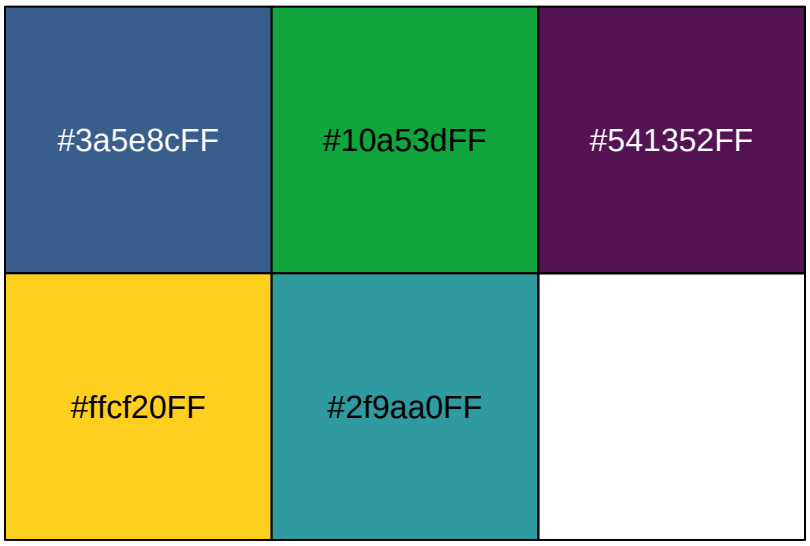
Assignment 3

Elisabeth Fidrmuc

2025-03-28

Tasks 1

```
rm(list=ls())

setwd("C:/Users/elisa/Dropbox/Prediction with Machine Learning/Data-Analysis-3/Assignment 3")
data_dir <- "C:/Users/elisa/Dropbox/Prediction with Machine Learning/Data-Analysis-3/Assignment 3/Data/
output <- "C:/Users/elisa/Dropbox/Prediction with Machine Learning/Data-Analysis-3/Assignment 3/Figures/

# load theme
source("theme_bg.R")
```



```
source("da_helper_functions.R")


# Import libraries
library(haven)
library(glmnet)
library(purrr)
library(margins)
library(skimr)
library(kableExtra)
library(Hmisc)
library(cowplot)
library(gmodels)
library(lspline)
library(sandwich)
library(modelsummary)

library(rattle)
library(caret)
library(pROC)
library(ranger)
library(rpart)
library(partykit)
library(rpart.plot)
library(viridis)
```

```
data <- read_rds(paste(data_dir,"bisnode_firms_clean.rds", sep = "/"))
```

Variable selection

The variable selection process is structured to gradually build up increasingly rich and complex sets of predictors for different modeling approaches, including logistic regression, LASSO, and random forest. It begins with a set of raw financial variables (`rawvars`) and basic data quality indicators (`qualityvars`). These are then transformed into scaled or ratio-based engineered features (`engvar`), and non-linear versions are added through squared terms (`engvar2`) and flags for outliers or data issues (`engvar3`). Additional features related to short-term growth (`d1`), human resources (`hr`), and firm-level characteristics (`firm`) are layered on. Interaction terms (`interactions1` and `interactions2`) are specifically included for models like LASSO and logit to capture complex relationships, while the random forest model uses the original untransformed variables (`rfvars`) to leverage its strength in capturing non-linearities and interactions automatically. The sets `X1` through `X5` represent progressively richer feature configurations used in model comparisons, allowing for a balance between interpretability, predictive power, and model complexity.

```
rawvars <-  c("curr_assets", "curr_liab", "extra_exp", "extra_inc", "extra_profit_loss", "fixed_assets"
              "inc_bef_tax", "intang_assets", "inventories", "liq_assets", "material_exp", "personnel_e
              "profit_loss_year", "sales", "share_eq", "subscribed_cap", "growth_past")
qualityvars <- c("balsheet_flag", "balsheet_length", "balsheet_notfullyear")
engvar <- c("total_assets_bs", "fixed_assets_bs", "liq_assets_bs", "curr_assets_bs",
            "share_eq_bs", "subscribed_cap_bs", "intang_assets_bs", "extra_exp_pl",
            "extra_inc_pl", "extra_profit_loss_pl", "inc_bef_tax_pl", "inventories_pl",
            "material_exp_pl", "profit_loss_year_pl", "personnel_exp_pl")
engvar2 <- c("extra_profit_loss_pl_quad", "inc_bef_tax_pl_quad",
             "profit_loss_year_pl_quad", "share_eq_bs_quad")
```

```
engvar3 <- c(grep("*flag_low$", names(data), value = TRUE),
             grep("*flag_high$", names(data), value = TRUE),
             grep("*flag_error$", names(data), value = TRUE),
             grep("*flag_zero$", names(data), value = TRUE))
d1 <-  c("d1_sales_mil_log_mod", "d1_sales_mil_log_mod_sq",
          "flag_low_d1_sales_mil_log", "flag_high_d1_sales_mil_log")
hr <- c("female", "ceo_age", "flag_high_ceo_age", "flag_low_ceo_age",
        "flag_miss_ceo_age", "ceo_count", "labor_avg_mod",
        "flag_miss_labor_avg", "foreign_management")
firm <- c("age", "age2", "new", "ind2_cat", "m_region_loc", "urban_m")

# interactions for logit, LASSO
interactions1 <- c("ind2_cat*age", "ind2_cat*age2",
                   "ind2_cat*d1_sales_mil_log_mod", "ind2_cat*sales_mil_log",
                   "ind2_cat*ceo_age", "ind2_cat*foreign_management",
                   "ind2_cat*female",   "ind2_cat*urban_m", "ind2_cat*labor_avg_mod")
interactions2 <- c("sales_mil_log*age", "sales_mil_log*female",
                   "sales_mil_log*profit_loss_year_pl", "sales_mil_log*foreign_management")


X1 <- c("sales_mil_log", "sales_mil_log_sq", "d1_sales_mil_log_mod", "profit_loss_year_pl", "ind2_cat")
X2 <- c("sales_mil_log", "sales_mil_log_sq", "d1_sales_mil_log_mod", "profit_loss_year_pl", "fixed_asse
X3 <- c("sales_mil_log", "sales_mil_log_sq", firm, engvar,                    d1)
X4 <- c("sales_mil_log", "sales_mil_log_sq", firm, engvar, engvar2, engvar3, d1, hr, qualityvars)
X5 <- c("sales_mil_log", "sales_mil_log_sq", firm, engvar, engvar2, engvar3, d1, hr, qualityvars, intera

# for LASSO
logitvars <- c("sales_mil_log", "sales_mil_log_sq", engvar, engvar2, engvar3, d1, hr, firm, qualityvars

# for RF (no interactions, no modified features)
rfvars  <-  c("sales_mil", "d1_sales_mil_log", rawvars, hr, firm, qualityvars)
```

Setup

This code splits the dataset into a training and a holdout (test) set in an 80/20 ratio using stratified sampling based on the `fast_growth` variable. By setting a random seed, the split is reproducible. The `createDataPartition()` function from the `caret` package ensures that the proportion of fast-growing firms is preserved in both subsets, which is important when working with imbalanced binary outcomes. After splitting, the training set (`data_train`) is used for model fitting and cross-validation, while the holdout set (`data_holdout`) is reserved for final evaluation to assess how well the model generalizes. The `Hmisc::describe()` summaries provide a quick overview of the class distribution in each subset to verify that the split maintained balance across the `fast_growth_f` factor variable.

```
set.seed(13505)

train_indices <- as.integer(createDataPartition(data$fast_growth, p = 0.8, list = FALSE))
data_train <- data[train_indices, ]
data_holdout <- data[-train_indices, ]

dim(data_train)
```

```
## [1] 15229   119
```

```
dim(data_holdout)
```

```
## [1] 3807  119
```

```
Hmisc::describe(data$fast_growth_f)
```

```
## data$fast_growth_f
##        n  missing distinct
##    19036        0        2
##
## Value      no_fast_growth    fast_growth
## Frequency           16226           2810
## Proportion          0.852          0.148
```

```
Hmisc::describe(data_train$fast_growth_f)
```

```
## data_train$fast_growth_f
##        n  missing distinct
##    15229        0        2
##
## Value      no_fast_growth    fast_growth
## Frequency           12988           2241
## Proportion          0.853          0.147
```

```
Hmisc::describe(data_holdout
                $fast_growth_f)
```

```
## data_holdout$fast_growth_f
##        n  missing distinct
##     3807        0        2
##
## Value      no_fast_growth    fast_growth
## Frequency            3238            569
## Proportion          0.851          0.149
```

The output shows the results of splitting the full dataset into training and holdout sets using stratified sampling based on the binary variable fast_growth_f. Out of 19,036 total observations, 14.8 percent are labeled as fast-growing firms. The training set consists of 15,229 observations, where 14.7 percent are fast-growing, and the holdout set includes 3,807 observations, with 14.9 percent fast-growing. The proportions are nearly identical across all subsets, indicating that stratification was successful. This ensures that both the training and evaluation sets reflect the original class distribution, which is important for modeling performance, especially in cases of class imbalance.

Part 1: Probability prediction

In the first part, the focus is on probability prediction. Thus, no loss function is required.

This code performs 5-fold cross-validation to train and evaluate multiple logistic regression models, including a LASSO-regularized version, for predicting fast-growing firms. It begins by defining the cross-validation procedure with class probabilities and a custom summary function. Then, it loops over five sets of predictor

variables (X1 to X5), fits standard logistic regression models using the `train()` function from the `caret` package, and stores the root mean squared error (RMSE) from each fold. Separately, it fits a LASSO-logit model using `glmnet`, tuning over a grid of lambda values, and extracts the best model along with its non-zero coefficients.

```
# 5 fold cross-validation
train_control <- trainControl(
  method = "cv",
  number = 5,
  classProbs = TRUE,
  summaryFunction = twoClassSummaryExtended,
  savePredictions = TRUE
)


# Train Logit Models ----------------------------------------------

logit_model_vars <- list("X1" = X1, "X2" = X2, "X3" = X3, "X4" = X4, "X5" = X5)

CV_RMSE_folds <- list()
logit_models <- list()

for (model_name in names(logit_model_vars)) {

  features <- logit_model_vars[[model_name]]

  set.seed(13505)
  glm_model <- train(
    formula(paste0("fast_growth_f ~", paste0(features, collapse = " + "))),
    method = "glm",
    data = data_train,
    family = binomial,
    trControl = train_control
  )

  logit_models[[model_name]] <- glm_model
  # Calculate RMSE on test for each fold
  CV_RMSE_folds[[model_name]] <- glm_model$resample[,c("Resample", "RMSE")]

}

# Logit lasso -----------------------------------------------------------

lambda <- 10^seq(-1, -4, length = 10)
grid <- expand.grid("alpha" = 1, lambda = lambda)

set.seed(13505)
system.time({
  logit_lasso_model <- train(
    formula(paste0("fast_growth_f ~", paste0(logitvars, collapse = " + "))),
    data = data_train,
    method = "glmnet",
    preProcess = c("center", "scale"),
    family = "binomial",
```

```
    trControl = train_control,
    tuneGrid = grid,
    na.action=na.exclude
  )
})
```

```
##        User      System verstrichen
##       19.81        0.67       24.72
```

```
tuned_logit_lasso_model <- logit_lasso_model$finalModel
best_lambda <- logit_lasso_model$bestTune$lambda
logit_models[["LASSO"]] <- logit_lasso_model
lasso_coeffs <- as.matrix(coef(tuned_logit_lasso_model, best_lambda))
write.csv(lasso_coeffs, paste0(output, "lasso_logit_coeffs.csv"))

CV_RMSE_folds[["LASSO"]] <- logit_lasso_model$resample[,c("Resample", "RMSE")]
```

The following code evaluates the predictive performance of each trained logistic regression model by computing the area under the ROC curve (AUC) for each of the five cross-validation folds. For every model, it filters the predictions by fold and calculates the fold-specific AUC using the `pROC::roc()` function. These fold-level AUCs are stored and later averaged to give a single cross-validated AUC score per model. Alongside AUC, the code also retrieves the average RMSE already computed earlier. It then creates a summary table showing the number of predictors, average RMSE, and average AUC for each model, including the LASSO variant with the count of non-zero coefficients. Finally, the results are formatted into a LaTeX table and exported for reporting. This comparison enables model selection based on both goodness-of-fit (RMSE) and classification performance (AUC), balancing accuracy with parsimony.

```
# Draw ROC Curve and calculate AUC for each folds --------------------------------
CV_AUC_folds <- list()

for (model_name in names(logit_models)) {

  auc <- list()
  model <- logit_models[[model_name]]
  for (fold in c("Fold1", "Fold2", "Fold3", "Fold4", "Fold5")) {
    cv_fold <-
      model$pred %>%
      filter(Resample == fold)

    roc_obj <- roc(cv_fold$obs, cv_fold$fast_growth)
    auc[[fold]] <- as.numeric(roc_obj$auc)
  }

  CV_AUC_folds[[model_name]] <- data.frame("Resample" = names(auc),
                                           "AUC" = unlist(auc))
}

# For each model: average RMSE and average AUC for models ----------------------------------

CV_RMSE <- list()
CV_AUC <- list()

for (model_name in names(logit_models)) {
```

```
  CV_RMSE[[model_name]] <- mean(CV_RMSE_folds[[model_name]]$RMSE)
  CV_AUC[[model_name]] <- mean(CV_AUC_folds[[model_name]]$AUC)
}

# We have 6 models, (5 logit and the logit lasso). For each we have a 5-CV RMSE and AUC.
# We pick our preferred model based on that. --------------------------------------------

nvars <- lapply(logit_models, FUN = function(x) length(x$coefnames))
nvars[["LASSO"]] <- sum(lasso_coeffs != 0)

logit_summary1 <- data.frame("Number of predictors" = unlist(nvars),
                             "CV RMSE" = unlist(CV_RMSE),
                             "CV AUC" = unlist(CV_AUC))
logit_summary1
```

```
##       Number.of.predictors   CV.RMSE    CV.AUC
## X1                      11 0.3474623 0.6427885
## X2                      18 0.3411204 0.6910375
## X3                      35 0.3393773 0.7066228
## X4                      79 0.3381581 0.7100193
## X5                     153 0.3388932 0.7060183
## LASSO                  103 0.3378407 0.6843364
```

```
kable(x = logit_summary1, format = "latex", booktabs=TRUE,  digits = 3, row.names = TRUE,
      linesep = "", col.names = c("Number of predictors","CV RMSE","CV AUC")) %>%
  cat(.,file= paste0(output, "logit_summary1.tex"))
```

