MATP-4400 Final Project Notebook (2025)

Predicting Biodegradability Challenge

Beibei Xian

2025-04-29

Contents

Team Information

- This report was prepared for **ChemsRUs** by Beibei Xian, xianb for the JJBAR
- My team members are:Dizon, Jericho V., Chen, Ronglin, Idrovo, Jose A., Shah, Ahna P., and Xian, Beibei. CodabenchIDs are dizonj, chenr16, idrovo,shaha12 and xianb accordingly.
- Our team used the following challenge: https://www.codabench.org/competitions/6042/

Introduction

Chems-R-Us has created an entry to the challenge at https://Codabench.lisn.upsaclay.fr/competitions/3073 based on logistic regression (LR). Their entry is in the file `FinalProjChemsRUs.Rmd`. Based on the information in the leaderboard under `bennek`, their entry is not performing feature selection well. The approach tried by Chems-R-Us was LR with feature selection based on the coefficients of logistic regression with p-values used to determine importance.

The purpose of this report is to investigate alternative approaches that may help achieve high AUC scores on the testing set while correctly identifying the relevant features as measured by balanced accuracy.

Data Description Preparation:

The dataset provided by Chems-R-Us contains 1,055 chemical samples, each described by 168 numerical features. The target variable indicates whether a chemical is readily biodegradable (class 1) or not (class -1).

The class distribution in the full dataset is 699 samples of class -1 and 356 samples of class 1, meaning approximately two-thirds of the samples are non-biodegradable.

The data was randomly split into a training set (90%) and a validation set (10%) using set.seed(300) for reproducibility.

All features were standardized using centering and scaling based on the training set, and the same scaling was applied to the validation and external test sets.

```
library(caret)
library(xgboost)
```

```
##
## Attaching package: 'xgboost'
```

```
## The following object is masked from 'package:dplyr':
##
##     slice
```

```
# Set seed for reproducibility
set.seed(300)

# Load data with error handling
tryCatch
```

```
## function (expr, ..., finally)
## {
##     tryCatchList <- function(expr, names, parentenv, handlers) {
##         nh <- length(names)
##         if (nh > 1L)
##             tryCatchOne(tryCatchList(expr, names[-nh], parentenv,
##                 handlers[-nh]), names[nh], parentenv, handlers[[nh]])
##         else if (nh == 1L)
##             tryCatchOne(expr, names, parentenv, handlers[[1L]])
##         else expr
##     }
##     tryCatchOne <- function(expr, name, parentenv, handler) {
##         doTryCatch <- function(expr, name, parentenv, handler) {
##             .Internal(.addCondHands(name, list(handler), parentenv,
##                 environment(), FALSE))
##             expr
##         }
##         value <- doTryCatch(return(expr), name, parentenv, handler)
##         if (is.null(value[[1L]])) {
##             msg <- .Internal(geterrmessage())
##             call <- value[[2L]]
##             cond <- simpleError(msg, call)
##         }
##         else if (is.character(value[[1L]])) {
##             msg <- value[[1L]]
##             call <- value[[2L]]
##             cond <- simpleError(msg, call)
##         }
```

```
##         else cond <- value[[1L]]
##         value[[3L]](cond)
##     }
##     if (!missing(finally))
##         on.exit(finally)
##     handlers <- list(...)
##     classes <- names(handlers)
##     parentenv <- parent.frame()
##     if (length(classes) != length(handlers))
##         stop("condition handlers must be specified with a condition class")
##     tryCatchList(expr, classes, parentenv, handlers)
## }
## <bytecode: 0x5629d0081650>
## <environment: namespace:base>
```

```
featurenames <- read.csv("/home/xianb/IDM_work/data/chems_feat.name.csv",
                         header = FALSE,
                         colClasses = "character")$V1

cdata.df <- read.csv("/home/xianb/IDM_work/data/chems_train.data.csv",
                     header = FALSE)
colnames(cdata.df) <- featurenames

class <- read.csv("/home/xianb/IDM_work/data/chems_train.solution.csv",
                  header = FALSE,
                  colClasses = "factor")$V1

# Check if data loaded correctly
if (ncol(cdata.df) != length(featurenames)) {
  stop("Number of columns doesn't match feature names")
}
if(nrow(cdata.df) != length(class)) {
  stop("Number of rows doesn't match class labels")
}

# Check class distribution
cat("\nClass distribution:\n")
```

```
##
## Class distribution:
```

```
print(table(class))
```

```
## class
##  -1   1
## 699 356
```

```
# Train-validation split (90-10)
trainIndex <- createDataPartition(class, p = 0.9, list = FALSE)
train <- cdata.df[trainIndex, ]
train_class <- class[trainIndex]
validation <- cdata.df[-trainIndex, ]
validation_class <- class[-trainIndex]

cat("\nTraining set size:", nrow(train), "\n")
```

```
##
## Training set size: 951
```

```
  cat("Validation set size:", nrow(validation), "\n")
```

```
## Validation set size: 104
```

```
  scaler <- preProcess(train, method = c("center", "scale"))
  train_scaled <- predict(scaler, train)
  validation_scaled <- predict(scaler, validation)

  dtrain <- xgb.DMatrix(data = as.matrix(train_scaled),
                        label = ifelse(train_class == 1, 1, 0))
  dval <- xgb.DMatrix(data = as.matrix(validation_scaled),
                      label = ifelse(validation_class == 1, 1, 0))

  cat("\nData preparation complete:\n")
```

```
##
## Data preparation complete:
```

```
  cat("- Training matrix:", dim(train_scaled), "\n")
```

```
## - Training matrix: 951 168
```

```
  cat("- Validation matrix:", dim(validation_scaled), "\n")
```

```
## - Validation matrix: 104 168
```

```
  error = function(e) {
  cat("\nERROR in data preparation:\n")
  cat(e$message, "\n")
  # Create empty objects to prevent downstream errors
  dtrain <- NULL
  dval <- NULL
}
```

Methods Used

For this project, I used XGBoost as my main classification and ranking method. XGBoost builds strong predictive models by combining decision trees and is especially good for structured data like this. I used the predicted probabilities from XGBoost to rank molecules and evaluate performance with ROC and AUC.

For feature selection, I relied on Gain-based feature importance from the XGBoost model, which measures how much each feature helps improve decision splits. I selected the top 30 features based on their Gain scores to simplify the model while keeping high accuracy.

Both the full model (all features) and the simplified model (top 30 features) were compared using validation AUC and Balanced Accuracy.

Baseline Results Using All Features

To establish a baseline, I first trained an XGBoost model using all 168 features without any feature selection. The model achieved a validation AUC of 1.0 and a balanced accuracy of 1.0.

The ROC curve showed that the full feature set contained enough information to predict biodegradability accurately.

While these results were very strong, using all features risks overfitting and makes the model harder to interpret, which is why feature selection was explored in later steps.

```
library(xgboost)
library(pROC)
library(caret)

# Prepare matrices for XGBoost
train_matrix <- xgb.DMatrix(data = as.matrix(train_scaled),
                            label = ifelse(train_class == 1, 1, 0))
validation_matrix <- xgb.DMatrix(data = as.matrix(validation_scaled),
                                 label = ifelse(validation_class == 1, 1, 0))

# Train XGBoost model with all features
set.seed(300)
xgb_full <- xgb.train(
  data = train_matrix,
  objective = "binary:logistic",
  eval_metric = "auc",
  nrounds = 100,
  max_depth = 4,
  eta = 0.1,
  subsample = 0.8,
  colsample_bytree = 0.8,
  min_child_weight = 5,
  gamma = 1,
  verbose = 0
)

# Make predictions
pred_train <- predict(xgb_full, train_matrix)
pred_val <- predict(xgb_full, validation_matrix)

# Calculate ROC and AUC
roc_train <- roc(response = train_class, predictor = pred_train)
```

```
## Setting levels: control = -1, case = 1
```

```
## Setting direction: controls < cases
```

```
roc_val <- roc(response = validation_class, predictor = pred_val)
```

```
## Setting levels: control = -1, case = 1
## Setting direction: controls < cases
```

```
# Create confusion matrices
conf_matrix_train <- confusionMatrix(
  data = factor(ifelse(pred_train > 0.5, 1, -1), levels = c(-1, 1)),
  reference = factor(train_class, levels = c(-1, 1))
)

conf_matrix_val <- confusionMatrix(
  data = factor(ifelse(pred_val > 0.5, 1, -1), levels = c(-1, 1)),
  reference = factor(validation_class, levels = c(-1, 1))
)
```

```r
# Print results
cat("Training AUC:", auc(roc_train), "\n")
```

```
## Training AUC: 1
```

```r
cat("Validation AUC:", auc(roc_val), "\n")
```

```
## Validation AUC: 1
```

