Project Part 1

Virginia Brame, Clay Harris, Hai Liu

2025-02-21

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

```
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)
```

Data loading

```
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(file.path("./HoldoutData", .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(file.path("./HoldoutData", .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)
```

```
# Set seed
set.seed(1353)

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

# 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")

# 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)

# Define cross-validation approach
resamples <- vfold_cv(train_data, v = 10, strata = BT)
custom_metrics <- metric_set(roc_auc, accuracy)
cv_control <- control_resamples(save_pred = TRUE)
```

```
# 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)
```

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

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

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

```r
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", "ROC-AUC")
  ) %>%
  kableExtra::kable_styling(full_width = FALSE, position = "center")
```

Table 1: Cross-validation performance metrics.

Model	Accuracy	ROC-AUC
Logistic regression	0.995	0.998
LDA	0.984	0.989
QDA	0.995	0.998

```r
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()
```

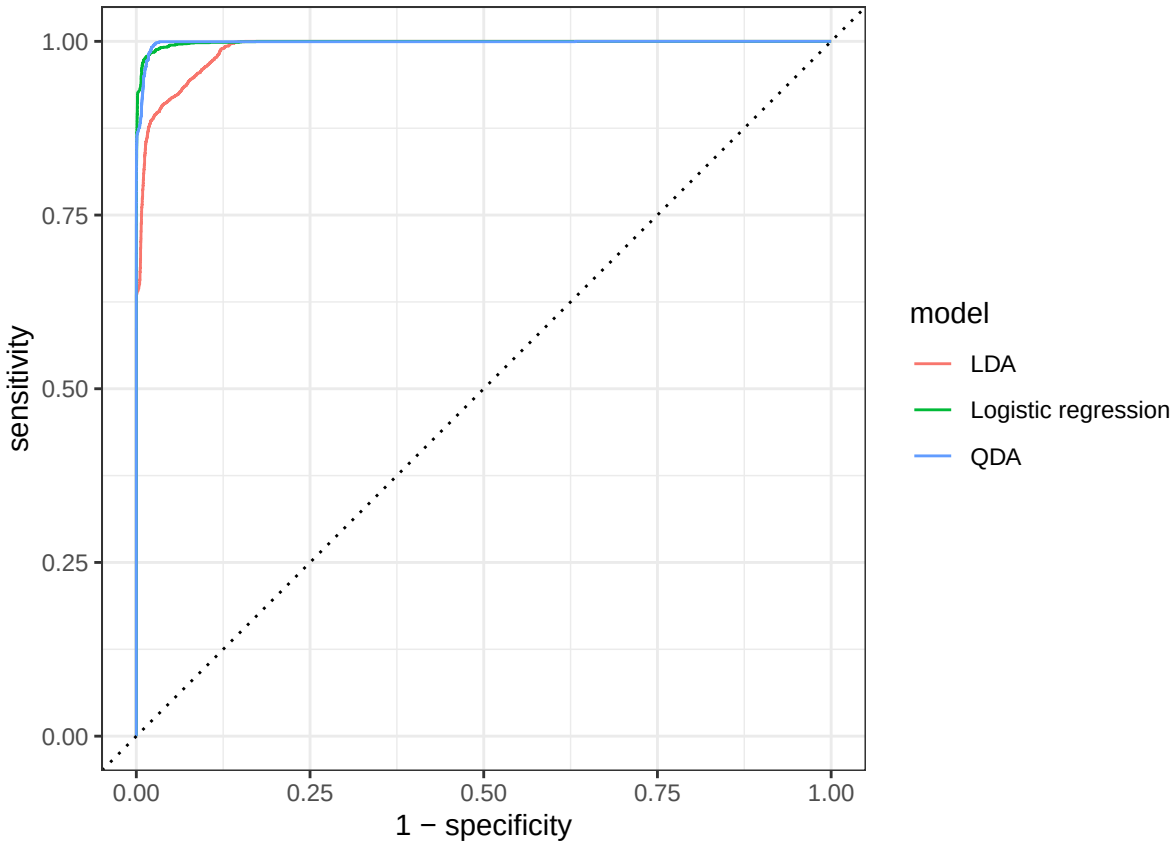


Figure 1: ROC curves for Logistic Regression, LDA, and QDA Models.

```
all_metrics <- bind_rows(
    calculate_metrics(final_logreg_fit, train_data, holdout_data, "logreg"),
    calculate_metrics(final_lda_fit, train_data, holdout_data, "LDA"),
    calculate_metrics(final_qda_fit, train_data, holdout_data, "QDA"),
)
all_metrics <- all_metrics %>% arrange(dataset)
```

```
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)
```

```
## Warning: Values from `.estimate` are not uniquely identified; output will contain
## list-cols.
## * Use `values_fn = list` to suppress this warning.
## * Use `values_fn = {summary_fun}` to summarise duplicates.
## * Use the following dplyr code to identify duplicates.
##   {data} |>
##   dplyr::summarise(n = dplyr::n(), .by = c(model, dataset, .estimator,
##   .metric)) |>
```

```
##   dplyr::filter(n > 1L)
```

