Machine Learning Project Modeling

```
library(caret)
library(rpart)
library(nnet)
library(ALEPlot)
library(class)
library(mgcv)
library(gbm)
library(MLmetrics)
library(randomForest)
library(xgboost)
library(yaImpute)
```

```
data <- read.csv("data_for_modeling_clean.csv", header = TRUE)
data$DEP_DEL15 <- factor(data$DEP_DEL15, levels = c(0, 1), labels = c("No", "Yes"))
```

Train Test Split

```
set.seed(1234)
trainIndex <- createDataPartition(data$DEP_DEL15, p = 0.8, list = FALSE)
train_data <- data[trainIndex, ]
test_data <- data[-trainIndex, ]
```

Standardize data

```
numeric_vars <- sapply(data, is.numeric)
train_mean <- colMeans(train_data[, numeric_vars], na.rm = TRUE)
train_sd <- apply(train_data[, numeric_vars], 2, sd, na.rm = TRUE)
train_data[, numeric_vars] <- scale(train_data[, numeric_vars], center = train_mean, scale = train_sd)

# scale the test data using the train mean and sd to prevent data leakage
test_data[, numeric_vars] <- scale(test_data[, numeric_vars], center = train_mean, scale = train_sd)
```

```
str(train_data)
```

```
## 'data.frame':    51685 obs. of  14 variables:
##  $ MONTH              : num  -1.65 -1.65 -1.65 -1.65 -1.65 ...
##  $ DAY_OF_WEEK        : num  1.53 1.53 1.53 1.53 1.53 ...
##  $ DEP_DEL15          : Factor w/ 2 levels "No","Yes": 1 1 1 1 1 1 2 1 1 1 ...
##  $ DISTANCE_GROUP     : num  -0.7653 1.3163 0.0673 2.5652 1.3163 ...
##  $ SEGMENT_NUMBER     : num  -1.1649 -0.5921 -0.5921 -0.5921 -0.0192 ...
##  $ NUMBER_OF_SEATS    : num  0.262 0.882 -0.101 -0.101 1.224 ...
##  $ GROUND_SERV_PER_PASS: num  0.832 -0.789 -0.599 -0.19 0.277 ...
##  $ PLANE_AGE          : num  -0.6509 -1.2255 -0.0762 -0.0762 -1.5128 ...
##  $ PRCP               : num  -0.301 -0.301 -0.301 -0.301 -0.301 ...
##  $ SNOW               : num  -0.103 -0.103 -0.103 -0.103 -0.103 ...
##  $ SNWD               : num  -0.126 -0.126 -0.126 -0.126 -0.126 ...
##  $ TMAX               : num  -0.361 -0.361 -0.361 -0.361 -0.361 ...
##  $ AWND               : num  -1.5 -1.5 -1.5 -1.5 -1.5 ...
##  $ DEP_TIME_START     : num  -1.399 -0.806 -1.202 -0.609 -0.214 ...
```

To store the result

```
model_results_df <- data.frame(
  Model = character(),
  Accuracy = numeric(),
  Precision = numeric(),
  Recall = numeric(),
  F1_Score = numeric(),
  stringsAsFactors = FALSE
)

# Function to add results
add_model_results <- function(model_name, acc, prec, rec, f1) {
  new_row <- data.frame(
    Model = model_name,
    Accuracy = round(acc, 4),
    Precision = round(prec, 4),
    Recall = round(rec, 4),
    F1_Score = round(f1, 4)
  )
  return(rbind(model_results_df, new_row))
}
```

1. Baseline

```
prop.table(table(train_data$DEP_DEL15))
```

```
##
##        No       Yes
## 0.8097707 0.1902293
```

2. Logistic Regression

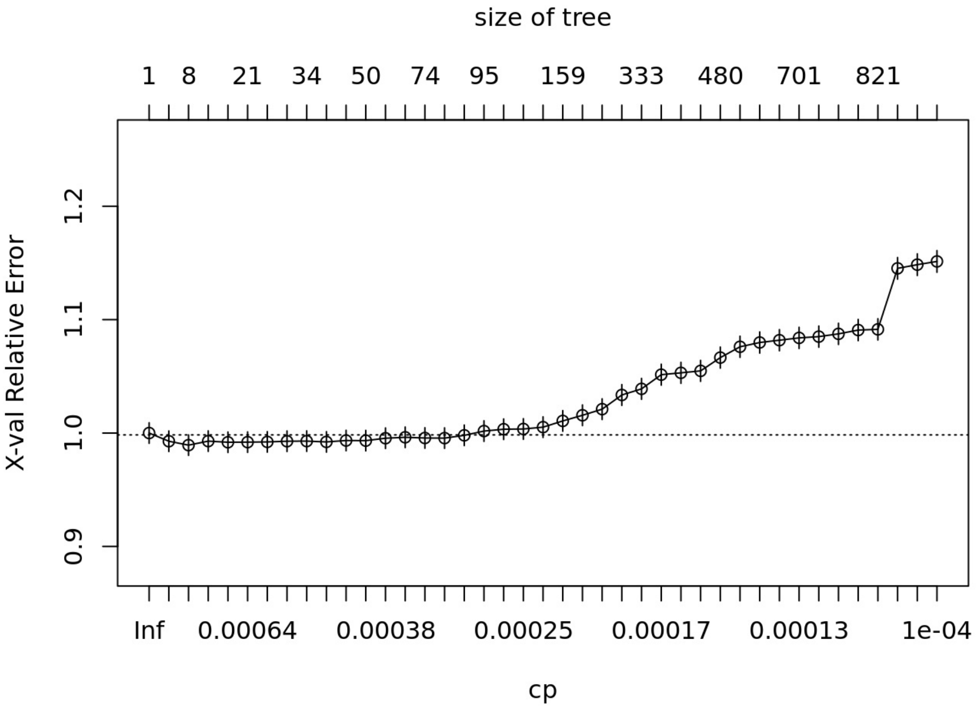
```
train_control <- trainControl(method = "cv", number = 5, classProbs = TRUE, summaryFunction = twoClassSummary)

logistic_model <- train(DEP_DEL15 ~ ., data = train_data, method = "glm", family = binomial, trControl = train_co
ntrol, metric = "ROC")

# Predict on test set
predictions <- predict(logistic_model, test_data)
conf_matrix <- confusionMatrix(predictions, test_data$DEP_DEL15, positive = "Yes")

# Store results
model_results_df <- add_model_results("Logistic Regression",
                                      conf_matrix$overall["Accuracy"],
                                      conf_matrix$byClass["Precision"],
                                      conf_matrix$byClass["Recall"],
                                      conf_matrix$byClass["F1"])
```

