Disaster Relief P2

Robert Andrews

8/14/2022

1 Introduction

In 2010, Haiti experienced a massive earthquake which displaced numerous civilians from their homes. Relief entities struggled to get necessary commodities like food and water to survivors due to the destruction of the country's communications and transportation infrastructure. A team from Rochester Institute of Technology collected thousands of images to try and decipher where survivors were located. It was determined that many survivors created temporary shelters using blue tarps, and these tarps became essential markers for relief efforts. However, it was difficult for teams of people cycle through thousands of pictures. Luckily, they determined that computer algorithms could be leveraged to rapidly identify locations of displaced persons by determining the existence of blue tarps in the images.

The goal of this analysis is to train a series of computer models that could be used for this kind of disaster relief effort. I will use pixel data collected from the Haiti disaster to train, validate, and select the most effective models to locate disaster victims. I will use statistical learning methods to produce a logistic regression model, linear discriminant analysis model (LDA), quadratic discriminant analysis model (QDA), k-nearest neighbors model (KNN), a lasso penalized regression model, a support vector machine model, and a random forest model.

It is my hope that one, if not all, of these models can be leveraged to save lives.

2 Data Pre-Processing

The data for this analysis consists of two separate data sets: one that will be used to train each model consisting of 63,241 observations, and a holdout set with 2,008,622 observations to test each model.

2.1 Training Data

I begin data pre-processing by loading the pixel training data contained in a single csv file.

```
# load the data
train = read.csv("HaitiPixels.csv")
```

This is a pretty clean training set, but to build proper classification models, I need to create a binary response variable. I create a binary factor variable called Is.BTarp with two classes: "Tarp" if the class is Blue Tarp, and "Other" if the class is not Blue Tarp. I set the Is.BTarp reference class to "Other" and verify that this reference selection was successful.

```
# create binary variable for Class
# YES if Blue Tarp, else NO
train <- train %>%
  mutate(Is.BTarp=ifelse(Class=='Blue Tarp','Tarp','Other'))
```

```
# remove Class column
train = train[,2:5]
```

```
# convert Is.BTarp to factor variable with two levels
train$Is.BTarp = as.factor(train$Is.BTarp)
```

```
#>       Tarp
#> Other    0
#> Tarp     1
```

```
# ensure that reference class is other
train$Is.BTarp<-relevel(train$Is.BTarp, ref = "Other")
```

```
contrasts(train$Is.BTarp)
```

```
#>       Tarp
#> Other    0
#> Tarp     1
```

I check to ensure that the classes were coded properly, which they were.

2.2 Holdout Data

The holdout data is stored in 8 separate text files, with two separate classes. The first class contains the data where blue tarps were identified, and the second class contains the data where other items were identified. The text files contain a bit of metadata and unnecessary predictors. Additionally, all of these text files will need to be merged into one data set, so this requires a bit of pre-processing. I first load the blue tarp data for pre-processing.

```
# load text files with data that correspond to blue tarps
tarp1 = read.table("HO_Blue_Tarps_1.txt", sep="", comment.char=";", header=FALSE)
tarp2 = read.table("HO_Blue_Tarps_2.txt", sep="", comment.char=";", header=FALSE)
tarp3 = read.table("HO_Blue_Tarps_3.txt", sep="", comment.char=";", header=FALSE)
tarp4 = read.table("HO_Blue_Tarps_4.txt", sep="", comment.char=";", header=FALSE)
```

I drop the unneeded data by sub-setting the columns that I want to use.

```
# subset tarp1,3, and 4
tarp1 = tarp1[,8:10]
tarp3 = tarp3[,8:10]
tarp4 = tarp4[,8:10]
```

I also rename the columns in the tarp2 subset to be consistent with the other subsets.

```
# rename tarp column heading to be congruent with other tarp data frames
tarp2 = tarp2 %>%
  rename(
    V8 = V1,
    V9 = V2,
    V10 = V3
  )
```

Next, I combine the four separate blue tarp data frames into one.

```
# tarps1-4 into one data set
HO_blue = rbind(tarp1, tarp2)
HO_blue = rbind(HO_blue, tarp3)
HO_blue = rbind(HO_blue, tarp4)
```

