DS 6030 Project Pt. 2

RJ Cubarrubia

2023-04-27

```
library(tidyverse)
library(caret)
library(ROCR)
library(kableExtra)
library(class)
```

Intro

After a catastrophic earthquake in 2010, rescue workers in Haiti rushed to aid displaced people who were using blue tarps as makeshift shelters. High resolution imagery was collected via aircraft with the hopes that a the images could be searched for blue tarps. If the blue tarps could be spotted, potential survivors could be identified and rescue workers and resources could be directly sent their way. But with thousands of images a day, aid workers weren't able to pore over all the pixels and communicate their findings to their fellow rescue workers on the ground in time. In came data-mining algorithms and predictive modeling — if the models could scan the images and identify the pixels with blue tarps well enough, the aid and rescue workers might be able to find survivors before it was too late.

This project involves training and test data sets with actual data collected from relief efforts. First, we'll be wrangling and exploring the training and test data. Then, we'll build many models, fit them to the training data and evaluate their performance with the test data. Finally, we'll sum up our results, compare the models and see how effective we were at aiding survivors of a devastating natural disaster.

Data Wrangling and EDA

Training Data

Let's start with our training data set.

```
haiti.train <- read.csv('HaitiPixels.csv', row.names = NULL)
```

```
nrow(haiti.train)
```

```
#> [1] 63241
```

```
ncol(haiti.train)
```

```
#> [1] 4
```

```
table(haiti.train$Class)
```

```
#>
#>        Blue Tarp          Rooftop             Soil Various Non-Tarp
#>             2022             9903            20566             4744
#>       Vegetation
#>            26006
```

```
2022 / nrow(haiti.train)
```

```
#> [1] 0.03197293
```

```
set.seed(710)

haiti.train.rows_shuffled <- sample(nrow(haiti.train))
haiti.train <- haiti.train[haiti.train.rows_shuffled, ]
```

```
haiti.train$isbluetarp <- ifelse(haiti.train$Class == 'Blue Tarp', 1, 0)
```

```
haiti.train$isbluetarp <- as.factor(haiti.train$isbluetarp)
```

```
levels(haiti.train$isbluetarp)
```

```
#> [1] "0" "1"
```

```
levels(haiti.train$isbluetarp) = c('False', 'True')
```

```
levels(haiti.train$isbluetarp)
```

```
#> [1] "False" "True"
```

```
haiti.train$isbluetarp <- factor(haiti.train$isbluetarp,
                           levels = c('True', 'False'))
```

```
levels(haiti.train$isbluetarp)
```

```
#> [1] "True"  "False"
```

```
sum(is.na(haiti.train))
```

```
#> [1] 0
```

```
haiti.train <- subset(haiti.train, select = -c(Class))
```

Looking at our training data, we have 63241 total observations. Our "Class" column has five different classes. Our class type of interest, "Blue Tarp", only accounts for about 3.2% of the observations. The other columns — "Red", "Blue" and "Green" — are our predictors. Since each of these is essential in determining overall pixel color, they'll all be used in our models. I created a new column, "isbluetarp", that labels "True" if the observation is a "Blue Tarp" and "False" if it's not. Since we're interested in only identifying blue tarps, we're dealing with a one-vs-rest type of classification for our models. We have no NaN values in our data frame. Since the data frame is ordered by classification (each class is grouped together in chunks), I shuffled the data so we don't violate our normality assumption that our error terms are uncorrelated with each other (just to be safe). Finally, I've removed the "Class" column since it's no longer important; we only needed the column to identify blue tarps from the rest and we're uninterested in the specific class labels of the other objects.

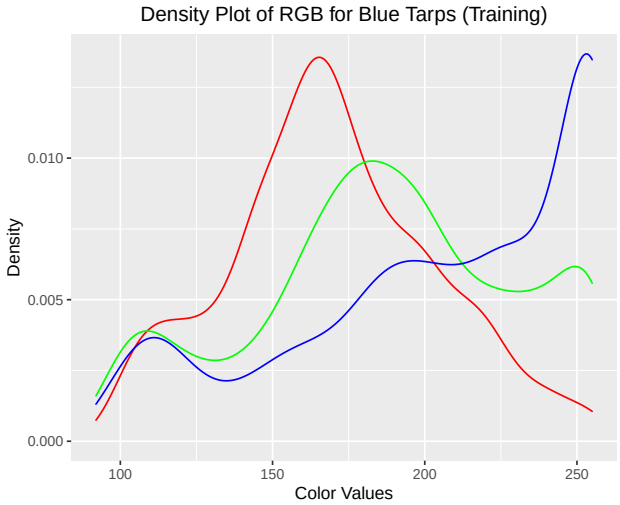
Let's take a look at the distributions of RGB in our training set.

```
haiti.train %>%
ggplot() +
  geom_density(aes(x = Red), color = 'Red') +
  geom_density(aes(x = Green), color = 'Green') +
  geom_density(aes(x = Blue), color = 'Blue') +
  labs(x = 'Color Values',
       y = 'Density') +
  ggtitle('Density Plot of RGB (Training)') +
  theme(plot.title = element_text(hjust = 0.5))
```

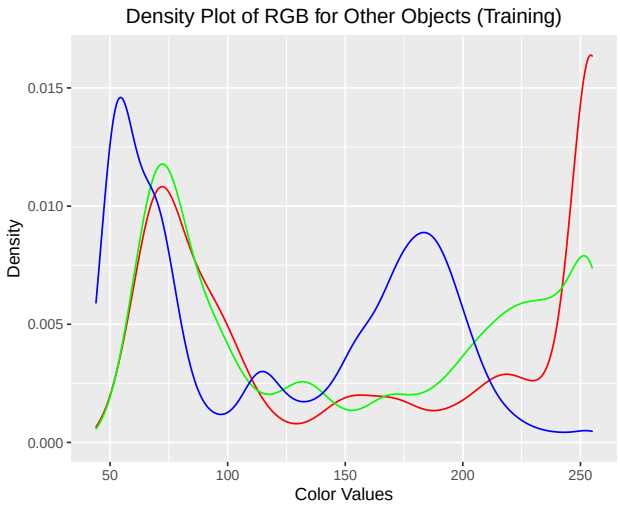
Our distributions are very different from normal and while the ranges are the same, the curves hardly overlap.

Let's compare the distributions of RGB values of blue tarps and the rest.

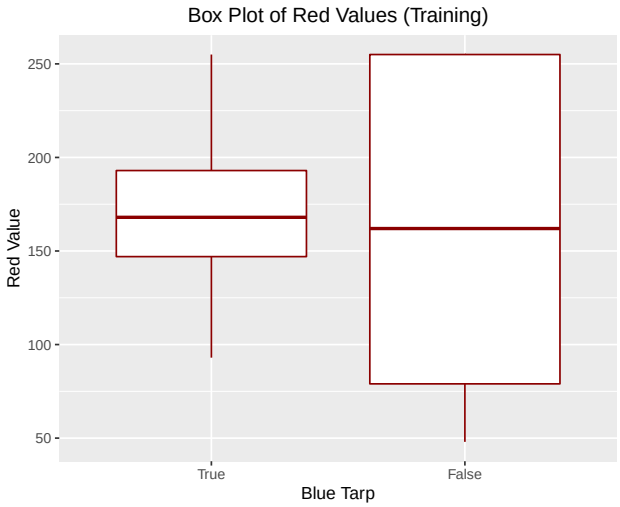
```
filter(haiti.train, isbluetarp == 'True') %>%
ggplot() +
  geom_density(aes(x = Red), color = 'Red') +
  geom_density(aes(x = Green), color = 'Green') +
  geom_density(aes(x = Blue), color = 'Blue') +
  labs(x = 'Color Values',
       y = 'Density') +
  ggtitle('Density Plot of RGB for Blue Tarps (Training)') +
  theme(plot.title = element_text(hjust = 0.5))
```



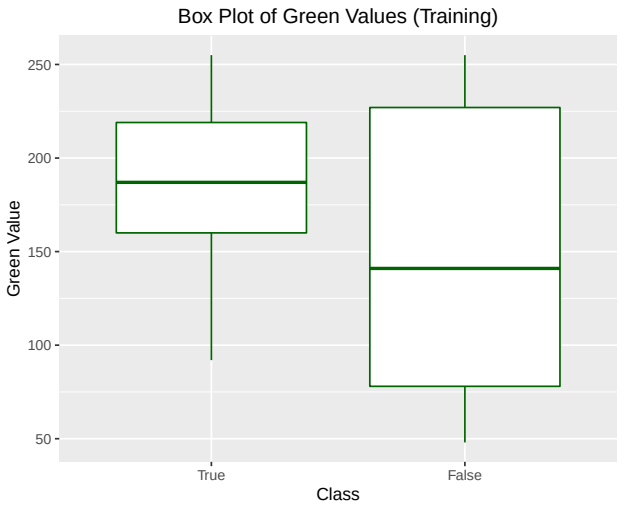

```
filter(haiti.train, isbluetarp == 'False') %>%
ggplot() +
  geom_density(aes(x = Red), color = 'Red') +
  geom_density(aes(x = Green), color = 'Green') +
  geom_density(aes(x = Blue), color = 'Blue') +
  labs(x = 'Color Values',
       y = 'Density') +
  ggtitle('Density Plot of RGB for Other Objects (Training)') +
  theme(plot.title = element_text(hjust = 0.5))
```




```
haiti.train %>%
ggplot(aes(x = isbluetarp, y = Red)) +
  geom_boxplot(col = 'dark red') +
  labs(x = 'Blue Tarp',
       y = 'Red Value') +
  ggtitle('Box Plot of Red Values (Training)') +
  theme(plot.title = element_text(hjust = 0.5))
```




```
haiti.train %>%
ggplot(aes(x = isbluetarp, y = Green)) +
  geom_boxplot(col = 'dark green') +
  labs(x = 'Class',
       y = 'Green Value') +
  ggtitle('Box Plot of Green Values (Training)') +
  theme(plot.title = element_text(hjust = 0.5))
```



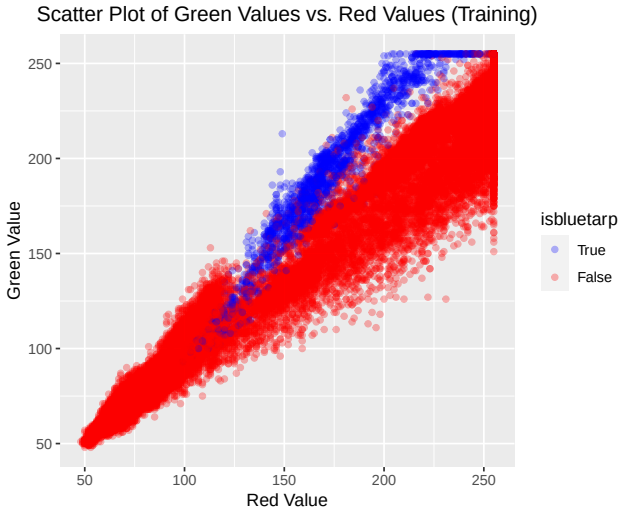

```
haiti.train %>%
ggplot(aes(x = isbluetarp, y = Blue)) +
  geom_boxplot(col = 'dark blue') +
  labs(x = 'Class',
       y = 'Blue Value') +
  ggtitle('Box Plot of Blue Values (Training)') +
  theme(plot.title = element_text(hjust = 0.5))
```

Unsurprisingly, blue tarps have a high density of high Blue values. They surprisingly have a high density of mid-range Red values. The other objects have a high density of high Red values and a high density of low Blue values. The distribution of Green values never hits the density peaks of Red or Blue values for either blue tarps or the other objects. Blue tarps have a pretty distinct distribution in their boxplots, with RGB values having pretty narrow inter-quartile ranges. Their Red values are generally lower than the Green values, which are lower than the Blue values. The other objects have predictably wide IQR's across their RGB values, especially with Red values, but their Blue values have a comparably smaller IQR to the other colors with generally lower values.

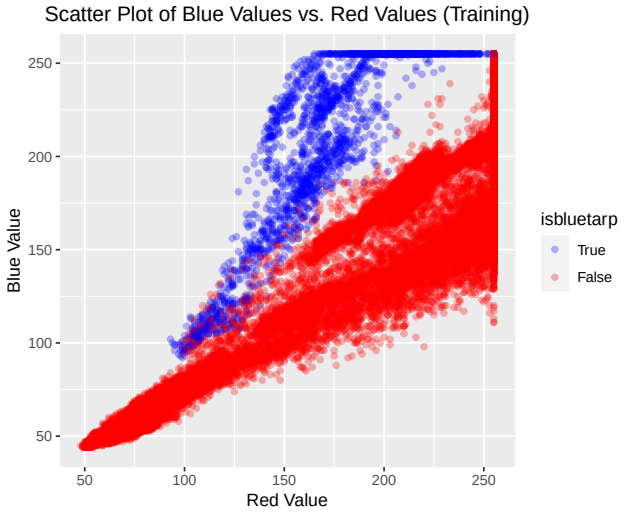
Let's check out some scatterplots to see how separable the blue tarps are from the other objects.

```
haiti.train %>%
ggplot(aes(x = Red, y = Green)) +
  geom_point(aes(color = isbluetarp),
             alpha = 0.3) +
```

```
  scale_color_manual(values=c('Blue', 'Red')) +
  labs(x = 'Red Value',
       y = 'Green Value') +
  ggtitle('Scatter Plot of Green Values vs. Red Values (Training)') +
  theme(plot.title = element_text(hjust = 0.5))
```

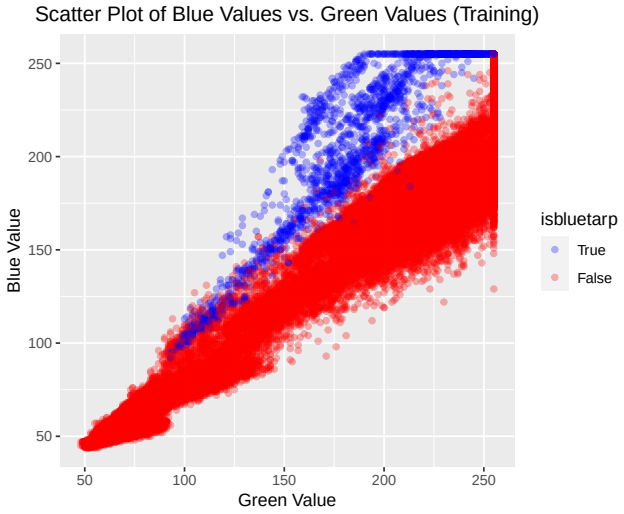



```
haiti.train %>%
ggplot(aes(x = Red, y = Blue)) +
  geom_point(aes(color = isbluetarp),
             alpha = 0.3) +
  scale_color_manual(values=c('Blue', 'Red')) +
  labs(x = 'Red Value',
       y = 'Blue Value') +
  ggtitle('Scatter Plot of Blue Values vs. Red Values (Training)') +
  theme(plot.title = element_text(hjust = 0.5))
```




```
haiti.train %>%
ggplot(aes(x = Green, y = Blue)) +
  geom_point(aes(color = isbluetarp),
             alpha = 0.3) +
  scale_color_manual(values=c('Blue', 'Red')) +
```

```
  labs(x = 'Green Value',
       y = 'Blue Value') +
  ggtitle('Scatter Plot of Blue Values vs. Green Values (Training)') +
  theme(plot.title = element_text(hjust = 0.5))
```



Our predictors all seem correlated. It also seems blue tarps are pretty clearly separable from the other objects. The separation appears somewhat close to linear but not quite. This suggests the more flexible models will likely have better performance.

Training Set

Now let's move on to our test data, which is quite large and in many different files. We have three reference images — two that are 831 pixels by 636 pixels and one that is 965 pixels by 826 pixels. These pictures individually contain 528516 pixels and 797090 pixels, respectively. Combined, these three pictures contain 1854122 pixels total.

Since the each raw data file contains either entirely blue tarps or entirely other objects, I'll start with the blue tarp files. One of the blue tarp files has a duplicate, with one version containing metadata as well as the complete data and the other only containing "B1", "B2" and "B3" values. These likely match up to Red, Green and Blue values, respectively, but we'll explore and confirm that later.

```
bluetarp.test.1 <- read.table('orthovnir067_ROI_Blue_Tarps.txt',
                              skip = 8)
```

Looking at the raw text file as well as the imported data, we have 10 features. Our raw data contains features for ID (really just a row number), as well as X and Y coordinates (given in two versions) and latitude and longitude. We don't need these. Our last three features are the "B1", "B2" and "B3" values and the ones we want to keep. Finally, since these are all blue tarps, we'll add a "isbluetarp" column like the one in our training data with everything labeled as "True".

```
bluetarp.test.1 <- subset(bluetarp.test.1, select = -c(V1, V2, V3, V4, V5, V6,
                                                       V7))
```

```
bluetarp.test.1$isbluetarp <- 'True'
```

We have two more similar blue tarp files, so we'll repeat the process.

```
bluetarp.test.2 <- read.table('orthovnir069_ROI_Blue_Tarps.txt',
                              skip = 8)
```

```
bluetarp.test.2 <- subset(bluetarp.test.2, select = -c(V1, V2, V3, V4, V5, V6,
                                                       V7))
```

```
bluetarp.test.2$isbluetarp <- 'True'
```

```
bluetarp.test.3 <- read.table('orthovnir078_ROI_Blue_Tarps.txt',
                              skip = 8)
```

```
bluetarp.test.3 <- subset(bluetarp.test.3, select = -c(V1, V2, V3, V4, V5, V6,
                                                       V7))
```

```
bluetarp.test.3$isbluetarp <- 'True'
```

```
nrow(bluetarp.test.1) + nrow(bluetarp.test.2) + nrow(bluetarp.test.3)
```

```
#> [1] 14480
```

So far we have 14480 blue tarp observations.

Now let's tackle the data containing the other objects. These are structured in the same way as our blue tarp files, so we'll read them in similarly but give the data sets a "False" label for the "isbluetarp" column.

```
notbluetarp.test.1 <- read.table('orthovnir057_ROI_NON_Blue_Tarps.txt',
                              skip = 8)
```

```
notbluetarp.test.1 <- subset(notbluetarp.test.1, select = -c(V1, V2, V3, V4,
                                                             V5, V6, V7))
```

```
notbluetarp.test.1$isbluetarp <- 'False'
```

```
notbluetarp.test.2 <- read.table('orthovnir067_ROI_NOT_Blue_Tarps.txt',
                              skip = 8)
```

```
notbluetarp.test.2 <- subset(notbluetarp.test.2, select = -c(V1, V2, V3, V4,
                                                             V5, V6, V7))
```

```
notbluetarp.test.2$isbluetarp <- 'False'
```

```
notbluetarp.test.3 <- read.table('orthovnir069_ROI_NOT_Blue_Tarps.txt',
                              skip = 8)
```

```
notbluetarp.test.3 <- subset(notbluetarp.test.3, select = -c(V1, V2, V3, V4,
                                                             V5, V6, V7))
```

```
notbluetarp.test.3$isbluetarp <- 'False'
```

```
notbluetarp.test.4 <- read.table('orthovnir078_ROI_NON_Blue_Tarps.txt',
                              skip = 8)
```

```
notbluetarp.test.4 <- subset(notbluetarp.test.4, select = -c(V1, V2, V3, V4,
                                                             V5, V6, V7))
```

```
notbluetarp.test.4$isbluetarp <- 'False'
```

```
nrow(notbluetarp.test.1) + nrow(notbluetarp.test.2) +
  nrow(notbluetarp.test.3) + nrow(notbluetarp.test.4)
```

```
#> [1] 1989697
```

We have almost 2 million(!) observations of non blue tarp objects.

Now, let's combine all the test data into one dataframe.

```
haiti.test <- rbind(bluetarp.test.1, bluetarp.test.2, bluetarp.test.3,
                    notbluetarp.test.1, notbluetarp.test.2,
                    notbluetarp.test.3, notbluetarp.test.4)
```

```
set.seed(710)

haiti.test.rows_shuffled <- sample(nrow(haiti.test))
haiti.test <- haiti.test[haiti.test.rows_shuffled, ]
```

```
nrow(haiti.test)
```

```
#> [1] 2004177
```

Our test set contains 2004177 observations(!).

```
14480 / 2004177
```

```
#> [1] 0.007224911
```

Blue tarps consist of less than 1% of our test data. This is a pretty big difference from our training set.

```
haiti.test$isbluetarp <- as.factor(haiti.test$isbluetarp)
```

```
levels(haiti.test$isbluetarp)
```

```
#> [1] "False" "True"
```

```
haiti.test$isbluetarp <- factor(haiti.test$isbluetarp,
                                levels = c('True', 'False'))
```

```
levels(haiti.test$isbluetarp)
```

```
#> [1] "True"  "False"
```

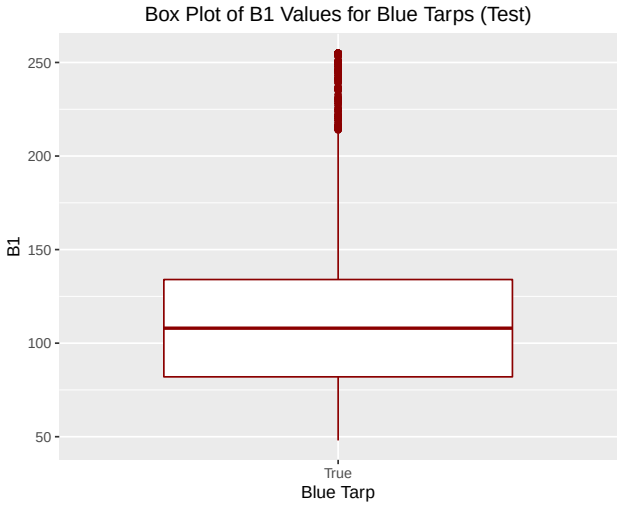
```
sum(is.na(haiti.test))
```

```
#> [1] 0
```

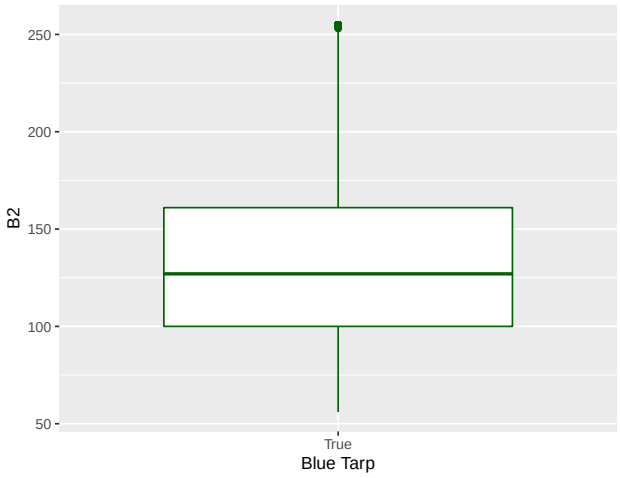
```
colnames(haiti.test)[colnames(haiti.test) == 'V8'] <- 'B1'
colnames(haiti.test)[colnames(haiti.test) == 'V9'] <- 'B2'
colnames(haiti.test)[colnames(haiti.test) == 'V10'] <- 'B3'
```

We still need to determine how the "B1", "B2" and "B3" values match up with Red, Green and Blue. The metadata in the raw text files for the test data each contains an entry regarding the ROI's (Region of Interest's) RGB value, suggesting an inherent order to the "B1", "B2" and "B3" features. Furthermore, RGB values are universally expressed in that order, so much so that RGB itself is a colloquialism for color models in general. Our raw data is also well curated and organized. It's likely that "B1", "B2" and "B3" match up with Red, Green and Blue, respectively, but we need further confirmation. Let's start by making boxplots of each feature for observations that are blue tarps. Since the blue tarps in the training data showed pretty distinct boxplots for Red, Green and Blue, this should give us a solid baseline for comparison.