3. Classification Tree

```
control <- rpart.control(minbucket = 5, cp = 0.0001, maxsurrogate = 0, usesurrogate = 0, xval = 10)
tree_model <- rpart(DEP_DEL15 ~ ., data = train_data, method = "class", control = control)
plotcp(tree_model)
```

```
best_cp <- tree_model$cptable[which.min(tree_model$cptable[, "xerror"]), "CP"]
prune_tree <- prune(tree_model, cp = best_cp)
prune_tree$variable.importance
```

```
plotcp(tree_model)
```




```
prune_tree$variable.importance
```

```
## DEP_TIME_START           PRCP SEGMENT_NUMBER
##      452.17667      191.84900       60.75027
```

```
cp_table_prune_tree = prune_tree$cptable[nrow(prune_tree$cptable),]
missclassified_cv_tree <- cp_table_prune_tree['xerror'] * (1 - max(prop.table(table(train_data$DEP_DEL15))))
missclassified_cv_tree
```

```
missclassified_cv_tree
```

```
##    xerror
## 0.1881977
```

```
test_preds <- predict(prune_tree, test_data, type = "class")
conf_matrix_tree <- confusionMatrix(test_preds, test_data$DEP_DEL15, positive = "Yes")

model_results_df <- add_model_results("Decision Tree",
                                      conf_matrix_tree$overall["Accuracy"],
                                      conf_matrix_tree$byClass["Precision"],
                                      conf_matrix_tree$byClass["Recall"],
                                      conf_matrix_tree$byClass["F1"])
```

Classification Tree with Modified Weight

```
tree_model_weighted <- rpart(DEP_DEL15 ~ ., data = train_data, method = "class",
                             control = rpart.control(cp = best_cp),
                             parms = list(loss = matrix(c(0, 2, 1, 0), nrow = 2)))
```

```
test_preds_tree_weighted <- predict(tree_model_weighted, test_data, type = "class")
conf_matrix_tree_weighted <- confusionMatrix(test_preds_tree_weighted, test_data$DEP_DEL15, positive = "Yes")

# Store results
model_results_df <- add_model_results("Decision Tree Weighted Ration 2:1",
                                      conf_matrix_tree_weighted$overall["Accuracy"],
                                      conf_matrix_tree_weighted$byClass["Precision"],
                                      conf_matrix_tree_weighted$byClass["Recall"],
                                      conf_matrix_tree_weighted$byClass["F1"])
```

Try Ratio 3:1 and 4:1 because the ratio of the imbalance class is 4:1

```
tree_model_weighted_2 <- rpart(DEP_DEL15 ~ ., data = train_data, method = "class",
                             control = rpart.control(cp = best_cp),
                             parms = list(loss = matrix(c(0, 3, 1, 0), nrow = 2)))
```

```
test_preds_tree_weighted_2 <- predict(tree_model_weighted_2, test_data, type = "class")
conf_matrix_tree_weighted_2 <- confusionMatrix(test_preds_tree_weighted_2, test_data$DEP_DEL15, positive = "Yes")

# Store results
model_results_df <- add_model_results("Decision Tree Weighted Ration 3:1",
                                      conf_matrix_tree_weighted_2$overall["Accuracy"],
                                      conf_matrix_tree_weighted_2$byClass["Precision"],
                                      conf_matrix_tree_weighted_2$byClass["Recall"],
                                      conf_matrix_tree_weighted_2$byClass["F1"])
```

```
tree_model_weighted_3 <- rpart(DEP_DEL15 ~ ., data = train_data, method = "class",
                             control = rpart.control(cp = best_cp),
                             parms = list(loss = matrix(c(0, 4, 1, 0), nrow = 2)))
```

```
test_preds_tree_weighted_3 <- predict(tree_model_weighted_3, test_data, type = "class")
conf_matrix_tree_weighted_3 <- confusionMatrix(test_preds_tree_weighted_3, test_data$DEP_DEL15, positive = "Yes")

# Store results
model_results_df <- add_model_results("Decision Tree Weighted Ration 4:1",
                                      conf_matrix_tree_weighted_3$overall["Accuracy"],
                                      conf_matrix_tree_weighted_3$byClass["Precision"],
                                      conf_matrix_tree_weighted_3$byClass["Recall"],
                                      conf_matrix_tree_weighted_3$byClass["F1"])
```

```
tree_model_weighted_3$variable.importance
```

```
##       DEP_TIME_START       SEGMENT_NUMBER                 PRCP                MONTH GROUND_SERV_PER_PASS
NUMBER_OF_SEATS                 TMAX       DISTANCE_GROUP
##          1193.374741           844.371221           362.512324           140.382269            62.885799
58.694956            49.637248            43.136116
##                 SNOW                 AWND            PLANE_AGE                 SNWD
##            18.978855             9.683105             7.748285             5.172017
```

