Disaster Relief Project

Khoi Tran (kt2np)

May 9, 2021

Contents

SYS 6018 | Spring 2021 | University of Virginia

Introduction

The 2010 Haiti earthquake was of a 7.0 magnitude on the Richter scale, killing hundreds of thousands, and prompting a multi-million dollar international response in humanitarian aid. With the collapse of the island nation's infrastructure worsened by dozens of aftershocks, the first crucial step in aiding the Haitians was rescue. Countless people were stranded, unable to receive incoming supplies and medical aid, and aerial surveillance and airlifts were a key step before recovery could even be considered.

In this dataset, colors, designated by red, green, and blue values, are supplied, along with a classifier for their corresponding surface. In disasters like this, the ability to provide accurate predictions in order to spur a quick response - even when given spotty or spurious information - is crucial, and perhaps a statistical model could be the solution to identifying the locations of disaster victims.

Training Data / EDA

Set-up

```
# Load Haiti Pixels dataset
HaitiPixels <- read.csv('HaitiPixels.csv')

# view type for each variable
str(HaitiPixels)
```

```
## 'data.frame':    63241 obs. of  4 variables:
##  $ Class: chr  "Vegetation" "Vegetation" "Vegetation" "Vegetation" ...
##  $ Red  : int  64 64 64 75 74 72 71 69 68 67 ...
##  $ Green: int  67 67 66 82 82 76 72 70 70 70 ...
##  $ Blue : int  50 50 49 53 54 52 51 49 49 50 ...
```

```
# change 'Class' to a factor variable
HaitiPixels$Class <- as.factor(HaitiPixels$Class)

# create binary factor variable to simplify 'Class', the one that matters is the Blue Tarp, designating
HaitiPixels$Tarp <- ifelse(HaitiPixels$Class == 'Blue Tarp',
                           yes = 1, no = 0) %>%
                      as.factor()
```

Subsetting Data

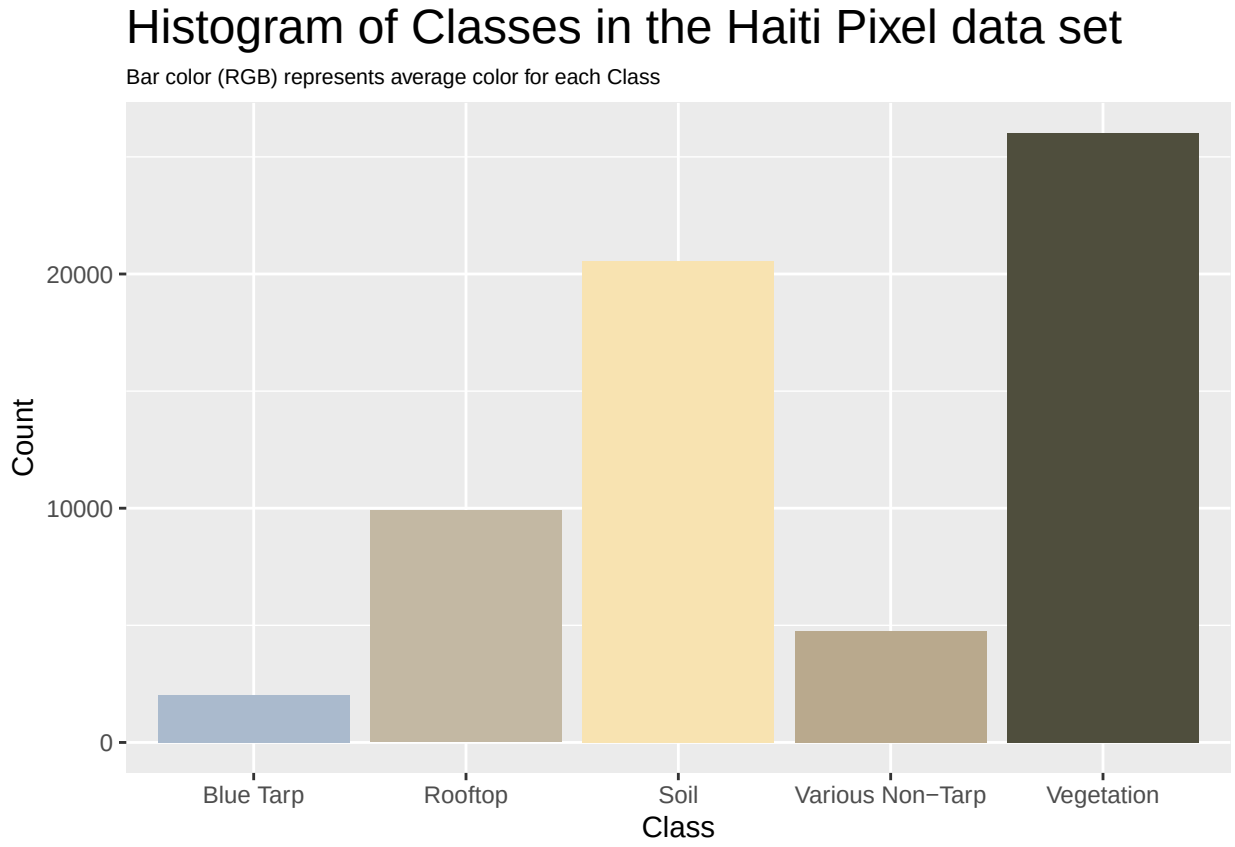
```
options(scipen = 999)

# preprocess to normalize values
# name as RGC, red-green chromatic
HP_RGC <- HaitiPixels[, -1] %>%
              transform(Red = Red / (Red + Green + Blue),
                        Green = Green / (Red + Green + Blue),
                        Blue = Blue / (Red + Green + Blue))
```

```
# apply RGB to HSV conversion from grDevices package
# use tidyr spread to convert from long to wide data, with hue, saturation, and value as columns
HP_HSV <- HaitiPixels
HP_HSV[, 1:4] <- mapply(rgb2hsv,
                        HaitiPixels$Red,
                        HaitiPixels$Blue,
                        HaitiPixels$Green) %>%
                    as.table() %>%
                    data.frame() %>%
                    spread(Var1, Freq)

# remove column for key
HP_HSV <- HP_HSV[, -1]
```

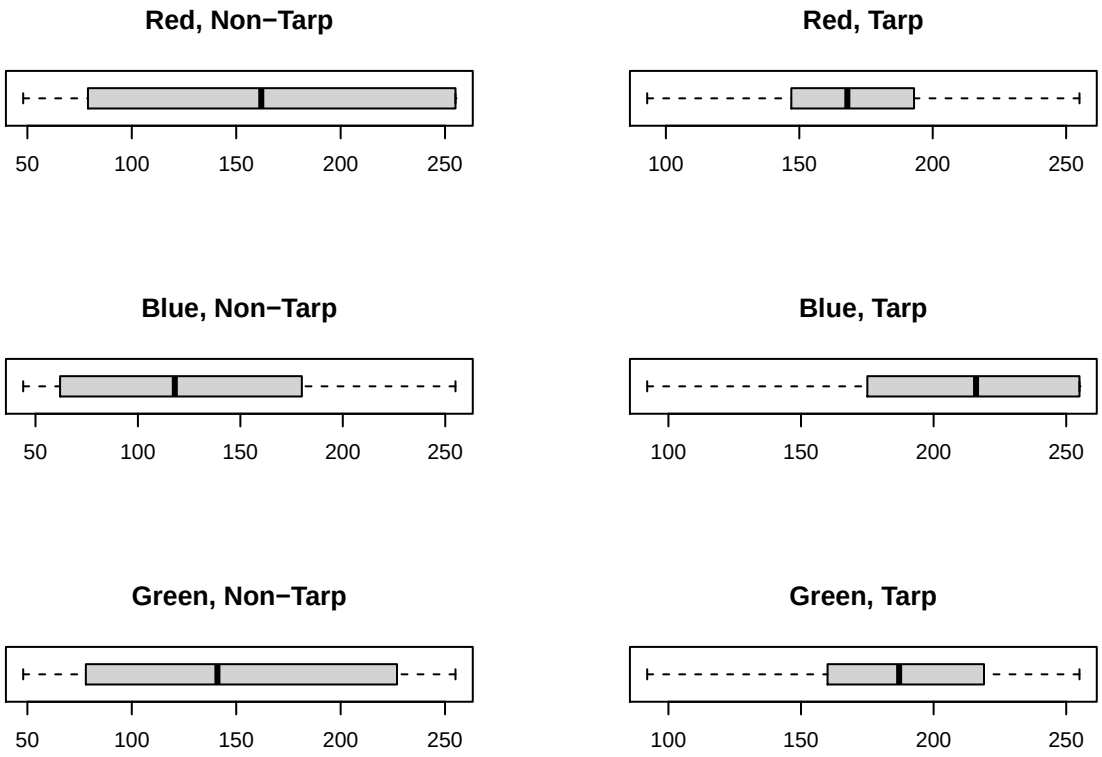
```r
# rename
names(HP_HSV) <- c('Hue', 'Saturation', 'Value', 'Tarp')
```

```r
# have standard RGB values for reference
HP_RGB <- HaitiPixels[, -1]
```

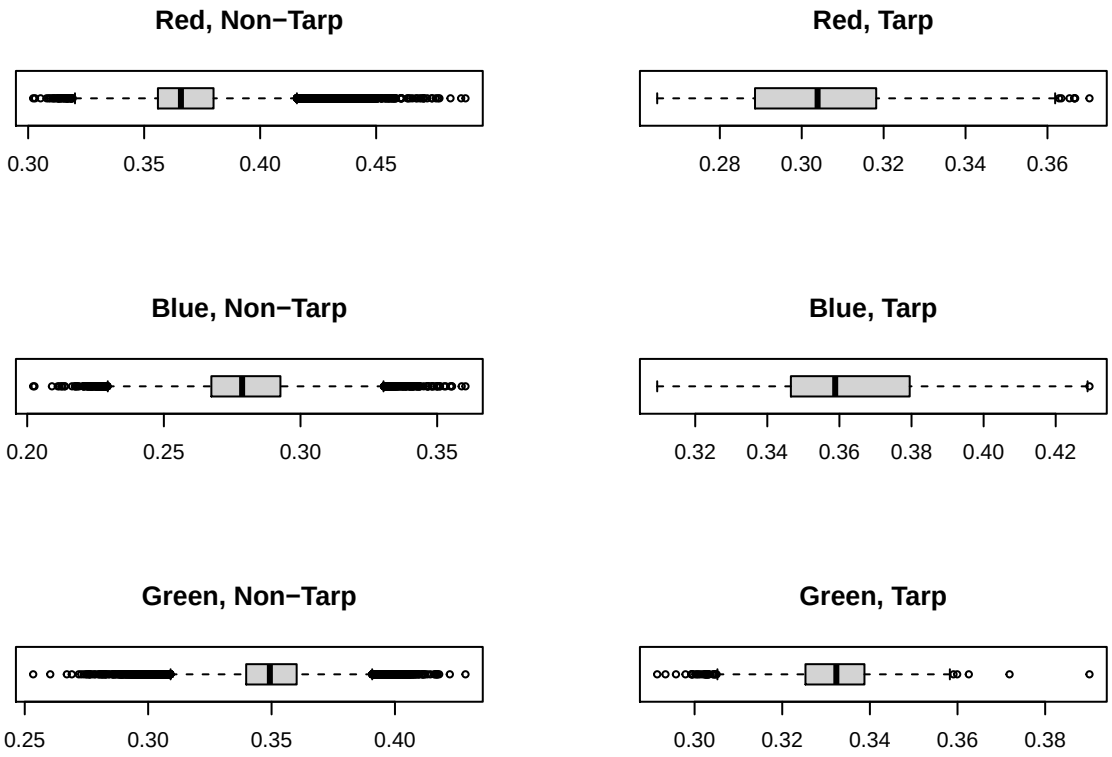
```r
# view frequency for 'Class'
# view average colors for each class
# divide each value by 255 to fit rgb() function

HPClass_viz <- merge(data.frame(HaitiPixels$Class),
                     data.frame(aggregate(HaitiPixels,
                                          by = list(HaitiPixels$Class),
                                          FUN = mean) %>%
                                  transform(avg_color = rgb(Red / 255,
                                                            Green / 255,
                                                            Blue / 255)))[, -c(2:6)],
                     by.x = 'HaitiPixels.Class', by.y = 'Group.1')
names(HPClass_viz) <- c('Class', 'Average_Color')

ggplot(HPClass_viz, aes(x = Class, fill = Average_Color)) +
  geom_bar() +
  scale_fill_manual(values = c('#AABACD' = '#AABACD', '#C3B8A3' = '#C3B8A3',
                               '#F8E3B1' = '#F8E3B1', '#B9A98D' = '#B9A98D',
                               '#4F4E3D' = '#4F4E3D')) +
  labs(title = 'Histogram of Classes in the Haiti Pixel data set',
       subtitle = 'Bar color (RGB) represents average color for each Class',
       x = 'Class', y = 'Count') +
  theme(plot.title = element_text(size = 18),
        plot.subtitle = element_text(size = 8),
        legend.position = 'none')
```



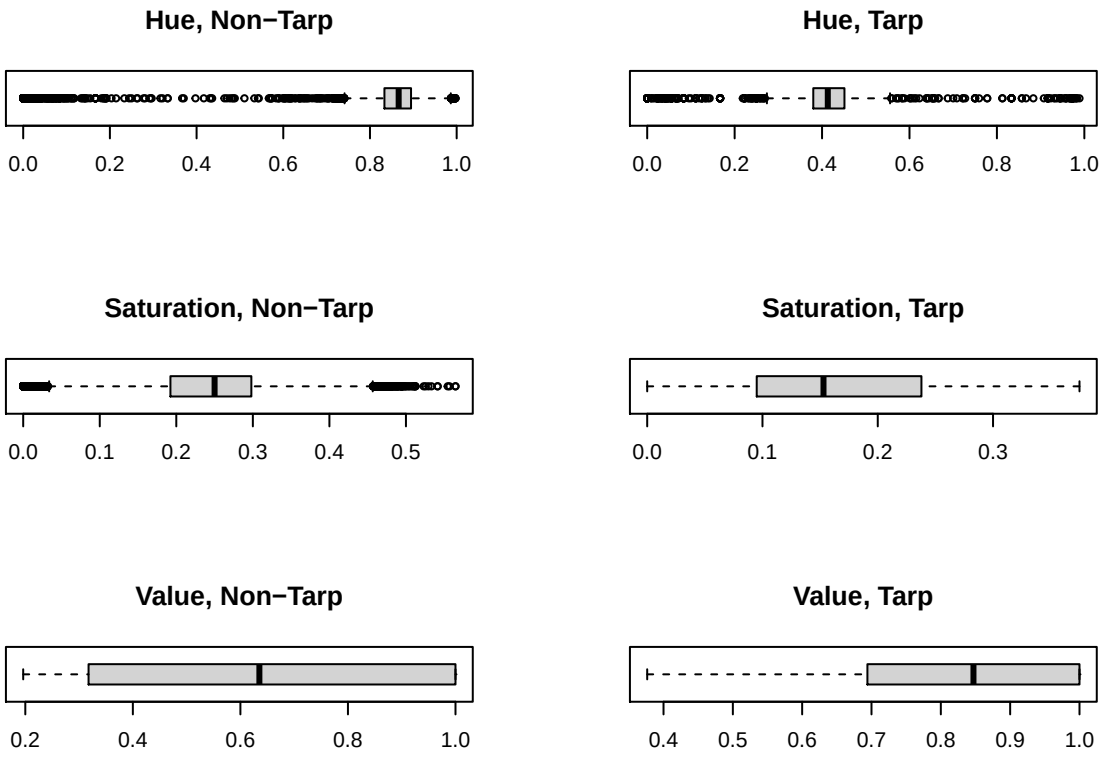

```
# plotting distribution of RGB values
par(mfrow = c(3, 2))
boxplot(HP_RGB[which(HP_RGB$Tarp == 0), ]$Red,
        horizontal = TRUE, main = 'Red, Non-Tarp')
boxplot(HP_RGB[which(HP_RGB$Tarp == 1), ]$Red,
        horizontal = TRUE, main = 'Red, Tarp')
boxplot(HP_RGB[which(HP_RGB$Tarp == 0), ]$Blue,
        horizontal = TRUE, main = 'Blue, Non-Tarp')
boxplot(HP_RGB[which(HP_RGB$Tarp == 1), ]$Blue,
        horizontal = TRUE, main = 'Blue, Tarp')
boxplot(HP_RGB[which(HP_RGB$Tarp == 0), ]$Green,
        horizontal = TRUE, main = 'Green, Non-Tarp')
boxplot(HP_RGB[which(HP_RGB$Tarp == 1), ]$Green,
        horizontal = TRUE, main = 'Green, Tarp')
```




```
# plotting distribution of normalized RGB values
par(mfrow = c(3, 2))
boxplot(HP_RGC[which(HP_RGC$Tarp == 0), ]$Red,
        horizontal = TRUE, main = 'Red, Non-Tarp')
boxplot(HP_RGC[which(HP_RGC$Tarp == 1), ]$Red,
        horizontal = TRUE, main = 'Red, Tarp')
boxplot(HP_RGC[which(HP_RGC$Tarp == 0), ]$Blue,
        horizontal = TRUE, main = 'Blue, Non-Tarp')
boxplot(HP_RGC[which(HP_RGC$Tarp == 1), ]$Blue,
        horizontal = TRUE, main = 'Blue, Tarp')
boxplot(HP_RGC[which(HP_RGC$Tarp == 0), ]$Green,
        horizontal = TRUE, main = 'Green, Non-Tarp')
boxplot(HP_RGC[which(HP_RGC$Tarp == 1), ]$Green,
        horizontal = TRUE, main = 'Green, Tarp')
```




```r
# plotting distribution of HSV values
par(mfrow = c(3, 2))
boxplot(HP_HSV[which(HP_HSV$Tarp == 0), ]$Hue,
        horizontal = TRUE, main = 'Hue, Non-Tarp')
boxplot(HP_HSV[which(HP_HSV$Tarp == 1), ]$Hue,
        horizontal = TRUE, main = 'Hue, Tarp')
boxplot(HP_HSV[which(HP_HSV$Tarp == 0), ]$Saturation,
        horizontal = TRUE, main = 'Saturation, Non-Tarp')
boxplot(HP_HSV[which(HP_HSV$Tarp == 1), ]$Saturation,
        horizontal = TRUE, main = 'Saturation, Tarp')
boxplot(HP_HSV[which(HP_HSV$Tarp == 0), ]$Value,
        horizontal = TRUE, main = 'Value, Non-Tarp')
boxplot(HP_HSV[which(HP_HSV$Tarp == 1), ]$Value,
        horizontal = TRUE, main = 'Value, Tarp')
```