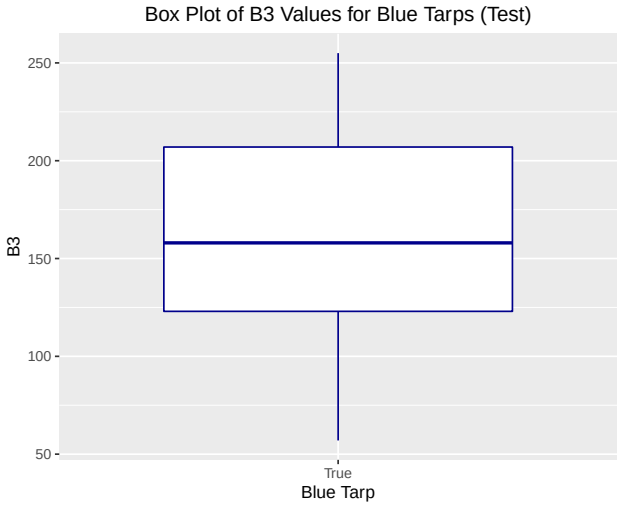
```
filter(haiti.test, isbluetarp == 'True') %>%
  ggplot(aes(x = isbluetarp, y = B1)) +
  geom_boxplot(col = 'dark red') +
  labs(x = 'Blue Tarp',
       y = 'B1') +
  ggtitle('Box Plot of B1 Values for Blue Tarps (Test)') +
  theme(plot.title = element_text(hjust = 0.5))
```




```
filter(haiti.test, isbluetarp == 'True') %>%
  ggplot(aes(x = isbluetarp, y = B2)) +
  geom_boxplot(col = 'dark green') +
  labs(x = 'Blue Tarp',
       y = 'B2') +
  ggtitle('Box Plot of B2 Values for Blue Tarps (Test)') +
  theme(plot.title = element_text(hjust = 0.5))
```




```
filter(haiti.test, isbluetarp == 'True') %>%
  ggplot(aes(x = isbluetarp, y = B3)) +
  geom_boxplot(col = 'dark blue') +
  labs(x = 'Blue Tarp',
       y = 'B3') +
  ggtitle('Box Plot of B3 Values for Blue Tarps (Test)') +
  theme(plot.title = element_text(hjust = 0.5))
```

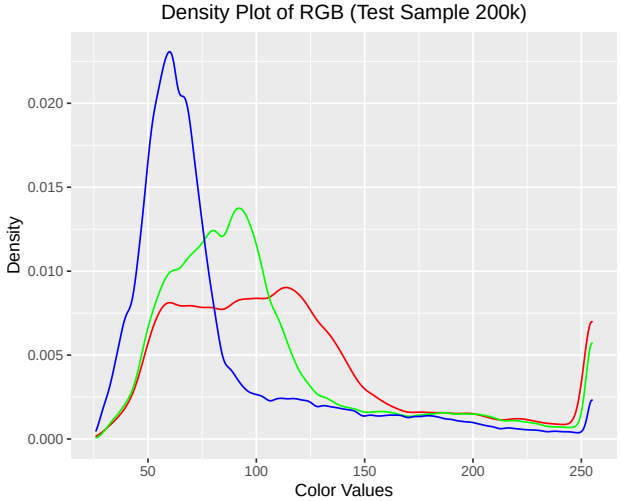


These test boxplots for blue tarps appear to be reasonable matches to the training boxplots. "B1" had a IQR and median that was less than "B2", which in turn was less than "B3". This follows the pattern from the training data, where Red's IQR and median was less than Green's IQR and median, which was then less than Blue's IQR and median. I think it's reasonable to relabel "B1" as Red, "B2" as Green and "B3" as Blue.

```
colnames(haiti.test)[colnames(haiti.test) == 'B1'] <- 'Red'
colnames(haiti.test)[colnames(haiti.test) == 'B2'] <- 'Green'
colnames(haiti.test)[colnames(haiti.test) == 'B3'] <- 'Blue'
```

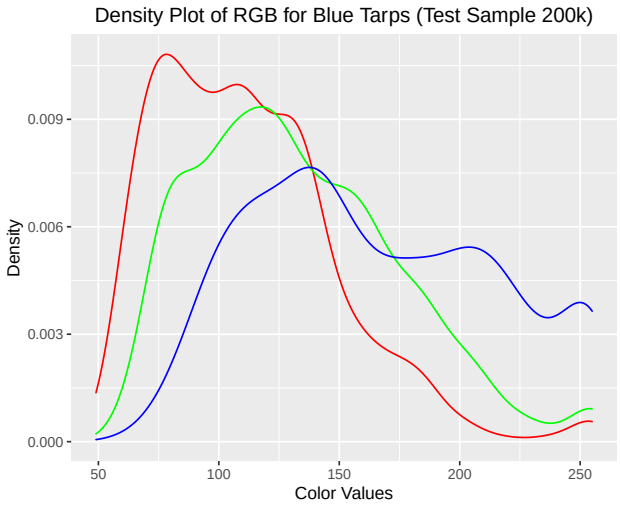
Let's check out the distribution of Red, Green and Blue in our test data. Our test data has over 2 million rows, so we'll randomly sample 200000 observations for density plot.

```
set.seed(710)

haiti.test[sample(nrow(haiti.test), 200000), ] %>%
ggplot() +
  geom_density(aes(x = Red), color = 'Red') +
  geom_density(aes(x = Green), color = 'Green') +
  geom_density(aes(x = Blue), color = 'Blue') +
  labs(x = 'Color Values',
       y = 'Density') +
  ggtitle('Density Plot of RGB (Test Sample 200k)') +
  theme(plot.title = element_text(hjust = 0.5))
```



This is considerably different than our training data. We have a very high density of low Blue values.

```
set.seed(710)

filter(haiti.test[sample(nrow(haiti.test), 200000), ], isbluetarp == 'True') %>%
ggplot() +
  geom_density(aes(x = Red), color = 'Red') +
  geom_density(aes(x = Green), color = 'Green') +
  geom_density(aes(x = Blue), color = 'Blue') +
  labs(x = 'Color Values',
       y = 'Density') +
  ggtitle('Density Plot of RGB for Blue Tarps (Test Sample 200k)') +
  theme(plot.title = element_text(hjust = 0.5))
```




```
set.seed(710)

filter(haiti.test[sample(nrow(haiti.test), 200000), ], isbluetarp == 'False') %>%
ggplot() +
  geom_density(aes(x = Red), color = 'Red') +
  geom_density(aes(x = Green), color = 'Green') +
  geom_density(aes(x = Blue), color = 'Blue') +
  labs(x = 'Color Values',
       y = 'Density') +
  ggtitle('Density Plot of RGB for Other Objects (Test Sample 200k)') +
  theme(plot.title = element_text(hjust = 0.5))
```

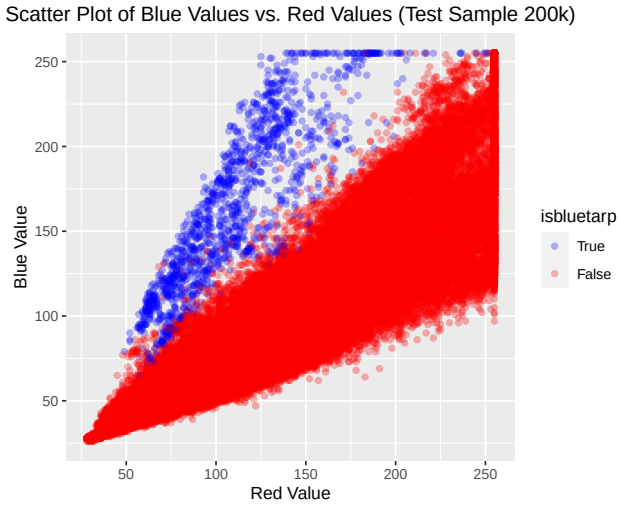
These also appear quite different than the ones from the training data, although the non blue tarp objects shared a very high density of low Blue values in both training and test data.

Finally, let's create some scatterplots with the test data like we did with the training data to see how separable the classes are. We'll again randomly sample 200000 observations for these.

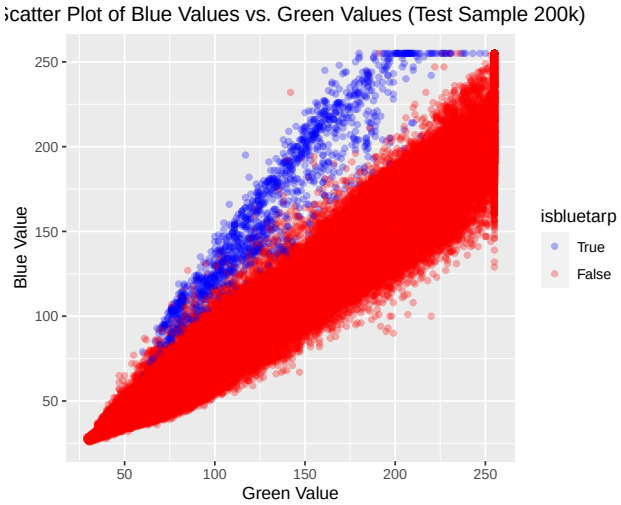
```
set.seed(710)

haiti.test[sample(nrow(haiti.test), 200000), ] %>%
ggplot(aes(x = Red, y = Green)) +
  geom_point(aes(color = isbluetarp),
             alpha = 0.3) +
  scale_color_manual(values=c('Blue', 'Red')) +
  labs(x = 'Red Value',
       y = 'Green Value') +
  ggtitle('Scatter Plot of Green Values vs. Red Values (Test Sample 200k)') +
  theme(plot.title = element_text(hjust = 0.5))
```


```
set.seed(710)

haiti.test[sample(nrow(haiti.test), 200000), ] %>%
ggplot(aes(x = Red, y = Blue)) +
  geom_point(aes(color = isbluetarp),
```

```
            alpha = 0.3) +
  scale_color_manual(values=c('Blue', 'Red')) +
  labs(x = 'Red Value',
       y = 'Blue Value') +
  ggtitle('Scatter Plot of Blue Values vs. Red Values (Test Sample 200k)') +
  theme(plot.title = element_text(hjust = 0.5))
```




```
set.seed(710)

haiti.test[sample(nrow(haiti.test), 200000), ] %>%
ggplot(aes(x = Green, y = Blue)) +
  geom_point(aes(color = isbluetarp),
             alpha = 0.3) +
  scale_color_manual(values=c('Blue', 'Red')) +
  labs(x = 'Green Value',
       y = 'Blue Value') +
  ggtitle('Scatter Plot of Blue Values vs. Green Values (Test Sample 200k)') +
  theme(plot.title = element_text(hjust = 0.5))
```



These test data scatterplots reasonably resemble the training data scatterplots, but they're somewhat less separable (but still pretty separable overall).

Now that we have a solid grasp on the data, let's move on to model building.

Model Building

For each of the models, I'll be focusing on two thresholds against predicted probabilities: the threshold that results in the maximum true positive rate (sensitivity) cross validated with training data; and the one that results in maximum accuracy cross validated with training data. I'll be evaluating a range of thresholds from 0.05 to 0.95 when fitting the model. The lower the threshold, the higher the sensitivity, so I suspect 0.05 to produce the highest sensitivity in our models. I could continuously lower the threshold to 0.01 and 0.001 and so forth to further maximize sensitivity. But I'm more interested in generally comparing performance between each model than pursuing an absolute ideal model, so a 0.05 threshold should do fine.

I'm interested in maximum sensitivity since it directly relates to the minimum false negative rate. If our ultimate goal is to potentially identify human survivors by identifying blue tarps, this means a false negative could lead to a stranded survivor — a devastating outcome. Thus, the threshold with the highest sensitivity would have the lowest false negative rate, minimizing these undesirable outcomes.

Conversely, I'm not so concerned about false positives, as the consequence of looking for survivors where we erroneously identified blue tarps isn't as grave as leaving potential survivors behind. This isn't free, though. Wasting time and resources on falsely identified blue tarps could lead to serious issues as well. But since I don't have context for who in particular would be using these models, I'll assume time and resources are less of a concern than maximizing life saving efforts. Maximum accuracy is less important to me in this context, but it's still an interesting indicator of general model performance, so I'll include it as well.

Regarding the data itself, I also don't see any compelling reason to normalize the values of "Red", "Green" and "Blue" (the values are all on the same scale) or to omit any these predictors/features since they're each essential in determining overall pixel color.

I'll be fitting each model with the same training controls in Caret, including cross-validation with 10 folds.

```
set.seed(710)

trControl <- caret::trainControl(method = 'cv',
                                 number = 10,
                                 savePredictions = TRUE,
                                 classProbs = TRUE,
                                 allowParallel = TRUE)
```

All training ROC curves and training AUC values will be plotted and calculated using out-of-fold predicted probabilities. I'll be creating the training ROC curves with this custom function. I'll be plotting these with about 2000 plot points to save resources.

```
draw_train_ROC <- function(model, title){
train.predob_cv <- ROCR::prediction(model$pred$False,
                            model$pred$obs,
                            label.ordering = c('True', 'False'))
train.roc_cv <- ROCR::performance(train.predob_cv,
                                  measure = 'tpr',
                                  x.measure = 'fpr')
plot(train.roc_cv,
     colorize = T,
     print.cutoffs.at = c(0, 0.1, 0.9, 1.0),
     downsampling = 0.035)
title(main = title)
lines(x = c(0,1),
      y = c(0,1),
      col = 'grey')}
```

I'll be creating the test ROC curves with this custom function. I'll also be plotting test ROC curves with about 2000 points to save resources.

```
draw_test_ROC <- function(preds, title){
test.predob <- ROCR::prediction(preds$False,
                                haiti.test$isbluetarp,
                                label.ordering = c('True', 'False'))
test.roc <- ROCR::performance(test.predob,
                              measure='tpr',
                              x.measure='fpr')
plot(test.roc,
     colorize = T,
     print.cutoffs.at = c(0, 0.1, 0.9, 1.0),
     downsampling = 0.001)
title(main = title)
lines(x = c(0,1),
      y = c(0,1),
      col = 'grey')}
```

Training AUC values will be calculated using this custom function.

```
get_train_AUC <- function(model){
predob_cv <- ROCR::prediction(model$pred$False,
                           model$pred$obs,
                           label.ordering = c('True', 'False'))

train.auc <- performance(predob_cv, measure = "auc")
train.auc <- train.auc@y.values[[1]]
return(train.auc)
}
```

Test AUC values will be calculated using this custom function.

```
get_test_AUC <- function(preds){
predob <- ROCR::prediction(preds$False,
                           haiti.test$isbluetarp,
                           label.ordering = c('True', 'False'))

test.auc <- performance(predob, measure = "auc")
test.auc <- test.auc@y.values[[1]]
return(test.auc)
}
```

I'll also be creating confusion matrices with our test data using this custom function.

```
create_test_conf_matrix <- function(preds, thresh){
  threshold.preds <- factor(
  ifelse(preds[ , 'True'] > thresh, 'True', 'False'))

threshold.preds <- relevel(threshold.preds,
                                  'True')

conf_matrix <- confusionMatrix(threshold.preds,
                haiti.test$isbluetarp)

return(conf_matrix)
}
```

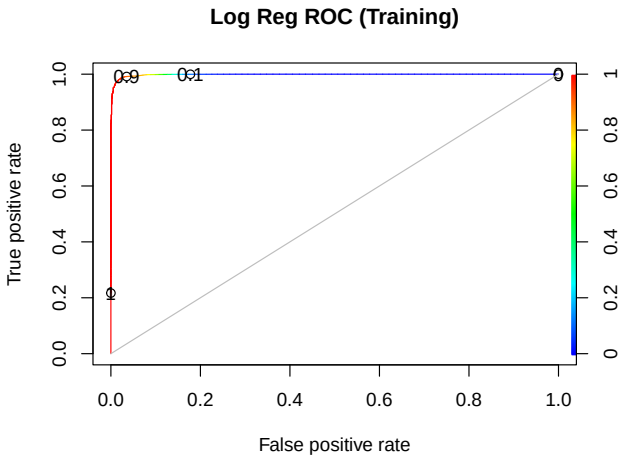
Logistic Regression

```
set.seed(710)

logreg.fit <- caret::train(isbluetarp ~ Red + Green + Blue,
                           data = haiti.train,
                           method = 'glm',
                           trControl = trControl)
```

```
results1.train.model <- c('Log Reg')
results2.train.model <- c('Log Reg')
```

```
draw_train_ROC(logreg.fit, 'Log Reg ROC (Training)')
```

Training




```
logreg.train.auc <- get_train_AUC(logreg.fit)
logreg.train.auc
```

```
#> [1] 0.9984656
```

Our ROC curve tightly hugs the upper left corner and our AUC value is very close to 1. This model seems like a very good fit.

```
results1.train.auc <- c(logreg.train.auc)
results2.train.auc <- c(logreg.train.auc)
```

```
logreg.train.threshold.stats <- caret::thresholder(logreg.fit,
                         threshold = seq(0.05, 0.95, by = 0.05),
                         statistics = "all")

logreg.train.threshold.stats$FNR <- (1 -
                                     logreg.train.threshold.stats$Sensitivity)
logreg.train.threshold.stats$FPR <- (1 -
                                     logreg.train.threshold.stats$Specificity)
```

```
logreg.train.threshold.stats %>%
  select("prob_threshold", "Accuracy", "Sensitivity", "Precision",
         "FNR", "FPR") %>%
  knitr::kable(digits=3) %>%
```

```
  kableExtra::kable_styling(full_width = FALSE,
                            latex_options = "HOLD_position")
```

prob_threshold	Accuracy	Sensitivity	Precision	FNR	FPR
0.05	0.987	0.976	0.717	0.024	0.013
0.10	0.991	0.965	0.792	0.035	0.008
0.15	0.992	0.949	0.820	0.051	0.007
0.20	0.994	0.930	0.881	0.070	0.004
0.25	0.996	0.919	0.947	0.081	0.002
0.30	0.996	0.911	0.953	0.089	0.002
0.35	0.996	0.902	0.955	0.098	0.001
0.40	0.995	0.896	0.958	0.104	0.001
0.45	0.995	0.890	0.963	0.110	0.001
0.50	0.995	0.886	0.964	0.114	0.001
0.55	0.995	0.882	0.966	0.118	0.001
0.60	0.995	0.874	0.970	0.126	0.001
0.65	0.995	0.869	0.972	0.131	0.001
0.70	0.995	0.864	0.975	0.136	0.001
0.75	0.995	0.856	0.976	0.144	0.001
0.80	0.995	0.850	0.979	0.150	0.001
0.85	0.994	0.839	0.982	0.161	0.001
0.90	0.994	0.824	0.985	0.176	0.000
0.95	0.993	0.800	0.989	0.200	0.000

For the highest accuracy, I'll go with a threshold of 0.35 since it's the closest to 0.5. This accuracy of about 99.6% is quite good. As expected, 0.05 resulted in the highest sensitivity of about 97.6%. I would say this is pretty good, but we'll see how it compares as we build more models.

```
logreg.thresh1 <- 0.05
logreg.thresh2 <- 0.35
```

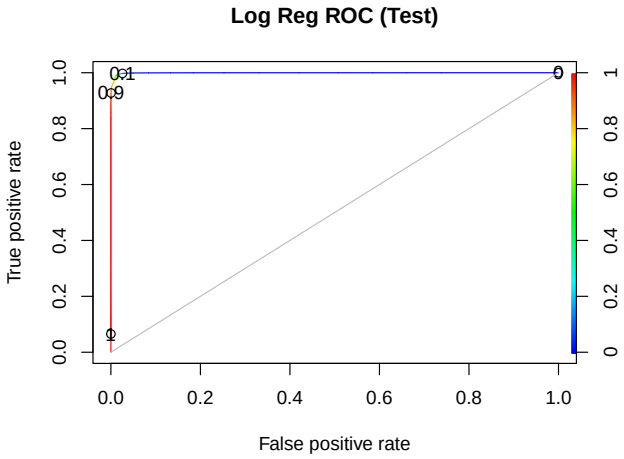
```
results1.train.thresh <- c(logreg.thresh1)
results2.train.thresh <- c(logreg.thresh2)

results1.train.accuracy <- c(0.987)
results1.train.sensitivity <- c(0.976)
results1.train.fpr <- c(0.013)
results1.train.precision <- c(0.717)

results2.train.accuracy <- c(0.996)
results2.train.sensitivity <- c(0.902)
results2.train.fpr <- c(0.001)
results2.train.precision <- c(0.955)
```

```
results1.test.model <- c('Log Reg')
results2.test.model <- c('Log Reg')
```

```
logreg.test.pred <- predict(logreg.fit, haiti.test, type = 'prob')
```

```
draw_test_ROC(logreg.test.pred, 'Log Reg ROC (Test)')
```

Testing




```
logreg.test.auc <- get_test_AUC(logreg.test.pred)
logreg.test.auc
```

```
#> [1] 0.9994131
```

Our ROC curve tightly hugs the upper left corner and our AUC value is very close to 1. This test AUC is greater than the training AUC, suggesting this model performs better on test data. This model seems to be performing very well on the holdout set.

```
results1.test.auc <- c(logreg.test.auc)
results2.test.auc <- c(logreg.test.auc)
```

```
results1.test.thresh <- c(logreg.thresh1)
results2.test.thresh <- c(logreg.thresh2)
```

When fitting this logistic regression model, I selected 0.05 as our threshold for maximum sensitivity and 0.35 as our threshold for maximum accuracy. Let's see how our model does with the test data at a 0.05 threshold and compare sensitivity.

```
logreg.test.conf_matrix1 <- create_test_conf_matrix(logreg.test.pred,
                                                    logreg.thresh1)
logreg.test.conf_matrix1
```

```
#> Confusion Matrix and Statistics
#>
#>           Reference
#> Prediction    True   False
#>      True    14479  193475
#>      False       1 1796222
#>
#>                Accuracy : 0.9035
#>                  95% CI : (0.9031, 0.9039)
#>     No Information Rate : 0.9928
#>     P-Value [Acc > NIR] : 1
#>
#>                   Kappa : 0.1183
#>
#>  Mcnemar's Test P-Value : <2e-16
#>
#>             Sensitivity : 0.999931
```

```
#>             Specificity : 0.902762
#>          Pos Pred Value : 0.069626
#>          Neg Pred Value : 0.999999
#>              Prevalence : 0.007225
#>          Detection Rate : 0.007224
#>    Detection Prevalence : 0.103760
#>       Balanced Accuracy : 0.951346
#>
#>        'Positive' Class : True
#>
```

Our sensitivity is very high, almost 100%. Looking at the confusion matrix, there was only one false negative. This model had a higher sensitivity at the 0.05 threshold with the test data than the training data.

```
# FPR
193475 / (193475 + 1796222)
```

```
#> [1] 0.09723842
```

```
# Precision
14479 / (14479 + 193475)
```

```
#> [1] 0.06962597
```

```
results1.test.accuracy <- c(0.904)
results1.test.sensitivity <- c(0.999)
results1.test.fpr <- c(0.097)
results1.test.precision <- c(0.069)
```

Now let's see how our model does with a threshold of 0.35 and compare accuracy.

```
logreg.test.conf_matrix2 <- create_test_conf_matrix(logreg.test.pred,
                                                    logreg.thresh2)
logreg.test.conf_matrix2
```

```
#> Confusion Matrix and Statistics
#>
#>           Reference
#> Prediction    True   False
#>      True    14356   38636
#>      False     124 1951061
#>
#>                Accuracy : 0.9807
#>                  95% CI : (0.9805, 0.9809)
#>     No Information Rate : 0.9928
#>     P-Value [Acc > NIR] : 1
#>
#>                   Kappa : 0.4189
#>
#>  Mcnemar's Test P-Value : <2e-16
#>
#>             Sensitivity : 0.991436
#>             Specificity : 0.980582
#>          Pos Pred Value : 0.270909
#>          Neg Pred Value : 0.999936
#>              Prevalence : 0.007225
#>          Detection Rate : 0.007163
#>    Detection Prevalence : 0.026441
```

```
#>        Balanced Accuracy : 0.986009
#>
#>         'Positive' Class : True
#>
```

Our accuracy is about 98.1%, which isn't really great in the context of our data (blue tarps consist of less than 1% of our test data). This test accuracy is also worse than training accuracy, but that's to be expected.

```
# FPR
38636 / (38636 + 1951061)
```

```
#> [1] 0.01941803
```

```
# Precision
14356 / (14356 + 38636)
```

```
#> [1] 0.2709088
```

```
results2.test.accuracy <- c(0.981)
results2.test.sensitivity <- c(0.991)
results2.test.fpr <- c(0.019)
results2.test.precision <- c(0.07)
```

Overall, I would say the logistic regression model did quite well with sensitivity but its accuracy was underwhelming.

