Disaster Relief Project: Part II

Quinton Mays

Aug 15, 2021

1 Introduction

At around 5:00 PM on January 12th, 2010, a magnitude 7.0 earthquake struck just west of Port-au-Prince, Haiti. The earthquake may have caused over 100,000 deaths, destroyed much of the islands infrastructure, and displaced millions of residents. In the immediate aftermath of the quake, many Haitians chose to shelter under tarps due to the risk of aftershocks causing buildings damaged by the earlier earthquake to collapse. Due to the damage to critical transportation and communications infrastructure, and the country's geography, coordination of relief efforts proved difficult because displaced persons were hard to locate.

To help locate refugees, a team from the Rochester Institute of Technology surveyed the affected region using aerial photography. Some examples from the aerial photography can be seen below:

Aerial Photo 1

Aerial Photo 2

Aerial Photo 3

As demonstrated in the photos above, human analysis of the photos to determine the presence of blue tarps is difficult. In addition, due to the scale of the disaster, a large land area needed to be covered, making the task of manually identifying photographs with blue tarps in them very resource intensive process. Therefore, an algorithm needed to be determined that could quickly and accurately locate potential displaced persons so that humanitarian aid resources could be deployed to those areas.

This project will focus on determining the appropriate algorithm for this application.

2 Training Data / EDA

The following R packages were utilized in this project:

```
library(tidyverse)
library(skimr)
library(GGally)
library(caret)
library(broom)
library(yardstick)
library(class)
library(plotly)
library(plotROC)
library(purrr)
library(patchwork)
library(doParallel)
library(cowplot)
library(ROCR)
```

The data was supplied in the form of a .csv file. The function `read_csv` from the package `tidyverse` was utilized to read in and store the data in a data frame object called `raw_data`. This data frame will serve as the record of the as-received data.

```
raw_data <- read_csv('HaitiPixels.csv',
                     col_types = list(col_factor(),
                                      col_double(),
                                      col_double(),
                                      col_double()))
```

Next, the `skimr` package was utilized to provide a summary of the imported data:

```
skim(raw_data)
```

Data summary

Name	raw_data
Number of rows	63241
Number of columns	4
_____	
Column type frequency:	
factor	1
numeric	3
_____	
Group variables	None

Variable type: factor

skim_variable	n_missing	complete_rate	ordered	n_unique	top_counts
Class	0	1	FALSE	5	Veg: 26006, Soi: 20566, Roo: 9903, Var: 4744

Variable type: numeric

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100	hist
Red	0	1	162.98	78.88	48	80	163	255	255	
Green	0	1	153.66	72.64	48	78	148	226	255	
Blue	0	1	125.14	61.39	44	63	123	181	255	

The skim indicates that the dataframe contains four variables: the `Red`, `Blue`, and `Green` (RGB) color intensity of each pixel, which ranges from 0 to 255, and the classification of the pixel contents 'Class`, which is summarized in the table below:

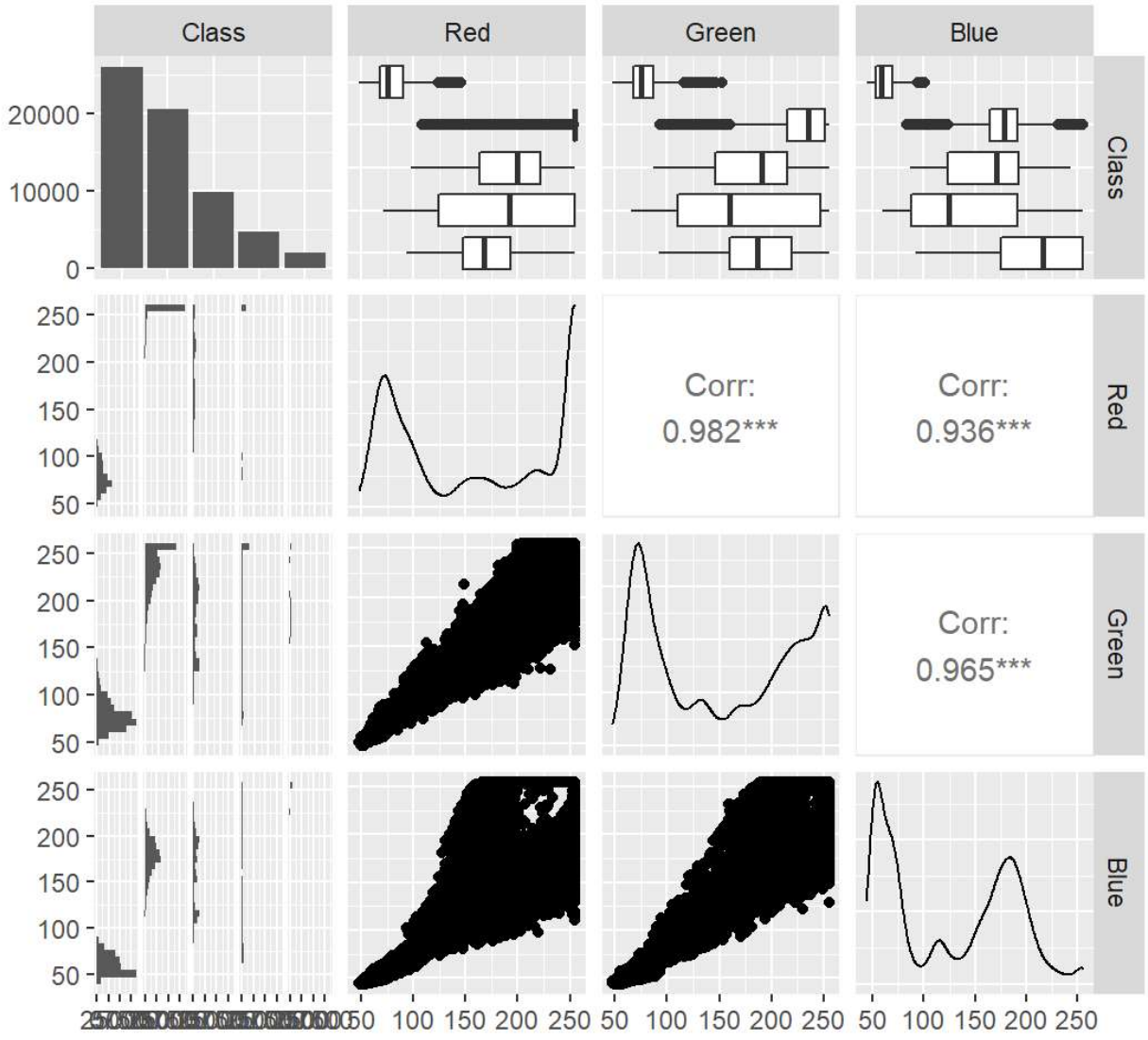
```
knitr::kable(table(raw_data$Class), col.names = c('Class',
                                                  'Observations'))
```

Class	Observations
Vegetation	26006
Soil	20566
Rooftop	9903
Various Non-Tarp	4744
Blue Tarp	2022

There are five levels within the categorical variable `Class`. The target class, Blue Tarp, only makes up a small portion of the data contained in this dataset. This is not surprising, as the dataset contains pixel information for a large area of land, with only a small portion of the land occupied by refugees.

Next, the function `ggpairs` from the package `Ggally` was used to visualize data distributions and correlations:

```
ggpairs(raw_data)
```



There are several points of interest in the `ggpairs` plot. First, the color values seem to be highly correlated with one another. Second, for the blue tarp classification, the blue value appears to be higher on average than the other classes. This can also be observed in the density curve for blue values, where there is a large concentration of blue values around 200.

3 Part I : Model Training

3.1 Setup

After conducting the exploratory data analysis, a working copy of the raw data was saved into a new dataframe called `df` .

```
df = raw_data
```

During exploratory data analysis, it was observed that the variable of interest, Class, has five levels. As the objective of this project is to predict if the pixel color values associated with a given area indicates if refugees are sheltering in this area, this categorical variable was collapsed from five classes to two classes. The new classes, `BT` (Blue Tarp) and `NBT` (Not Blue Tarp) will be the levels of the new response variable, `collpase.Class` . In addition, the seed 304 was chosen for repeatable random splits for cross validation.

```
set.seed(304)
df$collapse.Class = as.factor(ifelse(df$Class == "Blue Tarp", "BT", "NBT"))
df = subset(df, select = -Class)
df$collapse.Class = relevel(df$collapse.Class, ref = "NBT")
df$collapse.Class <- factor(df$collapse.Class, levels=rev(levels(df$collapse.Class)))
plot_preds = data.frame(df$collapse.Class)
```

After collapsing the response variable, the `contrasts` function was used to check the leveling in `collapse.Class` :

```
contrasts(df$collapse.Class)
```

```
#>     NBT
#> BT    0
#> NBT   1
```

A summary of the new variable `collapse.Class` can be found below:

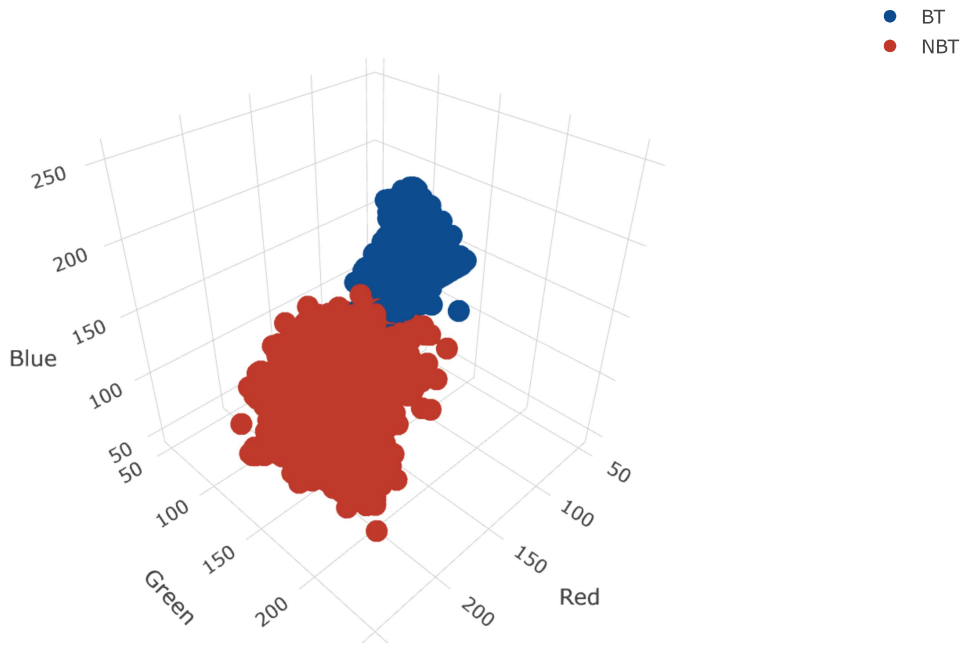
```
knitr::kable(table(df$collapse.Class), col.names = c('Class',
                                                     'Observations'))
```

Class	Observations
BT	2022
NBT	61219

It is clear that the count of pixels where the positive response, `BT` , is observed is sparse compared to the overall number of pixels observed. After the reduction in the number of levels in `Class` , the distribution of data in the 3D parameter space can be visualized with clarity. The package `plotly` was used to generate an interactive 3D scatter plot of each of the observations.

```
fig <- plot_ly(df, x = ~Red, y = ~Green, z = ~Blue, color = ~collapse.Class, colors = c('#0C4B8E','#BF382A'))
fig <- fig %>% add_markers()
fig <- fig %>% layout(title = 'Distribution of Observations in 3D Parameter Space',
                      scene = list(xaxis = list(title = 'Red'),
                                   yaxis = list(title = 'Green'),
                                   zaxis = list(title = 'Blue')))

fig
```

Distribution of Observations in 3D Parameter Space



The scatter plot indicates that there is a somewhat clear distinction between the classes `BT` and `NBT`, with BT pixels having higher Blue values than NBT pixels. However, it is unclear what form the decision boundary should take to create an optimal classification model for this data. Therefore, several classification models were considered including:

- Logistic Regression
- Linear Discriminant Analysis (LDA)
- Quadratic Discriminant Analysis (QDA)
- K-Nearest Neighbors (KNN)
- Penalized Logistic Regression (ElasticNet)

To begin the model selection a function, `trainControl`, from the `caret` package was utilized to set up a 10-fold cross validation to evaluate each of the models. Additional parameters were added to save the calculated probabilities for each observation, to save the predictions from the cross validation, and to allow for parallel processing.

```
ctrl = trainControl(method = 'cv',
                    number = 10,
                    classProbs = TRUE,
                    savePredictions = TRUE,
                    allowParallel = TRUE)
```

3.2 Logistic Regression

The first model examined was a logistic regression model. The function `train` from the package `caret` was used to fit a model to predict `collapse.Class` using the predictors `Red`, `Green`, and `Blue`. The model was then evaluated using a 10-fold cross validation that was setup with the `trainControl` function.

```
model.logistic = train(form = collapse.Class~.,
                       data = df,
                       method = 'glm',
                       trControl = ctrl,
                       family = 'binomial',
                       trace = FALSE)
model.logistic
```

```
#> Generalized Linear Model
#>
#> 63241 samples
#>     3 predictor
#>     2 classes: 'BT', 'NBT'
#>
#> No pre-processing
#> Resampling: Cross-Validated (10 fold)
#> Summary of sample sizes: 56917, 56917, 56917, 56917, 56917, 56917, ...
#> Resampling results:
#>
#>   Accuracy   Kappa
#>   0.9953037  0.9210554
```

```
model.logistic$finalModel %>% broom::tidy()
```

```
#> # A tibble: 4 x 5
#>   term        estimate std.error statistic   p.value
#>   <chr>          <dbl>     <dbl>     <dbl>     <dbl>
#> 1 (Intercept)   -0.210    0.185      -1.14 2.56e-  1
#> 2 Red            0.260    0.0126     20.6  1.42e- 94
#> 3 Green          0.218    0.0133     16.4  1.46e- 60
#> 4 Blue          -0.472    0.0155    -30.5  1.88e-204
```

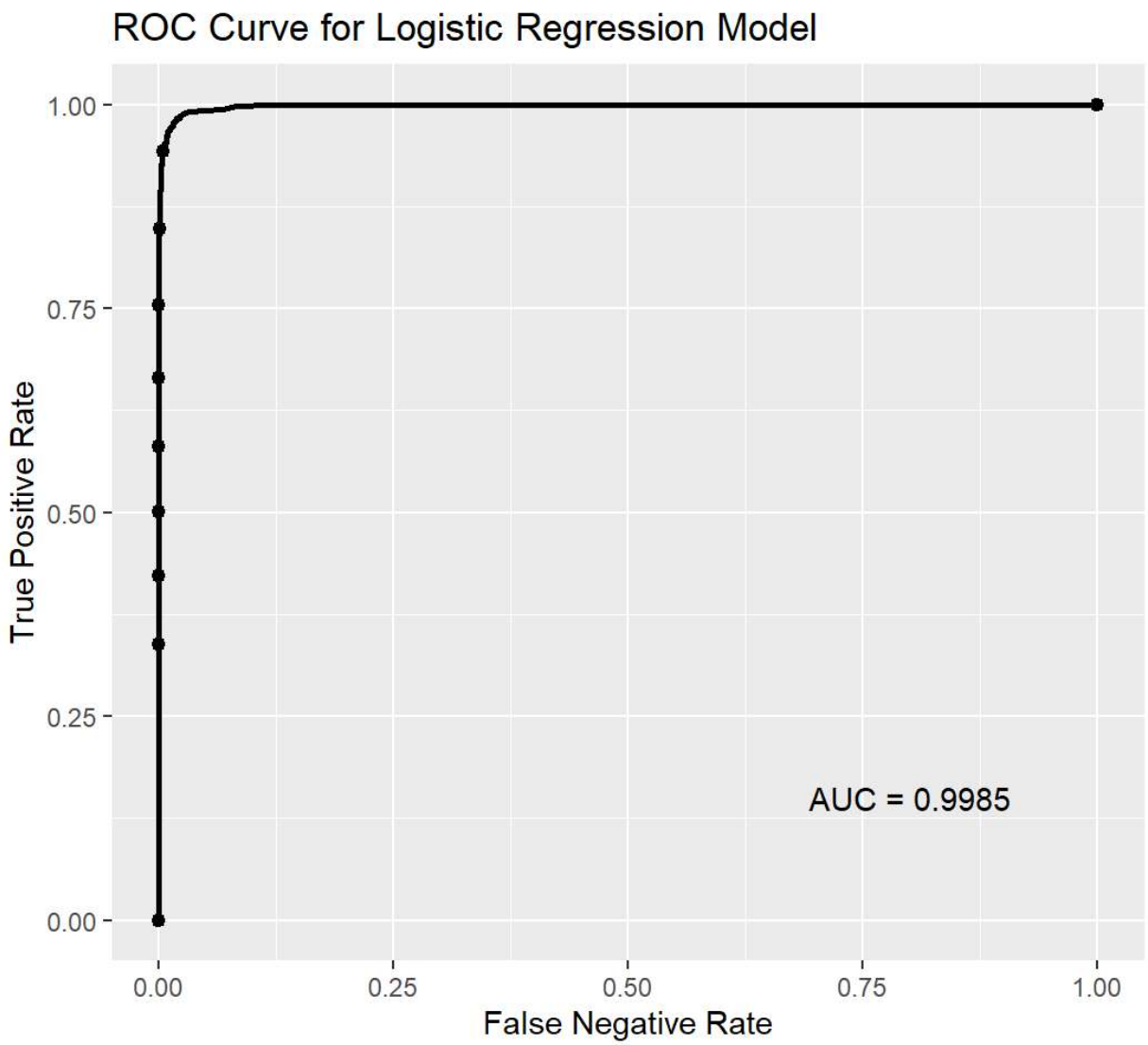
The output from the cross validation indicates that this model has very high accuracy (99.530367%). The model form also indicates that all of the predictors chosen (red, green, and blue) are highly important in predicting if a blue tarp is present (no p-values over 10^{-50}!) However, accuracy is not the only characteristic of the model that is important for this application, as it is important that the model correctly identifies areas that have blue tarps and does not waste aid resources by incorrectly labeling areas that do not contain blue tarps as containing them. To further investigate the logistic regression model's performance, a confusion matrix is needed:

```
confusionMatrix(model.logistic)
```

```
#> Cross-Validated (10 fold) Confusion Matrix
#>
#> (entries are percentual average cell counts across resamples)
#>
#>           Reference
#> Prediction   BT  NBT
#>        BT   2.8  0.1
#>        NBT  0.4 96.7
#>
#>  Accuracy (average) : 0.9953
```

The cross-validation confusion matrix indicates that the model has a sensitivity, or percentage of correctly identified blue tarp areas out of all blue tarp areas of 87.5% and a specificity, or percentage of correctly identified non-blue tarp areas out of all non-blue tarp areas of 99.9%. This confusion matrix is generated using a cutoff value of 0.5, which may not be optimal for this model, as identification of blue tarp areas is of more value than correct identification of non-blue tarp areas. To begin investigating the ideal cutoff value, the model's Receiver Operating Characteristic (ROC) curve is generated:

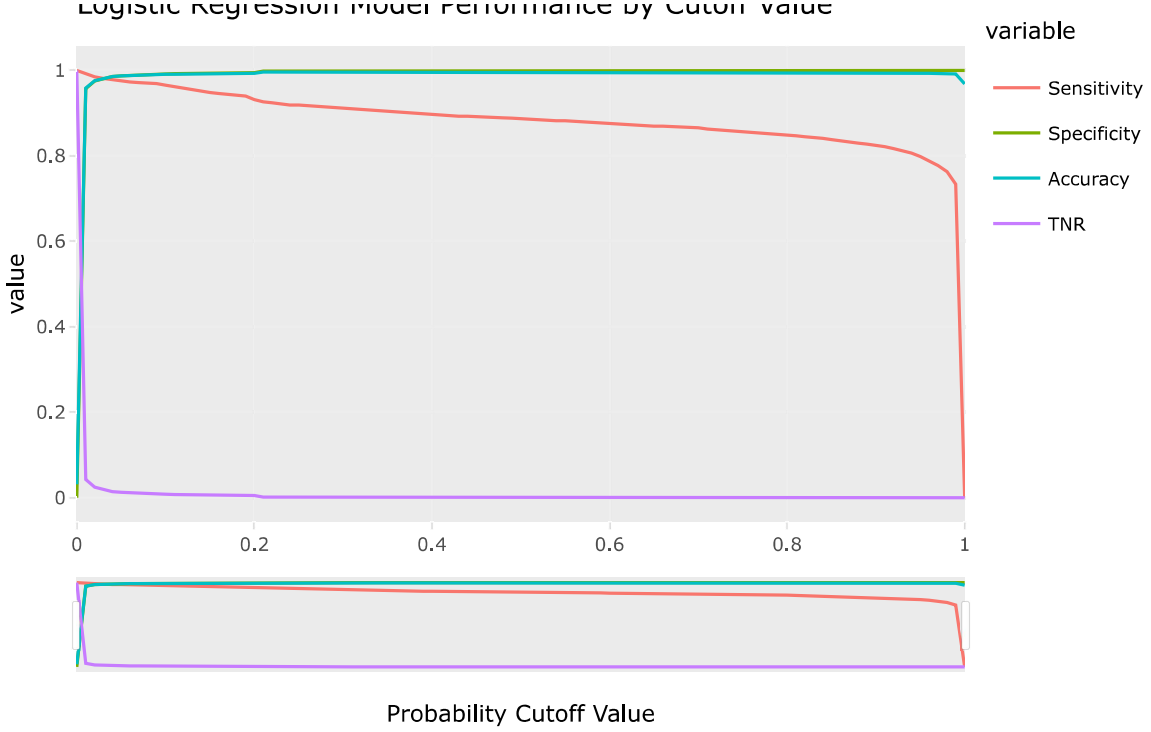
```
logistic.r = ggplot(model.logistic$pred, aes(m = BT, d = obs)) +
  geom_roc(labels = FALSE, increasing = FALSE) +
  labs(title = 'ROC Curve for Logistic Regression Model') +
  xlab('False Negative Rate') +
  ylab('True Positive Rate')
logistic.r + annotate("text", x = .8, y = .15, label = paste("AUC =", round(calc_auc(logistic.r)$AUC, 4)))
```



