Disaster Relief Project: Part II

Jesse Katz, wgq5tw

Aug 09, 2021

DS 6030 | Spring 2021 | University of Virginia

---

1 Introduction

This report examines imagery data collected during the relief efforts made in Haiti in response to the 2010 earthquake (over 63,000 distinct images). The goal of the analysis is to determine which classification method - Logistic Regression, LDA, KNN, QDA, K-nearest neighbors, Penalized Logistic Regression, Random Forest, or Support Vector Machines - performs the best at predicting where displaced persons living in makeshift shelters were located. The imagery data was collected as part of the rescue effort by a team from the Rochester Institute of Technology using an aircraft to collect high resolution geo-referenced imagery. It was known that the people displaced by the earthquake were creating temporary shelters using blue tarps, and these blue tarps would be good indicators of where the displaced persons were. The classification methods examined and compared in this analysis were evaluating prediction performance on whether or not a blue tarp was present in each images. The idea is that a successful algorithm at predicting whether or not a blue tarp was present in an image, could be used in efforts to locate displaced persons and communicate those locations to rescue workers so they could help those who needed it (or need in the future) in time. The classification methods are compared and tested using 10-fold cross validation techniques via the caret package in R and compared in terms of test prediction performance using accuracy, precision, True Positive Rate (tpr), False Positive Rate, and Area under the ROC curve on a set of Hold Out Data (instead of the full training dataset).

2 Training Data / EDA

Packages are loaded below.

```
# Load Required Packages
library(knitr)
library(tidyverse)
library(broom)
library(tidymodels)
library(randomForest)
library(dplyr)
library(tidyr)
library(ggplot2)
library(ggthemes)
library(GGally)
library(carat)
library(MASS)
library(class)
library(ROCR)
library(MLeval)
library(ggpubr)
library(kableExtra)
library(scatterplot3d)
```

The Data is then loaded using the read_csv function

```
#-- Load Data
data.dir <- "C:/Users/jkatz/Desktop/UVA/Semester 2/DS 6030/Project"
data <- read_csv(file.path(data.dir, "HaitiPixels.csv"))
```

```
head(data)
```

```
#> [38;5;246m# A tibble: 6 x 4[39m
#>   Class        Red Green  Blue
#>   [3m[38;5;246m<chr>[39m[23m      [3m[38;5;246m<dbl>[39m[23m [3m[38;5;246m<dbl>[39m[23m [3m[38;5;246m<dbl>[39m[23m [3
m[38;5;246m<dbl>[39m[23m
#> [38;5;250m1[39m Vegetation    64    67    50
#> [38;5;250m2[39m Vegetation    64    67    50
#> [38;5;250m3[39m Vegetation    64    66    49
#> [38;5;250m4[39m Vegetation    75    82    53
#> [38;5;250m5[39m Vegetation    74    82    54
#> [38;5;250m6[39m Vegetation    72    76    52
```

```
#What are the values in the data
unique(data$Class)
```

```
#> [1] "Vegetation"       "Soil"             "Rooftop"          "Various Non-Tarp"
#> [5] "Blue Tarp"
```

Looking at the data, it seems we have 5 distinct image classifications: Vegetation, Soil, Rooftop, Various Non-Tarp, and finally Blue Tarp. Because the goal of this analysis is to evaluate prediction performance of different classification methods on whether Blue Tarps are present in the images or not, we can reduce the class field into a binary class: Blue Tarp equals 1 and Other equals 0. The data is then turned into factor to ensure properly identified as a classification problem by R during analysis.

```
#-- Fix data set

#we are looking to predict blue tarps so, lets make a new column, if blue tarp 1, else 0
data$Tarp_dummy <- ifelse(data$Class == "Blue Tarp", 1, 0)

#check to see if a factor and if not convert
is.numeric(data$Tarp_dummy)
```

```
#> [1] TRUE
```

```
data$Tarp_dummy<-factor(data$Tarp_dummy)
contrasts(data$Tarp_dummy)
```

```
#>   1
#> 0 0
#> 1 1
```

```
#set the levels
levels(data$Tarp_dummy) <- c("Other","Blue_Tarp")
is.factor(data$Tarp_dummy)
```

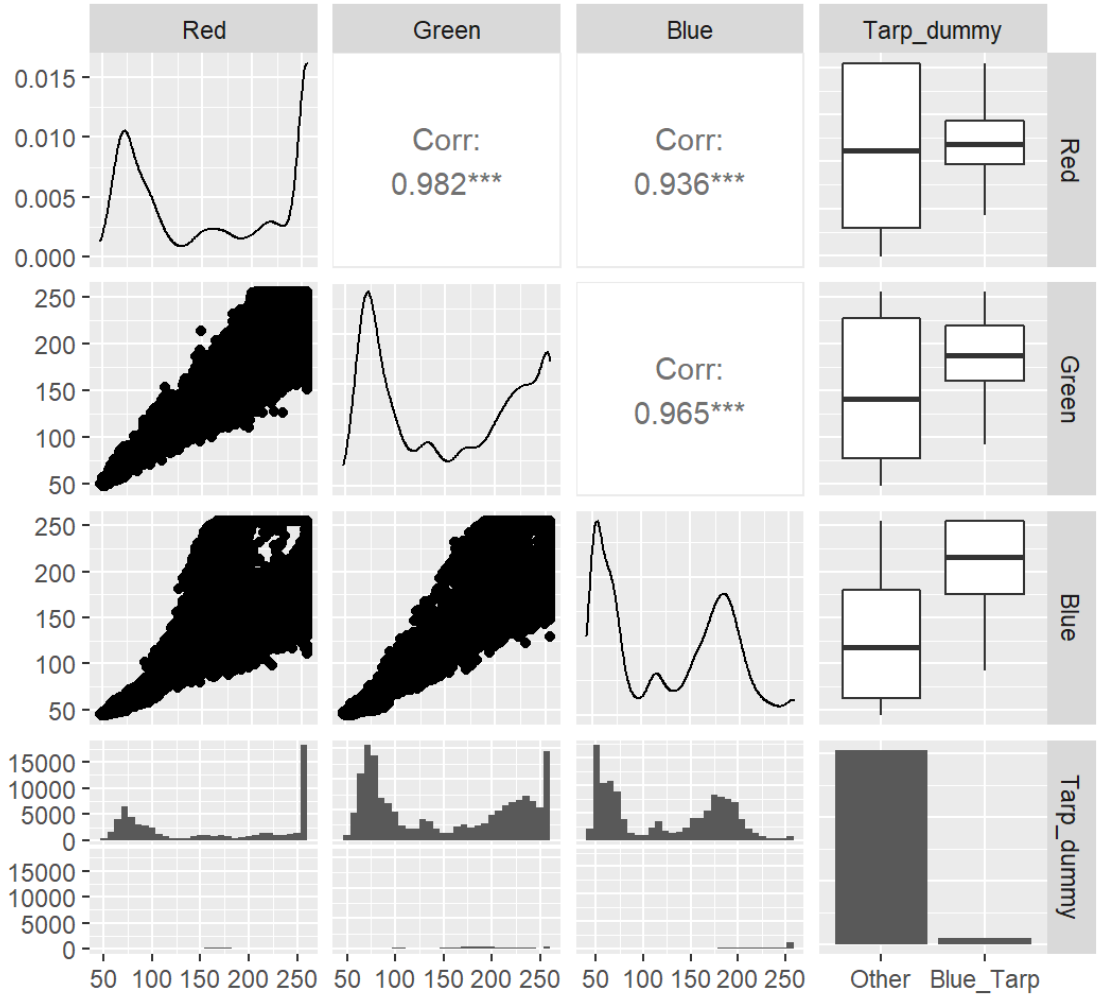
```
#> [1] TRUE
```

```
contrasts(data$Tarp_dummy)
```

```
#>           Blue_Tarp
#> Other             0
#> Blue_Tarp         1
```

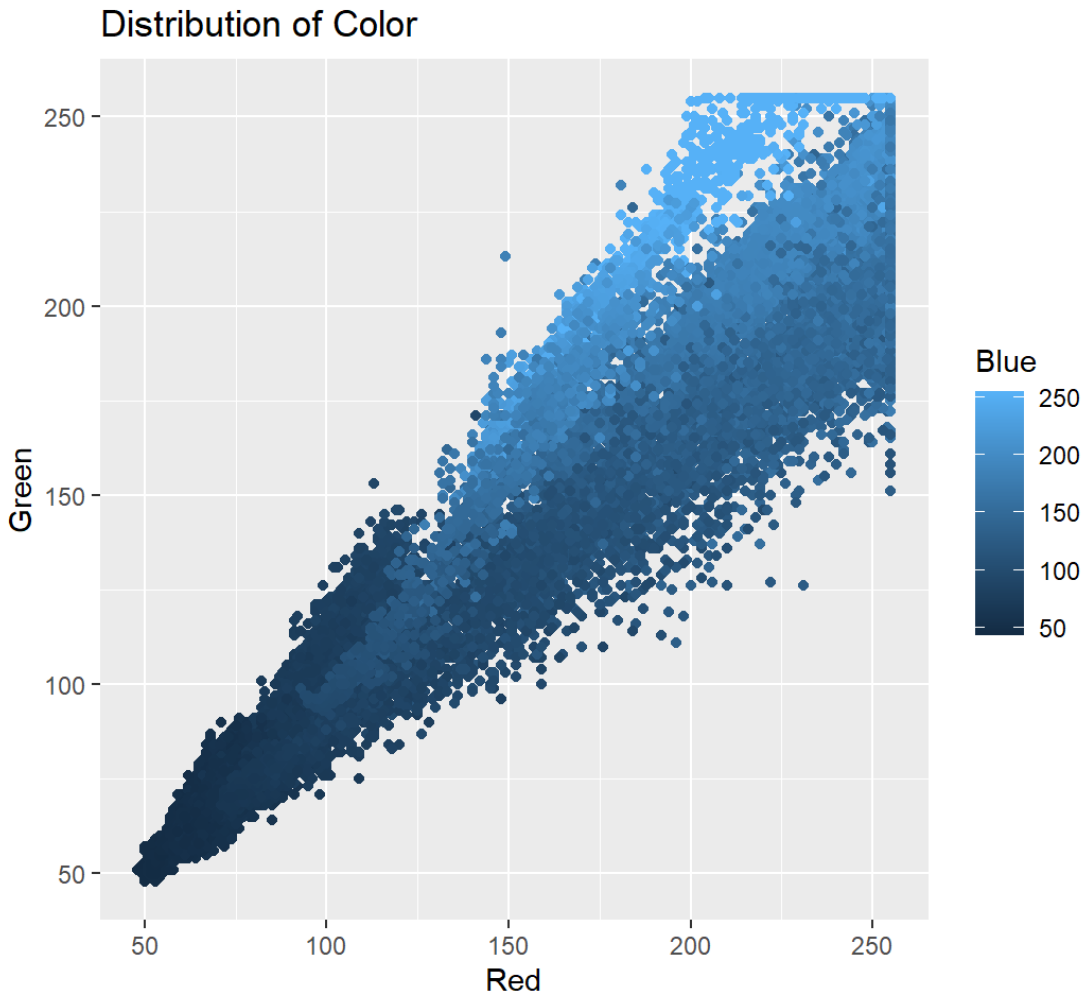
Let's explore the data using the ggpairs to see the distribution of the red, green and blue luminosity values and correlation of the pixel colors.

```
#-- Explore data
ggpairs(data[,2:5])
```



From the pairplot, we can see that there a very few blue tarp images in comparison to the Other types of images. As expected, the blue tarps have a certain distribution of pixel values that indicate a blue tarp (average luminosity of red pixels, but high blue and green luminosity values). This will bode well for a prediction model it seems because blue tarps seem to have very distinct pixel luminosities.This is likely because blue does not appear in nature very often, while green and red both appear frequently, in vegetation or soil. Additionally, There looks to be clear correlations between red, green & blue. Let's explore further

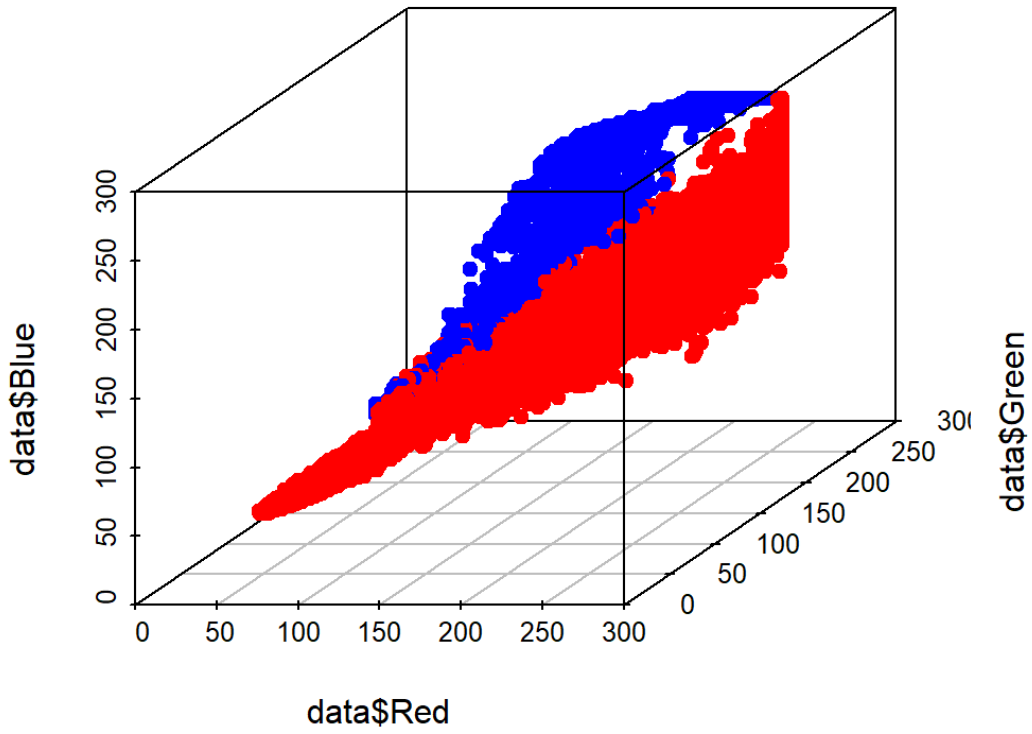
```
#-- Hue plot
pixelplot<-ggplot() + geom_point(data = data, aes(x = Red, y = Green, color = Blue))
print(pixelplot + ggtitle("Distribution of Color"))
```



As seen in the pairplot and confirmed above with a plot of the distribution of color values, the separate color pixel luminosity values are highly correlated. As red and green luminosity increases, so does blue. It is interesting, though, that it seems that high green and high blue values are more correlated than high red and high blue luminosity (and confirmed in the pairplot.

We can also plot a 3D plot of the data to see the boundary between Blue Tarps and Other Images. Overall the boundary seems to be relatively linear but still may be suited to a more flexible method of modeling. This will be worth investigating

```
scatterplot3d(x = data$Red, y = data$Green, z = data$Blue,
              color = c("red","blue")[as.factor(data$Tarp_dummy)], pch = 19)
```



3 Model Training

3.1 Set-up

Most of the model setup was done above, including setting a dummy class. We can also set a seed for when the models are run. Because we have a hold out set of data, we first use the main set to train the models using 10-fold cross validation, but we set the seed first.

```
#--
set.seed(123)
```

3.2 Logistic Regression

First, it is important to determine which form of the logistic regression (i.e. interaction terms or not, etc.) should be used in the prediction comparison using 10-fold cross validation. To do this, the performance metrics (i.e. AIC) of multiple different logistic regressions can be compared. First, the logistic regression without interaction is evaluated.

```
#-- Logistic Regression with no interaction variables
glm.logistic<- glm(Tarp_dummy ~ Red + Green + Blue, data = data, family = "binomial")
summary(glm.logistic)
```

```
#>
#> Call:
#> glm(formula = Tarp_dummy ~ Red + Green + Blue, family = "binomial",
#>     data = data)
#>
#> Deviance Residuals:
#>     Min       1Q   Median       3Q      Max
#> -3.4266  -0.0205  -0.0015   0.0000   3.7911
#>
#> Coefficients:
#>             Estimate Std. Error z value Pr(>|z|)
#> (Intercept)  0.20984    0.18455   1.137    0.256
#> Red         -0.26031    0.01262 -20.632   <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
```

Then the logistic regression with interaction can be fit and compared. It turns out that the AIC for the interaction Model is much lower than the additive Model, which indicates better performance. The p-value of this interaction term shows that the Green & Blue pixel values are not significant based on the Wald test of significance in the summary of the Model. This means that this term can be removed from the Model in the presence of all the other variables. Running the Model without the Green and Blue interaction term yields a better AIC and it is the one that will be used for cross-validation and performance comparison

```
#Lets try interaction
glm.logistic_int<- glm(Tarp_dummy ~ Red*Green*Blue, data = data, family = "binomial")
summary(glm.logistic_int)
```

```
#>
#> Call:
#> glm(formula = Tarp_dummy ~ Red * Green * Blue, family = "binomial",
#>     data = data)
#>
#> Deviance Residuals:
#>     Min       1Q   Median       3Q      Max
#> -3.2526  -0.0194  -0.0031   0.0000   3.9244
#>
#> Coefficients:
#>                   Estimate Std. Error z value Pr(>|z|)
#> (Intercept)    -4.598e+01  4.167e+00 -11.034  < 2e-16 ***
#> Red             3.329e-01  5.287e-02   6.296 3.06e-10 ***
#> Green           2.284e-01  6.469e-02   3.530 0.000415 ***
#> Blue            3.425e-01  4.784e-02   7.158 8.17e-13 ***
#> Red:Green      -3.826e-03  5.049e-04  -7.577 3.54e-14 ***
#> Red:Blue       -1.681e-03  4.135e-04  -4.065 4.80e-05 ***
#> Green:Blue      2.920e-05  2.666e-04   0.110 0.912786
#> Red:Green:Blue  1.021e-05  1.085e-06   9.416  < 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: 17902  on 63240  degrees of freedom
#> Residual deviance:  1195  on 63233  degrees of freedom
#> AIC: 1211
#>
#> Number of Fisher Scoring iterations: 14
```

```
#looks like we can remove Green Blue interaction, let's try
glm.logistic_int_red<- glm(Tarp_dummy ~ Red+Green+Blue+Red:Blue+Red:Green+Red:Green:Blue,
                           data = data, family = "binomial")
summary(glm.logistic_int_red)
```

```
#>
#> Call:
#> glm(formula = Tarp_dummy ~ Red + Green + Blue + Red:Blue + Red:Green +
#>     Red:Green:Blue, family = "binomial", data = data)
#>
#> Deviance Residuals:
#>     Min       1Q   Median       3Q      Max
#> -3.2542  -0.0193  -0.0031   0.0000   3.9250
#>
#> Coefficients:
#>                   Estimate Std. Error z value Pr(>|z|)
#> (Intercept)    -4.609e+01  4.044e+00 -11.396  < 2e-16 ***
#> Red             3.304e-01  4.768e-02   6.929 4.25e-12 ***
#> Green           2.316e-01  5.743e-02   4.033 5.52e-05 ***
#> Blue            3.442e-01  4.503e-02   7.645 2.10e-14 ***
#> Red:Blue       -1.671e-03  4.026e-04  -4.150 3.33e-05 ***
#> Red:Green      -3.823e-03  5.036e-04  -7.593 3.14e-14 ***
#> Red:Green:Blue  1.025e-05  1.039e-06   9.861  < 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:  1195.1  on 63234  degrees of freedom
#> AIC: 1209.1
#>
#> Number of Fisher Scoring iterations: 13
```

```
#rest of the variables are significant and with the best AIC (lowest)
```

The chosen Model above is trained and cross-validated using the caret package and 10-fold cross validation. Train Control is set with the predictions saved and 10-fold cross validation.

```
# Train the model

#evaluate with 10 fold
library(caret)

glm.start<-Sys.time()
set.seed(123)
train.control <- trainControl(method = "cv", number = 10,classProbs = TRUE,savePredictions ="final")

logistic_glm_model_int_red <- train(Tarp_dummy ~ Red+Green+Blue+Red:Blue+Red:Green+Red:Green:Blue,
                                    data = data, method = "glm",
                                    family="binomial",
                                    trControl = train.control)
logistic_glm_model_int_red$finalModel
```

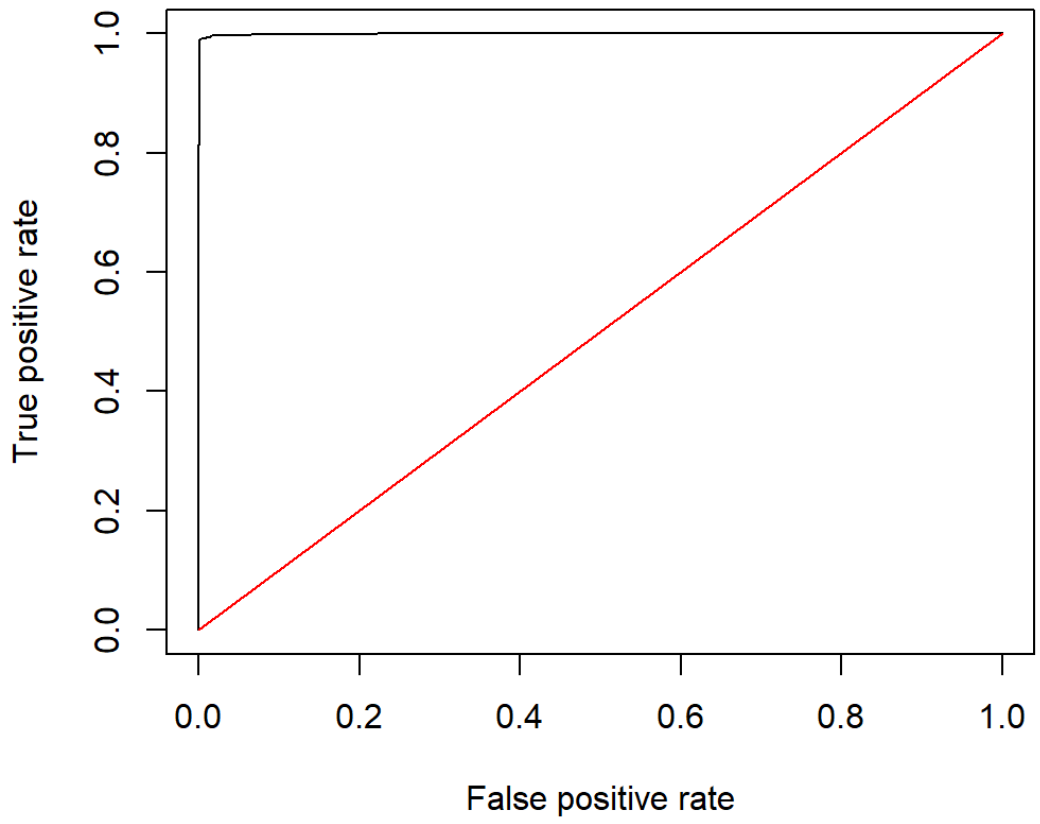
```
#>
#> Call:  NULL
#>
#> Coefficients:
#>      (Intercept)               Red             Green              Blue
#>       -4.609e+01         3.304e-01         2.316e-01         3.442e-01
#>       `Red:Blue`       `Red:Green`  `Red:Green:Blue`
#>       -1.671e-03        -3.823e-03         1.025e-05
#>
#> Degrees of Freedom: 63240 Total (i.e. Null);  63234 Residual
#> Null Deviance:       17900
#> Residual Deviance: 1195  AIC: 1209
```

```
glm.end<-Sys.time()
glm.runtime <-glm.end-glm.start
```

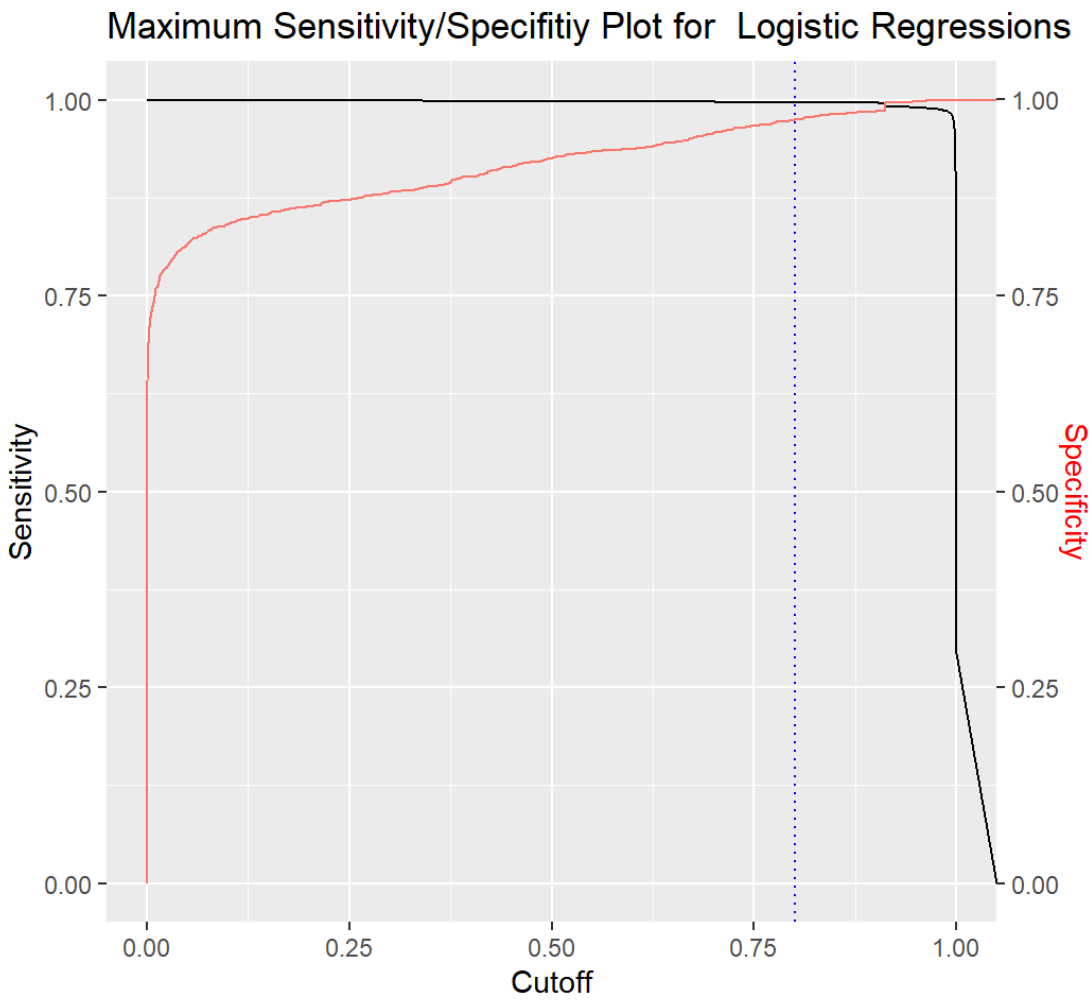
After the cross-validation, the Model is the evaluated on prediction performance first on the training set of data. To determine performance, the cross validated Model is used to on the full dataset to predict whether or not a blue tarp is present in each image (this is done on the hold out set later on). The True Positive Rate and False Positive rate are calculated and the ROC Curve is plotted. The ROC shows the the false positive rate vs. the true positive rate for all possible “thresholds” of prediction. Because the logistic regression calculates probabilities of a Blue Tarp being present in each image, a threshold has to be set when a probability is classified as a 1 for Blue Tarp or 0 for Other. The ROC curve shows the false positive and true positive rate for all thresholds that assign these probabilities. As shown by the ROC curve, here with the curve in the top left of the chart, this Model performs very well. This prediction again is on the training set so it is predicting whether the blue tarps are present in observations already used to train the Model. Predictive performance on the test set (hold out data) will be done later in the analysis.

