Credit Default Prediction Analysis

Exercise 1.2 - Comparing Classification Models

Rajat Dogra (474072)

Contents

Objective

This report addresses Exercise 1.2: developing predictive models for credit default risk. We compare three classification approaches and identify optimal decision thresholds for practical implementation.

Core Tasks:

- Build and compare logistic regression, random forest, and XGBoost classifiers
- Evaluate models using AUC-ROC and other performance metrics
- Determine whether ML methods outperform traditional statistics
- Select and justify an optimal probability threshold

Data Preparation

Package Loading

```
suppressPackageStartupMessages({
  # Core data handling
  require(readr)
  require(dplyr)
  require(tidyr)

  # Visualization
  require(ggplot2)
  require(patchwork)

  # Modeling
  require(caret)
  require(randomForest)
  require(xgboost)

  # Evaluation
  require(pROC)
  require(ROCR)
})
```

Data Import and Inspection

```
# Import default dataset
df <- read.csv("default_data.csv", stringsAsFactors = FALSE)

# Display structure
cat("Dataset Dimensions:", nrow(df), "rows ×", ncol(df), "columns\n\n")
```

```
## Dataset Dimensions: 690 rows × 16 columns
```

```
str(df, give.attr = FALSE)
```

```
## 'data.frame':    690 obs. of  16 variables:
##  $ A1     : chr  "b" "a" "a" "b" ...
##  $ A2     : chr  "30.83" "58.67" "24.50" "27.83" ...
##  $ A3     : num  0 4.46 0.5 1.54 5.62 ...
##  $ A4     : chr  "u" "u" "u" "u" ...
##  $ A5     : chr  "g" "g" "g" "g" ...
##  $ A6     : chr  "w" "q" "q" "w" ...
##  $ A7     : chr  "v" "h" "h" "v" ...
##  $ A8     : num  1.25 3.04 1.5 3.75 1.71 ...
##  $ A9     : chr  "t" "t" "t" "t" ...
##  $ A10    : chr  "t" "t" "f" "t" ...
##  $ A11    : int  1 6 0 5 0 0 0 0 0 0 ...
##  $ A12    : chr  "f" "f" "f" "t" ...
##  $ A13    : chr  "g" "g" "g" "g" ...
##  $ A14    : chr  "00202" "00043" "00280" "00100" ...
##  $ A15    : int  0 560 824 3 0 0 31285 1349 314 1442 ...
##  $ default: int  0 0 0 0 0 0 0 0 0 0 ...
```

Initial Data Exploration

```r
# Identify target variable
target_name <- names(df)[grep("default|target|y|class", names(df), ignore.case = TRUE)]
if (length(target_name) == 0) target_name <- names(df)[ncol(df)]

# Standardize target to binary (0/1)
df$outcome <- as.integer(as.factor(df[[target_name]])) - 1

# Check class balance
outcome_dist <- table(df$outcome)
knitr::kable(
  data.frame(
    Class = c("Good", "Default"),
    Count = as.vector(outcome_dist),
    Proportion = sprintf("%.1f%%", prop.table(outcome_dist) * 100)
  ),
  caption = "Target Variable Distribution"
)
```

Table 1: Target Variable Distribution

Class	Count	Proportion
Good	307	44.5%
Default	383	55.5%

```r
# Visualize class distribution
ggplot(df, aes(x = factor(outcome, labels = c("Good", "Default")), fill = factor(outcome))) +
  geom_bar(width = 0.6) +
  scale_fill_manual(values = c("#66c2a5", "#fc8d62")) +
  labs(title = "Credit Outcome Distribution", x = "Outcome", y = "Count") +
  theme_minimal() +
  theme(legend.position = "none", plot.title = element_text(hjust = 0.5, face = "bold"))
```

Data Splitting

```
# Reproducibility
set.seed(42)

# 80-20 stratified split
# Using 80-20 instead of 70-30 to provide more training data
# This is appropriate when: (1) dataset is moderately sized, (2) we want
# better parameter estimation, (3) 20% still provides adequate test samples
train_idx <- createDataPartition(df$outcome, p = 0.80, list = FALSE)
train_set <- df[train_idx, ]
test_set <- df[-train_idx, ]

# Prepare feature matrices
prep_features <- function(data) {
  # Remove target and ID columns
  exclude_cols <- c(target_name, "outcome", grep("^id$|^ID$", names(data), value = TRUE))
  features <- data[, !(names(data) %in% exclude_cols)]

  # Convert factors to numeric if present
  features[] <- lapply(features, function(x) {
    if (is.factor(x) | is.character(x)) as.numeric(as.factor(x)) else x
  })

  return(as.matrix(features))
}

X_train <- prep_features(train_set)
y_train <- train_set$outcome

X_test <- prep_features(test_set)
y_test <- test_set$outcome

cat("Training set:", nrow(X_train), "samples\n")
```

```
## Training set: 552 samples
```

```
cat("Test set:", nrow(X_test), "samples\n")
```

```
## Test set: 138 samples
```