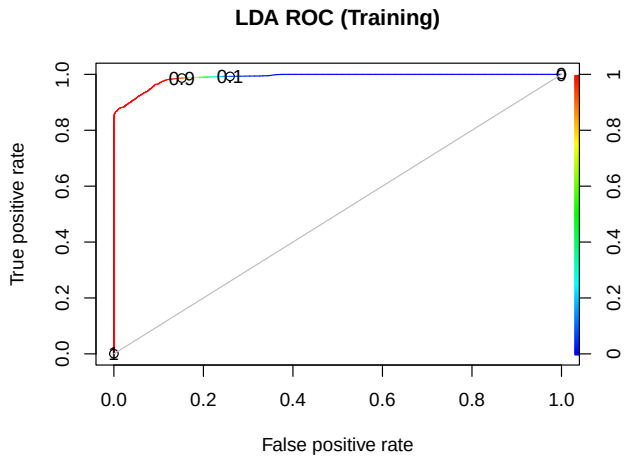
LDA

```
set.seed(710)

lda.fit <- caret::train(isbluetarp ~ Red + Green + Blue,
                        data = haiti.train,
                        method = 'lda',
                        trControl = trControl)
```

```
results1.train.model <- c(results1.train.model, 'LDA')
results2.train.model <- c(results2.train.model, 'LDA')
```

```
draw_train_ROC(lda.fit, 'LDA ROC (Training)')
```

Training



```
lda.train.auc <- get_train_AUC(lda.fit)
lda.train.auc
```

```
#> [1] 0.9888436
```

Our ROC curve does not tightly hug the upper left corner and our AUC value is relatively low. This model does not seem to be a great fit.

```
results1.train.auc <- c(results1.train.auc, lda.train.auc)
results2.train.auc <- c(results2.train.auc, lda.train.auc)
```

```
lda.train.threshold.stats <- caret::thresholder(lda.fit,
                          threshold = seq(0.05, 0.95, by = 0.05),
                          statistics = "all")

lda.train.threshold.stats$FNR <- (1 -
                                    lda.train.threshold.stats$Sensitivity)
lda.train.threshold.stats$FPR <- (1 -
                                    lda.train.threshold.stats$Specificity)
```

```
lda.train.threshold.stats %>%
  select("prob_threshold", "Accuracy", "Sensitivity", "Precision",
         "FNR", "FPR") %>%
  knitr::kable(digits = 3) %>%
  kableExtra::kable_styling(full_width = FALSE,
                            latex_options = "HOLD_position")
```

prob_threshold	Accuracy	Sensitivity	Precision	FNR	FPR
0.05	0.981	0.860	0.656	0.140	0.015
0.10	0.982	0.847	0.682	0.153	0.013
0.15	0.983	0.841	0.686	0.159	0.013
0.20	0.983	0.835	0.691	0.165	0.012
0.25	0.983	0.825	0.697	0.175	0.012
0.30	0.983	0.820	0.705	0.180	0.011
0.35	0.984	0.814	0.715	0.186	0.011
0.40	0.983	0.810	0.714	0.190	0.011
0.45	0.984	0.806	0.720	0.194	0.010
0.50	0.984	0.802	0.727	0.198	0.010
0.55	0.984	0.796	0.727	0.204	0.010
0.60	0.984	0.791	0.732	0.209	0.010
0.65	0.984	0.786	0.734	0.214	0.009
0.70	0.984	0.777	0.740	0.223	0.009
0.75	0.985	0.773	0.751	0.227	0.009
0.80	0.985	0.764	0.757	0.236	0.008
0.85	0.985	0.758	0.766	0.242	0.008
0.90	0.985	0.741	0.770	0.259	0.007
0.95	0.985	0.711	0.786	0.289	0.006

Our highest sensitivity of 86% was at the 0.05 threshold. Interestingly, our highest accuracy was actually at a threshold above 0.5, with a 98.5% accuracy at the 0.75 threshold (along with others, but we'll go with 0.75 since it's the closest to 0.5). Our accuracy at the 0.5 threshold is about 98.4%. This model does not seem like a good fit for the data.

```
lda.thresh1 <- 0.05
lda.thresh2 <- 0.75
```

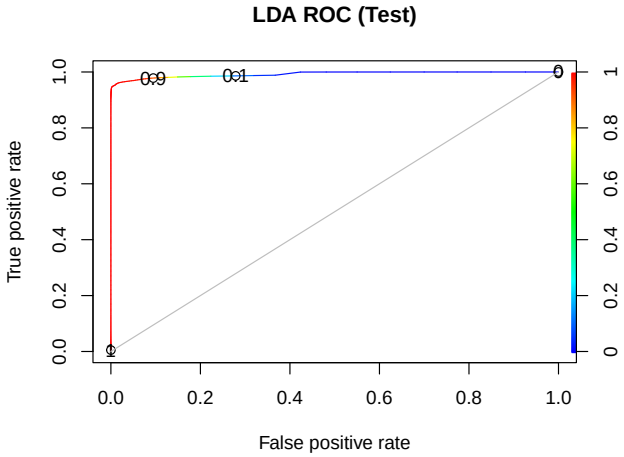
```
results1.train.thresh <- c(results1.train.thresh, lda.thresh1)
results2.train.thresh <- c(results2.train.thresh, lda.thresh2)

results1.train.accuracy <- c(results1.train.accuracy, 0.981)
results1.train.sensitivity <- c(results1.train.sensitivity, 0.860)
results1.train.fpr <- c(results1.train.fpr, 0.015)
results1.train.precision <- c(results1.train.precision, 0.656)

results2.train.accuracy <- c(results2.train.accuracy, 0.985)
results2.train.sensitivity <- c(results2.train.sensitivity, 0.773)
results2.train.fpr <- c(results2.train.fpr, 0.009)
results2.train.precision <- c(results2.train.precision, 0.751)
```

```
results1.test.model <- c(results1.test.model, 'LDA')
results2.test.model <- c(results2.test.model, 'LDA')
```

```
lda.test.pred <- predict(lda.fit, haiti.test, type = 'prob')
```

```
draw_test_ROC(lda.test.pred, 'LDA ROC (Test)')
```

Testing




```
lda.test.auc <- get_test_AUC(lda.test.pred)
lda.test.auc
```

```
#> [1] 0.9921155
```

Our test ROC curve looks a lot better than our training ROC curve, and our test AUC value is much improved. But the test ROC curve and test AUC values from our LDA model are worse than the ones from our logistic regression model.

```
results1.test.auc <- c(results1.test.auc, lda.test.auc)
results2.test.auc <- c(results2.test.auc, lda.test.auc)
```

```
results1.test.thresh <- c(results1.test.thresh, lda.thresh1)
results2.test.thresh <- c(results2.test.thresh, lda.thresh2)
```

When fitting our LDA model, I selected 0.05 as our threshold for maximum sensitivity and 0.75 for maximum accuracy. Let's see how our model does with the test data at a 0.05 threshold.

```
lda.test.conf_matrix1 <- create_test_conf_matrix(lda.test.pred,
                                                 lda.thresh1)
lda.test.conf_matrix1
```

```
#> Confusion Matrix and Statistics
#>
#>           Reference
#> Prediction    True   False
#>      True    13342   47249
#>      False    1138 1942448
#>
#>                Accuracy : 0.9759
#>                  95% CI : (0.9756, 0.9761)
#>     No Information Rate : 0.9928
#>     P-Value [Acc > NIR] : 1
#>
#>                   Kappa : 0.3478
#>
#>  Mcnemar's Test P-Value : <2e-16
#>
#>             Sensitivity : 0.921409
#>             Specificity : 0.976253
#>          Pos Pred Value : 0.220198
#>          Neg Pred Value : 0.999414
#>              Prevalence : 0.007225
#>          Detection Rate : 0.006657
#>    Detection Prevalence : 0.030232
#>       Balanced Accuracy : 0.948831
#>
#>        'Positive' Class : True
#>
```

Our sensitivity at the 0.05 threshold is only about 92.1%. This is pretty low, especially compared to our logistic regression model, but it's an improvement over the training set performance.

```
# FPR
47249 / (47249 + 1942448)
```

```
#> [1] 0.02374683
```

```
# Precision
13342 / (13342 + 47249)
```

```
#> [1] 0.2201977
```

```
results1.test.accuracy <- c(results1.test.accuracy, 0.976)
results1.test.sensitivity <- c(results1.test.sensitivity, 0.921)
results1.test.fpr <- c(results1.test.fpr, 0.024)
results1.test.precision <- c(results1.test.precision, 0.22)
```

Now let's check out the model at a 0.75 threshold for maximum accuracy.

```
lda.test.conf_matrix2 <- create_test_conf_matrix(lda.test.pred,
                                                 lda.thresh2)
lda.test.conf_matrix2
```

```
#> Confusion Matrix and Statistics
#>
#>           Reference
#> Prediction    True   False
#>      True    11449   31016
#>      False    3031 1958681
#>
#>                Accuracy : 0.983
#>                  95% CI : (0.9828, 0.9832)
#>     No Information Rate : 0.9928
#>     P-Value [Acc > NIR] : 1
#>
#>                   Kappa : 0.3956
#>
#>  Mcnemar's Test P-Value : <2e-16
#>
#>             Sensitivity : 0.790677
#>             Specificity : 0.984412
#>          Pos Pred Value : 0.269610
#>          Neg Pred Value : 0.998455
#>              Prevalence : 0.007225
#>          Detection Rate : 0.005713
#>    Detection Prevalence : 0.021188
#>       Balanced Accuracy : 0.887544
#>
#>        'Positive' Class : True
#>
```

Our model's accuracy at the 0.75 threshold is about 98.3%, slightly worse than our accuracy at this threshold with the training data.

```
# FPR
31016 / (31016 + 1958681)
```

```
#> [1] 0.0155883
```

```
# Precision
11449 / (11449 + 31016)
```

```
#> [1] 0.2696103
```

```
results2.test.accuracy <- c(results2.test.accuracy, 0.983)
results2.test.sensitivity <- c(results2.test.sensitivity, 0.791)
results2.test.fpr <- c(results2.test.fpr, 0.016)
results2.test.precision <- c(results2.test.precision, 0.27)
```

I don't think the LDA model performed well in either training or test data. Its sensitivity and accuracy performance with test data were lacking. This isn't unusual since our data wasn't quite linearly separable, but it's interesting that the logistic regression solidly outperformed it (at least in my opinion).

QDA

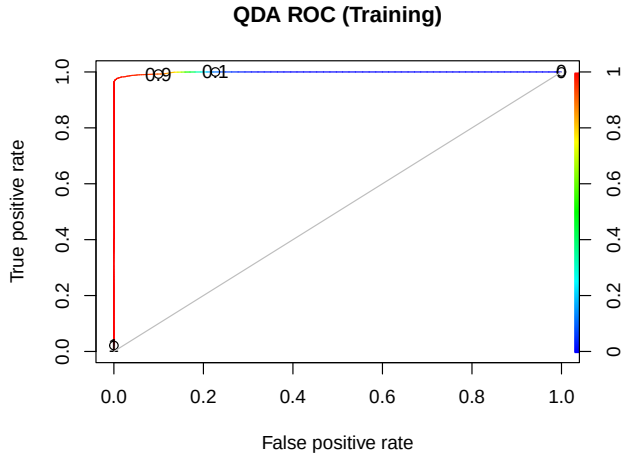
```
set.seed(710)

qda.fit <- caret::train(isbluetarp ~ Red + Green + Blue,
                        data = haiti.train,
```

```
                          method = 'qda',
                          trControl = trControl)
```

```
results1.train.model <- c(results1.train.model, 'QDA')
results2.train.model <- c(results2.train.model, 'QDA')
```

```
draw_train_ROC(qda.fit, 'QDA ROC (Training)')
```

Training




```
qda.train.auc <- get_train_AUC(qda.fit)
qda.train.auc
```

```
#> [1] 0.9981962
```

Our training ROC curve is somewhat strangely shaped, not unlike the LDA training ROC curve, but it hugs the upper left corner more closely than the LDA curve. Our AUC is very close to 1. This model seems like a better fit than the LDA (which was expected) and a good fit overall.

```
results1.train.auc <- c(results1.train.auc, qda.train.auc)
results2.train.auc <- c(results2.train.auc, qda.train.auc)
```

```
qda.train.threshold.stats <- caret::thresholder(qda.fit,
                          threshold = seq(0.05, 0.95, by = 0.05),
                          statistics = "all")

qda.train.threshold.stats$FNR <- (1 -
                                    qda.train.threshold.stats$Sensitivity)
qda.train.threshold.stats$FPR <- (1 -
                                    qda.train.threshold.stats$Specificity)
```

```
qda.train.threshold.stats %>%
  select("prob_threshold", "Accuracy", "Sensitivity", "Precision",
         "FNR", "FPR") %>%
  knitr::kable(digits = 3) %>%
  kableExtra::kable_styling(full_width = FALSE,
                            latex_options = "HOLD_position")
```

prob_threshold	Accuracy	Sensitivity	Precision	FNR	FPR
0.05	0.988	0.942	0.755	0.058	0.010
0.10	0.990	0.900	0.802	0.100	0.007
0.15	0.991	0.884	0.839	0.116	0.006
0.20	0.994	0.869	0.954	0.131	0.001
0.25	0.995	0.862	0.972	0.138	0.001
0.30	0.995	0.856	0.977	0.144	0.001
0.35	0.995	0.852	0.980	0.148	0.001
0.40	0.995	0.848	0.985	0.152	0.000
0.45	0.995	0.845	0.988	0.155	0.000
0.50	0.995	0.841	0.989	0.159	0.000
0.55	0.994	0.833	0.991	0.167	0.000
0.60	0.994	0.828	0.993	0.172	0.000
0.65	0.994	0.820	0.994	0.180	0.000
0.70	0.994	0.815	0.995	0.185	0.000
0.75	0.994	0.808	0.996	0.192	0.000
0.80	0.994	0.799	0.998	0.201	0.000
0.85	0.993	0.790	0.998	0.210	0.000
0.90	0.993	0.773	0.999	0.227	0.000
0.95	0.992	0.745	1.000	0.255	0.000

Our highest sensitivity was about 94.2% at the 0.05 threshold. Our highest accuracy was about 99.5% at the 0.5 threshold (I'll go with this over the other eligible thresholds since it's at the 0.5 threshold). Both of these metrics are improved from the LDA model but worse than the Logistic Regression model.

```
qda.thresh1 <- 0.05
qda.thresh2 <- 0.5
```

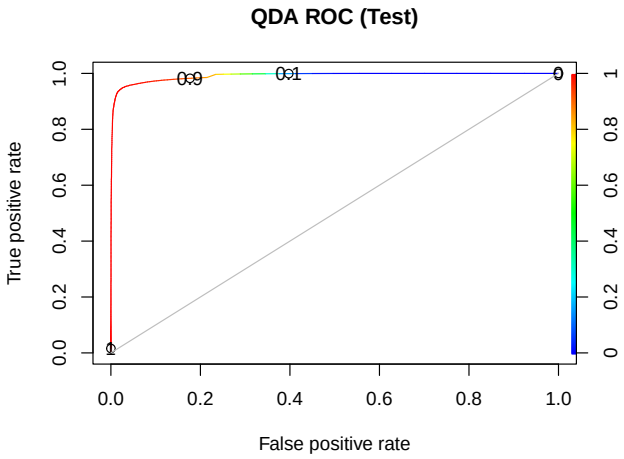
```
results1.train.thresh <- c(results1.train.thresh, qda.thresh1)
results2.train.thresh <- c(results2.train.thresh, qda.thresh2)

results1.train.accuracy <- c(results1.train.accuracy, 0.988)
results1.train.sensitivity <- c(results1.train.sensitivity, 0.942)
results1.train.fpr <- c(results1.train.fpr, 0.010)
results1.train.precision <- c(results1.train.precision, 0.755)

results2.train.accuracy <- c(results2.train.accuracy, 0.995)
results2.train.sensitivity <- c(results2.train.sensitivity, 0.841)
results2.train.fpr <- c(results2.train.fpr, 0.000)
results2.train.precision <- c(results2.train.precision, 0.989)
```

```
results1.test.model <- c(results1.test.model, 'QDA')
results2.test.model <- c(results2.test.model, 'QDA')
```

```
qda.test.pred <- predict(qda.fit, haiti.test, type = 'prob')
```

```
draw_test_ROC(qda.test.pred, 'QDA ROC (Test)')
```

Testing




```
qda.test.auc <- get_test_AUC(qda.test.pred)
qda.test.auc
```

```
#> [1] 0.9915001
```

The test ROC curve doesn't look bad but the test AUC value is actually less than ones for our previous models. Although this QDA model fit the training data pretty well, it seems like it's performing less well on the test data.

```
results1.test.auc <- c(results1.test.auc, qda.test.auc)
results2.test.auc <- c(results2.test.auc, qda.test.auc)
```

```
results1.test.thresh <- c(results1.test.thresh, qda.thresh1)
results2.test.thresh <- c(results2.test.thresh, qda.thresh2)
```

When fitting our QDA model, I selected 0.05 as the threshold for maximum sensitivity and 0.5 as the threshold for maximum accuracy. Let's see how our model performs at the 0.05 threshold.

```
qda.test.conf_matrix1 <- create_test_conf_matrix(qda.test.pred,
                                                 qda.thresh1)
qda.test.conf_matrix1
```

```
#> Confusion Matrix and Statistics
#>
#>           Reference
#> Prediction    True   False
#>      True    12598   44390
#>      False    1882 1945307
#>
#>                Accuracy : 0.9769
#>                  95% CI : (0.9767, 0.9771)
#>     No Information Rate : 0.9928
#>     P-Value [Acc > NIR] : 1
#>
#>                   Kappa : 0.345
#>
#>  Mcnemar's Test P-Value : <2e-16
#>
#>             Sensitivity : 0.870028
#>             Specificity : 0.977690
#>          Pos Pred Value : 0.221064
#>          Neg Pred Value : 0.999033
```

```
#>              Prevalence : 0.007225
#>          Detection Rate : 0.006286
#>    Detection Prevalence : 0.028435
#>       Balanced Accuracy : 0.923859
#>
#>        'Positive' Class : True
#>
```

Our model's sensitivity at the 0.05 threshold is about 87%. This is quite poor, especially compared to our previous models.

```
# FPR
44390 / (44390 + 1945307)
```

```
#> [1] 0.02230993
```

```
# Precision
12598 / (12598 + 44390)
```

```
#> [1] 0.2210641
```

```
results1.test.accuracy <- c(results1.test.accuracy, 0.977)
results1.test.sensitivity <- c(results1.test.sensitivity, 0.870)
results1.test.fpr <- c(results1.test.fpr, 0.022)
results1.test.precision <- c(results1.test.precision, 0.221)
```

Now let's see how our model does at the 0.5 threshold.

```
qda.test.conf_matrix2 <- create_test_conf_matrix(qda.test.pred,
                                                 qda.thresh2)
qda.test.conf_matrix2
```

```
#> Confusion Matrix and Statistics
#>
#>           Reference
#> Prediction    True   False
#>      True    10058    3651
#>      False    4422 1986046
#>
#>                Accuracy : 0.996
#>                  95% CI : (0.9959, 0.9961)
#>     No Information Rate : 0.9928
#>     P-Value [Acc > NIR] : < 2.2e-16
#>
#>                   Kappa : 0.7116
#>
#>  Mcnemar's Test P-Value : < 2.2e-16
#>
#>             Sensitivity : 0.694613
#>             Specificity : 0.998165
#>          Pos Pred Value : 0.733679
#>          Neg Pred Value : 0.997778
#>              Prevalence : 0.007225
#>          Detection Rate : 0.005019
#>    Detection Prevalence : 0.006840
#>       Balanced Accuracy : 0.846389
#>
#>        'Positive' Class : True
```

```
#>
```

Our accuracy is at about 99.6%. This greatly outperforms both our previous models. This isn't surprising, considering the data in the EDA looked somewhat, but not quite, linearly separable and our previous models created linear decision boundaries.

```
# FPR
3651 / (3651 + 1986046)
```

```
#> [1] 0.001834953
```

```
# Precision
10058 / (10058 + 3651)
```

```
#> [1] 0.7336786
```

```
results2.test.accuracy <- c(results2.test.accuracy, 0.996)
results2.test.sensitivity <- c(results2.test.sensitivity, 0.695)
results2.test.fpr <- c(results2.test.fpr, 0.002)
results2.test.precision <- c(results2.test.precision, 0.733)
```

For our primary metric, sensitivity, the QDA model performed (very) poorly. But it had strong accuracy with the test data, especially compared to the other models, so it certainly has some merit.