Model Training

```
set.seed(6018)
# training and testing, 75/25 split
RGB_subset <- sample(nrow(HP_RGB), nrow(HP_RGB) * 0.75)
RGB_train <- HP_RGB[RGB_subset, ]
RGB_test <- HP_RGB[-RGB_subset, ]

RGC_subset <- sample(nrow(HP_RGC), nrow(HP_RGC) * 0.75)
RGC_train <- HP_RGC[RGC_subset, ]
RGC_test <- HP_RGC[-RGC_subset, ]

HSV_subset <- sample(nrow(HP_HSV), nrow(HP_HSV) * 0.75)
HSV_train <- HP_HSV[HSV_subset, ]
HSV_test <- HP_HSV[-HSV_subset, ]
```

Logistic Regression

```
# RGB
glm(Tarp ~ Red + Green + Blue,
    data = HP_RGB,
```

```
    family = 'binomial') %>%
  summary()
```

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

```
##
## Call:
## glm(formula = Tarp ~ Red + Green + Blue, family = "binomial",
##     data = HP_RGB)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -3.4266  -0.0205  -0.0015   0.0000   3.7911
##
## Coefficients:
##             Estimate Std. Error z value            Pr(>|z|)
## (Intercept)  0.20984    0.18455   1.137               0.256
## Red         -0.26031    0.01262 -20.632 <0.0000000000000002 ***
## Green       -0.21831    0.01330 -16.416 <0.0000000000000002 ***
## Blue         0.47241    0.01548  30.511 <0.0000000000000002 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 17901.6  on 63240  degrees of freedom
## Residual deviance:  1769.5  on 63237  degrees of freedom
## AIC: 1777.5
##
## Number of Fisher Scoring iterations: 12
```

```
# normalized RGB
glm(Tarp ~ Red + Green + Blue,
    data = HP_RGC,
    family = 'binomial') %>%
  summary()
```

```
##
## Call:
## glm(formula = Tarp ~ Red + Green + Blue, family = "binomial",
##     data = HP_RGC)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -3.2135  -0.0118  -0.0026  -0.0007   3.4827
##
## Coefficients: (1 not defined because of singularities)
##             Estimate Std. Error z value            Pr(>|z|)
## (Intercept)  143.226      4.486   31.93 <0.0000000000000002 ***
## Red         -224.309      6.443  -34.82 <0.0000000000000002 ***
## Green       -206.993      8.613  -24.03 <0.0000000000000002 ***
## Blue              NA         NA      NA                   NA
```

```
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 17901.6  on 63240  degrees of freedom
## Residual deviance:  1798.5  on 63238  degrees of freedom
## AIC: 1804.5
##
## Number of Fisher Scoring iterations: 11
```

```
# HSV
glm(Tarp ~ Hue + Saturation + Value,
    data = HP_HSV,
    family = 'binomial') %>%
  summary()
```

```
##
## Call:
## glm(formula = Tarp ~ Hue + Saturation + Value, family = "binomial",
##     data = HP_HSV)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.8651  -0.1598  -0.1178  -0.0918   3.5686
##
## Coefficients:
##             Estimate Std. Error z value            Pr(>|z|)
## (Intercept)   1.8992     0.1370  13.868 <0.0000000000000002 ***
## Hue          -9.2903     0.1380 -67.307 <0.0000000000000002 ***
## Saturation   -0.6450     0.3721  -1.733               0.083 .
## Value         2.1886     0.1328  16.482 <0.0000000000000002 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 17901.6  on 63240  degrees of freedom
## Residual deviance:  8858.7  on 63237  degrees of freedom
## AIC: 8866.7
##
## Number of Fisher Scoring iterations: 8
```

```
## MAKE A FUNCTION TO GIVE RESULTS
# return vector of results
## inputs:
# 1) prediction(model, test$Tarp)
# 2) confusion matrix table (not obtained the same way for every model, can't just feed in the model an
# use deparse(substitute()) to turn object (model name) into a string, derive RGB or HSV from that, and
stats_label <- c('Color', 'AUROC', 'Accuracy',
                 'TPR', 'FPR', 'Precision', 'Model')
model_stats <- function(pred_model, test_y, cmatrix) {
  table_name <- deparse(substitute(cmatrix))
```

```
  Color <- str_split(table_name, '_')[[1]][2]
  AUROC <- performance(prediction(pred_model, test_y),
                       'auc')@y.values[[1]][1]
  Accuracy <- sum(diag(cmatrix)) / sum(cmatrix)
  TPR <- cmatrix[1] / sum(cmatrix[1:2])
  FPR <- cmatrix[3] / sum(cmatrix[3:4])
  Precision <- cmatrix[1] / sum(cmatrix[c(1, 3)])
  Model <- str_split(table_name, '_')[[1]][1] %>%
           toupper()
  return_stats <- c(Color, AUROC, Accuracy,
                    TPR, FPR, Precision, Model)
  names(return_stats) <- stats_label
  return(return_stats)
}
```

```
glm_RGB <- glm(Tarp ~ Red + Green + Blue,
               data = RGB_train,
               family = 'binomial')
```

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

```
glm_RGB_pred <- predict(glm_RGB,
                        RGB_test,
                        type = 'response')

glm_RGC <- glm(Tarp ~ Red + Green,
               data = RGC_train,
               family = 'binomial')
glm_RGC_pred <- predict(glm_RGC,
                        RGC_test,
                        type = 'response')


glm_HSV <- glm(Tarp ~ Hue + Saturation + Value,
               data = HSV_train,
               family = 'binomial')
glm_HSV_pred <- predict(glm_HSV,
                        HSV_test,
                        type = 'response')

glm_RGB_table <- table(ifelse(glm_RGB_pred > 0.5, 1, 0),
                       RGB_test$Tarp)
glm_RGC_table <- table(ifelse(glm_RGC_pred > 0.5, 1, 0),
                       RGC_test$Tarp)
glm_HSV_table <- table(ifelse(glm_HSV_pred > 0.5, 1, 0),
                       HSV_test$Tarp)

glm_RGB_acc <- model_stats(glm_RGB_pred,
                           RGB_test$Tarp,
                           glm_RGB_table)
glm_RGC_acc <- model_stats(glm_RGC_pred,
                           RGC_test$Tarp,
                           glm_RGC_table)
```

```
glm_HSV_acc <- model_stats(glm_HSV_pred,
                           HSV_test$Tarp,
                           glm_HSV_table)

glm_RGB_table
```

```
##
##         0     1
##   0 15272    67
##   1    12   460
```

```
glm_RGC_table
```

```
##
##         0     1
##   0 15296    57
##   1    11   447
```

```
glm_HSV_table
```

```
##
##         0     1
##   0 15170   314
##   1   139   188
```

```
lda_RGB <- lda(Tarp ~ Red + Green + Blue,
               data = RGB_train,
               family = 'binomial')
lda_RGB_pred <- predict(lda_RGB,
                        RGB_test)
lda_RGC <- lda(Tarp ~ Red + Green,
               data = RGC_train,
               family = 'binomial')
lda_RGC_pred <- predict(lda_RGC,
                        RGC_test)

lda_HSV <- lda(Tarp ~ Hue + Saturation + Value,
               data = HSV_train,
               family = 'binomial')
lda_HSV_pred <- predict(lda_HSV,
                        HSV_test)

lda_RGB_table <- table(lda_RGB_pred$class, RGB_test$Tarp)
lda_RGC_table <- table(lda_RGC_pred$class, RGC_test$Tarp)
lda_HSV_table <- table(lda_HSV_pred$class, HSV_test$Tarp)

lda_RGB_acc <- model_stats(lda_RGB_pred$posterior[,2],
                           RGB_test$Tarp,
                           lda_RGB_table)
lda_RGC_acc <- model_stats(lda_RGC_pred$posterior[,2],
                           RGC_test$Tarp,
```

```
                           lda_RGC_table)
lda_HSV_acc <- model_stats(lda_HSV_pred$posterior[,2],
                           HSV_test$Tarp,
                           lda_HSV_table)

lda_RGB_table
```

```
##
##         0     1
##   0 15116    98
##   1   168   429
```

```
lda_RGC_table
```

```
##
##         0     1
##   0 15300    59
##   1     7   445
```

```
lda_HSV_table
```

```
##
##         0     1
##   0 15166    19
##   1   143   483
```

```
qda_RGB <- qda(Tarp ~ Red + Green + Blue,
               data = RGB_train,
               family = 'binomial')
qda_RGB_pred <- predict(qda_RGB,
                        RGB_test)
qda_RGC <- qda(Tarp ~ Red + Green,
               data = RGC_train,
               family = 'binomial')
qda_RGC_pred <- predict(qda_RGC,
                        RGC_test)
qda_HSV <- qda(Tarp ~ Hue + Saturation + Value,
               data = HSV_train,
               family = 'binomial')
qda_HSV_pred <- predict(qda_HSV,
                        HSV_test)

qda_RGB_table <- table(qda_RGB_pred$class, RGB_test$Tarp)
qda_RGC_table <- table(qda_RGC_pred$class, RGC_test$Tarp)
qda_HSV_table <- table(qda_HSV_pred$class, HSV_test$Tarp)

qda_RGB_acc <- model_stats(qda_RGB_pred$posterior[, 2],
                           RGB_test$Tarp,
                           qda_RGB_table)
qda_RGC_acc <- model_stats(qda_RGC_pred$posterior[, 2],
                           RGC_test$Tarp,
```

```
                           qda_RGC_table)
qda_HSV_acc <- model_stats(qda_HSV_pred$posterior[, 2],
                           HSV_test$Tarp,
                           qda_HSV_table)

qda_RGB_table
```

```
##
##         0     1
##   0 15281    87
##   1     3   440
```

```
qda_RGC_table
```

```
##
##         0     1
##   0 15233    48
##   1    74   456
```

```
qda_HSV_table
```

```
##
##         0     1
##   0 15160    17
##   1   149   485
```

```
knn_RGB <- knn(train = as.matrix(RGB_train[, 1:3]),
                       test = as.matrix(RGB_test[, 1:3]),
                       cl = as.matrix(RGB_train$Tarp),
                       k = 1, prob = TRUE)
knn_RGC <- knn(train = as.matrix(RGC_train[, 1:2]),
                       test = as.matrix(RGC_test[, 1:2]),
                       cl = as.matrix(RGC_train$Tarp),
                       k = 1, prob = TRUE)
knn_HSV <- knn(train = as.matrix(HSV_train[, 1:3]),
                       test = as.matrix(HSV_test[, 1:3]),
                       cl = as.matrix(HSV_train$Tarp),
                       k = 1, prob = TRUE)

knn_RGB_table <- table(knn_RGB, RGB_test$Tarp)
knn_RGC_table <- table(knn_RGC, RGC_test$Tarp)
knn_HSV_table <- table(knn_HSV, HSV_test$Tarp)

knn_RGB_acc <- model_stats(as.numeric(knn_RGB),
                           RGB_test$Tarp,
                           knn_RGB_table)
knn_RGC_acc <- model_stats(as.numeric(knn_RGC),
                           RGC_test$Tarp,
                           knn_RGC_table)
knn_HSV_acc <- model_stats(as.numeric(knn_HSV),
                           HSV_test$Tarp,
```

```
                          knn_HSV_table)

knn_RGB_table
```

```
##
## knn_RGB     0     1
##       0 15264    33
##       1    20   494
```

```
knn_RGC_table
```

```
##
## knn_RGC     0     1
##       0 15291    38
##       1    16   466
```

```
knn_HSV_table
```

```
##
## knn_HSV     0     1
##       0 15286    28
##       1    23   474
```

```
# selecting best-k
# for loop, trying different K-values for KNN
# try k-values between 1 and 25
# create 50 row data frame for RGB and HSV values
bestk_DF <- data.frame(matrix(ncol = 7, nrow = 75))

# alternate rows in loops
for (i in seq(1, nrow(bestk_DF), 3)) {
  # k-value is 1:25
  k_value <- ceiling(i / 3)

  ## knn for RGB values
  loop_RGB <- knn(train = as.matrix(RGB_train[, 1:3]),
                  test = as.matrix(RGB_test[, 1:3]),
                  cl = as.matrix(RGB_train$Tarp),
                  k = k_value, prob = TRUE)
  knnbestk_RGB_table <- table(loop_RGB, RGB_test$Tarp)

  ## knn for normalized RGB values
  loop_RGC <- knn(train = as.matrix(RGC_train[, 1:2]),
                  test = as.matrix(RGC_test[, 1:2]),
                  cl = as.matrix(RGC_train$Tarp),
                  k = k_value, prob = TRUE)
  knnbestk_RGC_table <- table(loop_RGC, RGC_test$Tarp)

  ## knn for HSV values
  loop_HSV <- knn(train = as.matrix(HSV_train[, 1:3]),
                  test = as.matrix(HSV_test[, 1:3]),
```

```
                  cl = as.matrix(HSV_train$Tarp),
                  k = k_value, prob = TRUE)
  knnbestk_HSV_table <- table(loop_HSV, HSV_test$Tarp)

  bestk_DF[i, ] <- model_stats(as.numeric(loop_RGB), RGB_test$Tarp, knnbestk_RGB_table)
  bestk_DF[i + 1, ] <- model_stats(as.numeric(loop_RGC), RGC_test$Tarp, knnbestk_RGC_table)
  bestk_DF[i + 2, ] <- model_stats(as.numeric(loop_HSV), HSV_test$Tarp, knnbestk_HSV_table)
}
```

```
## data cleaning for best-k, RGB
# make new data frame, as so not require re-running of loop
bestk_RGB <- bestk_DF[which(bestk_DF$X1 == 'RGB'), ]
colnames(bestk_RGB) <- stats_label
bestk_RGB$K <- ceiling(as.numeric(row.names(bestk_RGB)) / 2)
bestk_RGB[, 2:6] <- sapply(bestk_RGB[, 2:6], as.numeric)

# percentile ranking of values
bestk_RGB$AUROCpct <- percent_rank(bestk_RGB$AUROC)
bestk_RGB$Accuracypct <- percent_rank(bestk_RGB$Accuracy)
bestk_RGB$TPRpct <- percent_rank(bestk_RGB$TPR)
bestk_RGB$FPRpct <- percent_rank(bestk_RGB$FPR)
bestk_RGB$Precisionpct <- percent_rank(bestk_RGB$Precision)

# variable for best k-values, decided by high percentile metric
# set equal to the k tuning parameter for graphing purposes
bestk_RGB$avg_stat <- (bestk_RGB$AUROC * 1.2 +
                       bestk_RGB$Accuracy * 1.2 +
                       bestk_RGB$TPR * 2.4 +
                       bestk_RGB$FPR * -0.2 +
                       bestk_RGB$Precision * 0.2) / 5

# have best-k be top 10th percentile
bestk_RGB$best_k <- ifelse(bestk_RGB$avg_stat >=
                           quantile(bestk_RGB$avg_stat, 0.80),
                           bestk_RGB$K, 0)

## view data with best k-values, sorted
# k = 7
subset(bestk_RGB[order(-bestk_RGB$avg_stat), ],
       best_k > 0,
       select = c(AUROC:Precision, best_k))
```

```
##        AUROC  Accuracy       TPR        FPR Precision best_k
## 13 0.9756266 0.9971539 0.9986914 0.04743833 0.9983648      7
## 10 0.9746778 0.9970906 0.9986914 0.04933586 0.9982995      5
## 28 0.9746778 0.9970906 0.9986914 0.04933586 0.9982995     14
## 7  0.9737945 0.9971539 0.9988223 0.05123340 0.9982345      4
## 40 0.9728784 0.9971539 0.9988877 0.05313093 0.9981693     20
```

```
bestk_RGB_acc <- subset(bestk_RGB[order(-bestk_RGB$avg_stat), ],
                        best_k > 0,
                        select = c(Color:Model))[1, ]
```

```r
## data cleaning for best-k, normalized RGB
# make new data frame, as so not require re-running of loop
bestk_RGC <- bestk_DF[which(bestk_DF$X1 == 'RGC'), ]
colnames(bestk_RGC) <- stats_label
bestk_RGC$K <- ceiling(as.numeric(row.names(bestk_RGC)) / 2)
bestk_RGC[, 2:6] <- sapply(bestk_RGC[, 2:6], as.numeric)

# percentile ranking of values
bestk_RGC$AUROCpct <- percent_rank(bestk_RGC$AUROC)
bestk_RGC$Accuracypct <- percent_rank(bestk_RGC$Accuracy)
bestk_RGC$TPRpct <- percent_rank(bestk_RGC$TPR)
bestk_RGC$FPRpct <- percent_rank(bestk_RGC$FPR)
bestk_RGC$Precisionpct <- percent_rank(bestk_RGC$Precision)

# variable for best k-values, decided by high percentile metric
# set equal to the k tuning parameter for graphing purposes
bestk_RGC$avg_stat <- (bestk_RGC$AUROC * 1.2 +
                       bestk_RGC$Accuracy * 1.2 +
                       bestk_RGC$TPR * 2.4 +
                       bestk_RGC$FPR * -0.2 +
                       bestk_RGC$Precision * 0.2) / 5

# have best-k be top 10th percentile
bestk_RGC$best_k <- ifelse(bestk_RGC$avg_stat >=
                           quantile(bestk_RGC$avg_stat, 0.80),
                           bestk_RGC$K, 0)

## view datawith best k-values, sorted
# k = 15
subset(bestk_RGC[order(-bestk_RGC$avg_stat), ],
       best_k > 0,
       select = c(AUROC:Precision, best_k))
```

```
##        AUROC  Accuracy       TPR        FPR Precision best_k
## 35 0.9739124 0.9977864 0.9994120 0.05158730 0.9983033     18
## 26 0.9738797 0.9977231 0.9993467 0.05158730 0.9983032     13
## 32 0.9728876 0.9976599 0.9993467 0.05357143 0.9982381     16
## 38 0.9728876 0.9976599 0.9993467 0.05357143 0.9982381     19
## 41 0.9728876 0.9976599 0.9993467 0.05357143 0.9982381     21
```

```r
bestk_RGC_acc <- subset(bestk_RGC[order(-bestk_RGC$avg_stat), ],
                        best_k > 0,
                        select = c(Color:Model))[1, ]
```

```r
## data cleaning for best-k, HSV
# make new data frame, as so not require re-running of loop
bestk_HSV <- bestk_DF[which(bestk_DF$X1 == 'HSV'), ]
colnames(bestk_HSV) <- stats_label
bestk_HSV$K <- ceiling(as.numeric(row.names(bestk_HSV)) / 2)
bestk_HSV[, 2:6] <- sapply(bestk_HSV[, 2:6], as.numeric)

# percentile ranking of values
bestk_HSV$AUROCpct <- percent_rank(bestk_HSV$AUROC)
```

```
bestk_HSV$Accuracypct <- percent_rank(bestk_HSV$Accuracy)
bestk_HSV$TPRpct <- percent_rank(bestk_HSV$TPR)
bestk_HSV$FPRpct <- percent_rank(bestk_HSV$FPR)
bestk_HSV$Precisionpct <- percent_rank(bestk_HSV$Precision)

# variable for best k-values, decided by high percentile metric
# set equal to the k tuning parameter for graphing purposes
bestk_HSV$avg_stat <- (bestk_HSV$AUROC * 1.2 +
                       bestk_HSV$Accuracy * 1.2 +
                       bestk_HSV$TPR * 2.4 +
                       bestk_HSV$FPR * -0.2 +
                       bestk_HSV$Precision * 0.2) / 5

# have best-k be top 10th percentile
bestk_HSV$best_k <- ifelse(bestk_HSV$avg_stat >=
                           quantile(bestk_HSV$avg_stat, 0.80),
                           bestk_HSV$K, 0)

## view data with best k-values, sorted
# k = 23
subset(bestk_HSV[order(-bestk_HSV$avg_stat), ],
       best_k > 0,
       select = c(AUROC:Precision, best_k))
```

```
##        AUROC  Accuracy       TPR        FPR Precision best_k
## 21 0.9783978 0.9973436 0.9986283 0.04183267 0.9986283     11
## 39 0.9783978 0.9973436 0.9986283 0.04183267 0.9986283     20
## 45 0.9783978 0.9973436 0.9986283 0.04183267 0.9986283     23
## 30 0.9774998 0.9974701 0.9988242 0.04382470 0.9985633     15
## 15 0.9774344 0.9973436 0.9986936 0.04382470 0.9985631      8
## 18 0.9774344 0.9973436 0.9986936 0.04382470 0.9985631      9
## 36 0.9774344 0.9973436 0.9986936 0.04382470 0.9985631     18
```

```
bestk_HSV_acc <- subset(bestk_HSV[order(-bestk_HSV$avg_stat), ],
                        best_k > 0,
                        select = c(Color:Model))[1, ]
```

