Project Part 1

Virginia Brame, Clay Harris, Hai Liu

2025-02-25

```
knitr::opts_chunk$set(echo=TRUE)
knitr::opts_chunk$set(cache=TRUE, autodep=TRUE)
knitr::opts_chunk$set(fig.align="center", fig.pos="H")
```

Data (loading, wrangling, EDA)

```
library(doParallel)
cl <- makePSOCKcluster(parallel::detectCores(logical = FALSE))
registerDoParallel(cl)
```

```
library(tidyverse)
library(tidymodels)
library(discrim)
library(leaflet)
library(terra)
library(htmlwidgets)
library(leafem)
library(colordistance)
library(jpeg)
library(patchwork)
library(probably)
library(gridExtra)
library(plotly)
library(mapview)
```

Data loading and wrangling

Since we are only interested in the level of "Blue Tarp", I create a new variable `BT` with only two classes, i.e., "TRUE" for "Blue Tarp" and "FALSE" for everything else.

```
col_names <- c('ID','X','Y','Map X','Map Y','Lat','Lon','Red','Green','Blue')

blue_files <- c(
  "orthovnir069_ROI_Blue_Tarps.txt",
  "orthovnir067_ROI_Blue_Tarps.txt",
  "orthovnir078_ROI_Blue_Tarps.txt"
)

non_blue_files <- c(
  "orthovnir057_ROI_NON_Blue_Tarps.txt",
  "orthovnir078_ROI_NON_Blue_Tarps.txt",
  "orthovnir067_ROI_NOT_Blue_Tarps.txt",
  "orthovnir069_ROI_NOT_Blue_Tarps.txt"
```

```
)

blue_data <- map_dfr(blue_files, ~
  read_table(.x, comment = ";", col_names = col_names) %>%
    select(Lat, Lon, Red, Green, Blue) %>%
    mutate(BT = "TRUE")
)

non_blue_data <- map_dfr(non_blue_files, ~
  read_table(.x, comment = ";", col_names = col_names) %>%
    select(Lat, Lon, Red, Green, Blue) %>%
    mutate(BT = "FALSE")
)

holdout_data <- bind_rows(blue_data, non_blue_data) %>%
  mutate(BT = factor(BT, levels = c("TRUE", "FALSE")))
```

```
train_data <- read_csv("HaitiPixels.csv") %>%
  mutate(BT = factor(if_else(Class == "Blue Tarp", "TRUE", "FALSE"), levels = c("TRUE", "FALSE"))) %>%
  select(Red, Green, Blue, BT)
```

EDA

```
# Convert
holdout_data_sp <- holdout_data %>%
  rename(x = Lon, y = Lat)
v <- terra::vect(holdout_data_sp, geom = c("x", "y"), crs = "EPSG:4326")

# Reproject to UTM (meters)
v_utm <- terra::project(v, "EPSG:32618")

# Create an empty raster
r_empty <- terra::rast(terra::ext(v_utm), resolution = 0.5, crs = "EPSG:32618")

# Rasterize
r_b1 <- terra::rasterize(v_utm, r_empty, field = "Red", overwrite = TRUE)
r_b2 <- terra::rasterize(v_utm, r_empty, field = "Green", overwrite = TRUE)
r_b3 <- terra::rasterize(v_utm, r_empty, field = "Blue", overwrite = TRUE)

# Combine
rgb_raster <- c(r_b1, r_b2, r_b3)

# Reproject
rgb_raster_wgs <- terra::project(rgb_raster, "EPSG:4326", overwrite = TRUE)

# Convert to brick for leaflet
rgb_brick <- raster::brick(rgb_raster_wgs)
```

```
# Create map
m <- leaflet(options = leafletOptions(maxZoom = 25)) %>%
  addTiles(options = tileOptions(maxZoom = 25)) %>%
  leafem::addRasterRGB(rgb_brick, r = 1, g = 2, b = 3)

if (knitr::is_html_output()) {
```

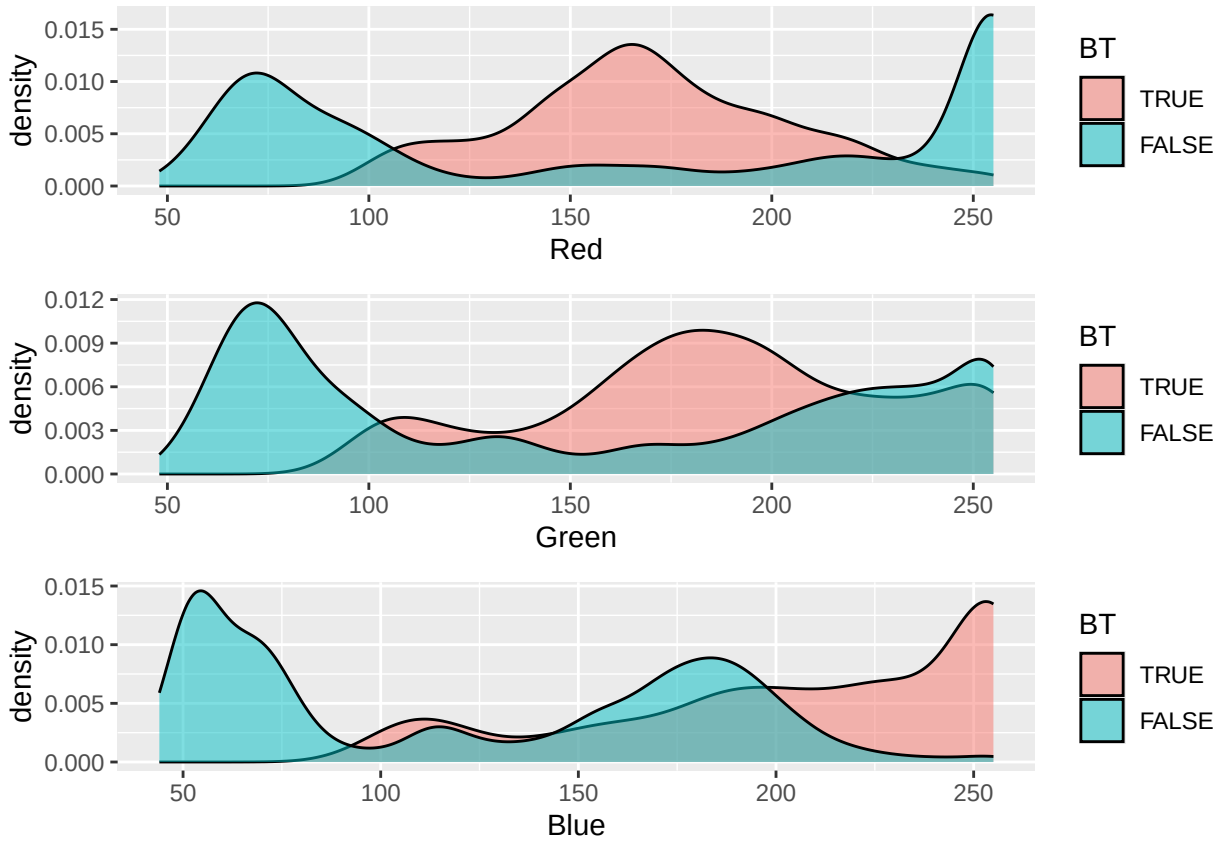
```
  htmlwidgets::saveWidget(m, "interactive_map.html")
  htmltools::includeHTML("interactive_map.html")
} else {
  # Save a static image for PDF output
  mapshot(m, file = "map_static.png")
  knitr::include_graphics("map_static.png")
}
```


re-Dame-
mmaculée-
nception
Route Nationale # 2
Mer Mapoe
Mariani
Route des Rails
Rue Christophe Chanel
Nan Couco
Lamatin 54
Route Nationale # 2
RN 2
Clocher
Brochette 99
Brochette
Morne Boyer
Rue Cayimite
Dufour
Degand
Morne Chandelle
Vieux Caille
Saint-
Rivière Froide
Treförend
Angibeau
Kafou
Morisset
Berot

```r
# Identify non-null pixels
non_na_mask <- !is.na(r_b1) | !is.na(r_b2) | !is.na(r_b3)

# Count cells
non_na_cells <- sum(terra::values(non_na_mask), na.rm = TRUE)

# Area of one pixel
cell_area_km2 <- (terra::res(r_empty)[1] * terra::res(r_empty)[2]) / 1e6

# Total area
total_area_km2 <- non_na_cells * cell_area_km2

# Print the result
total_area_km2
```

```
## [1] 0.01344958
```

```r
rgb1 <- train_data %>%
  ggplot(aes(x=Red, fill=BT))+
  geom_density(alpha = 0.5)
rgb2 <- train_data %>%
  ggplot(aes(x=Green, fill=BT))+
  geom_density(alpha = 0.5)
rgb3 <- train_data %>%
  ggplot(aes(x=Blue, fill=BT))+
  geom_density(alpha = 0.5)

grid.arrange(rgb1, rgb2, rgb3)
```