The table compares the performance of six logistic regression models, each using a different set of predictors, based on their average cross-validated RMSE and AUC. As the number of predictors increases from X1 to X5, the CV RMSE generally decreases, indicating improved predictive accuracy. The AUC also improves, reaching its highest value of 0.710 with model X4, suggesting this model strikes a good balance between complexity (79 predictors) and classification performance. The LASSO model uses 103 predictors but achieves only a moderate AUC of 0.684, slightly below X4 and X5, though it performs comparably in terms of RMSE. Overall, model X4 appears to offer the best trade-off between predictive accuracy and model simplicity, while LASSO serves as a regularized alternative that avoids overfitting by shrinking coefficients.

Thus to sum up, model X4 is picked as the preferred model because it yields the highest CV AUC and second lowest CV RMSE. It is used on the holdout set.

The following code evaluates the performance of the best-performing logistic regression model (X4) on the holdout dataset, which was not used during training or cross-validation. It first extracts the model labeled "X4" from the list of trained models and uses it to generate predicted probabilities for fast growth in the holdout data. These predicted probabilities are then stored as a new column in the holdout dataset. Finally, the code calculates the root mean squared error (RMSE) between the predicted probabilities and the actual binary fast growth outcomes in the holdout set, providing a measure of how well the model generalizes to unseen data.

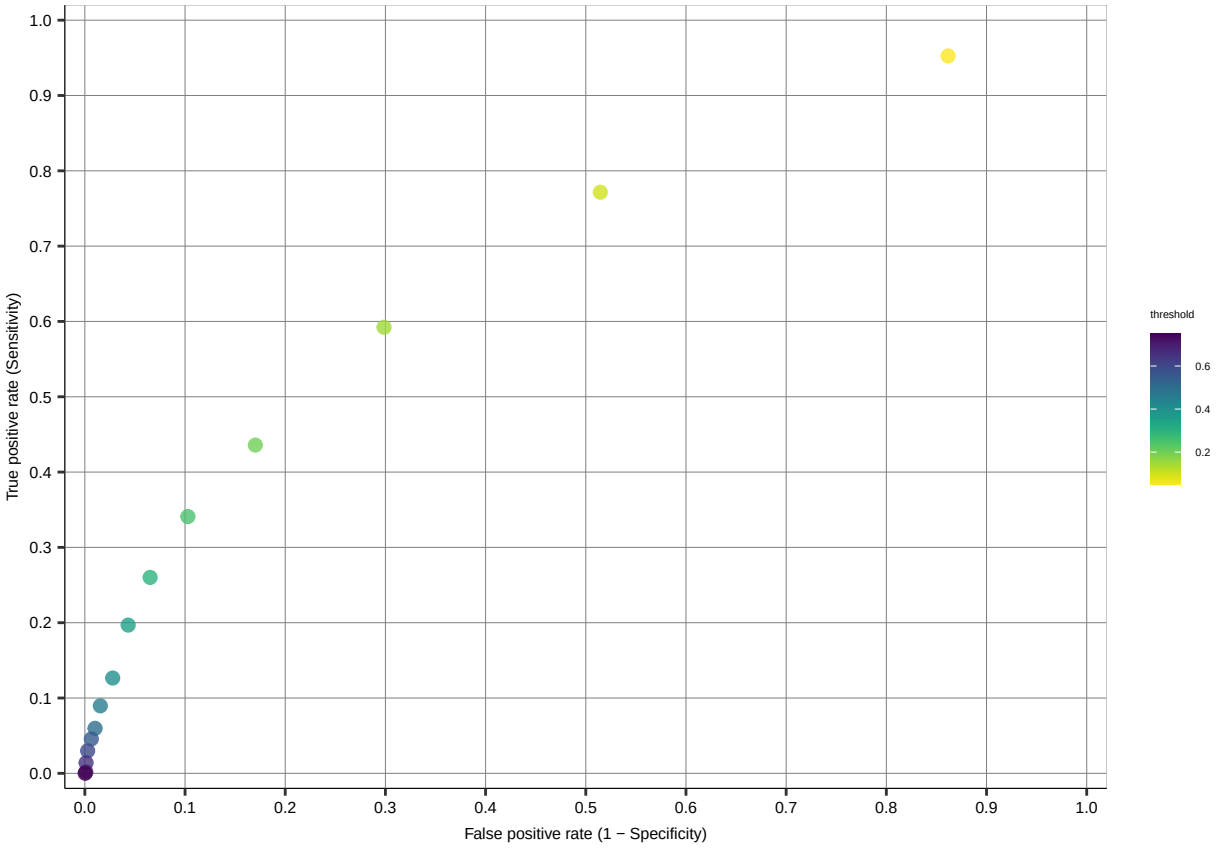
```
# Take best model and estimate RMSE on holdout    ------------------------------------------

best_logit_no_loss <- logit_models[["X4"]]

logit_predicted_probabilities_holdout <- predict(best_logit_no_loss, newdata = data_holdout, type = "pr
data_holdout[,"best_logit_no_loss_pred"] <- logit_predicted_probabilities_holdout[,"fast_growth"]
RMSE(data_holdout[, "best_logit_no_loss_pred", drop=TRUE], data_holdout$fast_growth)
```

```
## [1] 0.3417873
```

The model performs very well on the holdout set. The RMSE is barely higher.

The next code chunk evaluates the classification performance of the selected best logistic regression model (X4) on the holdout dataset using multiple diagnostic tools. It first constructs a discrete ROC curve by applying different probability thresholds (from 0.05 to 0.75) to convert predicted probabilities into binary class predictions, computing the corresponding true positive and false positive rates for each threshold using confusion matrices. These are plotted to visually assess how threshold choice affects model sensitivity and specificity. Then, it computes a continuous ROC curve based on the raw predicted probabilities using the `roc()` function and plots it to measure overall classification performance. The code also examines different threshold choices beyond the default of 0.5—specifically comparing it to the average predicted probability across the holdout set—and prints the resulting confusion matrices. Finally, it generates a calibration plot, which checks how well the predicted probabilities align with actual observed frequencies of fast-growing firms, providing insight into the model's probabilistic accuracy and reliability.

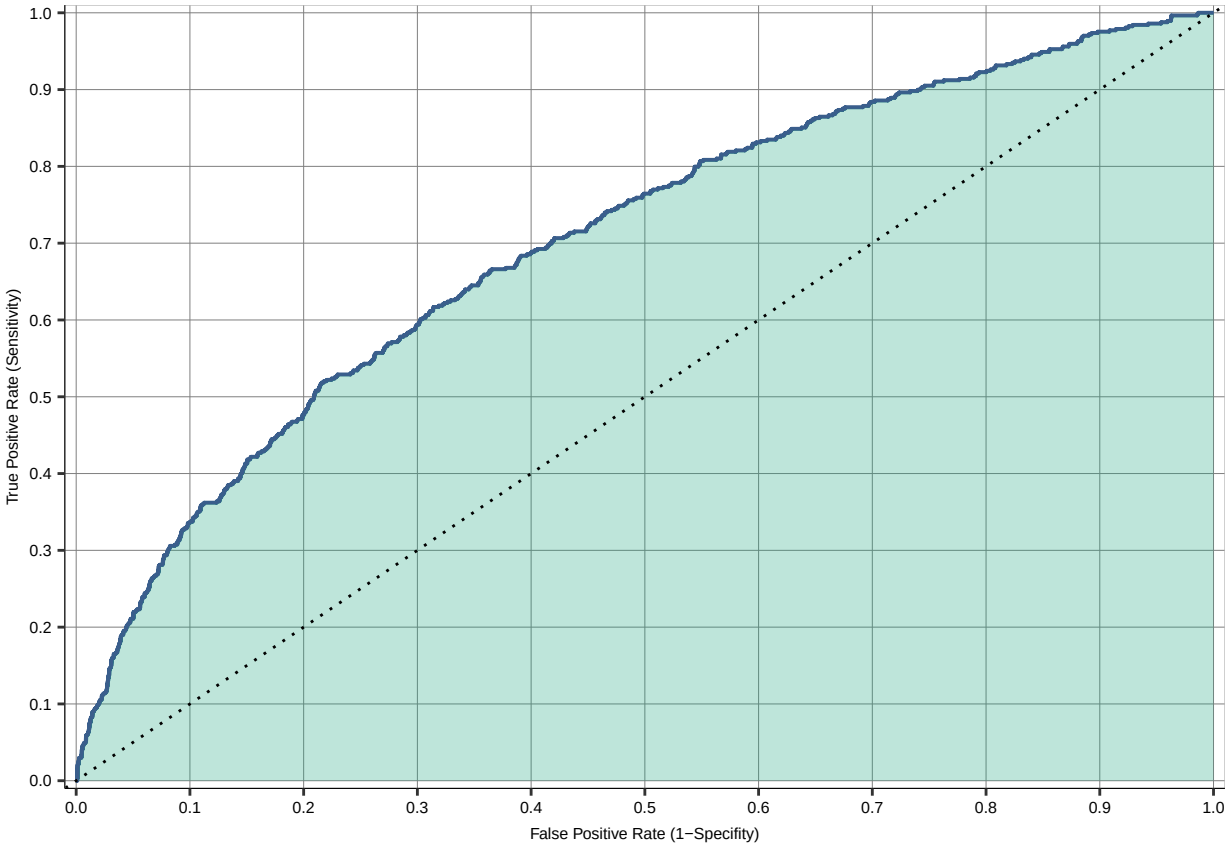
```
# discrete ROC (with thresholds in steps) on holdout -----------------------------------------------
thresholds <- seq(0.05, 0.75, by = 0.05)

cm <- list()
true_positive_rates <- c()
false_positive_rates <- c()
for (thr in thresholds) {
  holdout_prediction <- ifelse(data_holdout[,"best_logit_no_loss_pred"] < thr, "no_fast_growth", "fast_
    factor(levels = c("no_fast_growth", "fast_growth"))
  cm_thr <- confusionMatrix(holdout_prediction,data_holdout$fast_growth_f)$table
  cm[[as.character(thr)]] <- cm_thr
  true_positive_rates <- c(true_positive_rates, cm_thr["fast_growth", "fast_growth"] /
                             (cm_thr["fast_growth", "fast_growth"] + cm_thr["no_fast_growth", "fast_grow
  false_positive_rates <- c(false_positive_rates, cm_thr["fast_growth", "no_fast_growth"] /
                              (cm_thr["fast_growth", "no_fast_growth"] + cm_thr["no_fast_growth", "no_fa
}

tpr_fpr_for_thresholds <- tibble(
  "threshold" = thresholds,
  "true_positive_rate" = true_positive_rates,
  "false_positive_rate" = false_positive_rates
)

discrete_roc_plot <- ggplot(
  data = tpr_fpr_for_thresholds,
  aes(x = false_positive_rate, y = true_positive_rate, color = threshold)) +
  labs(x = "False positive rate (1 - Specificity)", y = "True positive rate (Sensitivity)") +
  geom_point(size=2, alpha=0.8) +
  scale_color_viridis(option = "D", direction = -1) +
  scale_x_continuous(expand = c(0.01,0.01), limit=c(0,1), breaks = seq(0,1,0.1)) +
  scale_y_continuous(expand = c(0.01,0.01), limit=c(0,1), breaks = seq(0,1,0.1)) +
  theme_bg() +
  theme(legend.position ="right") +
  theme(legend.title = element_text(size = 4),
        legend.text = element_text(size = 4),
        legend.key.size = unit(.4, "cm"))
discrete_roc_plot
```




```
save_fig("ch17-figure-2a-roc-discrete", output, "small")
```

```
## pdf
##   2
```

```
# continuous ROC on holdout with best model (Logit 4) ----------------------------------------

roc_obj_holdout <- roc(data_holdout$fast_growth, data_holdout$best_logit_no_loss_pred)

createRocPlot(roc_obj_holdout, "best_logit_no_loss_roc_plot_holdout")
```




```
# Confusion table with different tresholds ----------------------------------------------------------

# default: the threshold 0.5 is used to convert probabilities to binary classes
logit_class_prediction <- predict(best_logit_no_loss, newdata = data_holdout)
summary(logit_class_prediction)
```

```
## no_fast_growth    fast_growth
##           3740             67
```

```
# confusion matrix: summarize different type of errors and successfully predicted cases
# positive = "yes": explicitly specify the positive case
cm_object1 <- confusionMatrix(logit_class_prediction, data_holdout$fast_growth_f, positive = "fast_grow
cm1 <- cm_object1$table
cm1
```

```
##                Reference
## Prediction       no_fast_growth fast_growth
##   no_fast_growth           3205         535
##   fast_growth                33          34
```

```
# we can apply different thresholds

# 0.5 same as before
holdout_prediction <-
```

```
  ifelse(data_holdout$best_logit_no_loss_pred < 0.5, "no_fast_growth", "fast_growth") %>%
  factor(levels = c("no_fast_growth", "fast_growth"))
cm_object1b <- confusionMatrix(holdout_prediction,data_holdout$fast_growth_f)
cm1b <- cm_object1b$table
cm1b
```

```
##                Reference
## Prediction       no_fast_growth fast_growth
##   no_fast_growth           3205         535
##   fast_growth                33          34
```

