EDA_and_modelling_5oct

Michael Wang

2025-10-10

```r
library(tidyverse)
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.4     v readr     2.1.5
## v forcats   1.0.0     v stringr   1.5.2
## v ggplot2   4.0.0     v tibble    3.3.0
## v lubridate 1.9.4     v tidyr     1.3.1
## v purrr     1.1.0
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

```r
library(glmnet)
```

```
## Loading required package: Matrix
##
## Attaching package: 'Matrix'
##
## The following objects are masked from 'package:tidyr':
##
##     expand, pack, unpack
##
## Loaded glmnet 4.1-10
```

```r
library(randomForest)
```

```
## randomForest 4.7-1.2
## Type rfNews() to see new features/changes/bug fixes.
##
## Attaching package: 'randomForest'
##
## The following object is masked from 'package:dplyr':
##
##     combine
##
## The following object is masked from 'package:ggplot2':
##
##     margin
```

```r
library(xgboost)
```

```
##
## Attaching package: 'xgboost'
##
## The following object is masked from 'package:dplyr':
##
##     slice
```

```r
library(MASS)
```

```
##
## Attaching package: 'MASS'
##
## The following object is masked from 'package:dplyr':
##
##     select
```

```r
library(caret)
```

```
## Loading required package: lattice
##
## Attaching package: 'caret'
##
## The following object is masked from 'package:purrr':
##
##     lift
```

```r
library(pROC)
```

```
## Type 'citation("pROC")' for a citation.
##
## Attaching package: 'pROC'
##
## The following objects are masked from 'package:stats':
##
##     cov, smooth, var
```

```r
select <- dplyr::select

setwd('/Users/michaelwang/Desktop/unsw/unsw\ actl/4305/Assignment')

df <- read_csv('/Users/michaelwang/Desktop/unsw/unsw actl/4305/Assignment/actl4305-ggins-datathon/standa
```

```
## Rows: 70000 Columns: 77
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr   (7): destinations, traveller_ages, platform, discount, convert, quote_...
## dbl  (45): quote_price, extra_cancellation, quote_hour, trip_length, lead_le...
## lgl   (5): has_child_U12, has_teen_O13, has_adult_O18, has_senior_O65, is_fa...
```

```
## dttm  (1): quote_create_time
## date (19): trip_start_date, trip_end_date, boost_1_start_date, boost_1_end_d...
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```r
crime_indices <- read_csv('/Users/michaelwang/Desktop/unsw/unsw actl/4305/Assignment/cleaned_crime_indi
```

```
## Rows: 70000 Columns: 2
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (1): crime_index
## dbl (1): quote_id
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```r
df_trim <- df %>%
  select(-destinations, -traveller_ages, -quote_create_time, -matches("^boost_.*_date$"))

df_clean <- df_trim %>%
  mutate(
    across(starts_with("boost_"), ~ if_else(is.na(.x), 0, .x)),
    quote_hour = if_else(quote_hour > 23, 24, quote_hour)
    )

any(is.na(df_clean))
```

```
## [1] FALSE
```

```r
df_discount <- df_clean %>%
  mutate(
    discount = as.numeric(str_extract(discount, "^[0-9]*"))
  )


character_colnames <- names(df_discount)[sapply(df_discount, is.character)]

df_factors <- df_discount %>%
  mutate(
    across(all_of(character_colnames), ~ as.factor(.x))
  )

df_crime_scores <- df_factors %>%
  mutate(quote_id = row_number()) %>%
  left_join(crime_indices, by = "quote_id") %>%
  select(-quote_id)

df_crime_scores <- df_crime_scores %>%
  rowwise() %>%
  mutate(
    median_crime_index = median(as.numeric(unlist(strsplit(crime_index, ", "))))
```

```r
  ) %>%
  ungroup %>%
  select(-crime_index)

df_clean_quote_hour <- df_crime_scores %>%
  mutate(quote_hour = if_else(quote_hour > 23, 24, quote_hour))

df_extra_cancellation <- df_clean_quote_hour %>%
  mutate(
    extra_cancel_tier = case_when(
      extra_cancellation == 0 ~ 0,
      extra_cancellation <= 5000 ~ 1,
      extra_cancellation > 5000 ~ 2,
      TRUE ~ 1
    )
  ) %>%
    select(-extra_cancellation)

df_final <- df_extra_cancellation %>%
  select(-quote_time)


df_eda <- df_final


df_large_quote_price <- df_eda[df_eda$quote_price > 5000,]
df_large_lengths <- df_eda[df_eda$trip_length > 365,]

df_capped <- df_eda

cap_trip_len <- quantile(df_capped$trip_length, 0.99)
df_capped$trip_length_capped <- pmin(df_capped$trip_length, cap_trip_len)

cap_price <- quantile(df_capped$quote_price, 0.99)
df_capped$quote_price_capped <- pmin(df_capped$quote_price, cap_price)

cap_lead_len <- quantile(df_capped$lead_length, 0.99)
df_capped$lead_length_capped <- pmin(df_capped$lead_length, cap_lead_len)

boost_cols_1_2 <- paste0("boost_", 1:2, "_length")
for (col in boost_cols_1_2) {
  cap_val <- quantile(df_capped[[col]], 0.99)
  df_capped[[col]] <- pmin(df_capped[[col]], cap_val)
}

boost_cols_3_4_5 <- paste0("boost_", 3:5, "_length")
for (col in boost_cols_3_4_5) {
  cap_val <- quantile(df_capped[[col]], 0.999)
  df_capped[[col]] <- pmin(df_capped[[col]], cap_val)
}

boost_cols_6_7_8 <- paste0("boost_", 6:8, "_length")
for (col in boost_cols_6_7_8) {
```

```r
  cap_val <- quantile(df_capped[[col]], 0.999)
  df_capped[[col]] <- pmin(df_capped[[col]], cap_val)
}

cap_age_range <- quantile(df_capped$age_range, 0.99)
df_capped$age_range_capped <- pmin(df_capped$age_range, cap_age_range)

df_capped <- df_capped %>%
  select(-trip_length, -quote_price, -lead_length, -age_range)

df_correlation <- df_capped %>%
  select(-median_age, -min_age, -max_age)


df_correlation <- df_correlation %>%
  select(-generations, -age_range_capped, -boost_4_length, -boost_5_length, -boost_6_length, -boost_7_le

df_pretransform_modelling <- df_correlation

df_transformations <- df_correlation

box_cox_cols <- c(
  "mean_age",
  "trip_length_capped",
  "quote_price_capped",
  "lead_length_capped",
  paste0("boost_", 1:3, "_length")
)

small <- 1e-6
for (col in box_cox_cols) {
  zero_flag <- min(df_transformations[[col]])
  if (zero_flag <= 0) {
    df_transformations[[col]] <- df_transformations[[col]] + small
  }
}

X <- as.data.frame(df_transformations[, box_cox_cols])

preprocess_boxcox <- preProcess(X, method = "BoxCox")
X_boxcox  <- predict(preprocess_boxcox, X)

df_transformations[paste0(names(X_boxcox), "_boxcox")] <- X_boxcox

z_cols <- paste0(box_cox_cols, "_boxcox")
X <- as.data.frame(df_transformations[, z_cols])
preprocess_z <- preProcess(X, method = c("center", "scale"))
X_z <- predict(preprocess_z, X)

df_transformations[paste0(box_cox_cols, "_boxcox_z")] <- X_z

df_transformations <- df_transformations %>%
  select(-ends_with("_boxcox"), -all_of(box_cox_cols))
```

```r
df_transformations_1 <- df_transformations

remaining_z_cols <- c(
  "median_crime_index",
  "num_adults",
  "boost_num",
  "num_travellers",
  "discount"
)

small <- 1e-6
for (col in remaining_z_cols) {
  zero_flag <- min(df_transformations_1[[col]])
  if (zero_flag <= 0) {
    df_transformations_1[[col]] <- df_transformations_1[[col]] + small
  }
}