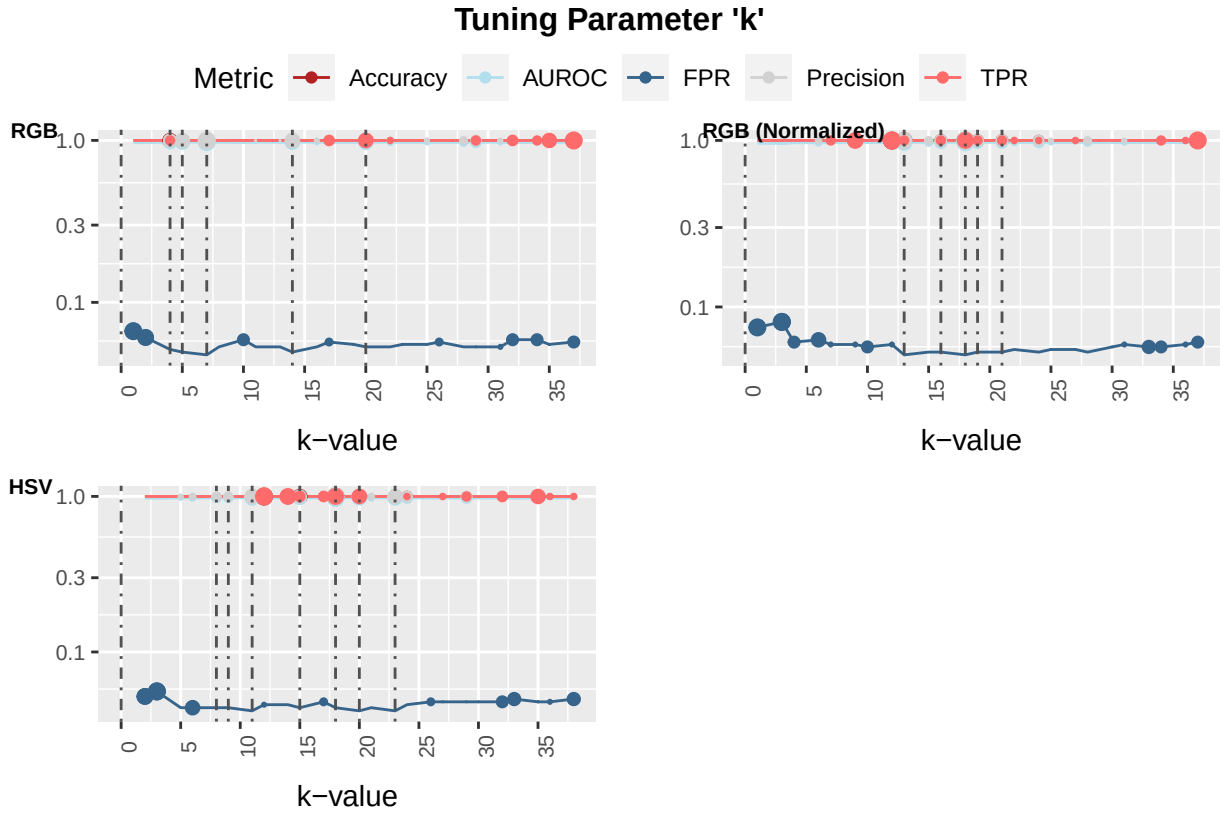
```
# transforming data for tuning parameter k for visualization
bestk_viz <- merge(rbind(bestk_HSV, bestk_RGC, bestk_RGB)[, c(1:6, 8, 15)] %>%
                     pivot_longer(cols = AUROC:Precision,
                                  names_to = 'type'),
                   rbind(bestk_HSV, bestk_RGC, bestk_RGB)[, c(8:13, 15)] %>%
                     pivot_longer(cols = AUROCpct:Precisionpct,
                                  names_to = 'type',
                                  names_pattern = '(\\w+)pct',
                                  values_to = 'pct'),
                   by = c('type', 'best_k', 'K'))
```

```
## ggplot for best-k knn tuning
RGB_bestk_plot <- ggplot(bestk_viz[which(bestk_viz$Color == 'RGB'),],
                         aes(x = K, color = type)) +
                  geom_line(aes(y = value)) +
```

```
                  geom_point(aes(y = value, size = pct)) +
                  scale_x_continuous(breaks = 0:20 * 5) +
                  scale_y_continuous(name = '',
                                     trans = 'log10') +
                  scale_size(range = c(-7.5, 2.5), guide = FALSE) +
                  geom_vline(aes(xintercept = best_k),
                             color = '#515151',
                             linetype = 'dotdash',
                             size = 0.5) +
                  scale_color_manual(values = c('firebrick', 'lightblue2',
                                                'steelblue4', 'grey82',
                                                'indianred1')) +
                  labs(x = 'k-value', color = 'Metric') +
                    theme(plot.title = element_text(size = 18),
                          plot.subtitle = element_text(size = 8,
                                                       margin = ggplot2::margin(t = 2.5, r = 0, b = 7.5
                          axis.text.x = element_text(size = 7.5, angle = 90),
                          axis.text.y = element_text(size = 7.5),
                          axis.title.x = element_text(margin = ggplot2::margin(t = 10, r = 0, b = 0, l =
                          axis.title.y.left = element_text(margin = ggplot2::margin(t = 0, r = 10, b = 0

RGC_bestk_plot <- ggplot(bestk_viz[which(bestk_viz$Color == 'RGC'),],
                       aes(x = K, color = type)) +
                  geom_line(aes(y = value)) +
                  geom_point(aes(y = value, size = pct)) +
                  scale_x_continuous(breaks = 0:20 * 5) +
                  scale_y_continuous(name = '',
                                     trans = 'log10') +
                  scale_size(range = c(-7.5, 2.5), guide = FALSE) +
                  geom_vline(aes(xintercept = best_k),
                             color = '#515151',
                             linetype = 'dotdash',
                             size = 0.5) +
                  scale_color_manual(values = c('firebrick', 'lightblue2',
                                                'steelblue4', 'grey82',
                                                'indianred1')) +
                  labs(x = 'k-value', color = 'Metric') +
                    theme(plot.title = element_text(size = 18),
                          plot.subtitle = element_text(size = 8,
                                                       margin = ggplot2::margin(t = 2.5, r = 0, b = 7.5
                          axis.text.x = element_text(size = 7.5, angle = 90),
                          axis.text.y = element_text(size = 7.5),
                          axis.title.x = element_text(margin = ggplot2::margin(t = 10, r = 0, b = 0, l =
                          axis.title.y.left = element_text(margin = ggplot2::margin(t = 0, r = 10, b = 0

HSV_bestk_plot <- ggplot(bestk_viz[which(bestk_viz$Color == 'HSV'),],
                       aes(x = K, color = type)) +
                  geom_line(aes(y = value)) +
                  geom_point(aes(y = value, size = pct)) +
                  scale_x_continuous(breaks = 0:20 * 5) +
                  scale_y_continuous(name = '',
                                     trans = 'log10') +
                  scale_size(range = c(-7.5, 2.5), guide = FALSE) +
```

```
                  geom_vline(aes(xintercept = best_k),
                             color = '#515151',
                             linetype = 'dotdash',
                             size = 0.5) +
                  scale_color_manual(values = c('firebrick', 'lightblue2',
                                                'steelblue4', 'grey82',
                                                'indianred1')) +
                  labs(x = 'k-value', color = 'Metric') +
                    theme(plot.title = element_text(size = 18),
                          plot.subtitle = element_text(size = 8,
                                                       margin = ggplot2::margin(t = 2.5, r = 0, b = 7.5
                          axis.text.x = element_text(size = 7.5, angle = 90),
                          axis.text.y = element_text(size = 7.5),
                          axis.title.x = element_text(margin = ggplot2::margin(t = 10, r = 0, b = 0, l =
                          axis.title.y.left = element_text(margin = ggplot2::margin(t = 0, r = 10, b = 0
```

```
annotate_figure(ggarrange(RGB_bestk_plot, RGC_bestk_plot, HSV_bestk_plot,
                          ncol = 2, nrow = 2, common.legend = TRUE,
                          labels = c('RGB', 'RGB (Normalized)', 'HSV'),
                          font.label = list(size = 7.5)),
                top = text_grob('Tuning Parameter \'k\'', face = 'bold', size = 12),
                bottom = text_grob('Dot size represents value percentile, relative to metrics for other
                                   face = 'italic', size = 7.5))
```



Dot size represents value percentile, relative to metrics for other values 'k'

Penalized Logistic Regression (ElasticNet)

```
# force data into matrices
RGB_matrix_X <- model.matrix(Tarp ~ poly(Red, 4, raw = TRUE) +
                                    poly(Green, 4, raw = TRUE) +
                                    poly(Blue, 4, raw = TRUE),
                             data = HP_RGB)[,-1]
RGB_matrix_Y <- HP_RGB$Tarp %>%
                    as.numeric() %>%
                    as.matrix()

RGC_matrix_X <- model.matrix(Tarp ~ poly(Red, 4, raw = TRUE) +
                                    poly(Green, 4, raw = TRUE),
                             data = HP_RGC)[,-1]
RGC_matrix_Y <- HP_RGC$Tarp %>%
                    as.numeric() %>%
                    as.matrix()

HSV_matrix_X <- model.matrix(Tarp ~ poly(Hue, 4, raw = TRUE) +
                                    poly(Saturation, 4, raw = TRUE) +
                                    poly(Value, 4, raw = TRUE),
                             data = HP_HSV)[,-1]
HSV_matrix_Y <- HP_HSV$Tarp %>%
                    as.numeric() %>%
                    as.matrix()
```

```
## normalized RGB values, penalized regression
## standard cv glm provided as a reference for comparison
## setting up parallel
## use 'parallel' package to help with training times

## allow for parallelism, setup new cluster
reg_cluster <- parallel::makeCluster(detectCores() - 2,
                                     setup_strategy = 'sequential')
registerDoParallel(reg_cluster)

## training
## RGB values, penalized regression
RGB_cvglm <- cv.glmnet(RGB_matrix_X,
                       RGB_matrix_Y,
                       family = 'binomial',
                       nfolds = 10,
                       parallel = TRUE)

RGB_ridge <- cv.glmnet(RGB_matrix_X,
                       RGB_matrix_Y,
                       lambda = 10^seq(10, -2, length = 100),
                       alpha = 0,
                       nfolds = 10,
                       parallel = TRUE)

RGB_lasso <- cv.glmnet(RGB_matrix_X,
                       RGB_matrix_Y,
                       lambda = 10^seq(10, -2, length = 100),
                       alpha = 1,
```

```
                        nfolds = 10,
                        parallel = TRUE)
## normalized RGB values, penalized regression
RGC_cvglm <- cv.glmnet(RGC_matrix_X,
                        RGC_matrix_Y,
                        family = 'binomial',
                        nfolds = 10,
                        parallel = TRUE)

RGC_ridge <- cv.glmnet(RGC_matrix_X,
                        RGC_matrix_Y,
                        lambda = 10^seq(10, -2, length = 100),
                        alpha = 0,
                        nfolds = 10,
                        parallel = TRUE)

RGC_lasso <- cv.glmnet(RGC_matrix_X,
                        RGC_matrix_Y,
                        lambda = 10^seq(10, -2, length = 100),
                        alpha = 1,
                        nfolds = 10,
                        parallel = TRUE)
## HSV values, penalized regression
HSV_cvglm <- cv.glmnet(HSV_matrix_X,
                        HSV_matrix_Y,
                        family = 'binomial',
                        nfolds = 10,
                        parallel = TRUE)

HSV_ridge <- cv.glmnet(HSV_matrix_X,
                        HSV_matrix_Y,
                        lambda = 10^seq(10, -2, length = 100),
                        alpha = 0,
                        nfolds = 10,
                        parallel = TRUE)

HSV_lasso <- cv.glmnet(HSV_matrix_X,
                        HSV_matrix_Y,
                        lambda = 10^seq(10, -2, length = 100),
                        alpha = 1,
                        nfolds = 10,
                        parallel = TRUE)
# end cluster
stopCluster(reg_cluster)
```

```
## predict on test data
cv_reg_list <- list(RGB_cvglm, RGB_ridge, RGB_lasso,
                    RGC_cvglm, RGC_ridge, RGC_lasso,
                    HSV_cvglm, HSV_ridge, HSV_lasso)
cv_newx_list <- list(RGB_matrix_X[-RGB_subset, ],
                     RGB_matrix_X[-RGB_subset, ],
                     RGB_matrix_X[-RGB_subset, ],
                     RGC_matrix_X[-RGC_subset, ],
```

```
                     RGC_matrix_X[-RGC_subset, ],
                     RGC_matrix_X[-RGC_subset, ],
                     HSV_matrix_X[-HSV_subset, ],
                     HSV_matrix_X[-HSV_subset, ],
                     HSV_matrix_X[-HSV_subset, ])

# 9-column matrix, predicted values per column
# in order: 3/ea RGB, RGC, HSV of cvglm, ridge, lasso
cv_reg_pred0 <- mapply(function(x, y)
                       predict(x, newx = y,
                               s = x$lambda.min),
                       cv_reg_list, cv_newx_list)

# thresholds for finding predicted values
cv_reg_thresholds <- (apply(cv_reg_pred0, 2, max) +
                      apply(cv_reg_pred0, 2, min)) / 2

# 9-column matrix, fit for table
cv_reg_pred1 <- c()
for (i in 1:ncol(cv_reg_pred0)) {
  cv_reg_predvals <- unlist(lapply(cv_reg_pred0[, i],
                                   function(x)
                                   ifelse(x >= cv_reg_thresholds[i], 1, 0)))
  cv_reg_pred1 <- cbind(cv_reg_pred1, cv_reg_predvals)
}
```

```
# MSE values
mean((RGB_matrix_Y[-RGB_subset, ] - cv_reg_pred0[, 1])^2)
```

```
## [1] 531.6031
```

```
mean((RGB_matrix_Y[-RGB_subset, ] - cv_reg_pred0[, 2])^2)
```

```
## [1] 0.01872933
```

```
mean((RGB_matrix_Y[-RGB_subset, ] - cv_reg_pred0[, 3])^2)
```

```
## [1] 0.02088232
```

```
mean((RGC_matrix_Y[-RGC_subset, ] - cv_reg_pred0[, 4])^2)
```

```
## [1] 176.8053
```

```
mean((RGC_matrix_Y[-RGC_subset, ] - cv_reg_pred0[, 5])^2)
```

```
## [1] 0.01657143
```

```r
mean((RGC_matrix_Y[-RGC_subset, ] - cv_reg_pred0[, 6])^2)
```

```
## [1] 0.01881276
```

```r
mean((HSV_matrix_Y[-HSV_subset, ] - cv_reg_pred0[, 7])^2)
```

```
## [1] 66.04885
```

```r
mean((HSV_matrix_Y[-HSV_subset, ] - cv_reg_pred0[, 8])^2)
```

```
## [1] 0.01442848
```

```r
mean((HSV_matrix_Y[-HSV_subset, ] - cv_reg_pred0[, 9])^2)
```

```
## [1] 0.01561427
```

```r
# shrinkage methods show lower mean squared error
```

```r
# RGB stats
cvglm_RGB_table <- table(cv_reg_pred1[, 1], RGB_test$Tarp)
cvglm_RGB_table
```

```
##
##        0    1
##   0 9812    0
##   1 5472  527
```

```r
ridge_RGB_table <- table(cv_reg_pred1[, 2], RGB_test$Tarp)
ridge_RGB_table
```

```
##
##         0     1
##   0 15067   106
##   1   217   421
```

```r
lasso_RGB_table <- table(cv_reg_pred1[, 3], RGB_test$Tarp)
lasso_RGB_table
```

```
##
##         0     1
##   0 15062   179
##   1   222   348
```

```r
cvglm_RGB_acc <- model_stats(cv_reg_pred0[, 1], RGB_test$Tarp, cvglm_RGB_table)
ridge_RGB_acc <- model_stats(cv_reg_pred0[, 2], RGB_test$Tarp, ridge_RGB_table)
lasso_RGB_acc <- model_stats(cv_reg_pred0[, 3], RGB_test$Tarp, lasso_RGB_table)
```

```
# normalized RGB stats
cvglm_RGC_table <- table(cv_reg_pred1[, 4], RGC_test$Tarp)
cvglm_RGC_table
```

```
##
##         0     1
##   0 14967     2
##   1   340   502
```

```
ridge_RGC_table <- table(cv_reg_pred1[, 5], RGC_test$Tarp)
ridge_RGC_table
```

```
##
##         0     1
##   0 15207    33
##   1   100   471
```

```
lasso_RGC_table <- table(cv_reg_pred1[, 6], RGC_test$Tarp)
lasso_RGC_table
```

```
##
##         0     1
##   0 14626     2
##   1   681   502
```

```
cvglm_RGC_acc <- model_stats(cv_reg_pred0[, 4], RGC_test$Tarp, cvglm_RGC_table)
ridge_RGC_acc <- model_stats(cv_reg_pred0[, 5], RGC_test$Tarp, ridge_RGC_table)
lasso_RGC_acc <- model_stats(cv_reg_pred0[, 6], RGC_test$Tarp, lasso_RGC_table)
```

```
# HSV stats
cvglm_HSV_table <- table(cv_reg_pred1[, 7], HSV_test$Tarp)
cvglm_HSV_table
```

```
##
##         0     1
##   0 15281    39
##   1    28   463
```

```
ridge_HSV_table <- table(cv_reg_pred1[, 8], HSV_test$Tarp)
ridge_HSV_table
```

```
##
##         0     1
##   0 15152    13
##   1   157   489
```

```
lasso_HSV_table <- table(cv_reg_pred1[, 9], HSV_test$Tarp)
lasso_HSV_table
```

```
##
##         0     1
##   0 15150    11
##   1   159   491
```

```
cvglm_HSV_acc <- model_stats(cv_reg_pred0[, 7], HSV_test$Tarp, cvglm_HSV_table)
ridge_HSV_acc <- model_stats(cv_reg_pred0[, 8], HSV_test$Tarp, ridge_HSV_table)
lasso_HSV_acc <- model_stats(cv_reg_pred0[, 9], HSV_test$Tarp, lasso_HSV_table)
```

Tuning Parameters

KNN Tuning When using the k-nearest neighbors algorithm, subset selection, Ridge regression, and Lasso regression, various parameters were tuned in order to find a 'best model'. For the k-nearest neighbors algorithm (knn), I ran a loop from 1 to 100, calculating AUROC, accuracy, true positive rate, false positive rate, and precision for each value of k. A 'best-k' was then selected, based upon a 'k-value that could produce a model with high overall values for each metric, weighted at a 6:6:12:-1:1 ratio for AUROC, accuracy, true positive rate, false positive rate, and precision, respectively. A negative weighting was chosen, as a lower false positive rating is obviously more desirable, however, the magnitude of the weighting is still small, as I believe that in a well-funded rescue scenario such as that in Haiti, it is less important to avoid false positives than it is to be thorough in searching. Additionally, while blue tarps are an indicator of survivors seeking rescue, the *lack* of a blue tarp does not guarantee the lack of someone in need of help.