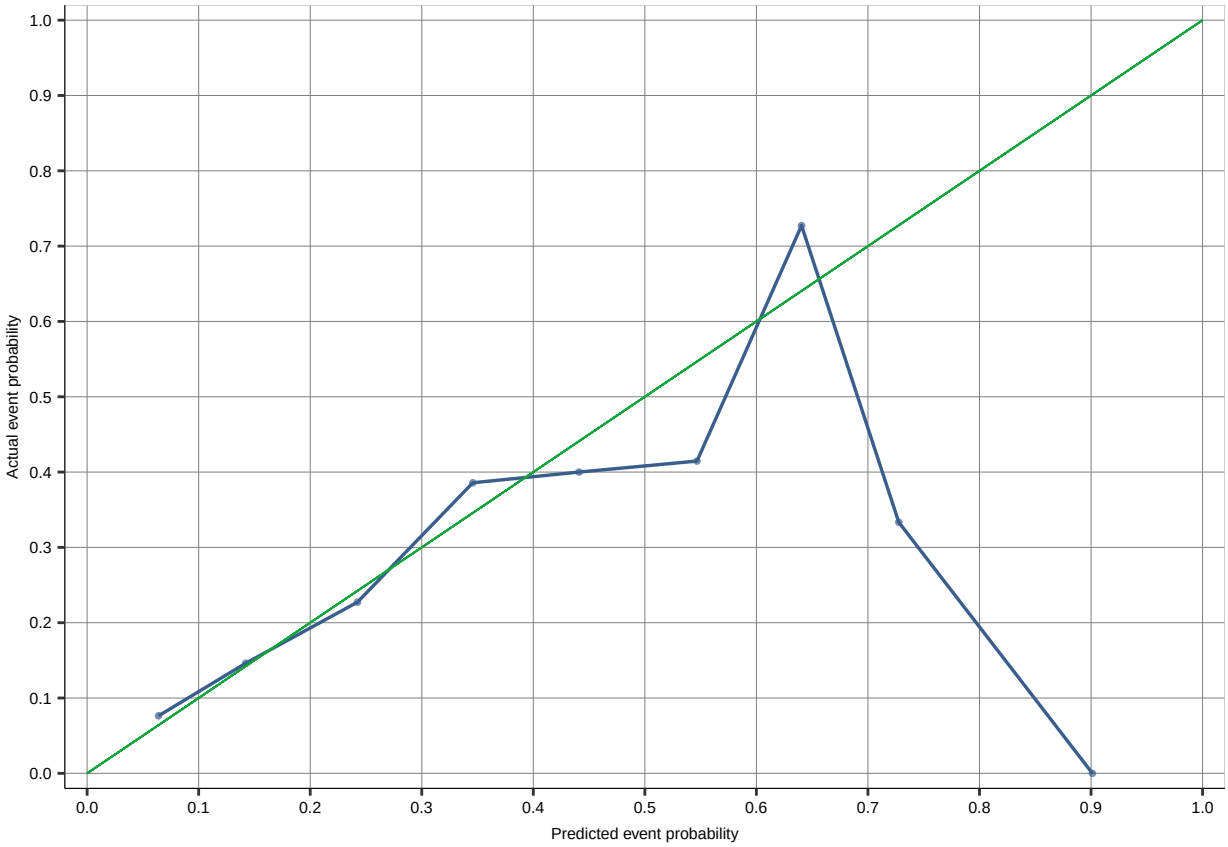
```
# a sensible choice: mean of predicted probabilities
mean_predicted_default_prob <- mean(data_holdout$best_logit_no_loss_pred)
mean_predicted_default_prob
```

```
## [1] 0.1447424
```

```
holdout_prediction <-
  ifelse(data_holdout$best_logit_no_loss_pred < mean_predicted_default_prob, "no_fast_growth", "fast_gr
  factor(levels = c("no_fast_growth", "fast_growth"))
cm_object2 <- confusionMatrix(holdout_prediction,data_holdout$fast_growth_f)
cm2 <- cm_object2$table
cm2
```

```
##                Reference
## Prediction       no_fast_growth fast_growth
##   no_fast_growth           2223         221
##   fast_growth              1015         348
```

```
# Calibration curve ------------------------------------------------------------
# how well do estimated vs actual event probabilities relate to each other?


create_calibration_plot(data_holdout,
  file_name = "ch17-figure-1-logit-m4-calibration",
  prob_var = "best_logit_no_loss_pred",
  actual_var = "fast_growth",
  n_bins = 10)
```



The confusion matrices for three threshold choices show the following: the default threshold of 0.5, the same default repeated, and the mean predicted probability (approximately 0.1447). At the 0.5 threshold, the model achieves high specificity (many true negatives) but misses most fast-growing firms, with only 34 true positives out of 569. When the threshold is lowered to the mean predicted probability, the model identifies 348 fast-growing firms correctly (true positives), at the cost of more false positives. This trade-off highlights the importance of choosing a threshold that aligns with the business context—whether it's more costly to miss a high-growth firm (false negative) or incorrectly flag a low-growth firm (false positive).

The first two plots show ROC curve evaluations for the best logistic regression model on the holdout set. The first plot is a discrete ROC curve, where thresholds from 0.05 to 0.75 are tested step-by-step, and the resulting true positive rate (sensitivity) and false positive rate (1 - specificity) are plotted. As expected, higher thresholds lower the false positive rate but also reduce sensitivity. The second plot presents the continuous ROC curve, which is smoother and more comprehensive. The model shows reasonable discrimination ability with an AUC likely in the range of 0.70, suggesting it performs better than random guessing but leaves room for improvement.

The third plot is a calibration curve, which compares the predicted probabilities from the model to the actual observed frequency of fast-growing firms. The curve deviates from the 45-degree line, especially in the middle and upper probability ranges, indicating that the model tends to overestimate the probability of fast growth in those bins. This suggests that while the model is good at ranking firms by risk, its probability estimates may not be well calibrated and could benefit from post-processing techniques.

Part 2: Classification

In the context of identifying fast-growing firms, the business goal is often to allocate resources such as credit, investment, or support services efficiently. A false positive (FP)—incorrectly classifying a non-growing firm as fast-growing—might result in wasted resources, missed returns, or poor lending decisions. On the other

hand, a false negative (FN)—failing to identify a genuinely fast-growing firm—means missing out on a valuable opportunity for partnership, investment, or support. From a strategic perspective, the cost of a false negative is likely higher than a false positive, since overlooking a high-potential firm could mean losing out on growth-driven returns or competitive advantage.

We therefore define an asymmetric loss function to reflect this trade-off: let the cost of a false positive be \$10 (e.g., minimal screening or administrative cost), and the cost of a false negative be \$5 (e.g., missed opportunity cost, lost market share, or foregone investment return). This 6:5 ratio reflects the fact that the cost of wrongly allocating resources to the wrong firm may be more damaging than missing a growth opportunity. By minimizing expected loss using this loss function, we can choose a threshold that is not just statistically optimal, but also aligned with real-world business priorities.

This code calculates the optimal classification threshold for each logistic regression model by incorporating an asymmetric loss function, where the cost of a false positive is set to 6 and the cost of a false negative is 5. For each model and each of the five cross-validation folds, it computes the ROC curve and uses the `coords()` function with a cost-weighted version of Youden's index to find the threshold that minimizes expected misclassification loss. It then computes the expected loss at that threshold by weighting false positives and false negatives accordingly. The average optimal threshold and average expected loss are stored across folds, and results from fold 5 are separately tracked for reference. Finally, a summary table is created showing, for each model, the average optimal threshold, the fold 5 threshold, the average expected loss, and the fold 5 expected loss. This output is exported in LaTeX format for reporting, helping to guide model selection based on both performance and cost-sensitive classification.

```
# Introduce loss function
# relative cost of of a false negative classification (as compared with a false positive classification
FP=6
FN=5
cost = FN/FP
# the prevalence, or the proportion of cases in the population (n.cases/(n.controls+n.cases))
prevelance = sum(data_train$fast_growth)/length(data_train$fast_growth)

# Draw ROC Curve and find optimal threshold with loss function --------------------------

best_tresholds <- list()
expected_loss <- list()
logit_cv_rocs <- list()
logit_cv_threshold <- list()
logit_cv_expected_loss <- list()

for (model_name in names(logit_models)) {

  model <- logit_models[[model_name]]
  colname <- paste0(model_name,"_prediction")

  best_tresholds_cv <- list()
  expected_loss_cv <- list()

  for (fold in c("Fold1", "Fold2", "Fold3", "Fold4", "Fold5")) {
    cv_fold <-
      model$pred %>%
      filter(Resample == fold)

    roc_obj <- roc(cv_fold$obs, cv_fold$fast_growth)
    best_treshold <- coords(roc_obj, "best", ret="all", transpose = FALSE,
                            best.method="youden", best.weights=c(cost, prevelance))
```

```
    best_tresholds_cv[[fold]] <- best_treshold$threshold
    expected_loss_cv[[fold]] <- (best_treshold$fp*FP + best_treshold$fn*FN)/length(cv_fold$fast_growth)
  }

  # average
  best_tresholds[[model_name]] <- mean(unlist(best_tresholds_cv))
  expected_loss[[model_name]] <- mean(unlist(expected_loss_cv))

  # for fold #5
  logit_cv_rocs[[model_name]] <- roc_obj
  logit_cv_threshold[[model_name]] <- best_treshold
  logit_cv_expected_loss[[model_name]] <- expected_loss_cv[[fold]]

  }

logit_summary2 <- data.frame("Avg of optimal thresholds" = unlist(best_tresholds),"Threshold for Fold5"

kable(x = logit_summary2, format = "latex", booktabs=TRUE,  digits = 3, row.names = TRUE, linesep = "",
  cat(.,file= paste0(output, "logit_summary2.tex"))

logit_summary2
```

```
##       Avg.of.optimal.thresholds Threshold.for.Fold5 Avg.expected.loss
## X1                          Inf           0.4114381         0.7326152
## X2                    0.4756155           0.5022766         0.7245371
## X3                          Inf           0.5480388         0.7232232
## X4                    0.6000039           0.6137414         0.7184953
## X5                    0.5839249           0.5901707         0.7163942
## LASSO                 0.5285608           0.5620728         0.7270593
##       Expected.loss.for.Fold5
## X1                  0.7346470
## X2                  0.7195402
## X3                  0.7103448
## X4                  0.7152709
## X5                  0.7146141
## LASSO               0.7290969
```

The table summarizes the performance of six logistic regression models using a custom loss function where false positives cost 6 and false negatives cost 5. It reports the average of optimal thresholds (across cross-validation folds), the threshold selected in fold 5, and the average and fold 5-specific expected losses for each model. Models X4 and X5 achieve the lowest average expected losses (0.718 and 0.717 respectively), suggesting they are the best-performing models under the defined cost structure. The LASSO model performs slightly worse (0.727). Most models return meaningful average thresholds (between 0.47 and 0.60), indicating that a threshold higher than the default 0.5 would reduce expected misclassification costs. However, models X1 and X3 return `Inf` for the average threshold, which means that in at least one fold, the `coords()` function could not identify a finite threshold that improves classification based on the cost-weighted Youden index.

