Disaster Relief Project: Part 2

Mariska Batavia

2022-08-06

Introduction

The purpose of this project was to use aerial photographs of Haiti after the destructive 2010 earthquake to locate blue tarps, which are indicative of displaced persons in need of humanitarian aid. The training data set consisted of 63,241 individual pixels, each with a red, green, and blue color value (all numeric), as well as a class assignment that indicated whether a pixel was a blue tarp, vegetation, soil, rooftop, or something else. Using this data set, I built seven classification models using different methods: logistic regression, linear discriminant analysis, quadratic discriminant analysis, K-nearest neighbor, penalized fit logistic regression (ridge regression), random forest, and the support vector machine. I then tested these seven models on a larger holdout data set of over 2 million individual pixels.

Data Wrangling and Exploratory Data Analysis

My first step was to import the data and convert the "Class" variable to a factor with five levels. I also created a variable called "tarp," which collapses the levels of "Class" into a binary outcome: either the pixel is a blue tarp and has a value of "tarp," or it is any of the other four categories in "Class", and has a value of "other."

```
#import the data
haiti <- read.csv("HaitiPixels.csv", header=TRUE)

#convert class to factor
haiti$Class <- factor(haiti$Class)

#create a binary variable to identify blue tarp (yes/no)
haiti <- haiti%>%
  mutate(tarp=fct_collapse(Class, tarp=c("Blue Tarp"),
                           other=c("Vegetation", "Soil", "Rooftop",
                                   "Various Non-Tarp")))
```

Next, I checked for any pixels with missing values; there were none.

```
#return rows with missing values
haiti[!complete.cases(haiti),]
```

```
## [1] Class Red   Green Blue  tarp
## <0 rows> (or 0-length row.names)
```

I looked at a simple summary of each variable in the data set, as well as a correlation matrix for all three colors, shown below. Blue tarps are the least numerous category, making up only (2022/(61219+2022)) = 3.2% of all pixels. Most pixels are coded as soil or vegetation. All three color values have a maximum of 255 (which is the maximum possible), but the lowest values are 48 (red and green) and 44 (blue). Color values are highly correlated.

```
summary(haiti)
```

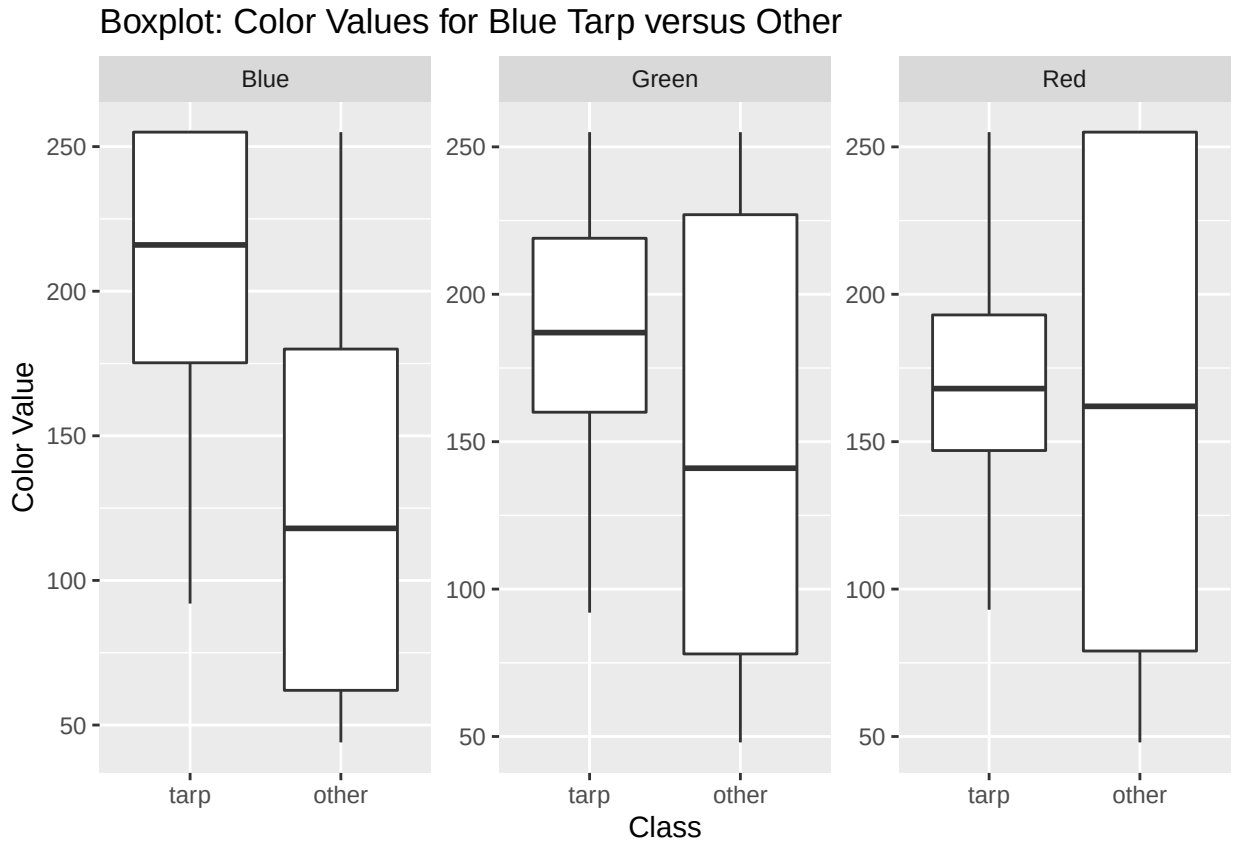
```
##               Class            Red          Green            Blue
##  Blue Tarp       : 2022   Min.   : 48   Min.   : 48.0   Min.   : 44.0
##  Rooftop         : 9903   1st Qu.: 80   1st Qu.: 78.0   1st Qu.: 63.0
##  Soil            :20566   Median :163   Median :148.0   Median :123.0
##  Various Non-Tarp: 4744   Mean   :163   Mean   :153.7   Mean   :125.1
##  Vegetation      :26006   3rd Qu.:255   3rd Qu.:226.0   3rd Qu.:181.0
##                           Max.   :255   Max.   :255.0   Max.   :255.0
##     tarp
##  tarp : 2022
##  other:61219
##
##
##
##
```

```
round((cor(haiti[2:4])), 2)
```

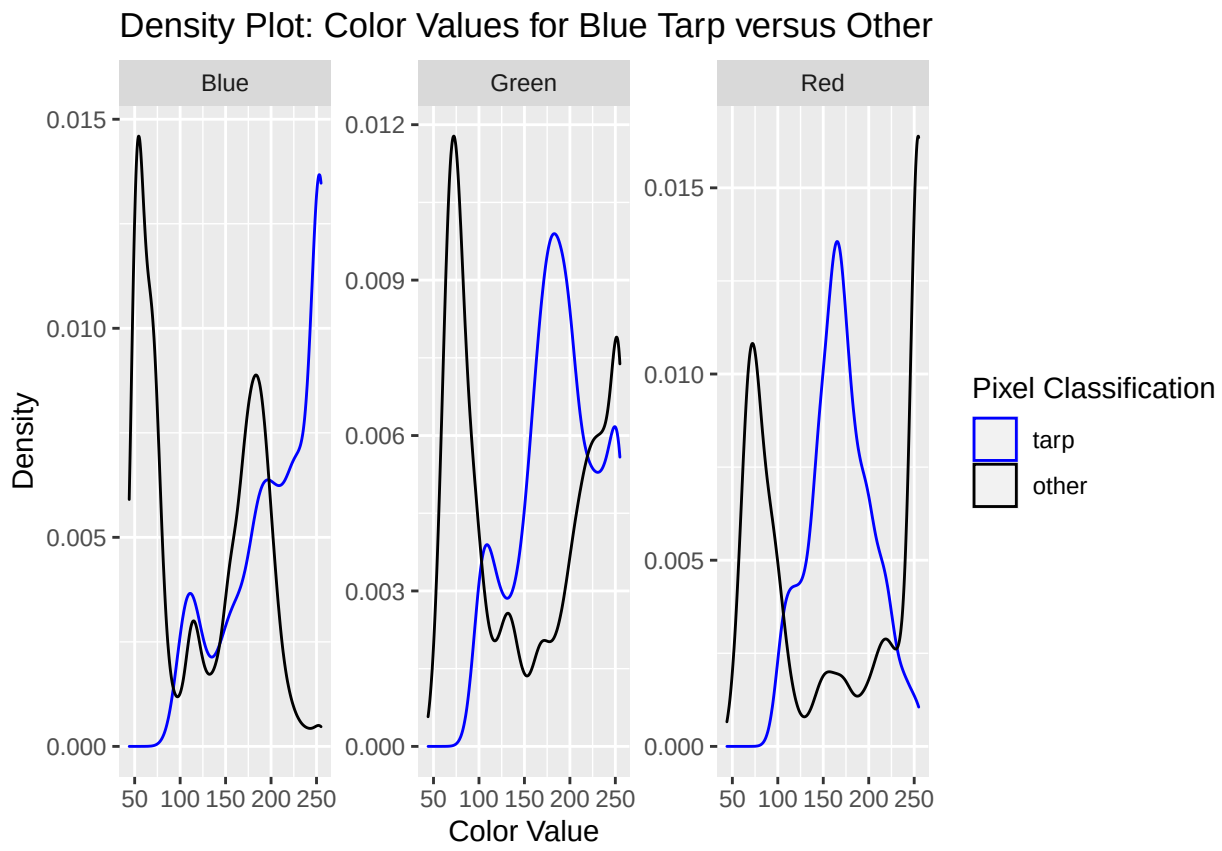
```
##        Red Green Blue
## Red   1.00  0.98 0.94
## Green 0.98  1.00 0.96
## Blue  0.94  0.96 1.00
```

From the boxplots and density plots below, I noted that pixels coded as blue tarps have substantially higher median blue and green color values as compared to pixels coded as other categories. Red color values had similar median values in both classes of pixels, though from the density plot, it appears that there is a bimodal distribution of values for pixels that are not tarps. Indeed, though the green values are lower, on average, in non-tarp pixels, they also show a similar bimodal distribution. For all three colors, the range of values was much narrower in blue tarp pixels than in pixels coded as other categories. This is unsurprising, since non-blue tarp pixels span four different categories (vegetation, soil, rooftops, and various non-tarp).

```
#long pivot for side-by-side plots

long <- haiti[,2:5]%>%
  pivot_longer(cols=-tarp, names_to = "Color", values_to = "Value")

#boxplot with color as facet wrap
ggplot(long, aes(x=tarp, y=Value)) +
  geom_boxplot() +
  facet_wrap(facets='Color', ncol=4, scales = 'free_y')+
  labs(x="Class", y="Color Value",
       title = "Boxplot: Color Values for Blue Tarp versus Other")
```

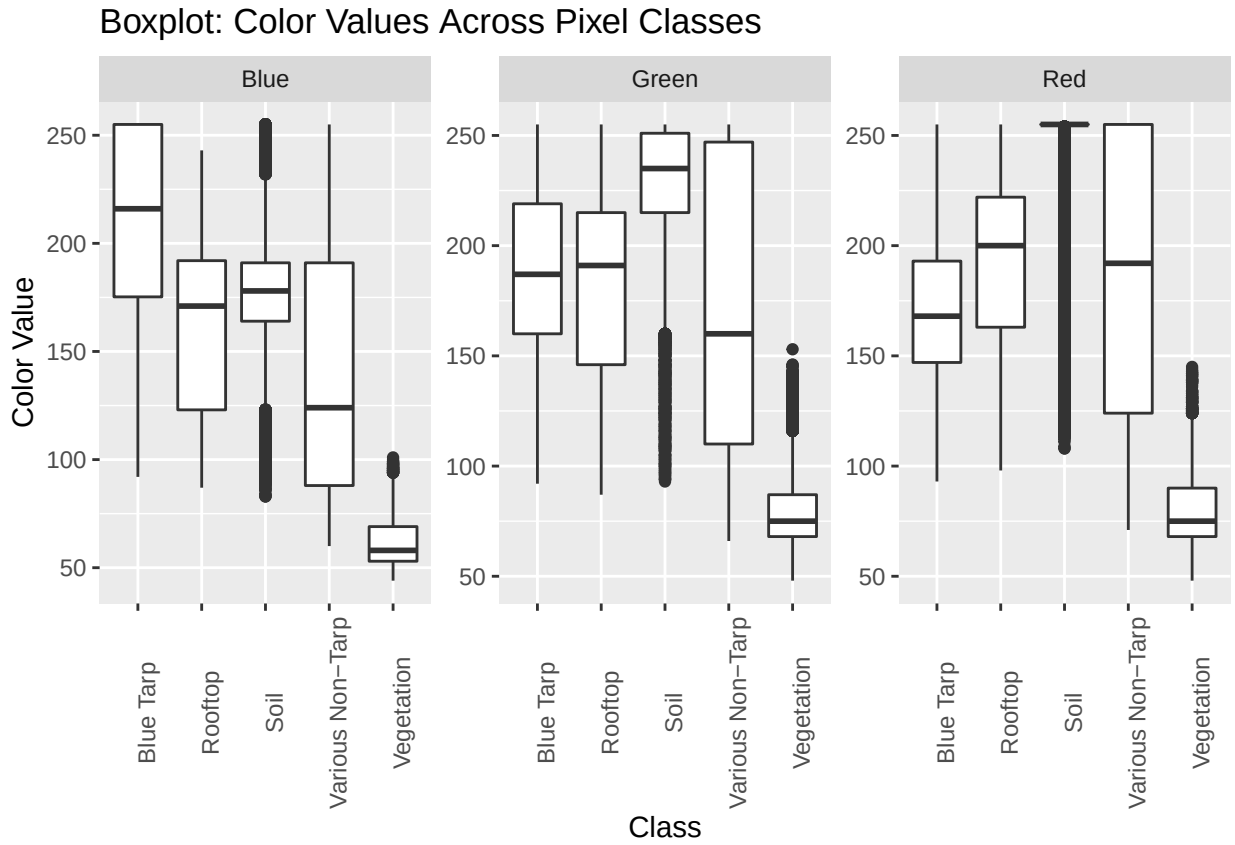



```
#density plots with color as facet wrap
p <- ggplot(long, aes(x=Value, color=tarp)) +
  geom_density() +
  facet_wrap(facets='Color', ncol=4, scales = 'free_y')+
  labs(x="Color Value", y="Density",
       title = "Density Plot: Color Values for Blue Tarp versus Other",
       color="Pixel Classification")

#manually change the color of the lines
p+scale_color_manual(values=c("blue", "black"))
```

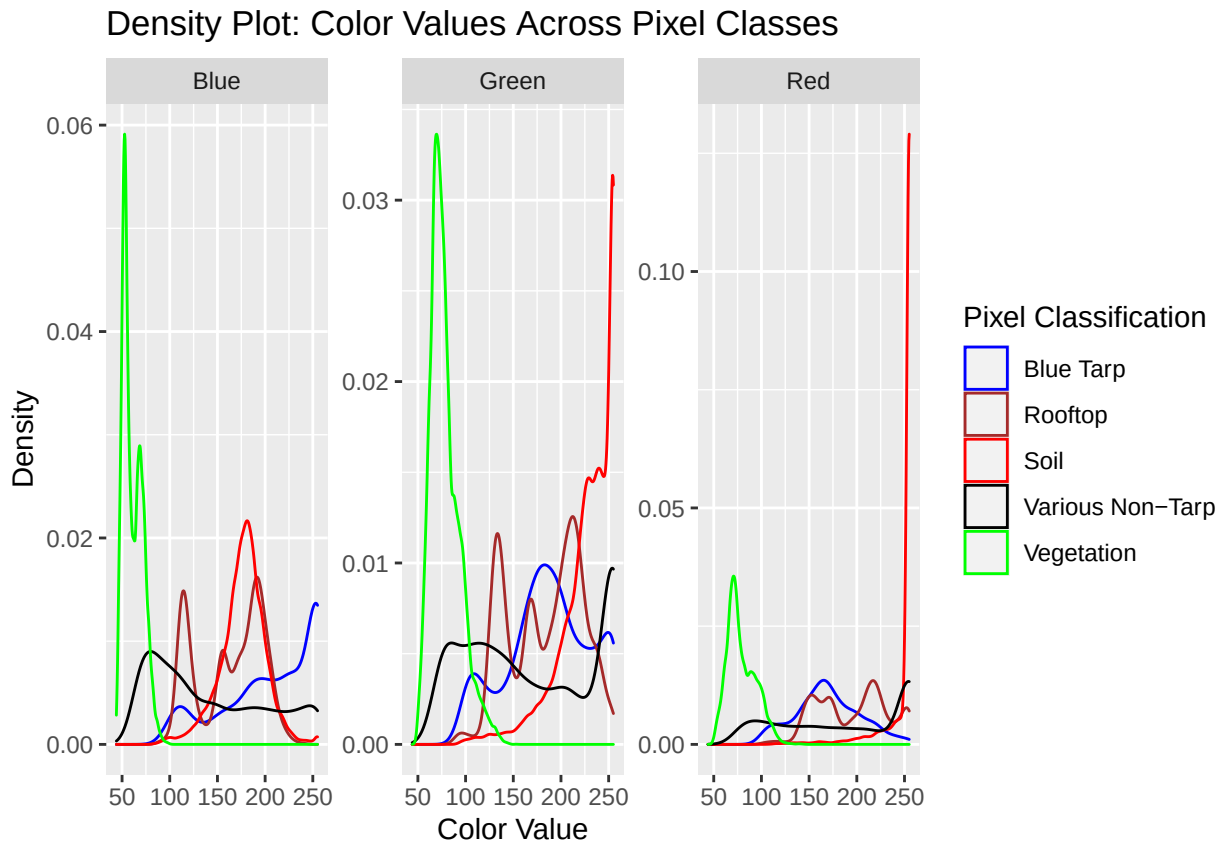


Plotting color values across all five original categories (below), it appears that blue tarp pixels have notably higher blue color values than other categories, but there is substantial overlap with other categories in red and green color values. For instance, rooftops have similar green color values (in terms of both median and IQR) compared to blue tarps, and their density plots greatly overlap. Two other observations worth noting are first, that vegetation appears to have low values for red, green, and blue, suggesting those pixels are darker overall. Second, the density plots show that a majority of soil pixels have very high green and red values.

```
#long pivot for side-by-side plots
long2 <- haiti[,1:4]%>%
  pivot_longer(cols=-Class, names_to = "Color", values_to = "Value")

#ggplot with color as facet wrap
ggplot(long2, aes(x=Class, y=Value)) +
  geom_boxplot() +
  facet_wrap(facets='Color', ncol=4, scales = 'free_y')+
  labs(x="Class", y="Color Value", title = "Boxplot: Color Values Across Pixel Classes")+
  theme(axis.text.x = element_text(angle=90))
```

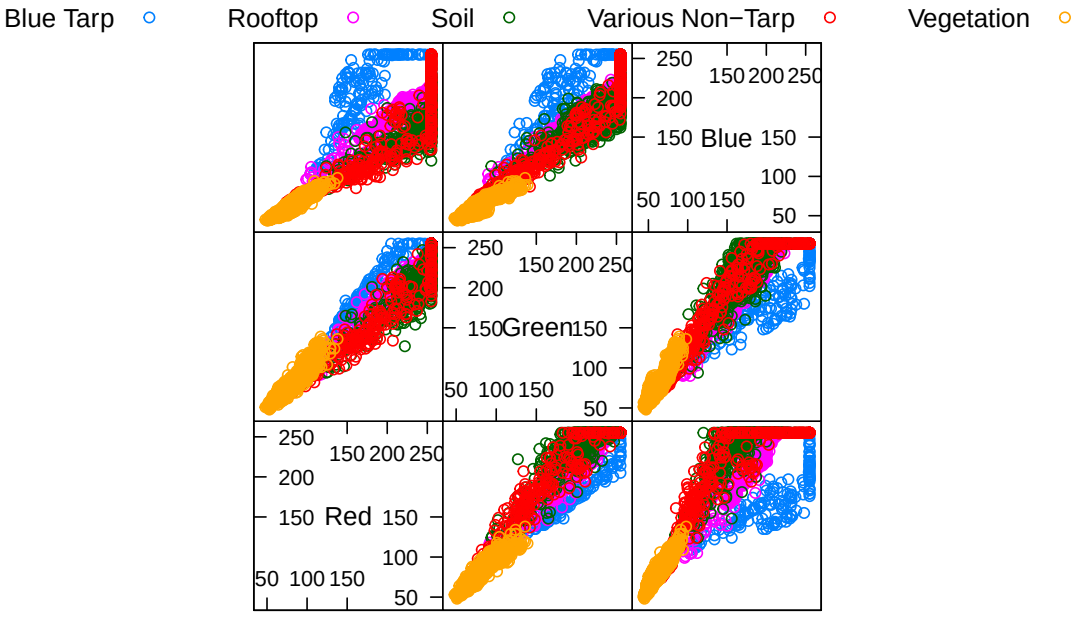



```
#density plots with color as facet wrap
p2 <- ggplot(long2, aes(x=Value, color=Class)) +
  geom_density() +
  facet_wrap(facets='Color', ncol=4, scales = 'free_y')+
  labs(x="Color Value", y="Density",
       title = "Density Plot: Color Values Across Pixel Classes",
       color="Pixel Classification")

#manually change the color of the lines
p2 + scale_color_manual(values=c("blue", "brown", "red", "black", "green"))
```



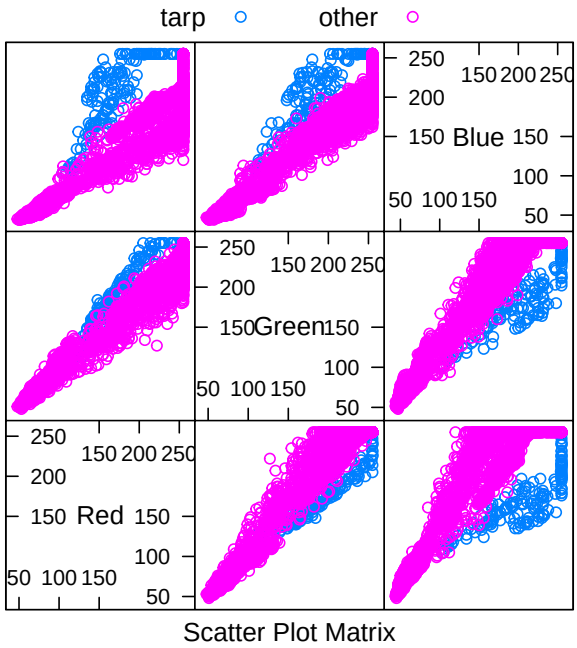
I randomly sampled 5000 observations from the data set and created a scatter plot matrix (below). Looking at this matrix, we can see that all pairs of color values are positively correlated, as noted previously. Pixels with low values for one color tend to have low values for other colors, though in higher value ranges, there is more variation. Many blue tarp pixels stand out in having relatively high blue values for a given green or red value. Separation of other classes is not always as clear. Vegetation (orange on this plot) stands out as having low color values, as noted previously, but rooftop, soil, and various non-tarp categories have considerable overlap in their ratios of red:blue values, green:blue values, and red:green values.

```
#take a sample of 5000 points from the haiti data set
set.seed(1)
sample <- sample(nrow(haiti), 5000)
haiti_sample <- haiti[sample,]

#scatter plot grid with points colored by class
featurePlot(x = haiti_sample[, 2:4],
            y = haiti_sample$Class,
            plot = "pairs",
            auto.key = list(columns = 5))
```



Because some classes have substantial overlap, and because the most important objective is to identify blue tarps from all other points, I generated this same scatter plot matrix using the "tarp" classifier added to the dataframe previously, which collapsed all non-tarp pixels into a single category (shown below).

```
#scatter plot grid with points colored by class
featurePlot(x = haiti_sample[, 2:4],
            y = haiti_sample$tarp,
            plot = "pairs",
            auto.key = list(columns = 2))
```



In the scatter plots above, the decision boundaries between blue tarps and other pixels appear to be approximately linear. Also, the variance in color values and covariance between pairs of colors is not the same across all classes, particularly when non-tarp classes are considered separately (i.e., "vegetation," "soil," etc.)

