Code

1 Credit Card Analogy

Goal: Identify customers likely to default on loan. Positive (1): Customer predicted to default Negative (0): Customer predicted not to default

False Positive (FP): Predicts default, but customer actually pays → Moderate cost (lost interest or customer trust if credit is restricted unnecessarily) False Negative (FN): Predicts no default, but customer actually defaults → High cost (financial loss due to unpaid debt)

Strategy: Since missing actual defaulters (false negatives) is very costly, we keep the classification threshold higher to be more conservative. This reduces false negatives (missed defaulters), even if it slightly increases false positives (good customers flagged as risky).

Goal:

Reduce false negatives — you don't want to miss people who will actually default. We should decrease the classification threshold to catch more potential defaulters — even if it means mistakenly flagging some safe customers.

1.1 Load the Dataset

Hide

```
holdout=readRDS("holdout_df_Boost.rds")
```

1.2 Performance Comparison Table Across All Models with Accuracy,Sensitivity,Specificity,F1 and AUC

Hide

```
# Extract true labels
  truth <- holdout$loan_default

# Get model names based on .prob columns
  model_names <- names(holdout) %>%
  stringr::str_subset("\\.prob$") %>%
  stringr::str_remove("\\.prob$")

    get_metricss <- function(model_name) {

    }
# Function to compute metrics for each model
  get_metrics <- function(model_name) {
  prob_col <- paste0(model_name, ".prob")
  class_col <- paste0(model_name, ".class")

    # Extract predictions
  prob <- holdout[[prob_col]]
  pred <- factor(holdout[[class_col]], levels = c("No", "Yes"))

    # Confusion matrix
  cm <- caret::confusionMatrix(pred, truth, positive = "Yes")

  # AUC
  roc_obj <- pROC::roc(truth, prob, quiet = TRUE)

  # Metrics
  data.frame(
    Model = model_name,
    Accuracy = cm$overall["Accuracy"],
    Sensitivity = cm$byClass["Sensitivity"],
    Specificity = cm$byClass["Specificity"],
    F1 = MLmetrics::F1_Score(
      y_pred = pred,
      y_true = truth,
      positive = "Yes"
    ),
    AUC = as.numeric(roc_obj$auc)
  )
}

# Apply to all models
results_df <- dplyr::bind_rows(lapply(model_names, get_metrics))

# Remove row names and display
```

```
rownames(results_df) <- NULL
print(results_df)
```

```
##     Model  Accuracy Sensitivity Specificity        F1       AUC
## 1 default 0.8209913  0.06910317   1.0000000 0.1292732 0.5345516
## 2 entropy 0.8209913  0.06910317   1.0000000 0.1292732 0.6567661
## 3  ostree 0.6554563  0.63775896   0.6596696 0.4158514 0.6903745
## 4  pruned 0.6632122  0.65737491   0.6646020 0.4287959 0.7153128
## 5   rf_cv 0.8223546  0.10785326   0.9924622 0.1892969 0.7001855
## 6     xgb 0.8236516  0.14130360   0.9861043 0.2355709 0.7414753
```

