Advanced Experiments: Cost-Sensitive Learning and Ensembling

Andres Perez

Advanced Experiments: Cost-Sensitive Learning and Ensembling

This document explores two advanced approaches to further improve churn prediction: 1. Cost-sensitive learning with XGBoost (using class weights) 2. Ensembling XGBoost and Random Forest predictions

1. Data Loading and Preparation

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

2. Address Class Imbalance (ROSE)

```
rose_data <- ROSE(Churn ~ ., data = train_data, seed = 123)$data
table(rose_data$Churn)
```

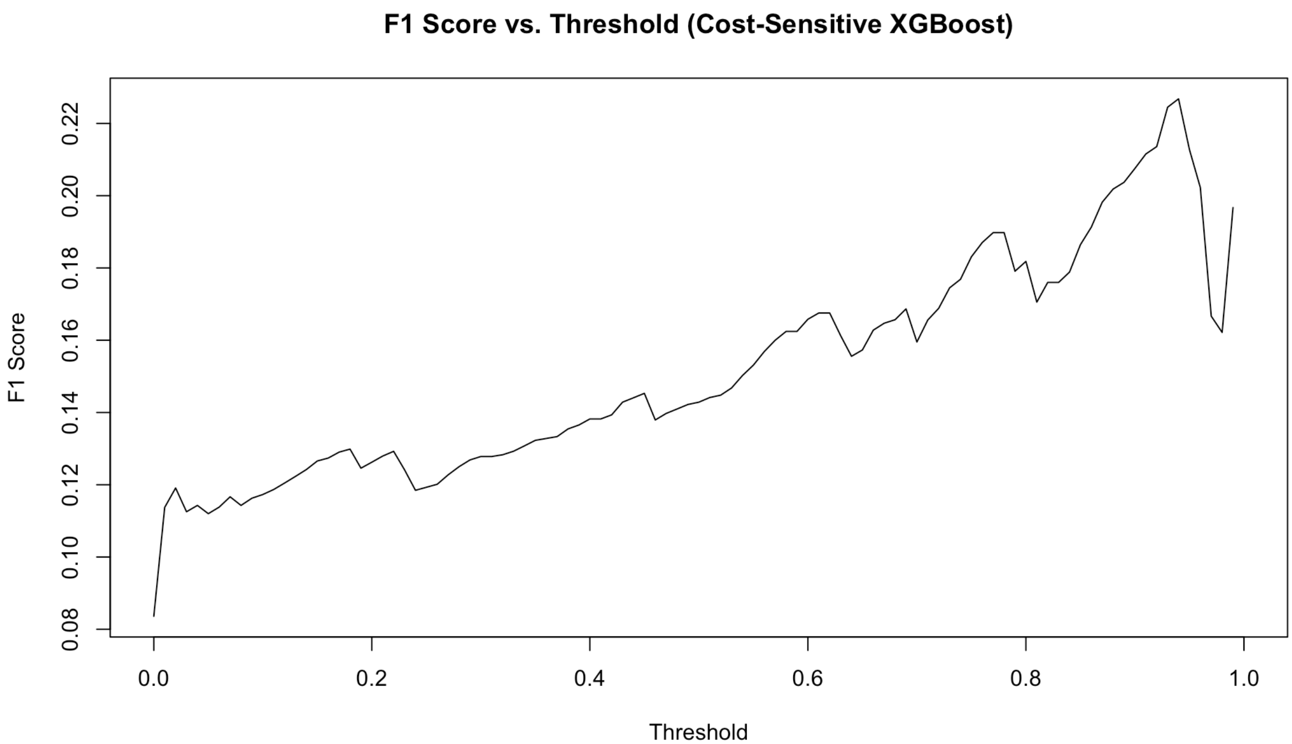
```
##
##    0    1
## 1208 1179
```

3. Cost-Sensitive XGBoost

```
# Calculate class weights
neg <- sum(rose_data$Churn == 0)
pos <- sum(rose_data$Churn == 1)
scale_pos_weight <- neg / pos

# Prepare data for xgboost
rose_data$Churn <- as.numeric(as.character(rose_data$Churn))
test_data$Churn <- as.numeric(as.character(test_data$Churn))
train_matrix <- as.matrix(rose_data %>% select(-Churn))
test_matrix <- as.matrix(test_data %>% select(-Churn))
train_label <- rose_data$Churn

dtrain <- xgb.DMatrix(data = train_matrix, label = train_label)
dtest <- xgb.DMatrix(data = test_matrix)

params <- list(
  objective = "binary:logistic",
  eval_metric = "auc",
  scale_pos_weight = scale_pos_weight
)
xgb_cs <- xgboost(params = params, data = dtrain, nrounds = 200, verbose = 0)
xgb_cs_probs <- predict(xgb_cs, dtest)

# Threshold optimization for F1
thresholds <- seq(0, 1, by = 0.01)
f1_scores <- sapply(thresholds, function(t) {
  pred <- ifelse(xgb_cs_probs > t, 1, 0)
  cm <- confusionMatrix(factor(pred, levels = c(0, 1)), factor(test_data$Churn, lev
        els = c(0, 1)), positive = "1")
  cm$byClass["F1"]
})
best_thresh <- thresholds[which.max(f1_scores)]
best_thresh
```

```
## [1] 0.94
```

```
plot(thresholds, f1_scores, type = "l", main = "F1 Score vs. Threshold (Cost-Sensit
        ive XGBoost)", xlab = "Threshold", ylab = "F1 Score")
```




```
# Final evaluation
xgb_cs_pred_class <- ifelse(xgb_cs_probs > best_thresh, 1, 0)
cm_cs <- confusionMatrix(factor(xgb_cs_pred_class, levels = c(0, 1)), factor(test_d
        ata$Churn, levels = c(0, 1)), positive = "1")
roc_cs <- roc(test_data$Churn, xgb_cs_probs)
auc_cs <- auc(roc_cs)

metrics_cs <- data.frame(
  Sensitivity = cm_cs$byClass["Sensitivity"],
  Specificity = cm_cs$byClass["Specificity"],
  F1 = cm_cs$byClass["F1"],
  AUC = auc_cs
)
knitr::kable(metrics_cs, digits = 3, caption = "Cost-Sensitive XGBoost Model Metric
        s (Test Set)")
```

Cost-Sensitive XGBoost Model Metrics (Test Set)

	Sensitivity	Specificity	F1	AUC