Preparation for Model Comparison and Analysis

Before starting the model-building process, I set trainControl parameters to be used by all models. These settings establish the use of 10-fold cross validation in creating models.

```
#set trainControl parameters to be used in all models
trControl <- caret::trainControl(method='cv', number=10,
                                 savePredictions=TRUE, classProbs = TRUE,
                                 allowParallel = TRUE)
```

I also established an empty data frame to store key aspects of each model's characteristics and performance.

```
columns <- c("Model", "Model Tuning Parameters", "AUC", "Threshold",
             "Accuracy", "Sensitivity/TPR", "FPR", "Precision")
performance_table <- data.frame(matrix(nrow = 0, ncol = length(columns)))
colnames(performance_table) <- columns
```

Finally, I built a function to help me more efficiently evaluate and compare different models and identify appropriate thresholds. Each time I call the function on one of my models, it will print a ROC curve and a table showing the model's 10-point cross-validated error and accuracy (assuming the default threshold of 0.5) and AUC. Finally, it will test a variety of threshold values so I can see accuracy, sensitivity, the false positive rate, and precision of the model as the threshold changes.

```
#function to evaluate models
evaluate_model <- function(model){

  #CV accuracy with threshold=0.5
  accuracy <- model$results$Accuracy

  #CV prediction error estimate with threshold=0.5
  error <- 1-model$results$Accuracy

  #AUC
  #model$pred['tarp'] and model$pred['obs'] are the predicted prob and observed
  #values for the validation sets.
  #Total number of values in model$pred is the number of observations multiplied
  #by the number of combinations of tuneGrid parameters.
  #So, to generate a ROC or AUC for a specific set of tune parameters,
  #you have to build a new model.
  #Otherwise it will include all suboptimal parameters!
  rates <- prediction(model$pred['tarp'], model$pred['obs'])
  auc_val<- performance(rates, measure="auc")
  auc <- auc_val@y.values[[1]]

  #ROC
  roc_result <- performance(rates, measure="tpr", x.measure="fpr")
  plot(roc_result, main="ROC Curve")
  lines(x=c(0,1), y=c(0,1), col="red")

  #threshold evaluation
  threshold.stats <- thresholder(model, threshold=seq(0.05, 0.5, by=0.05),
                                 statistics="all")
```

```
  threshold.stats$FPR <- 1-threshold.stats$Specificity #add FPR

  table <- threshold.stats %>%
    select("prob_threshold","Accuracy", "Sensitivity", "FPR", "Precision",
           "Detection Prevalence")
  print(table)

  return(tibble(CV_accuracy = accuracy,
                CV_error=error, AUC=auc))

}
```

Logistic Regression

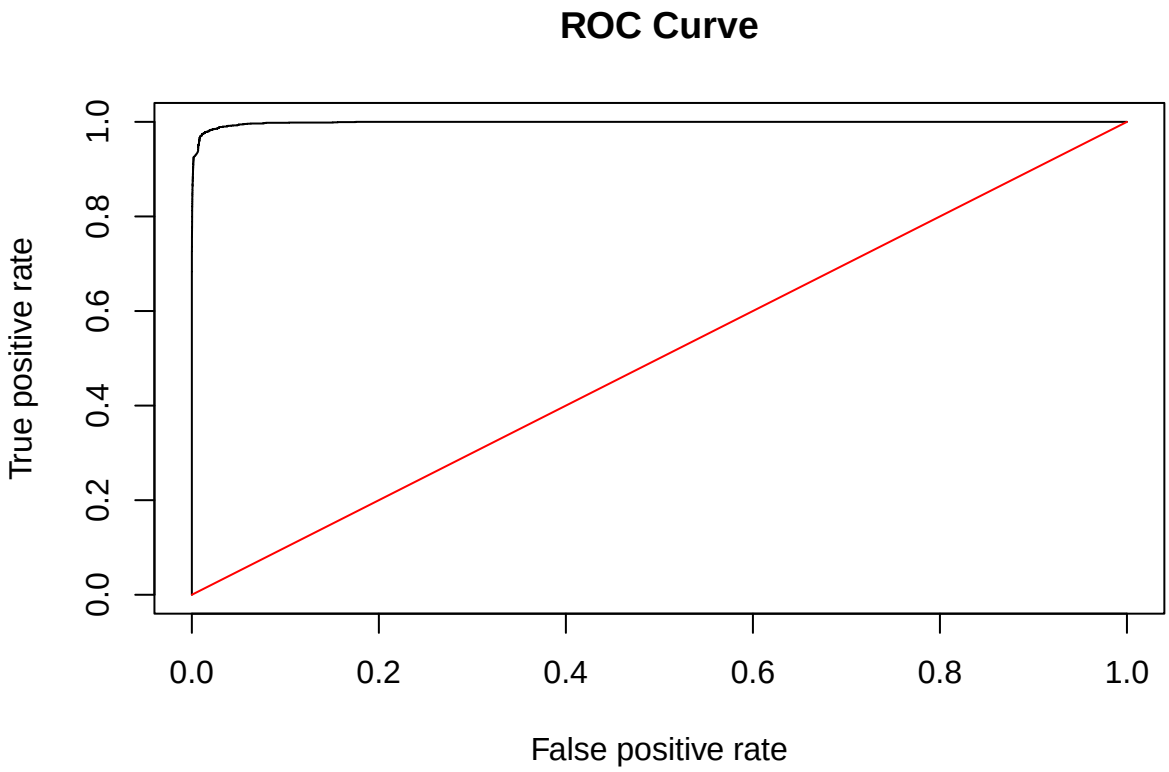
The logistic regression model I built and tested included all three color values as individual predictors, since it appeared from the EDA that all three predictors would be important in separating classes. The summary output (below) showed that all three predictors were highly significant.

```
#logistic regression
set.seed(1)
logistic <- train(tarp~Red+Green+Blue, data=haiti,
                  method ='glm', family='binomial', trControl=trControl)

summary(logistic)
```

```
##
## Call:
## NULL
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -3.7911   0.0000   0.0015   0.0205   3.4266
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) -0.20984    0.18455  -1.137    0.256
## Red          0.26031    0.01262  20.632   <2e-16 ***
## Green        0.21831    0.01330  16.416   <2e-16 ***
## Blue        -0.47241    0.01548 -30.511   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 17901.6  on 63240  degrees of freedom
## Residual deviance:  1769.5  on 63237  degrees of freedom
## AIC: 1777.5
##
## Number of Fisher Scoring iterations: 12
```

As shown in the outputs below, this model had very high prediction accuracy (99.52%), low prediction error (0.48%), and an AUC close to 1 (0.998). The ROC curve confirms that the model overall performs very well, as the curve is far from the diagonal line and close to the upper left corner.

```
evaluate_model(logistic)
```




```
##    prob_threshold  Accuracy Sensitivity         FPR Precision
## 1            0.05 0.9868598   0.9757743 0.012773796 0.7167123
## 2            0.10 0.9908446   0.9663757 0.008347060 0.7931307
## 3            0.15 0.9917142   0.9495586 0.006893276 0.8202407
## 4            0.20 0.9934378   0.9312613 0.004508373 0.8774153
## 5            0.25 0.9957464   0.9188972 0.001715166 0.9469413
## 6            0.30 0.9956357   0.9099985 0.001535483 0.9517632
## 7            0.35 0.9955725   0.9035702 0.001388470 0.9558311
## 8            0.40 0.9954144   0.8956543 0.001290463 0.9583466
## 9            0.45 0.9953986   0.8912110 0.001159784 0.9621785
## 10           0.50 0.9952404   0.8852826 0.001127115 0.9629622
##    Detection Prevalence
## 1            0.04356349
## 2            0.03897787
## 3            0.03703293
## 4            0.03413924
## 5            0.03104001
## 6            0.03058145
## 7            0.03023357
## 8            0.02988569
## 9            0.02961689
## 10           0.02939552
```

```
## # A tibble: 1 x 3
```

```
##   CV_accuracy CV_error   AUC
##         <dbl>    <dbl> <dbl>
## 1       0.995  0.00476 0.998
```

Because the purpose of this model is to find as many people under blue tarps as possible, I determined that misidentifying a pixel as a blue tarp when it was something else would be less consequential than mididentifying a tarp as something else when it is really a blue tarp; that is, in this circumstance, a false negative is much more problematic than a false positive, since false negatives could cost people their lives.

Looking at the table of performance metrics at different threshold values, we can see that at a threshold of P=0.05, model sensitivity is 0.976, which means that 97.6% of blue tarps would be identified, and those people using them could receive aid. The false positive rate at this threshold is at its peak - 0.013 or 1.3%, and precision is at its lowest point - 0.717 or 71.7%. This means that some pixels would be misidentified as tarps when they are not, but again, this is less consequential than missing some tarps, which could cost lives. Finally, even with a threshold set at P=0.05, overall model accuracy is still 98.7% (with a corresponding error of 1.3%), which is only slightly lower than peak model accuracy of 99.5% when the threshold is set at P=0.5.

I tested some additional thresholds below p=0.05 to see if I could make the model even more sensitive without compromising overall accuracy and precision too much. Those results are below. At P=0.005, the sensitivity went to 99.7%, but precision dropped to about 31.9%, which means that the majority of the pixels identified as tarps would actually be something else. Unsurprisingly, detection prevalence also rose considerably, to approximately 10%. If resources and aid workers were in limited supply, I would be concerned about misidentifying too many tarp pixels, as they would detract attention and time from the pixels that were truly tarps. Thus, given that a threshold of P=0.05 correctly captured 97.6% of blue tarps and still maintained a much higher precision, I chose to keep the threshold at P=0.05 for this model.

```
threshold.stats <- thresholder(logistic, threshold=seq(0.005, 0.05, by=0.005),
                               statistics="all")
 threshold.stats$FPR <- 1-threshold.stats$Specificity #add FPR

 table <- threshold.stats %>%
   select("prob_threshold","Accuracy", "Sensitivity", "FPR", "Precision",
          "Detection Prevalence")
 print(table)
```

```
##    prob_threshold  Accuracy Sensitivity        FPR Precision
## 1           0.005 0.9317374   0.9965420 0.07040294 0.3191597
## 2           0.010 0.9581127   0.9920890 0.04300946 0.4337474
## 3           0.015 0.9692447   0.9891235 0.03141178 0.5106952
## 4           0.020 0.9753167   0.9851680 0.02500852 0.5666464
## 5           0.025 0.9788429   0.9831927 0.02130053 0.6049418
## 6           0.030 0.9811041   0.9822026 0.01893199 0.6326836
## 7           0.035 0.9834443   0.9792323 0.01641645 0.6640822
## 8           0.040 0.9849781   0.9782422 0.01479932 0.6865302
## 9           0.045 0.9860218   0.9767619 0.01367222 0.7028119
## 10          0.050 0.9868598   0.9757743 0.01277380 0.7167123
##    Detection Prevalence
## 1            0.10001417
## 2            0.07335424
## 3            0.06203247
## 4            0.05570747
## 5            0.05205478
## 6            0.04973033
## 7            0.04720034
## 8            0.04560330
## 9            0.04446481
```

```
## 10           0.04356349
```

Key model characteristics and metrics were saved for reference later.

```
#insert all key metrics into performance data frame

logisticinfo <- data.frame("Logistic regression", "predictors=3",
                           0.998, 0.05, 0.987, 0.976, 0.013, 0.717)
colnames(logisticinfo) <- columns
performance_table <- rbind(performance_table, logisticinfo)
```

Linear Discriminant Analysis

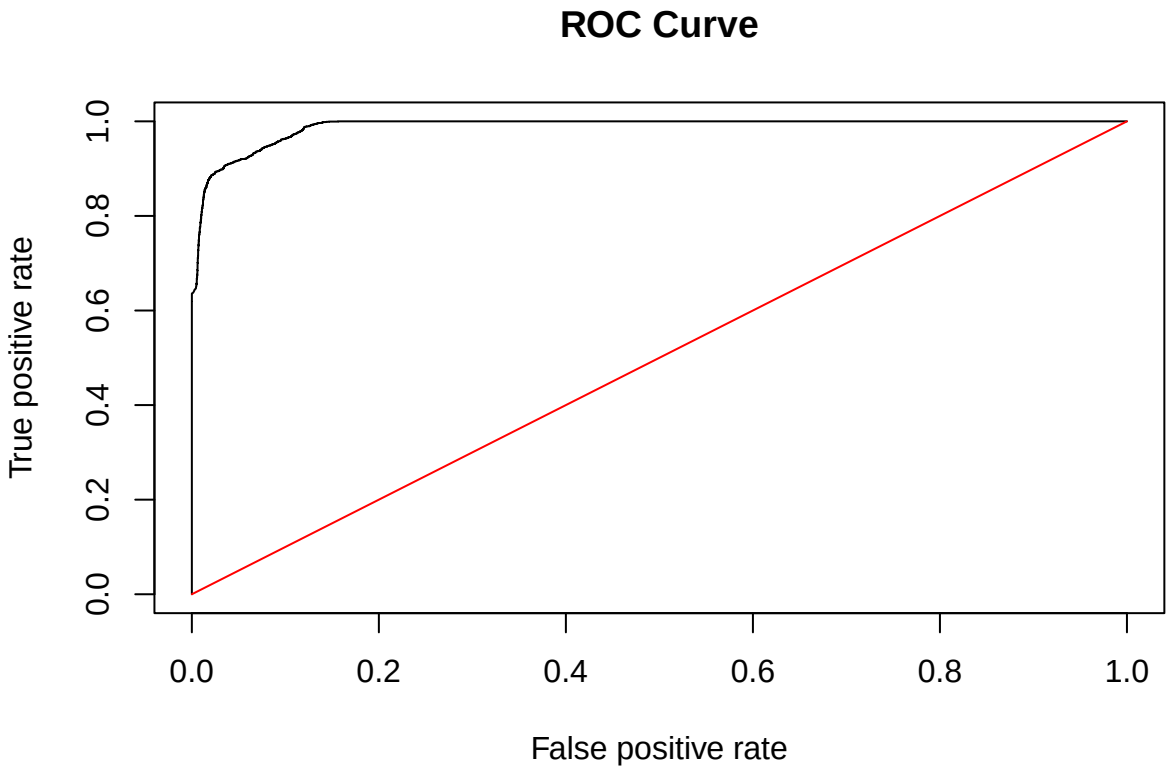
The LDA model I built and tested included all three color values as individual predictors, since it appeared from the EDA that all three predictors would be important in separating classes.

```
#LDA
set.seed(1)
lda <- train(tarp~Red+Green+Blue, data=haiti,
                  method ='lda', trControl=trControl)

lda$finalModel
```

```
## Call:
## lda(x, grouping = y)
##
## Prior probabilities of groups:
##       tarp      other
## 0.03197293 0.96802707
##
## Group means:
##            Red    Green     Blue
## tarp  169.6627 186.4149 205.0371
## other 162.7604 152.5808 122.4993
##
## Coefficients of linear discriminants:
##               LD1
## Red    0.02896984
## Green  0.02328544
## Blue  -0.06923974
```

As shown in the outputs below, this model had very high prediction accuracy (98.39%), low prediction error (1.61%), and an AUC close to 1 (0.989), though all three of these values are slightly lower than the corresponding values seen for the logistic regression. The ROC curve confirms that the model overall performs very well, though consistent with other slightly worse performance metrics, it cuts away from the upper left corner as compared to the ROC curve for the logistic regression.

```
evaluate_model(lda)
```




```
##    prob_threshold  Accuracy Sensitivity        FPR Precision
## 1            0.05 0.9810408   0.8600522 0.01496261 0.6561506
## 2            0.10 0.9824640   0.8476881 0.01308412 0.6822231
## 3            0.15 0.9825589   0.8407575 0.01275742 0.6859436
## 4            0.20 0.9827170   0.8348169 0.01239806 0.6906078
## 5            0.25 0.9828435   0.8244403 0.01192435 0.6962166
## 6            0.30 0.9832072   0.8199873 0.01140163 0.7046099
## 7            0.35 0.9836183   0.8145442 0.01079725 0.7143653
## 8            0.40 0.9834760   0.8096010 0.01078091 0.7133988
## 9            0.45 0.9837290   0.8061381 0.01040522 0.7196920
## 10           0.50 0.9839187   0.8016851 0.01006219 0.7254389
##    Detection Prevalence
## 1            0.04198223
## 2            0.03976848
## 3            0.03923084
## 4            0.03869320
## 5            0.03790259
## 6            0.03725427
## 7            0.03649526
## 8            0.03632133
## 9            0.03584697
## 10           0.03537259
```

```
## # A tibble: 1 x 3
##   CV_accuracy CV_error   AUC
##         <dbl>    <dbl> <dbl>
## 1       0.984   0.0161 0.989
```

When one class is rare compared to the other, the accuracy can be a misleading indicator of overall model performance, and in a closer examination of metrics across different thresholds, it is clear that this model's performance is substantially worse than the logistic regression. As before, if we want to prioritize sensitivity to help as many people as possible, we can only achieve a maximum sensitivity of 0.860 at P=0.05 with this model; this means that we would only be expected to detect about 86% of all tarp pixels. As before, I tried some additional, lower thresholds to see if I could improve the sensitivity (shown below).

Even with the threshold lowered to P=0.001, the sensitivity only reached 93.6%, and at this threshold, precision was only 30.8%, which means that a majority of pixels identified as tarps would actually be something else. I ultimately selected a threshold of P=0.005; this yielded a sensitivity of 89.4%, but also a precision of 51.7%. Setting the threshold this low would mean that the detection prevalence is about 5.54%, which is higher than the detection prevalence for the threshold I selected for the logistic regression above (about 4.36%). If resources and personnel were readily available to invesitgate all identified tarp pixels, I would probably choose to lower this threshold further and increase sensitivity, with the recognition that this would dramatically increase the number of pixels identified (many erroneously) as tarps.

```
threshold.stats <- thresholder(lda, threshold=seq(0.001, 0.01, by=0.001),
                               statistics="all")
  threshold.stats$FPR <- 1-threshold.stats$Specificity #add FPR

  table <- threshold.stats %>%
    select("prob_threshold","Accuracy", "Sensitivity", "FPR", "Precision",
           "Detection Prevalence")
  print(table)
```

```
##    prob_threshold  Accuracy Sensitivity        FPR Precision
## 1           0.001 0.9307253   0.9362118 0.06945551 0.3083639
## 2           0.002 0.9540330   0.9144589 0.04465930 0.4043272
## 3           0.003 0.9631884   0.9050554 0.03489110 0.4623052
## 4           0.004 0.9673471   0.8971419 0.03033369 0.4949991
## 5           0.005 0.9697981   0.8941716 0.02770379 0.5168592
## 6           0.006 0.9722174   0.8916988 0.02512290 0.5405806
## 7           0.007 0.9733717   0.8887309 0.02383243 0.5531910
## 8           0.008 0.9745419   0.8872506 0.02257464 0.5660072
## 9           0.009 0.9750637   0.8862630 0.02200293 0.5717898
## 10          0.010 0.9760599   0.8857679 0.02095749 0.5839245
##    Detection Prevalence
## 1            0.09716803
## 2            0.07246882
## 3            0.06271252
## 4            0.05804781
## 5            0.05540711
## 6            0.05282967
## 7            0.05148561
## 8            0.05022058
## 9            0.04963552
## 10           0.04860770
```

I saved key metrics in my data frame for later use.

```
#insert all key metrics into performance data frame

ldainfo <- data.frame("LDA", "predictors=3",
                      0.989, 0.005, 0.970, 0.894, 0.028, 0.517)
colnames(ldainfo) <- columns
performance_table <- rbind(performance_table, ldainfo)
```

Quadratic Discriminant Analysis

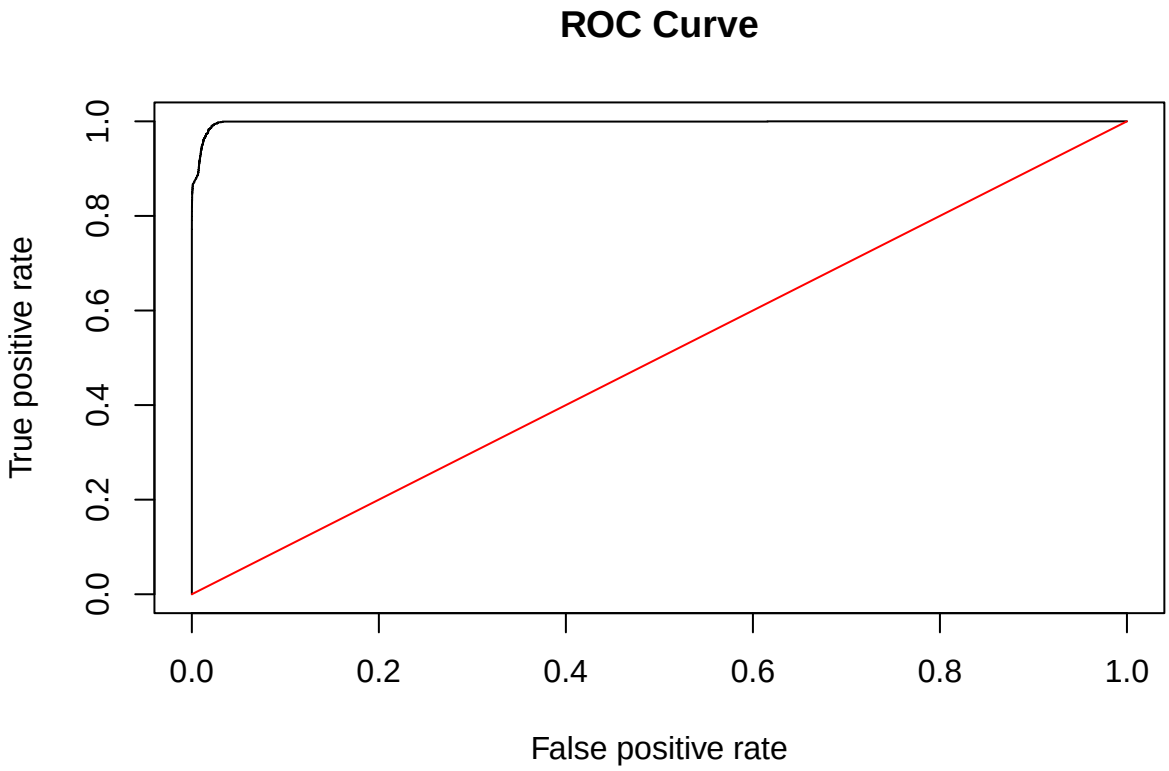
The QDA model I built and tested included all three color values as individual predictors, since it appeared from the EDA that all three predictors would be important in separating classes.

```
#QDA
set.seed(1)
qda <- train(tarp~Red+Green+Blue, data=haiti,
                  method ='qda', trControl=trControl)

qda$finalModel
```

```
## Call:
## qda(x, grouping = y)
##
## Prior probabilities of groups:
##       tarp      other
## 0.03197293 0.96802707
##
## Group means:
##            Red    Green     Blue
## tarp  169.6627 186.4149 205.0371
## other 162.7604 152.5808 122.4993
```

As shown in the outputs below, this model had very high prediction accuracy (99.46%), low prediction error (0.54%), and an AUC close to 1 (0.998). This model outperformed the LDA model and was comparable to the logistic regression model. The ROC curve confirms that the model overall performs very well, as the curve is far from the diagonal line and close to the upper left corner.

```
evaluate_model(qda)
```




```
##    prob_threshold  Accuracy Sensitivity          FPR Precision
## 1            0.05 0.9882355   0.9401673 0.0101765597 0.7533461
## 2            0.10 0.9896744   0.9010998 0.0073996360 0.8017922
## 3            0.15 0.9903228   0.8862581 0.0062398763 0.8249355
## 4            0.20 0.9941494   0.8684632 0.0016988046 0.9469747
## 5            0.25 0.9947503   0.8620348 0.0008657408 0.9705065
## 6            0.30 0.9947186   0.8560991 0.0007023955 0.9758192
## 7            0.35 0.9947186   0.8521412 0.0005717193 0.9801290
## 8            0.40 0.9947503   0.8481856 0.0004083686 0.9856167
## 9            0.45 0.9947345   0.8452178 0.0003266960 0.9883892
## 10           0.50 0.9945605   0.8392845 0.0003103614 0.9888861
##    Detection Prevalence
## 1            0.03991081
## 2            0.03597350
## 3            0.03437643
## 4            0.02941132
## 5            0.02839931
## 6            0.02805144
## 7            0.02779844
## 8            0.02751381
## 9            0.02733987
## 10           0.02713431
```

```
## # A tibble: 1 x 3
##   CV_accuracy CV_error   AUC
##         <dbl>    <dbl> <dbl>
## 1       0.995  0.00544 0.998
```

Looking through the list of possible thresholds, P=0.05 showed the highest sensitivity (94.01%) and still had a very low false positive rate (1.02%) and high precision (75.33%). I tried a few more (lower) threshold values to see if I could improve sensitivity further without compromising the precision too much. This output is shown below. As has been the case in other models, there is a clear tradeoff between increasing sensitivity and increasing precision. I selected a threshold of P=0.025, which gave a sensitivity of 98.37%, a precision of 64.15%, and a detection prevalence of 4.91%. As was true with the logistic model, this model has the potential to have a very high sensitivity if the threshold is lowered enough, and this model's precision does not suffer quite as much at very low thresholds, compared to the logistic regression.

```
threshold.stats <- thresholder(qda, threshold=seq(0.005, 0.05, by=0.005),
                               statistics="all")
  threshold.stats$FPR <- 1-threshold.stats$Specificity #add FPR

  table <- threshold.stats %>%
    select("prob_threshold","Accuracy", "Sensitivity", "FPR", "Precision",
           "Detection Prevalence")
  print(table)
```

```
##    prob_threshold  Accuracy Sensitivity        FPR Precision
## 1           0.005 0.9564839   0.9995050 0.04493697 0.4239463
## 2           0.010 0.9670783   0.9995050 0.03399265 0.4934758
## 3           0.015 0.9725969   0.9980247 0.02824280 0.5394159
## 4           0.020 0.9775620   0.9911013 0.02288500 0.5895956
## 5           0.025 0.9818472   0.9836853 0.01821326 0.6415486
## 6           0.030 0.9843140   0.9723041 0.01528933 0.6780956
## 7           0.035 0.9858162   0.9648905 0.01349252 0.7030287
## 8           0.040 0.9869231   0.9550017 0.01202239 0.7244523
## 9           0.045 0.9879983   0.9475857 0.01066660 0.7461725
## 10          0.050 0.9882355   0.9401673 0.01017656 0.7533461
##    Detection Prevalence
## 1            0.07545737
## 2            0.06486296
## 3            0.05924951
## 4            0.05384163
## 5            0.04908206
## 6            0.04588792
## 7            0.04391137
## 8            0.04217200
## 9            0.04062238
## 10           0.03991081
```

I saved key metrics in my data frame for later use.

```
#insert all key metrics into performance data frame

qdainfo <- data.frame("QDA", "predictors=3",
                      0.998, 0.025, 0.982,  0.984,  0.018,  0.642)
colnames(qdainfo) <- columns
performance_table <- rbind(performance_table, qdainfo)
```

K-Nearest Neighbor

For K-nearest neighbor, I again used all three predictors. I used carat's train function to test values of k between 3 and 15. The value of k that minimized cross validation error was k=7.