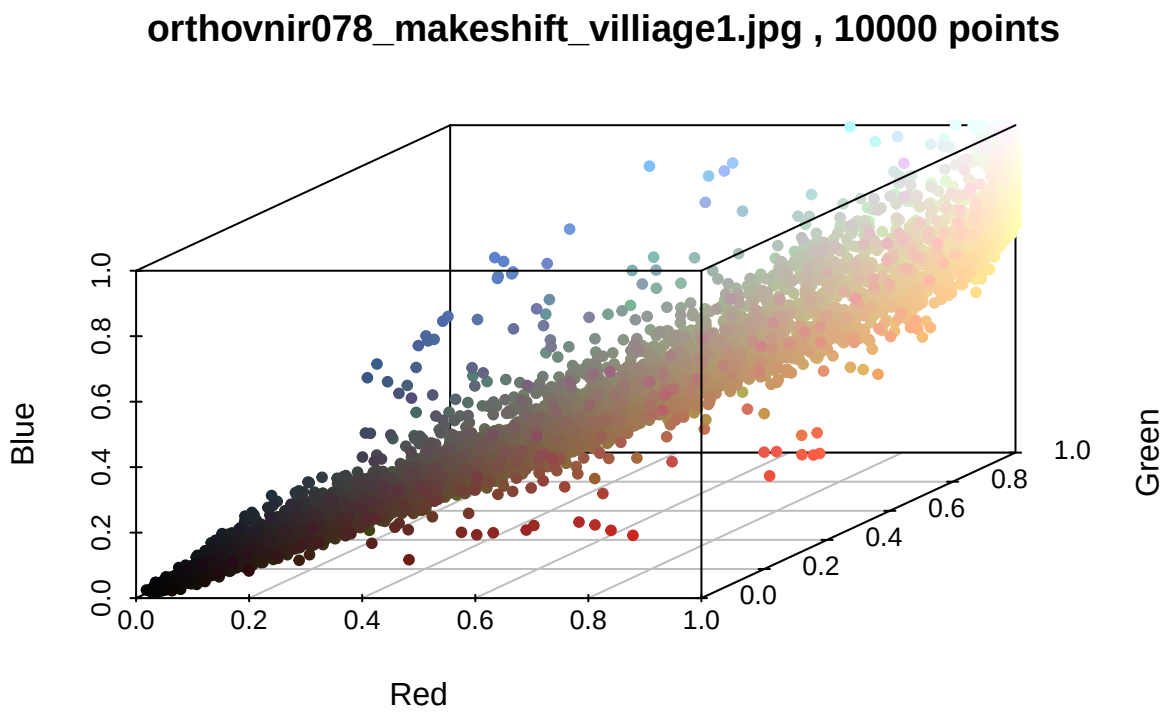
Understanding that: - Blue is 0,0,255 - Green is 0,255,0 - Red is 255,0,0

```
plot_ly(train_data, x = ~Red, y = ~Green, z = ~Blue, color = ~BT,
        colors = c("orange", "lightblue")) %>%
  add_markers() %>%
  layout(title = "Test Data | 3D RGB Plot by BlueTarp",
         scene = list(xaxis = list(title = 'Red'),
                      yaxis = list(title = 'Green'),
                      zaxis = list(title = 'Blue')))
```

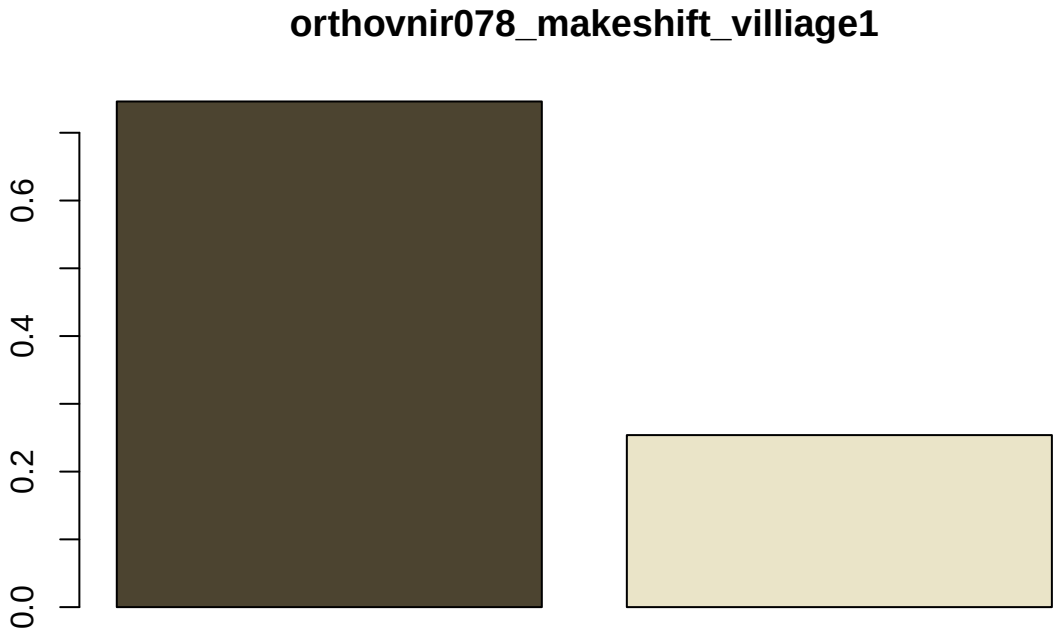
WebGL is not supported by your browser - visit https://get.webgl.org for more info

Here we can clearly see the separation of the two levels of BlueTarp in the training data.

```
image_path <- "orthovnir078_makeshift_villiage1.jpg"
colordistance::plotPixels(image_path)
```



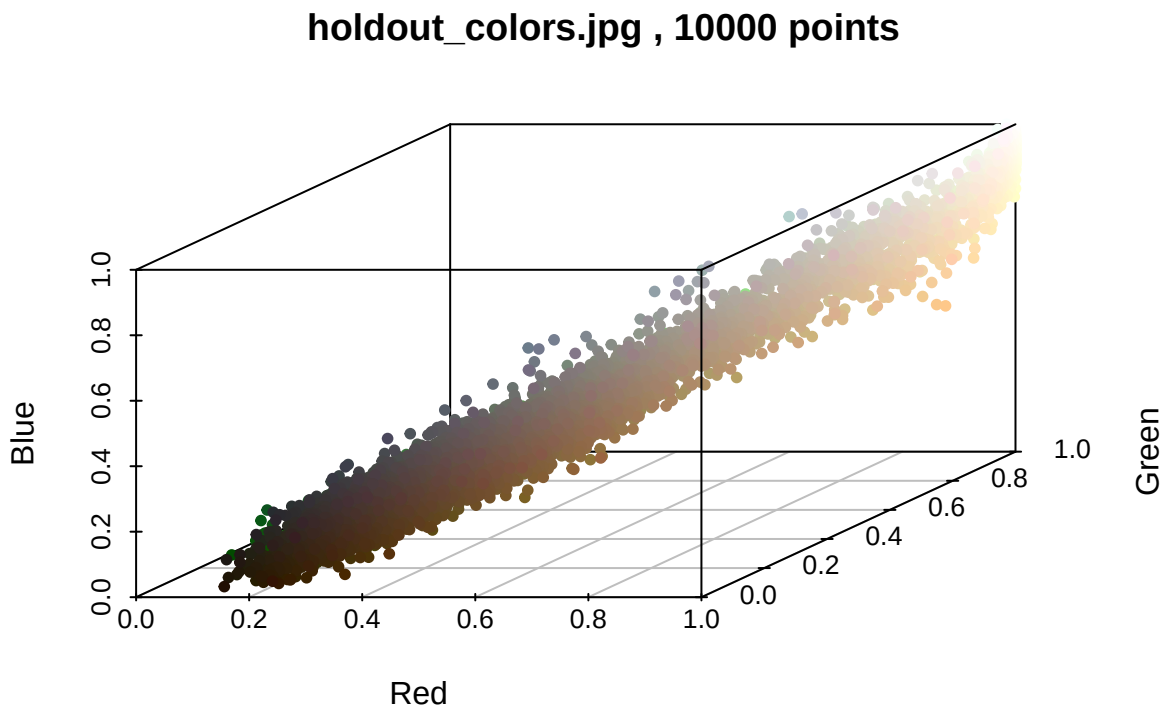

```
H8hist <- colordistance::getImageHist(image_path, bins=c(1, 1, 2))
```




```
# Number of pixels
n <- nrow(holdout_data)
maxHeight <- 65500
height <- min(n, maxHeight)
width <- ceiling(n / height)
total_pixels <- height * width
```

```r
# Normalize RGB
r <- holdout_data$Red / 255
g <- holdout_data$Green / 255
b <- holdout_data$Blue / 255

# Calculate padding
pad <- total_pixels - n

# Pad
if(pad > 0){
  r <- c(r, rep(0, pad))
  g <- c(g, rep(1, pad))
  b <- c(b, rep(0, pad))
}

# Make array
img_array <- array(c(matrix(r, nrow = height, ncol = width),
                     matrix(g, nrow = height, ncol = width),
                     matrix(b, nrow = height, ncol = width)),
                   dim = c(height, width, 3))

# Write to jpg
writeJPEG(img_array, target = "holdout_colors.jpg")
```