KNN

Training Let's first evaluate a wide range of potential k neighbors values to help find a potentially optimal one, or at least a good starting point. I'll start by evaluating k values from 1 to 30. Our data is very dense and the classes appeared quite distinct in our earlier visualizations, so I don't expect the optimal k value to be very large.

```
set.seed(710)

tuneGrid = expand.grid(k = 1:30)

knn.fit <- caret::train(isbluetarp ~ Red + Green + Blue,
                        data = haiti.train,
                        method = 'knn',
                        trControl = trControl,
                        tuneGrid = tuneGrid)
```

```
results1.train.model <- c(results1.train.model, 'KNN')
results2.train.model <- c(results2.train.model, 'KNN')
```

```
knn.fit
```

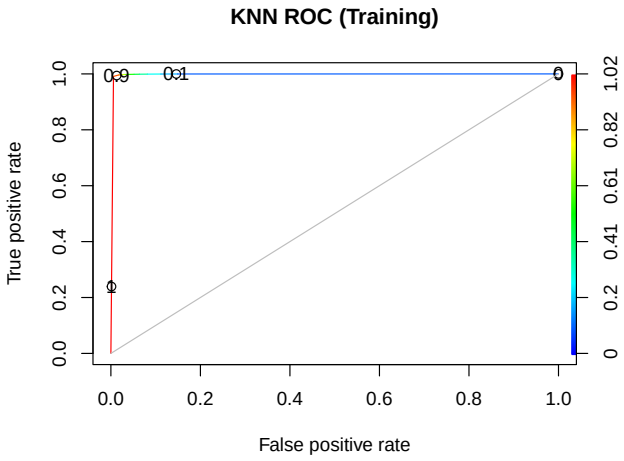
```
#> k-Nearest Neighbors
#>
#> 63241 samples
#>     3 predictor
#>     2 classes: 'True', 'False'
#>
#> No pre-processing
#> Resampling: Cross-Validated (10 fold)
#> Summary of sample sizes: 56916, 56917, 56916, 56918, 56917, 56917, ...
#> Resampling results across tuning parameters:
#>
#>   k   Accuracy   Kappa
#>    1  0.9967268  0.9468299
```

```
#>    2  0.9967900  0.9480746
#>    3  0.9972170  0.9549225
#>    4  0.9972486  0.9556565
#>    5  0.9973435  0.9572203
#>    6  0.9972644  0.9559515
#>    7  0.9971854  0.9547211
#>    8  0.9970905  0.9533705
#>    9  0.9971537  0.9542139
#>   10  0.9971379  0.9539077
#>   11  0.9970430  0.9523874
#>   12  0.9970905  0.9531188
#>   13  0.9971537  0.9540927
#>   14  0.9969798  0.9513641
#>   15  0.9970589  0.9526085
#>   16  0.9970430  0.9522722
#>   17  0.9970905  0.9530598
#>   18  0.9971063  0.9533242
#>   19  0.9970589  0.9525551
#>   20  0.9969956  0.9515716
#>   21  0.9970430  0.9523258
#>   22  0.9970905  0.9531428
#>   23  0.9970114  0.9517957
#>   24  0.9969482  0.9508075
#>   25  0.9969482  0.9507619
#>   26  0.9969166  0.9502383
#>   27  0.9968375  0.9489224
#>   28  0.9968217  0.9486260
#>   29  0.9968375  0.9488341
#>   30  0.9967742  0.9478219
#>
#> Accuracy was used to select the optimal model using the largest value.
#> The final value used for the model was k = 5.
```

It looks like our best k value is 5.

```
results.params.model <- c('KNN')
results.params.param <- c('K')
results.params.value <- c(5)
```

```
draw_train_ROC(knn.fit, 'KNN ROC (Training)')
```



```
knn.train.auc <- get_train_AUC(knn.fit)
knn.train.auc
```

```
#> [1] 0.9978522
```

The ROC curve tightly hugs the upper left corner and the AUC value is very close to 1 (although not quite as close as the Logistic Regression's test AUC). This model seems like a very good fit.

```
results1.train.auc <- c(results1.train.auc, knn.train.auc)
results2.train.auc <- c(results2.train.auc, knn.train.auc)
```

```
knn.train.threshold.stats <- caret::thresholder(knn.fit,
                          threshold = seq(0.05, 0.95, by = 0.05),
                          statistics = "all")

knn.train.threshold.stats$FNR <- (1 -
                                    knn.train.threshold.stats$Sensitivity)
knn.train.threshold.stats$FPR <- (1 -
                                    knn.train.threshold.stats$Specificity)
```

```
knn.train.threshold.stats %>%
  select("prob_threshold", "Accuracy", "Sensitivity", "Precision",
         "FNR", "FPR") %>%
  knitr::kable(digits = 3) %>%
  kableExtra::kable_styling(full_width = FALSE,
                            latex_options = "HOLD_position")
```

prob_threshold	Accuracy	Sensitivity	Precision	FNR	FPR
0.05	0.992	0.996	0.797	0.004	0.008
0.10	0.996	0.983	0.900	0.017	0.004
0.15	0.996	0.983	0.902	0.017	0.004
0.20	0.997	0.973	0.927	0.027	0.003
0.25	0.997	0.973	0.927	0.027	0.003
0.30	0.997	0.973	0.927	0.027	0.003
0.35	0.997	0.972	0.934	0.028	0.002
0.40	0.997	0.963	0.953	0.037	0.002
0.45	0.997	0.963	0.954	0.037	0.002
0.50	0.997	0.958	0.957	0.042	0.001
0.55	0.997	0.958	0.957	0.042	0.001
0.60	0.997	0.935	0.968	0.065	0.001
0.65	0.997	0.935	0.969	0.065	0.001
0.70	0.997	0.928	0.969	0.072	0.001
0.75	0.997	0.926	0.970	0.074	0.001
0.80	0.996	0.886	0.982	0.114	0.001
0.85	0.996	0.874	0.986	0.126	0.000
0.90	0.995	0.870	0.987	0.130	0.000
0.95	0.995	0.869	0.987	0.131	0.000

Our highest sensitivity is at the 0.05 threshold while our best accuracy is at various thresholds, but I'll go with 0.5. This sensitivity and accuracy both seem pretty good overall (or maybe the last couple of models were so poor that this performance looks strong in comparison).

```
knn.thresh1 <- 0.05
knn.thresh2 <- 0.5
```

```
results1.train.thresh <- c(results1.train.thresh, knn.thresh1)
results2.train.thresh <- c(results2.train.thresh, knn.thresh2)

results1.train.accuracy <- c(results1.train.accuracy, 0.992)
results1.train.sensitivity <- c(results1.train.sensitivity, 0.996)
results1.train.fpr <- c(results1.train.fpr, 0.008)
results1.train.precision <- c(results1.train.precision, 0.797)

results2.train.accuracy <- c(results2.train.accuracy, 0.997)
results2.train.sensitivity <- c(results2.train.sensitivity, 0.958)
results2.train.fpr <- c(results2.train.fpr, 0.001)
results2.train.precision <- c(results2.train.precision, 0.957)
```

```
results1.test.model <- c(results1.test.model, 'KNN')
results2.test.model <- c(results2.test.model, 'KNN')
```

```
knn.test.pred <- predict(knn.fit, haiti.test, type = 'prob')
```

```
# this results in an esoteric error that I couldn't debug

# draw_test_ROC(knn.test.pred, 'KNN ROC (Test)')
```

```
knn.test.auc <- get_test_AUC(knn.test.pred)
knn.test.auc
```

Testing

```
#> [1] 0.9494872
```

I unfortunately had technical issues drawing the test ROC curve for the KNN model, but the AUC value is very low. There's a huge discrepancy between the training and test AUC values. Despite not having the ROC curve, I think this AUC value is enough to show that this model is not performing well on the test data.

```
results1.test.auc <- c(results1.test.auc, knn.test.auc)
results2.test.auc <- c(results2.test.auc, knn.test.auc)
```

```
results1.test.thresh <- c(results1.test.thresh, knn.thresh1)
results2.test.thresh <- c(results2.test.thresh, knn.thresh2)
```

When fitting our KNN model, I selected 0.05 as the threshold for maximum sensitivity and 0.5 as the threshold for maximum accuracy. Let's see how our model performs at the 0.05 threshold.

```
knn.test.conf_matrix1 <- create_test_conf_matrix(knn.test.pred,
                                                 knn.thresh1)
knn.test.conf_matrix1
```

```
#> Confusion Matrix and Statistics
#>
#>           Reference
#> Prediction    True   False
#>      True    13115   45850
#>      False    1365 1943847
#>
```

```
#>                Accuracy : 0.9764
#>                  95% CI : (0.9762, 0.9767)
#>     No Information Rate : 0.9928
#>     P-Value [Acc > NIR] : 1
#>
#>                   Kappa : 0.3496
#>
#>  Mcnemar's Test P-Value : <2e-16
#>
#>             Sensitivity : 0.905732
#>             Specificity : 0.976956
#>          Pos Pred Value : 0.222420
#>          Neg Pred Value : 0.999298
#>              Prevalence : 0.007225
#>          Detection Rate : 0.006544
#>    Detection Prevalence : 0.029421
#>       Balanced Accuracy : 0.941344
#>
#>        'Positive' Class : True
#>
```

Our sensitivity at the 0.05 threshold is about 90.6%, which is not good.

```
# FPR
45850 / (45850 + 1943847)
```

```
#> [1] 0.02304371
```

```
# Precision
13115 / (13115 + 45850)
```

```
#> [1] 0.2224201
```

```
results1.test.accuracy <- c(results1.test.accuracy, 0.976)
results1.test.sensitivity <- c(results1.test.sensitivity, 0.906)
results1.test.fpr <- c(results1.test.fpr, 0.023)
results1.test.precision <- c(results1.test.precision, 0.222)
```

Now let's see how our model does at the 0.5 threshold.

```
knn.test.conf_matrix2 <- create_test_conf_matrix(knn.test.pred,
                                                 knn.thresh2)
knn.test.conf_matrix2
```

```
#> Confusion Matrix and Statistics
#>
#>           Reference
#> Prediction    True   False
#>      True    11816   12544
#>      False    2664 1977153
#>
#>                Accuracy : 0.9924
#>                  95% CI : (0.9923, 0.9925)
#>     No Information Rate : 0.9928
#>     P-Value [Acc > NIR] : 1
#>
#>                   Kappa : 0.6049
#>
```

```
#>  Mcnemar's Test P-Value : <2e-16
#>
#>             Sensitivity : 0.816022
#>             Specificity : 0.993696
#>          Pos Pred Value : 0.485057
#>          Neg Pred Value : 0.998654
#>              Prevalence : 0.007225
#>          Detection Rate : 0.005896
#>    Detection Prevalence : 0.012155
#>       Balanced Accuracy : 0.904859
#>
#>        'Positive' Class : True
#>
```

Our accuracy at the 0.5 threshold was about 99.2%, which seems pretty strong overall (at least compared to our previous models).

```
# FPR
12544 / (12544 + 1977153)
```

```
#> [1] 0.006304478
```

```
# Precision
11816 / (11816 + 12544)
```

```
#> [1] 0.4850575
```

```
results2.test.accuracy <- c(results2.test.accuracy, 0.992)
results2.test.sensitivity <- c(results2.test.sensitivity, 0.816)
results2.test.fpr <- c(results2.test.fpr, 0.006)
results2.test.precision <- c(results2.test.precision, 0.485)
```

Overall, this model had underwhelming sensitivity but had pretty impressive accuracy. It's not the best for our primary metric but still seems like a useful model.

Penalized Logistic Regression — Ridge

```
set.seed(710)

lambdas <- 10 ^ seq(-4, 2, 0.2)
tuneGrid <- expand.grid(alpha = 0,
                        lambda = lambdas)

ridge.fit <- caret::train(isbluetarp ~ Red + Green + Blue,
                          data = haiti.train,
                          method = 'glmnet',
                          trControl = trControl,
                          tuneGrid = tuneGrid)
```

```
results1.train.model <- c(results1.train.model, 'Ridge')
results2.train.model <- c(results2.train.model, 'Ridge')
```

```
ridge.fit$bestTune$lambda
```

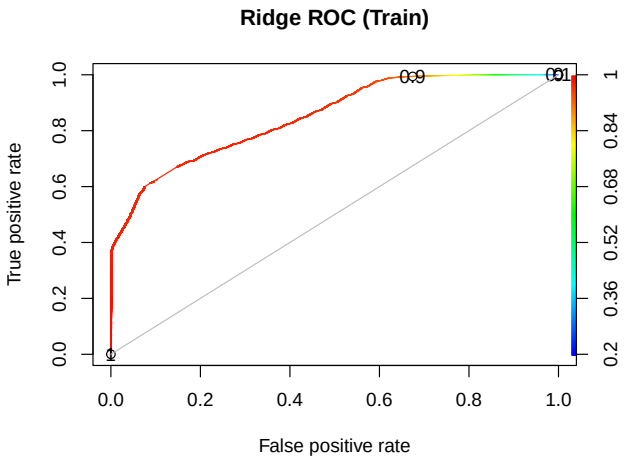
Training

```
#> [1] 0.003981072
```

```
results.params.model <- c(results.params.model, 'Ridge')
results.params.param <- c(results.params.param, 'lambda')
results.params.value <- c(results.params.value, 0.004)
```

```
results.params.model <- c(results.params.model, 'Ridge')
results.params.param <- c(results.params.param, 'alpha')
results.params.value <- c(results.params.value, 0)
```

```
draw_train_ROC(ridge.fit, 'Ridge ROC (Train)')
```




```
ridge.train.auc <- get_train_AUC(ridge.fit)
ridge.train.auc
```

```
#> [1] 0.85112
```

This ROC curve and AUC value are abysmal. Even after tuning, our Ridge Regression seems like a very poor fit.

```
results1.train.auc <- c(results1.train.auc, ridge.train.auc)
results2.train.auc <- c(results2.train.auc, ridge.train.auc)
```

```
ridge.train.threshold.stats <- caret::thresholder(ridge.fit,
                          threshold = seq(0.05, 0.95, by = 0.05),
                          statistics = "all")

ridge.train.threshold.stats$FNR <- (1 -
                                    ridge.train.threshold.stats$Sensitivity)
ridge.train.threshold.stats$FPR <- (1 -
                                    ridge.train.threshold.stats$Specificity)
```

```
ridge.train.threshold.stats %>%
  select("prob_threshold", "Accuracy", "Sensitivity", "Precision",
         "FNR", "FPR") %>%
  knitr::kable(digits = 3) %>%
  kableExtra::kable_styling(full_width = FALSE,
                            latex_options = "HOLD_position")
```

prob_threshold	Accuracy	Sensitivity	Precision	FNR	FPR
0.05	0.917	0.895	0.264	0.105	0.083
0.10	0.980	0.816	0.645	0.184	0.015
0.15	0.982	0.746	0.716	0.254	0.010
0.20	0.983	0.662	0.770	0.338	0.007
0.25	0.987	0.582	0.999	0.418	0.000
0.30	0.985	0.528	1.000	0.472	0.000
0.35	0.983	0.479	1.000	0.521	0.000
0.40	0.982	0.442	1.000	0.558	0.000
0.45	0.980	0.385	1.000	0.615	0.000
0.50	0.978	0.310	1.000	0.690	0.000
0.55	0.975	0.218	1.000	0.782	0.000
0.60	0.973	0.146	1.000	0.854	0.000
0.65	0.971	0.085	1.000	0.915	0.000
0.70	0.969	0.045	1.000	0.955	0.000
0.75	0.969	0.018	1.000	0.982	0.000
0.80	0.968	0.000	NaN	1.000	0.000
0.85	0.968	0.000	NaN	1.000	0.000
0.90	0.968	0.000	NaN	1.000	0.000
0.95	0.968	0.000	NaN	1.000	0.000

Our highest sensitivity is found at the 0.05 threshold while our best accuracy is found at the 0.25 threshold. Both seem not great compared to what we've seen from our models thus far.

```
ridge.thresh1 <- 0.05
ridge.thresh2 <- 0.25
```

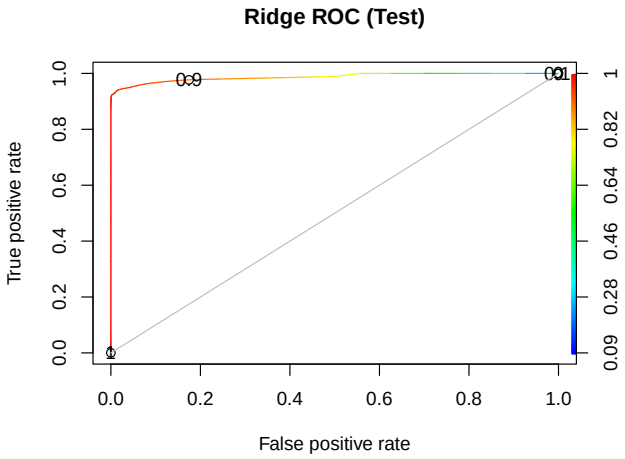
```
results1.train.thresh <- c(results1.train.thresh, ridge.thresh1)
results2.train.thresh <- c(results2.train.thresh, ridge.thresh2)

results1.train.accuracy <- c(results1.train.accuracy, 0.917)
results1.train.sensitivity <- c(results1.train.sensitivity, 0.895)
results1.train.fpr <- c(results1.train.fpr, 0.083)
results1.train.precision <- c(results1.train.precision, 0.264)

results2.train.accuracy <- c(results2.train.accuracy, 0.987)
results2.train.sensitivity <- c(results2.train.sensitivity, 0.582)
results2.train.fpr <- c(results2.train.fpr, 0.000)
results2.train.precision <- c(results2.train.precision, 0.999)
```

```
results1.test.model <- c(results1.test.model, 'Ridge')
results2.test.model <- c(results2.test.model, 'Ridge')
```

```
ridge.test.pred <- predict(ridge.fit, haiti.test, type = 'prob')
```

```
draw_test_ROC(ridge.test.pred, 'Ridge ROC (Test)')
```

Testing




```
ridge.test.auc <- get_test_AUC(ridge.test.pred)
ridge.test.auc
```

```
#> [1] 0.9875184
```

This ROC curve pretty closely hugs the upper left corner and AUC is pretty close to 1. Overall, I wouldn't say this is particularly good compared to some of our other models, but this is a massive improvement over the training set performance (and somewhat of a surprise).

```
results1.test.auc <- c(results1.test.auc, ridge.test.auc)
results2.test.auc <- c(results2.test.auc, ridge.test.auc)
```

```
results1.test.thresh <- c(results1.test.thresh, ridge.thresh1)
results2.test.thresh <- c(results2.test.thresh, ridge.thresh2)
```

I selected 0.05 as the threshold for maximum sensitivity and 0.25 as the threshold for maximum accuracy when fitting our model. Let's see how our it performs at the 0.05 threshold.

```
ridge.test.conf_matrix1 <- create_test_conf_matrix(ridge.test.pred,
                                                   ridge.thresh1)
ridge.test.conf_matrix1
```

```
#> Confusion Matrix and Statistics
#>
#>           Reference
#> Prediction    True   False
#>      True    13860   99224
#>      False     620 1890473
#>
#>                Accuracy : 0.9502
#>                  95% CI : (0.9499, 0.9505)
#>     No Information Rate : 0.9928
#>     P-Value [Acc > NIR] : 1
#>
#>                   Kappa : 0.2071
#>
#>  Mcnemar's Test P-Value : <2e-16
#>
#>             Sensitivity : 0.957182
#>             Specificity : 0.950131
#>          Pos Pred Value : 0.122564
#>          Neg Pred Value : 0.999672
```

```
#>              Prevalence : 0.007225
#>          Detection Rate : 0.006916
#>    Detection Prevalence : 0.056424
#>       Balanced Accuracy : 0.953657
#>
#>        'Positive' Class : True
#>
```

Our sensitivity at the 0.05 threshold is about 95.7%. I wouldn't say this is good overall, but compared to our training results, this is surprisingly high.

```
# FPR
99224 / (99224 + 1890473)
```

```
#> [1] 0.0498689
```

```
# Precision
13860 / (13860 + 99224)
```

```
#> [1] 0.1225638
```

```
results1.test.accuracy <- c(results1.test.accuracy, 0.950)
results1.test.sensitivity <- c(results1.test.sensitivity, 0.957)
results1.test.fpr <- c(results1.test.fpr, 0.050)
results1.test.precision <- c(results1.test.precision, 0.123)
```

Now let's see how the model does at the 0.25 threshold.

```
ridge.test.conf_matrix2 <- create_test_conf_matrix(ridge.test.pred,
                                                   ridge.thresh2)
ridge.test.conf_matrix2
```

```
#> Confusion Matrix and Statistics
#>
#>           Reference
#> Prediction    True   False
#>      True     7036     529
#>      False    7444 1989168
#>
#>                Accuracy : 0.996
#>                  95% CI : (0.9959, 0.9961)
#>     No Information Rate : 0.9928
#>     P-Value [Acc > NIR] : < 2.2e-16
#>
#>                   Kappa : 0.6365
#>
#>  Mcnemar's Test P-Value : < 2.2e-16
#>
#>             Sensitivity : 0.485912
#>             Specificity : 0.999734
#>          Pos Pred Value : 0.930073
#>          Neg Pred Value : 0.996272
#>              Prevalence : 0.007225
#>          Detection Rate : 0.003511
#>    Detection Prevalence : 0.003775
#>       Balanced Accuracy : 0.742823
#>
#>        'Positive' Class : True
```

```
#>
```

Our accuracy of 99.6% is quite good, among the best of our models. This is surprising, considering how the model performed with the training data.

```
# FPR
529 / (529 + 1989168)
```

```
#> [1] 0.0002658696
```

```
# Precision
7036 / (7036 + 529)
```

```
#> [1] 0.9300727
```

```
results2.test.accuracy <- c(results2.test.accuracy, 0.996)
results2.test.sensitivity <- c(results2.test.sensitivity, 0.486)
results2.test.fpr <- c(results2.test.fpr, 0.000)
results2.test.precision <- c(results2.test.precision, 0.930)
```

This model had unusual behavior: it performed very poorly with training data but performed surprisingly well with test data. Our highest sensitivity was still not great, but our best accuracy was unexpectedly high.