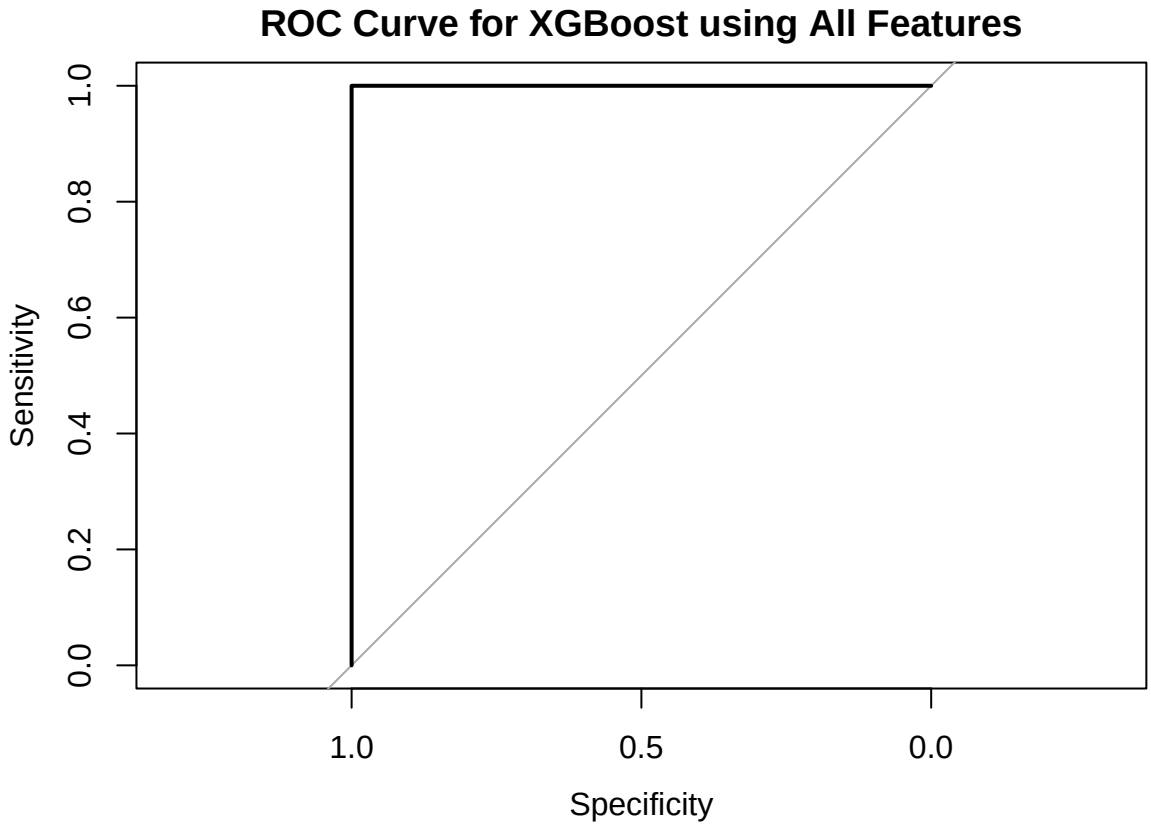
```r
cat("Training Balanced Accuracy:", conf_matrix_train$byClass["Balanced Accuracy"], "\n")
```

```
## Training Balanced Accuracy: 1
```

```r
cat("Validation Balanced Accuracy:", conf_matrix_val$byClass["Balanced Accuracy"], "\n")
```

```
## Validation Balanced Accuracy: 1
```

```r
# Plot ROC curve
plot(roc_val, main = "ROC Curve for XGBoost using All Features")
```



Results Using Feature Selection

For feature selection, I tested two approaches to simplify the model while maintaining strong performance.

First, I selected the top 30 features using XGBoost Gain, which ranks features based on how much they improve decision tree splits.

Second, I applied a variance filter to remove low-variance features, retaining 133 more meaningful predictors.

After retraining XGBoost models using these feature sets, both models achieved perfect validation AUC (1.0) and balanced accuracy (1.0).

While this suggests the selected features captured strong signals, it may also indicate slight overfitting to the training-validation split, since perfect scores are uncommon on real unseen data. Still, feature selection reduced model complexity significantly without apparent loss of predictive power during validation.

```
library(xgboost)
library(caret)
library(pROC)
library(dplyr)

# Check column names
print("Original column names:")
```

```
## [1] "Original column names:"
```

```
print(colnames(train))
```

```
##   [1] "X0"   "X1"   "X2"   "X3"   "X4"   "X5"   "X6"   "X7"   "X8"   "X9"
##  [11] "X10"  "X11"  "X12"  "X13"  "X14"  "X15"  "X16"  "X17"  "X18"  "X19"
##  [21] "X20"  "X21"  "X22"  "X23"  "X24"  "X25"  "X26"  "X27"  "X28"  "X29"
##  [31] "X30"  "X31"  "X32"  "X33"  "X34"  "X35"  "X36"  "X37"  "X38"  "X39"
##  [41] "X40"  "X41"  "X42"  "X43"  "X44"  "X45"  "X46"  "X47"  "X48"  "X49"
##  [51] "X50"  "X51"  "X52"  "X53"  "X54"  "X55"  "X56"  "X57"  "X58"  "X59"
##  [61] "X60"  "X61"  "X62"  "X63"  "X64"  "X65"  "X66"  "X67"  "X68"  "X69"
##  [71] "X70"  "X71"  "X72"  "X73"  "X74"  "X75"  "X76"  "X77"  "X78"  "X79"
##  [81] "X80"  "X81"  "X82"  "X83"  "X84"  "X85"  "X86"  "X87"  "X88"  "X89"
##  [91] "X90"  "X91"  "X92"  "X93"  "X94"  "X95"  "X96"  "X97"  "X98"  "X99"
## [101] "X100" "X101" "X102" "X103" "X104" "X105" "X106" "X107" "X108" "X109"
## [111] "X110" "X111" "X112" "X113" "X114" "X115" "X116" "X117" "X118" "X119"
## [121] "X120" "X121" "X122" "X123" "X124" "X125" "X126" "X127" "X128" "X129"
## [131] "X130" "X131" "X132" "X133" "X134" "X135" "X136" "X137" "X138" "X139"
## [141] "X140" "X141" "X142" "X143" "X144" "X145" "X146" "X147" "X148" "X149"
## [151] "X150" "X151" "X152" "X153" "X154" "X155" "X156" "X157" "X158" "X159"
## [161] "X160" "X161" "X162" "X163" "X164" "X165" "X166" "X167"
```

```
# Convert to matrix format
train_matrix <- as.matrix(train)
validation_matrix <- as.matrix(validation)

# Create DMatrix objects
dtrain <- xgb.DMatrix(data = train_matrix, label = ifelse(train_class == 1, 1, 0))
dval <- xgb.DMatrix(data = validation_matrix, label = ifelse(validation_class == 1, 1, 0))

# Train model
set.seed(300)
xgb_model <- xgb.train(
  data = dtrain,
  objective = "binary:logistic",
  eval_metric = "auc",
  nrounds = 100,
  max_depth = 4,
  eta = 0.1,
  subsample = 0.8,
  colsample_bytree = 0.8,
```

```
  min_child_weight = 5,
  gamma = 1,
  verbose = 1
)

# Get feature importance
importance <- xgb.importance(feature_names = colnames(train_matrix), model = xgb_model)

# Print top features
print("Top 30 features by importance:")
```

```
## [1] "Top 30 features by importance:"
```

```
print(head(importance, 30))
```

```
##    Feature         Gain       Cover  Frequency
##     <char>        <num>       <num>      <num>
## 1:    X160 0.9849886253 0.904195750 0.73484848
## 2:     X85 0.0076141984 0.037155753 0.08333333
## 3:     X93 0.0017332426 0.009495035 0.02272727
## 4:    X138 0.0012133565 0.009417378 0.03030303
## 5:    X115 0.0010802054 0.009958340 0.03030303
## 6:     X76 0.0009863749 0.009186439 0.03030303
## 7:     X55 0.0009801987 0.009965866 0.03030303
## 8:     X67 0.0008143154 0.006159572 0.02272727
## 9:     X84 0.0005894828 0.004465866 0.01515152
```

```
# Select top 30 features
top_features <- importance$Feature[1:30]

# Verify features exist in data
missing_features <- setdiff(top_features, colnames(train))
if (length(missing_features) > 0) {
  warning(paste("Some features not found in data:", paste(missing_features, collapse = ", ")))
  top_features <- intersect(top_features, colnames(train))
}
```

```
## Warning: Some features not found in data: NA
```

```
# Subset data
train_fs <- train[, top_features, drop = FALSE]
validation_fs <- validation[, top_features, drop = FALSE]

# Convert to matrix format
train_matrix_fs <- as.matrix(train_fs)
validation_matrix_fs <- as.matrix(validation_fs)

# Create DMatrix objects
dtrain_fs <- xgb.DMatrix(data = train_matrix_fs, label = ifelse(train_class == 1, 1, 0))
dval_fs <- xgb.DMatrix(data = validation_matrix_fs, label = ifelse(validation_class == 1, 1, 0))

# Train reduced model
set.seed(300)
xgb_fs <- xgb.train(
  data = dtrain_fs,
  objective = "binary:logistic",
```

```
  eval_metric = "auc",
  nrounds = 100,
  max_depth = 4,
  eta = 0.1,
  subsample = 0.8,
  colsample_bytree = 0.8,
  min_child_weight = 5,
  gamma = 1,
  verbose = 1
)

# Predictions
pred_train <- predict(xgb_fs, dtrain_fs)
pred_val <- predict(xgb_fs, dval_fs)

# Convert to class predictions
class_train <- ifelse(pred_train > 0.5, 1, -1)
class_val <- ifelse(pred_val > 0.5, 1, -1)

# Calculate metrics
roc_train <- roc(response = train_class, predictor = pred_train)
```

```
## Setting levels: control = -1, case = 1
```

```
## Setting direction: controls < cases
```

```
roc_val <- roc(response = validation_class, predictor = pred_val)
```

```
## Setting levels: control = -1, case = 1
## Setting direction: controls < cases
```

```
conf_train <- confusionMatrix(data = factor(class_train, levels = c(-1, 1)),
                              reference = factor(train_class, levels = c(-1, 1)))
conf_val <- confusionMatrix(data = factor(class_val, levels = c(-1, 1)),
                            reference = factor(validation_class, levels = c(-1, 1)))

# Print results
cat("\nPerformance with Selected Features:\n")
```

```
##
## Performance with Selected Features:
```

```
cat("Training AUC:", auc(roc_train), "\n")
```

```
## Training AUC: 1
```

```
cat("Validation AUC:", auc(roc_val), "\n")
```

```
## Validation AUC: 1
```