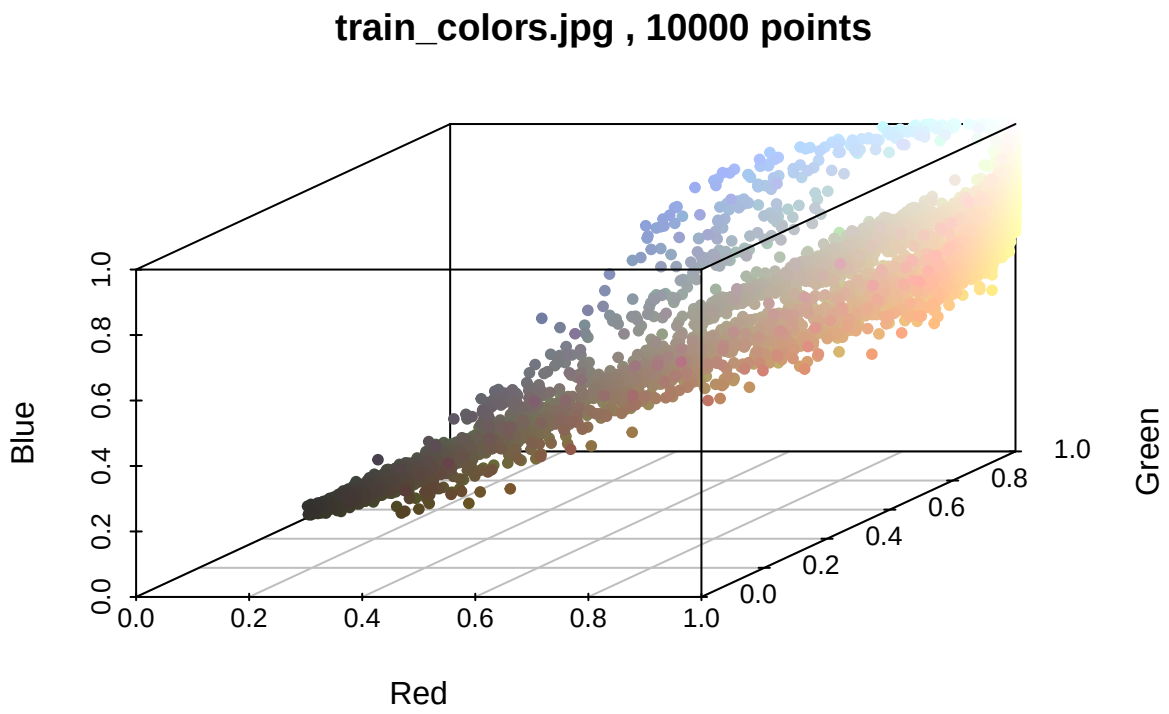
```r
image_path <- "holdout_colors.jpg"
colordistance::plotPixels(image_path)
```




```r
# Number of pixels
n <- nrow(train_data)
maxHeight <- 65500
height <- min(n, maxHeight)
width <- ceiling(n / height)
```

```
total_pixels <- height * width

# Normalize RGB
r <- train_data$Red / 255
g <- train_data$Green / 255
b <- train_data$Blue / 255

# Calculate padding
pad <- total_pixels - n

# Pad
if(pad > 0){
  r <- c(r, rep(0, pad))
  g <- c(g, rep(1, pad))
  b <- c(b, rep(0, pad))
}

# Make array
img_array <- array(c(matrix(r, nrow = height, ncol = width),
                     matrix(g, nrow = height, ncol = width),
                     matrix(b, nrow = height, ncol = width)),
                   dim = c(height, width, 3))

# Write to jpg
writeJPEG(img_array, target = "train_colors.jpg")
```

```
image_path <- "train_colors.jpg"
colordistance::plotPixels(image_path)
```

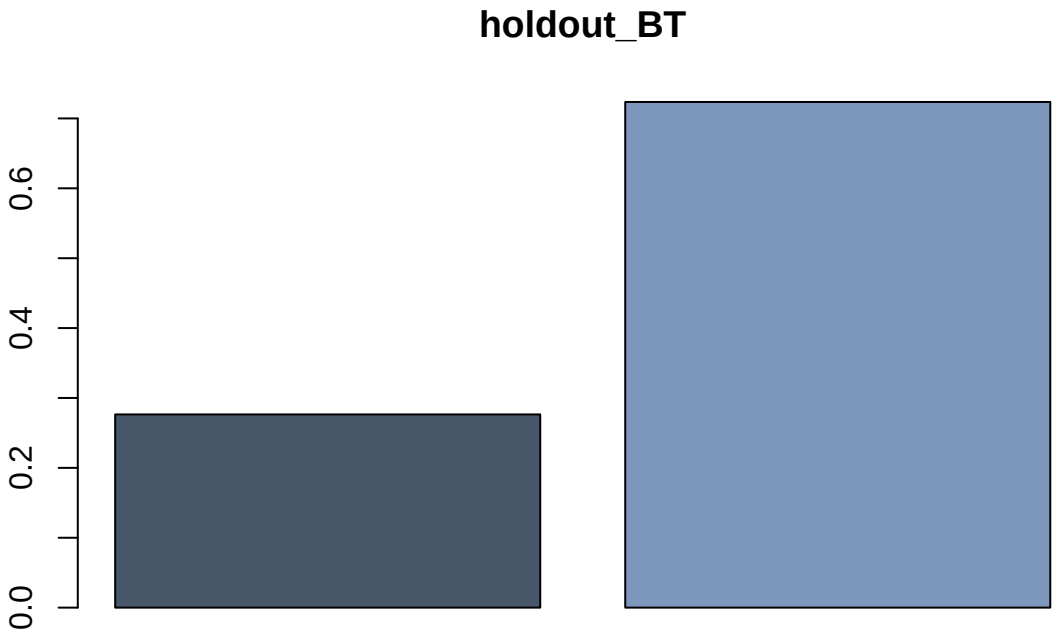



```
bt_data <- holdout_data %>%
  filter(BT == "TRUE")
```

```
n <- nrow(bt_data)
maxHeight <- 65500
height <- min(n, maxHeight)
width <- ceiling(n / height)
total_pixels <- height * width

r <- bt_data$Red / 255
g <- bt_data$Green / 255
b <- bt_data$Blue / 255

pad <- total_pixels - n

if(pad > 0){
  r <- c(r, rep(0, pad))
  g <- c(g, rep(1, pad))
  b <- c(b, rep(0, pad))
}

img_array <- array(
  c(matrix(r, nrow = height, ncol = width),
    matrix(g, nrow = height, ncol = width),
    matrix(b, nrow = height, ncol = width)),
  dim = c(height, width, 3)
)

writeJPEG(img_array, target = "holdout_BT.jpg")
```

```
image_path <- "holdout_BT.jpg"
H8hist <- colordistance::getImageHist(image_path, bins=c(1, 1, 2))
```