To choose the threshold of probability that gives the best prediction performance, Caret’s thresholder function is utilized. This allows sampling different threshold probabilities to how the threshold performs on prediction of the blue tarps vs. other for each observation in the training data. This thresholder function calculates multiple different statistics from the prediction and confusion matrix by threshold. Chosen here is the one with the best accuracy.Once chosen, this threshold is then utilized to calculate certain performance metrics like TPR & FPR for the training set of data

```
#-- Test and see how it performs on the data
glm.probs<- predict(logistic_glm_model_int_red, data, type="prob")
rates.glm <- prediction(1-glm.probs[,2], data$Tarp_dummy)
perf.glm <- performance(rates.glm,"tpr","fpr")


plot(perf.glm, main="ROC Curve for Logistic Regression")
lines(x = c(0,1), y = c(0,1), col="red")
```




```
auc.glm <- performance(rates.glm, measure = "auc")
auc.glm<-auc.glm@y.values[[1]]


sens.glm <- data.frame(x=unlist(performance(rates.glm, "sens")@x.values),
                       y=unlist(performance(rates.glm, "sens")@y.values))
spec.glm <- data.frame(x=unlist(performance(rates.glm, "spec")@x.values),
                       y=unlist(performance(rates.glm, "spec")@y.values))

thres.glm <-thresholder(logistic_glm_model_int_red,
                        threshold = seq(0.1,1,by=.02),
                        final=TRUE,statistics = 'all' )
thres.glm.index=which.max(thres.glm$Accuracy)
threshold.glm <- thres.glm$prob_threshold[thres.glm.index]

glm.plot.threshold<-sens.glm %>% ggplot(aes(x,y)) +
  geom_line() +
  geom_line(data=spec.glm, aes(x,y,col="red")) +
  geom_vline(xintercept = threshold.glm,col="blue",linetype="dotted")+
  scale_y_continuous(sec.axis = sec_axis(~., name = "Specificity")) +
  labs(x='Cutoff', y="Sensitivity") +
  theme(axis.title.y.right = element_text(colour = "red"),
        legend.position="none") +
  ggtitle("Maximum Sensitivity/Specifitiy Plot for  Logistic Regressions")
plot(glm.plot.threshold)
```




```
glm.pred<-rep("Other",63241)
glm.pred[glm.probs[,2]>threshold.glm]="Tarp"
table(glm.pred,data$Tarp_dummy)
```

```
#>
#> glm.pred Other Blue_Tarp
#>    Other 61175       274
#>    Tarp     44      1748
```

```
mean(glm.pred==data$Tarp_dummy)
```

```
#> [1] 0.9673313
```

```
bottom_right.glm<-table(glm.pred,data$Tarp_dummy)[2,2]
bottom_left.glm<-table(glm.pred,data$Tarp_dummy)[2,1]
top_left.glm<-table(glm.pred,data$Tarp_dummy)[1,1]
top_right.glm<-table(glm.pred,data$Tarp_dummy)[1,2]

tpr.glm = bottom_right.glm/(bottom_right.glm+top_right.glm)
fpr.glm = bottom_left.glm/(bottom_left.glm+top_left.glm)
precision.glm = bottom_right.glm/(bottom_right.glm+bottom_left.glm)
accuracy.glm = (top_left.glm+bottom_right.glm)/sum(table(glm.pred,data$Tarp_dummy))
```

3.3 LDA

Next, the Linear Discriminant Analysis Model (LDA) is is trained using 10 fold cross validation (no interaction terms) on the main training set of data. The same method from the caret package is used. LDA creates a new axis and projects data onto a new axis in a way to maximize the separation of the two classes (Blue Tarps & Other images). LDA assumes the following:

```
1. Maximize the distance between means of the two classes
2. Minimize the variation within each class
3. All classes have the same covariance matrix
```

The 10-fold cross validation using caret uses the same Train Control Method as use in logistic regression.

```
#-- LDA 10-Fold Cross Validation

lda.start<-Sys.time()
set.seed(123)
#10 fold cross validation
lda_model_no_int <- train(Tarp_dummy ~ Red + Green + Blue, data = data,
                          method = "lda",
               trControl = train.control)
lda_model_no_int$finalModel
```

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

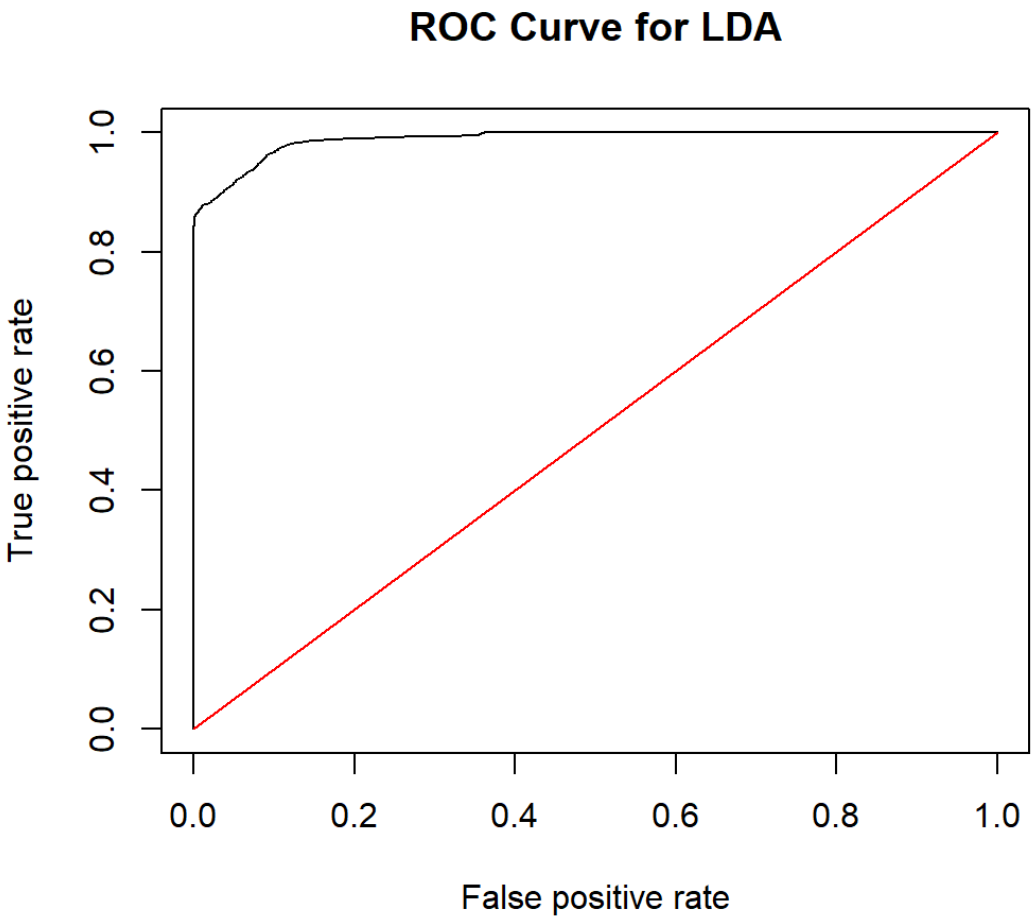
```
lda.end<-Sys.time()
lda.runtime <-lda.end-lda.start
```

Just like how we used the cross-validated Logistic regression to predict the observation of blue tarps in each image (for the training data), we do the same for LDA. The same ROC Plot is using the cross validated model to predict the image class with different thresholds of probability. The same method is used to determine which threshold probability value should be used for classification.

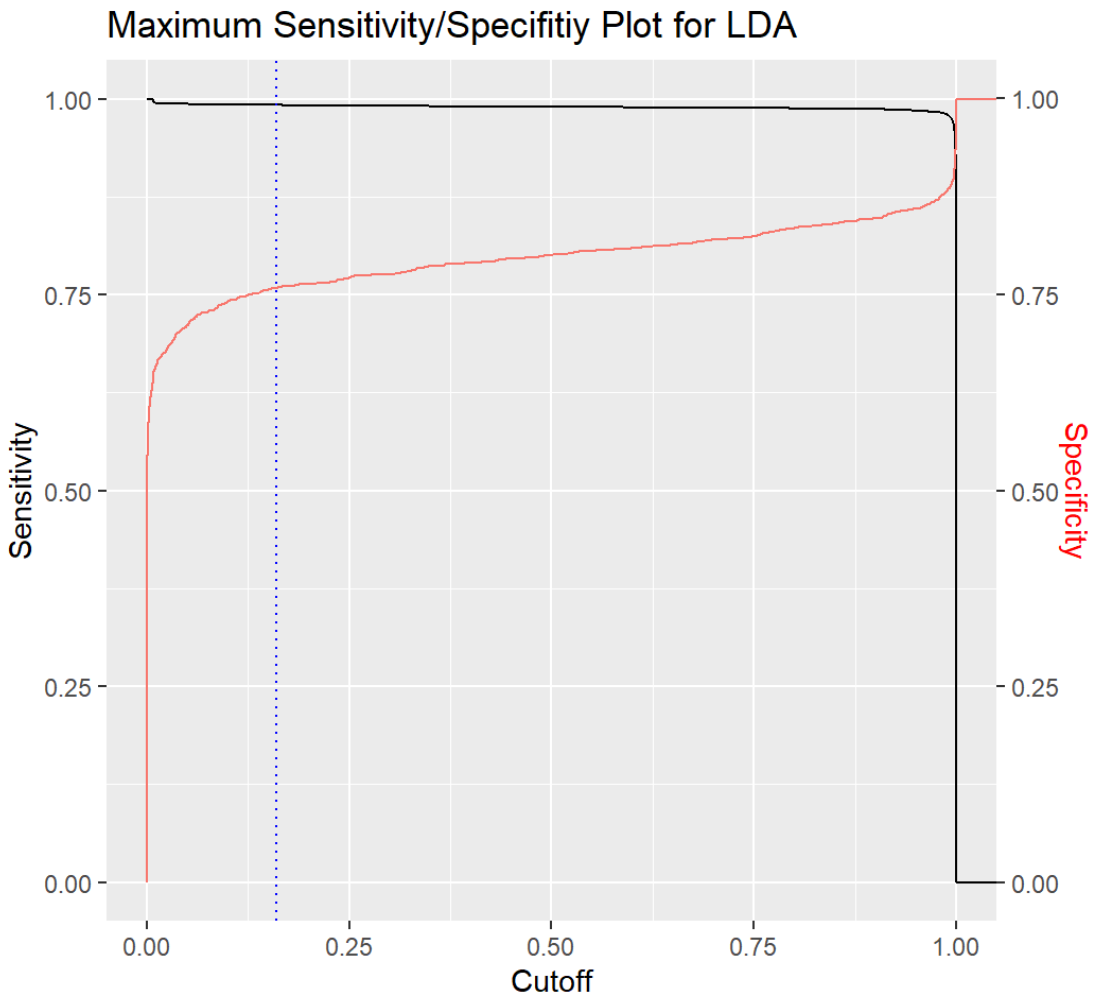
To choose the threshold of probability that gives the best prediction performance, Caret's thresholder function is utilized. This allows sampling different threshold probabilities to how the threshold performs on prediction of the blue tarps vs. other for each observation in the training data. This thresholder function calculates multiple different statistics

from the prediction and confusion matrix by threshold. Chosen here is the one with the best accuracy.Once chosen, this threshold is then utilized to calculate certain performance metrics like TPR & FPR for the training set of data

```
#-- LDA Prediction Matrix
lda.probs<- predict(lda_model_no_int, data, type="prob")
lda.pred<-rep("Other",63241)
rates.lda <- prediction(1-lda.probs[,2], data$Tarp_dummy)
perf.lda <- performance(rates.lda,"tpr","fpr")

plot(perf.lda, main="ROC Curve for LDA")
lines(x = c(0,1), y = c(0,1), col="red")
```



```
auc.lda <- performance(rates.lda, measure = "auc")
auc.lda<-auc.lda@y.values[[1]]


sens.lda <- data.frame(x=unlist(performance(rates.lda, "sens")@x.values),
                   y=unlist(performance(rates.lda, "sens")@y.values))
spec.lda <- data.frame(x=unlist(performance(rates.lda, "spec")@x.values),
                   y=unlist(performance(rates.lda, "spec")@y.values))

thres.lda <-thresholder(lda_model_no_int,threshold = seq(0.1,1,by=.02),final=TRUE,statistics = 'all' )
thres.lda.index=which.max(thres.lda$Accuracy)
threshold.lda <- thres.lda$prob_threshold[thres.lda.index]

lda.plot.threshold<-sens.lda %>% ggplot(aes(x,y)) +
  geom_line() +
  geom_line(data=spec.lda, aes(x,y,col="red")) +
  geom_vline(xintercept = threshold.lda,col="blue",linetype="dotted")+
  scale_y_continuous(sec.axis = sec_axis(~., name = "Specificity")) +
  labs(x='Cutoff', y="Sensitivity") +
  theme(axis.title.y.right = element_text(colour = "red"), legend.position="none")+
  ggtitle("Maximum Sensitivity/Specifitiy Plot for LDA")
plot(lda.plot.threshold)
```




```
lda.pred[lda.probs[,2]>threshold.lda]="Tarp"
table(lda.pred,data$Tarp_dummy)
```

```
#>
#> lda.pred Other Blue_Tarp
#>    Other 60454       324
#>    Tarp    765      1698
```

```
mean(lda.pred==data$Tarp_dummy)
```

```
#> [1] 0.9559305
```

```
bottom_right.lda<-table(lda.pred,data$Tarp_dummy)[2,2]
bottom_left.lda<-table(lda.pred,data$Tarp_dummy)[2,1]
top_left.lda<-table(lda.pred,data$Tarp_dummy)[1,1]
top_right.lda<-table(lda.pred,data$Tarp_dummy)[1,2]

tpr.lda = bottom_right.lda/(bottom_right.lda+top_right.lda)
fpr.lda = bottom_left.lda/(bottom_left.lda+top_left.lda)
precision.lda = bottom_right.lda/(bottom_right.lda+bottom_left.lda)
accuracy.lda = (top_left.lda+bottom_right.lda)/sum(table(lda.pred,data$Tarp_dummy))
```

3.4 QDA

The same process is then repeated for Quadratic Discriminant Analysis (QDA) as done for the logistic regression and LDA. QDA uses the same maximum likelihood rule and Bayesian rules on the Gaussian distribution used in LDA, but the last assumption that the variances are equal on both classes is removed. This means that the quadratic term in the likelihood function is not canceled out.

```
#-- QDA 10-Fold Cross Validation
#10 fold cross validation

qda.start<-Sys.time()
set.seed(123)
qda_model_no_int <- train(Tarp_dummy ~ Red + Green + Blue, data = data, method = "qda",
               trControl = train.control)

qda_model_no_int$finalModel
```

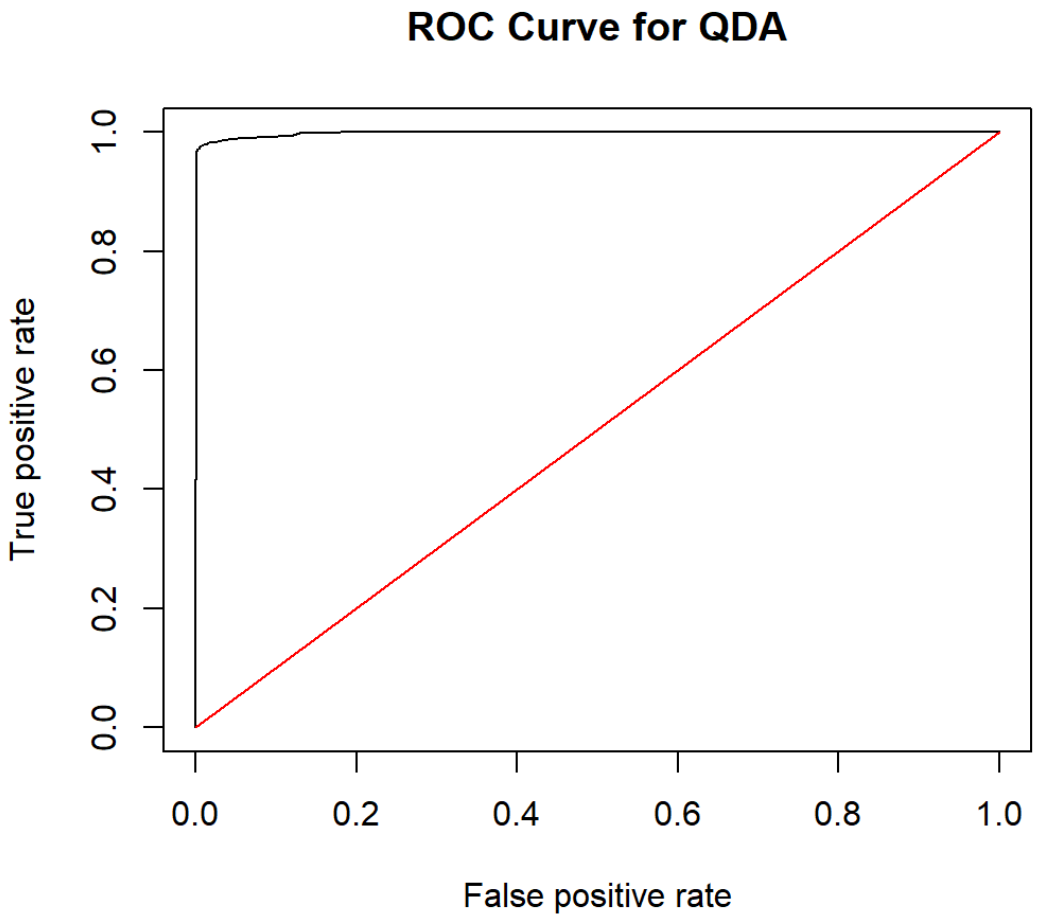
```
#> Call:
#> qda(x, grouping = y)
#>
#> Prior probabilities of groups:
#>      Other  Blue_Tarp
#> 0.96802707 0.03197293
#>
#> Group means:
#>                Red    Green     Blue
#> Other     162.7604 152.5808 122.4993
#> Blue_Tarp 169.6627 186.4149 205.0371
```

```
qda.end<-Sys.time()
qda.runtime <-qda.end-qda.start
```

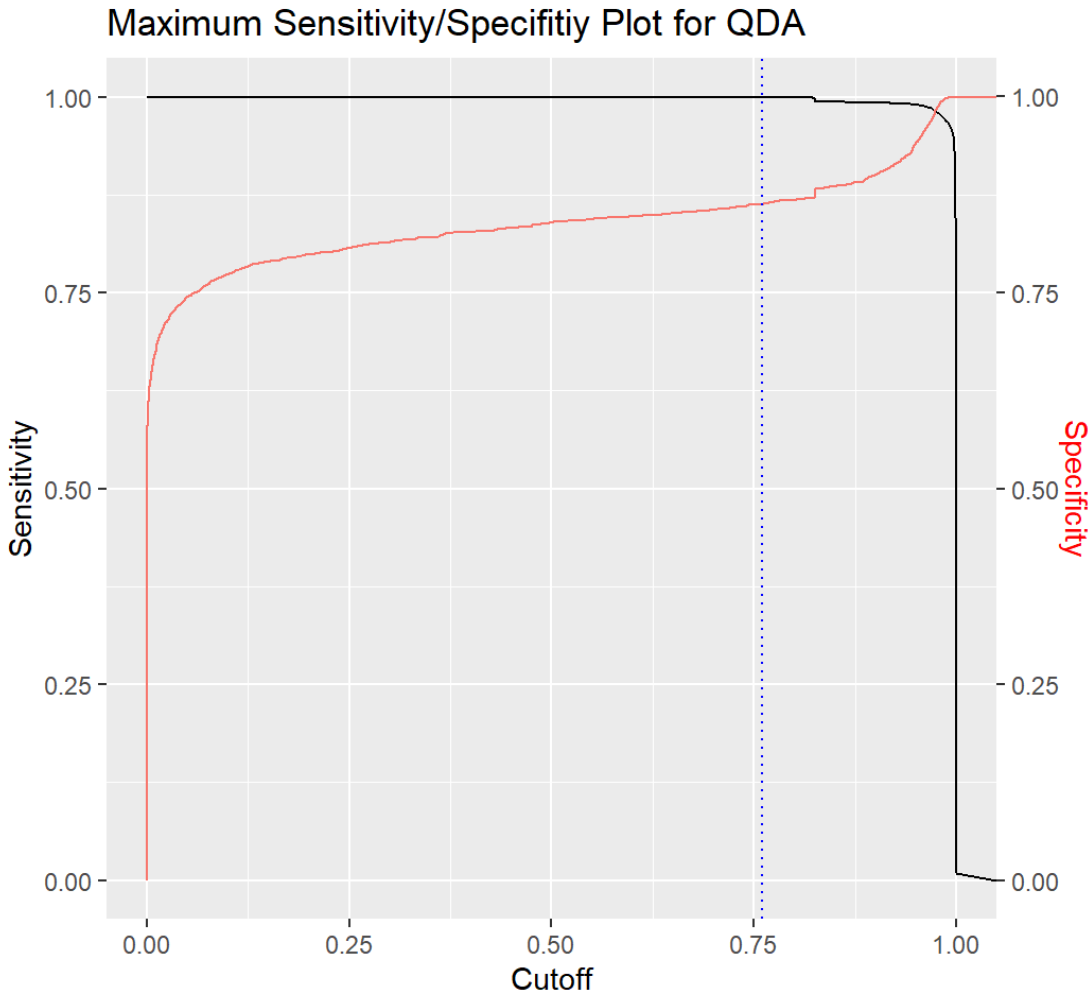
After training the QDA Model on all the data with cross-validation, the prediction performance on the training data (hold out data comes later) is evaluated using the same process the was used on LDA & Logistic regression. The same ROC Plot is generated. The same method is used to determine which threshold probability value should be used for classification.

To choose the threshold of probability that gives the best prediction performance, Caret's thresholder function is again utilized with the thresold with the set accuracy chosen.

```
#-- QDA Prediction Matrix
qda.probs<- predict(qda_model_no_int, data, type="prob")
qda.pred<-rep("Other",63241)
rates.qda <- prediction(1-qda.probs[,2], data$Tarp_dummy)
perf.qda <- performance(rates.qda,"tpr","fpr")

plot(perf.qda, main="ROC Curve for QDA")
lines(x = c(0,1), y = c(0,1), col="red")
```



```
auc.qda <- performance(rates.qda, measure = "auc")
auc.qda<-auc.qda@y.values[[1]]



sens.qda <- data.frame(x=unlist(performance(rates.qda, "sens")@x.values),
                       y=unlist(performance(rates.qda, "sens")@y.values))
spec.qda <- data.frame(x=unlist(performance(rates.qda, "spec")@x.values),
                       y=unlist(performance(rates.qda, "spec")@y.values))

thres.qda <-thresholder(qda_model_no_int,threshold = seq(0.1,1,by=.02),final=TRUE,statistics = 'all' )
thres.qda.index=which.max(thres.qda$Accuracy)
threshold.qda <- thres.qda$prob_threshold[thres.qda.index]


qda.plot.threshold<-sens.qda %>% ggplot(aes(x,y)) +
  geom_line() +
  geom_line(data=spec.qda, aes(x,y,col="red")) +
  geom_vline(xintercept = threshold.qda,col="blue",linetype="dotted")+
  scale_y_continuous(sec.axis = sec_axis(~., name = "Specificity")) +
  labs(x='Cutoff', y="Sensitivity") +
  theme(axis.title.y.right = element_text(colour = "red"), legend.position="none") +
  ggtitle("Maximum Sensitivity/Specifitiy Plot for QDA")
plot(qda.plot.threshold)
```



```
qda.pred[qda.probs[,2]>threshold.qda]="Tarp"
table(qda.pred,data$Tarp_dummy)
```

```
#>
#> qda.pred Other Blue_Tarp
#>    Other 61213       395
#>    Tarp      6      1627
```

```
mean(qda.pred==data$Tarp_dummy)
```

```
#> [1] 0.9679322
```

```
bottom_right.qda<-table(qda.pred,data$Tarp_dummy)[2,2]
bottom_left.qda<-table(qda.pred,data$Tarp_dummy)[2,1]
top_left.qda<-table(qda.pred,data$Tarp_dummy)[1,1]
top_right.qda<-table(qda.pred,data$Tarp_dummy)[1,2]

tpr.qda = bottom_right.qda/(bottom_right.qda+top_right.qda)
fpr.qda = bottom_left.qda/(bottom_left.qda+top_left.qda)
precision.qda = bottom_right.qda/(bottom_right.qda+bottom_left.qda)
accuracy.qda = (top_left.qda+bottom_right.qda)/sum(table(qda.pred,data$Tarp_dummy))
```

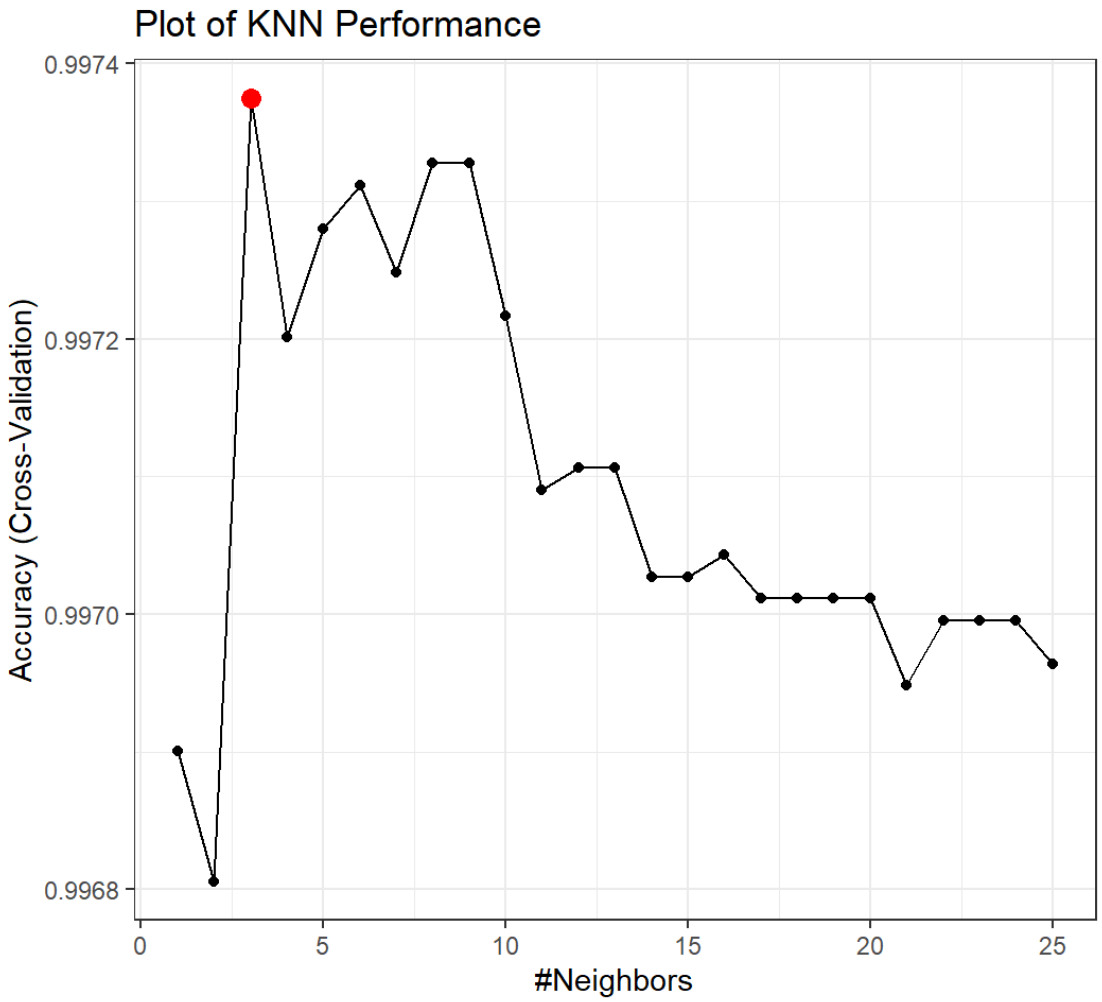
3.5 KNN

For K Nearest Neighbors, a tuning parameter is needed to determine how many (K) nearest points should be used to determine whether the image contains a blue tarp or not. To determine this, 10-fold cross validation is ran with multiple different values of K to see which performs the best in prediction. In this case, 1 - 25 is used for K. This is done below and plotted. It is important to note the time complexity of this run. Time to run this via 10-fold cross validation takes a while to complete which should be kept in mind in choosing the appropriate predictor. As shown in the chart below of k vs. accuracy on cross validation, K = 3 is the ideal Model.

```
#-- KNN


knn.start <-Sys.time()
set.seed(123)
knn_model <- train(
  Tarp_dummy ~ Red + Green + Blue,
  data = data,
  method = "knn",
  trControl = train.control,
  preProcess = c("center", "scale"), #this scales  the predictors to have mean 0 and variance 1
  tuneGrid = expand.grid(k = seq(1, 25, by = 1)) #check all k's 0 to 25
)
knn_model$finalModel
```

```
#> 3-nearest neighbor model
#> Training set outcome distribution:
#>
#>     Other Blue_Tarp
#>     61219      2022
```

```
knn.end<-Sys.time()
knn.runtime <-knn.end-knn.start


best_k <-knn_model$bestTune[1,1]
accuracy_knn_top<-knn_model$results[best_k,]$Accuracy
knn_plot_data <- data.frame(best_k,accuracy_knn_top)

knn_plot<-  ggplot(knn_model) + theme_bw()
print(knn_plot + ggtitle("Plot of KNN Performance")+geom_point(data=knn_plot_data,
             mapping = aes(x=best_k,y=accuracy_knn_top),
             color='red',
             size=3))
```

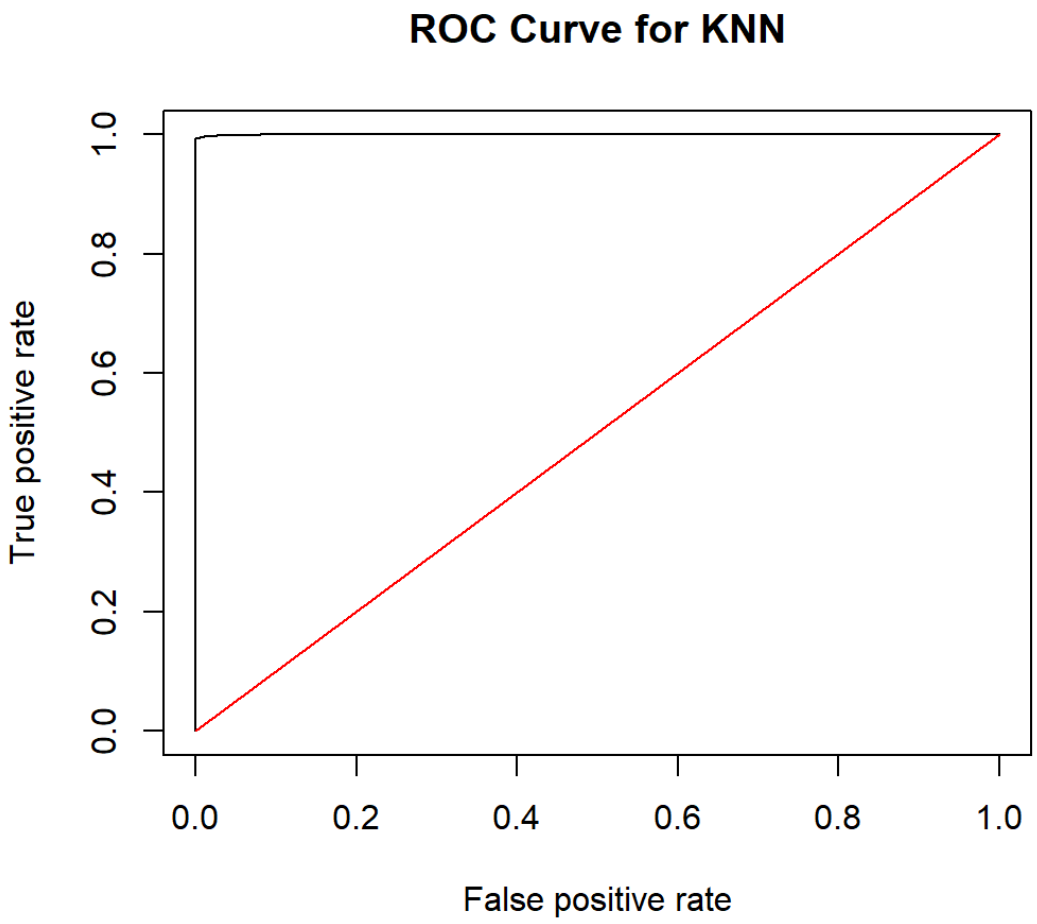


We then run the with just the best tune (3 Nearest Neighbors)