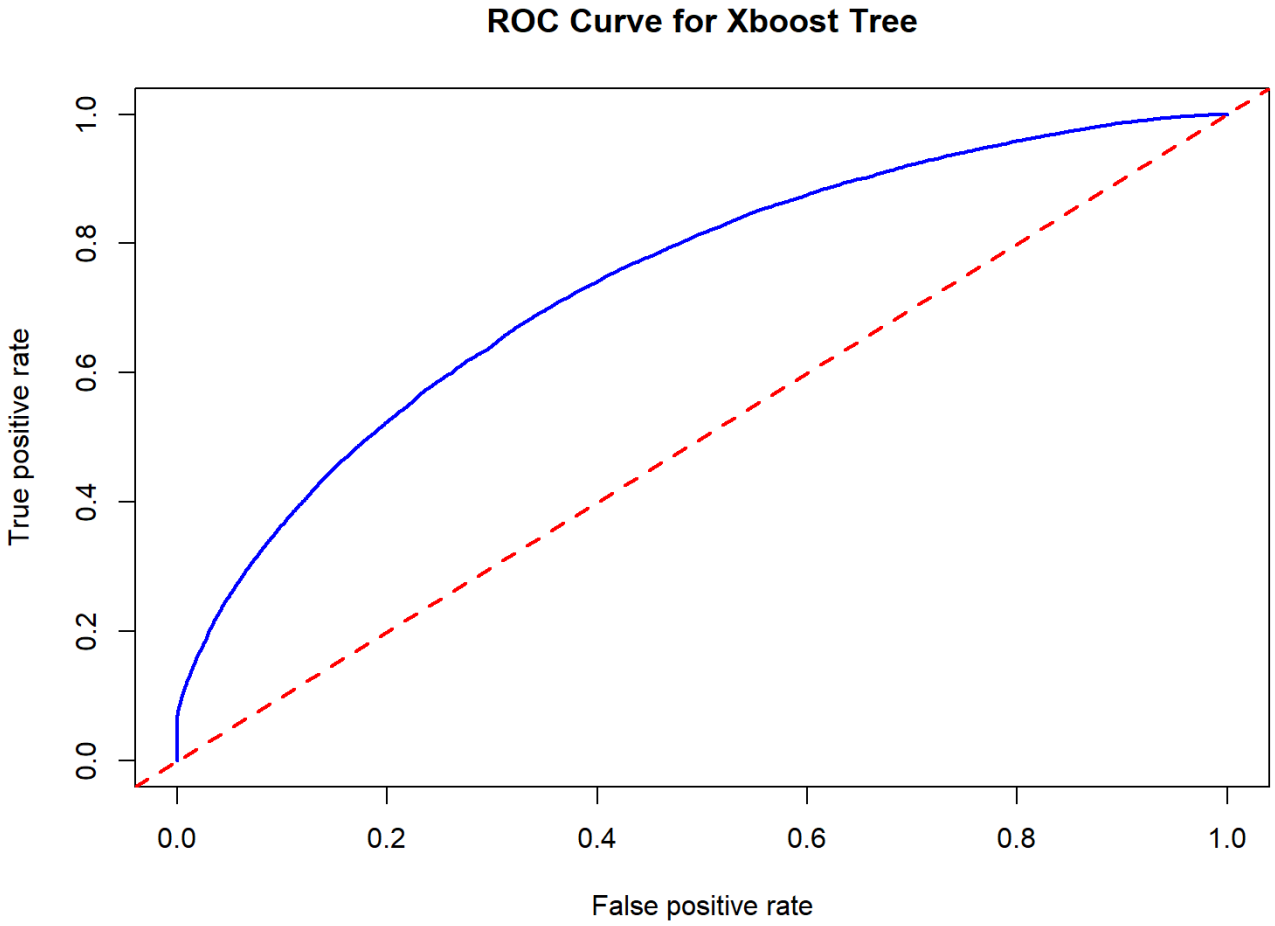
1.3 Analysis:

By seeing the table Xgb has highest AUC value(74%) and next highest is rf_cv(70%).

1.4 ROC Curve for XBoost Tree and Confusion Matrix

Hide

```
pred <- ROCR::prediction(holdout$xgb.prob,holdout$loan_default)
perf <- ROCR::performance(pred, "tpr", "fpr")
# Plot Curve
plot(perf, col = "blue", lwd = 2, main = "ROC Curve for Xboost Tree")
abline(a = 0, b = 1, col = "red", lty = 2, lwd = 2)
```



Hide

```
confusionMatrix(holdout$xgb.class,
                holdout$loan_default,positive="Yes")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    No   Yes
##        No  60178 12476
##        Yes   848  2053
##
##                Accuracy : 0.8237
##                  95% CI : (0.8209, 0.8264)
##     No Information Rate : 0.8077
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.1833
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.14130
##             Specificity : 0.98610
##          Pos Pred Value : 0.70769
##          Neg Pred Value : 0.82828
##              Prevalence : 0.19230
##          Detection Rate : 0.02717
##    Detection Prevalence : 0.03840
##       Balanced Accuracy : 0.56370
##
##        'Positive' Class : Yes
##
```

1.5 Analysis:

FP: 12476 and TP:2053 Specificity : 98% Sensitivity : 14% Accuracy : 82% and Balanced Accuracy: 56%

1.6 Step 1- Finding the Optimal Threshold using ROCR package

Generally we use the F1 Score to find the optimal threshold. Using the ROCR package, we can compute the precision and recall for each threshold We use this vector of values to compute the F1 Score for each threshold. The threshold that gives the highest F1 Score is the optimal threshold.

Hide

```
pred <- ROCR::prediction(holdout$xgb.prob,holdout$loan_default)
prec <- ROCR::performance(pred, "prec")
rec <- ROCR::performance(pred, "rec")
precision <- prec@y.values[[1]]
recall <- rec@y.values[[1]]
f1=2*precision*recall/(precision+recall)
f1[is.nan(f1)]=0
cutoffs=prec@x.values[[1]]
df_f1=data.frame(f1,cutoffs)

opt_idx <- which.max(f1)
opt_f1 <- df_f1[opt_idx,]
opt_f1
```

	f1 <dbl>	cutoffs <dbl>
22640	0.4474334	0.2266894

1 row

Hide

```
print(opt_f1$cutoffs)
```

```
## [1] 0.2266894
```

1.7 Analysis

Our best possible cutoff is 0.2266894.

1.8 Step-2 Finding the model performance Using Youden's J statistic

Hide

```
pred <- ROCR::prediction(holdout$xgb.prob,holdout$loan_default)
perf=performance(pred,"tpr","fpr")
tpr=perf@y.values[[1]]
fpr=perf@x.values[[1]]
thr=perf@alpha.values[[1]]
j=tpr-fpr
best_j=thr[which.max(j)]
print(best_j)
```

```
## [1] 0.1886004
```

1.9 ## Analysis

Our best possible cutoff is 0.1886004.

