Student

Chirui GUO

2024-12-11

Student Capstone Project

Introduction

One of the challenges faced by higher education institutions worldwide is addressing the different learning styles and academic performances of students. This is important for improving students' learning experiences and the overall efficiency of the institution. Predicting and identifying potential difficulties that students might face is crucial for institutions that want to develop strategies to support and guide students at risk of failing or dropping out.

The dataset was developed as part of a project focused on decreasing academic dropout and failure rates in higher education. It utilizes machine learning methods to early identify students who are at risk, allowing for the implementation of support strategies to assist them in their academic journey.

Dataset

Predict Students' Dropout and Academic Success is a dataset created from a higher education institution (acquired from several disjoint databases) related to students enrolled in different undergraduate degrees, such as agronomy, design, education, nursing, journalism, management, social service, and technologies. The dataset includes information known at the time of student enrollment (academic path, demographics, and social-economic factors) and the students' academic performance at the end of the first and second semesters. The data is used to build classification models to predict students' dropout and academic sucess. The problem is formulated as a three category classification task, in which there is a strong imbalance towards one of the classes. [1]

The main target for this project is to **improve the machine learning performance by oversampling techniques for class imbalance problem**.

Firstly, download and import the needed libraries: caret, tidyverse, gbm, smotefamily, doParallel, ggtext

```
if(!require(tidyverse)) install.packages("tidyverse", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: tidyverse
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.4     v readr     2.1.5
## v forcats   1.0.0     v stringr   1.5.1
## v ggplot2   3.5.1     v tibble    3.2.1
## v lubridate 1.9.3     v tidyr     1.3.1
## v purrr     1.0.2
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

```r
if(!require(ggtext)) install.packages("ggtext", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: ggtext
```

```r
if(!require(caret)) install.packages("caret", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: caret
## Loading required package: lattice
##
## Attaching package: 'caret'
##
## The following object is masked from 'package:purrr':
##
##     lift
```

```r
if(!require(gbm)) install.packages("gbm", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: gbm
## Loaded gbm 2.2.2
## This version of gbm is no longer under development. Consider transitioning to gbm3, https://github.c
```

```r
if(!require(smotefamily)) install.packages("smotefamily", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: smotefamily
```

```r
if(!require(doParallel)) install.packages("doParallel", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: doParallel
## Loading required package: foreach
##
## Attaching package: 'foreach'
##
## The following objects are masked from 'package:purrr':
##
##     accumulate, when
##
## Loading required package: iterators
## Loading required package: parallel
```

```r
library(tidyverse)
library(ggtext)
library(caret)
library(gbm)
library(smotefamily)
library(doParallel)
```

Methods

Data preprocessing

Download **Predict Students' Dropout and Academic Success** dataset from link. and load into memory.

There are 4424 instance and 37 columns in the dataset, and each instance is a student in the dataset.

```r
options(timeout = 120)
dl <- "student.zip"
if(!file.exists(dl))
  download.file("https://archive.ics.uci.edu/static/public/697/predict+students+dropout+and+academic+su
```

```
data <- "data.csv"
if(!file.exists(data))
  unzip(dl, data)