4. Neural Network

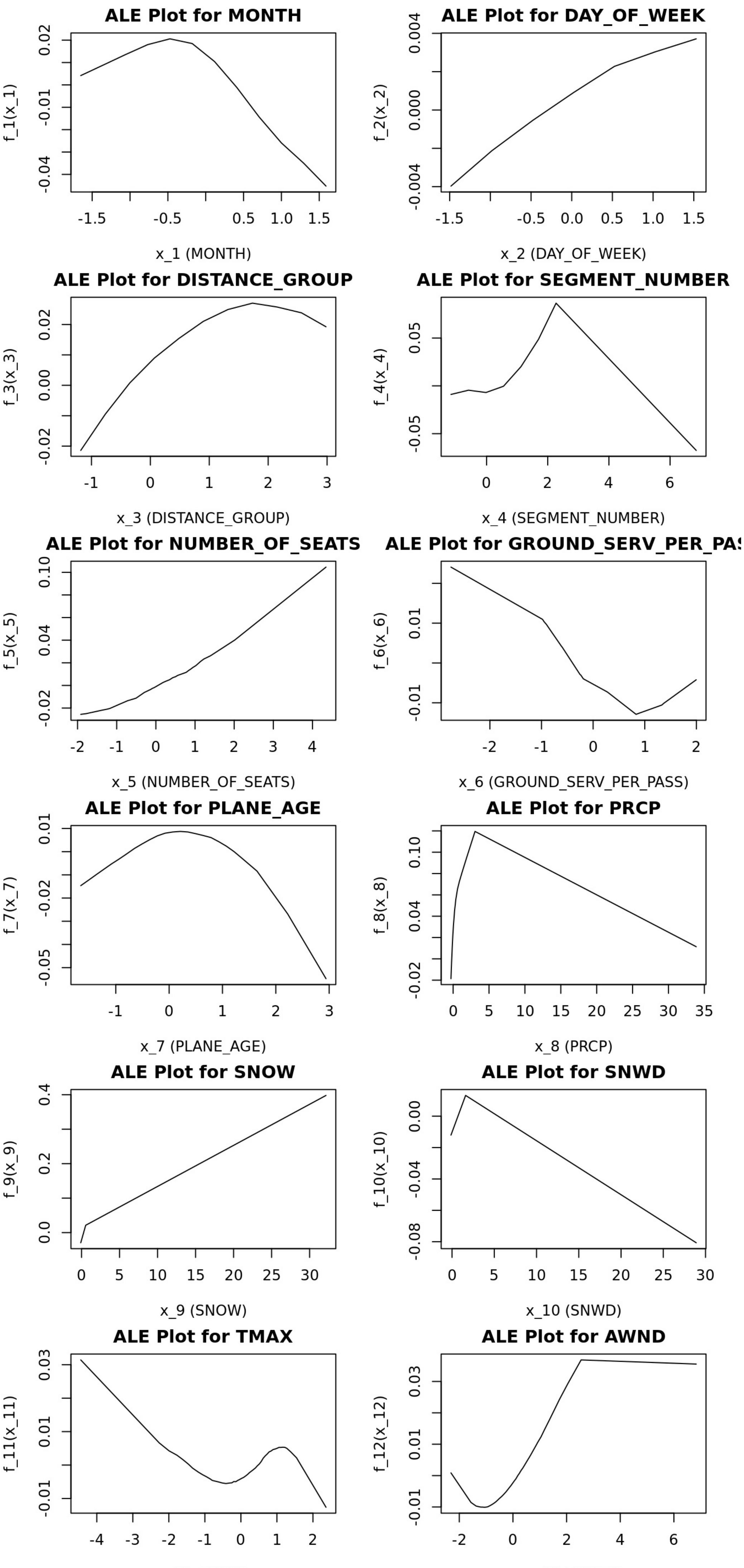
```
# create the cross validation function
set.seed(123)
CVInd <- function(n,K) {
 m<-floor(n/K) #approximate size of each part
 r<-n-m*K
 I<-sample(n,n) #random reordering of the indices
 Ind<- vector("list", K)
 for (k in 1:K) {
 if (k <= r) kpart <- ((m+1)*(k-1)+1):((m+1)*k)
 else kpart<-((m+1)*r+m*(k-r-1)+1):((m+1)*r+m*(k-r))
 Ind[[k]] <- I[kpart] #indices for kth part of data
 }
 return(Ind)
}
```

```r
set.seed(123)
K <- 5
hidden_nodes <- c(3, 5, 10)
decay_values <- c(0.1, 0.3, 0.5)
n.models <- length(hidden_nodes) * length(decay_values)
n <- nrow(train_data)
y <- train_data$DEP_DEL15
CV_metrics <- matrix(0, n.models, 4)
model_list <- list()

Ind <- CVInd(n, K)
model_index <- 1
for (h in hidden_nodes) {
  for (d in decay_values) {
    yhat <- numeric(n)
    for (k in 1:K) {
      test_idx <- Ind[[k]]
      train_idx <- setdiff(1:n, test_idx)

      nn_model <- nnet(DEP_DEL15 ~ ., data = train_data[train_idx, ], size = h, decay = d, maxit = 500, linout =
FALSE, skip = FALSE, trace = FALSE)
      yhat[test_idx] <- predict(nn_model, train_data[test_idx, ], type = "class")
    }

    confusion <- confusionMatrix(factor(yhat), factor(y), positive = "Yes")
    CV_metrics[model_index, ] <- c(confusion$overall["Accuracy"], confusion$byClass["Precision"], confusion$byCla
ss["Recall"], confusion$byClass["F1"])
    model_list[[model_index]] <- list(model = nn_model, size = h, decay = d)
    model_index <- model_index + 1
  }
}

# Identify the best model
best_model_index <- which.max(CV_metrics[, 4]) # best F1 score
best_model <- model_list[[best_model_index]]$model
```

```r
best_hideen_nodes <- model_list[[best_model_index]]$size
best_decay <- model_list[[best_model_index]]$decay

best_nn_model <- nnet(DEP_DEL15 ~ ., data = train_data, size = best_hideen_nodes, decay = best_decay, maxit = 500
, linout = FALSE, skip = FALSE, trace = FALSE)
```

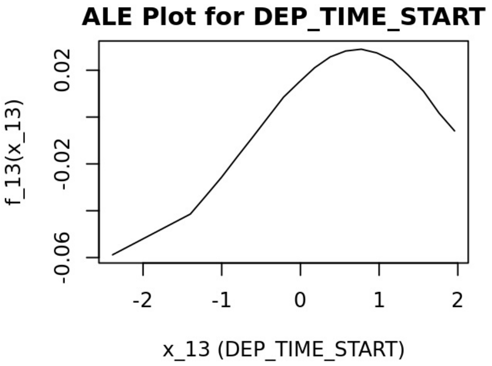
```r
nn_pred <- predict(best_nn_model, test_data, type = "class")
conf_matrix_nnet <- confusionMatrix(factor(nn_pred), factor(test_data$DEP_DEL15), positive = "Yes")

# Store results
model_results_df <- add_model_results("Neural Network",
                                      conf_matrix_nnet$overall["Accuracy"],
                                      conf_matrix_nnet$byClass["Precision"],
                                      conf_matrix_nnet$byClass["Recall"],
                                      conf_matrix_nnet$byClass["F1"])
```

```r
library(ALEPlot)

ale_results <- list()
feature_names <- colnames(train_data)[colnames(train_data) != "DEP_DEL15"]

pred.fun <- function(X.model, newdata) {
  predict(best_nn_model, newdata, type = "raw")
}

par(mfrow = c(2, 2))
par(mar = c(4, 4, 2, 2))

for (feature in feature_names) {
  ale_plot <- ALEPlot(
    X = train_data[, feature_names],
    pred.fun = pred.fun,
    J = which(feature_names == feature),
    K = 50
  )

  title(main = paste("ALE Plot for", feature))
  ale_results[[feature]] <- ale_plot
}
```


ALE Plot for MONTH
f_1(x_1)
x_1 (MONTH)
ALE Plot for DAY_OF_WEEK
f_2(x_2)
x_2 (DAY_OF_WEEK)
ALE Plot for DISTANCE_GROUP
f_3(x_3)
x_3 (DISTANCE_GROUP)
ALE Plot for SEGMENT_NUMBER
f_4(x_4)
x_4 (SEGMENT_NUMBER)
ALE Plot for NUMBER_OF_SEATS
f_5(x_5)
x_5 (NUMBER_OF_SEATS)
ALE Plot for GROUND_SERV_PER_PA
f_6(x_6)
x_6 (GROUND_SERV_PER_PASS)
ALE Plot for PLANE_AGE
f_7(x_7)
x_7 (PLANE_AGE)
ALE Plot for PRCP
f_8(x_8)
x_8 (PRCP)
ALE Plot for SNOW
f_9(x_9)
x_9 (SNOW)
ALE Plot for SNWD
f_10(x_10)
x_10 (SNWD)
ALE Plot for TMAX
f_11(x_11)
ALE Plot for AWND
f_12(x_12)

x_11 (TMAX)   x_12 (AWND)

```
par(mfrow = c(1, 1))
```



5. K Nearest Neighbours

```
set.seed(123)
K <- 5  # K-fold CV
k_values <- seq(3, 40, 2)  # Candidate values for k
n <- nrow(train_data)
y <- train_data$DEP_DEL15
X <- train_data[, colnames(train_data) != "DEP_DEL15"]
CV_metrics <- matrix(0, length(k_values), 4)

Ind <- CVInd(n, K)
for (i in 1:length(k_values)) {
  k <- k_values[i]
  yhat <- factor(rep(NA, n), levels = levels(y))
  for (fold in 1:K) {
    test_idx <- Ind[[fold]]
    train_idx <- setdiff(1:n, test_idx)
    yhat[test_idx] <- knn(train = X[train_idx, ], test = X[test_idx, ], cl = y[train_idx], k = k)
  }

  confusion <- confusionMatrix(yhat, y, positive = "Yes")
  CV_metrics[i, ] <- c(confusion$overall["Accuracy"], confusion$byClass["Precision"], confusion$byClass["Recall"]
, confusion$byClass["F1"])
}

# Identify the best k
best_k_index <- which.max(CV_metrics[, 4])
best_k <- k_values[best_k_index]

# Print the best k
cat("Best k:", best_k, "\n")
```

```r
X_train <- train_data[, colnames(train_data) != "DEP_DEL15"]
y_train <- train_data$DEP_DEL15
X_test <- test_data[, colnames(test_data) != "DEP_DEL15"]
y_test <- test_data$DEP_DEL15

knn_test_preds <- knn(train = X_train, test = X_test, cl = y_train, k = best_k)
conf_matrix_knn <- confusionMatrix(knn_test_preds, y_test, positive = "Yes")

model_results_df <- add_model_results("KNN",
                                      conf_matrix_knn$overall["Accuracy"],
                                      conf_matrix_knn$byClass["Precision"],
                                      conf_matrix_knn$byClass["Recall"],
                                      conf_matrix_knn$byClass["F1"])
```

