Improved Modeling for Churn Prediction

Andres Perez

Improved Modeling for Churn Prediction

This document covers advanced modeling steps to improve churn prediction, including class imbalance handling, hyperparameter tuning, cross-validation, and threshold optimization.

1. Data Loading

```
data <- read.csv("data/EngineeredChurnData.csv")
# Use only the top 10 features (plus Churn)
selected_features <- c(
  "Customer.Months",
  "Days.Since.Last.Login",
  "CHI.Score.Mon0",
  "Activity_Score",
  "CHI.Score",
  "Logins",
  "Views_log",
  "Logins_log",
  "Views",
  "Login_View_Interaction",
  "Churn"
)
selected_features <- intersect(selected_features, colnames(data))
data <- data[, selected_features, drop=FALSE]
# Remove rows with NA/NaN/Inf
data <- data[complete.cases(data) & apply(data, 1, function(row) all(is.finite(as.n
        umeric(row)))), ]
if(!is.factor(data$Churn)) data$Churn <- as.factor(data$Churn)
```

2. Train-Test Split

```
set.seed(123)
train_index <- createDataPartition(data$Churn, p = 0.8, list = FALSE)
train_data <- data[train_index, ]
test_data <- data[-train_index, ]
```

3. Address Class Imbalance

```
# Use ROSE for class balancing
rose_data <- ROSE(Churn ~ ., data = train_data, seed = 123)$data
table(rose_data$Churn)
```

```
##
##    0    1
## 1208 1179
```

4. Hyperparameter Tuning & Cross-Validation (XGBoost Example)

```
set.seed(123)
xgb_grid <- expand.grid(
  nrounds = c(100, 200),
  max_depth = c(3, 5, 7),
  eta = c(0.01, 0.1, 0.3),
  gamma = 0,
  colsample_bytree = 1,
  min_child_weight = 1,
  subsample = 1
)
train_control <- trainControl(
  method = "cv",
  number = 5,
  summaryFunction = twoClassSummary,
  classProbs = TRUE,
  verboseIter = FALSE
)
rose_data$Churn <- factor(ifelse(rose_data$Churn == 1, "yes", "no"), levels = c("n
        o", "yes"))
test_data$Churn <- factor(ifelse(test_data$Churn == 1, "yes", "no"), levels = c("n
        o", "yes"))
invisible(capture.output(
  xgb_tuned <- train(
    Churn ~ .,
    data = rose_data,
    method = "xgbTree",
    trControl = train_control,
    tuneGrid = xgb_grid,
    metric = "Sens",
    verbose = 0
  ),
  file = '/dev/null'
))
```

```r
xgb_tuned
```