df <- read_delim(data, delim=";", show_col_types = FALSE)
rm(data, dl)
dim(df)
```

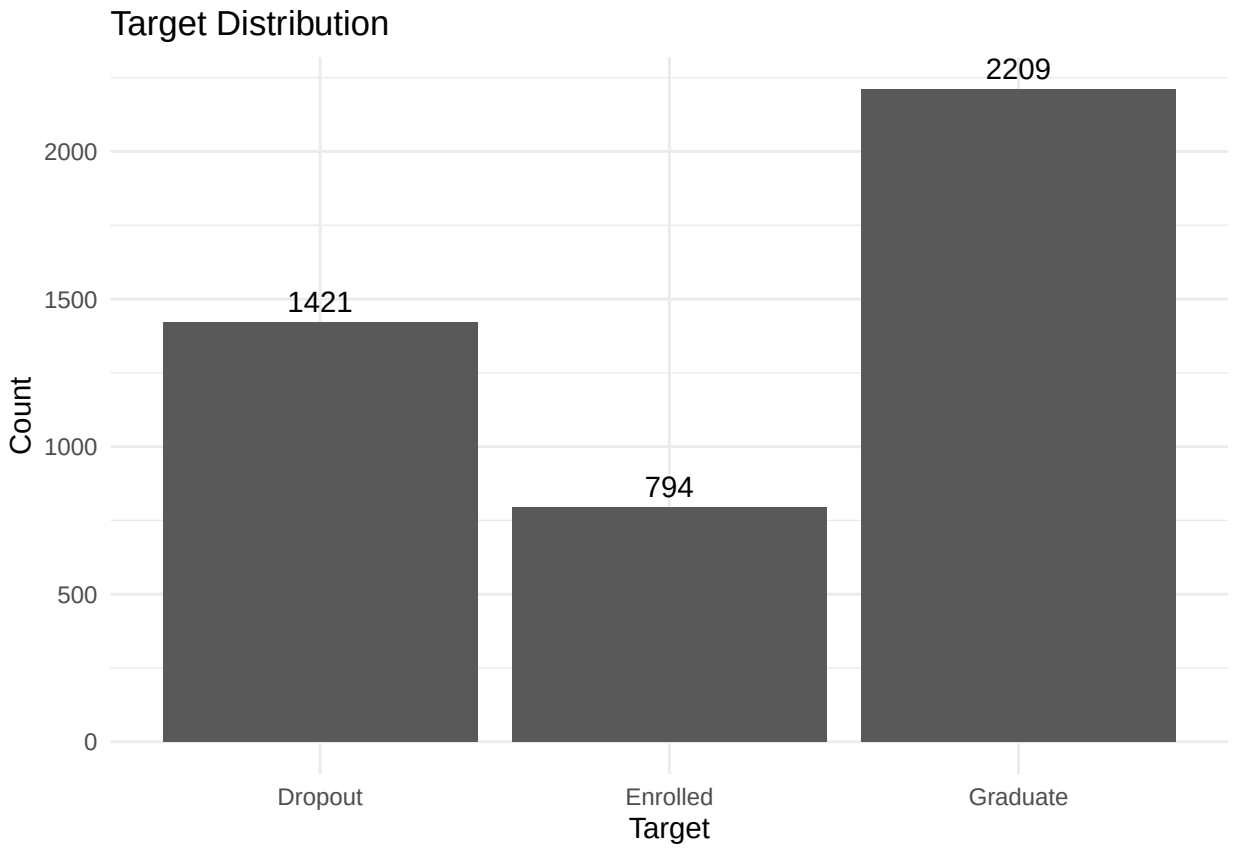
```
## [1] 4424   37
```

Exploratory Analysis

We find that the target is imbalanced in this dataset. There are 1421 students dropout, 794 students enrolled and 2209 students graduate.

It is challenging for multi-class classification problem.

```
df |>
  ggplot(aes(x = Target)) +
  geom_bar() +
  geom_text(stat = 'count', aes(label = after_stat(count)), vjust = -0.5) +
  scale_x_discrete(labels = c("1" = "A", "2" = "B", "3" = "C")) +
  labs(title = "Target Distribution", x = "Target", y = "Count") +
  theme_minimal()
```



The next step is data transformation, which facilitates subsequent exploratory analysis and model training.

```
df <- df |>
  mutate(across(where(is.factor), as.numeric))|>  # transform factor into numeric
```

```
  mutate(across(where(is.character), as.factor)) # transform character into factor
```

Train and Test set Then split the raw dataset into 80% train set and 20% test set.

The 80/20 split ratio is optimal for this dataset because it:

- Provides sufficient data for both training and testing
- Handles class imbalance effectively
- Ensures reliable model evaluation
- Aligns with established practices in educational data analysis [1]

```
# Set random seed
set.seed(42)

# 1.Data preprocessing
# Split train set and test set
index <- createDataPartition(df$Target, p = 0.8, list = FALSE)
train_data <- df[index, ]
test_data <- df[-index, ]