X_z_df <- as.data.frame(df_transformations_1[, remaining_z_cols])
preprocess_z <- preProcess(X_z_df, method = c("center", "scale"))
X_z <- predict(preprocess_z, X_z_df)
df_transformations_1[paste0(remaining_z_cols, "_z")] <- X_z

df_transformations_1 <- df_transformations_1 %>%
  select(-all_of(remaining_z_cols))


df_modelling <- df_transformations_1
```

No imb data proc

Train/Test/Val

```r
# model.matrix converts to numeric matrix to run regression, [, -1] removes convert column
X <- model.matrix(convert ~ ., df_modelling)[, -1]
y <- ifelse(df_modelling$convert == "YES", 1, 0)

set.seed(111)
# regular sampling:
# train_index <- sample(seq_len(nrow(X)), 0.8 * nrow(X))

# stratified sampling to make sure test and train have equal proportions of converted and non-converted
train_index <- createDataPartition(y, p = 0.8, list = FALSE)
# 0.25 of 0.8 is 0.2 of total

X_train <- X[train_index,]
y_train <- y[train_index]

validation_index <- createDataPartition(y_train, p = 0.25, list = FALSE)
X_val <- X_train[validation_index,]
```

```r
y_val <- y_train[validation_index]
X_train_inner <- X_train[-validation_index, ]
y_train_inner <- y_train[-validation_index]

X_test <- X[-train_index,]
y_test <- y[-train_index]

# length(y_train_inner) / length(y)
# length(y_val) / length(y)
# length(y_test) / length(y)

Train_Data <- df_modelling[train_index,]
validation_index_df <- createDataPartition(Train_Data$convert, p = 0.25, list = FALSE)
Validation_Data <- Train_Data[validation_index_df, ]
Train_Data_inner<- Train_Data[-validation_index_df, ]
Test_Data <- df_modelling[-train_index,]
y_test_glm <- ifelse(Test_Data$convert == "YES", 1, 0)
y_validation_glm <- ifelse(Validation_Data$convert == "YES", 1, 0)
```

GLM/Shrinkage

```r
glm_basic <- glm(convert ~ ., data = Train_Data, family = binomial)
# summary(glm_basic)
```

```r
# ridge (alpha = 0), lasso (alpha = 1), elastic Net (0 < alpha < 1)
# then check best one. At the same time, also runs default 10 fold cv on training data to pick best lam
# hyperparameter for shrinkage to control degree of penalty applied.
cv_lasso <- cv.glmnet(X_train, y_train, family = "binomial", alpha = 1)
cv_ridge <- cv.glmnet(X_train, y_train, family = "binomial", alpha = 0)
cv_enet <- cv.glmnet(X_train, y_train, family = "binomial", alpha = 0.5)
```

```r
# # cv_lasso, etc. are cv.glmnet objects - running coef on them returns a sparse coeff matrix with one
# # fitted coeffecients, with intercept in first row. This is with the best lambda, stored in the cv.gl
coef_lasso <- coef(cv_lasso, s = "lambda.min")
# coef_ridge <- coef(cv_ridge, s = "lambda.min")
# coef_enet <- coef(cv_enet, s = "lambda.min")

# # this extracts that column of coeffecients, keeps only the non 0 ones, and keeps it as a matrix (dro
# prevent dropping into a vector); then returns top 20.
lasso_nonzero <- coef_lasso[coef_lasso[,1] != 0, , drop = FALSE]
sort(lasso_nonzero[,1], decreasing = TRUE)[1:20]
```

```
##                   (Intercept)                    Antarctica
##                    51.9998897                     0.7423665
##                    discount_z    trip_length_capped_boxcox_z
##                     0.6093428                     0.6016991
##              has_child_U12TRUE                 South_America
##                     0.5882245                     0.5609581
##                 North_America       group_typesingle_old_O60
##                     0.4973767                     0.4836544
```

```
## group_typeolder_parents_family                           Africa
##                       0.4073433                        0.3744716
##                     boost_num_z                      gadget_cover
##                       0.3408810                        0.3398086
##                      snowsports                      Central_Asia
##                       0.3185885                        0.3177019
##                     Middle_East       group_typesingle_middle_U60
##                       0.2988412                        0.2614825
##     group_typesingle_young_U30                   has_teen_013TRUE
##                       0.2526183                        0.2312086
##                motorcycle_cover                   Domestic_Cruise
##                       0.2247859                        0.2224792
```

```
# ridge_nonzero <- coef_ridge[coef_ridge[,1] != 0, , drop = FALSE]
# sort(ridge_nonzero[,1], decreasing = TRUE)[1:20]

# enet_nonzero <- coef_enet[coef_enet[,1] != 0, , drop = FALSE]
# sort(enet_nonzero[,1], decreasing = TRUE)[1:20]

# # cv error vs log(lambda)
# plot(cv_lasso)
# plot(cv_ridge)
# plot(cv_enet)
```

```
# type = response means to give probabilities for logistic reg instead of values for normal regression
# predict() as a universal apply data to model base r function

# idea of using dataframe for glm prediction and matrix for shrinkage prediction i think is because of
# the glm model saw the original data, so it was the test data in the same form to help process. The sh
# saw the dataframe only a matrix rep, so thus it wants it in the same format.

pred_glm <- predict(glm_basic, newdata = Test_Data, type = "response")
pred_lasso <- predict(cv_lasso, newx = X_test, type = "response", s = "lambda.min")
pred_ridge <- predict(cv_ridge, newx = X_test, type = "response", s = "lambda.min")
pred_enet <- predict(cv_enet, newx = X_test, type = "response", s = "lambda.min")

glm_class <- ifelse(pred_glm > 0.5, 1, 0)
lasso_class <- ifelse(pred_lasso > 0.5, 1, 0)
ridge_class <- ifelse(pred_ridge > 0.5, 1, 0)
enet_class <- ifelse(pred_enet > 0.5, 1, 0)

acc_glm <- mean(glm_class == y_test_glm)
acc_lasso <- mean(lasso_class == y_test)
acc_ridge <- mean(ridge_class == y_test)
acc_enet <- mean(enet_class == y_test)

c(glm = acc_glm, Lasso = acc_lasso, Ridge = acc_ridge, ElasticNet = acc_enet)
```

```
##       glm     Lasso     Ridge ElasticNet
## 0.8824286 0.8822857 0.8806429  0.8823571
```

```r
calculate_metrics <- function(predictions, actual) {
  tp <- sum(predictions == 1 & actual == 1)
  tn <- sum(predictions == 0 & actual == 0)
  fp <- sum(predictions == 1 & actual == 0)
  fn <- sum(predictions == 0 & actual == 1)

  accuracy <- (tp + tn) / (tp + tn + fp + fn)
  precision <- tp / (tp + fp)
  recall <- tp / (tp + fn)
  sensitivity <- recall
  specificity <- tn / (tn + fp)
  f1 <- 2 * precision * recall / (precision + recall)

  return(c(
    Accuracy = accuracy,
    Precision = precision,
    Recall = recall,
    Specificity = specificity,
    F1_Score = f1
  ))
}

metrics_glm <- calculate_metrics(glm_class, y_test_glm)
metrics_lasso <- calculate_metrics(lasso_class, y_test)
metrics_ridge <- calculate_metrics(ridge_class, y_test)
metrics_enet  <- calculate_metrics(enet_class, y_test)

comparison_table <- data.frame(
  GLM = metrics_glm,
  Lasso = metrics_lasso,
  Ridge = metrics_ridge,
  ElasticNet = metrics_enet
)

print(round(comparison_table, 4))
```

```
##                GLM  Lasso  Ridge ElasticNet
## Accuracy    0.8824 0.8823 0.8806     0.8824
## Precision   0.6159 0.6147 0.6238     0.6161
## Recall      0.1550 0.1533 0.1110     0.1533
## Specificity 0.9862 0.9863 0.9905     0.9864
## F1_Score    0.2477 0.2454 0.1884     0.2455
```

