part of a category, and therefore a true positive outcome accounts for the correct prediction of class membership. Thus, we aim for a high recall, especially for the categories of failure and relative success. Conversely, this objective would ensure a low false negative rate, which would account for our classifications missing flagging at risk for students that are actually at risk, which is what we aim to minimize. The F1 score, defined as:

$$\text{F1 Score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

emphasizes the harmonic mean of precision and recall, making it particularly useful for imbalanced datasets, like our case. With it, we allow for a balance in the objective of reducing false positives but also false negatives, where the later it is the most crucial. Furthermore, Cohen's Kappa provides an evaluation of chance by measuring agreement, which helps understand the model's reliability, and its essential given the category imbalance. Finally, the OOB score, specific to ensemble methods (i.e., Bagged Trees and Random Forests), offers an unbiased estimate of model generalization error, as it uses the data not being selected during the resampling, which prevents overfitting.

3.5 Feature Importance

For every predicting feature, we consider its feature importance, which is measured as the magnitude of change in a model's prediction error, measured in our case by accuracy when the corresponding predictor is premutated. Consequently, a large increase in a model's predictive error, denotes a large importance of the feature given that when its value is alternated the association between the outcome and the predictor are destroyed, leading to a loss of predictive capacity which is absorbed by the increase in the model's predictive error.

4 Analysis and insights

The results of the predictions are given in Table 2 from which we depict that when considering the oversampled data the performance across methods is enhanced, compared to using the undersampled and unbalanced data. This is likely since the minority groups are represented as the same as the majority group compared to the unbalanced data, and also have more number of observations than the undersampled data. Although our case does not have extreme cases of minority group, such as in fraud detection (where less than 1% are fraud transactions), the gap in between the largest and the smallest group is large enough to under-represent the minority group, resulting in less accuracy. With unbalanced data, the decision boundary can be skewed towards the majority group, resulting in less accuracy.

Comparing different models, the bagged trees with oversampling perform the best based on the metrics. The recall, F1 score and the out of the bag score are the largest for this specification compared to the rest. Because our goal is to predict students who succeed in university, we aim to avoid the false negatives in the prediction. Thus, as stated in Section 3.4, the F1 score and the recall are more important measures in this case. Therefore, although the accuracy and Cohen's kappa is smaller than the oversampled random forest model, we believe the bagged trees with oversampling performed the best for our goals of predicting academic success, and should be considered as the preferred model.

Considering the above preferences in model selection and data balancing, we also believe in the need to display measures of feature importance, as done in Figures 1 and Figure 2. They are to be understood as the predictors that have most predictive power. Across the models, we observe that some predictors are consistently selected across the top 10 most predictive features. In terms of academic predictors, the most predictive are the average grade on curricular units in the 1st and 2nd semester, as well as the number of units approved in those semesters. Moreover, previous qualification grade is less predictive but still within the top 10. In terms of demographics, they are age at enrolment. In terms of socioeconomic, they are whether the student has paid tuition fees paid up to date.

We believe these predictors should be considered as the dimensions on which university administrators should focus not only for predicting

students' success but also designing policies and interventions to improve academic success, especially for those in risk of not graduating or graduating but with more than 3 years of delay.

However, such selected predictors are not necessarily causal drivers of academic success, and therefore a causality study should be aimed for this, from which this paper abstains. However, in this work we shed light on which are the dimensions with most potential in affecting academic success given their high predictive power and on which future studies should focus on.

Table 2: Comparison of Model Performance under Different Data Balancing Techniques