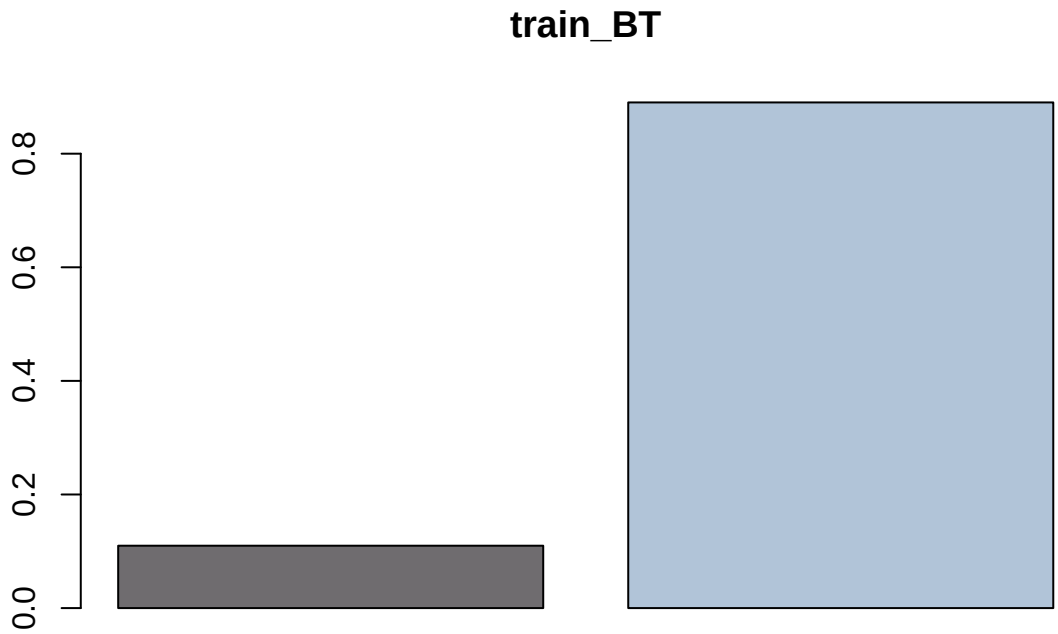
```
## RGB and HSV are device-dependent, perceptually non-uniform color spaces. See 'Color spaces' vignette
##
## Using 1*1*2 = 2 bins
```



```r
bt_data <- train_data %>%
  filter(BT == "TRUE")

n <- nrow(bt_data)
maxHeight <- 65500
height <- min(n, maxHeight)
width <- ceiling(n / height)
total_pixels <- height * width

r <- bt_data$Red / 255
g <- bt_data$Green / 255
b <- bt_data$Blue / 255

pad <- total_pixels - n

if(pad > 0){
  r <- c(r, rep(0, pad))
  g <- c(g, rep(1, pad))
  b <- c(b, rep(0, pad))
}

img_array <- array(
  c(matrix(r, nrow = height, ncol = width),
    matrix(g, nrow = height, ncol = width),
    matrix(b, nrow = height, ncol = width)),
  dim = c(height, width, 3)
)

writeJPEG(img_array, target = "train_BT.jpg")
```

```r
image_path <- "train_BT.jpg"
H8hist <- colordistance::getImageHist(image_path, bins=c(1, 1, 2))
```

```
## RGB and HSV are device-dependent, perceptually non-uniform color spaces. See 'Color spaces' vignette
##
## Using 1*1*2 = 2 bins
```



Have a look at the distributioin of the two classes for the outcome named "BT" (for BlueTarp).

```
train_data |>
    ggplot(aes(x=BT, fill=BT)) +
    geom_bar(position="dodge")
```

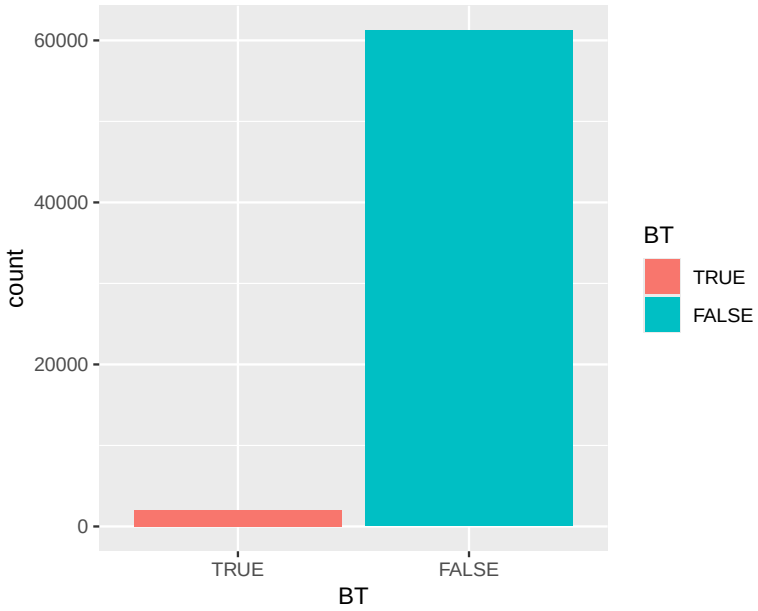


Figure 1: Distribution of Blue Tarp among all the observations.

I can see that the two outcome classes are extremely unbalanced. I will keep this in mind and deal with it later.

Methods

Build three classification models, *i.e.*, LDA, QDA, and logistic regression, with cross-validation.

Prepare model workflows Define the preprocessing steps. In this case, we normalize all numeric predictors.

```
# Formula and recipe
formula <- BT ~ Red + Green + Blue
BT_recipe <- recipe(formula, data = train_data) %>%
  step_normalize(all_numeric_predictors())
```

Specify the three models.

```
# Specify models
logreg_spec <- logistic_reg(mode="classification", engine="glm")
lda_spec <- discrim_linear(mode="classification", engine="MASS")
qda_spec <- discrim_quad(mode="classification", engine="MASS")
```

Combine preprocessing steps and model specification in workflow.

```
# Define workflows
logreg_wf <- workflow() %>%
    add_recipe(BT_recipe) %>%
    add_model(logreg_spec)

lda_wf <- workflow() %>%
    add_recipe(BT_recipe) %>%
    add_model(lda_spec)

qda_wf <- workflow() %>%
    add_recipe(BT_recipe) %>%
    add_model(qda_spec)
```

Cross-validation Define cross-validation approach - 10-fold cross-validation using stratified sampling - Measure performance using ROC-AUC (we also collect accuracy) - Save resample predictions, so that we can build ROC curves using cross-validation results

```
# Define cross-validation approach
set.seed(6030)

resamples <- vfold_cv(train_data, v = 10, strata = BT)
custom_metrics <- metric_set(roc_auc, accuracy, precision, f_meas)
cv_control <- control_resamples(save_pred = TRUE)
```

Cross-validation

```
# Cross-validate
logreg_cv <- fit_resamples(logreg_wf, resamples, metrics=custom_metrics, control=cv_control)
lda_cv <- fit_resamples(lda_wf, resamples, metrics=custom_metrics, control=cv_control)
qda_cv <- fit_resamples(qda_wf, resamples, metrics=custom_metrics, control=cv_control)
```

```
# Fit to train_data
final_logreg_fit <- logreg_wf %>% fit(data = train_data)
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
final_lda_fit    <- lda_wf %>% fit(data = train_data)
final_qda_fit    <- qda_wf %>% fit(data = train_data)
```

Model performance before threshold selection

The performance metrics estimated using 10-fold cross-validation.

```
cv_metrics <- bind_rows(
    collect_metrics(logreg_cv) %>%
        mutate(model="Logistic regression"),
    collect_metrics(lda_cv) %>%
        mutate(model="LDA"),
    collect_metrics(qda_cv) %>%
        mutate(model="QDA")
)

cv_metrics %>%
    select(model, .metric, mean) %>%
    pivot_wider(names_from=".metric", values_from="mean") %>%
    knitr::kable(
      caption="Cross-validation performance metrics.",
      digits=3,
      col.names = c("Model", "Accuracy", "F-measure", "Precision", "ROC-AUC")
  ) %>%
  kableExtra::kable_styling(full_width = FALSE, position = "center")
```

Table 1: Cross-validation performance metrics.

Model	Accuracy	F-measure	Precision	ROC-AUC
Logistic regression	0.995	0.923	0.964	0.998
LDA	0.984	0.761	0.725	0.989
QDA	0.995	0.908	0.989	0.998

Visualization of the same data

```
ggplot(cv_metrics, aes(x=mean, y=model, xmin=mean - std_err, xmax=mean + std_err)) +
    geom_point() +
    geom_linerange() +
    facet_wrap(~ .metric)
```