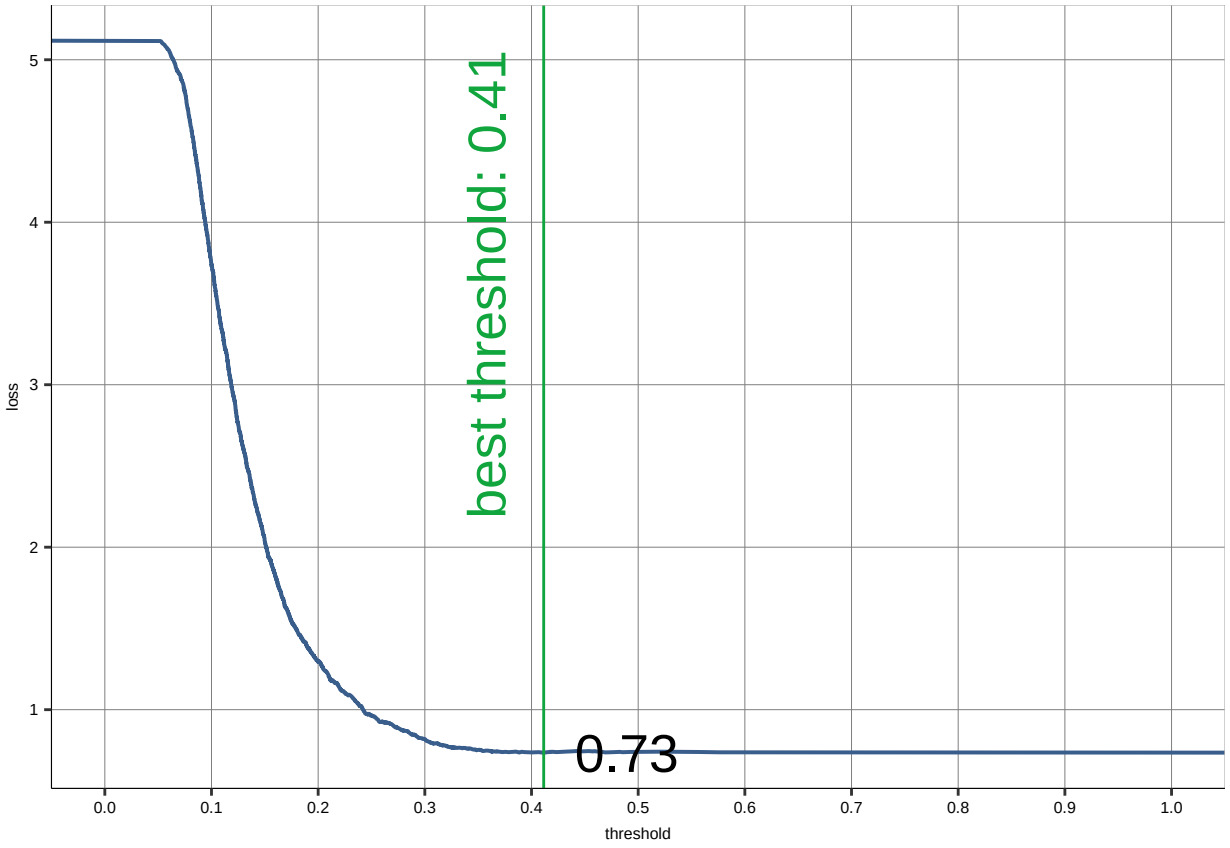
This can happen when the model performs poorly in a particular fold—specifically, when the ROC curve does not provide any cutoff that improves both sensitivity and specificity in a way that balances your cost function. If the classifier's predicted probabilities for fast growth are not well-separated from those of non-growers, or if one class dominates, the ROC object may not contain valid coordinates for determining a cost-optimal threshold, causing `Inf` to appear. Reducing the asymmetry of the cost ratio (by setting FP = 6 and FN = 5 instead of FP = 10 and FN = 5 as I had before) makes the loss function more forgiving and

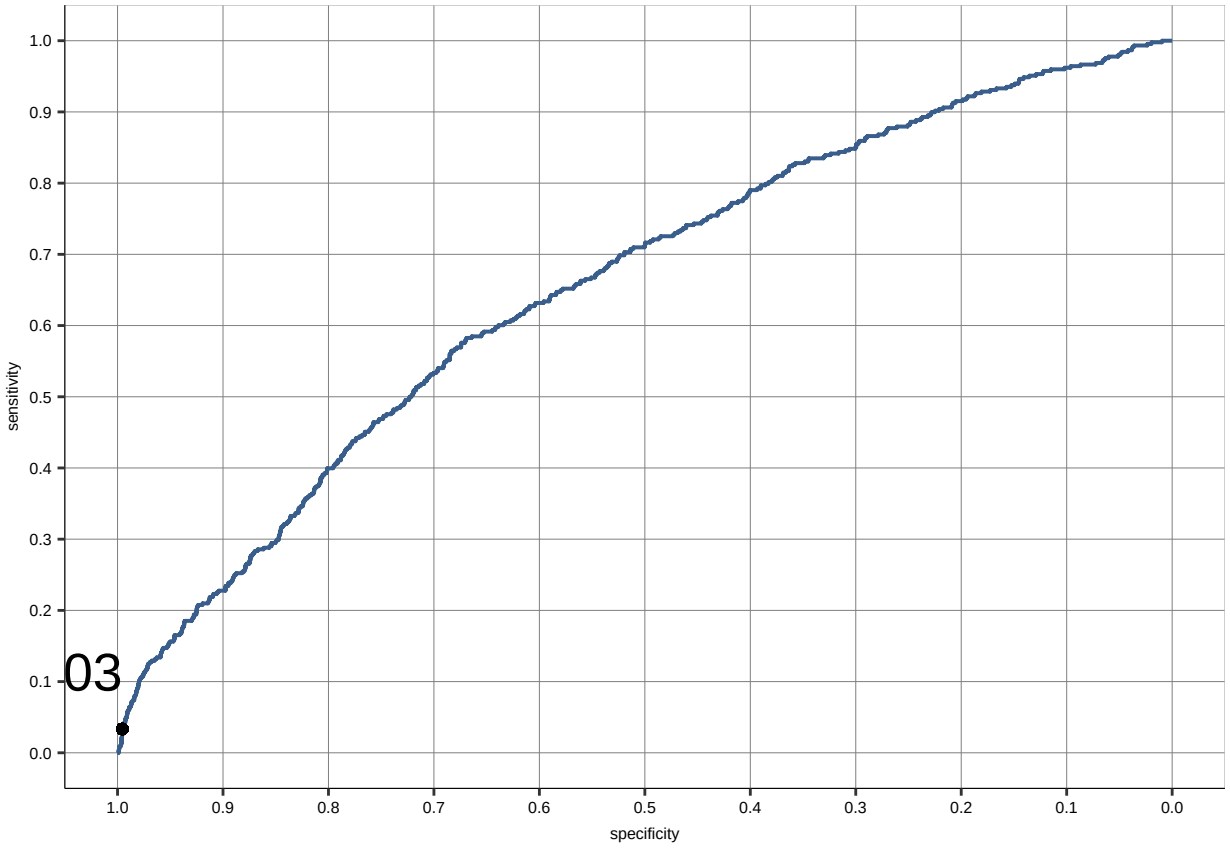
allows the optimization routine to find usable thresholds for more models. This suggests that models X1 and X3 may be underfitting or poorly calibrated compared to the others.

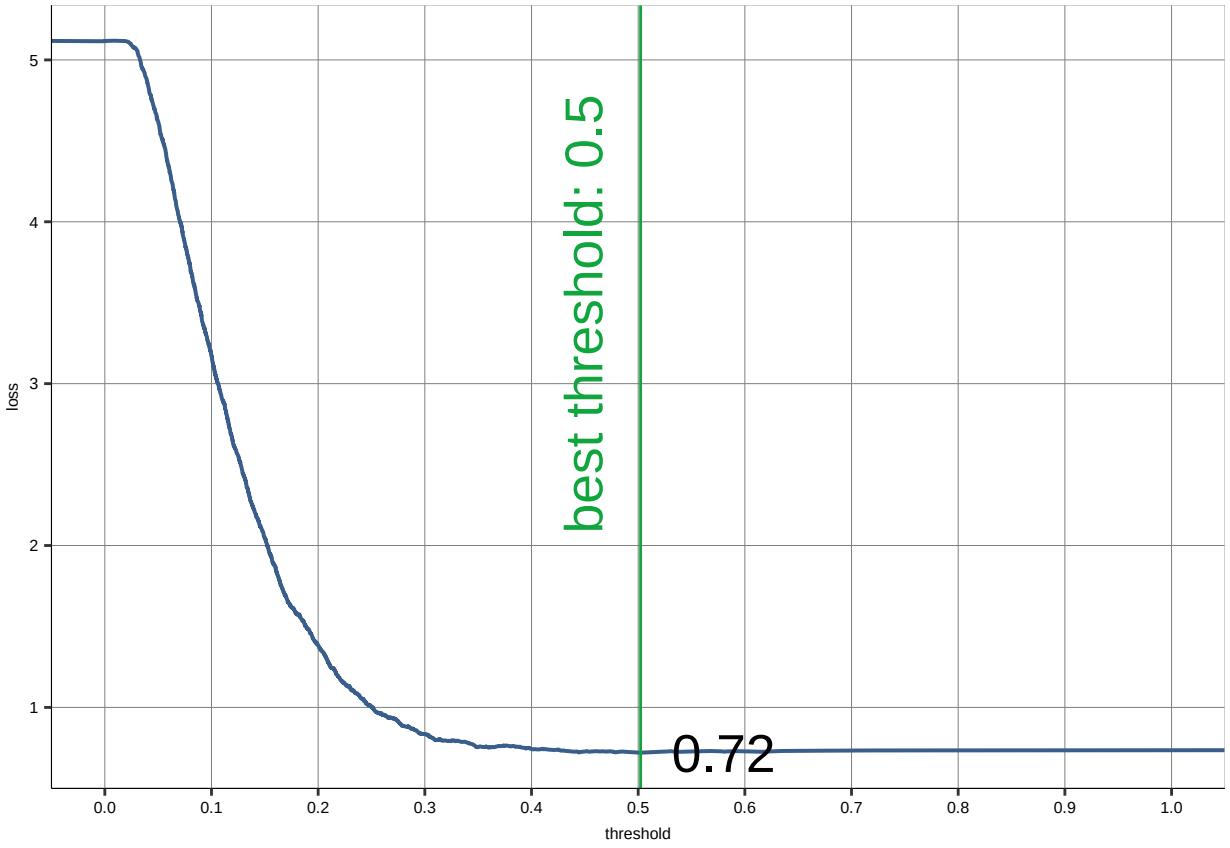
Overall, X4 appears to offer the most balanced and consistent performance, with a clearly defined optimal threshold (0.60) and the lowest fold-specific loss (0.715). In contrast, models X1 and X3 are less reliable, and their higher losses suggest weaker classification performance. This output helps identify which models minimize misclassification costs under the asymmetric loss function and provides optimal thresholds that can be used for final classification.

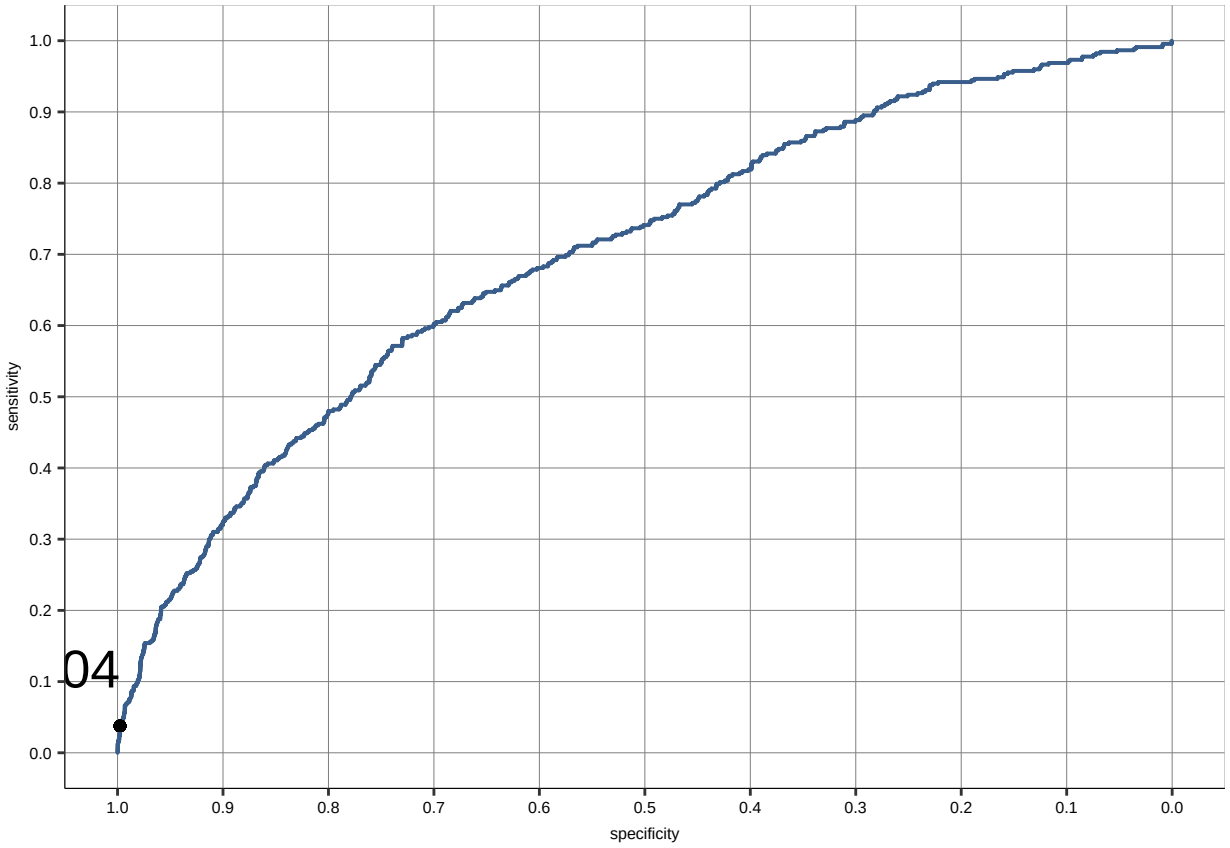
This code generates and displays plots that visualize the performance of each logistic regression model based on fold 5 of cross-validation. For each model, it retrieves the ROC curve (`r`) and the optimal threshold (`best_coords`) identified for fold 5. It then calls two custom functions—`createLossPlot()` and `createRocPlotWithOptimal()`—to generate a loss curve and an ROC curve that highlights the selected threshold. The plots are both saved (likely to file, using the `paste0(model_name, "_loss_plot")` naming convention) and displayed in the console or notebook using `print()`. This ensures that users can both visually inspect model performance interactively and have persistent copies for reporting. The loss plots help assess how expected misclassification costs vary across thresholds, while the ROC plots illustrate each model's true and false positive trade-offs, including the threshold chosen based on the specified loss function.