Penalized Logistic Regression (Lasso)

```
set.seed(710)

lambdas <- 10 ^ seq(-4, 2, 0.2)
tuneGrid <- expand.grid(alpha = 1,
                        lambda = lambdas)

lasso.fit <- caret::train(isbluetarp ~ Red + Green + Blue,
                          data = haiti.train,
                          method = 'glmnet',
                          trControl = trControl,
                          tuneGrid = tuneGrid)
```

```
results1.train.model <- c(results1.train.model, 'Lasso')
results2.train.model <- c(results2.train.model, 'Lasso')
```

```
lasso.fit$bestTune$lambda
```

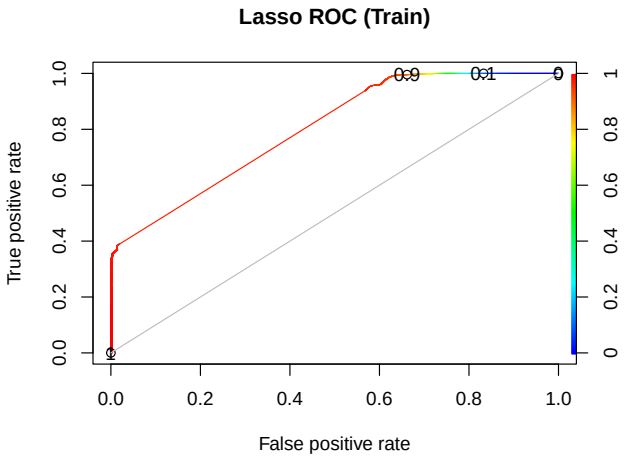
Training

```
#> [1] 1e-04
```

```
results.params.model <- c(results.params.model, 'Lasso')
results.params.param <- c(results.params.param, 'lambda')
results.params.value <- c(results.params.value, 0.0001)
```

```
results.params.model <- c(results.params.model, 'Lasso')
results.params.param <- c(results.params.param, 'alpha')
results.params.value <- c(results.params.value, 1)
```

```
draw_train_ROC(lasso.fit, 'Lasso ROC (Train)')
```




```
lasso.train.auc <- get_train_AUC(lasso.fit)
lasso.train.auc
```

```
#> [1] 0.8005947
```

This ROC curve and AUC value are dismal. Even after tuning, our Lasso Regression doesn't seem like a good fit.

```
results1.train.auc <- c(results1.train.auc, lasso.train.auc)
results2.train.auc <- c(results2.train.auc, lasso.train.auc)
```

```
lasso.train.threshold.stats <- caret::thresholder(lasso.fit,
                          threshold = seq(0.05, 0.95, by = 0.05),
                          statistics = "all")

lasso.train.threshold.stats$FNR <- (1 -
                                    lasso.train.threshold.stats$Sensitivity)
lasso.train.threshold.stats$FPR <- (1 -
                                    lasso.train.threshold.stats$Specificity)
```

```
lasso.train.threshold.stats %>%
  select("prob_threshold", "Accuracy", "Sensitivity", "Precision",
         "FNR", "FPR") %>%
  knitr::kable(digits = 3) %>%
  kableExtra::kable_styling(full_width = FALSE,
                            latex_options = "HOLD_position")
```

prob_threshold	Accuracy	Sensitivity	Precision	FNR	FPR
0.05	0.987	0.979	0.717	0.021	0.013
0.10	0.991	0.961	0.795	0.039	0.008
0.15	0.991	0.940	0.812	0.060	0.007
0.20	0.991	0.924	0.825	0.076	0.007
0.25	0.993	0.906	0.893	0.094	0.004
0.30	0.995	0.895	0.960	0.105	0.001
0.35	0.995	0.886	0.966	0.114	0.001
0.40	0.995	0.877	0.969	0.123	0.001
0.45	0.995	0.873	0.974	0.127	0.001
0.50	0.995	0.867	0.976	0.133	0.001
0.55	0.995	0.862	0.978	0.138	0.001
0.60	0.995	0.856	0.981	0.144	0.001
0.65	0.995	0.848	0.984	0.152	0.000
0.70	0.995	0.840	0.986	0.160	0.000
0.75	0.994	0.829	0.987	0.171	0.000
0.80	0.994	0.817	0.987	0.183	0.000
0.85	0.993	0.798	0.990	0.202	0.000
0.90	0.993	0.777	0.994	0.223	0.000
0.95	0.992	0.737	0.998	0.263	0.000

Our highest sensitivity is at the 0.05 threshold while our highest accuracy is at many thresholds, but I'll pick 0.5. This is actually pretty good and not what I expected, given our ROC curve and AUC value.

```
lasso.thresh1 <- 0.05
lasso.thresh2 <- 0.5
```

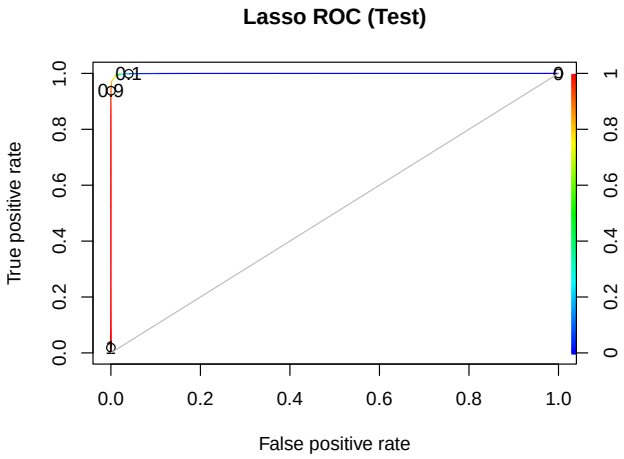
```
results1.train.thresh <- c(results1.train.thresh, lasso.thresh1)
results2.train.thresh <- c(results2.train.thresh, lasso.thresh2)

results1.train.accuracy <- c(results1.train.accuracy, 0.987)
results1.train.sensitivity <- c(results1.train.sensitivity, 0.979)
results1.train.fpr <- c(results1.train.fpr, 0.013)
results1.train.precision <- c(results1.train.precision, 0.717)

results2.train.accuracy <- c(results2.train.accuracy, 0.995)
results2.train.sensitivity <- c(results2.train.sensitivity, 0.867)
results2.train.fpr <- c(results2.train.fpr, 0.001)
results2.train.precision <- c(results2.train.precision, 0.976)
```

```
results1.test.model <- c(results1.test.model, 'Lasso')
results2.test.model <- c(results2.test.model, 'Lasso')
```

```
lasso.test.pred <- predict(lasso.fit, haiti.test, type = 'prob')
```

```
draw_test_ROC(lasso.test.pred, 'Lasso ROC (Test)')
```

Testing




```
lasso.test.auc <- get_test_AUC(lasso.test.pred)
lasso.test.auc
```

```
#> [1] 0.9996042
```

This ROC curve very tightly hugs the upper left corner and our AUC value is extremely close to 1. Our model seems to perform very well on the test data, despite the low training AUC value and poor training ROC curve.

```
results1.test.auc <- c(results1.test.auc, lasso.test.auc)
results2.test.auc <- c(results2.test.auc, lasso.test.auc)
```

```
results1.test.thresh <- c(results1.test.thresh, lasso.thresh1)
results2.test.thresh <- c(results2.test.thresh, lasso.thresh2)
```

Let's see how our model does at our chosen 0.05 threshold for highest sensitivity.

```
lasso.test.conf_matrix1 <- create_test_conf_matrix(lasso.test.pred,
                                                   lasso.thresh1)
lasso.test.conf_matrix1
```

```
#> Confusion Matrix and Statistics
#>
#>           Reference
#> Prediction    True   False
#>      True    14480  189006
#>      False       0 1800691
#>
#>                Accuracy : 0.9057
#>                  95% CI : (0.9053, 0.9061)
#>     No Information Rate : 0.9928
#>     P-Value [Acc > NIR] : 1
#>
#>                   Kappa : 0.121
#>
#>  Mcnemar's Test P-Value : <2e-16
#>
#>             Sensitivity : 1.000000
#>             Specificity : 0.905008
#>          Pos Pred Value : 0.071160
#>          Neg Pred Value : 1.000000
#>              Prevalence : 0.007225
```

```
#>          Detection Rate : 0.007225
#>    Detection Prevalence : 0.101531
#>       Balanced Accuracy : 0.952504
#>
#>        'Positive' Class : True
#>
```

Our model predicted zero false negatives and had a sensitivity of 1! This is our best model thus far regarding sensitivity.

```
# FPR
189006 / (189006  + 1800691)
```

```
#> [1] 0.09499235
```

```
# Precision
14480 / (14480 + 189006)
```

```
#> [1] 0.07115969
```

```
results1.test.accuracy <- c(results1.test.accuracy, 0.906)
results1.test.sensitivity <- c(results1.test.sensitivity, 1.000)
results1.test.fpr <- c(results1.test.fpr, 0.095)
results1.test.precision <- c(results1.test.precision, 0.071)
```

Now let's see how our model does at the 0.5 threshold for maximum accuracy.

```
lasso.test.conf_matrix2 <- create_test_conf_matrix(lasso.test.pred,
                                                   lasso.thresh2)
lasso.test.conf_matrix2
```

```
#> Confusion Matrix and Statistics
#>
#>           Reference
#> Prediction    True   False
#>      True    14249    8988
#>      False     231 1980709
#>
#>                Accuracy : 0.9954
#>                  95% CI : (0.9953, 0.9955)
#>     No Information Rate : 0.9928
#>     P-Value [Acc > NIR] : < 2.2e-16
#>
#>                   Kappa : 0.7534
#>
#>  Mcnemar's Test P-Value : < 2.2e-16
#>
#>             Sensitivity : 0.984047
#>             Specificity : 0.995483
#>          Pos Pred Value : 0.613203
#>          Neg Pred Value : 0.999883
#>              Prevalence : 0.007225
#>          Detection Rate : 0.007110
#>    Detection Prevalence : 0.011594
#>       Balanced Accuracy : 0.989765
#>
#>        'Positive' Class : True
#>
```

Our accuracy of almost 99.6% is very good compared to our other models so far, another surprising result!

```
# FPR
8988 / (8988  + 1980709)
```

```
#> [1] 0.004517271
```

```
# Precision
14249 / (14249 + 8988)
```

```
#> [1] 0.6132031
```

```
results2.test.accuracy <- c(results2.test.accuracy, 0.996)
results2.test.sensitivity <- c(results2.test.sensitivity, 0.984)
results2.test.fpr <- c(results2.test.fpr, 0.005)
results2.test.precision <- c(results2.test.precision, 0.613)
```

This model was our best so far. Like our Ridge Regression, our Lasso Regression performed much better with the test data than with the training data. The test sensitivity and test accuracy at our selected thresholds were both impressive, especially considering how poor of a fit the model seemed with the training data.

Random Forest

Training Let's first tune our random forest for the best value of mtry, which dictates how many features each tree in our random forest will consider, and how many trees our random forest model should have.

```
rf.list <- list()

for (ntree in c(50, 100, 150, 200, 250, 300, 350, 400)){
  set.seed(710)

  rf.fit <- caret::train(isbluetarp ~ Red + Green + Blue,
                         data = haiti.train,
                         method = 'rf',
                         ntree = ntree,
                         trControl = trControl,
                         tuneGrid = data.frame(mtry = c(1 : 3)))

  rf.list.key <- toString(ntree)
  rf.list[[rf.list.key]] <- rf.fit
}
```

```
results1.train.model <- c(results1.train.model, 'RF')
results2.train.model <- c(results2.train.model, 'RF')
```

```
rf.list
```

```
#> $`50`
#> Random Forest
#>
#> 63241 samples
#>     3 predictor
#>     2 classes: 'True', 'False'
#>
#> No pre-processing
#> Resampling: Cross-Validated (10 fold)
#> Summary of sample sizes: 56916, 56917, 56916, 56918, 56917, 56917, ...
#> Resampling results across tuning parameters:
```

```
#>
#>   mtry  Accuracy   Kappa
#>   1     0.9969324  0.9499841
#>   2     0.9969165  0.9498667
#>   3     0.9967110  0.9465112
#>
#> Accuracy was used to select the optimal model using the largest value.
#> The final value used for the model was mtry = 1.
#>
#> $`100`
#> Random Forest
#>
#> 63241 samples
#>     3 predictor
#>     2 classes: 'True', 'False'
#>
#> No pre-processing
#> Resampling: Cross-Validated (10 fold)
#> Summary of sample sizes: 56916, 56917, 56916, 56918, 56917, 56917, ...
#> Resampling results across tuning parameters:
#>
#>   mtry  Accuracy   Kappa
#>   1     0.9970114  0.9512892
#>   2     0.9969482  0.9502973
#>   3     0.9966952  0.9461317
#>
#> Accuracy was used to select the optimal model using the largest value.
#> The final value used for the model was mtry = 1.
#>
#> $`150`
#> Random Forest
#>
#> 63241 samples
#>     3 predictor
#>     2 classes: 'True', 'False'
#>
#> No pre-processing
#> Resampling: Cross-Validated (10 fold)
#> Summary of sample sizes: 56916, 56917, 56916, 56918, 56917, 56917, ...
#> Resampling results across tuning parameters:
#>
#>   mtry  Accuracy   Kappa
#>   1     0.9970430  0.9517872
#>   2     0.9970114  0.9513193
#>   3     0.9966793  0.9459087
#>
#> Accuracy was used to select the optimal model using the largest value.
#> The final value used for the model was mtry = 1.
#>
#> $`200`
#> Random Forest
#>
#> 63241 samples
#>     3 predictor
```

```
#>     2 classes: 'True', 'False'
#>
#> No pre-processing
#> Resampling: Cross-Validated (10 fold)
#> Summary of sample sizes: 56916, 56917, 56916, 56918, 56917, 56917, ...
#> Resampling results across tuning parameters:
#>
#>   mtry  Accuracy   Kappa
#>   1     0.9969798  0.9507798
#>   2     0.9969798  0.9508252
#>   3     0.9967110  0.9464470
#>
#> Accuracy was used to select the optimal model using the largest value.
#> The final value used for the model was mtry = 1.
#>
#> $`250`
#> Random Forest
#>
#> 63241 samples
#>     3 predictor
#>     2 classes: 'True', 'False'
#>
#> No pre-processing
#> Resampling: Cross-Validated (10 fold)
#> Summary of sample sizes: 56916, 56917, 56916, 56918, 56917, 56917, ...
#> Resampling results across tuning parameters:
#>
#>   mtry  Accuracy   Kappa
#>   1     0.9969798  0.9507483
#>   2     0.9969640  0.9505599
#>   3     0.9966477  0.9454189
#>
#> Accuracy was used to select the optimal model using the largest value.
#> The final value used for the model was mtry = 1.
#>
#> $`300`
#> Random Forest
#>
#> 63241 samples
#>     3 predictor
#>     2 classes: 'True', 'False'
#>
#> No pre-processing
#> Resampling: Cross-Validated (10 fold)
#> Summary of sample sizes: 56916, 56917, 56916, 56918, 56917, 56917, ...
#> Resampling results across tuning parameters:
#>
#>   mtry  Accuracy   Kappa
#>   1     0.9969640  0.9504689
#>   2     0.9969165  0.9497742
#>   3     0.9966319  0.9451479
#>
#> Accuracy was used to select the optimal model using the largest value.
#> The final value used for the model was mtry = 1.
```

```
#>
#> $`350`
#> Random Forest
#>
#> 63241 samples
#>     3 predictor
#>     2 classes: 'True', 'False'
#>
#> No pre-processing
#> Resampling: Cross-Validated (10 fold)
#> Summary of sample sizes: 56916, 56917, 56916, 56918, 56917, 56917, ...
#> Resampling results across tuning parameters:
#>
#>   mtry  Accuracy   Kappa
#>   1     0.9969798  0.9506913
#>   2     0.9969482  0.9502955
#>   3     0.9966477  0.9453992
#>
#> Accuracy was used to select the optimal model using the largest value.
#> The final value used for the model was mtry = 1.
#>
#> $`400`
#> Random Forest
#>
#> 63241 samples
#>     3 predictor
#>     2 classes: 'True', 'False'
#>
#> No pre-processing
#> Resampling: Cross-Validated (10 fold)
#> Summary of sample sizes: 56916, 56917, 56916, 56918, 56917, 56917, ...
#> Resampling results across tuning parameters:
#>
#>   mtry  Accuracy   Kappa
#>   1     0.9970272  0.9515138
#>   2     0.9969640  0.9505657
#>   3     0.9966793  0.9459634
#>
#> Accuracy was used to select the optimal model using the largest value.
#> The final value used for the model was mtry = 1.
```

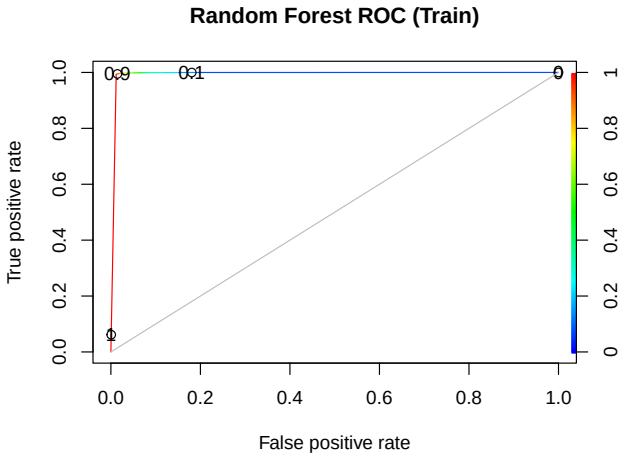
All of our models selected 1 as the best value for mtry. Considering the accuracy of each random forest, I'll go with 300 trees. Although it doesn't have the absolute highest accuracy among our random forests, it's still very close to the highest accuracy, lies in a stable region among accuracy values and is a reasonably large (but not too large) choice for the size of the random forest.

```
set.seed(710)

rf.fit.tuned <- caret::train(isbluetarp ~ Red + Green + Blue,
                             data = haiti.train,
                             method = 'rf',
                             ntree = 300,
                             trControl = trControl,
                             tuneGrid = data.frame(mtry = 1))
```

```
results.params.model <- c(results.params.model, 'RF')
results.params.param <- c(results.params.param, 'mtry')
results.params.value <- c(results.params.value, 1)
```

```
results.params.model <- c(results.params.model, 'RF')
results.params.param <- c(results.params.param, 'ntrees')
results.params.value <- c(results.params.value, 300)
```

```
draw_train_ROC(rf.fit, 'Random Forest ROC (Train)')
```




```
rf.train.auc <- get_train_AUC(rf.fit)
rf.train.auc
```

```
#> [1] 0.9959696
```

This ROC curve very tightly hugs the upper left corner and our AUC value is very close to 1. This model seems like a good fit.

```
results1.train.auc <- c(results1.train.auc, rf.train.auc)
results2.train.auc <- c(results2.train.auc, rf.train.auc)
```

```
rf.train.threshold.stats <- caret::thresholder(rf.fit,
                          threshold = seq(0.05, 0.95, by = 0.05),
                          statistics = "all")

rf.train.threshold.stats$FNR <- (1 -
                                   rf.train.threshold.stats$Sensitivity)
rf.train.threshold.stats$FPR <- (1 -
                                   rf.train.threshold.stats$Specificity)
```

```
rf.train.threshold.stats %>%
  select("prob_threshold", "Accuracy", "Sensitivity", "Precision",
         "FNR", "FPR") %>%
  knitr::kable(digits = 3) %>%
  kableExtra::kable_styling(full_width = FALSE,
                            latex_options = "HOLD_position")
```

prob_threshold	Accuracy	Sensitivity	Precision	FNR	FPR
0.05	0.988	0.998	0.736	0.002	0.012
0.10	0.991	0.995	0.775	0.005	0.010
0.15	0.996	0.983	0.900	0.017	0.004
0.20	0.996	0.978	0.913	0.022	0.003
0.25	0.997	0.974	0.922	0.026	0.003
0.30	0.997	0.971	0.932	0.029	0.002
0.35	0.997	0.965	0.943	0.035	0.002
0.40	0.997	0.959	0.949	0.041	0.002
0.45	0.997	0.951	0.957	0.049	0.001
0.50	0.997	0.945	0.962	0.055	0.001
0.55	0.997	0.935	0.968	0.065	0.001
0.60	0.997	0.927	0.972	0.073	0.001
0.65	0.997	0.922	0.976	0.078	0.001
0.70	0.996	0.907	0.979	0.093	0.001
0.75	0.995	0.858	0.984	0.142	0.000
0.80	0.993	0.805	0.987	0.195	0.000
0.85	0.993	0.773	0.994	0.227	0.000
0.90	0.992	0.738	0.995	0.262	0.000
0.95	0.990	0.679	0.997	0.321	0.000

Our highest sensitivity was at the 0.05 threshold and our highest accuracy was at many thresholds, but we'll go with 0.5. Our model did well with training data.

```
rf.thresh1 <- 0.05
rf.thresh2 <- 0.5
```

```
results1.train.thresh <- c(results1.train.thresh, rf.thresh1)
results2.train.thresh <- c(results2.train.thresh, rf.thresh2)

results1.train.accuracy <- c(results1.train.accuracy, 0.988)
results1.train.sensitivity <- c(results1.train.sensitivity, 0.998)
results1.train.fpr <- c(results1.train.fpr, 0.012)
results1.train.precision <- c(results1.train.precision, 0.736)

results2.train.accuracy <- c(results2.train.accuracy, 0.997)
results2.train.sensitivity <- c(results2.train.sensitivity, 0.945)
results2.train.fpr <- c(results2.train.fpr, 0.001)
results2.train.precision <- c(results2.train.precision, 0.962)
```

```
results1.test.model <- c(results1.test.model, 'RF')
results2.test.model <- c(results2.test.model, 'RF')
```

```
rf.test.pred <- predict(rf.fit.tuned, haiti.test, type = 'prob')
```

```
# this also results in an esoteric error that I couldn't debug

# draw_test_ROC(rf.test.pred, 'Random Forest ROC (Test)')
```

```
rf.test.auc <- get_test_AUC(rf.test.pred)
```

```
rf.test.auc
```

Testing

```
#> [1] 0.9825992
```

Unfortunately, I had technical issues with drawing the test ROC curve for this model, but our test AUC value is lower than our training AUC. This isn't bad but it's not too promising.

```
results1.test.auc <- c(results1.test.auc, rf.test.auc)
results2.test.auc <- c(results2.test.auc, rf.test.auc)
```

```
results1.test.thresh <- c(results1.test.thresh, rf.thresh1)
results2.test.thresh <- c(results2.test.thresh, rf.thresh2)
```

Let's see how our model does at our chosen 0.05 threshold for highest sensitivity.

```
rf.test.conf_matrix1 <- create_test_conf_matrix(rf.test.pred,
                                                rf.thresh1)
rf.test.conf_matrix1
```

```
#> Confusion Matrix and Statistics
#>
#>           Reference
#> Prediction    True   False
#>      True    14016   67794
#>      False     464 1921903
#>
#>                Accuracy : 0.9659
#>                  95% CI : (0.9657, 0.9662)
#>     No Information Rate : 0.9928
#>     P-Value [Acc > NIR] : 1
#>
#>                   Kappa : 0.2823
#>
#>  Mcnemar's Test P-Value : <2e-16
#>
#>             Sensitivity : 0.967956
#>             Specificity : 0.965927
#>          Pos Pred Value : 0.171324
#>          Neg Pred Value : 0.999759
#>              Prevalence : 0.007225
#>          Detection Rate : 0.006993
#>    Detection Prevalence : 0.040820
#>       Balanced Accuracy : 0.966942
#>
#>        'Positive' Class : True
#>
```

Our sensitivity of about 96.8% isn't the worst we've seen but doesn't compare to our best models either.

```
# FPR
67794 / (67794 + 1921903)
```

```
#> [1] 0.03407252
```

```
# Precision
14016 / (14016 + 67794)
```

```
#> [1] 0.1713238
```

```
results1.test.accuracy <- c(results1.test.accuracy, 0.966)
results1.test.sensitivity <- c(results1.test.sensitivity, 0.968)
results1.test.fpr <- c(results1.test.fpr, 0.034)
results1.test.precision <- c(results1.test.precision, 0.171)
```

Now let's see how our model does at the 0.5 threshold for maximum accuracy.

```
rf.test.conf_matrix2 <- create_test_conf_matrix(rf.test.pred,
                                                rf.thresh2)
rf.test.conf_matrix2
```

```
#> Confusion Matrix and Statistics
#>
#>           Reference
#> Prediction    True   False
#>      True    11190    7203
#>      False    3290 1982494
#>
#>                Accuracy : 0.9948
#>                  95% CI : (0.9947, 0.9949)
#>     No Information Rate : 0.9928
#>     P-Value [Acc > NIR] : < 2.2e-16
#>
#>                   Kappa : 0.6782
#>
#>  Mcnemar's Test P-Value : < 2.2e-16
#>
#>             Sensitivity : 0.772790
#>             Specificity : 0.996380
#>          Pos Pred Value : 0.608384
#>          Neg Pred Value : 0.998343
#>              Prevalence : 0.007225
#>          Detection Rate : 0.005583
#>    Detection Prevalence : 0.009177
#>       Balanced Accuracy : 0.884585
#>
#>        'Positive' Class : True
#>
```

Our accuracy of about 99.5% is very good compared to our other models.

```
# FPR
7203 / (7203 + 1982494)
```

```
#> [1] 0.003620149
```

```
# Precision
11190 / (11190 + 7203)
```

```
#> [1] 0.6083836
```

```
results2.test.accuracy <- c(results2.test.accuracy, 0.995)
results2.test.sensitivity <- c(results2.test.sensitivity, 0.773)
results2.test.fpr <- c(results2.test.fpr, 0.004)
results2.test.precision <- c(results2.test.precision, 0.608)
```

Overall, I think model proved useful, but its sensitivity wasn't as good as our best models.