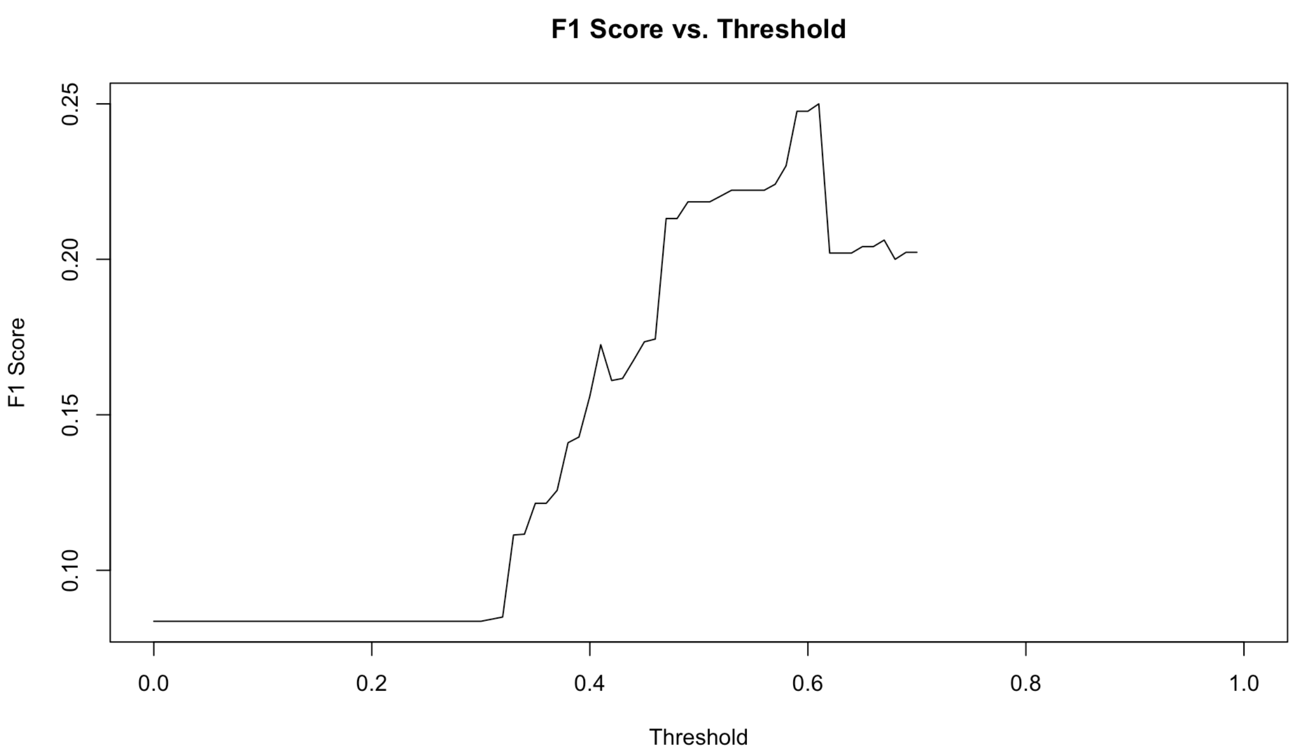
```
## eXtreme Gradient Boosting
##
## 2387 samples
##   10 predictor
##    2 classes: 'no', 'yes'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 1909, 1910, 1909, 1910, 1910
## Resampling results across tuning parameters:
##
##   eta   max_depth  nrounds  ROC        Sens       Spec
##   0.01  3          100      0.8091334  0.8104660  0.6403750
##   0.01  3          200      0.8267526  0.7839855  0.6870501
##   0.01  5          100      0.8275745  0.7533315  0.7396069
##   0.01  5          200      0.8372364  0.7516649  0.7565525
##   0.01  7          100      0.8352164  0.7582868  0.7472340
##   0.01  7          200      0.8407705  0.7549570  0.7599387
##   0.10  3          100      0.8514458  0.7574363  0.7734908
##   0.10  3          200      0.8564480  0.7632214  0.7862099
##   0.10  5          100      0.8528194  0.7665409  0.7819834
##   0.10  5          200      0.8523785  0.7706800  0.7743635
##   0.10  7          100      0.8558034  0.7640513  0.7743455
##   0.10  7          200      0.8539429  0.7607387  0.7726578
##   0.30  3          100      0.8547996  0.7623744  0.7743419
##   0.30  3          200      0.8460564  0.7574260  0.7692571
##   0.30  5          100      0.8467940  0.7557628  0.7616264
##   0.30  5          200      0.8470292  0.7557834  0.7599171
##   0.30  7          100      0.8450801  0.7541339  0.7633069
##   0.30  7          200      0.8455084  0.7541408  0.7709556
##
## Tuning parameter 'gamma' was held constant at a value of 0
## Tuning
##
## Tuning parameter 'min_child_weight' was held constant at a value of 1
##
## Tuning parameter 'subsample' was held constant at a value of 1
## Sens was used to select the optimal model using the largest value.
## The final values used for the model were nrounds = 100, max_depth = 3, eta
##  = 0.01, gamma = 0, colsample_bytree = 1, min_child_weight = 1 and subsample
##  = 1.
```

5. Threshold Optimization

```
# Predict probabilities on test set
xgb_probs <- predict(xgb_tuned, newdata = test_data, type = "prob")[, "yes"]
# Find best threshold for sensitivity/F1
thresholds <- seq(0, 1, by = 0.01)
f1_scores <- sapply(thresholds, function(t) {
  pred <- ifelse(xgb_probs > t, "yes", "no")
  cm <- confusionMatrix(factor(pred, levels = c("no", "yes")), test_data$Churn, pos
        itive = "yes")
  cm$byClass["F1"]
})
best_thresh <- thresholds[which.max(f1_scores)]
best_thresh
```

```
## [1] 0.61
```

```
plot(thresholds, f1_scores, type = "l", main = "F1 Score vs. Threshold", xlab = "Th
        reshold", ylab = "F1 Score")
```