Model	**Data Balancing**	**Accuracy**	**Recall**	**F1 Score**	**Kappa**	**OOB Score**
Single Decision Tree	Unbalanced	0.682	0.574	0.561	0.487	-
	Oversampled	0.743	0.843	0.830	0.615	-
	Undersampled	0.593	0.704	0.674	0.389	-
Pruned Tree	Unbalanced	0.755	0.595	0.595	0.581	-
	Oversampled	0.732	0.819	0.804	0.597	-
	Undersampled	0.667	0.746	0.713	0.500	-
Bagged Trees	Unbalanced	0.759	0.680	0.669	0.596	0.771
	Oversampled	0.808	0.888	0.861	0.712	0.817
	Undersampled	0.723	0.809	0.781	0.584	0.705
Pruned and Bagged Trees	Unbalanced	0.757	0.606	0.626	0.589	0.768
	Oversampled	0.776	0.866	0.839	0.664	0.789
	Undersampled	0.692	0.738	0.732	0.538	0.688
Random Forests	Unbalanced	0.757	0.645	0.651	0.585	0.766
	Oversampled	0.812	0.811	0.811	0.716	0.824
	Undersampled	0.721	0.720	0.720	0.582	0.699

Note: "OOB Score" is not applicable for single decision tree and pruned tree.

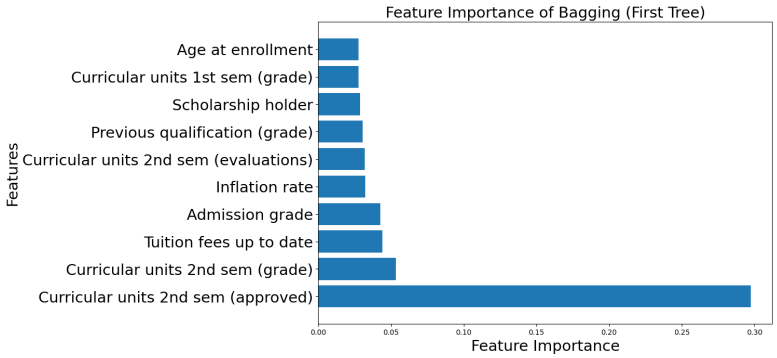

Figure 1: 10 Most Important Features for Bagged / Pruned Trees

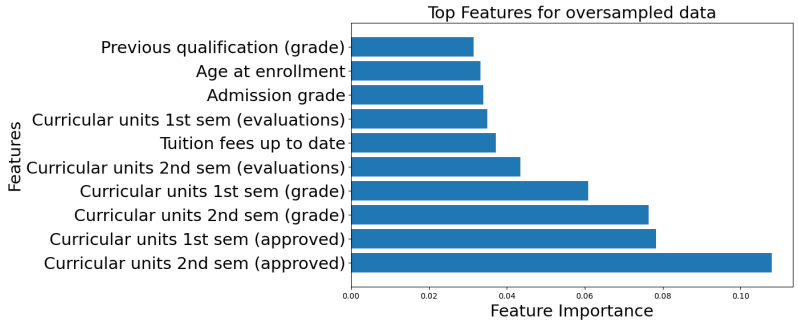

Figure 2: 10 Most Important Features for Random Forests Trees

5 Conclusion

In this paper, we predicted the academic success of university students using Supervised Machine Learning methods, such as Decision Trees, Bagged Decision Trees, Random Forests and some of their improvements. Our models considered academic, demographic and socio-economic pre- and post-enrolment predictors, coming from 4424 students from a Portuguese university. Academic success was measured through whether the student graduated on time, with a delay of at most 3 years, or took more than 3 years to do so or did not do so. Our data suffers from unbalancing and in attempts to solve it both undersampling and oversampling are considered.

Our results indicate that the considered models predict with a very similar level of accuracy, which varies equally across models in terms of what type of balancing the data was exposed to. We argue that our main focus is not accuracy but the correct identification of true positives to decrease the prevalence of false negatives, which would amount to missing classifying some students to be at risk of dropping out or experiencing serious delay. Therefore, we select the best model based on F1 score and recall rates, which brings us to favour decision trees with bagging for oversampled data. Moreover, our paper also shows what are predictors with the most predictive power consistently across different models, which we argue should be the focus for university administrators when assessing the potential for success of students.

Further work on the causal dimension should focus on uncovering the causal impact of the selected most predictive variables, as well as interventions or policies aimed at changing them, e.g. help in keeping up to date with payments. Parallelly, further steps on the predictive dimension aimed to improve model performance should explore alternative oversampling methods, such as ADASYN, which weighs the information about how much the majority group exists near the minority group. Moreover, opting for more computer intensive prediction models such as XGBoost and LightGBM given that we face a small dataset may have potential to yield better predictive performance.