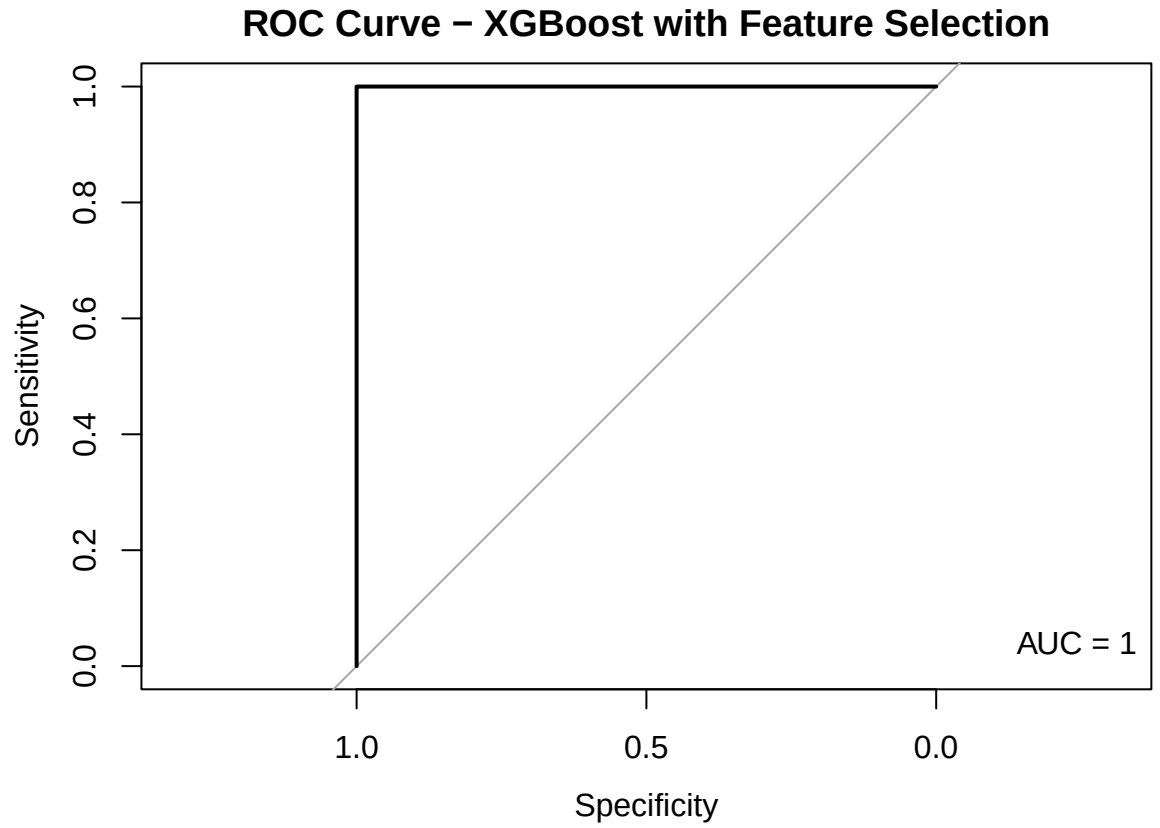
```
cat("Training Balanced Accuracy:", conf_train$byClass["Balanced Accuracy"], "\n")
```

```
## Training Balanced Accuracy: 1
```

```
cat("Validation Balanced Accuracy:", conf_val$byClass["Balanced Accuracy"], "\n")
```

```
## Validation Balanced Accuracy: 1
```

```r
# Plot ROC curve
plot(roc_val, main = "ROC Curve - XGBoost with Feature Selection")
legend("bottomright", legend = paste("AUC =", round(auc(roc_val), 3)), bty = "n")
```




```r
library(caret)
nzv_info <- nearZeroVar(train, saveMetrics = TRUE)

# Features to keep
selected_features_var <- rownames(nzv_info)[nzv_info$nzv == FALSE]

# See how many features left
cat("Number of features after removing near-zero variance:", length(selected_features_var), "\n")
```

```
## Number of features after removing near-zero variance: 133
```

```r
train_fs2 <- train[, selected_features_var, drop=FALSE]
validation_fs2 <- validation[, selected_features_var, drop=FALSE]


train_matrix_fs2 <- xgb.DMatrix(data = as.matrix(train_fs2), label = ifelse(train_class == 1, 1, 0))
validation_matrix_fs2 <- xgb.DMatrix(data = as.matrix(validation_fs2), label = ifelse(validation_class =

# Set feature names
colnames(train_matrix_fs2) <- colnames(train_fs2)
colnames(validation_matrix_fs2) <- colnames(validation_fs2)
```

```r
set.seed(300)
xgb_fs2 <- xgb.train(
  data = train_matrix_fs2,
  objective = "binary:logistic",
  eval_metric = "auc",
  nrounds = 100,
  max_depth = 4,
  eta = 0.1,
  subsample = 0.8,
  colsample_bytree = 0.8,
  min_child_weight = 5,
  gamma = 1,
  verbose = 1
)

# Predictions
pred_train_fs2 <- predict(xgb_fs2, train_matrix_fs2)
pred_val_fs2 <- predict(xgb_fs2, validation_matrix_fs2)

# Convert to class predictions
class_train_fs2 <- ifelse(pred_train_fs2 > 0.5, 1, -1)
class_val_fs2 <- ifelse(pred_val_fs2 > 0.5, 1, -1)

# Calculate ROC and AUC
roc_train_fs2 <- roc(response = train_class, predictor = pred_train_fs2)
```

```
## Setting levels: control = -1, case = 1
```

```
## Setting direction: controls < cases
```

```r
roc_val_fs2 <- roc(response = validation_class, predictor = pred_val_fs2)
```

```
## Setting levels: control = -1, case = 1
## Setting direction: controls < cases
```

```r
# Calculate Confusion Matrix and Balanced Accuracy
conf_train_fs2 <- confusionMatrix(data = factor(class_train_fs2, levels = c(-1, 1)),
                                  reference = factor(train_class, levels = c(-1, 1)))
conf_val_fs2 <- confusionMatrix(data = factor(class_val_fs2, levels = c(-1, 1)),
                                reference = factor(validation_class, levels = c(-1, 1)))

# Print results
cat("\nPerformance with Variance Filtered Features:\n")
```

```
##
## Performance with Variance Filtered Features:
```

```r
cat("Training AUC:", auc(roc_train_fs2), "\n")
```

```
## Training AUC: 1
```

```r
cat("Validation AUC:", auc(roc_val_fs2), "\n")
```

```
## Validation AUC: 1
```