The ROC curve's proximity to the top left of the plot and it's associated area under the curve (AUC) above indicates that the logistic regression model performs very well at predicting if a given pixel represents a blue tarp or not. To determine the optimal cutoff value for the model, an interactive plot was created to explore the model's performance metrics for different cutoff values:

```
sens = c()
spec = c()
acc = c()
tnr = c()
cutoff = seq(0, 1, by=0.01)
for (i in 1:length(cutoff)){
  newpred = factor(ifelse(model.logistic$pred$BT > cutoff[i], "BT", "NBT"), levels = c("BT", "NBT"))
  sens[i] = sens_vec(model.logistic$pred$obs, newpred)
  spec[i] = spec_vec(model.logistic$pred$obs, newpred)
  acc[i] = accuracy_vec(model.logistic$pred$obs, newpred)
  tnr[i] = 1 - spec[i]
}
cutoff_plot_data = data.frame(Cutoff = cutoff, Sensitivity = sens, Specificity = spec, Accuracy = acc, TNR = tn
        r)
p = ggplot(reshape2::melt(cutoff_plot_data, id.var='Cutoff')) +
  geom_line(aes(x = Cutoff, y = value, color = variable)) +
  scale_x_continuous(limits = c(0,1), breaks = seq(0,1,0.1)) +
  labs(title = 'Logistic Regression Model Performance by Cutoff Value') +
  xlab('Probability Cutoff Value')
ggplotly(p, dynamicTicks = TRUE) %>% rangeslider() %>% layout(hovermode = 'x')
```

Logistic Regression Model Performance by Cutoff Value



After investigating the interactive plot, a cutoff value of 0.1 was selected for the model. The associated confusion matrix for this value is shown below:

```
newpred = factor(ifelse(model.logistic$pred$BT > 0.1, "BT", "NBT"), levels = c("BT", "NBT"))
plot_preds = cbind(plot_preds,newpred)
logistic.cm = confusionMatrix(newpred, model.logistic$pred$obs)
logistic.cm
```

```
#> Confusion Matrix and Statistics
#>
#>           Reference
#> Prediction    BT   NBT
#>        BT   1952   510
#>        NBT    70 60709
#>
#>                Accuracy : 0.9908
#>                  95% CI : (0.9901, 0.9916)
#>     No Information Rate : 0.968
#>     P-Value [Acc > NIR] : < 2.2e-16
#>
#>                   Kappa : 0.8659
#>
#>  Mcnemar's Test P-Value : < 2.2e-16
#>
#>             Sensitivity : 0.96538
#>             Specificity : 0.99167
#>          Pos Pred Value : 0.79285
#>          Neg Pred Value : 0.99885
#>              Prevalence : 0.03197
#>          Detection Rate : 0.03087
#>    Detection Prevalence : 0.03893
#>       Balanced Accuracy : 0.97853
#>
#>        'Positive' Class : BT
#>
```

Using a cutoff of 0.1 improves the model's sensitivity from 87.5% to 96.5% while only reducing its specificity from 0.995 to 0.991, which represents a large improvement in the model's utility to aid workers.

A summary of the logistic regression model with the selected cutoff value can be found below:

```
logistic.results = data.frame(Model = 'Logistic Regression', Tuning = 'NA', AUROC = round(calc_auc(logistic.r)$
        AUC, 4), Threshold = 0.1, Accuracy = round(logistic.cm$overall[[1]],4), TPR = round(logistic.cm$byClas
        s[[1]],4), FNR = round(1-logistic.cm$byClass[[2]],4), Precision = round(logistic.cm$byClass[[5]],4))
knitr::kable(logistic.results)
```

Model	Tuning	AUROC	Threshold	Accuracy	TPR	FNR	Precision
Logistic Regression	NA	0.9985	0.1	0.9908	0.9654	0.0083	0.7929

3.3 LDA

The next model that was explored was a linear discriminant analysis (LDA) model. LDA may improve on the performance of the logistic regression model by generating a linear decision boundary that maximized the separation between the two classes. The model was trained using the `train` method of the `caret` class and evaluated using a 10-fold cross validation:

```
model.lda = train(collapse.Class~.,
                  data = df,
                  method = 'lda',
                  trControl = ctrl,
                  trace = FALSE)
model.lda
```

```
#> Linear Discriminant Analysis
#>
#> 63241 samples
#>     3 predictor
#>     2 classes: 'BT', 'NBT'
#>
#> No pre-processing
#> Resampling: Cross-Validated (10 fold)
#> Summary of sample sizes: 56918, 56917, 56917, 56916, 56917, 56917, ...
#> Resampling results:
#>
#>   Accuracy   Kappa
#>   0.9839661  0.7533183
```

```
model.lda$finalModel
```

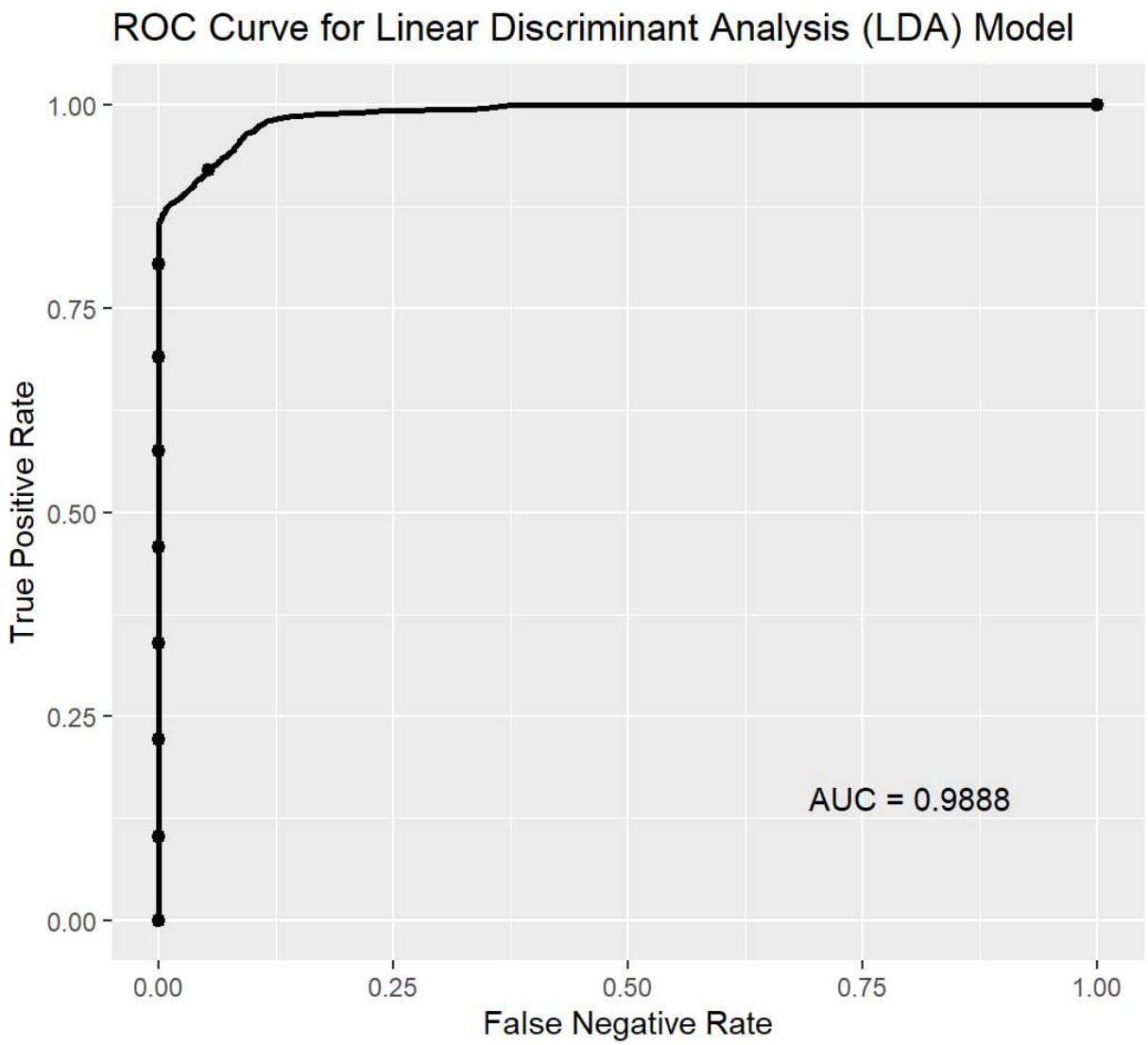
```
#> Call:
#> lda(x, grouping = y, trace = FALSE)
#>
#> Prior probabilities of groups:
#>         BT        NBT
#> 0.03197293 0.96802707
#>
#> Group means:
#>          Red    Green     Blue
#> BT  169.6627 186.4149 205.0371
#> NBT 162.7604 152.5808 122.4993
#>
#> Coefficients of linear discriminants: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

Again, the initial model fit shows promising accuracy. Further investigation using the model's confusion matrix and ROC curve is shown below:

```
confusionMatrix(model.lda)
```

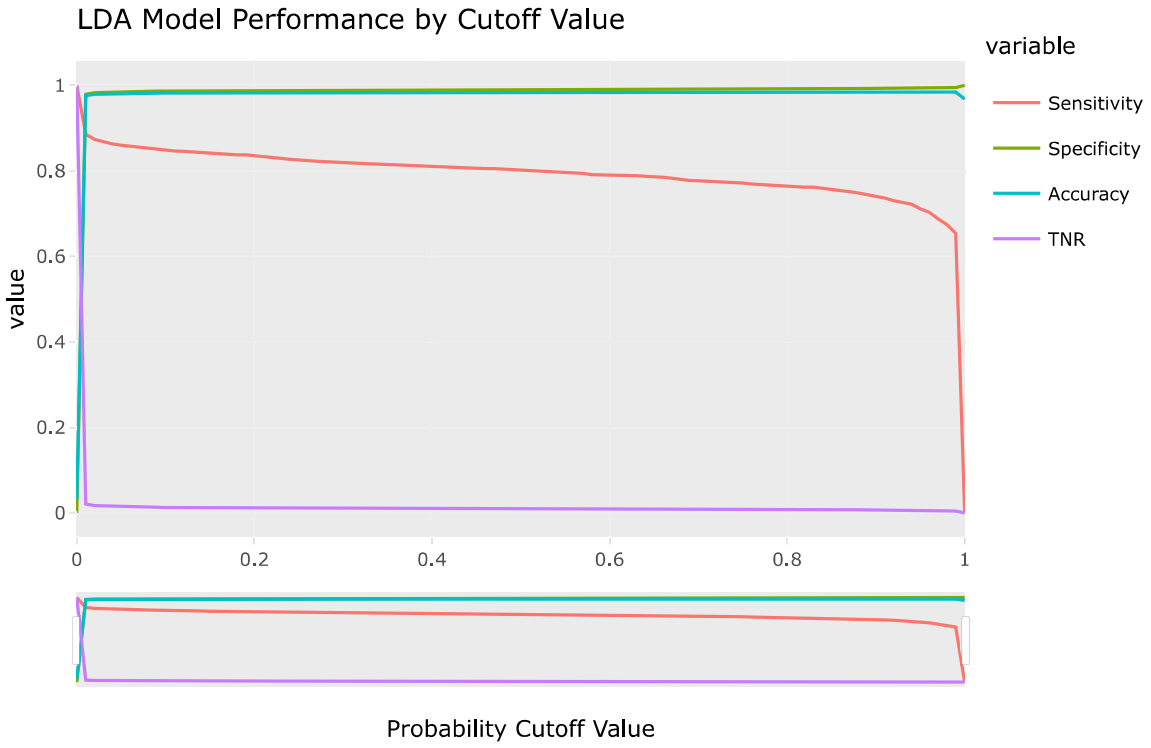
```
#> Cross-Validated (10 fold) Confusion Matrix
#>
#> (entries are percentual average cell counts across resamples)
#>
#>           Reference
#> Prediction   BT  NBT
#>        BT   2.6  1.0
#>        NBT  0.6 95.8
#>
#>  Accuracy (average) : 0.984
```

```
lda.r = ggplot(model.lda$pred, aes(m = BT, d = obs)) +
  geom_roc(labels = FALSE, increasing = FALSE) +
  labs(title = 'ROC Curve for Linear Discriminant Analysis (LDA) Model') +
  xlab('False Negative Rate') +
  ylab('True Positive Rate')
auc = round(calc_auc(lda.r)$AUC, 4)
lda.r + annotate("text", x = .8, y = .15, label = paste("AUC =", round(calc_auc(lda.r)$AUC, 4)))
```



The confusion matrix for the LDA model indicates that the model has an accuracy of 98.3966059%. The ROC curve proximity to the upper left of the plot area and associated $AUC = 0.9888$, indicates that this model performs very well at categorizing blue tarp vs not blue tarp pixels overall. However, this is not the ideal measure of the performance of the model, as the 0.5 probability threshold used when generating the confusion matrix yields a True Positive Rate of $\frac{2.6}{2.6 + 0.6} = 0.8125$. The low TPR associated with the 0.5 cutoff is most likely due to the relative sparsity of BT data points compared to NBT data points, as seen in the EDA section. A low TPR is not ideal, as the model may classify blue tarp areas as non-blue tarp areas, causing the deployment of resources to endangered people to be slowed. Choosing another value for the cutoff value may improve the performance of the model. The performance metrics for the LDA model at different cutoff values is shown below:

```
sens = c()
spec = c()
acc = c()
tnr = c()
cutoff = seq(0, 1, by=0.01)
for (i in 1:length(cutoff)){
  newpred = factor(ifelse(model.lda$pred$BT > cutoff[i], "BT", "NBT"), levels = c("BT", "NBT"))
  sens[i] = sens_vec(model.lda$pred$obs, newpred)
  spec[i] = spec_vec(model.lda$pred$obs, newpred)
  acc[i] = accuracy_vec(model.lda$pred$obs, newpred)
  tnr[i] = 1 - spec[i]
}
cutoff_plot_data = data.frame(Cutoff = cutoff, Sensitivity = sens, Specificity = spec, Accuracy = acc, TNR = tn
        r)
p = ggplot(reshape2::melt(cutoff_plot_data, id.var='Cutoff')) +
  geom_line(aes(x = Cutoff, y = value, color = variable)) +
  scale_x_continuous(limits = c(0,1), breaks = seq(0,1,0.1)) +
  labs(title = 'LDA Model Performance by Cutoff Value') +
  xlab('Probability Cutoff Value')
ggplotly(p, dynamicTicks = TRUE) %>% rangeslider() %>% layout(hovermode = 'x')
```



The interactive plot above indicates that choosing a cutoff value of 0.05 to produce a blue tarp prediction improves the TPR of the model by 5%, to 86%. While not ideal, this improvement could result in the correct identification of many more temporary shelters to bring relief aid to. The confusion matrix for a cutoff of 0.05 is shown below:

```
newpred = factor(ifelse(model.lda$pred$BT > 0.05, "BT", "NBT"), levels = c("BT", "NBT"))
plot_preds = cbind(plot_preds,newpred)
lda.cm = confusionMatrix(newpred, model.lda$pred$obs)
lda.cm
```

```
#> Confusion Matrix and Statistics
#>
#>           Reference
#> Prediction    BT   NBT
#>        BT   1739   911
#>        NBT   283 60308
#>
#>                Accuracy : 0.9811
#>                  95% CI : (0.98, 0.9822)
#>     No Information Rate : 0.968
#>     P-Value [Acc > NIR] : < 2.2e-16
#>
#>                   Kappa : 0.7348
#>
#>  Mcnemar's Test P-Value : < 2.2e-16
#>
#>             Sensitivity : 0.86004
#>             Specificity : 0.98512
#>          Pos Pred Value : 0.65623
#>          Neg Pred Value : 0.99533
#>              Prevalence : 0.03197
#>          Detection Rate : 0.02750
#>    Detection Prevalence : 0.04190
#>       Balanced Accuracy : 0.92258
#>
#>        'Positive' Class : BT
#>
```

A summary table for the LDA model performance is shown below:

```
lda.results = data.frame(Model = 'LDA', Tuning = 'NA', AUROC = round(calc_auc(lda.r)$AUC, 4), Threshold = 0.1,
         Accuracy = round(lda.cm$overall[[1]],4), TPR = round(lda.cm$byClass[[1]],4), FNR = round(1-lda.cm$byCl
         ass[[2]],4), Precision = round(lda.cm$byClass[[5]],4))
knitr::kable(lda.results)
```

Model	Tuning	AUROC	Threshold	Accuracy	TPR	FNR	Precision
LDA	NA	0.9888	0.1	0.9811	0.86	0.0149	0.6562

3.4 QDA

The next model explored was a quadratic discriminant analysis (QDA) model. Like LDA, QDA generates a decision boundary between the classes that maximizes the separation between each class, however QDA does not assume that the classes have equal variance, and therefore provides a more flexible model fit and quadratic decision boundary, which may be useful in this case, where the decision boundary may be non-linear. Again, the model was trained using the `train` method of the `caret` class.

```
model.qda = train(collapse.Class~.,
                  data = df,
                  method = 'qda',
                  trControl = ctrl,
                  trace = FALSE)
model.qda
```

```
#> Quadratic Discriminant Analysis
#>
#> 63241 samples
#>     3 predictor
#>     2 classes: 'BT', 'NBT'
#>
#> No pre-processing
#> Resampling: Cross-Validated (10 fold)
#> Summary of sample sizes: 56917, 56917, 56917, 56916, 56917, 56917, ...
#> Resampling results:
#>
#>   Accuracy   Kappa
#>   0.9945446  0.9045071
```

```
model.qda$finalModel
```

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

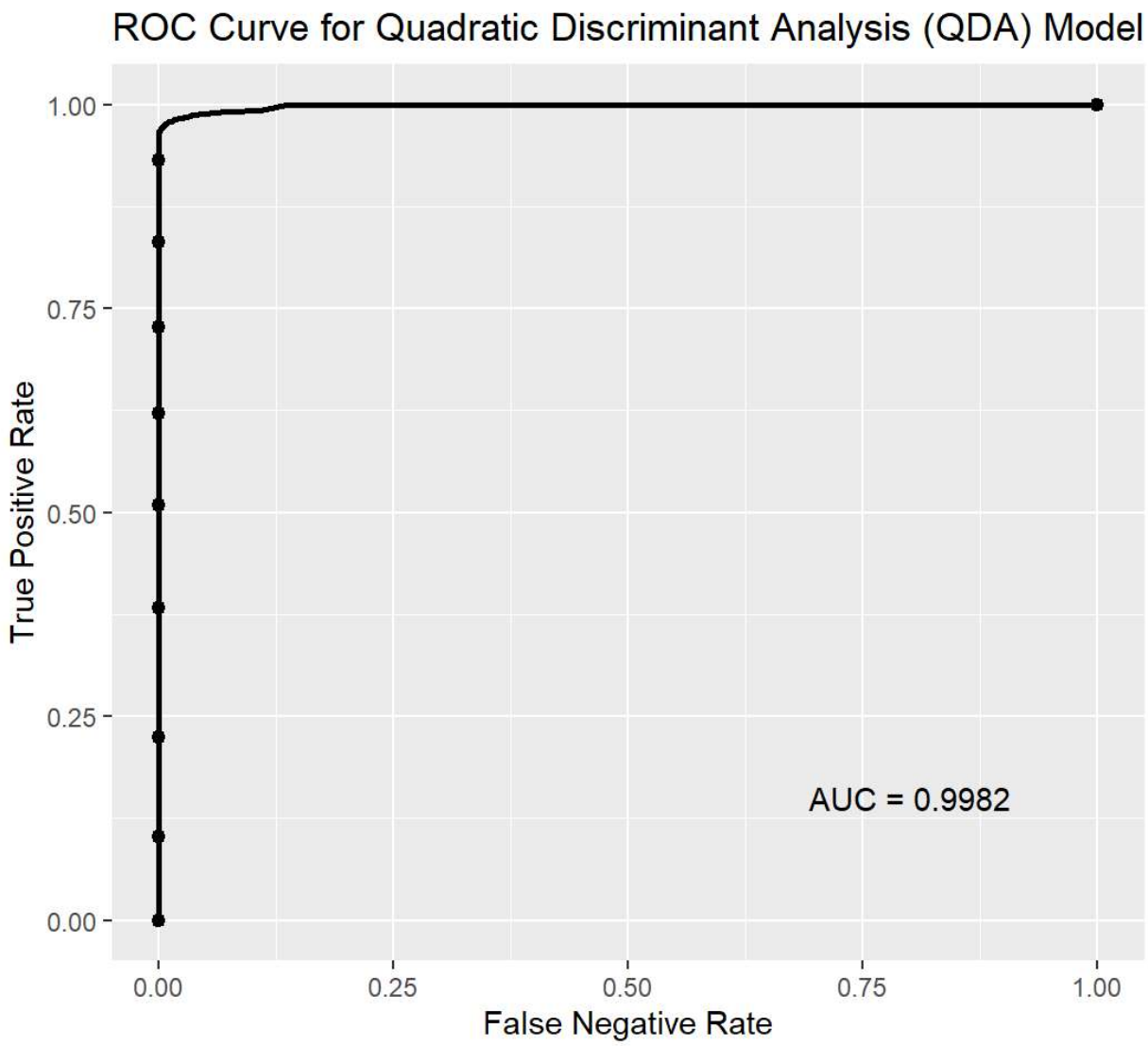
The QDA results shown above are promising, the overall model accuracy has improved to 99%. A confusion matrix was generated for the 10-fold cross validation to visualize model performance at a cutoff value of 0.5:

```
confusionMatrix(model.qda)
```

```
#> Cross-Validated (10 fold) Confusion Matrix
#>
#> (entries are percentual average cell counts across resamples)
#>
#>           Reference
#> Prediction   BT  NBT
#>        BT   2.7  0.0
#>        NBT  0.5 96.8
#>
#>  Accuracy (average) : 0.9945
```

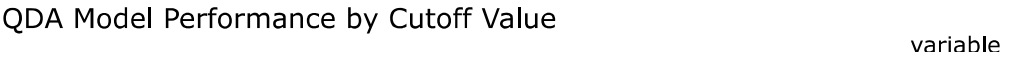
The confusion matrix indicates that the QDA model outperforms the linear discriminant analysis and is nearly identical to the performance of the logistic regression model. Next a receiver operating characteristic (ROC) curve was generated to visualize the model performance across all cutoff values:

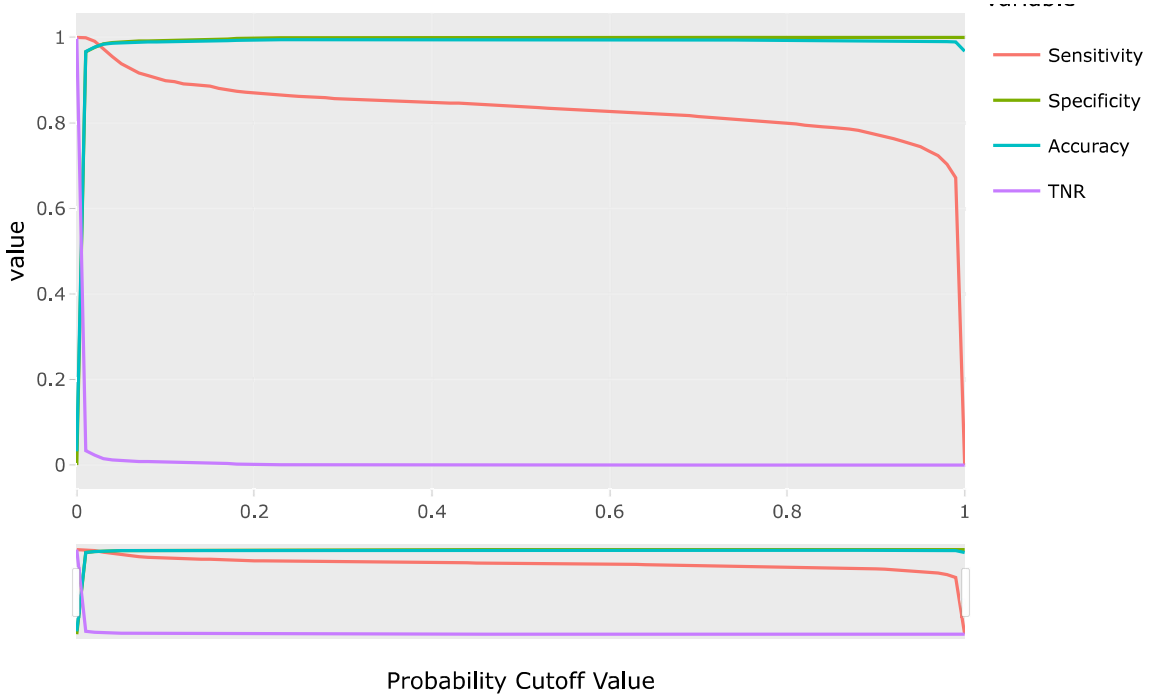
```
qda.r = ggplot(model.qda$pred, aes(m = BT, d = obs)) +
  geom_roc(labels = FALSE, increasing = FALSE) +
  labs(title = 'ROC Curve for Quadratic Discriminant Analysis (QDA) Model') +
  xlab('False Negative Rate') +
  ylab('True Positive Rate')
auc = round(calc_auc(qda.r)$AUC, 4)
qda.r + annotate("text", x = .8, y = .15, label = paste("AUC =", round(calc_auc(qda.r)$AUC, 4)))
```



The performance of the QDA model is nearly identical to the logistic regression model, with an AUC of 0.9982. Next, the cutoff value was optimized to balance sensitivity and specificity. The interactive plot below was utilized to select an optimal cutoff value:

```
sens = c()
spec = c()
acc = c()
tnr = c()
cutoff = seq(0, 1, by=0.01)
for (i in 1:length(cutoff)){
  newpred = factor(ifelse(model.qda$pred$BT > cutoff[i], "BT", "NBT"), levels = c("BT", "NBT"))
  sens[i] = sens_vec(model.qda$pred$obs, newpred)
  spec[i] = spec_vec(model.qda$pred$obs, newpred)
  acc[i] = accuracy_vec(model.qda$pred$obs, newpred)
  tnr[i] = 1 - spec[i]
}
cutoff_plot_data = data.frame(Cutoff = cutoff, Sensitivity = sens, Specificity = spec, Accuracy = acc, TNR = tn
        r)
p = ggplot(reshape2::melt(cutoff_plot_data, id.var='Cutoff')) +
  geom_line(aes(x = Cutoff, y = value, color = variable)) +
  scale_x_continuous(limits = c(0,1), breaks = seq(0,1,0.1)) +
  labs(title = 'QDA Model Performance by Cutoff Value') +
  xlab('Probability Cutoff Value')
ggplotly(p, dynamicTicks = TRUE) %>% rangeslider() %>% layout(hovermode = 'x')
```





It is clear that beyond a cutoff of 0.1, model performance is not very much improved, therefore a confusion matrix was generated using a cutoff value of 0.1.

```
newpred = factor(ifelse(model.qda$pred$BT > 0.1, "BT", "NBT"), levels = c("BT", "NBT"))
plot_preds = cbind(plot_preds,newpred)
qda.cm = confusionMatrix(newpred, model.qda$pred$obs)
qda.cm
```

```
#> Confusion Matrix and Statistics
#>
#>           Reference
#> Prediction    BT   NBT
#>        BT   1818   456
#>        NBT   204 60763
#>
#>                Accuracy : 0.9896
#>                  95% CI : (0.9887, 0.9903)
#>     No Information Rate : 0.968
#>     P-Value [Acc > NIR] : < 2.2e-16
#>
#>                   Kappa : 0.841
#>
#>  Mcnemar's Test P-Value : < 2.2e-16
#>
#>             Sensitivity : 0.89911
#>             Specificity : 0.99255
#>          Pos Pred Value : 0.79947
#>          Neg Pred Value : 0.99665
#>              Prevalence : 0.03197
#>          Detection Rate : 0.02875
#>    Detection Prevalence : 0.03596
#>       Balanced Accuracy : 0.94583
#>
#>        'Positive' Class : BT
#>
```

Using a cutoff of 0.1, the model's TPR improves from 84.4% to 90.0%. The chosen cutoff has increased the sensitivity by nearly 6% while only decreasing the global model accuracy by less than a percent.

A summary table of the QDA model results can be found below:

```
qda.results = data.frame(Model = 'QDA', Tuning = 'NA', AUROC = round(calc_auc(qda.r)$AUC, 4), Threshold = 0.1,
        Accuracy = round(qda.cm$overall[[1]],4), TPR = round(qda.cm$byClass[[1]],4), FNR = round(1-qda.cm$byCl
        ass[[2]],4), Precision = round(qda.cm$byClass[[5]],4))
knitr::kable(qda.results)
```

Model	Tuning	AUROC	Threshold	Accuracy	TPR	FNR	Precision
QDA	NA	0.9982	0.1	0.9896	0.8991	0.0074	0.7995

3.5 KNN

The next model tested was a K-Nearest Neighbors (KNN) model. While the parametric models have all shown to be quite accurate in predicting the presence of tarps, perhaps the non-parametric and highly flexible KNN model will yield better results.

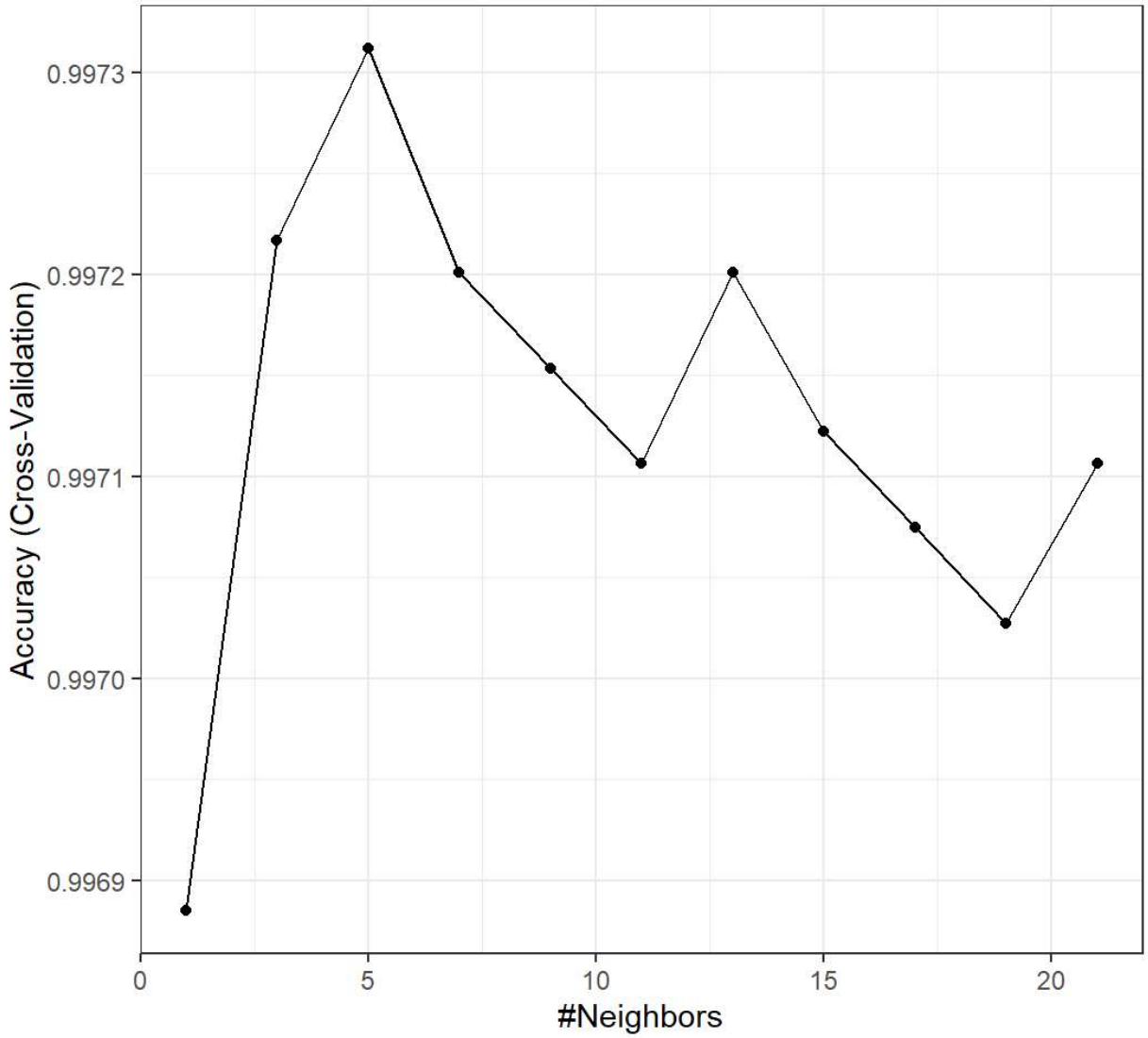
3.5.1 Tuning Parameter k

The first step in selecting the KNN model is to determine the optimal value for k, or how many neighbors will "vote" on each point to determine its class. To obtain an optimal value for k the `train` method of the `caret` package is utilized:

```
model.knn = train(collapse.Class~.,
                  data = df,
                  method = 'knn',
                  trControl = ctrl,
                  tuneGrid = expand.grid(k = seq(1, 21, by = 2)))
model.knn
```

```
#> k-Nearest Neighbors
#>
#> 63241 samples
#>     3 predictor
#>     2 classes: 'BT', 'NBT'
#>
#> No pre-processing
#> Resampling: Cross-Validated (10 fold)
#> Summary of sample sizes: 56917, 56918, 56916, 56917, 56917, 56917, ...
#> Resampling results across tuning parameters:
#>
#>   k   Accuracy   Kappa
#>    1  0.9968849  0.9494116
#>    3  0.9972170  0.9550278
#>    5  0.9973119  0.9567646
#>    7  0.9972012  0.9550364
#>    9  0.9971538  0.9542424
#>   11  0.9971063  0.9534146
#>   13  0.9972012  0.9548647
#>   15  0.9971221  0.9536106
#>   17  0.9970747  0.9527917
#>   19  0.9970273  0.9520740
#>   21  0.9971063  0.9533568
#>
#> Accuracy was used to select the optimal model using the largest value.
#> The final value used for the model was k = 5.
```

```
ggplot(model.knn) + theme_bw()
```



The plot and model summary above indicate that $k = 5$ is optimal based on a 10-fold cross validation. It appears that model performance increases to this point and then drops off as k continues to increase. The confusion matrix for $k = 5$ is shown below:

```
knn.cm = confusionMatrix(model.knn$pred$pred,model.knn$pred$obs)
plot_preds = cbind(plot_preds,model.knn$pred$pred)
knn.cm
```

```
#> Confusion Matrix and Statistics
#>
#>           Reference
#> Prediction     BT    NBT
#>        BT   21279   1035
#>        NBT    963 672374
#>
#>                Accuracy : 0.9971
#>                  95% CI : (0.997, 0.9973)
#>     No Information Rate : 0.968
#>     P-Value [Acc > NIR] : <2e-16
#>
#>                   Kappa : 0.9537
#>
#>  Mcnemar's Test P-Value : 0.1122
#>
#>             Sensitivity : 0.95670
#>             Specificity : 0.99846
#>          Pos Pred Value : 0.95362
#>          Neg Pred Value : 0.99857
#>              Prevalence : 0.03197
#>          Detection Rate : 0.03059
#>    Detection Prevalence : 0.03208
#>       Balanced Accuracy : 0.97758
#>
#>        'Positive' Class : BT
#>
```

It appears that this model is the best performing model explored so far, with an accuracy of 0.9971% and a TPR of 0.9567%. Since the class of each prediction is determined by its k-nearest neighbor's "votes", there is no cutoff value to select, and this model can interprted as-is. A table of the KNN-model's results can be found below:

```
knn.results = data.frame(Model = 'KNN', Tuning = 'k=5', AUROC = 'NA', Threshold = 'NA', Accuracy = round(knn.cm
         $overall[[1]],4), TPR = round(knn.cm$byClass[[1]],4), FNR = round(1-knn.cm$byClass[[2]],4), Precision
          = round(knn.cm$byClass[[5]],4))
knitr::kable(knn.results)
```

Model	Tuning	AUROC	Threshold	Accuracy	TPR	FNR	Precision
KNN	k=5	NA	NA	0.9971	0.9567	0.0015	0.9536

3.6 Penalized Logistic Regression (ElasticNet)

The final model examined in this exploration is a penalized logistic regression using ElasticNet. This model is similar to Least Absolute Shrinkage and Selection Operator (LASSO) and Ridge regression, in that it includes a penalty term. However, ElasticNet regression combines the best of both of those models, in that it allows for some of the coefficients to shrink, and some to go to 0. This property may allow the model to outperform the others that have already been examined.

3.6.1 Tuning Parameters

To begin, the tuning parameters, α and λ must be selected. The method `train` from the `caret` package was utilized to select these parameters:

```
model.elnet = train(
  collapse.Class ~ .,
  data = df,
  method = "glmnet",
  family = 'binomial',
  preProcess = c('center','scale'),
  trControl = ctrl,
  allowParallel = TRUE)
alpha = model.elnet$bestTune$alpha
lambda = model.elnet$bestTune$lambda
```

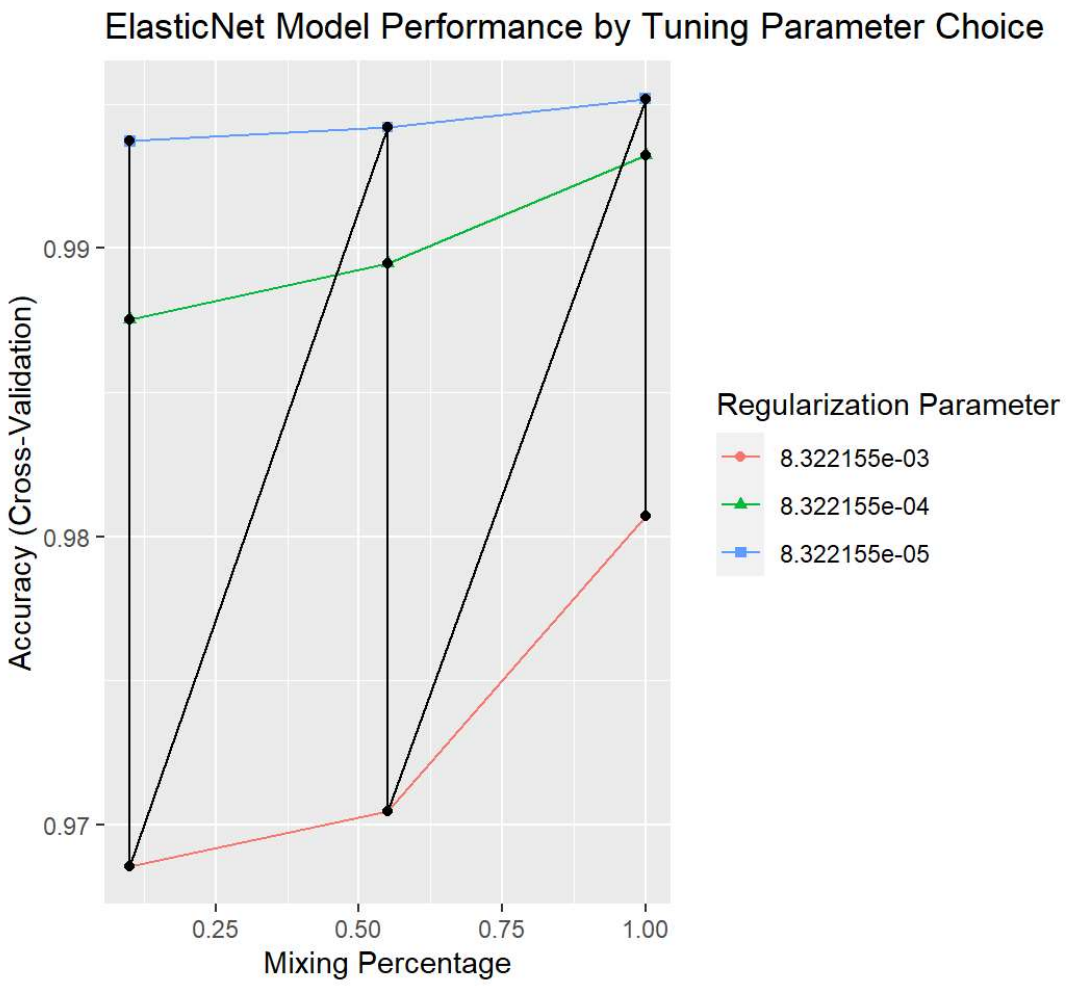
The `train` method examined 3 values for α and λ, and selected $\alpha = 1$ and $\lambda = 8.3221545 \times 10^{-5}$, indicating that the ElasticNet regression actually decided on a LASSO Regression due to $\alpha = 1$. The coefficients of the model selected is shown below:

```
coef(model.elnet$finalModel, model.elnet$finalModel$lambdaOpt)
```

```
#> 4 x 1 sparse Matrix of class "dgCMatrix"
#>                      1
#> (Intercept)  12.96764
#> Red          16.33537
#> Green        10.57268
#> Blue        -21.57395
```

A visualization of the model performance at each value of α and λ is shown below:

```
ggplot(model.elnet) + geom_line() + geom_point() + labs(title = 'ElasticNet Model Performance by Tuning Paramet
         er Choice')
```



The graphic indicates that the best value of the mixing percentage, α is 1 and the best value for the regularization parameter, λ is 0.00832. The value of λ indicates that the selected LASSO model very closely approximates a normal logistic regression model, as the penalty term is not heavily weighted. The low weight to the penalty term indicates that all three variables (`Red` , `Green` , and `Blue`) likely provide utility in classification of `BT` pixels vs `NBT` pixels.

3.6.2 Threshold Selection

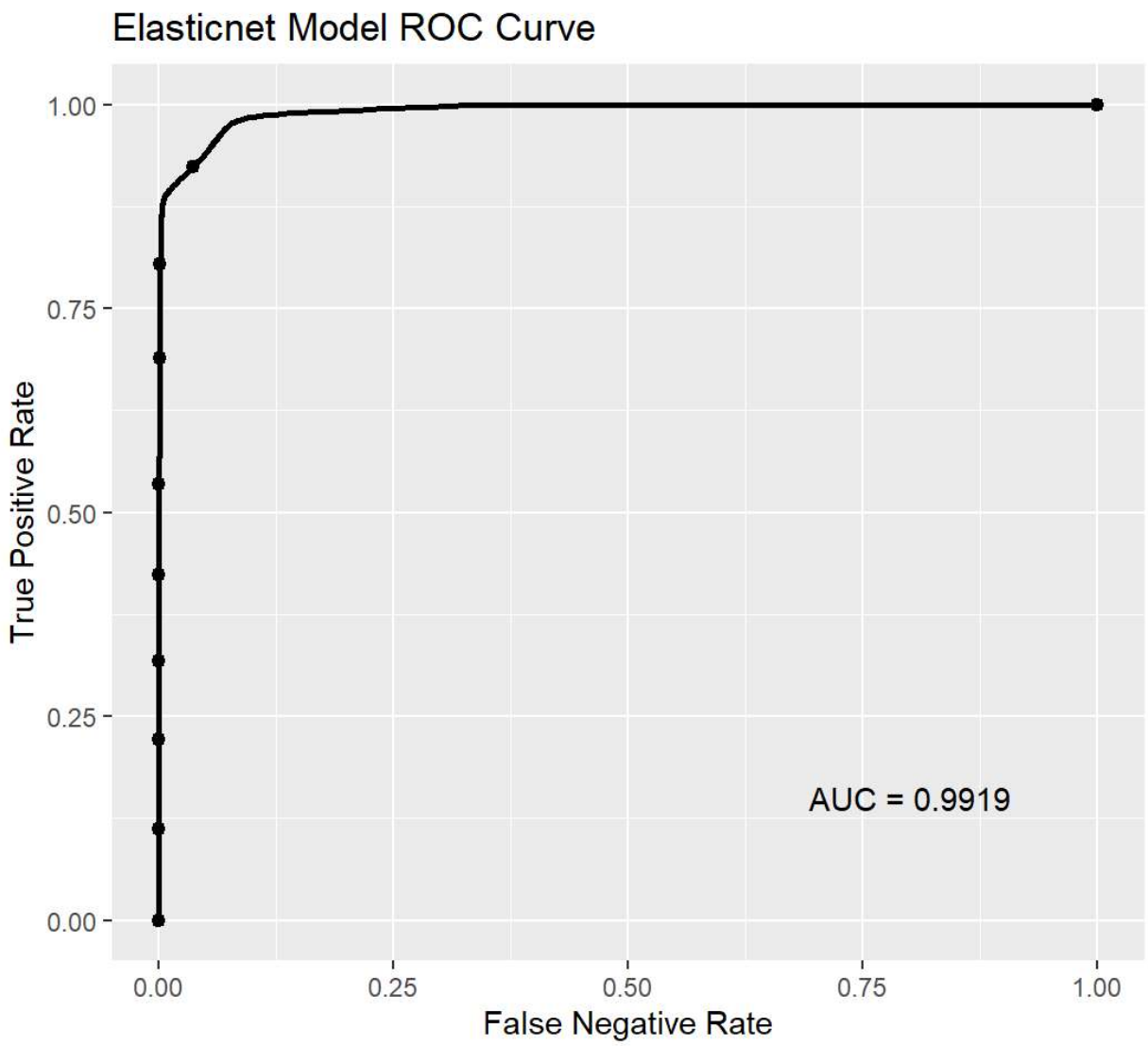
The model parameters above create a model with the following confusion matrix for a cutoff of 0.5:

```
confusionMatrix(model.elnet)
```

```
#> Cross-Validated (10 fold) Confusion Matrix
#>
#> (entries are percentual average cell counts across resamples)
#>
#>           Reference
#> Prediction   BT  NBT
#>        BT   2.8  0.1
#>        NBT  0.4 96.7
#>
#>  Accuracy (average) : 0.9952
```

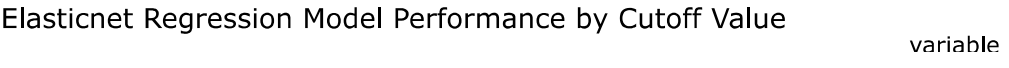
It appears that the ElasticNet model has very good performance, matching the performance of the other parametric models with an accuracy of 0.9952 and a TPR of 0.875. The ROC curve for the ElasticNet model is shown below:

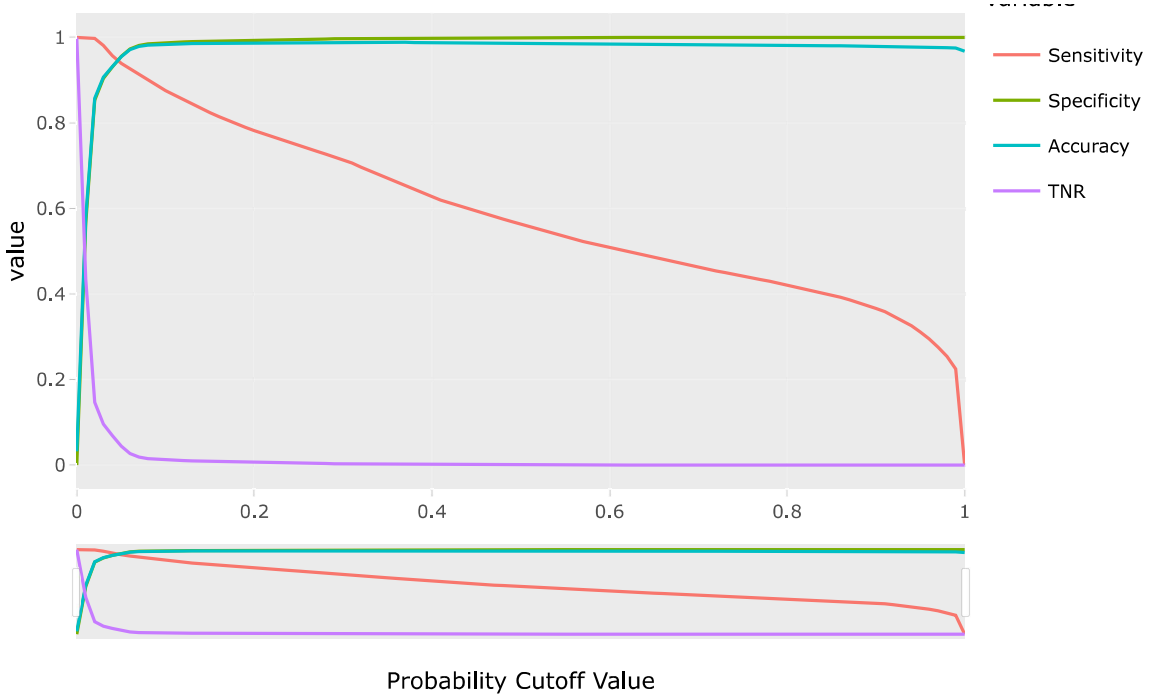
```
elnet.r = ggplot(model.elnet$pred, aes(m = BT, d = obs)) +
  geom_roc(labels = FALSE, increasing = FALSE) +
  labs(title = 'Elasticnet Model ROC Curve') +
  xlab('False Negative Rate') +
  ylab('True Positive Rate')
auc = round(calc_auc(elnet.r)$AUC, 4)
elnet.r + annotate("text", x = .8, y = .15, label = paste("AUC =", round(calc_auc(elnet.r)$AUC, 4)))
```



The ROC curve for the model indicates that the ROC curve has an $AUC = 0.9919$, which indicates a very impressive model. Finally, the interactive plot below was utilized to improve the TPR of the model by selecting a more optimal cutoff value:

```
sens = c()
spec = c()
acc = c()
tnr = c()
cutoff = seq(0, 1, by=0.01)
for (i in 1:length(cutoff)){
  newpred = factor(ifelse(model.elnet$pred$BT > cutoff[i], "BT", "NBT"), levels = c("BT", "NBT"))
  sens[i] = sens_vec(model.elnet$pred$obs, newpred)
  spec[i] = spec_vec(model.elnet$pred$obs, newpred)
  acc[i] = accuracy_vec(model.elnet$pred$obs, newpred)
  tnr[i] = 1 - spec[i]
}
cutoff_plot_data = data.frame(Cutoff = cutoff, Sensitivity = sens, Specificity = spec, Accuracy = acc, TNR = tn
        r)
p = ggplot(reshape2::melt(cutoff_plot_data, id.var='Cutoff')) +
  geom_line(aes(x = Cutoff, y = value, color = variable)) +
  scale_x_continuous(limits = c(0,1), breaks = seq(0,1,0.1)) +
  labs(title = 'Elasticnet Regression Model Performance by Cutoff Value') +
  xlab('Probability Cutoff Value')
ggplotly(p, dynamicTicks = TRUE) %>% rangeslider() %>% layout(hovermode = 'x')
```





A cutoff of 0.05 was selected, as TPR did not increase significantly beyond this point. The associated confusion matrix for the cutoff of 0.05 is shown below:

```
newpred = factor(ifelse(model.elnet$pred$BT > 0.05, "BT", "NBT"), levels = c("BT", "NBT"))
plot_preds = cbind(plot_preds,newpred)
```

```
#> Error in data.frame(..., check.names = FALSE): arguments imply differing number of rows: 695651, 569169
```

```
elnet.cm = confusionMatrix(newpred, model.elnet$pred$obs)
elnet.cm
```

```
#> Confusion Matrix and Statistics
#>
#>           Reference
#> Prediction     BT    NBT
#>        BT   17095  24445
#>        NBT   1103 526526
#>
#>                Accuracy : 0.9551
#>                  95% CI : (0.9546, 0.9556)
#>     No Information Rate : 0.968
#>     P-Value [Acc > NIR] : 1
#>
#>                   Kappa : 0.5524
#>
#>  Mcnemar's Test P-Value : <2e-16
#>
#>             Sensitivity : 0.93939
#>             Specificity : 0.95563
#>          Pos Pred Value : 0.41153
#>          Neg Pred Value : 0.99791
#>              Prevalence : 0.03197
#>          Detection Rate : 0.03004
#>    Detection Prevalence : 0.07298
#>       Balanced Accuracy : 0.94751
#>
#>        'Positive' Class : BT
#>
```

By decreasing the cutoff value to 0.05, the model's TPR increases to 0.939! However, the overall performance of the model suffers in this case, reducing the overall model accuracy to 0.955. However, this value was chosen to make sure that as many people in need were reached as possible. A summary table of the model results can be found below:

```
elnet.results = data.frame(Model = 'Elasticnet', Tuning = c('alpha = 1, lambda = 8.32e-0.5'), AUROC = round(cal
         c_auc(elnet.r)$AUC, 4), Threshold = 0.1, Accuracy = round(elnet.cm$overall[[1]],4), TPR = round(elnet.
         cm$byClass[[1]],4), FNR = round(1-elnet.cm$byClass[[2]],4), Precision = round(elnet.cm$byClass[[5]],4
         ))
knitr::kable(elnet.results)
```

Model	Tuning	AUROC	Threshold	Accuracy	TPR	FNR	Precision
Elasticnet	alpha = 1, lambda = 8.32e-0.5	0.9919	0.1	0.9551	0.9394	0.0444	0.4115

4 Part I : Results (Cross-Validation)

The summary of the results from each of the model fits can be found below:

```
results = rbind(logistic.results, lda.results, qda.results, knn.results, elnet.results)
knitr::kable(results)
```

Model	Tuning	AUROC	Threshold	Accuracy	TPR	FNR	Precision
Logistic Regression	NA	0.9985	0.1	0.9908	0.9654	0.0083	0.7929
LDA	NA	0.9888	0.1	0.9811	0.8600	0.0149	0.6562
QDA	NA	0.9982	0.1	0.9896	0.8991	0.0074	0.7995
KNN	k=5	NA	NA	0.9971	0.9567	0.0015	0.9536
Elasticnet	alpha = 1, lambda = 8.32e-0.5	0.9919	0.1	0.9551	0.9394	0.0444	0.4115

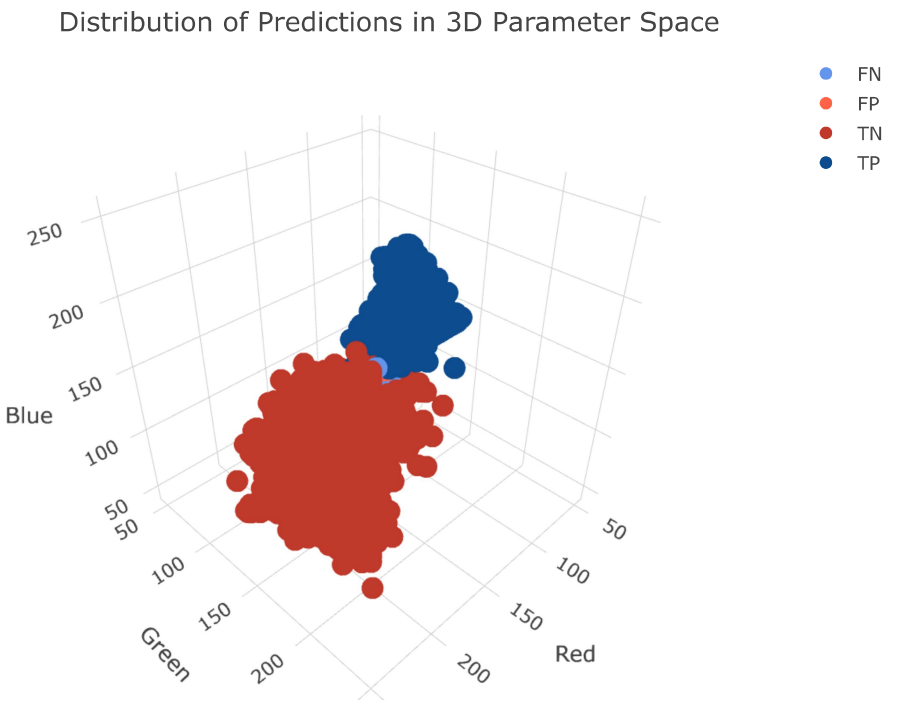
Each of the results was the average generated from a 10-fold cross validation of the data and the selection of the optimal cutoff value.

5 Part I : Conclusions

5.0.1 Conclusion #1

Out of the 5 models tested, K-Nearest Neighbors with $k = 5$ appears to be the best performing model. This is not surprising, as the 3D scatterplot generated during the exploratory data analysis indicated that the decision boundary may be highly non-linear. The flexible KNN model created a highly non-linear decision boundary which allowed the model to predict the presence of blue tarps very accurately. A visualization of the model predictions is shown below:

```
preds = predict(model.knn, newdata = df)
classification = preds == df$collapse.Class
classification = data.frame(preds, classification)
classification = ifelse(classification$preds == 'BT',
                        ifelse(classification$classification == TRUE, 'TP', 'FP'),
                        ifelse(classification$classification == TRUE, 'TN', 'FN'))
classification = as.factor(classification)
fig <- plot_ly(df, x = ~Red, y = ~Green, z = ~Blue, color = ~classification, colors = c('cornflowerblue', 'toma
        to', '#BF382A','#0C4B8E'))
fig <- fig %>% add_markers()
fig <- fig %>% layout(title = 'Distribution of Predictions in 3D Parameter Space',
                      scene = list(xaxis = list(title = 'Red'),
                                   yaxis = list(title = 'Green'),
                                   zaxis = list(title = 'Blue')))

fig
```



The traditional drawback of using a flexible model such as KNN is that there is high variance associated with these models. However, due to the large number of data points in this dataset the KNN model's 10-fold cross validation performance was excellent. If a smaller dataset was used to train each model, KNN may have had a larger cross validation error due to its natural higher variance. Overall, there can be a moderately-high level of confidence in the cross-validation results, as the dataset is very large compared to the number of variables and classes to predict.

5.0.2 Conclusion #2

The selected model, K-Nearest Neighbors, would be very helpful to aid humanitarian relief efforts for two reasons. First, aid services could have a high degree of confidence in the suggestions, as the KNN model has a very high TPR of 95.63%. This indicates that when the model indicates that an area contains blue tarps, the user can have a high degree of confidence in it. Second, the model has an overall accuracy of over 99%, therefore the aid services could use the model to aid decision making in where to deploy resources. High accuracy is important as false postives and false negatives both waste precious time and resources that could be deployed to help people in need.

5.0.3 Conclusion #3

To further improve this model's performance and utility there are several options that could be explored in future projects. First, supplemental algorithms could also be developed to indicate if a prediction falls sufficiently close to the prediction boundary and to flag it for further review by humans.

Next, additional data on latitude and longitude locations of each of the data points could also allow for algorithms to identify clusters of blue tarps in physical space. The addition of location data would allow aid services to deploy their resources even more effectively.

Finally, data pre-processing and feature engineering could be explored further to increase the true positive rate further. Upsampling methods such as SMOTE could be used to increase the prevalence of the blue tarp feature in the dataset, which could improve the model's predictive power for that class. Likewise, downsampling techniques could be used to balance the data by removing some of the NBT observations from the training dataset.

5.0.4 Conclusion #4

While training and evaluating each of these models, the run time of each algorithm was also observed. The following observations were made:

1. While KNN was the best performing model for this application, it was also the slowest to train, with a significantly longer run time than any of the other models. **I**f there were significantly more data points, it would be prudent to split the data into training and test sets, perform the model fit and cross validation on the training set, and then evaluate the final model performance using the test set.
2. The run time of the ElasticNet logistic regression model was also somewhat longer than that of the normal logistic regression model, LDA, and QDA models. For larger data sets it may also be appropriate to perform the training-test split on this model as well.
3. The run time of the Logistic Regression, LDA, and QDA models were similar and much smaller than that of the KNN or ElasticNet models. This property could make them useful in situations where aid resources had a larger data set to process and limited access to large computing resources.

5.0.5 Conclusion #5

This dataset is a very good candidate for predictive modeling tools for three reasons. First, the dataset contains only features that are important for prediction, making the task of feature selection easier. Second, each feature also has the same scaling as all of the other features (0-255 for RBG values), removing the need for some pre-processing activities. Finally, the dataset is large compared to the number of features in each of the models, which reduces the amount of variance in the final models.

6 Part II : Hold-out Data / EDA

After evaluating the cross-validation performance of the logistic regression, LDA, QDA, KNN, and Elasticnet models, additional holdout data was introduced. The data was provided in the form of 7 .txt files. To import the data the following process was run:

1. Find all text files in the current directory using the `list.files` function and regular expressions
2. Using a `dplyr` pipeline, feed the vector of text files into `set_names` to send the file names to
3. `map_df` which then combines them into a single dataframe using `read_table` which contains all rows from each text file except the first 7, which do not contain useful information
4. Create a `class` column using regular expression searches of the `FileName` column that looks for lines from files names NOT or NON_Blue_Tarps, if it is from such a file, set `class` to `NBT` , else `BT`
5. Remove unnecessary columns using `dplyr::select`

```
txt_files = list.files(pattern = '\\.txt$')
ho_df = txt_files %>%
  set_names(.) %>%
  map_df(read_table, skip = 7, .id = 'FileName')

ho_df$class = ifelse(grepl('NO[TN]_Blue_Tarps',
                           ho_df$FileName,
                           ignore.case = TRUE,
                           perl = TRUE),
                     'NBT',
                     'BT')
ho_df$class = as.factor(ho_df$class)
ho_df = ho_df %>%
  dplyr::select(-c(';','FileName'))
```

Next, the `skim` function from the `skimr` package was used to provide a concise visualization of the data.