6. Generalized Additive Model (GAM)

```r
K <- 5  # K-fold CV
n <- nrow(train_data)
y <- train_data$DEP_DEL15
X <- train_data[, colnames(train_data) != "DEP_DEL15"]
CV_metrics <- matrix(0, K, 4)

Ind <- CVInd(n, K)
yhat <- factor(rep(NA, n), levels = levels(y))

numeric_vars <- names(X)[sapply(X, is.numeric)]
categorical_vars <- names(X)[sapply(X, is.factor)]

gam_formula <- as.formula(paste("DEP_DEL15 ~",
                                paste(c(
                                  paste0("s(", numeric_vars, ", k=5)"),
                                  categorical_vars
                                ), collapse = " + ")))

for (fold in 1:K) {
  test_idx <- Ind[[fold]]
  train_idx <- setdiff(1:n, test_idx)

  gam_model <- gam(gam_formula, data = train_data[train_idx, ], family = binomial)
  pred_probs <- predict(gam_model, train_data[test_idx, ], type = "response")
  yhat[test_idx] <- factor(ifelse(pred_probs > 0.5, "Yes", "No"), levels = levels(y))

  confusion <- confusionMatrix(yhat[test_idx], y[test_idx], positive = "Yes")
  CV_metrics[fold, ] <- c(confusion$overall["Accuracy"], confusion$byClass["Precision"], confusion$byClass["Recal
l"], confusion$byClass["F1"])
}
```

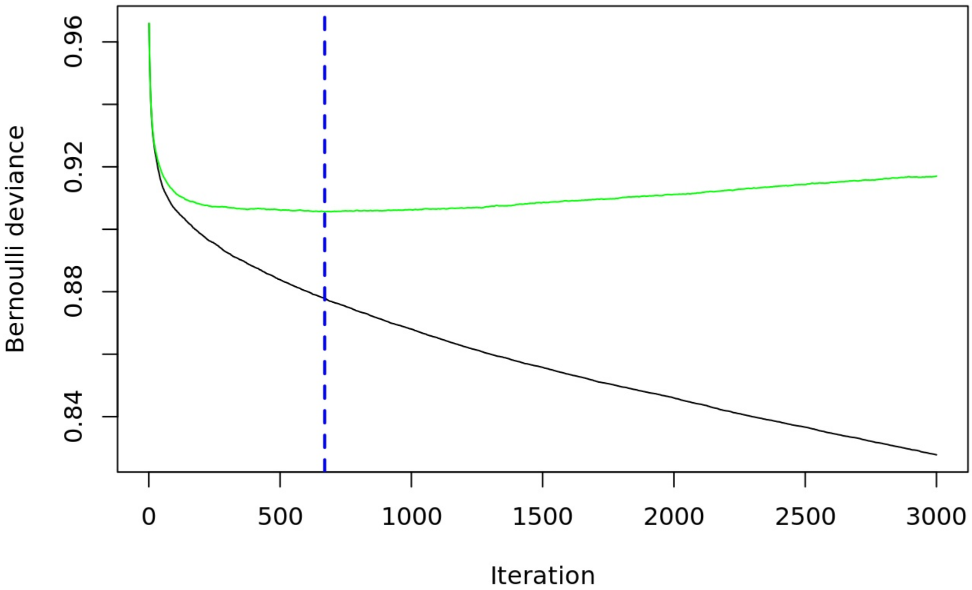
```r
best_gam_model <- gam(gam_formula, data = train_data, family = binomial)
gam_probs <- predict(best_gam_model, test_data, type = "response")
gam_preds <- factor(ifelse(gam_probs > 0.5, "Yes", "No"), levels = levels(test_data$DEP_DEL15))

conf_matrix_gam <- confusionMatrix(gam_preds, test_data$DEP_DEL15, positive = "Yes")

model_results_df <- add_model_results("GAM",
                                      conf_matrix_gam$overall["Accuracy"],
                                      conf_matrix_gam$byClass["Precision"],
                                      conf_matrix_gam$byClass["Recall"],
                                      conf_matrix_gam$byClass["F1"])
```

7. Boosted Tree

Boosted tree has a built in CV already

```r
gbm1 <- gbm(as.numeric(train_data$DEP_DEL15 == "Yes")~., data = train_data, distribution = "bernoulli", n.trees =
3000,
            shrinkage = 0.1, interaction.depth = 3, bag.fraction = 0.5, train.fraction = 1,
            n.minobsinnode = 10, cv.folds = 5, keep.data = TRUE, verbose = FALSE)
best.iter <- gbm.perf(gbm1, method = "cv")
```

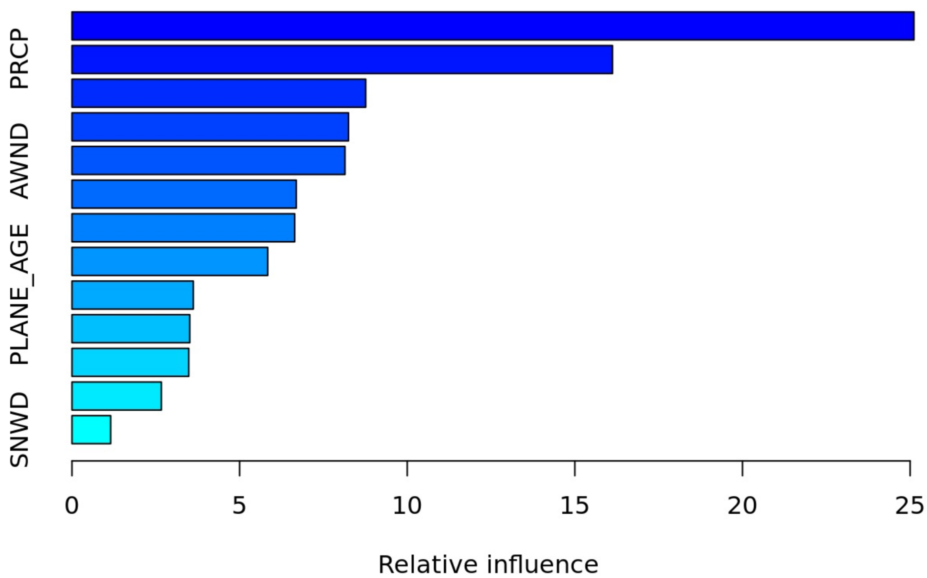
```r
best.iter <- gbm.perf(gbm1, method = "cv")
```




```
best.iter
```

```
## [1] 670
```

```
summary(gbm1, n.trees = best.iter)
```




```
##                                      var   rel.inf
## DEP_TIME_START             DEP_TIME_START 25.115232
## PRCP                                 PRCP 16.122586
## TMAX                                 TMAX  8.761933
## SEGMENT_NUMBER             SEGMENT_NUMBER  8.248157
## AWND                                 AWND  8.143648
## NUMBER_OF_SEATS           NUMBER_OF_SEATS  6.690024
## GROUND_SERV_PER_PASS GROUND_SERV_PER_PASS  6.643598
## MONTH                               MONTH  5.838251
## PLANE_AGE                       PLANE_AGE  3.619657
## DISTANCE_GROUP             DISTANCE_GROUP  3.513701
## SNOW                                 SNOW  3.484108
## DAY_OF_WEEK                   DAY_OF_WEEK  2.665675
## SNWD                                 SNWD  1.153431
```

```
gbm1_prob <- predict(gbm1, test_data, n.trees = best.iter, type = "response")
gbm1_pred <- ifelse(gbm1_prob > 0.5, "Yes", "No")
gbm1_pred <- factor(gbm1_pred, levels = levels(test_data$DEP_DEL15))
conf_matrix_gbm1 <- confusionMatrix(gbm1_pred, test_data$DEP_DEL15, positive = "Yes")

model_results_df <- add_model_results("Boosted Tree",
                                      conf_matrix_gbm1$overall["Accuracy"],
                                      conf_matrix_gbm1$byClass["Precision"],
                                      conf_matrix_gbm1$byClass["Recall"],
                                      conf_matrix_gbm1$byClass["F1"])
```