Since the unified data set that corresponds to images where blue tarps were located, I add the Is.BTarp predictor column with every observation indicating that a blue tarp was identified. After this. I convert it into a factor variable.

```
# add Is.BTarp Column to HO_blue data
HO_blue['Is.BTarp'] = c('Tarp')
```

```
# convert Is.BTarp to factor variable with two levels
HO_blue$Is.BTarp = as.factor(HO_blue$Is.BTarp)
```

I then repeat this process, but now for the data that correspond to images without blue tarps.

```
# load text files with data that correspond to blue tarps
other1 = read.table("HO_Other_1.txt", sep="", comment.char=";", header=FALSE)
other2 = read.table("HO_Other_2.txt", sep="", comment.char=";", header=FALSE)
other3 = read.table("HO_Other_3.txt", sep="", comment.char=";", header=FALSE)
other4 = read.table("HO_Other_4.txt", sep="", comment.char=";", header=FALSE)
```

```
# select only pixel columns from each data subet
# subset tarp1,3, and 4
other1 = other1[,8:10]
other2 = other2[,8:10]
other3 = other3[,8:10]
other4 = other4[,8:10]
```

```
# others1-4 into one data set
HO_other = rbind(other1,other2)
HO_other = rbind(HO_other,other3)
HO_other = rbind(HO_other,other4)
```

I add Is.BTarp to this dataframe with every corresponding observation as “Other” since there were no blue tarps associated with this data.

```
# add Is.BTarp Column to HO_blue data
HO_other['Is.BTarp'] = c('Other')
```

```
# convert Is.BTarp to factor variable with two levels
HO_other$Is.BTarp = as.factor(HO_other$Is.BTarp)
```

Now that the blue tarp and other holdout data subsets are prepared, I combine them.

```
# merge HO_blue and HO_other
holdout = rbind(HO_blue,HO_other)
```

The pixel columns are not labeled descriptively, so I update the column names to reflect the pixel colors observed.

```
# rename V1, V2, and V3 to Red, Green, and Blue
holdout = holdout %>%
  rename(
    Red = V8,
    Green = V9,
    Blue = V10
  )
```

```
# switch Is.BTarp reference class to "Other"
holdout$Is.BTarp<-relevel(holdout$Is.BTarp, ref = "Other")
```

Finally, I do one last check to ensure that the Is.BTarp classes are correctly coded.

```
contrasts(train$Is.BTarp)
```

```
#>       Tarp
#> Other    0
#> Tarp     1
```

3 Exploratory Data Analysis

Next, I proceed with exploratory data analysis to inspect any additional patterns and relationships that can be identified by inspection.

I begin Exploratory Data Analysis (EDA) by evaluating initial data patterns with the average number of pixels for each Class. For this analysis, I am most interested in the concentration of Red, Blue, and Green Pixels associated with the Blue Tarp Class.

```
# sort the mean of predictors by class
holdout %>%
  group_by(Is.BTarp) %>%
  summarize(Avg.Blue = mean(Blue), Avg.Green = mean(Green), Avg.Red = mean(Red)) %>%
  kable(digits=3)
```

Is.BTarp	Avg.Blue	Avg.Green	Avg.Red
Other	81.759	105.173	118.308
Tarp	167.976	135.764	112.144

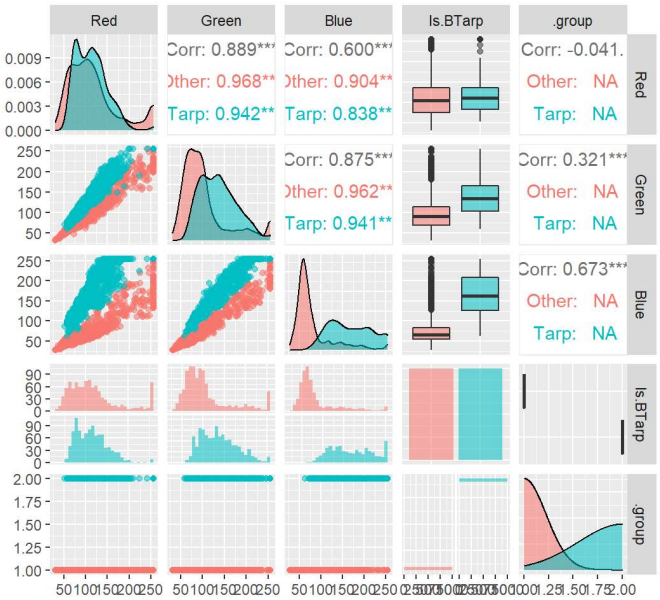
The mean Blue Pixel values by Is.BTarp are highest compared to Red and Green when Blue Tarp, which indicates the identification of a blue tarp, is the corresponding Class. This suggests that higher concentrations of Blue relative to the other two pixel colors corresponds to the location of blue tarps.