Table 2: Metrics for the classification models.

model	dataset	accuracy	kap	roc_auc	f_meas	sens
logreg	holdout	0.9897793	0.5785505	0.9994131	0.5828446, 0.7731457	0.9882597
LDA	holdout	0.9817496	0.3924352	0.9921155	0.399126, 0.582286	0.8389503
QDA	holdout	0.9959719	0.7115849	0.9915001	0.7136117, 0.7020899	0.6946133
logreg	train	0.9952879	0.9207303	0.9985069	0.9231563, 0.9000402	0.8852621
LDA	train	0.9839661	0.7533611	0.9888768	0.7616361, 0.7848837	0.8011869
QDA	train	0.9946079	0.90604	0.9982175	0.9087991, 0.8663947	0.8402572

```
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)
}
```

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

g1 / g2 / g3
```

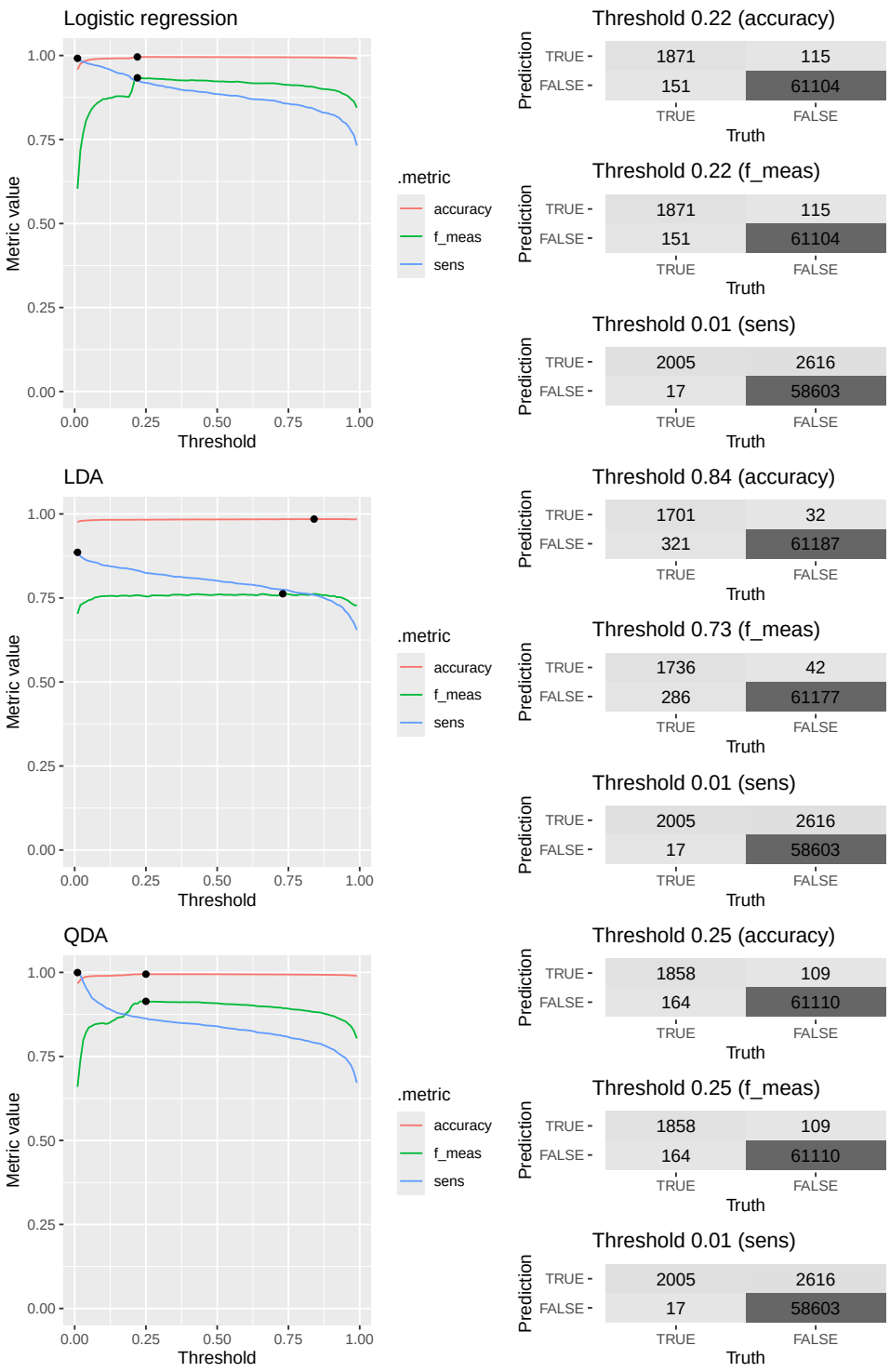
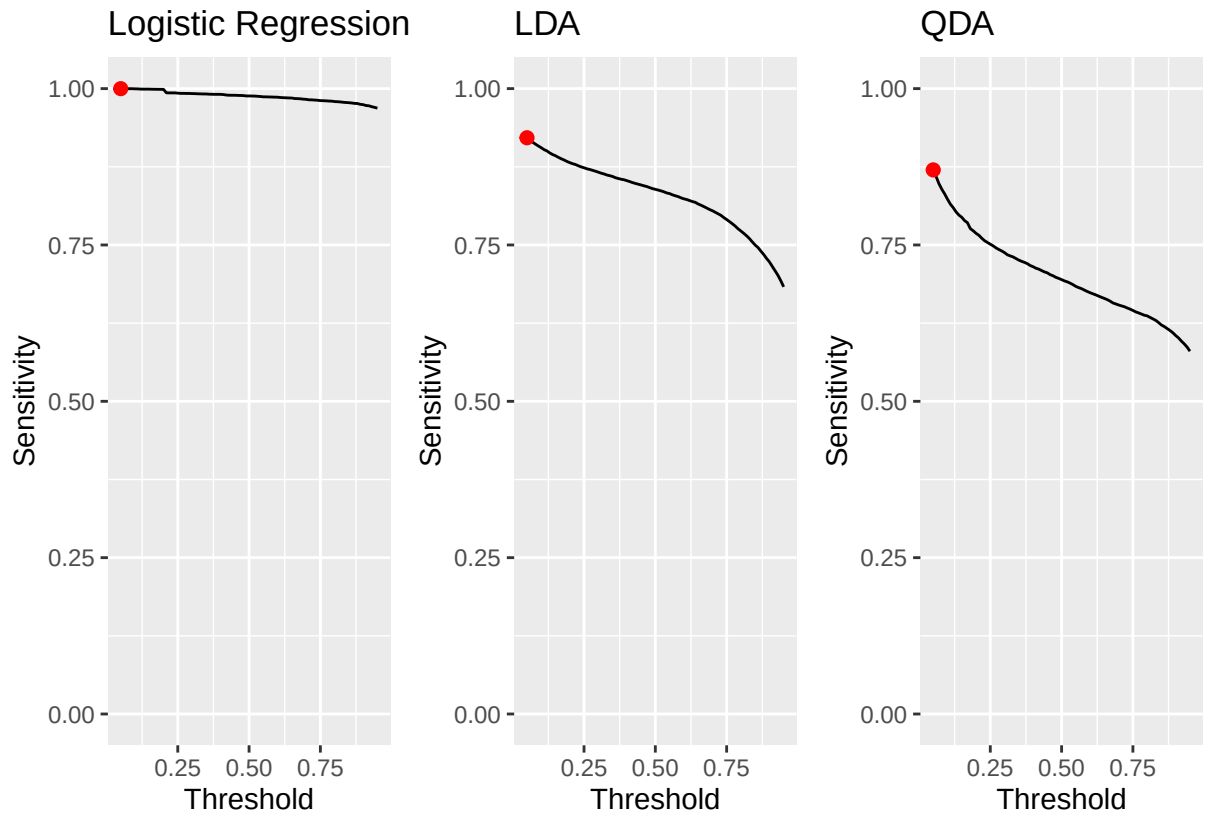


Figure 2: Metrics as a function of model performance

```
# 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(0.05, 0.95, 0.01),
      event_level = "first",
      metrics = metric_set(sens)
    )
  opt_threshold <- threshold_data %>%
    arrange(desc(.estimate)) %>%
    slice(1)
  list(
    model_name = model_name,
    threshold = opt_threshold$.threshold,
    threshold_data = threshold_data,
    opt_threshold = opt_threshold
  )
}