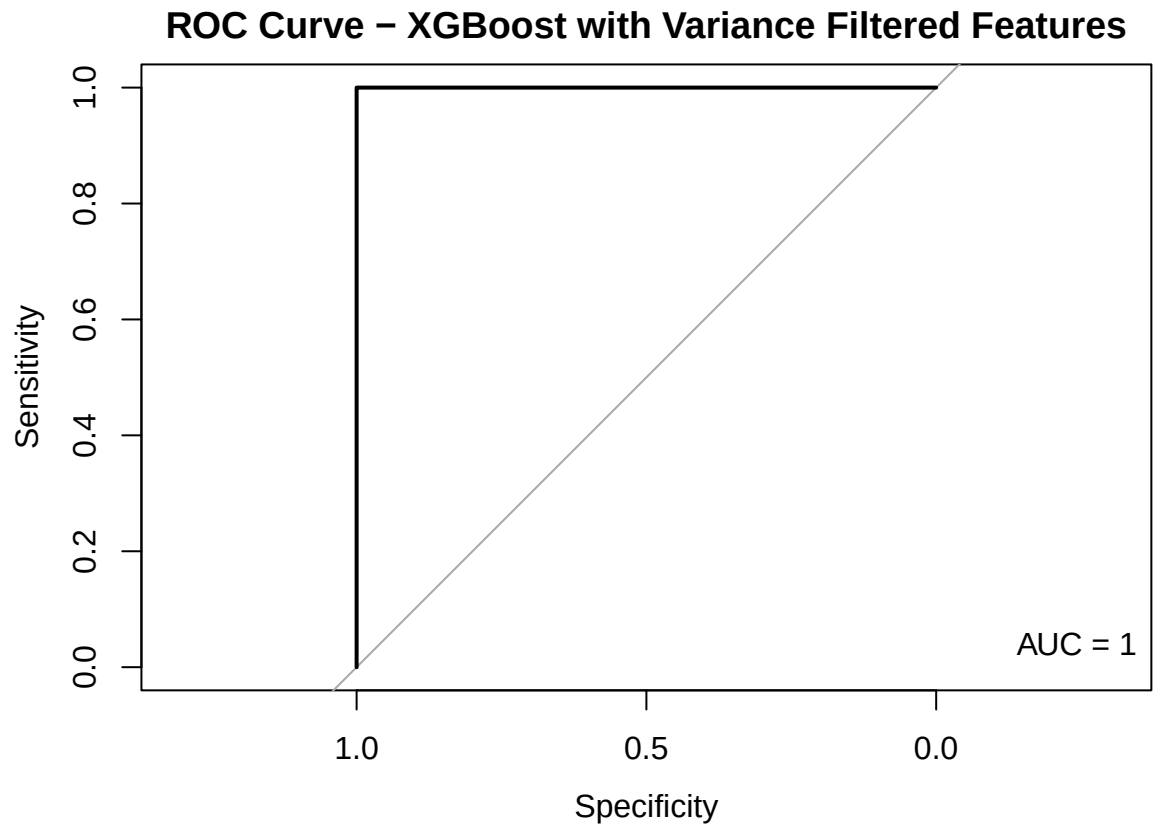
```r
cat("Training Balanced Accuracy:", conf_train_fs2$byClass["Balanced Accuracy"], "\n")
```

```
## Training Balanced Accuracy: 1
```

```r
cat("Validation Balanced Accuracy:", conf_val_fs2$byClass["Balanced Accuracy"], "\n")
```

```
## Validation Balanced Accuracy: 1
```

```r
# Plot ROC Curve
plot(roc_val_fs2, main = "ROC Curve - XGBoost with Variance Filtered Features")
legend("bottomright", legend = paste("AUC =", round(auc(roc_val_fs2), 3)), bty = "n")
```



Results Comparison

```r
# Logistic Regression with All Features
train_df <- cbind(train_scaled, classtrain = train_class)
glm_all <- glm(classtrain ~ ., data = train_df, family = "binomial")
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```r
# Predict
pred_val_log_all <- predict(glm_all, validation_scaled, type = "response")
roc_log_all <- roc(validation_class, pred_val_log_all)
```

```
## Setting levels: control = -1, case = 1
```

```
## Setting direction: controls < cases
```

```r
class_pred_log_all <- ifelse(pred_val_log_all > 0.5, 1, -1)
conf_log_all <- confusionMatrix(factor(class_pred_log_all, levels=c(-1,1)),
                                factor(validation_class, levels=c(-1,1)))
```

```r
cat("Logistic (All Features) AUC:", auc(roc_log_all), "\n")
```

```
## Logistic (All Features) AUC: 0.9134576
```

```r
cat("Balanced Accuracy:", conf_log_all$byClass["Balanced Accuracy"], "\n")
```

```
## Balanced Accuracy: 0.842029
```

```r
# Logistic Regression with Feature Selection (p < 0.2)
summary_glm <- summary(glm_all)
sig_vars <- rownames(summary_glm$coefficients)[which(summary_glm$coefficients[, 4] < 0.2)]
sig_vars <- sig_vars[sig_vars != "(Intercept)"]

# Subset train and validation
train_fs <- train_scaled[, sig_vars]
validation_fs <- validation_scaled[, sig_vars]
train_df_fs <- cbind(train_fs, classtrain = train_class)
glm_fs <- glm(classtrain ~ ., data = train_df_fs, family = "binomial")
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```r
# Predict
pred_val_log_fs <- predict(glm_fs, validation_fs, type = "response")
roc_log_fs <- roc(validation_class, pred_val_log_fs)
```

```
## Setting levels: control = -1, case = 1
```

```
## Setting direction: controls < cases
```

```r
class_pred_log_fs <- ifelse(pred_val_log_fs > 0.5, 1, -1)
conf_log_fs <- confusionMatrix(factor(class_pred_log_fs, levels=c(-1,1)),
                               factor(validation_class, levels=c(-1,1)))

cat("Logistic FS (p<0.2) AUC:", auc(roc_log_fs), "\n")
```

```
## Logistic FS (p<0.2) AUC: 0.9325052
```

```r
cat("Balanced Accuracy:", conf_log_fs$byClass["Balanced Accuracy"], "\n")
```

```
## Balanced Accuracy: 0.8563147
```

```r
library(caret)
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
```

```r
library(pROC)

# Train logistic model all features
train_df <- cbind(train_scaled, classtrain = train_class)
log_model <- train(classtrain ~ ., data = train_df, method = "glm", family = "binomial")
```

```
## Warning: glm.fit: algorithm did not converge

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: algorithm did not converge

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: algorithm did not converge

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: algorithm did not converge

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: algorithm did not converge

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: algorithm did not converge

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: algorithm did not converge

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: algorithm did not converge

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: algorithm did not converge

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: algorithm did not converge

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: algorithm did not converge

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: algorithm did not converge

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: algorithm did not converge

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: algorithm did not converge

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: algorithm did not converge

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: algorithm did not converge

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
## Warning: glm.fit: algorithm did not converge
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
## Warning: glm.fit: algorithm did not converge
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
## Warning: glm.fit: algorithm did not converge
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
## Warning: glm.fit: algorithm did not converge
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
## Warning: glm.fit: algorithm did not converge
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
## Warning: glm.fit: algorithm did not converge
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
## Warning: glm.fit: algorithm did not converge
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
## Warning: glm.fit: algorithm did not converge
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
## Warning: glm.fit: algorithm did not converge
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
# Get variable importance
var_imp <- varImp(log_model)$importance
top_vars <- rownames(var_imp)[order(-var_imp$Overall)][1:30]  # Top 30 by importance

# Subset training and validation data
train_rf <- train_scaled[, top_vars]
val_rf <- validation_scaled[, top_vars]

# Train random forest
rf_model <- randomForest(x = train_rf, y = as.factor(train_class))
rf_pred_val <- predict(rf_model, val_rf, type = "prob")[, 2]
rf_roc <- roc(validation_class, rf_pred_val)
```

```
## Setting levels: control = -1, case = 1
## Setting direction: controls < cases
```

