ppgecon\_ml\_projeto\_final\_v00.R

DELL

2025-01-31

# SETUP ----  
  
# Pacotes utilizados no modelo de previsão de inadimplência  
  
# dplyr: Manipulação eficiente de dados com funções do tidyverse  
library(dplyr)

## Warning: pacote 'dplyr' foi compilado no R versão 4.4.2

##   
## Anexando pacote: 'dplyr'

## Os seguintes objetos são mascarados por 'package:stats':  
##   
## filter, lag

## Os seguintes objetos são mascarados por 'package:base':  
##   
## intersect, setdiff, setequal, union

# ggplot2: Criação de gráficos avançados e customizáveis  
library(ggplot2)

## Warning: pacote 'ggplot2' foi compilado no R versão 4.4.2

# gridExtra: Combinação de múltiplos gráficos em um único layout  
library(gridExtra)

##   
## Anexando pacote: 'gridExtra'

## O seguinte objeto é mascarado por 'package:dplyr':  
##   
## combine

# recipes: Pré-processamento de dados, incluindo normalização, imputação e balanceamento  
library(recipes)

##   
## Anexando pacote: 'recipes'

## O seguinte objeto é mascarado por 'package:stats':  
##   
## step

# caret: Utilizado para particionar os dados, treinar modelos e calcular métricas de desempenho  
library(caret)

## Warning: pacote 'caret' foi compilado no R versão 4.4.2

## Carregando pacotes exigidos: lattice

# themis: Técnicas de balanceamento de classes, incluindo SMOTE  
library(themis)

## Warning: pacote 'themis' foi compilado no R versão 4.4.2

# class: Implementação do algoritmo KNN (K-Nearest Neighbors)  
library(class)  
  
# randomForest: Implementação do algoritmo de Random Forest para classificação e regressão  
library(randomForest)

## randomForest 4.7-1.2

## Type rfNews() to see new features/changes/bug fixes.

##   
## Anexando pacote: 'randomForest'

## O seguinte objeto é mascarado por 'package:gridExtra':  
##   
## combine

## O seguinte objeto é mascarado por 'package:ggplot2':  
##   
## margin

## O seguinte objeto é mascarado por 'package:dplyr':  
##   
## combine

# xgboost: Implementação otimizada de Gradient Boosting para aprendizado supervisionado  
library(xgboost)

## Warning: pacote 'xgboost' foi compilado no R versão 4.4.2

##   
## Anexando pacote: 'xgboost'

## O seguinte objeto é mascarado por 'package:dplyr':  
##   
## slice

# ELT ----  
# Carregar os dados (substituir pelo caminho correto)  
data <- read.csv2("C:/Users/DELL/OneDrive/R/Rprojetos/ufpr\_ppgecon/mineracao\_dados/projeto\_final/data/credit\_data.csv")  
  
