

STAT 4620 Final Project

2025-12-14

```
# Import Libraries
library(tidyverse)

## -- Attaching core tidyverse packages ----- tidyverse 2.0.0 --
## v dplyr     1.1.4     v readr     2.1.6
## v forcats   1.0.1     v stringr   1.6.0
## v ggplot2   4.0.1     v tibble    3.3.0
## v lubridate 1.9.4     v tidyr    1.3.1
## v purrr    1.2.0

## -- 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

library(rpart)
library(rpart.plot)
library(caret)

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

library(pROC)

## Type 'citation("pROC")' for a citation.
##
## Attaching package: 'pROC'
##
## The following objects are masked from 'package:stats':
##
##     cov, smooth, var

library(class)
library(pls)

## 
## Attaching package: 'pls'
```

```

##  

## The following object is masked from 'package:caret':  

##  

##      R2  

##  

## The following object is masked from 'package:stats':  

##  

##      loadings  

library(glmnet)  

## Loading required package: Matrix  

##  

## Attaching package: 'Matrix'  

##  

## The following objects are masked from 'package:tidyverse':  

##  

##      expand, pack, unpack  

##  

## Loaded glmnet 4.1-10  

# Setup and Import Data  

set.seed(4620)  

train <- read.csv("data/train.csv")  

test <- read.csv("data/test.csv")

```

EDA

Model Analysis

Classification/Regression Tree

```

# Setup Classification Data  

train_class <- train  

test_class <- test  

train_class$Bankrupt. <- factor(train_class$Bankrupt.)  

test_class$Bankrupt. <- factor(test_class$Bankrupt.)  

predictor_cols <- setdiff(colnames(train_class), "Bankrupt.")  

# Apply CART Model  

cart_full <- rpart(  

  formula = as.formula(paste("Bankrupt. ~", paste(predictor_cols, collapse = " + "))),  

  data = train_class,  

  method = "class",  

  cp = 0.0001,  

  minsplit = 5
)