```
skim(ho_df)
```

Data summary

Name	ho_df
Number of rows	2004177
Number of columns	11
_____	
Column type frequency:	
factor	1
numeric	10
_____	
Group variables	None

Variable type: factor

skim_variable	n_missing	complete_rate	ordered	n_unique	top_counts
class	0	1	FALSE	2	NBT: 1989697, BT: 14480

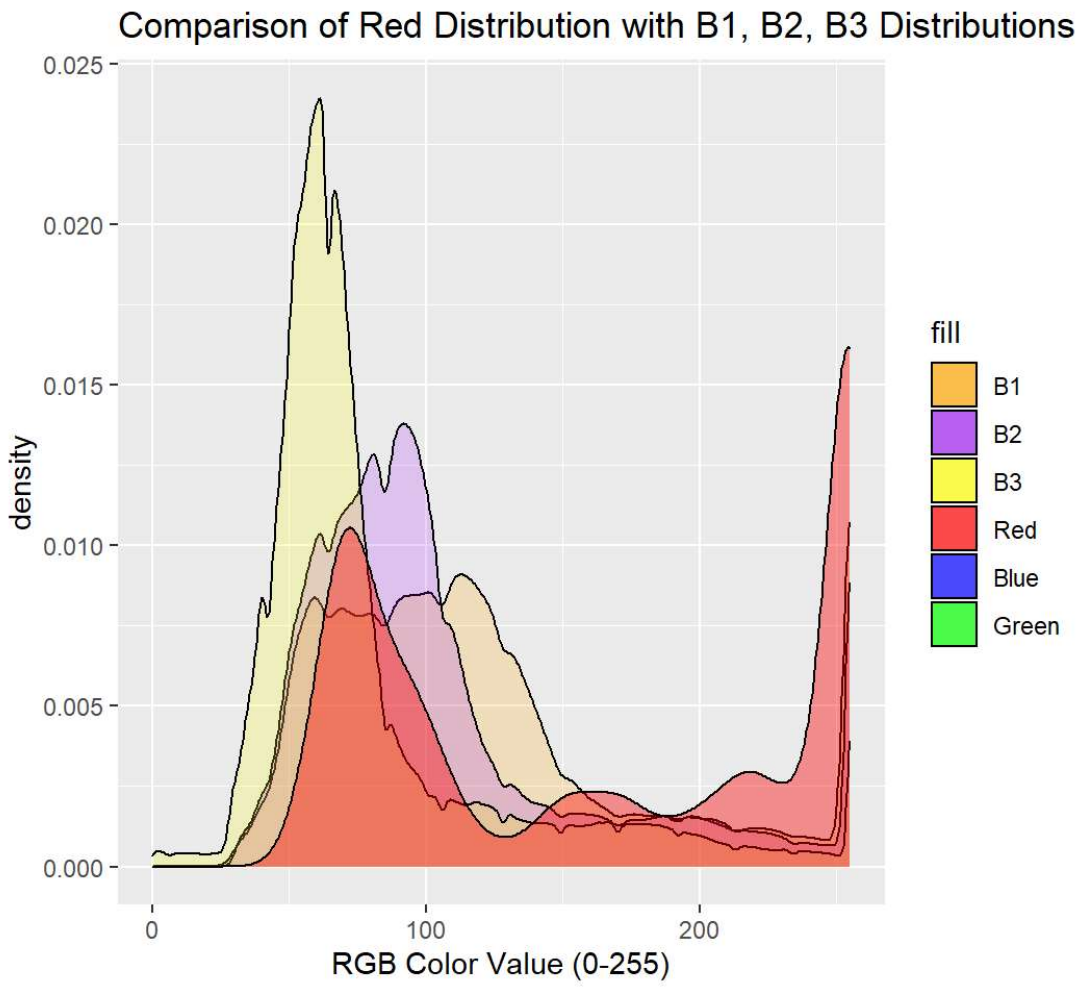
Variable type: numeric

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100	hist
ID	0	1	4967.72	2885.21	0.00	2468.00	4947.00	7464.00	9999.00	
X	0	1	2265.82	771.23	14.00	1729.00	2233.00	2746.00	4139.00	
Y	0	1	2298.44	1532.44	0.00	1451.00	1752.00	3034.00	6226.00	
Map X	0	1	771865.58	2127.09	769800.56	769879.84	772389.11	773028.22	775373.37	
Map Y	0	1	2049782.46	132.65	2049516.65	2049648.21	2049866.26	2049887.30	2050019.86	
Lat	0	1	18.52	0.00	18.52	18.52	18.52	18.52	18.52	
Lon	0	1	-72.42	0.02	-72.44	-72.44	-72.42	-72.41	-72.39	
B1	0	1	118.11	56.97	0.00	76.00	107.00	139.00	255.00	
B2	0	1	105.38	52.72	28.00	71.00	91.00	117.00	255.00	

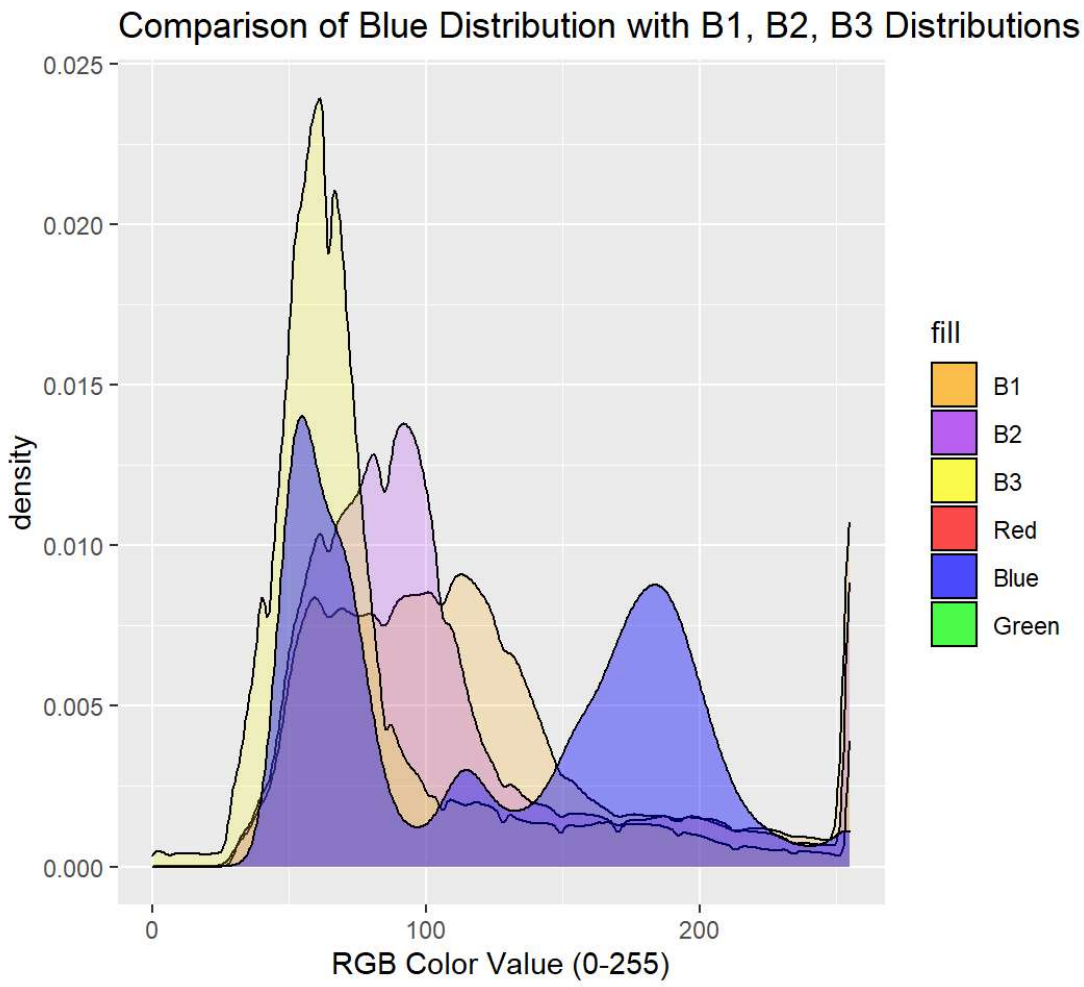
skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100	hist
B3	0	1	79.81	46.50	0.00	54.00	65.00	83.00	255.00	

The dataframe contains four variables of interest (within the scope of this project), `B1`, `B2`, `B3`, and `class`. `B1`, `B2`, and `B3` are the 0-255 values for `red`, `green`, and `blue`, and `class` is analogous to `collapse.Class` from the training data in section 1. To determine which of `B1`, `B2`, `B3` are `red`, `green`, and `blue`, density plots were created for `B1`, `B2`, `B3` and overlaid onto `red`, `green`, and `blue` from the training data:

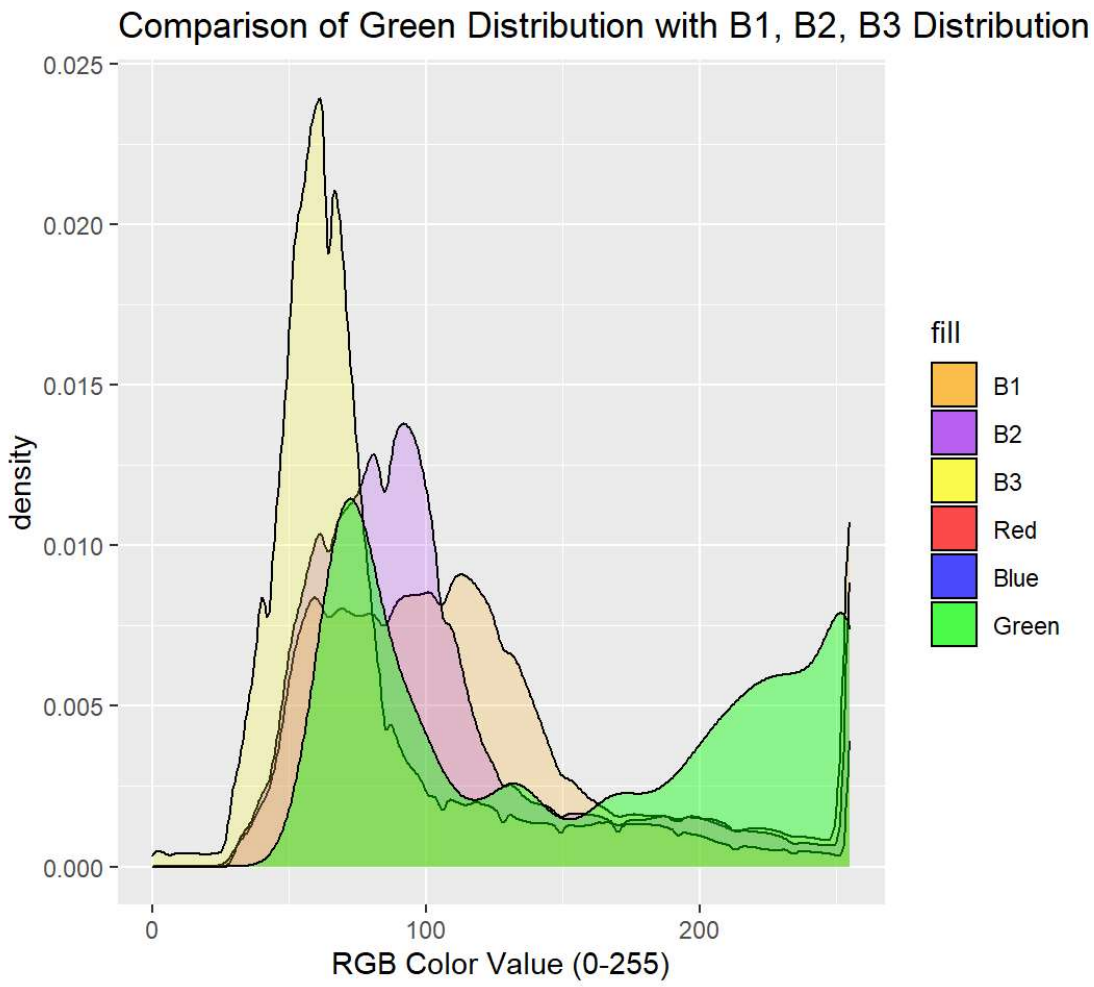
```
colors = c('B1' = 'orange',
           'B2' = 'purple',
           'B3' = 'Yellow',
           'Red' = 'red',
           'Blue' = 'blue',
           'Green' = 'green')
red_hist = ggplot() +
  geom_density(data = ho_df, aes(x = B1, fill = "B1"), alpha = 0.2) +
  geom_density(data = ho_df, aes(x = B2, fill = "B2"), alpha = 0.2) +
  geom_density(data = ho_df, aes(x = B3, fill = "B3"), alpha = 0.2) +
  geom_density(data = df, aes(x = Red, fill = 'Red'), alpha = 0.4) +
  labs(x = 'RGB Color Value (0-255)',
       title = 'Comparison of Red Distribution with B1, B2, B3 Distributions',
       color = 'Legend') +
  scale_fill_manual(values = colors)

blue_hist = ggplot() +
  geom_density(data = ho_df, aes(x = B1, fill = "B1"), alpha = 0.2) +
  geom_density(data = ho_df, aes(x = B2, fill = "B2"), alpha = 0.2) +
  geom_density(data = ho_df, aes(x = B3, fill = "B3"), alpha = 0.2) +
  geom_density(data = df, aes(x = Blue, fill = 'Blue'), alpha = 0.4) +
  labs(x = 'RGB Color Value (0-255)',
       title = 'Comparison of Blue Distribution with B1, B2, B3 Distributions',
       color = 'Legend') +
  scale_fill_manual(values = colors)

green_hist = ggplot() +
  geom_density(data = ho_df, aes(x = B1, fill = "B1"), alpha = 0.2) +
  geom_density(data = ho_df, aes(x = B2, fill = "B2"), alpha = 0.2) +
  geom_density(data = ho_df, aes(x = B3, fill = "B3"), alpha = 0.2) +
  geom_density(data = df, aes(x = Green, fill = 'Green'), alpha = 0.4) +
  labs(x = 'RGB Color Value (0-255)',
       title = 'Comparison of Green Distribution with B1, B2, B3 Distributions',
       color = 'Legend') +
  scale_fill_manual(values = colors)


red_hist
```


Comparison of Red Distribution with B1, B2, B3 Distributions
0.025
0.020
0.015
0.010
0.005
0.000
density
0
100
200
RGB Color Value (0-255)
fill
B1
B2
B3
Red
Blue
Green

```
blue_hist
```


Comparison of Blue Distribution with B1, B2, B3 Distributions
0.025
0.020
0.015
0.010
0.005
0.000
density
0
100
200
RGB Color Value (0-255)
fill
B1
B2
B3
Red
Blue
Green

```
green_hist
```



Although it is difficult to tell, it appears that the variables `B1` , `B2` , and `B3` are `Red` , `Green` , and `Blue` respectively (There are similarities in the distributions, positions, and heights of the major peaks in the histograms). This assumption was also tested when predictions were made on the holdout data, as incorrectly specified variables leads to poor holdout data prediction performance for the models, see supplementary materials for example confusion matrix test. The data frame column names are then reassigned to represent this assumption:

```
ho_df = ho_df %>%
  dplyr::select(c('B1', 'B2', 'B3', 'class'))
colnames(ho_df) = c('Red', 'Green', 'Blue', 'class')
```

In addition to their usefulness in determining the unknown variable identities, the density plot visualizations indicate that the holdout data differs from the training data, which may present some difficulties for any models that have been over-trained.

7 Part II : Model Training

In addition to the five models evaluated in Part I, two additional models, random forest and support vector machine (SVM), were trained on the training data and evaluated using cross validation in part II.

One consideration for training the support vector machine and random forest models was training time. Many of these models can take over an hour to train on such a large dataset on a laptop with 16GB of RAM. To reduce some of the training times the package `doParallel` was utilized to allow for parallel processing of model training calculations.

```
cl = makePSOCKcluster(4)
registerDoParallel(cl)
```

7.1 Random Forest

The first of the two new models to be trained is a Random Forest Model. A random forest model uses an ensemble of decision trees to determine the correct class for an observation. Random Forest models tend to be high performing classification models and therefore could be very useful in predicting the presence of blue tarps.

7.1.1 Tuning Parameters

A random forest model has two tuning parameters `mtry`, the number of variables that are randomly sampled at each split, and `ntrees`, the number of trees grown in the forest, must be selected. Due to high computation times, a loop is not constructed to test different values of `ntrees` in this report. However, several values of `ntrees` were tested during the construction of this report, and it was determined that there was no significant performance gain from increasing the number of trees beyond the default parameter $ntrees = 500$ (see Supplementary Materials for example). Therefore, for this model, `ntrees` was set to the default value $ntrees = 500$. The parameter `mtry` was then selected by cross-validation using the `train` function from the `caret` package.

```
set.seed(304)
model.rf = train(collapse.Class ~ .,
                 data = df,
                 method = 'rf',
                 trControl = ctrl,
                 allowParallel = TRUE)
```

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

```
model.rf
```

```
#> Random Forest
#>
#> 63241 samples
#>     3 predictor
#>     2 classes: 'BT', 'NBT'
#>
#> 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     0.9970905  0.9527201
#>   3     0.9968533  0.9489141
#>
#> Accuracy was used to select the optimal model using the largest value.
#> The final value used for the model was mtry = 2.
```

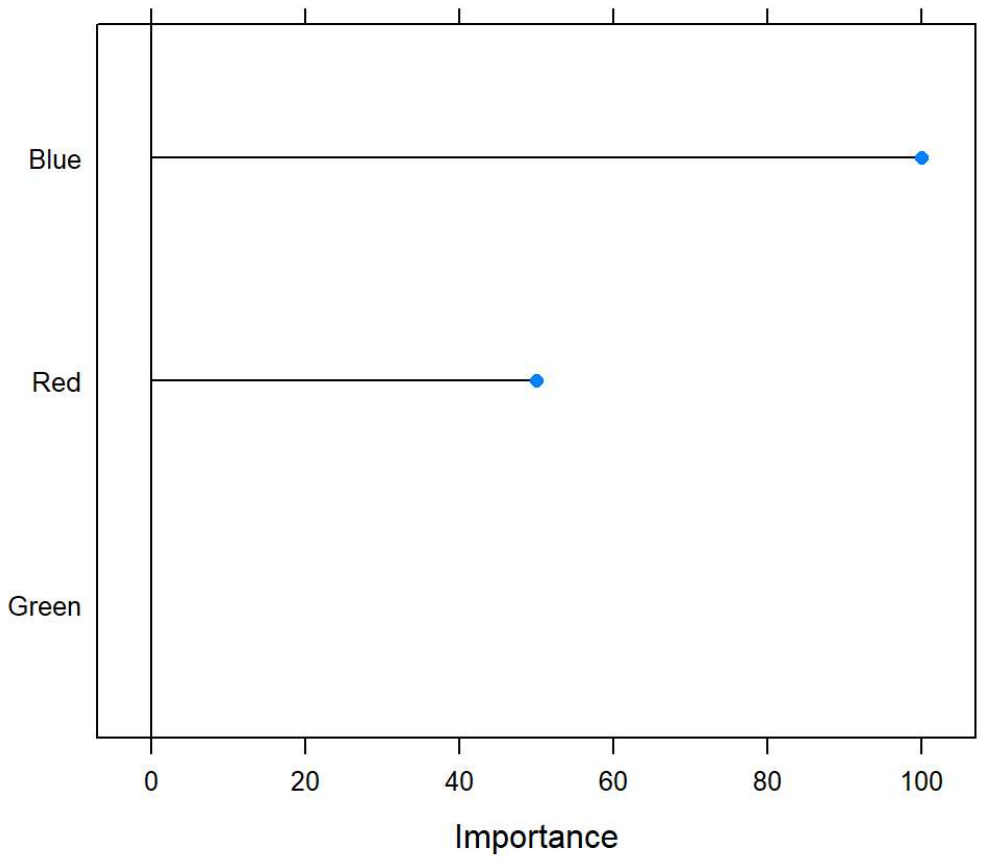
Cross-Validation selected $mtry = 2$ as the optimal tuning parameter from those tested. The cross-validation confusion matrix for the selected random forest model is shown below:

```
rf.cm = confusionMatrix(model.rf$pred$pred,model.rf$pred$obs)
rf.cm
```

```
#> Confusion Matrix and Statistics
#>
#>           Reference
#> Prediction     BT    NBT
#>        BT    3828    167
#>        NBT    216 122271
#>
#>                Accuracy : 0.997
#>                  95% CI : (0.9967, 0.9973)
#>     No Information Rate : 0.968
#>     P-Value [Acc > NIR] : < 2e-16
#>
#>                   Kappa : 0.9508
#>
#>  Mcnemar's Test P-Value : 0.01418
#>
#>             Sensitivity : 0.94659
#>             Specificity : 0.99864
#>          Pos Pred Value : 0.95820
#>          Neg Pred Value : 0.99824
#>              Prevalence : 0.03197
#>          Detection Rate : 0.03027
#>    Detection Prevalence : 0.03159
#>       Balanced Accuracy : 0.97261
#>
#>        'Positive' Class : BT
#>
```

In addition to model selection, the cross validation of the random forest model can also produce a variable importance plot which illustrates the importance of each variable in the model as determined by mean decrease in node impurity from splits based on that variable. The variable importance plot for the selected model is shown below:

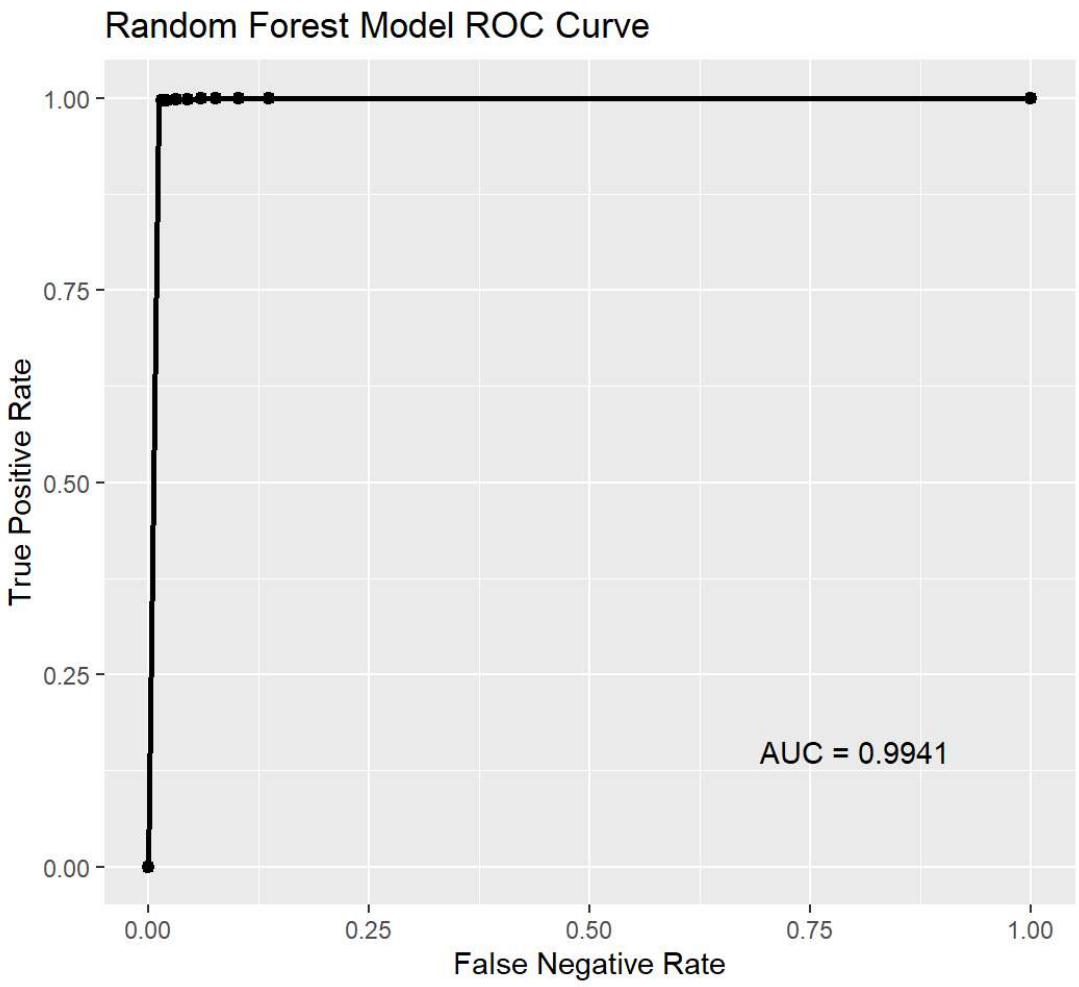
```
plot(varImp(model.rf), main = 'Variable Importance Plot for Random Forest Model')
```



The variable importance plot indicates that only two of the predictors, `Red` and `Blue` are important at predicting each observation's `class`. This is an interesting result, as many of the models tested before the random forest model have been highly effective at predicting class using all three variables. Further investigation could be conducted to see if these models' predictive power would change by not including `Green`.

After selecting the model parameters `mtry` and `ntrees` and examining the default confusion matrix, a ROC curve and associated AUC is generated:

```
rf.r = ggplot(model.rf$pred, aes(m = BT, d = obs)) +
  geom_roc(labels = FALSE, increasing = FALSE) +
  labs(title = 'Random Forest Model ROC Curve') +
  xlab('False Negative Rate') +
  ylab('True Positive Rate')
auc = round(calc_auc(rf.r)$AUC, 4)
rf.r + annotate("text", x = .8, y = .15, label = paste("AUC =", round(calc_auc(rf.r)$AUC, 4)))
```

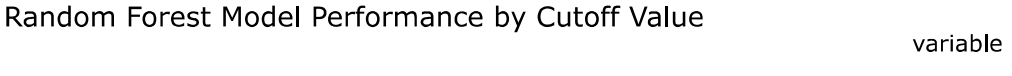


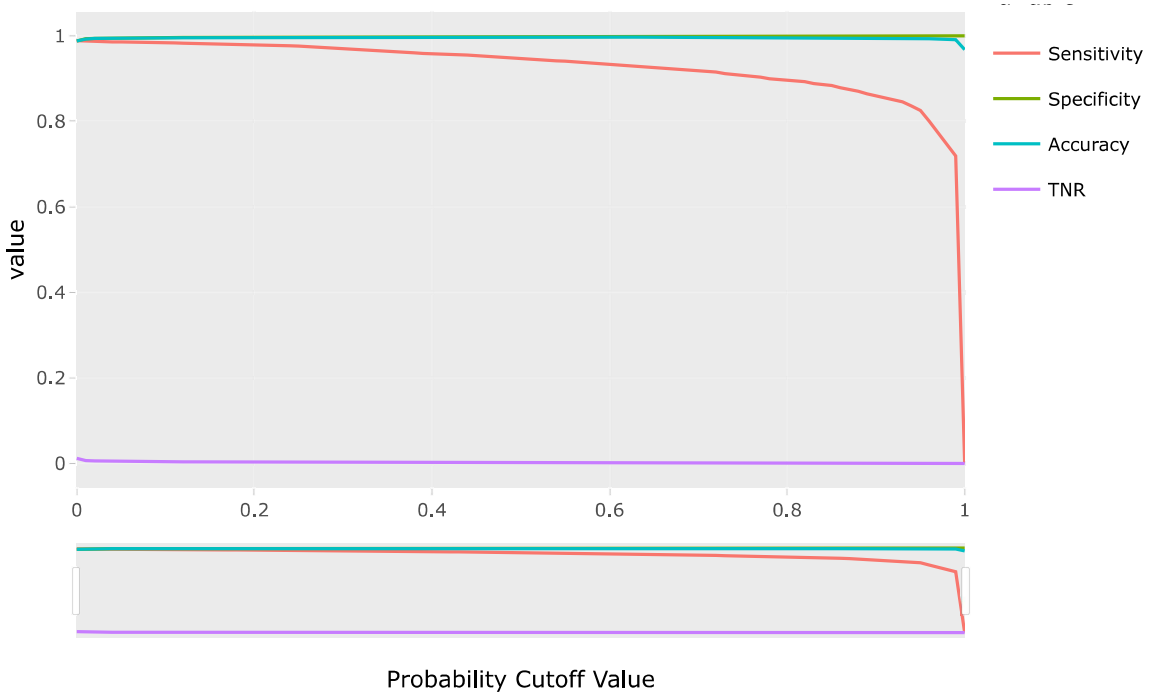
Based on the ROC curve and associated AUC, the random forest model is among the highest performing models examined.

7.1.2 Threshold Selection

To maximize model performance, an appropriate probability cutoff threshold must be selected for classifying an observation as a Blue Tarp. An interactive chart was generated using the `plotly` package to determine the threshold value:

```
sens = c()
spec = c()
acc = c()
tnr = c()
cutoff = seq(0, 1, by=0.01)
for (i in 1:length(cutoff)){
  newpred = factor(ifelse(model.rf$pred$BT > cutoff[i], "BT", "NBT"), levels = c("BT", "NBT"))
  sens[i] = sens_vec(model.rf$pred$obs, newpred)
  spec[i] = spec_vec(model.rf$pred$obs, newpred)
  acc[i] = accuracy_vec(model.rf$pred$obs, newpred)
  tnr[i] = 1 - spec[i]
}
cutoff_plot_data = data.frame(Cutoff = cutoff, Sensitivity = sens, Specificity = spec, Accuracy = acc, TNR = tn
        r)
p = ggplot(reshape2::melt(cutoff_plot_data, id.var='Cutoff')) +
  geom_line(aes(x = Cutoff, y = value, color = variable)) +
  scale_x_continuous(limits = c(0,1), breaks = seq(0,1,0.1)) +
  labs(title = 'Random Forest Model Performance by Cutoff Value') +
  xlab('Probability Cutoff Value')
ggplotly(p, dynamicTicks = TRUE) %>% rangeslider() %>% layout(hovermode = 'x')
```





The plot indicates that $cutoff = 0.05$ represents the optimal model performance. The confusion matrix at the selected cutoff value is shown below:

```
newpred = factor(ifelse(model.rf$pred$BT > 0.05, "BT", "NBT"), levels = c("BT", "NBT"))
plot_preds = cbind(plot_preds,model.knn$pred$pred)
rf.cm = confusionMatrix(newpred, model.rf$pred$obs)
rf.cm
```

```
#> Confusion Matrix and Statistics
#>
#>           Reference
#> Prediction     BT    NBT
#>        BT    3988    594
#>        NBT     56 121844
#>
#>                Accuracy : 0.9949
#>                  95% CI : (0.9945, 0.9952)
#>     No Information Rate : 0.968
#>     P-Value [Acc > NIR] : < 2.2e-16
#>
#>                   Kappa : 0.922
#>
#>  Mcnemar's Test P-Value : < 2.2e-16
#>
#>             Sensitivity : 0.98615
#>             Specificity : 0.99515
#>          Pos Pred Value : 0.87036
#>          Neg Pred Value : 0.99954
#>              Prevalence : 0.03197
#>          Detection Rate : 0.03153
#>    Detection Prevalence : 0.03623
#>       Balanced Accuracy : 0.99065
#>
#>        'Positive' Class : BT
#>
```

The confusion matrix indicates that the Random Forest model with paramters $mtry = 2$ and $n_{trees} = 500$ with $cutoff = 0.05$ is an excellent model for predicting the presence of blue tarps given a `Red`, `Green`, and `Blue` value. The model is over 99% accurate overall and also has a very high sensitivity (over 98%), which is important due to the imbalanced data set. Therefore, the random forest model is an excellent candidate model for use in production according to cross-validation.

A table of the Random Forest model's results can be found below:

```
rf.results = data.frame(Model = 'Random Forest', Tuning = c('mtry = 2,ntrees = 500'), AUROC = round(calc_auc(r
        f.r)$AUC, 4), Threshold = 0.05, Accuracy = round(rf.cm$overall[[1]],4), TPR = round(rf.cm$byClass[[1
        ]],4), FNR = round(1-rf.cm$byClass[[2]],4), Precision = round(rf.cm$byClass[[5]],4))
knitr::kable(rf.results)
```

Model	Tuning	AUROC	Threshold	Accuracy	TPR	FNR	Precision
Random Forest	mtry = 2,ntrees = 500	0.9941	0.05	0.9949	0.9862	0.0049	0.8704

7.2 Support Vector Machine (SVM)

The final model to be tested is a support vector machine model. These models use a kernel function to project the data into a higher-dimensional space, where accurate hyper-plane separations of the data may be possible.

7.2.1 Tuning Parameters

To create a support vector machine model, three parameters must be chosen. The first parameter, the type of kernel used to create the model, determines which function is used to transform the data into a higher-dimensional space where it may be separated with greater ease than in the original space. Three kernels were examined for this problem:

1. Linear Kernel (Support Vector Classifier)
2. Radial Kernel
3. Polynomial Kernel

In addition to the type of kernel used, the tuning parameters C (the error margin for the support vectors) and σ (the amount of curvature in the decision boundaries) must be determined using cross-validation.

The `train` method of the `caret` package was used to train models for each of the three kernels and to select values of σ (sigma) and C for each of the kernel models (only C must be selected in the case of the linear kernel):

```
model.svm.linear = train(collapse.Class ~ .,
                         data = df,
                         method = 'svmLinear',
                         trControl = ctrl,
                         tuneLength = 4,
                         allowParallel = TRUE)
model.svm.linear
```

```
#> Support Vector Machines with Linear Kernel
#>
#> 63241 samples
#>     3 predictor
#>     2 classes: 'BT', 'NBT'
#>
#> No pre-processing
#> Resampling: Cross-Validated (10 fold)
#> Summary of sample sizes: 56917, 56917, 56916, 56917, 56918, 56917, ...
#> Resampling results:
#>
#>   Accuracy   Kappa
#>   0.9953827  0.9223697
#>
#> Tuning parameter 'C' was held constant at a value of 1
```

The selected linear model has very good cross validation accuracy (over 99%). Cross-validation selcted the tuning parameter $C = 1$ as the best performing tune for this model.

```
model.svm.radial.tg = expand.grid(C = c(1, 4, 8, 16, 32),
                                  sigma = c(1,5,10))
model.svm.radial = train(collapse.Class ~ .,
                         data = df,
                         method = 'svmRadial',
                         trControl = ctrl,
                         tuneLength = 8,
                         tuneGrid = model.svm.radial.tg,
                         allowParallel = TRUE)
model.svm.radial
```

```
#> Support Vector Machines with Radial Basis Function Kernel
#>
#> 63241 samples
#>     3 predictor
#>     2 classes: 'BT', 'NBT'
#>
#> No pre-processing
#> Resampling: Cross-Validated (10 fold)
#> Summary of sample sizes: 56917, 56917, 56916, 56917, 56917, 56917, ...
#> Resampling results across tuning parameters:
#>
#>   C   sigma  Accuracy   Kappa
#>    1   1     0.9963473  0.9404414
#>    1   5     0.9968059  0.9474774
#>    1  10     0.9969956  0.9508147
#>    4   1     0.9966161  0.9447398
#>    4   5     0.9969798  0.9505496
#>    4  10     0.9970589  0.9518081
#>    8   1     0.9967426  0.9468715
#>    8   5     0.9970905  0.9525361
#>    8  10     0.9971221  0.9528663
#>   16   1     0.9969166  0.9497220
#>   16   5     0.9971379  0.9534583
#>   16  10     0.9972328  0.9547782
#>   32   1     0.9969008  0.9495269
#>   32   5     0.9973435  0.9570087
#>   32  10     0.9973593  0.9569741
#>
#> Accuracy was used to select the optimal model using the largest value.
#> The final values used for the model were sigma = 10 and C = 32.
```

The radial kenerl SVM model has even better cross validation accuracy than the linear model, with 99.7% accuracy, indicating that it may be a stronger model. Cross-validation selected the tuning parameters $sigma = 10$ and $C = 32$ as the best performing tune for this model.

```
model.svm.poly = train(collapse.Class ~ .,
                       data = df,
                       method = 'svmPoly',
                       trControl = ctrl,
                       tuneLength = 4,
                       allowParallel = TRUE)
model.svm.poly
```

```
#> Support Vector Machines with Polynomial Kernel
#>
#> 63241 samples
#>     3 predictor
#>     2 classes: 'BT', 'NBT'
#>
#> No pre-processing
#> Resampling: Cross-Validated (10 fold)
#> Summary of sample sizes: 56917, 56917, 56917, 56917, 56917, 56916, ...
#> Resampling results across tuning parameters:
#>
#>   degree  scale  C     Accuracy   Kappa
#>   1       0.001  0.25        NaN        NaN
#>   1       0.001  0.50        NaN        NaN
#>   1       0.001  1.00  0.9882355  0.8216255
#>   1       0.001  2.00  0.9882829  0.8221081
#>   1       0.010  0.25  0.9887573  0.8263776
#>   1       0.010  0.50  0.9894689  0.8325940
#>   1       0.010  1.00  0.9917934  0.8661770
#>   1       0.010  2.00  0.9949084  0.9125656
#>   1       0.100  0.25  0.9949874  0.9140806
#>   1       0.100  0.50  0.9951139  0.9165088
#>   1       0.100  1.00  0.9951930  0.9184521
#>   1       0.100  2.00  0.9952879  0.9204874
#>   1       1.000  0.25  0.9953195  0.9211519
#>   1       1.000  0.50  0.9954144  0.9228499
#>   1       1.000  1.00  0.9953827  0.9223477
#>   1       1.000  2.00  0.9953827  0.9223559
#>   2       0.001  0.25        NaN        NaN
#>   2       0.001  0.50  0.9883462  0.8236764
#>   2       0.001  1.00  0.9883936  0.8239193
#>   2       0.001  2.00  0.9895954  0.8358580
#>   2       0.010  0.25  0.9950823  0.9167473
#>   2       0.010  0.50  0.9954460  0.9245793
#>   2       0.010  1.00  0.9959520  0.9339484
#>   2       0.010  2.00  0.9958255  0.9316166
#>   2       0.100  0.25  0.9957939  0.9310820
#>   2       0.100  0.50  0.9958888  0.9327680
#>   2       0.100  1.00  0.9959204  0.9332587
#>   2       0.100  2.00  0.9958730  0.9323586
#>   2       1.000  0.25  0.9956832  0.9288474
#>   2       1.000  0.50  0.9957464  0.9297906
#>   2       1.000  1.00  0.9957148  0.9290913
#>   2       1.000  2.00  0.9957306  0.9293409
#>   3       0.001  0.25        NaN        NaN
#>   3       0.001  0.50  0.9885043  0.8261978
#>   3       0.001  1.00  0.9894531  0.8349382
#>   3       0.001  2.00  0.9948767  0.9120536
#>   3       0.010  0.25  0.9953669  0.9227827
#>   3       0.010  0.50  0.9959046  0.9332754
#>   3       0.010  1.00  0.9958097  0.9314475
#>   3       0.010  2.00  0.9957781  0.9308048
#>   3       0.100  0.25  0.9959046  0.9334267
#>   3       0.100  0.50  0.9959362  0.9339233
#>   3       0.100  1.00  0.9959520  0.9341493
#>   3       0.100  2.00  0.9960153  0.9347560
#>   3       1.000  0.25  0.9962841  0.9390584
#>   3       1.000  0.50  0.9963473  0.9401576
#>   3       1.000  1.00  0.9963473  0.9401793
#>   3       1.000  2.00  0.9964580  0.9419290
#>
```

```
#> 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.
```

```
stopCluster(cl)
```

```
#> Error in summary.connection(connection): invalid connection
```

The polynomial kernel also has very high cross-validation accuracy, at over 99.6%. Cross-validation selected the tuning parameters $sigma = 3$ and $C = 1$ as the best performing tune for this model.

Despite the addition of parallel processing, many of the training times for each of these models exceeded an hour. Due to the long computation time it became difficult to conduct a more exhaustive search of the parameter space for more ideal models. However, each of the three kernels produced a model with greater than 99% accuracy.

To compare the performance of the three kernels, a table of the results was produced and can be seen below:

```
svm.linear.cv.results = confusionMatrix(model.svm.linear$pred$pred,
                                        model.svm.linear$pred$obs)

svm.radial.cv.results = confusionMatrix(model.svm.radial$pred$pred,
                                        model.svm.radial$pred$obs)

svm.poly.cv.results = confusionMatrix(model.svm.poly$pred$pred,
                                      model.svm.poly$pred$obs)

models = list(model.svm.linear, model.svm.radial, model.svm.poly)

model_types = c('Linear', 'Radial', 'Polynomial')
model_accs = c(svm.linear.cv.results$overall[[1]],
               svm.radial.cv.results$overall[[1]],
               svm.poly.cv.results$overall[[1]])
model_sens = c(svm.linear.cv.results$byClass[[1]],
               svm.radial.cv.results$byClass[[1]],
               svm.poly.cv.results$byClass[[1]])
model_spec = c(svm.linear.cv.results$byClass[[2]],
               svm.radial.cv.results$byClass[[2]],
               svm.poly.cv.results$byClass[[2]])
svm.kernel.table = data.frame(KernelType = model_types,
                              ModelAccuracy = model_accs,
                              ModelSensitivity = model_sens,
                              ModelSpecificity = model_spec)
knitr::kable(svm.kernel.table)
```

KernelType	ModelAccuracy	ModelSensitivity	ModelSpecificity
Linear	0.9953827	0.8877349	0.9989382
Radial	0.9969766	0.9420706	0.9987901
Polynomial	0.9941695	0.8985366	0.9973281

All three SVM kernels produced very high-performing models.However, based on the results in the table above, the best performing kernel is the radial kernel due to the higher sensitivity performance. Therefore, the radial kernel with tuning parameters $sigma = 10$ and $C = 32$ was selected as the candidate support vector machine model.

```
model.svm = model.svm.radial
```

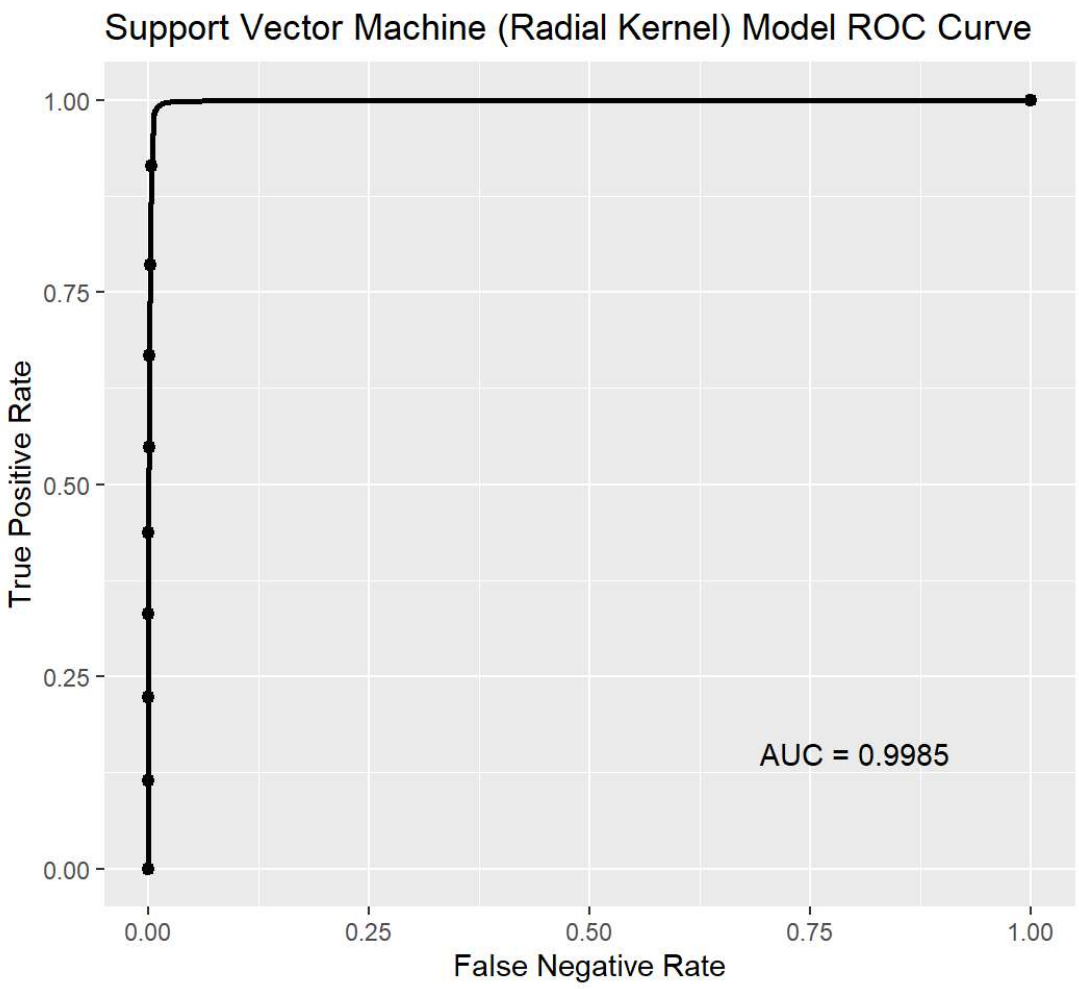
The default confusion matrix for the SVM model (using $cutoff = 0.5$) is shown below:

```
svm.cm = confusionMatrix(model.svm$pred$pred,model.svm$pred$obs)
svm.cm
```

```
#> Confusion Matrix and Statistics
#>
#>           Reference
#> Prediction     BT    NBT
#>        BT   28573   1111
#>        NBT   1757 917174
#>
#>                Accuracy : 0.997
#>                  95% CI : (0.9969, 0.9971)
#>     No Information Rate : 0.968
#>     P-Value [Acc > NIR] : < 2.2e-16
#>
#>                   Kappa : 0.9507
#>
#>  Mcnemar's Test P-Value : < 2.2e-16
#>
#>             Sensitivity : 0.94207
#>             Specificity : 0.99879
#>          Pos Pred Value : 0.96257
#>          Neg Pred Value : 0.99809
#>              Prevalence : 0.03197
#>          Detection Rate : 0.03012
#>    Detection Prevalence : 0.03129
#>       Balanced Accuracy : 0.97043
#>
#>        'Positive' Class : BT
#>
```

The selected model's ROC curve and AUC is shown below:

```
svm.r = ggplot(model.svm$pred, aes(m = BT, d = obs)) +
  geom_roc(labels = FALSE, increasing = FALSE) +
  labs(title = 'Support Vector Machine (Radial Kernel) Model ROC Curve') +
  xlab('False Negative Rate') +
  ylab('True Positive Rate')
auc = round(calc_auc(svm.r)$AUC, 4)
svm.r + annotate("text", x = .8, y = .15, label = paste("AUC =", round(calc_auc(svm.r)$AUC, 4)))
```

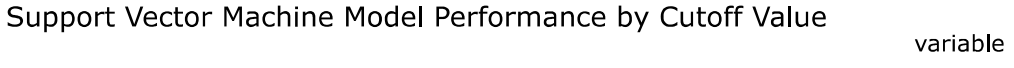


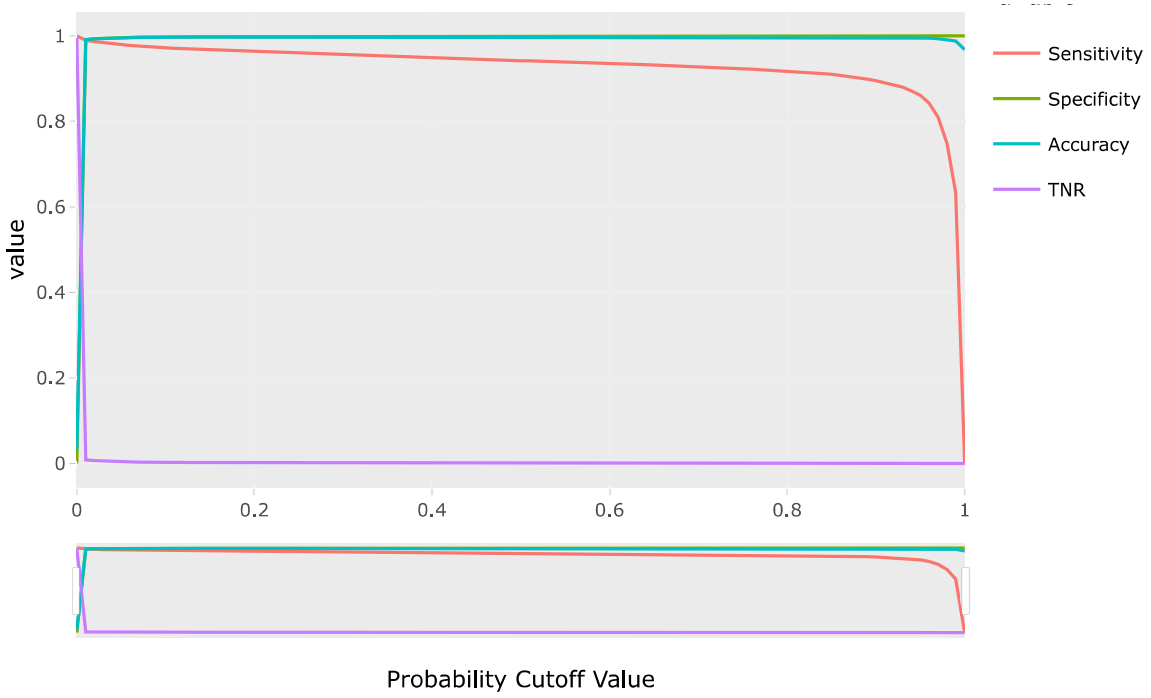
The radial kernel SVM AUC is among the highest AUCs of all models investigated in this report at over 0.998.

7.2.2 Threshold Selection

Next, the cutoff value for classification must be selected for the selected SVM model. To investigate the model's performance across a range of cutoff values the following interactive plot was created:

```
sens = c()
spec = c()
acc = c()
tnr = c()
cutoff = seq(0, 1, by=0.01)
for (i in 1:length(cutoff)){
  newpred = factor(ifelse(model.svm$pred$BT > cutoff[i], "BT", "NBT"), levels = c("BT", "NBT"))
  sens[i] = sens_vec(model.svm$pred$obs, newpred)
  spec[i] = spec_vec(model.svm$pred$obs, newpred)
  acc[i] = accuracy_vec(model.svm$pred$obs, newpred)
  tnr[i] = 1 - spec[i]
}
cutoff_plot_data = data.frame(Cutoff = cutoff, Sensitivity = sens, Specificity = spec, Accuracy = acc, TNR = tn
        r)
p = ggplot(reshape2::melt(cutoff_plot_data, id.var='Cutoff')) +
  geom_line(aes(x = Cutoff, y = value, color = variable)) +
  scale_x_continuous(limits = c(0,1), breaks = seq(0,1,0.1)) +
  labs(title = 'Support Vector Machine Model Performance by Cutoff Value') +
  xlab('Probability Cutoff Value')
ggplotly(p, dynamicTicks = TRUE) %>% rangeslider() %>% layout(hovermode = 'x')
```





Based on the interactive plot, it appears that the optimal model performance is achieved with a cutoff value of 0.1. The associated confusion matrix is shown below:

```
newpred = factor(ifelse(model.svm$pred$BT > 0.1, "BT", "NBT"), levels = c("BT", "NBT"))
svm.cm = confusionMatrix(newpred, model.svm$pred$obs)
svm.cm
```

```
#> Confusion Matrix and Statistics
#>
#>           Reference
#> Prediction     BT    NBT
#>        BT   29481   2094
#>        NBT    849 916191
#>
#>                Accuracy : 0.9969
#>                  95% CI : (0.9968, 0.997)
#>     No Information Rate : 0.968
#>     P-Value [Acc > NIR] : < 2.2e-16
#>
#>                   Kappa : 0.9509
#>
#>  Mcnemar's Test P-Value : < 2.2e-16
#>
#>             Sensitivity : 0.97201
#>             Specificity : 0.99772
#>          Pos Pred Value : 0.93368
#>          Neg Pred Value : 0.99907
#>              Prevalence : 0.03197
#>          Detection Rate : 0.03108
#>    Detection Prevalence : 0.03329
#>       Balanced Accuracy : 0.98486
#>
#>        'Positive' Class : BT
#>
```

The confusion matrix at the chosen cutoff value indicates that the radial kernel support vector machine is an excellent model when evaluated using cross-validation on the training data. The model performs very well in accuracy, sensitivity, and specificity. The support vector machine model is an excellent choice for a model to evaluate the presence of blue tarps.

The cross-validation performance of the support vector machine model at the given cutoff value is shown in the table below.

```
sig = as.character(model.svm.radial$bestTune[[1]])
Cparam = as.character(model.svm.radial$bestTune[[2]])
svm.results = data.frame(Model = 'SVM Radial Kernel', Tuning = c(sprintf('sigma = %s, C = %s', sig, Cparam)), A
        UROC = round(calc_auc(svm.r)$AUC, 4), Threshold = 0.1, Accuracy = round(svm.cm$overall[[1]],4), TPR =
        round(svm.cm$byClass[[1]],4), FNR = round(1-svm.cm$byClass[[2]],4), Precision = round(svm.cm$byClass[[
        5]],4))
knitr::kable(svm.results)
```

Model	Tuning	AUROC	Threshold	Accuracy	TPR	FNR	Precision
SVM Radial Kernel	sigma = 10, C = 32	0.9985	0.1	0.9969	0.972	0.0023	0.9337

8 Part II: Final Results (Cross-Validation)

```
results.cv = rbind(logistic.results, lda.results, qda.results, knn.results, elnet.results, rf.results, svm.resu
        lts)
knitr::kable(results.cv,
             caption = 'Model Cross-Validation Performance')
```

Model Cross-Validation Performance

Model	Tuning	AUROC	Threshold	Accuracy	TPR	FNR	Precision
Logistic Regression	NA	0.9985	0.1	0.9908	0.9654	0.0083	0.7929
LDA	NA	0.9888	0.1	0.9811	0.8600	0.0149	0.6562
QDA	NA	0.9982	0.1	0.9896	0.8991	0.0074	0.7995
KNN	k=5	NA	NA	0.9971	0.9567	0.0015	0.9536
Elasticnet	alpha = 1, lambda = 8.32e-0.5	0.9919	0.1	0.9551	0.9394	0.0444	0.4115
Random Forest	mtry = 2,ntrees = 500	0.9941	0.05	0.9949	0.9862	0.0049	0.8704
SVM Radial Kernel	sigma = 10, C = 32	0.9985	0.1	0.9969	0.9720	0.0023	0.9337

9 Part II: Final Results (Hold-Out)

```
ho.probs.logistic = predict(model.logistic,
                            newdata = ho_df[,1:3],
                            type = 'prob')
ho.preds.logistic = as.factor(ifelse(ho.probs.logistic$BT > 0.1, 'BT', 'NBT'))
ho.cm.logistic = confusionMatrix(ho.preds.logistic,
                                 ho_df$class)
ho.rocr_prediction.logistic = prediction(predictions = ho.probs.logistic$BT,
                                         labels = ho_df$class)
ho.auc.logistic.obj =  performance(ho.rocr_prediction.logistic,
                                   measure = 'auc')
ho.auc.logistic = round(1 - ho.auc.logistic.obj@y.values[[1]],4)

ho.probs.lda = predict(model.lda, newdata = ho_df[,1:3], type = 'prob')
ho.preds.lda = as.factor(ifelse(ho.probs.lda$BT > 0.1, 'BT', 'NBT'))
ho.cm.lda = confusionMatrix(ho.preds.lda,
                            ho_df$class)
ho.rocr_prediction.lda = prediction(predictions = ho.probs.lda$BT,
                                    labels = ho_df$class)
ho.auc.lda.obj =  performance(ho.rocr_prediction.lda,
                              measure = 'auc')
ho.auc.lda = round(1 - ho.auc.lda.obj@y.values[[1]],4)

ho.probs.qda = predict(model.qda, newdata = ho_df[,1:3], type = 'prob')
ho.preds.qda = as.factor(ifelse(ho.probs.qda$BT > 0.1, 'BT', 'NBT'))
ho.cm.qda = confusionMatrix(ho.preds.qda,
                            ho_df$class)
ho.rocr_prediction.qda = prediction(predictions = ho.probs.qda$BT,
                                    labels = ho_df$class)
ho.auc.qda.obj =  performance(ho.rocr_prediction.qda,
                              measure = 'auc')
ho.auc.qda = round(1 - ho.auc.qda.obj@y.values[[1]],4)

ho.preds.knn = predict(model.knn, newdata = ho_df[,1:3])
ho.cm.knn = confusionMatrix(ho.preds.knn,
                            ho_df$class)

ho.probs.elnet = predict(model.elnet, newdata = ho_df[,1:3], type = 'prob')
ho.preds.elnet = as.factor(ifelse(ho.probs.elnet$BT > 0.1, 'BT', 'NBT'))
ho.cm.elnet = confusionMatrix(ho.preds.elnet,
                              ho_df$class)
ho.rocr_prediction.elnet = prediction(predictions = ho.probs.elnet$BT,
                                      labels = ho_df$class)
ho.auc.elnet.obj =  performance(ho.rocr_prediction.elnet,
                                measure = 'auc')
ho.auc.elnet = round(1 - ho.auc.elnet.obj@y.values[[1]],4)

ho.probs.rf = predict(model.rf, newdata = ho_df[,1:3], type = 'prob')
ho.preds.rf = as.factor(ifelse(ho.probs.rf$BT > 0.05, 'BT', 'NBT'))
ho.cm.rf = confusionMatrix(ho.preds.rf,
                           ho_df$class)
ho.rocr_prediction.rf = prediction(predictions = ho.probs.rf$BT,
                                   labels = ho_df$class)
ho.auc.rf.obj =  performance(ho.rocr_prediction.rf,
                             measure = 'auc')
ho.auc.rf = round(1 - ho.auc.rf.obj@y.values[[1]],4)

ho.probs.svm = predict(model.svm, newdata = ho_df[,1:3], type = 'prob')
ho.preds.svm = as.factor(ifelse(ho.probs.svm$BT > 0.1, 'BT', 'NBT'))
ho.cm.svm = confusionMatrix(ho.preds.svm,
                            ho_df$class)
ho.rocr_prediction.svm = prediction(predictions = ho.probs.svm$BT,
                                    labels = ho_df$class)
```

```
ho.auc.svm.obj =  performance(ho.rocr_prediction.svm,
                                  measure = 'auc')
ho.auc.svm = round(1 - ho.auc.svm.obj@y.values[[1]],4)

ho_cms = list(ho.cm.logistic, ho.cm.lda, ho.cm.qda,
              ho.cm.knn, ho.cm.elnet, ho.cm.rf, ho.cm.svm)
model_names = c('Logistic Regression',
                'LDA',
                'QDA',
                'KNN',
                'Elasticnet',
                'Random Forest',
                'SVM Radial Kernel')
cutoffs = c('0.1', '0.1', '0.1', 'NA', '0.1', '0.05', '0.1')
tuning_params = c('NA', 'NA', 'NA',
                  'K = 5', 'alpha = 1, lambda = 8.32e-0.5',
                  'mtry = 2, ntrees = 500','sigma = 10, C = 32')
aucs = c(ho.auc.logistic, ho.auc.lda, ho.auc.qda,
         'N/A', ho.auc.elnet, ho.auc.rf, ho.auc.svm)
accs = c()
sens = c()
specs = c()
precs = c()
for (i in 1:length(ho_cms)){
  cur_model = ho_cms[[i]]
  accs = append(accs, round(cur_model$overall[[1]],4))
  sens = append(sens, round(cur_model$byClass[[1]],4))
  specs = append(specs, round(1 - cur_model$byClass[[2]], 4))
  precs = append(precs, round(cur_model$byClass[[5]],4))
}
ho_table_data = data.frame(model_names, tuning_params, cutoffs, aucs,
                           accs, sens, specs, precs)
colnames(ho_table_data) = c('Model Name', 'Tuning Parameter(s)',
                            'Cutoff Value', 'AUROC',
                            'Accuracy',
                            'TPR',
                            'FNR',
                            'Precision')
knitr::kable(ho_table_data,
             caption = 'Model Holdout Data Performance')
```

Model Holdout Data Performance

Model Name	Tuning Parameter(s)	Cutoff Value	AUROC	Accuracy	TPR	FNR	Precision
Logistic Regression	NA	0.1	0.9996	0.9283	1.0000	0.0722	0.0916
LDA	NA	0.1	0.9931	0.9784	0.9239	0.0212	0.2407
QDA	NA	0.1	0.7877	0.9754	0.6209	0.0220	0.1706
KNN	K = 5	NA	N/A	0.9923	0.8217	0.0064	0.4819
Elasticnet	alpha = 1, lambda = 8.32e-0.5	0.1	0.9998	0.9376	1.0000	0.0628	0.1038
Random Forest	mtry = 2, ntrees = 500	0.05	0.9802	0.9806	0.9571	0.0192	0.2663
SVM Radial Kernel	sigma = 10, C = 32	0.1	0.9307	0.9888	0.4291	0.0072	0.3037

The table above illustrates each model's predictive performance on the 2 million row holdout dataset. No cutoff value or AUROC is provided for the KNN model as a cutoff value was not selected in part **I** because the decision was made to have the majority class of each observations K-Nearest Neighbors determine the observation's class.

10 Part II : Final Conclusions

10.0.1 Conclusion #1

The best performing model for the cross-validation data is the Radial Kernel Support Vector Machine. This model has an AUC of over 0.998 and a true positive rate of over 98% at the selected cutoff value of 0.1. This strong performance indicates that this model is very effective at learning the structure and patterns of the data within the training set. The SVM model has slightly better cross-validation performance than the random forest or k-nearest-neighbors models. The SVM model was chosen over these other models because of its higher true positive rate, indicating that this model's predictions of the presence of blue tarps would be more accurate than the random forest or KNN models when evaluated on the training set.

10.0.2 Conclusion #2

The best performing model on the holdout data is the Random Forest model. While the random forest model did not perform as well on the training data as the radial kernel SVM (94.66% TPR v 98% TPR), it significantly outperformed the SVM on the holdout data. The random forest model with $mtry = 2$ and $ntrees = 500$ avoided any potential over-training and achieved over 95% sensitivity and specificity when evaluated on the holdout data. In addition to the random forest model, the KNN and LDA models also performed very well on the holdout data. The random forest model was chosen over these models because despite having lower specificity, its loss in specificity is made up by its large increase in sensitivity. As the objective of the model is to predict the presence of blue tarps accurately, a 1% decrease in true negative rate compared the KNN and LDA models is more than compensated by an increase in sensitivity of 13% and 3% over the KNN and LDA models respectively.

10.0.3 Conclusion #3

As stated in conclusion 1, the SVM model is the best performing model on the cross-validation data. However, this model does not perform as well on the holdout data, where its performance suffers significantly. Despite its accuracy only dropping to 98%, the model's sensitivity, or TPR, drops from over 98% to under 50%, indicating that this model was significantly over-trained on the cross-validation data. This level of over-training did not allow the model to correctly interpret the holdout data, which had different distributions than the training data as observed in the exploratory data analysis section. The poor performance of the model on the holdout data removes it from contention to be put into production as the best overall model.

Therefore, the best performing model overall is the random forest model, which performed very well on both the training and holdout datasets. While this model did not achieve as high of a true positive rate on the training data as the radial kernel SVM, it did not over-train on the cross-validation data, leading to robust performance on the holdout dataset.

10.0.4 Conclusion #4

The Random Forest Model is also a good choice for a production model due to its relatively fast prediction time. Some of the other top performing models on the holdout data, notably the KNN model, were slow at predicting the large holdout dataset, which contained over 2 million rows. As each row contained a pixel, and high resolution photos can contain millions to billions of pixels, the chosen model must also have quick prediction performance to process new data quickly so that aid can be deployed.

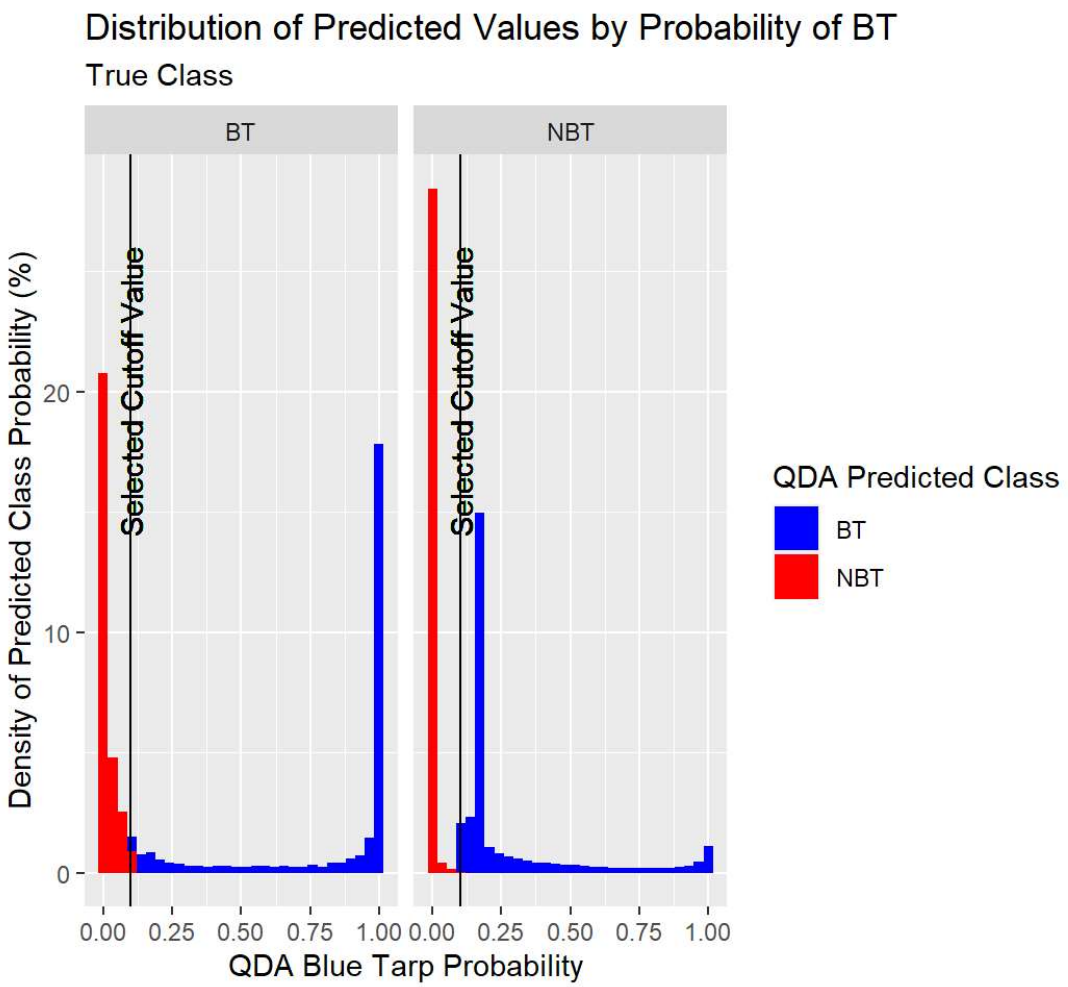
10.0.5 Conclusion #5

Many of the models trained on the cross-validation training data performed poorly on the holdout data, most notably the SVM and QDA models. This performance can be attributed to two issues: over-training and low cutoff values.

When examined during the holdout data exploratory data analysis, the holdout data appears to have much different distributions than the training data. Therefore, any models that are over-fitted on the training data may fail to correctly predict observations in the holdout data. In addition to over-training, the extremely low cutoff values selected during model training may have also contributed to the poor model performance because the low cutoff value allows the model to classify points which have very low probability of being a blue tarp as blue tarp points.

For example, this effect is illustrated in the following visualization of the QDA model predictions:

```
hist_df = data.frame(ho.probs.qda$BT, ho.preds.qda, ho_df$class)
ggplot(hist_df) +
  geom_histogram(aes(x = ho.probs.qda.BT,
                     y = ..density..,
                     fill = ho.preds.qda )) +
  geom_vline(xintercept = 0.1) +
  annotate('text', x = 0.1, y = 20, label = 'Selected Cutoff Value', angle = 90,
           text=element_text(size=11)) +
  facet_grid(~ho_df.class) +
  scale_fill_manual(values=c("blue", "red")) +
  labs(title = 'Distribution of Predicted Values by Probability of BT',
       subtitle = 'True Class') +
  xlab('QDA Blue Tarp Probability') +
  ylab('Density of Predicted Class Probability (%)') +
  guides(fill=guide_legend(title="QDA Predicted Class"))
```

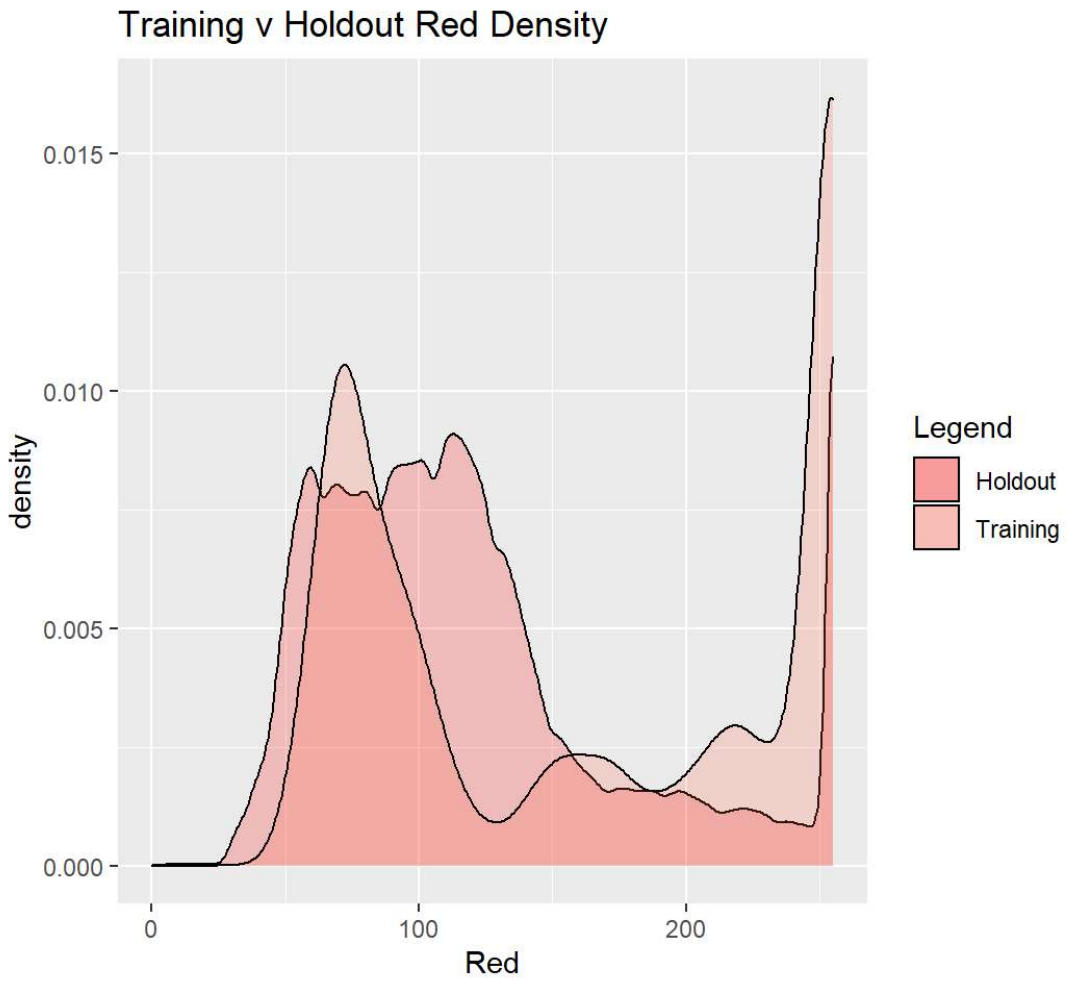


From the plot, it can be seen that if the cutoff has been raised to 0.2, the QDA accuracy would increase because the model would then correctly classify the large peak in the NBT class properly while only incorrectly classifying a small amount of BT class. However, a large part of the model's sensitivity loss is due to the large peak at the left hand side of the BT plot, which would not be correctly classified without drastically reducing the cutoff value and thereby reducing the model's overall performance greatly. The presence of the large peak at the left side of the plot is likely due to over-training of the model. This visualization is also reflected in the QDA model's low holdout AUC (0.7877).

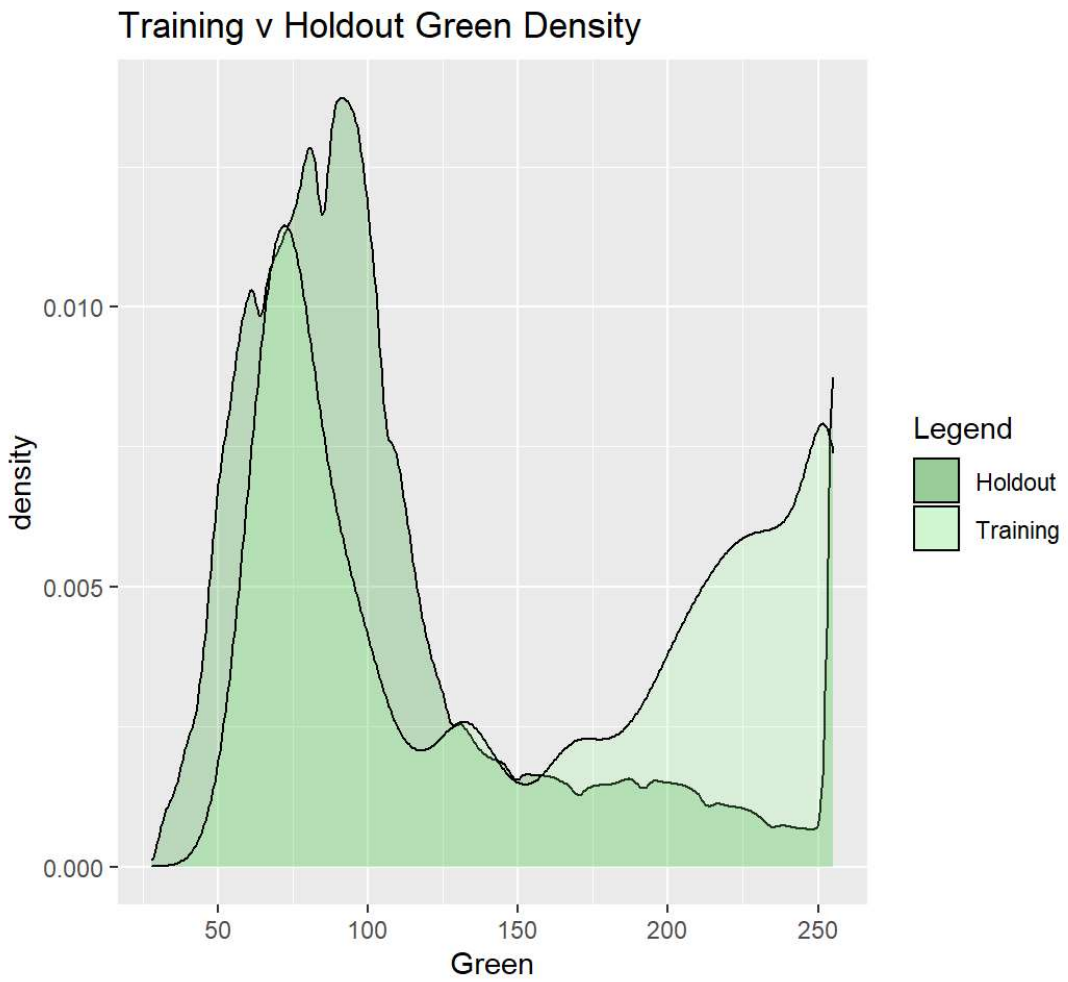
10.0.6 Conclusion #6

As observed in Part II exploratory data analysis, the training data and holdout data had some significant differences in their respective `Red`, `Green`, and `Blue` distributions. This effect is visualized in the density plots below:

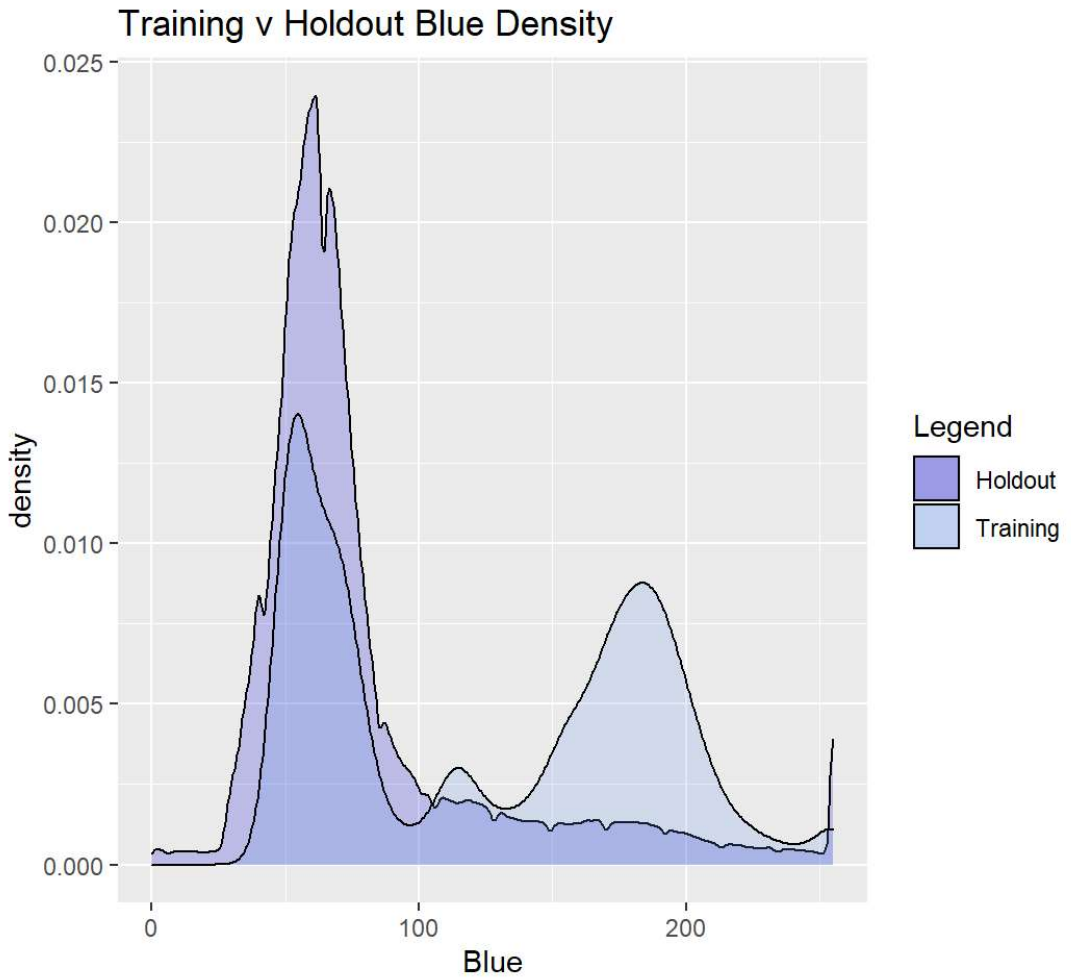
```
p1 = ggplot() +
  geom_density(data = ho_df, aes(x = Red, fill = 'Holdout'), alpha = 0.2) +
  geom_density(data = df, aes(x = Red, fill = 'Training'), alpha = 0.2) +
  scale_fill_manual('Legend', values = c('red', 'tomato')) +
  labs(title = 'Training v Holdout Red Density')

p2 = ggplot() +
  geom_density(data = ho_df, aes(x = Green, fill = 'Holdout'), alpha = 0.2) +
  geom_density(data = df, aes(x = Green, fill = 'Training'), alpha = 0.2) +
  scale_fill_manual('Legend', values = c('green4', 'palegreen'))+
  labs(title = 'Training v Holdout Green Density')

p3 = ggplot() +
  geom_density(data = ho_df, aes(x = Blue, fill = 'Holdout'), alpha = 0.2) +
  geom_density(data = df, aes(x = Blue, fill = 'Training'), alpha = 0.2) +
  scale_fill_manual('Legend', values = c('blue3', 'cornflowerblue'))+
  labs(title = 'Training v Holdout Blue Density')

p1
```




```
p2
```


Training v Holdout Green Density
density
0.010
0.005
0.000
50
100
150
200
250
Green
Legend
Holdout
Training

p3



The density plots above indicate that there are large differences in the distributions of color values between the training and holdout data. This difference exposed much of the over-training that was observed in some models when predictions were made on the holdout data.

To prevent this condition from occurring for future search and rescue initiatives where data might not exist to indicate over-training until issues have arisen from incorrect predictions, it may be prudent to examine training prior to model training to ensure that it is representative of the various environments in the search area. For example, if it is known that the search area has five biome types: city, forest, desert, swamp, and coastline, it may be possible to use k-means clustering on the training data with k=5 to create clusters of data and then sample from each cluster to ensure that the training data contains representative samples of each biome. While an expansive validation process for the training data may not be possible in every situation, this process could help avoid some of the performance loss that was observed in this report when the models were exposed to new data.

Another process that could avoid the performance loss from new data would be to avoid using static models and instead continuously update the model as new data arrives by performing the same training/test split on new data. The new training data could then be used to create an updated model that, when trained, could be placed into use as the production model. However, this approach is only viable if computational and human resources are available to run this process.

11 Supplementary Materials

11.1 Test Holdout Variable Assignment

The following confusion matrix was generated as an example of model performance when variables were incorrectly assigned:

```
ho_df2 = ho_df
colnames(ho_df2) = c('Red', 'Blue', 'Green', 'class')

confusionMatrix(predict(model.logistic,
                        newdata = ho_df2[,1:3]),
                ho_df2$class)
```

```
#> Confusion Matrix and Statistics
#>
#>           Reference
#> Prediction      BT     NBT
#>        BT     5503 1254690
#>        NBT    8977  735007
#>
#>                Accuracy : 0.3695
#>                  95% CI : (0.3688, 0.3702)
#>     No Information Rate : 0.9928
#>     P-Value [Acc > NIR] : 1
#>
#>                   Kappa : -0.0057
#>
#>  Mcnemar's Test P-Value : <2e-16
#>
#>             Sensitivity : 0.380041
#>             Specificity : 0.369406
#>          Pos Pred Value : 0.004367
#>          Neg Pred Value : 0.987934
#>              Prevalence : 0.007225
#>          Detection Rate : 0.002746
#>    Detection Prevalence : 0.628783
#>       Balanced Accuracy : 0.374724
#>
#>        'Positive' Class : BT
#>
```

```
remove(ho_df2)
```

It is clear from the poor model performance statistics and confusion matrix that the variable assignment is incorrect.

11.2 RF ntrees

```
cl = makePSOCKcluster(4)
registerDoParallel(cl)

set.seed(304)
model.rf_test = train(collapse.Class ~ .,
                      data = df,
                      method = 'rf',
                      ntree = 1000,
                      trControl = ctrl,
                      allowParallel = TRUE)
```

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

```
model.rf_test
```

```
#> Random Forest
#>
#> 63241 samples
#>     3 predictor
#>     2 classes: 'BT', 'NBT'
#>
#> 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     0.9970747  0.9524527
#>   3     0.9968849  0.9494033
#>
#> Accuracy was used to select the optimal model using the largest value.
#> The final value used for the model was mtry = 2.
```

```
stopCluster(cl)
```

```
model_ntrees = c(500,1000)
model.rf_test.cm = confusionMatrix(model.rf_test$pred$pred, model.rf_test$pred$obs)
model.rf.cm = confusionMatrix(model.rf$pred$pred, model.rf$pred$obs)
model_accs = c(model.rf.cm$overall[[1]], model.rf_test.cm$overall[[1]])
model_sens = c(model.rf.cm$byClass[[1]], model.rf_test.cm$byClass[[1]])
model_spec = c(model.rf.cm$byClass[[2]], model.rf_test.cm$byClass[[2]])

knitr::kable(data.frame(N_Trees = model_ntrees,
                        Accuracy = model_accs,
                        Sensitivity = model_sens,
                        specificity = model_spec))
```

N_Trees	Accuracy	Sensitivity	specificity
500	0.9969719	0.9465875	0.9986360
1000	0.9969798	0.9463403	0.9986524

As demonstrated in the table above, there is no discernible change in model performance between $n_{trees} = 500$ and $n_{trees} = 1000$, therefore $n_{trees} = 500$ was used.

11.3 Environment

This project was completed in the following environment:

```
sessionInfo()
```

```
#> R version 4.1.0 (2021-05-18)
#> Platform: x86_64-w64-mingw32/x64 (64-bit)
#> Running under: Windows 10 x64 (build 19042)
#>
#> Matrix products: default
#>
#> locale:
#> [1] LC_COLLATE=English_United States.1252
#> [2] LC_CTYPE=English_United States.1252
#> [3] LC_MONETARY=English_United States.1252
#> [4] LC_NUMERIC=C
#> [5] LC_TIME=English_United States.1252
#>
#> attached base packages:
#> [1] stats     graphics  grDevices utils     datasets  methods   base
#>
#> other attached packages:
#>  [1] vroom_1.5.3     plotROC_2.2.1   plotly_4.9.4.1  class_7.3-19
#>  [5] yardstick_0.0.8 broom_0.7.8     caret_6.0-88    lattice_0.20-44
#>  [9] GGally_2.1.2    skimr_2.1.3     forcats_0.5.1   stringr_1.4.0
#> [13] dplyr_1.0.6     purrr_0.3.4     readr_1.4.0     tidyr_1.1.3
#> [17] tibble_3.1.2    ggplot2_3.3.5   tidyverse_1.3.1
#>
#> loaded via a namespace (and not attached):
#>  [1] nlme_3.1-152         fs_1.5.0             bit64_4.0.5
#>  [4] lubridate_1.7.10     RColorBrewer_1.1-2   httr_1.4.2
#>  [7] repr_1.1.3           tools_4.1.0          backports_1.2.1
#> [10] bslib_0.2.5.1        utf8_1.2.1           R6_2.5.0
#> [13] rpart_4.1-15         lazyeval_0.2.2       DBI_1.1.1
#> [16] colorspace_2.0-1     nnet_7.3-16          withr_2.4.2
#> [19] tidyselect_1.1.1     bit_4.0.4            compiler_4.1.0
#> [22] glmnet_4.1-1         cli_2.5.0            rvest_1.0.0
#> [25] xml2_1.3.2           labeling_0.4.2       sass_0.4.0
#> [28] scales_1.1.1         proxy_0.4-26         digest_0.6.27
#> [31] rmarkdown_2.9        base64enc_0.1-3      pkgconfig_2.0.3
#> [34] htmltools_0.5.1.1    highr_0.9            dbplyr_2.1.1
#> [37] htmlwidgets_1.5.3    rlang_0.4.11         readxl_1.3.1
#> [40] rstudioapi_0.13      shape_1.4.6          farver_2.1.0
#> [43] jquerylib_0.1.4      generics_0.1.0       jsonlite_1.7.2
#> [46] crosstalk_1.1.1      ModelMetrics_1.2.2.2 magrittr_2.0.1
#> [49] Matrix_1.3-3         Rcpp_1.0.6           munsell_0.5.0
#> [52] fansi_0.5.0          lifecycle_1.0.0      stringi_1.6.2
#> [55] pROC_1.17.0.1        yaml_2.2.1           MASS_7.3-54
#> [58] plyr_1.8.6           recipes_0.1.16       grid_4.1.0
#> [61] crayon_1.4.1         haven_2.4.1          splines_4.1.0
#> [64] hms_1.1.0            knitr_1.33           pillar_1.6.1
#> [67] reshape2_1.4.4       codetools_0.2-18     stats4_4.1.0
#> [70] reprex_2.0.0         glue_1.4.2           evaluate_0.14
#> [73] data.table_1.14.0    modelr_0.1.8         tzdb_0.1.2
#> [76] vctrs_0.3.8          foreach_1.5.1        cellranger_1.1.0
#> [79] gtable_0.3.0         reshape_0.8.8        assertthat_0.2.1
#> [82] xfun_0.24            gower_0.2.2          prodlim_2019.11.13
#> [85] e1071_1.7-7          viridisLite_0.4.0    survival_3.2-11
#> [88] timeDate_3043.102    iterators_1.0.13     lava_1.6.9
#> [91] ellipsis_0.3.2       ipred_0.9-11
```