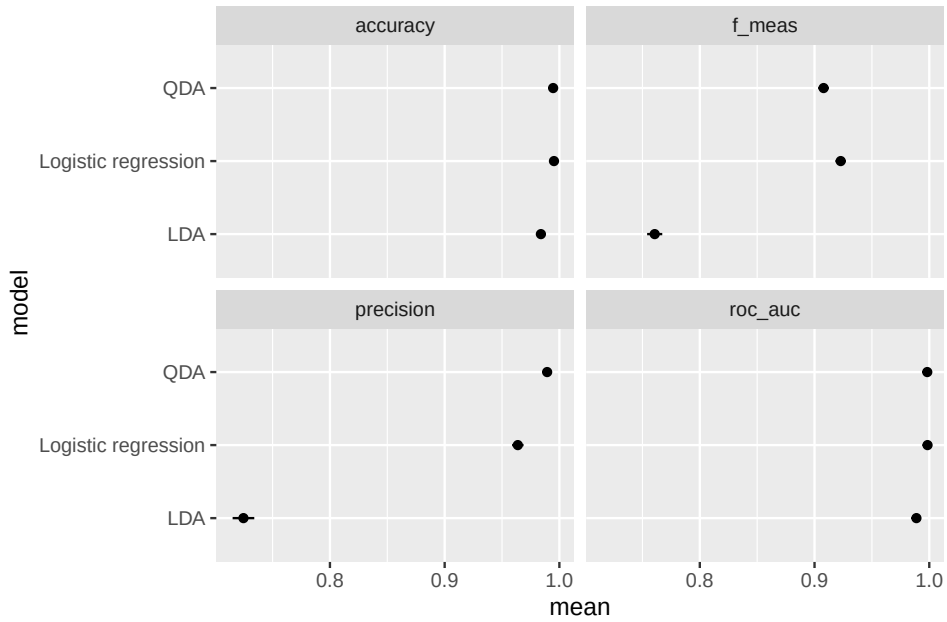


Figure 2: Cross-validation performance metrics

Cross-validation ROC curves

```
bind_rows(
    collect_predictions(logreg_cv) %>% mutate(model="Logistic regression"),
    collect_predictions(lda_cv) %>% mutate(model="LDA"),
    collect_predictions(qda_cv) %>% mutate(model="QDA")
) %>%
    group_by(model) %>%
    roc_curve(truth=BT, .pred_TRUE, event_level="first") %>%
    autoplot()
```

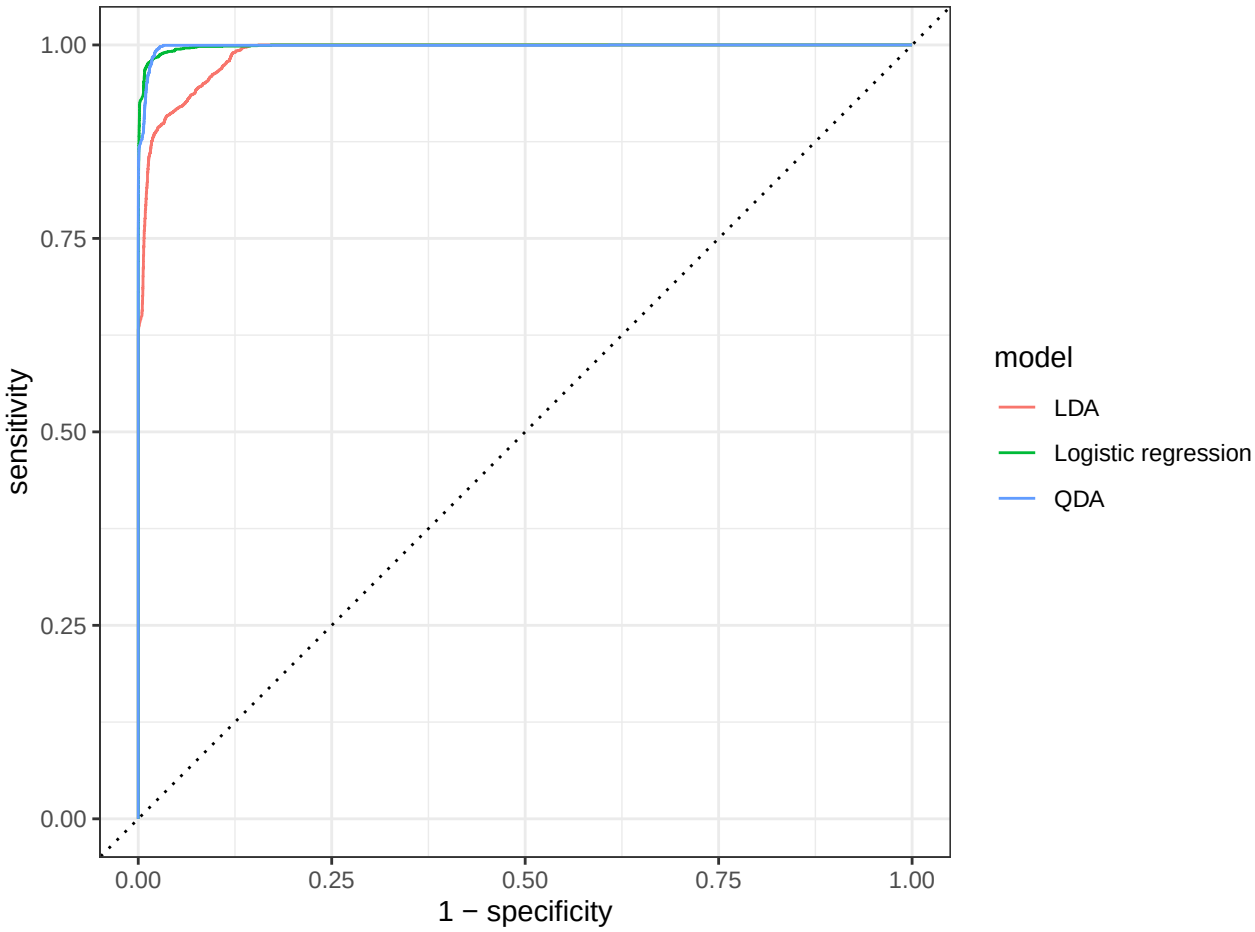


Figure 3: Overlay of cross-validation ROC curves

Threshold selection/Optimization

It is clear that our outcome classes are heavily imbalanced, so we need to adjust the threshold to improve its predictive accuracy and precision.

Use package `probably` to explore the threshold. We define two functions to look at the effect of threshold selection on performance metrics and the associated confusion matrices:

```
# Create metric set
class_metrics <- metric_set(accuracy, sens, f_meas)

# Compute metrics
compute_my_metrics <- function(data) {
  res_class <- class_metrics(data, truth = BT, estimate = .pred_class)
  res_prob  <- roc_auc(data, truth = BT, .pred_TRUE, event_level = "first")
  bind_rows(res_class, res_prob)
}
```

```
calculate_metrics_cv <- function(cv_object, holdout, model_name, workflow, train_data) {
  # Use CV preds
  train_aug <- collect_predictions(cv_object)

  # Augment with holdout
```

```
  final_fit   <- workflow %>% fit(train_data)
  holdout_aug <- augment(final_fit, new_data = holdout)

  bind_rows(
    bind_cols(
      model   = model_name,
      dataset = "train",
      compute_my_metrics(train_aug)
    ),
    bind_cols(
      model   = model_name,
      dataset = "holdout",
      compute_my_metrics(holdout_aug)
    )
  )
}
```

```
all_metrics <- bind_rows(
    calculate_metrics_cv(logreg_cv, holdout_data, "logreg", logreg_wf, train_data),
    calculate_metrics_cv(lda_cv, holdout_data, "LDA", lda_wf, train_data),
    calculate_metrics_cv(qda_cv, holdout_data, "QDA", qda_wf, train_data)
) %>% arrange(dataset)
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
all_metrics %>%
        pivot_wider(names_from=.metric, values_from=.estimate) %>%
        select(-.estimator) %>%
        knitr::kable(
          caption= "Metrics for the classification models.",
          digits=3) %>%
        kableExtra::kable_styling(full_width=FALSE)
```

Table 2: Metrics for the classification models.

model	dataset	accuracy	sens	f_meas	roc_auc
logreg	holdout	0.990	0.988	0.583	0.999
LDA	holdout	0.982	0.839	0.399	0.992
QDA	holdout	0.996	0.695	0.714	0.992
logreg	train	0.995	0.886	0.923	0.998
LDA	train	0.984	0.801	0.761	0.989
QDA	train	0.995	0.839	0.908	0.998

```
threshold_graph <- function(model_cv, model_name) {
    performance <- probably::threshold_perf(collect_predictions(model_cv), BT, .pred_TRUE,
        thresholds=seq(0.01, 0.99, 0.01), event_level="first",
        metrics=metric_set(f_meas, accuracy, sens))
    max_metrics <- performance %>%
        drop_na() %>%
        group_by(.metric) %>%
        filter(.estimate == max(.estimate))
    g <- ggplot(performance, aes(x=.threshold, y=.estimate, color=.metric)) +
        geom_line() +
```

```
        geom_point(data=max_metrics, color="black") +
        labs(title=model_name, x="Threshold", y="Metric value") +
        coord_cartesian(ylim=c(0, 1))
    thresholds <- max_metrics %>%
        select(.metric, .threshold) %>%
        deframe()
    return(list(graph=g, thresholds=thresholds))
}

visualize_conf_mat <- function(model_cv, thresholds, metric) {
    threshold <- thresholds[metric]
    cm <- collect_predictions(logreg_cv) %>%
        mutate(
            .pred_class = make_two_class_pred(.pred_TRUE, c("TRUE", "FALSE"), threshold=threshold),
        ) %>%
        conf_mat(truth=BT, estimate=.pred_class)
    autoplot(cm, type="heatmap") +
        labs(title=sprintf("Threshold %.2f (%s)", threshold, metric))
}

overview_model <- function(model_cv, model_name) {
    tg <- threshold_graph(model_cv, model_name)
    g1 <- visualize_conf_mat(model_cv, tg$thresholds, "accuracy")
    g2 <- visualize_conf_mat(model_cv, tg$thresholds, "f_meas")
    g3 <- visualize_conf_mat(model_cv, tg$thresholds, "sens")
    tg$graph + (g1 / g2 / g3)
}
```