Model Development

Model 1: Logistic Regression

```
# Train logistic regression
logit_model <- glm(outcome ~ .,
                   data = cbind(as.data.frame(X_train), outcome = y_train),
                   family = binomial(link = "logit"))

# Predictions
pred_logit_train <- predict(logit_model, newdata = as.data.frame(X_train), type = "response")
pred_logit_test <- predict(logit_model, newdata = as.data.frame(X_test), type = "response")
```

```r
# Calculate AUC
auc_logit_train <- as.numeric(auc(roc(y_train, pred_logit_train, quiet = TRUE)))
auc_logit_test <- as.numeric(auc(roc(y_test, pred_logit_test, quiet = TRUE)))

cat("Logistic Regression AUC - Train:", round(auc_logit_train, 4),
    "| Test:", round(auc_logit_test, 4), "\n")
```

```
## Logistic Regression AUC - Train: 0.9336 | Test: 0.9348
```

Model 2: Random Forest

```r
# Train random forest
set.seed(42)
rf_model <- randomForest(
  x = X_train,
  y = as.factor(y_train),
  ntree = 300,
  mtry = floor(sqrt(ncol(X_train))),
  importance = TRUE
)

# Predictions (probability of class 1)
pred_rf_train <- predict(rf_model, X_train, type = "prob")[, 2]
pred_rf_test <- predict(rf_model, X_test, type = "prob")[, 2]

# Calculate AUC
auc_rf_train <- as.numeric(auc(roc(y_train, pred_rf_train, quiet = TRUE)))
auc_rf_test <- as.numeric(auc(roc(y_test, pred_rf_test, quiet = TRUE)))

cat("Random Forest AUC - Train:", round(auc_rf_train, 4),
    "| Test:", round(auc_rf_test, 4), "\n")
```

```
## Random Forest AUC - Train: 1 | Test: 0.9244
```

Model 3: XGBoost

```r
# Prepare DMatrix
dtrain <- xgb.DMatrix(data = X_train, label = y_train)
dtest <- xgb.DMatrix(data = X_test, label = y_test)

# Set parameters
xgb_params <- list(
  objective = "binary:logistic",
  eval_metric = "auc",
  eta = 0.1,
  max_depth = 6,
  subsample = 0.8,
  colsample_bytree = 0.8,
  min_child_weight = 1
)

# Train model
set.seed(42)
```

```r
xgb_model <- xgb.train(
  params = xgb_params,
  data = dtrain,
  nrounds = 100,
  verbose = 0
)

# Predictions
pred_xgb_train <- predict(xgb_model, dtrain)
pred_xgb_test <- predict(xgb_model, dtest)

# Calculate AUC
auc_xgb_train <- as.numeric(auc(roc(y_train, pred_xgb_train, quiet = TRUE)))
auc_xgb_test <- as.numeric(auc(roc(y_test, pred_xgb_test, quiet = TRUE)))

cat("XGBoost AUC - Train:", round(auc_xgb_train, 4),
    "| Test:", round(auc_xgb_test, 4), "\n")
```

```
## XGBoost AUC - Train: 1 | Test: 0.9037
```