```r
library(pROC)
roc_glm <- roc(y_test_glm, as.vector(pred_glm))
```

```
## Setting levels: control = 0, case = 1
```

```
## Setting direction: controls < cases
```

```r
roc_lasso <- roc(y_test, as.vector(pred_lasso))
```

```
## Setting levels: control = 0, case = 1
## Setting direction: controls < cases
```

```r
roc_ridge <- roc(y_test, as.vector(pred_ridge))
```

```
## Setting levels: control = 0, case = 1
## Setting direction: controls < cases
```

```r
roc_enet <- roc(y_test, as.vector(pred_enet))
```

```
## Setting levels: control = 0, case = 1
## Setting direction: controls < cases
```

```r
auc(roc_glm)
```

```
## Area under the curve: 0.8047
```

```r
auc(roc_lasso)
```

```
## Area under the curve: 0.8046
```

```r
auc(roc_ridge)
```

```
## Area under the curve: 0.8011
```

```r
auc(roc_enet)
```

```
## Area under the curve: 0.8045
```

LASSO for more interpretability (feature selection)

ENET for slightly more stable coeffecients

HORRENDOUS F1 SCORE when not treating any imbalanced data stuff.

RF

```r
df_tree_modelling <- df_eda %>%
  select(-median_age, -min_age, -max_age, -generations, -age_range)
```

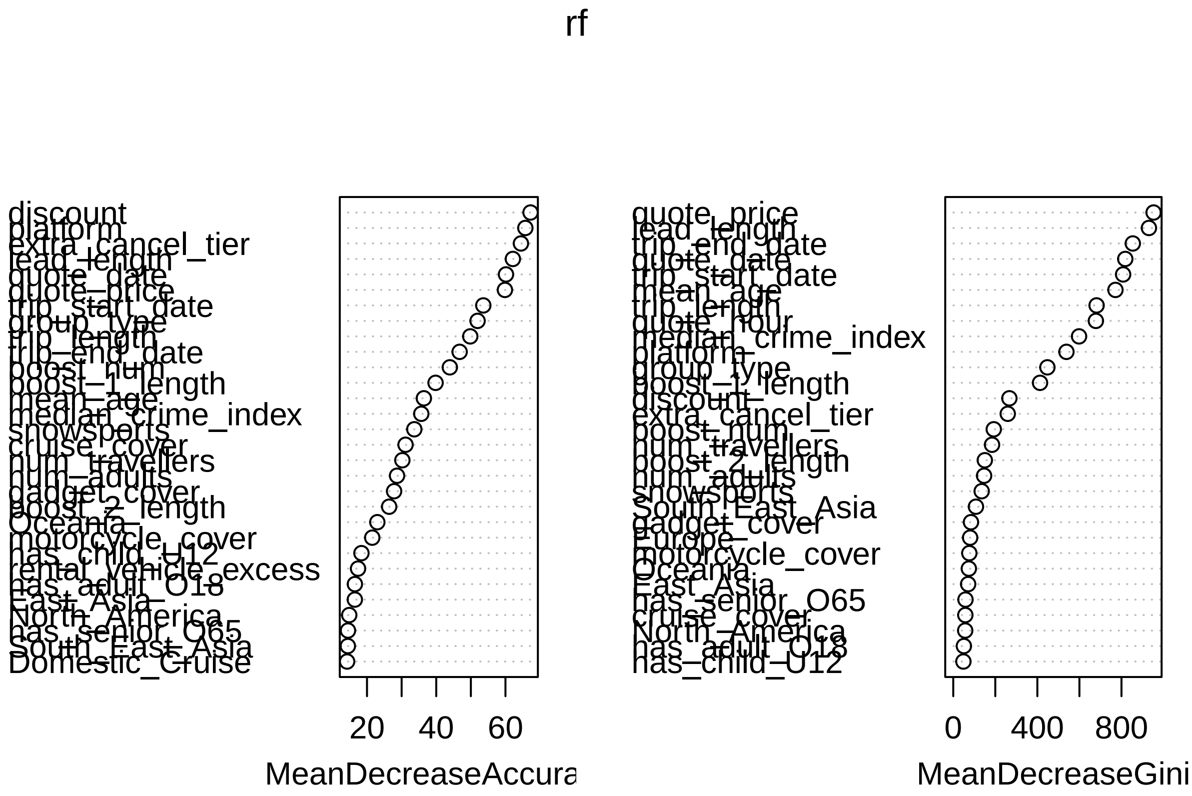
```r
set.seed(111)
df_tree_modelling$convert <- factor(df_tree_modelling$convert, , levels = c("NO","YES"))

y_tree <- df_tree_modelling$convert
train_index <- createDataPartition(y_tree, p = 0.8, list = FALSE)

y_train_tree <- y[train_index]
validation_index <- createDataPartition(y_train_tree, p = 0.25, list = FALSE)

tree_train <- df_tree_modelling[train_index, ]
tree_train_inner <- df_tree_modelling[-validation_index, ]
tree_val <- tree_train[validation_index, ]
tree_test <- df_tree_modelling[-train_index, ]
y_val_tree <- as.numeric(ifelse(tree_val$convert == "YES", 1, 0))
y_test_tree <- as.numeric(ifelse(tree_test$convert == "YES", 1, 0))

set.seed(222)
rf <- randomForest(
  convert ~ .,
  data = tree_train,
  ntree = 500,
  mtry = floor(sqrt(ncol(tree_train) - 1)),
  importance = TRUE
)

# type = prob returns probabilities instead of class labels. This is for several reasons,
# first is calculating metrics like ROC or just probability in general, and second for
# later decision threshold tuning. [, "YES"] means we select the YES probability column.
pred_test <- predict(rf, tree_test, type = "prob")[, "YES"]
pred_class <- ifelse(pred_test >= 0.5, 1, 0)
rf_metrics <- calculate_metrics(pred_class, y_test_tree)

# RUN THESE ONLY
rf_metrics
```

```
##    Accuracy   Precision      Recall Specificity    F1_Score
##   0.8703479   0.4836601   0.1233333   0.9805722   0.1965471
```

```r
print(rf)
```

```
##
## Call:
##  randomForest(formula = convert ~ ., data = tree_train, ntree = 500,      mtry = floor(sqrt(ncol(tre
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 7
##
##         OOB estimate of  error rate: 12.86%
## Confusion matrix:
##        NO YES class.error
## NO  47906 894  0.01831967
## YES  6309 892  0.87612832
```

```
varImpPlot(rf)
```



Imb data procs

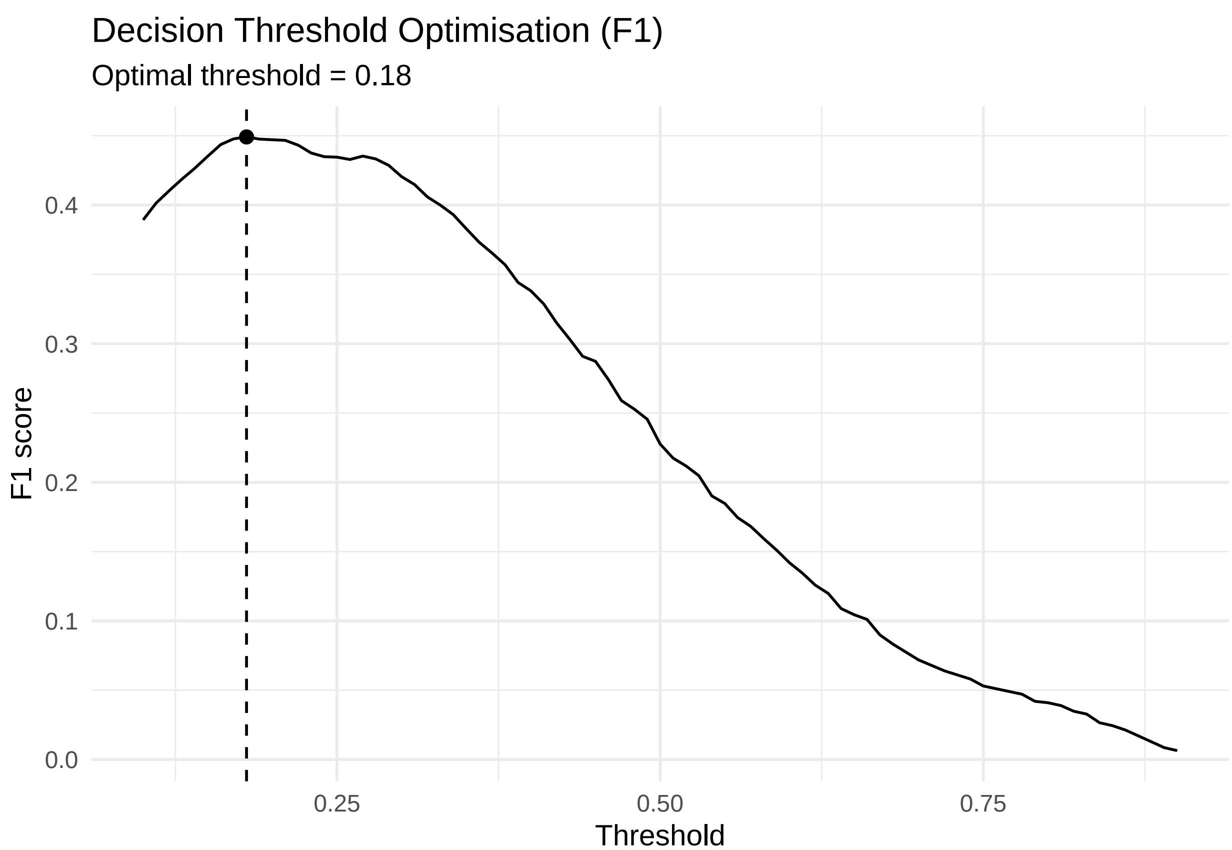
GLM DTO