GBM with weights 2:1

```
class_weights <- ifelse(train_data$DEP_DEL15 == "Yes", 2, 1)
gbm2 <- gbm(as.numeric(train_data$DEP_DEL15 == "Yes") ~ ., data = train_data, distribution = "bernoulli", n.trees
= 3000,
            shrinkage = 0.1, interaction.depth = 3, bag.fraction = 0.5, train.fraction = 1,n.minobsinnode = 10,
            cv.folds = 5, keep.data = TRUE, verbose = FALSE,weights = class_weights)
best.iter2 <- gbm.perf(gbm2, method = "cv")
```

```
gbm2_prob <- predict(gbm2, test_data, n.trees = best.iter2, type = "response")
gbm2_pred <- ifelse(gbm2_prob > 0.5, "Yes", "No")
gbm2_pred <- factor(gbm2_pred, levels = levels(test_data$DEP_DEL15))
conf_matrix_gbm2 <- confusionMatrix(gbm2_pred, test_data$DEP_DEL15, positive = "Yes")

model_results_df <- add_model_results("Boosted Tree Weighted Ratio 2:1",
                                      conf_matrix_gbm2$overall["Accuracy"],
                                      conf_matrix_gbm2$byClass["Precision"],
                                      conf_matrix_gbm2$byClass["Recall"],
                                      conf_matrix_gbm2$byClass["F1"])
```

GBM with weights 3:1

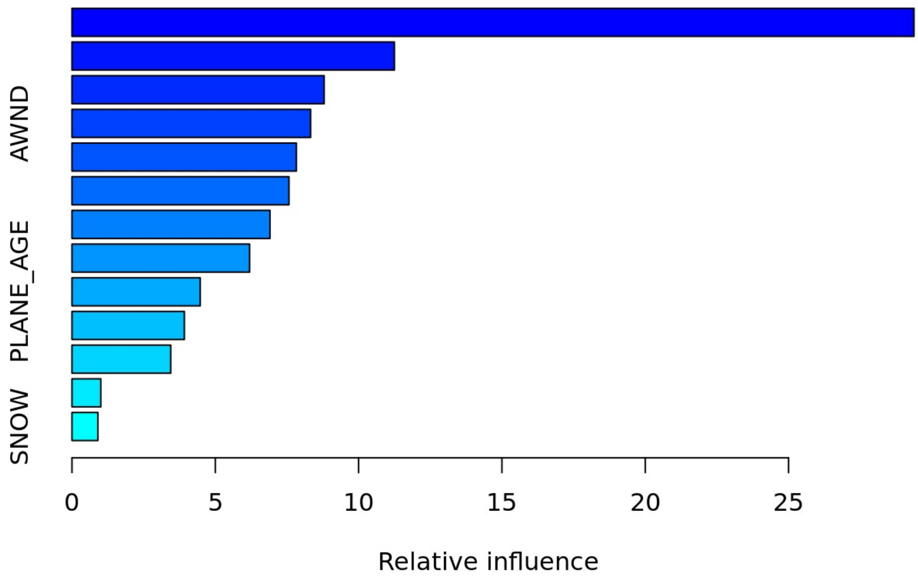
```
class_weights <- ifelse(train_data$DEP_DEL15 == "Yes", 3, 1)
gbm3 <- gbm(as.numeric(train_data$DEP_DEL15 == "Yes") ~ ., data = train_data, distribution = "bernoulli", n.trees
= 3000,
            shrinkage = 0.1, interaction.depth = 3, bag.fraction = 0.5, train.fraction = 1,n.minobsinnode = 10,
            cv.folds = 5, keep.data = TRUE, verbose = FALSE,weights = class_weights)
best.iter3 <- gbm.perf(gbm3, method = "cv")
```

```
gbm3_prob <- predict(gbm3, test_data, n.trees = best.iter3, type = "response")
gbm3_pred <- ifelse(gbm3_prob > 0.5, "Yes", "No")
gbm3_pred <- factor(gbm3_pred, levels = levels(test_data$DEP_DEL15))
conf_matrix_gbm3 <- confusionMatrix(gbm3_pred, test_data$DEP_DEL15, positive = "Yes")

model_results_df <- add_model_results("Boosted Tree Weighted Ratio 3:1",
                                      conf_matrix_gbm3$overall["Accuracy"],
                                      conf_matrix_gbm3$byClass["Precision"],
                                      conf_matrix_gbm3$byClass["Recall"],
                                      conf_matrix_gbm3$byClass["F1"])
```