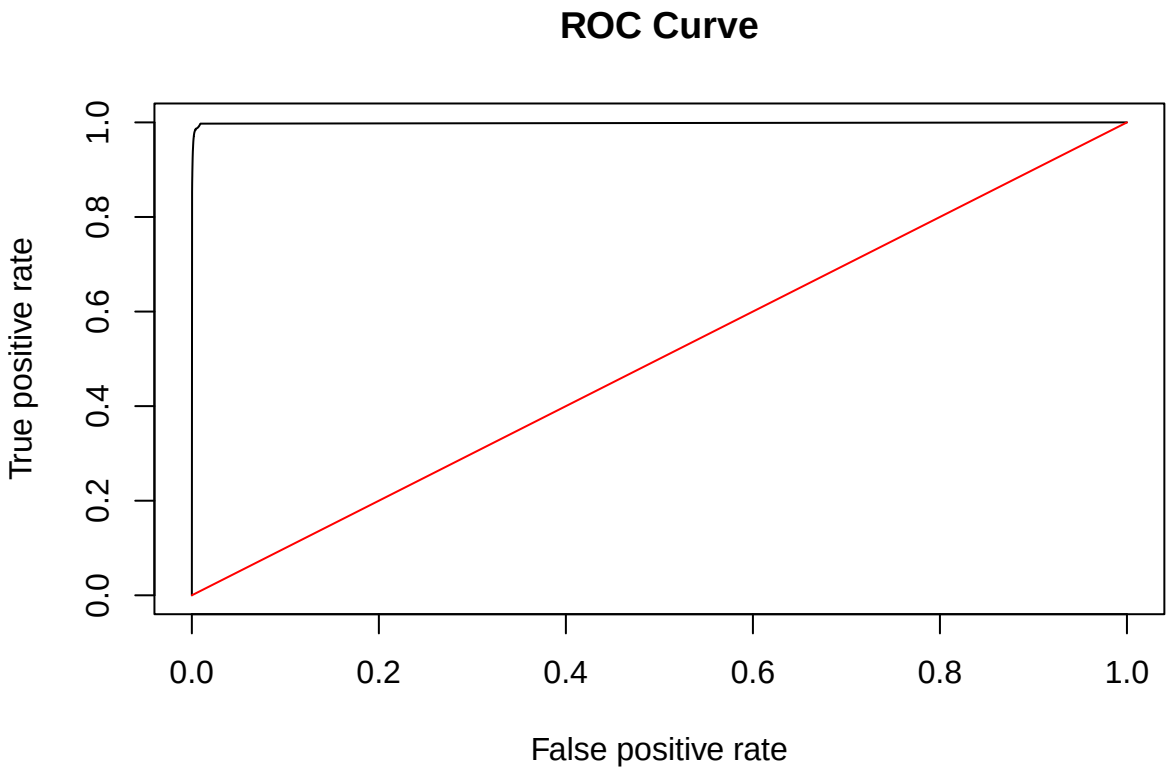
```
#determine number of cores and initialize cluster
no_cores <- detectCores()-1
cl <- makePSOCKcluster(no_cores)
registerDoParallel(cl)

set.seed(1)

#define range of k to explore
kvals <- seq(3, 15, by=2)

#set tuneGrid
tuneGrid <- expand.grid(k=kvals)

knn <- train(tarp~Red+Green+Blue, data=haiti, method ='knn',
             preProcess = c("center","scale"), #data must be normalized
             trControl=trControl, tuneGrid=tuneGrid)

knn
```

```
## k-Nearest Neighbors
##
## 63241 samples
##     3 predictor
##     2 classes: 'tarp', 'other'
##
## Pre-processing: centered (3), scaled (3)
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 56918, 56917, 56917, 56917, 56916, 56917, ...
## Resampling results across tuning parameters:
##
##   k   Accuracy   Kappa
##    3  0.9971854  0.9544871
##    5  0.9972328  0.9553425
##    7  0.9972803  0.9562512
##    9  0.9972328  0.9554283
##   11  0.9970589  0.9526162
##   13  0.9970905  0.9530855
##   15  0.9970747  0.9528944
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was k = 7.
```

```
#stop cluster
stopCluster(cl)
registerDoSEQ()
```

The benefit to using carat for the knn model is that it returns probabilities of being a tarp for each data point, and therefore I was able to generate a ROC curve and a corresponding AUC value. However, in evaluating my KNN model with my evaluate_model function (below), I realized that the AUC value was the same for all values of k, which made me suspect that while CV accuracy and CV error were specific to a certain value of k, the ROC curve and corresponding AUC value was an average over all values of k in the tuning grid.

```
evaluate_model(knn)
```




```
##    prob_threshold  Accuracy Sensitivity         FPR Precision
## 1            0.05 0.9912873   0.9980247 0.008935130 0.7872226
## 2            0.10 0.9955092   0.9856631 0.004165386 0.8867135
## 3            0.15 0.9964106   0.9812125 0.003087274 0.9131995
## 4            0.20 0.9964106   0.9812125 0.003087274 0.9131995
## 5            0.25 0.9964422   0.9802248 0.003021936 0.9147948
## 6            0.30 0.9970589   0.9742867 0.002188862 0.9365364
## 7            0.35 0.9970589   0.9742867 0.002188862 0.9365364
## 8            0.40 0.9970747   0.9742867 0.002172527 0.9369567
## 9            0.45 0.9973119   0.9643955 0.001600811 0.9522508
## 10           0.50 0.9972803   0.9629127 0.001584476 0.9526353
##    Detection Prevalence
## 1            0.04055912
## 2            0.03554657
## 3            0.03436062
## 4            0.03436062
## 5            0.03426574
## 6            0.03326955
## 7            0.03326955
## 8            0.03325374
## 9            0.03238404
## 10           0.03232080
```

```
## # A tibble: 7 x 3
```

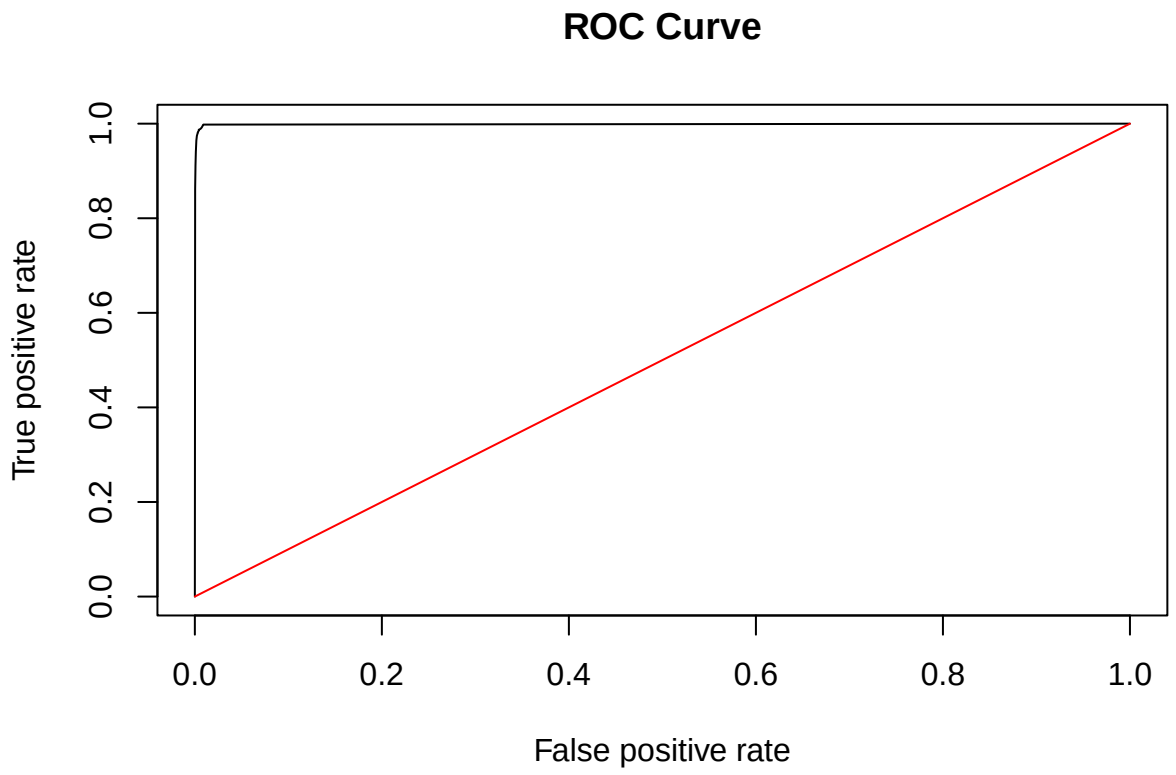
```
##   CV_accuracy CV_error   AUC
##         <dbl>    <dbl> <dbl>
## 1       0.997  0.00281 0.998
## 2       0.997  0.00277 0.998
## 3       0.997  0.00272 0.998
## 4       0.997  0.00277 0.998
## 5       0.997  0.00294 0.998
## 6       0.997  0.00291 0.998
## 7       0.997  0.00293 0.998
```

Therefore To generate a ROC curve and AUC unique to the specific selected value of k, I rebuilt the model with k=7, to see a ROC curve and AUC specific to this value.

```
set.seed(1)

#set tuneGrid
tuneGrid <- expand.grid(k=7)

knn_final <- train(tarp~Red+Green+Blue, data=haiti, method ='knn',
             preProcess = c("center","scale"), #data must be normalized
             trControl=trControl, tuneGrid=tuneGrid)
```

As shown in the outputs below, for k=7, this model had very high prediction accuracy (99.73%), low prediction error (0.27%), and an AUC close to 1 (0.999). The ROC curve confirms that the model overall performs very well, as the curve is far from the diagonal line and close to the upper left corner. As compared to the other two high performing models (logistic regression and QDA), this model is even better.

```
evaluate_model(knn_final)
```



```
##    prob_threshold  Accuracy Sensitivity         FPR Precision
## 1            0.05 0.9912873   0.9980247 0.008935130 0.7872226
## 2            0.10 0.9955092   0.9856631 0.004165386 0.8867135
## 3            0.15 0.9964106   0.9812125 0.003087274 0.9131995
## 4            0.20 0.9964106   0.9812125 0.003087274 0.9131995
## 5            0.25 0.9964422   0.9802248 0.003021936 0.9147948
## 6            0.30 0.9970589   0.9742867 0.002188862 0.9365364
## 7            0.35 0.9970589   0.9742867 0.002188862 0.9365364
## 8            0.40 0.9970747   0.9742867 0.002172527 0.9369567
## 9            0.45 0.9973119   0.9643955 0.001600811 0.9522508
## 10           0.50 0.9972803   0.9629127 0.001584476 0.9526353
##    Detection Prevalence
## 1            0.04055912
## 2            0.03554657
## 3            0.03436062
## 4            0.03436062
## 5            0.03426574
## 6            0.03326955
## 7            0.03326955
## 8            0.03325374
## 9            0.03238404
## 10           0.03232080
```

```
## # A tibble: 1 x 3
##   CV_accuracy CV_error   AUC
##         <dbl>    <dbl> <dbl>
## 1       0.997  0.00269 0.999
```

Looking at the various probability thresholds above, at P=0.05, the sensitivity is 99.80%, with a corresponding precision that is also quite good compared to other models, at 78.72%. This indicates that this model would identify almost all blue tarps, and that more than three quarters of pixels identified as tarps would actually be tarps. In a situation where aid workers and/or resources were limited, decreasing the threshold to 0.10 would still give a sensitivity of 98.57%, the precision would be 88.67%, and the detection rpevalence would be only 3.55%. Because I would expect resources to be constrained in a diaster situation, I selected 0.10 as the threshold, and saved key metrics in my data frame.

```
#insert all key metrics into performance data frame

knninfo <- data.frame("KNN", "predictors=3; k=7",
                      0.998, 0.10, 0.996,   0.986,  0.004,  0.887)
colnames(knninfo) <- columns
performance_table <- rbind(performance_table, knninfo)
```

Penalized Logistic Regression (Ridge Regression)

For the penalized logistic regression model, I chose ridge regression, given that there were few predictors to begin with, and all showed a strong relationship with pixel identification. I fit the model using all three color values as predictors, as in other models. I set a range of λ values to try, and the model selected $\lambda = 0.003981072$. It was noteworthy that the accuracy for all values below $\lambda = 0.01$ was 97.79%, and accuracy was slightly lower (96.80%) for all λ values 0.01 or greater. The higher accuracy for all values very close to 0 suggests to me that a shrinkage penalty may not substantially improve predictive ability beyond the regular logistic regression (where $\lambda = 0$).

```
set.seed(1)

#define range of lambdas to explore
```

```
lambdas <- 10^seq(1, -3, by=-0.2)

#set tuneGrid
tuneGrid <- expand.grid(alpha=0, lambda=lambdas)

#build ridge regression model
ridge <- train(tarp~Red+Green+Blue, data=haiti, method ='glmnet',
               trControl=trControl, tuneGrid=tuneGrid)

ridge
```

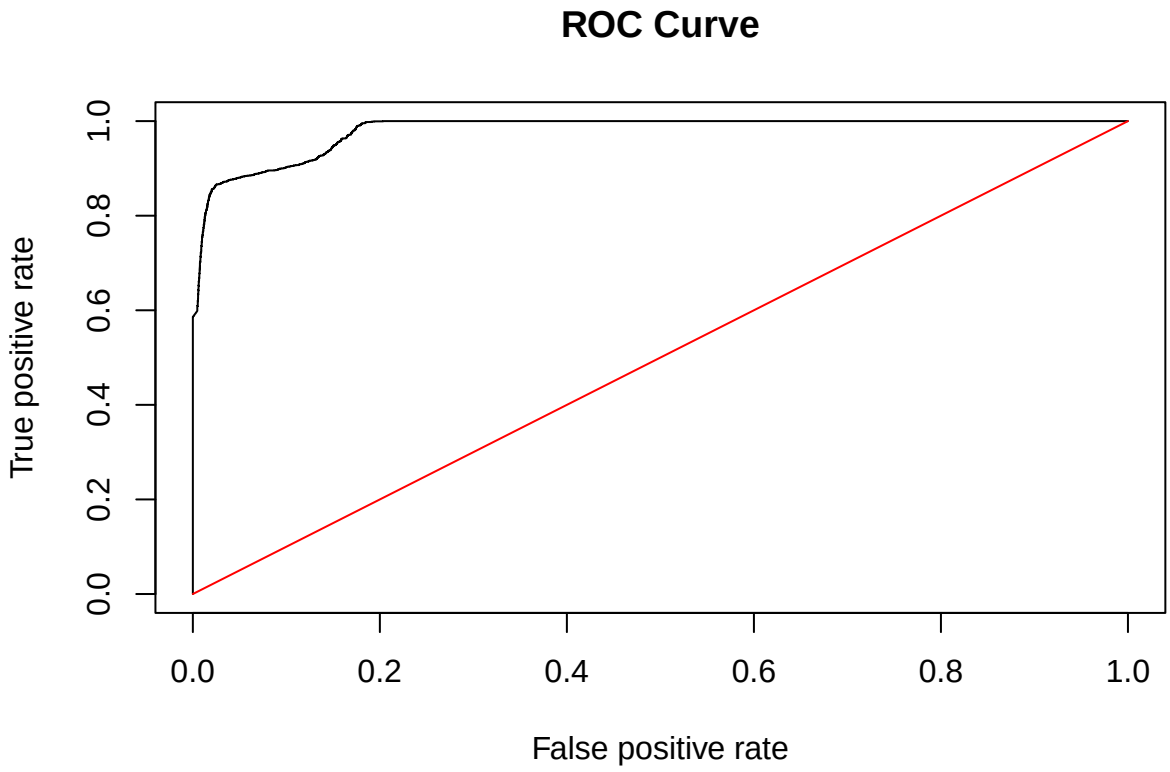
```
## glmnet
##
## 63241 samples
##     3 predictor
##     2 classes: 'tarp', 'other'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 56918, 56917, 56917, 56917, 56916, 56917, ...
## Resampling results across tuning parameters:
##
##   lambda        Accuracy   Kappa
##    0.001000000  0.9778783  0.4624899
##    0.001584893  0.9778783  0.4624899
##    0.002511886  0.9778783  0.4624899
##    0.003981072  0.9778783  0.4624899
##    0.006309573  0.9708575  0.1581568
##    0.010000000  0.9680271  0.0000000
##    0.015848932  0.9680271  0.0000000
##    0.025118864  0.9680271  0.0000000
##    0.039810717  0.9680271  0.0000000
##    0.063095734  0.9680271  0.0000000
##    0.100000000  0.9680271  0.0000000
##    0.158489319  0.9680271  0.0000000
##    0.251188643  0.9680271  0.0000000
##    0.398107171  0.9680271  0.0000000
##    0.630957344  0.9680271  0.0000000
##    1.000000000  0.9680271  0.0000000
##    1.584893192  0.9680271  0.0000000
##    2.511886432  0.9680271  0.0000000
##    3.981071706  0.9680271  0.0000000
##    6.309573445  0.9680271  0.0000000
##   10.000000000  0.9680271  0.0000000
##
## Tuning parameter 'alpha' was held constant at a value of 0
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were alpha = 0 and lambda = 0.003981072.
```

As with the KNN model, since I wanted to see a ROC curve and AUC corresponding to only the best value of λ, I rebuilt this model using only the optimized value.

```
#set tuneGrid
tuneGrid <- expand.grid(alpha=0, lambda=0.003981072)
```

```r
#build ridge regression model
ridge_final <- train(tarp~Red+Green+Blue, data=haiti, method ='glmnet',
             trControl=trControl, tuneGrid=tuneGrid)
```

```r
evaluate_model(ridge_final)
```




```
##    prob_threshold  Accuracy Sensitivity          FPR Precision
## 1            0.05 0.9167469   0.8956470 8.255616e-02 0.2639822
## 2            0.10 0.9796493   0.8150393 1.491370e-02 0.6442677
## 3            0.15 0.9824007   0.7463176 9.800895e-03 0.7165335
## 4            0.20 0.9827644   0.6622592 6.648285e-03 0.7672281
## 5            0.25 0.9866226   0.5821416 1.633453e-05 0.9991667
## 6            0.30 0.9849307   0.5287104 0.000000e+00 1.0000000
## 7            0.35 0.9833336   0.4787495 0.000000e+00 1.0000000
## 8            0.40 0.9821477   0.4416427 0.000000e+00 1.0000000
## 9            0.45 0.9803292   0.3847559 0.000000e+00 1.0000000
## 10           0.50 0.9778783   0.3080988 0.000000e+00 1.0000000
##    Detection Prevalence
## 1           0.108553128
## 2           0.040495944
## 3           0.033348679
## 4           0.027608746
## 5           0.018627189
## 6           0.016903611
## 7           0.015306545
## 8           0.014120580
```

```
## 9           0.012302128
## 10          0.009851192
```

```
## # A tibble: 1 x 3
##   CV_accuracy CV_error   AUC
##         <dbl>    <dbl> <dbl>
## 1       0.978   0.0221 0.980
```

When I looked at performance of this model (above), cross-validated prediction accuracy was 97.79%, and the corresponding prediction error was 2.21%. The ROC curve has a pronounced dip in the upper left corner, though the AUC was still quite high, at 0.980.

As I lowered the probability threshold to P=0.05, overall accuracy and precision dropped substantially, and sensitivity peaked at 89.47%. I tried some additional, lower threshold values (below). Though it was possible to improve sensitivity, the precision was extremely poor. Therefore, I settled on a threshold of P=0.10, as a compromise between sensitivity (81.50%) and precision (64.48%). Obviously many tarps would be missed with this model, so if resources were abundant to further investigate pixels identified as tarps, I would lower the threshold, even though it would come at the expense of dramatically reducing precision and increasing detection prevalence.

```
threshold.stats <- thresholder(ridge_final, threshold=seq(0.005, 0.05, by=0.005),
                               statistics="all")
  threshold.stats$FPR <- 1-threshold.stats$Specificity #add FPR

  table <- threshold.stats %>%
    select("prob_threshold","Accuracy", "Sensitivity", "FPR", "Precision",
           "Detection Prevalence")
  print(table)
```

```
##    prob_threshold   Accuracy Sensitivity        FPR  Precision
## 1           0.005 0.04800683   1.0000000 0.98343655 0.03249401
## 2           0.010 0.17226144   1.0000000 0.85507772 0.03719173
## 3           0.015 0.63482541   1.0000000 0.37723576 0.08053198
## 4           0.020 0.76989620   1.0000000 0.23770395 0.12203219
## 5           0.025 0.80495254   0.9995050 0.20147336 0.14080685
## 6           0.030 0.82683708   0.9910964 0.17858830 0.15492168
## 7           0.035 0.84448386   0.9629127 0.15942761 0.16632322
## 8           0.040 0.86143485   0.9297737 0.14082229 0.17906881
## 9           0.045 0.88703528   0.9085109 0.11367389 0.20887402
## 10          0.050 0.91674693   0.8956470 0.08255616 0.26398219
##    Detection Prevalence
## 1             0.9839661
## 2             0.8597115
## 3             0.3971475
## 4             0.2620767
## 5             0.2269888
## 6             0.2045666
## 7             0.1851172
## 8             0.1660473
## 9             0.1390870
## 10            0.1085531
```

I saved key metrics in my summary data frame.

```
#insert all key metrics into performance data frame

ridgeinfo <- data.frame("Ridge regression", "predictors=3; lambda=0.00398",
```

```
                      0.980, 0.10, 0.980,   0.815, 0.015,   0.645)
colnames(ridgeinfo) <- columns
performance_table <- rbind(performance_table, ridgeinfo)
```

Random Forest

I fit a random forest model including all three color values as possible predictors, as in other models. In building this model, I varied two tuning parameters: the number of predictors considered at each node (m=1 or m=2), and the total number of trees (1-100). For the number of predictors at each node, I did not try m=3, as there were only three total predictors. For the number of trees, I chose this relatively low range of values based on some preliminary runs of this chunk of code, in which the default setting for ntree (500) did not produce a substantially superior accuracy as compared to a ntree value below 40. Setting ntree to very high values took a much longer time to run, and since it did not appear to offer any substantive advantage, I restricted my exploration to this handful of lower values.

```
#initialize cluster
no_cores <- detectCores()-1
cl <- makePSOCKcluster(no_cores)
registerDoParallel(cl)

#define range of tuning parameters to explore
mtry <- c(1,2)
ntrees <- c(1, 20, 40, 60, 80, 100)

params <- expand.grid(mtry = mtry, ntrees = ntrees)

#create a vector to store accuracy for each set of parameters
accuracy <- rep(0, nrow(params))

#build random forest for all values of mtry and ntree
for (i in 1:nrow(params)){
  mtry = params[i,1]
  ntree <- params[i,2]

  #set tuneGrid
  tuneGrid <- expand.grid(mtry=mtry)

  #build model
  set.seed(1)
  rf <- train(tarp~Red+Green+Blue,data = haiti, method = "rf", importance=TRUE,
              trControl=trControl, tuneGrid = tuneGrid, ntree = ntree)

  #store accuracy
  accuracy[i] <-  rf$results[1,2]
}

#stop cluster
stopCluster(cl)
registerDoSEQ()
```