```

```

# Examine Output
printcp(cart_full)

## 
## Classification tree:
## rpart(formula = as.formula(paste("Bankrupt. ~", paste(predictor_cols,
##   collapse = " + "))), data = train_class, method = "class",
##   cp = 1e-04, minsplit = 5)
##
## Variables actually used in tree construction:
## [1] Accounts.Receivable.Turnover
## [2] After.tax.Net.Profit.Growth.Rate
## [3] Allocation.rate.per.person
## [4] Borrowing.dependency
## [5] Cash.Flow.to.Sales
## [6] Current.Asset.Turnover.Rate
## [7] Current.Liabilities.Liability
## [8] Current.Ratio
## [9] Fixed.Assets.to.Assets
## [10] Fixed.Assets.Turnover.Frequency
## [11] Interest.bearing.debt.interest.rate
## [12] Interest.Coverage.Ratio..Interest.expense.to.EBIT.
## [13] Interest.Expense.Ratio
## [14] Inventory.Turnover.Rate..times.
## [15] Net.Income.to.Stockholder.s.Equity
## [16] Net.Income.to.Total.Assets
## [17] Net.Value.Growth.Rate
## [18] Net.Value.Per.Share..B.
## [19] No.credit.Interval
## [20] Non.industry.income.and.expenditure.revenue
## [21] Operating.Profit.Growth.Rate
## [22] Operating.profit.per.person
## [23] Operating.Profit.Rate
## [24] Per.Share.Net.profit.before.tax..Yuan...
## [25] Realized.Sales.Gross.Profit.Growth.Rate
## [26] Research.and.development.expense.rate
## [27] Revenue.per.person
## [28] Revenue.Per.Share..Yuan...
## [29] ROA.A..before.interest.and...after.tax
## [30] ROA.B..before.interest.and.depreciation.after.tax
## [31] ROA.C..before.interest.and.depreciation.before.interest
## [32] Total.Asset.Return.Growth.Rate.Ratio
## [33] Total.debt.Total.net.worth
## [34] Working.capitcal.Turnover.Rate
## [35] X
##
## Root node error: 140/4773 = 0.029332
##
## n= 4773
##
##          CP nsplit rel error  xerror      xstd
## 1  0.0428571      0    1.00000 1.00000 0.083267
## 2  0.0357143      3    0.87143 0.99286 0.082978

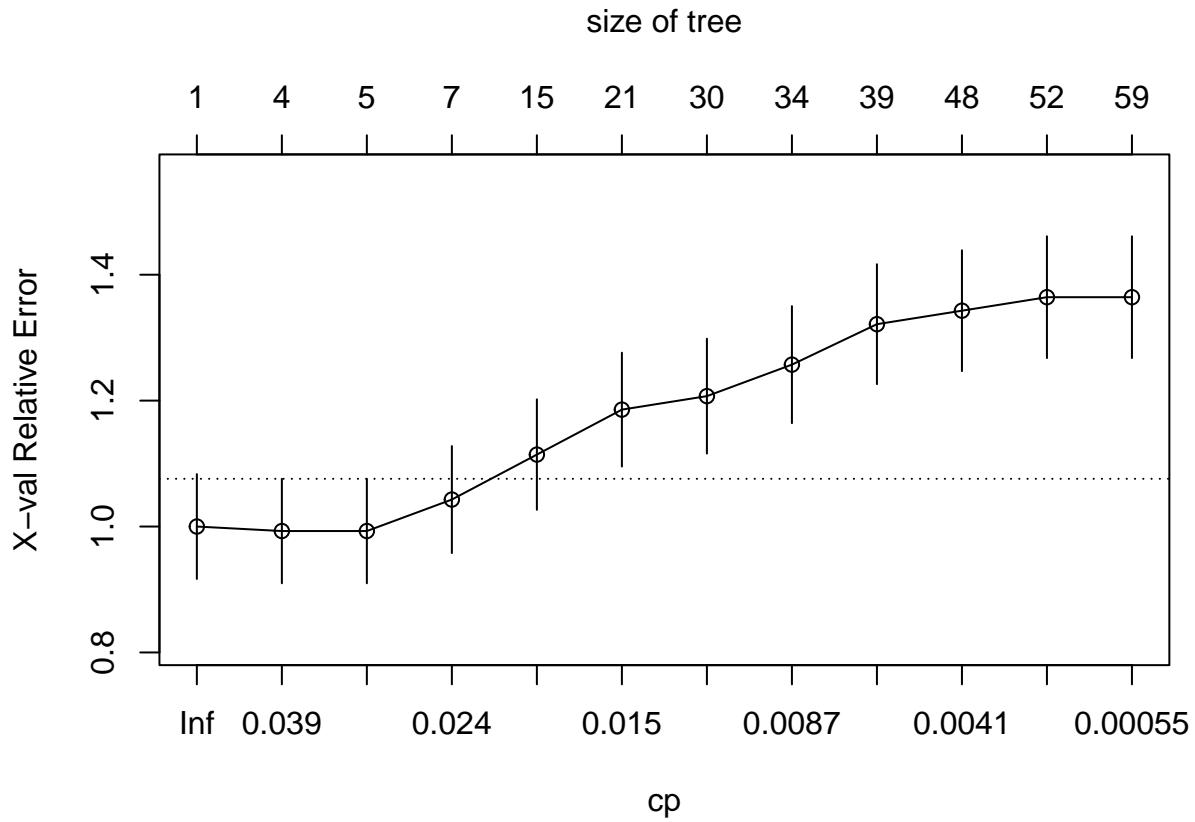
```

```

## 3 0.0321429      4 0.83571 0.99286 0.082978
## 4 0.0178571      6 0.77143 1.04286 0.084977
## 5 0.0166667      14 0.60000 1.11429 0.087744
## 6 0.0142857      20 0.50000 1.18571 0.090415
## 7 0.0107143      29 0.37143 1.20714 0.091198
## 8 0.0071429      33 0.32857 1.25714 0.092997
## 9 0.0047619      38 0.29286 1.32143 0.095252
## 10 0.0035714     47 0.25000 1.34286 0.095990
## 11 0.0030612     51 0.23571 1.36429 0.096721
## 12 0.0001000     58 0.21429 1.36429 0.096721