SVM — Linear Kernel

Training Let's tune for our cost parameter, C.

```
set.seed(710)

tuneGrid <- expand.grid(C = c(0.01, 0.1, 1, 10, 100))

svm_linear.fit <- caret::train(isbluetarp ~ Red + Green + Blue,
                           data = haiti.train,
                           method = 'svmLinear',
                           trControl = trControl,
                           tuneGrid = tuneGrid)
```

```
results1.train.model <- c(results1.train.model, 'SVM - L')
results2.train.model <- c(results2.train.model, 'SVM - L')
```

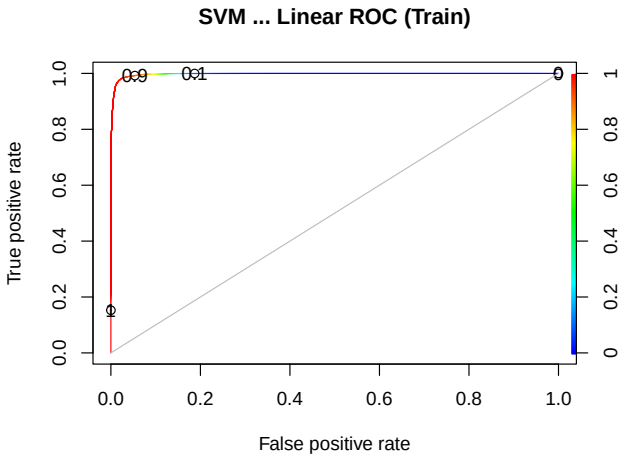
```
svm_linear.fit
```

```
#> Support Vector Machines with Linear Kernel
#>
#> 63241 samples
#>     3 predictor
#>     2 classes: 'True', 'False'
#>
#> No pre-processing
#> Resampling: Cross-Validated (10 fold)
#> Summary of sample sizes: 56916, 56917, 56916, 56918, 56917, 56917, ...
#> Resampling results across tuning parameters:
#>
#>   C      Accuracy   Kappa
#>   1e-02  0.9922993  0.8725117
#>   1e-01  0.9951930  0.9183423
#>   1e+00  0.9953985  0.9225662
#>   1e+01  0.9954144  0.9227467
#>   1e+02  0.9954144  0.9226789
#>
#> Accuracy was used to select the optimal model using the largest value.
#> The final value used for the model was C = 10.
```

Our optimal value of C was 10.

```
results.params.model <- c(results.params.model, 'SVM - L')
results.params.param <- c(results.params.param, 'C')
results.params.value <- c(results.params.value, 10)
```

```
draw_train_ROC(svm_linear.fit, 'SVM - Linear ROC (Train)')
```




```
svm_linear.train.auc <- get_train_AUC(svm_linear.fit)
svm_linear.train.auc
```

```
#> [1] 0.9977279
```

This ROC curve tightly hugs the upper left corner and our AUC value is very close to 1. This model seems like a very good fit.

```
results1.train.auc <- c(results1.train.auc, svm_linear.train.auc)
results2.train.auc <- c(results2.train.auc, svm_linear.train.auc)
```

```
svm_linear.train.threshold.stats <- caret::thresholder(svm_linear.fit,
                         threshold = seq(0.05, 0.95, by = 0.05),
                         statistics = "all")

svm_linear.train.threshold.stats$FNR <- (1 -
                                  svm_linear.train.threshold.stats$Sensitivity)
svm_linear.train.threshold.stats$FPR <- (1 -
                                  svm_linear.train.threshold.stats$Specificity)
```

```
svm_linear.train.threshold.stats %>%
  select("prob_threshold", "Accuracy", "Sensitivity", "Precision",
         "FNR", "FPR") %>%
  knitr::kable(digits = 3) %>%
  kableExtra::kable_styling(full_width = FALSE,
                            latex_options = "HOLD_position")
```

prob_threshold	Accuracy	Sensitivity	Precision	FNR	FPR
0.05	0.984	0.974	0.668	0.026	0.016
0.10	0.990	0.958	0.775	0.042	0.009
0.15	0.995	0.935	0.919	0.065	0.003
0.20	0.996	0.926	0.943	0.074	0.002
0.25	0.996	0.916	0.950	0.084	0.002
0.30	0.996	0.908	0.952	0.092	0.002
0.35	0.996	0.903	0.958	0.097	0.001
0.40	0.996	0.898	0.960	0.102	0.001
0.45	0.995	0.892	0.963	0.108	0.001
0.50	0.995	0.887	0.967	0.113	0.001
0.55	0.995	0.881	0.970	0.119	0.001
0.60	0.995	0.876	0.972	0.124	0.001
0.65	0.995	0.871	0.973	0.129	0.001
0.70	0.995	0.866	0.975	0.134	0.001
0.75	0.995	0.860	0.978	0.140	0.001
0.80	0.995	0.851	0.980	0.149	0.001
0.85	0.994	0.842	0.983	0.158	0.000
0.90	0.994	0.830	0.986	0.170	0.000
0.95	0.994	0.809	0.988	0.191	0.000

Our threshold for highest sensitivity is 0.05. Among the thresholds with highest accuracy, I'll go with 0.4 since it's the closest to 0.5. The accuracy at 0.4 is pretty impressive while the sensitivity at 0.05 seems decent, but not great.

```
svm_linear.thresh1 <- 0.05
svm_linear.thresh2 <- 0.4
```

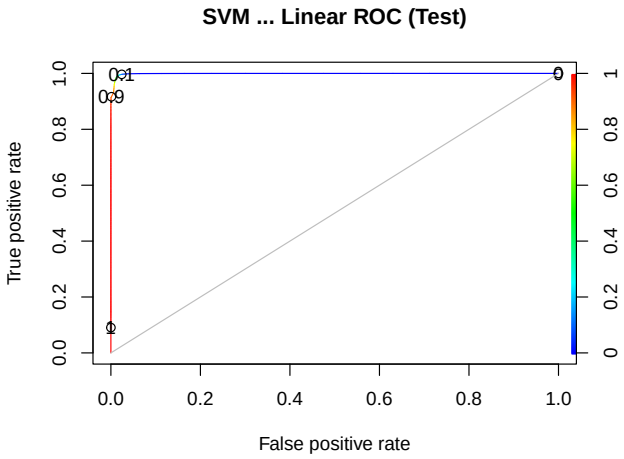
```
results1.train.thresh <- c(results1.train.thresh, svm_linear.thresh1)
results2.train.thresh <- c(results2.train.thresh, svm_linear.thresh2)

results1.train.accuracy <- c(results1.train.accuracy, 0.984)
results1.train.sensitivity <- c(results1.train.sensitivity, 0.974)
results1.train.fpr <- c(results1.train.fpr, 0.016)
results1.train.precision <- c(results1.train.precision, 0.668)

results2.train.accuracy <- c(results2.train.accuracy, 0.996)
results2.train.sensitivity <- c(results2.train.sensitivity, 0.898)
results2.train.fpr <- c(results2.train.fpr, 0.001)
results2.train.precision <- c(results2.train.precision, 0.960)
```

```
results1.test.model <- c(results1.test.model, 'SVM - Linear')
results2.test.model <- c(results2.test.model, 'SVM - Linear')
```

```
svm_linear.test.pred <- predict(svm_linear.fit,
                                haiti.test,
                                type = 'prob')
```

```
draw_test_ROC(svm_linear.test.pred, 'SVM - Linear ROC (Test)')
```

Testing




```
svm_linear.test.auc <- get_test_AUC(svm_linear.test.pred)
svm_linear.test.auc
```

```
#> [1] 0.9991331
```

This ROC curve pretty tightly hugs the upper left corner and our AUC value is very close to 1. This model seems to be performing well.

```
results1.test.auc <- c(results1.test.auc, svm_linear.test.auc)
results2.test.auc <- c(results2.test.auc, svm_linear.test.auc)
```

```
results1.test.thresh <- c(results1.test.thresh, svm_linear.thresh1)
results2.test.thresh <- c(results2.test.thresh, svm_linear.thresh2)
```

Let's see how our model does at our chosen 0.05 threshold for highest sensitivity.

```
svm_linear.test.conf_matrix1 <- create_test_conf_matrix(
  svm_linear.test.pred, svm_linear.thresh1)
svm_linear.test.conf_matrix1
```

```
#> Confusion Matrix and Statistics
#>
#>           Reference
#> Prediction    True   False
#>      True    14479  214620
#>      False       1 1775077
#>
#>                Accuracy : 0.8929
#>                  95% CI : (0.8925, 0.8933)
#>     No Information Rate : 0.9928
#>     P-Value [Acc > NIR] : 1
#>
#>                   Kappa : 0.1067
#>
#>  Mcnemar's Test P-Value : <2e-16
#>
#>             Sensitivity : 0.999931
#>             Specificity : 0.892134
#>          Pos Pred Value : 0.063200
#>          Neg Pred Value : 0.999999
#>              Prevalence : 0.007225
#>          Detection Rate : 0.007224
```

```
#>    Detection Prevalence : 0.114311
#>       Balanced Accuracy : 0.946033
#>
#>        'Positive' Class : True
#>
```

Our sensitivity with this model is extremely close to 1. This is our best model yet for this metric!

```
# FPR
214620 / (214620 + 1775077)
```

```
#> [1] 0.1078657
```

```
# Precision
14479 / (14479 + 214620)
```

```
#> [1] 0.06319975
```

```
results1.test.accuracy <- c(results1.test.accuracy, 0.893)
results1.test.sensitivity <- c(results1.test.sensitivity, 0.999)
results1.test.fpr <- c(results1.test.fpr, 0.108)
results1.test.precision <- c(results1.test.precision, 0.063)
```

Now let's see how our model does at the 0.4 threshold for maximum accuracy.

```
svm_linear.test.conf_matrix2 <- create_test_conf_matrix(
  svm_linear.test.pred, svm_linear.thresh2)
svm_linear.test.conf_matrix2
```

```
#> Confusion Matrix and Statistics
#>
#>           Reference
#> Prediction    True   False
#>      True    14341   50244
#>      False     139 1939453
#>
#>                Accuracy : 0.9749
#>                  95% CI : (0.9746, 0.9751)
#>     No Information Rate : 0.9928
#>     P-Value [Acc > NIR] : 1
#>
#>                   Kappa : 0.3552
#>
#>  Mcnemar's Test P-Value : <2e-16
#>
#>             Sensitivity : 0.990401
#>             Specificity : 0.974748
#>          Pos Pred Value : 0.222048
#>          Neg Pred Value : 0.999928
#>              Prevalence : 0.007225
#>          Detection Rate : 0.007156
#>    Detection Prevalence : 0.032225
#>       Balanced Accuracy : 0.982574
#>
#>        'Positive' Class : True
#>
```

Our accuracy is about 97.5%, which is pretty underwhelming compared to our other models. Interestingly,

our sensitivity at this threshold is still quite high.

```
# FPR
50244 / (50244 + 1939453)
```

```
#> [1] 0.02525209
```

```
# Precision
14341 / (14341 + 50244)
```

```
#> [1] 0.2220485
```

```
results2.test.accuracy <- c(results2.test.accuracy, 0.975)
results2.test.sensitivity <- c(results2.test.sensitivity, 0.990)
results2.test.fpr <- c(results2.test.fpr, 0.025)
results2.test.precision <- c(results2.test.precision, 0.222)
```

This model performed very well regarding sensitivity but was underwhelming for accuracy.

SVM — Polynomial Kernel

Training Let's tune for cost (C) and which degree of polynomial kernel. I chose 1 for the scale parameter because according to the documentation for the kernlab and caret packages on parsnip, the default value for the scale is 1.

```
set.seed(710)

tuneGrid <- expand.grid(degree = c(2, 3, 4, 5),
                        scale = 1,
                        C = c(0.01, 0.1, 1, 10, 100))

svm_poly.fit <- caret::train(isbluetarp ~ Red + Green + Blue,
                             data = haiti.train,
                             method = 'svmPoly',
                             trControl = trControl,
                             tuneGrid = tuneGrid)
```

```
results1.train.model <- c(results1.train.model, 'SVM - P')
results2.train.model <- c(results2.train.model, 'SVM - P')
```

```
svm_poly.fit
```

```
#> Support Vector Machines with Polynomial Kernel
#>
#> 63241 samples
#>     3 predictor
#>     2 classes: 'True', 'False'
#>
#> No pre-processing
#> Resampling: Cross-Validated (10 fold)
#> Summary of sample sizes: 56916, 56917, 56916, 56918, 56917, 56917, ...
#> Resampling results across tuning parameters:
#>
#>   degree  C      Accuracy   Kappa
#>   2       1e-02  0.9954302  0.9250679
#>   2       1e-01  0.9957148  0.9294863
#>   2       1e+00  0.9957306  0.9293196
#>   2       1e+01  0.9958097  0.9303919
```

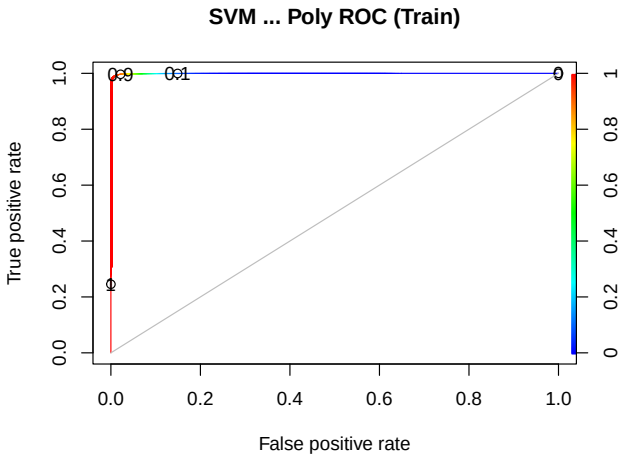
```
#>   2       1e+02  0.9963947  0.9403474
#>   3       1e-02  0.9962050  0.9386992
#>   3       1e-01  0.9961259  0.9365966
#>   3       1e+00  0.9964264  0.9413801
#>   3       1e+01  0.9968217  0.9482762
#>   3       1e+02  0.9968691  0.9489779
#>   4       1e-02  0.9963473  0.9414028
#>   4       1e-01  0.9963631  0.9404365
#>   4       1e+00  0.9967110  0.9461222
#>   4       1e+01  0.9969640  0.9505110
#>   4       1e+02  0.9970114  0.9511101
#>   5       1e-02  0.9965845  0.9443778
#>   5       1e-01  0.9966319  0.9446956
#>   5       1e+00  0.9969007  0.9491803
#>   5       1e+01  0.9970905  0.9522463
#>   5       1e+02  0.9970905  0.9520588
#>
#> Tuning parameter 'scale' was held constant at a value of 1
#> Accuracy was used to select the optimal model using the largest value.
#> The final values used for the model were degree = 5, scale = 1 and C = 100.
```

Our tuned degree was 5 and our tuned cost was 100.

```
results.params.model <- c(results.params.model, 'SVM - P')
results.params.param <- c(results.params.param, 'C')
results.params.value <- c(results.params.value, 100)
```

```
results.params.model <- c(results.params.model, 'SVM - P')
results.params.param <- c(results.params.param, 'degree')
results.params.value <- c(results.params.value, 5)
```

```
draw_train_ROC(svm_poly.fit, 'SVM - Poly ROC (Train)')
```




```
svm_poly.train.auc <- get_train_AUC(svm_poly.fit)
svm_poly.train.auc
```

```
#> [1] 0.9995237
```

This ROC curve tightly hugs the upper left corner and our AUC value is very close to 1. This model seems like a good fit.

```
results1.train.auc <- c(results1.train.auc, svm_poly.train.auc)
results2.train.auc <- c(results2.train.auc, svm_poly.train.auc)
```

```
svm_poly.train.threshold.stats <- caret::thresholder(svm_poly.fit,
                          threshold = seq(0.05, 0.95, by = 0.05),
                          statistics = "all")

svm_poly.train.threshold.stats$FNR <- (1 -
                                  svm_poly.train.threshold.stats$Sensitivity)
svm_poly.train.threshold.stats$FPR <- (1 -
                                  svm_poly.train.threshold.stats$Specificity)
```

```
svm_poly.train.threshold.stats %>%
  select("prob_threshold", "Accuracy", "Sensitivity", "Precision",
         "FNR", "FPR") %>%
  knitr::kable(digits = 3) %>%
  kableExtra::kable_styling(full_width = FALSE,
                            latex_options = "HOLD_position")
```

prob_threshold	Accuracy	Sensitivity	Precision	FNR	FPR
0.05	0.989	0.987	0.773	0.013	0.011
0.10	0.996	0.978	0.918	0.022	0.003
0.15	0.997	0.967	0.939	0.033	0.002
0.20	0.997	0.961	0.944	0.039	0.002
0.25	0.997	0.956	0.954	0.044	0.002
0.30	0.997	0.952	0.958	0.048	0.001
0.35	0.997	0.948	0.962	0.052	0.001
0.40	0.997	0.946	0.965	0.054	0.001
0.45	0.997	0.940	0.970	0.060	0.001
0.50	0.997	0.935	0.973	0.065	0.001
0.55	0.997	0.929	0.976	0.071	0.001
0.60	0.997	0.921	0.979	0.079	0.001
0.65	0.997	0.915	0.982	0.085	0.001
0.70	0.997	0.907	0.985	0.093	0.000
0.75	0.996	0.897	0.989	0.103	0.000
0.80	0.996	0.876	0.992	0.124	0.000
0.85	0.995	0.849	0.995	0.151	0.000
0.90	0.994	0.816	0.996	0.184	0.000
0.95	0.993	0.782	0.999	0.218	0.000

Our threshold for best sensitivity is 0.05. I'll go with 0.5 as our threshold for best accuracy. Our sensitivity at 0.05 and accuracy at 0.5 both seem fairly good compared to our other models.

```
svm_poly.thresh1 <- 0.05
svm_poly.thresh2 <- 0.5
```

```
results1.train.thresh <- c(results1.train.thresh, svm_poly.thresh1)
results2.train.thresh <- c(results2.train.thresh, svm_poly.thresh2)

results1.train.accuracy <- c(results1.train.accuracy, 0.989)
results1.train.sensitivity <- c(results1.train.sensitivity, 0.987)
results1.train.fpr <- c(results1.train.fpr, 0.011)
results1.train.precision <- c(results1.train.precision, 0.773)
```

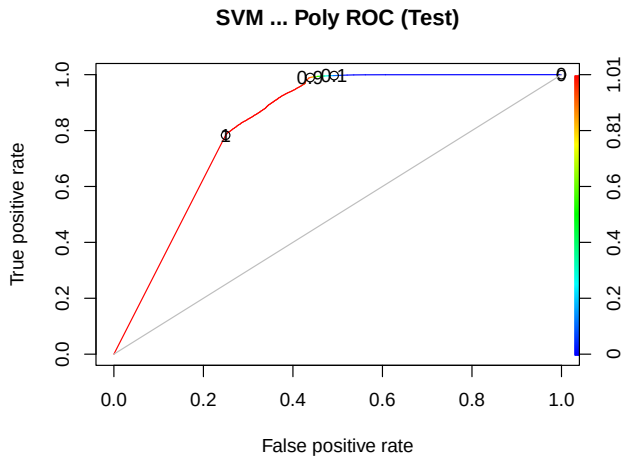
```
results2.train.accuracy <- c(results2.train.accuracy, 0.997)
results2.train.sensitivity <- c(results2.train.sensitivity, 0.935)
results2.train.fpr <- c(results2.train.fpr, 0.001)
results2.train.precision <- c(results2.train.precision, 0.973)
```

```
results1.test.model <- c(results1.test.model, 'SVM - Poly')
results2.test.model <- c(results2.test.model, 'SVM - Poly')
```

```
svm_poly.test.pred <- predict(svm_poly.fit,
                              haiti.test,
                              type = 'prob')
```

```
draw_test_ROC(svm_poly.test.pred, 'SVM - Poly ROC (Test)')
```

Testing




```
svm_poly.test.auc <- get_test_AUC(svm_poly.test.pred)
svm_poly.test.auc
```

```
#> [1] 0.836577
```