References

Beaulac, C. and Rosenthal, J. S. (2019). Predicting university students' academic success and major using random forests. *Research in Higher Education*, 60(7):1048–1064.

Blockeel, H. and Struyf, J. (2002). Efficient algorithms for decision tree cross-validation. *Journal of Machine Learning Research*, 3:621–650.

Bradford, J. P., Kunz, C., Kohavi, R., Brunk, C., and Brodley, C. E. (1998). Pruning decision trees with misclassification costs. In Nédellec, C. and Rouveirol, C., editors, *Machine Learning: ECML-98, 10th European Conference on Machine Learning*, volume 1398 of *Lecture Notes in Computer Science*, pages 131–136. Springer, Berlin, Heidelberg.

Chawla, N. V., Bowyer, K. W., Hall, L. O., and Kegelmeyer, W. P. (2002). Smote: Synthetic minority over-sampling technique. *Journal of Artificial Intelligence Research*, 16:321–357.

Council of the European Union (2021). Council resolution on a strategic framework for european cooperation in education and training towards the european education area and beyond (2021-2030). Official Journal of the European Union, C 66, 26.2.2021, p. 1-21. Accessed: 26 February 2021.

Eurostat (2024). Early leavers from education and training. *Statistics Explained*. Accessed: 23 May 2024.

Grina, F., Elouedi, Z., and Lefevre, E. (2020). A preprocessing approach for class-imbalanced data using smote and belief function theory. In Analide, C., Novais, P., Camacho, D., and Yin, H., editors, *Intelligent Data Engineering and Automated Learning – IDEAL 2020*, pages 3–11, Cham. Springer International Publishing.

Hasanin, T., Khoshgoftaar, T. M., Leevy, J. L., et al. (2019). Severely imbalanced big data challenges: investigating data sampling approaches. *Journal of Big Data*, 6(1):107.

Hoffait, A.-S. and Schyns, M. (2017). Early detection of university students with potential difficulties. *Decision Support Systems*, 101:1–11.

Martins, M. V., Tolledo, D., Machado, J., Baptista, L., and Realinho, V. (2021). *Early Prediction of student's Performance in Higher Education: A Case Study*.

Miguéis, V., Freitas, A., Garcia, P. J., and Silva, A. (2018). Early segmentation of students according to their academic performance: A predictive modelling approach. *Decision Support Systems*, 115:36–51.

Mingers, J. (1989). An empirical comparison of pruning methods for decision tree induction. *Machine Learning*, 4(2):227–243.

Ministry of Education, Culture, Sports, Science and Technology (2023). Annual report on education 2023. http://www.mext.go.jp/en/. Accessed: 2023-09-30.

Ramezan, C. A., Warner, T. A., and Maxwell, A. E. (2019). Evaluation of sampling and cross-validation tuning strategies for regional-scale machine learning classification. *Remote Sensing*, 11(2):185. Submission received: 8 December 2018 / Revised: 13 January 2019 / Accepted: 16 January 2019 / Published: 18 January 2019. This article belongs to the Special Issue Region Based Classification (RBC), Object Based Image Analysis (OBIA) and Deep Learning (DL) for Remote Sensing Applications.

Realinho, Valentim, V. M. M. M. J. and Baptista, L. (2021). Predict Students' Dropout and Academic Success. UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C5MC89.

Thammasiri, D., Delen, D., Meesad, P., and Kasap, N. (2014). A critical assessment of imbalanced class distribution problem: The case of predicting freshmen student attrition. *Expert Systems with Applications*, 41(2):321–330.

U.S. Department of Education (2023). Educational statistics 2023. https://www.ed.gov/data. Accessed: 2023-09-30.

A Appendix

A.1 Data Balancing

In this section, we undersample the two larger groups to the number of observations of the smallest group. From the second chunk, we oversample the two minority groups using SMOTE.