```

```
plotcp(cart_full)
```



```

# Find Optimal Value For Pruning
cptable <- cart_full$cptable
min_row <- which.min(cptable[, "xerror"])

xerr_min <- cptable[min_row, "xerror"]
xerr_SE <- cptable[min_row, "xstd"]

threshold <- xerr_min + xerr_SE
opt_row <- which(cptable[, "xerror"] <= threshold)[1]
opt_cp <- cptable[opt_row, "CP"]

```

```

# Prune Tree
cart_pruned <- prune(cart_full, cp = opt_cp)

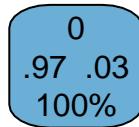
cart_pruned$variable.importance

## logical(0)

# Plot Updated Tree
rpart.plot(
  cart_pruned,
  type = 2,
  fallen.leaves = TRUE,
  extra = 104,
  main = "Pruned CART Tree"
)

```

Pruned CART Tree



```

# Predict on Test Data
pred_probs_class <- predict(cart_pruned, newdata = test, type = "prob")[,2]
pred_class_class <- factor(ifelse(pred_probs_class > 0.5, "1", "0"), levels = c("0","1"))

conf_mat_class <- confusionMatrix(pred_class_class, test_class$Bankrupt.)
conf_mat_class

## Confusion Matrix and Statistics

```

```

##          Reference
## Prediction 0     1
##           0 1966   80
##           1     0     0
##
##          Accuracy : 0.9609
##          95% CI  : (0.9516, 0.9689)
##          No Information Rate : 0.9609
##          P-Value [Acc > NIR] : 0.5297
##
##          Kappa : 0
##
##  Mcnemar's Test P-Value : <2e-16
##
##          Sensitivity : 1.0000
##          Specificity  : 0.0000
##          Pos Pred Value : 0.9609
##          Neg Pred Value :      NaN
##          Prevalence    : 0.9609
##          Detection Rate : 0.9609
##          Detection Prevalence : 1.0000
##          Balanced Accuracy : 0.5000
##
##          'Positive' Class : 0
##

```

```

accuracy_class <- conf_mat_class$overall["Accuracy"]
sensitivity_class <- conf_mat_class$byClass["Sensitivity"]
specificity_class <- conf_mat_class$byClass["Specificity"]

```

```
accuracy_class
```

```

##  Accuracy
## 0.9608993

```

```
sensitivity_class
```

```

## Sensitivity
## 1

```

```
specificity_class
```

```

## Specificity
## 0

```

KNN

```

# Setup KNN Data
train_knn <- train
test_knn <- test

train_knn$Bankrupt. <- factor(train_knn$Bankrupt.)
test_knn$Bankrupt. <- factor(test_knn$Bankrupt.)

predictor_cols <- setdiff(colnames(train), c("Bankrupt.", "Unnamed..0"))

# Standardize Predictors
preproc <- preProcess(train[, predictor_cols], method = c("center", "scale"))

## Warning in preProcess.default(train[, predictor_cols], method = c("center", :
## These variables have zero variances: Net.Income.Flag

train_scaled <- predict(preproc, train_knn[, predictor_cols])
test_scaled <- predict(preproc, test_knn[, predictor_cols])

# Find Best K Value
K_values <- seq(1, 51, by = 2)
acc_results <- c()

for (k in K_values) {
  knn_pred <- knn(
    train = train_scaled,
    test = test_scaled,
    cl = train_knn$Bankrupt.,
    k = k,
    prob = TRUE
  )

  acc <- mean(knn_pred == test_knn$Bankrupt.)
  acc_results <- c(acc_results, acc)
}

best_k <- K_values[which.max(acc_results)]
best_k

## [1] 11

# Fit KNN Model and
knn_pred <- knn(
  train = train_scaled,
  test = test_scaled,
  cl = train_knn$Bankrupt.,
  k = best_k,
  prob = TRUE
)

conf_mat_knn <- confusionMatrix(knn_pred, test_knn$Bankrupt.)
conf_mat_knn