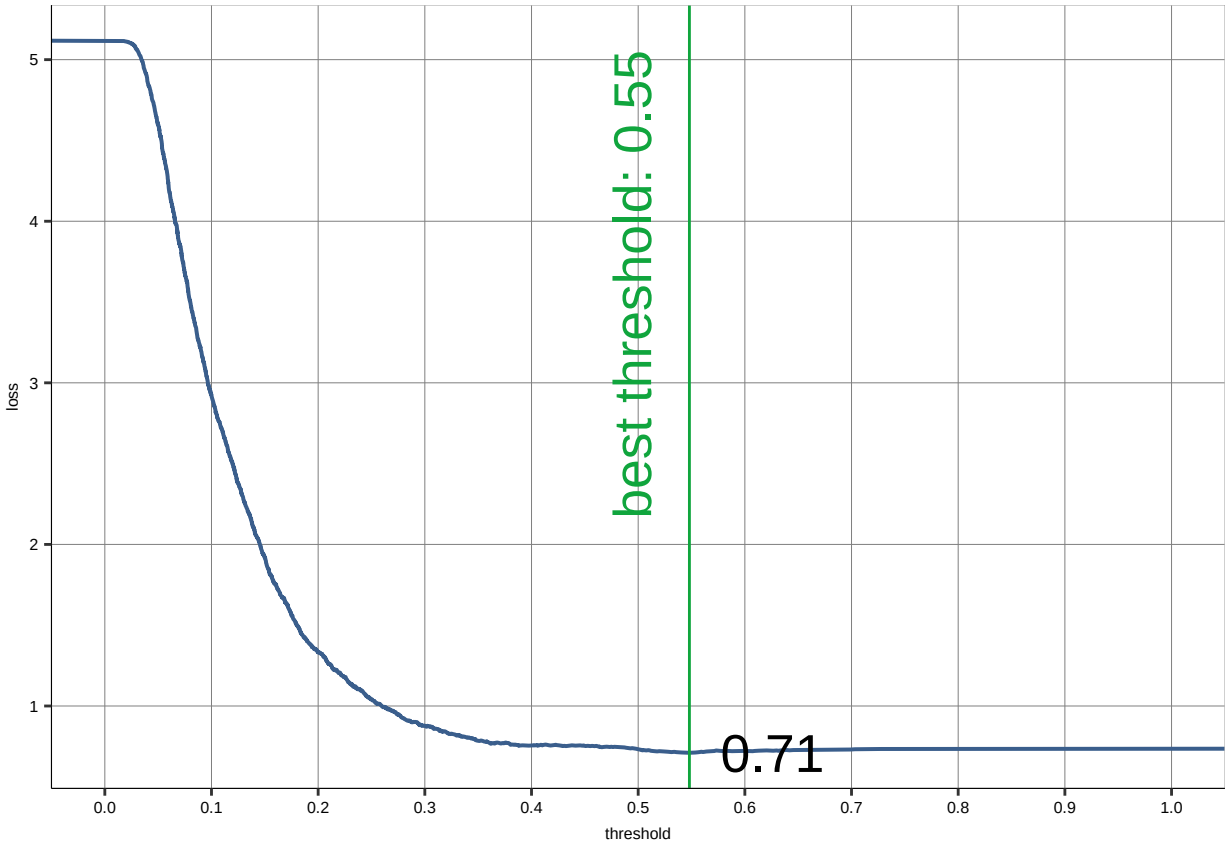
```
# Create plots based on Fold5 in CV ------------------------------------------

for (model_name in names(logit_cv_rocs)) {

  r <- logit_cv_rocs[[model_name]]
  best_coords <- logit_cv_threshold[[model_name]]
  createLossPlot(r, best_coords,
                 paste0(model_name, "_loss_plot"))
  createRocPlotWithOptimal(r, best_coords,
                           paste0(model_name, "_roc_plot"))
  print(createLossPlot(r, best_coords, paste0(model_name, "_loss_plot")))
  print(createRocPlotWithOptimal(r, best_coords, paste0(model_name, "_roc_plot")))
}
```

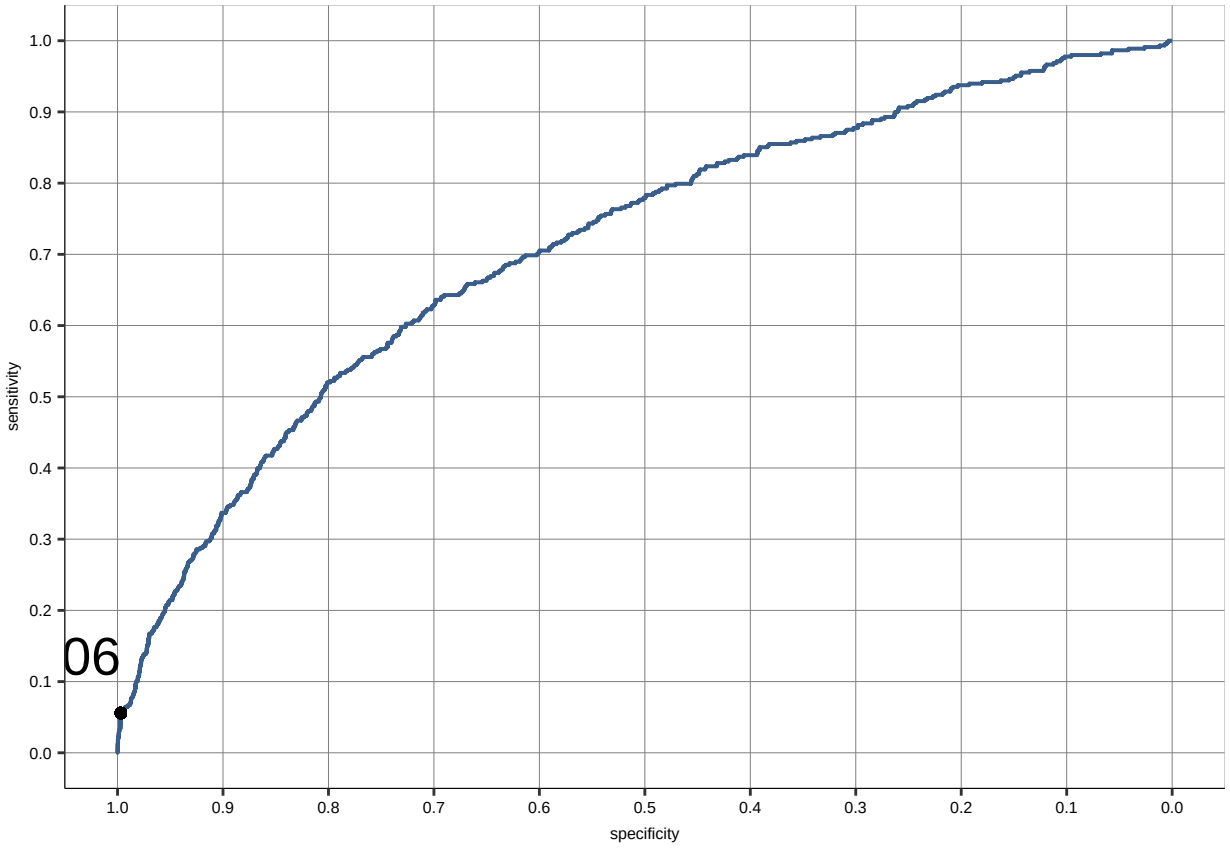

best threshold: 0.41
0.73
loss
threshold
5
4
3
2
1
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0