GBM with weights 4:1

```
class_weights <- ifelse(train_data$DEP_DEL15 == "Yes", 4, 1)
gbm4 <- gbm(as.numeric(train_data$DEP_DEL15 == "Yes") ~ ., data = train_data, distribution = "bernoulli", n.trees
= 3000,
            shrinkage = 0.1, interaction.depth = 3, bag.fraction = 0.5, train.fraction = 1,n.minobsinnode = 10,
            cv.folds = 5, keep.data = TRUE, verbose = FALSE,weights = class_weights)
best.iter4 <- gbm.perf(gbm4, method = "cv")
```

```
summary(gbm4, n.trees = best.iter)
```




```
##                                           var    rel.inf
## DEP_TIME_START             DEP_TIME_START 29.3765314
## PRCP                                 PRCP 11.2486043
## TMAX                                 TMAX  8.7974920
## AWND                                 AWND  8.3244235
## MONTH                               MONTH  7.8286752
## GROUND_SERV_PER_PASS GROUND_SERV_PER_PASS  7.5703899
## NUMBER_OF_SEATS           NUMBER_OF_SEATS  6.9090412
## SEGMENT_NUMBER             SEGMENT_NUMBER  6.1967568
## PLANE_AGE                       PLANE_AGE  4.4751587
## DISTANCE_GROUP             DISTANCE_GROUP  3.9207492
## DAY_OF_WEEK                   DAY_OF_WEEK  3.4441294
## SNWD                                 SNWD  1.0046599
## SNOW                                 SNOW  0.9033883
```

```
gbm4_prob <- predict(gbm4, test_data, n.trees = best.iter3, type = "response")
gbm4_pred <- ifelse(gbm4_prob > 0.5, "Yes", "No")
gbm4_pred <- factor(gbm4_pred, levels = levels(test_data$DEP_DEL15))
conf_matrix_gbm4 <- confusionMatrix(gbm4_pred, test_data$DEP_DEL15, positive = "Yes")

model_results_df <- add_model_results("Boosted Tree Weighted Ratio 4:1",
                                      conf_matrix_gbm4$overall["Accuracy"],
                                      conf_matrix_gbm4$byClass["Precision"],
                                      conf_matrix_gbm4$byClass["Recall"],
                                      conf_matrix_gbm4$byClass["F1"])
```

8. Random Forest

Based on the prof lecture, for RF, the most important hyperparameter is the nodesize. Number of tree won’t make it overfit, unlike the Boosted Tree. For the mtry (number of randomized variable), it is mentioned on the lecture that usually we can just set to 1/3 of the total predictor variables.

```r
# custom f1 score function bcs there is no f1 score on the caret rf
custom_summary <- function(data, lev = NULL, model = NULL) {
  precision <- posPredValue(data$pred, data$obs, positive = "Yes")
  recall <- sensitivity(data$pred, data$obs, positive = "Yes")
  F1 <- ifelse(precision + recall > 0, 2 * (precision * recall) / (precision + recall), 0)

  return(c(Accuracy = mean(data$pred == data$obs), Precision = precision, Recall = recall, F1 = F1))
}

set.seed(123)

train_control <- trainControl(
  method = "cv",
  number = 5,
  classProbs = TRUE,
  summaryFunction = custom_summary,  # Use custom F1 function
  savePredictions = "final"
)

nodesize_values <- c(3, 5, 10, 15, 20)
fixed_mtry <- 4

rf_results <- data.frame(nodesize = nodesize_values, F1 = NA)

for (i in 1:length(nodesize_values)) {

  rf_model <- train(
    DEP_DEL15 ~ .,
    data = train_data,
    method = "rf",
    trControl = train_control,
    tuneGrid = expand.grid(mtry = fixed_mtry),
    metric = "F1",
    importance = TRUE,
    nodesize = nodesize_values[i],
    ntree = 500
  )

  rf_results$F1[i] <- max(rf_model$results$F1, na.rm = TRUE)
}

# Find the best nodesize based on highest F1-score
best_nodesize <- rf_results$nodesize[which.max(rf_results$F1)]

# Print results
cat("Best nodesize:", best_nodesize, "\n")
```

```r
rf_model <- randomForest(DEP_DEL15 ~ ., data = train_data, mtry = 4, ntree = 500, nodesize = best_nodesize, impor
tance = TRUE)
plot(rf_model)
```

```r
importance(rf_model)
```

```
##                              No        Yes MeanDecreaseAccuracy MeanDecreaseGini
## MONTH                 46.665682   1.211133             49.06817        965.33782
## DAY_OF_WEEK            8.352290   9.666601             12.06687        968.22392
## DISTANCE_GROUP        59.617093 -20.550144             51.14339        970.71923
## SEGMENT_NUMBER        98.181442 -75.767962             96.01273        769.64417
## NUMBER_OF_SEATS       70.498757 -24.805815             66.75949       1170.66256
## GROUND_SERV_PER_PASS  60.359977 -12.679712             59.06658        867.92560
## PLANE_AGE             51.630390 -19.827828             43.72597       1469.73670
## PRCP                  35.676266  45.128582             53.83092        952.25461
## SNOW                  18.062501   8.919377             22.76465        105.31242
## SNWD                   9.695626   5.280322             11.92881         88.99459
## TMAX                  46.114665  -8.529692             48.21230       1946.21209
## AWND                  18.704049  12.209826             23.50755       1913.50862
## DEP_TIME_START       108.752236 -41.461648            125.76073       1422.28710
```

```
rf_test_preds <- predict(rf_model, test_data, type = "class")
conf_matrix_rf <- confusionMatrix(as.factor(rf_test_preds), as.factor(test_data$DEP_DEL15), positive = "Yes")

model_results_df <- add_model_results("Random Forest",
                                      conf_matrix_rf$overall["Accuracy"],
                                      conf_matrix_rf$byClass["Precision"],
                                      conf_matrix_rf$byClass["Recall"],
                                      conf_matrix_rf$byClass["F1"])
```

Try weighted RF Model

```
rf_model_weighted <- randomForest(DEP_DEL15 ~ ., data = train_data,
                                  mtry = 4, ntree = 500, nodesize = best_nodesize,
                                  importance = TRUE, classwt = c("No" = 1, "Yes" = 2))

rf_test_preds_weighted <- predict(rf_model_weighted, test_data, type = "class")
conf_matrix_rf2 <- confusionMatrix(as.factor(rf_test_preds_weighted), as.factor(test_data$DEP_DEL15), positive =
"Yes")

model_results_df <- add_model_results("Random Forest Weighted Ratio 2:1",
                                      conf_matrix_rf2$overall["Accuracy"],
                                      conf_matrix_rf2$byClass["Precision"],
                                      conf_matrix_rf2$byClass["Recall"],
                                      conf_matrix_rf2$byClass["F1"])
```

```
rf_model_weighted3 <- randomForest(DEP_DEL15 ~ ., data = train_data,
                                  mtry = 4, ntree = 500, nodesize = best_nodesize,
                                  importance = TRUE, classwt = c("No" = 1, "Yes" = 3))

rf_test_preds_weighted3 <- predict(rf_model_weighted3, test_data, type = "class")
conf_matrix_rf3 <- confusionMatrix(as.factor(rf_test_preds_weighted3), as.factor(test_data$DEP_DEL15), positive =
"Yes")

model_results_df <- add_model_results("Random Forest Weighted Ratio 3:1",
                                      conf_matrix_rf3$overall["Accuracy"],
                                      conf_matrix_rf3$byClass["Precision"],
                                      conf_matrix_rf3$byClass["Recall"],
                                      conf_matrix_rf3$byClass["F1"])
```

```
rf_model_weighted4 <- randomForest(DEP_DEL15 ~ ., data = train_data,
                                  mtry = 4, ntree = 500, nodesize = best_nodesize,
                                  importance = TRUE, classwt = c("No" = 1, "Yes" = 4))

rf_test_preds_weighted4 <- predict(rf_model_weighted4, test_data, type = "class")
conf_matrix_rf4 <- confusionMatrix(as.factor(rf_test_preds_weighted4), as.factor(test_data$DEP_DEL15), positive =
"Yes")

model_results_df <- add_model_results("Random Forest Weighted Ratio 3:1",
                                      conf_matrix_rf4$overall["Accuracy"],
                                      conf_matrix_rf4$byClass["Precision"],
                                      conf_matrix_rf4$byClass["Recall"],
                                      conf_matrix_rf4$byClass["F1"])
```

```
importance(rf_model_weighted4)
```

```
##                             No        Yes MeanDecreaseAccuracy MeanDecreaseGini
## MONTH               -41.653942  60.701544          -11.9285941       1155.08056
## DAY_OF_WEEK          -1.477843  16.883133            6.3164044       1146.28319
## DISTANCE_GROUP      -17.368891  22.005448           -6.8709205       1103.58279
## SEGMENT_NUMBER       45.677742 -23.022057           46.6508213        723.75263
## NUMBER_OF_SEATS     -26.268686  47.300663           -6.8130616       1387.76806
## GROUND_SERV_PER_PASS -16.792325  32.355629           -1.2387320       1028.18426
## PLANE_AGE           -21.025805  26.289885           -9.4833345       1791.00083
## PRCP                -10.128096  52.639884           20.5885484        857.14752
## SNOW                -16.784694  20.577288           -6.6832286         76.14540
## SNWD                -11.273417   9.303987           -7.5285044         86.83097
## TMAX                -55.385279  66.843059          -26.1223035       2271.25186
## AWND                -17.605076  31.647628            0.5568613       2226.97649
## DEP_TIME_START       58.903262  13.554315           79.5356013       1652.98001
```