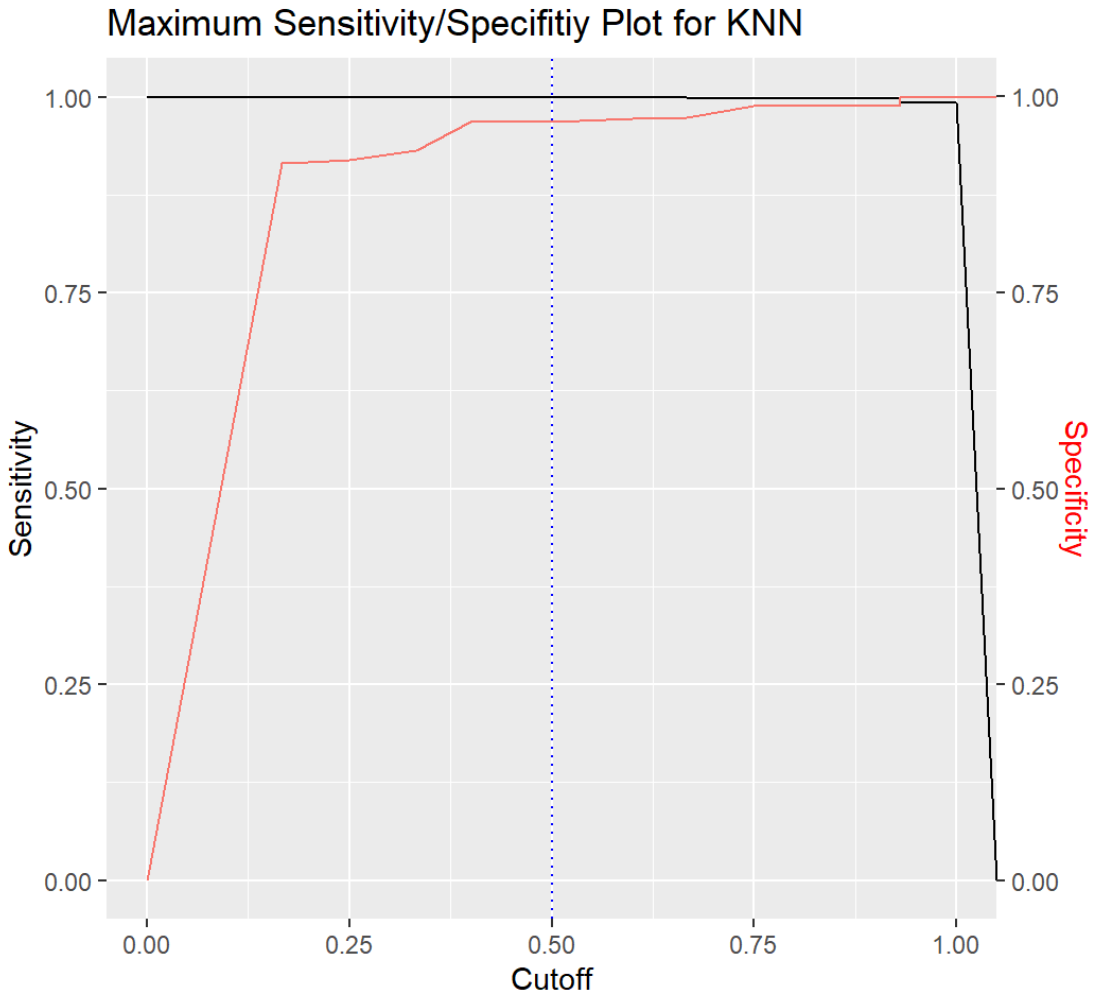
```
#-- KNN best results
set.seed(123)

knn_final = train(
   Tarp_dummy ~ Red + Green + Blue,
  data = data,
  method = "knn",
  trControl = train.control,
  tuneGrid = expand.grid(k = best_k)
)
```

And then use the same process for prediction and determining the probability threshold using the thresholder function. Once the threshold was chosen, the confusion matrix is used to determine the accuracy, precision, tpr, fpr, etc. for the training data.

```
knn.probs<- predict(knn_final, data, type="prob")
knn.pred<-rep("Other",63241)
rates.knn <- prediction(1-knn.probs[,2], data$Tarp_dummy)
perf.knn <- performance(rates.knn,"tpr","fpr")

plot(perf.knn, main="ROC Curve for KNN")
lines(x = c(0,1), y = c(0,1), col="red")
```



```
auc.knn <- performance(rates.knn, measure = "auc")
auc.knn<-auc.knn@y.values[[1]]



sens.knn <- data.frame(x=unlist(performance(rates.knn, "sens")@x.values),
                       y=unlist(performance(rates.knn, "sens")@y.values))
spec.knn <- data.frame(x=unlist(performance(rates.knn, "spec")@x.values),
                       y=unlist(performance(rates.knn, "spec")@y.values))

thres.knn <-thresholder(knn_final,threshold = seq(0.1,1,by=.02),final=TRUE,statistics = 'all' )
thres.knn.index=which.max(thres.knn$Accuracy)
threshold.knn <- thres.knn$prob_threshold[thres.knn.index]


knn.plot.threshold<-sens.knn %>% ggplot(aes(x,y)) +
  geom_line() +
  geom_line(data=spec.knn, aes(x,y,col="red")) +
  geom_vline(xintercept = threshold.knn,col="blue",linetype="dotted")+
  scale_y_continuous(sec.axis = sec_axis(~., name = "Specificity")) +
  labs(x='Cutoff', y="Sensitivity") +
  theme(axis.title.y.right = element_text(colour = "red"), legend.position="none")+
  ggtitle("Maximum Sensitivity/Specifitiy Plot for KNN")
plot(knn.plot.threshold)
```



```
knn.pred[knn.probs[,2]>threshold.knn]="Tarp"
table(knn.pred,data$Tarp_dummy)
```

```
#>
#> knn.pred Other Blue_Tarp
#>    Other 61163        64
#>    Tarp     56      1958
```

```
mean(knn.pred==data$Tarp_dummy)
```

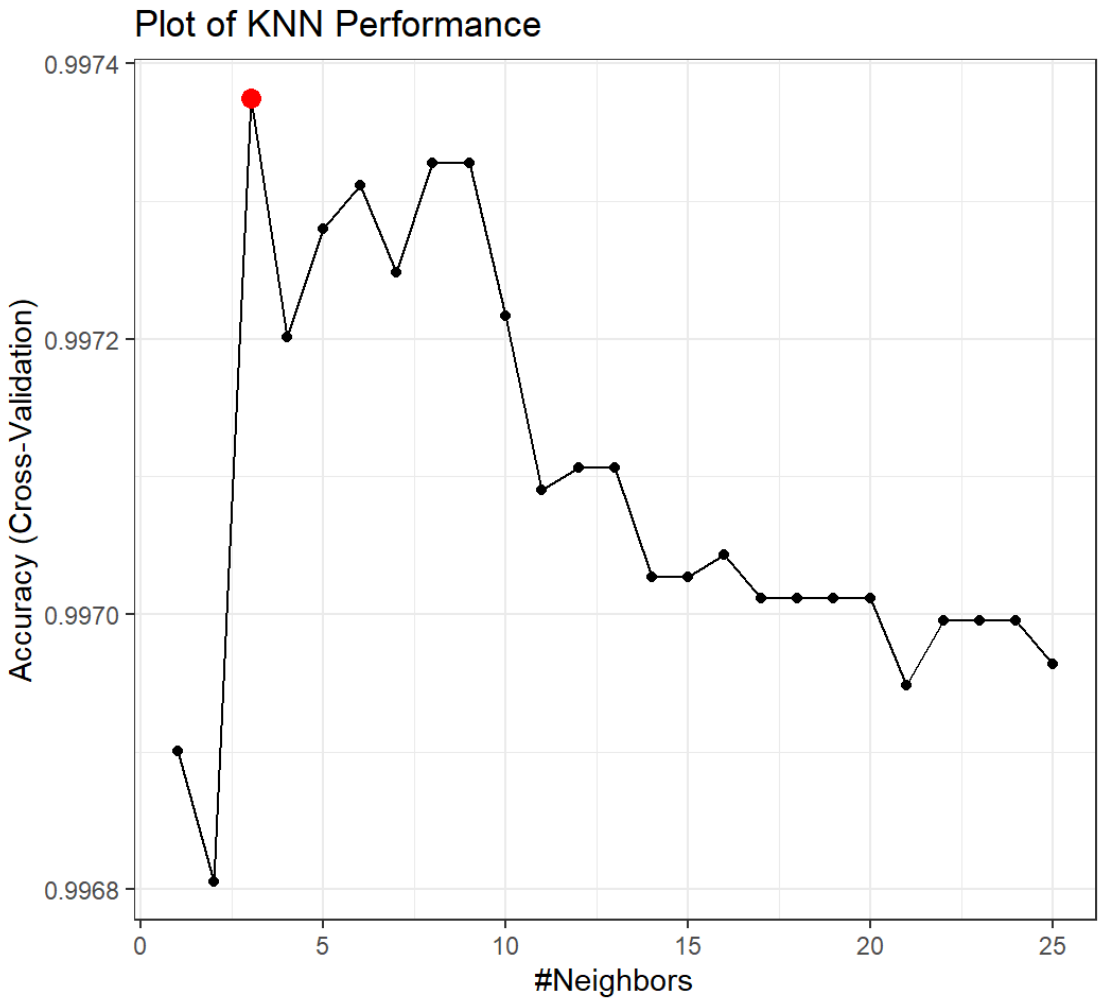
```
#> [1] 0.9671416
```

```
bottom_right.knn<-table(knn.pred,data$Tarp_dummy)[2,2]
bottom_left.knn<-table(knn.pred,data$Tarp_dummy)[2,1]
top_left.knn<-table(knn.pred,data$Tarp_dummy)[1,1]
top_right.knn<-table(knn.pred,data$Tarp_dummy)[1,2]

tpr.knn = bottom_right.knn/(bottom_right.knn+top_right.knn)
fpr.knn = bottom_left.knn/(bottom_left.knn+top_left.knn)
precision.knn = bottom_right.knn/(bottom_right.knn+bottom_left.knn)
accuracy.knn = (top_left.knn+bottom_right.knn)/sum(table(knn.pred,data$Tarp_dummy))
```

3.5.1 Tuning Parameter k

As mentioned above, the tuning parameter k was determined via 10-fold cross validation over 25 different values of K. This was done using the caret package and using a tuning grid of 1 - 25. Plotting the cross validation model accuracy vs. k # of neighbors shows that 3 ends up being the best value of k with the most accurate performance Model in terms of prediction. The plot is replicated below which displays this.

```
best_k <-knn_model$bestTune[1,1]
accuracy_knn_top<-knn_model$results[best_k,]$Accuracy
knn_plot_data <- data.frame(best_k,accuracy_knn_top)

knn_plot<-  ggplot(knn_model) + theme_bw()
print(knn_plot + ggtitle("Plot of KNN Performance")+geom_point(data=knn_plot_data,
             mapping = aes(x=best_k,y=accuracy_knn_top),
             color='red',
             size=3))
```



3.6 Penalized Logistic Regression

The same process is then repeated for for Penalized Logistic Regression here, but similarly to KNN there is a tuning parameter λ that needs to be determined. This λ value determines how much weighting the bias term (or the penalized term) gets in fitting the model. A tuning grid can also be used just as before. ElasticNet is used which is a combination of Ridge & Lasso Penalized regression.

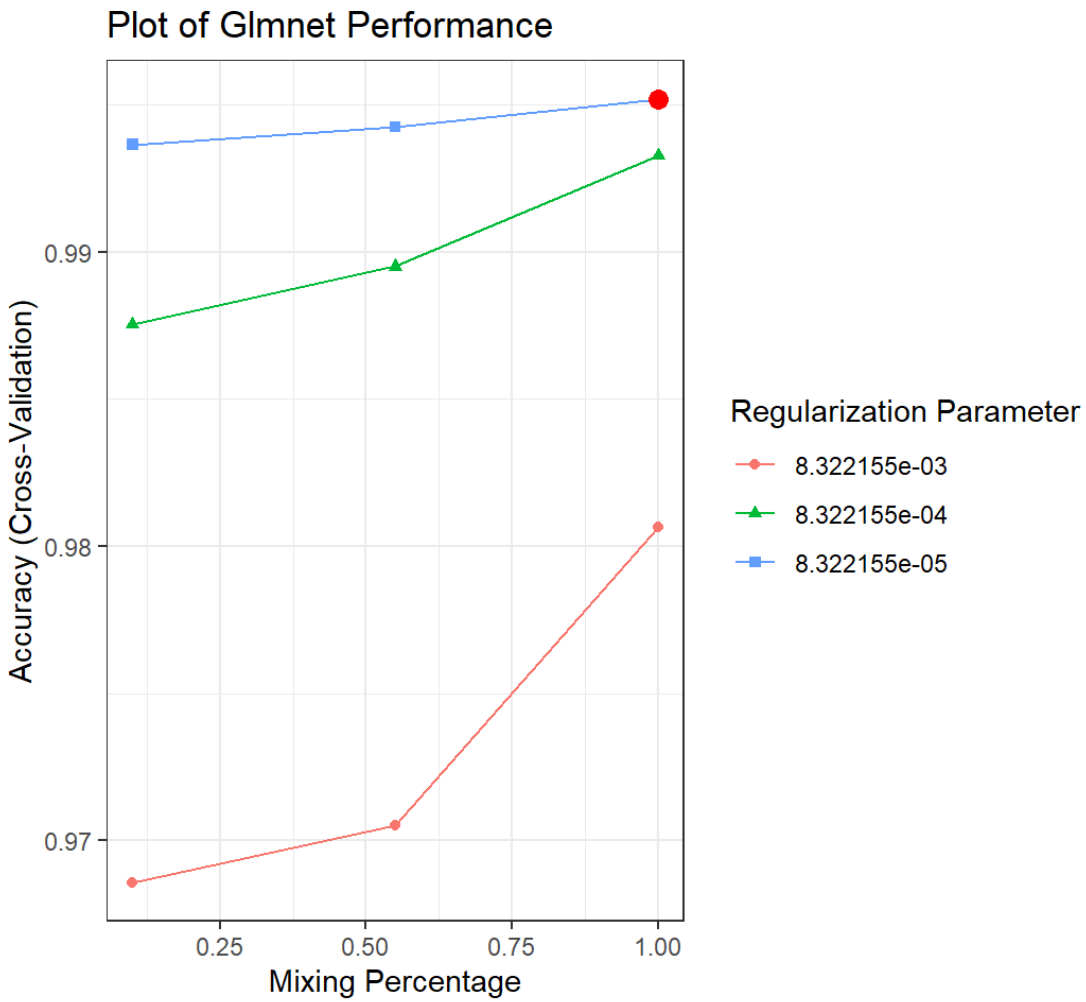
```
#-- 10-Fold Cross Validation
#10 fold cross validation
set.seed(123)
grid <- 10^seq(10,-2,length= 100)
ridge.start<-Sys.time()
ridge.fit<-train(Tarp_dummy ~ Red+Green+Blue, data = data,
                 method = "glmnet",family="binomial",
                 trControl = train.control,
                 TuneGrid = expand.grid(lambda=grid ))

ridge.fit$bestTune
```

```
#>   alpha       lambda
#> 7     1 8.322155e-05
```

```
ridge.end<-Sys.time()
ridge.runtime <-ridge.end-ridge.start
```

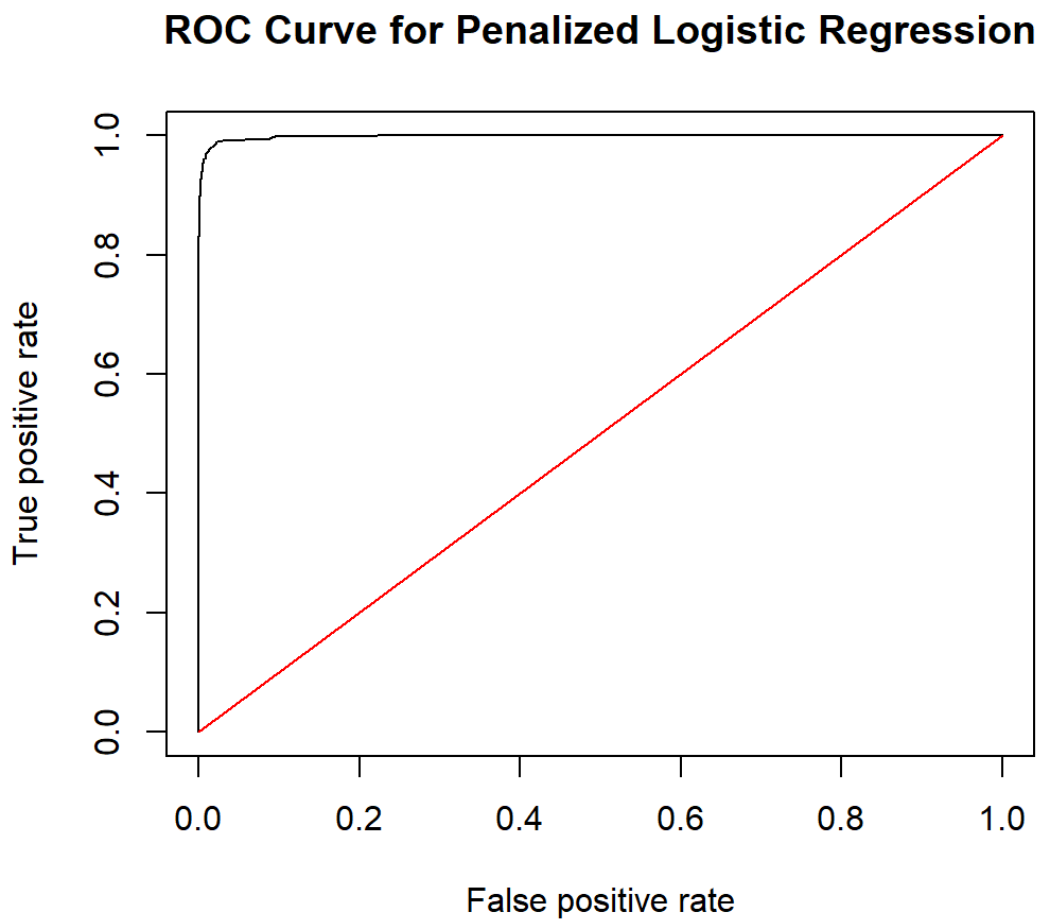
As shown below, the Model with the best accuracy is the model with alpha = 1, which is Lasso, with a λ of $8.322e^{-5}$.

```
best_index<-as.numeric(rownames(ridge.fit$bestTune[1,0]))
best_alpha<- ridge.fit$bestTune[1,1]
best_lambda<- ridge.fit$bestTune[1,2]
accuracy_lambda_top<-ridge.fit$results[best_index,]$Accuracy
lambda_plot_data <- data.frame(best_alpha,accuracy_lambda_top)

ridge_plot<-  ggplot(ridge.fit, xvar = "lambda") + theme_bw()
print(ridge_plot + ggtitle("Plot of Glmnet Performance")+geom_point(data=lambda_plot_data,
             mapping = aes(x=best_alpha,y=accuracy_lambda_top),
             color='red',
             size=3)
)
```

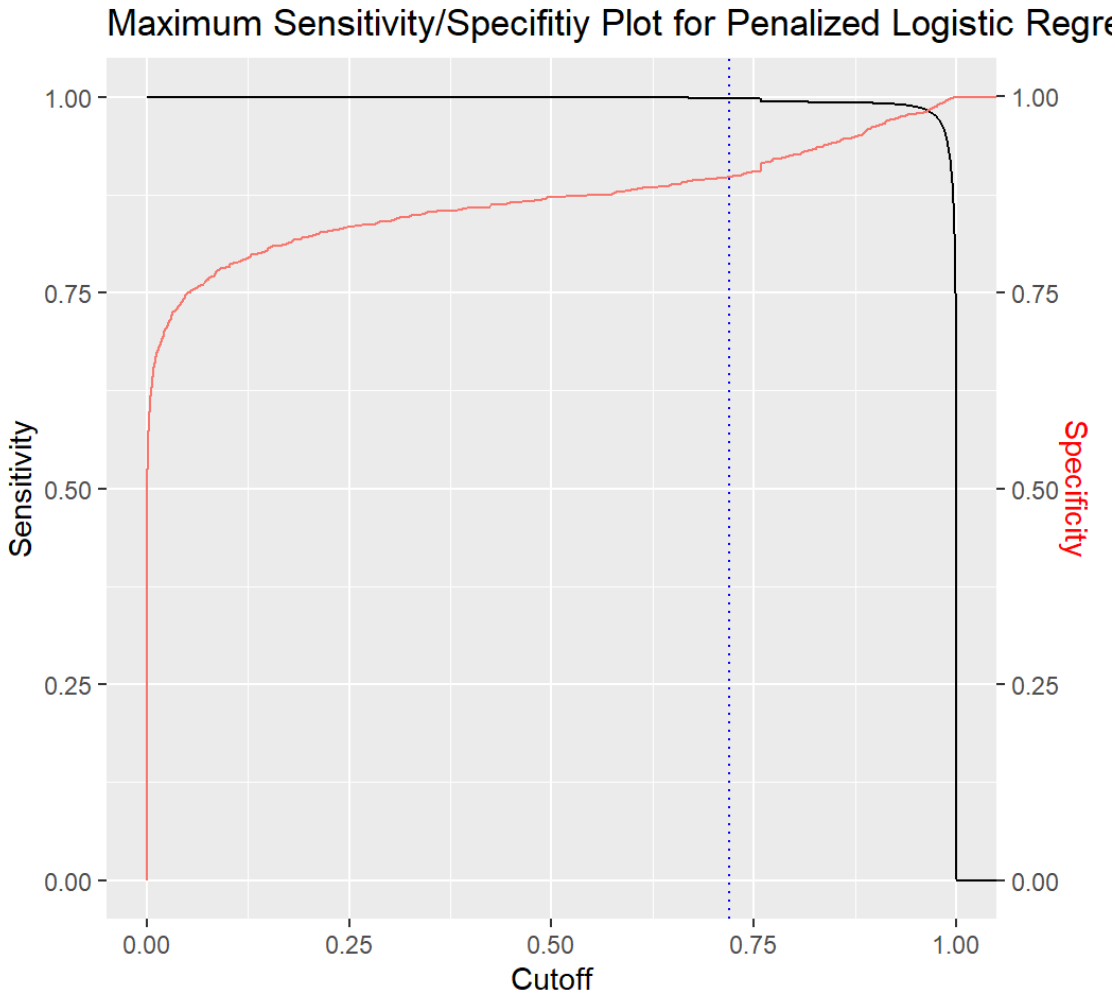


For the ElasticNet Model, the same process was used to evaluate prediction performance as was done for all the other Model: the Model predicts the probability of a class (Blue Tarp or Other) for all the data points and then the ROC plot is plotted. The probability threshold to classify Blue Tarps is chosen using thresholder function in caret and then applied to the predicted probabilities to determine a confusion matrix and relevant prediction statistics.

```
#-- glmnet Prediction Matrix

#fit the best Model

ridge.probs<- predict(ridge.fit, data, type="prob")
ridge.pred<-rep("Other",63241)
rates.ridge <- prediction(1-ridge.probs[,2], data$Tarp_dummy)
perf.ridge <- performance(rates.ridge,"tpr","fpr")

plot(perf.ridge, main="ROC Curve for Penalized Logistic Regression")
lines(x = c(0,1), y = c(0,1), col="red")
```



```
auc.ridge <- performance(rates.ridge, measure = "auc")
auc.ridge<-auc.ridge@y.values[[1]]



sens.ridge <- data.frame(x=unlist(performance(rates.ridge, "sens")@x.values),
                         y=unlist(performance(rates.ridge, "sens")@y.values))
spec.ridge <- data.frame(x=unlist(performance(rates.ridge, "spec")@x.values),
                         y=unlist(performance(rates.ridge, "spec")@y.values))

thres.ridge <-thresholder(ridge.fit,threshold = seq(0.1,1,by=.02),final=TRUE,statistics = 'all' )
thres.ridge.index=which.max(thres.ridge$Accuracy)
threshold.ridge <- thres.ridge$prob_threshold[thres.ridge.index]

ridge.plot.threshold<-sens.ridge %>% ggplot(aes(x,y)) +
  geom_line() +
  geom_line(data=spec.ridge, aes(x,y,col="red")) +
  geom_vline(xintercept = threshold.ridge,col="blue",linetype="dotted")+
  scale_y_continuous(sec.axis = sec_axis(~., name = "Specificity")) +
  labs(x='Cutoff', y="Sensitivity") +
  theme(axis.title.y.right = element_text(colour = "red"), legend.position="none")+
  ggtitle("Maximum Sensitivity/Specifitiy Plot for Penalized Logistic Regressions")

plot(ridge.plot.threshold)
```



```
ridge.pred[ridge.probs[,2]>threshold.ridge]="Tarp"
table(ridge.pred,data$Tarp_dummy)
```

```
#>
#> ridge.pred Other Blue_Tarp
#>      Other 61194       329
#>      Tarp     25      1693
```

```
mean(ridge.pred==data$Tarp_dummy)
```

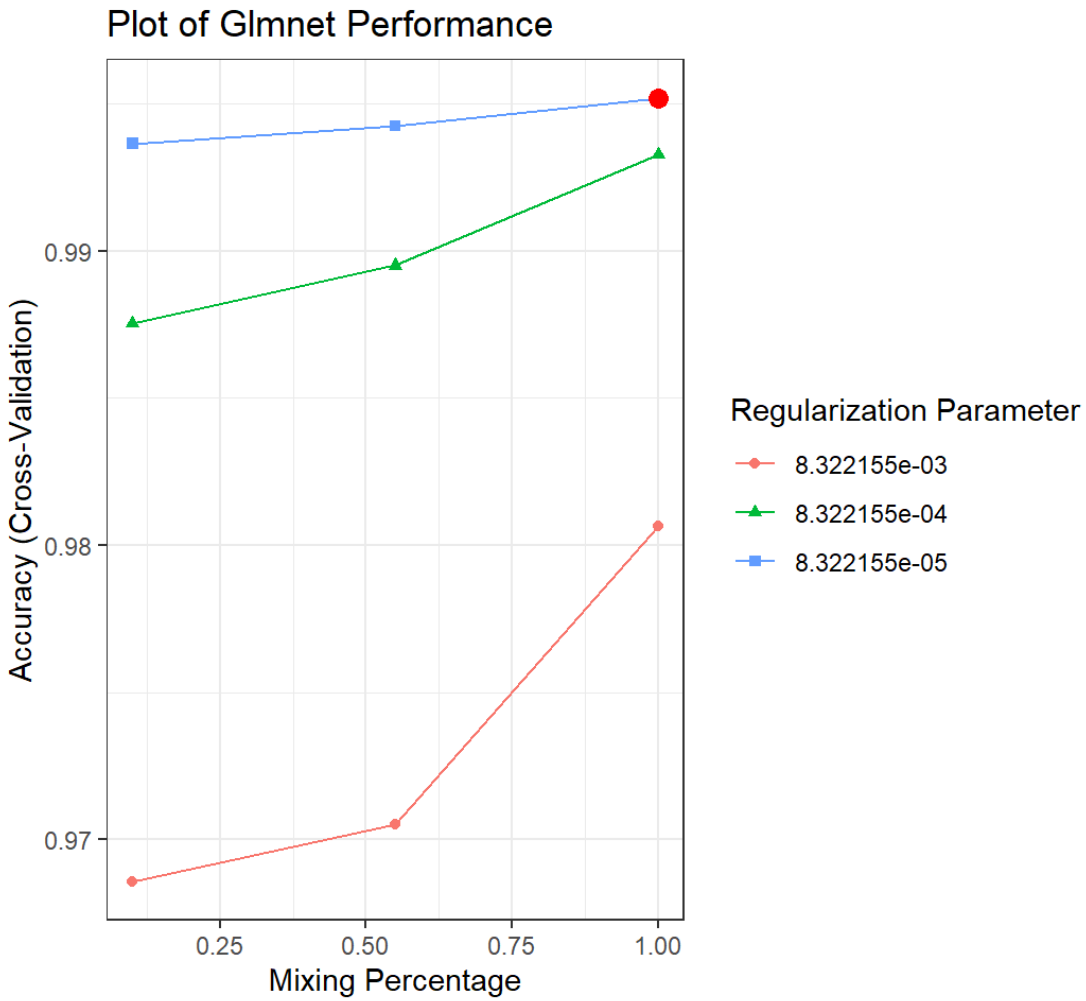
```
#> [1] 0.9676318
```

```
bottom_right.ridge<-table(ridge.pred,data$Tarp_dummy)[2,2]
bottom_left.ridge<-table(ridge.pred,data$Tarp_dummy)[2,1]
top_left.ridge<-table(ridge.pred,data$Tarp_dummy)[1,1]
top_right.ridge<-table(ridge.pred,data$Tarp_dummy)[1,2]

tpr.ridge = bottom_right.ridge/(bottom_right.ridge+top_right.ridge)
fpr.ridge = bottom_left.ridge/(bottom_left.ridge+top_left.ridge)
precision.ridge = bottom_right.ridge/(bottom_right.ridge+bottom_left.ridge)
accuracy.ridge = (top_left.ridge+bottom_right.ridge)/sum(table(ridge.pred,data$Tarp_dummy))
```