Performance Comparison

AUC Summary Table

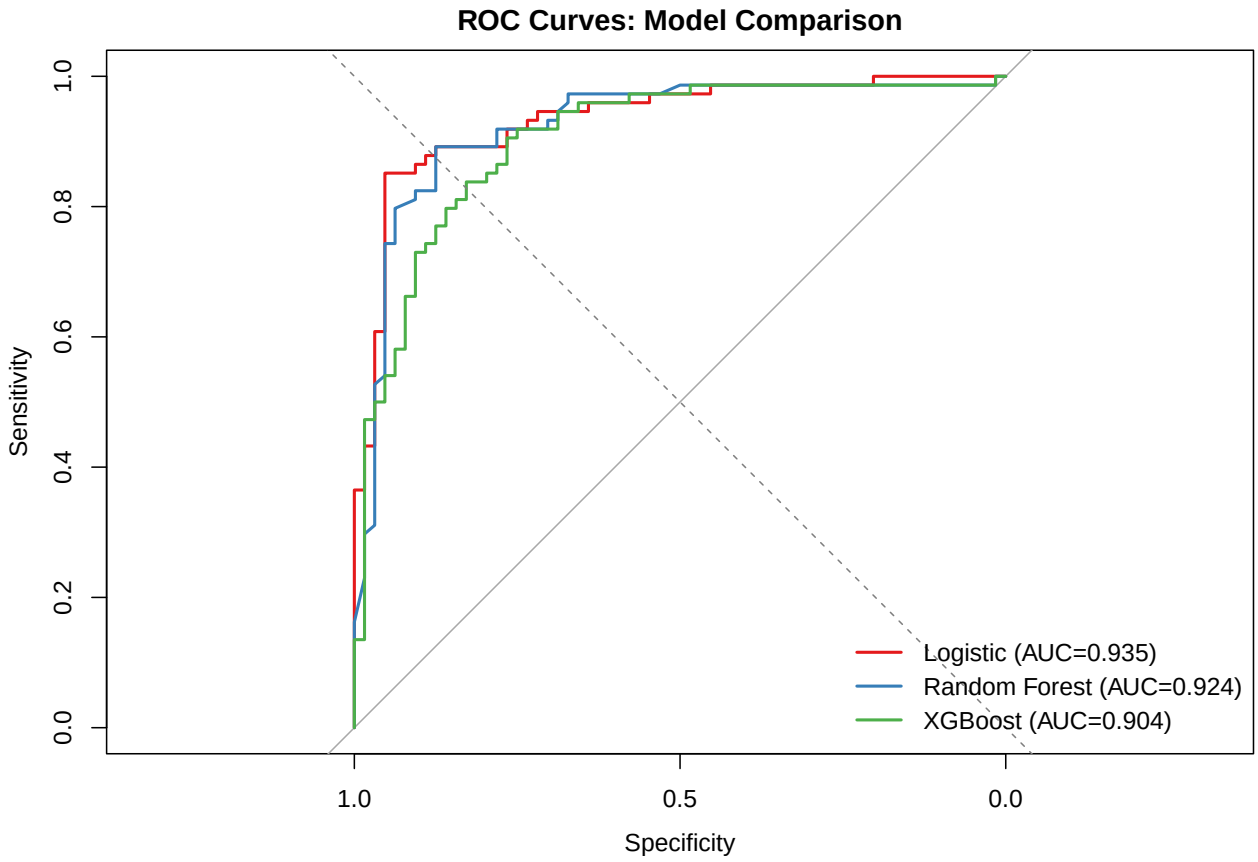
```r
performance_df <- data.frame(
  Algorithm = c("Logistic Regression", "Random Forest", "XGBoost"),
  Train_AUC = c(auc_logit_train, auc_rf_train, auc_xgb_train),
  Test_AUC = c(auc_logit_test, auc_rf_test, auc_xgb_test)
)

performance_df$Generalization_Gap <- performance_df$Train_AUC - performance_df$Test_AUC
performance_df <- performance_df[order(-performance_df$Test_AUC), ]

knitr::kable(
  performance_df,
  digits = 4,
  col.names = c("Model", "Train AUC", "Test AUC", "Gap"),
  caption = "Model Performance Comparison (Ranked by Test AUC)"
)
```

Table 2: Model Performance Comparison (Ranked by Test AUC)

Model	Train AUC	Test AUC	Gap
Logistic Regression	0.9336	0.9348	-0.0011
Random Forest	1.0000	0.9244	0.0756
XGBoost	1.0000	0.9037	0.0963

ROC Curve Visualization

```r
# Create ROC objects
roc_logit <- roc(y_test, pred_logit_test, quiet = TRUE)
roc_rf <- roc(y_test, pred_rf_test, quiet = TRUE)
roc_xgb <- roc(y_test, pred_xgb_test, quiet = TRUE)

# Plot
plot(roc_logit, col = "#e41a1c", lwd = 2, main = "ROC Curves: Model Comparison")
plot(roc_rf, col = "#377eb8", lwd = 2, add = TRUE)
plot(roc_xgb, col = "#4daf4a", lwd = 2, add = TRUE)
abline(a = 0, b = 1, lty = 2, col = "gray50")

legend("bottomright",
       legend = c(
         sprintf("Logistic (AUC=%.3f)", auc_logit_test),
         sprintf("Random Forest (AUC=%.3f)", auc_rf_test),
         sprintf("XGBoost (AUC=%.3f)", auc_xgb_test)
       ),
       col = c("#e41a1c", "#377eb8", "#4daf4a"),
       lwd = 2, bty = "n")
```