6. Model Evaluation

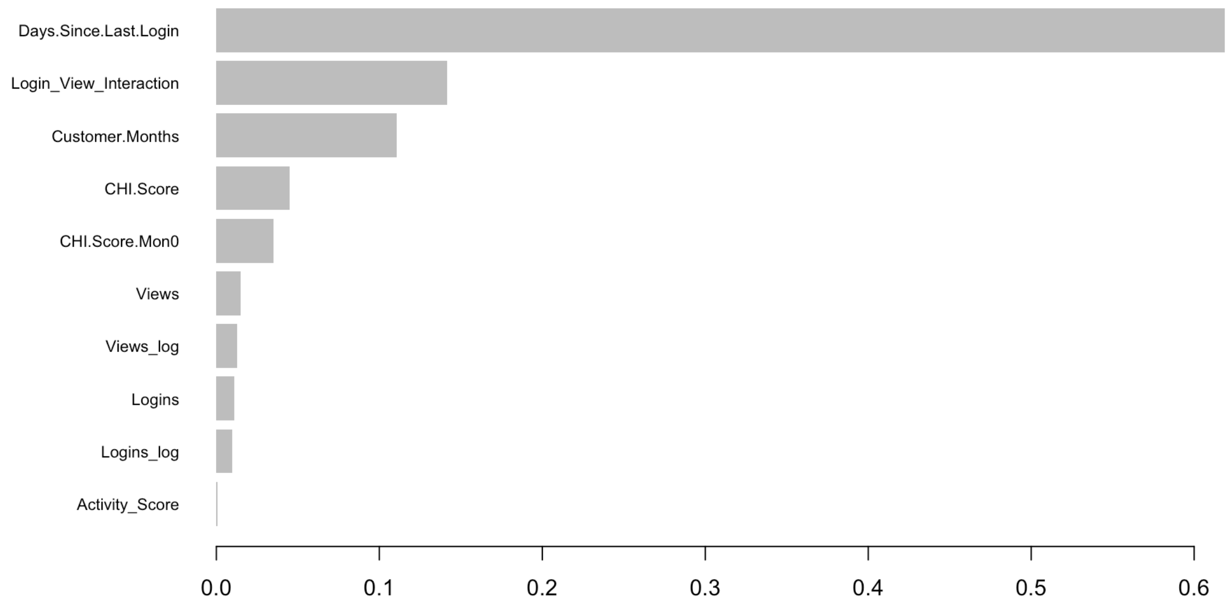
```
# Use best threshold to classify
xgb_pred_class <- ifelse(xgb_probs > best_thresh, "yes", "no")
cm <- confusionMatrix(factor(xgb_pred_class, levels = c("no", "yes")), test_data$Ch
        urn, positive = "yes")
roc_xgb <- roc(as.numeric(test_data$Churn == "yes"), xgb_probs)
auc_xgb <- auc(roc_xgb)

metrics <- data.frame(
  Sensitivity = cm$byClass["Sensitivity"],
  Specificity = cm$byClass["Specificity"],
  F1 = cm$byClass["F1"],
  AUC = auc_xgb
)
knitr::kable(metrics, digits = 3, caption = "Improved XGBoost Model Metrics (Test S
        et)")
```

Improved XGBoost Model Metrics (Test Set)

	Sensitivity	Specificity	F1	AUC
Sensitivity	0.5	0.886	0.25	0.7727733

7. Interpretation (Optional)

```
# Feature importance plot
xgb.importance <- xgb.importance(model = xgb_tuned$finalModel)
xgb.plot.importance(xgb.importance)
```



8. Conclusion

- Addressing class imbalance and tuning XGBoost improved model sensitivity and overall performance.
- Further improvements could include ensembling, additional feature engineering, or using other advanced algorithms.