For the remaining EDA, I will use a stratified random sample of the holdout data to generate visualizations.

```
set.seed(7)
# get random sample for holdout set
eda_set <- holdout %>%
  group_by(Is.BTarp) %>%
  sample_n(size=1000)
```

I use the ggpairs function to efficiently compare the three quantitative predictors, Red, Blue and, Green, and the response variable, Is.BTarp. I exclude Class at this point in the analysis as Is.BTarp takes over as the response variable of interest.

```
# ggpairs plot
eda_set %>%
  ggpairs(aes(color=Is.BTarp,alpha=0.2))
```



The Red, Blue, and Green predictors are highly correlated with one another upon inspection of the scatter plots and of the correlation statistics displayed. Hopefully this is coincidental and will not result in multi-collinearity during the model building process. I will be able to determine this during evaluation of the predictors during feature selection.

The box plots yield a similar conclusion to the average pixel tables from the data wrangling section: there is a higher concentration of blue pixels for the Tarp Class than for the Other class. The average number of Green Pixels appear to be slightly higher for blue tarps than for other, but this may have a negligible impact. Conversely, the concentration of red pixels is roughly the same regardless of Class, although the spread of the red pixels for each Is.BTarp class is different.

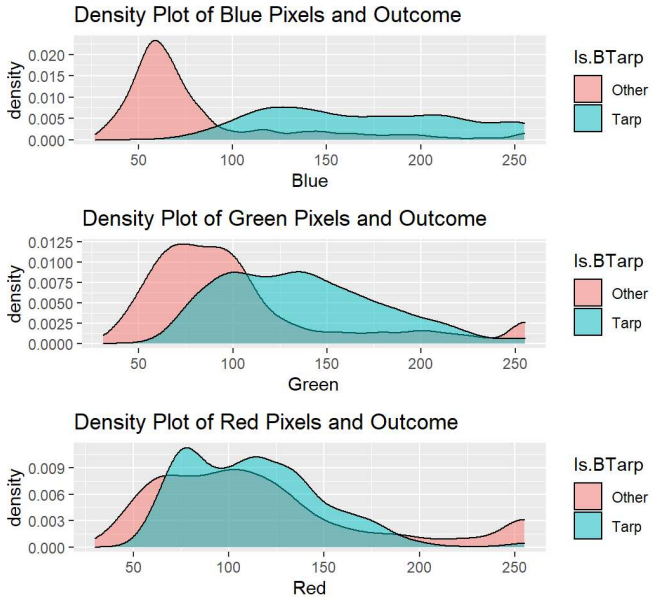
I want to take a closer look at the density plots, so I generate individual plots and grid them into one graph to get a better look.

```
# create a grid of density plots
blue = eda_set %>%
  ggplot(aes(x=Blue, fill=Is.BTarp))+
  geom_density(alpha=0.5)+
  labs(title= "Density Plot of Blue Pixels and Outcome")

green = eda_set %>%
  ggplot(aes(x=Green, fill=Is.BTarp))+
  geom_density(alpha=0.5)+
  labs(title= "Density Plot of Green Pixels and Outcome")

red = eda_set %>%
  ggplot(aes(x=Red, fill=Is.BTarp))+
  geom_density(alpha=0.5)+
  labs(title= "Density Plot of Red Pixels and Outcome")

# grid these density plots
grid.arrange(blue, green, red)
```




```
#
```

The relationship between the Tarp and red pixels and Tarp and green Pixels is dubious solely from inspection of the density graphs. However, it is clear that as the concentration of blue pixels increases, the frequency of Tarp classification increases.

Now that I have to determined potential relationships that may exist vai EDA, I begin the model building process by selecting features.