Ultimately, k-value of 7, 15, and 23 were best for both RGB, normalized RGB, and HSV values, respectively.

Penalized Logistic Regression Tuning For penalized logistic regression, I ultimately chose to show results from a cross-validated logistic regression alongside ridge and lasso regression, to show the effects of various shrinkage methods. For these methods, I chose to expand the original regressions - `Tarp ~ Red + Green + Blue` for RGB values, `Tarp ~ Red + Green` for normalized RGB values, and `Tarp ~ Hue + Saturation + Value` were transformed into a polynomial regressions with degrees up to four, in order to see which variables would be shrunk or eliminated.

Both lasso and ridge regression showed lower mean squared error values (MSE) than a standard cross-validated logistic regression did, for both normalized RGB and HSV values.

Mean Squared Error values:

- CV Logistic regression, RGB: 1.014621
- Ridge regression, RGB: 0.01872933
- Lasso regression, RGB: 0.02088232
- CV Logistic regression, normalized RGB: 1.004663
- Ridge regression, normalized RGB: 0.01657143
- Lasso regression, normalized RGB: 0.01881276
- CV Logistic regression, HSV: 1.010136
- Ridge regression, HSV: 0.01442848
- Lasso regression, HSV: 0.01561427

Models used for Penalized Logistic Regression:

- RGB: `Tarp ~ Red + Red^2 + Red^3 + Red^4 + Green + Green^2 + Green^3 + Green^4 + Blue + Blue^2 + Blue^3 + Blue^4`
- Normalized RGB: `Tarp ~ Red + Red^2 + Red^3 + Red^4 + Green + Green^2 + Green^3 + Green^4`

- HSV: `Tarp ~ Hue + Hue^2 + Hue^3 + Hue^4 + Saturation + Saturation^2 + Saturation^3 + Saturation^4 + Value + Value^2 + Value^3 + Value^4`

When analyzing the results, ridge regression yielded the best best results. However, these differences in performance are overall not very considerable between (cross-validated) logistic, lasso, and ridge regression, and the standardized logistic model without any polynomial or interaction variables ended up outperforming shrinkage methods.

For all penalized logistic regression methods, I chose to produce confusion matrices by comparing t he predicted values to a threshold value that sits in between the minimum and maximum values of the predicted values. In these confusion matrices, models using HSV color values had overall lower true negative values, false positive, and false negative values. Of these shrinkage methods, ridge regression seemed to yield the best results.

Performance Table

```
perf_table <- rbind(glm_RGB_acc, lda_RGB_acc,
                    qda_RGB_acc, knn_RGB_acc,
                    bestk_RGB_acc, cvglm_RGB_acc,
                    ridge_RGB_acc, lasso_RGB_acc,
                    glm_RGC_acc, lda_RGC_acc,
                    qda_RGC_acc, knn_RGC_acc,
                    bestk_RGC_acc, cvglm_RGC_acc,
                    ridge_RGC_acc, lasso_RGC_acc,
                    glm_HSV_acc, lda_HSV_acc,
                    qda_HSV_acc, knn_HSV_acc,
                    bestk_HSV_acc, cvglm_HSV_acc,
                    ridge_HSV_acc, lasso_HSV_acc) %>%
  data.frame()

perf_table[, 2:6] <- lapply(perf_table[, 2:6],
                            as.numeric)
perf_table[, 2:6] <- perf_table[, 2:6] %>%
                         round(4)

perf_table <- perf_table[order(-(perf_table$AUROC * 1.2 +
                                 perf_table$Accuracy * 1.2 +
                                 perf_table$TPR * 2.4 +
                                 perf_table$FPR * -0.2 +
                                 perf_table$Precision * 0.2) / 5), ]

rownames(perf_table) <- NULL
perf_table
```

```
##    Color  AUROC Accuracy    TPR    FPR Precision    Model
## 1    HSV 0.9990   0.9958 0.9982 0.0777    0.9975    CVGLM
## 2    RGC 0.9990   0.9957 0.9993 0.1131    0.9963      GLM
## 3    RGC 0.9989   0.9958 0.9995 0.1171    0.9962      LDA
## 4    RGB 0.9983   0.9950 0.9992 0.1271    0.9956      GLM
## 5    RGC 0.9988   0.9916 0.9935 0.0655    0.9978    RIDGE
## 6    HSV 0.9784   0.9973 0.9986 0.0418    0.9986 KNNBESTK
## 7    RGC 0.9979   0.9923 0.9952 0.0952    0.9969      QDA
```

```
## 8    RGB 0.9983   0.9943 0.9998 0.1651    0.9943      QDA
## 9    RGB 0.9756   0.9972 0.9987 0.0474    0.9984 KNNBESTK
## 10   RGC 0.9739   0.9978 0.9994 0.0516    0.9983 KNNBESTK
## 11   HSV 0.9714   0.9968 0.9985 0.0558    0.9982      KNN
## 12   HSV 0.9872   0.9895 0.9903 0.0339    0.9989      QDA
## 13   RGB 0.9680   0.9966 0.9987 0.0626    0.9978      KNN
## 14   RGC 0.9618   0.9966 0.9990 0.0754    0.9975      KNN
## 15   HSV 0.9770   0.9892 0.9896 0.0219    0.9993    LASSO
## 16   HSV 0.9769   0.9898 0.9907 0.0378    0.9987      LDA
## 17   HSV 0.9771   0.9892 0.9897 0.0259    0.9991    RIDGE
## 18   RGC 0.9990   0.9784 0.9778 0.0040    0.9999    CVGLM
## 19   RGB 0.9883   0.9832 0.9890 0.1860    0.9936      LDA
## 20   RGB 0.9769   0.9796 0.9858 0.2011    0.9930    RIDGE
## 21   RGC 0.9989   0.9568 0.9555 0.0040    0.9999    LASSO
## 22   RGB 0.9647   0.9746 0.9855 0.3397    0.9883    LASSO
## 23   HSV 0.9774   0.9713 0.9909 0.6255    0.9797      GLM
## 24   RGB 0.9996   0.6539 0.6420 0.0000    1.0000    CVGLM
```

```
## grouped performance statistics
perf_table %>%
  group_by(Color) %>%
  summarize(mAUROC = mean(AUROC), mAcc = mean(Accuracy),
            mTPR = mean(TPR), mFPR = mean(FPR),
            mPrc = mean(Precision))
```

```
## # A tibble: 3 x 6
##   Color mAUROC  mAcc  mTPR   mFPR  mPrc
##   <chr>  <dbl> <dbl> <dbl>  <dbl> <dbl>
## 1 HSV    0.981 0.990 0.993 0.115  0.996
## 2 RGB    0.984 0.947 0.950 0.141  0.995
## 3 RGC    0.991 0.988 0.990 0.0657 0.998
```

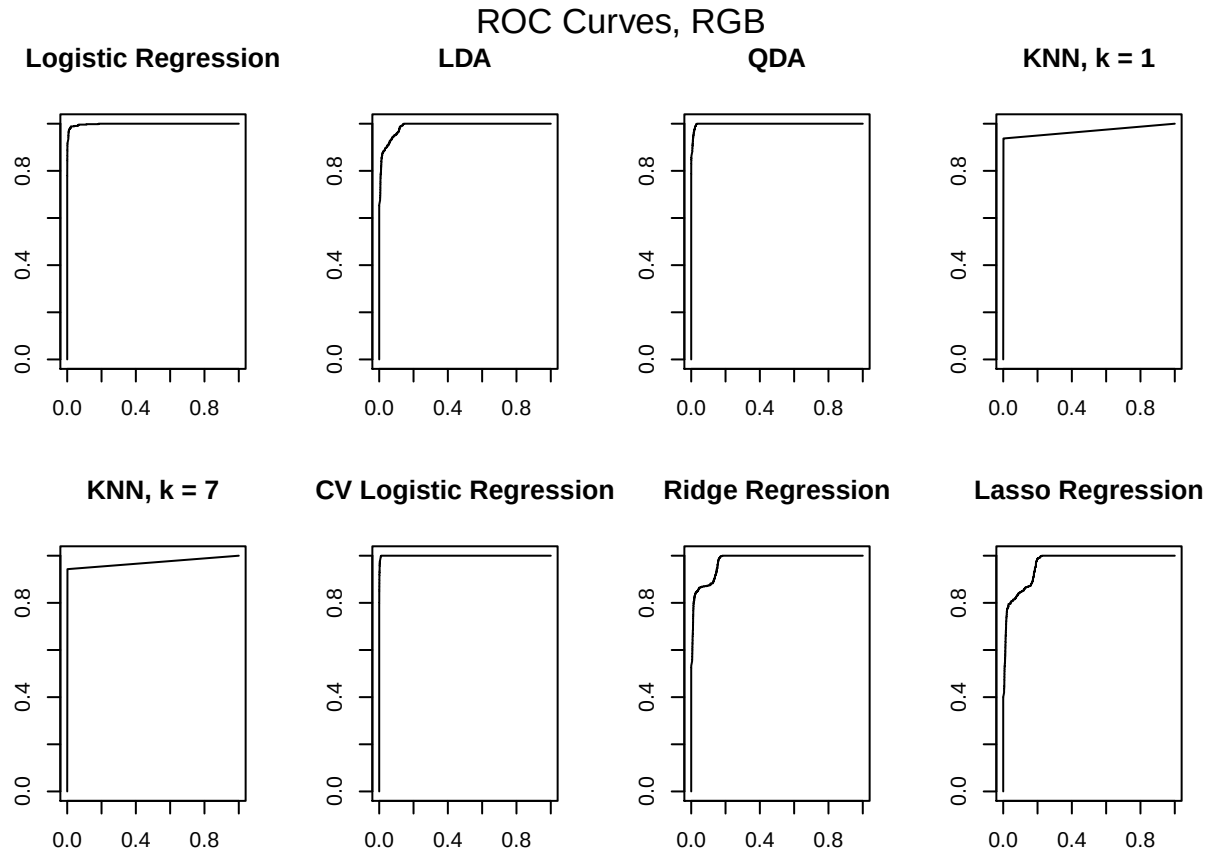
```
perf_table %>%
  group_by(Model) %>%
  summarize(mAUROC = mean(AUROC), mAcc = mean(Accuracy),
            mTPR = mean(TPR), mFPR = mean(FPR),
            mPrc = mean(Precision))
```

```
## # A tibble: 8 x 6
##   Model    mAUROC  mAcc  mTPR   mFPR  mPrc
##   <chr>     <dbl> <dbl> <dbl>  <dbl> <dbl>
## 1 CVGLM     0.999 0.876 0.873 0.0272 0.999
## 2 GLM       0.992 0.987 0.996 0.289  0.991
## 3 KNN       0.967 0.997 0.999 0.0646 0.998
## 4 KNNBESTK  0.976 0.997 0.999 0.0469 0.998
## 5 LASSO     0.980 0.974 0.977 0.122  0.996
## 6 LDA       0.988 0.990 0.993 0.114  0.996
## 7 QDA       0.994 0.992 0.995 0.0981 0.997
## 8 RIDGE     0.984 0.987 0.990 0.0975 0.997
```

ROC Curves

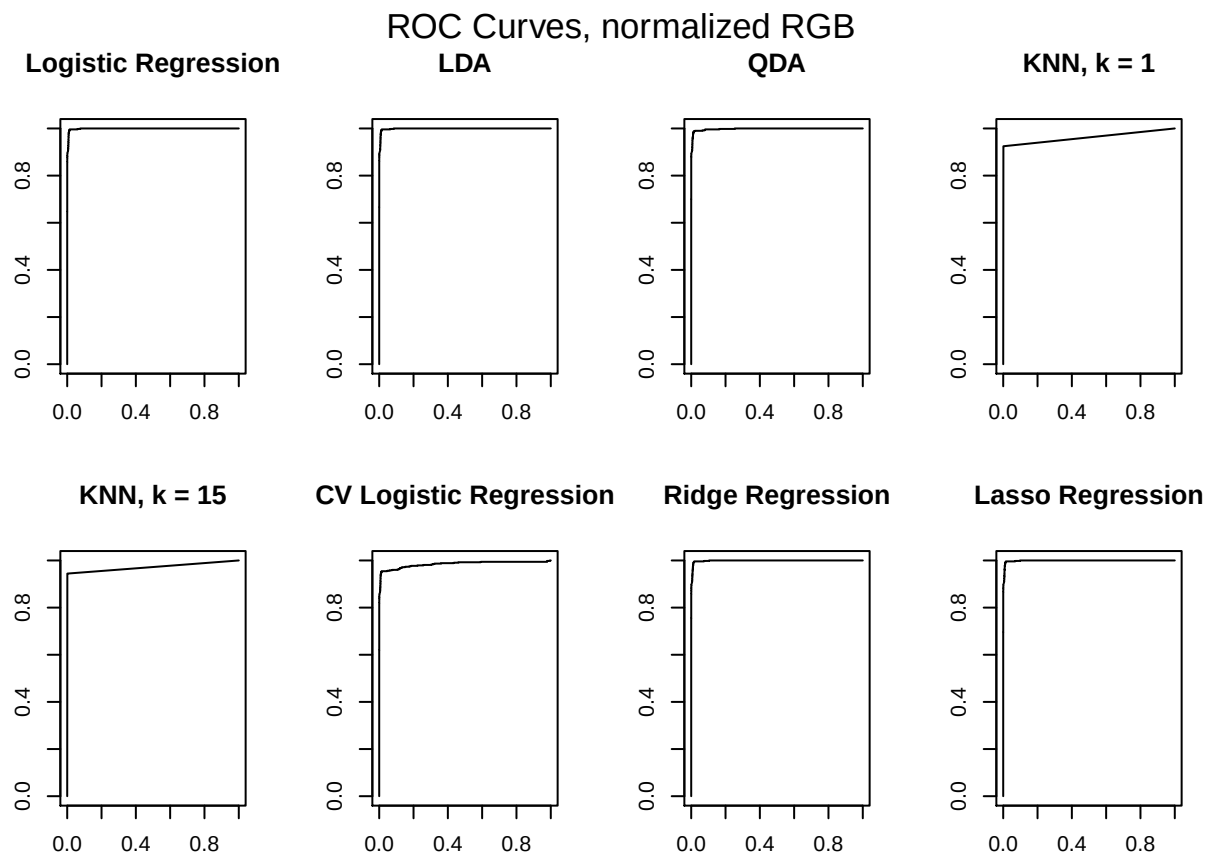
```
# plotting ROC curves, RGB values
par(mfrow = c(2, 4),
    mar = c(2, 2, 4, 2) + 0.5)

# log reg
prediction(glm_RGB_pred,
           RGB_test$Tarp) %>%
  performance('tpr', 'fpr') %>%
  plot(main = 'Logistic Regression')

# lda
prediction(lda_RGB_pred$posterior[, 2],
           RGB_test$Tarp)  %>%
  performance('tpr', 'fpr') %>%
  plot(main = 'LDA')

# qda
prediction(qda_RGB_pred$posterior[, 2],
           RGB_test$Tarp)  %>%
  performance('tpr', 'fpr') %>%
  plot(main = 'QDA')

# knn
prediction(as.numeric(knn_RGB) - 1,
           RGB_test$Tarp)  %>%
    performance('tpr', 'fpr') %>%
    plot(main = 'KNN, k = 1')

# knn, best-k
prediction(as.numeric(knn(train = as.matrix(RGB_train[, 1:3]),
                          test = as.matrix(RGB_test[, 1:3]),
                          cl = as.matrix(RGB_train$Tarp),
                          k = 7, prob = TRUE)) - 1,
           RGB_test$Tarp)  %>%
    performance('tpr', 'fpr') %>%
    plot(main = 'KNN, k = 7')

# cv log reg
prediction(cv_reg_pred0[, 1],
           RGB_test$Tarp)  %>%
  performance('tpr', 'fpr') %>%
  plot(main = 'CV Logistic Regression')

# ridge regression
prediction(cv_reg_pred0[, 2],
           RGB_test$Tarp)  %>%
  performance('tpr', 'fpr') %>%
  plot(main = 'Ridge Regression')

# lasso regression
prediction(cv_reg_pred0[, 3],
```

```
            RGB_test$Tarp)  %>%
  performance('tpr', 'fpr') %>%
  plot(main = 'Lasso Regression')

mtext('ROC Curves, RGB',
      outer = TRUE,
      cex = 1.1,
      line = -1.5)
```