# 2. Applying ADASYN algorithm and SMOTE algorithm
# prepared train set
X_train <- train_data |>
  select(-Target)
y_train <- train_data |>
  select(Target)
```

Oversampling Techniques

In order to generate balanced dataset for class imbalance problem, we choose four method for comparing:

1. Baseline Baseline: Train the model without Oversampling.

2. SMOTE SMOTE (Synthetic Minority Oversampling Technique) is a data augmentation method that generates synthetic samples for the minority class to address class imbalance. The method works by interpolating between existing minority class samples and their nearest neighbors in the feature space.

3. ADASYN ADASYN (Adaptive Synthetic Sampling) is a data oversampling technique designed to handle imbalanced datasets. It improves upon SMOTE by adaptively generating synthetic samples based on the difficulty of learning minority class samples. The harder it is to classify a sample (i.e., closer to the decision boundary), the more synthetic samples are generated for it.

4. Borderline-SMOTE Borderline-SMOTE is an enhancement of the original SMOTE (Synthetic Minority Oversampling Technique) designed to focus on **hard-to-classify samples** near the decision boundary between the minority and majority classes. Instead of applying synthetic oversampling uniformly to all minority class samples, Borderline-SMOTE targets only those minority samples that are close to the majority class, as these are more likely to be misclassified.

```
# use SMOTE, ADASYN and Borderline-SMOTE to balance data
SMOTE_balanced <- SMOTE(X_train, y_train, K = 5)
ADASYN_balanced <- ADAS(X_train, y_train, K = 5)
BLSMOTE_balanced <- BLSMOTE(X_train, y_train, K = 5)
```

```
## [1] "Borderline-SMOTE done"
```

```
# get tibble
SMOTE_balanced_train <- SMOTE_balanced$data |>
  rename(Target = class)
ADASYN_balanced_train <- ADASYN_balanced$data |>
  rename(Target = class)
BLSMOTE_balanced_train <- BLSMOTE_balanced$data |>
  rename(Target = class)

rm(df, X_train, y_train, SMOTE_balanced, ADASYN_balanced, BLSMOTE_balanced) # free memory
```

As we know gradient boosting performs well in classification problems, we choose `gbm` as our base machine learning method. [2]

To reduce time consumption, we use `doParallel` package for parallel computing.

Also we use 5-fold cross validation for better performance.

```
# 3. Train the model
# use the balanced data to train the model

## Cross validation is too slow so we should do parallel computing
## Detect the number of cores and create a parallel cluster.
num_cores <- detectCores() - 1
cl <- makeCluster(num_cores)
registerDoParallel(cl)

# choose gradient boosting as machine learning model
ML_model <- "gbm"

Baseline_model <- train(
  Target ~ .,
  data = train_data,
  method = ML_model,
  trControl = trainControl(
    method = "cv",
    number = 5
  )
)
```

```
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        1.0986             nan     0.1000    0.1475
##      2        0.9935             nan     0.1000    0.1114
##      3        0.9167             nan     0.1000    0.0851
##      4        0.8566             nan     0.1000    0.0598
##      5        0.8137             nan     0.1000    0.0458
##      6        0.7809             nan     0.1000    0.0373
##      7        0.7534             nan     0.1000    0.0255
##      8        0.7334             nan     0.1000    0.0223
##      9        0.7149             nan     0.1000    0.0200
##     10        0.7003             nan     0.1000    0.0179
##     20        0.6266             nan     0.1000    0.0037
##     40        0.5721             nan     0.1000    0.0010
##     60        0.5432             nan     0.1000   -0.0011
##     80        0.5243             nan     0.1000   -0.0012
##    100        0.5085             nan     0.1000   -0.0013
```

```r
SMOTE_model <- train(
  Target ~ .,
  data = SMOTE_balanced_train,
  method = ML_model,
  trControl = trainControl(
    method = "cv",
    number = 5
  )
)
```