```
optimise_threshold <- function(predictions, actual, metric = "f1") {
  thresholds <- seq(0.1, 0.9, by = 0.01)
  results <- data.frame(
    threshold = thresholds,
    accuracy = NA,
    precision = NA,
    recall = NA,
    f1 = NA,
    specificity = NA
  )

  # quick logic for this loop: predictions is a numeric vector of probabilities. Use an if else to chan
  # into class labels 1 or 0 according to the new threshold. Scan across all the thresholds, calculate
  # neg, false pos and false neg. Then use these to calculate the diff metrics to eventually get to F1

  # At the end, we have a results data frame. Append each metric into the result data frame, such that
```

```
  # the result data frame is a list of the metrics for each threshold level.

  for (i in seq_along(thresholds)) {
    threshold <- thresholds[i]
    pred_class <- ifelse(predictions > threshold, 1, 0)

    tp <- sum(pred_class == 1 & actual == 1)
    tn <- sum(pred_class == 0 & actual == 0)
    fp <- sum(pred_class == 1 & actual == 0)
    fn <- sum(pred_class == 0 & actual == 1)

    results$accuracy[i] <- (tp + tn) / (tp + tn + fp + fn)
    results$precision[i] <- tp / (tp + fp)
    results$recall[i] <- tp / (tp + fn)
    results$f1[i] <- 2 * results$precision[i] * results$recall[i] / (results$precision[i] + results$reca
    results$specificity[i] <- tn / (tn + fp)
  }

  # which picks out the index of the list which is the highest. We can decide which metric to use. Here
  # to F1 since its the best indicator overall esp for this imbalanced data.
  # then we extract the threshold that corresponds do it, and return everything.

  optimal_index <- which.max(results[[metric]])
  optimal_threshold <- results$threshold[optimal_index]

  return(list(
    optimal_threshold = optimal_threshold,
    all_results = results,
    best_metrics = results[optimal_index, ]
  ))
}
```

```
glm_dt_opt <- glm(convert ~ ., data = Train_Data_inner, family = binomial)
cv_lasso_dt_opt <- cv.glmnet(X_train_inner, y_train_inner,
                             family = "binomial", alpha = 1)
cv_ridge_dt_opt <- cv.glmnet(X_train_inner, y_train_inner,
                             family = "binomial", alpha = 0)
cv_enet_dt_opt <- cv.glmnet(X_train_inner, y_train_inner,
                            family = "binomial", alpha = 0.5)

pred_glm_dt_opt_val <- predict(glm_dt_opt, newdata = Validation_Data, type = "response")
pred_lasso_dt_opt_val <- predict(cv_lasso_dt_opt, newx = X_val, type = "response",
                       s = "lambda.min")
pred_ridge_dt_opt_val <- predict(cv_ridge_dt_opt, newx = X_val, type = "response",
                       s = "lambda.min")
pred_enet_dt_opt_val <- predict(cv_enet_dt_opt,  newx = X_val, type = "response",
                      s = "lambda.min")

opt_glm_dt_opt <- optimise_threshold(pred_glm_dt_opt_val, y_validation_glm, metric = "f1")
opt_lasso_dt_opt <- optimise_threshold(pred_lasso_dt_opt_val, y_val, metric = "f1")
opt_ridge_dt_opt <- optimise_threshold(pred_ridge_dt_opt_val, y_val, metric = "f1")
opt_enet_dt_opt <- optimise_threshold(pred_enet_dt_opt_val, y_val, metric = "f1")
```

```r
res_lasso_dt_opt <- opt_lasso_dt_opt$all_results

ggplot(res_lasso_dt_opt, aes(x = threshold, y = f1)) +
  geom_line() +
  geom_vline(xintercept = opt_lasso_dt_opt$optimal_threshold, linetype = 2) +
  geom_point(data = subset(res_lasso_dt_opt,
                           threshold == opt_lasso_dt_opt$optimal_threshold),
             aes(x = threshold, y = f1), size = 2) +
  labs(title = "Decision Threshold Optimisation (F1)",
       subtitle = paste("Optimal threshold =",
                        round(opt_lasso_dt_opt$optimal_threshold, 3)),
       x = "Threshold", y = "F1 score") +
  theme_minimal()
```




```r
# ValidatioN Results After Tuning Decision Threshold
res_glm_dt_opt <- opt_glm_dt_opt$all_results
res_ridge_dt_opt <- opt_ridge_dt_opt$all_results
res_enet_dt_opt <- opt_enet_dt_opt$all_results

res_glm_dt_opt[res_glm_dt_opt$threshold == opt_glm_dt_opt$optimal_threshold,]
```

```
##    threshold  accuracy precision    recall        f1 specificity
## 11       0.2 0.8109421 0.3517266 0.5446527 0.4274281   0.8505785
```

```r
res_lasso_dt_opt[res_lasso_dt_opt$threshold == opt_lasso_dt_opt$optimal_threshold,]
```

```
##   threshold  accuracy precision    recall        f1 specificity
## 9      0.18 0.7970714 0.3510155 0.6232508 0.4490983   0.8236699
```

```r
res_ridge_dt_opt[res_ridge_dt_opt$threshold == opt_ridge_dt_opt$optimal_threshold,]
```

```
##   threshold  accuracy precision    recall        f1 specificity
## 8      0.17 0.7864286 0.3406532 0.6512379 0.4473198   0.8071158
```

```r
res_enet_dt_opt[res_enet_dt_opt$threshold == opt_enet_dt_opt$optimal_threshold,]
```

```
##   threshold  accuracy precision    recall        f1 specificity
## 9      0.18 0.7976429 0.3518687 0.6232508 0.4497961   0.8243288
```