Notes:

- `f_meas` cannot be calculated for high threshold values. In this case, the function `threshold_perf` returns `NA` for the F-measure. We filter out these values using `drop_na()`.

```
g1 <- overview_model(logreg_cv, "Logistic regression")
g2 <- overview_model(lda_cv, "LDA")
g3 <- overview_model(qda_cv, "QDA")

g1 / g2 / g3
```

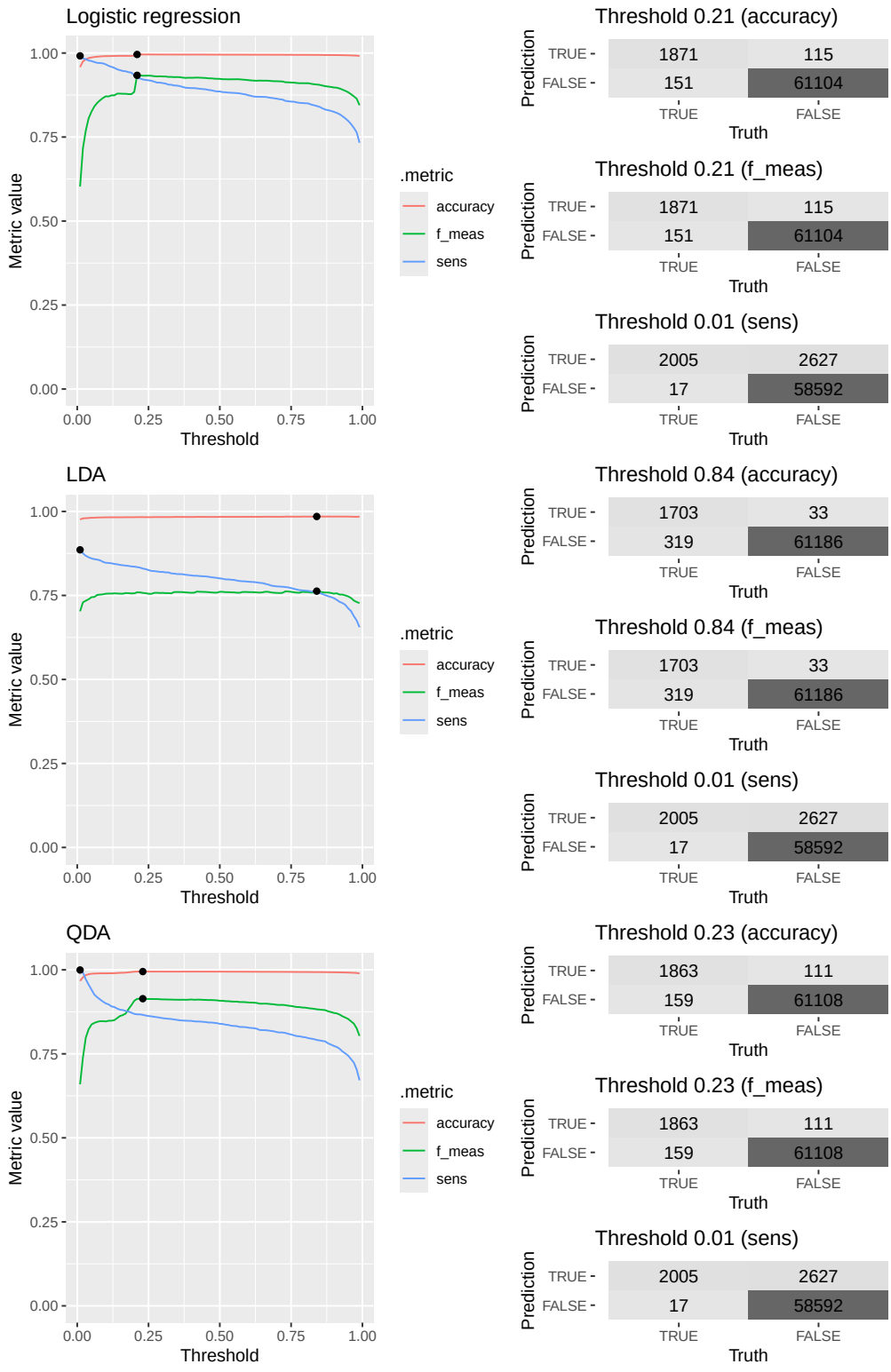


Figure 4: Metrics as a function of model performance

```
# Tweak f measure
f_meas_adj2 <- metric_tweak("f_meas_adj2", f_meas, beta = 2)
```

```r
f_meas_adj3 <- metric_tweak("f_meas_adj3", f_meas, beta = 3)

threshold_graph <- function(model_cv, model_name) {
    performance <- probably::threshold_perf(collect_predictions(model_cv), BT, .pred_TRUE,
        thresholds=seq(0.01, 0.99, 0.01), event_level="first",
        metrics=metric_set(f_meas, f_meas_adj2, f_meas_adj3))
    max_metrics <- performance %>%
        drop_na() %>%
        group_by(.metric) %>%
        filter(.estimate == max(.estimate))
    g <- ggplot(performance, aes(x=.threshold, y=.estimate, color=.metric)) +
        geom_line() +
        geom_point(data=max_metrics, color="black") +
        labs(title=model_name, x="Threshold", y="Metric value") +
        coord_cartesian(ylim=c(0, 1))
    thresholds <- max_metrics %>%
        select(.metric, .threshold) %>%
        deframe()
    return(list(graph=g, thresholds=thresholds))
}

visualize_conf_mat <- function(model_cv, thresholds, metric) {
    threshold <- thresholds[metric]
    cm <- collect_predictions(logreg_cv) %>%
        mutate(
            .pred_class = make_two_class_pred(.pred_TRUE, c("TRUE", "FALSE"), threshold=threshold),
        ) %>%
        conf_mat(truth=BT, estimate=.pred_class)
    autoplot(cm, type="heatmap") +
        labs(title=sprintf("Threshold %.2f (%s)", threshold, metric))
}

overview_model <- function(model_cv, model_name) {
    tg <- threshold_graph(model_cv, model_name)
    g1 <- visualize_conf_mat(model_cv, tg$thresholds, "f_meas")
    g2 <- visualize_conf_mat(model_cv, tg$thresholds, "f_meas_adj2")
    g3 <- visualize_conf_mat(model_cv, tg$thresholds, "f_meas_adj3")
    tg$graph + (g1 / g2 / g3)
}
```

```r
g1 <- overview_model(logreg_cv, "Logistic regression")
g2 <- overview_model(lda_cv, "LDA")
g3 <- overview_model(qda_cv, "QDA")

g1 / g2 / g3
```