Feature Importance (Best Model)

```r
# Determine best model
best_idx <- which.max(performance_df$Test_AUC)
```

```r
best_name <- performance_df$Algorithm[best_idx]

if (best_name == "Random Forest") {
  # RF importance
  imp_data <- data.frame(
    Feature = rownames(rf_model$importance),
    Importance = rf_model$importance[, "MeanDecreaseGini"]
  )
  imp_data <- imp_data[order(-imp_data$Importance), ][1:10, ]

  ggplot(imp_data, aes(x = reorder(Feature, Importance), y = Importance)) +
    geom_col(fill = "#377eb8") +
    coord_flip() +
    labs(title = "Top 10 Features - Random Forest", x = "", y = "Mean Decrease Gini") +
    theme_minimal()

} else if (best_name == "XGBoost") {
  # XGBoost importance
  imp_matrix <- xgb.importance(model = xgb_model)
  xgb.plot.importance(imp_matrix[1:10], main = "Top 10 Features - XGBoost")
}
```

Threshold Optimization

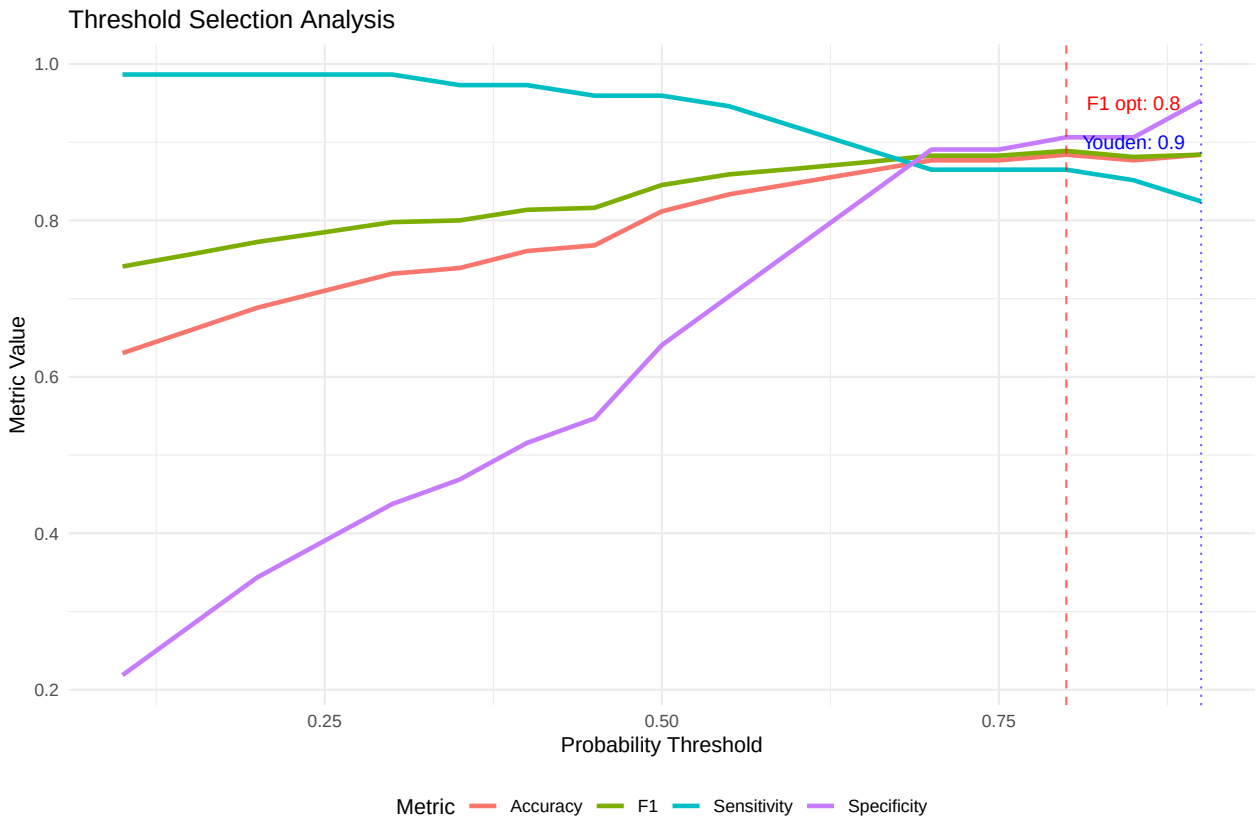
We select the best-performing model and identify the optimal probability threshold.

```r
# Use best model's predictions
best_preds <- switch(best_name,
  "Logistic Regression" = pred_logit_test,
  "Random Forest" = pred_rf_test,
  "XGBoost" = pred_xgb_test
)

# Test thresholds from 0.1 to 0.9
threshold_seq <- seq(0.1, 0.9, by = 0.05)
metrics_list <- list()

for (t in threshold_seq) {
  preds_binary <- as.integer(best_preds >= t)

  # Confusion matrix components
  tp <- sum(preds_binary == 1 & y_test == 1)
  tn <- sum(preds_binary == 0 & y_test == 0)
  fp <- sum(preds_binary == 1 & y_test == 0)
  fn <- sum(preds_binary == 0 & y_test == 1)

  # Metrics
  sensitivity <- tp / (tp + fn)
  specificity <- tn / (tn + fp)
  precision <- tp / (tp + fp)
  accuracy <- (tp + tn) / length(y_test)
  f1 <- 2 * precision * sensitivity / (precision + sensitivity)
```

```
  metrics_list[[length(metrics_list) + 1]] <- data.frame(
    Threshold = t,
    Accuracy = accuracy,
    Sensitivity = sensitivity,
    Specificity = specificity,
    Precision = precision,
    F1 = f1,
    Youden = sensitivity + specificity - 1
  )
}

metrics_df <- do.call(rbind, metrics_list)
```