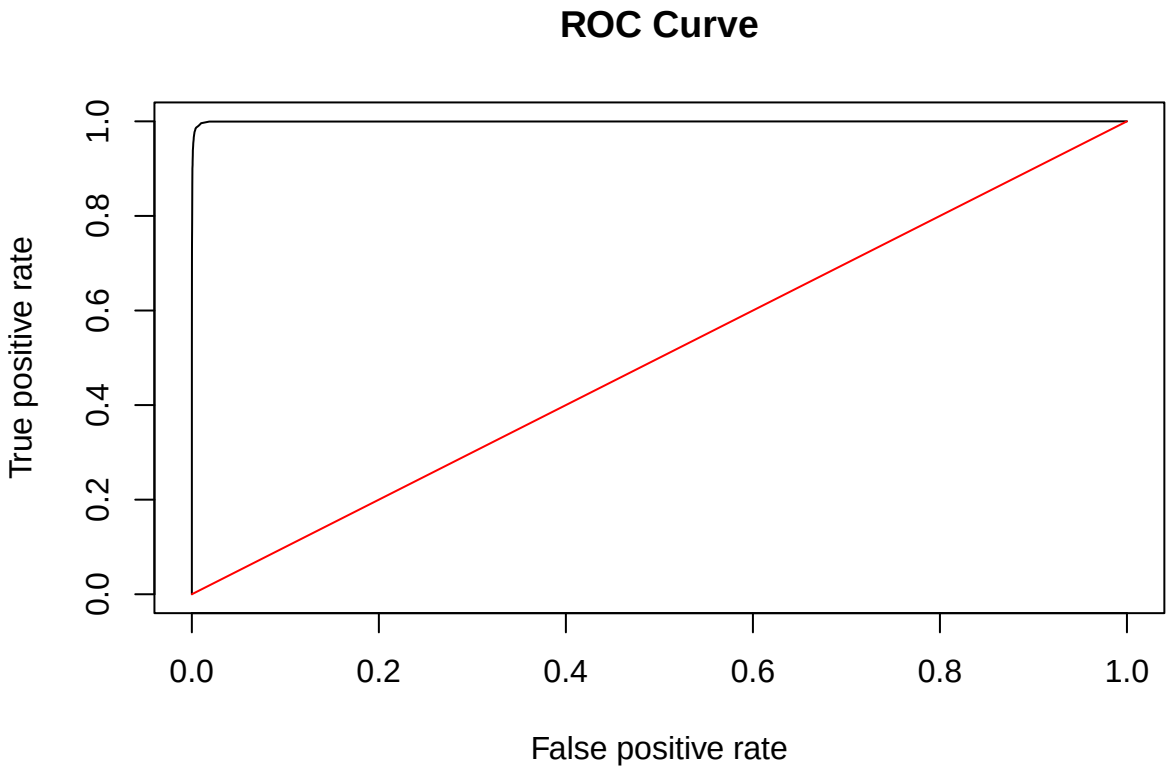
```
#view results and choose the optimal set of tuning parameters

results <- data.frame(params, accuracy)
results
```

```
##    mtry ntrees  accuracy
## 1     1      1 0.9944815
## 2     2      1 0.9959836
## 3     1     20 0.9968849
## 4     2     20 0.9968849
## 5     1     40 0.9970114
## 6     2     40 0.9967901
## 7     1     60 0.9969640
## 8     2     60 0.9968533
## 9     1     80 0.9970431
## 10    2     80 0.9968217
## 11    1    100 0.9968849
## 12    2    100 0.9969482
```

As shown in the table above, the best combinations of tuning parameters are either m=1, ntree=40 (with an accuracy of 99.70%) or m=1, ntree=80 (with an accuracy of 99.70%). Since the computation time will be much shorter with only 40 trees, I chose ntree=40 for my final model. I created a new model using these optimal tuning parameters.

```
#initialize cluster
no_cores <- detectCores()-1
cl <- makePSOCKcluster(no_cores)
registerDoParallel(cl)

#set tuneGrid
tuneGrid <- expand.grid(mtry=1)

#build random forest model
set.seed(1)
rf_final <- train(tarp~Red+Green+Blue,data = haiti, method = "rf", importance=TRUE,
              trControl=trControl, tuneGrid = tuneGrid, ntree = 40)

#stop cluster
stopCluster(cl)
registerDoSEQ()
```

```
evaluate_model(rf_final)
```




```
##    prob_threshold  Accuracy Sensitivity         FPR Precision
## 1            0.05 0.9902753   0.9960420 0.009915215 0.7708394
## 2            0.10 0.9930267   0.9901112 0.006876918 0.8301765
## 3            0.15 0.9954460   0.9851705 0.004214376 0.8876190
## 4            0.20 0.9959204   0.9817076 0.003609985 0.9023781
## 5            0.25 0.9965845   0.9747866 0.002695248 0.9227942
## 6            0.30 0.9968217   0.9688485 0.002254203 0.9342321
## 7            0.35 0.9969007   0.9619226 0.001943844 0.9423411
## 8            0.40 0.9970114   0.9559894 0.001633485 0.9509198
## 9            0.45 0.9969798   0.9480735 0.001404799 0.9571625
## 10           0.50 0.9969798   0.9411452 0.001176113 0.9636869
##    Detection Prevalence
## 1            0.04144458
## 2            0.03831372
## 3            0.03557820
## 4            0.03488246
## 5            0.03377557
## 6            0.03315886
## 7            0.03263705
## 8            0.03214686
## 9            0.03167248
## 10           0.03122971
```

```
## # A tibble: 1 x 3
##   CV_accuracy CV_error   AUC
##         <dbl>    <dbl> <dbl>
## 1       0.997  0.00299 0.999
```

When I looked at performance of this model (above), cross-validated prediction accuracy was 99.70%, and the corresponding prediction error was 0.30%. The ROC curve comes very close to the upper left corner, and the AUC was 0.999 - the highest yet, and nearly 1, which is the maximum possible.

At probability thresholds below 0.5, sensitivity increased and precision declined. Peak sensitivity (99.60%) was reached at P=0.05, but precision was only 77.08 at this threshold. As with past models, I sought to balance sensitivity with precision, with the recognition that resources would likely be limited. I settled on a threshold of 0.15, which gave a sensitivity of 98.52%, a precision of 88.76%, and a detection prevalence of 3.56%. Given the high sensitivity and precision, this model is on par with the k-nearest neighbors model.

I saved key metrics in my summary data frame.

```
#insert all key metrics into performance data frame

rfinfo <- data.frame("Random forest", "predictors=3; mtry=1; ntree=40",
                     0.999, 0.15, 0.995,   0.985, 0.004,   0.888)
colnames(rfinfo) <- columns
performance_table <- rbind(performance_table, rfinfo)
```

Support Vector Machine

I tried three different support vector machines, each using a different kernel (linear, polynomial, and radial basis function). For all three of these models, I incorporated all three predictors (as in all other models) and tuned the cost parameter. For the polynomial kernel, I also tuned the degree of the polynomial, and for the radial basis function kernel, I also tuned the sigma parameter.

```
#SVM with linear kernel

#set range of cost values to try
cost <- c(0.01, 0.1, 0.5, 1, 5, 10)

#set tuneGrid
tuneGrid <- expand.grid(C = cost)


#initialize cluster
no_cores <- detectCores()-1
cl <- makePSOCKcluster(no_cores)
registerDoParallel(cl)

set.seed(1)

#build model
svm_linear <- train(tarp~Red+Green+Blue, data=haiti, method ='svmLinear',
              trControl=trControl, tuneGrid = tuneGrid)

svm_linear
```

```
## Support Vector Machines with Linear Kernel
##
## 63241 samples
##     3 predictor
##     2 classes: 'tarp', 'other'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
```

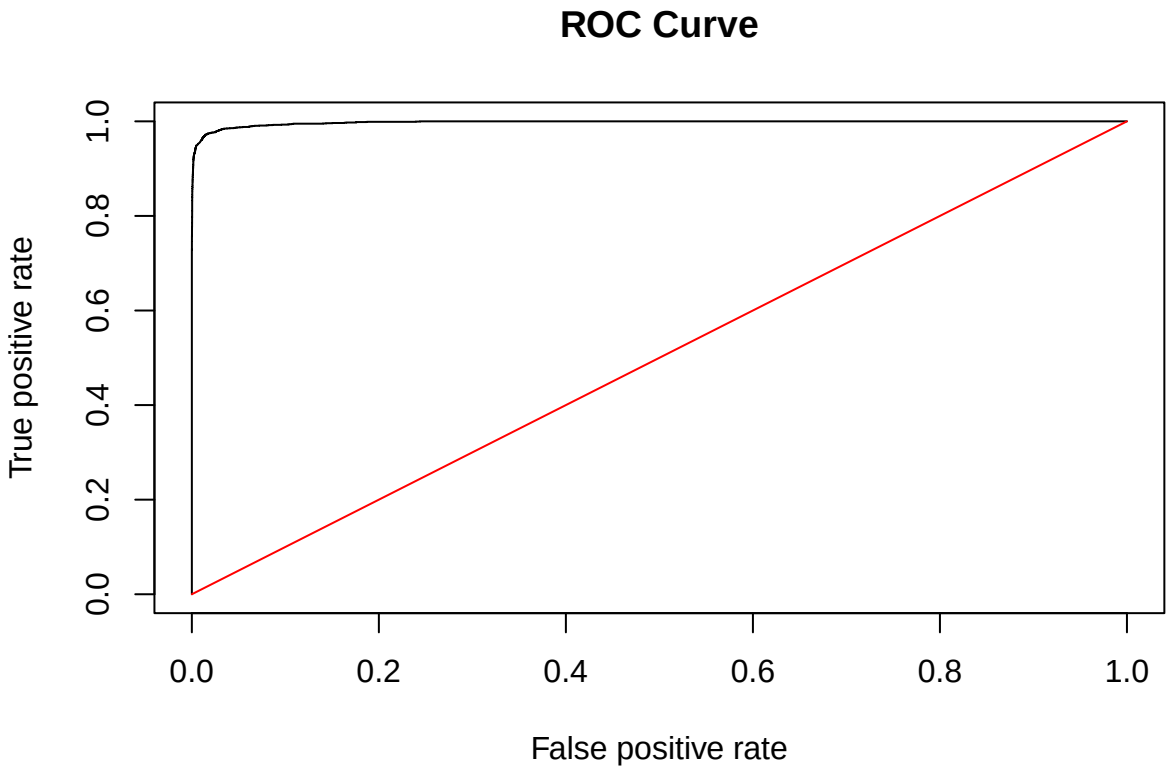
```
## Summary of sample sizes: 56918, 56917, 56917, 56917, 56916, 56917, ...
## Resampling results across tuning parameters:
##
##   C      Accuracy   Kappa
##    0.01  0.9910343  0.8549687
##    0.10  0.9952088  0.9186876
##    0.50  0.9953986  0.9224700
##    1.00  0.9953986  0.9225159
##    5.00  0.9953828  0.9222408
##   10.00  0.9953669  0.9220081
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was C = 0.5.
```

```
#stop cluster
stopCluster(cl)
registerDoSEQ()
```

According to the above output, the best corss-validated prediction accuracy obtained by a linear kernel with an optimized cost = 0.5 was 99.54%. I rebuilt this model with its optimized cost parameter for further evaluation.

```
#set tuneGrid
tuneGrid <- expand.grid(C = 0.5)

#initialize cluster
no_cores <- detectCores()-1
cl <- makePSOCKcluster(no_cores)
registerDoParallel(cl)

set.seed(1)

#build model
svm_linear_final <- train(tarp~Red+Green+Blue, data=haiti, method ='svmLinear',
             trControl=trControl, tuneGrid = tuneGrid)

#stop cluster
stopCluster(cl)
registerDoSEQ()
```

```
evaluate_model(svm_linear_final)
```




```
##    prob_threshold  Accuracy Sensitivity         FPR Precision
## 1            0.05 0.9834760   0.9723065 0.016155089 0.6661940
## 2            0.10 0.9939122   0.9465932 0.004524743 0.8737868
## 3            0.15 0.9952088   0.9322587 0.002711572 0.9193726
## 4            0.20 0.9957148   0.9233527 0.001894838 0.9418111
## 5            0.25 0.9957148   0.9134590 0.001568147 0.9507705
## 6            0.30 0.9956199   0.9075257 0.001470137 0.9533536
## 7            0.35 0.9955409   0.9025825 0.001388465 0.9555677
## 8            0.40 0.9955251   0.8986222 0.001274120 0.9588875
## 9            0.45 0.9954144   0.8926938 0.001192445 0.9611734
## 10           0.50 0.9953986   0.8872628 0.001029097 0.9661406
##    Detection Prevalence
## 1            0.04672596
## 2            0.03464524
## 3            0.03243150
## 4            0.03135626
## 5            0.03072375
## 6            0.03043912
## 7            0.03020194
## 8            0.02996475
## 9            0.02969594
## 10           0.02936389
```

```
## # A tibble: 1 x 3
##   CV_accuracy CV_error   AUC
##         <dbl>    <dbl> <dbl>
## 1       0.995  0.00460 0.998
```

The SVM with linear kernel had an AUC value of 0.997. To attain high sensitivity of 97.23%, the threshold has to be lowered to 0.05, and at this threshold, the precision is a relatively low 66.62% and detection prevalence is 4.67%. Detecting this many "tarp" points - many of which would be erroneously identified - could considerably strain resources. Next, I tried a polynomial kernel to see if I could improve upon these results.

```
#SVM with polynomial kernel

#set range of parameter values to try
cost <- c(0.1, 0.5, 1)
degree <- c(2, 3, 4)
scale <- c(0.001, 0.01, 0.1, 1)

#set tuneGrid
tuneGrid <- expand.grid(C = cost, degree = degree, scale = scale)

#initialize cluster
no_cores <- detectCores()-1
cl <- makePSOCKcluster(no_cores)
registerDoParallel(cl)

set.seed(1)

#build model
svm_poly <- train(tarp~Red+Green+Blue, data=haiti, method ='svmPoly',
             trControl=trControl, tuneGrid = tuneGrid)

svm_poly
```

```
## Support Vector Machines with Polynomial Kernel
##
## 63241 samples
##     3 predictor
##     2 classes: 'tarp', 'other'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 56918, 56917, 56917, 56917, 56916, 56917, ...
## Resampling results across tuning parameters:
##
##   C    degree  scale  Accuracy   Kappa
##   0.1  2       0.001  0.9883936  0.8240979
##   0.1  2       0.010  0.9894214  0.8395990
##   0.1  2       0.100  0.9958888  0.9329030
##   0.1  2       1.000  0.9957307  0.9297412
##   0.1  3       0.001  0.9884411  0.8251859
##   0.1  3       0.010  0.9919199  0.8717207
##   0.1  3       0.100  0.9957983  0.9314075
##   0.1  3       1.000  0.9962208  0.9381743
##   0.1  4       0.001  0.9888048  0.8303626
##   0.1  4       0.010  0.9953550  0.9218082
##   0.1  4       0.100  0.9959677  0.9345730
##   0.1  4       1.000  0.9963789  0.9406732
##   0.5  2       0.001  0.9883462  0.8234196
##   0.5  2       0.010  0.9954776  0.9250629
```

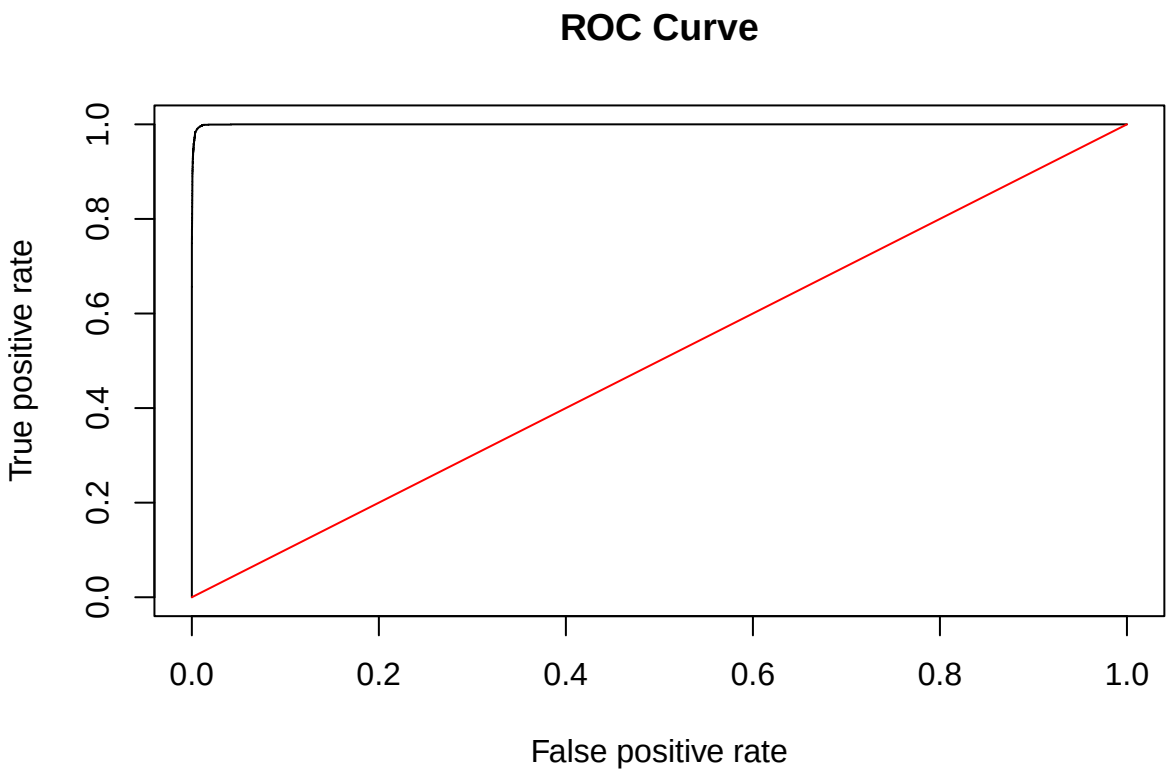
```
##   0.5  2       0.100  0.9959520  0.9338078
##   0.5  2       1.000  0.9956832  0.9286936
##   0.5  3       0.001  0.9885202  0.8262779
##   0.5  3       0.010  0.9959362  0.9337456
##   0.5  3       0.100  0.9960311  0.9353916
##   0.5  3       1.000  0.9962999  0.9392570
##   0.5  4       0.001  0.9887890  0.8299251
##   0.5  4       0.010  0.9958712  0.9325768
##   0.5  4       0.100  0.9959836  0.9347906
##   0.5  4       1.000  0.9966319  0.9447475
##   1.0  2       0.001  0.9883462  0.8232025
##   1.0  2       0.010  0.9959678  0.9341635
##   1.0  2       0.100  0.9959362  0.9334546
##   1.0  2       1.000  0.9956990  0.9288208
##   1.0  3       0.001  0.9894373  0.8346116
##   1.0  3       0.010  0.9958185  0.9314467
##   1.0  3       0.100  0.9960311  0.9353247
##   1.0  3       1.000  0.9963473  0.9402376
##   1.0  4       0.001  0.9905758  0.8502711
##   1.0  4       0.010  0.9956403  0.9284150
##   1.0  4       0.100  0.9960152  0.9348620
##   1.0  4       1.000  0.9966319  0.9448110
##
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were degree = 4, scale = 1 and C = 1.
```

```
#stop cluster
stopCluster(cl)
registerDoSEQ()
```

Of the tuning parameters I evaluated with this kernel, the best prediction accuracy (99.66%) was achieved with a fourth degree polynomial, with cost = 1 and scale = 1. The same high accuracy was also achieved with a fourth degree polynomial with cost = 0.5 and scale =1. Nonetheless, I selected the parameters that the output identified and rebuilt the final version of this model for subsequent evaluation.

```
#set tuneGrid
tuneGrid <- expand.grid(C = 1, degree = 4, scale = 1)

#initialize cluster
no_cores <- detectCores()-1
cl <- makePSOCKcluster(no_cores)
registerDoParallel(cl)

set.seed(1)

#build model
svm_poly_final <- train(tarp~Red+Green+Blue, data=haiti, method ='svmPoly',
            trControl=trControl, tuneGrid = tuneGrid)

#stop cluster
stopCluster(cl)
registerDoSEQ()
```

```
evaluate_model(svm_poly_final)
```




```
##    prob_threshold  Accuracy Sensitivity         FPR Precision
## 1            0.05 0.9899274   0.9965444 0.010290931 0.7621869
## 2            0.10 0.9960943   0.9802297 0.003381299 0.9054528
## 3            0.15 0.9961576   0.9723162 0.003054603 0.9131546
## 4            0.20 0.9964264   0.9663757 0.002580896 0.9252564
## 5            0.25 0.9965687   0.9599449 0.002221534 0.9345555
## 6            0.30 0.9967268   0.9564795 0.001943844 0.9420688
## 7            0.35 0.9966477   0.9510413 0.001845837 0.9445169
## 8            0.40 0.9966952   0.9455982 0.001617153 0.9508316
## 9            0.45 0.9967742   0.9411476 0.001388465 0.9572518
## 10           0.50 0.9966794   0.9352144 0.001290457 0.9599485
##    Detection Prevalence
## 1            0.04182415
## 2            0.03461363
## 3            0.03404437
## 4            0.03339606
## 5            0.03284263
## 6            0.03246312
## 7            0.03219431
## 8            0.03179900
## 9            0.03143530
## 10           0.03115068
```

```
## # A tibble: 1 x 3
##   CV_accuracy CV_error   AUC
##         <dbl>    <dbl> <dbl>
## 1       0.997  0.00332  1.00
```

As seen in the above output, the polynomial kernel with optimized tuning parameters has an AUC of 0.9997, a cross-validated prediction accuracy of 99.67%, and a prediction error of 0.33%. The ROC curve reflects this high performance and comes very close to the upper left corner of the plot. At a probability threshold of 0.1, the sensitivity is 98.02% and the precision is 90.54%. This model not only outperforms the linear kernel, but also the other models that have been evaluated thus far. The last kernel I tried was the radial basis function kernel.

```
#SVM with radial basis function kernel

#set range of parameter values to try
cost <- c(0.1, 0.5, 1)
sigma <- c(0.01, 0.1, 1)

#set tuneGrid
tuneGrid <- expand.grid(C = cost, sigma = sigma)

#initialize cluster
no_cores <- detectCores()-1
cl <- makePSOCKcluster(no_cores)
registerDoParallel(cl)

set.seed(1)

#build model
svm_radial <- train(tarp~Red+Green+Blue, data=haiti, method ='svmRadial',
             trControl=trControl, tuneGrid = tuneGrid)

svm_radial
```

```
## Support Vector Machines with Radial Basis Function Kernel
##
## 63241 samples
##     3 predictor
##     2 classes: 'tarp', 'other'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 56918, 56917, 56917, 56917, 56916, 56917, ...
## Resampling results across tuning parameters:
##
##   C    sigma  Accuracy   Kappa
##   0.1  0.01   0.9897061  0.8448085
##   0.1  0.10   0.9950191  0.9179066
##   0.1  1.00   0.9958413  0.9321465
##   0.5  0.01   0.9955567  0.9267475
##   0.5  0.10   0.9958255  0.9319314
##   0.5  1.00   0.9961101  0.9365225
##   1.0  0.01   0.9959240  0.9334966
##   1.0  0.10   0.9958888  0.9332310
##   1.0  1.00   0.9963789  0.9408698
##
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were sigma = 1 and C = 1.
```

```
#stop cluster
stopCluster(cl)
registerDoSEQ()
```