```r
# Model Eval Using Test Set
pred_glm_dt_opt <- predict(glm_dt_opt, newdata = Test_Data, type = "response")
pred_lasso_dt_opt <- predict(cv_lasso_dt_opt, newx = X_test, type = "response",
                      s = "lambda.min")
pred_ridge_dt_opt <- predict(cv_ridge_dt_opt, newx = X_test, type = "response",
                      s = "lambda.min")
pred_enet_dt_opt <- predict(cv_enet_dt_opt,  newx = X_test, type = "response",
                     s = "lambda.min")

glm_class_dt_opt <- ifelse(pred_glm_dt_opt > opt_glm_dt_opt$optimal_threshold, 1, 0)
lasso_class_dt_opt <- ifelse(pred_lasso_dt_opt > opt_lasso_dt_opt$optimal_threshold,
                             1, 0)
ridge_class_dt_opt <- ifelse(pred_ridge_dt_opt > opt_ridge_dt_opt$optimal_threshold,
                             1, 0)
enet_class_dt_opt <- ifelse(pred_enet_dt_opt  > opt_enet_dt_opt$optimal_threshold, 1, 0)

metrics_glm_dt_opt <- calculate_metrics(glm_class_dt_opt, y_test_glm)
metrics_lasso_dt_opt <- calculate_metrics(lasso_class_dt_opt, y_test)
metrics_ridge_dt_opt <- calculate_metrics(ridge_class_dt_opt, y_test)
metrics_enet_dt_opt <- calculate_metrics(enet_class_dt_opt,  y_test)

comparison_table <- data.frame(
  GLM = metrics_glm_dt_opt,
  Lasso = metrics_lasso_dt_opt,
  Ridge = metrics_ridge_dt_opt,
  ElasticNet = metrics_enet_dt_opt
)
comparison_table
```

```
##                   GLM     Lasso     Ridge ElasticNet
## Accuracy    0.8196429 0.8014286 0.7905714  0.8012857
## Precision   0.3536723 0.3320312 0.3194021  0.3317057
## Recall      0.5371854 0.5835240 0.5989703  0.5829519
## Specificity 0.8599412 0.8325171 0.8179073  0.8324355
## F1_Score    0.4265274 0.4232365 0.4166335  0.4228216
```

RF DTO

```r
set.seed(222)
rf_dto <- randomForest(
  convert ~ .,
  data = tree_train_inner,
  ntree = 500,
  mtry  = floor(sqrt(ncol(tree_train_inner) - 1)),
  importance = TRUE
)

pred_val <- predict(rf_dto, tree_val, type = "prob")[, "YES"]
opt_rf_dto <- optimise_threshold(pred_val, y_val_tree, metric = "f1")
res_opt_rf_dto <- opt_rf_dto$all_results
cat("Validation Results")
```

```
## Validation Results
```

```r
res_opt_rf_dto[res_opt_rf_dto$threshold == opt_rf_dto$optimal_threshold,]
```

```
##    threshold accuracy precision    recall        f1 specificity
## 29      0.38 0.971502 0.9266169 0.8413326 0.8819177   0.9903516
```

```r
pred_test_dto <- predict(rf, tree_test, type = "prob")[, "YES"]
rf_class_dt_opt <- ifelse(pred_test_dto > opt_rf_dto$optimal_threshold, 1, 0)
metrics_rf_dto <- calculate_metrics(rf_class_dt_opt, y_test_tree)
metrics_rf_dto
```

```
##    Accuracy   Precision      Recall Specificity    F1_Score
##   0.8659904   0.4656420   0.2861111   0.9515534   0.3544391
```

U/D Sampling

glmnet / cv.glmnet (logistic) - for family="binomial", glmnet expects a binary outcome. It accepts either numeric 0/1, or a 2-level factor (internally converted to 0/1 with the first level as 0). Turning it into numeric makes it simple and less error prone (0/1 encoding)

randomForest - in rf, the task is inferred from the type of y. if y is factor, it is classification forest (type="prob" gives class probs as mentioned earlier). if y is numeric, we will get a regression forest (continuous outputs; no type="prob").

Thus pass numeric into glm but factor into

```r
X_train_matrix <- as.data.frame(X_train)
y_train_fac <- droplevels(as.factor(y_train))

set.seed(123)
train_up <- upSample(x = X_train_matrix, y = y_train_fac, yname = "convert")
X_train_up <- data.matrix(subset(train_up, select = -convert))
y_train_up <- as.numeric(as.character(train_up$convert))
```

```r
set.seed(123)
train_down <- downSample(x = X_train_matrix, y = y_train_fac, yname = "convert")
X_train_down <- data.matrix(subset(train_down, select = -convert))
y_train_down <- as.numeric(as.character(train_down$convert))

Train_Data_up <- Train_Data
Train_Data_up$convert <- factor(Train_Data_up$convert, levels = c("NO","YES"))

up_train <- upSample(x = subset(Train_Data_up, select = -convert),
                     y = Train_Data_up$convert, yname = "convert")

Train_Data_down <- Train_Data
Train_Data_down$convert <- factor(Train_Data_down$convert, levels = c("NO","YES"))

down_train <- upSample(x = subset(Train_Data_down, select = -convert),
                     y = Train_Data_down$convert, yname = "convert")
# prop table is just table but in proportions
# prop.table(table(y_train))
# prop.table(table(y_train_up))
# prop.table(table(y_train_down))

X_train_inner_matrix <- as.data.frame(X_train_inner)
y_train_inner_fac <- droplevels(as.factor(y_train_inner))

set.seed(123)
train_up_inner <- upSample(x = X_train_inner_matrix, y = y_train_inner_fac, yname = "convert")
X_train_inner_up <- data.matrix(subset(train_up_inner, select = -convert))
y_train_inner_up <- as.numeric(as.character(train_up_inner$convert))

set.seed(123)
train_down_inner <- downSample(x = X_train_inner_matrix, y = y_train_inner_fac, yname = "convert")
X_train_inner_down <- data.matrix(subset(train_down_inner, select = -convert))
y_train_inner_down <- as.numeric(as.character(train_down_inner$convert))

Train_Data_inner_up <- Train_Data_inner
Train_Data_inner_up$convert <- factor(Train_Data_inner_up$convert, levels = c("NO","YES"))

up_train_inner <- upSample(x = subset(Train_Data_inner_up, select = -convert),
                     y = Train_Data_inner_up$convert, yname = "convert")

Train_Data_inner_down <- Train_Data_inner
Train_Data_inner_down$convert <- factor(Train_Data_inner_down$convert, levels = c("NO","YES"))

down_train_inner <- upSample(x = subset(Train_Data_inner_down, select = -convert),
                     y = Train_Data_inner_down$convert, yname = "convert")
```

```r
# set seed is a random number SEQUENCE setter - use it again before each random act to make sure each r
set.seed(111)
glm_up <- glm(convert ~ ., data = up_train, family = binomial)
cv_lasso_up  <- cv.glmnet(X_train_up, y_train_up, family = "binomial", alpha = 1)
cv_ridge_up  <- cv.glmnet(X_train_up, y_train_up, family = "binomial", alpha = 0)
cv_enet_up   <- cv.glmnet(X_train_up, y_train_up, family = "binomial", alpha = 0.5)
```

```r
set.seed(111)
glm_down <- glm(convert ~ ., data = down_train, family = binomial)
cv_lasso_down <- cv.glmnet(X_train_down, y_train_down, family = "binomial", alpha = 1)
cv_ridge_down <- cv.glmnet(X_train_down, y_train_down, family = "binomial", alpha = 0)
cv_enet_down <- cv.glmnet(X_train_down, y_train_down, family = "binomial", alpha = 0.5)
```

```r
pred_glm_up <- predict(glm_up, newdata = Test_Data, type = "response")
pred_lasso_up <- predict(cv_lasso_up, newx = X_test, type = "response", s = "lambda.min")
pred_ridge_up <- predict(cv_ridge_up, newx = X_test, type = "response", s = "lambda.min")
pred_enet_up  <- predict(cv_enet_up,  newx = X_test, type = "response", s = "lambda.min")

glm_class_up <- ifelse(pred_glm_up > 0.5, 1, 0)
lasso_class_up <- ifelse(pred_lasso_up > 0.5, 1, 0)
ridge_class_up <- ifelse(pred_ridge_up > 0.5, 1, 0)
enet_class_up  <- ifelse(pred_enet_up  > 0.5, 1, 0)

metrics_glm_up <- calculate_metrics(glm_class_up, y_test_glm)
metrics_lasso_up <- calculate_metrics(lasso_class_up, y_test)
metrics_ridge_up <- calculate_metrics(ridge_class_up, y_test)
metrics_enet_up  <- calculate_metrics(enet_class_up,  y_test)

comparison_table <- data.frame(
  GLM = metrics_glm_up,
  Lasso = metrics_lasso_up,
  Ridge = metrics_ridge_up,
  ElasticNet = metrics_enet_up
)

print(round(comparison_table, 4))
```

```
##                GLM  Lasso  Ridge ElasticNet
## Accuracy    0.7341 0.7356 0.7231     0.7356
## Precision   0.2824 0.2837 0.2731     0.2837
## Recall      0.7328 0.7328 0.7328     0.7328
## Specificity 0.7343 0.7360 0.7217     0.7360
## F1_Score    0.4077 0.4091 0.3979     0.4091
```