```

```

## Confusion Matrix and Statistics
##
##             Reference
## Prediction   0      1
##           0 1964    74
##           1      2     6
##
##                   Accuracy : 0.9629
##                   95% CI : (0.9537, 0.9706)
##       No Information Rate : 0.9609
##       P-Value [Acc > NIR] : 0.3504
##
##                   Kappa : 0.1302
##
## Mcnemar's Test P-Value : 3.816e-16
##
##                   Sensitivity : 0.9990
##                   Specificity : 0.0750
##       Pos Pred Value : 0.9637
##       Neg Pred Value : 0.7500
##       Prevalence : 0.9609
##       Detection Rate : 0.9599
## Detection Prevalence : 0.9961
##       Balanced Accuracy : 0.5370
##
##       'Positive' Class : 0
##

accuracy_knn <- conf_mat_knn$overall[["Accuracy"]]
sensitivity_knn <- conf_mat_knn$byClass[["Sensitivity"]]
specificity_knn <- conf_mat_knn$byClass[["Specificity"]]

accuracy_knn

## Accuracy
## 0.9628543

sensitivity_knn

## Sensitivity
## 0.9989827

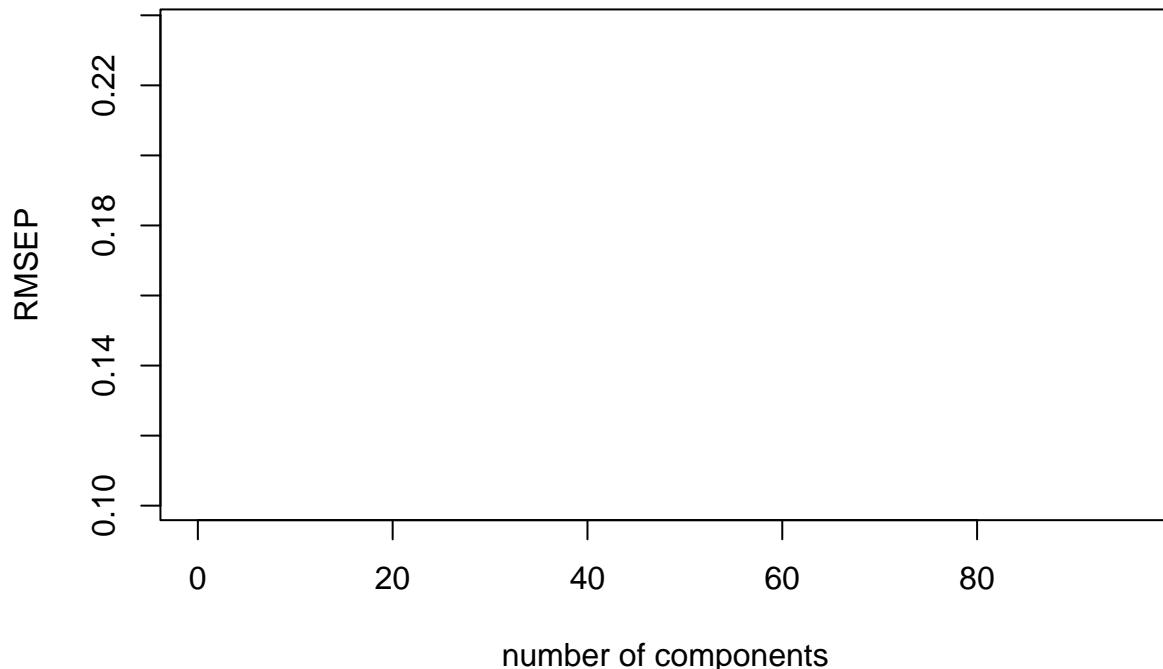
specificity_knn

## Specificity
## 0.075