9. XGBoost

```r
# XGBoost requires matrix input
train_matrix <- model.matrix(DEP_DEL15 ~ ., data = train_data)[,-1]  # Remove intercept
train_label <- ifelse(train_data$DEP_DEL15 == "Yes", 1, 0)

# Convert to XGBoost format
dtrain <- xgb.DMatrix(data = train_matrix, label = train_label)
```

```r
params <- list(
  objective = "binary:logistic",
  eval_metric = "logloss",
  max_depth = 3,
  eta = 0.1,
  subsample = 0.8,
  colsample_bytree = 0.8
)

xgb_cv <- xgb.cv(
  params = params,
  data = dtrain,
  nrounds = 500,
  nfold = 5,
  stratified = TRUE,
  verbose = 0,
  early_stopping_rounds = 10
)

# best iter
best_nrounds <- xgb_cv$best_iteration
print(best_nrounds)
```

```r
xgb_model <- xgboost(
  params = params,
  data = dtrain,
  nrounds = best_nrounds,
  verbose = 0
)

# Convert test data into XGBoost matrix
test_matrix <- model.matrix(DEP_DEL15 ~ ., data = test_data)[,-1]
test_label <- as.numeric(test_data$DEP_DEL15 == "Yes")
dtest <- xgb.DMatrix(data = test_matrix, label = test_label)
xgb_probs <- predict(xgb_model, dtest)
xgb_preds <- ifelse(xgb_probs > 0.5, "Yes", "No")
xgb_preds <- factor(xgb_preds, levels = levels(test_data$DEP_DEL15))
conf_matrix_xgb <- confusionMatrix(xgb_preds, test_data$DEP_DEL15, positive = "Yes")

model_results_df <- add_model_results("XGBoost",
                                      conf_matrix_xgb$overall["Accuracy"],
                                      conf_matrix_xgb$byClass["Precision"],
                                      conf_matrix_xgb$byClass["Recall"],
                                      conf_matrix_xgb$byClass["F1"])
```

Try Weighted XGBoost Weight 2:1

```
params <- list(
  objective = "binary:logistic",
  eval_metric = "logloss",
  max_depth = 3,
  eta = 0.1,
  subsample = 0.8,
  colsample_bytree = 0.8,
  scale_pos_weight = 2
)

xgb_cv_weighted <- xgb.cv(
  params = params,
  data = dtrain,
  nrounds = 500,
  nfold = 5,
  stratified = TRUE,
  verbose = 0,
  early_stopping_rounds = 10
)

best_nrounds_weighted <- xgb_cv_weighted$best_iteration

# train the best nrounds
xgb_model_weighted <- xgboost(
  params = params,
  data = dtrain,
  nrounds = best_nrounds_weighted,
  verbose = 0
)
```

```
# Convert test data into XGBoost matrix
test_matrix <- model.matrix(DEP_DEL15 ~ ., data = test_data)[,-1]
test_label <- as.numeric(test_data$DEP_DEL15 == "Yes")
dtest <- xgb.DMatrix(data = test_matrix, label = test_label)

xgb_probs_weighted <- predict(xgb_model_weighted, dtest)
xgb_preds_weighted <- ifelse(xgb_probs_weighted > 0.5, "Yes", "No")
xgb_preds_weighted <- factor(xgb_preds_weighted, levels = levels(test_data$DEP_DEL15))

conf_matrix_xgb_weighted <- confusionMatrix(xgb_preds_weighted, test_data$DEP_DEL15, positive = "Yes")

model_results_df <- add_model_results("XGBoost Weighted Ratio 2:1",
                                      conf_matrix_xgb_weighted$overall["Accuracy"],
                                      conf_matrix_xgb_weighted$byClass["Precision"],
                                      conf_matrix_xgb_weighted$byClass["Recall"],
                                      conf_matrix_xgb_weighted$byClass["F1"])
```

Try Weighted XGBoost Weight 3:1

```r
params <- list(
  objective = "binary:logistic",
  eval_metric = "logloss",
  max_depth = 3,
  eta = 0.1,
  subsample = 0.8,
  colsample_bytree = 0.8,
  scale_pos_weight = 3
)

xgb_cv_weighted <- xgb.cv(
  params = params,
  data = dtrain,
  nrounds = 500,
  nfold = 5,
  stratified = TRUE,
  verbose = 0,
  early_stopping_rounds = 10
)

best_nrounds_weighted <- xgb_cv_weighted$best_iteration

# train the best nrounds
xgb_model_weighted <- xgboost(
  params = params,
  data = dtrain,
  nrounds = best_nrounds_weighted,
  verbose = 0
)
```

```r
# Convert test data into XGBoost matrix
test_matrix <- model.matrix(DEP_DEL15 ~ ., data = test_data)[,-1]
test_label <- as.numeric(test_data$DEP_DEL15 == "Yes")
dtest <- xgb.DMatrix(data = test_matrix, label = test_label)

xgb_probs_weighted <- predict(xgb_model_weighted, dtest)
xgb_preds_weighted <- ifelse(xgb_probs_weighted > 0.5, "Yes", "No")
xgb_preds_weighted <- factor(xgb_preds_weighted, levels = levels(test_data$DEP_DEL15))

conf_matrix_xgb_weighted <- confusionMatrix(xgb_preds_weighted, test_data$DEP_DEL15, positive = "Yes")

model_results_df <- add_model_results("XGBoost Weighted Ratio 3:1",
                                      conf_matrix_xgb_weighted$overall["Accuracy"],
                                      conf_matrix_xgb_weighted$byClass["Precision"],
                                      conf_matrix_xgb_weighted$byClass["Recall"],
                                      conf_matrix_xgb_weighted$byClass["F1"])
```