  
# Remover a coluna ID, pois não é relevante para a modelagem  
data <- data %>% select(-ID)  
  
# Renomear a coluna alvo corretamente  
colnames(data)[colnames(data) == "default.payment.next.month"] <- "Y"  
  
# Converter a variável Y para fator  
data$Y <- as.factor(data$Y)  
  
# Verificar a estrutura corrigida dos dados  
str(data)

## 'data.frame': 30000 obs. of 24 variables:  
## $ LIMIT\_BAL: int 20000 120000 90000 50000 50000 50000 500000 100000 140000 20000 ...  
## $ SEX : int 2 2 2 2 1 1 1 2 2 1 ...  
## $ EDUCATION: int 2 2 2 2 2 1 1 2 3 3 ...  
## $ MARRIAGE : int 1 2 2 1 1 2 2 2 1 2 ...  
## $ AGE : int 24 26 34 37 57 37 29 23 28 35 ...  
## $ PAY\_0 : int 2 -1 0 0 -1 0 0 0 0 -2 ...  
## $ PAY\_2 : int 2 2 0 0 0 0 0 -1 0 -2 ...  
## $ PAY\_3 : int -1 0 0 0 -1 0 0 -1 2 -2 ...  
## $ PAY\_4 : int -1 0 0 0 0 0 0 0 0 -2 ...  
## $ PAY\_5 : int -2 0 0 0 0 0 0 0 0 -1 ...  
## $ PAY\_6 : int -2 2 0 0 0 0 0 -1 0 -1 ...  
## $ BILL\_AMT1: int 3913 2682 29239 46990 8617 64400 367965 11876 11285 0 ...  
## $ BILL\_AMT2: int 3102 1725 14027 48233 5670 57069 412023 380 14096 0 ...  
## $ BILL\_AMT3: int 689 2682 13559 49291 35835 57608 445007 601 12108 0 ...  
## $ BILL\_AMT4: int 0 3272 14331 28314 20940 19394 542653 221 12211 0 ...  
## $ BILL\_AMT5: int 0 3455 14948 28959 19146 19619 483003 -159 11793 13007 ...  
## $ BILL\_AMT6: int 0 3261 15549 29547 19131 20024 473944 567 3719 13912 ...  
## $ PAY\_AMT1 : int 0 0 1518 2000 2000 2500 55000 380 3329 0 ...  
## $ PAY\_AMT2 : int 689 1000 1500 2019 36681 1815 40000 601 0 0 ...  
## $ PAY\_AMT3 : int 0 1000 1000 1200 10000 657 38000 0 432 0 ...  
## $ PAY\_AMT4 : int 0 1000 1000 1100 9000 1000 20239 581 1000 13007 ...  
## $ PAY\_AMT5 : int 0 0 1000 1069 689 1000 13750 1687 1000 1122 ...  
## $ PAY\_AMT6 : int 0 2000 5000 1000 679 800 13770 1542 1000 0 ...  
## $ Y : Factor w/ 2 levels "0","1": 2 2 1 1 1 1 1 1 1 1 ...

# Separar em treino (70%) e teste (30%)  
set.seed(123)  
index <- createDataPartition(data$Y, p = 0.7, list = FALSE)  
train\_data <- data[index, ]  
test\_data <- data[-index, ]  
  
# Normalizar os dados (exceto a variável resposta)  
norm\_params <- preProcess(train\_data[,-ncol(train\_data)], method = c("range"))  
train\_data[,-ncol(train\_data)] <- predict(norm\_params, train\_data[,-ncol(train\_data)])  
test\_data[,-ncol(train\_data)] <- predict(norm\_params, test\_data[,-ncol(train\_data)])  
  
# Definir os preditores e a variável resposta  
train\_x <- train\_data[, -ncol(train\_data)]  
train\_y <- train\_data$Y  
test\_x <- test\_data[, -ncol(test\_data)]  
test\_y <- test\_data$Y  
  
  
# MODELOS ----  
## KNN ----  
# Treinar modelo KNN (k=5)  
k\_value <- 5  
knn\_model <- knn(train = train\_x, test = test\_x, cl = train\_y, k = k\_value)  
  
# Avaliação do modelo  
conf\_matrix <- confusionMatrix(knn\_model, test\_y)  
print(conf\_matrix)

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction 0 1  
## 0 6407 1309  
## 1 602 681  
##   
## Accuracy : 0.7876   
## 95% CI : (0.779, 0.7961)  
## No Information Rate : 0.7789   
## P-Value [Acc > NIR] : 0.02269   
##   
## Kappa : 0.2937   
##   
## Mcnemar's Test P-Value : < 2e-16   
##   
## Sensitivity : 0.9141   
## Specificity : 0.3422   
## Pos Pred Value : 0.8304   
## Neg Pred Value : 0.5308   
## Prevalence : 0.7789   
## Detection Rate : 0.7120   
## Detection Prevalence : 0.8574   
## Balanced Accuracy : 0.6282   
##   
## 'Positive' Class : 0   
##

# Métricas de performance  
accuracy <- conf\_matrix$overall["Accuracy"]  
precision <- conf\_matrix$byClass["Precision"]  
recall <- conf\_matrix$byClass["Recall"]  
F1\_score <- 2 \* (precision \* recall) / (precision + recall)  
  
cat("\nDesempenho do modelo KNN:\n")

##   
## Desempenho do modelo KNN:

cat("Acurácia:", accuracy, "\n")

## Acurácia: 0.7876431

cat("Precisão:", precision, "\n")

## Precisão: 0.8303525

cat("Recall:", recall, "\n")