This ROC curve and AUC value are quite poor. Although this model seemed like a good fit, it's not looking great with the test data.

```
results1.test.auc <- c(results1.test.auc, svm_poly.test.auc)
results2.test.auc <- c(results2.test.auc, svm_poly.test.auc)
```

```
results1.test.thresh <- c(results1.test.thresh, svm_poly.thresh1)
results2.test.thresh <- c(results2.test.thresh, svm_poly.thresh2)
```

Let's see how our model does at our chosen 0.05 threshold for highest sensitivity.

```
svm_poly.test.conf_matrix1 <- create_test_conf_matrix(
  svm_poly.test.pred, svm_poly.thresh1)
svm_poly.test.conf_matrix1
```

```
#> Confusion Matrix and Statistics
#>
#>           Reference
```

```
#> Prediction    True   False
#>      True     8278   47883
#>      False    6202 1941814
#>
#>                Accuracy : 0.973
#>                  95% CI : (0.9728, 0.9732)
#>     No Information Rate : 0.9928
#>     P-Value [Acc > NIR] : 1
#>
#>                   Kappa : 0.2255
#>
#>  Mcnemar's Test P-Value : <2e-16
#>
#>             Sensitivity : 0.571685
#>             Specificity : 0.975935
#>          Pos Pred Value : 0.147398
#>          Neg Pred Value : 0.996816
#>              Prevalence : 0.007225
#>          Detection Rate : 0.004130
#>    Detection Prevalence : 0.028022
#>       Balanced Accuracy : 0.773810
#>
#>        'Positive' Class : True
#>
```

The sensitivity at our 0.05 threshold is very poor, but interestingly, the accuracy is pretty high. Still, it's not a good result.

```
# FPR
47883 / (47883 + 1941814)
```

```
#> [1] 0.02406547
```

```
# Precision
8278 / (8278 + 47883)
```

```
#> [1] 0.1473977
```

```
results1.test.accuracy <- c(results1.test.accuracy, 0.973)
results1.test.sensitivity <- c(results1.test.sensitivity, 0.572)
results1.test.fpr <- c(results1.test.fpr, 0.024)
results1.test.precision <- c(results1.test.precision, 0.147)
```

Now, let's see how our model does at our chosen 0.5 threshold for highest accuracy.

```
svm_poly.test.conf_matrix2 <- create_test_conf_matrix(
  svm_poly.test.pred, svm_poly.thresh2)
svm_poly.test.conf_matrix2
```

```
#> Confusion Matrix and Statistics
#>
#>           Reference
#> Prediction    True   False
#>      True     7876   11976
#>      False    6604 1977721
#>
#>                Accuracy : 0.9907
#>                  95% CI : (0.9906, 0.9909)
```

```
#>     No Information Rate : 0.9928
#>     P-Value [Acc > NIR] : 1
#>
#>                   Kappa : 0.4543
#>
#>  Mcnemar's Test P-Value : <2e-16
#>
#>             Sensitivity : 0.543923
#>             Specificity : 0.993981
#>          Pos Pred Value : 0.396736
#>          Neg Pred Value : 0.996672
#>              Prevalence : 0.007225
#>          Detection Rate : 0.003930
#>    Detection Prevalence : 0.009905
#>       Balanced Accuracy : 0.768952
#>
#>        'Positive' Class : True
#>
```

Our accuracy isn't too bad, but it's not great overall. Given how accurate the model was at the 0.05 threshold, I expected better at the 0.5 threshold.

```
# FPR
11976 / (11976 + 1977721)
```

```
#> [1] 0.006019007
```

```
# Precision
7876 / (7876 + 11976)
```

```
#> [1] 0.3967358
```

```
results2.test.accuracy <- c(results2.test.accuracy, 0.991)
results2.test.sensitivity <- c(results2.test.sensitivity, 0.544)
results2.test.fpr <- c(results2.test.fpr, 0.006)
results2.test.precision <- c(results2.test.precision, 0.397)
```

This model seemed like a good fit but performed disappointingly with the test data. It did quite poorly with our primary metric, sensitivity, and its accuracy was unimpressive.

SVM — Radial Kernel

Training Let's start by tuning for our sigma and cost (C) parameters.

```
set.seed(710)

tuneGrid <- expand.grid(sigma = c(0.1, 1, 10),
                        C = c(0.01, 0.1, 1, 10, 100))

svm_radial.fit <- caret::train(isbluetarp ~ Red + Green + Blue,
                           data = haiti.train,
                           method = 'svmRadial',
                           trControl = trControl,
                           tuneGrid = tuneGrid)
```

```
results1.train.model <- c(results1.train.model, 'SVM - R')
results2.train.model <- c(results2.train.model, 'SVM - R')
```

```
svm_radial.fit
```

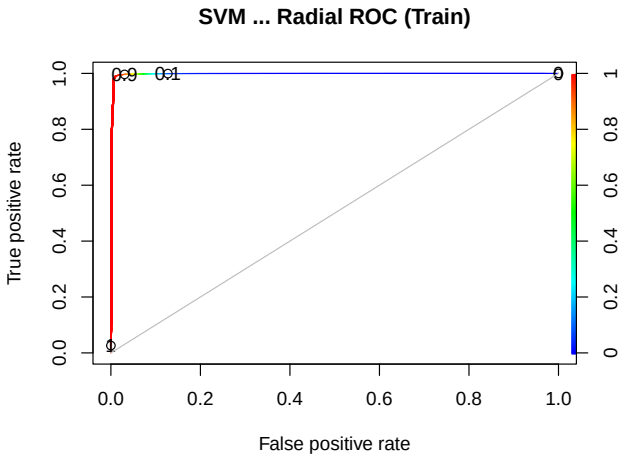
```
#> Support Vector Machines with Radial Basis Function Kernel
#>
#> 63241 samples
#>     3 predictor
#>     2 classes: 'True', 'False'
#>
#> No pre-processing
#> Resampling: Cross-Validated (10 fold)
#> Summary of sample sizes: 56916, 56917, 56916, 56918, 56917, 56917, ...
#> Resampling results across tuning parameters:
#>
#>   sigma  C      Accuracy   Kappa
#>    0.1   1e-02  0.9893740  0.8369285
#>    0.1   1e-01  0.9950507  0.9184880
#>    0.1   1e+00  0.9957622  0.9309845
#>    0.1   1e+01  0.9960943  0.9360717
#>    0.1   1e+02  0.9962999  0.9394763
#>    1.0   1e-02  0.9944656  0.9076059
#>    1.0   1e-01  0.9959203  0.9335405
#>    1.0   1e+00  0.9964422  0.9419419
#>    1.0   1e+01  0.9967584  0.9470780
#>    1.0   1e+02  0.9970905  0.9526646
#>   10.0   1e-02  0.9960310  0.9362244
#>   10.0   1e-01  0.9964896  0.9424379
#>   10.0   1e+00  0.9969482  0.9499854
#>   10.0   1e+01  0.9971696  0.9536879
#>   10.0   1e+02  0.9974858  0.9592018
#>
#> Accuracy was used to select the optimal model using the largest value.
#> The final values used for the model were sigma = 10 and C = 100.
```

Our tuned values of sigma and C are 10 and 100, respectively.

```
results.params.model <- c(results.params.model, 'SVM - R')
results.params.param <- c(results.params.param, 'C')
results.params.value <- c(results.params.value, 100)
```

```
results.params.model <- c(results.params.model, 'SVM - R')
results.params.param <- c(results.params.param, 'sigma')
results.params.value <- c(results.params.value, 10)
```

```
draw_train_ROC(svm_radial.fit, 'SVM - Radial ROC (Train)')
```




```
svm_radial.train.auc <- get_train_AUC(svm_radial.fit)
svm_radial.train.auc
```

```
#> [1] 0.9985334
```

This ROC curve tightly hugs the upper left corner and our AUC value is close to 1. This model seems like a good fit.

```
results1.train.auc <- c(results1.train.auc, svm_radial.train.auc)
results2.train.auc <- c(results2.train.auc, svm_radial.train.auc)
```

```
svm_radial.train.threshold.stats <- caret::thresholder(svm_radial.fit,
                         threshold = seq(0.05, 0.95, by = 0.05),
                         statistics = "all")

svm_radial.train.threshold.stats$FNR <- (1 -
                                  svm_radial.train.threshold.stats$Sensitivity)
svm_radial.train.threshold.stats$FPR <- (1 -
                                  svm_radial.train.threshold.stats$Specificity)
```

```
svm_radial.train.threshold.stats %>%
  select("prob_threshold", "Accuracy", "Sensitivity", "Precision",
         "FNR", "FPR") %>%
  knitr::kable(digits = 3) %>%
  kableExtra::kable_styling(full_width = FALSE,
                            latex_options = "HOLD_position")
```

prob_threshold	Accuracy	Sensitivity	Precision	FNR	FPR
0.05	0.997	0.976	0.942	0.024	0.002
0.10	0.997	0.971	0.948	0.029	0.002
0.15	0.997	0.968	0.952	0.032	0.002
0.20	0.997	0.966	0.955	0.034	0.002
0.25	0.998	0.965	0.957	0.035	0.001
0.30	0.997	0.963	0.959	0.037	0.001
0.35	0.997	0.960	0.960	0.040	0.001
0.40	0.997	0.958	0.961	0.042	0.001
0.45	0.997	0.957	0.962	0.043	0.001
0.50	0.997	0.955	0.966	0.045	0.001
0.55	0.997	0.952	0.966	0.048	0.001
0.60	0.997	0.950	0.967	0.050	0.001
0.65	0.997	0.945	0.968	0.055	0.001
0.70	0.997	0.942	0.970	0.058	0.001
0.75	0.997	0.940	0.974	0.060	0.001
0.80	0.997	0.933	0.975	0.067	0.001
0.85	0.997	0.926	0.977	0.074	0.001
0.90	0.997	0.910	0.980	0.090	0.001
0.95	0.994	0.831	0.986	0.169	0.000

A threshold of 0.05 has our highest sensitivity and a threshold of 0.25 has our highest accuracy.

```
svm_radial.thresh1 <- 0.05
svm_radial.thresh2 <- 0.25
```

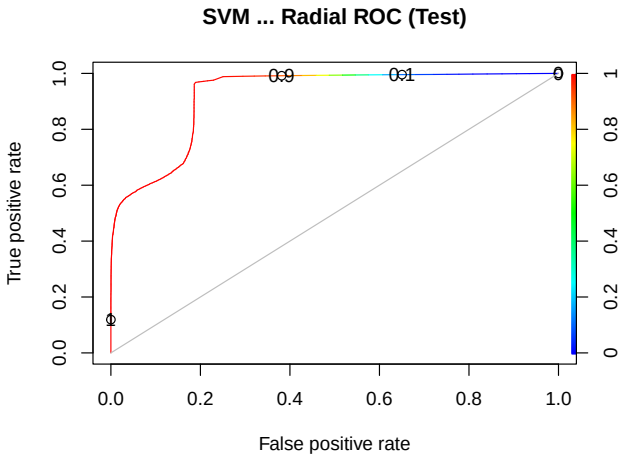
```
results1.train.thresh <- c(results1.train.thresh, svm_radial.thresh1)
results2.train.thresh <- c(results2.train.thresh, svm_radial.thresh2)

results1.train.accuracy <- c(results1.train.accuracy, 0.997)
results1.train.sensitivity <- c(results1.train.sensitivity, 0.976)
results1.train.fpr <- c(results1.train.fpr, 0.002)
results1.train.precision <- c(results1.train.precision, 0.942)

results2.train.accuracy <- c(results2.train.accuracy, 0.998)
results2.train.sensitivity <- c(results2.train.sensitivity, 0.965)
results2.train.fpr <- c(results2.train.fpr, 0.001)
results2.train.precision <- c(results2.train.precision, 0.957)
```

```
results1.test.model <- c(results1.test.model, 'SVM - R')
results2.test.model <- c(results2.test.model, 'SVM - R')
```

```
svm_radial.test.pred <- predict(svm_radial.fit,
                                haiti.test,
                                type = 'prob')
```

```
draw_test_ROC(svm_radial.test.pred, 'SVM - Radial ROC (Test)')
```

Testing




```
svm_radial.test.auc <- get_test_AUC(svm_radial.test.pred)
svm_radial.test.auc
```

```
#> [1] 0.9216632
```

This ROC curve has a bizarre shape but its AUC value, while relatively low, isn't the worst we've seen among our models.

```
results1.test.auc <- c(results1.test.auc, svm_radial.test.auc)
results2.test.auc <- c(results2.test.auc, svm_radial.test.auc)
```

```
results1.test.thresh <- c(results1.test.thresh, svm_radial.thresh1)
results2.test.thresh <- c(results2.test.thresh, svm_radial.thresh2)
```

Let's see how our model does at our chosen 0.05 threshold for highest sensitivity.

```
svm_radial.test.conf_matrix1 <- create_test_conf_matrix(
  svm_radial.test.pred, svm_radial.thresh1)
svm_radial.test.conf_matrix1
```

```
#> Confusion Matrix and Statistics
#>
#>           Reference
#> Prediction    True   False
#>      True     9585   18427
#>      False    4895 1971270
#>
#>                Accuracy : 0.9884
#>                  95% CI : (0.9882, 0.9885)
#>     No Information Rate : 0.9928
#>     P-Value [Acc > NIR] : 1
#>
#>                   Kappa : 0.4459
#>
#>  Mcnemar's Test P-Value : <2e-16
#>
#>             Sensitivity : 0.661948
#>             Specificity : 0.990739
#>          Pos Pred Value : 0.342175
#>          Neg Pred Value : 0.997523
#>              Prevalence : 0.007225
#>          Detection Rate : 0.004783
```

```
#>    Detection Prevalence : 0.013977
#>       Balanced Accuracy : 0.826343
#>
#>        'Positive' Class : True
#>
```

Our sensitivity is quite poor.

```
# FPR
18427 / (18427 + 1971270)
```

```
#> [1] 0.009261209
```

```
# Precision
9585 / (9585 + 18427)
```

```
#> [1] 0.3421748
```

```
results1.test.accuracy <- c(results1.test.accuracy, 0.988)
results1.test.sensitivity <- c(results1.test.sensitivity, 0.662)
results1.test.fpr <- c(results1.test.fpr, 0.009)
results1.test.precision <- c(results1.test.precision, 0.342)
```

Now, let's see how our model does at our chosen 0.5 threshold for highest accuracy.

```
svm_radial.test.conf_matrix2 <- create_test_conf_matrix(
  svm_radial.test.pred, svm_radial.thresh2)
svm_radial.test.conf_matrix2
```

```
#> Confusion Matrix and Statistics
#>
#>           Reference
#> Prediction    True   False
#>      True     7905   14046
#>      False    6575 1975651
#>
#>                Accuracy : 0.9897
#>                  95% CI : (0.9896, 0.9899)
#>     No Information Rate : 0.9928
#>     P-Value [Acc > NIR] : 1
#>
#>                   Kappa : 0.429
#>
#>  Mcnemar's Test P-Value : <2e-16
#>
#>             Sensitivity : 0.545925
#>             Specificity : 0.992941
#>          Pos Pred Value : 0.360120
#>          Neg Pred Value : 0.996683
#>              Prevalence : 0.007225
#>          Detection Rate : 0.003944
#>    Detection Prevalence : 0.010953
#>       Balanced Accuracy : 0.769433
#>
#>        'Positive' Class : True
#>
```

Our accuracy isn't too poor but it's also not good.

```
# FPR
14046 / (14046 + 1975651)
```

```
#> [1] 0.007059366
```

```
# Precision
7905 / (7905 + 14046)
```

```
#> [1] 0.3601203
```

```
results2.test.accuracy <- c(results2.test.accuracy, 0.990)
results2.test.sensitivity <- c(results2.test.sensitivity, 0.546)
results2.test.fpr <- c(results2.test.fpr, 0.007)
results2.test.precision <- c(results2.test.precision, 0.360)
```

Overall, this model was sub-par. Our sensitivity was bad and our accuracy was decent, at best.

Results

Training Results

Here are our *training results* at the ideal threshold *for sensitivity*.

```
train.results.1 <- data.frame(
  results1.train.model,
  results1.train.thresh,
  round(results1.train.auc, 3),
  results1.train.accuracy,
  results1.train.sensitivity,
  results1.train.fpr,
  results1.train.precision
)

colnames(train.results.1) <- c('Model',
                               'Threshold',
                               'AUC Value',
                               'Accuracy',
                               'Sensitivity',
                               'False Positive Rate',
                               'Precision')
```

```
knitr::kable(train.results.1, 'simple')
```

Model	Threshold	AUC Value	Accuracy	Sensitivity	False Positive Rate	Precision
Log Reg	0.05	0.998	0.987	0.976	0.013	0.717
LDA	0.05	0.989	0.981	0.860	0.015	0.656
QDA	0.05	0.998	0.988	0.942	0.010	0.755
KNN	0.05	0.998	0.992	0.996	0.008	0.797
Ridge	0.05	0.851	0.917	0.895	0.083	0.264
Lasso	0.05	0.801	0.987	0.979	0.013	0.717
RF	0.05	0.996	0.988	0.998	0.012	0.736
SVM — L	0.05	0.998	0.984	0.974	0.016	0.668
SVM — P	0.05	1.000	0.989	0.987	0.011	0.773
SVM — R	0.05	0.999	0.997	0.976	0.002	0.942

Here are our *training results* at the ideal threshold *for accuracy*.

```
train.results.2 <- data.frame(
  results2.train.model,
  results2.train.thresh,
  round(results2.train.auc, 3),
  results2.train.accuracy,
  results2.train.sensitivity,
  results2.train.fpr,
  results2.train.precision
)

colnames(train.results.2) <- c('Model',
                               'Threshold',
                               'AUC Value',
                               'Accuracy',
                               'Sensitivity',
                               'False Positive Rate',
                               'Precision')
```

```
knitr::kable(train.results.2, 'simple')
```

Model	Threshold	AUC Value	Accuracy	Sensitivity	False Positive Rate	Precision
Log Reg	0.35	0.998	0.996	0.902	0.001	0.955
LDA	0.75	0.989	0.985	0.773	0.009	0.751
QDA	0.50	0.998	0.995	0.841	0.000	0.989
KNN	0.50	0.998	0.997	0.958	0.001	0.957
Ridge	0.25	0.851	0.987	0.582	0.000	0.999
Lasso	0.50	0.801	0.995	0.867	0.001	0.976
RF	0.50	0.996	0.997	0.945	0.001	0.962
SVM — L	0.40	0.998	0.996	0.898	0.001	0.960
SVM — P	0.50	1.000	0.997	0.935	0.001	0.973
SVM — R	0.25	0.999	0.998	0.965	0.001	0.957

Here are our *final parameters* for the models that needed *tuning*.

```
param.results <- data.frame(
  results.params.model,
  results.params.param,
  results.params.value
)

colnames(param.results) <- c('Model',
                             'Parameter',
                             'Value')
```

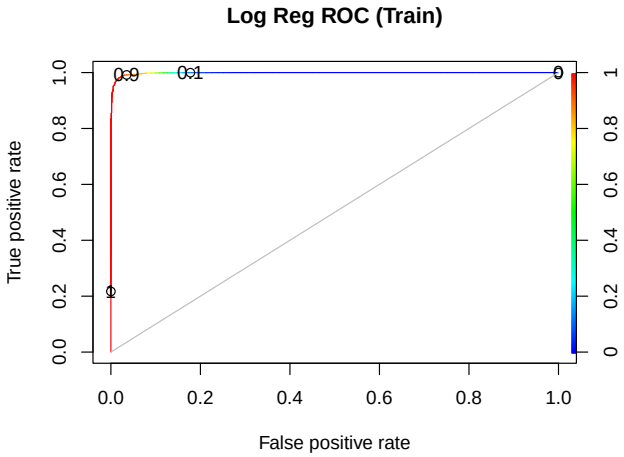
```
knitr::kable(param.results, 'simple')
```

Model	Parameter	Value
KNN	K	5e+00
Ridge	lambda	4e-03
Ridge	alpha	0e+00
Lasso	lambda	1e-04
Lasso	alpha	1e+00

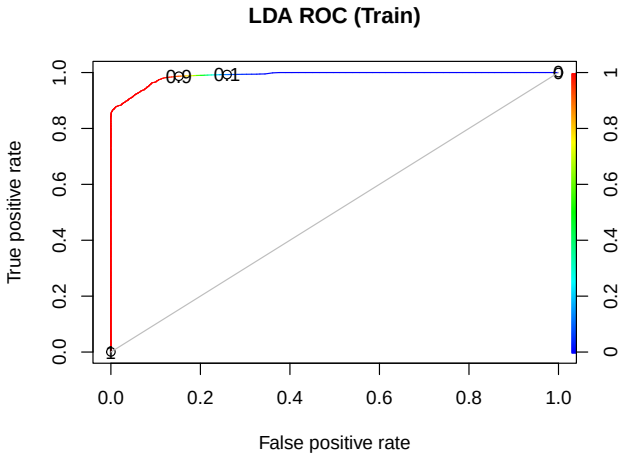
Model	Parameter	Value
RF	mtry	1e+00
RF	ntrees	3e+02
SVM — L	C	1e+01
SVM — P	C	1e+02
SVM — P	degree	5e+00
SVM — R	C	1e+02
SVM — R	sigma	1e+01

Here are our *training ROC curves*.