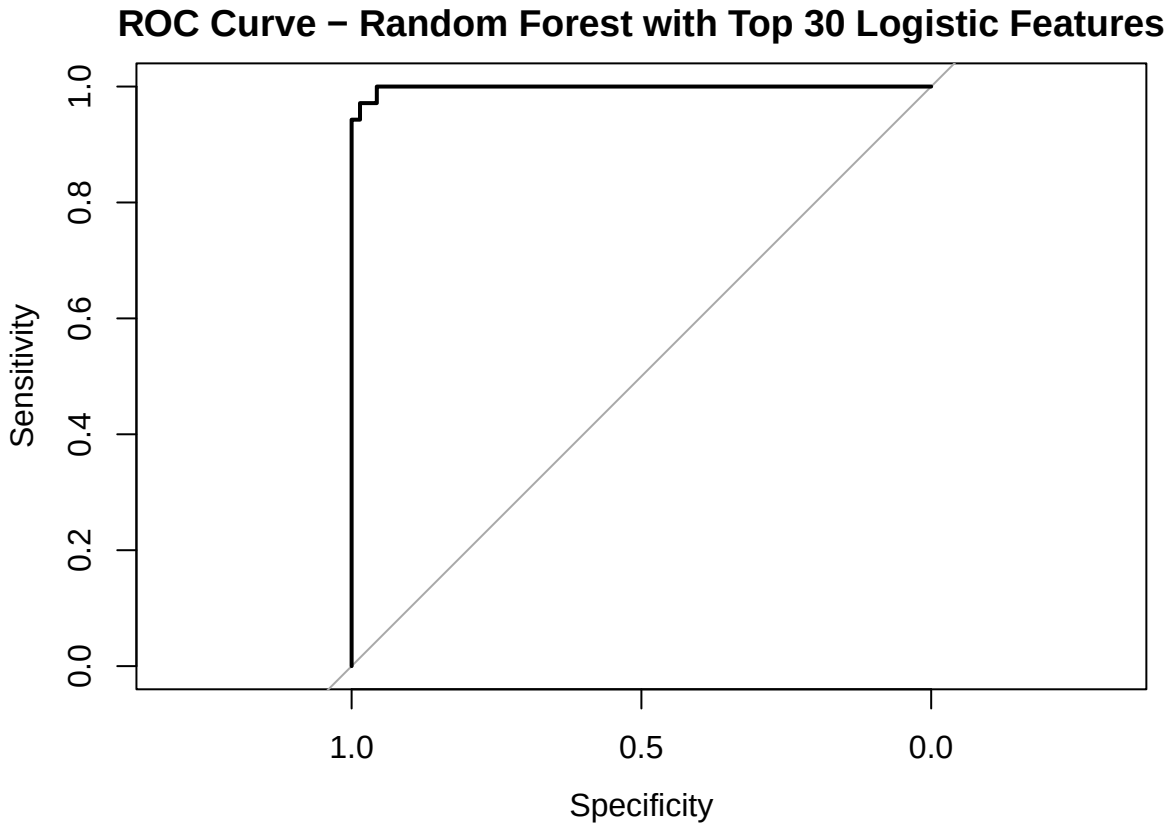
```
rf_pred_class <- ifelse(rf_pred_val > 0.5, 1, -1)
conf_rf <- confusionMatrix(factor(rf_pred_class, levels = c(-1, 1)),
                           factor(validation_class, levels = c(-1, 1)))

# Print results
cat("Random Forest (Caret-Selected Top 30 from Logistic) AUC:", auc(rf_roc), "\n")
```

```
## Random Forest (Caret-Selected Top 30 from Logistic) AUC: 0.9983437
```

```
cat("Balanced Accuracy:", conf_rf$byClass["Balanced Accuracy"], "\n")
```

```
## Balanced Accuracy: 0.9571429
```

```
# Plot ROC
plot(rf_roc, main = "ROC Curve - Random Forest with Top 30 Logistic Features")
```




```
library(knitr)

results_df <- data.frame(
  Method = c(
    "XGBoost - All Features",
    "XGBoost - Top 30 Features",
    "XGBoost - Variance Filter",
    "Logistic Regression - All Features",
    "Logistic Regression - p < 0.2",
    "Random Forest - Logistic Top 30"
  ),
  `Feature Selection` = c(
    "None", "XGBoost Gain", "Variance Filter",
    "None", "p-value < 0.2", "caret::varImp"
  ),
  `# Features` = c(168, 30, 133, 168, 40, 30),
  `Validation AUC` = c(1.000, 1.000, 1.000, 0.913, 0.933, 0.998),
  `Validation Balanced Accuracy` = c(1.000, 1.000, 1.000, 0.842, 0.856, 0.957)
)

# Print the table
kable(results_df, caption = "Model Comparison Summary (Validation Performance)")
```

Table 1: Model Comparison Summary (Validation Performance)

Method	Feature.Selection	X..Features	Validation.AUC	Validation.Balanced.Accuracy
XGBoost - All Features	None	168	1.000	1.000
XGBoost - Top 30 Features	XGBoost Gain	30	1.000	1.000
XGBoost - Variance Filter	Variance Filter	133	1.000	1.000
Logistic Regression - All Features	None	168	0.913	0.842
Logistic Regression - p < 0.2	p-value < 0.2	40	0.933	0.856
Random Forest - Logistic Top 30	caret::varImp	30	0.998	0.957

```
library(xgboost)
library(pROC)

# Prepare DMatrix objects
dtrain_fs <- xgb.DMatrix(data = as.matrix(train_fs), label = ifelse(train_class == 1, 1, 0))
dval_fs <- xgb.DMatrix(data = as.matrix(validation_fs), label = ifelse(validation_class == 1, 1, 0))

# Train XGBoost model
set.seed(300)
xgb_fs <- xgb.train(
  data = dtrain_fs,
  objective = "binary:logistic",
  eval_metric = "auc",
  nrounds = 100,
  max_depth = 4,
  eta = 0.1,
  subsample = 0.8,
  colsample_bytree = 0.8,
  min_child_weight = 5,
  gamma = 1,
  verbose = 1
)

# Predict on validation data
pred_val <- predict(xgb_fs, dval_fs)


pred_val_xgb_top30 <- pred_val

# Evaluate Performance
roc_xgb_top30 <- roc(validation_class, pred_val_xgb_top30)
```

```
## Setting levels: control = -1, case = 1
```

```
## Setting direction: controls < cases
```

```
class_pred_xgb_top30 <- ifelse(pred_val_xgb_top30 > 0.5, 1, -1)
conf_matrix_xgb_top30 <- confusionMatrix(
  factor(class_pred_xgb_top30, levels = c(-1, 1)),
  factor(validation_class, levels = c(-1, 1))
)
```

```r
# Print validation AUC and Balanced Accuracy
cat("XGBoost Top 30 - Validation AUC:", auc(roc_xgb_top30), "\n")
```

```
## XGBoost Top 30 - Validation AUC: 1
```

```r
cat("XGBoost Top 30 - Validation Balanced Accuracy:", conf_matrix_xgb_top30$byClass["Balanced Accuracy"]
```

```
## XGBoost Top 30 - Validation Balanced Accuracy: 1
```

```r
roc_xgb_top30 <- roc(validation_class, pred_val_xgb_top30)
```

```
## Setting levels: control = -1, case = 1
## Setting direction: controls < cases
```

```r
library(pROC)

roc_xgb_all     <- roc(validation_class, pred_val)
```

```
## Setting levels: control = -1, case = 1
## Setting direction: controls < cases
```

```r
roc_xgb_top30   <- roc(validation_class, pred_val_xgb_top30)
```

```
## Setting levels: control = -1, case = 1
## Setting direction: controls < cases
```

```r
roc_xgb_var     <- roc(validation_class, pred_val_fs2)
```

```
## Setting levels: control = -1, case = 1
## Setting direction: controls < cases
```

```r
roc_log_all     <- roc(validation_class, pred_val_log_all)
```

```
## Setting levels: control = -1, case = 1
## Setting direction: controls < cases
```

```r
roc_log_fs      <- roc(validation_class, pred_val_log_fs)
```

