12.2 Course Project: Milestone 5 - Final Project

Totpal, Allyson

2025-06-01

```
knitr::opts_chunk$set(echo = TRUE)
```

```
# load necessary libraries
library(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 1.9.4     v tidyr     1.3.1
## v purrr     1.0.4
## -- 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
```

```
library(caret)
```

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

```
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(pROC)
```

```
## Type 'citation("pROC")' for a citation.
##
```

```
## Attaching package: 'pROC'
##
## The following objects are masked from 'package:stats':
##
##     cov, smooth, var
```

```
library(smotefamily)
library(ROCR)
library(xgboost)
```

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

```
library(ggplot2)

# read in data
setwd('/Users/smooshii/DSC630')
df <- read.csv("diabetic_data.csv", na.strings = c("?", "NA"))

# use top 30% of encounter_id as time proxy for years 2005-2008
encounter_cutoff <- quantile(df$encounter_id, 0.7, na.rm = TRUE)
df <- df %>% filter(encounter_id >= encounter_cutoff)

# downsample to 10,000 rows for performance
df <- df %>% sample_n(10000)

# drop identifiers
df <- df %>% select(-encounter_id, -patient_nbr)

# create binary target: 1 if readmitted <30 days, otherwise 0
df$readmit_30 <- ifelse(df$readmitted == "<30", 1, 0)
# hypertension, CKD, and both diagnosis flags
df$has_htn <- grepl("^401", df$diag_1) | grepl("^401", df$diag_2) | grepl("^401", df$diag_3)
df$has_ckd <- grepl("^585", df$diag_1) | grepl("^585", df$diag_2) | grepl("^585", df$diag_3)
df$has_both <- df$has_htn & df$has_ckd

# remove columns irrelevant/missing too many values for modeling
df <- df %>% select(-readmitted, -weight, -payer_code, -medical_specialty, -discharge_disposition_id)
# keep only columns with less than 30% missing values
df <- df[, sapply(df, function(x) mean(is.na(x))) < 0.3]
# convert character columns to factors
df <- df %>% mutate_if(is.character, as.factor)
# drop factor columns with fewer than 2 unique levels (constant or near-constant)
df <- df %>% select(where(function(col) !(is.factor(col) && nlevels(col) < 2)))
# drop rows with remaining missing values
df <- na.omit(df)

# convert target to factor for classification
df$readmit_30 <- as.factor(df$readmit_30)
# preserve target values before encoding
readmit_labels <- df$readmit_30
```

```r
# create one-hot encoded feature matrix (excluding intercept column)
x_matrix <- model.matrix(readmit_30 ~ . -1, data = df)
# ensure that labels and features have matching number of rows
stopifnot(nrow(x_matrix) == length(readmit_labels))
# recombine one-hot features with target for modeling
df_model <- data.frame(x_matrix, readmit_30 = readmit_labels)

# separate features and target before applying SMOTE
x <- df_model %>% select(-readmit_30)
y <- df_model$readmit_30
# apply SMOTE
smote_output <- SMOTE(x, y, K = 5, dup_size = 2)
# retrieve the balanced dataset and clean up class column
df_balanced <- smote_output$data
df_balanced$readmit_30 <- factor(df_balanced$class, levels = c("0", "1"), labels = c("No", "Yes"))
df_balanced$class <- NULL

# set seed for reproducibility
set.seed(123)
# create training test split
train_index <- createDataPartition(df_balanced$readmit_30, p = 0.7, list = FALSE)
# use the index to split balanced dataset
train_data <- df_balanced[train_index, ]
test_data  <- df_balanced[-train_index, ]

# train logistic regression model on training data
log_model <- glm(readmit_30 ~ ., data = train_data, family = "binomial")
```