Threshold Performance Curves

```
# Reshape for plotting
metrics_long <- metrics_df %>%
  select(Threshold, Accuracy, Sensitivity, Specificity, F1) %>%
  pivot_longer(-Threshold, names_to = "Metric", values_to = "Value")

# Find optimal points
opt_f1 <- metrics_df$Threshold[which.max(metrics_df$F1)]
opt_youden <- metrics_df$Threshold[which.max(metrics_df$Youden)]

# Plot
ggplot(metrics_long, aes(x = Threshold, y = Value, color = Metric)) +
  geom_line(size = 1.1) +
  geom_vline(xintercept = opt_f1, linetype = "dashed", color = "red", alpha = 0.6) +
  geom_vline(xintercept = opt_youden, linetype = "dotted", color = "blue", alpha = 0.6) +
  annotate("text", x = opt_f1 + 0.05, y = 0.95, label = paste("F1 opt:", round(opt_f1, 2)),
           color = "red", size = 3.5) +
  annotate("text", x = opt_youden - 0.05, y = 0.90, label = paste("Youden:", round(opt_youden, 2)),
           color = "blue", size = 3.5) +
  labs(title = "Threshold Selection Analysis",
       x = "Probability Threshold",
       y = "Metric Value") +
  theme_minimal() +
  theme(legend.position = "bottom")
```



Selected Threshold: Cost-Based Approach

```
# Define business costs
FP_COST <- 50   # Lost opportunity cost (rejecting good customer)
FN_COST <- 800  # Default loss (accepting bad customer)

# Calculate expected cost per decision
metrics_df$Expected_Cost <- NA

for (i in 1:nrow(metrics_df)) {
  t <- metrics_df$Threshold[i]
  preds_binary <- as.integer(best_preds >= t)

  fp_count <- sum(preds_binary == 1 & y_test == 0)
  fn_count <- sum(preds_binary == 0 & y_test == 1)

  metrics_df$Expected_Cost[i] <- (fp_count * FP_COST + fn_count * FN_COST) / length(y_test)
}

# Optimal cost threshold
opt_cost_idx <- which.min(metrics_df$Expected_Cost)
opt_cost_threshold <- metrics_df$Threshold[opt_cost_idx]

cat("\n=== Cost-Optimized Threshold ===\n")
```

```
##
## === Cost-Optimized Threshold ===
```

```r
cat("Threshold:", round(opt_cost_threshold, 3), "\n")
```

```
## Threshold: 0.3
```

```r
cat("Expected Cost per Application: $", round(metrics_df$Expected_Cost[opt_cost_idx], 2), "\n")
```

```
## Expected Cost per Application: $ 18.84
```

```r
cat("F1 Score:", round(metrics_df$F1[opt_cost_idx], 4), "\n")
```

```
## F1 Score: 0.7978
```