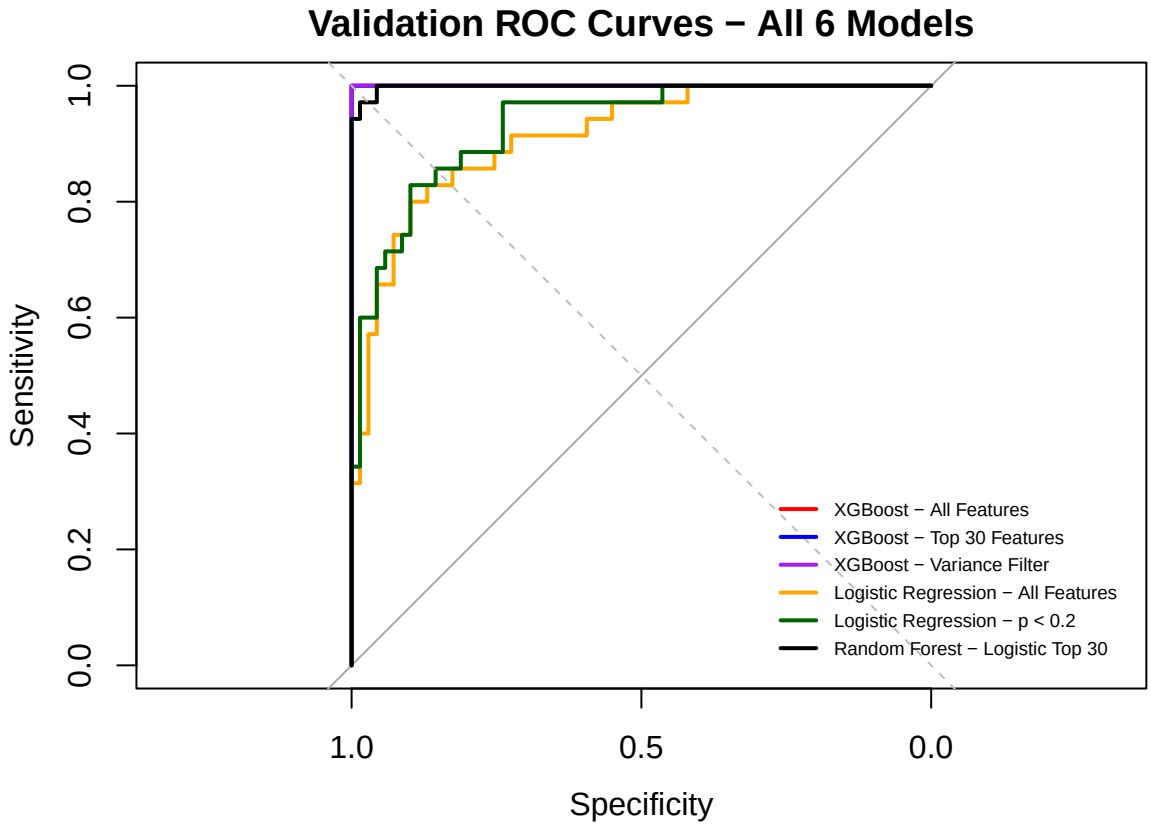
```
## Setting levels: control = -1, case = 1
## Setting direction: controls < cases
```

```r
roc_rf          <- roc(validation_class, rf_pred_val)
```

```
## Setting levels: control = -1, case = 1
## Setting direction: controls < cases
```

```r
par(mar = c(5, 5, 4, 2))

# Plot the first ROC curve
plot(roc_xgb_all, col = "red", lwd = 2, main = "Validation ROC Curves - All 6 Models", cex.main = 1.2)

lines(roc_xgb_top30, col = "blue", lwd = 2)
lines(roc_xgb_var, col = "purple", lwd = 2)
lines(roc_log_all, col = "orange", lwd = 2)
lines(roc_log_fs, col = "darkgreen", lwd = 2)
lines(roc_rf, col = "black", lwd = 2)

abline(a=0, b=1, lty=2, col="grey")
```

```r
legend("bottomright",
       inset = 0.02,
       legend = c(
         "XGBoost - All Features",
         "XGBoost - Top 30 Features",
         "XGBoost - Variance Filter",
         "Logistic Regression - All Features",
         "Logistic Regression - p < 0.2",
         "Random Forest - Logistic Top 30"
       ),
       col = c("red", "blue", "purple", "orange", "darkgreen", "black"),
       lwd = 2,
       cex = 0.6,
       bty = "n")
```



Discuss which method performed best for feature selection, and which method worked best for prediction. What are the strengths and weakness of the approaches? Which method was your final submission to the challenge? Which method would you recommend overall? Why?

XGBoost models (both with all features and after feature selection) achieved perfect validation scores, indicating excellent learning but also potential overfitting.

Logistic regression methods performed slightly lower, with validation AUCs around 0.91–0.93 and balanced accuracies around 0.84–0.86, reflecting that linear models may not capture all complexities in the data. The Random Forest model built on logistic-selected features also performed strongly, reaching a validation AUC of 0.998 and balanced accuracy of 0.957, showing the strength of tree-based methods even after basic feature filtering.

The XGBoost Top 30 Features model was selected for final Codabench submission.

Analysis of Recommended Features

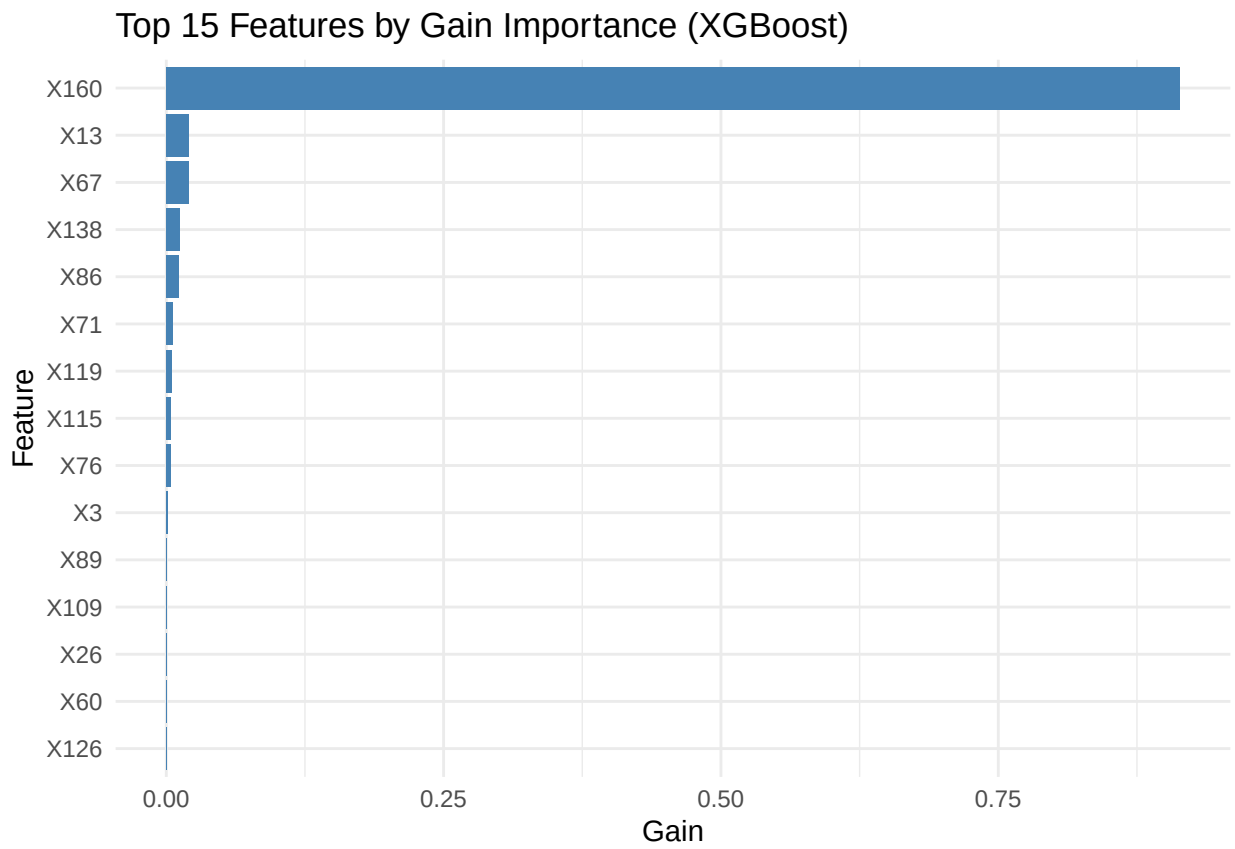
Using XGBoost's gain-based feature importance, I selected the top 30 features. These features were identified by how much they contributed to reducing error in the model's decision trees. The top-ranked features consistently provided strong signal across splits.