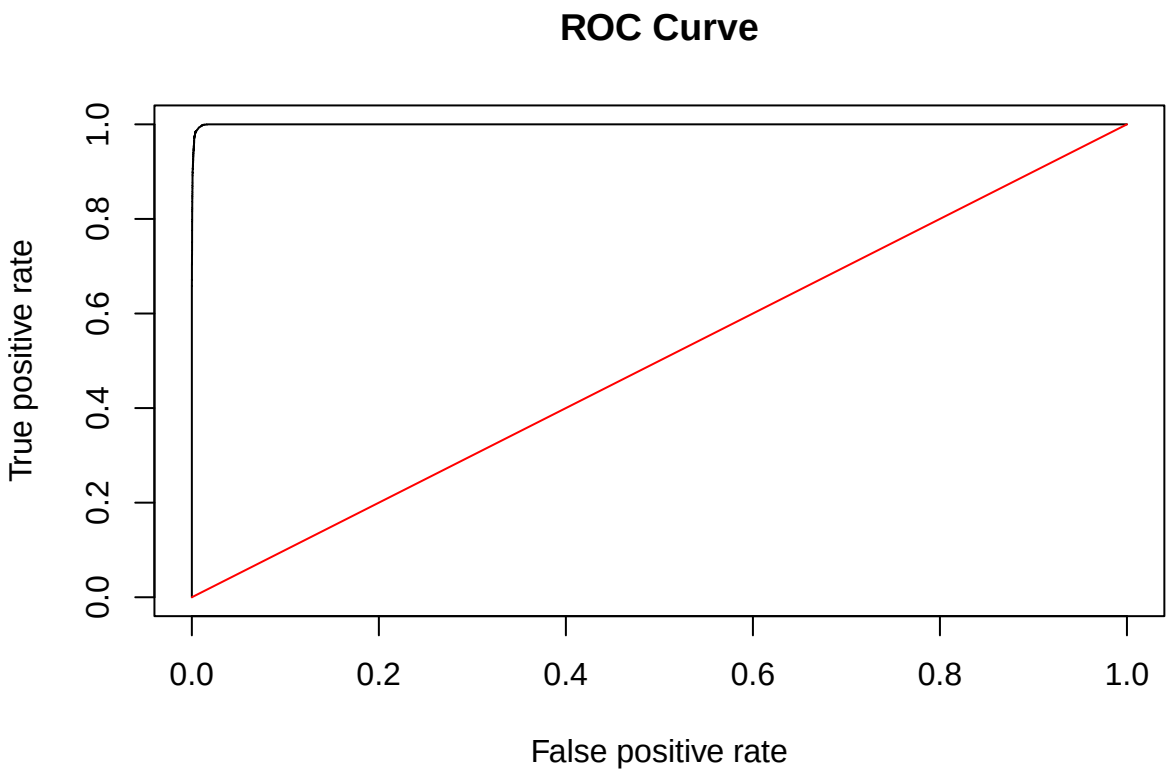
The output above indicates that the optimal tuning parameters are cost = 1 and sigma = 1, so I rebuilt this model with optimal tuning parameters for further evaluation.

```
#set tuneGrid
tuneGrid <- expand.grid(C = 1, sigma = 1)

#initialize cluster
no_cores <- detectCores()-1
cl <- makePSOCKcluster(no_cores)
registerDoParallel(cl)

set.seed(1)

#build model
svm_radial_final <- train(tarp~Red+Green+Blue, data=haiti, method ='svmRadial',
             trControl=trControl, tuneGrid = tuneGrid)

svm_radial_final
```

```
## Support Vector Machines with Radial Basis Function Kernel
##
## 63241 samples
##     3 predictor
##     2 classes: 'tarp', 'other'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 56918, 56917, 56917, 56917, 56916, 56917, ...
## Resampling results:
##
##   Accuracy   Kappa
##   0.9963631  0.9406054
##
## Tuning parameter 'sigma' was held constant at a value of 1
## Tuning parameter
##  'C' was held constant at a value of 1
```

```
#stop cluster
stopCluster(cl)
registerDoSEQ()
```

```
evaluate_model(svm_radial_final)
```




```
##    prob_threshold  Accuracy Sensitivity         FPR Precision
## 1            0.05 0.9957148   0.9831903 0.003871353 0.8935748
## 2            0.10 0.9961576   0.9767644 0.003201619 0.9097470
## 3            0.15 0.9965371   0.9728089 0.002678906 0.9230920
## 4            0.20 0.9965212   0.9663757 0.002482891 0.9278662
## 5            0.25 0.9964264   0.9599522 0.002368550 0.9305825
## 6            0.30 0.9964580   0.9559918 0.002205202 0.9348278
## 7            0.35 0.9964580   0.9505487 0.002025519 0.9394611
## 8            0.40 0.9964580   0.9465932 0.001894840 0.9428887
## 9            0.45 0.9964422   0.9406550 0.001715160 0.9477039
## 10           0.50 0.9963631   0.9332342 0.001551815 0.9521480
##    Detection Prevalence
## 1            0.03518288
## 2            0.03432901
## 3            0.03369651
## 4            0.03330120
## 5            0.03298495
## 6            0.03270032
## 7            0.03235244
## 8            0.03209944
## 9            0.03173575
## 10           0.03134043
```

```
## # A tibble: 1 x 3
##   CV_accuracy CV_error   AUC
##         <dbl>    <dbl> <dbl>
## 1       0.996  0.00364  1.00
```

The AUC for this model was 0.9996, with a cross-validated prediction accuracy of 99.64% and a corresponding prediction error rate of 0.36%. This model's performance is very strong relative to the others, and is comparable to both the KNN model and the random forest model. At a threshold of 0.05, the sensitivity is 98.32% and precision is 89.36%. Since this model very slightly underperfoms the polynomial kernel and I wanted to choose just one SVM model for comparison to others, I decided to keep the polynomial kernel as my representative SVM model. Thus, I saved the optimized tuning parameters and performance metrics for the polynomial kernel in my summary data frame.

```
#insert all key metrics into performance data frame

svminfo <- data.frame("Support vector machine",
                      "predictors=3; kernel=polynomial; cost=1; degree=4; scale=1",
                      0.9997, 0.10, 0.996,  0.980,  0.003,  0.905)

colnames(svminfo) <- columns
performance_table <- rbind(performance_table, svminfo)
```

Performance Table

A summary of model performance is shown in the below table. Accuracy, Sensitivity, FPR, and Precision are for the given threshold value.

```
performance_table
```

```
##                    Model
## 1    Logistic regression
## 2                    LDA
## 3                    QDA
## 4                    KNN
## 5       Ridge regression
## 6          Random forest
## 7 Support vector machine
##                                       Model Tuning Parameters    AUC Threshold
## 1                                                 predictors=3 0.9980     0.050
## 2                                                 predictors=3 0.9890     0.005
## 3                                                 predictors=3 0.9980     0.025
## 4                                           predictors=3; k=7 0.9980     0.100
## 5                              predictors=3; lambda=0.00398 0.9800     0.100
## 6                            predictors=3; mtry=1; ntree=40 0.9990     0.150
## 7 predictors=3; kernel=polynomial; cost=1; degree=4; scale=1 0.9997     0.100
##   Accuracy Sensitivity/TPR   FPR Precision
## 1    0.987           0.976 0.013     0.717
## 2    0.970           0.894 0.028     0.517
## 3    0.982           0.984 0.018     0.642
## 4    0.996           0.986 0.004     0.887
## 5    0.980           0.815 0.015     0.645
## 6    0.995           0.985 0.004     0.888
## 7    0.996           0.980 0.003     0.905
```

Holdout Data Wrangling and EDA

For each of the eight test data files, I imported the file, taking care to omit header rows and select relevant columns as needed. I added a tarp column that identified the pixel as either "tarp" or "other," which I inferred from the file names.

Because color value columns were not labeled, I had to make a determination about which column corresponded

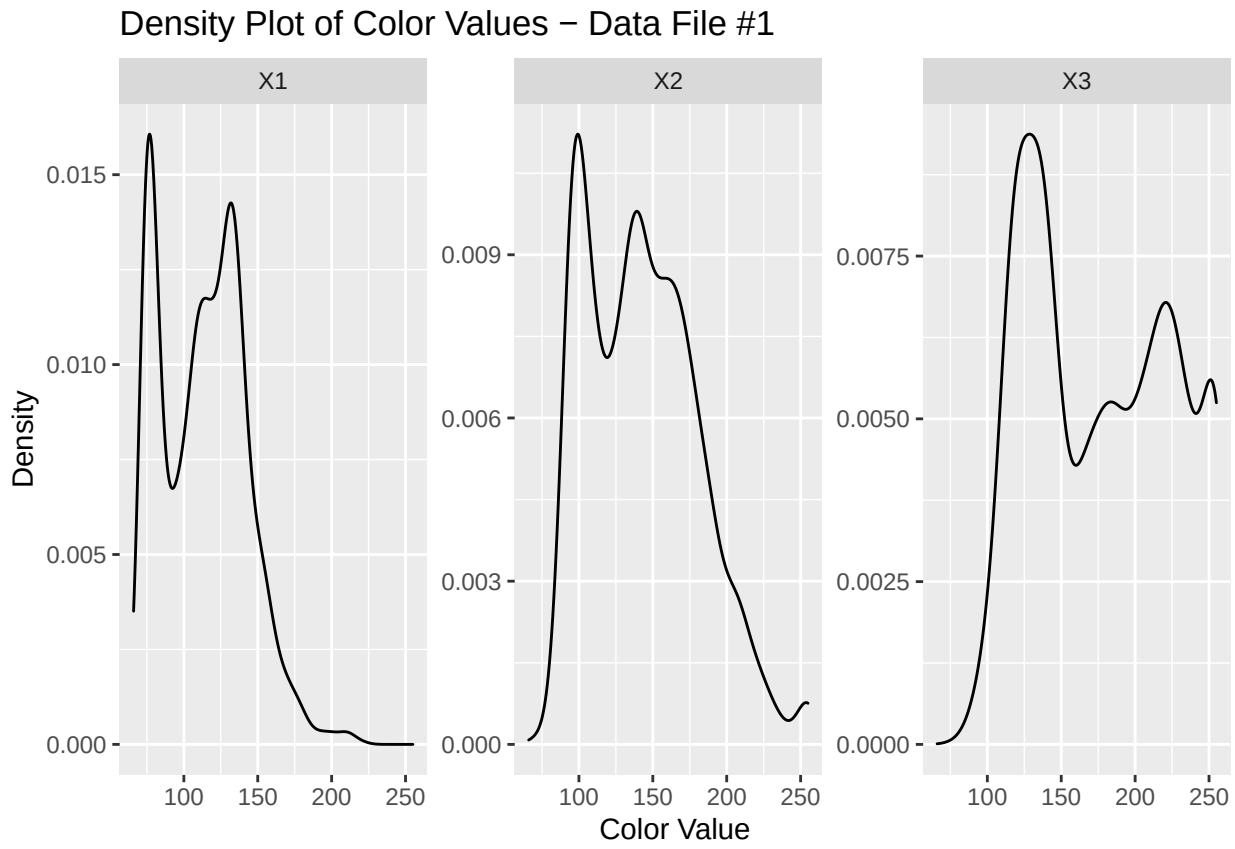
to which color. I did this by comparing density plots of each column to those in the EDA section for the training data. If pixels came from a "tarp" file, I compared color value density curves to those for tarps in the training set. If they came from a "non-tarp" file, I compared color value density curves to those for "other" pixels in the training set.

```
#Import data file #1

tarp1 <- read_table('orthovnir067_ROI_Blue_Tarps_data.txt', skip=1, col_names=FALSE)
```

```
##
## -- Column specification --------------------------------------------------------
## cols(
##   X1 = col_double(),
##   X2 = col_double(),
##   X3 = col_double()
## )
```

```
#add class label, change it to factor

tarp1$tarp <- rep('tarp', nrow(tarp1))
tarp1$tarp <- factor(tarp1$tarp)
```

```
#Column identification for data file #1

#long pivot for side-by-side plots

long_tarp1 <- tarp1[,1:4]%>%
  pivot_longer(cols=-tarp, names_to = "Color", values_to = "Value")

#density plots with color as facet wrap
ggplot(long_tarp1, aes(x=Value)) +
  geom_density() +
  facet_wrap(facets='Color', ncol=4, scales = 'free_y')+
  labs(x="Color Value", y="Density",
       title = "Density Plot of Color Values - Data File #1")
```



Comparing these density curves to those for tarp pixels in the training set, I believe X3 most clearly corresponds to blue. Neither of the other two plots clearly resemble the red or green values for training set tarp pixels, but since green values tended to be a little higher than red values, I assigned X2 to green and X1 to red.

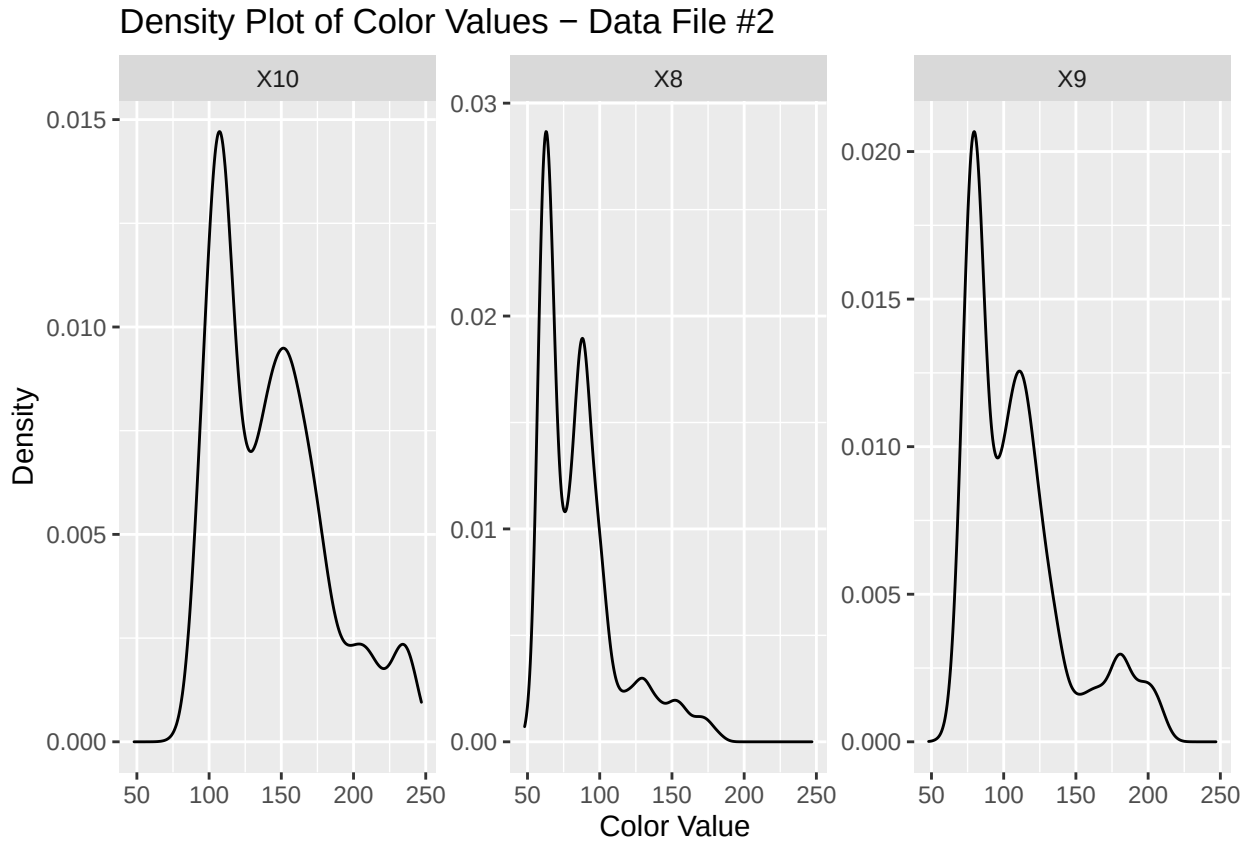
```
tarp1 <- tarp1%>%
  rename(Red=X1, Green=X2, Blue=X3)
```

```
#Import data file #2

tarp2 <- read_table('orthovnir078_ROI_Blue_Tarps.txt', skip=8, col_names=FALSE)
```

```
##
## -- Column specification --------------------------------------------------------
## cols(
##   X1 = col_double(),
##   X2 = col_double(),
##   X3 = col_double(),
##   X4 = col_double(),
##   X5 = col_double(),
##   X6 = col_double(),
##   X7 = col_double(),
##   X8 = col_double(),
##   X9 = col_double(),
##   X10 = col_double()
## )
```

```
#select last three columns
tarp2 <- tarp2%>%
  select(X8, X9, X10)

#add class label, change it to factor

tarp2$tarp <- rep('tarp', nrow(tarp2))
tarp2$tarp <- factor(tarp2$tarp)
```

```
#Column identification for data file #2

#long pivot for side-by-side plots

long_tarp2 <- tarp2[,1:4]%>%
  pivot_longer(cols=-tarp, names_to = "Color", values_to = "Value")

#density plots with color as facet wrap
ggplot(long_tarp2, aes(x=Value)) +
  geom_density() +
  facet_wrap(facets='Color', ncol=4, scales = 'free_y')+
  labs(x="Color Value", y="Density",
       title = "Density Plot of Color Values - Data File #2")
```



None of these density curves resemble those for color values for blue tarp pixels in the training set; nonetheless, I assigned the curve with the highest values (X10) to blue, the curve with second highest values (X9) to green, and lowest values (X8) to red.

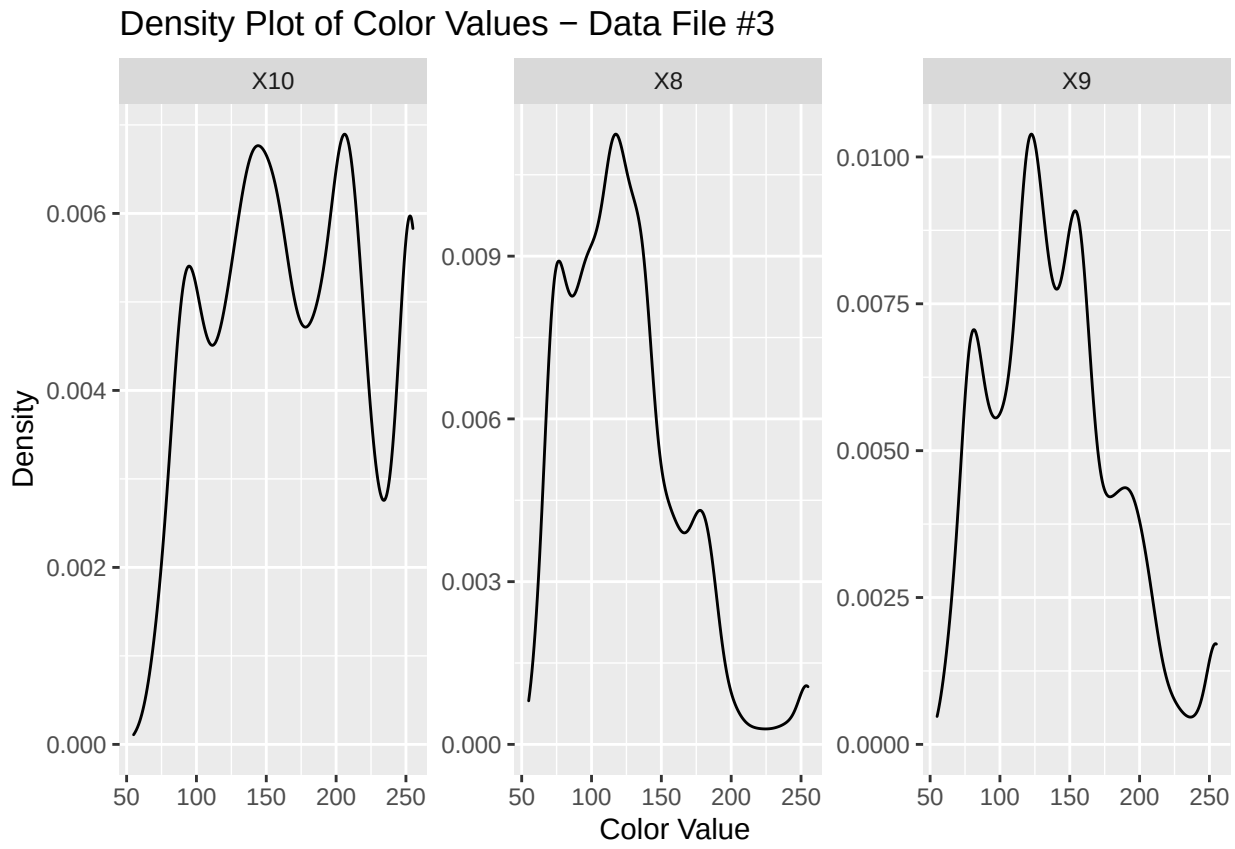
```
tarp2 <- tarp2%>%
  rename(Red=X8, Green=X9, Blue=X10)
```

```
#Import data file #3

tarp3 <- read_table('orthovnir069_ROI_Blue_Tarps.txt', skip=8, col_names=FALSE)
```

```
##
## -- Column specification --------------------------------------------------------
## cols(
##   X1 = col_double(),
##   X2 = col_double(),
##   X3 = col_double(),
##   X4 = col_double(),
##   X5 = col_double(),
##   X6 = col_double(),
##   X7 = col_double(),
##   X8 = col_double(),
##   X9 = col_double(),
##   X10 = col_double()
## )
```

```
#select last three columns
tarp3 <- tarp3%>%
  select(X8, X9, X10)

#add class label, change it to factor

tarp3$tarp <- rep('tarp', nrow(tarp3))
tarp3$tarp <- factor(tarp3$tarp)
```

```
#Column identification for data file #3

#long pivot for side-by-side plots

long_tarp3 <- tarp3[,1:4]%>%
  pivot_longer(cols=-tarp, names_to = "Color", values_to = "Value")

#density plots with color as facet wrap
ggplot(long_tarp3, aes(x=Value)) +
  geom_density() +
  facet_wrap(facets='Color', ncol=4, scales = 'free_y')+
  labs(x="Color Value", y="Density",
       title = "Density Plot of Color Values - Data File #3")
```



The density plot for X10 most clearly corresponded to the training set's blue density curve. As with the previous two files, I assigned X9 to green and X8 to red because values were slightly higher in X9, and this more closely matched green color values in the training set.

```
tarp3 <- tarp3%>%
  rename(Red=X8, Green=X9, Blue=X10)
```

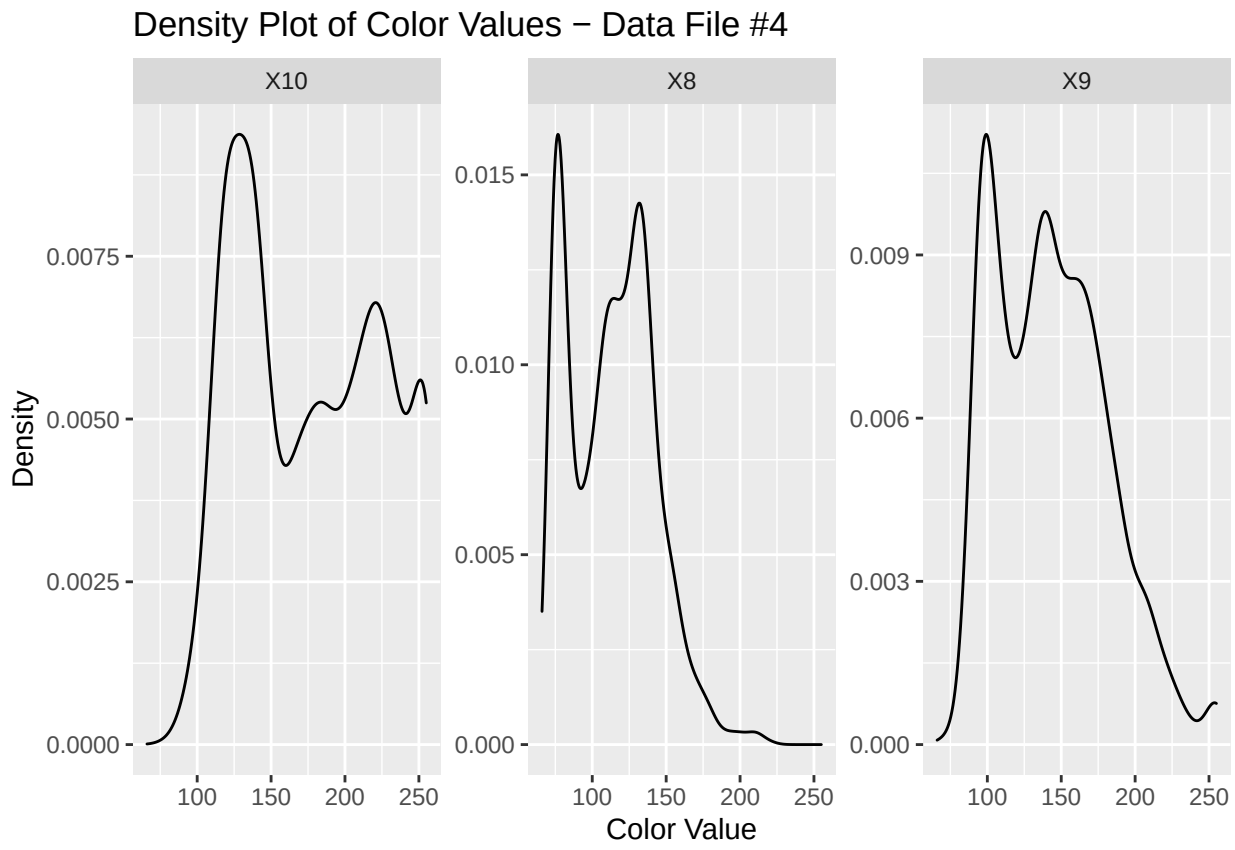
```
#Import data file #4

tarp4 <- read_table('orthovnir067_ROI_Blue_Tarps.txt', skip=8, col_names=FALSE)
```

```
##
## -- Column specification --------------------------------------------------------
## cols(
##   X1 = col_double(),
##   X2 = col_double(),
##   X3 = col_double(),
##   X4 = col_double(),
##   X5 = col_double(),
##   X6 = col_double(),
##   X7 = col_double(),
##   X8 = col_double(),
##   X9 = col_double(),
##   X10 = col_double()
## )
```

```
#select last three columns
tarp4 <- tarp4%>%
```

```
  select(X8, X9, X10)

#add class label, change it to factor

tarp4$tarp <- rep('tarp', nrow(tarp4))
tarp4$tarp <- factor(tarp4$tarp)
```

```
#Column identification for data file #4

#long pivot for side-by-side plots

long_tarp4 <- tarp4[,1:4]%>%
  pivot_longer(cols=-tarp, names_to = "Color", values_to = "Value")

#density plots with color as facet wrap
ggplot(long_tarp4, aes(x=Value)) +
  geom_density() +
  facet_wrap(facets='Color', ncol=4, scales = 'free_y')+
  labs(x="Color Value", y="Density",
       title = "Density Plot of Color Values - Data File #4")
```



Once more, the column X10 most closely matches the training set for blue, X9 most closely matches to green, and X8 most closely matches to red. It also makes sense that labeling would be consistent among all these files, though I did not want to assume this would be the case.