Python Code

```
#Determine the size of the smallest group
min_size = df['Target'].value_counts().min()
#Set the seed for reproducibility
seed = 123
#Sample each group to the size of the smallest group
df_graduate = df[df['Target'] ==
 ↪  'Graduate'].sample(min_size, random_state=seed)
df_enrolled = df[df['Target'] ==
 ↪  'Enrolled'].sample(min_size, random_state=seed)
df_dropout = df[df['Target'] ==
 ↪  'Dropout'].sample(min_size, random_state=seed)
#Combine the balanced groups into a new DataFrame
df_balanced = pd.concat([df_graduate, df_enrolled,
 ↪  df_dropout])
```

Python Code

```
y = df['Target']
x = df.drop(columns=['Target'])
#Choose the seed for SMOTE
sm = SMOTE(random_state=654321)
#Resample using SMOTE
x_res, y_res = sm.fit_resample(x, y)
```

A.2 Decision Tree and Bagging

In this section, we train various Decision Tree models, including a basic decision tree, a pruned decision tree, and a decision tree with bagging. The models are evaluated using cross-validation, and we visualize the decision trees and feature importance to interpret the results.

Python Code

```

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X,
 ↪  y, test_size=0.2, stratify=y, random_state=42)

scaler = StandardScaler()
X_train[numerical_cols] =
 ↪  scaler.fit_transform(X_train[numerical_cols])
X_test[numerical_cols] =
 ↪  scaler.transform(X_test[numerical_cols])