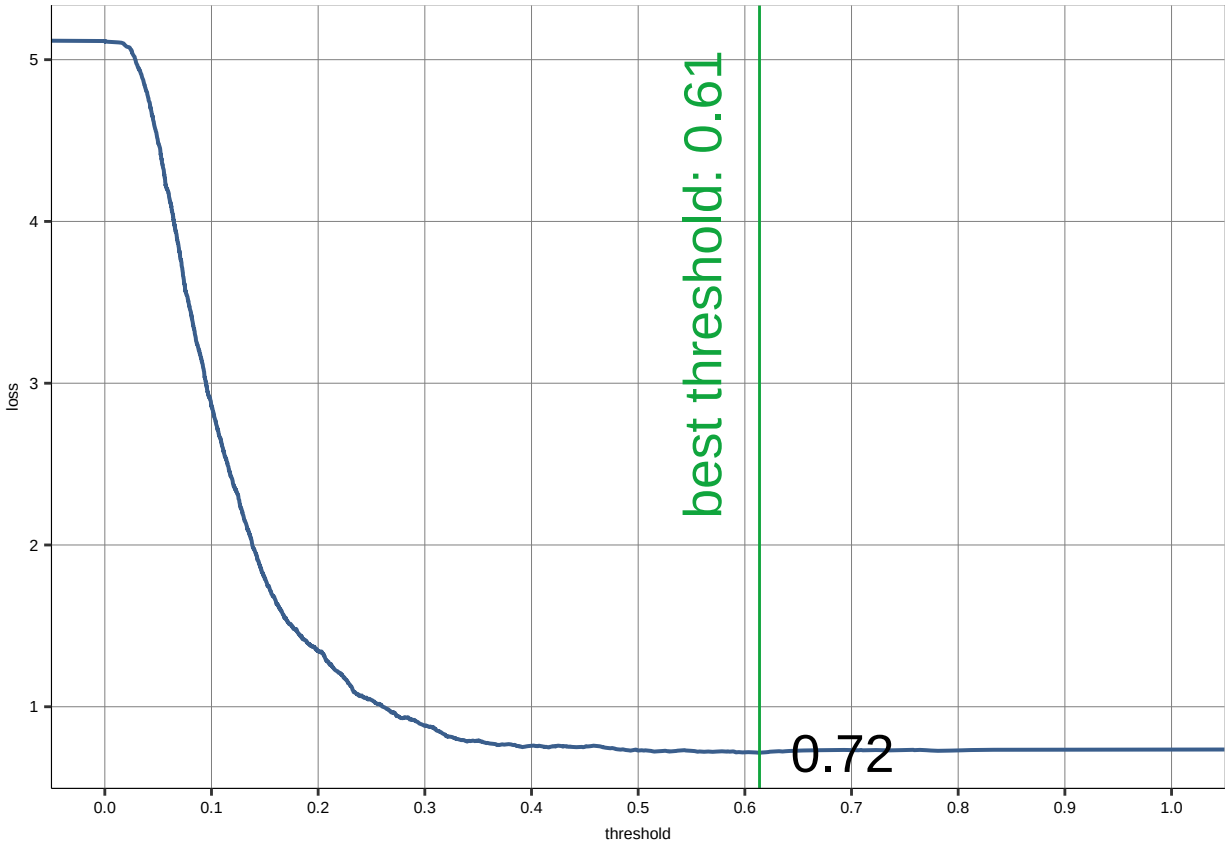

sensitivity
specificity
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
03

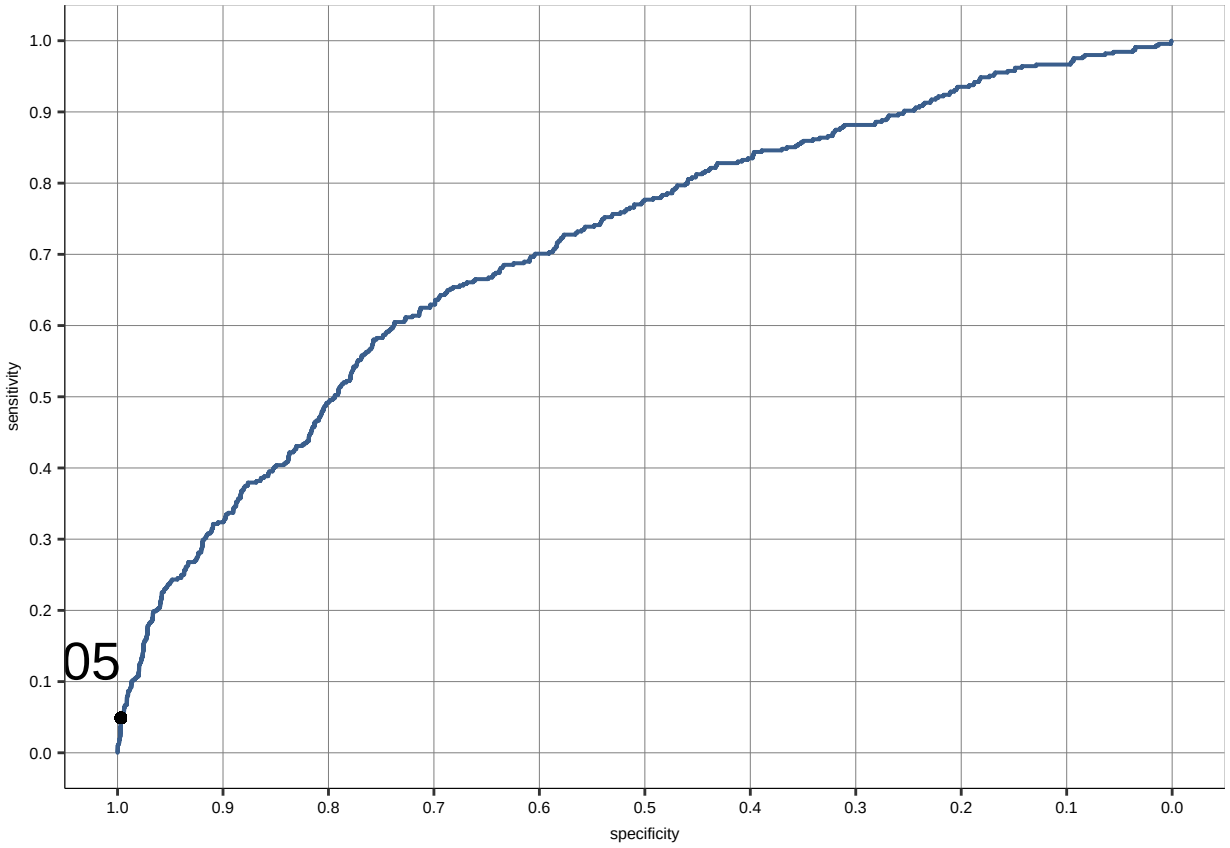

best threshold: 0.5
0.72
loss
threshold
5
4
3
2
1
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0

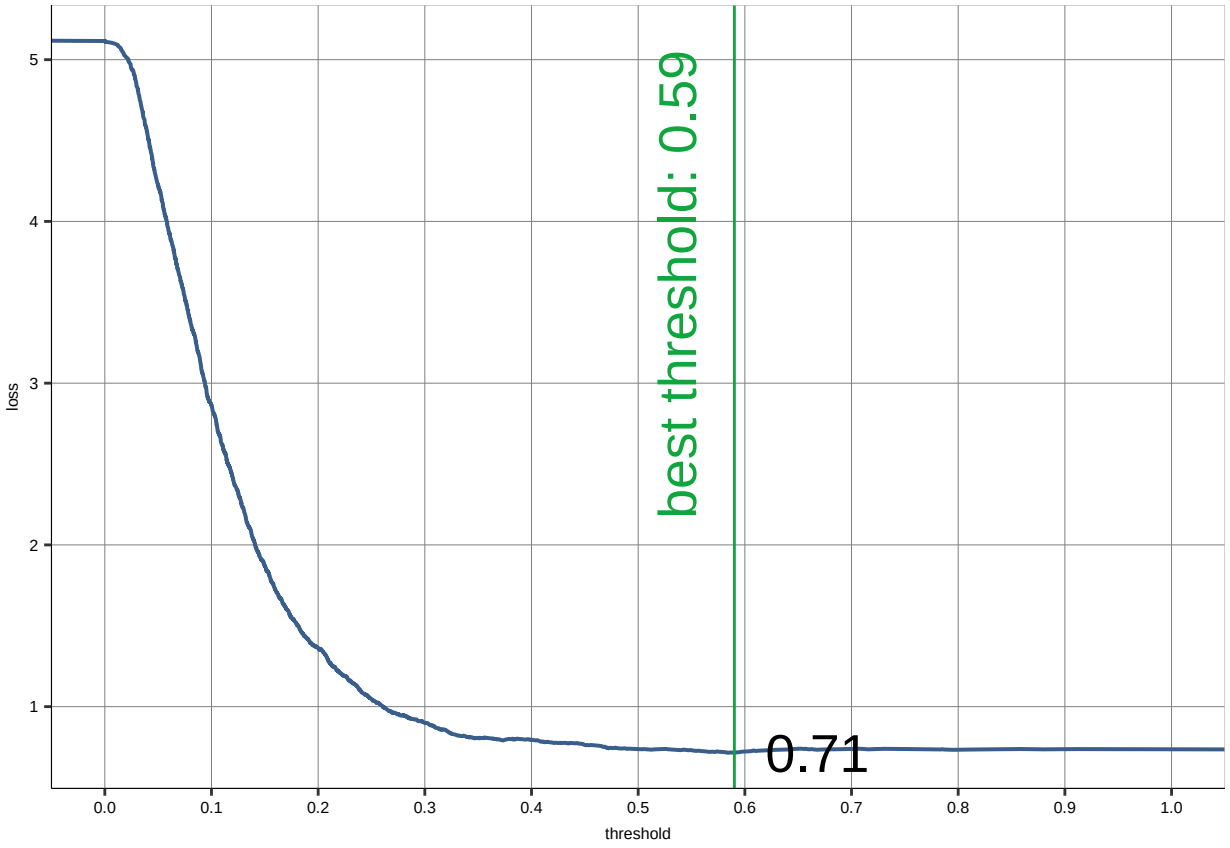

1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
sensitivity
04
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
specificity

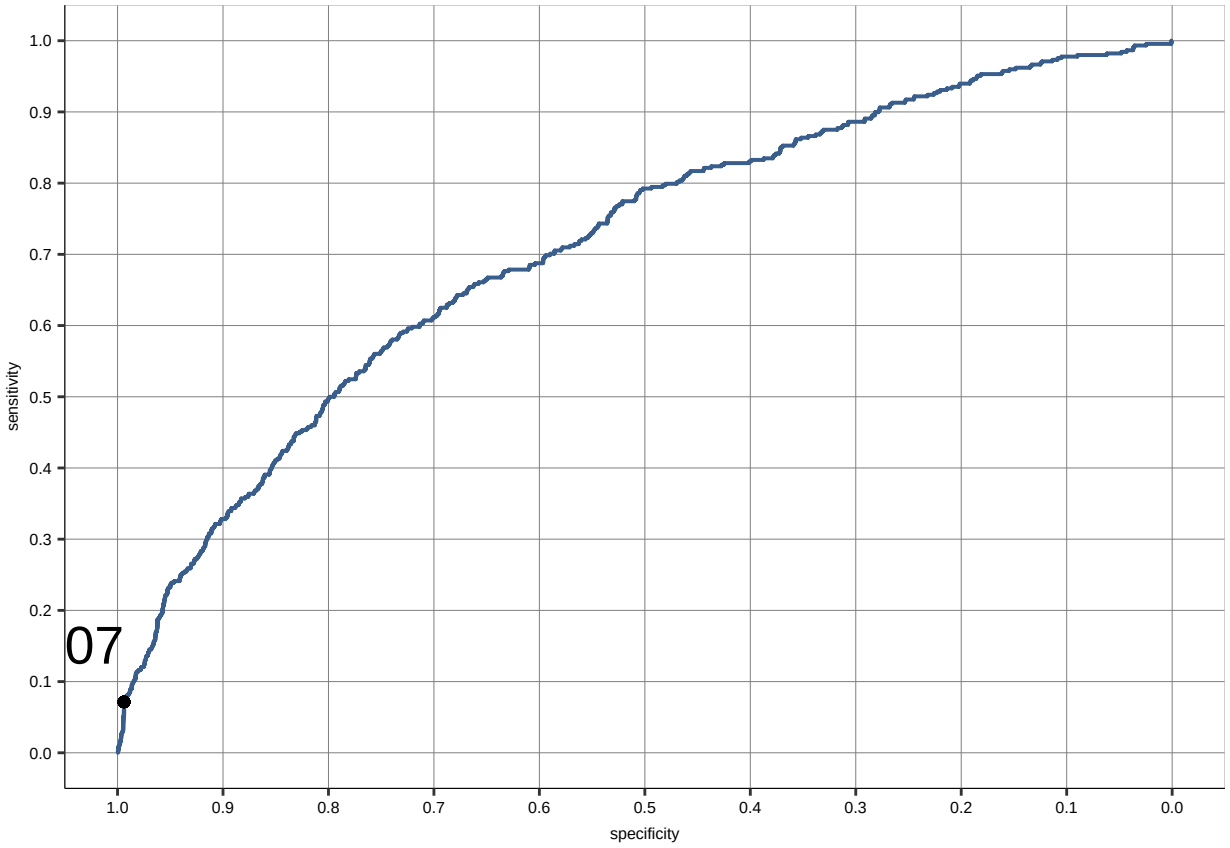

best threshold: 0.55
0.71
loss
threshold
5
4
3
2
1
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0

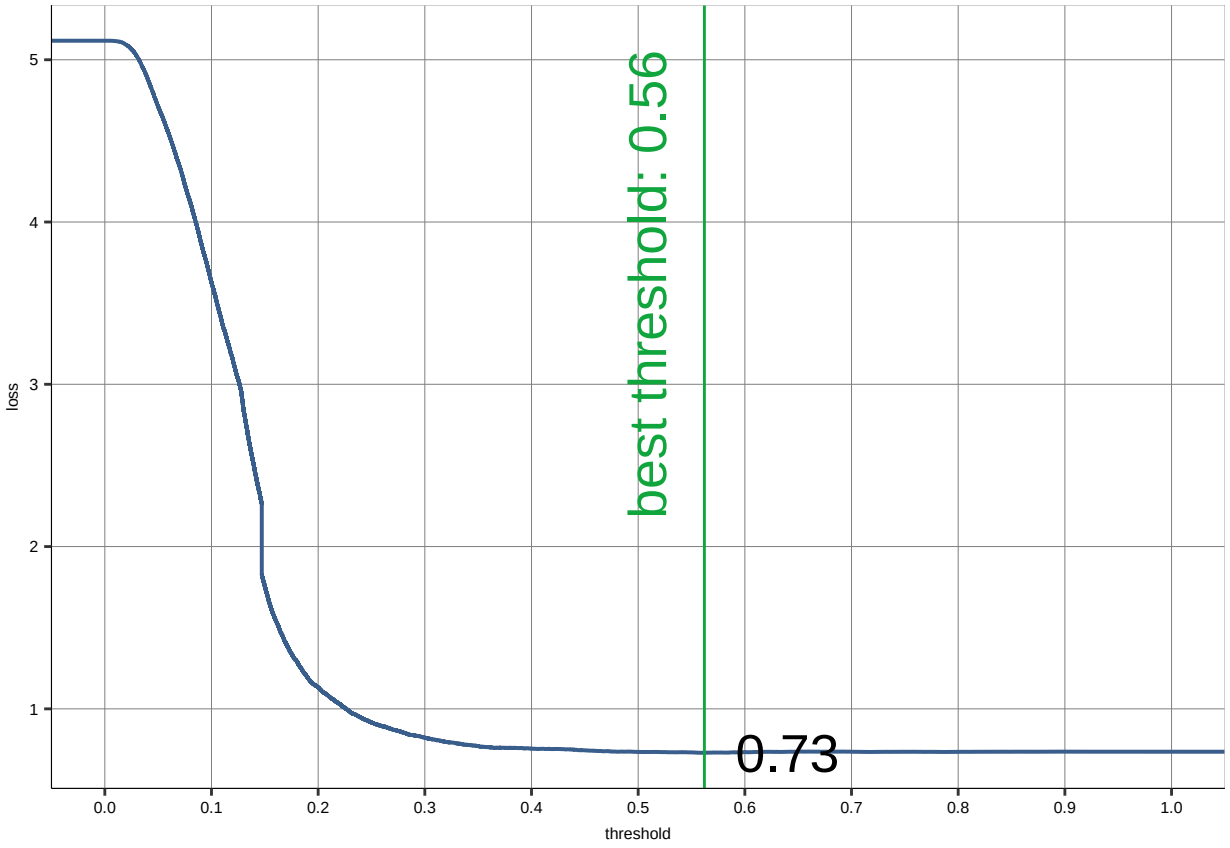

1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
sensitivity
06
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
specificity