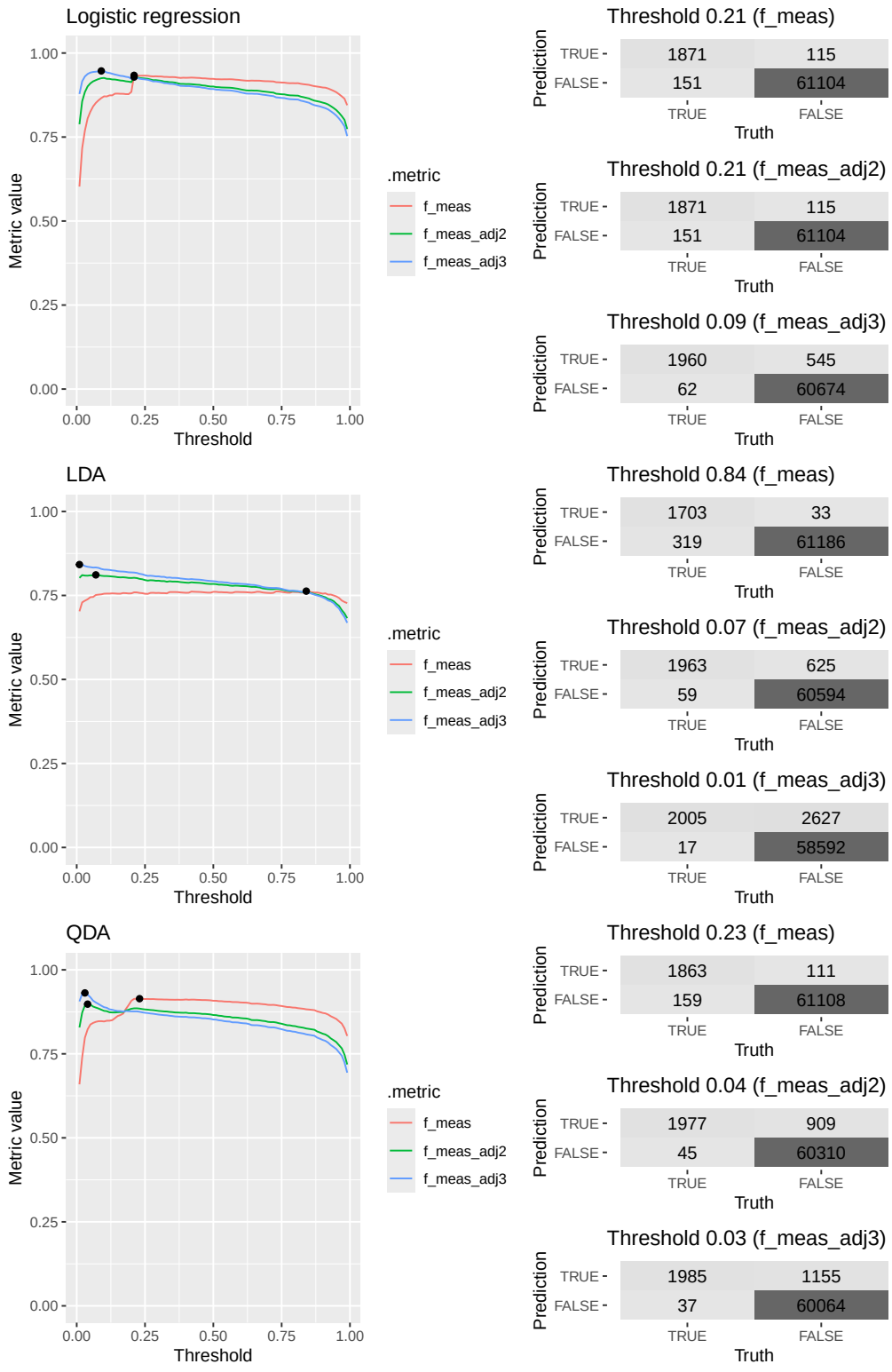


Figure 5: Metrics as a function of model performance

Next, I compared the ROC curves for the logistic regression between the cross-validation predictions and the predictions on the full training data set to see if the logistic regression was overfitting the training data.

```r
# Generate ROC plot
get_roc_overlay_autoplot <- function(cv_object, wf, data, model_name) {
  # Fit the full model
  full_fit <- wf %>% fit(data)

  # Get CV preds
  cv_preds <- collect_predictions(cv_object) %>%
    mutate(source = "CV")

  # Get preds for full model
  full_preds <- augment(full_fit, new_data = data) %>%
    mutate(source = "Full")

  # Compute ROC
  roc_data <- bind_rows(cv_preds, full_preds) %>%
    group_by(source) %>%
    roc_curve(truth = BT, .pred_TRUE, event_level = "first")

  # Use autoplot
  autoplot(roc_data) +
    ggtitle(paste(model_name)) +
    theme_minimal()
}

# Generate ROC overlay plots for each model
p_logreg <- get_roc_overlay_autoplot(logreg_cv, logreg_wf, train_data, "Logistic Regression")
p_lda    <- get_roc_overlay_autoplot(lda_cv, lda_wf, train_data, "LDA")
p_qda    <- get_roc_overlay_autoplot(qda_cv, qda_wf, train_data, "QDA")

# Patchwork the three plots into one graphic
combined_roc <- p_logreg + p_lda + p_qda +
  plot_annotation(title = "Overlay of ROC Curves (CV vs. Full Data Predictions)")
combined_roc
```

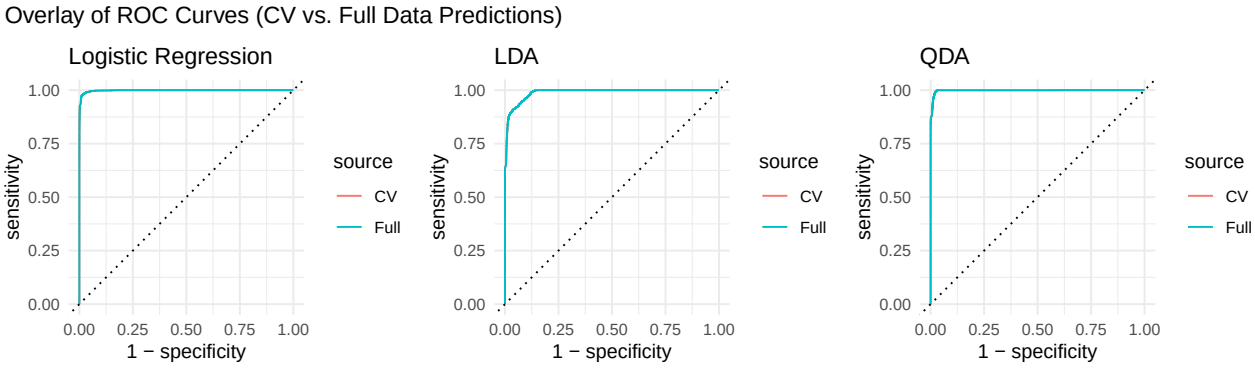


Figure 6: ROC curve comarison between corss-valication and full data set predictions

As we can see that the ROC curve for the cross-validation predictions is almost identical to the ROC curve for the predictions on the full training set, indicating that the logistic regression model is not overfitting the training data.

```r
# Find optimal thresholds
threshold_scan <- function(model, data, model_name) {
  threshold_data <- model %>%
    augment(data) %>%
    probably::threshold_perf(
      truth = BT,
      estimate = .pred_TRUE,
      thresholds = seq(0.01, 0.99, 0.01),
      event_level = "first",
      metrics = metric_set(f_meas)
    )
  opt_threshold <- threshold_data %>%
    drop_na() %>%
    arrange(desc(.estimate)) %>%
    slice(1)
  list(
    threshold = opt_threshold$.threshold,
    threshold_data = threshold_data,
    opt_threshold = opt_threshold,
    model_name = model_name
  )
}
```