4 Model Fitting, Tuning, Parameter Selection, and Evaluation

4.1 Feature Selection

Based on EDA, I would like to use all three predictors, Red, Green, and Blue for my models. However, I want evidence that all the features will be useful, and I collect this evidence using automated search procedures. I use forward selection, backward selection, and stepwise selection to determine which features should be included in my initial model. I will use the Caret package for this purpose. Additionally, I will use a stratified random sample of the data computational efficiency since the holdout set contains more than 2,000,000 observations.

```
set.seed(7)
# generate large sample for holdout automated selection
auto_set <- holdout %>%
  group_by(Is.BTarp) %>%
  sample_n(size=10000)
```

I will use the recommended predictors from this feature selection to simplify the feature selection process for each model that I use, even though this search parameter is specified for generalize linear models. I previously attempted to use automated feature selection for the non generalized linear models, however, these procedures recommended models that provided counter-intuitive, and arguably counterproductive, results.Additionally, this data only contain three candidate predictors, so the need to eliminate features is not as pressing as it would be for a data set with many parameters.

```
set.seed(7)
# use automated search procedure to narrow down predictors
trControl = caret::trainControl(method="none")
model = caret::train(Is.BTarp~., data=auto_set, family='binomial', trControl=trControl,
                     method="glmStepAIC", direction="forward")
```

```
#> Start:  AIC=27727.89
#> .outcome ~ 1
#>
#>         Df Deviance   AIC
#> + Blue   1    16689 16693
#> + Green  1    25754 25758
#> + Red    1    27630 27634
#> <none>        27726 27728
#>
#> Step:  AIC=16693.4
#> .outcome ~ Blue
#>
#>         Df Deviance     AIC
#> + Red    1    971.5   977.5
#> + Green  1   1392.9  1398.9
#> <none>      16689.4 16693.4
#>
#> Step:  AIC=977.51
#> .outcome ~ Blue + Red
#>
#>         Df Deviance    AIC
#> + Green  1   890.46 898.46
#> <none>       971.51 977.51
#>
#> Step:  AIC=898.46
#> .outcome ~ Blue + Red + Green
```

```
# look at final model coefficients
summary(model$finalModel)
```

```
#>
#> Call:
#> NULL
#>
#> Deviance Residuals:
#>     Min       1Q   Median       3Q      Max
#> -5.8836  -0.0001   0.0000   0.0000   2.7433
#>
#> Coefficients:
#>             Estimate Std. Error z value Pr(>|z|)
#> (Intercept) -2.47447    0.19232 -12.867  < 2e-16 ***
#> Blue         0.54765    0.02703  20.259  < 2e-16 ***
#> Red         -0.34236    0.02292 -14.939  < 2e-16 ***
#> Green       -0.19532    0.02525  -7.735 1.04e-14 ***
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> (Dispersion parameter for binomial family taken to be 1)
#>
#>     Null deviance: 27725.89  on 19999  degrees of freedom
#> Residual deviance:   890.46  on 19996  degrees of freedom
#> AIC: 898.46
#>
#> Number of Fisher Scoring iterations: 12
```

Forward selection suggests that Blue, Green, and Red should all be predictors as they result in the lowest AIC value of 898.46. Additionally, each recommended predictor is highly significant – having p-values that are very close to 0 as displayed in the summary statistics.

Next, I will use backward selection and compare the results to forward selection.

```
set.seed(7)
# use automated search procedure to narrow down predictors
trControl = caret::trainControl(method="none")
model = caret::train(Is.BTarp~., data=auto_set, family='binomial', trControl=trControl,
                     method="glmStepAIC", direction="backward")
```

```
#> Start:  AIC=898.46
#> .outcome ~ Red + Green + Blue
#>
#>         Df Deviance    AIC
#> <none>        890.5  898.5
#> - Green  1    971.5  977.5
#> - Red    1   1392.9 1398.9
#> - Blue   1   7179.1 7185.1
```

```
# check summary statistics for the final model
summary(model$finalModel)
```

```
#>
#> Call:
#> NULL
#>
#> Deviance Residuals:
#>     Min       1Q   Median       3Q      Max
#> -5.8836  -0.0001   0.0000   0.0000   2.7433
#>
#> Coefficients:
#>             Estimate Std. Error z value Pr(>|z|)
#> (Intercept) -2.47447    0.19232 -12.867  < 2e-16 ***
#> Red         -0.34236    0.02292 -14.939  < 2e-16 ***
#> Green       -0.19532    0.02525  -7.735 1.04e-14 ***
#> Blue         0.54765    0.02703  20.259  < 2e-16 ***
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> (Dispersion parameter for binomial family taken to be 1)
#>
#>     Null deviance: 27725.89  on 19999  degrees of freedom
#> Residual deviance:   890.46  on 19996  degrees of freedom
#> AIC: 898.46
#>
#> Number of Fisher Scoring iterations: 12
```