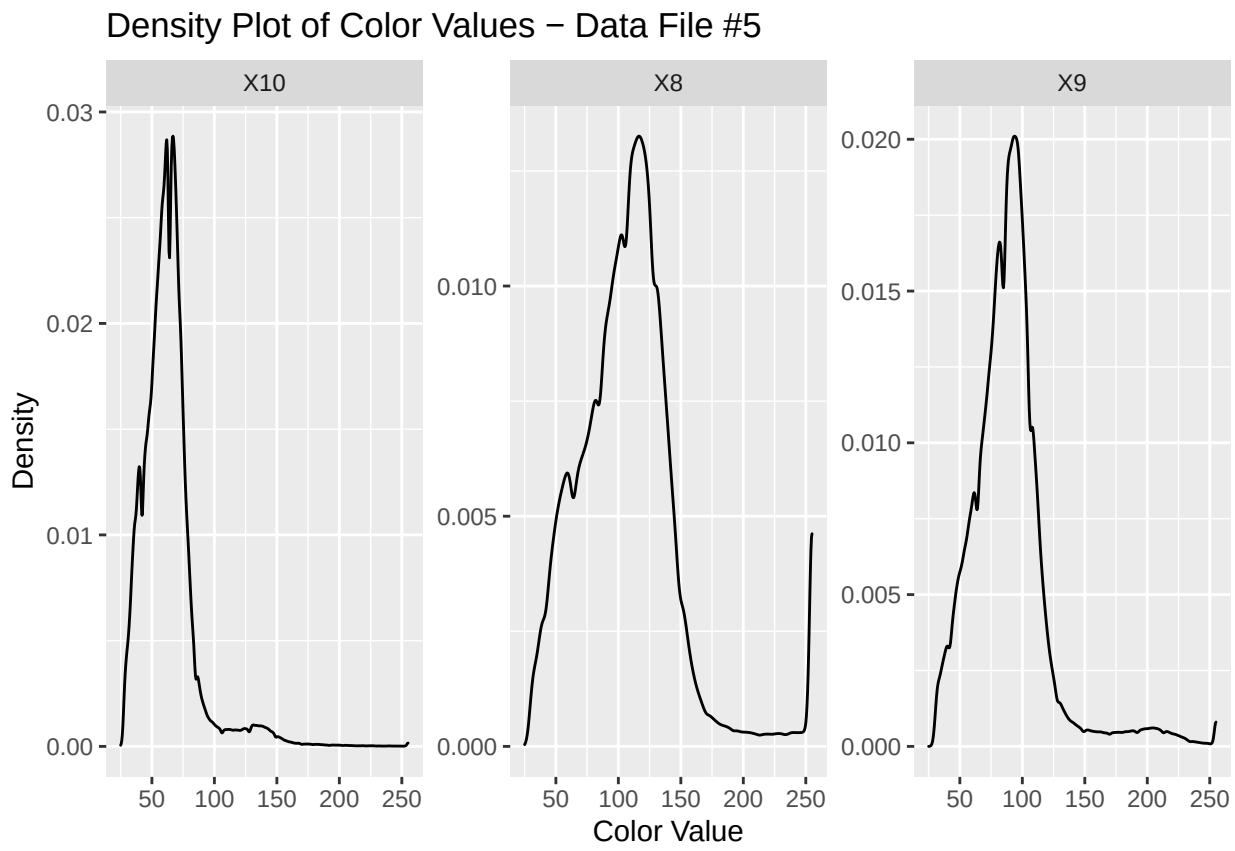
```
tarp4 <- tarp4%>%
  rename(Red=X8, Green=X9, Blue=X10)
```

```r
#Import data file #5

other1 <- read_table('orthovnir057_ROI_NON_Blue_Tarps.txt',
                     skip=8,col_names=FALSE)
```

```
##
## -- Column specification --------------------------------------------------------
## cols(
##   X1 = col_double(),
##   X2 = col_double(),
##   X3 = col_double(),
##   X4 = col_double(),
##   X5 = col_double(),
##   X6 = col_double(),
##   X7 = col_double(),
##   X8 = col_double(),
##   X9 = col_double(),
##   X10 = col_double()
## )
```

```r
#select last three columns
other1 <- other1%>%
  select(X8, X9, X10)

#add class label, change it to factor

other1$tarp <- rep('other', nrow(other1))
other1$tarp <- factor(other1$tarp)
```

```r
#Column identification for data file #5

#long pivot for side-by-side plots

long_other1 <- other1[,1:4]%>%
  pivot_longer(cols=-tarp, names_to = "Color", values_to = "Value")

#density plots with color as facet wrap
ggplot(long_other1, aes(x=Value)) +
  geom_density() +
  facet_wrap(facets='Color', ncol=4, scales = 'free_y')+
  labs(x="Color Value", y="Density",
       title = "Density Plot of Color Values - Data File #5")
```



These plots do not closely resemble those for non-tarp pixels in the training set. However, the spike in very high color values for column X8 leads me to believe that this column corresponds to red. Since X10 has the lowest values, I believe this corresponds to blue, and the slightly higher values of X9 compared to X10 lead me to believe it corresponds to green.

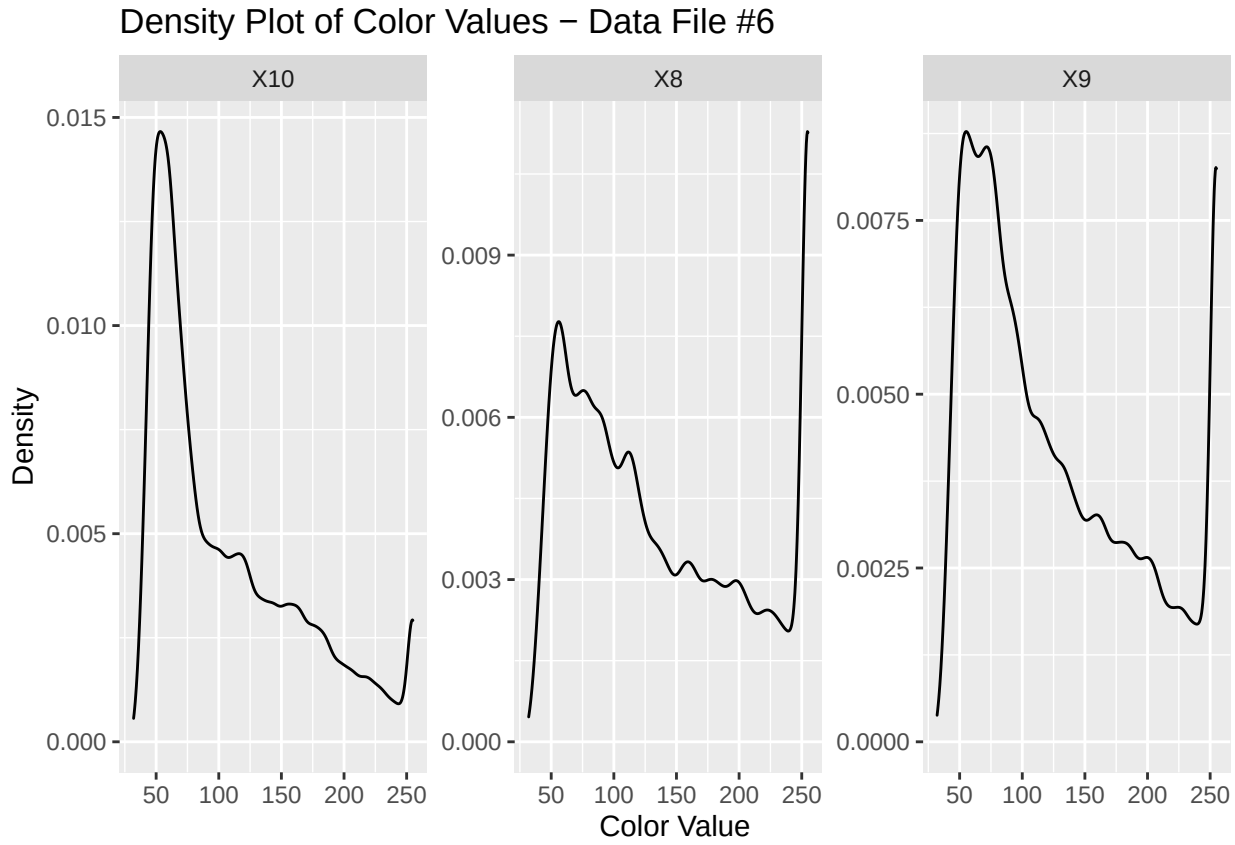
```
other1 <- other1%>%
  rename(Red=X8, Green=X9, Blue=X10)
```

```
#Import data file #6

other2 <- read_table('orthovnir078_ROI_NON_Blue_Tarps.txt',
                     skip=8,col_names=FALSE)
```

```
##
## -- Column specification --------------------------------------------------------
## cols(
##   X1 = col_double(),
##   X2 = col_double(),
##   X3 = col_double(),
##   X4 = col_double(),
##   X5 = col_double(),
##   X6 = col_double(),
##   X7 = col_double(),
##   X8 = col_double(),
##   X9 = col_double(),
##   X10 = col_double()
## )
```

```r
#select last three columns
other2 <- other2%>%
  select(X8, X9, X10)

#add class label, change it to factor

other2$tarp <- rep('other', nrow(other2))
other2$tarp <- factor(other2$tarp)
```

```r
#Column identification for data file #6

#long pivot for side-by-side plots

long_other2 <- other2[,1:4]%>%
  pivot_longer(cols=-tarp, names_to = "Color", values_to = "Value")

#density plots with color as facet wrap
ggplot(long_other2, aes(x=Value)) +
  geom_density() +
  facet_wrap(facets='Color', ncol=4, scales = 'free_y')+
  labs(x="Color Value", y="Density",
       title = "Density Plot of Color Values - Data File #6")
```



As with the previous file, these plots do not closely resemble the training data, and show a very similar pattern to one another, making it difficult to distinguish between colors. Nonetheless, based on past precedent of other data files, the fact that the X8 column has the highest values and X10 has the lowest values, I

assigned these columns in the same way as I have done for all other files.

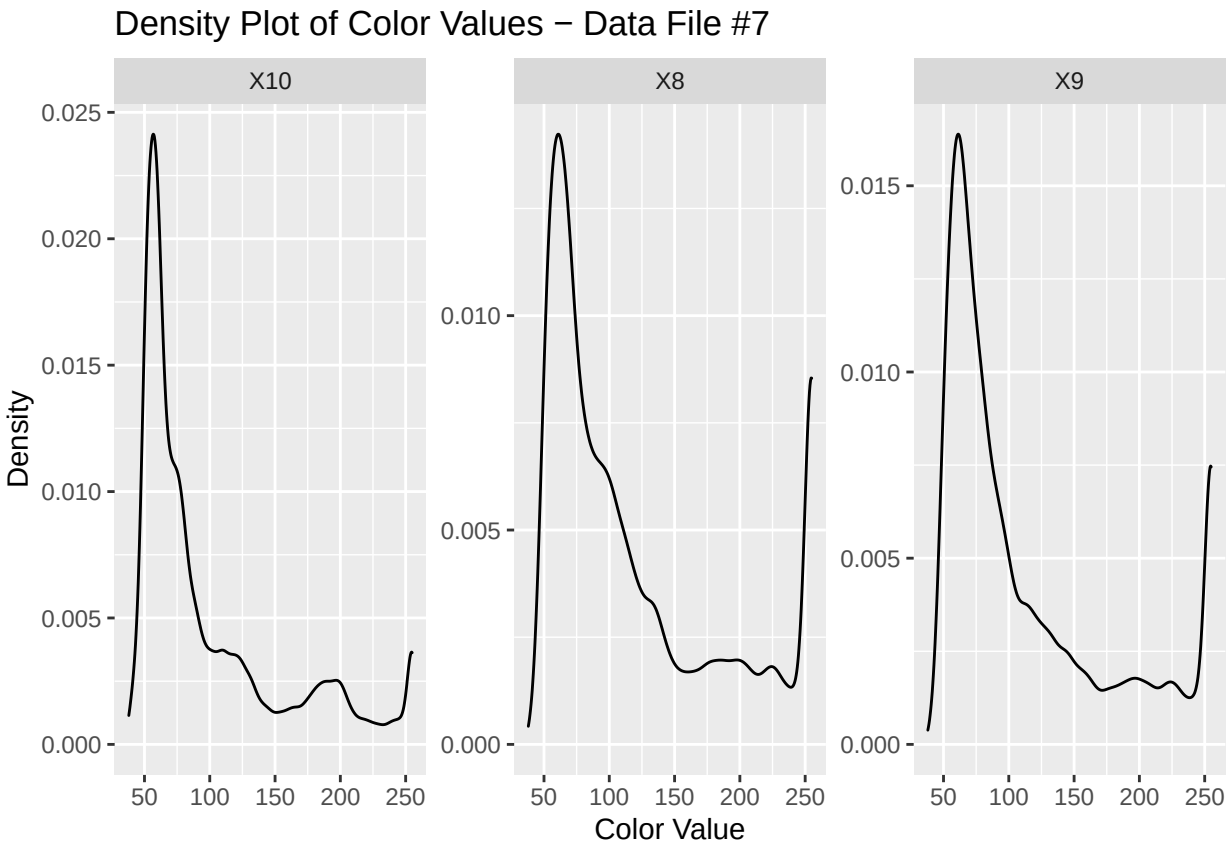
```
other2 <- other2%>%
  rename(Red=X8, Green=X9, Blue=X10)
```

```
#Import data file #7

other3 <- read_table('orthovnir069_ROI_NOT_Blue_Tarps.txt',
                     skip=8,col_names=FALSE)
```

```
##
## -- Column specification --------------------------------------------------------
## cols(
##   X1 = col_double(),
##   X2 = col_double(),
##   X3 = col_double(),
##   X4 = col_double(),
##   X5 = col_double(),
##   X6 = col_double(),
##   X7 = col_double(),
##   X8 = col_double(),
##   X9 = col_double(),
##   X10 = col_double()
## )
```

```
#select last three columns
other3 <- other3%>%
  select(X8, X9, X10)

#add class label, change it to factor

other3$tarp <- rep('other', nrow(other3))
other3$tarp <- factor(other3$tarp)
```

```
#Column identification for data file #7

#long pivot for side-by-side plots

long_other3 <- other3[,1:4]%>%
  pivot_longer(cols=-tarp, names_to = "Color", values_to = "Value")

#density plots with color as facet wrap
ggplot(long_other3, aes(x=Value)) +
  geom_density() +
  facet_wrap(facets='Color', ncol=4, scales = 'free_y')+
  labs(x="Color Value", y="Density",
       title = "Density Plot of Color Values - Data File #7")
```



From this set of plots, column X10 most clearly corresponds to blue, given the low density of high value points, but X8 and X9 are very similar to each other. I assigned X8 to red because it had slightly higher density at high values, and to be consistent with other files.

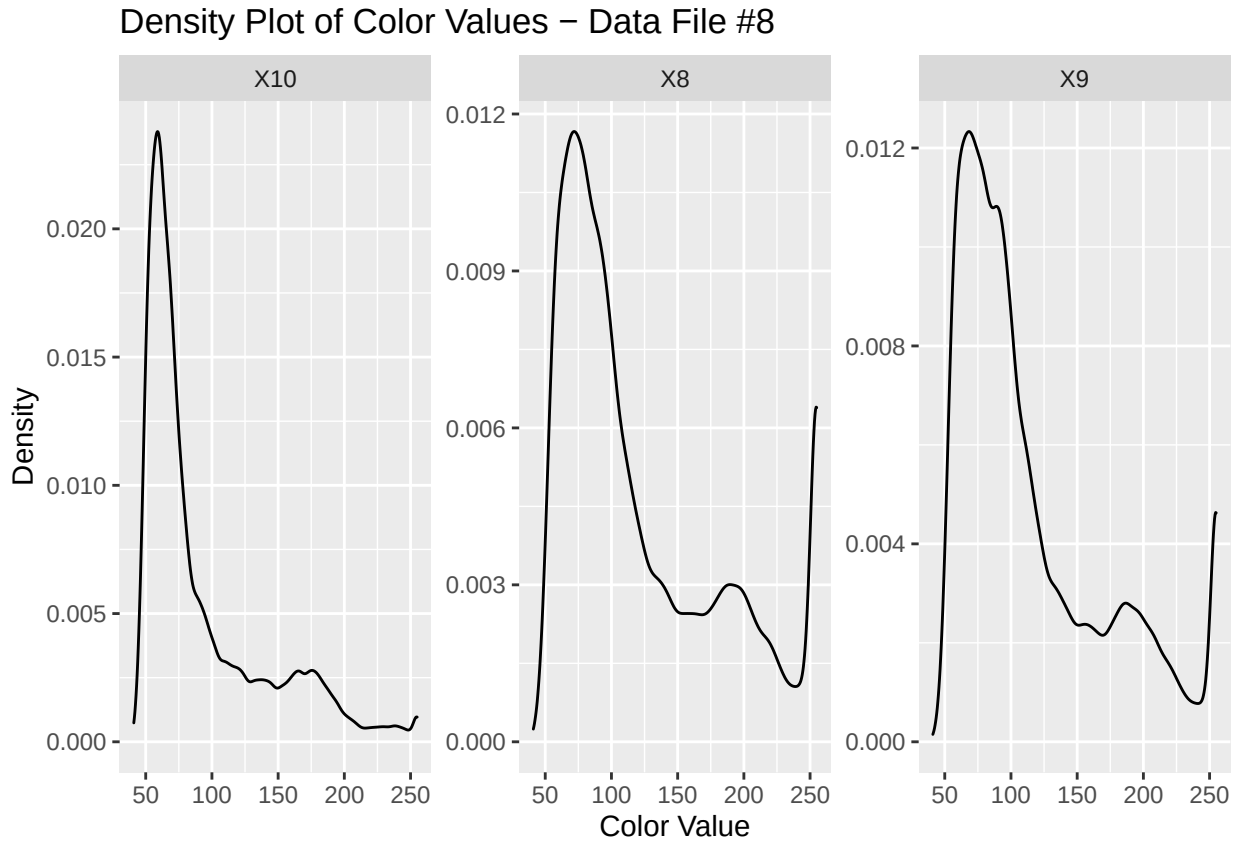
```
other3 <- other3%>%
  rename(Red=X8, Green=X9, Blue=X10)
```

```
#Import data file #8

other4 <- read_table('orthovnir067_ROI_NOT_Blue_Tarps.txt',
                     skip=8,col_names=FALSE)
```

```
##
## -- Column specification --------------------------------------------------------
## cols(
##   X1 = col_double(),
##   X2 = col_double(),
##   X3 = col_double(),
##   X4 = col_double(),
##   X5 = col_double(),
##   X6 = col_double(),
##   X7 = col_double(),
##   X8 = col_double(),
##   X9 = col_double(),
##   X10 = col_double()
## )
```

```r
#select last three columns
other4 <- other4%>%
  select(X8, X9, X10)

#add class label, change it to factor

other4$tarp <- rep('other', nrow(other4))
other4$tarp <- factor(other4$tarp)
```

```r
#Column identification for data file #8

#long pivot for side-by-side plots

long_other4 <- other4[,1:4]%>%
  pivot_longer(cols=-tarp, names_to = "Color", values_to = "Value")

#density plots with color as facet wrap
ggplot(long_other4, aes(x=Value)) +
  geom_density() +
  facet_wrap(facets='Color', ncol=4, scales = 'free_y')+
  labs(x="Color Value", y="Density",
       title = "Density Plot of Color Values - Data File #8")
```



Consistent with all other data files, I assigned column X8 to red, X9 to green, and X10 to blue, as those were the training data curves that most closely matched these distributions.

```
other4 <- other4%>%
  rename(Red=X8, Green=X9, Blue=X10)
```

With all eight files imported, cleaned up, and consistently labeled, my next step was to join these eight files into a single test data set.

```
test_final <- rbind(tarp1, tarp2, tarp3, tarp4, other1, other2, other3, other4)
```

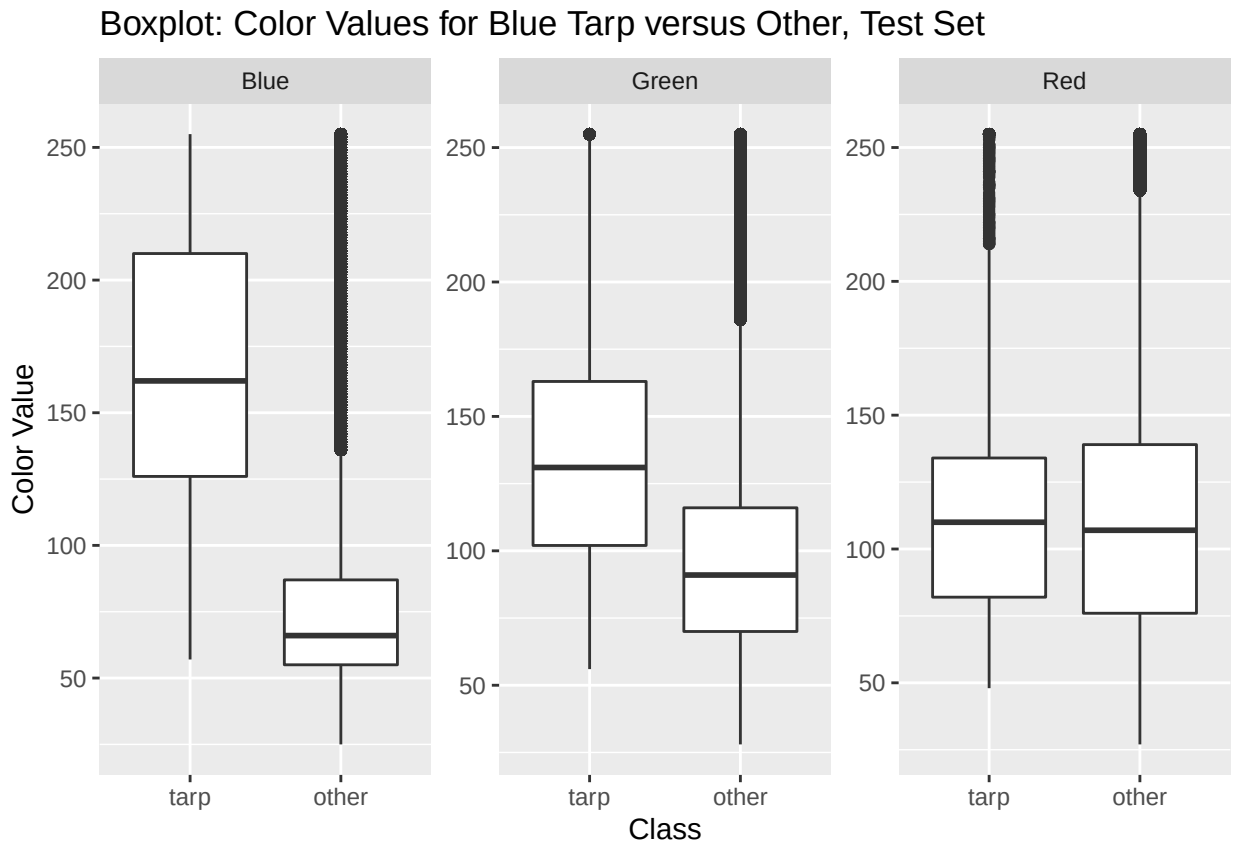
Before testing my models on this data set, I did some additional EDA.

```
summary(test_final)
```

```
##       Red             Green            Blue           tarp
##  Min.   : 27.0   Min.   : 28.0   Min.   : 25.00   tarp :  18926
##  1st Qu.: 76.0   1st Qu.: 71.0   1st Qu.: 55.00   other:1989697
##  Median :107.0   Median : 91.0   Median : 66.00
##  Mean   :118.3   Mean   :105.5   Mean   : 82.57
##  3rd Qu.:139.0   3rd Qu.:117.0   3rd Qu.: 88.00
##  Max.   :255.0   Max.   :255.0   Max.   :255.00
```