```r
pred_glm_down <- predict(glm_down, newdata = Test_Data, type = "response")
pred_lasso_down <- predict(cv_lasso_down, newx = X_test, type = "response",
                           s = "lambda.min")
pred_ridge_down <- predict(cv_ridge_down, newx = X_test, type = "response",
                           s = "lambda.min")
pred_enet_down <- predict(cv_enet_down, newx = X_test, type = "response",
                          s = "lambda.min")

glm_class_down <- ifelse(pred_glm_down > 0.5, 1, 0)
lasso_class_down <- ifelse(pred_lasso_down > 0.5, 1, 0)
ridge_class_down <- ifelse(pred_ridge_down > 0.5, 1, 0)
enet_class_down <- ifelse(pred_enet_down  > 0.5, 1, 0)

metrics_glm_down <- calculate_metrics(glm_class_down, y_test_glm)
metrics_lasso_down <- calculate_metrics(lasso_class_down, y_test)
```

```r
metrics_ridge_down <- calculate_metrics(ridge_class_down, y_test)
metrics_enet_down <- calculate_metrics(enet_class_down,  y_test)

comparison_table <- data.frame(
  GLM = metrics_glm_down,
  Lasso = metrics_lasso_down,
  Ridge = metrics_ridge_down,
  ElasticNet = metrics_enet_down
)
print(round(comparison_table, 4))
```

```
##                GLM  Lasso  Ridge ElasticNet
## Accuracy    0.7355 0.7314 0.7179     0.7314
## Precision   0.2831 0.2807 0.2710     0.2808
## Recall      0.7300 0.7368 0.7454     0.7374
## Specificity 0.7363 0.7306 0.7139     0.7306
## F1_Score    0.4080 0.4065 0.3975     0.4068
```

U/D Samp+DTO

```r
# attempt for u/d sampling with earlier optimal decision threshold
# up:
glm_class_up <- ifelse(pred_glm_up > opt_glm_dt_opt$optimal_threshold, 1, 0)
lasso_class_up <- ifelse(pred_lasso_up > opt_lasso_dt_opt$optimal_threshold, 1, 0)
ridge_class_up <- ifelse(pred_ridge_up > opt_ridge_dt_opt$optimal_threshold, 1, 0)
enet_class_up <- ifelse(pred_enet_up > opt_enet_dt_opt$optimal_threshold, 1, 0)

metrics_glm_up <- calculate_metrics(glm_class_up, y_test_glm)
metrics_lasso_up <- calculate_metrics(lasso_class_up, y_test)
metrics_ridge_up <- calculate_metrics(ridge_class_up, y_test)
metrics_enet_up <- calculate_metrics(enet_class_up, y_test)

comparison_table <- data.frame(
  GLM = metrics_glm_up,
  Lasso = metrics_lasso_up,
  Ridge = metrics_ridge_up,
  ElasticNet = metrics_enet_up
)

print(round(comparison_table, 4))
```

```
##                GLM  Lasso  Ridge ElasticNet
## Accuracy    0.4081 0.3845 0.3203     0.3843
## Precision   0.1692 0.1648 0.1530     0.1648
## Recall      0.9565 0.9662 0.9794     0.9662
## Specificity 0.3299 0.3015 0.2262     0.3013
## F1_Score    0.2875 0.2816 0.2646     0.2815
```

```r
glm_class_down <- ifelse(pred_glm_down > opt_glm_dt_opt$optimal_threshold, 1, 0)
lasso_class_down <- ifelse(pred_lasso_down > opt_lasso_dt_opt$optimal_threshold, 1, 0)
```

```r
ridge_class_down <- ifelse(pred_ridge_down > opt_ridge_dt_opt$optimal_threshold, 1, 0)
enet_class_down <- ifelse(pred_enet_down > opt_enet_dt_opt$optimal_threshold, 1, 0)

metrics_glm_down <- calculate_metrics(glm_class_down, y_test_glm)
metrics_lasso_down <- calculate_metrics(lasso_class_down, y_test)
metrics_ridge_down <- calculate_metrics(ridge_class_down, y_test)
metrics_enet_down <- calculate_metrics(enet_class_down, y_test)

comparison_table <- data.frame(
  GLM = metrics_glm_down,
  Lasso = metrics_lasso_down,
  Ridge = metrics_ridge_down,
  ElasticNet = metrics_enet_down
)
print(round(comparison_table, 4))
```

```
##                GLM  Lasso  Ridge ElasticNet
## Accuracy    0.4061 0.3872 0.3210     0.3872
## Precision   0.1687 0.1654 0.1532     0.1654
## Recall      0.9559 0.9657 0.9805     0.9657
## Specificity 0.3277 0.3047 0.2269     0.3047
## F1_Score    0.2867 0.2824 0.2650     0.2824
```

```r
# Actual optimisation:
set.seed(111)
glm_up_inner <- glm(convert ~ ., data = up_train_inner, family = binomial)
cv_lasso_up_inner <- cv.glmnet(X_train_inner_up, y_train_inner_up,
                         family = "binomial", alpha = 1)
cv_ridge_up_inner <- cv.glmnet(X_train_inner_up, y_train_inner_up,
                         family = "binomial", alpha = 0)
cv_enet_up_inner <- cv.glmnet(X_train_inner_up, y_train_inner_up,
                         family = "binomial", alpha = 0.5)

set.seed(111)
glm_down_inner <- glm(convert ~ ., data = down_train_inner, family = binomial)
cv_lasso_down_inner <- cv.glmnet(X_train_inner_down, y_train_inner_down,
                                family = "binomial", alpha = 1)
cv_ridge_down_inner <- cv.glmnet(X_train_inner_down, y_train_inner_down,
                                family = "binomial", alpha = 0)
cv_enet_down_inner <- cv.glmnet(X_train_inner_down, y_train_inner_down,
                               family = "binomial", alpha = 0.5)
```

```r
pred_glm_up_inner <- predict(glm_up_inner, newdata = Validation_Data, type = "response")
pred_lasso_up_inner <- predict(cv_lasso_up_inner, newx = X_val, type = "response", s = "lambda.min")
pred_ridge_up_inner <- predict(cv_ridge_up_inner, newx = X_val, type = "response", s = "lambda.min")
pred_enet_up_inner <- predict(cv_enet_up_inner,  newx = X_val, type = "response", s = "lambda.min")

pred_glm_down_inner <- predict(glm_down_inner, newdata = Validation_Data,
                              type = "response")
pred_lasso_down_inner <- predict(cv_lasso_down_inner, newx = X_val, type = "response", s = "lambda.min"
pred_ridge_down_inner <- predict(cv_ridge_down_inner, newx = X_val, type = "response", s = "lambda.min"
pred_enet_down_inner <- predict(cv_enet_down_inner,  newx = X_val, type = "response", s = "lambda.min")
```

```r
# up
opt_glm_up_inner <- optimise_threshold(pred_glm_up_inner,
                                       y_validation_glm, metric = "f1")
opt_lasso_up_inner <- optimise_threshold(pred_lasso_up_inner, y_val, metric = "f1")
opt_ridge_up_inner <- optimise_threshold(pred_ridge_up_inner, y_val, metric = "f1")
opt_enet_up_inner <- optimise_threshold(pred_enet_up_inner, y_val, metric = "f1")

res_glm_up_inner <- opt_glm_up_inner$all_results
res_lasso_up_inner <- opt_lasso_up_inner$all_results
res_ridge_up_inner <- opt_ridge_up_inner$all_results
res_enet_up_inner <- opt_enet_up_inner$all_results

res_glm_up_inner[res_glm_up_inner$threshold == opt_glm_up_inner$optimal_threshold,]
```