```
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        1.0986             nan     0.1000    0.1752
##      2        0.9876             nan     0.1000    0.1255
##      3        0.9052             nan     0.1000    0.0923
##      4        0.8433             nan     0.1000    0.0731
##      5        0.7952             nan     0.1000    0.0539
##      6        0.7588             nan     0.1000    0.0519
##      7        0.7254             nan     0.1000    0.0464
##      8        0.6949             nan     0.1000    0.0346
##      9        0.6714             nan     0.1000    0.0304
##     10        0.6502             nan     0.1000    0.0233
##     20        0.5361             nan     0.1000    0.0148
##     40        0.4404             nan     0.1000    0.0036
##     60        0.3953             nan     0.1000    0.0024
##     80        0.3684             nan     0.1000   -0.0005
##    100        0.3482             nan     0.1000   -0.0002
##    120        0.3326             nan     0.1000   -0.0006
##    140        0.3185             nan     0.1000   -0.0008
##    150        0.3124             nan     0.1000   -0.0005
```

```r
ADASYN_model <- train(
  Target ~ .,
  data = ADASYN_balanced_train,
  method = ML_model,
  trControl = trainControl(
    method = "cv",
    number = 5
  )
)
```

```
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        1.0986             nan     0.1000    0.1758
##      2        0.9842             nan     0.1000    0.1331
##      3        0.8996             nan     0.1000    0.0946
##      4        0.8383             nan     0.1000    0.0760
##      5        0.7878             nan     0.1000    0.0699
##      6        0.7422             nan     0.1000    0.0533
##      7        0.7096             nan     0.1000    0.0447
##      8        0.6801             nan     0.1000    0.0353
##      9        0.6568             nan     0.1000    0.0359
##     10        0.6334             nan     0.1000    0.0225
##     20        0.5191             nan     0.1000    0.0062
##     40        0.4236             nan     0.1000    0.0050
##     60        0.3778             nan     0.1000    0.0024
##     80        0.3513             nan     0.1000    0.0009
```

```
##    100        0.3318             nan     0.1000   -0.0005
##    120        0.3166             nan     0.1000   -0.0000
##    140        0.3021             nan     0.1000   -0.0004
##    150        0.2956             nan     0.1000   -0.0011
```

```
BLSMOTE_model <- train(
  Target ~ .,
  data = BLSMOTE_balanced_train,
  method = ML_model,
  trControl = trainControl(
    method = "cv",
    number = 5
  )
)
```

```
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        1.0986             nan     0.1000    0.1761
##      2        0.9810             nan     0.1000    0.1323
##      3        0.8937             nan     0.1000    0.1014
##      4        0.8283             nan     0.1000    0.0731
##      5        0.7793             nan     0.1000    0.0610
##      6        0.7391             nan     0.1000    0.0445
##      7        0.7090             nan     0.1000    0.0412
##      8        0.6816             nan     0.1000    0.0403
##      9        0.6546             nan     0.1000    0.0245
##     10        0.6365             nan     0.1000    0.0240
##     20        0.5228             nan     0.1000    0.0161
##     40        0.4252             nan     0.1000    0.0013
##     60        0.3843             nan     0.1000   -0.0001
##     80        0.3587             nan     0.1000   -0.0011
##    100        0.3380             nan     0.1000   -0.0003
##    120        0.3220             nan     0.1000   -0.0003
##    140        0.3085             nan     0.1000   -0.0008
##    150        0.3017             nan     0.1000   -0.0012
```

```
stopCluster(cl) #stop
registerDoSEQ() # recover sequential computing
rm(cl) # free the memory
```

Result

Then we use test set for evaluating these techniques.