## Recall: 0.9141104

cat("F1-score:", F1\_score, "\n")

## F1-score: 0.8702207

## RANDOM FOREST ----  
  
# Treinar modelo Random Forest  
set.seed(123)  
rf\_model.1 <- randomForest(Y ~ .,   
 data = train\_data,   
 ntree = 100,   
 mtry = sqrt(ncol(train\_data) - 1),   
 importance = TRUE)  
  
# Fazer previsões  
rf\_pred.1 <- predict(rf\_model.1, test\_data, type = "class")  
  
# Avaliação do modelo  
conf\_matrix.1 <- confusionMatrix(rf\_pred.1, test\_data$Y)  
print(conf\_matrix.1)

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction 0 1  
## 0 6596 1253  
## 1 413 737  
##   
## Accuracy : 0.8149   
## 95% CI : (0.8067, 0.8228)  
## No Information Rate : 0.7789   
## P-Value [Acc > NIR] : < 2.2e-16   
##   
## Kappa : 0.3669   
##   
## Mcnemar's Test P-Value : < 2.2e-16   
##   
## Sensitivity : 0.9411   
## Specificity : 0.3704   
## Pos Pred Value : 0.8404   
## Neg Pred Value : 0.6409   
## Prevalence : 0.7789   
## Detection Rate : 0.7330   
## Detection Prevalence : 0.8722   
## Balanced Accuracy : 0.6557   
##   
## 'Positive' Class : 0   
##

# Métricas de performance  
accuracy.1 <- conf\_matrix.1$overall["Accuracy"]  
precision.1 <- conf\_matrix.1$byClass["Precision"]  
recall.1 <- conf\_matrix.1$byClass["Recall"]  
F1\_score.1 <- 2 \* (precision \* recall) / (precision + recall)  
  
cat("\nDesempenho do modelo Random Forest:\n")

##   
## Desempenho do modelo Random Forest:

cat("Acurácia:", accuracy.1, "\n")

## Acurácia: 0.8148683

cat("Precisão:", precision.1, "\n")

## Precisão: 0.8403618

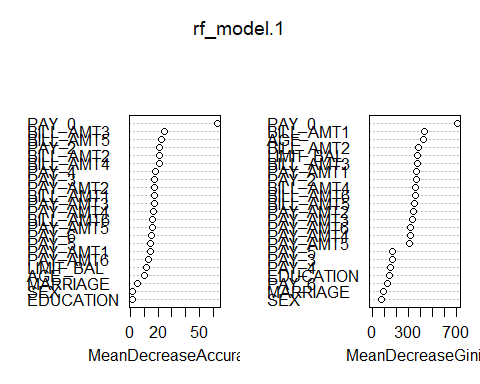
cat("Recall:", recall.1, "\n")

## Recall: 0.9410758

cat("F1-score:", F1\_score.1, "\n")

## F1-score: 0.8702207

# Importância das variáveis  
varImpPlot(rf\_model.1)



# Random Forest pode ser ajustado para dar mais peso para inadimplentes (classwt)  
rf\_model.2 <- randomForest(Y ~ .,   
 data = train\_data,   
 ntree = 100,   
 classwt = c(0.5, 1.5))  
  
# Fazer previsões  
rf\_pred.2 <- predict(rf\_model.2, test\_data, type = "class")  
  
# Avaliação do modelo  
conf\_matrix.2 <- confusionMatrix(rf\_pred.2, test\_data$Y)  
print(conf\_matrix.2)

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction 0 1  
## 0 6504 1259  
## 1 505 731  
##   
## Accuracy : 0.804   
## 95% CI : (0.7956, 0.8121)  
## No Information Rate : 0.7789   
## P-Value [Acc > NIR] : 3.171e-09   
##   
## Kappa : 0.3416   
##   
## Mcnemar's Test P-Value : < 2.2e-16   
##   
## Sensitivity : 0.9279   
## Specificity : 0.3673   
## Pos Pred Value : 0.8378   
## Neg Pred Value : 0.5914   
## Prevalence : 0.7789   
## Detection Rate : 0.7227   
## Detection Prevalence : 0.8627   
## Balanced Accuracy : 0.6476   
##   
## 'Positive' Class : 0   
##