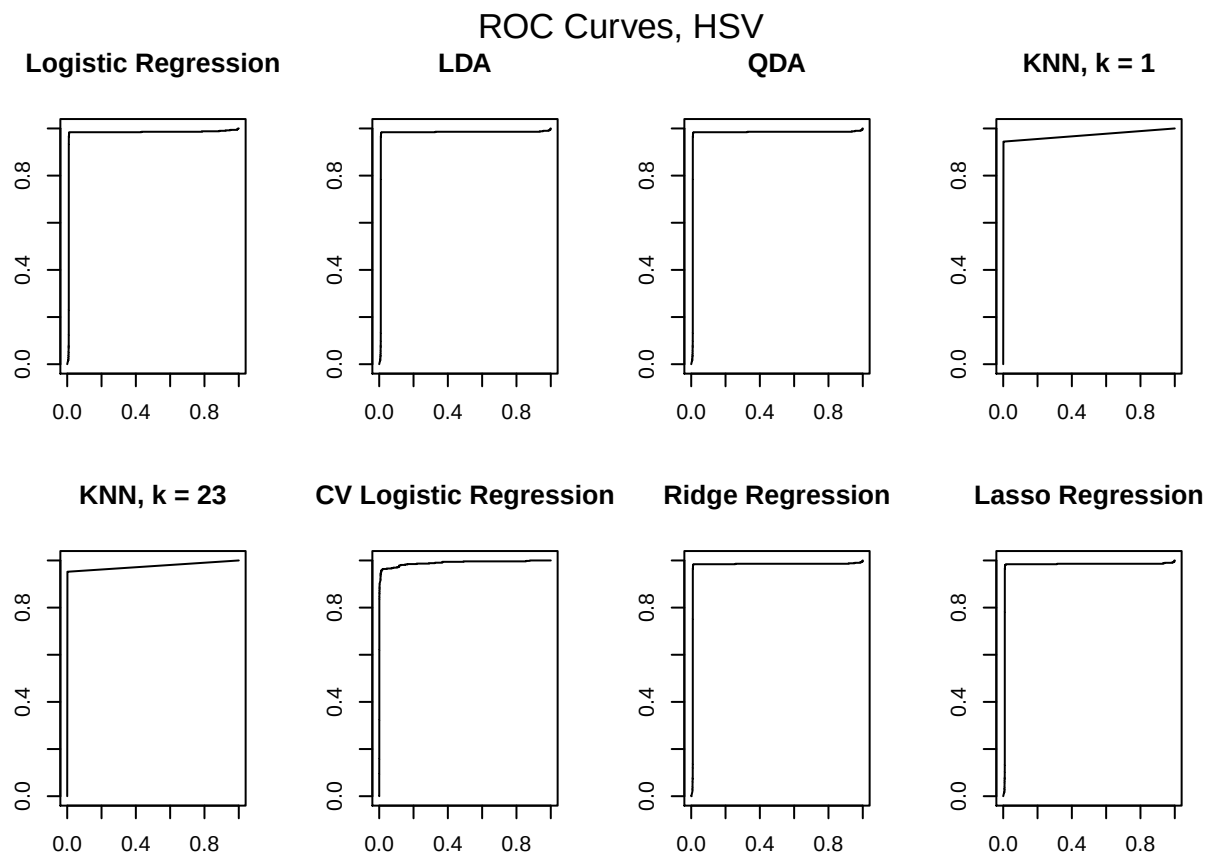

```
# plotting ROC curves, normalized RGB values
par(mfrow = c(2, 4),
    mar = c(2, 2, 4, 2) + 0.5)

# log reg
prediction(glm_RGC_pred,
           RGC_test$Tarp) %>%
  performance('tpr', 'fpr') %>%
  plot(main = 'Logistic Regression')

# lda
prediction(lda_RGC_pred$posterior[, 2],
           RGC_test$Tarp)  %>%
  performance('tpr', 'fpr') %>%
  plot(main = 'LDA')
```

```r
# qda
prediction(qda_RGC_pred$posterior[, 2],
           RGC_test$Tarp)  %>%
  performance('tpr', 'fpr') %>%
  plot(main = 'QDA')

# knn
prediction(as.numeric(knn_RGC) - 1,
           RGC_test$Tarp)  %>%
    performance('tpr', 'fpr') %>%
    plot(main = 'KNN, k = 1')

# knn, best-k
prediction(as.numeric(knn(train = as.matrix(RGC_train[, 1:2]),
                          test = as.matrix(RGC_test[, 1:2]),
                          cl = as.matrix(RGC_train$Tarp),
                          k = 15, prob = TRUE)) - 1,
           RGC_test$Tarp)  %>%
    performance('tpr', 'fpr') %>%
    plot(main = 'KNN, k = 15')

# cv log reg
prediction(cv_reg_pred0[, 4],
           RGB_test$Tarp)  %>%
  performance('tpr', 'fpr') %>%
  plot(main = 'CV Logistic Regression')

# ridge regression
prediction(cv_reg_pred0[, 5],
           RGC_test$Tarp)  %>%
  performance('tpr', 'fpr') %>%
  plot(main = 'Ridge Regression')

# lasso regression
prediction(cv_reg_pred0[, 6],
           RGC_test$Tarp)  %>%
  performance('tpr', 'fpr') %>%
  plot(main = 'Lasso Regression')

mtext('ROC Curves, normalized RGB',
      outer = TRUE,
      cex = 1.1,
      line = -1.5)
```




```
# plotting ROC curves, HSV values
par(mfrow = c(2, 4),
    mar = c(2, 2, 4, 2) + 0.5)

# log reg
prediction(glm_HSV_pred,
           HSV_test$Tarp) %>%
  performance('tpr', 'fpr') %>%
  plot(main = 'Logistic Regression')

# lda
prediction(lda_HSV_pred$posterior[, 2],
           HSV_test$Tarp)  %>%
  performance('tpr', 'fpr') %>%
  plot(main = 'LDA')

# qda
prediction(lda_HSV_pred$posterior[, 2],
           HSV_test$Tarp) %>%
  performance('tpr', 'fpr') %>%
  plot(main = 'QDA')

# knn
prediction(as.numeric(knn_HSV) - 1,
           HSV_test$Tarp) %>%
  performance('tpr', 'fpr') %>%
```

```
    plot(main = 'KNN, k = 1')

# knn, best-k
prediction(as.numeric(knn(train = as.matrix(HSV_train[, 1:3]),
                          test = as.matrix(HSV_test[, 1:3]),
                          cl = as.matrix(HSV_train$Tarp),
                          k = 23, prob = TRUE)) - 1,
           HSV_test$Tarp) %>%
  performance('tpr', 'fpr') %>%
  plot(main = 'KNN, k = 23')

# cv log reg
prediction(cv_reg_pred0[, 7],
           RGB_test$Tarp)  %>%
  performance('tpr', 'fpr') %>%
  plot(main = 'CV Logistic Regression')

# ridge regression
prediction(cv_reg_pred0[, 8],
           HSV_test$Tarp)  %>%
  performance('tpr', 'fpr') %>%
  plot(main = 'Ridge Regression')

# lasso regression
prediction(cv_reg_pred0[, 9],
           HSV_test$Tarp)  %>%
  performance('tpr', 'fpr') %>%
  plot(main = 'Lasso Regression')

mtext('ROC Curves, HSV',
      outer = TRUE,
      cex = 1.1,
      line = -1.5)
```



Cross-Validation Performance

```
### random forest
## setting up parallel
## use 'parallel' package to help with training times
# can use a lot of RAM

# setup
RF_tr_control <- caret::trainControl(method = 'cv',
                                     number = 10,
                                     summaryFunction = twoClassSummary,
                                     allowParallel = TRUE,
                                     classProbs = TRUE,
                                     savePredictions = TRUE,
                                     returnResamp = 'all',
                                     verboseIter = TRUE)

## allow for parallelism, setup new cluster
RF_cluster <- parallel::makeCluster(detectCores() - 2,
                                    setup_strategy = 'sequential')
registerDoParallel(RF_cluster)

# random forest, 10-fold
set.seed(6018)
```

```
rf_RGB <- caret::train(Tarp ~ Red + Green + Blue, data = RGB_train %>%
                         mutate(Tarp = as.factor(ifelse(RGB_train$Tarp == 1,
                                                        'Yes', 'No'))),
                       method = 'rf',
                       tuneGrid = data.frame(mtry = 1),
                       trControl = RF_tr_control)
```

```
## Aggregating results
## Fitting final model on full training set
```

```
rf_RGC <- caret::train(Tarp ~ Red + Green, data = RGC_train %>%
                         mutate(Tarp = as.factor(ifelse(RGC_train$Tarp == 1,
                                                        'Yes', 'No'))),
                       method = 'rf',
                       tuneGrid = data.frame(mtry = 1),
                       trControl = RF_tr_control)
```

```
## Aggregating results
## Fitting final model on full training set
```

```
rf_HSV <- caret::train(Tarp ~ Hue + Saturation + Value, data = HSV_train %>%
                         mutate(Tarp = as.factor(ifelse(HSV_train$Tarp == 1,
                                                        'Yes', 'No'))),
                       method = 'rf',
                       tuneGrid = data.frame(mtry = 1),
                       trControl = RF_tr_control)
```

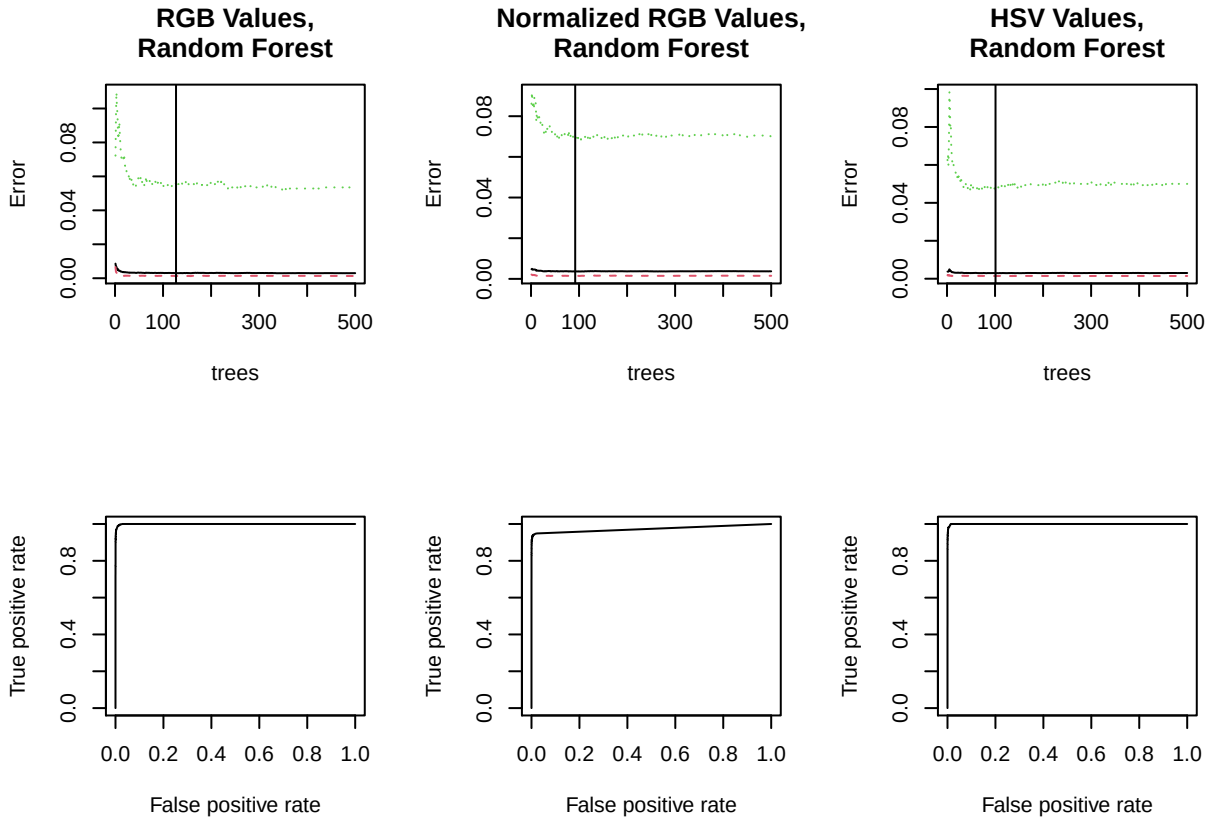
```
## Aggregating results
## Fitting final model on full training set
```

```
# end cluster
stopCluster(RF_cluster)
```

```
rf_RGB_pred0 <- predict(rf_RGB$finalModel,
                        RGB_test, type = 'prob')
rf_RGC_pred0 <- predict(rf_RGC$finalModel,
                        RGC_test, type = 'prob')
rf_HSV_pred0 <- predict(rf_HSV$finalModel,
                        HSV_test, type = 'prob')
```

```
## plotting ntrees and error, second row plotting ROC curves
par(mfrow = c(2, 3))

# ntrees and error
plot(rf_RGB$finalModel,
     main = 'RGB Values,\nRandom Forest')
abline(v = which.min(rf_RGB$finalModel$err.rate[, 1]))
plot(rf_RGC$finalModel,
     main = 'Normalized RGB Values,\nRandom Forest')
abline(v = which.min(rf_RGC$finalModel$err.rate[, 1]))
plot(rf_HSV$finalModel,
```

```
     main = 'HSV Values,\nRandom Forest')
abline(v = which.min(rf_HSV$finalModel$err.rate[, 1]))

# ROC
rf_RGB_pred1 <- prediction(rf_RGB_pred0[, 2],
                           RGB_test$Tarp)
rf_RGC_pred1 <- prediction(rf_RGC_pred0[, 2],
                           RGB_test$Tarp)
rf_HSV_pred1 <- prediction(rf_HSV_pred0[, 2],
                           HSV_test$Tarp)

performance(rf_RGB_pred1,
            'tpr', 'fpr') %>%
  plot()
performance(rf_RGC_pred1,
            'tpr', 'fpr') %>%
  plot()
performance(rf_HSV_pred1,
            'tpr', 'fpr') %>%
  plot()
```




```
# show confusion matricies
rf_RGB_table <- table(ifelse(rf_RGB_pred0[, 2] > 0.5, 1, 0),
                      RGB_test$Tarp)
rf_RGC_table <- table(ifelse(rf_RGC_pred0[, 2] > 0.5, 1, 0),
```

```
                      RGC_test$Tarp)
rf_HSV_table <- table(ifelse(rf_HSV_pred0[, 2] > 0.5, 1, 0),
                      HSV_test$Tarp)
rf_RGB_table
```

```
##
##         0     1
##   0 15269    35
##   1    15   492
```

```
rf_RGC_table
```