3.6.1 Tuning Parameters

As mentioned above, the tuning parameter of λ was determined via 10-fold cross validation over a grid of 100 different values of λ . This was done using the caret package and using a tuning grid of $10^{seq(10,-2,length=100)}$. Plotting the cross validation model accuracy vs. of the ElasticNet Model for different mixes of Ridge vs. Lasso for some of the top performing λ values shows that the Lasso Model performing the best with a λ of $8.322e^{-5}$ with the most accurate performance Model in terms of prediction. The plot is replicated below which displays this. Lasso shrinkage likely performs the best because compared to other mixes of lasso & ridge and just pure ridge regression because it is able to shrink coefficients of unneeded variables and because the number of variables is much less than the number of images.

```
best_index<-as.numeric(rownames(ridge.fit$bestTune[1,0]))
best_alpha<- ridge.fit$bestTune[1,1]
best_lambda<- ridge.fit$bestTune[1,2]
accuracy_lambda_top<-ridge.fit$results[best_index,]$Accuracy
lambda_plot_data <- data.frame(best_alpha,accuracy_lambda_top)

ridge_plot<-  ggplot(ridge.fit, xvar = "lambda") + theme_bw()
print(ridge_plot + ggtitle("Plot of Glmnet Performance")+geom_point(data=lambda_plot_data,
             mapping = aes(x=best_alpha,y=accuracy_lambda_top),
             color='red',
             size=3)
)
```



3.7 Random Forest

The same process is then repeated for for Random Forest here, but similarly to KNN and Penalized logistic there are some tuning parameters that need to be determined. This mtry value determines the Number of variables available for splitting at each tree node. A tuning grid can also be used just as before. Theoretically tuning could be done on the number of trees, but we do know that mtry is the most important tuning metric as the performance variance tends to decrease as number of trees increase. We use 500 trees which can be considered standard.

```
#-- 10-Fold Cross Validation
#10 fold cross validation
set.seed(123)
p=3

train.control.forest <- trainControl(method = "cv", number = 10,
                                     classProbs = TRUE,
                                     savePredictions ="final",search='grid')
tunegrid <-  expand.grid(.mtry = c(p,p/3,p/2,sqrt(p))) #test p/3, p,p/2,sqrt(p)
forest.start<-Sys.time()
forest.fit<-train(Tarp_dummy ~ Red+Green+Blue, data = data,
                  method = "rf",
                  trControl = train.control.forest,
                  TuneGrid = tunegrid,
                  ntree = 500)
```

```
#> note: only 2 unique complexity parameters in default grid. Truncating the grid to 2 .
```

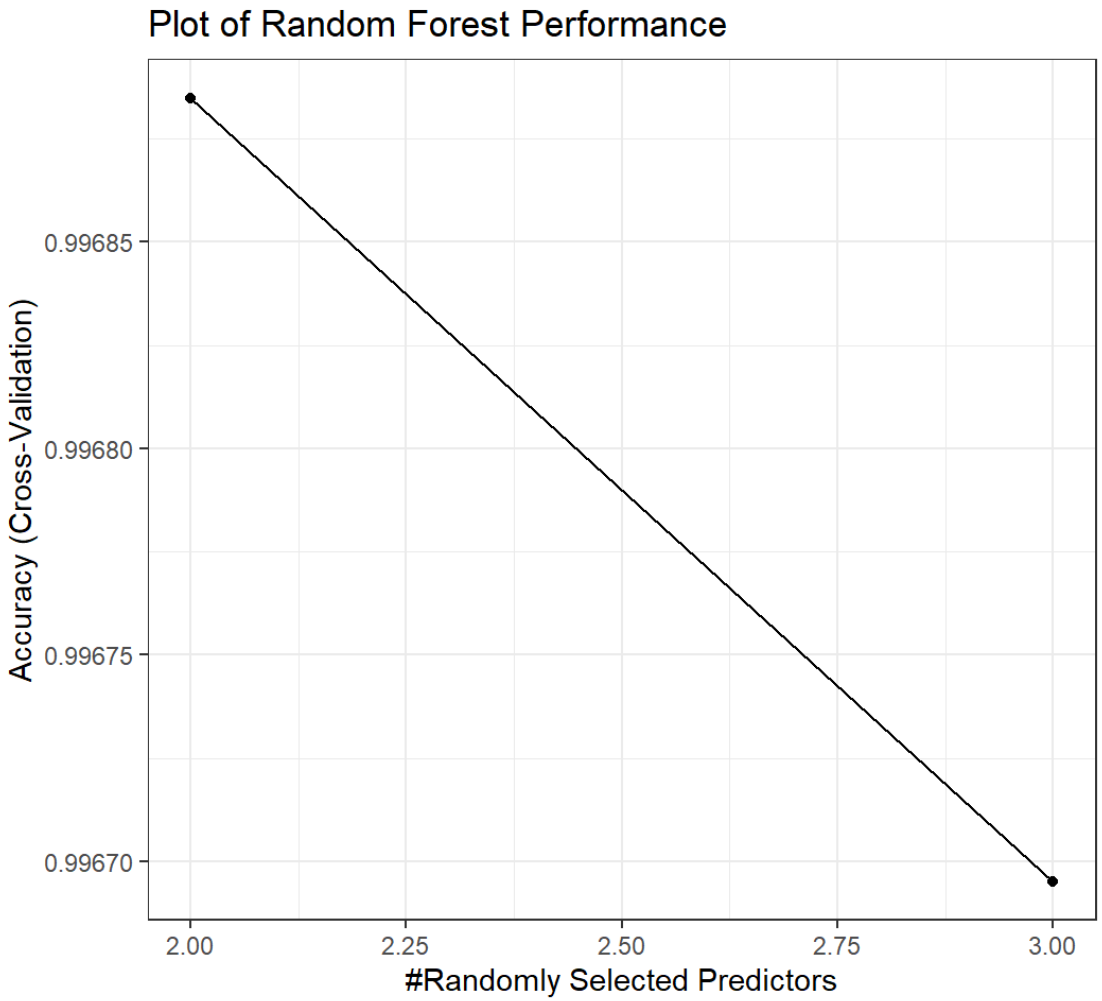
```
forest.fit
```

```
#> Random Forest
#>
#> 63241 samples
#>     3 predictor
#>     2 classes: 'Other', 'Blue_Tarp'
#>
#> No pre-processing
#> Resampling: Cross-Validated (10 fold)
#> Summary of sample sizes: 56917, 56917, 56917, 56917, 56917, 56917, ...
#> Resampling results across tuning parameters:
#>
#>   mtry  Accuracy   Kappa
#>   2     0.9968849  0.9492949
#>   3     0.9966952  0.9462423
#>
#> Accuracy was used to select the optimal model using the largest value.
#> The final value used for the model was mtry = 2.
```

```
forest.end<-Sys.time()
forest.runtime <-forest.end-forest.start
```

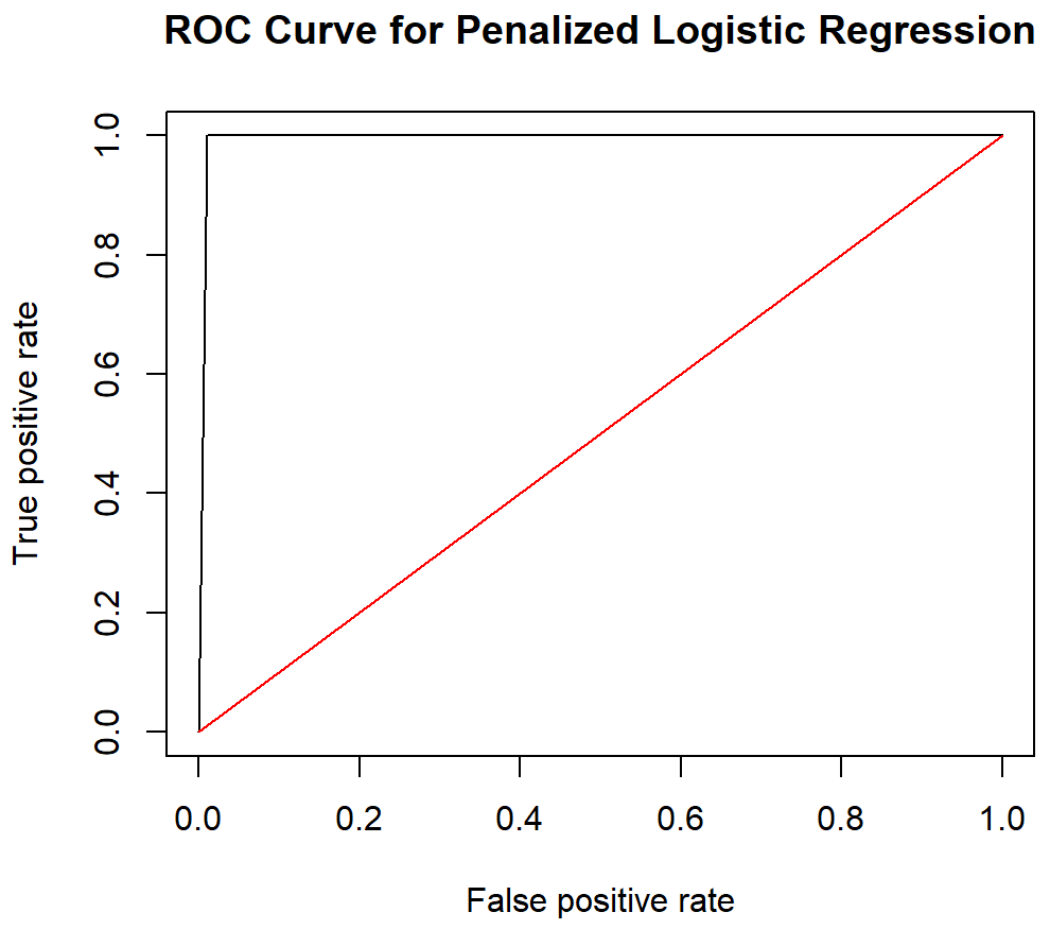
As shown above, the Model with the best accuracy is the model with mtry = 2. Investigated further below

```
best_mtry <- forest.fit$bestTune[1,1]

forest_plot<-  ggplot(forest.fit, xvar = "Randomly Selected Variables") + theme_bw()
print(forest_plot + ggtitle("Plot of Random Forest Performance")
)
```

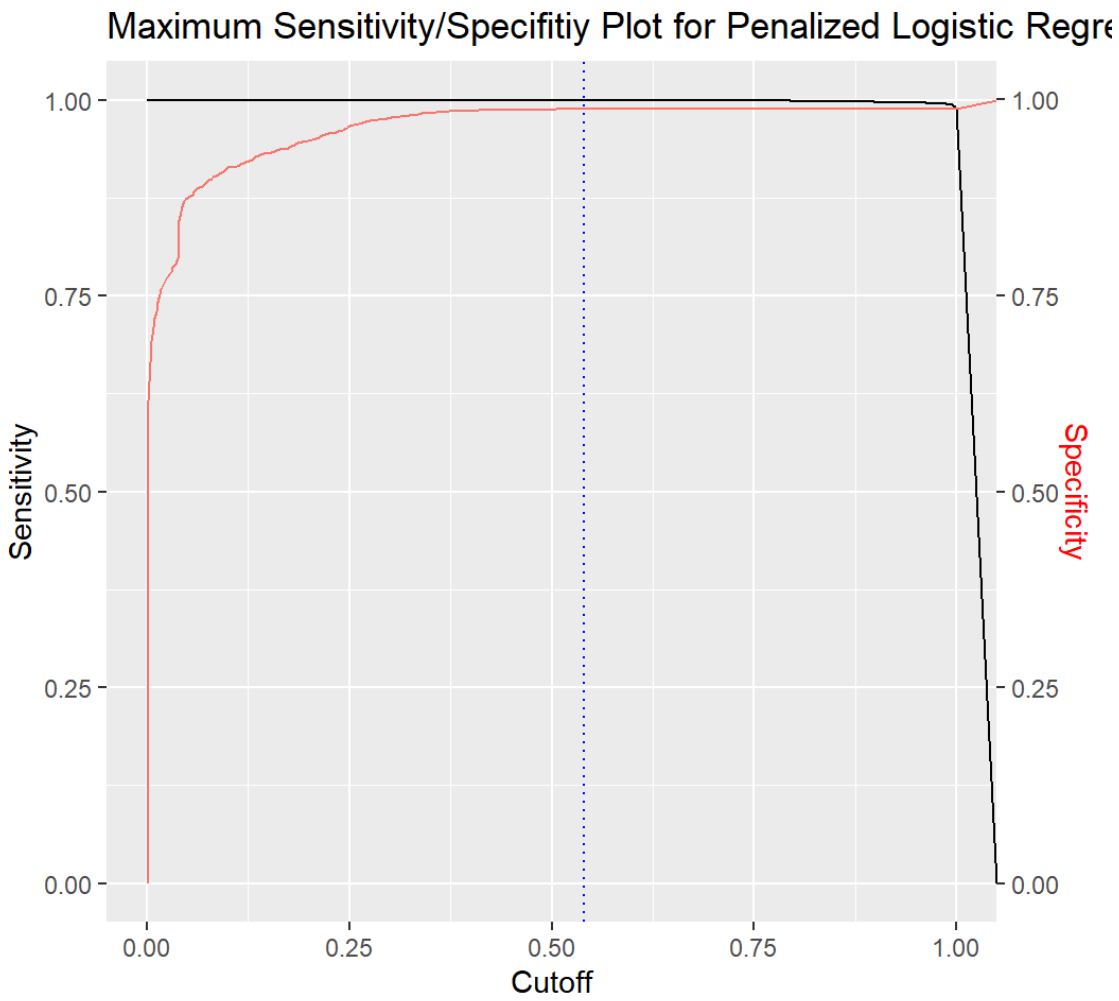


For the Random Forest , the same process was used to evaluate prediction performance as was done for all the other Models: the Model predicts the probability of a class (Blue Tarp or Other) for all the data points and then the ROC plot is plotted. The probability threshold to classify Blue Tarps is chosen using the thersholder function and maximum accuracy threshold, and then applied to the predicted probabilities to determine a confusion matrix and relevant prediction statistics.

```
#--

#fit the best Model

forest.probs<- predict(forest.fit, data, type="prob")
rates.forest <- prediction(1-forest.probs[,2], data$Tarp_dummy)
perf.forest <- performance(rates.forest,"tpr","fpr")

plot(perf.forest, main="ROC Curve for Penalized Logistic Regression")
lines(x = c(0,1), y = c(0,1), col="red")
```




```
auc.forest <- performance(rates.forest, measure = "auc")
auc.forest<-auc.forest@y.values[[1]]



sens.forest <- data.frame(x=unlist(performance(rates.forest, "sens")@x.values),
                  y=unlist(performance(rates.forest, "sens")@y.values))
spec.forest <- data.frame(x=unlist(performance(rates.forest, "spec")@x.values),
                  y=unlist(performance(rates.forest, "spec")@y.values))

thres.forest <-thresholder(forest.fit,threshold = seq(0.1,1,by=.02),final=TRUE,statistics = 'all' )
thres.forest.index=which.max(thres.forest$Accuracy)
threshold.forest <- thres.forest$prob_threshold[thres.forest.index]

forest.plot.threshold<-sens.forest %>% ggplot(aes(x,y)) +
  geom_line() +
  geom_line(data=spec.forest, aes(x,y,col="red")) +
  geom_vline(xintercept = threshold.forest,col="blue",linetype="dotted")+
  scale_y_continuous(sec.axis = sec_axis(~., name = "Specificity")) +
  labs(x='Cutoff', y="Sensitivity") +
  theme(axis.title.y.right = element_text(colour = "red"), legend.position="none")+
  ggtitle("Maximum Sensitivity/Specifitiy Plot for Penalized Logistic Regressions")

plot(forest.plot.threshold)
```




```
forest.pred<-rep("Other",63241)
forest.pred[forest.probs[,2]>threshold.forest]="Tarp"
table(forest.pred,data$Tarp_dummy)
```

```
#>
#> forest.pred Other Blue_Tarp
#>       Other 61218        25
#>       Tarp      1      1997
```

```
mean(forest.pred==data$Tarp_dummy)
```

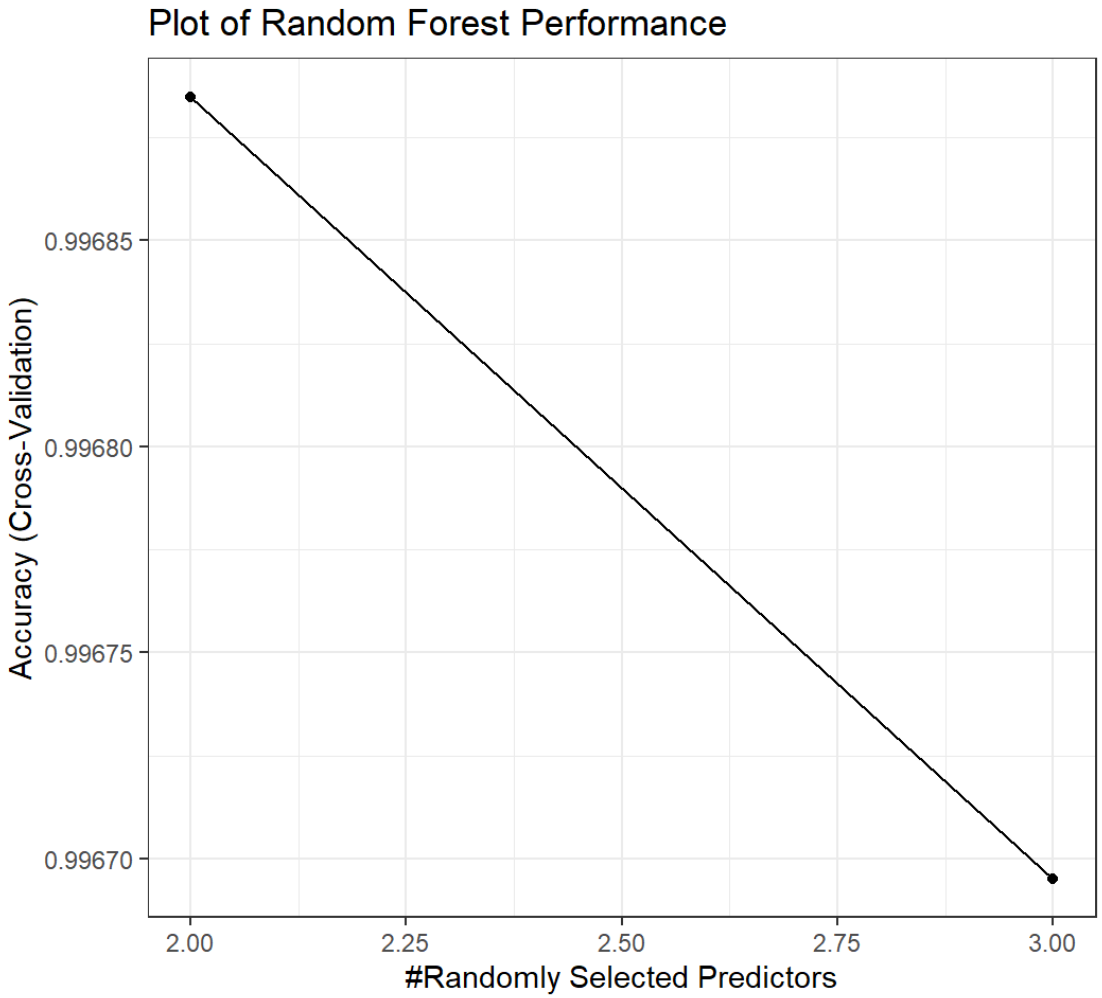
```
#> [1] 0.9680113
```

```
bottom_right.forest<-table(forest.pred,data$Tarp_dummy)[2,2]
bottom_left.forest<-table(forest.pred,data$Tarp_dummy)[2,1]
top_left.forest<-table(forest.pred,data$Tarp_dummy)[1,1]
top_right.forest<-table(forest.pred,data$Tarp_dummy)[1,2]

tpr.forest = bottom_right.forest/(bottom_right.forest+top_right.forest)
fpr.forest = bottom_left.forest/(bottom_left.forest+top_left.forest)
precision.forest = bottom_right.forest/(bottom_right.forest+bottom_left.forest)
accuracy.forest = (top_left.forest+bottom_right.forest)/sum(table(forest.pred,data$Tarp_dummy))
```

3.7.1 Tuning Parameters

As mentioned above, the tuning parameter of mtry was determined via 10-fold cross validation over a grid of multiple different values of mtry. This was done using the caret package and using a tuning grid. Plotting the cross validation model accuracy vs. different number of random variables used at each node shows (this is the mtry variable) that the model with **2** random variables to split at each node is most accurate performance Model in terms of prediction on the training data .The plot is replicated below which displays this. There are only 2 values of mtry used because there are only 3 variables total used in creating the model so can only be a max of 3 and a min of 2 to tune

```
forest_plot<-  ggplot(forest.fit, xvar = "Randomly Selected Variables") + theme_bw()
print(forest_plot + ggtitle("Plot of Random Forest Performance")
)
```



3.8 SVM

A similar process is used here for Support Vector Machines here, but similarly to some of the there other variables there are some tuning parameters that need to be determined. We have to choose the cost function which controls training errors and margins. For example, a small cost creates a large margin (a soft margin) and allows more misclassifications. On the other hand, a large cost creates a narrow margin (a hard margin) and permits fewer misclassifications. The kernel also has to be decided Radial, Linear, or a polynomial type. This set of mathematical functions are used to provide the window to manipulate the data. We do a few different trains to determine the best kernel and cost function using tuneLength of 4 to iterate through. Scaling is not important here and is not considered because the three parameters have the same scale already and therefore it is unnecessary.

```
#-- 10-Fold Cross Validation
#10 fold cross validation
set.seed(123)

svm.start<-Sys.time()
svm.fit.linear<-train(Tarp_dummy ~ Red+Green+Blue, data = data,
                  method = "svmLinear",
                  trControl = train.control,
                  tuneLength = 4 )


set.seed(123)
svm.fit.radial<-train(Tarp_dummy ~ Red+Green+Blue, data = data,
                  method = "svmRadial",
                  trControl = train.control,
                  tuneLength = 4)
set.seed(123)

# this takes a long time so ran once to test and then used radial moving forward.
svm.fit.poly<-train(Tarp_dummy ~ Red+Green+Blue, data = data,
                  method = "svmPoly",
                  trControl = train.control,
                  tuneLength = 4)


svm.end<-Sys.time()
svm.fit.linear
```

```
#> Support Vector Machines with Linear Kernel
#>
#> 63241 samples
#>     3 predictor
#>     2 classes: 'Other', 'Blue_Tarp'
#>
#> No pre-processing
#> Resampling: Cross-Validated (10 fold)
#> Summary of sample sizes: 56917, 56917, 56917, 56917, 56917, 56917, ...
#> Resampling results:
#>
#>   Accuracy   Kappa
#>   0.9953827  0.9223195
#>
#> Tuning parameter 'C' was held constant at a value of 1
```

```
svm.fit.radial
```

```
#> Support Vector Machines with Radial Basis Function Kernel
#>
#> 63241 samples
#>     3 predictor
#>     2 classes: 'Other', 'Blue_Tarp'
#>
#> No pre-processing
#> Resampling: Cross-Validated (10 fold)
#> Summary of sample sizes: 56917, 56917, 56917, 56917, 56917, 56917, ...
#> Resampling results across tuning parameters:
#>
#>   C     Accuracy   Kappa
#>   0.25  0.9967110  0.9460442
#>   0.50  0.9968217  0.9479055
#>   1.00  0.9969482  0.9500479
#>   2.00  0.9970272  0.9513932
#>
#> Tuning parameter 'sigma' was held constant at a value of 8.378477
#> Accuracy was used to select the optimal model using the largest value.
#> The final values used for the model were sigma = 8.378477 and C = 2.
```

```
svm.fit.poly
```

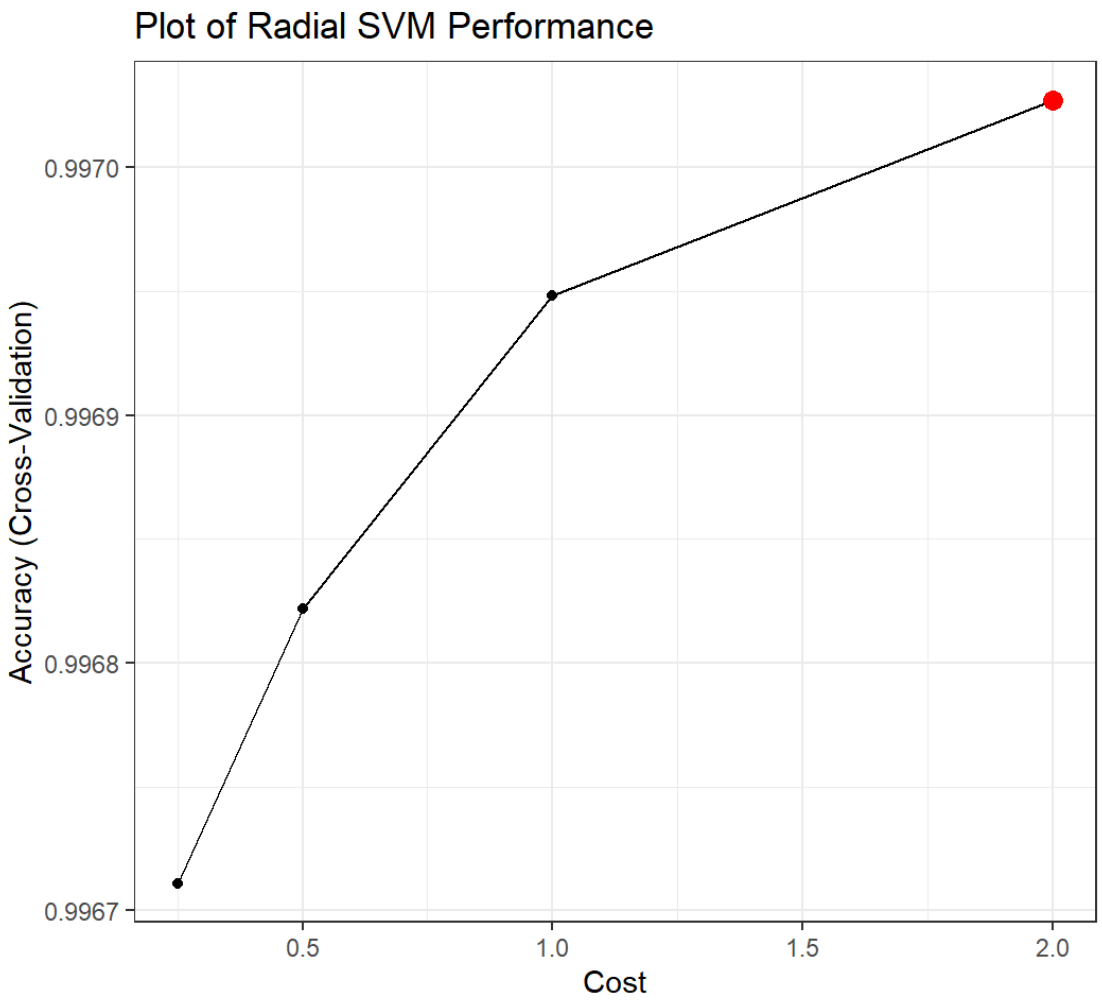
```
#> Support Vector Machines with Polynomial Kernel
#>
#> 63241 samples
#>     3 predictor
#>     2 classes: 'Other', 'Blue_Tarp'
#>
#> No pre-processing
#> Resampling: Cross-Validated (10 fold)
#> Summary of sample sizes: 56917, 56917, 56917, 56917, 56917, 56917, ...
#> Resampling results across tuning parameters:
#>
#>   degree  scale  C     Accuracy   Kappa
#>   1       0.001  0.25  0.9882829  0.8225107
#>   1       0.001  0.50  0.9882355  0.8217855
#>   1       0.001  1.00  0.9882671  0.8222311
#>   1       0.001  2.00  0.9883146  0.8227362
#>   1       0.010  0.25  0.9887731  0.8266893
#>   1       0.010  0.50  0.9895321  0.8336918
#>   1       0.010  1.00  0.9920463  0.8697531
#>   1       0.010  2.00  0.9948925  0.9121850
#>   1       0.100  0.25  0.9950190  0.9146255
#>   1       0.100  0.50  0.9951614  0.9173089
#>   1       0.100  1.00  0.9951930  0.9184302
#>   1       0.100  2.00  0.9952720  0.9201371
#>   1       1.000  0.25  0.9953195  0.9211101
#>   1       1.000  0.50  0.9954144  0.9228025
#>   1       1.000  1.00  0.9953827  0.9223195
#>   1       1.000  2.00  0.9954144  0.9228341
#>   2       0.001  0.25  0.9883304  0.8234090
#>   2       0.001  0.50  0.9883304  0.8234338
#>   2       0.001  1.00  0.9883146  0.8229879
#>   2       0.001  2.00  0.9896270  0.8365653
#>   2       0.010  0.25  0.9950507  0.9160835
#>   2       0.010  0.50  0.9954934  0.9253432
#>   2       0.010  1.00  0.9959362  0.9336534
#>   2       0.010  2.00  0.9958097  0.9312976
#>   2       0.100  0.25  0.9957781  0.9307740
#>   2       0.100  0.50  0.9958255  0.9316284
#>   2       0.100  1.00  0.9958413  0.9318605
#>   2       0.100  2.00  0.9957939  0.9309577
#>   2       1.000  0.25  0.9956674  0.9285832
#>   2       1.000  0.50  0.9957306  0.9295084
#>   2       1.000  1.00  0.9957306  0.9294242
#>   2       1.000  2.00  0.9957781  0.9301339
#>   3       0.001  0.25  0.9884727  0.8259197
#>   3       0.001  0.50  0.9885359  0.8267310
#>   3       0.001  1.00  0.9895637  0.8367274
#>   3       0.001  2.00  0.9948767  0.9120248
#>   3       0.010  0.25  0.9954144  0.9234927
#>   3       0.010  0.50  0.9959046  0.9332383
#>   3       0.010  1.00  0.9958255  0.9316566
#>   3       0.010  2.00  0.9957781  0.9307153
#>   3       0.100  0.25  0.9959362  0.9338717
#>   3       0.100  0.50  0.9959520  0.9341490
#>   3       0.100  1.00  0.9958729  0.9328135
#>   3       0.100  2.00  0.9959678  0.9339043
#>   3       1.000  0.25  0.9962682  0.9388015
#>   3       1.000  0.50  0.9963473  0.9400778
#>   3       1.000  1.00  0.9964738  0.9422585
```

```
#>   3       1.000  2.00  0.9964896  0.9424122
#>
#> Accuracy was used to select the optimal model using the largest value.
#> The final values used for the model were degree = 3, scale = 1 and C = 2.
```