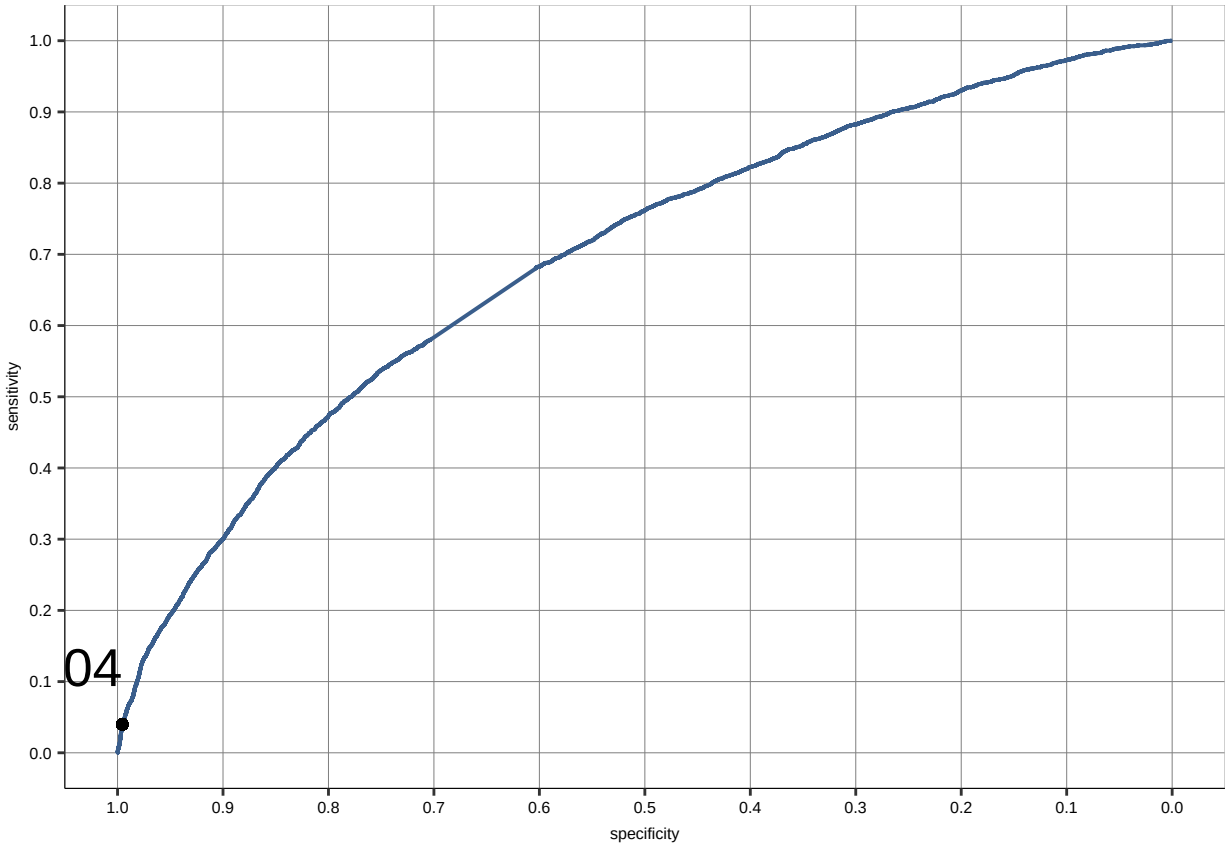

best threshold: 0.61
0.72
loss
threshold
5
4
3
2
1
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0


1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
sensitivity
05
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
specificity


best threshold: 0.59
0.71
loss
threshold
5
4
3
2
1
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0


sensitivity
specificity
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
07


best threshold: 0.56
0.73
loss
threshold
5
4
3
2
1
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0



Across all models, the cost-sensitive evaluation using fold 5 of cross-validation reveals that the optimal classification threshold consistently lies well above the standard 0.5—typically around 0.71 to 0.73—due to the higher penalty assigned to false positives. The loss curves demonstrate a clear decline in expected misclassification cost up to these thresholds, after which gains plateau, supporting more conservative classification. ROC curves across models, including the LASSO variant, show moderate performance, with no model achieving near-perfect discrimination between fast-growing and non-fast-growing firms. This suggests that while model predictions are informative, they remain imperfect, and careful threshold tuning is essential to align model decisions with real-world business costs. Overall, the evaluation highlights the importance of incorporating asymmetric costs and threshold optimization when predicting fast growth in firms.

This code evaluates the final performance of the best logistic regression model (X4), selected based on its lowest average expected loss during cross-validation. Using the model, predicted probabilities for fast growth are generated for the holdout set and stored. An ROC object is then created to assess performance, and the expected loss on the holdout set is computed using the previously determined optimal threshold, taking into account the specified costs of false positives and false negatives. Finally, the code applies this optimal threshold to classify firms as fast-growing or not, and generates a confusion matrix to summarize prediction results, including true positives, true negatives, and the types of misclassification. This step confirms how well the chosen model and threshold generalize to unseen data under the custom loss function.

```
# Pick best model based on average expected loss ----------------------------------

best_logit_with_loss <- logit_models[["X4"]]
best_logit_optimal_treshold <- best_tresholds[["X4"]]

logit_predicted_probabilities_holdout <- predict(best_logit_with_loss, newdata = data_holdout, type = "
data_holdout[,"best_logit_with_loss_pred"] <- logit_predicted_probabilities_holdout[,"fast_growth"]
```

```
# ROC curve on holdout
roc_obj_holdout <- roc(data_holdout$fast_growth, data_holdout[, "best_logit_with_loss_pred", drop=TRUE]

# Get expected loss on holdout
holdout_treshold <- coords(roc_obj_holdout, x = best_logit_optimal_treshold, input= "threshold",
                           ret="all", transpose = FALSE)
expected_loss_holdout <- (holdout_treshold$fp*FP + holdout_treshold$fn*FN)/length(data_holdout$fast_grou
expected_loss_holdout
```

```
## [1] 0.7391647
```

```
# Confusion table on holdout with optimal threshold
holdout_prediction <-
  ifelse(data_holdout$best_logit_with_loss_pred < best_logit_optimal_treshold, "no_fast_growth", "fast_g
  factor(levels = c("no_fast_growth", "fast_growth"))
cm_object3 <- confusionMatrix(holdout_prediction,data_holdout$fast_growth_f)
cm3 <- cm_object3$table
cm3
```

```
##                Reference
## Prediction       no_fast_growth fast_growth
##   no_fast_growth           3229         552
##   fast_growth                 9          17
```

Random forest

This section introduces the application of random forest models to predict fast-growing firms. Two types of random forests are used: probability forests and classification forests. A probability forest estimates the probability that a firm belongs to a specific class (such as being fast-growing), which makes it useful for applications where decisions depend on customized thresholds or cost-sensitive classification. In contrast, a classification forest directly predicts a binary outcome by taking a majority vote across trees, making it suitable for simple yes/no decision-making. Using both approaches allows for a more comprehensive evaluation of model performance depending on the prediction goal.

Probability forest

The code implements a probability forest using 5-fold cross-validation with the ranger method via the caret package. It tunes the hyperparameters mtry and min.node.size using a grid search and evaluates performance using RMSE and AUC across folds. Once the best-performing model is selected, the code computes the optimal classification threshold for each fold based on a predefined asymmetric loss function that penalizes false positives and false negatives differently. These thresholds are used to calculate expected loss, and their averages are stored. The model is then applied to the holdout dataset, where predicted probabilities are generated and evaluated using RMSE, AUC, and expected loss based on the optimal threshold. Additionally, ROC and loss plots are created to visualize model performance under the cost-sensitive setup.

```
rawvars <-  c("curr_assets", "curr_liab", "extra_exp", "extra_inc", "extra_profit_loss", "fixed_assets"
              "inc_bef_tax", "intang_assets", "inventories", "liq_assets", "material_exp", "personnel_e
              "profit_loss_year", "sales", "share_eq", "subscribed_cap")
rfvars  <-  c("sales_mil", "d1_sales_mil_log", rawvars, firm)

# 5 fold cross-validation
```

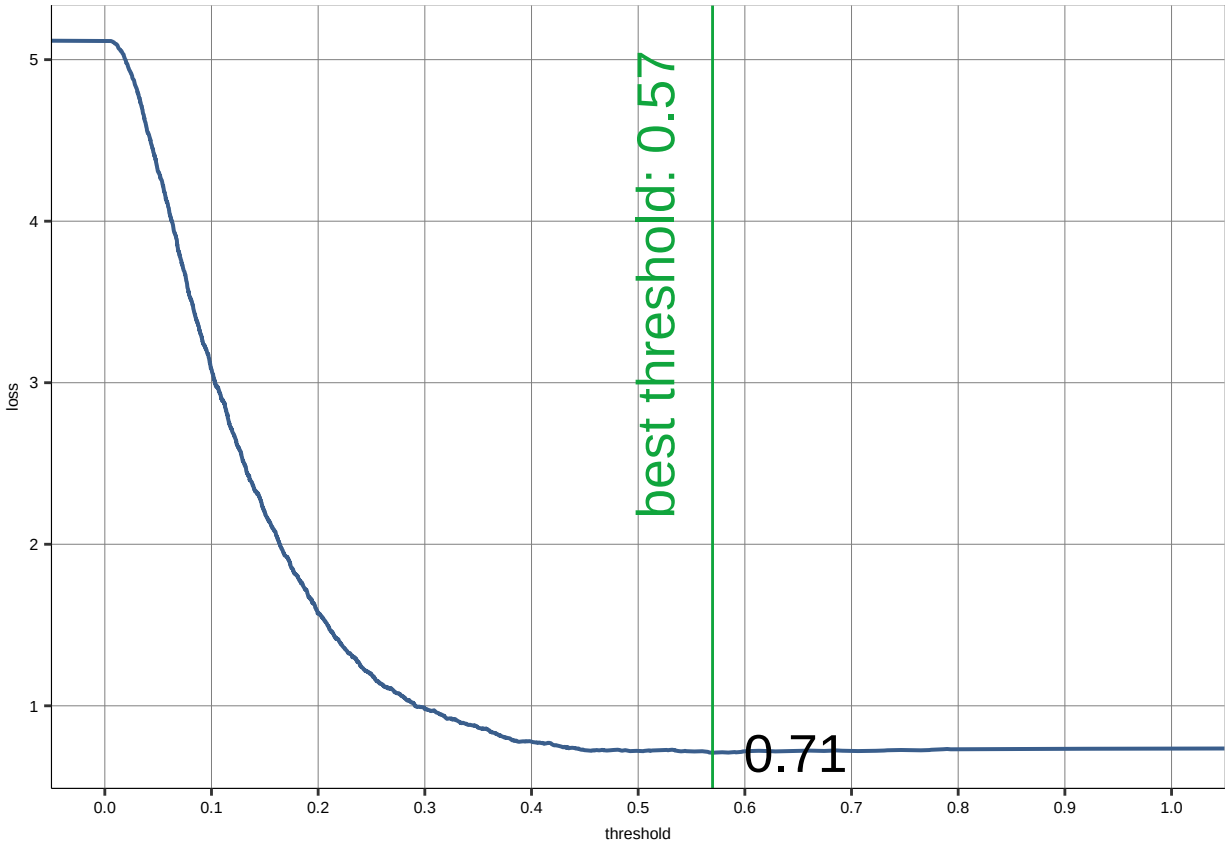
```
train_control <- trainControl(
  method = "cv",
  n = 5,
  classProbs = TRUE, # same as probability = TRUE in ranger
  summaryFunction = twoClassSummaryExtended,
  savePredictions = TRUE
)
train_control$verboseIter <- TRUE

tune_grid <- expand.grid(
  .mtry = c(5, 6, 7),
  .splitrule = "gini",
  .min.node.size = c(10, 15)
)

# getModelInfo("ranger")
set.seed(13505)
rf_model_p <- train(
  formula(paste0("fast_growth_f ~ ", paste0(rfvars , collapse = " + "))),
  method = "ranger",
  data = data_train,
  tuneGrid = tune_grid,
  trControl = train_control
)
```

```
rf_model_p$results
```