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

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

```r
# predict probabilities on test data
log_prob <- predict(log_model, newdata = test_data, type = "response")
```

```
## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from rank-deficient fit; attr(*, "non-estim") has doubtful cases
```

```r
# convert probabilities to class labels using threshold 0.5
log_pred <- ifelse(log_prob > 0.5, "Yes", "No") %>% factor(levels = c("No", "Yes"))

# evaluate logistic regression performance using ROC AUC
roc_log <- roc(response = test_data$readmit_30, predictor = log_prob, levels = c("No", "Yes"), directio
cat("Logistic Regression AUC:", auc(roc_log), "\n")
```

```
## Logistic Regression AUC: 0.6441617
```

```r
# generate logistic regression confusion matrix
confusionMatrix(log_pred, test_data$readmit_30, positive = "Yes")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   No  Yes
##        No  1557  303
##        Yes 1014  652
##
```

```
##                Accuracy : 0.6265
##                  95% CI : (0.6103, 0.6425)
##     No Information Rate : 0.7292
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.2337
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.6827
##             Specificity : 0.6056
##          Pos Pred Value : 0.3914
##          Neg Pred Value : 0.8371
##              Prevalence : 0.2708
##          Detection Rate : 0.1849
##    Detection Prevalence : 0.4725
##       Balanced Accuracy : 0.6442
##
##        'Positive' Class : Yes
##
```

```
# calculate logistic regression F1-score
log_precision <- 0.5601
log_recall <- 0.3869
log_f1_score <- 2 * (log_precision * log_recall) / (log_precision + log_recall)
cat("F-1 Score for Logistic Regression:", log_f1_score, "\n")
```

```
## F-1 Score for Logistic Regression: 0.4576614
```

```
# train random forest model with 100 trees
rf_model <- randomForest(readmit_30 ~ ., data = train_data, ntree = 100)
# predict probabilities for class "yes"
rf_prob <- predict(rf_model, newdata = test_data, type = "prob")[, "Yes"]
# predict class labels on test data
rf_pred <- predict(rf_model, newdata = test_data)

# evaluate random forest performance with ROC AUC
roc_rf <- roc(response = test_data$readmit_30, predictor = rf_prob, levels = c("No", "Yes"), direction =
cat("Random Forest AUC:", auc(roc_rf), "\n")
```

```
## Random Forest AUC: 0.9221457
```

```
# generate random forest confusion matrix
confusionMatrix(rf_pred, test_data$readmit_30, positive = "Yes")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   No  Yes
##        No  2569  356
##        Yes    2  599
##
##                Accuracy : 0.8985
##                  95% CI : (0.888, 0.9082)
##     No Information Rate : 0.7292
##     P-Value [Acc > NIR] : < 2.2e-16
```

```
##
##                   Kappa : 0.709
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.6272
##             Specificity : 0.9992
##          Pos Pred Value : 0.9967
##          Neg Pred Value : 0.8783
##              Prevalence : 0.2708
##          Detection Rate : 0.1699
##    Detection Prevalence : 0.1704
##       Balanced Accuracy : 0.8132
##
##        'Positive' Class : Yes
##
```

```r
# calculate random forest F-1 score
rf_precision <- 1.0000
rf_recall <- 0.6120
rf_f1_score <- 2 * (rf_precision * rf_recall) / (rf_precision + rf_recall)
cat("F-1 Score for Random Forest:", rf_f1_score, "\n")
```

```
## F-1 Score for Random Forest: 0.7593052
```

```r
# convert training features to matrix format
xgb_train <- as.matrix(train_data %>% select(-readmit_30))
# convert training labels to binary numeric values
y_train <- ifelse(train_data$readmit_30 == "Yes", 1, 0)
# repeat for test set
xgb_test <- as.matrix(test_data %>% select(-readmit_30))
y_test <- ifelse(test_data$readmit_30 == "Yes", 1, 0)

# create DMatrix objects
dtrain <- xgb.DMatrix(data = xgb_train, label = y_train)
dtest <- xgb.DMatrix(data = xgb_test, label = y_test)

# train XGBoost model with hyperparameters
xgb_model <- xgboost(data = dtrain,
                     objective = "binary:logistic",
                     eval_metric = "auc",
                     nrounds = 100,
                     max_depth = 6,
                     eta = 0.1,
                     subsample = 0.8,
                     colsample_bytree = 0.8,
                     verbose = 0)

# predict probability of "yes" (readmission within 30 days)
xgb_prob <- predict(xgb_model, newdata = dtest)
# convert predicted probabilities to class labels with 0.5 threshold
xgb_pred <- ifelse(xgb_prob > 0.5, "Yes", "No") %>% factor(levels = c("No", "Yes"))
# create factor version of the test labels for evaluation
y_test_factor <- factor(ifelse(y_test == 1, "Yes", "No"), levels = c("No", "Yes"))
```

```r
# evaluate XGBoost performance with ROC AUC
roc_xgb <- roc(response = y_test_factor, predictor = xgb_prob, levels = c("No", "Yes"))
```