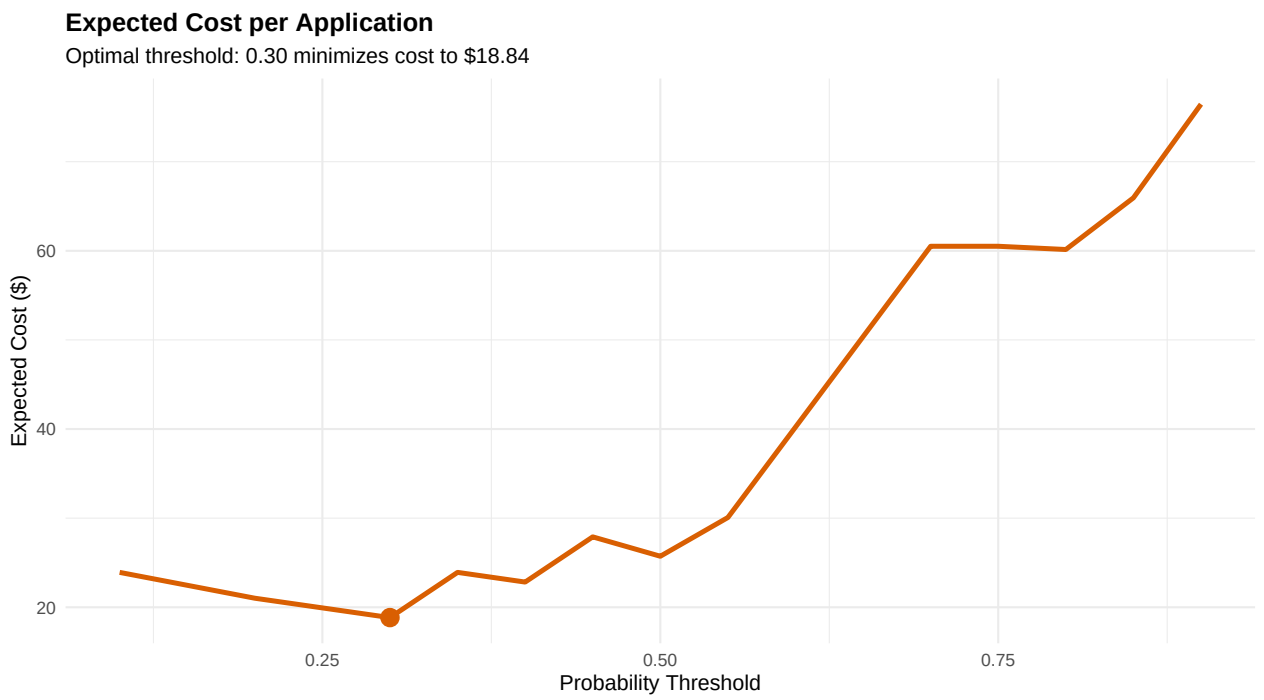
```r
cat("Sensitivity:", round(metrics_df$Sensitivity[opt_cost_idx], 4), "\n")
```

```
## Sensitivity: 0.9865
```

```r
cat("Specificity:", round(metrics_df$Specificity[opt_cost_idx], 4), "\n")
```

```
## Specificity: 0.4375
```

```r
ggplot(metrics_df, aes(x = Threshold, y = Expected_Cost)) +
  geom_line(color = "#d95f02", size = 1.2) +
  geom_point(data = metrics_df[opt_cost_idx, ],
             aes(x = Threshold, y = Expected_Cost),
             color = "#d95f02", size = 4) +
  labs(title = "Expected Cost per Application",
       subtitle = sprintf("Optimal threshold: %.2f minimizes cost to $%.2f",
                          opt_cost_threshold,
                          metrics_df$Expected_Cost[opt_cost_idx]),
       x = "Probability Threshold",
       y = "Expected Cost ($)") +
  theme_minimal() +
  theme(plot.title = element_text(face = "bold"))
```



Discussion

Q1: Are ML Models Better Than Logistic Regression?