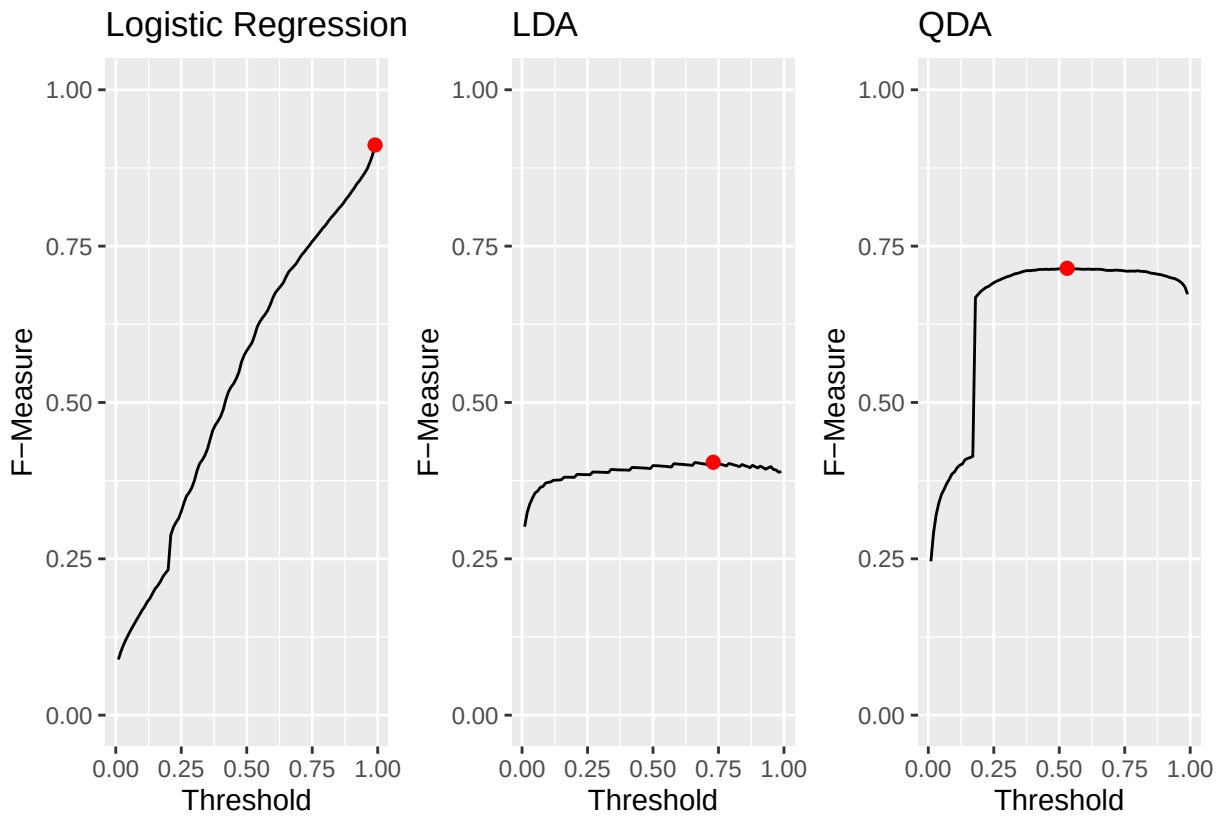
```r
# Fitted models
logreg_result <- threshold_scan(final_logreg_fit, holdout_data, "Logistic Regression")
lda_result    <- threshold_scan(final_lda_fit, holdout_data, "LDA")
qda_result    <- threshold_scan(final_qda_fit, holdout_data, "QDA")
```

```r
# Optimal thresholds
logreg_holdout_threshold <- logreg_result$threshold
lda_holdout_threshold    <- lda_result$threshold
qda_holdout_threshold    <- qda_result$threshold
```

```r
threshold_scan_cv <- function(cv_obj, model_name) {
  threshold_data <- cv_obj %>%
    collect_predictions() %>%
    probably::threshold_perf(
      truth = BT,
      estimate = .pred_TRUE,
      thresholds = seq(0.05, 0.95, 0.01),
      event_level = "first",
      metrics = metric_set(f_meas)
    )
  opt_threshold <- threshold_data %>%
    drop_na() %>%
    arrange(desc(.estimate)) %>%
    slice(1)
  list(
    threshold = opt_threshold$.threshold
  )
}

# CV objects
logreg_train_result <- threshold_scan_cv(logreg_cv, "Logistic Regression")
```

```
lda_train_result    <- threshold_scan_cv(lda_cv, "LDA")
qda_train_result    <- threshold_scan_cv(qda_cv, "QDA")

# Optimal thresholds
logreg_train_threshold <- logreg_train_result$threshold
lda_train_threshold    <- lda_train_result$threshold
qda_train_threshold    <- qda_train_result$threshold
```

```
# Plot and combine threshold graphs
plot_threshold <- function(result) {
  ggplot(result$threshold_data, aes(x = .threshold, y = .estimate)) +
    geom_line() +
    geom_point(data = result$opt_threshold, color = "red", size = 2) +
    labs(title = result$model_name, x = "Threshold", y = "F-Measure") +
    coord_cartesian(ylim = c(0, 1))
}

g1 <- plot_threshold(logreg_result)
g2 <- plot_threshold(lda_result)
g3 <- plot_threshold(qda_result)

# Combine the plots
g1 + g2 + g3
```




```
predict_at_threshold <- function(model, data, threshold) {
  model %>%
    augment(data) %>%
    mutate(.pred_class = make_two_class_pred(.pred_TRUE,
```

```
                    c("TRUE", "FALSE"),
                    threshold = threshold))
}

calculate_metrics_at_threshold <- function(model, train, holdout, model_name, train_threshold, holdout_
  bind_rows(
    # Metrics for the training set
    bind_cols(
      model = model_name,
      dataset = "train",
      threshold = train_threshold,
      metrics(predict_at_threshold(model, train, train_threshold), truth = BT, estimate = .pred_class)
    ),
    bind_cols(
      model = model_name,
      dataset = "train",
      threshold = train_threshold,
      roc_auc(model %>% augment(train), BT, .pred_TRUE, event_level = "first")
    ),
    bind_cols(
      model = model_name,
      dataset = "train",
      threshold = train_threshold,
      f_meas(predict_at_threshold(model, train, train_threshold), truth = BT, estimate = .pred_class)
    ),
    bind_cols(
      model = model_name,
      dataset = "train",
      threshold = train_threshold,
      sens(predict_at_threshold(model, train, train_threshold), truth = BT, estimate = .pred_class)
    ),
    # Metrics for the holdout set
    bind_cols(
      model = model_name,
      dataset = "holdout",
      threshold = holdout_threshold,
      metrics(predict_at_threshold(model, holdout, holdout_threshold), truth = BT, estimate = .pred_cla
    ),
    bind_cols(
      model = model_name,
      dataset = "holdout",
      threshold = holdout_threshold,
      roc_auc(model %>% augment(holdout), BT, .pred_TRUE, event_level = "first")
    ),
    bind_cols(
      model = model_name,
      dataset = "holdout",
      threshold = holdout_threshold,
      f_meas(predict_at_threshold(model, holdout, holdout_threshold), truth = BT, estimate = .pred_clas
    ),
    bind_cols(
      model = model_name,
      dataset = "holdout",
```

```
      threshold = holdout_threshold,
      sens(predict_at_threshold(model, holdout, holdout_threshold), truth = BT, estimate = .pred_class)
    )
  )
}

metrics_at_threshold <- bind_rows(
    calculate_metrics_at_threshold(final_logreg_fit, train_data, holdout_data, "Logistic regression", lo
    calculate_metrics_at_threshold(final_lda_fit, train_data, holdout_data, "LDA", lda_train_threshold,
    calculate_metrics_at_threshold(final_qda_fit, train_data, holdout_data, "QDA", qda_train_threshold,
) %>% arrange(dataset)
```

```
metrics_at_threshold %>%
        pivot_wider(names_from=.metric, values_from=.estimate) %>%
        select(-.estimator) %>%
        knitr::kable(
          caption= "Performance metrics for models at ideal threshold.",
          digits=3) %>%
        kableExtra::kable_styling(full_width=FALSE)
```

Table 3: Performance metrics for models at ideal threshold.

model	dataset	threshold	accuracy	kap	roc_auc	f_meas	sens
Logistic regression	holdout	0.99	0.999	0.911	0.999	0.912	0.953
LDA	holdout	0.73	0.983	0.398	0.992	0.404	0.797
QDA	holdout	0.53	0.996	0.712	0.992	0.714	0.689
Logistic regression	train	0.21	0.996	0.932	0.999	0.934	0.926
LDA	train	0.84	0.985	0.755	0.989	0.763	0.759
QDA	train	0.23	0.995	0.911	0.998	0.914	0.866

```
visualize_conf_mat_holdout <- function(model, data, threshold, metric_label) {
  cm <- model %>%
    augment(data) %>%
    mutate(.pred_class = make_two_class_pred(.pred_TRUE, c("TRUE", "FALSE"), threshold = threshold)) %>%
    conf_mat(truth = BT, estimate = .pred_class)

  autoplot(cm, type = "heatmap") +
    labs(title = sprintf("Threshold %.2f (%s)", threshold, metric_label))
}

overview_model_holdout <- function(model, model_name, threshold) {
  cm_plot <- visualize_conf_mat_holdout(model, holdout_data, threshold, "Holdout")
  cm_plot + labs(title = sprintf("%s \n (Threshold = %.2f)", model_name, threshold))
}

logreg_cm <- overview_model_holdout(final_logreg_fit, "Logistic Regression", logreg_holdout_threshold)
lda_cm    <- overview_model_holdout(final_lda_fit, "LDA", lda_holdout_threshold)
qda_cm    <- overview_model_holdout(final_qda_fit, "QDA", qda_holdout_threshold)
```

```
logreg_cm + lda_cm + qda_cm
```

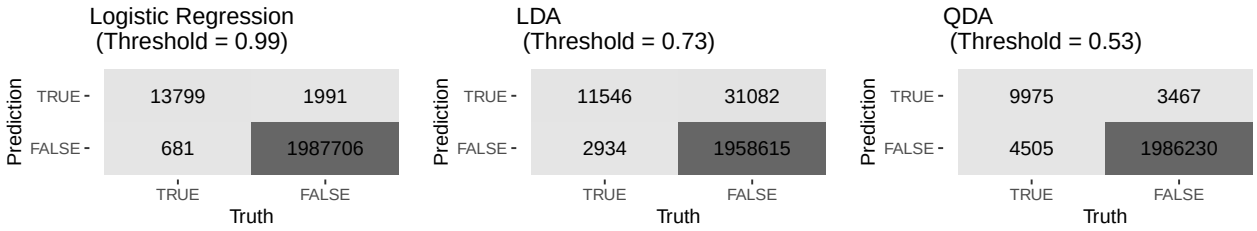


Figure 7: ROC curve comarison between corss-valication and full data set predictions

```
stopCluster(cl)
registerDoSEQ()
```