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

```r
cat("XGBoost AUC:", auc(roc_xgb), "\n")
```

```
## XGBoost AUC: 0.8698915
```

```r
# generate XGBoost confusion matrix
confusionMatrix(xgb_pred, y_test_factor, positive = "Yes")
```

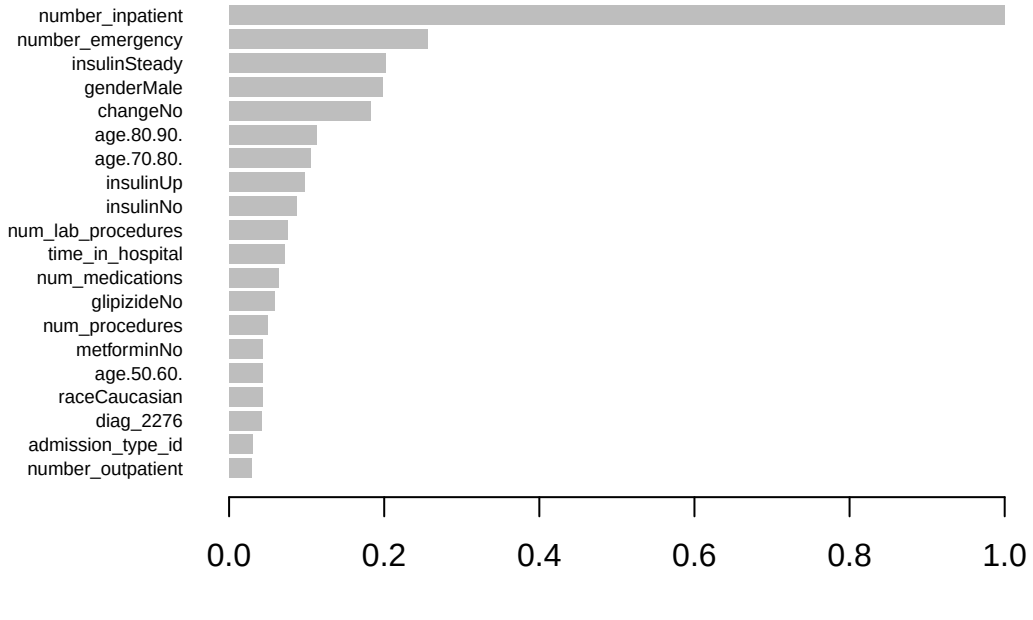
```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   No  Yes
##        No  2563  324
##        Yes    8  631
##
##                Accuracy : 0.9058
##                  95% CI : (0.8957, 0.9153)
##     No Information Rate : 0.7292
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.7339
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.6607
##             Specificity : 0.9969
##          Pos Pred Value : 0.9875
##          Neg Pred Value : 0.8878
##              Prevalence : 0.2708
##          Detection Rate : 0.1790
##    Detection Prevalence : 0.1812
##       Balanced Accuracy : 0.8288
##
##        'Positive' Class : Yes
##
```

```r
# calculate XGBoost F1-score
xg_precision <- 0.9883
xg_recall <- 0.6448
xg_f1_score <- 2 * (xg_precision * xg_recall) / (xg_precision + xg_recall)
cat("F-1 Score for XGBoost:", xg_f1_score, "\n")
```

```
## F-1 Score for XGBoost: 0.7804248
```