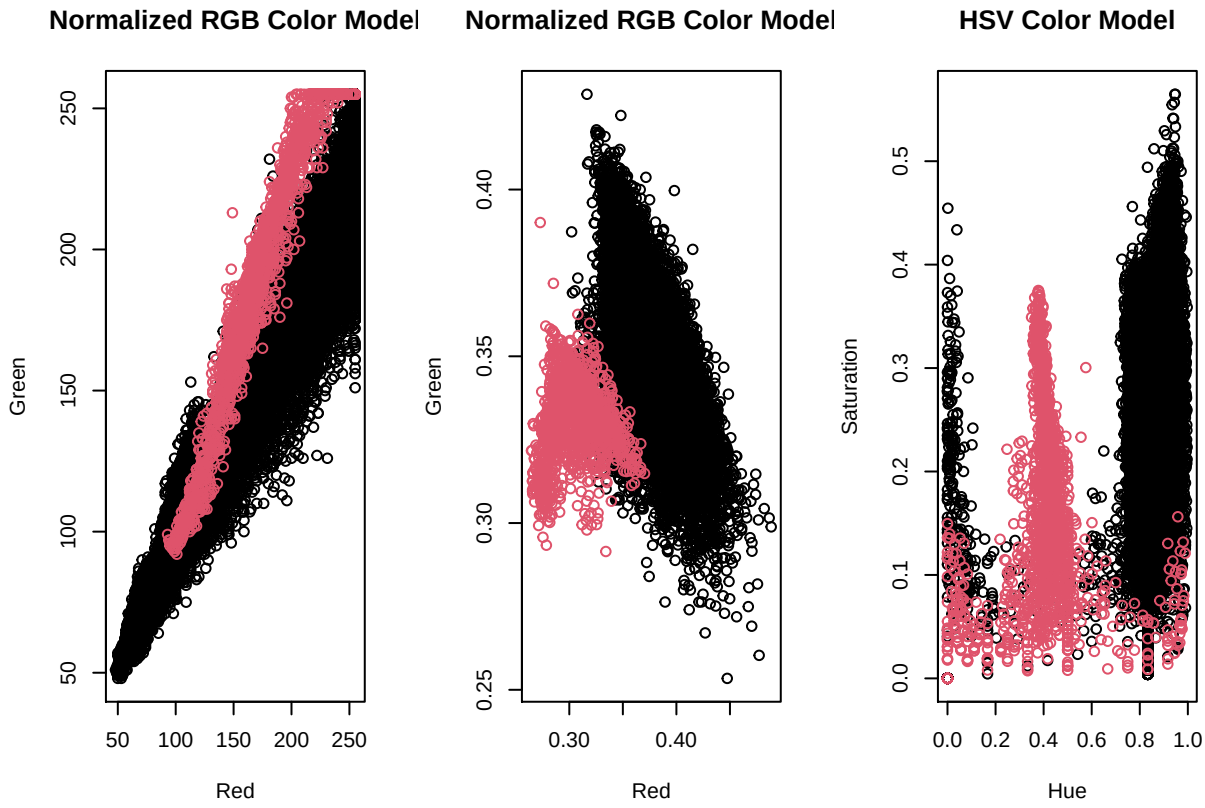
```
##
##         0     1
##   0 15298    32
##   1     9   472
```

```
rf_HSV_table
```

```
##
##         0     1
##   0 15289    25
##   1    20   477
```

```
rf_RGB_acc <- model_stats(rf_RGB_pred0[, 2],
                          RGB_test$Tarp,
                          rf_RGB_table)
rf_RGC_acc <- model_stats(rf_RGC_pred0[, 2],
                          RGC_test$Tarp,
                          rf_RGC_table)
rf_HSV_acc <- model_stats(rf_HSV_pred0[, 2],
                          HSV_test$Tarp,
                          rf_HSV_table)
```

```
### support vector machines
## pre-sum plotting of points, assuming two most relevant variables per color model
# red&green for RGB and normalized RGB, for sake of comparison, hue&saturation for HSV
par(mfrow = c(1, 3))
plot(HP_RGB$Red, HP_RGB$Green,
     col = as.numeric(HP_RGB$Tarp),
     main = 'Normalized RGB Color Model',
     xlab = 'Red',
     ylab = 'Green')
plot(HP_RGC$Red, HP_RGC$Green,
     col = as.numeric(HP_RGC$Tarp),
     main = 'Normalized RGB Color Model',
     xlab = 'Red',
     ylab = 'Green')
plot(HP_HSV$Hue, HP_HSV$Saturation,
     col = as.numeric(HP_HSV$Tarp),
     main = 'HSV Color Model',
     xlab = 'Hue',
     ylab = 'Saturation')
```




```
## use caret::train for better performance
# define trimmed data frame for training
# use even smaller training sets for the sake of time, 1/4 of the num of obs
set.seed(6018)

RGB_SVM_subset <- sample(nrow(HP_RGB), nrow(HP_RGB) * 0.25)
RGB_SVM_train <- HP_RGB[RGB_SVM_subset, ] %>%
                  mutate(Tarp = as.factor(ifelse(Tarp == 1,
                                                 'Yes', 'No')))
RGB_SVM_test <- HP_RGB[-RGB_SVM_subset, ] %>%
                  mutate(Tarp = as.factor(ifelse(Tarp == 1,
                                                 'Yes', 'No')))

RGC_SVM_subset <- sample(nrow(HP_RGC), nrow(HP_RGC) * 0.25)
RGC_SVM_train <- HP_RGC[RGC_SVM_subset, ] %>%
                  mutate(Tarp = as.factor(ifelse(Tarp == 1,
                                                 'Yes', 'No')))
RGC_SVM_test <- HP_RGC[-RGC_SVM_subset, ] %>%
                  mutate(Tarp = as.factor(ifelse(Tarp == 1,
                                                 'Yes', 'No')))

HSV_SVM_subset <- sample(nrow(HP_HSV), nrow(HP_HSV) * 0.25)
HSV_SVM_train <- HP_HSV[HSV_SVM_subset, ] %>%
                  mutate(Tarp = as.factor(ifelse(Tarp == 1,
                                                 'Yes', 'No')))
HSV_SVM_test <- HP_HSV[-HSV_SVM_subset, ] %>%
```

```
                  mutate(Tarp = as.factor(ifelse(Tarp == 1,
                                                 'Yes', 'No')))
```

```
## support vector machines, 10-fold cv
## setting up parallel
## use 'parallel' package to help with CV SVM times
# will use a lot of RAM

# setup
# disable return of training data
svm_cost <- 10^c(0:2)
SVM_trcontrol0 <- caret::trainControl(allowParallel = TRUE,
                                      classProbs = TRUE,
                                      method = 'cv',
                                      number = 10,
                                      returnData = F,
                                      savePredictions = TRUE,
                                      verboseIter = TRUE)

## allow for parallelism, setup new cluster
SVM_cluster <- parallel::makeCluster(detectCores() - 2,
                                     setup_strategy = 'sequential')
registerDoParallel(SVM_cluster)

## linear SVM
svm_RGB_linear <- train(Tarp ~ Red + Green + Blue, data = RGB_SVM_train,
                        method = 'svmLinear',
                        preProcess = c('center','scale'),
                        tuneGrid = expand.grid(C = svm_cost),
                        trControl = SVM_trcontrol0)
```

```
## Aggregating results
## Selecting tuning parameters
## Fitting C = 1 on full training set
```

```
svm_RGC_linear <- train(Tarp ~ Red + Green, data = RGC_SVM_train,
                        method = 'svmLinear',
                        preProcess = c('center','scale'),
                        tuneGrid = expand.grid(C = svm_cost),
                        trControl = SVM_trcontrol0)
```

```
## Aggregating results
## Selecting tuning parameters
## Fitting C = 1 on full training set
```

```
svm_HSV_linear <- train(Tarp ~ Hue + Saturation + Value, data = HSV_SVM_train,
                        method = 'svmLinear',
                        preProcess = c('center', 'scale'),
                        tuneGrid = expand.grid(C = svm_cost),
                        trControl = SVM_trcontrol0)
```

```
## Aggregating results
```

```
## Selecting tuning parameters
## Fitting C = 10 on full training set
```

```
## radial SVM
# use sigest() from kernlab package for automated parameter estimation for radial SVM
svm_RGB_radial <- train(Tarp  ~ Red + Green + Blue, data = RGB_SVM_train,
                        method = 'svmRadial',
                        preProcess = c('center','scale'),
                        tuneGrid = expand.grid(C = svm_cost,
                                               sigma = sigest(as.matrix(RGB_SVM_train[, 1:2]),
                                                              scaled = TRUE)),
                        trControl = SVM_trcontrol0)
```

```
## Aggregating results
## Selecting tuning parameters
## Fitting sigma = 28.4, C = 10 on full training set
```

```
svm_RGC_radial <- train(Tarp  ~ Red + Green, data = RGC_SVM_train,
                        method = 'svmRadial',
                        preProcess = c('center','scale'),
                        tuneGrid = expand.grid(C = svm_cost,
                                               sigma = sigest(as.matrix(RGC_SVM_train[, 1:2]),
                                                              scaled = TRUE)),
                        trControl = SVM_trcontrol0)
```

```
## Aggregating results
## Selecting tuning parameters
## Fitting sigma = 3.66, C = 100 on full training set
```

```
svm_HSV_radial <- train(Tarp  ~ Hue + Saturation + Value, data = HSV_SVM_train,
                        method = 'svmRadial',
                        preProcess = c('center','scale'),
                        tuneGrid = expand.grid(C = svm_cost,
                                               sigma = sigest(as.matrix(HSV_SVM_train[, 1:2]),
                                                              scaled = TRUE)),
                        trControl = SVM_trcontrol0)
```

```
## Aggregating results
## Selecting tuning parameters
## Fitting sigma = 0.103, C = 100 on full training set
```

```
## polynomial SVM
svm_RGB_polynm <- train(Tarp  ~ Red + Green + Blue, data = RGB_SVM_train,
                        method = 'svmPoly',
                        tuneGrid = expand.grid(C = 1,
                                               degree = c(1:2),
                                               scale = 10^c(1:2)),
                        trControl = SVM_trcontrol0)
```

```
## Aggregating results
## Selecting tuning parameters
## Fitting degree = 2, scale = 100, C = 1 on full training set
```

```
svm_RGC_polynm <- train(Tarp  ~ Red + Green, data = RGC_SVM_train,
                        method = 'svmPoly',
                        tuneGrid = expand.grid(C = 1,
                                               degree = c(1:2),
                                               scale = 10^c(1:2)),
                        trControl = SVM_trcontrol0)
```

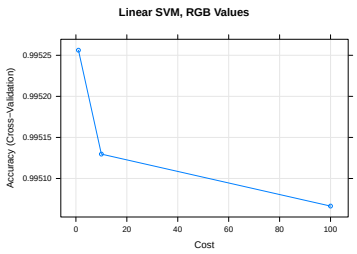
```
## Aggregating results
## Selecting tuning parameters
## Fitting degree = 1, scale = 100, C = 1 on full training set
```

```
svm_HSV_polynm <- train(Tarp  ~ Hue + Saturation + Value, data = HSV_SVM_train,
                        method = 'svmPoly',
                        tuneGrid = expand.grid(C = 1,
                                               degree = 2^c(0:1),
                                               scale = 10^c(1:2)),
                        trControl = SVM_trcontrol0)
```

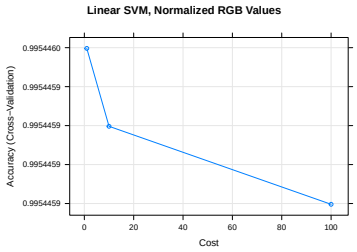
```
## Aggregating results
## Selecting tuning parameters
## Fitting degree = 2, scale = 10, C = 1 on full training set
```

```
# end cluster
stopCluster(SVM_cluster)
```

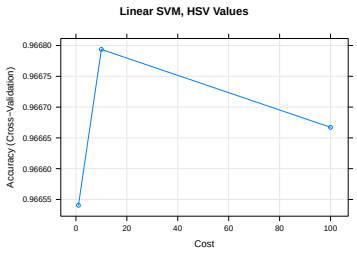
```
## plotting SVMs
kernlab::plot(svm_RGB_linear,
              main = 'Linear SVM, RGB Values')
```



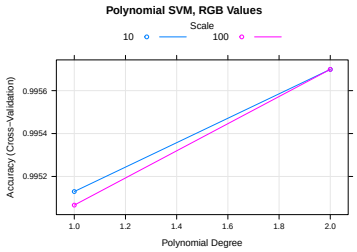

```
kernlab::plot(svm_RGC_linear,
              main = 'Linear SVM, Normalized RGB Values')
```



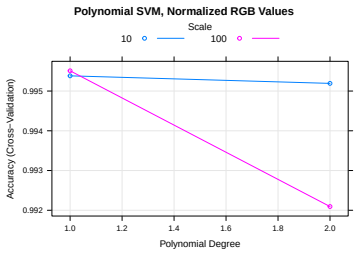
```
kernlab::plot(svm_HSV_linear,
              main = 'Linear SVM, HSV Values')
```



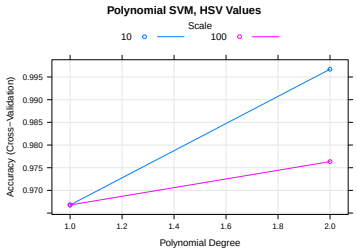

```
kernlab::plot(svm_RGB_polynm,
              main = 'Polynomial SVM, RGB Values')
```



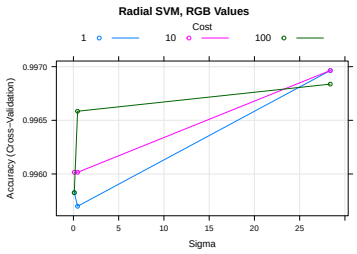

```
kernlab::plot(svm_RGC_polynm,
              main = 'Polynomial SVM, Normalized RGB Values')
```



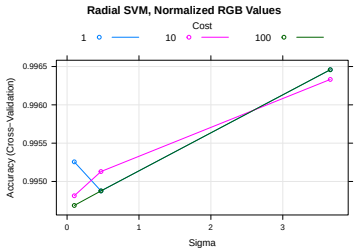

```
kernlab::plot(svm_HSV_polynm,
              main = 'Polynomial SVM, HSV Values')
```



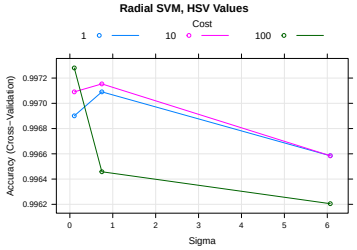

```
kernlab::plot(svm_RGB_radial,
              main = 'Radial SVM, RGB Values')
```




```
kernlab::plot(svm_RGC_radial,
              main = 'Radial SVM, Normalized RGB Values')
```




```
kernlab::plot(svm_HSV_radial,
              main = 'Radial SVM, HSV Values')
```




```
## predicting with test data, values and probabilities
svm_list <- list(svm_RGB_linear, svm_RGB_polynm, svm_RGB_radial,
                 svm_RGC_linear, svm_RGC_radial, svm_RGC_polynm,
                 svm_HSV_linear, svm_HSV_polynm, svm_HSV_radial)
svm_test_list <- list(RGB_SVM_test, RGB_SVM_test, RGB_SVM_test,
                      RGC_SVM_test, RGC_SVM_test, RGC_SVM_test,
                      HSV_SVM_test, HSV_SVM_test, HSV_SVM_test)

# 9-column matrix, predicted probabilities per column
# in order: 3/ea RGB, RGC, HSV of linear, polynomial, radial
svm_pred0 <- mapply(function(x, y)
                    predict(x, y, type = 'prob'),
                    svm_list, svm_test_list)

# 9-column matrix, fitted values per column
svm_pred1 <- mapply(function(x, y)
                    predict(x, y),
                    svm_list, svm_test_list)
```

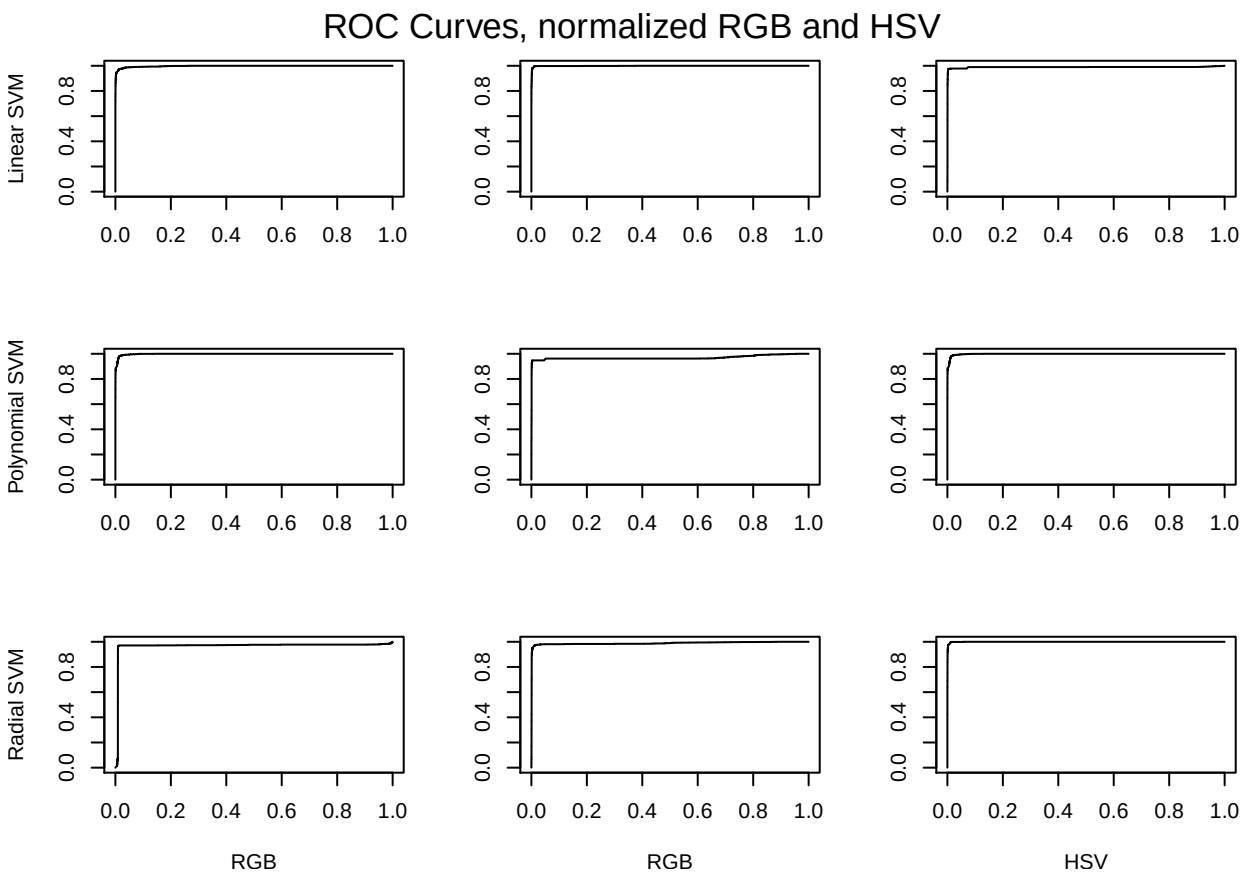
```
# 9-column matrix, prediction values per column (for ROC)
svm_pred2 <- list()
for (i in 1:ncol(svm_pred0)) {
  svm_pred2[[i]] <- prediction(svm_pred0[, i]$Yes,
                               svm_test_list[[i]]$Tarp)
}
```

```
# order: RGB linear, polynomial, radial, then same for RGC / normalized RGB, then HSV
for (i in 1:ncol(svm_pred1)) {
  print(table(svm_pred1[, i], svm_test_list[[i]]$Tarp))
}
```

```
##
##          No   Yes
##   No  45843   179
##   Yes    41  1368
##
##          No   Yes
##   No  45839   175
##   Yes    45  1372
##
##          No   Yes
##   No  45812    94
##   Yes    72  1453
##
##          No   Yes
##   No  45877   210
##   Yes    29  1315
##
##          No   Yes
##   No  45851   145
##   Yes    55  1380
##
##          No   Yes
##   No  45877   202
##   Yes    29  1323
##
##          No   Yes
##   No  45511  1114
##   Yes   411   395
##
##          No   Yes
##   No  45833   119
##   Yes    89  1390
##
##          No   Yes
##   No  45858    84
##   Yes    64  1425
```

```
## plotting ROC
par(mfrow = c(3, 3),
    mai = rep(0, 4),
    mar = c(3.8, 3.8, 2.2, 0.8))
```

```
performance(svm_pred2[[1]],
            'tpr', 'fpr') %>%
  plot(main = '', xlab = '', ylab = 'Linear SVM',
       xaxt = 'n', yaxt = 'n')
performance(svm_pred2[[2]],
            'tpr', 'fpr') %>%
  plot(main = '', xlab = '', ylab = '',
       xaxt = 'n', yaxt = 'n')
performance(svm_pred2[[3]],
            'tpr', 'fpr') %>%
  plot(main = '', xlab = '', ylab = '',
       xaxt = 'n', yaxt = 'n')
performance(svm_pred2[[4]],
            'tpr', 'fpr') %>%
  plot(main = '', xlab = '', ylab = 'Polynomial SVM',
       xaxt = 'n', yaxt = 'n')
performance(svm_pred2[[5]],
            'tpr', 'fpr') %>%
  plot(main = '', xlab = '', ylab = '',
       xaxt = 'n', yaxt = 'n')
performance(svm_pred2[[6]],
            'tpr', 'fpr') %>%
  plot(main = '', xlab = '', ylab = '',
       xaxt = 'n', yaxt = 'n')
performance(svm_pred2[[7]],
            'tpr', 'fpr') %>%
  plot(main = '', xlab = 'RGB', ylab = 'Radial SVM',
       xaxt = 'n', yaxt = 'n')
performance(svm_pred2[[8]],
            'tpr', 'fpr') %>%
  plot(main = '', xlab = 'RGB', ylab = '',
       xaxt = 'n', yaxt = 'n')
performance(svm_pred2[[9]],
            'tpr', 'fpr') %>%
  plot(main = '', xlab = 'HSV', ylab = '',
       xaxt = 'n', yaxt = 'n')

mtext('ROC Curves, normalized RGB and HSV',
      outer = TRUE,
      cex = 1.1,
      line = -1.5)
```




```
## bind results to be used in performance table
svm_results_df <- matrix(ncol = 7, nrow = 9)
for (i in 1:ncol(svm_pred0)) {
  svm_stats <- model_stats(svm_pred0[, i]$Yes,
                           svm_test_list[[i]]$Tarp,
                           table(svm_pred1[, i],
                                 svm_test_list[[i]]$Tarp))
  svm_results_df[i, ] <- svm_stats
}

## turn into data frame, clean and name properly
svm_results_df <- data.frame(svm_results_df)
colnames(svm_results_df) <- stats_label

svm_results_df$Color <- rep(c('RGB', 'RGC', 'HSV'), each = 3)
svm_results_df$Model <- rep(c('SVML', 'SVMP', 'SVMR'), 3)
```