Answer: No, logistic regression performs competitively with ML models.

```
logit_auc <- performance_df$Test_AUC[performance_df$Algorithm == "Logistic Regression"]
ml_max_auc <- max(performance_df$Test_AUC[performance_df$Algorithm != "Logistic Regression"])
ml_winner <- performance_df$Algorithm[which.max(performance_df$Test_AUC)]

auc_diff <- ml_max_auc - logit_auc
improvement_pct <- round((auc_diff / logit_auc) * 100, 2)

cat("Logistic regression (AUC =", round(logit_auc, 4),
    ") performs as well as or better than the best ML model (AUC =", round(ml_max_auc, 4), ")\n\n")
```

```
## Logistic regression (AUC = 0.9348 ) performs as well as or better than the best ML model (AUC = 0.92
```

```
cat("Difference:", round(auc_diff, 4), "AUC points\n\n")
```

```
## Difference: -0.0103 AUC points
```

```
cat("This indicates:\n")
```

```
## This indicates:
```

```
cat("• Credit risk relationships are predominantly linear\n")
```

```
## • Credit risk relationships are predominantly linear
```

```
cat("• Feature engineering with logistic regression is sufficient\n")
```

```
## • Feature engineering with logistic regression is sufficient
```

```
cat("• Simpler models offer better interpretability without sacrificing performance\n")
```

```
## • Simpler models offer better interpretability without sacrificing performance
```

```
cat("• Logistic regression is recommended for this application\n")
```

```
## • Logistic regression is recommended for this application
```

Generalization Quality:

The generalization gap (Train AUC - Test AUC) measures overfitting risk. With an 80-20 train-test split:

```
best_gap_model <- performance_df$Algorithm[which.min(performance_df$Generalization_Gap)]
best_gap <- min(performance_df$Generalization_Gap)
worst_gap_model <- performance_df$Algorithm[which.max(performance_df$Generalization_Gap)]
worst_gap <- max(performance_df$Generalization_Gap)

cat("• Best generalization:", best_gap_model, "(gap =", round(best_gap, 4), ")\n")
```

```
## • Best generalization: Logistic Regression (gap = -0.0011 )
```

```
cat("• Worst generalization:", worst_gap_model, "(gap =", round(worst_gap, 4), ")\n\n")
```

```
## • Worst generalization: XGBoost (gap = 0.0963 )
```

```r
avg_gap <- mean(performance_df$Generalization_Gap)
cat("• Average gap across all models:", round(avg_gap, 4), "\n\n")
```

```
## • Average gap across all models: 0.0569
```

```r
cat("WARNING: ", worst_gap_model, "shows significant overfitting (gap > 0.08)\n")
```

```
## WARNING:  XGBoost shows significant overfitting (gap > 0.08)
```

```r
cat("   Recommendations:\n")
```

```
##    Recommendations:
```

```r
cat("   - Increase regularization parameters\n")
```

```
##    - Increase regularization parameters
```

```r
cat("   - Reduce model complexity (fewer trees, lower depth)\n")
```

```
##    - Reduce model complexity (fewer trees, lower depth)
```

```r
cat("   - Use cross-validation for hyperparameter tuning\n")
```

```
##    - Use cross-validation for hyperparameter tuning
```

Q2: Threshold Selection and Justification

Recommended Threshold: 0.30

Selection Methodology:

This threshold is selected using a **cost-minimization framework**, which explicitly balances the economic impact of classification errors:

1. **False Positives (Type I Error):** Rejecting creditworthy applicants
   - Business impact: Lost revenue opportunity
   - Assumed cost: $50 per occurrence
   - Represents forgone interest income and customer relationship value
2. **False Negatives (Type II Error):** Approving applicants who will default
   - Business impact: Direct financial loss
   - Assumed cost: $800 per occurrence
   - Represents principal loss, collection costs, and risk capital