```

```
9
10 # Function to calculate additional metrics
11 def calculate_metrics(y_true, y_pred):
12     cm = confusion_matrix(y_true, y_pred)
13     TN = cm[0, 0]
14     FP = cm[0, 1]
15     FN = cm[1, 0]
16     TP = cm[1, 1]
17
18     # Calculate metrics
19     sensitivity = TP / (TP + FN)
20     specificity = TN / (TN + FP)
21     precision = TP / (TP + FP)
22     neg_pred_value = TN / (TN + FN)
23     f1 = 2 * (precision * sensitivity) / (precision +
       ↪ sensitivity)
24     prevalence = (TP + FN) / (TP + TN + FP + FN)
25     detection_rate = TP / (TP + TN + FP + FN)
26     detection_prevalence = (TP + FP) / (TP + TN + FP +
       ↪ FN)
27     balanced_accuracy = (sensitivity + specificity) / 2
28
29     metrics = {
30         'Sensitivity (Recall)': sensitivity,
31         'Specificity': specificity,
32         'Pos Pred Value (Precision)': precision,
33         'Neg Pred Value': neg_pred_value,
34         'F1 Score': f1,
35         'Prevalence': prevalence,
36         'Detection Rate': detection_rate,
37         'Detection Prevalence': detection_prevalence,
38         'Balanced Accuracy': balanced_accuracy
39     }
40
41     return metrics
42
43
44 # Cross-validation setup
45 def evaluate_model(model, X_train, y_train, X_test,
   ↪ y_test):
46     # Cross-validation
47     accuracy_scores = cross_val_score(model, X_train,
       ↪ y_train, cv=10, scoring='accuracy')
48     kappa_scorer = make_scorer(cohen_kappa_score)
49     kappa_scores = cross_val_score(model, X_train,
       ↪ y_train, cv=10, scoring=kappa_scorer)
50
51     model.fit(X_train, y_train)
52     y_pred = model.predict(X_test)
53     accuracy = accuracy_score(y_test, y_pred)
54     kappa = cohen_kappa_score(y_test, y_pred)
```

```
    metrics = calculate_metrics(y_test, y_pred)
    metrics['Accuracy'] = accuracy
    metrics['Cohen\'s Kappa'] = kappa

    return metrics, accuracy_scores.mean(),
     ↪ accuracy_scores.std(), kappa_scores.mean(),
     ↪ kappa_scores.std()


def plot_decision_tree(model, feature_names, title):
    plt.figure(figsize=(20, 10))
    plot_tree(model, feature_names=feature_names,
     ↪ filled=True, rounded=True,
     ↪ class_names=label_encoder.classes_)
    plt.title(title)
    plt.show()


def plot_top_feature_importance(model, feature_names,
 ↪ title, top_n=10):
    importances = model.feature_importances_
    indices = np.argsort(importances)[::-1]
    top_indices = indices[:top_n]
    top_importances = importances[top_indices]

    plt.figure(figsize=(10, 6))
    plt.title(title)
    plt.barh(range(top_n), top_importances,
     ↪ align="center")
    plt.yticks(range(top_n), [feature_names[i] for i in
     ↪ top_indices])
    plt.xlabel("Feature Importance")
    plt.ylabel("Features")
    plt.ylim([-1, top_n])
    plt.show()

# 1. Single decision tree without any filtering
tree_no_filter = DecisionTreeClassifier(random_state=42)
metrics_no_filter, acc_mean_no_filter,
 ↪ acc_std_no_filter, kappa_mean_no_filter,
 ↪ kappa_std_no_filter = evaluate_model(
    tree_no_filter, X_train, y_train, X_test, y_test)
plot_decision_tree(tree_no_filter, X.columns, "Decision
 ↪ Tree without Filtering")
plot_top_feature_importance(tree_no_filter, X.columns,
 ↪ "Feature Importance without Filtering")

# 2. Single decision tree with pruning
dt_model = DecisionTreeClassifier(random_state=42)
params = {
    'max_depth': [None, 10, 20, 30],
```

```
    'min_samples_split': randint(2, 20),
    'ccp_alpha': uniform(0.0, 0.1)
}
dt_random_search = RandomizedSearchCV(dt_model,
    param_distributions=params, n_iter=100, cv=10,
    n_jobs=-1,
                                      random_state=42)
dt_random_search.fit(X_train, y_train)
best_params = dt_random_search.best_params_
pruned_tree = DecisionTreeClassifier(random_state=42,
    **best_params)
metrics_pruned_tree, acc_mean_pruned, acc_std_pruned,
    kappa_mean_pruned, kappa_std_pruned =
    evaluate_model(
    pruned_tree, X_train, y_train, X_test, y_test)
plot_decision_tree(pruned_tree, X.columns, "Pruned
    Decision Tree")
plot_top_feature_importance(pruned_tree, X.columns,
    "Feature Importance of Pruned Tree")

# 3. Single decision tree with bagging
bagging_tree =
    BaggingClassifier(estimator=DecisionTreeClassifier(random_state=42,
    ccp_alpha=0),
                                  n_estimators=100,
                                   random_state=42,
                                   oob_score=True)
metrics_bagging_tree, acc_mean_bagging, acc_std_bagging,
    kappa_mean_bagging, kappa_std_bagging =
    evaluate_model(
    bagging_tree, X_train, y_train, X_test, y_test)
plot_decision_tree(bagging_tree.estimators_[0],
    X.columns, "Bagging Decision Tree (First Tree)")
plot_top_feature_importance(bagging_tree.estimators_[0],
    X.columns, "Feature Importance of Bagging (First
    Tree)")

# 4. Single decision tree with pruning and bagging
bagging_pruned_tree =
    BaggingClassifier(estimator=DecisionTreeClassifier(random_state=42,
    **best_params),

                                          n_estimators=100,
                                          random_state=42,
                                          oob_score=True)
metrics_bagging_pruned_tree, acc_mean_bagging_pruned,
    acc_std_bagging_pruned, kappa_mean_bagging_pruned,
    kappa_std_bagging_pruned = evaluate_model(
    bagging_pruned_tree, X_train, y_train, X_test,
     y_test)
```

```
plot_decision_tree(bagging_pruned_tree.estimators_[0], X.columns, "Bagging Pruned Decision Tree (First Tree)")
plot_top_feature_importance(bagging_pruned_tree.estimators_[0], X.columns,
                            "Feature Importance of Bagging Pruned (First Tree)")

```

A.3 Random Forests

In this section, we train a basic Random Forest model by using the scikit-learn libraries. We also train the hyperparameters and perform cross-validation, as stated in the methodology. We iterate through the three data samples (unbalanced, oversample, undersample) to calculate the relevant statistics.