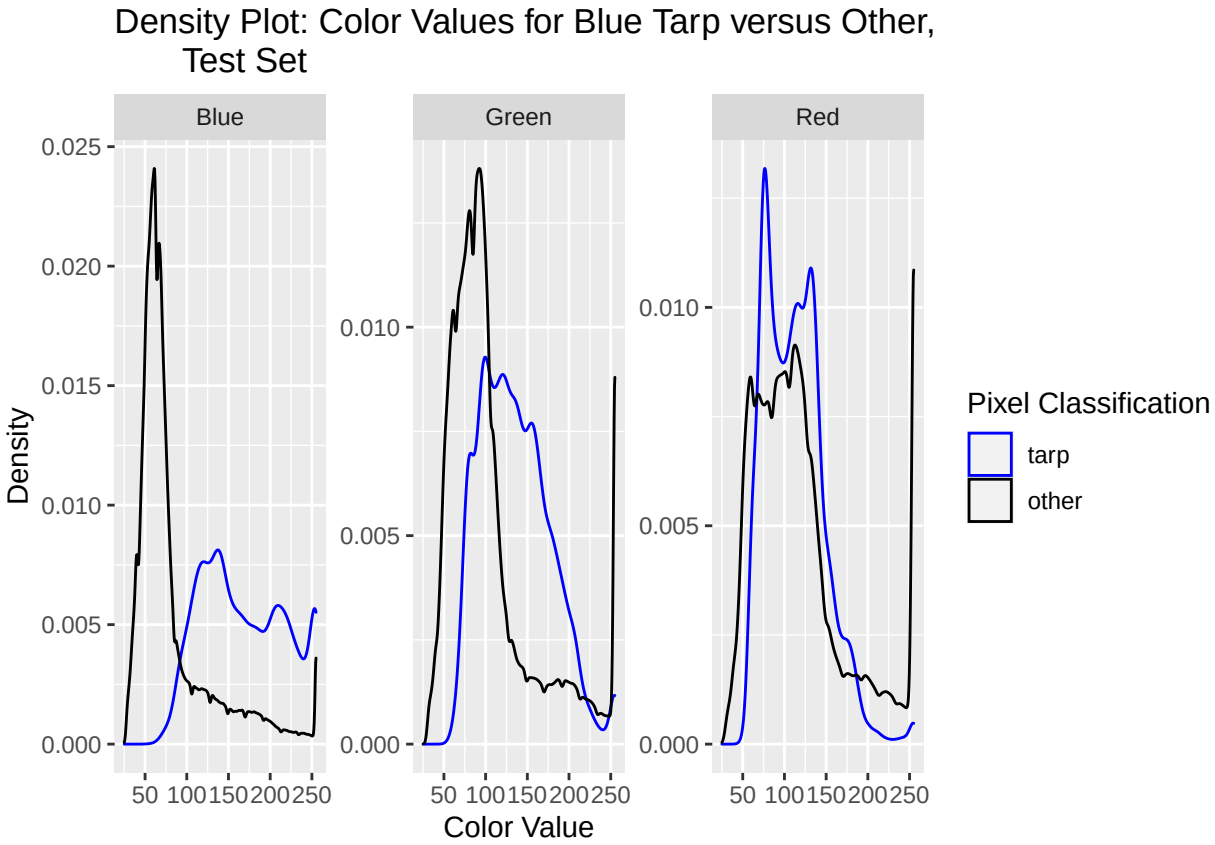
The above summary shows that 18926/(18926+1989697) = 0.94% of the points were blue tarps (compared to 3.2% in the training set). The median color values are notably lower than those in the training set: 107 versus 163 for red, 91 versus 148 for green, and 66 versus 123 for blue.

```
#long pivot for side-by-side plots

long_test <- test_final[,1:4]%>%
  pivot_longer(cols=-tarp, names_to = "Color", values_to = "Value")

#boxplot with color as facet wrap
ggplot(long_test, aes(x=tarp, y=Value)) +
  geom_boxplot() +
  facet_wrap(facets='Color', ncol=4, scales = 'free_y')+
  labs(x="Class", y="Color Value",
       title = "Boxplot: Color Values for Blue Tarp versus Other, Test Set")
```



These boxplots show that, broadly speaking, the differences between tarp pixels and non-tarp pixels show the same pattern as was seen in the training set. Median blue and green color values tend to be higher in tarp pixels than non-tarp pixels, and median red color values are roughly comparable. As mentioned above, however, we again see that median color values are lower in the test set than in the training set, and the IQR is also much narrower in the test set than in the training set.

```
#density plots with color as facet wrap
p <- ggplot(long_test, aes(x=Value, color=tarp)) +
  geom_density() +
  facet_wrap(facets='Color', ncol=4, scales = 'free_y')+
  labs(x="Color Value", y="Density",
       title = "Density Plot: Color Values for Blue Tarp versus Other,
       Test Set", color="Pixel Classification")

#manually change the color of the lines
p+scale_color_manual(values=c("blue", "black"))
```



Consistent with the density plots I generated for each individual test data file, these plots, which combine all the test data together, show some substantial differences as compared to the training set. While blue values have some overlap between tarp and non-tarp pixels, they are reasonably well separated between the two classes, as was also the case with the training set. The density curves for red overlap almost completely in the test set, and this was not the case in the training set. This leads me to believe that the models will underperform in the test data compared to the training data, given that red color values may be a poor predictor of class in the test set. Green values had considerable overlap in the training set, and this continues to be the case in the test set, though the shape of the density curves are somewhat different and are shifted toward lower values that in the training set.

Model Testing

To test the seven models I built on the holdout data, I wrote a function that takes the model and predetermined threshold as inputs, and using caret's confusionMatrix function, it generates a confusion matrix and returns accuracy, sensitivity, the false positive rate, and precision for the selected threshold. It also plots a ROC curve and computes the AUC for the model, which are more general model performance metrics independent of the selected threshold.

```
#function to test models
test_model <- function(model, threshold){
  prediction <- as.factor(ifelse(predict(model,
                                         newdata=test_final,
                                         type='prob')$tarp>threshold,
                                 'tarp', 'other'))

  cm <- confusionMatrix(data=prediction, reference=test_final$tarp)
```

```
  print(cm$table)

  #AUC
  preds <- predict(model, newdata=test_final, type='prob')$tarp
  rates <- prediction(preds, test_final$tarp)
  auc_val <- performance(rates, measure='auc')
  auc <- auc_val@y.values[[1]]

  #ROC (with downsampling, because test set is huge)
  roc_result <- performance(rates, measure='tpr', x.measure='fpr')
  plot(roc_result,  downsampling = 0.001, main='ROC Curve')
  lines(x=c(0,1), y=c(0,1), col='red')


  return(tibble(AUC=auc,
                threshold = threshold,
                accuracy = cm$overall['Accuracy'],
                sensitivity = cm$byClass['Sensitivity'],
                FPR=(1-cm$byClass['Specificity']),
                precision = cm$byClass['Precision']))

}
```

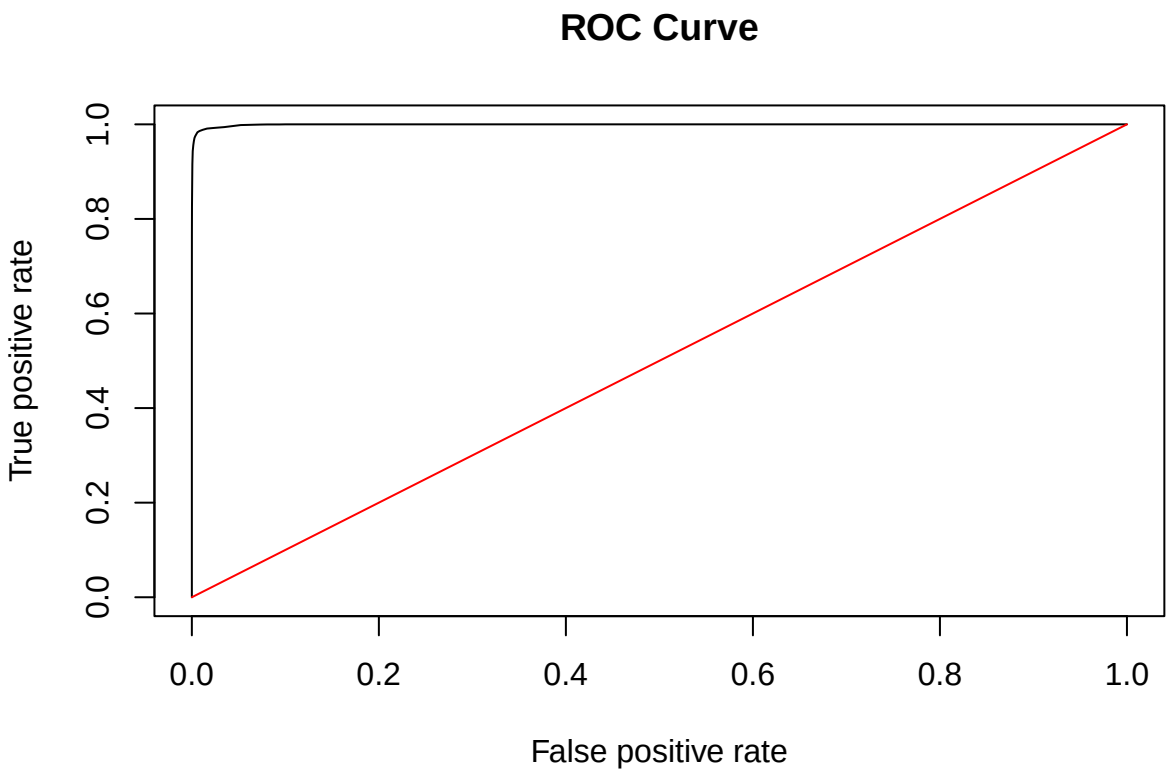
I also set up an empty data frame to store important metrics capturing each model's performance on the test data.

```
columns <- c("Model", "AUC", "Threshold",
             "Accuracy", "Sensitivity/TPR", "FPR", "Precision")
test_table <- data.frame(matrix(nrow = 0, ncol = length(columns)))
colnames(test_table) <- columns
```

Using each of the seven candidate models and the final, aggregated holdout data set, I tested each model's performance on the test data set using the probability threshold that I specified as optimal for each model. I captured key performance metrics for each model in my test performance data frame.

```
#Test the logistic model
test_model(logistic, 0.05)
```

```
##            Reference
## Prediction    tarp   other
##      tarp    18924  193475
##      other       2 1796222
```




```
## # A tibble: 1 x 6
##     AUC threshold accuracy sensitivity    FPR precision
##   <dbl>     <dbl>    <dbl>       <dbl>  <dbl>     <dbl>
## 1 0.999      0.05    0.904        1.00 0.0972    0.0891
```

As the ROC and AUC show, this model performed very well on the test data set, with a ROC curve that came very close to the upper left corner and a AUC value of 0.999. At the threshold I set using training data (0.05), overall accuracy was 90.38% and the sensitivity was 99.99%. Though the overall accuracy was lower than on the training data, the sensitivity (99.99%) was better, indicating that virtually no tarps were missed. Unfortunately, the false positive rate and precision were substantially worse than on the training set, indicating that many points (more than 193,000!) were misidentified as tarps. The AUC and ROC suggest that this model has strong potential, and that it would be worthwhile to further tweak the threshold to reduce the number of false positives.

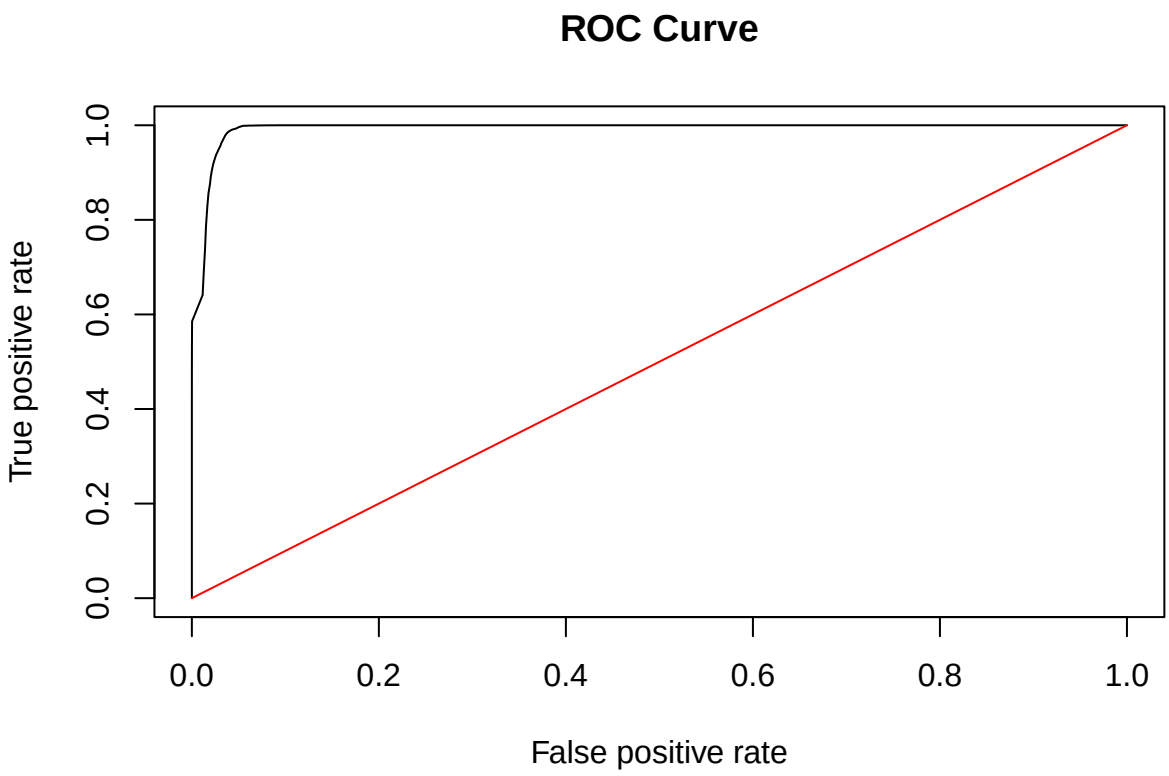
I saved key metrics in my summary table.

```
#insert all key metrics into test data frame

logistic_test_info <- data.frame("Logistic regression",
                      0.999, 0.05, 0.904,   0.9999, 0.097,  0.089)

colnames(logistic_test_info) <- columns
test_table <- rbind(test_table, logistic_test_info)
```

```
#Test the LDA model
test_model(lda, 0.005)
```

```
##           Reference
```

```
## Prediction    tarp   other
##      tarp    18565   73047
##      other     361 1916650
```




```
## # A tibble: 1 x 6
##     AUC threshold accuracy sensitivity    FPR precision
##   <dbl>     <dbl>    <dbl>       <dbl>  <dbl>     <dbl>
## 1 0.992     0.005    0.963       0.981 0.0367     0.203
```

The LDA's AUC (0.992) was slightly lower than the logistic regression's, and its ROC curve did not come as close to the upper left corner. At the selected threshold of 0.005, overall accuracy was 96.35%, and sensitivity was 98.09%. The precision was still substantially lower than in the training set, at 20.26%, again indicated many points falsely misclassified as tarps.

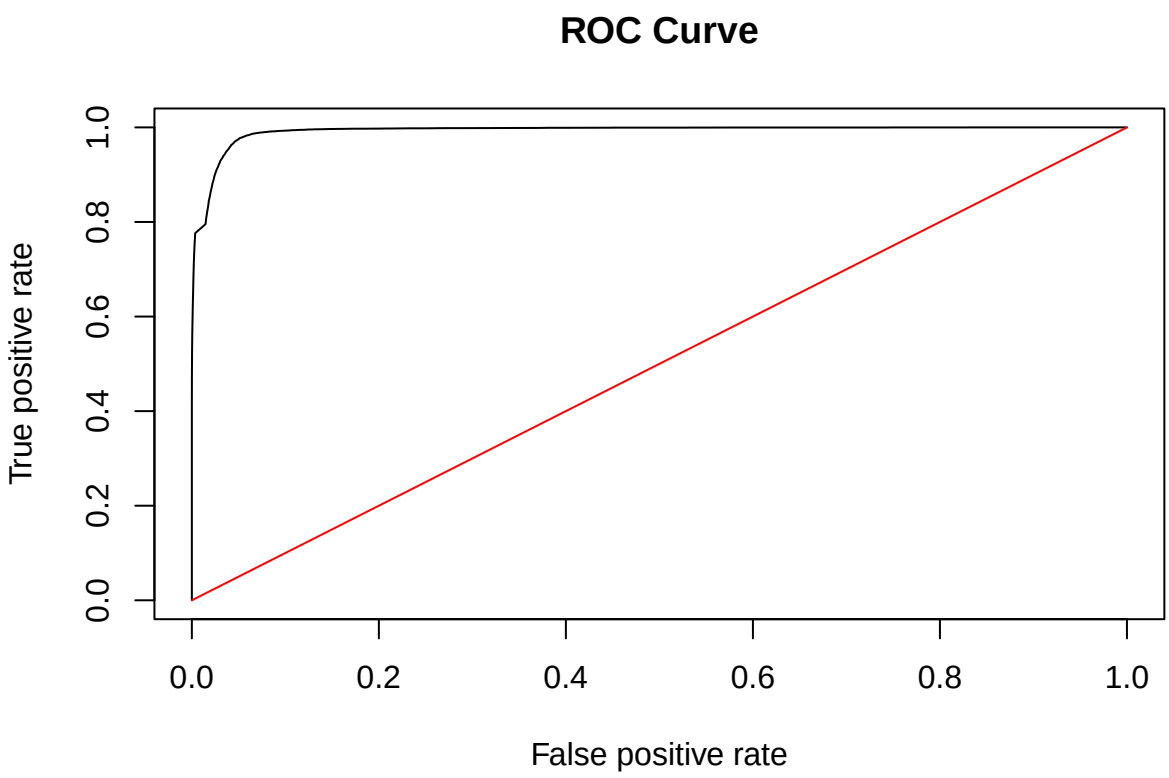
I saved these metrics in my test performance data frame.

```
#insert all key metrics into test data frame

lda_test_info <- data.frame("LDA",
                      0.992, 0.005, 0.963,  0.981,  0.037,  0.201)

colnames(lda_test_info) <- columns
test_table <- rbind(test_table, lda_test_info)
```

```
#Test the QDA model
test_model(qda, 0.025)
```

```
##           Reference
## Prediction    tarp   other
```

```
##       tarp    17472   58242
##       other    1454 1931455
```




```
## # A tibble: 1 x 6
##     AUC threshold accuracy sensitivity    FPR precision
##   <dbl>     <dbl>    <dbl>       <dbl>  <dbl>     <dbl>
## 1 0.992     0.025    0.970       0.923 0.0293     0.231
```

The QDA's AUC is 0.992, and similar to the LDA ROC, it does not come as close to the upper left corner of the plot as compared to the logistic regression. The accuracy at the selected threshold of 0.025 is the highest we have seen thus far at 97.03%, though the sensitivity is much lower (92.32%) than the other two models. As the confusion matrix shows, this model misses quite a few tarp points, but also does not identify as many false positives.

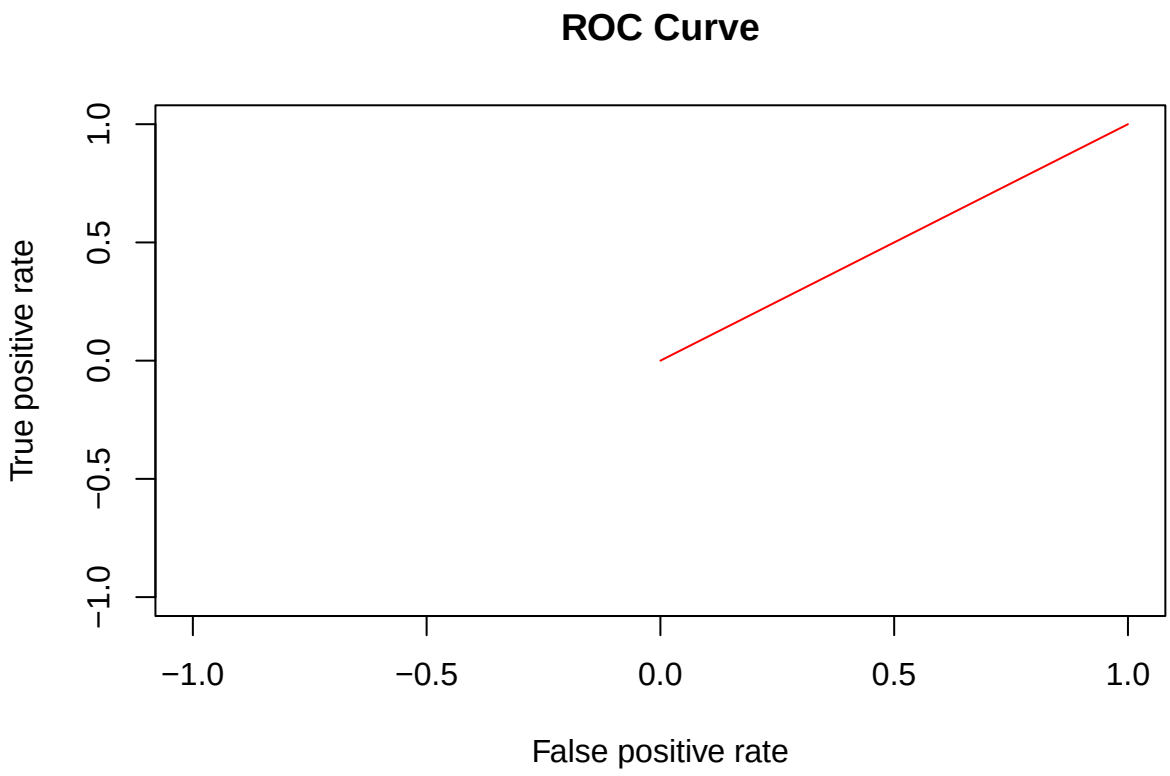
I saved these metrics in my test performance data frame.

```
#insert all key metrics into test data frame

qda_test_info <- data.frame("QDA",
                      0.992, 0.025, 0.970,  0.923,  0.029,  0.231)

colnames(qda_test_info) <- columns
test_table <- rbind(test_table, qda_test_info)
```

```
#determine number of cores and initialize cluster
no_cores <- detectCores()-1
cl <- makePSOCKcluster(no_cores)
registerDoParallel(cl)
```

```
#Test the KNN model
test_model(knn_final, 0.1)
```

```
##           Reference
## Prediction    tarp   other
##      tarp    17832   27664
##      other    1094 1962033
```



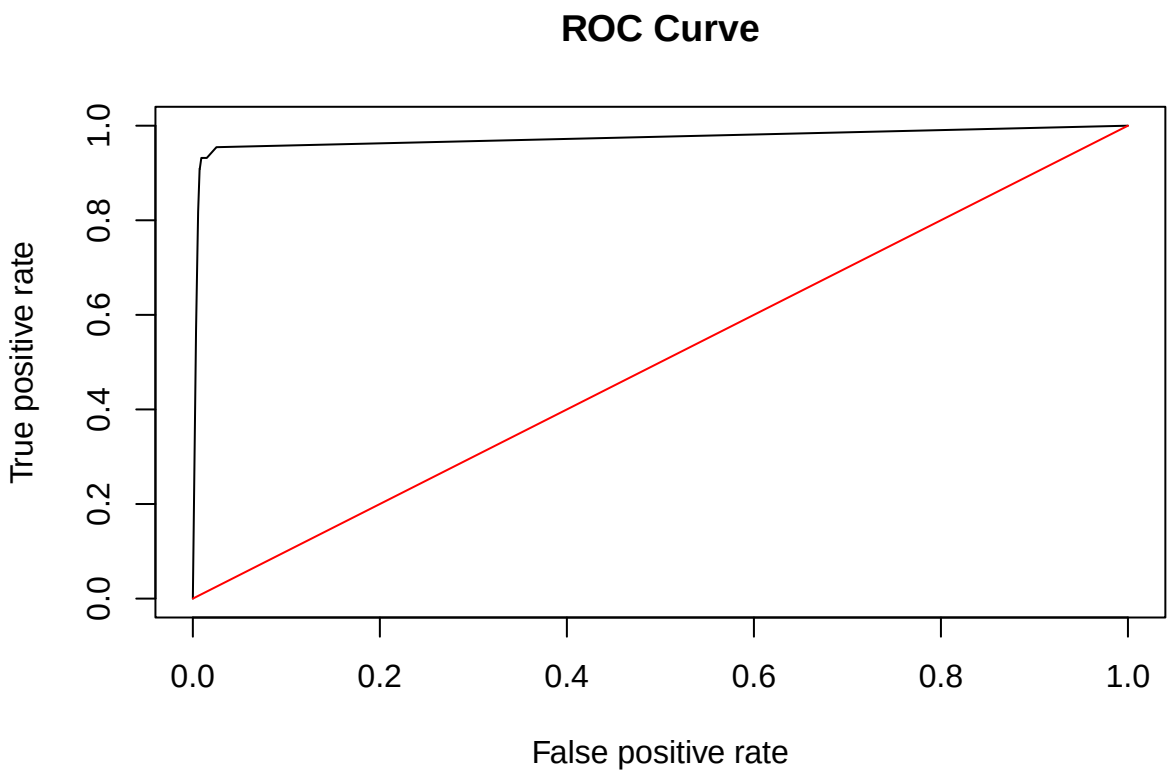

```
## # A tibble: 1 x 6
##     AUC threshold accuracy sensitivity    FPR precision
##   <dbl>     <dbl>    <dbl>       <dbl>  <dbl>     <dbl>
## 1 0.970       0.1    0.986       0.942 0.0139     0.392
```

```
#stop cluster
stopCluster(cl)
registerDoSEQ()
```