```
# 4. Prediction and Evaluation

## Baseline
predictions <- predict(Baseline_model, test_data)
cm <- confusionMatrix(predictions, as.factor(test_data$Target))
accuracy <- as.numeric(cm$overall["Accuracy"]) # get accuracy
f1_scores <- cm$byClass[, "F1"]
macro_f1 <- mean(f1_scores, na.rm = TRUE) # get f1 score
f1_results <- tibble(method = "Baseline", accuracy = accuracy, f1 = macro_f1)

## SMOTE
predictions <- predict(SMOTE_model, test_data)
cm <- confusionMatrix(predictions, as.factor(test_data$Target))
```

```r
accuracy <- as.numeric(cm$overall["Accuracy"]) # get accuracy
f1_scores <- cm$byClass[, "F1"]
macro_f1 <- mean(f1_scores, na.rm = TRUE) # get f1 score
f1_results <- bind_rows(f1_results, tibble(method="SMOTE", accuracy = accuracy, f1 = macro_f1))

## ADASYN
predictions <- predict(ADASYN_model, test_data)
cm <- confusionMatrix(predictions, as.factor(test_data$Target))
accuracy <- as.numeric(cm$overall["Accuracy"]) # get accuracy
f1_scores <- cm$byClass[, "F1"]
macro_f1 <- mean(f1_scores, na.rm = TRUE) # get f1 score
f1_results <- bind_rows(f1_results, tibble(method="ADASYN", accuracy = accuracy, f1 = macro_f1))

## BLSMOTE
predictions <- predict(BLSMOTE_model, test_data)
cm <- confusionMatrix(predictions, as.factor(test_data$Target))
accuracy <- as.numeric(cm$overall["Accuracy"]) # get accuracy
f1_scores <- cm$byClass[, "F1"]
macro_f1 <- mean(f1_scores, na.rm = TRUE) # get f1 score
f1_results <- bind_rows(f1_results, tibble(method="BLSMOTE", accuracy = accuracy, f1 = macro_f1))

f1_results
```

```
## # A tibble: 4 x 3
##   method   accuracy    f1
##   <chr>       <dbl> <dbl>
## 1 Baseline    0.775 0.696
## 2 SMOTE       0.769 0.709
## 3 ADASYN      0.776 0.719
## 4 BLSMOTE     0.773 0.712
```

In the context of predicting students' dropout and academic success, we are dealing with a **multi-class classification problem** where the classes might be imbalanced (e.g., "Dropout", "Enrolled", "Graduate"). F1 Score is particularly suitable for this scenario because:

1. **Handles Class Imbalance**:
   - Educational datasets often have uneven class distributions
   - Some outcomes (like dropouts) may be less frequent than others
   - F1 Score balances precision and recall, preventing bias towards majority classes
2. **Comprehensive Evaluation**:
   - Combines both precision and recall into a single metric
   - Formula:
   $$F1 = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$
   - Provides a balanced assessment of model performance

The experimental results show that **ADASYN achieved an F1 score of 0.71917**, which is higher than other oversampling techniques. This superior performance can be attributed to:

1. **Better Handling of Class Boundaries**:
   - ADASYN focuses on difficult-to-learn examples
   - Generates more synthetic samples for minority class instances that are harder to learn
2. **Adaptive Nature**:
   - Adjusts the generation of synthetic samples based on local data density
   - Reduces noise in the oversampling process

Conclusion

Based on the F1 score of 0.71917, ADASYN demonstrates the best performance among the tested oversampling techniques for the Students' Dropout and Academic Success dataset. This selection:

1. **Ensures Better Prediction Quality**:
   - Higher F1 score indicates better balance between precision and recall
   - More reliable predictions for all classes
2. **Practical Benefits**:
   - More accurate identification of at-risk students
   - Better resource allocation for student support
   - Improved decision-making for educational interventions

Therefore, ADASYN should be implemented as the final oversampling technique for this educational dataset.

Reference

1. Realinho, V., Vieira Martins, M., Machado, J., & Baptista, L. (2021). Predict Students' Dropout and Academic Success Dataset. UCI Machine Learning Repository. https://doi.org/10.24432/C5MC89.
2. Martins, M. V., Tolledo, D., Machado, J., Baptista, L. M., & Realinho, V. (2021). Early prediction of student's performance in higher education: a case study. In *Trends and Applications in Information Systems and Technologies: Volume 1 9* (pp. 166-175). Springer International Publishing.