# Métricas de performance  
accuracy.2 <- conf\_matrix.2$overall["Accuracy"]  
precision.2 <- conf\_matrix.2$byClass["Precision"]  
recall.2 <- conf\_matrix.2$byClass["Recall"]  
F1\_score.2 <- 2 \* (precision \* recall) / (precision + recall)  
  
cat("\nDesempenho do modelo Random Forest:\n")

##   
## Desempenho do modelo Random Forest:

cat("Acurácia:", accuracy.2, "\n")

## Acurácia: 0.8039782

cat("Precisão:", precision.2, "\n")

## Precisão: 0.8378204

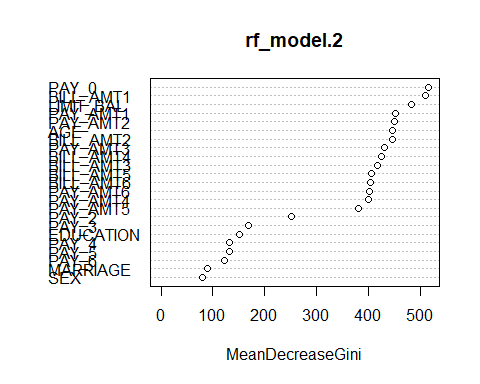
cat("Recall:", recall.2, "\n")

## Recall: 0.9279498

cat("F1-score:", F1\_score.2, "\n")

## F1-score: 0.8702207

# Importância das variáveis  
varImpPlot(rf\_model.2)



# Usar Oversampling/SMOTE  
# Como há mais adimplentes do que inadimplentes, técnicas como SMOTE podem ajudar a equilibrar as classes.  
  
train\_data.3 <- train\_data %>%   
 recipe(Y ~ .) %>% # pré-processamento para a variável resposta `Y` com todas as outras variáveis como preditoras  
 step\_smote(Y) %>% # SMOTE para balancear as classes da variável `Y` (gera novos exemplos sintéticos da classe minoritária)  
 prep() %>% # Prepara a receita, estimando os parâmetros necessários para a transformação  
 juice() # Extrai os dados transformados para serem usados no modelo  
  
  
  
# Aumentar ntree para 500 ou 1000 pode melhorar o desempenho  
rf\_model.3 <- randomForest(Y ~ .,   
 data = train\_data.3,   
 ntree = 1000)  
  
  
# Fazer previsões  
rf\_pred.3 <- predict(rf\_model.3,   
 test\_data,   
 type = "class")  
  
# Avaliação do modelo  
conf\_matrix.3 <- confusionMatrix(rf\_pred.3, test\_data$Y)  
print(conf\_matrix.3)

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction 0 1  
## 0 6185 1025  
## 1 824 965  
##   
## Accuracy : 0.7945   
## 95% CI : (0.786, 0.8028)  
## No Information Rate : 0.7789   
## P-Value [Acc > NIR] : 0.0001606   
##   
## Kappa : 0.3811   
##   
## Mcnemar's Test P-Value : 3.301e-06   
##   
## Sensitivity : 0.8824   
## Specificity : 0.4849   
## Pos Pred Value : 0.8578   
## Neg Pred Value : 0.5394   
## Prevalence : 0.7789   
## Detection Rate : 0.6873   
## Detection Prevalence : 0.8012   
## Balanced Accuracy : 0.6837   
##   
## 'Positive' Class : 0   
##

# Métricas de performance  
accuracy.3 <- conf\_matrix.3$overall["Accuracy"]  
precision.3 <- conf\_matrix.3$byClass["Precision"]  
recall.3 <- conf\_matrix.3$byClass["Recall"]  
F1\_score.3 <- 2 \* (precision \* recall) / (precision + recall)  
  
cat("\nDesempenho do modelo Random Forest:\n")

##   
## Desempenho do modelo Random Forest:

cat("Acurácia:", accuracy.3, "\n")

## Acurácia: 0.7945327

cat("Precisão:", precision.3, "\n")

## Precisão: 0.8578363

cat("Recall:", recall.3, "\n")

## Recall: 0.8824369

cat("F1-score:", F1\_score.3, "\n")

## F1-score: 0.8702207

# Importância das variáveis  
varImpPlot(rf\_model.3)