Python Code

```
import pandas as pd
import numpy as np

# Modelling
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import (accuracy_score, confusion_matrix, precision_score, recall_score,
                             ConfusionMatrixDisplay, cohen_kappa_score, f1_score)
from sklearn.model_selection import RandomizedSearchCV, train_test_split
from scipy.stats import randint
from sklearn.preprocessing import LabelEncoder

# File paths for the different datasets
file_paths = {
    'unbalanced': '/content/drive/Shareddrives/Data Analysis and Machine Learning/code/data balancing/unbalanced_data.xlsx',
    'oversampled': '/content/drive/Shareddrives/Data Analysis and Machine Learning/code/data balancing/data_oversample.xlsx',
    'undersampled': '/content/drive/Shareddrives/Data Analysis and Machine Learning/code/data balancing/data_undersample.xlsx'
}

# Loop through each file path and perform the operations
for balance_type, file_path in file_paths.items():
    print(f"\nProcessing {balance_type} dataset\n")
```

```
22
23      # Load data
24      dataset = pd.read_excel(file_path)
25
26      X = dataset.drop('Target', axis=1)
27      y = dataset['Target']
28
29      # Encode target labels to numerical values
30      label_encoder = LabelEncoder()
31      y = label_encoder.fit_transform(y)
32
33      # Split the data into training and test sets
34      X_train, X_test, y_train, y_test =
         ↪  train_test_split(X, y, test_size=0.2,
         ↪  random_state=42)
35
36      # Fitting/Evaluating Model
37      rf = RandomForestClassifier()
38      rf.fit(X_train, y_train)
39
40      # Make sure it's making accurate predictions
41      y_pred = rf.predict(X_test)
42
43      # Printing accuracy score
44      accuracy = accuracy_score(y_test, y_pred)
45      kappa = cohen_kappa_score(y_test, y_pred)
46      print("Initial Model Accuracy:", accuracy)
47      print("Initial Model Cohen's kappa:", kappa)
48
49      # Hyperparameters tuning
50      param_dist = {'n_estimators': randint(50, 500),
51                    'max_depth': randint(1, 20)}
52
53      # Create a random forest classifier
54      rf = RandomForestClassifier()
55
56      # Use random search to find the best
         ↪  hyperparameters
57      rand_search = RandomizedSearchCV(rf,
58
                                         ↪  param_distributions=param_dist,
59                                       n_iter=5,
60                                       cv=10,
61                                       random_state=42)
62
63      # Fit the random search object to the data
64      rand_search.fit(X_train, y_train)
65
66      # Create a variable for the best model
67      best_rf = rand_search.best_estimator_
68
```

```
    # Print the best hyperparameters
    print('Best hyperparameters:',
     ↪  rand_search.best_params_)

    # Generate predictions with the best model
    y_pred = best_rf.predict(X_test)

    # Create the confusion matrix
    cm = confusion_matrix(y_test, y_pred)
    ConfusionMatrixDisplay(confusion_matrix=cm).plot()

    # Final accuracy, precision, and recall using the
     ↪  best model
    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred,
     ↪  average='macro')
    recall = recall_score(y_test, y_pred,
     ↪  average='macro')
    f1 = f1_score(y_test, y_pred, average='macro')

    print("Tuned Model Accuracy:", accuracy)
    print("Tuned Model Precision:", precision)
    print("Tuned Model Recall:", recall)
    print("Tuned Model F1 Score:", f1)

    # Calculate Cohen's kappa score
    kappa = cohen_kappa_score(y_test, y_pred)
    print("Tuned Model Cohen's kappa:", kappa)

    # Instantiate the RandomForestClassifier with
     ↪  oob_score=True
    rf = RandomForestClassifier(oob_score=True,
     ↪  random_state=42)

    # Fit the model to the training data
    rf.fit(X_train, y_train)

    # Calculate the OOB score
    oob_score = rf.oob_score_
    print("OOB score:", oob_score)
```