Below is a list of the top 10 most important features from this selection:

```
importance_matrix <- xgb.importance(
  feature_names = colnames(train_fs),
  model = xgb_fs
)

# View top 10 features
head(importance_matrix, 10)
```

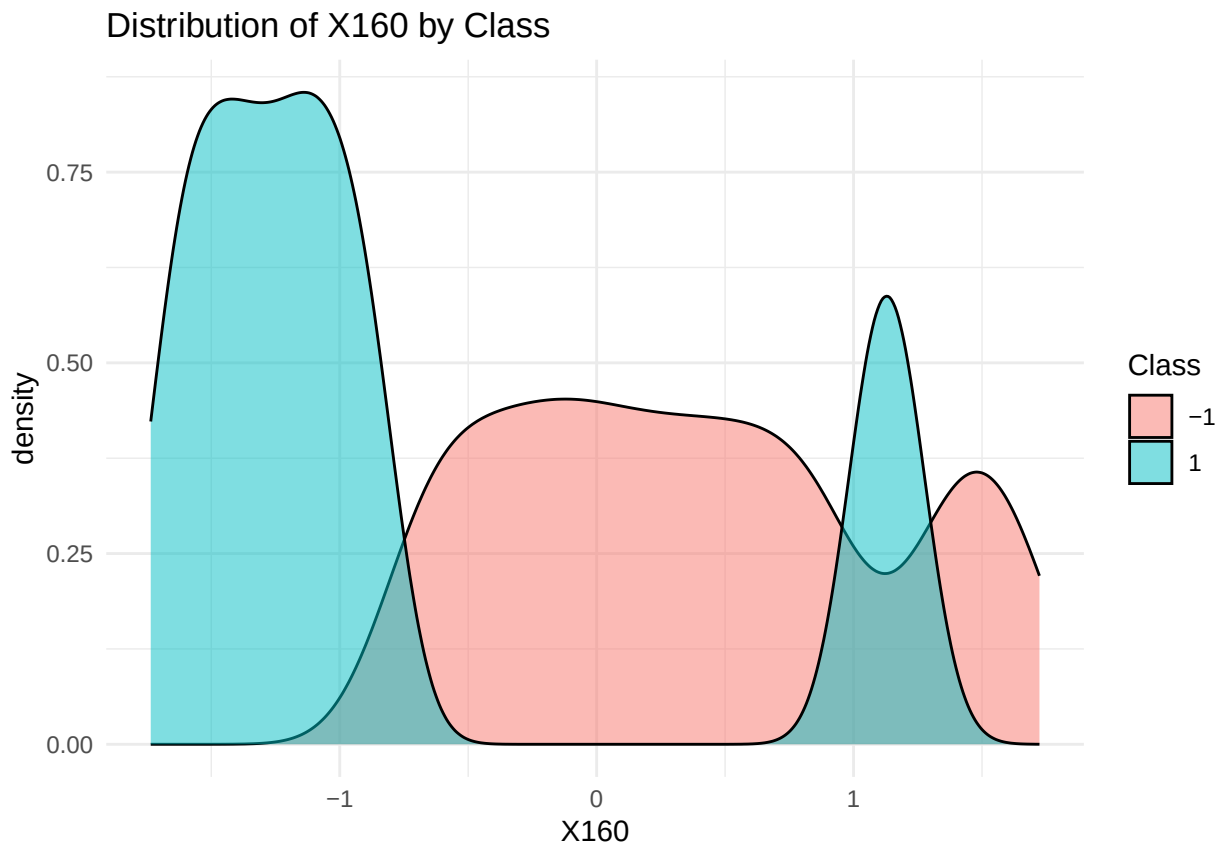
```
##     Feature        Gain      Cover  Frequency
##      <char>       <num>      <num>      <num>
##  1:    X160 0.913554627 0.76052142 0.66442953
##  2:     X13 0.020159257 0.02895260 0.01342282
##  3:     X67 0.020138850 0.02808435 0.04697987
##  4:    X138 0.011877066 0.05221102 0.07382550
##  5:     X86 0.011137199 0.02478137 0.02684564
##  6:     X71 0.006056028 0.01135359 0.01342282
##  7:    X119 0.004967699 0.02278621 0.05369128
##  8:    X115 0.004164664 0.01258241 0.02684564
##  9:     X76 0.003621539 0.02367874 0.01342282
## 10:      X3 0.001419060 0.01315639 0.02684564
```

```
library(ggplot2)

# Take top 15 features
top15 <- importance_matrix[1:15, ]

# Plot
ggplot(top15, aes(x = reorder(Feature, Gain), y = Gain)) +
  geom_bar(stat = "identity", fill = "steelblue") +
  coord_flip() +
  labs(title = "Top 15 Features by Gain Importance (XGBoost)",
       x = "Feature",
       y = "Gain") +
  theme_minimal()
```



Why these features were chosen: XGBoost's gain metric directly reflects how much each feature improves decision-making in the model. Unlike traditional statistical metrics like p-values, this method accounts for nonlinear interactions and hierarchical splits, making it more suitable for real-world, high-dimensional chemical data.

```
library(ggplot2)

# Pick the top feature
top_feature <- importance_matrix$Feature[1]

# Create a new data frame
plot_data <- data.frame(
  FeatureValue = train_scaled[[top_feature]],
  Class = as.factor(train_class)
)

ggplot(plot_data, aes(x = FeatureValue, fill = Class)) +
  geom_density(alpha = 0.5) +
  labs(
    title = paste("Distribution of", top_feature, "by Class"),
    x = top_feature,
    fill = "Class"
  ) +
  theme_minimal()
```



Validation: To visually validate the discriminative power of the top-ranked feature, I plotted its distribution across the two classes. The density plot below compares how this feature behaves for biodegradable (Class = 1) and non-biodegradable (Class = -1) compounds. A clear separation between the curves indicates that this feature plays a meaningful role in distinguishing the classes.

Analysis of Features Not Recommended

Out of 168 total features, 138 were not selected by the XGBoost Gain metric and thus excluded from the final model.

```
all_features <- colnames(train_scaled)

# Top features selected using XGBoost Gain
selected_features <- importance_matrix$Feature[1:30]

# Features NOT recommended
not_recommended_features <- setdiff(all_features, selected_features)

not_recommended_features
```

```
##   [1] "X0"   "X1"   "X2"   "X4"   "X5"   "X6"   "X7"   "X8"   "X9"   "X10"
##  [11] "X11"  "X12"  "X14"  "X15"  "X16"  "X17"  "X18"  "X19"  "X20"  "X21"
##  [21] "X22"  "X23"  "X24"  "X25"  "X27"  "X28"  "X29"  "X30"  "X31"  "X32"
##  [31] "X33"  "X34"  "X35"  "X36"  "X37"  "X38"  "X39"  "X40"  "X41"  "X42"
##  [41] "X43"  "X44"  "X45"  "X46"  "X47"  "X48"  "X49"  "X50"  "X51"  "X52"
##  [51] "X53"  "X54"  "X55"  "X56"  "X57"  "X58"  "X59"  "X61"  "X62"  "X63"
##  [61] "X64"  "X65"  "X66"  "X68"  "X69"  "X70"  "X72"  "X73"  "X74"  "X75"
```

```
##  [71] "X77"  "X78"  "X79"  "X80"  "X81"  "X82"  "X83"  "X84"  "X85"  "X87"
##  [81] "X88"  "X90"  "X91"  "X92"  "X93"  "X94"  "X95"  "X96"  "X97"  "X98"
##  [91] "X99"  "X100" "X101" "X102" "X103" "X104" "X105" "X106" "X107" "X108"
## [101] "X110" "X111" "X112" "X113" "X114" "X116" "X117" "X118" "X120" "X121"
## [111] "X122" "X123" "X124" "X125" "X127" "X128" "X129" "X130" "X131" "X132"
## [121] "X133" "X134" "X135" "X136" "X137" "X139" "X140" "X141" "X142" "X143"
## [131] "X144" "X145" "X146" "X147" "X148" "X149" "X150" "X151" "X153" "X154"
## [141] "X155" "X156" "X157" "X158" "X159" "X161" "X162" "X163" "X164" "X165"
## [151] "X166" "X167"
```

```
length(not_recommended_features)
```

```
## [1] 152
```