Try Weighted XGBoost Weight 4:1

```
params <- list(
  objective = "binary:logistic",
  eval_metric = "logloss",
  max_depth = 3,
  eta = 0.1,
  subsample = 0.8,
  colsample_bytree = 0.8,
  scale_pos_weight = 4
)

xgb_cv_weighted <- xgb.cv(
  params = params,
  data = dtrain,
  nrounds = 500,
  nfold = 5,
  stratified = TRUE,
  verbose = 0,
  early_stopping_rounds = 10
)

best_nrounds_weighted <- xgb_cv_weighted$best_iteration

# train the best nrounds
xgb_model_weighted <- xgboost(
  params = params,
  data = dtrain,
  nrounds = best_nrounds_weighted,
  verbose = 0
)
```

```
# Convert test data into XGBoost matrix
test_matrix <- model.matrix(DEP_DEL15 ~ ., data = test_data)[,-1]
test_label <- as.numeric(test_data$DEP_DEL15 == "Yes")
dtest <- xgb.DMatrix(data = test_matrix, label = test_label)

xgb_probs_weighted <- predict(xgb_model_weighted, dtest)
xgb_preds_weighted <- ifelse(xgb_probs_weighted > 0.5, "Yes", "No")
xgb_preds_weighted <- factor(xgb_preds_weighted, levels = levels(test_data$DEP_DEL15))

conf_matrix_xgb_weighted <- confusionMatrix(xgb_preds_weighted, test_data$DEP_DEL15, positive = "Yes")

model_results_df <- add_model_results("XGBoost Weighted Ratio 4:1",
                                      conf_matrix_xgb_weighted$overall["Accuracy"],
                                      conf_matrix_xgb_weighted$byClass["Precision"],
                                      conf_matrix_xgb_weighted$byClass["Recall"],
                                      conf_matrix_xgb_weighted$byClass["F1"])
```

```
feature_names <- colnames(dtrain)
importance_matrix <- xgb.importance(feature_names = feature_names, model = xgb_model_weighted)
print(importance_matrix)
```

```
##                 Feature        Gain      Cover  Frequency
##                  <char>       <num>      <num>      <num>
##  1:      DEP_TIME_START 0.313081033 0.09455181 0.10156581
##  2:                PRCP 0.130172254 0.09901154 0.08675413
##  3:      SEGMENT_NUMBER 0.095261097 0.05802716 0.06644096
##  4:               MONTH 0.075538088 0.08406023 0.08336860
##  5:                TMAX 0.075071001 0.12764090 0.13711384
##  6: GROUND_SERV_PER_PASS 0.072176389 0.10572095 0.09098603
##  7:     NUMBER_OF_SEATS 0.068943901 0.10899330 0.11172239
##  8:                AWND 0.062351895 0.11728749 0.12103259
##  9:      DISTANCE_GROUP 0.032421892 0.04001193 0.04655099
## 10:           PLANE_AGE 0.029011573 0.06354122 0.06686416
## 11:         DAY_OF_WEEK 0.025181095 0.05350339 0.04909014
## 12:                SNOW 0.011590278 0.03120789 0.02327550
## 13:                SNWD 0.009199504 0.01644219 0.01523487
```

```
model_results_df
```

```
##                                      Model Accuracy Precision Recall F1_Score
## Accuracy                Logistic Regression   0.8092    0.4314 0.0090   0.0175
## Accuracy1                     Decision Tree   0.8113    0.6786 0.0155   0.0302
## Accuracy2  Decision Tree Weighted Ration 2:1   0.8022    0.4378 0.1404   0.2126
## Accuracy3  Decision Tree Weighted Ration 3:1   0.7180    0.3145 0.4089   0.3555
## Accuracy4  Decision Tree Weighted Ration 4:1   0.6506    0.2847 0.5533   0.3760
## Accuracy5                    Neural Network   0.8116    0.5645 0.0427   0.0794
## Accuracy6                               KNN   0.7690    0.3014 0.1627   0.2114
## Accuracy7                               GAM   0.8106    0.5632 0.0199   0.0385
## Accuracy8                      Boosted Tree   0.8119    0.5603 0.0529   0.0967
## Accuracy9    Boosted Tree Weighted Ratio 2:1   0.7980    0.4324 0.1977   0.2714
## Accuracy10   Boosted Tree Weighted Ratio 3:1   0.7444    0.3515 0.4068   0.3771
## Accuracy11   Boosted Tree Weighted Ratio 4:1   0.6585    0.2981 0.5867   0.3953
## Accuracy12                    Random Forest   0.8127    0.5633 0.0688   0.1226
## Accuracy13  Random Forest Weighted Ratio 2:1   0.8112    0.5215 0.0887   0.1516
## Accuracy14  Random Forest Weighted Ratio 3:1   0.8118    0.5327 0.0862   0.1485
## Accuracy15  Random Forest Weighted Ratio 3:1   0.8112    0.5238 0.0850   0.1463
## Accuracy16                          XGBoost   0.8140    0.6571 0.0468   0.0874
## Accuracy17        XGBoost Weighted Ratio 2:1   0.7982    0.4311 0.1896   0.2634
## Accuracy18        XGBoost Weighted Ratio 3:1   0.7420    0.3456 0.3983   0.3701
## Accuracy19        XGBoost Weighted Ratio 4:1   0.6609    0.2995 0.5846   0.3961
```

```r
# Load DT package
library(DT)

# Display table with pagination
datatable(model_results_df,
          options = list(pageLength = nrow(model_results_df)),  # Show all rows
          rownames = FALSE)
```

Show 10 entries   Search:

Model	Accuracy	Precision	Recall	F1_Score
Logistic Regression	0.8092	0.4314	0.009	0.0175
Decision Tree	0.8113	0.6786	0.0155	0.0302
Decision Tree Weighted Ration 2:1	0.8022	0.4378	0.1404	0.2126
Decision Tree Weighted Ration 3:1	0.718	0.3145	0.4089	0.3555
Decision Tree Weighted Ration 4:1	0.6506	0.2847	0.5533	0.376
Neural Network	0.8116	0.5645	0.0427	0.0794
KNN	0.769	0.3014	0.1627	0.2114
GAM	0.8106	0.5632	0.0199	0.0385
Boosted Tree	0.8119	0.5603	0.0529	0.0967
Boosted Tree Weighted Ratio 2:1	0.798	0.4324	0.1977	0.2714
Boosted Tree Weighted Ratio 3:1	0.7444	0.3515	0.4068	0.3771
Boosted Tree Weighted Ratio 4:1	0.6585	0.2981	0.5867	0.3953
Random Forest	0.8127	0.5633	0.0688	0.1226
Random Forest Weighted Ratio 2:1	0.8112	0.5215	0.0887	0.1516
Random Forest Weighted Ratio 3:1	0.8118	0.5327	0.0862	0.1485
Random Forest Weighted Ratio 3:1	0.8112	0.5238	0.085	0.1463
XGBoost	0.814	0.6571	0.0468	0.0874
XGBoost Weighted Ratio 2:1	0.7982	0.4311	0.1896	0.2634
XGBoost Weighted Ratio 3:1	0.742	0.3456	0.3983	0.3701
XGBoost Weighted Ratio 4:1	0.6609	0.2995	0.5846	0.3961

Showing 1 to 20 of 20 entries   Previous 1 Next