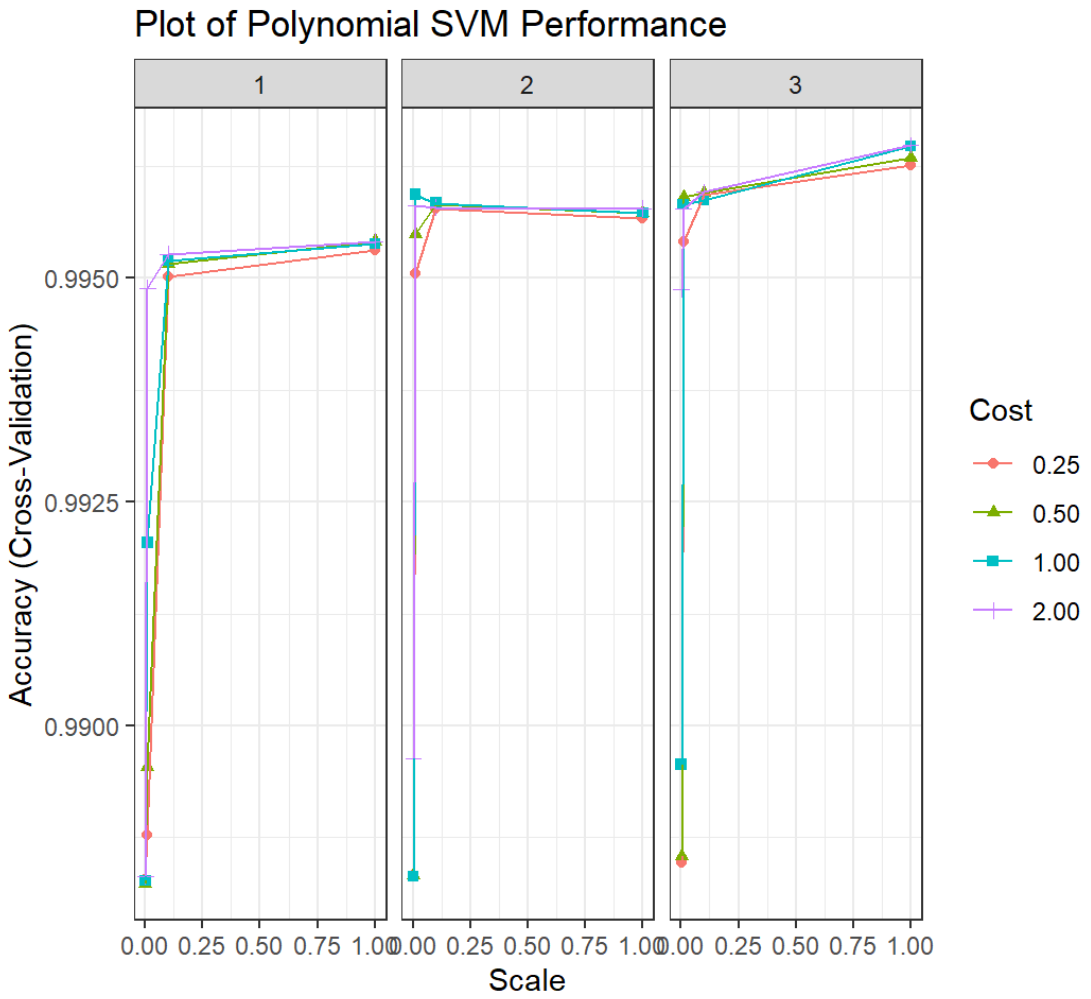
```
svm.runtime <-svm.end-svm.start
```

We an search for the best tune for each, and compare accuracy. Best Accuracy for Linear was with a Cost function of 1, with accuracy of 0.9953827 over ten fold cross-validation. For Radial Kernel, the best tune was with a Cost function of 2 and a sigma value of 8.378 for an accuracy of 0.9970272 (equivalent to the gamma which defines how far the influence of a single training example reaches, with low values meaning 'far' and high values meaning 'close'. The gamma parameters can be seen as the inverse of the radius of influence of samples selected by the model as support vectors.). With a Polynomial Kernel, the best tune was an accuracy of 0.9964896 with a Cost function of 2 and a scale of 1 and a degree of 3. Clearly the Radial best tune performs the best and is the Model we continue with. We can see some plots of the three cross validation runs below

```
best_index.radial<-as.numeric(rownames(svm.fit.radial$bestTune[1,0]))
best_cost<- svm.fit.radial$bestTune[1,2]
best_sigma<- svm.fit.radial$bestTune[1,1]
accuracy_cost_top<-svm.fit.radial$results[best_index.radial,]$Accuracy
cost_plot_data <- data.frame(best_cost,accuracy_cost_top)

svm.radial.plot <- ggplot(svm.fit.radial)+theme_bw()
print(svm.radial.plot +ggtitle("Plot of Radial SVM Performance")+geom_point(data=cost_plot_data,
             mapping = aes(x=best_cost,y=accuracy_cost_top),
             color='red',
             size=3))
```

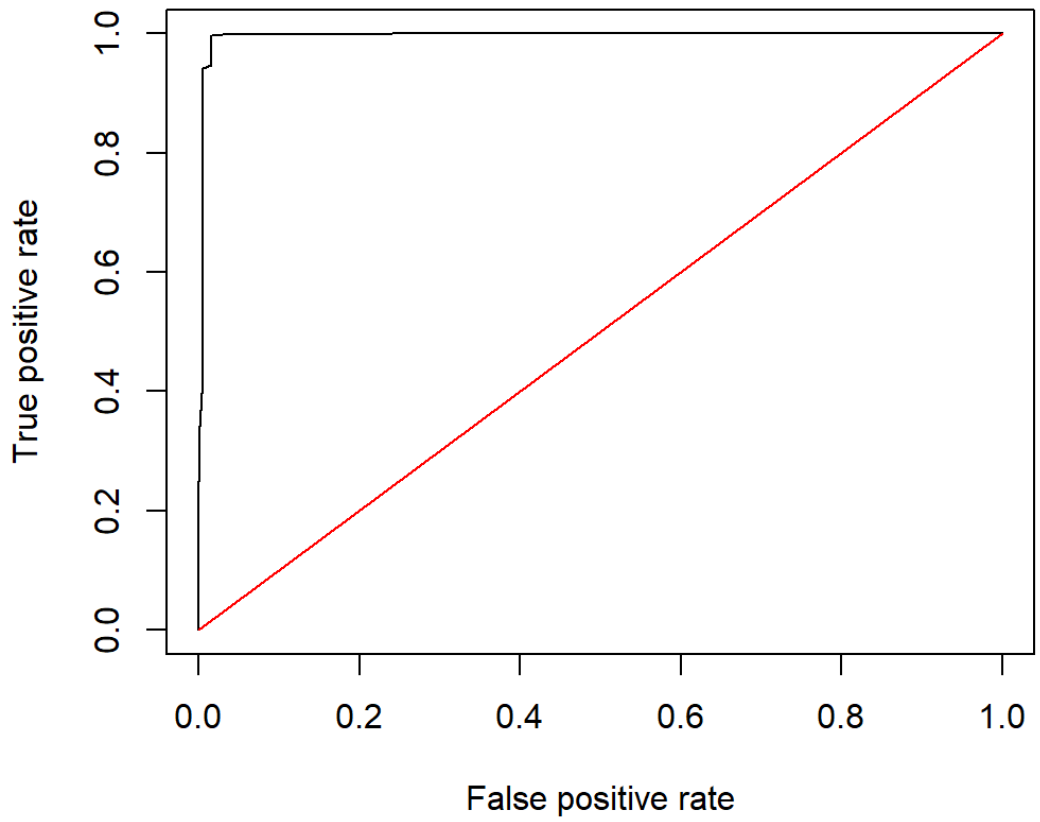



```
svm.poly.plot <- ggplot(svm.fit.poly)+theme_bw()
print(svm.poly.plot +ggtitle("Plot of Polynomial SVM Performance"))
```

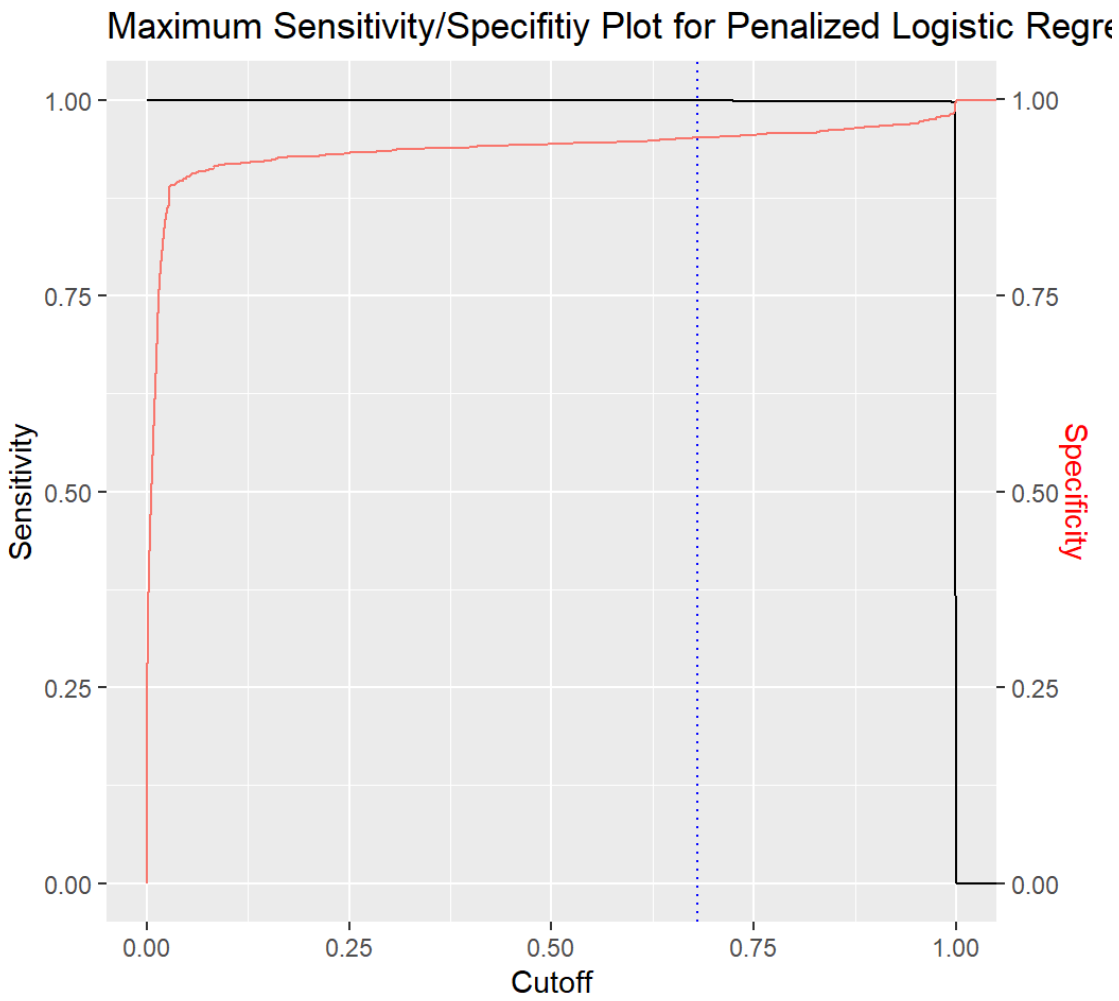


For the Support Vector Machines, the same process was used to evaluate prediction performance as was done for all the other Model: the Model predicts the probability of a class (Blue Tarp or Other) for all the data points and then the ROC plot is plotted. The probability threshold to classify Blue Tarps is chosen using the geometric mean and then applied to the predicted probabilities to determine a confusion matrix and relevant prediction statistics.

```
#--

#fit the best Model

svm.probs<- predict(svm.fit.radial, data, type="prob")
rates.svm <- prediction(1-svm.probs[,2], data$Tarp_dummy)
perf.svm <- performance(rates.svm,"tpr","fpr")

plot(perf.svm, main="ROC Curve for Penalized Logistic Regression")
lines(x = c(0,1), y = c(0,1), col="red")
```




```
auc.svm <- performance(rates.svm, measure = "auc")
auc.svm<-auc.svm@y.values[[1]]




sens.svm <- data.frame(x=unlist(performance(rates.svm, "sens")@x.values),
                       y=unlist(performance(rates.svm, "sens")@y.values))
spec.svm <- data.frame(x=unlist(performance(rates.svm, "spec")@x.values),
                       y=unlist(performance(rates.svm, "spec")@y.values))

thres.radial <-thresholder(svm.fit.radial,threshold = seq(0.1,1,by=.02),final=TRUE,statistics = 'all' )
thres.radial.index=which.max(thres.radial$Accuracy)
threshold.svm <- thres.radial$prob_threshold[thres.radial.index]

svm.plot.threshold<-sens.svm %>% ggplot(aes(x,y)) +
  geom_line() +
  geom_line(data=spec.svm, aes(x,y,col="red")) +
  geom_vline(xintercept = threshold.svm,col="blue",linetype="dotted")+
  scale_y_continuous(sec.axis = sec_axis(~., name = "Specificity")) +
  labs(x='Cutoff', y="Sensitivity") +
  theme(axis.title.y.right = element_text(colour = "red"), legend.position="none")+
  ggtitle("Maximum Sensitivity/Specifitiy Plot for Penalized Logistic Regressions")

plot(svm.plot.threshold)
```




```
svm.pred<-rep("Other",63241)
svm.pred[svm.probs[,2]>threshold.svm]="Tarp"
table(svm.pred,data$Tarp_dummy)
```

```
#>
#> svm.pred Other Blue_Tarp
#>    Other 61165       127
#>    Tarp     54      1895
```

```
mean(svm.pred==data$Tarp_dummy)
```

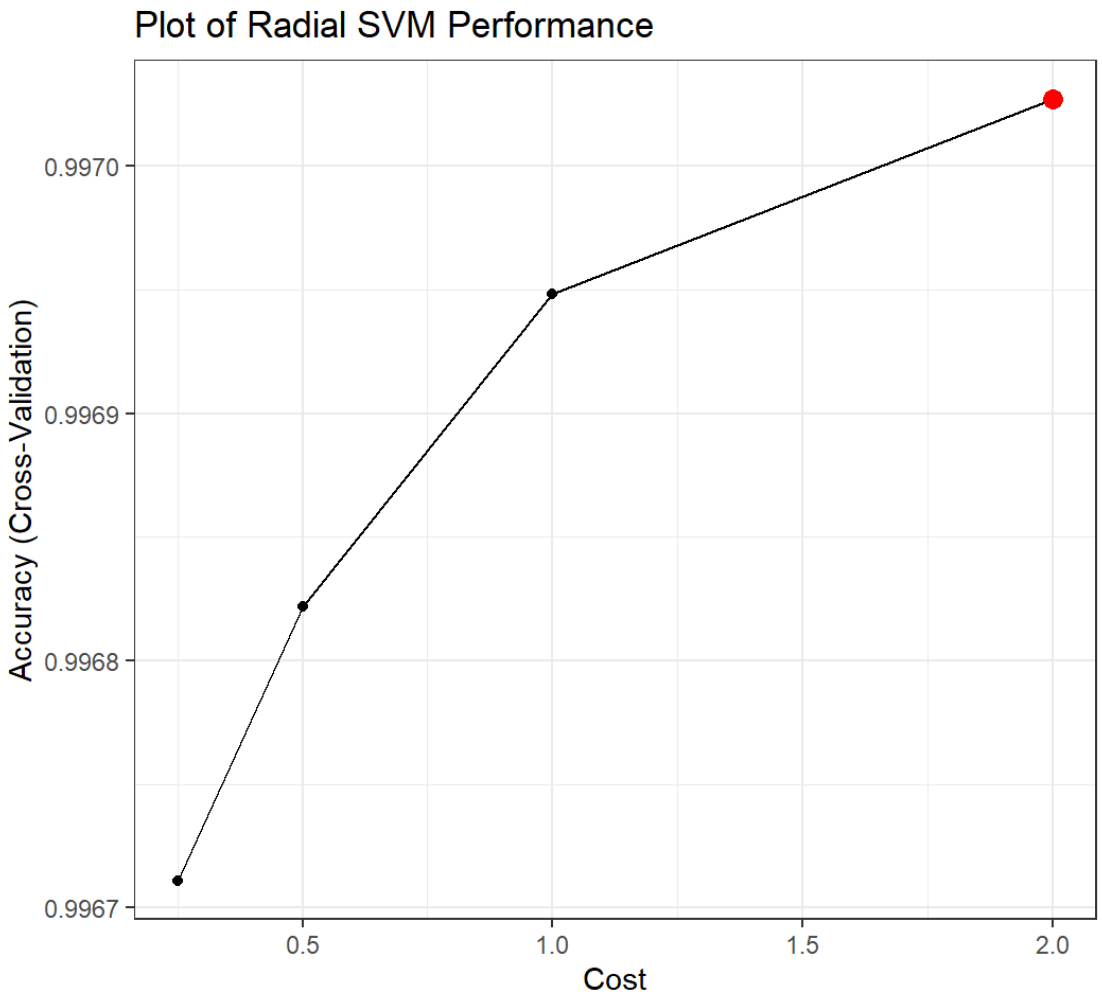
```
#> [1] 0.9671732
```

```
bottom_right.svm<-table(svm.pred,data$Tarp_dummy)[2,2]
bottom_left.svm<-table(svm.pred,data$Tarp_dummy)[2,1]
top_left.svm<-table(svm.pred,data$Tarp_dummy)[1,1]
top_right.svm<-table(svm.pred,data$Tarp_dummy)[1,2]

tpr.svm = bottom_right.svm/(bottom_right.svm+top_right.svm)
fpr.svm = bottom_left.svm/(bottom_left.svm+top_left.svm)
precision.svm = bottom_right.svm/(bottom_right.svm+bottom_left.svm)
accuracy.svm = (top_left.svm+bottom_right.svm)/sum(table(svm.pred,data$Tarp_dummy))
```

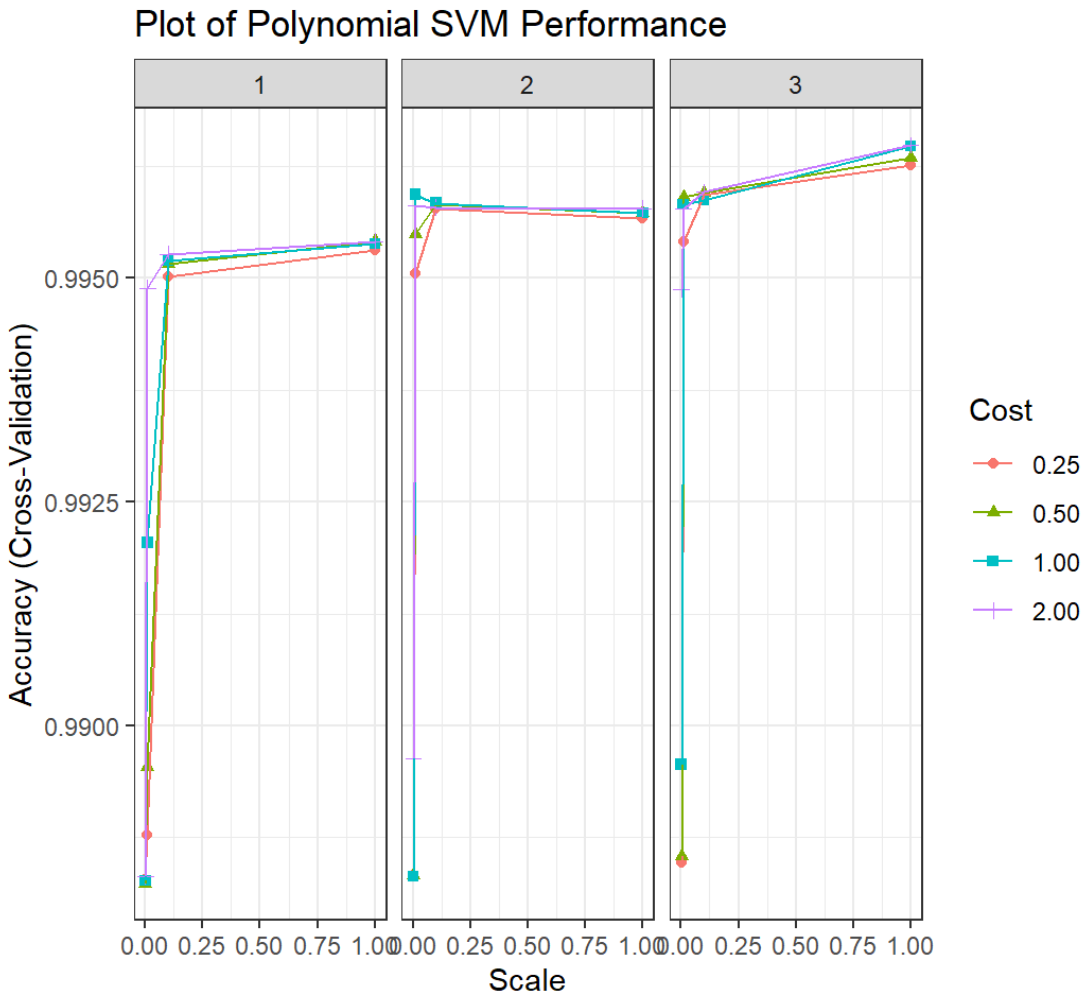
3.8.1 Tuning Parameters

As mentioned above, the tuning parameter of the Cost Function was determined via 10-fold cross validation over a grid of multiple different values and kernels for the Support vector machine . This was done using the caret package and using a tuning grid and running multiple different trains for each kernel. For radial kernel, there is the cost function to optimize and the gamma (sigma value) to optimize. The polynomial kernel has the degree and the cost function to optimize. After evaluating all the kernels (linear, radial, and polynomial) and multiple tuned values, it was determine the radial kernel was the most accurate performing with Cost function of 2 and a sigma value of 8.378. The plot for radial and polynomial kernels are replicated below which displays this.

```
svm.radial.plot <- ggplot(svm.fit.radial)+theme_bw()
print(svm.radial.plot +ggtitle("Plot of Radial SVM Performance")+geom_point(data=cost_plot_data,
             mapping = aes(x=best_cost,y=accuracy_cost_top),
             color='red',
             size=3))
```



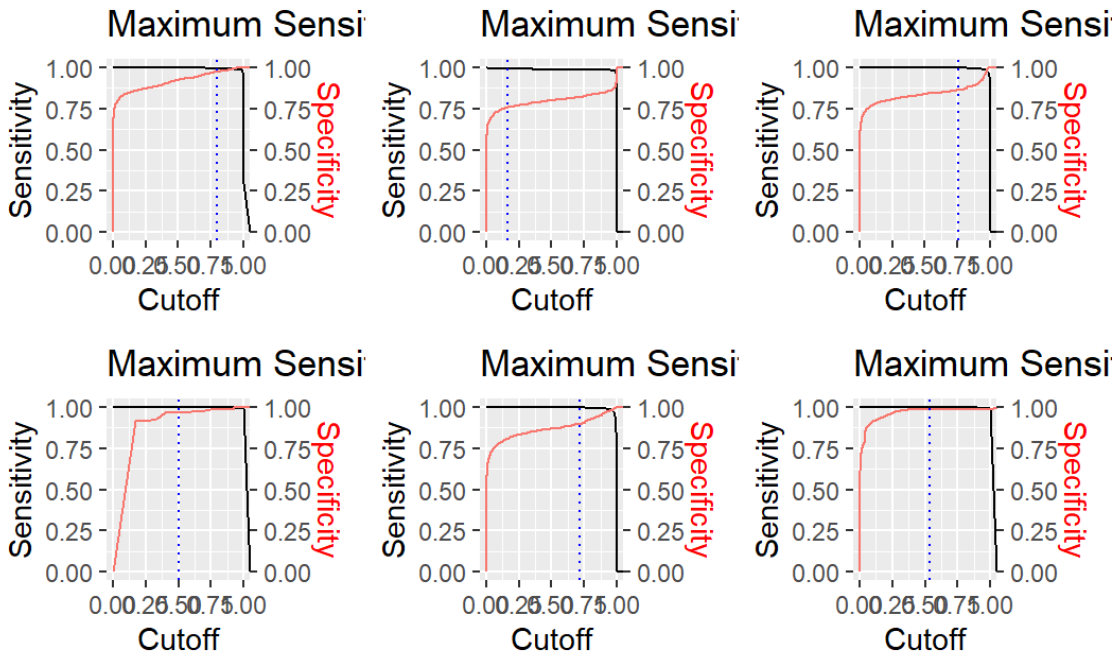

```
svm.poly.plot <- ggplot(svm.fit.poly)+theme_bw()
print(svm.poly.plot +ggtitle("Plot of Polynomial SVM Performance"))
```



3.9 Threshold Selection

As mentioned previously, each Model trained using caret and cross validation, when used for prediction, predicts a probability of each observation being assigned to a class. In this context, each image is assigned a predicted probability, based on the Red, Green, & Blue luminosity values, that the image contains the Blue Tarp or Other Tars. Because of these probabilities, in order to evaluate predictive performance a “threshold” for probability values needs to be chosen to assign each image to the “Blue Tarp” class or the “Other” class. To do this, the thresholder function from caret is used to resample each Model and test different thresholds values to determine the maximal. We chose the one that maximizes the model accuracy which is what we are looking for here to most accurately predict where blue tarps are and where blue tarps aren’t. This prioritizes resources as best as possible to help the survivors. See below for each chart and the thresholds themselves can be seen in the table:

```
figure <- ggarrange(glm.plot.threshold, lda.plot.threshold, qda.plot.threshold,knn.plot.threshold,
                    ridge.plot.threshold,forest.plot.threshold,
                    ncol = 3, nrow = 3)

plot(figure)
```



4 Results (Cross-Validation)

The results from all the Model's cross validation and prediction performances are accumulated and displayed in the table below using the kableExtra package. This table combines the:

```
 1. Tuning parameters: the k chosen for K-Nearest Neighbors and the $\lambda$ for the Penalized Logistic Regr
ession)

 2. Area Under the ROC Curve (AUROC) which describes the  ability to disriminate between classes. THe AUROC c
loser to one is a stronger   predictive performance. An AUROC of 0.8 means that the model has good discrimina
tory ability: 80% of the time, the model will correctly   assign a Blue Tarp to a randomly selected image wit
h a Blue Tarp than to a randomly selected image without a Blue Tarp.

 3. Threshold: The probability threshold chosen to assign each image to the Blue Tarp class based on the Mode
l's predicted probability

 4. Accuracy: Overall, how often is the classifier correct? (True Predicted Blue Tarps +True Predicted Negati
ve Tarps)/Total Images

 5. True Positive Rate (TPR): # True positives (Actual Blue Tarps) / # positives (Predicted Blue Tarps)

 6. False Positive Rate (FPR): # False Positives (Falsely Predicted Blue Tarp Images) / # negatives (Actual O
ther Images)

 7. Precision: # True positives / # predicted positives
```

```
results<-data.frame("Model" = c("Log Reg","LDA","QDA","KNN","Penalized Log Reg",
                                "Random Forest (ntrees=500)","SVM (kernel=Radial)"),
                    "Tuning"= c("n/a","n/a","n/a",best_k,best_lambda,best_mtry,best_cost),
                    "AUROC"=c(0,0,0,0,0,0,0),"Threshold"=c(0,0,0,0,0,0,0),"Accuracy"=c(0,0,0,0,0,0,0),
                    "TPR"=c(0,0,0,0,0,0,0),"FPR"=c(0,0,0,0,0,0,0),"Precision"=c(0,0,0,0,0,0,0))

results[,4]=c(threshold.glm, threshold.lda,threshold.qda,threshold.knn,
              threshold.ridge,threshold.forest,threshold.svm)

results[,3]=c(auc.glm,auc.lda,auc.qda,auc.knn,auc.ridge,auc.forest,auc.svm)

results[,5]=c(accuracy.glm,accuracy.lda,accuracy.qda,accuracy.knn,
              accuracy.ridge,accuracy.forest,accuracy.svm)
results[,6]=c(tpr.glm,tpr.lda,tpr.qda,tpr.knn,tpr.ridge,tpr.forest,tpr.svm)

results[,7]=c(fpr.glm,fpr.lda,fpr.qda,fpr.knn,fpr.ridge,fpr.forest,fpr.svm)

results[,8]=c(precision.glm,precision.lda,precision.qda,precision.knn,
              precision.ridge,precision.forest,precision.svm)


results  %>%
  kbl(caption = "Cross-Validation Results") %>%
  kable_styling(bootstrap_options = c("striped", "hover", "condensed", "responsive"))
```

Cross-Validation Results

Model	Tuning	AUROC	Threshold	Accuracy	TPR	FPR	Precision
Log Reg	n/a	0.9995442	0.80	0.9949716	0.8644906	0.0007187	0.9754464
LDA	n/a	0.9888768	0.16	0.9827802	0.8397626	0.0124961	0.6894032
QDA	n/a	0.9982175	0.76	0.9936592	0.8046489	0.0000980	0.9963258
KNN	3	0.9998980	0.50	0.9981025	0.9683482	0.0009147	0.9721946
Penalized Log Reg	8.32215451789909e-05	0.9985033	0.72	0.9944024	0.8372898	0.0004084	0.9854482
Random Forest (ntrees=500)	2	0.9944970	0.54	0.9995889	0.9876360	0.0000163	0.9994995
SVM (kernel=Radial)	2	0.9960265	0.68	0.9971379	0.9371909	0.0008821	0.9722935

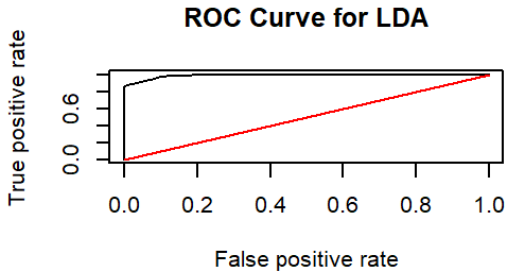
Because the Caret Package was used to Train the Models over cross validation, the following list below describes how each of the performance metrics were calculated. Overall, the final Model chosen through the Caret package was used to predict the classes of each image. This final Model is based on the mean of all the resamplings in 10-fold cross validation and then refit on the total dataset used.