Backward selection produces near-identical results as forward selection, with an AIC of 898.46. I will now conduct stepwise selection before making a final determination on which predictors to use in my initial model.

```
set.seed(7)
# use automated search procedure to narrow down predictors
trControl = caret::trainControl(method="none")
model = caret::train(Is.BTarp~., data=auto_set, family = 'binomial', trControl=trControl,
                     method="glmStepAIC", direction="both")
```

```
#> Start:  AIC=898.46
#> .outcome ~ Red + Green + Blue
#>
#>         Df Deviance    AIC
#> <none>        890.5  898.5
#> - Green  1    971.5  977.5
#> - Red    1   1392.9 1398.9
#> - Blue   1   7179.1 7185.1
```

```
# look at summary for final model
summary(model$finalModel)
```

```
#>
#> Call:
#> NULL
#>
#> Deviance Residuals:
#>     Min       1Q   Median       3Q      Max
#> -5.8836  -0.0001   0.0000   0.0000   2.7433
#>
#> Coefficients:
#>             Estimate Std. Error z value Pr(>|z|)
#> (Intercept) -2.47447    0.19232 -12.867  < 2e-16 ***
#> Red         -0.34236    0.02292 -14.939  < 2e-16 ***
#> Green       -0.19532    0.02525  -7.735 1.04e-14 ***
#> Blue         0.54765    0.02703  20.259  < 2e-16 ***
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> (Dispersion parameter for binomial family taken to be 1)
#>
#>     Null deviance: 27725.89  on 19999  degrees of freedom
#> Residual deviance:   890.46  on 19996  degrees of freedom
#> AIC: 898.46
#>
#> Number of Fisher Scoring iterations: 12
```

Conveniently, stepwise selection yields nearly identical results to forward and backward selection, with an AIC 898.46. I am ready to begin training and validating a model that regresses Is.BTarp against Red, Green, and Blue.

4.2 Logistic Regression

4.2.1 Training and Validation

Per my feature selection results, I fit a logistic regression model that regression Is.BTarp against Red, Green, and Blue, all available predictors. I perform 10-fold Cross Validation (10-fold CV) on the model before testing the model on the holdout data set. 10-fold CV is the best validation option for this data. This data set is far too large to perform Leave One Out Cross Validation (LOOCV). 10-fold CV has a major computational advantage over LOOCV. Additionally, 10-fold CV will hedge against over fitting by training the data on 10 different subsets of stratified data rather than training on a single training subset.

```
# train model with Blue, Red, and Green as predictors
# method: k-fold cross validation with 10 folds
# set seed to 7
set.seed(7)
trControl <- caret::trainControl(method='cv', number=10, savePredictions = TRUE,
                                 classProbs = TRUE)
logistic <- caret::train(Is.BTarp ~ ., data=train,
                     method='glm',
                     family = 'binomial',
                     trControl=trControl)

logistic
```

```
#> Generalized Linear Model
#>
#> 63241 samples
#>     3 predictor
#>     2 classes: 'Other', 'Tarp'
#>
#> No pre-processing
#> Resampling: Cross-Validated (10 fold)
#> Summary of sample sizes: 56916, 56917, 56917, 56917, 56917, 56917, ...
#> Resampling results:
#>
#>   Accuracy  Kappa
#>   0.995272  0.9203101
```

The output above shows that the training accuracy of the logistic model is high with a value of 0.995272. I am ultimately more interested in the test accuracy than the train accuracy.

```
# get summary of logistic model
summary(logistic)
```

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

I extract the results from the model produced by cross validation.

```
# pull final model
logistic$finalModel
```

```
#>
#> Call:  NULL
#>
#> Coefficients:
#> (Intercept)          Red        Green         Blue
#>      0.2098      -0.2603      -0.2183       0.4724
#>
#> Degrees of Freedom: 63240 Total (i.e. Null);  63237 Residual
#> Null Deviance:       17900
#> Residual Deviance: 1770  AIC: 1778
```

Cross Validation produces a final model
$P = (e(0.2098 - -0.2603Red - 0.2183Green + 0.4724Blue)/(1 + e(0.2098 - 0.2603Red - 0.2183Green + 0.4724Blue)))$.