```

PLS

PLS CV – RMSEP vs Components



```
opt_M <- which.min(RMSEP(pls_fit)$val[,1,])
opt_M

## (Intercept)
##           1

# Predict on Test Data With Optimal Components
pls_pred_probs <- predict(pls_fit, newdata = as.matrix(X_test_pls), ncomp = opt_M)

pls_pred_probs <-pls_pred_probs[,1,1]

pls_pred_class <- ifelse(pls_pred_probs > .5, "1", "0")
pls_pred_class<- factor(pls_pred_class, levels = c("0", "1"))

# Evaluate Performance
conf_mat_pls <- confusionMatrix(pls_pred_class, test_pls$Bankrupt.)
conf_mat_pls

## Confusion Matrix and Statistics
##
##             Reference
## Prediction 0 1
##           0 0 0
##           1 0 0
```

```

##                               Accuracy : NaN
##                               95% CI : (NA, NA)
##      No Information Rate : NA
##      P-Value [Acc > NIR] : NA
##
##                               Kappa : NaN
##
##      Mcnemar's Test P-Value : NA
##
##      Sensitivity : NA
##      Specificity : NA
##      Pos Pred Value : NA
##      Neg Pred Value : NA
##      Prevalence : NaN
##      Detection Rate : NaN
##      Detection Prevalence : NaN
##      Balanced Accuracy : NA
##
##      'Positive' Class : 0
##

accuracy_pls <- conf_mat_pls$overall["Accuracy"]
sensitivity_pls <- conf_mat_pls$byClass["Sensitivity"]
specificity_pls <- conf_mat_pls$byClass["Specificity"]

accuracy_pls

## Accuracy
##      NaN

sensitivity_pls

## Sensitivity
##      NA

specificity_pls

## Specificity
##      NA

```

Ridge Regression

```

# Setup Ridge Data
train_ridge <- train
test_ridge <- test

y_train_ridge <- as.numeric(as.character(train_ridge$Bankrupt.))
y_test_ridge <- as.numeric(as.character(test_ridge$Bankrupt.))

```

```

# Cleanup Columns
predictor_cols_ridge <- setdiff(colnames(train_ridge), c("Bankrupt.", "Unnamed..0"))

X_train_ridge <- as.matrix(train_ridge[, predictor_cols_ridge])
X_test_ridge <- as.matrix(test_ridge[, predictor_cols_ridge])

# Fit Ridge Logistic Regression
ridge_cv <- cv.glmnet(
  x = X_train_ridge,
  y = y_train_ridge,
  alpha = 0,
  family = "binomial",
  standardize = TRUE,
  type.measure = "deviance"
)

best_lambda <- ridge_cv$lambda.min
best_lambda

## [1] 0.03472234

# Fit Final Ridge Model
ridge_fit <- glmnet(
  x = X_train_ridge,
  y = y_train_ridge,
  alpha = 0,
  family = "binomial",
  lambda = best_lambda,
  standardize = TRUE
)

## Warning: from glmnet C++ code (error code -1); Convergence for 1th lambda value
## not reached after maxit=100000 iterations; solutions for larger lambdas
## returned

## Warning in getcoef(fit, nvars, nx, vnames): an empty model has been returned;
## probably a convergence issue

# Predict on Test Data
ridge_prob <- predict(ridge_fit, newx = X_test_ridge, type = "response")

ridge_pred <- ifelse(ridge_prob > 0.5, "1", "0")
ridge_pred <- factor(ridge_pred, levels = c("0", "1"))

# Evaluate Ridge Performance
y_test_factor <- factor(test_ridge$Bankrupt., levels = c("0", "1"))
conf_mat_ridge <- confusionMatrix(ridge_pred, y_test_factor)
conf_mat_ridge

## Confusion Matrix and Statistics
##

```

```

##             Reference
## Prediction    0     1
##            0 1966   80
##            1     0     0
##
##                  Accuracy : 0.9609
##                  95% CI : (0.9516, 0.9689)
##      No Information Rate : 0.9609
##      P-Value [Acc > NIR] : 0.5297
##
##                  Kappa : 0
##
## McNemar's Test P-Value : <2e-16
##
##                  Sensitivity : 1.0000
##                  Specificity : 0.0000
##      Pos Pred Value : 0.9609
##      Neg Pred Value :      NaN
##      Prevalence : 0.9609
##      Detection Rate : 0.9609
##      Detection Prevalence : 1.0000
##      Balanced Accuracy : 0.5000
##
##      'Positive' Class : 0
##

accuracy_ridge <- conf_mat_ridge$overall["Accuracy"]
sensitivity_ridge <- conf_mat_ridge$byClass["Sensitivity"]
specificity_ridge <- conf_mat_ridge$byClass["Specificity"]

accuracy_ridge

##  Accuracy
## 0.9608993

sensitivity_ridge

## Sensitivity
##           1

specificity_ridge

## Specificity
##           0

#=====
# Setup LASSO Data
#=====

train_lasso <- train
test_lasso <- test