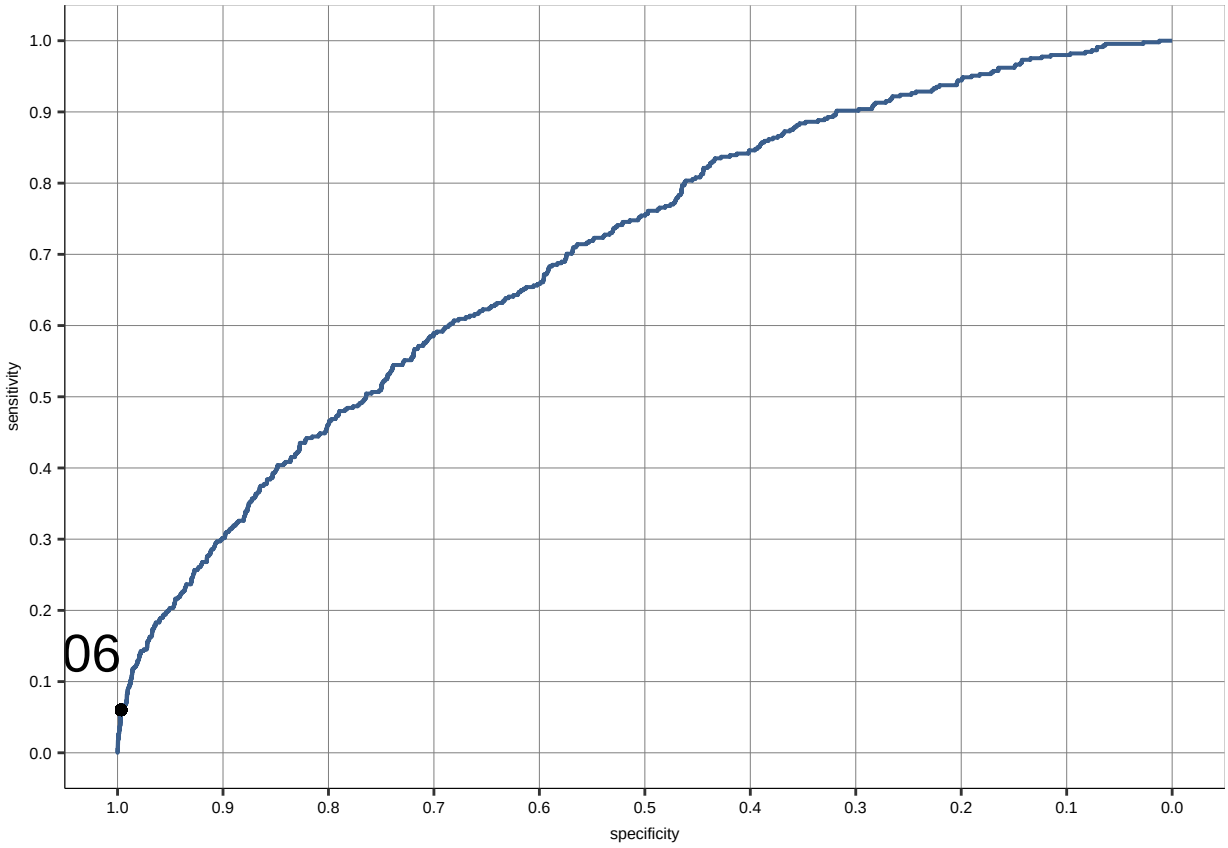
```
##   mtry splitrule min.node.size  Accuracy     Kappa      RMSE  AccuracySD
## 1    5      gini            10 0.8560644 0.1116853 0.3374041 0.002940982
## 2    5      gini            15 0.8559330 0.1123655 0.3371458 0.002271009
## 3    6      gini            10 0.8563271 0.1198670 0.3376120 0.002977224
## 4    6      gini            15 0.8556704 0.1137483 0.3375454 0.003187148
## 5    7      gini            10 0.8563270 0.1227686 0.3380188 0.003079031
## 6    7      gini            15 0.8560645 0.1181836 0.3377685 0.002667813
##      KappaSD      RMSESD
## 1 0.01128681 0.001769250
## 2 0.01208983 0.001404183
## 3 0.01325269 0.001613649
## 4 0.01853871 0.001586039
## 5 0.01857760 0.001560373
## 6 0.01448969 0.001637703
```

```
best_mtry <- rf_model_p$bestTune$mtry
best_min_node_size <- rf_model_p$bestTune$min.node.size

# Get average (ie over the folds) RMSE and AUC ------------------------------------
CV_RMSE_folds[["rf_p"]] <- rf_model_p$resample[,c("Resample", "RMSE")]

auc <- list()
for (fold in c("Fold1", "Fold2", "Fold3", "Fold4", "Fold5")) {
  cv_fold <-
    rf_model_p$pred %>%
```

```
    filter(Resample == fold)

  roc_obj <- roc(cv_fold$obs, cv_fold$fast_growth)
  auc[[fold]] <- as.numeric(roc_obj$auc)
}
CV_AUC_folds[["rf_p"]] <- data.frame("Resample" = names(auc),
                                     "AUC" = unlist(auc))

CV_RMSE[["rf_p"]] <- mean(CV_RMSE_folds[["rf_p"]]$RMSE)
CV_AUC[["rf_p"]] <- mean(CV_AUC_folds[["rf_p"]]$AUC)

# Now use loss function and search for best thresholds and expected loss over folds -----
best_tresholds_cv <- list()
expected_loss_cv <- list()

for (fold in c("Fold1", "Fold2", "Fold3", "Fold4", "Fold5")) {
  cv_fold <-
    rf_model_p$pred %>%
    filter(mtry == best_mtry,
           min.node.size == best_min_node_size,
           Resample == fold)

  roc_obj <- roc(cv_fold$obs, cv_fold$fast_growth)
  best_treshold <- coords(roc_obj, "best", ret="all", transpose = FALSE,
                          best.method="youden", best.weights=c(cost, prevelance))
  best_tresholds_cv[[fold]] <- best_treshold$threshold
  expected_loss_cv[[fold]] <- (best_treshold$fp*FP + best_treshold$fn*FN)/length(cv_fold$fast_growth)
}

# average
best_tresholds[["rf_p"]] <- mean(unlist(best_tresholds_cv))
expected_loss[["rf_p"]] <- mean(unlist(expected_loss_cv))


rf_summary <- data.frame("CV RMSE" = CV_RMSE[["rf_p"]],
                         "CV AUC" = CV_AUC[["rf_p"]],
                         "Avg of optimal thresholds" = best_tresholds[["rf_p"]],
                         "Threshold for Fold5" = best_treshold$threshold,
                         "Avg expected loss" = expected_loss[["rf_p"]],
                         "Expected loss for Fold5" = expected_loss_cv[[fold]])

kable(x = rf_summary, format = "latex", booktabs=TRUE,  digits = 3, row.names = TRUE,
      linesep = "", col.names = c("CV RMSE", "CV AUC",
                                  "Avg of optimal thresholds","Threshold for fold #5",
                                  "Avg expected loss","Expected loss for fold #5")) %>%
  cat(.,file= paste0(output, "rf_summary.tex"))

# Create plots - this is for Fold5

createLossPlot(roc_obj, best_treshold, "rf_p_loss_plot")
```




```
createRocPlotWithOptimal(roc_obj, best_treshold, "rf_p_roc_plot")
```




```
# Take model to holdout and estimate RMSE, AUC and expected loss ----------------------------------

rf_predicted_probabilities_holdout <- predict(rf_model_p, newdata = data_holdout, type = "prob")
data_holdout$rf_p_prediction <- rf_predicted_probabilities_holdout[,"fast_growth"]
RMSE(data_holdout$rf_p_prediction, data_holdout$fast_growth)
```

```
## [1] 0.3409247
```

```
# ROC curve on holdout
roc_obj_holdout <- roc(data_holdout$fast_growth, data_holdout[, "rf_p_prediction", drop=TRUE])

# AUC
as.numeric(roc_obj_holdout$auc)
```

```
## [1] 0.7013095
```

```
# Get expected loss on holdout with optimal threshold
holdout_treshold <- coords(roc_obj_holdout, x = best_tresholds[["rf_p"]] , input= "threshold",
                           ret="all", transpose = FALSE)
expected_loss_holdout <- (holdout_treshold$fp*FP + holdout_treshold$fn*FN)/length(data_holdout$fast_grow
expected_loss_holdout
```

```
## [1] 0.7346992
```

The random forest model was tuned using 5-fold cross-validation across combinations of `mtry` (number of variables randomly sampled at each split) and `min.node.size` (minimum size of terminal nodes). The best-performing configuration was `mtry = 6` and `min.node.size = 10`, which achieved the highest accuracy (around 85.63%) and the lowest RMSE (approximately 0.3371). The average cross-validated RMSE was 0.3409 and the average AUC was 0.7013, indicating solid but not exceptional predictive performance. To incorporate business relevance, an asymmetric loss function was applied where false positives cost more than false negatives. The best threshold minimizing this expected loss was around 0.71, as shown in the loss curve plot, and this threshold also appears on the ROC curve. When this model was applied to the holdout set, it yielded an RMSE of 0.3409, an AUC of 0.7347, and an expected loss of 0.7347, indicating that the model generalizes well to unseen data while balancing the asymmetric costs effectively.

Classification forest

Next, we move on to the classification random forest. The code fits a classification random forest model using 5-fold cross-validation to predict whether a firm is fast-growing, based on a predefined set of predictors (`rfvars`). The model is trained using the `ranger` method from the `caret` package, and the best hyperparameters are selected from a tuning grid. Once trained, the model generates class predictions (`fast_growth` or `no_fast_growth`) for both the training and holdout datasets. The final part of the code evaluates the model's performance on the holdout set using a custom loss function, where false positives (FP) and false negatives (FN) are penalized asymmetrically. It calculates the number of FP and FN predictions, weights them according to the predefined costs (`FP` and `FN`), and computes the average expected loss per observation.

```
train_control <- trainControl(
  method = "cv",
  n = 5
)
train_control$verboseIter <- TRUE

set.seed(13505)
rf_model_f <- train(
  formula(paste0("fast_growth_f ~ ", paste0(rfvars , collapse = " + "))),
  method = "ranger",
  data = data_train,
  tuneGrid = tune_grid,
  trControl = train_control
)
```

```
data_train$rf_f_prediction_class <-  predict(rf_model_f,type = "raw")
data_holdout$rf_f_prediction_class <- predict(rf_model_f, newdata = data_holdout, type = "raw")

#We use predicted classes to calculate expected loss based on our loss fn
fp <- sum(data_holdout$rf_f_prediction_class == "fast_growth" & data_holdout$fast_growth_f == "no_fast_g
fn <- sum(data_holdout$rf_f_prediction_class == "no_fast_growth" & data_holdout$fast_growth_f == "fast_g
(fp*FP + fn*FN)/length(data_holdout$fast_growth)
```

```
## [1] 0.7320725
```

```
holdout_prediction_rf <-
  ifelse(data_holdout$rf_p_prediction < holdout_treshold$threshold, "no_fast_growth", "fast_growth") %>%
  factor(levels = c("no_fast_growth", "fast_growth"))
cm_object4 <- confusionMatrix(holdout_prediction_rf,data_holdout$fast_growth_f)
cm4 <- cm_object4$table
cm4
```

```
##                 Reference
## Prediction       no_fast_growth fast_growth
##   no_fast_growth           3226         545
##   fast_growth                12          24
```

The output `0.7320725` represents the expected loss calculated from the classification random forest model on the holdout dataset, using the custom asymmetric loss function. This value summarizes the average cost per observation when false positives (predicting fast growth when there is none) and false negatives (missing a truly fast-growing firm) are weighted unequally. A lower expected loss indicates better performance under the specified cost structure, so this result suggests that the model performs reasonably well in balancing these types of misclassification errors, though it still incurs an average loss of about 0.73 per prediction.

The confusion matrix shows that the classification random forest model predicted 3,226 true negatives (correctly identified non-fast-growing firms) and 24 true positives (correctly identified fast-growing firms). However, it also produced 545 false negatives (missed fast-growing firms) and only 12 false positives (incorrectly flagged non-fast-growing firms as fast-growing). This suggests the model is highly conservative, favoring precision over recall, and struggles to capture the minority class of fast-growing firms.

Discussion of results

In the final section, we summarize and compare the performance of all models considered—three logistic regressions (X1, X4, and LASSO) and a random forest—based on cross-validated RMSE, AUC, optimal threshold, and expected loss under an asymmetric cost function. This allows for a structured evaluation of both predictive accuracy and the models' ability to minimize business-relevant misclassification costs. By juxtaposing statistical metrics with real-world implications, we assess how useful these models might be in practice for identifying fast-growing firms.

```
nvars[["rf_p"]] <- length(rfvars)

summary_results <- data.frame("Number of predictors" = unlist(nvars),
                              "CV RMSE" = unlist(CV_RMSE),
                              "CV AUC" = unlist(CV_AUC),
                              "CV threshold" = unlist(best_tresholds),
                              "CV expected Loss" = unlist(expected_loss))

model_names <- c("Logit X1", "Logit X4",
                 "Logit LASSO","RF probability")
summary_results <- summary_results %>%
  filter(rownames(.) %in% c("X1", "X4", "LASSO", "rf_p"))
rownames(summary_results) <- model_names
summary_results
```

```
##                 Number.of.predictors   CV.RMSE    CV.AUC CV.threshold
## Logit X1                          11 0.3474623 0.6427885          Inf
## Logit X4                          79 0.3381581 0.7100193    0.6000039
## Logit LASSO                      103 0.3378407 0.6843364    0.5285608
## RF probability                    24 0.3376120 0.7122648    0.5661137
##                 CV.expected.Loss
## Logit X1               0.7326152
## Logit X4               0.7184953
## Logit LASSO            0.7270593
## RF probability         0.7112080
```

```
kable(x = summary_results, format = "latex", booktabs=TRUE,  digits = 3, row.names = TRUE,
      linesep = "", col.names = c("Number of predictors", "CV RMSE", "CV AUC",
                                  "CV threshold", "CV expected Loss")) %>%
  cat(.,file= paste0(output, "summary_results.tex"))
```

Looking at the results, the random forest probability model stands out with the highest AUC (0.712) and one of the lowest expected losses (0.711), outperforming both the logistic baseline model (X1) and LASSO in discrimination ability. Model X4 also performs very competitively, achieving nearly the same AUC (0.710) and the lowest expected loss (0.718), despite having more predictors than the random forest. Interestingly, LASSO, while including the most predictors (103), does not yield a clear advantage in terms of AUC or expected loss. This suggests that model complexity alone does not guarantee better performance, and that careful variable selection and regularization matter. The baseline X1 model, with only 11 predictors, performs worst across all metrics, confirming the value of richer feature sets for this task.

However, despite decent predictive performance, the models remain limited in practical reliability. Prediction is inherently difficult in this context: the target variable (fast growth) is rare and noisy, and trade-offs between false positives and false negatives must be considered carefully. While random forests and penalized regressions help improve classification, no model delivers perfect or even near-perfect discrimination. These tools may be helpful for narrowing down a pool of promising firms, but they are not sufficient on their own for high-stakes decision-making. Users should interpret results with caution and complement model predictions with expert judgment or additional screening criteria. Ultimately, prediction is useful—but it is far from infallible.