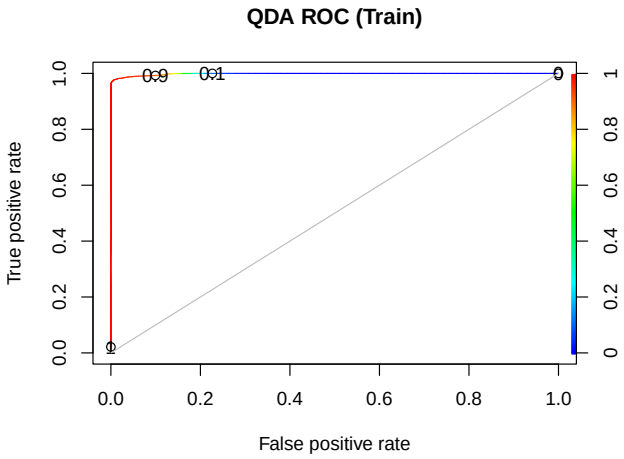
```
draw_train_ROC(logreg.fit, 'Log Reg ROC (Train)')
```



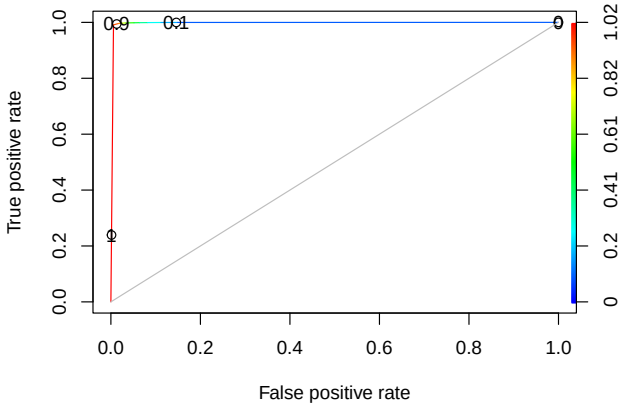

```
draw_train_ROC(lda.fit, 'LDA ROC (Train)')
```



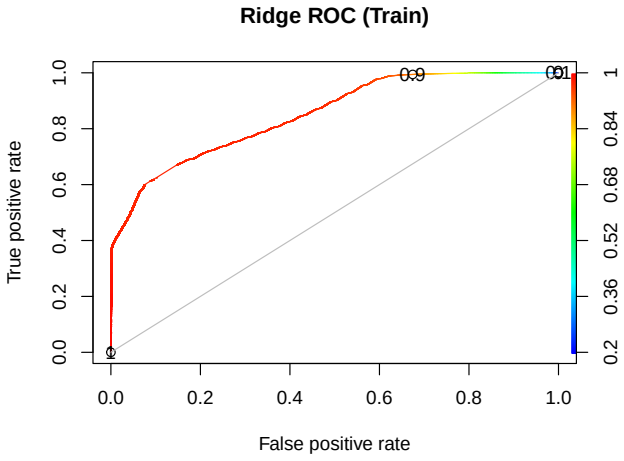

```
draw_train_ROC(qda.fit, 'QDA ROC (Train)')
```



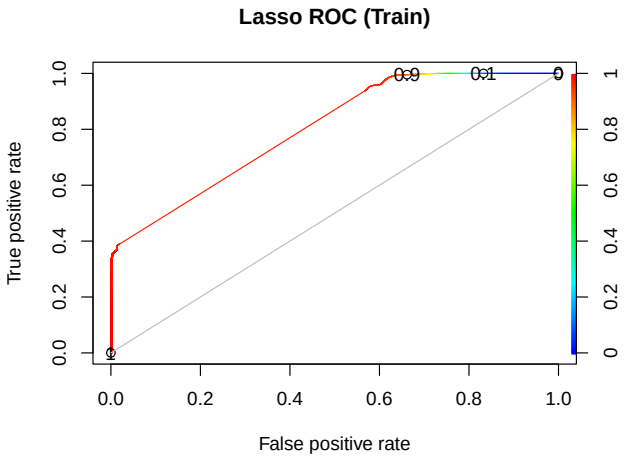

```
draw_train_ROC(knn.fit, 'KNN ROC (Train)')
```



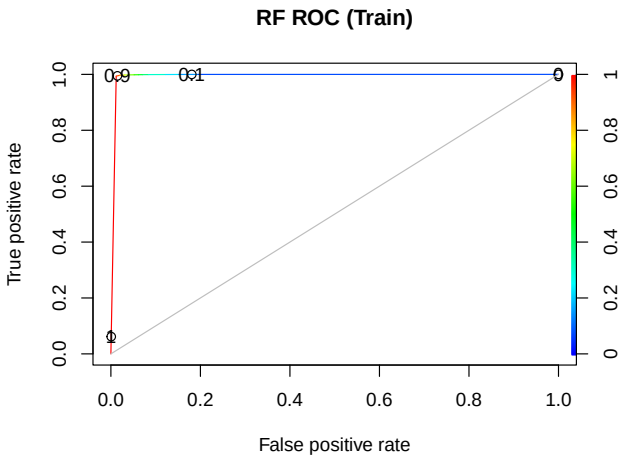

```
draw_train_ROC(ridge.fit, 'Ridge ROC (Train)')
```



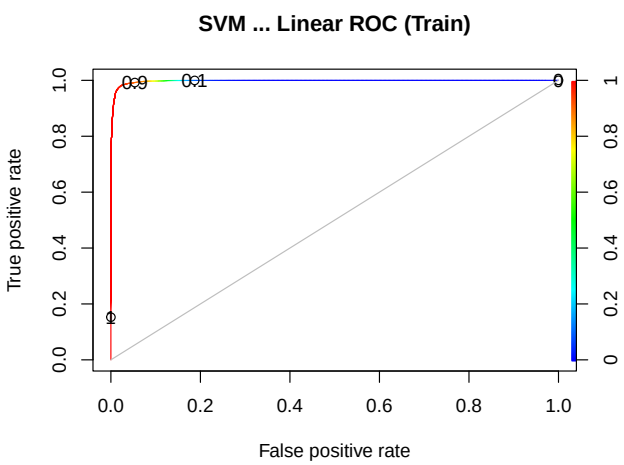

```
draw_train_ROC(lasso.fit, 'Lasso ROC (Train)')
```



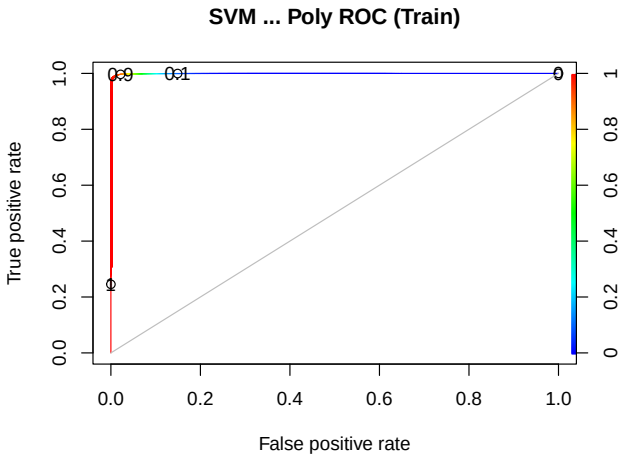

```
draw_train_ROC(rf.fit, 'RF ROC (Train)')
```



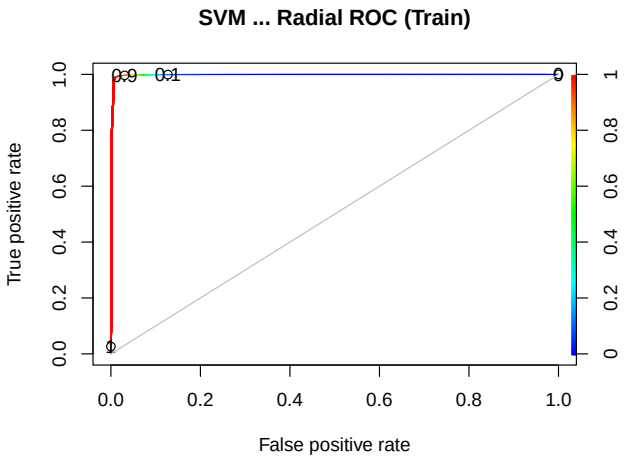

```
draw_train_ROC(svm_linear.fit, 'SVM - Linear ROC (Train)')
```




```
draw_train_ROC(svm_poly.fit, 'SVM - Poly ROC (Train)')
```




```
draw_train_ROC(svm_radial.fit, 'SVM - Radial ROC (Train)')
```



Testing Results

Here are our *test results* at the ideal threshold *for sensitivity*.

```
test.results.1 <- data.frame(
  results1.test.model,
  results1.test.thresh,
  round(results1.test.auc, 3),
  results1.test.accuracy,
  results1.test.sensitivity,
  results1.test.fpr,
  results1.test.precision
)

colnames(test.results.1) <- c('Model',
                              'Threshold',
                              'AUC Value',
                              'Accuracy',
                              'Sensitivity',
                              'False Positive Rate',
                              'Precision')
```

```
knitr::kable(test.results.1, 'simple')
```

Model	Threshold	AUC Value	Accuracy	Sensitivity	False Positive Rate	Precision
Log Reg	0.05	0.999	0.904	0.999	0.097	0.069
LDA	0.05	0.992	0.976	0.921	0.024	0.220
QDA	0.05	0.992	0.977	0.870	0.022	0.221
KNN	0.05	0.949	0.976	0.906	0.023	0.222
Ridge	0.05	0.988	0.950	0.957	0.050	0.123
Lasso	0.05	1.000	0.906	1.000	0.095	0.071
RF	0.05	0.983	0.966	0.968	0.034	0.171
SVM - Linear	0.05	0.999	0.893	0.999	0.108	0.063
SVM - Poly	0.05	0.837	0.973	0.572	0.024	0.147
SVM - R	0.05	0.922	0.988	0.662	0.009	0.342

Here are our *test results* at the ideal threshold *for accuracy*.

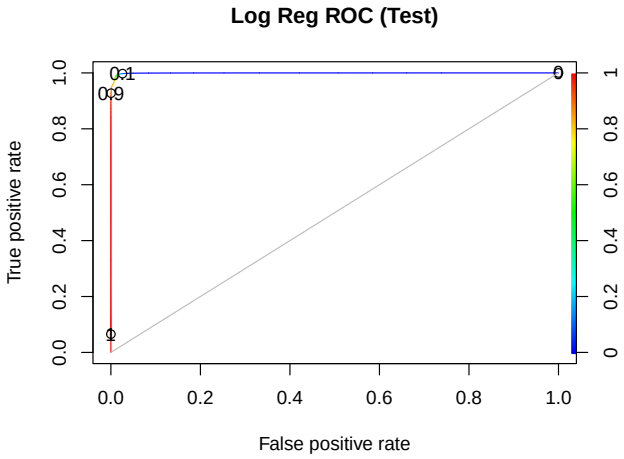
```
test.results.2 <- data.frame(
  results2.test.model,
  results2.test.thresh,
  round(results2.test.auc, 3),
  results2.test.accuracy,
  results2.test.sensitivity,
  results2.test.fpr,
  results2.test.precision
)

colnames(test.results.2) <- c('Model',
                              'Threshold',
                              'AUC Value',
                              'Accuracy',
                              'Sensitivity',
                              'False Positive Rate',
                              'Precision')
```

```
knitr::kable(test.results.2, 'simple')
```

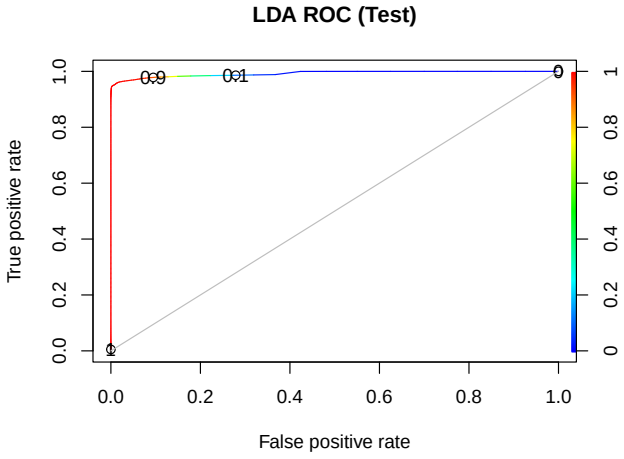
Model	Threshold	AUC Value	Accuracy	Sensitivity	False Positive Rate	Precision
Log Reg	0.35	0.999	0.981	0.991	0.019	0.070
LDA	0.75	0.992	0.983	0.791	0.016	0.270
QDA	0.50	0.992	0.996	0.695	0.002	0.733
KNN	0.50	0.949	0.992	0.816	0.006	0.485
Ridge	0.25	0.988	0.996	0.486	0.000	0.930
Lasso	0.50	1.000	0.996	0.984	0.005	0.613
RF	0.50	0.983	0.995	0.773	0.004	0.608
SVM — Linear	0.40	0.999	0.975	0.990	0.025	0.222
SVM — Poly	0.50	0.837	0.991	0.544	0.006	0.397
SVM — R	0.25	0.922	0.990	0.546	0.007	0.360

Here are our *test ROC curves*. The ROC curves for the KNN and Random Forest models are omitted due to technical issues.

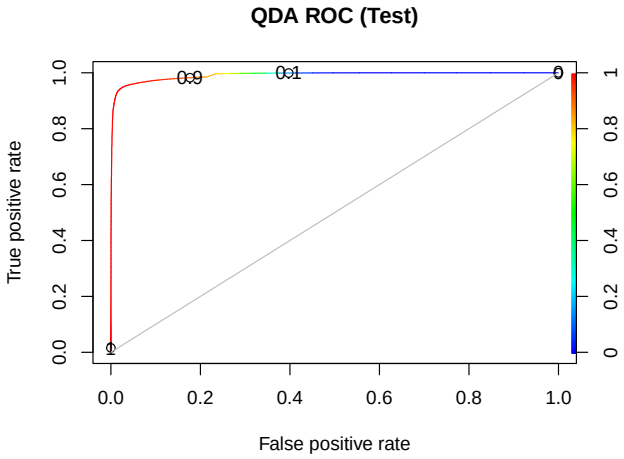
```
draw_test_ROC(logreg.test.pred, 'Log Reg ROC (Test)')
```



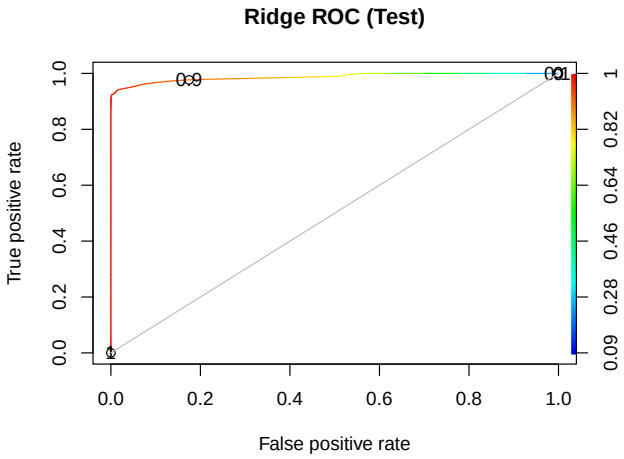

```
draw_test_ROC(lda.test.pred, 'LDA ROC (Test)')
```



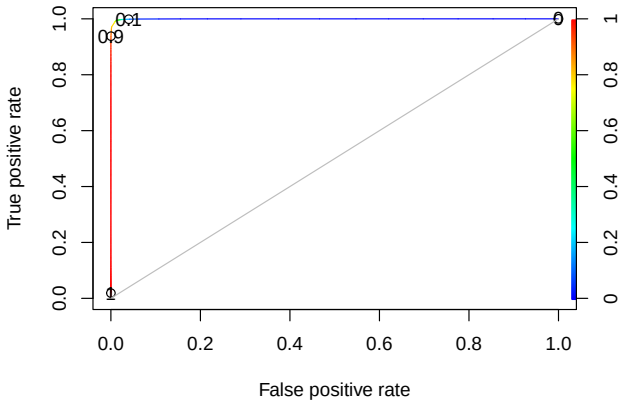

```
draw_test_ROC(qda.test.pred, 'QDA ROC (Test)')
```



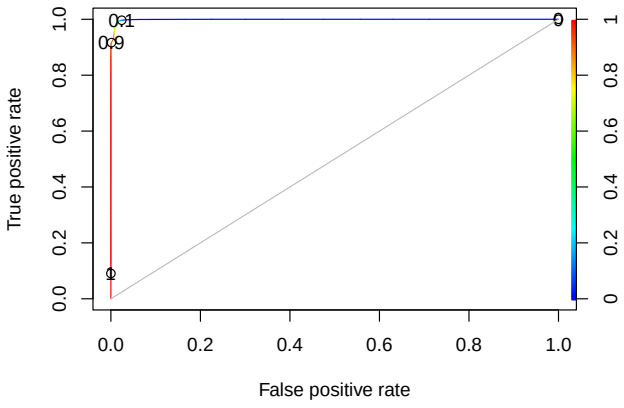

```
draw_test_ROC(ridge.test.pred, 'Ridge ROC (Test)')
```



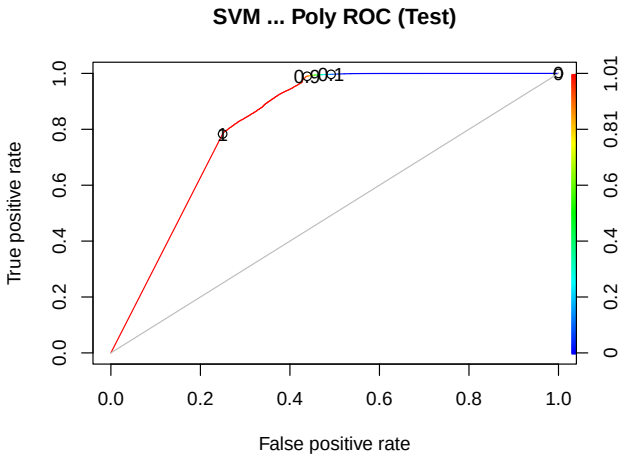

```
draw_test_ROC(lasso.test.pred, 'Lasso ROC (Test)')
```



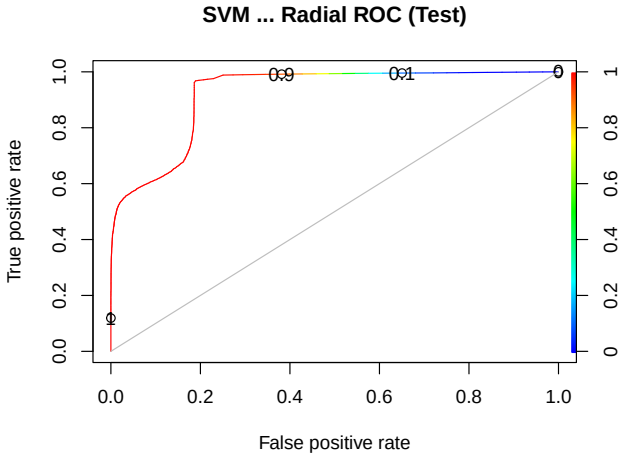

```
draw_test_ROC(svm_linear.test.pred, 'SVM - Linear ROC (Test)')
```




```
draw_test_ROC(svm_poly.test.pred, 'SVM - Poly ROC (Test)')
```




```
draw_test_ROC(svm_radial.test.pred, 'SVM - Radial ROC (Test)')
```



Conclusions

And the winner is... Lasso! This model had the best test performance regarding sensitivity at the 0.05 threshold among all our models, hitting a sensitivity of 1 while actually making predictions (as in, the model didn't just predict everything as one class). It also tied for the best accuracy at our chosen thresholds for accuracy. It performed very well overall on both our metrics, particularly with our main metric in sensitivity, and didn't take long to train or test. It was surprising to see because it had the worst training ROC curve and AUC value among our models. I suspect that our test data isn't well represented by our training data and the Lasso model, being parameterized, had good flexibility when dealing with the test data. My main hangup with this model is the massive discrepancy between training fit and test performance, which gives me pause in applying this model to different data sets. But in my opinion, it's the winner in this context considering both our metrics of interest.

But is there really a winner? Although our Lasso model did well on our chosen metrics, if we had different goals, our target model might change. In this scenario, I prioritized sensitivity with the idea that if our objective is to identify potential survivors through identifying blue tarps, then a false negative could have dire consequences — human suffering or death. But this can be resource intensive. An organization like FEMA might be able to afford these costs, but what about, say, a non-profit organization aiding in relief efforts? Their priorities might be different given their limitations, so they might focus on a model with maximum accuracy to limit waste and ensure the best allocation of their resources, and they would probably want to avoid a model that takes a very long time to train and test.

I also explored thresholds for maximum accuracy because accuracy seems like a better metric for general

comparison between models, especially lacking the context of a particular scenario or problem. Although the Lasso did well in this regard too, I would also consider the Ridge, QDA and Random Forest models. The QDA and Random Forest models showed more consistency between their training fit and test performance, especially with accuracy, so they might instill more faith in different use cases. But really, it comes down to your particular problem, and there are valid reasons to prefer other performance metrics depending on context.

The model that best fit the training data was... Random Forest! With 300 trees in the random forest, each tree only having to make one decision and using only one feature to make that decision, our Random Forest model was very simple due to the data set having only two classes and three features. It had the highest sensitivity and second highest accuracy with our training data, had a relatively high accuracy with test data, but an underwhelming sensitivity with the test data.

The KNN model seemed promising in an earlier version of this project, where we fit some of these models to training data but didn't test them. But after fitting more models, it fell in the rankings considering both training sensitivity and training accuracy. Our SVM models also fit the training data well regarding accuracy, but they were somewhat disappointing with their performance on the test set. Plus, I found other annoying drawbacks with them, which we'll get to later.

The best models in training weren't the best models in testing. Lasso, in my opinion, was our best model with the test data but it seemed like the worst fit among our models with its terribly shaped ROC curve and weak AUC value. It then beat or matched metrics with other models like Logistic Regression, QDA, KNN and the SVM's that looked promising in the training data. Ridge Regression also showed a similar pattern in that it did much better on the test set than its training ROC curve and AUC value would suggest. As mentioned before, I suspect this is because the test data isn't well represented by the training data (we can sort of see this in the EDA section with the IQR differences in the RGB boxplots in the training and test sets), but I expected SVM's in particular to be more flexible and have better accuracy with the test data.

I was pleasantly surprised by how well Logistic Regression did, considering it was the most basic model. Since the test set was so large, I expected the more flexible models to perform better overall, but this wasn't the case. It might be that it's easy to over-tune and over-fit the more flexible models to the training data.

Computation and run time really matter. I've never dealt with data sets this large, so this was my first experience with some long run times in training some of these models, and I even had to limit the ranges of some tuning parameters to ease the computational burden on my machine. The Random Forest model took a while to tune (maybe about 15 minutes), mainly because I looped through different numbers of trees. It performed pretty well on the test set, with relatively high sensitivity and accuracy, though it wasn't the best by either metric.

The SVM models took even longer to train, each probably taking 20+ minutes, and would sometimes fail if I expanded my tuning grid too much. And ultimately, although they fit the training data well (especially regarding accuracy), they didn't beat out other models with their test data performance. The Linear Kernel SVM did well with sensitivity, though, coming in second after the Lasso, but it was also inferior in accuracy compared to the Lasso. The other SVM kernels did poorly regarding sensitivity with the test set but had alright accuracy, though far from our best models. It just doesn't seem to be worth all that time and computational effort for those disappointing results (although, to be fair, we wouldn't have known unless we tested them!).

I think these models seem pretty effective in relief efforts (with some caveats). Now we've seen how our models do with a large test data set with over 2 million observations, I have more faith that these models would be effective in relief efforts. The logic of associating blue tarps with potential survivors is sound and our best models appear effective in identifying them among the other objects in our pictures. But really, how big is 2 million observations? Our data also includes three reference images, two that are 831 pixels by 636 pixels and one that is 965 pixels by 826 pixels. These pictures individually contain 528516 pixels and 797090 pixels, respectively. These three pictures combined contain 1854122 total pixels. Our models are really only identifying individual pixels as blue tarps or non blue tarp objects. Taking a step back, that means our models have been tested on about 3.5 images' worth of pixels — not really that large of a data set in the big

picture. Is this enough? I'm not totally sure, but the test set doesn't seem trivial, we've seen differences in how the models performed in training and testing, and our best models' results were strong, so I think these models would be effective in relief efforts. It certainly beats us humans searching through thousands of images a day.