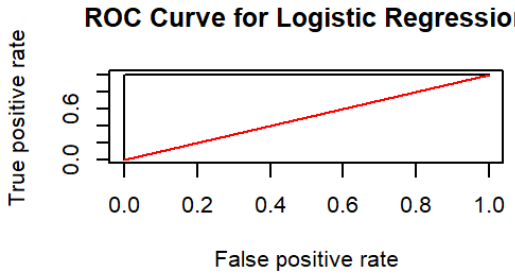
```
  1.Tuning: The Tuning chosen for the best Model as determined by the Caret Model. This is the K for the best performing Model in   K-Nearest Neighbors & the best lambda value for the Penalized Logistic Regression

  2. AUROC: This is the AUROC for the based on the final Model (see above as the average of all the cross validations) with different probability thresholds.

  3. Threshold: Best Probability Threshold

  4. Accuracy, TPR, FPR & Precision are based on the final Model (average of all the cross validations) using the maximum probabilty threshold. So this this just the Maxmium number for the final model
```

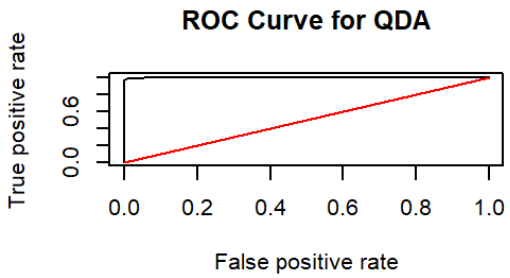
4.1 ROC Curves

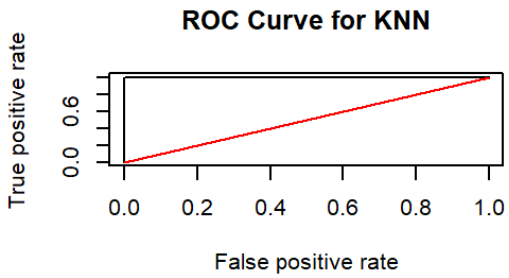
The ROC is *receiver operating characteristic curve*, is a graphical plot that illustrates the diagnostic ability of a binary classifier system as its discrimination threshold is varied. In layman terms, for the final Model used in each Model fit to all the image data, the probability threshold of determining whether the image has a blue tarp or not is varied for all possible variables and shows the True Positive Rate & False Positive Rate. A ROC further to the top left (higher True Positive and lower false positive rates) is best. ROC curves are shown below and all have been created using in-sample data. This is because the data used to fit the Model is what the ROC curve is evaluating in this situation. It is not predicting new values on new data not used to fit the Model. This means that the data used in the RO curves is the totality of the imagery data from the Haitian Earthquake crisis used in fitting the Models (without the Hold Out set that will be introduced later)

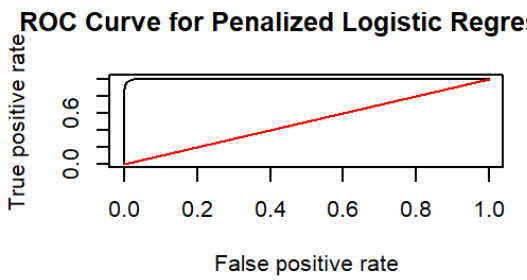
```
par(mfrow = c(3, 2))


plot(perf.lda, main="ROC Curve for LDA")
lines(x = c(0,1), y = c(0,1), col="red")

plot(perf.glm, main="ROC Curve for Logistic Regression")
lines(x = c(0,1), y = c(0,1), col="red")

plot(perf.qda, main="ROC Curve for QDA")
lines(x = c(0,1), y = c(0,1), col="red")

plot(perf.knn, main="ROC Curve for KNN")
lines(x = c(0,1), y = c(0,1), col="red")

plot(perf.ridge, main="ROC Curve for Penalized Logistic Regression")
lines(x = c(0,1), y = c(0,1), col="red")

plot(perf.forest, main="ROC Curve for Random Forest")
lines(x = c(0,1), y = c(0,1), col="red")
```

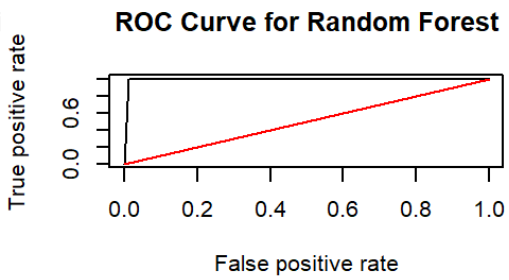




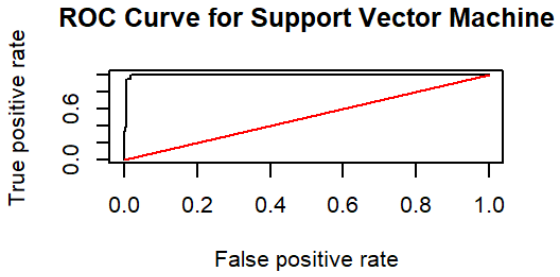








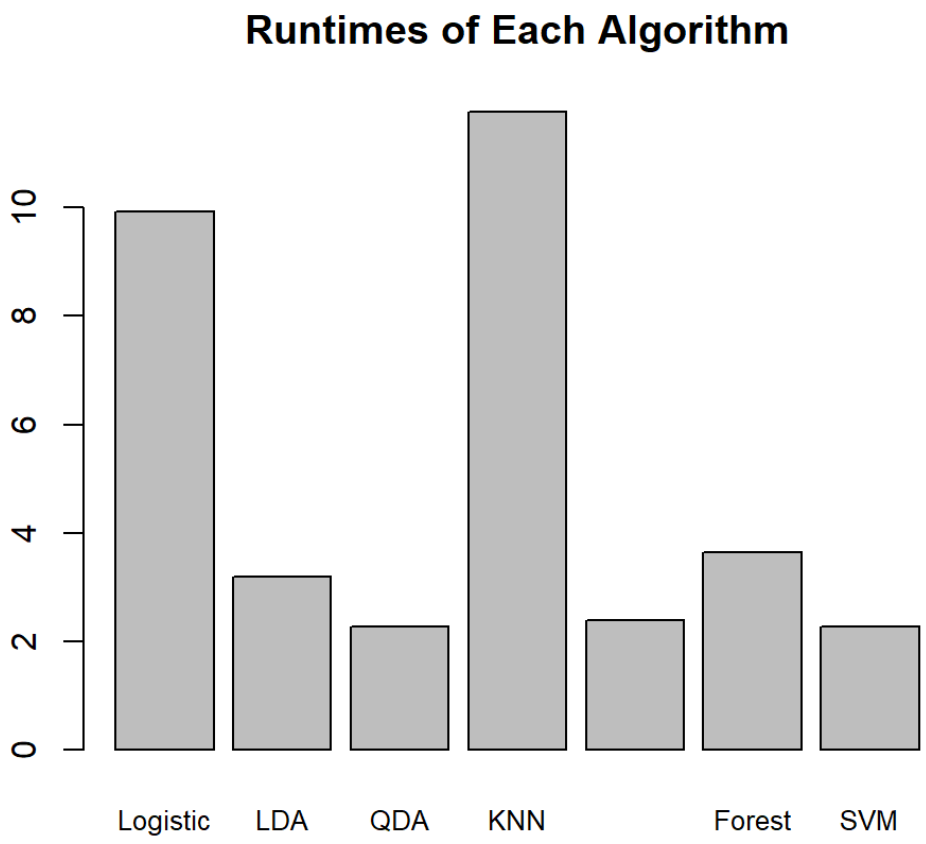

```
plot(perf.svm, main="ROC Curve for Support Vector Machine")
lines(x = c(0,1), y = c(0,1), col="red")
```



Time Complexity

Below is a bar chart comparing the time complexity of each Model Run. As you can see, KNN, Random Forest & SVM are by far the longest and worse performing when running 10-Fold Cross Validation & Tuning

```
runtimes<-data_frame("Model"=c('Logistic','LDA','QDA','KNN','Pen Logistic','Forest','SVM'),
                     "time"=c(as.numeric(glm.runtime),as.numeric(lda.runtime),as.numeric(qda.runtime),
                              as.numeric(knn.runtime),as.numeric(ridge.runtime),as.numeric(forest.runtime),
                              as.numeric(svm.runtime)))


barplot(runtimes$time, main="Runtimes of Each Algorithm",
        horiz=FALSE, names.arg=runtimes$Model, cex.names=0.8)
```



5 Hold-out Data / EDA

Now we want to load the hold out data. This will be used to evaluate the predictive performance of our trained models on a new set of test or hold-out data. We first have to load the data files which are not in a clean format. Working through the fixed widths using regular expressions to find the widths for each file allows loading into a list of dataframes. Each is then reduced to just the last 3 columns which contain the luminosity values (these are still unknown which refers to which). Additionally, there was one duplicate file ignored. Based on the names, values for the dummy class variable are assigned either “Blue_tarp” or “Other” to refer to what is in the file.

```
#-- Load Data
data.dir <- "C:/Users/jkatz/Desktop/UVA/Semester 2/DS 6030/Project 2"
files <-list.files(path=data.dir, pattern="\\.txt", full.names=T)
files_raw <-list.files(path=data.dir, pattern="\\.txt", full.names=F)
hdr <- c('ID','X','Y','Map.X','Map.Y','Lat','Lon','B1','B2','B3')
one.line <- lapply(files, function(x) readLines(x, n = 9)[9]) # char string with 1st line of data
widths<-data.frame()
for (i in 1:length(one.line)){
tmp <- diff(c(0, gregexpr("\\S(?=\\s)", paste(one.line[i], ""), perl = TRUE)[[1]]))
widths<-rbind(widths,tmp)
}


data_load<-lapply(files, function(x) read.fwf(x, widths[match(x,files),], skip = 8,col.names=hdr))
names(data_load) <- files_raw


orthovnir057_ROI_NON_Blue_Tarps.txt <-data_load[1]$orthovnir057_ROI_NON_Blue_Tarps.txt[,c(8,9,10)]
orthovnir067_ROI_Blue_Tarps.txt <-data_load[2]$orthovnir067_ROI_Blue_Tarps.txt[,c(8,9,10)]
orthovnir067_ROI_NOT_Blue_Tarps.txt <-data_load[3]$orthovnir067_ROI_NOT_Blue_Tarps.txt[,c(8,9,10)]
orthovnir069_ROI_Blue_Tarps.txt <-data_load[4]$orthovnir069_ROI_Blue_Tarps.txt[,c(8,9,10)]
orthovnir069_ROI_NOT_Blue_Tarps.txt <-data_load[5]$orthovnir069_ROI_NOT_Blue_Tarps.txt[,c(8,9,10)]
orthovnir078_ROI_Blue_Tarps.txt <-data_load[6]$orthovnir078_ROI_Blue_Tarps.txt[,c(8,9,10)]
orthovnir078_ROI_NON_Blue_Tarps.txt <-data_load[7]$orthovnir078_ROI_NON_Blue_Tarps.txt[,c(8,9,10)]

orthovnir057_ROI_NON_Blue_Tarps.txt$Class <-"Other"
orthovnir067_ROI_Blue_Tarps.txt$Class <-"Blue_Tarp"
orthovnir067_ROI_NOT_Blue_Tarps.txt$Class <-"Other"
orthovnir069_ROI_Blue_Tarps.txt$Class <-"Blue_Tarp"
orthovnir069_ROI_NOT_Blue_Tarps.txt$Class <-"Other"
orthovnir078_ROI_Blue_Tarps.txt$Class <- "Blue_Tarp"
orthovnir078_ROI_NON_Blue_Tarps.txt$Class <-"Other"

hold_out_data <- rbind(orthovnir057_ROI_NON_Blue_Tarps.txt,orthovnir067_ROI_Blue_Tarps.txt
                       ,orthovnir067_ROI_NOT_Blue_Tarps.txt,orthovnir069_ROI_Blue_Tarps.txt
                       ,orthovnir069_ROI_Blue_Tarps.txt
                       ,orthovnir078_ROI_Blue_Tarps.txt,orthovnir078_ROI_NON_Blue_Tarps.txt)

hold_out_data$Tarp_dummy <- ifelse(hold_out_data$Class == "Blue_Tarp", 1, 0)

#check to see if a factor and if not convert
is.numeric(hold_out_data$Tarp_dummy)
```

```
#> [1] TRUE
```

```
hold_out_data$Tarp_dummy<-factor(hold_out_data$Tarp_dummy)
contrasts(hold_out_data$Tarp_dummy)
```

```
#>   1
#> 0 0
#> 1 1
```

```
#set the levels
levels(hold_out_data$Tarp_dummy) <- c("Other","Blue_Tarp")
is.factor(hold_out_data$Tarp_dummy)
```

```
#> [1] TRUE
```

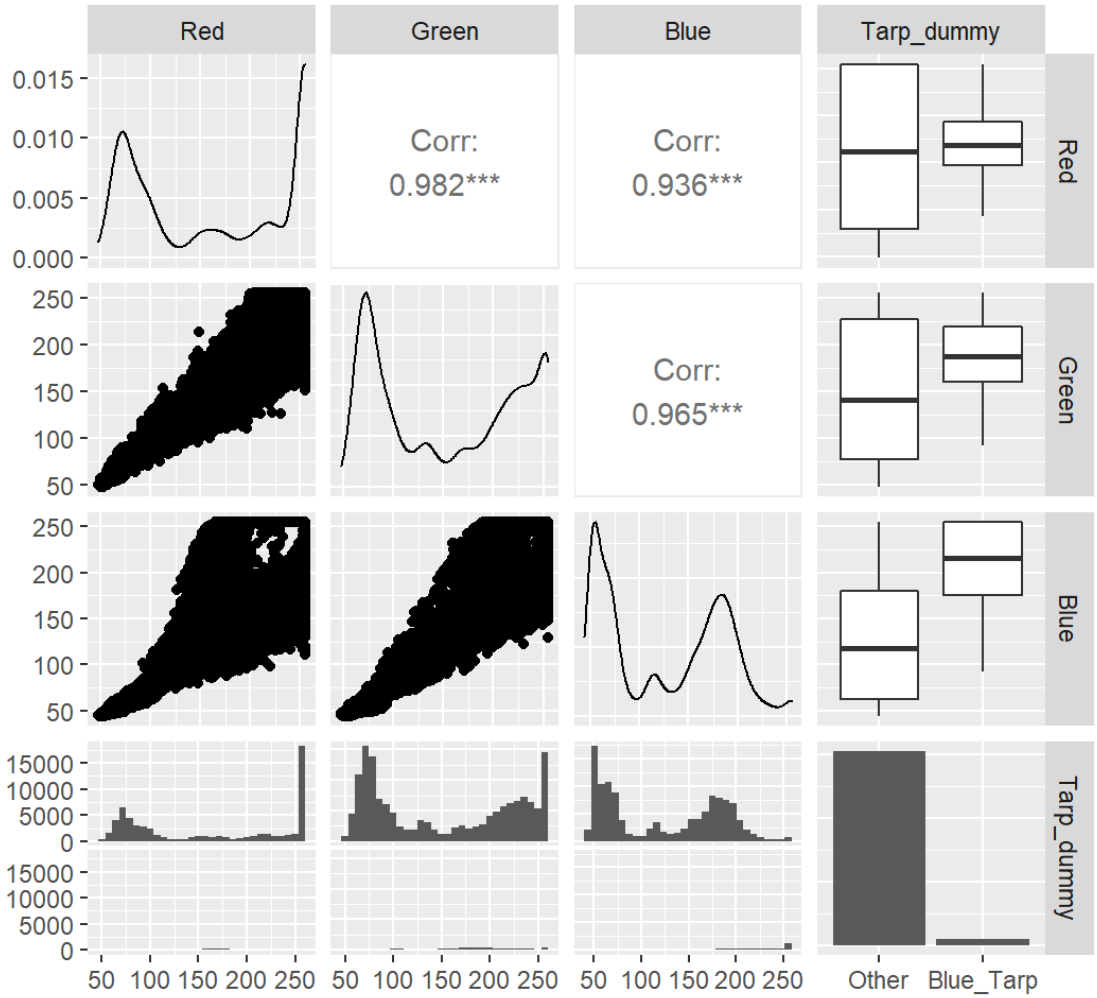
```
contrasts(hold_out_data$Tarp_dummy)
```

```
#>           Blue_Tarp
#> Other             0
#> Blue_Tarp         1
```

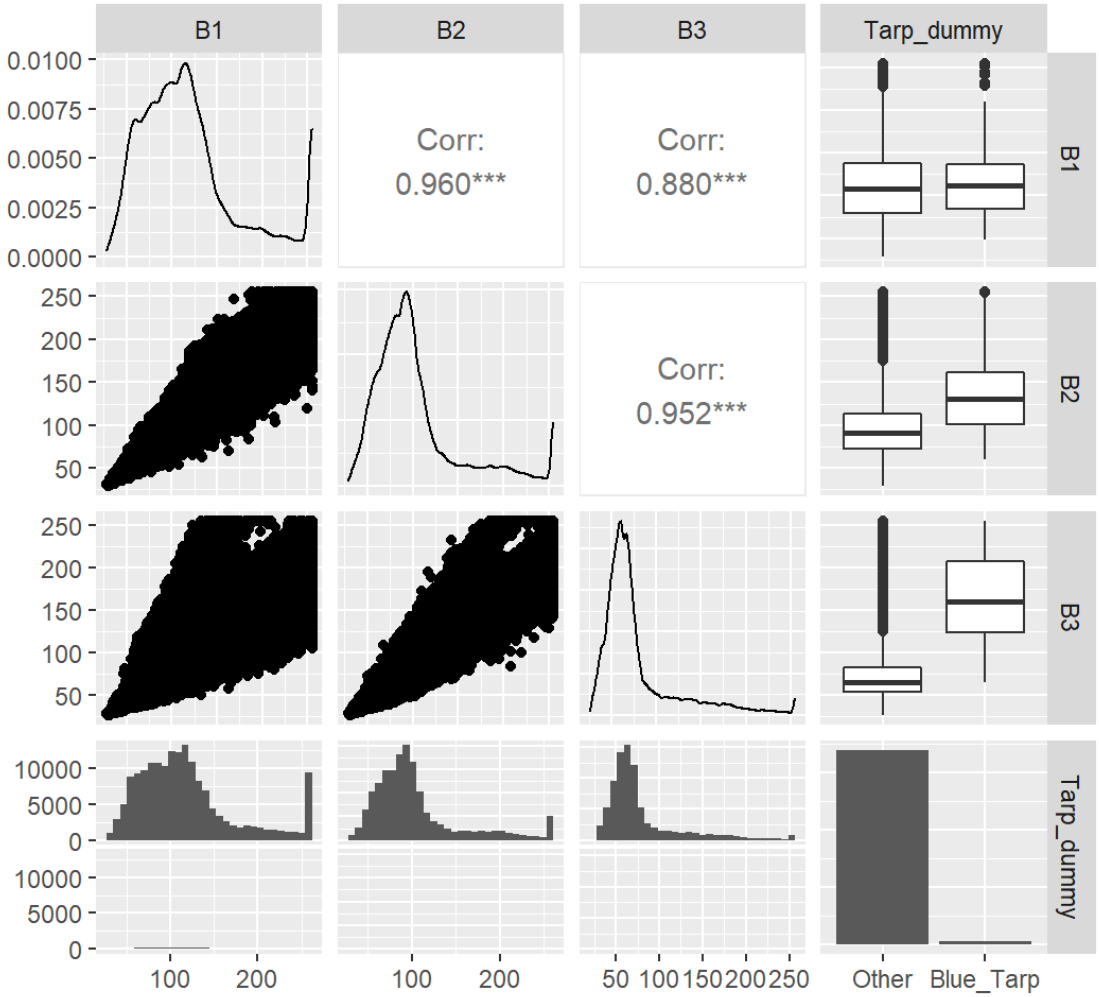
```
hold_out_data <-hold_out_data[,c(4,1,2,3,5)]
```

Let's compare the distributions of the two datasets to see if we can tell if B1 = Red, B2 = Green & B3 = Blue.

```
#compare
ggpairs(data[,2:5])
```




```
ggpairs(hold_out_data[,2:5])
```



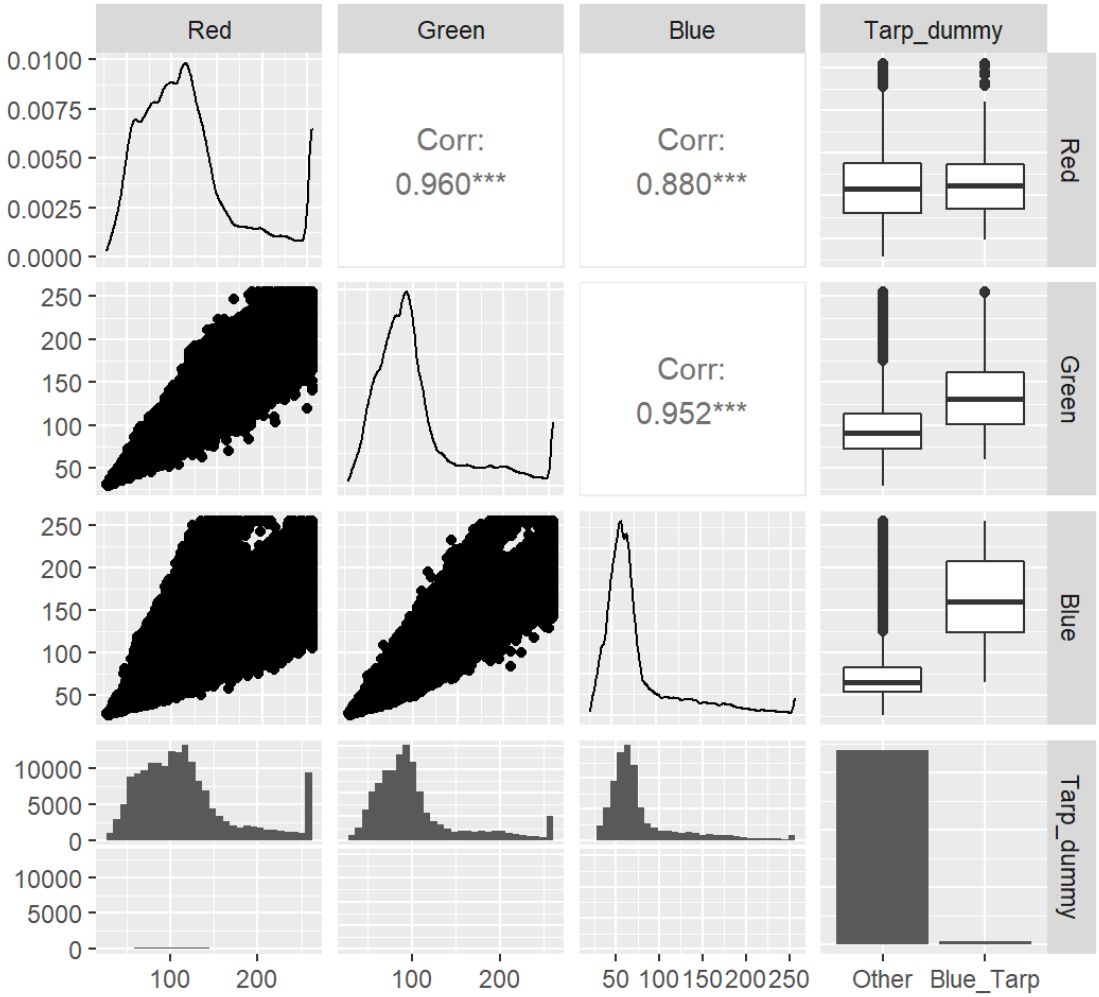
Comparing the two, the boxplot distributions seem to match B1 = Red, B2 = Green & B3 = Blue and relative correlatons, RGB makes the most sense in this context. The boxplots show that Blue has the largest difference between class means, then green, then red for both data sets. Let's assume that those are the column orders.

```
names(hold_out_data) <- c("Class","Red","Green","Blue","Tarp_dummy")
head(hold_out_data)
```

```
#>   Class Red Green Blue Tarp_dummy
#> 1 Other 104    89   63      Other
#> 2 Other 103    84   63      Other
#> 3 Other  98    78   57      Other
#> 4 Other  96    69   50      Other
#> 5 Other 115    94   75      Other
#> 6 Other  97    78   63      Other
```

We can also do similar data exploration as we did for the training dataset.

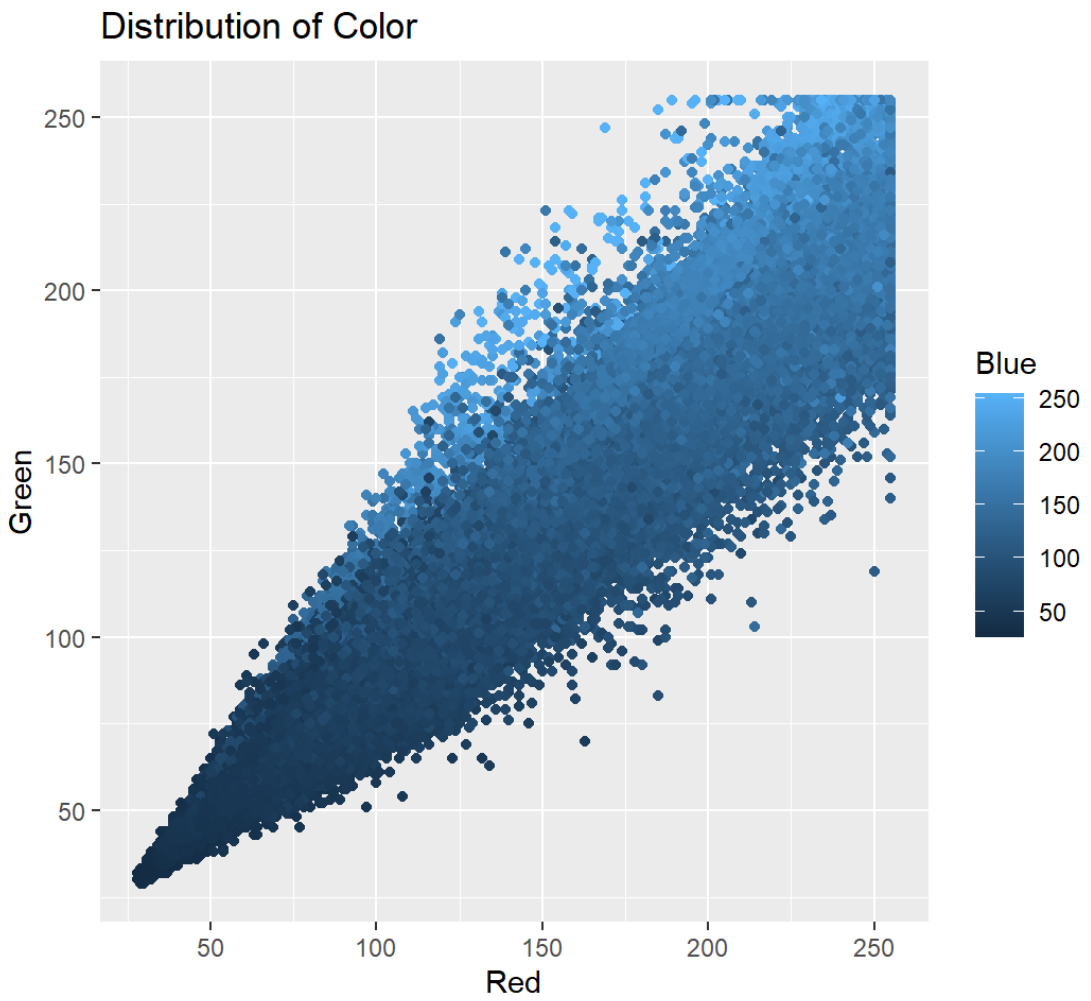
```
#-- Explore data
ggpairs(hold_out_data[,2:5])
```



From the pairplot, we can see that there a very few blue tarp images in comparison to the Other types of images. As expected, the blue tarps have a certain distribution of pixel values that indicate a blue tarp (average luminosity of red pixels, but high blue and green luminosity values). This is very similar to the data we saw in the training data and will hopefully bode well for prediction on based on the models created from the training data. There does seem to be a more distinct separation on the Blue Pixels for Blue Tarps vs. non-blue tarps. This is interesting and bodes the question could the models be trained solely on the blue pixels alone.