The ROC curve did not render properly for this model, despite working well for logistic regression, LDA, QDA, ridge regression, and SVM. In troubleshooting the problem, I discovered that the downsampling argument of the plot function that did not work with certain models (KNN and random forests). To get around this issue and generate a usable ROC curve, I simply took a random sample of the holdout set and used this smaller data set to estimate a ROC curve.

```
#create a mini test set of 5000 points

set.seed(1)
sample2 <- sample(nrow(test_final), 5000)
mini_test <- test_final[sample2,]
```

```
#ROC curve
preds <- predict(knn_final, newdata=mini_test, type='prob')$tarp
rates <- prediction(preds, mini_test$tarp)
roc_result <- performance(rates, measure='tpr', x.measure='fpr')
plot(roc_result, main='ROC Curve')
lines(x=c(0,1), y=c(0,1), col='red')
```



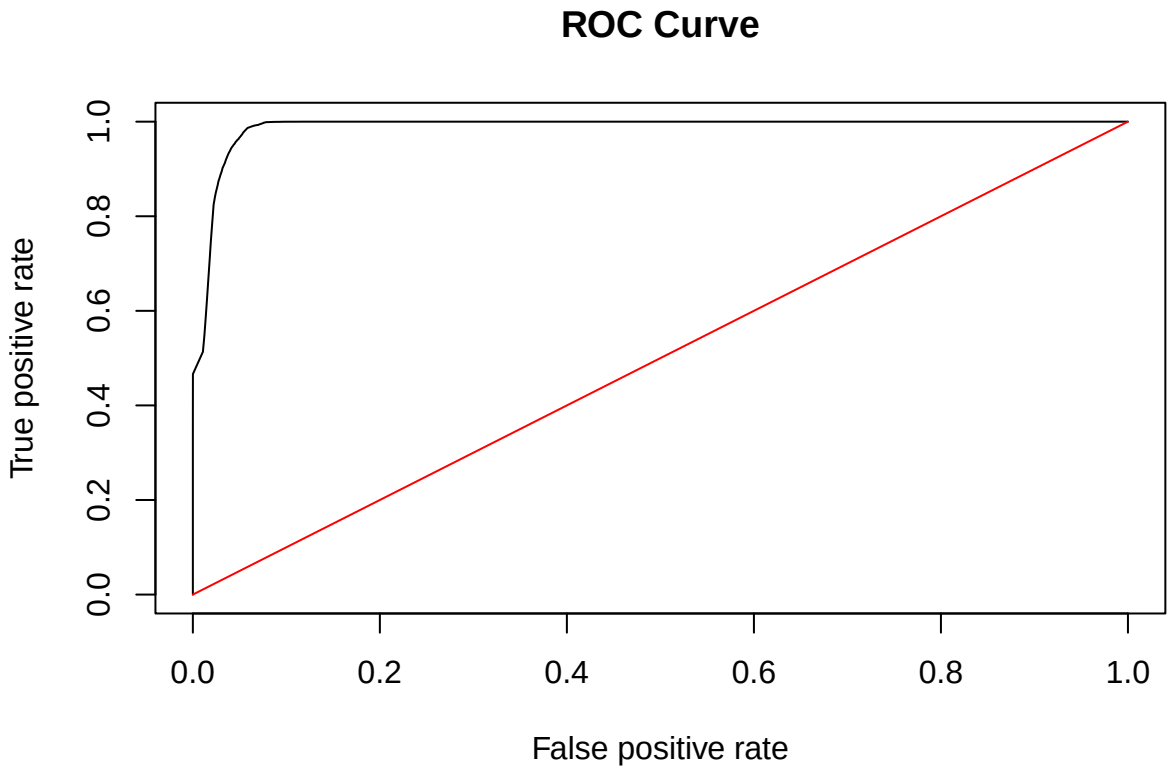
The AUC value of 0.970 was a bit lower for this model than for others thus far, and the ROC curve did not fit as neatly into the upper left corner of the plot as for logistic regression, which is the strongest (so far) in terms of ROC/AUC. The accuracy at the selected threshold of 0.1 was 98.57%, the sensitivity was 94.22%, and importantly, the precision was higher than other models, at 39.19%. There were some missed tarp pixels, but of those identified as tarps, a higher proportion were actually tarps. The precision is still far below what was seen with the training set, however. Though the KNN model was the top performer with the training data, its AUC did not match the overall performance of the logistic regression model when used with the test data.

I saved these metrics in my test performance data frame.

```
#insert all key metrics into test data frame

knn_test_info <- data.frame("KNN",
                      0.970, 0.1, 0.986,    0.942,  0.014,  0.392)

colnames(knn_test_info) <- columns
test_table <- rbind(test_table, knn_test_info)
```

```
#Test the ridge model
test_model(ridge_final, 0.1)
```

```
##           Reference
## Prediction    tarp   other
##      tarp    15823   45805
##      other    3103 1943892
```




```
## # A tibble: 1 x 6
##     AUC threshold accuracy sensitivity    FPR precision
##   <dbl>     <dbl>    <dbl>       <dbl>  <dbl>     <dbl>
## 1 0.988       0.1    0.976       0.836 0.0230     0.257
```

The AUC for the ridge regression model (0.988) is lower than for logistic regression, LDA, and QDA, but higher than for the KNN model. The corresponding ROC curve is farther from the upper left corner than was seen in logistic regression, LDA, and QDA. At the threshold of 0.1, sensitivity was only 83.60%, indicating that many tarp pixels were missed, and precision (25.68%) was not much better than in the logistic regression, LDA, and QDA. The ridge regression model's relatively low sensitivity was not surprising, given that this was also the case with the training data.

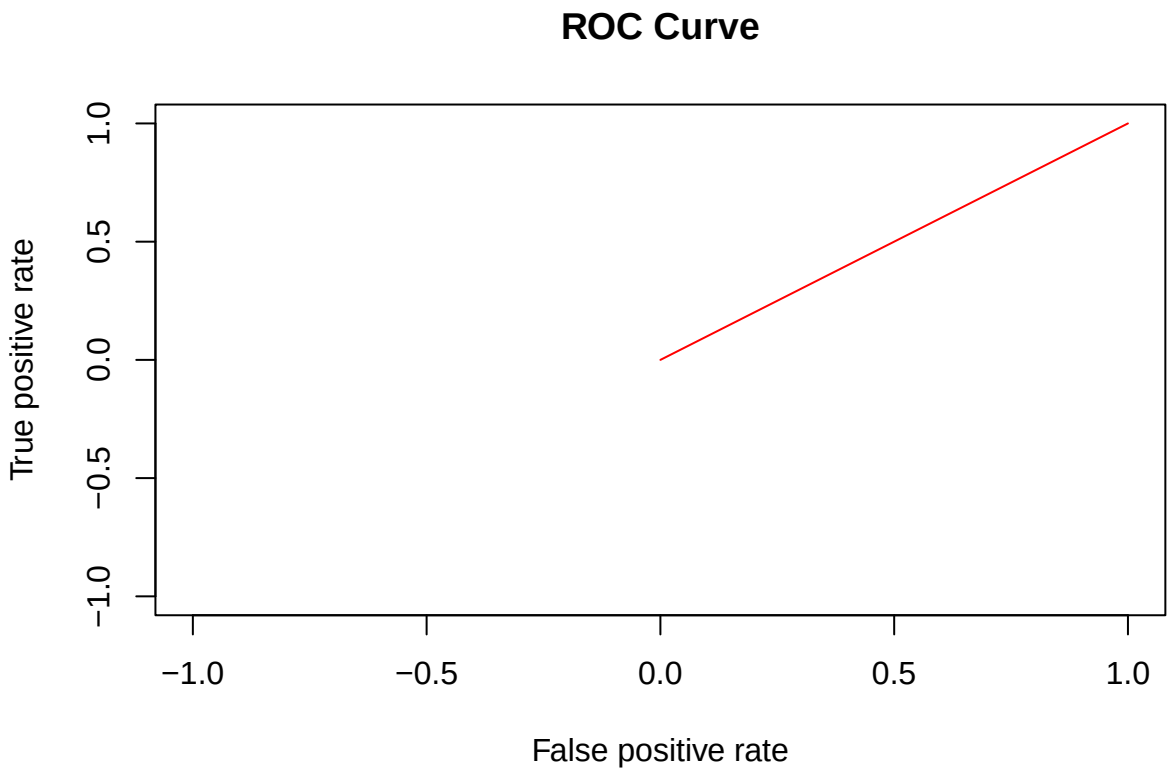
I saved these metrics in my test performance data frame.

```
#insert all key metrics into test data frame

ridge_test_info <- data.frame("Ridge regression",
                      0.988, 0.1, 0.976,    0.836,  0.023,  0.257)

colnames(ridge_test_info) <- columns
```

```
test_table <- rbind(test_table, ridge_test_info)
```

```
#Test the random forest model
test_model(rf_final, 0.15)
```

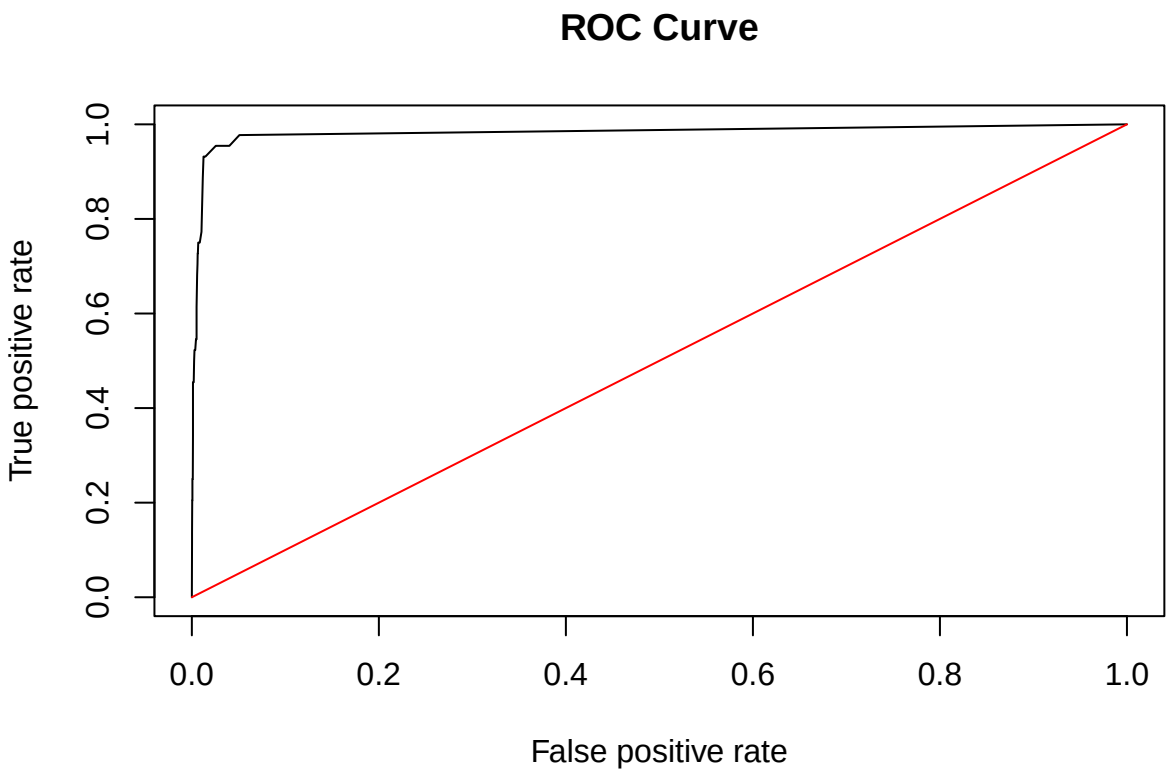
```
##            Reference
## Prediction    tarp   other
##      tarp    18374   49330
##      other     552 1940367
```




```
## # A tibble: 1 x 6
##     AUC threshold accuracy sensitivity    FPR precision
##   <dbl>     <dbl>    <dbl>       <dbl>  <dbl>     <dbl>
## 1 0.987      0.15    0.975       0.971 0.0248     0.271
```

As was true with the KNN model, the downsampling argument of the plot function caused the ROC curve to render improperly, and I used the same method described above to generate a usable ROC curve.

```
#ROC curve
preds <- predict(rf_final, newdata=mini_test, type='prob')$tarp
rates <- prediction(preds, mini_test$tarp)
roc_result <- performance(rates, measure='tpr', x.measure='fpr')
plot(roc_result, main='ROC Curve')
lines(x=c(0,1), y=c(0,1), col='red')
```



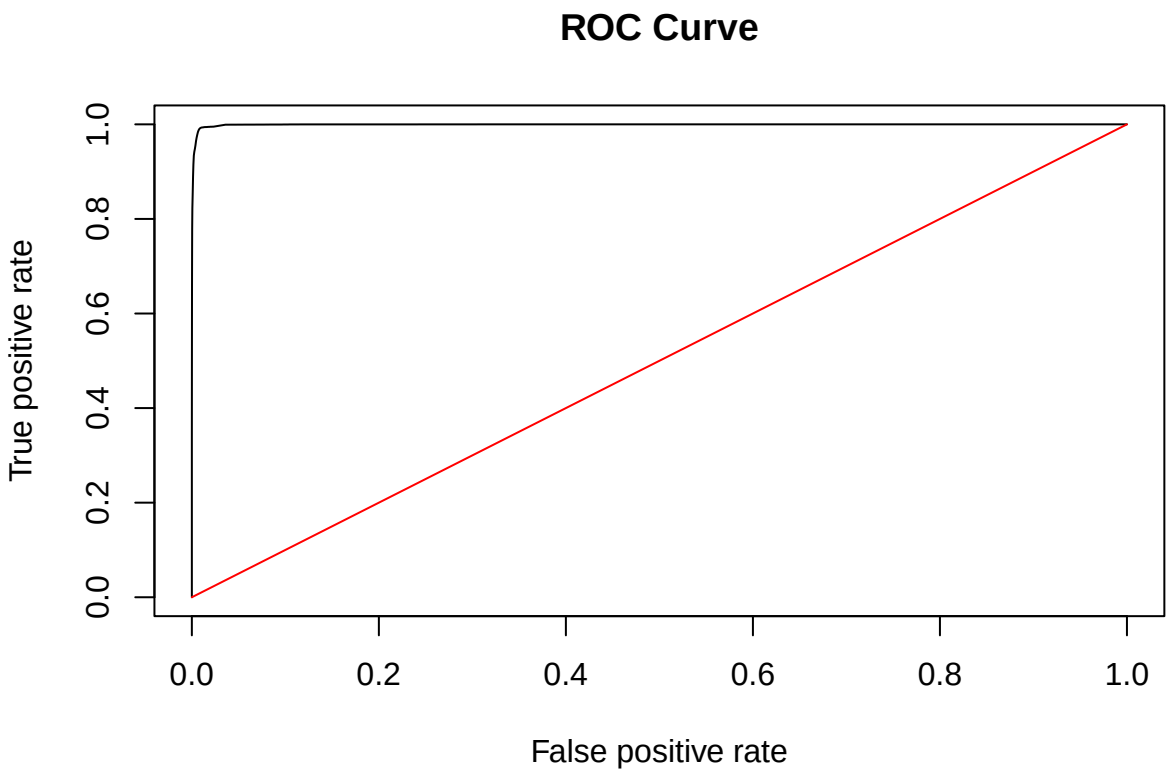
The AUC of 0.987 is comparable to the AUC of ridge regression, and is relatively low compared to better performing models (though not as low as the KNN model). The ROC curve cuts away a bit from the upper left corner. At the selected threshold of 0.15, the model's accuracy was 97.52%, sensitivity was 97.08%, and precision was 27.14%.

I saved these metrics in my test performance data frame.

```
#insert all key metrics into test data frame

rf_test_info <- data.frame("Random forest",
                      0.987, 0.15, 0.975,   0.971,  0.025,  0.271)

colnames(rf_test_info) <- columns
test_table <- rbind(test_table, rf_test_info)
```

```
#determine number of cores and initialize cluster
no_cores <- detectCores()-1
cl <- makePSOCKcluster(no_cores)
registerDoParallel(cl)

#Test the SVM model
test_model(svm_poly_final, 0.1)
```

```
##            Reference
## Prediction    tarp   other
##      tarp    18832   42995
##      other      94 1946702
```




```
## # A tibble: 1 x 6
##     AUC threshold accuracy sensitivity    FPR precision
##   <dbl>     <dbl>    <dbl>       <dbl>  <dbl>     <dbl>
## 1 0.999       0.1    0.979       0.995 0.0216     0.305
```

```
#stop cluster
stopCluster(cl)
registerDoSEQ()
```

This model's ROC curve came very close to the upper left corner of the plot, and the corresponding AUC value was 0.999, the same as the logistic regression model. At the selected threshold of 0.1, the sensitivity was 99.50% and the corresponding precision was 30.46%. For this level of sensitivity, the precision appears much better than the logistic regression, though as noted before, it would be worthwhile to further explore different thresholds for the logistic regession model.

I saved these metrics in my test performance data frame.

```
#insert all key metrics into test data frame

svm_test_info <- data.frame("Support vector machine (polynomial kernel)",
                      0.999, 0.1, 0.979,    0.995,  0.022,  0.305)

colnames(svm_test_info) <- columns
test_table <- rbind(test_table, svm_test_info)
```

Holdout Set Performance Table

A summary of model performance is shown in the below table. Tuning parameters for each model are in the original table (above). Accuracy, Sensitivity, FPR, and Precision are for the given threshold value.

```
test_table
```

```
##                                       Model   AUC Threshold Accuracy
## 1                       Logistic regression 0.999     0.050    0.904
## 2                                       LDA 0.992     0.005    0.963
## 3                                       QDA 0.992     0.025    0.970
## 4                                       KNN 0.970     0.100    0.986
## 5                          Ridge regression 0.988     0.100    0.976
## 6                             Random forest 0.987     0.150    0.975
## 7 Support vector machine (polynomial kernel) 0.999     0.100    0.979
##   Sensitivity/TPR   FPR Precision
## 1          0.9999 0.097     0.089
## 2          0.9810 0.037     0.201
## 3          0.9230 0.029     0.231
## 4          0.9420 0.014     0.392
## 5          0.8360 0.023     0.257
## 6          0.9710 0.025     0.271
## 7          0.9950 0.022     0.305
```

Conclusions

When I trained the seven models described above, all had fairly high and similar AUC values, ranging from a low of 0.980 (ridge regression) to a high of 0.9997 (SVM model with polynomial kernel). Based on AUC values alone, the top performer was the SVM model, followed very closely by the random forest (0.9990), and then by the logistic regression, QDA, and KNN models (all 0.998). With such a tiny amount of variation among these top performers, the AUC alone seemed insufficient to distinguish them, and I largely relied on sensitivity and precision to further judge their performance, and to select a threshold for each model.

As noted several times in the discussion throughout this report, sensitivity and precision seemed like logical metrics to consider, since they most directly reflect aid workers' ability to locate people needing help (sensitivity) without simultaneously being overwhelmed by false positives (precision). I also considered the related metric of detection prevalence (how many total pixels would be identified as tarps), since too high a number would easily overwhelm limited people and resources. Without knowing exactly how personnel and resources might be constrained, I attempted to balance these factors, and roughly aimed, where possible, for sensitivity of >97-98%, with corresponding precision >70%. However, threshold choices could be very different with additional information. For example, if misidentifying pixels as tarps placed undue strain on a very limited number of workers, and thereby prevented them from assessing pixels that were more unambiguously tarps, I would choose higher threshold values. Alternatively, if there were plenty of resources available to check out every pixel identified as a tarp, I might have further lowered the threshold for some models and increase the sensitivity further.

Based on these criteria, the SVM model, random forest, and K-nearest neighbor models were the strongest performers with the training data. While logistic regression and QDA were also able to achieve a high level of sensitivity at certain thresholds, the SVM, random forest, and KNN models were able to do so with the highest corresponding precision. The SVM model's strong performance was likely due to the fact that the classes were reasonably well separated, as evident in the scatter plots. The original data set had only three predictors, so a polynomial kernel added some additional dimensions and flexibility, and ultimely helped in finding a highly accurate separating hyperplane. The random forest model's success was also unsurprising, given that the three predictors were highly correlated to each other, and random forests tend to perform well when this is the case. The SVM and random forest models also had no requirement about the distribution of the input data, which may have offered an advantage in this case. The strong performance of the KNN

model was likely attributable to the large number of observations; KNN requires a large training set, and may not have performed as well on a smaller data set.

The relatively poor performance of LDA relative to QDA with the training data was unsurprising, given that the variance was not constant between classes, and this is required for LDA. It is similarly unsurprising that ridge regression performed relatively poorly, since this method is meant to reduce the contributions of less significant predictors. In this case, since there were so few predictors and they were all important, ridge regression offered little advantage.

When I tested the models on the holdout data set, the first thing I noticed was that precision was comparatively poor across all models. Whereas precision was 88-91% among top performing models in the training set, it dropped to under 40% in the holdout set, and for some thresholds and models, was substantially lower than this. Given the size of the holdout set (>2 million pixels) and the low precision of the models, there were tens of thousands of false positives in some cases, and given my concerns about constrained resources, this seems unacceptably high. Thus, in a second iteration of training, I would likely consider raising the thresholds to reduce the number of false positives at the expense of some sensitivity. Nonetheless, in the present analysis, I used the thresholds I had identified during training to evaluate each model, along with the models' ROC curves and AUC scores.

Looking at AUC scores, there was marginally more variation with the holdout data than was seen with the training data, and values did not follow the same pattern as was seen with training data. With the training set, the SVM and random forest were the top performers, followed by the logistic regression, QDA, and KNN models. With the holdout set, the two best models were the logistic regression and the SVM model (both 0.999), followed by the LDA and QDA models (both 0.992). The random forest had an AUC of 0.987 - comparable to the ridge regression - and the KNN model had the lowest AUC of all the models, at 0.970.

Looking more carefully at each model's performance with the threshold I identified, the logistic regression had outstanding sensitivity, detecting 99.99% of tarp pixels and exceeding its performance on the taining set, but had an abysmally low precision of only 8.9%. This would correspond to an untenable number of false positive pixels, but given that the AUC value for this model was so high, and it thus seemed to be a strong model overall, I think it would be worthwhile to continue tweaking this model with some different threshold values before rejecting it outright. The other model with the highest AUC score, the SVM model with the polynomial kernel, attained 99.5% sensitivity, which also exceeded its performance with the training data. Though the sensitivity is only slightly lower than the logistic regression, this model has a much improved precision of 30.5%. As noted above, this precision is perhaps still too low, but in further testing and tweaking of thresholds, I speculate that this model would offer the highet sensitivity for a given level of precision. It is therefore my top model, and the one I would choose to use in a real-world situation.

The KNN and random forest models - both strong contenders based on the training sets - performed decently with the holdout data, though their sensitivity was slightly lower than was seen with the training data. While their sensitivities were not as high as the logistic regression or the SVM, the KNN model had considerably better precision (39.2%), and the random forest model had precision of 27.1% - not as strong as the SVM, but nowhere close to as bad as the logistic regression. The differences between sensitivity and precision among these mid-range performers are likely due in part to varying threshold values.

In considering the performance of these models, it is worthwhile to note that there were some substantial differences between the training and testing data sets, which became apparent during the import and density plot assessment for the test data files. Some files contined data that were more similar to the training set than others, but generally speaking, color values were lower in the holdout set than in the training set, and the density plots for each color looked different for the training and holdout sets. For instance, red color values were a valuable predictor of class in the training data, but classes had much more overlap in the test data, and red color value was therefore not as informative as a predictor for the holdout set. Though some of the models performed very well with the testing data despite these differences, I speculate that the lower overall precision may be, in part, due to differently distributed training and holdout sets. To mitigate this issue, one possible strategy I could try would be to engineer a feature that represents each color value as a proportion of the total for that pixel; that is, for each pixel, what proportion of its total color score is red, green, and blue? This may help to control for the overall darkness of an image. I might also consider whether

different models would be needed to assess darker versus brighter images.

In addition to changes mentioned in the above paragraphs, there are several other ideas I have to further improve these models. I could experiment more with adjusting the number of parameters. Early on, I did try different subsets of predictors, but quickly discovered that all three color values were important, so reducing to fewer predictors did not make sense. I also experimented with interactions, but the improvement in model performance was marginal at best. I did not attempt to build any polynomial terms into the models, and this could be an avenue for further exploration. I could also try a broader range of tuning parameters, particularly in the SVM models, which show the most promise based on current results. To further enhance the model's ability to correctly identify tarps, it would be interesting to factor a pixel's surrounding pixels into a model. A pixel that is surrounded by other "tarp" pixels is more likely to itself be a tarp than a pixel whose color values identify it as a tarp, but which is surrounded by "non-tarp" pixels. Finally, it would be interesting to try an approach that is known to excel with image recognition, such as a neural network.