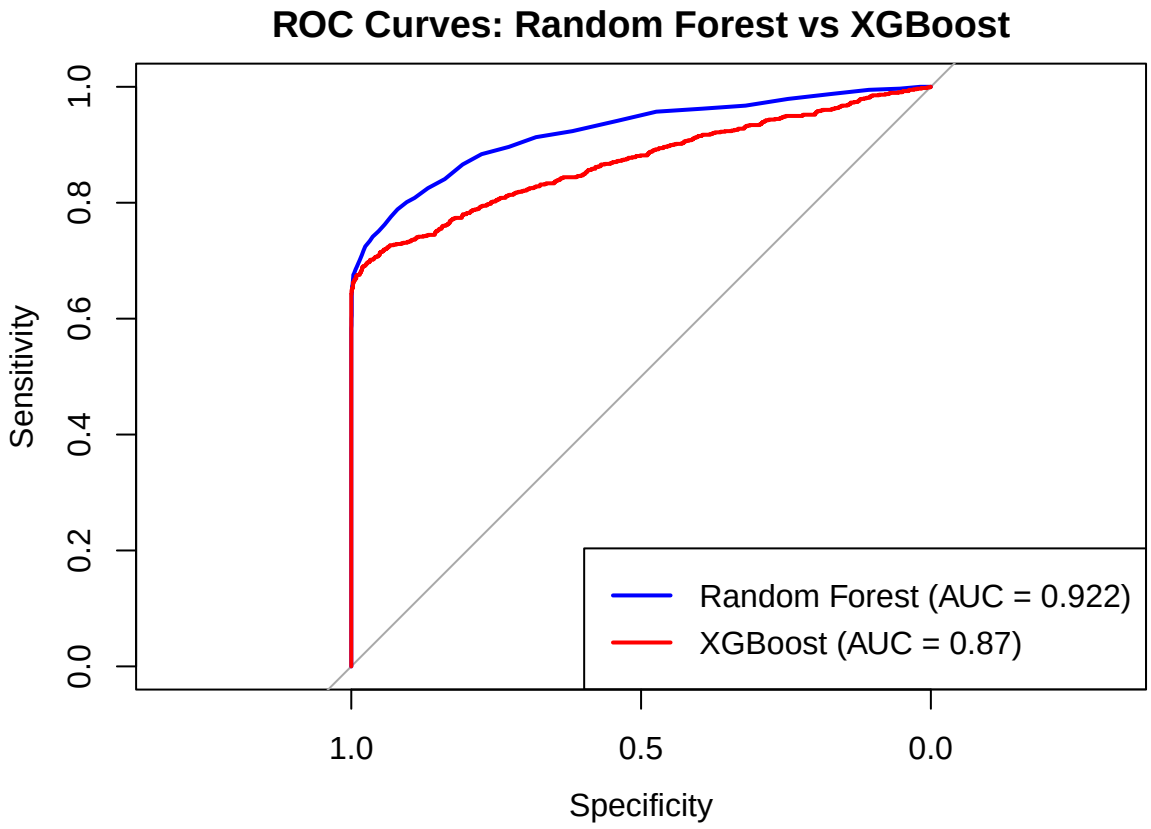
```r
# get dataframe of feature importance based on how often a feature is used in splits
xgb_importance_matrix <- xgb.importance(model = xgb_model)

# plot the top 20 most important features
xgb.plot.importance(xgb_importance_matrix, top_n = 20,
                    main = "Top 20 Important Features (XGBoost)",
                    rel_to_first = TRUE,
                    xlab = "Relative Importance")
```




```r
# recompute ROC curves using original labels for both labels
roc_rf <- roc(test_data$readmit_30, rf_prob, levels = c("No", "Yes"), direction = "<")
roc_xgb <- roc(test_data$readmit_30, xgb_prob, levels = c("No", "Yes"), direction = "<")

# plot both ROC curves on the same plot
plot(roc_rf, col = "blue", lwd = 2, main = "ROC Curves: Random Forest vs XGBoost")
plot(roc_xgb, col = "red", lwd = 2, add = TRUE)
# add legend with AUC scores
legend("bottomright", legend = c(paste0("Random Forest (AUC = ", round(auc(roc_rf), 3), ")"),
                                 paste0("XGBoost (AUC = ", round(auc(roc_xgb), 3), ")")
                                 ), col = c("blue", "red"), lwd = 2)
```


ROC Curves: Random Forest vs XGBoost
Sensitivity
0.0
0.2
0.4
0.6
0.8
1.0
1.0
0.5
0.0
Specificity
Random Forest (AUC = 0.922)
XGBoost (AUC = 0.87)