The correlation between blue and red is actually much less for the hold out data compared to the training data. Let's explore further.

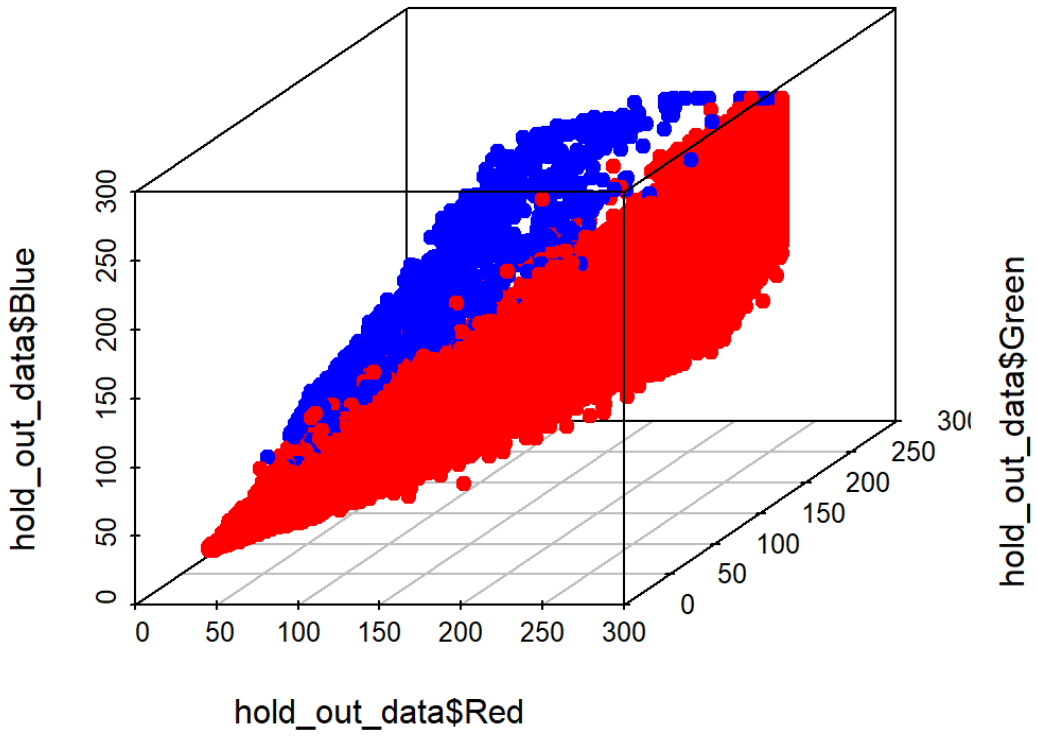
```
#-- Hue plot
pixelplot<-ggplot() + geom_point(data = hold_out_data, aes(x = Red, y = Green, color = Blue))
print(pixelplot + ggtitle("Distribution of Color"))
```



As seen in the pairplot and confirmed above with a plot of the distribution of color values, the separate color pixel luminosity values are highly correlated for the hold out data just like the trainig data. As red and green luminosity increases, so does blue. It is interesting, though, that it seems that high green and high blue values are more correlated than high red and high blue luminosity (and confirmed in the pairplot) and even more so than the training data.

We can also plot a 3D plot of the data to see the boundary between Blue Tarps and Other Images just like we did before.

```
scatterplot3d(x = hold_out_data$Red, y = hold_out_data$Green, z = hold_out_data$Blue,
              color = c("red","blue")[as.factor(hold_out_data$Tarp_dummy)], pch = 19)
```

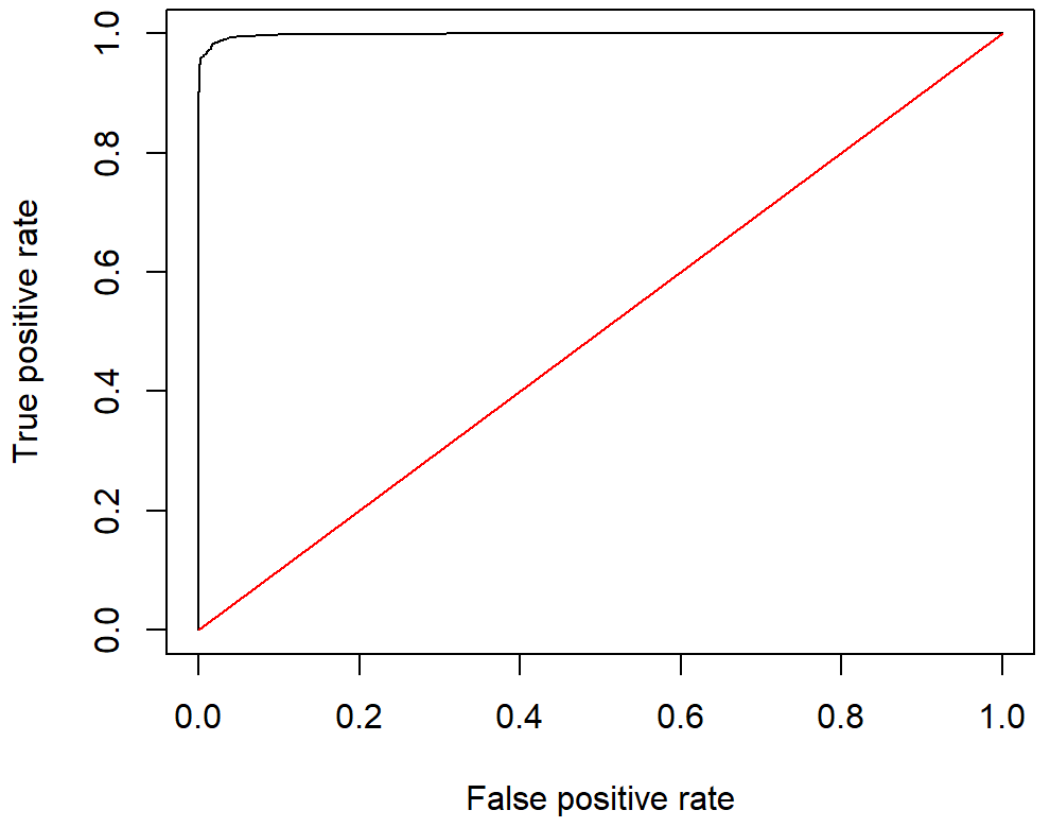


This plot is interesting because there seems to be less separation in the hold out data than the training data. There are some high levels of blue pixel luminosity that are not blue tarps

6 Results (Hold-Out)

Now we can predict on the Hold Out set to see how the trained models on the training set of data performs. The same probability threshold for each model is used as chosen in the training prediction. We also take a look at each of the ROC Curves and the Accuracy of the prediction for each model on the hold out data.

```
#-- Test and see how it performs on the data
glm.probs.hold_out<- predict(logistic_glm_model_int_red, hold_out_data, type="prob")
rates.glm.hold_out <- prediction(1-glm.probs.hold_out[,2], hold_out_data$Tarp_dummy)
perf.glm.hold_out <- performance(rates.glm.hold_out,"tpr","fpr")

plot(perf.glm.hold_out, main="ROC Curve for Logistic on Hold Out")
lines(x = c(0,1), y = c(0,1), col="red")
```




```
auc.glm.hold_out <- performance(rates.glm.hold_out, measure = "auc")
auc.glm.hold_out<-auc.glm.hold_out@y.values[[1]]


glm.pred.hold_out<-rep("Other",nrow(hold_out_data))
glm.pred.hold_out[glm.probs.hold_out[,2]>threshold.glm]="Tarp"
glm.hold_out.table<-table(glm.pred.hold_out,hold_out_data$Tarp_dummy)
glm.hold_out.table
```

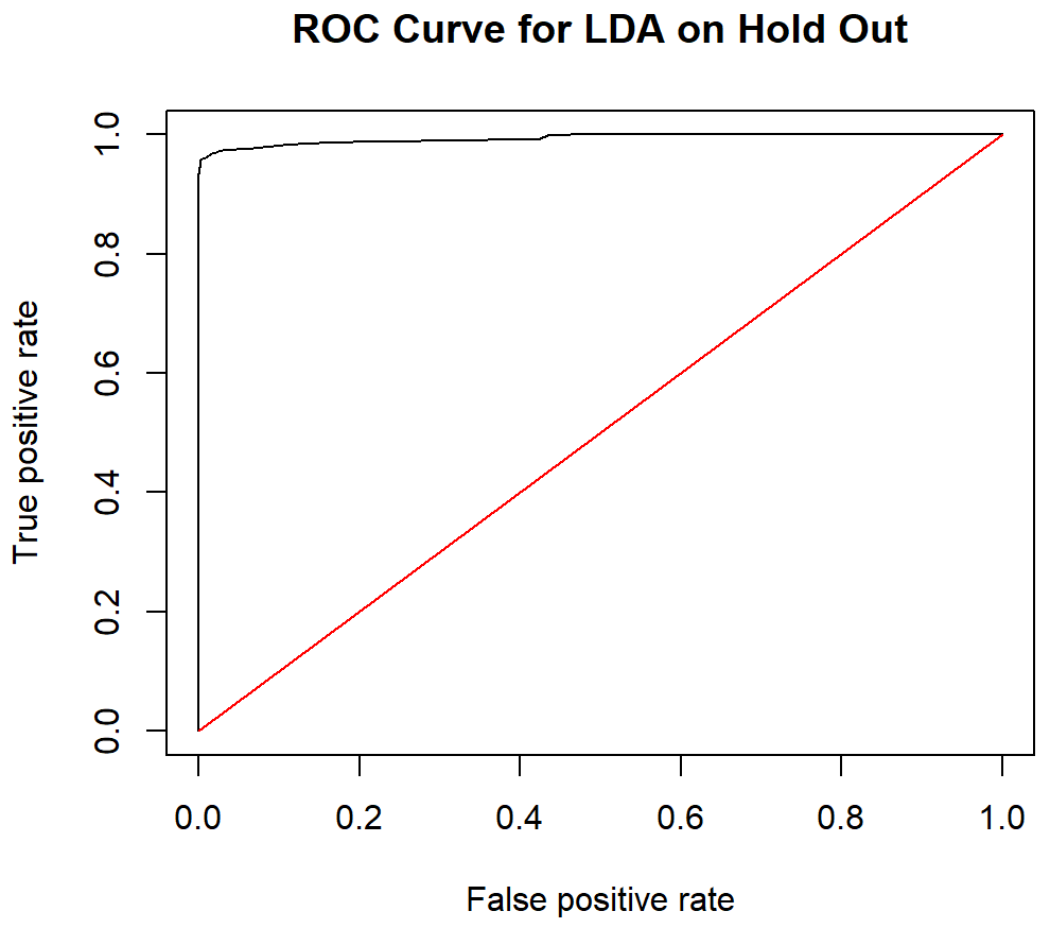
```
#>
#> glm.pred.hold_out  Other Blue_Tarp
#>             Other 169112       211
#>             Tarp     305      1917
```

```
mean(glm.pred.hold_out==hold_out_data$Tarp_dummy)
```

```
#> [1] 0.9858171
```

```
bottom_right.glm.hold_out<-table(glm.pred.hold_out,hold_out_data$Tarp_dummy)[2,2]
bottom_left.glm.hold_out<-table(glm.pred.hold_out,hold_out_data$Tarp_dummy)[2,1]
top_left.glm.hold_out<-table(glm.pred.hold_out,hold_out_data$Tarp_dummy)[1,1]
top_right.glm.hold_out<-table(glm.pred.hold_out,hold_out_data$Tarp_dummy)[1,2]

tpr.glm.hold_out = bottom_right.glm.hold_out/(bottom_right.glm.hold_out+top_right.glm.hold_out)
fpr.glm.hold_out = bottom_left.glm.hold_out/(bottom_left.glm.hold_out+top_left.glm.hold_out)
precision.glm.hold_out = bottom_right.glm.hold_out/(bottom_right.glm.hold_out+bottom_left.glm.hold_out)
accuracy.glm.hold_out = (top_left.glm.hold_out+bottom_right.glm.hold_out)/sum(glm.hold_out.table)
```

```
#-- Test and see how it performs on the data
lda.probs.hold_out<- predict(lda_model_no_int, hold_out_data, type="prob")
rates.lda.hold_out <- prediction(1-lda.probs.hold_out[,2], hold_out_data$Tarp_dummy)
perf.lda.hold_out <- performance(rates.lda.hold_out,"tpr","fpr")

plot(perf.lda.hold_out, main="ROC Curve for LDA on Hold Out")
lines(x = c(0,1), y = c(0,1), col="red")
```



```
auc.lda.hold_out <- performance(rates.lda.hold_out, measure = "auc")
auc.lda.hold_out<-auc.lda.hold_out@y.values[[1]]


lda.pred.hold_out<-rep("Other",nrow(hold_out_data))
lda.pred.hold_out[lda.probs.hold_out[,2]>threshold.lda]="Tarp"
lda.hold_out.table<-table(lda.pred.hold_out,hold_out_data$Tarp_dummy)
lda.hold_out.table
```

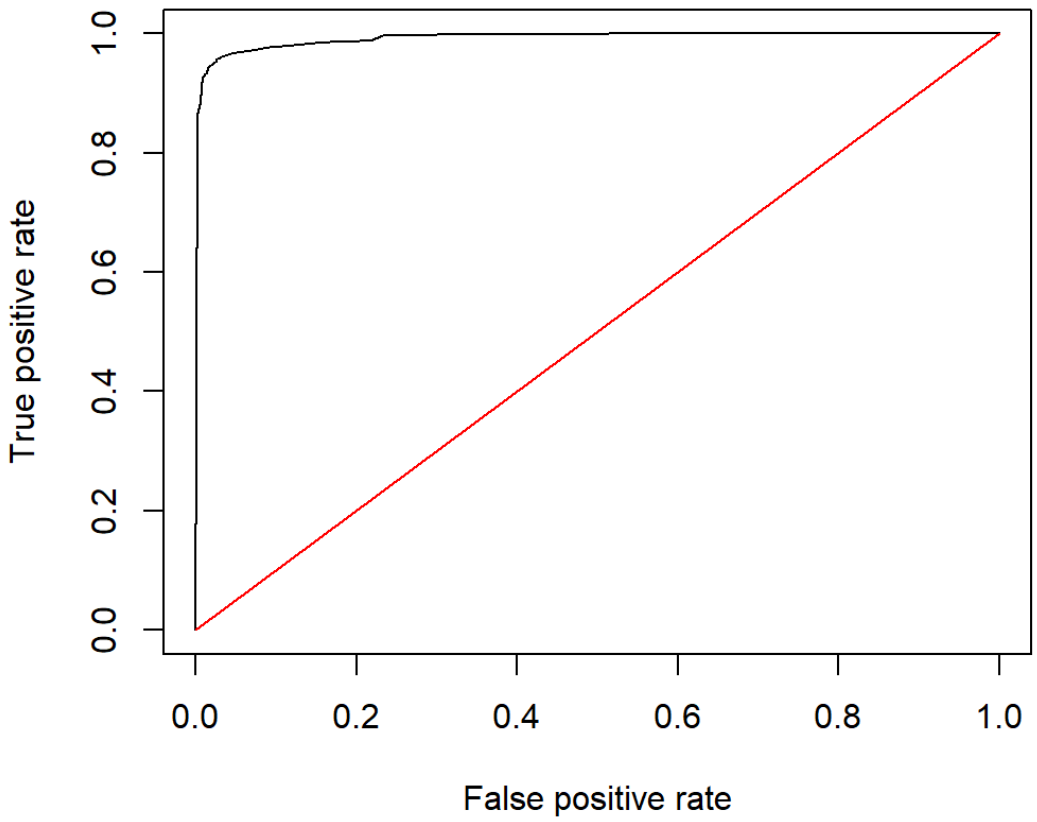
```
#>
#> lda.pred.hold_out  Other Blue_Tarp
#>              Other 166796       279
#>              Tarp    2621      1849
```

```
mean(lda.pred.hold_out==hold_out_data$Tarp_dummy)
```

```
#> [1] 0.9723163
```

```
bottom_right.lda.hold_out<-table(lda.pred.hold_out,hold_out_data$Tarp_dummy)[2,2]
bottom_left.lda.hold_out<-table(lda.pred.hold_out,hold_out_data$Tarp_dummy)[2,1]
top_left.lda.hold_out<-table(lda.pred.hold_out,hold_out_data$Tarp_dummy)[1,1]
top_right.lda.hold_out<-table(lda.pred.hold_out,hold_out_data$Tarp_dummy)[1,2]

tpr.lda.hold_out = bottom_right.lda.hold_out/(bottom_right.lda.hold_out+top_right.lda.hold_out)
fpr.lda.hold_out = bottom_left.lda.hold_out/(bottom_left.lda.hold_out+top_left.lda.hold_out)
precision.lda.hold_out = bottom_right.lda.hold_out/(bottom_right.lda.hold_out+bottom_left.lda.hold_out)
accuracy.lda.hold_out = (top_left.lda.hold_out+bottom_right.lda.hold_out)/sum(lda.hold_out.table)
```

```
#-- Test and see how it performs on the data
qda.probs.hold_out<- predict(qda_model_no_int, hold_out_data, type="prob")
rates.qda.hold_out <- prediction(1-qda.probs.hold_out[,2], hold_out_data$Tarp_dummy)
perf.qda.hold_out <- performance(rates.qda.hold_out,"tpr","fpr")

plot(perf.qda.hold_out, main="ROC Curve for QDA on Hold Out")
lines(x = c(0,1), y = c(0,1), col="red")
```




```
auc.qda.hold_out <- performance(rates.qda.hold_out, measure = "auc")
auc.qda.hold_out<-auc.qda.hold_out@y.values[[1]]


qda.pred.hold_out<-rep("Other",nrow(hold_out_data))
qda.pred.hold_out[qda.probs.hold_out[,2]>threshold.qda]="Tarp"
qda.hold_out.table<-table(qda.pred.hold_out,hold_out_data$Tarp_dummy)
qda.hold_out.table
```

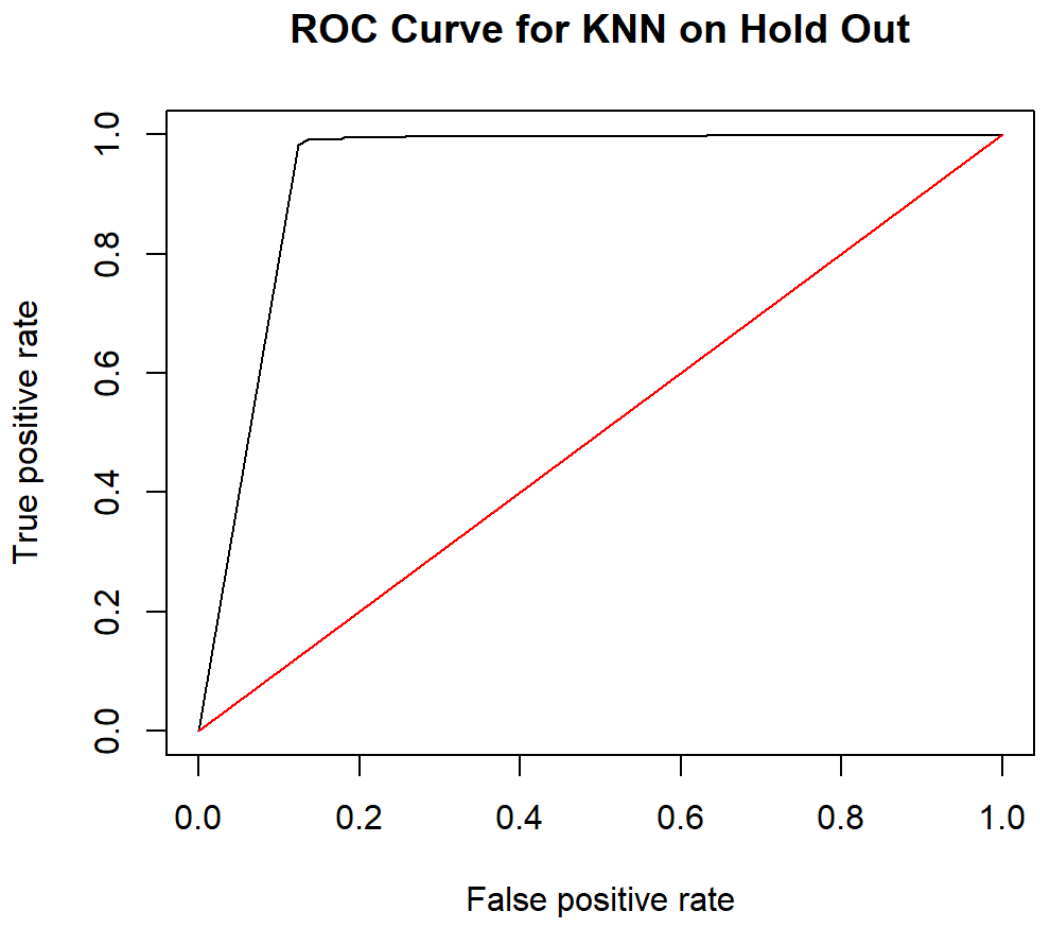
```
#>
#> qda.pred.hold_out  Other Blue_Tarp
#>              Other 169206       741
#>              Tarp     211      1387
```

```
mean(qda.pred.hold_out==hold_out_data$Tarp_dummy)
```

```
#> [1] 0.9863651
```

```
bottom_right.qda.hold_out<-table(qda.pred.hold_out,hold_out_data$Tarp_dummy)[2,2]
bottom_left.qda.hold_out<-table(qda.pred.hold_out,hold_out_data$Tarp_dummy)[2,1]
top_left.qda.hold_out<-table(qda.pred.hold_out,hold_out_data$Tarp_dummy)[1,1]
top_right.qda.hold_out<-table(qda.pred.hold_out,hold_out_data$Tarp_dummy)[1,2]

tpr.qda.hold_out = bottom_right.qda.hold_out/(bottom_right.qda.hold_out+top_right.qda.hold_out)
fpr.qda.hold_out = bottom_left.qda.hold_out/(bottom_left.qda.hold_out+top_left.qda.hold_out)
precision.qda.hold_out = bottom_right.qda.hold_out/(bottom_right.qda.hold_out+bottom_left.qda.hold_out)
accuracy.qda.hold_out = (top_left.qda.hold_out+bottom_right.qda.hold_out)/sum(qda.hold_out.table)
```

```
#-- Test and see how it performs on the data
knn.probs.hold_out<- predict(knn_final, hold_out_data, type="prob")
rates.knn.hold_out <- prediction(1-knn.probs.hold_out[,2], hold_out_data$Tarp_dummy)
perf.knn.hold_out <- performance(rates.knn.hold_out,"tpr","fpr")

plot(perf.knn.hold_out, main="ROC Curve for KNN on Hold Out")
lines(x = c(0,1), y = c(0,1), col="red")
```



```
auc.knn.hold_out <- performance(rates.knn.hold_out, measure = "auc")
auc.knn.hold_out<-auc.knn.hold_out@y.values[[1]]


knn.pred.hold_out<-rep("Other",nrow(hold_out_data))
knn.pred.hold_out[knn.probs.hold_out[,2]>threshold.knn]="Tarp"
knn.hold_out.table<-table(knn.pred.hold_out,hold_out_data$Tarp_dummy)
knn.hold_out.table
```

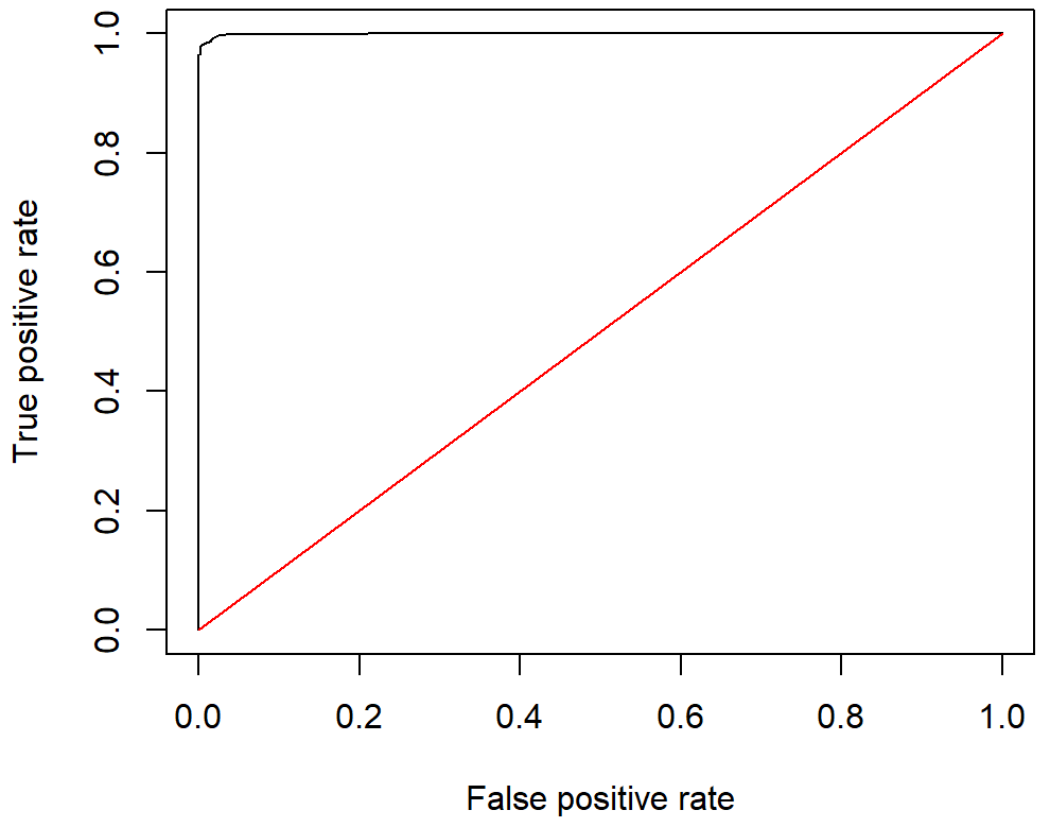
```
#>
#> knn.pred.hold_out  Other Blue_Tarp
#>              Other 168490       383
#>              Tarp     927      1745
```

```
mean(knn.pred.hold_out==hold_out_data$Tarp_dummy)
```

```
#> [1] 0.9821913
```

```
bottom_right.knn.hold_out<-table(knn.pred.hold_out,hold_out_data$Tarp_dummy)[2,2]
bottom_left.knn.hold_out<-table(knn.pred.hold_out,hold_out_data$Tarp_dummy)[2,1]
top_left.knn.hold_out<-table(knn.pred.hold_out,hold_out_data$Tarp_dummy)[1,1]
top_right.knn.hold_out<-table(knn.pred.hold_out,hold_out_data$Tarp_dummy)[1,2]

tpr.knn.hold_out = bottom_right.knn.hold_out/(bottom_right.knn.hold_out+top_right.knn.hold_out)
fpr.knn.hold_out = bottom_left.knn.hold_out/(bottom_left.knn.hold_out+top_left.knn.hold_out)
precision.knn.hold_out = bottom_right.knn.hold_out/(bottom_right.knn.hold_out+bottom_left.knn.hold_out)
accuracy.knn.hold_out = (top_left.knn.hold_out+bottom_right.knn.hold_out)/sum(knn.hold_out.table)
```

```
#-- Test and see how it performs on the data
ridge.probs.hold_out<- predict(ridge.fit, hold_out_data, type="prob")
rates.ridge.hold_out <- prediction(1-ridge.probs.hold_out[,2], hold_out_data$Tarp_dummy)
perf.ridge.hold_out <- performance(rates.ridge.hold_out,"tpr","fpr")

plot(perf.ridge.hold_out, main="ROC Curve for Penalized Logistic Regres on Hold Out")
lines(x = c(0,1), y = c(0,1), col="red")
```




```
auc.ridge.hold_out <- performance(rates.ridge.hold_out, measure = "auc")
auc.ridge.hold_out<-auc.ridge.hold_out@y.values[[1]]


ridge.pred.hold_out<-rep("Other",nrow(hold_out_data))
ridge.pred.hold_out[ridge.probs.hold_out[,2]>threshold.ridge]="Tarp"
ridge.hold_out.table<-table(ridge.pred.hold_out,hold_out_data$Tarp_dummy)
ridge.hold_out.table
```