```

```

y_train_lasso <- as.numeric(as.character(train_lasso$Bankrupt.))
y_test_lasso <- as.numeric(as.character(test_lasso$Bankrupt.))

predictor_cols_lasso <- setdiff(colnames(train_lasso), c("Bankrupt.", "Unnamed..0"))

X_train_lasso <- as.matrix(train_lasso[, predictor_cols_lasso])
X_test_lasso <- as.matrix(test_lasso[, predictor_cols_lasso])

#=====
# Fit LASSO Logistic Regression
#=====

lasso_cv <- cv.glmnet(
  x = X_train_lasso,
  y = y_train_lasso,
  alpha = 1,           # <-- LASSO
  family = "binomial",
  standardize = TRUE,
  type.measure = "deviance"
)

best_lambda_lasso <- lasso_cv$lambda.min
best_lambda_lasso

## [1] 0.005792036

#=====
# Fit Final LASSO Model
#=====

lasso_fit <- glmnet(
  x = X_train_lasso,
  y = y_train_lasso,
  alpha = 1,           # <-- LASSO
  family = "binomial",
  lambda = best_lambda_lasso,
  standardize = TRUE
)

## Warning: from glmnet C++ code (error code -1); Convergence for 1th lambda value
## not reached after maxit=100000 iterations; solutions for larger lambdas
## returned

## Warning in getcoef(fit, nvars, nx, vnames): an empty model has been returned;
## probably a convergence issue

#=====
# Predict on Test Data
#=====

lasso_prob <- predict(lasso_fit, newx = X_test_lasso, type = "response")

```

```

lasso_pred <- ifelse(lasso_prob > 0.5, "1", "0")
lasso_pred <- factor(lasso_pred, levels = c("0", "1"))

#=====
# Evaluate LASSO Performance
#=====

y_test_factor_lasso <- factor(test_lasso$Bankrupt., levels = c("0", "1"))

conf_mat_lasso <- confusionMatrix(lasso_pred, y_test_factor_lasso)
conf_mat_lasso

## Confusion Matrix and Statistics
##
##          Reference
## Prediction      0      1
##           0 1966    80
##           1      0      0
##
##          Accuracy : 0.9609
##                 95% CI : (0.9516, 0.9689)
##     No Information Rate : 0.9609
##     P-Value [Acc > NIR] : 0.5297
##
##          Kappa : 0
##
## McNemar's Test P-Value : <2e-16
##
##          Sensitivity : 1.0000
##          Specificity : 0.0000
##     Pos Pred Value : 0.9609
##     Neg Pred Value :      NaN
##          Prevalence : 0.9609
##     Detection Rate : 0.9609
## Detection Prevalence : 1.0000
##     Balanced Accuracy : 0.5000
##
## 'Positive' Class : 0
##

accuracy_lasso     <- conf_mat_lasso$overall["Accuracy"]
sensitivity_lasso <- conf_mat_lasso$byClass["Sensitivity"]
specificity_lasso <- conf_mat_lasso$byClass["Specificity"]

accuracy_lasso

## Accuracy
## 0.9608993

```

```

sensitivity_lasso

## Sensitivity
##           1

specificity_lasso

## Specificity
##           0

library(randomForest)

## randomForest 4.7-1.2

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

## 
## Attaching package: 'randomForest'

## The following object is masked from 'package:dplyr':
## 
##     combine

## The following object is masked from 'package:ggplot2':
## 
##     margin

library(caret)

train_rf <- train
test_rf  <- test

train_rf$Bankrupt. <- factor(train_rf$Bankrupt., levels = c("0","1"))
test_rf$Bankrupt. <- factor(test_rf$Bankrupt., levels = c("0","1"))

predictor_cols <- setdiff(colnames(train_rf), c("Bankrupt.", "Unnamed..0"))

X_train <- train_rf[, predictor_cols]
y_train <- train_rf$Bankrupt.

X_test  <- test_rf[, predictor_cols]
y_test  <- test_rf$Bankrupt.

# --- FIXED: compute sampsize correctly ---
class_counts <- table(y_train)
minority_size <- min(class_counts)
sampsize_vec <- rep(minority_size, length(class_counts))

# Fit balanced RF
rf_model <- randomForest(