1.10 Comparision of Two optimal cutoff for selecting the best performance model based on the confusion matrix

We are curious to know with the two optimal cutoffs impact on tp,fn values.We checked with the two optimal cutoffs of 0.18 and 0.23. Both of the thersholds either decreasing amount of tp or increasing fp. so we made the in between level 0.2 as possible thershold.The code given below reveals the best accuracy rate for loan_defaults.

Hide

```
holdout$xgbopt_cutoff.class <- factor(ifelse(holdout$xgb.prob >=  0.2, "Yes",
"No"), levels = c("No", "Yes"))
holdout$xgbopt_cutoff.prob <- holdout$xgb.prob

confusionMatrix(holdout$xgb.class,
                holdout$loan_default,positive="Yes")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    No   Yes
##        No  60178 12476
##        Yes   848  2053
##
##                Accuracy : 0.8237
##                  95% CI : (0.8209, 0.8264)
##     No Information Rate : 0.8077
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.1833
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.14130
##             Specificity : 0.98610
##          Pos Pred Value : 0.70769
##          Neg Pred Value : 0.82828
##              Prevalence : 0.19230
##          Detection Rate : 0.02717
##    Detection Prevalence : 0.03840
##       Balanced Accuracy : 0.56370
##
##        'Positive' Class : Yes
##
```

Hide

```
confusionMatrix(holdout$xgbopt_cutoff.class,
                holdout$loan_default,positive="Yes")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    No   Yes
##        No  42829  5232
##        Yes 18197  9297
##
##               Accuracy : 0.6899
##                 95% CI : (0.6866, 0.6932)
##    No Information Rate : 0.8077
##    P-Value [Acc > NIR] : 1
##
##                  Kappa : 0.255
##
##  Mcnemar's Test P-Value : <2e-16
##
##            Sensitivity : 0.6399
##            Specificity : 0.7018
##         Pos Pred Value : 0.3381
##         Neg Pred Value : 0.8911
##             Prevalence : 0.1923
##         Detection Rate : 0.1230
##   Detection Prevalence : 0.3639
##      Balanced Accuracy : 0.6709
##
##       'Positive' Class : Yes
##
```

Hide

```
# Extract true labels
  truth <- holdout$loan_default

# Get model names based on .prob columns
  model_names <- names(holdout) %>%
  stringr::str_subset("\\.prob$") %>%
  stringr::str_remove("\\.prob$")

    get_metricss <- function(model_name) {

    }
# Function to compute metrics for each model
  get_metrics <- function(model_name) {
  prob_col <- paste0(model_name, ".prob")
  class_col <- paste0(model_name, ".class")

    # Extract predictions
  prob <- holdout[[prob_col]]
  pred <- factor(holdout[[class_col]], levels = c("No", "Yes"))

    # Confusion matrix
  cm <- caret::confusionMatrix(pred, truth, positive = "Yes")

  # AUC
  roc_obj <- pROC::roc(truth, prob, quiet = TRUE)

  # Metrics
  data.frame(
    Model = model_name,
    Accuracy = cm$overall["Accuracy"],
    Sensitivity = cm$byClass["Sensitivity"],
    Specificity = cm$byClass["Specificity"],
    F1 = MLmetrics::F1_Score(
      y_pred = pred,
      y_true = truth,
      positive = "Yes"
    ),
    AUC = as.numeric(roc_obj$auc)
  )
}

# Apply to all models
results_df <- dplyr::bind_rows(lapply(model_names, get_metrics))

# Remove row names and display
```

```
rownames(results_df) <- NULL
print(results_df)
```

```
##           Model  Accuracy Sensitivity Specificity        F1       AUC
## 1       default 0.8209913  0.06910317   1.0000000 0.1292732 0.5345516
## 2       entropy 0.8209913  0.06910317   1.0000000 0.1292732 0.6567661
## 3        ostree 0.6554563  0.63775896   0.6596696 0.4158514 0.6903745
## 4        pruned 0.6632122  0.65737491   0.6646020 0.4287959 0.7153128
## 5         rf_cv 0.8223546  0.10785326   0.9924622 0.1892969 0.7001855
## 6           xgb 0.8236516  0.14130360   0.9861043 0.2355709 0.7414753
## 7 xgbopt_cutoff 0.6899080  0.63989263   0.7018156 0.4424720 0.7414753
```

1.11 Result

By reviewing all the table metrics, xgbopt_cutoff has best in capturing the tp(Actual Defaulters),F1(capturing defaulters and avoiding false alarms) and AUC(Identify the difference between defaulting and non-defaulting borrowers)

Recall = 0.64 → catches ~64% of actual defaulters

F1 = 0.44 → strong balance of precision & recall

AUC = 0.74 → reliable ranking capability

Best Model for Loan Defaults: xgbopt_cutoff