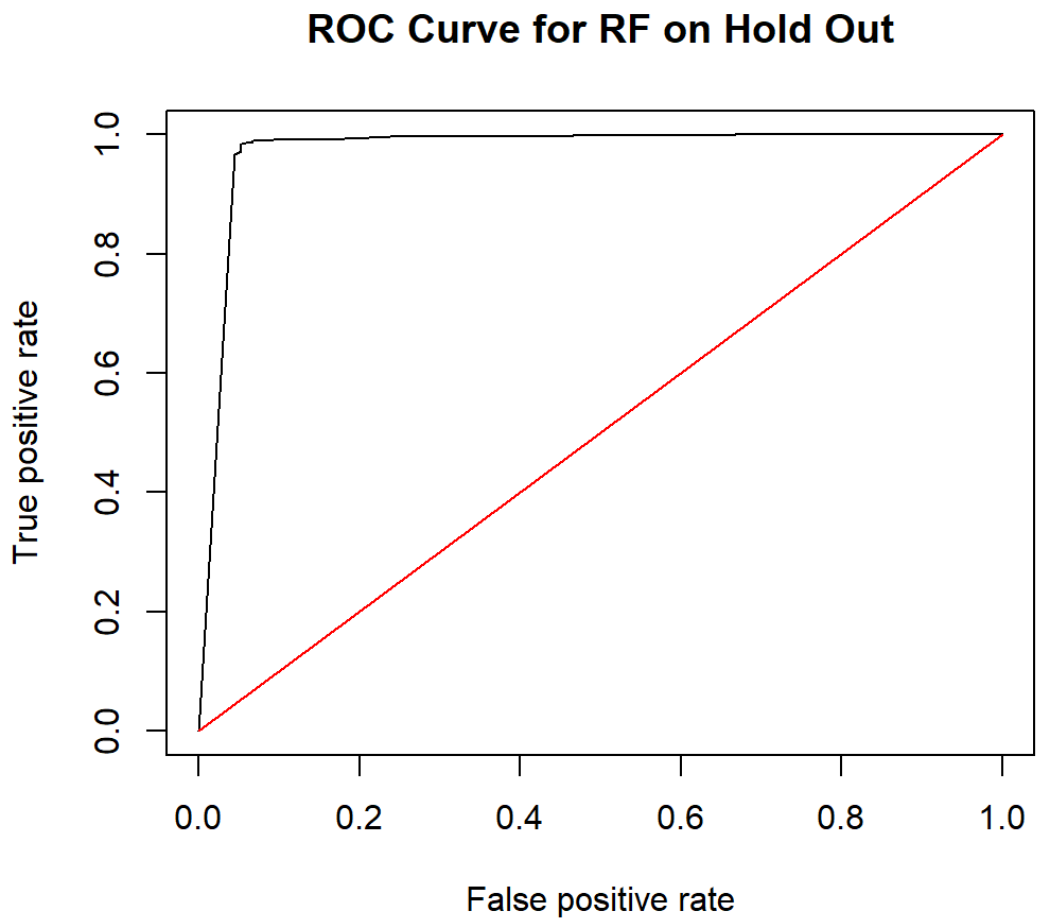
```
#>
#> ridge.pred.hold_out  Other Blue_Tarp
#>               Other 168948        54
#>               Tarp     469      2074
```

```
mean(ridge.pred.hold_out==hold_out_data$Tarp_dummy)
```

```
#> [1] 0.9848611
```

```
bottom_right.ridge.hold_out<-table(ridge.pred.hold_out,hold_out_data$Tarp_dummy)[2,2]
bottom_left.ridge.hold_out<-table(ridge.pred.hold_out,hold_out_data$Tarp_dummy)[2,1]
top_left.ridge.hold_out<-table(ridge.pred.hold_out,hold_out_data$Tarp_dummy)[1,1]
top_right.ridge.hold_out<-table(ridge.pred.hold_out,hold_out_data$Tarp_dummy)[1,2]

tpr.ridge.hold_out = bottom_right.ridge.hold_out/(bottom_right.ridge.hold_out+top_right.ridge.hold_out)
fpr.ridge.hold_out = bottom_left.ridge.hold_out/(bottom_left.ridge.hold_out+top_left.ridge.hold_out)
precision.ridge.hold_out = bottom_right.ridge.hold_out/(bottom_right.ridge.hold_out
                                                        +bottom_left.ridge.hold_out)
accuracy.ridge.hold_out = (top_left.ridge.hold_out+bottom_right.ridge.hold_out)/sum(ridge.hold_out.table)
```

```
#-- Test and see how it performs on the data
forest.probs.hold_out<- predict(forest.fit, hold_out_data, type="prob")
rates.forest.hold_out <- prediction(1-forest.probs.hold_out[,2], hold_out_data$Tarp_dummy)
perf.forest.hold_out <- performance(rates.forest.hold_out,"tpr","fpr")

plot(perf.forest.hold_out, main="ROC Curve for RF on Hold Out")
lines(x = c(0,1), y = c(0,1), col="red")
```



```
auc.forest.hold_out <- performance(rates.forest.hold_out, measure = "auc")
auc.forest.hold_out<-auc.forest.hold_out@y.values[[1]]


forest.pred.hold_out<-rep("Other",nrow(hold_out_data))
forest.pred.hold_out[forest.probs.hold_out[,2]>threshold.forest]="Tarp"
forest.hold_out.table<-table(forest.pred.hold_out,hold_out_data$Tarp_dummy)
forest.hold_out.table
```

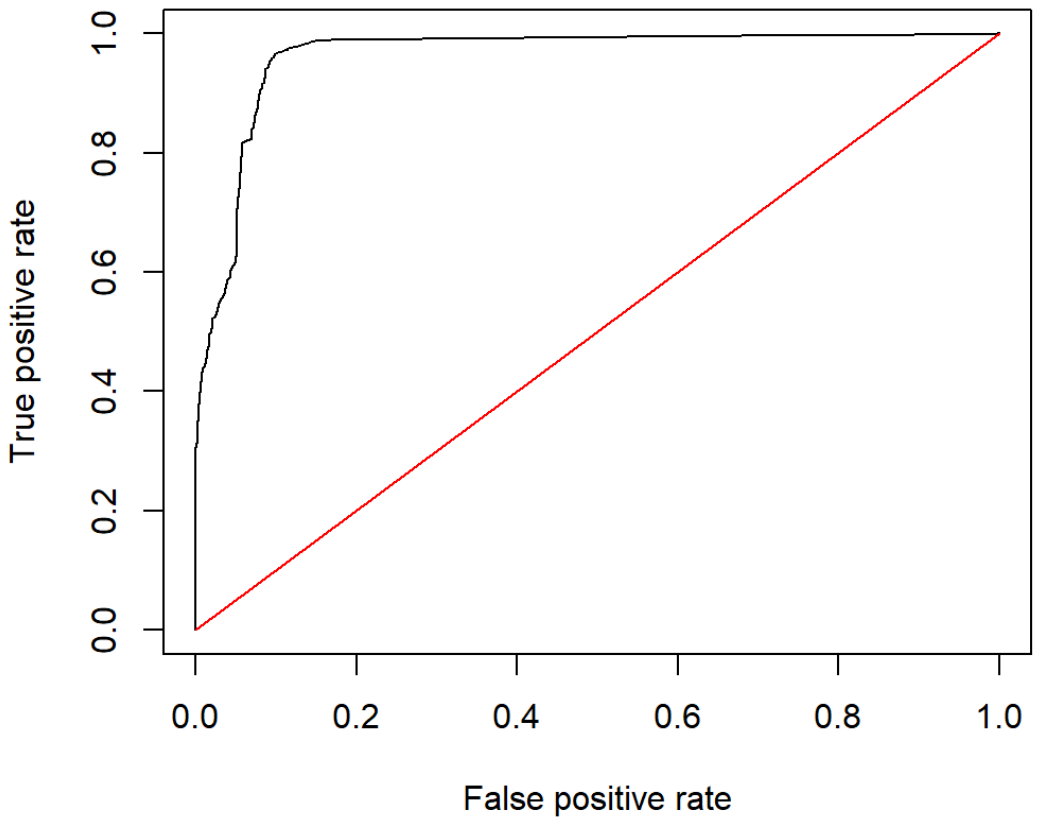
```
#>
#> forest.pred.hold_out  Other Blue_Tarp
#>                Other 168693       481
#>                Tarp     724      1647
```

```
mean(forest.pred.hold_out==hold_out_data$Tarp_dummy)
```

```
#> [1] 0.9833746
```

```
bottom_right.forest.hold_out<-table(forest.pred.hold_out,hold_out_data$Tarp_dummy)[2,2]
bottom_left.forest.hold_out<-table(forest.pred.hold_out,hold_out_data$Tarp_dummy)[2,1]
top_left.forest.hold_out<-table(forest.pred.hold_out,hold_out_data$Tarp_dummy)[1,1]
top_right.forest.hold_out<-table(forest.pred.hold_out,hold_out_data$Tarp_dummy)[1,2]

tpr.forest.hold_out = bottom_right.forest.hold_out/(bottom_right.forest.hold_out+top_right.forest.hold_out)
fpr.forest.hold_out = bottom_left.forest.hold_out/(bottom_left.forest.hold_out+top_left.forest.hold_out)
precision.forest.hold_out = bottom_right.forest.hold_out/(bottom_right.forest.hold_out
                                                          +bottom_left.forest.hold_out)
accuracy.forest.hold_out = (top_left.forest.hold_out+bottom_right.forest.hold_out)/sum(forest.hold_out.table)
```

```
#-- Test and see how it performs on the data
svm.probs.hold_out<- predict(svm.fit.radial, hold_out_data, type="prob")
rates.svm.hold_out <- prediction(1-svm.probs.hold_out[,2], hold_out_data$Tarp_dummy)
perf.svm.hold_out <- performance(rates.svm.hold_out,"tpr","fpr")

plot(perf.svm.hold_out, main="ROC Curve for SVM on Hold Out")
lines(x = c(0,1), y = c(0,1), col="red")
```




```
auc.svm.hold_out <- performance(rates.svm.hold_out, measure = "auc")
auc.svm.hold_out<-auc.svm.hold_out@y.values[[1]]


svm.pred.hold_out<-rep("Other",nrow(hold_out_data))
svm.pred.hold_out[svm.probs.hold_out[,2]>threshold.svm]="Tarp"
svm.hold_out.table <- table(svm.pred.hold_out,hold_out_data$Tarp_dummy)
svm.hold_out.table
```

```
#>
#> svm.pred.hold_out  Other Blue_Tarp
#>             Other 168593      1258
#>             Tarp     824       870
```

```
mean(svm.pred.hold_out==hold_out_data$Tarp_dummy)
```

```
#> [1] 0.9827917
```

```
bottom_right.svm.hold_out<-table(svm.pred.hold_out,hold_out_data$Tarp_dummy)[2,2]
bottom_left.svm.hold_out<-table(svm.pred.hold_out,hold_out_data$Tarp_dummy)[2,1]
top_left.svm.hold_out<-table(svm.pred.hold_out,hold_out_data$Tarp_dummy)[1,1]
top_right.svm.hold_out<-table(svm.pred.hold_out,hold_out_data$Tarp_dummy)[1,2]

tpr.svm.hold_out = bottom_right.svm.hold_out/(bottom_right.svm.hold_out+top_right.svm.hold_out)
fpr.svm.hold_out = bottom_left.svm.hold_out/(bottom_left.svm.hold_out+top_left.svm.hold_out)
precision.svm.hold_out = bottom_right.svm.hold_out/(bottom_right.svm.hold_out+bottom_left.svm.hold_out)
accuracy.svm.hold_out = (top_left.svm.hold_out+bottom_right.svm.hold_out)/sum(svm.hold_out.table)
```

Now that we have all the predictions on the hold out data by each of the Models, we can summarize in a table just like we did for the training data to compare performance on this hold-out test set.

```
results.hold_out<-data.frame("Model" = c("Log Reg","LDA","QDA","KNN","Penalized Log Reg",
                                         "Random Forest (ntrees=500)","SVM (kernel=Radial)"),
                             "Tuning"= c("n/a","n/a","n/a",best_k,best_lambda,best_mtry,best_cost),
                             "AUROC"=c(0,0,0,0,0,0,0),"Threshold"=c(0,0,0,0,0,0,0),
                             "Accuracy"=c(0,0,0,0,0,0,0),"TPR"=c(0,0,0,0,0,0,0),
                             "FPR"=c(0,0,0,0,0,0,0),"Precision"=c(0,0,0,0,0,0,0))

results.hold_out[,4]=c(threshold.glm, threshold.lda,threshold.qda,
                       threshold.knn,threshold.ridge,threshold.forest,threshold.svm)
results.hold_out[,3]=c(auc.glm.hold_out,auc.lda.hold_out,auc.qda.hold_out,
                       auc.knn.hold_out,auc.ridge.hold_out,auc.forest.hold_out,auc.svm.hold_out)
results.hold_out[,5]=c(accuracy.glm.hold_out,accuracy.lda.hold_out,accuracy.qda.hold_out,
                       accuracy.knn.hold_out,accuracy.ridge.hold_out,accuracy.forest.hold_out
                       ,accuracy.svm.hold_out)
results.hold_out[,6]=c(tpr.glm.hold_out,tpr.lda.hold_out,tpr.qda.hold_out,
                       tpr.knn.hold_out,tpr.ridge.hold_out,tpr.forest.hold_out,tpr.svm.hold_out)
results.hold_out[,7]=c(fpr.glm.hold_out,fpr.lda.hold_out,fpr.qda.hold_out,
                       fpr.knn.hold_out,fpr.ridge.hold_out,fpr.forest.hold_out,fpr.svm.hold_out)
results.hold_out[,8]=c(precision.glm.hold_out,precision.lda.hold_out,precision.qda.hold_out,
                       precision.knn.hold_out,precision.ridge.hold_out,precision.forest.hold_out
                       ,precision.svm.hold_out)


results.hold_out  %>%
  kbl(caption = "Hold-Out Results") %>%
  kable_styling(bootstrap_options = c("striped", "hover", "condensed", "responsive"))
```

Hold-Out Results

Model	Tuning	AUROC	Threshold	Accuracy	TPR	FPR	Precision
Log Reg	n/a	0.9987635	0.80	0.9969920	0.9008459	0.0018003	0.8627363
LDA	n/a	0.9935578	0.16	0.9830948	0.8688910	0.0154707	0.4136465
QDA	n/a	0.9922695	0.76	0.9944504	0.6517857	0.0012454	0.8679599
KNN	3	0.9351855	0.50	0.9923635	0.8200188	0.0054717	0.6530689
Penalized Log Reg	8.32215451789909e-05	0.9995238	0.72	0.9969512	0.9746241	0.0027683	0.8155722

Model	Tuning	AUROC	Threshold	Accuracy	TPR	FPR	Precision
Random Forest (ntrees=500)	2	0.9743995	0.54	0.9929756	0.7739662	0.0042735	0.6946436
SVM (kernel=Radial)	2	0.9630707	0.68	0.9878632	0.4088346	0.0048637	0.5135773

7 Final Conclusions

7.1 Conclusion #1 A discussion of the best performing algorithm(s) in the cross-validation and hold-out data

Compared to previous attempts on the training data alone, the hold out data creates a twist in determining the best performing algorithm on the cross-validation data and the hold data. Taking a look at the performance table on cross-validation, shown below, the K-Nearest Neighbor & Random Forest Models seem to perform the best in predicting which images from the Haitian Earthquake crisis contain Blue Tarps (to assist in identifying temporary shelters).

```
results %>%
  kbl(caption = "Cross-Validation Results") %>%
  kable_styling(bootstrap_options = c("striped", "hover", "condensed", "responsive"))
```

Cross-Validation Results

Model	Tuning	AUROC	Threshold	Accuracy	TPR	FPR	Precision
Log Reg	n/a	0.9995442	0.80	0.9949716	0.8644906	0.0007187	0.9754464
LDA	n/a	0.9888768	0.16	0.9827802	0.8397626	0.0124961	0.6894032
QDA	n/a	0.9982175	0.76	0.9936592	0.8046489	0.0000980	0.9963258
KNN	3	0.9998980	0.50	0.9981025	0.9683482	0.0009147	0.9721946
Penalized Log Reg	8.32215451789909e-05	0.9985033	0.72	0.9944024	0.8372898	0.0004084	0.9854482
Random Forest (ntrees=500)	2	0.9944970	0.54	0.9995889	0.9876360	0.0000163	0.9994995
SVM (kernel=Radial)	2	0.9960265	0.68	0.9971379	0.9371909	0.0008821	0.9722935

In terms of prediction evaluation, both the KNN Model and Random forest performed strongly via cross-validation on the training data set.

```
1.KNN with K = 3, has the highest AUROC att 99.98980% meaning the model will correctly assign a Blue Tarp to
a randomly selected image with a Blue Tarp than to a randomly selected image without a Blue Tarp (on the cros
s-validation data). For the other metrics,including Accuracy, TPR, FPR, Precision, KNN performed in the top 2
-3 compared to the rest of the Models

2. Random Forest with mtry = 2 performed strongly for all metrics: it was the strongest performer over cross
valdiation for all metrics besides AUROC, in which it performed 2nd best.
```

Based on the comparison above, and in terms of cross-validation, it can be determined that the Random Forest Model performed best in terms of predicting which images from the Haitian Earthquake crisis contain Blue Tarps (to assist in identifying temporary shelters) on the cross validation data.

Looking at the hold-out data prediction performance (shown below), the results are a little different than via cross-validation. Note that all of these use the same thresholds as used as in the cross-trained models.

```
results.hold_out  %>%
  kbl(caption = "Hold-Out Results") %>%
  kable_styling(bootstrap_options = c("striped", "hover", "condensed", "responsive"))
```

Hold-Out Results

Model	Tuning	AUROC	Threshold	Accuracy	TPR	FPR	Precision
Log Reg	n/a	0.9987635	0.80	0.9969920	0.9008459	0.0018003	0.8627363
LDA	n/a	0.9935578	0.16	0.9830948	0.8688910	0.0154707	0.4136465
QDA	n/a	0.9922695	0.76	0.9944504	0.6517857	0.0012454	0.8679599
KNN	3	0.9351855	0.50	0.9923635	0.8200188	0.0054717	0.6530689
Penalized Log Reg	8.32215451789909e-05	0.9995238	0.72	0.9969512	0.9746241	0.0027683	0.8155722
Random Forest (ntrees=500)	2	0.9743995	0.54	0.9929756	0.7739662	0.0042735	0.6946436
SVM (kernel=Radial)	2	0.9630707	0.68	0.9878632	0.4088346	0.0048637	0.5135773

Interestingly, the once strong performing KNN Model on cross validation data, are now one of the weaker performing models, which may be a sign of over-training on the training data . Random Forest is still decently strong with AUROC over 97% but is middle tier in terms of the rest of the Models. As shown in the table above, Penalized Logistic regression clearly performs the best with the highest AUROC & Accuracy. This was a decently strong performer over cross validation, but rose to the top on the Hold Out Data. This may indicate a good balance of training and prediction performance on new data. We will discuss this later in the conclusions

7.2 Conclusion #2 A discussion or analysis justifying why your findings above are compatible or reconcilable.

The results of the model evaluation on predicting which images from the Haitian Earthquake crisis contain Blue Tarps (to assist in identifying temporary shelters), are reconcilable because any differences between hold-data performance and cross-validation data performance can be explained by how models are trained. Each Model is different in many ways, including its flexibility. Flexibility can be defined as how much is model's behavior influenced by characteristics of the data; therefore, the flexibility or inflexibility of a model is a characteristic that should not be overlooked. It can mean the difference between a useful or useless Model. Flexible Models fit more easily to the data and put less assumptions on distribution of the data before fitting the Model: in our analysis the most flexible Models are the K-Nearest Neighbor Model, Random Forest, & Support Vector Machines. It is shown that the most flexible Model (KNN), which performs very strongly on cross-validation data (AUROC of 99.98980%), performs considerably worse on the hold-out data (AUROC of 93.51855% which is the worst). This is due to over-training. The KNN model being the most flexible, especially with a low k value of 3 (number of nearest neighbors to include in the majority of the voting process),

fits very closely to the data in the training set. This fit is actually too close and does not allow the model to predict as well on a new data set not in the training set. A similar result is seen in the other flexible models, with Support Vector Machines and Random forest having slightly reduced performance on prediction in the hold out data. This is not as prevalent as the KNN model.

Conversely, the other less flexible models performed similarly in the Hold-Out data as they performed on the cross-validation data. This is expected as they take in more assumptions on how the data actually looks and is distributed and therefore if they perform poorly on the training data (or relatively in terms of this data set) they will likely perform poorly on the hold-out data and visa-versa for the strong performing models. The fact that penalized logistic regression and logistic regression both performed very well on both data sets in terms of prediction might speak about how the data fits well to a logistic regression in general. More investigation could likely be done to investigate the true distribution of the data and this could be featured in future work.

7.3 Conclusion #3 A recommendation and rationale regarding which algorithm to use for detection of blue tarps.

As discussed previously, there are significant differences between which models perform the best in terms predicting which images from the Haitian Earthquake crisis contain Blue Tarps (to assist in identifying temporary shelters), on the results of cross-validation on the training data and hold-data. This can be attributed to over-training as discussed above due to flexibility of Models.

Overall, the Model that performed the best in predicting which images contained Blue Tarps was the Penalized Logistic Regression. This Model performs well on both the training via cross-validation and hold-out data as well. On cross-validation, this Model had the 2nd strongest AUROC and strong performance in all of the other metrics. Additionally, as discussed before, on the Hold Out data the Penalized Linear Regression has the strong AUROC and strongest True Positive Rate, as well as 3rd lowest False Positive Rate and Precision score. QDA, Logistic Regression & Random Forests also get honorable mentions as they performed well in both the training and hold-out.

One additional thing to take into account is the training times of the Models on Cross Validation. Random Forest performed strongly but took decently long to train as shown by the Algorithm Runtimes and therefore was not chosen as the top Model. Penalized linear regression, conversely, ran relatively quickly which is vital if the Model ever needs to be refit. This is because response time to save the displaced citizens is key to saving lives. This time complexity was the final decision to choose Penalized Logistic REgression as the strongest performing algorithm for detection of blue tarps.

7.4 Conclusion #4 A discussion of the relevance of the metrics calculated in the tables to this application context.

The metrics used in this analysis for evaluating algorithm performance on predicting which images from the Haitian Earthquake crisis contain Blue Tarps (to assist in identifying temporary shelters) were the following (as seen in the performance tables): AUROC, Accuracy, TPR, FPR and Precision.

```
1. AUROC represents the Area under the ROC curve andit tells  about the model's ability to discriminate betwe
en blue tarps  (positive examples) and non-blue-tarps (negative examples.) For example, an AUROC of 0.8 means
that the model has good   discriminatory ability: 80% of the time, the model will correctly assign a blue tar
p to a randomly image containing a blue tarp than to a randomly selected image without Blue Tarp.

2. Accuracy describes overall how often the algorithm correctly identifies whether a blue tarp is present or
not

3. True Positive Rate describes when there is actually a blue tarp, how often does the Model predict that the
re is.

4. False Positive Rate describes when there actually is not a blue tarp in the image, how often does the algo
rithm predict that there is

5. Precision represents when when an image is predicted to have a blue tarp, how often is the algorithm actua
lly correct.
```

As one can see, all these metrics measure different things. Often, accuracy is used as an indicator of how well a model performs, but it is important to understand this is not an end all be all. Different metrics should be prioritized under different circumstances. In terms of this analysis, it is very important to prioritize predicting the which images contain the blue tarps correctly because the this is a life or death situation. If this is done incorrectly, different areas of Haiti could have been searched instead of the correct areas where survivors are displaced. This incorrect search could waste valuable time in saving lives.

Digging into this further, prioritizing True Positive Rate, False Positive Rate and Precision should be key for this analysis. These metrics deal with how the algorithms perform in correctly predicting which images contain blue tarps. Particularly, Precision, which determines when a Blue Tarp is predicted how often is the prediction correct, seems to be the most relevant. This is because Precision lets the evaluators of the Model determine if the blue tarp predictions can truly be trusted. If the Precision is low, this means the Model poorly predicts blue tarps and if not considered, rescue efforts could be prioritized in incorrect areas of Haiti not containing displaced citizens. Similarly, False Positive Rate and True Positive Rate should be taken into account for similar reasons: ensuring the prediction of blue tarps is as accurate as possible.

Thinking of Precision as a key metric, it is good to see that Penalized Logistic Regression performs strongly for both the cross-validation data and the hold-out data and is one of the main reasons why it is chosen as the top performing algorithm overall.

7.5 Conclusion #5: Do you think this analysis could be helpful in saving human lives?

Overall, because all of these algorithms have such strong performance in terms of predicting which images have blue tarps, it is clear that this Modeling could have been very helpful in saving human lives during the Haitian earthquake disaster and during future disasters. If the disaster effort takes pictures like they had done here, as new images were taken, they can be fed into the algorithm to determine which have blue tarps without having to sift through each image one-by-one. The images that are predicted to have blue tarps could be prioritized and searched first to find and save survivors of natural disasters. This would have to be done quickly and efficiently and if a disaster was actually going on, data should be live fed into the Model to save time. Additionally, as mentioned before, if the Model has to be refit and an algorithm other than Penalized Logistic Regression is used with large training time complexity (Random Forest, SVM, KNN, etc.), this time complexity could be an issue depending on the number of images taken. A plan would have to be in place to maximize the area covered with the fewest images.

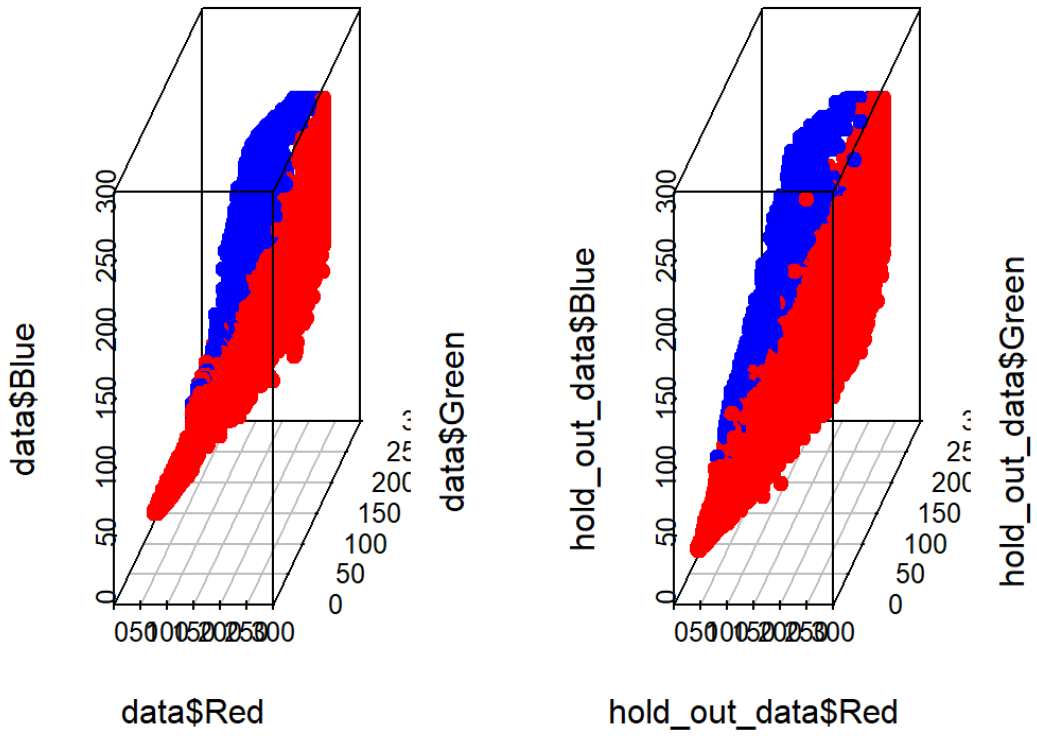
It is important to note that the Models built here would only be particularly effective when the survivors are using blue tarps as shelter. Otherwise the Models would need to be rebuilt to identify another type of shelter. Maybe the relief effort could airdrop blue tarps so that survivors can find them. This would ensure that these are used and would be identified by the Models.

7.6 Conclusion #6: Is the dataset particularly suited for a specific algorithm?

Previously, before Hold-Out Data was added into the analysis, we discussed that imagery data from the Haitian Earthquake disaster seemed to be suited particularly well for prediction via K-Nearest Neighbors. K-Nearest Neighbors Algorithm is one of the most flexible classification method of all of the Models in this analysis, which means it can find much wider range of possible shapes to estimate the data. One of the downfalls of the flexible Models is that they can be more difficult to interpret. In this analysis, the goal is not to interpret the fits, but the most accurately predict where Blue Tarps are to quickly save survivors of the Haitian Earthquake.

With the Hold-Out Data added, this assumption seems to be incorrect; the KNN model performs poorly on the Hold-Out data, while Logistic and Penalized Logistic Regressions performed very well on both Training via cross validation and the hold-out test data. This means that the data is likely strongly suited for a logistic type regression. Logistic regression types perform well when there are a large number of observations and relatively few predictors which is the case in this data set. Additionally, logistic regressions perform well when the data set is linearly separable which could be argued based on the distribution of the data of blue tarps and not as shown in the plots below:

```
par(mfrow = c(1, 2))
scatterplot3d(x = data$Red, y = data$Green, z = data$Blue,
              color = c("red","blue")[as.factor(data$Tarp_dummy)], pch = 19)
scatterplot3d(x = hold_out_data$Red, y = hold_out_data$Green, z = hold_out_data$Blue,
              color =c("red","blue")[as.factor(hold_out_data$Tarp_dummy)], pch = 19)
```



Additionally, because it is shown that the data set via cross validation is prone to over-training as seen by the reduction in performance by the more flexible methods like KNN on the Hold-Out Data, this may be an indicator again that the data set is not suited to a flexible method and may be more suited to a logistic type regression (a more biased approach). Furthermore, because of this over-training issue, this validates why Penalized Logistic Regression (ElasticNet) performs so well on both data sets because it's goal as an algorithm is to introduce penalization methods so that overfitting is avoided by imposing a penalty on large fluctuations on the estimated parameters and thus on the fitted curve. Thus, the dataset may truly be suited to the Penalized Logistic Regression Algorithm