# Apply to final 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")

# Extract the optimal thresholds
logreg_threshold <- logreg_result$threshold
lda_threshold    <- lda_result$threshold
qda_threshold    <- qda_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 = "Sensitivity") +
    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) {
    return(
        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, threshold) {
  bind_rows(
    # Accuracy of training set
    bind_cols(
      model = model_name,
      dataset = "train",
      threshold = threshold,
      metrics(predict_at_threshold(model, train, threshold), truth = BT, estimate = .pred_class)
    ),
    # ROC-AUC of training set
    bind_cols(
      model = model_name,
      dataset = "train",
      threshold = threshold,
      roc_auc(model %>% augment(train), BT, .pred_TRUE, event_level = "first")
    ),
```

```
    # F-measure of training set
    bind_cols(
      model = model_name,
      dataset = "train",
      threshold = threshold,
      f_meas(predict_at_threshold(model, train, threshold), truth = BT, estimate = .pred_class)
    ),
    # Sensitivity (Recall) of training set
    bind_cols(
      model = model_name,
      dataset = "train",
      threshold = threshold,
      sens(predict_at_threshold(model, train, threshold), truth = BT, estimate = .pred_class)
    ),
    # Accuracy of holdout set
    bind_cols(
      model = model_name,
      dataset = "holdout",
      threshold = threshold,
      metrics(predict_at_threshold(model, holdout, threshold), truth = BT, estimate = .pred_class)
    ),
    # ROC-AUC of holdout set
    bind_cols(
      model = model_name,
      dataset = "holdout",
      threshold = threshold,
      roc_auc(model %>% augment(holdout), BT, .pred_TRUE, event_level = "first")
    ),
    # F-measure of holdout set
    bind_cols(
      model = model_name,
      dataset = "holdout",
      threshold = threshold,
      f_meas(predict_at_threshold(model, holdout, threshold), truth = BT, estimate = .pred_class)
    ),
    # Sensitivity (Recall) of holdout set
    bind_cols(
      model = model_name,
      dataset = "holdout",
      threshold = threshold,
      sens(predict_at_threshold(model, 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_threshold),
    calculate_metrics_at_threshold(final_qda_fit, train_data, holdout_data, "QDA", qda_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	0.05	0.903	0.118	0.999	0.130	1.000
LDA	holdout	0.05	0.976	0.348	0.992	0.355	0.921
QDA	holdout	0.05	0.977	0.345	0.992	0.353	0.870
Logistic regression	train	0.05	0.987	0.820	0.999	0.827	0.976
LDA	train	0.05	0.981	0.735	0.989	0.745	0.860
QDA	train	0.05	0.988	0.832	0.998	0.838	0.942

“‘{, eval=FALSE} # Convert for terra 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 (resolution in meters)

v_utm <- terra::project(v, “EPSG:32618”)

Create empty raster

r_empty <- terra::rast(terra::ext(v_utm), resolution = 0.1, crs = “EPSG:32618”)

3 bands

r_b1 <- terra::rasterize(v_utm, r_empty, field = “B1”, filename = “r_b1.tif”, overwrite = TRUE) r_b2 <- terra::rasterize(v_utm, r_empty, field = “B2”, filename = “r_b2.tif”, overwrite = TRUE) r_b3 <- terra::rasterize(v_utm, r_empty, field = “B3”, filename = “r_b3.tif”, overwrite = TRUE)

Combine bands

rgb_raster <- c(r_b1, r_b2, r_b3)

Reproject back to wgs84

rgb_raster_wgs <- terra::project(rgb_raster, “EPSG:4326”, filename = “rgb_raster_wgs.tif”, overwrite = TRUE)

Convert to brick for leaflet

rgb_brick <- raster::brick(rgb_raster_wgs)

```{,}
# Create map
m <- leaflet() %>%
  addTiles() %>%
  leafem::addRasterRGB(rgb_brick, r = 1, g = 2, b = 3)

# Save the map
htmlwidgets::saveWidget(m, "interactive_map.html")
```

{, eval=FALSE} #| message: FALSE # Aggregate raster (factor 10) rgb_brick_coarse <- raster::aggregate(rgb_brick, fact = 10, fun = mean)

“‘{, eval=FALSE} # Create map m <- leaflet() %>% addTiles() %>% leafem::addRasterRGB(rgb_brick_coarse, r = 1, g = 2, b = 3)

htmlwidgets::saveWidget(m, “interactive_map_coarse.html”)

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

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

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

# Normalize RGB
r <- holdout_data$B1 / 255
g <- holdout_data$B2 / 255
b <- holdout_data$B3 / 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")
```

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

```
test_data <- read.csv("HaitiPixels.csv")
```

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

# Normalize RGB
r <- test_data$Red / 255
g <- test_data$Green / 255
b <- test_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 = "test_colors.jpg")
```

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