```
##    threshold accuracy precision    recall        f1 specificity
## 51       0.6 0.797943 0.3390422 0.5893054 0.4304409   0.8289981
```

```r
res_lasso_up_inner[res_lasso_up_inner$threshold == opt_lasso_up_inner$optimal_threshold,]
```

```
##    threshold  accuracy precision    recall        f1 specificity
## 51       0.6 0.7979286 0.3504774 0.6124865 0.4458374   0.8263054
```

```r
res_ridge_up_inner[res_ridge_up_inner$threshold == opt_ridge_up_inner$optimal_threshold,]
```

```
##    threshold  accuracy precision    recall        f1 specificity
## 49      0.58 0.7879286   0.33959 0.6329386 0.4420222   0.8116455
```

```r
res_enet_up_inner[res_enet_up_inner$threshold == opt_enet_up_inner$optimal_threshold,]
```

```
##    threshold  accuracy precision    recall        f1 specificity
## 51       0.6 0.7977857 0.3502616 0.6124865 0.4456628   0.8261407
```

```r
# down
opt_glm_down_inner <- optimise_threshold(pred_glm_down_inner,
                                         y_validation_glm, metric = "f1")
opt_lasso_down_inner <- optimise_threshold(pred_lasso_down_inner, y_val, metric = "f1")
opt_ridge_down_inner <- optimise_threshold(pred_ridge_down_inner, y_val, metric = "f1")
opt_enet_down_inner <- optimise_threshold(pred_enet_down_inner, y_val, metric = "f1")

res_glm_down_inner <- opt_glm_down_inner$all_results
res_lasso_down_inner <- opt_lasso_down_inner$all_results
res_ridge_down_inner <- opt_ridge_down_inner$all_results
res_enet_down_inner <- opt_enet_down_inner$all_results
res_glm_down_inner[res_glm_down_inner$threshold == opt_glm_down_inner$optimal_threshold,]
```

```
##    threshold  accuracy precision    recall        f1 specificity
## 50      0.59 0.7910864 0.3321245 0.6058434 0.4290455   0.8186592
```

```r
res_lasso_down_inner[res_lasso_down_inner$threshold ==
                       opt_lasso_down_inner$optimal_threshold,]
```

```
##    threshold  accuracy precision    recall        f1 specificity
## 53      0.62 0.8046429 0.3563708 0.5855759 0.4430869    0.838165
```

```r
res_ridge_down_inner[res_ridge_down_inner$threshold ==
                       opt_ridge_down_inner$optimal_threshold,]
```

```
##    threshold  accuracy precision    recall        f1 specificity
## 51       0.6 0.7992857 0.3500315 0.5979548 0.4415739   0.8300939
```

```r
res_enet_down_inner[res_enet_down_inner$threshold ==
                      opt_enet_down_inner$optimal_threshold,]
```

```
##    threshold  accuracy precision    recall       f1 specificity
## 53      0.62 0.8057857 0.3580613 0.5844995 0.444081   0.8396475
```

```r
# test set
# up
pred_glm_up_inner <- predict(glm_up_inner, newdata = Test_Data, type = "response")
pred_lasso_up_inner <- predict(cv_lasso_up_inner, newx = X_test, type = "response",
                      s = "lambda.min")
pred_ridge_up_inner <- predict(cv_ridge_up_inner, newx = X_test, type = "response",
                      s = "lambda.min")
pred_enet_up_inner <- predict(cv_enet_up_inner,  newx = X_test, type = "response",
                     s = "lambda.min")

glm_class_up_inner <- ifelse(pred_glm_up_inner >
                               opt_glm_up_inner$optimal_threshold, 1, 0)
lasso_class_up_inner <- ifelse(pred_lasso_up_inner >
                                 opt_lasso_up_inner$optimal_threshold, 1, 0)
ridge_class_up_inner <- ifelse(pred_ridge_up_inner >
                                 opt_ridge_up_inner$optimal_threshold, 1, 0)
enet_class_up_inner  <- ifelse(pred_enet_up_inner  >
                                 opt_enet_up_inner$optimal_threshold, 1, 0)

metrics_glm_up_inner <- calculate_metrics(glm_class_up_inner, y_test_glm)
metrics_lasso_up_inner <- calculate_metrics(lasso_class_up_inner, y_test)
metrics_ridge_up_inner <- calculate_metrics(ridge_class_up_inner, y_test)
metrics_enet_up_inner <- calculate_metrics(enet_class_up_inner, y_test)

comparison_table_up_inner <- data.frame(
  GLM = metrics_glm_up_inner,
  Lasso = metrics_lasso_up_inner,
  Ridge = metrics_ridge_up_inner,
  ElasticNet = metrics_enet_up_inner
)
comparison_table_up_inner
```

```
##                    GLM     Lasso     Ridge ElasticNet
```

```
## Accuracy    0.8038571 0.8016429 0.7951429  0.8015000
## Precision   0.3328868 0.3286713 0.3234552  0.3282239
## Recall      0.5686499 0.5646453 0.5869565  0.5635011
## Specificity 0.8374143 0.8354554 0.8248449  0.8354554
## F1_Score    0.4199409 0.4154915 0.4170732  0.4148242
```

```r
# down

pred_glm_down_inner <- predict(glm_down_inner, newdata = Test_Data, type = "response")
pred_lasso_down_inner <- predict(cv_lasso_down_inner, newx = X_test, type = "response",
                      s = "lambda.min")
pred_ridge_down_inner <- predict(cv_ridge_down_inner, newx = X_test, type = "response",
                      s = "lambda.min")
pred_enet_down_inner <- predict(cv_enet_down_inner,  newx = X_test, type = "response",
                     s = "lambda.min")

glm_class_down_inner <- ifelse(pred_glm_down_inner >
                                 opt_glm_down_inner$optimal_threshold, 1, 0)
lasso_class_down_inner <- ifelse(pred_lasso_down_inner >
                                   opt_lasso_down_inner$optimal_threshold, 1, 0)
ridge_class_down_inner <- ifelse(pred_ridge_down_inner >
                                   opt_ridge_down_inner$optimal_threshold, 1, 0)
enet_class_down_inner  <- ifelse(pred_enet_down_inner >
                                   opt_enet_down_inner$optimal_threshold, 1, 0)

glm_class_up_inner <- ifelse(pred_glm_up_inner > 0.5, 1, 0)
lasso_class_up_inner <- ifelse(pred_lasso_up_inner > 0.5, 1, 0)
ridge_class_up_inner <- ifelse(pred_ridge_up_inner > 0.5, 1, 0)
enet_class_up_inner  <- ifelse(pred_enet_up_inner > 0.5, 1, 0)

metrics_glm_down_inner <- calculate_metrics(glm_class_down_inner, y_test_glm)
metrics_lasso_down_inner <- calculate_metrics(lasso_class_down_inner, y_test)
metrics_ridge_down_inner <- calculate_metrics(ridge_class_down_inner, y_test)
metrics_enet_down_inner <- calculate_metrics(enet_class_down_inner, y_test)

comparison_table_down_inner <- data.frame(
  GLM = metrics_glm_down_inner,
  Lasso = metrics_lasso_down_inner,
  Ridge = metrics_ridge_down_inner,
  ElasticNet = metrics_enet_down_inner
)
comparison_table_down_inner
```

```
##                   GLM     Lasso     Ridge ElasticNet
## Accuracy    0.7984286 0.8090000 0.8040714  0.8095714
## Precision   0.3277742 0.3366267 0.3301468  0.3368870
## Recall      0.5846682 0.5457666 0.5532037  0.5423341
## Specificity 0.8289259 0.8465557 0.8398629  0.8476983
## F1_Score    0.4200575 0.4164120 0.4135129  0.4156072
```

seems like overall performace is better without up or down sampling maybe? need to think about what metrics we really want to be using.

Next Steps - tuning, feature selection, weights, analysis of error and residuals

To do: - tune hyper parameters - do better feature selection —> hybrid/forward/backward subset feature selection - try the weights 4 to 1 thing for lasso/enet after optimising decision boundary - check if under or overfitting by comparing pure train and test error

```
cv_lasso <- cv.glmnet(X_train, y_train, family = "binomial", alpha = 1)
train_preds <- predict(cv_lasso, newx = X_train, s = "lambda.min", type = "response")
test_preds <- predict(cv_lasso, newx = X_test, s = "lambda.min", type = "response")

opt_threshold <- opt_lasso_dt_opt$optimal_threshold
train_class <- ifelse(train_preds > opt_threshold, 1, 0)
test_class <- ifelse(test_preds > opt_threshold, 1, 0)

train_error <- mean(train_class != y_train)
test_error <- mean(test_class != y_test)

cat("Training error:", round(train_error, 3), "\n")
```

```
## Training error: 0.203
```

```
cat("Test error:", round(test_error, 3))
```

```
## Test error: 0.198
```

underfitting? maybe a discuss for next week.