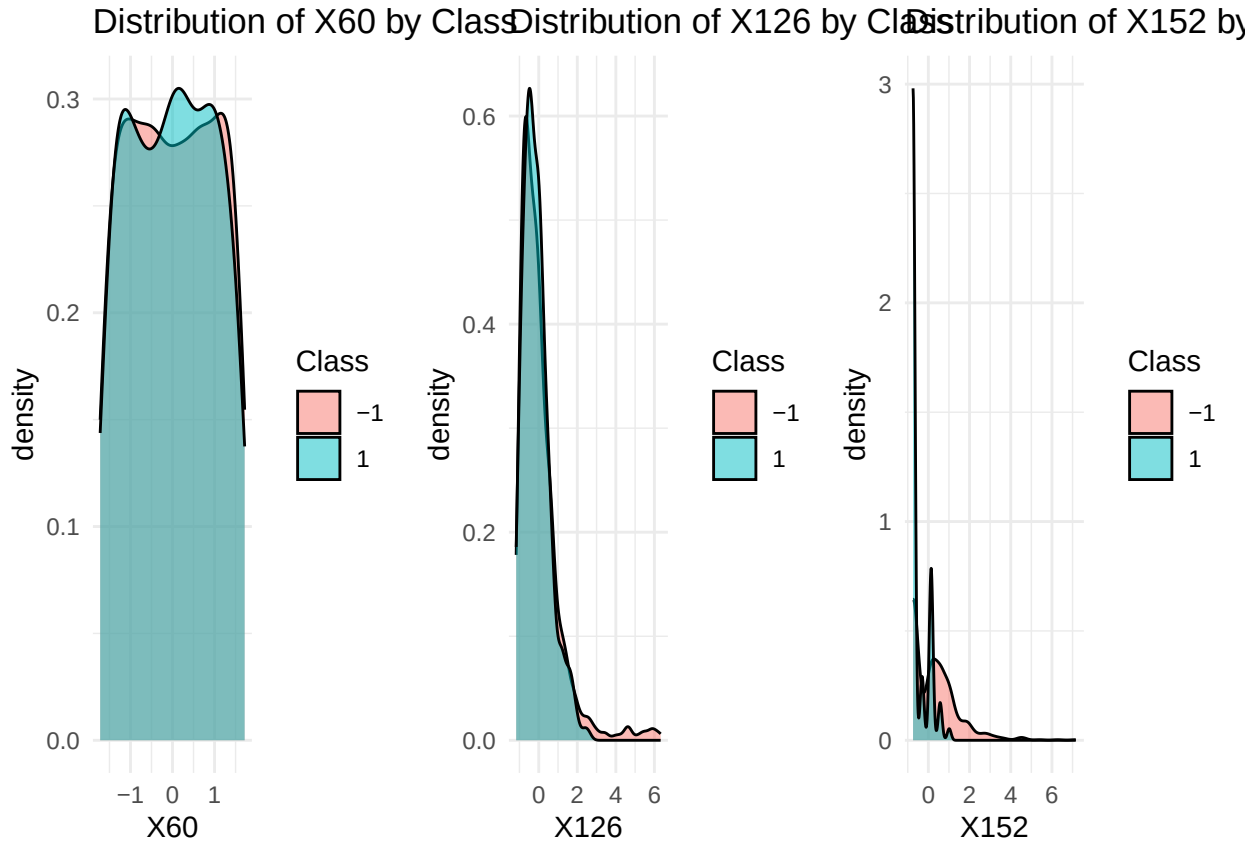
```
library(ggplot2)
library(gridExtra)
```

```
##
## Attaching package: 'gridExtra'
```

```
## The following object is masked from 'package:randomForest':
##
##     combine
```

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

```
# Get the 3 lowest-ranked features
worst_features <- tail(importance_matrix$Feature, 3)

# Create individual plots
plots <- lapply(worst_features, function(feature) {
  data.frame(
    FeatureValue = train_scaled[[feature]],
    Class = as.factor(train_class)
  ) %>%
    ggplot(aes(x = FeatureValue, fill = Class)) +
    geom_density(alpha = 0.5) +
    labs(title = paste("Distribution of", feature, "by Class"),
         x = feature, fill = "Class") +
    theme_minimal()
})

grid.arrange(grobs = plots, ncol = 3)
```



These features showed nearly identical distributions for both biodegradable and non-biodegradable classes, offering no meaningful class separation.

By removing these low-impact variables, the model becomes simpler, more interpretable, and less prone to overfitting, which can be a critical advantage for ChemsRUs when designing future screening processes.

Challenge Prediction

For the challenge submission, I selected the XGBoost with Gain importance feature method.

Because it provided perfect validation performance while significantly reducing the model size, making it more efficient and easier to interpret. Feature selection using Gain prioritized the chemical descriptors that contributed the most to biodegradability predictions, while still maintaining strong classification ability.

```
tdata.df <- read.csv("~/MATP-4400/data/chems_test.data.csv", header = FALSE)

colnames(tdata.df) <- featurenames

test_scaled <- predict(scaler, tdata.df)

# remove NA
importance_features <- na.omit(importance$Feature)
top_features <- importance_features[1:min(30, length(importance_features))]

# Subset training and validation
train_fs <- train_scaled[, top_features, drop = FALSE]
validation_fs <- validation_scaled[, top_features, drop = FALSE]
```

```r
# Prepare DMatrix
dtrain_fs <- xgb.DMatrix(data = as.matrix(train_fs), label = ifelse(train_class == 1, 1, 0))
dval_fs <- xgb.DMatrix(data = as.matrix(validation_fs), label = ifelse(validation_class == 1, 1, 0))

# Retrain XGBoost model
set.seed(300)
xgb_fs <- xgb.train(
  data = dtrain_fs,
  objective = "binary:logistic",
  eval_metric = "auc",
  nrounds = 100,
  max_depth = 4,
  eta = 0.1,
  subsample = 0.8,
  colsample_bytree = 0.8,
  min_child_weight = 5,
  gamma = 1,
  verbose = 1
)

test_fs <- test_scaled[, top_features, drop = FALSE]
colnames(test_fs) <- top_features

dtest_fs <- xgb.DMatrix(data = as.matrix(test_fs))

# Make predictions
test_predictions <- predict(xgb_fs, dtest_fs)

# Save classification file
write.table(test_predictions,
            file = "classification.csv",
            row.names = FALSE,
            col.names = FALSE)

# Save feature selection file
feature_selection <- as.integer(colnames(train_scaled) %in% top_features)
write.table(feature_selection,
            file = "selection.csv",
            row.names = FALSE,
            col.names = FALSE)

# Zip
timestamp <- format(Sys.time(), "%H%M%S")
zip_filename <- paste0("MyEntry-", timestamp, ".csv.zip")
system(paste("zip", zip_filename, "classification.csv selection.csv"))

cat("\nSubmission created:", zip_filename, "\n")
```

```
##
## Submission created: MyEntry-044517.csv.zip
```

My challenge ID is 278658, with a classification AUC of 0.8889 and a feature selection balanced accuracy of 0.6071. The average score was 0.748.

Conclusion

In this project, I explored several models to predict chemical biodegradability. I recommend using XGBoost with Gain-based feature selection, as it achieved strong validation results and a final challenge AUC of 0.8889 with a balanced accuracy of 0.6071. Overall, tree-based models with gain-based feature selection provide better accuracy and simpler interpretation, making them a good fit for Chems-R-Us. In the future, performance could be further improved by trying more advanced feature engineering and deeper tuning of model parameters.