```

```

x = X_train,
y = y_train,
ntree = 500,
mtry = floor(sqrt(ncol(X_train))),
sampsize = sampsize_vec,    # <-- FIXED
importance = TRUE
)

# Predictions
rf_prob <- predict(rf_model, X_test, type = "prob")[,2]
rf_pred <- factor(ifelse(rf_prob > 0.5, "1", "0"), levels = c("0","1"))

conf_mat_rf <- confusionMatrix(rf_pred, y_test, positive = "1")
conf_mat_rf

## Confusion Matrix and Statistics
##
##             Reference
## Prediction      0      1
##           0 1769    20
##           1   197    60
##
##                 Accuracy : 0.8939
##                 95% CI : (0.8798, 0.907)
##     No Information Rate : 0.9609
##     P-Value [Acc > NIR] : 1
##
##                 Kappa : 0.3152
##
##     Mcnemar's Test P-Value : <2e-16
##
##                 Sensitivity : 0.75000
##                 Specificity : 0.89980
##     Pos Pred Value : 0.23346
##     Neg Pred Value : 0.98882
##                 Prevalence : 0.03910
##                 Detection Rate : 0.02933
##     Detection Prevalence : 0.12561
##                 Balanced Accuracy : 0.82490
##
##                 'Positive' Class : 1
##


library(randomForest)
library(caret)

#=====
# 1. Prepare Data
#=====

train_bag <- train
test_bag <- test

```

```

# Response must be factor
train_bag$Bankrupt. <- factor(train_bag$Bankrupt., levels = c("0","1"))
test_bag$Bankrupt. <- factor(test_bag$Bankrupt., levels = c("0","1"))

# Predictor columns
predictor_cols <- setdiff(colnames(train_bag), c("Bankrupt.", "Unnamed..0"))

X_train <- train_bag[, predictor_cols]
y_train <- train_bag$Bankrupt.

X_test <- test_bag[, predictor_cols]
y_test <- test_bag$Bankrupt.

#=====
# 2. Fit Bagging Model
#   Bagging = Random Forest with mtry = all predictors
#=====

set.seed(123)

bag_model <- randomForest(
  x = X_train,
  y = y_train,
  ntree = 500,                      # many trees -> stable
  mtry = ncol(X_train),              # <-- key for bagging
  importance = TRUE
)

#=====
# 3. Predict on Test Data
#=====

bag_prob <- predict(bag_model, X_test, type = "prob")[,2]

# threshold 0.5 (can adjust later)
bag_pred <- factor(ifelse(bag_prob > 0.5, "1", "0"), levels = c("0","1"))

#=====
# 4. Evaluate Model
#=====

conf_mat_bag <- confusionMatrix(
  bag_pred,
  y_test,
  positive = "1"
)

conf_mat_bag

## Confusion Matrix and Statistics
##
##             Reference
## Prediction      0      1

```

```

##          0 1959    60
##          1     7    20
##
##          Accuracy : 0.9673
##         95% CI : (0.9586, 0.9745)
##      No Information Rate : 0.9609
##      P-Value [Acc > NIR] : 0.07412
##
##          Kappa : 0.3612
##
##  Mcnemar's Test P-Value : 2.114e-10
##
##          Sensitivity : 0.250000
##          Specificity  : 0.996439
##      Pos Pred Value : 0.740741
##      Neg Pred Value : 0.970282
##          Prevalence  : 0.039101
##          Detection Rate : 0.009775
##      Detection Prevalence : 0.013196
##          Balanced Accuracy : 0.623220
##
##          'Positive' Class : 1
##


# Extract metrics
accuracy_bag      <- conf_mat_bag$overall["Accuracy"]
sensitivity_bag   <- conf_mat_bag$byClass["Sensitivity"]
specificity_bag   <- conf_mat_bag$byClass["Specificity"]

accuracy_bag

##  Accuracy
##  0.9672532

sensitivity_bag

##  Sensitivity
##  0.25

specificity_bag

##  Specificity
##  0.9964395

#####
# 5. Variable Importance Plot (optional)
#####

varImpPlot(bag_model, main = "Bagging Variable Importance")

```

Bagging Variable Importance