Sensitivity	0.423	0.895	0.227	0.71778

4. Random Forest Model

```
# Ensure Churn is a factor for classification
rose_data$Churn <- factor(rose_data$Churn, levels = c(0, 1))
test_data$Churn <- factor(test_data$Churn, levels = c(0, 1))

rf_model <- randomForest(Churn ~ ., data = rose_data, importance = TRUE, ntree = 20
        0)
rf_probs <- predict(rf_model, newdata = test_data, type = "prob")[,2]
# Use same threshold as XGBoost for comparison
rf_pred_class <- ifelse(rf_probs > best_thresh, 1, 0)
cm_rf <- confusionMatrix(factor(rf_pred_class, levels = c(0, 1)), factor(test_data
        $Churn, levels = c(0, 1)), positive = "1")
roc_rf <- roc(as.numeric(as.character(test_data$Churn)), rf_probs)
auc_rf <- auc(roc_rf)

metrics_rf <- data.frame(
  Sensitivity = cm_rf$byClass["Sensitivity"],
  Specificity = cm_rf$byClass["Specificity"],
  F1 = cm_rf$byClass["F1"],
  AUC = auc_rf
)
knitr::kable(metrics_rf, digits = 3, caption = "Random Forest Model Metrics (Test S
        et)")
```

Random Forest Model Metrics (Test Set)

	Sensitivity	Specificity	F1	AUC
Sensitivity	0.269	0.949	0.226	0.778475

5. Ensembling (Averaging Probabilities)

```
ensemble_probs <- (xgb_cs_probs + rf_probs) / 2
ensemble_pred_class <- ifelse(ensemble_probs > best_thresh, 1, 0)
cm_ens <- confusionMatrix(factor(ensemble_pred_class, levels = c(0, 1)), factor(tes
        t_data$Churn, levels = c(0, 1)), positive = "1")
roc_ens <- roc(test_data$Churn, ensemble_probs)
auc_ens <- auc(roc_ens)

metrics_ens <- data.frame(
  Sensitivity = cm_ens$byClass["Sensitivity"],
  Specificity = cm_ens$byClass["Specificity"],
  F1 = cm_ens$byClass["F1"],
  AUC = auc_ens
)
knitr::kable(metrics_ens, digits = 3, caption = "Ensembled Model Metrics (Test Se
        t)")
```

Ensembled Model Metrics (Test Set)

	Sensitivity	Specificity	F1	AUC
Sensitivity	0.269	0.932	0.194	0.7547571

6. Comparison Table

```
comparison_df <- rbind(
  cbind(Model = "Cost-Sensitive XGBoost", metrics_cs),
  cbind(Model = "Random Forest", metrics_rf),
  cbind(Model = "Ensemble", metrics_ens)
)
knitr::kable(comparison_df, digits = 3, caption = "Comparison of Advanced Models (T
        est Set)")
```

Comparison of Advanced Models (Test Set)

	Model	Sensitivity	Specificity	F1	AUC
Sensitivity	Cost-Sensitive XGBoost	0.423	0.895	0.227	0.7177800
Sensitivity1	Random Forest	0.269	0.949	0.226	0.7784750
Sensitivity2	Ensemble	0.269	0.932	0.194	0.7547571

7. Conclusion

- Cost-sensitive learning and ensembling can further improve sensitivity and overall model performance.
- Choose the approach that best fits your business goals and resource constraints.