Hold-Out Test Sample Performance

```
## load in .txt data
# store filenames in lists for easy calling
HO_blue_files <- list.files(path = '.',
                            pattern = 'orthovnir.*_ROI_Blue_Tarps.txt')
HO_nonblue_files <- list.files(path = '.',
```

```
                                pattern = 'orthovnir.*_ROI_NO._Blue_Tarps.*.txt')

read_holdout <- function(file_list) {
  holdout_data <- data.frame()
  ## loop through files
  for (i in 1:length(file_list)) {
    filename <- file_list[i]
    # record the file number as a variable
    Sample <- str_extract(filename, '\\d+') %>%
              as.numeric()
    # set Tarp value based upon document name, NOT a blue tarp == 0, is a blue tarp == 1
    Tarp <- ifelse(str_detect(filename,
                              '(NO[NT])\\_Blue\\_Tarps'),
                   0, 1) %>%
              as.factor()
    # reading in without colnames, as they will be filled in at some point anyways
    # skips extraneous data
    HO_read <- read.table(filename,
                          skip = 8,
                          header = FALSE) %>%
                 data.frame()
    HO_read$Sample <- Sample
    HO_read$Tarp <- Tarp

    holdout_data <- rbind(holdout_data,
                          HO_read)
  }
  return(holdout_data)
}
```

```
## use function, read in files
HP_HoldOut <- rbind(read_holdout(HO_blue_files),
                    read_holdout(HO_nonblue_files))
# remove first column, the key leftover from rbind()
# take last 11 columns, to be error-proof
HP_HoldOut <- HP_HoldOut[, (ncol(HP_HoldOut) - 10):ncol(HP_HoldOut)]

## rename columns
colnames(HP_HoldOut) <- c('X', 'Y', 'MapX', 'MapY',
                          'Latitude', 'Longitude',
                          'Red', 'Green', 'Blue',
                          'Sample', 'Tarp')

## make RGB, normalized RGB, and HSV versions of the hold-out data
options(scipen = 999)

## normal RGB
HO_RGB <- HP_HoldOut[, c(7:9, 11)]

## normalized RGB
# preprocess to normalize values
HO_RGC <- HP_HoldOut[, c(7:9, 11)] %>%
              transform(Red = Red / (Red + Green + Blue),
```

```
                        Green = Green / (Red + Green + Blue),
                        Blue = Blue / (Red + Green + Blue))

##HSV
# apply RGB to HSV conversion from grDevices package
HoldOut_HSV <- mapply(rgb2hsv, HP_HoldOut$Red, HP_HoldOut$Blue, HP_HoldOut$Green)
# use tidyr spread to convert from long to wide data, with hue, saturation, and value as columns
HoldOut_HSV <- spread(as.data.frame(as.table(HoldOut_HSV)), Var1, Freq)
# duplicate holdout df, substitute RGB columns for HSV values
HO_HSV <- HP_HoldOut[, c(7:9, 11)]
HO_HSV[, 1:3] <- HoldOut_HSV[, -1]

# rename columns
colnames(HO_HSV) <- c('Hue', 'Saturation',
                      'Value', 'Tarp')

# remove objects to avoid memory errors
rm(HoldOut_HSV, HP_HoldOut)
```

```
#### random forest
## randomforest, RGB
rf_RGB_HOpred0 <- predict(rf_RGB,
                          HO_RGB,
                          type = 'prob')
# table requires column flip
rf_RGB_HOtable <- table(ifelse(rf_RGB_HOpred0[, 2] > 0.5, 1, 0),
                        HO_RGB$Tarp)[, 2:1]
rf_RGB_HOresults <- model_stats(rf_RGB_HOpred0[, 2],
                                HO_RGB$Tarp,
                                rf_RGB_HOtable)
## randomforest, normalized RGB
rf_RGC_HOpred0 <- predict(rf_RGC,
                          HO_RGC,
                          type = 'prob')
# table requires column flip
rf_RGC_HOtable <- table(ifelse(rf_RGC_HOpred0[, 2] > 0.5, 1, 0),
                        HO_RGC$Tarp)[, 2:1]
rf_RGC_HOresults <- model_stats(rf_RGC_HOpred0[, 2],
                                HO_RGC$Tarp,
                                rf_RGC_HOtable)
## randomforest, HSV
rf_HSV_HOpred0 <- predict(rf_HSV,
                          HO_HSV,
                          type = 'prob')
# table requires column flip
rf_HSV_HOtable <- table(ifelse(rf_HSV_HOpred0[, 2] > 0.5, 1, 0),
                        HO_HSV$Tarp)[, 2:1]
rf_HSV_HOresults <- model_stats(rf_HSV_HOpred0[, 2],
                                HO_HSV$Tarp,
                                rf_HSV_HOtable)

## remove objects to avoid memory errors with the rest of the project
rm(rf_RGB_HOpred0, rf_RGC_HOpred0, rf_HSV_HOpred0)
```

```r
# output confusion matricies
rf_RGB_HOtable
```

```
##
##           0       1
##   0 1983464    3554
##   1    6233   10926
```

```r
rf_RGC_HOtable
```

```
##
##           0       1
##   0 1945877     119
##   1   43820   14361
```

```r
rf_HSV_HOtable
```

```
##
##           0       1
##   0 1934618    3360
##   1   55079   11120
```

```r
#### cross-validated glm, HSV
# create matrix for testing
cvglm_HSV_HO_test <- model.matrix(Tarp ~ poly(Hue, 4, raw = TRUE) +
                                         poly(Saturation, 4, raw = TRUE) +
                                         poly(Value, 4, raw = TRUE),
                                  data = HO_HSV)[,-1]
cvglm_HSV_HOpred0 <- predict(HSV_cvglm,
                             cvglm_HSV_HO_test,
                             s = HSV_cvglm$lambda.min)
# create threshold similar to that with testing
cvglm_HSV_HOthresh <- (max(cvglm_HSV_HOpred0) + min(cvglm_HSV_HOpred0)) / 2
cvglm_HSV_HOpred1 <- ifelse(cvglm_HSV_HOpred0 > cvglm_HSV_HOthresh, 1, 0)
# table requires column flip
cvglm_HSV_HOtable <- table(cvglm_HSV_HOpred1, HO_HSV$Tarp)[, 2:1]
cvglm_HSV_HOresults <- model_stats(cvglm_HSV_HOpred0,
                                   HO_HSV$Tarp,
                                   cvglm_HSV_HOtable)

## remove objects to avoid memory errors with the rest of the project
rm(cvglm_HSV_HO_test, cvglm_HSV_HOpred0, cvglm_HSV_HOpred1)
# output confusion matrix
cvglm_HSV_HOtable
```

```
##
## cvglm_HSV_HOpred1       0       1
##                 0 1961094      94
##                 1   28603   14386
```

```
#### svm
## linear svm, RGB
svmL_RGB_HOpred0 <- predict(svm_RGB_linear,
                            newdata = HO_RGB)
svmL_RGB_HOpred1 <- predict(svm_RGB_linear,
                            newdata = HO_RGB,
                            type = 'prob')[, 2]
# table requires column flip
svmL_RGB_HOtable <- table(svmL_RGB_HOpred0,
                          HO_RGB$Tarp)[, 2:1]
svmL_RGB_HOresults <- model_stats(svmL_RGB_HOpred1,
                                  HO_RGB$Tarp,
                                  svmL_RGB_HOtable)
## polynomial svm, RGB
svmP_RGB_HOpred0 <- predict(svm_RGB_polynm,
                            newdata = HO_RGB)
svmP_RGB_HOpred1 <- predict(svm_RGB_polynm,
                            newdata = HO_RGB,
                            type = 'prob')[, 2]
# table requires column flip
svmP_RGB_HOtable <- table(svmP_RGB_HOpred0,
                          HO_RGB$Tarp)[, 2:1]
svmP_RGB_HOresults <- model_stats(svmP_RGB_HOpred1,
                                  HO_RGB$Tarp,
                                  svmP_RGB_HOtable)
## radial svm, RGB
svmR_RGB_HOpred0 <- predict(svm_RGB_radial,
                            newdata = HO_RGB)
svmR_RGB_HOpred1 <- predict(svm_RGB_radial,
                            newdata = HO_RGB,
                            type = 'prob')[, 2]
# table requires column flip
svmR_RGB_HOtable <- table(svmR_RGB_HOpred0,
                          HO_RGB$Tarp)[, 2:1]
svmR_RGB_HOresults <- model_stats(svmR_RGB_HOpred1,
                                  HO_RGB$Tarp,
                                  svmR_RGB_HOtable)

## linear svm, normalized RGB
svmL_RGC_HOpred0 <- predict(svm_RGC_linear,
                            newdata = HO_RGC)
svmL_RGC_HOpred1 <- predict(svm_RGC_linear,
                            newdata = HO_RGC,
                            type = 'prob')[, 2]
# table requires column flip
svmL_RGC_HOtable <- table(svmL_RGC_HOpred0,
                          HO_RGC$Tarp)[, 2:1]
svmL_RGC_HOresults <- model_stats(svmL_RGC_HOpred1,
                                  HO_RGC$Tarp,
                                  svmL_RGC_HOtable)

## radial svm, HSV
svmR_HSV_HOpred0 <- predict(svm_RGC_radial,
```

```
                            newdata = HO_RGC)
svmR_HSV_HOpred1 <- predict(svm_RGC_radial,
                            newdata = HO_RGC,
                            type = 'prob')[, 2]
# table requires column flip
svmR_HSV_HOtable <- table(svmR_HSV_HOpred0,
                          HO_RGC$Tarp)[, 2:1]
svmR_HSV_HOresults <- model_stats(svmR_HSV_HOpred1,
                                  HO_RGC$Tarp,
                                  svmR_HSV_HOtable)

# remove objects to avoid memory errors
rm(svmL_RGB_HOpred0, svmL_RGB_HOpred1, svmP_RGB_HOpred0,
   svmP_RGB_HOpred1, svmR_RGB_HOpred0, svmR_RGB_HOpred1,
   svmL_RGC_HOpred0, svmL_RGC_HOpred1, svmR_HSV_HOpred0,
   svmR_HSV_HOpred1)
# output confusion matricies
svmL_RGB_HOtable
```

```
##
## svmL_RGB_HOpred0       0       1
##              No  1960907     180
##              Yes   28790   14300
```

```
svmP_RGB_HOtable
```

```
##
## svmP_RGB_HOpred0       0       1
##              No  1982417     608
##              Yes    7280   13872
```

```
svmR_RGB_HOtable
```

```
##
## svmR_RGB_HOpred0       0       1
##              No  1977122   10914
##              Yes   12575    3566
```

```
svmL_RGC_HOtable
```

```
##
## svmL_RGC_HOpred0       0       1
##              No  1974588     221
##              Yes   15109   14259
```

```
svmR_HSV_HOtable
```

```
##
## svmR_HSV_HOpred0       0       1
##              No  1946528    6810
##              Yes   43169    7670
```

```
#### glm / lda
## glm, RGB
glm_RGB_HOpred0 <- predict(glm_RGB,
                           newdata = HO_RGB,
                           type = 'response')
# table requires column flip
glm_RGB_HOtable <- table(ifelse(glm_RGB_HOpred0 > 0.5, 1, 0),
                         HO_RGB$Tarp)[, 2:1]
glm_RGB_HOresults <- model_stats(glm_RGB_HOpred0,
                                 HO_RGB$Tarp,
                                 glm_RGB_HOtable)

## glm, normalized RGB
glm_RGC_HOpred0 <- predict(glm_RGC,
                           newdata = HO_RGC,
                           type = 'response')
# table requires column flip
glm_RGC_HOtable <- table(ifelse(glm_RGC_HOpred0 > 0.5, 1, 0),
                         HO_RGC$Tarp)[, 2:1]
glm_RGC_HOresults <- model_stats(glm_RGC_HOpred0,
                                 HO_RGC$Tarp,
                                 glm_RGC_HOtable)
## lda, normalized RGB
lda_RGC_HOpred0 <- predict(lda_RGC,
                           newdata = HO_RGC)
# table requires column flip
lda_RGC_HOtable <- table(lda_RGC_HOpred0$class,
                         HO_RGC$Tarp)[, 2:1]
lda_RGC_HOresults <- model_stats(lda_RGC_HOpred0$posterior[, 2],
                                 HO_RGC$Tarp,
                                 lda_RGC_HOtable)

# remove objects to avoid memory errors
rm(glm_RGB_HOpred0, glm_RGC_HOpred0, lda_RGC_HOpred0)
# output confusion matricies
glm_RGB_HOtable
```

```
##
##           0       1
##   0 1971807     176
##   1   17890   14304
```

```
glm_RGC_HOtable
```

```
##
##           0       1
##   0 1972553     178
##   1   17144   14302
```

```
lda_RGC_HOtable
```

```
##
```

```
##           0       1
##   0 1972674     178
##   1   17023   14302
```

Cross-Validation Performance Table

```
## comparing cv models to existing
cv_perf_table <- rbind(perf_table, rf_RGB_acc,
                       rf_RGC_acc, rf_HSV_acc,
                       svm_results_df) %>%
                    data.frame()

cv_perf_table[, 2:6] <- lapply(cv_perf_table[, 2:6],
                               as.numeric)
cv_perf_table[, 2:6] <- cv_perf_table[, 2:6] %>%
                          round(4)

cv_perf_table <- cv_perf_table[order(-(cv_perf_table$AUROC * 1.2 +
                                       cv_perf_table$Accuracy * 1.2 +
                                       cv_perf_table$TPR * 2.4 +
                                       cv_perf_table$FPR * -0.2 +
                                       cv_perf_table$Precision * 0.2) / 5), ]

rownames(cv_perf_table) <- NULL
cv_perf_table[1:12, ]
```

```
##    Color  AUROC Accuracy    TPR    FPR Precision Model
## 1    HSV 0.9997   0.9972 0.9987 0.0498    0.9984    RF
## 2    HSV 0.9995   0.9969 0.9986 0.0557    0.9982  SVMR
## 3    RGB 0.9996   0.9968 0.9990 0.0664    0.9977    RF
## 4    RGC 0.9927   0.9974 0.9994 0.0635    0.9979    RF
## 5    HSV 0.9990   0.9958 0.9982 0.0777    0.9975 CVGLM
## 6    RGC 0.9990   0.9957 0.9993 0.1131    0.9963   GLM
## 7    RGC 0.9989   0.9958 0.9995 0.1171    0.9962   LDA
## 8    RGB 0.9991   0.9954 0.9990 0.1131    0.9962  SVMP
## 9    RGB 0.9891   0.9965 0.9984 0.0608    0.9980  SVMR
## 10   RGB 0.9975   0.9954 0.9991 0.1157    0.9961  SVML
## 11   RGB 0.9983   0.9950 0.9992 0.1271    0.9956   GLM
## 12   RGC 0.9985   0.9951 0.9994 0.1325    0.9956  SVMR
```

```
## grouped performance statistics
cv_perf_table %>%
  group_by(Color) %>%
  summarize(mAUROC = mean(AUROC), mAcc = mean(Accuracy),
            mTPR = mean(TPR), mFPR = mean(FPR),
            mPrc = mean(Precision))
```

```
## # A tibble: 3 x 6
##   Color mAUROC  mAcc  mTPR   mFPR  mPrc
##   <chr>  <dbl> <dbl> <dbl>  <dbl> <dbl>
## 1 HSV    0.983 0.990 0.994 0.154  0.995
```

```
## 2 RGB    0.988 0.963 0.966 0.124  0.996
## 3 RGC    0.991 0.991 0.993 0.0796 0.997
```

```
cv_perf_table %>%
  group_by(Model) %>%
  summarize(mAUROC = mean(AUROC), mAcc = mean(Accuracy),
            mTPR = mean(TPR), mFPR = mean(FPR),
            mPrc = mean(Precision))
```

```
## # A tibble: 12 x 6
##    Model    mAUROC  mAcc  mTPR   mFPR  mPrc
##    <chr>     <dbl> <dbl> <dbl>  <dbl> <dbl>
##  1 CVGLM     0.999 0.876 0.873 0.0272 0.999
##  2 GLM       0.992 0.987 0.996 0.289  0.991
##  3 KNN       0.967 0.997 0.999 0.0646 0.998
##  4 KNNBESTK  0.976 0.997 0.999 0.0469 0.998
##  5 LASSO     0.980 0.974 0.977 0.122  0.996
##  6 LDA       0.988 0.990 0.993 0.114  0.996
##  7 QDA       0.994 0.992 0.995 0.0981 0.997
##  8 RF        0.997 0.997 0.999 0.0599 0.998
##  9 RIDGE     0.984 0.987 0.990 0.0975 0.997
## 10 SVML      0.988 0.986 0.997 0.331  0.989
## 11 SVMP      0.986 0.996 0.999 0.0957 0.997
## 12 SVMR      0.996 0.996 0.999 0.083  0.997
```

Hold-out Test Performance Table

```
## comparing cv models to existing
ho_perf_table <- rbind(rf_RGB_HOresults, rf_RGC_HOresults, rf_HSV_HOresults,
                       cvglm_HSV_HOresults, svmL_RGB_HOresults, svmP_RGB_HOresults,
                       svmR_RGB_HOresults, svmL_RGC_HOresults, svmR_HSV_HOresults,
                       glm_RGB_HOresults, glm_RGC_HOresults, lda_RGC_HOresults) %>%
                    data.frame()

ho_perf_table[, 2:6] <- lapply(lapply(ho_perf_table[, 2:6],
                                      as.character), as.numeric)
ho_perf_table[, 2:6] <- ho_perf_table[, 2:6] %>%
                          round(4)

ho_perf_table <- ho_perf_table[order(-(ho_perf_table$AUROC * 1.2 +
                                       ho_perf_table$Accuracy * 1.2 +
                                       ho_perf_table$TPR * 2.4 +
                                       ho_perf_table$FPR * -0.2 +
                                       ho_perf_table$Precision * 0.2) / 5), ]

rownames(ho_perf_table) <- NULL
ho_perf_table
```

```
##    Color  AUROC Accuracy    TPR    FPR Precision Model
## 1    RGB 0.9982   0.9961 0.9963 0.0420    0.9997  SVMP
## 2    RGC 0.9995   0.9924 0.9924 0.0153    0.9999  SVML
```

```
## 3    RGC 0.9995   0.9914 0.9914 0.0123    0.9999   GLM
## 4    RGC 0.9995   0.9914 0.9914 0.0123    0.9999   LDA
## 5    RGB 0.9994   0.9910 0.9910 0.0122    0.9999   GLM
## 6    RGB 0.9991   0.9855 0.9855 0.0124    0.9999  SVML
## 7    HSV 0.9929   0.9857 0.9856 0.0065    1.0000 CVGLM
## 8    RGC 0.9956   0.9781 0.9780 0.0082    0.9999    RF
## 9    RGB 0.9793   0.9951 0.9969 0.2454    0.9982    RF
## 10   HSV 0.9853   0.9708 0.9723 0.2320    0.9983    RF
## 11   HSV 0.9776   0.9751 0.9783 0.4703    0.9965  SVMR
## 12   RGB 0.9763   0.9883 0.9937 0.7537    0.9945  SVMR
```

```
## grouped performance statistics
ho_perf_table %>%
  group_by(Color) %>%
  summarize(mAUROC = mean(AUROC), mAcc = mean(Accuracy),
            mTPR = mean(TPR), mFPR = mean(FPR),
            mPrc = mean(Precision))
```

```
## # A tibble: 3 x 6
##   Color mAUROC  mAcc  mTPR   mFPR  mPrc
##   <chr>  <dbl> <dbl> <dbl>  <dbl> <dbl>
## 1 HSV    0.985 0.977 0.979 0.236  0.998
## 2 RGB    0.990 0.991 0.993 0.213  0.998
## 3 RGC    0.999 0.988 0.988 0.0120 1.00
```

```
ho_perf_table %>%
  group_by(Model) %>%
  summarize(mAUROC = mean(AUROC), mAcc = mean(Accuracy),
            mTPR = mean(TPR), mFPR = mean(FPR),
            mPrc = mean(Precision))
```

```
## # A tibble: 7 x 6
##   Model mAUROC  mAcc  mTPR   mFPR  mPrc
##   <chr>  <dbl> <dbl> <dbl>  <dbl> <dbl>
## 1 CVGLM  0.993 0.986 0.986 0.0065 1
## 2 GLM    0.999 0.991 0.991 0.0122 1.00
## 3 LDA    1.00  0.991 0.991 0.0123 1.00
## 4 RF     0.987 0.981 0.982 0.162  0.999
## 5 SVML   0.999 0.989 0.989 0.0138 1.00
## 6 SVMP   0.998 0.996 0.996 0.042  1.00
## 7 SVMR   0.977 0.982 0.986 0.612  0.996
```

Conclusions

To preface my conclusions for this project, I would like to discuss some early transformations I performed on the Haiti Pixels data set. The initial `Class` variable - designating the different ground surfaces marked by their color (in RGB: red, green, blue) - was overcomplicated. The objective of this analysis was to specifically identify blue tarps, which mark survivors, so I simplified the `Class` variable by creating the `Tarp` variable, which is a simple binary categorical variable that designates whether or not the observation is a blue tarp. For the purpose of this analysis, there is no use of differentiating between the other classes - rescue crews do not need to search for areas of soil amongst areas of vegetation.