```
library(MASS)
step_model <- stepAIC(glm_basic, direction = "both", trace = FALSE)
summary(step_model)
```

```
##
## Call:
## glm(formula = convert ~ trip_start_date + trip_end_date + platform +
##     quote_date + quote_hour + has_child_U12 + has_teen_O13 +
##     has_adult_O18 + has_senior_O65 + group_type + Africa + Europe +
##     Oceania + Multi_Region + Domestic_Cruise + North_America +
##     South_America + Middle_East + South_East_Asia + Antarctica +
##     snowsports + cruise_cover + medical_conditions + gadget_cover +
##     motorcycle_cover + rental_vehicle_excess + specified_items +
##     extra_cancel_tier + trip_length_capped_boxcox_z + quote_price_capped_boxcox_z +
##     lead_length_capped_boxcox_z + boost_2_length_boxcox_z + boost_3_length_boxcox_z +
##     boost_num_z + num_travellers_z + discount_z, family = binomial,
##     data = Train_Data)
##
## Coefficients:
##                                  Estimate Std. Error z value Pr(>|z|)
## (Intercept)                    51.9896700 11.5899906   4.486 7.27e-06 ***
```

```
## trip_start_date                      0.0058859  0.0007968   7.387 1.50e-13 ***
## trip_end_date                       -0.0044913  0.0007177  -6.258 3.91e-10 ***
## platformqw                          -1.1431971  0.0461346 -24.780  < 2e-16 ***
## platformweb                         -1.4623446  0.0433361 -33.744  < 2e-16 ***
## quote_date                          -0.0040665  0.0006786  -5.993 2.07e-09 ***
## quote_hour                           0.0060232  0.0027875   2.161 0.030712 *
## has_child_U12TRUE                    0.7192795  0.1258277   5.716 1.09e-08 ***
## has_teen_O13TRUE                     0.2685638  0.0858783   3.127 0.001764 **
## has_adult_O18TRUE                   -0.1346085  0.0848409  -1.587 0.112604
## has_senior_O65TRUE                   0.2385898  0.0853812   2.794 0.005200 **
## group_typemiddle_couple_U60          0.5968600  0.2594158   2.301 0.021404 *
## group_typemiddle_peer_group_U60     -0.0556439  0.3646781  -0.153 0.878727
## group_typemixed_generation_family    0.4519563  0.2561232   1.765 0.077630 .
## group_typeold_couple_O60             0.7063987  0.2553325   2.767 0.005665 **
## group_typeolder_parents_family       0.6935275  0.2675870   2.592 0.009548 **
## group_typeolder_peer_group_O60       0.2846445  0.3720303   0.765 0.444205
## group_typeparents_plus_kids         -0.2237712  0.2167544  -1.032 0.301898
## group_typeparents_plus_teens         0.3364114  0.2614978   1.286 0.198276
## group_typesingle_middle_U60          1.0065086  0.2830151   3.556 0.000376 ***
## group_typesingle_old_O60             1.2623870  0.2832342   4.457 8.31e-06 ***
## group_typesingle_young_U30           0.9427224  0.2834049   3.326 0.000880 ***
## group_typeyoung_couple_U30           0.4600432  0.2660326   1.729 0.083760 .
## group_typeyoung_peer_group_U30       0.2382936  0.3011587   0.791 0.428795
## Africa                               0.3932084  0.1027016   3.829 0.000129 ***
## Europe                               0.0669381  0.0461674   1.450 0.147086
## Oceania                             -0.4490016  0.0530016  -8.471  < 2e-16 ***
## Multi_Region                        -0.3200958  0.1051790  -3.043 0.002340 **
## Domestic_Cruise                      0.2243940  0.0681832   3.291 0.000998 ***
## North_America                        0.4952722  0.0616095   8.039 9.07e-16 ***
## South_America                        0.5507066  0.1259625   4.372 1.23e-05 ***
## Middle_East                          0.2967711  0.1054927   2.813 0.004905 **
## South_East_Asia                      0.1241377  0.0408088   3.042 0.002351 **
## Antarctica                           0.7703325  0.4041080   1.906 0.056617 .
## snowsports                           0.3022167  0.0650045   4.649 3.33e-06 ***
## cruise_cover                        -0.5082200  0.0710317  -7.155 8.38e-13 ***
## medical_conditions                  -2.0665083  0.4760769  -4.341 1.42e-05 ***
## gadget_cover                         0.3344531  0.0716914   4.665 3.08e-06 ***
## motorcycle_cover                     0.2142588  0.0807262   2.654 0.007951 **
## rental_vehicle_excess               -0.5450064  0.1063316  -5.126 2.97e-07 ***
## specified_items                     -0.6935199  0.1520318  -4.562 5.07e-06 ***
## extra_cancel_tier                   -1.9299735  0.0656929 -29.379  < 2e-16 ***
## trip_length_capped_boxcox_z          0.6393268  0.0422400  15.136  < 2e-16 ***
## quote_price_capped_boxcox_z         -1.1246251  0.0402617 -27.933  < 2e-16 ***
## lead_length_capped_boxcox_z         -0.5365335  0.0209170 -25.651  < 2e-16 ***
## boost_2_length_boxcox_z              0.1153383  0.0188556   6.117 9.54e-10 ***
## boost_3_length_boxcox_z             -0.0338023  0.0155822  -2.169 0.030061 *
## boost_num_z                          0.3914406  0.0362338  10.803  < 2e-16 ***
## num_travellers_z                     0.1060001  0.0405087   2.617 0.008878 **
## discount_z                           0.6260933  0.0259373  24.139  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
```

```
##     Null deviance: 43173  on 55999  degrees of freedom
## Residual deviance: 34509  on 55950  degrees of freedom
## AIC: 34609
##
## Number of Fisher Scoring iterations: 6
```

```r
formula(step_model)
```

```
## convert ~ trip_start_date + trip_end_date + platform + quote_date +
##     quote_hour + has_child_U12 + has_teen_O13 + has_adult_O18 +
##     has_senior_O65 + group_type + Africa + Europe + Oceania +
##     Multi_Region + Domestic_Cruise + North_America + South_America +
##     Middle_East + South_East_Asia + Antarctica + snowsports +
##     cruise_cover + medical_conditions + gadget_cover + motorcycle_cover +
##     rental_vehicle_excess + specified_items + extra_cancel_tier +
##     trip_length_capped_boxcox_z + quote_price_capped_boxcox_z +
##     lead_length_capped_boxcox_z + boost_2_length_boxcox_z + boost_3_length_boxcox_z +
##     boost_num_z + num_travellers_z + discount_z
```

- stratified samplied didn't do much, the original sampling was already relatively well split between train and test for the target variable.
- tested model with more significant predictors, didn't change model too much
- tested using weights in cv.glmnet, worse overall than decision threshold optimisation

To Do:

- loads of testing with different models, with different params, before and after box cox or z scaling
- loads of testing to do with hyperparams