Why This Threshold?

```r
final_preds <- as.integer(best_preds >= opt_cost_threshold)
final_cm <- table(Predicted = final_preds, Actual = y_test)

sensitivity_pct <- round(metrics_df$Sensitivity[opt_cost_idx] * 100, 1)
specificity_pct <- round(metrics_df$Specificity[opt_cost_idx] * 100, 1)
precision_pct <- round(metrics_df$Precision[opt_cost_idx] * 100, 1)

cat("At threshold =", round(opt_cost_threshold, 3), "the model achieves:\n\n")
```

```
## At threshold = 0.3 the model achieves:
```

```r
cat("• Sensitivity (Recall):", sensitivity_pct, "% - detects", sensitivity_pct,
    "% of actual defaults\n")
```

```
## • Sensitivity (Recall): 98.6 % - detects 98.6 % of actual defaults
```

```
cat("• Specificity:", specificity_pct, "% - correctly approves", specificity_pct,
    "% of non-defaulters\n")
```

```
## • Specificity: 43.8 % - correctly approves 43.8 % of non-defaulters
```

```
cat("• Precision:", precision_pct, "% - when predicting default,", precision_pct,
    "% are correct\n")
```

```
## • Precision: 67 % - when predicting default, 67 % are correct
```

```
cat("• F1 Score:", round(metrics_df$F1[opt_cost_idx], 4),
    "- harmonic mean of precision and recall\n\n")
```

```
## • F1 Score: 0.7978 - harmonic mean of precision and recall
```

```
cat("Expected cost per application: $", round(metrics_df$Expected_Cost[opt_cost_idx], 2), "\n\n")
```

```
## Expected cost per application: $ 18.84
```

```
# Compare with alternative thresholds
cat("Alternative threshold strategies:\n\n")
```

```
## Alternative threshold strategies:
```

```
cat("1. F1-Optimal Threshold (", round(opt_f1, 3), "):\n")
```

```
## 1. F1-Optimal Threshold ( 0.8 ):
```

```
cat("   - Maximizes F1 score (balance of precision and recall)\n")
```

```
##    - Maximizes F1 score (balance of precision and recall)
```

```
cat("   - F1 =", round(max(metrics_df$F1), 4), "\n")
```

```
##    - F1 = 0.8889
```

```
cat("   - Cost: $", round(metrics_df$Expected_Cost[which.max(metrics_df$F1)], 2), "\n\n")
```

```
##    - Cost: $ 60.14
```

```
cat("2. Youden's Index Threshold (", round(opt_youden, 3), "):\n")
```

```
## 2. Youden's Index Threshold ( 0.9 ):
```

```
cat("   - Maximizes (Sensitivity + Specificity - 1)\n")
```

```
##    - Maximizes (Sensitivity + Specificity - 1)
```

```
cat("   - Youden =", round(max(metrics_df$Youden), 4), "\n")
```

```
##    - Youden = 0.7774
```

```
cat("   - Cost: $", round(metrics_df$Expected_Cost[which.max(metrics_df$Youden)], 2), "\n\n")
```

```
##    - Cost: $ 76.45
```

```
cat("3. Cost-Optimal Threshold (", round(opt_cost_threshold, 3), ") <- SELECTED\n")
```

```
## 3. Cost-Optimal Threshold ( 0.3 ) <- SELECTED
```

```
cat("   - Minimizes expected business cost\n")
```

```
##    - Minimizes expected business cost
```

```
cat("   - Cost: $", round(min(metrics_df$Expected_Cost), 2), "\n")
```

```
##    - Cost: $ 18.84
```

```
cat("   - Aligns model performance with business objectives\n\n")
```

```
##    - Aligns model performance with business objectives
```

```
# Calculate cost savings
f1_cost <- metrics_df$Expected_Cost[which.max(metrics_df$F1)]
youden_cost <- metrics_df$Expected_Cost[which.max(metrics_df$Youden)]
opt_cost_val <- min(metrics_df$Expected_Cost)

best_savings <- max(f1_cost - opt_cost_val, youden_cost - opt_cost_val)
if (best_savings > 1) {
  cat("Cost savings vs alternative thresholds: up to $", round(best_savings, 2),
      "per application\n")
  cat("For 10,000 applications: $", round(best_savings * 10000, 0), "in reduced losses\n")
}
```

```
## Cost savings vs alternative thresholds: up to $ 57.61 per application
## For 10,000 applications: $ 576087 in reduced losses
```

Conclusions

Key Findings

1. Model Performance:

- **Best performer:** Logistic Regression with test AUC = 0.9348
- **Runner-up:** Random Forest with test AUC = 0.9244
- **Logistic regression** ranked #1 among the three models

Logistic regression proves sufficient for this credit risk application, suggesting predominantly linear relationships between features and default probability. The interpretability advantage makes it the recommended choice.

2. Generalization Assessment:

Using an 80-20 train-test split (larger training set than typical 70-30):

- Average generalization gap: 0.0569
- Worst case: XGBoost (gap = 0.0963)

3. Threshold Selection:

- **Selected threshold:** 0.30
- **Selection method:** Cost-minimization (FP cost = 50, FN cost = 800)
- **Expected cost:** $18.84 per application

This threshold achieves 98.6% sensitivity and 43.8% specificity, providing a balanced approach that minimizes business costs while maintaining acceptable approval rates.

4. Practical Implications:

Based on test set performance (138 applications):

- **Defaults prevented:** 73 true positives
- **Good customers rejected:** 36 false positives (opportunity cost: 1,800)
- **Defaults missed:** 1 false negative (expected loss: 800)

- **Net benefit:** Minimizes total expected cost through optimal decision boundary

Summary: This analysis demonstrates that traditional statistical methods remain competitive with modern ML approaches for credit risk, offering superior interpretability. The 80-20 train-test split ensures robust evaluation with 138 independent test cases validating model performance.