When processing my imagery data from Haiti, I chose to initially compare three different formats to quantify the color data - the standard RGB format, as provided by the dataset, a normalized RGB format (rg chromaticity), and the HSV format, which represents hue, saturation, and intensity. The reasoning behind this methodology is to further the goal of accurately identifying the *blue* tarps, so perhaps a reformatting of the data in order to numerically separate blue tarp observations further would improve my models.

While the standard RGB format in the dataset provides red, green, and blue values as integers up to 255, the normalized RGB format divides each color value by the total value of red, blue, and green combined, in order to make each color value (for red, green, and blue) a decimal value out of 1. This format is commonly used in computer vision techniques, as it reduces the effects of varying lighting in the identification of a color.

Figure 1: Source: Jephraim Manansala via Medium

Above is an example from a Medium blog of standard images, compared to how they appear with normalized RGB values. With normalized values (and then further transformation), key colors are heavily emphasized, as lighting is equalized. As the blog observes, normalizing color data is especially important for computer vision for identifying low-contrast objects that would be otherwise difficult to detect without a human's sense of dimesionality and depth, which in the blog's example, are fruits hidden within the tree's leaves. For our example, being able to spot blue tarps from the air after a natural disaster in Haiti would be an equally useful application of this technique.

As the boxplots show, normalizing the RGB values greatly reduces the variance of the values, potentially making blue tarps more identifiable in modeling. Red and blue values for blue tarp vs. non-tarp values experience little overlap.

The second color format shown is the HSV color model, representing hue, saturation, and value. Instead of displaying color through a mixture of red, green, and blue values, hue alone represents color as part of a 360 degree wheel, while saturation and value dictate the intensity and brightness of the color.

While normalized RGB is useful to isolate color - whether or not the observation is blue, which may mean it is a blue tarp - HSV would allow us to analyze whether saturation and value alone could be useful predictors.

Boxplots of HSV values show even greater separation between blue tarp and non-tarp samples; hues for observations where there is a blue tarp are very clearly separate from those of non-tarp observations! Saturation and value experience some overlap between blue tarp and non-tarp observations, but have clearly different distributions.

Conclusion #1

When sorting variables by the weighted average of the five metrics (AUROC, Accuracy, TPR, FPR, Precision), the random forest model conducted on HSV color data come out on top when considering more cross-validated models, though most results are relatively similar in performance.

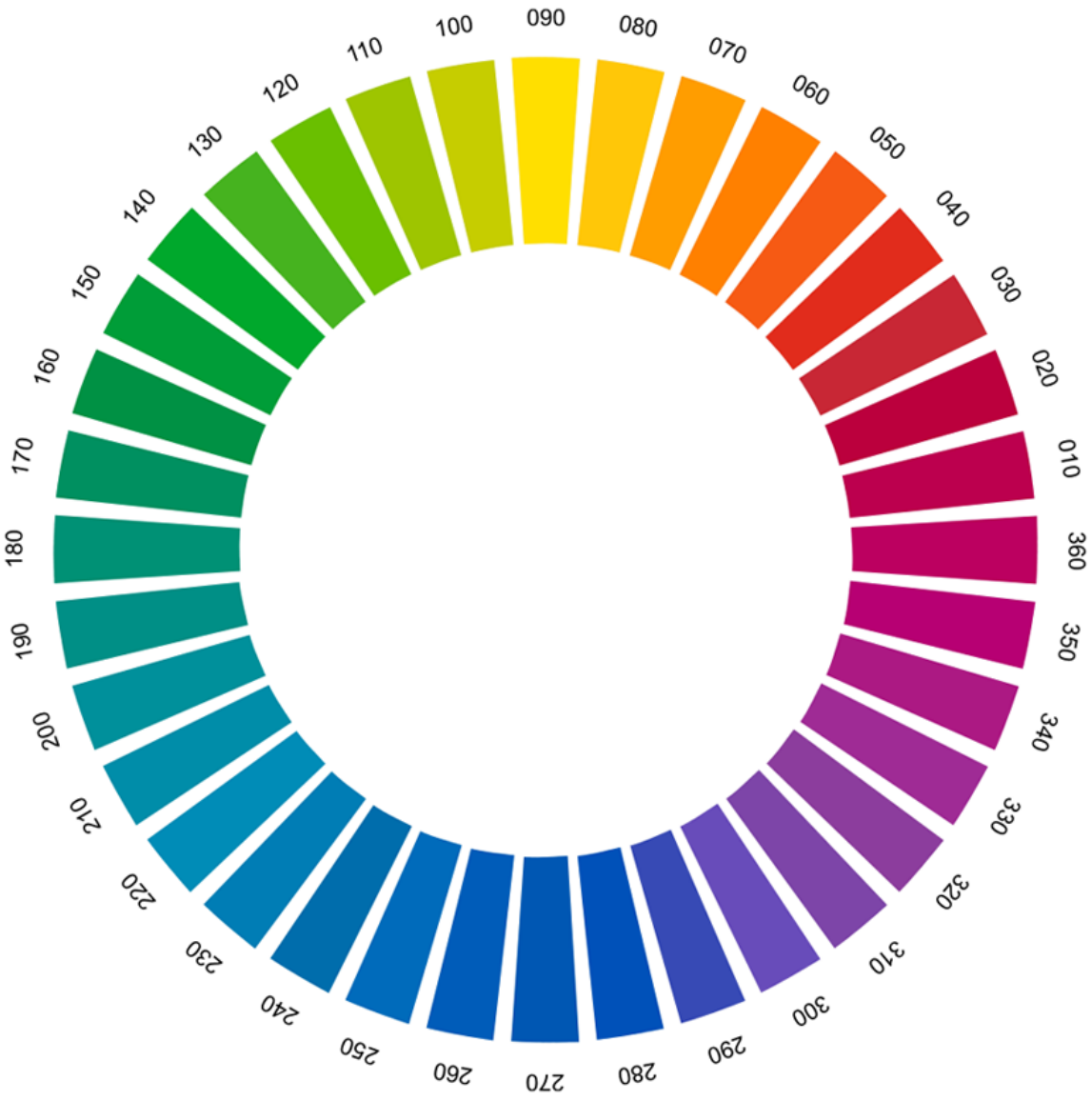


Figure 2: Source: Stack Exchange

When not considering the cross-validated models, the simpler logistic regression, linear discriminant analysis, and quadratic discriminant analysis models came up on top when testing against the original data. Perhaps their simplicity helped against overfit in comparison to other more complex models.

Given how most models had very similarly good performance, it's also worth considering how my weighting of metrics found these models favorable. When training on the normalized RGB data, these three models had *relatively* high false positive rates compared to others, but as this held little weight, it didn't negatively impact their "rankings".

Conclusion #2

Simplifying the RGB color model into the normalized RGB and HSV color models seemed to yield very accurate models, but did not seem to yield significantly better results in model prediction than standard RGB values. In the traditional RGB model, there is heavy interaction between the red, green, and blue variables in order to produce a color - in this case, the blue of a blue tarp that we are looking for - but a normalized RGB model simplifies color down to interactions between red and green alone. The HSV color model, which performed slightly better than the standard and normalized RGB models, simplifies color even further, as hue captures the basics of a "color" (whether something is red, blue, green, or even orange, purple, etc.) into one variable - hue.

As imagery data may be taken at different times of day, resulting in variable levels of sunlight, or blue tarps may be under varying amounts of shade, and it is important to identify tarps regardless of lighting levels. Initial values showed blue values in the normalized RGB model to be rather insignificant as well, which makes sense due to the nature of normalized values. All values are calculated relative to red and green values, so it makes more sense for other models to be calculated based upon those two colors in the datasets with normalized RGB data.

So while the models based upon normalized RGB data did not have the *absolute best* results, they did tend to do better on average than models based upon standard RGB values, especially in terms of false positive rate.

As a final note, I was slightly shocked to see that models based upon HSV data performed worse on average than standard RGB data. Perhaps this data does not consist of very bright and punchy colors, which would be best captured by variables such as hue and saturation.

Table 1: Average Metrics for Models, Grouped by Color Model

Color	mAUROC	mAcc	mTPR	mFPR	mPrc
HSV	0.9832750	0.9897000	0.9944167	0.1535750	0.9950083
RGB	0.9879167	0.9632083	0.9661833	0.1237500	0.9957500
RGC	0.9906000	0.9906917	0.9930167	0.0795583	0.9973750

Table 2: Average Metrics when Testing on Hold-Out Data, Grouped by Color Model

Color	mAUROC	mAcc	mTPR	mFPR	mPrc
HSV	0.9852667	0.977200	0.9787333	0.2362667	0.9982667
RGB	0.9904600	0.991200	0.9926800	0.2131400	0.9984400
RGC	0.9985250	0.988325	0.9883000	0.0120250	0.9999000

Conclusion #3

As briefly discussed in class, weighting the five metrics used to evaluate models - area under the ROC curve (AUROC), accuracy, true positive rate, false positive rate, and precision - is a subjective process that requires some qualitative thinking.

When ranking models, I started with very low weightings for false positive rate and precision, and specifically used a negative value for weighting false positive rate (as higher values are worse). Within the context of disaster response, resources are presumed to be relatively plentiful, and a slight waste of manpower and resources chasing false positives - in this case, what we believe to be blue tarps, but are not - is not very critical. Likewise, precision is not as key as true positive rate, as it does not deal in the success in avoiding false negatives, which is truly critical. Missing false negatives - what a model could not correctly identify as a blue tarp - is critical to Haiti's earthquake response, which is why I weighted true positive rate the heaviest. Accuracy and AUROC are next, as it they are good overall measures of model performance, but within the context of earthquake response, I do not believe them to be the most important metrics.

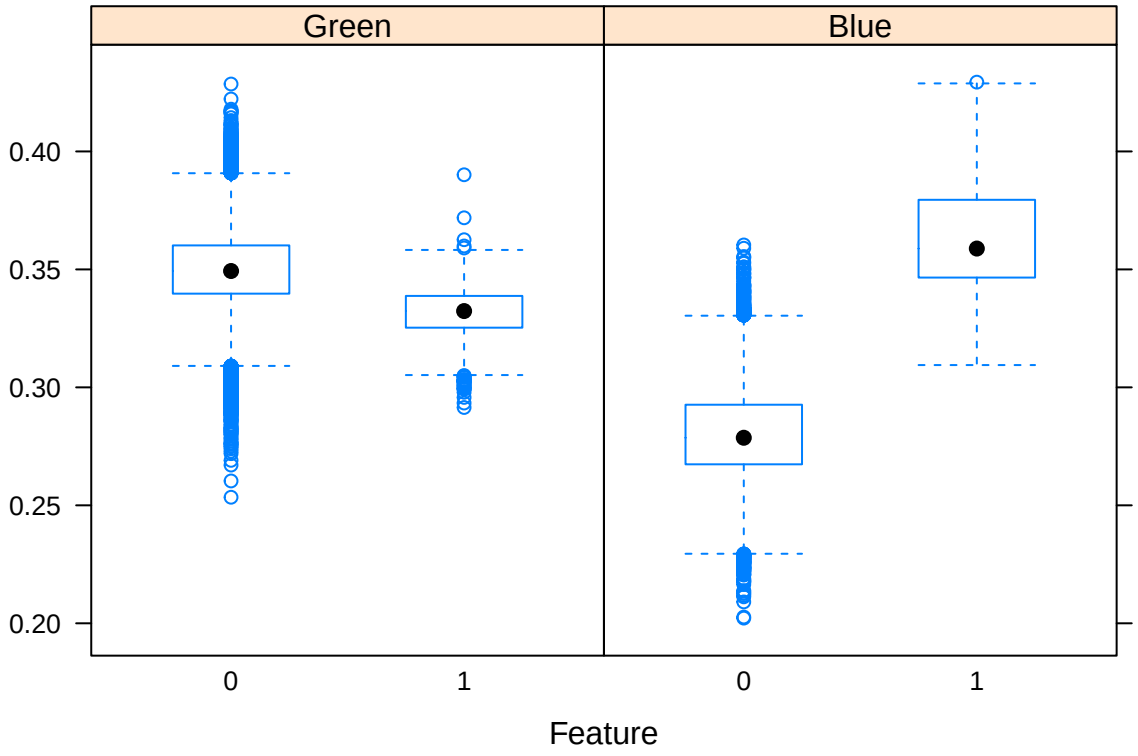
Conclusion #4

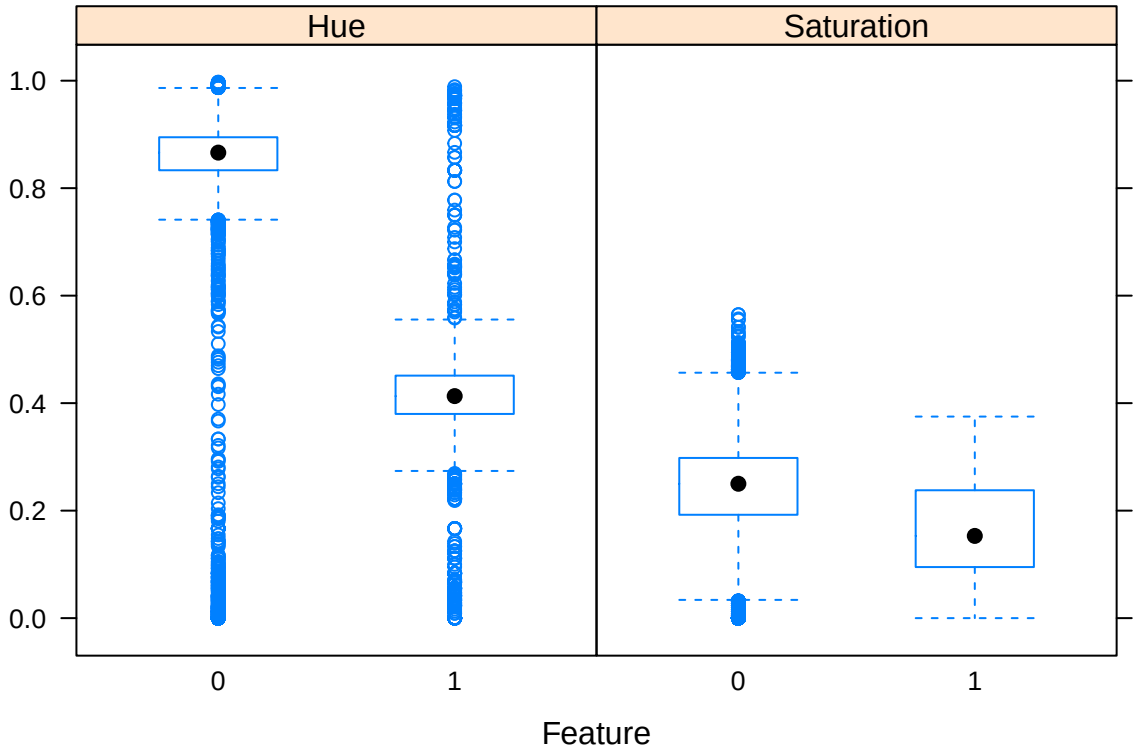
Support vector machines were top performers when testing on the hold-out data, surprisingly with a variety of kernels and using different formats for the color data. A linear and polynomial (with a degree of 2) SVM for the RGB data ended up being the top performing model when testing on the hold-out data. A total of five out of the nine total support vector machines I created were ultimately chosen to be within my top-twelve to use in testing with the hold-out data, with radial and linear models being the most popular.

As noted by Manuel Fernández-Delgado, Eva Cernadas, Senén Barro, and Dinani Amorim's *Do we Need Hundreds of Classifiers to Solve Real World Classification Problems?*, support vector machines were consistently top performers for a variety of prediction problems across a wide range of datasets, and it seems that the Haiti earthquake data is no exception.

Conclusion #5

Fernández-Delgado and his cohorts deemed random forests to be the overall best prediction method in their experiments, and the Haiti earthquake data is no exception here either. While it fell behind when testing with the hold-out data, and it was notably the best model when testing with just the original Haiti Pixels data.


Green
Blue
0.40
0.35
0.30
0.25
0.20
0
1
0
1
Feature



Due to the predictor variables having overlapping but distinct distributions, the ability for a random forest model to find "splits" is incredibly powerful - while able to avoid the overfitting issues that could prove to be disastrous with the inevitably large quantity of testing data that would come with the imagery data necessary for disaster response. Despite the fact that it did not prove to be the absolute best model when testing against the hold-out data, I would not discount it when considering tests with future data.

Conclusion #6

My final conclusion ponders possibilities if the original training data had as many parameters as the hold-out data, or more. Given how the hold-out data contains coordinates for latitude and longitude, perhaps the observation's town or proximity to the nearest town could have been used, and perhaps the population or population density of the town could be a factor. Any presence of blue in a (formerly, given the earthquake) populated area like that would be more likely to be a blue tarp, making location or predictors derived from location a potentially valuable predictor for training.

While the timeframe of the Haiti earthquake itself is well-known, more precise timestamps for the observations along with a factor variable for weather conditions may prove to be a useful predictor for training. As shown by the normalized RGB and HSV color models, normalizing color data to account for lighting can be a very useful technique for modeling and analysis of imagery data, but having variables that may give some indication to the conditions in which the photo was taken would allow for greater insight, and potentially further transformations to the color data to better isolate desired colors. Knowing the time of day the photo was taken or how cloudy it was when the photo was taken could be very crucial in improving the model.

Overall, I still believe that the modeling done with the training data at hand proved to be effective. The normalization and transformation of the original RGB values given still gave very accurate models for predicting the presence of other earthquake survivors seeking rescue.