4.2.2 Results

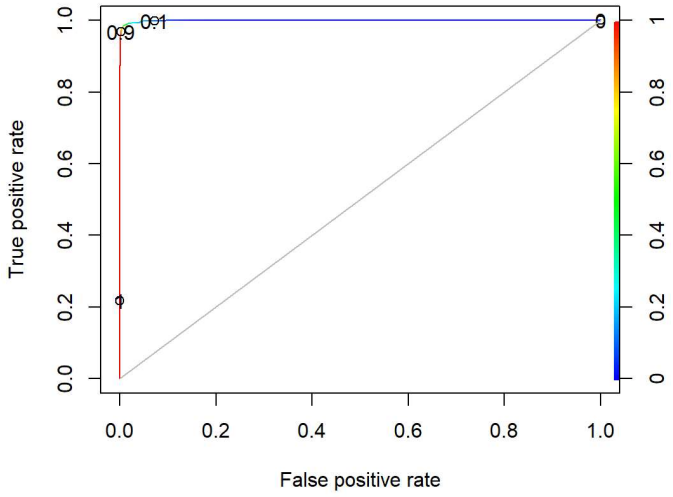
It is time to test this model on the holdout data. I firest apply this model to generate estimated probabilities. These probabilities will be used to interpret the results via an ROC curve and to select a prediction threshold.

```
# select best threshold value
pred.prob <- predict(logistic, holdout, type='prob')

predob <- ROCR::prediction(pred.prob$Tarp, holdout$Is.BTarp, label.ordering=c('Other', 'Tarp'))

model.roc <- ROCR::performance(predob, measure='tpr', x.measure='fpr')

log.ROC = plot(model.roc, colorize=T, print.cutoffs.at=c(0, 0.1, 0.9, 1.0))+
  lines(x=c(0,1), y=c(0,1), col='grey')
```




```
log.ROC
```

```
#> integer(0)
```

The shape of the ROC curve indicates that this is a highly accurate model, having a curve that approaches the point [0,1], which is the location of a perfect model.

```
# get AUC value for this curve
auc.calc = performance(predob, "auc")
auc.log = auc.calc@y.values
auc.log
```

```
#> [[1]]
#> [1] 0.9993877
```

The AUC value of 0.9993877 suggests that the logistic model is far better than random guessing, given that 0.9993877 > 0.5.

I now use the probabilities to evaluate the best prediction thresholds based on Accuracy, Sensitivity, False Positive Rate (FPR), and Precision.

```
# use thresholder function to analyze and select the best threshold value
# primary goal is to maximize TPR and minimize FNR
thresh.stats = thresholder(logistic,
                           threshold = seq(0.05, 0.95, by = 0.05),
                           statistics = 'all')

thresh.stats$FPR = 1-thresh.stats$Sensitivity

thresh.stats$ERR.Rate = 1-thresh.stats$Accuracy
```

Next, I generate a table to that contains the Error Rate, Accuracy, Sensitivty, False Positive Rate (FPR), and Precision performance metrics by varying probability thresholds.

```
# create table to view various threshold metrics for threshold selection
log.thresh = thresh.stats %>%
    dplyr::select("prob_threshold", "ERR.Rate", "Accuracy", "Sensitivity", "FPR", "Precision")%>%
    knitr::kable(digits=4) %>%
    kableExtra::kable_styling(full_width=FALSE)

# Highest Accuracy
log.Acc = thresh.stats %>% dplyr::slice_max(Accuracy) %>%
    dplyr::select("prob_threshold", "ERR.Rate", "Accuracy", "Sensitivity", "FPR", "Precision")%>%
    knitr::kable(digits=4) %>%
    kableExtra::kable_styling(full_width=FALSE)
log.Acc
```

prob_threshold	ERR.Rate	Accuracy	Sensitivity	FPR	Precision
0.75	0.0042	0.9958	0.9983	0.0017	0.9973

```
# Highest TPR
log.Sen = thresh.stats %>% dplyr::slice_max(Sensitivity) %>%
    dplyr::select("prob_threshold", "ERR.Rate", "Accuracy", "Sensitivity", "FPR", "Precision")%>%
    knitr::kable(digits=4) %>%
    kableExtra::kable_styling(full_width=FALSE)
log.Sen
```

prob_threshold	ERR.Rate	Accuracy	Sensitivity	FPR	Precision
0.05	0.0068	0.9932	0.9997	3e-04	0.9934

```
# Lowest FPR
log.FPR = thresh.stats %>% dplyr::slice_min(FPR) %>%
    dplyr::select("prob_threshold", "ERR.Rate", "Accuracy", "Sensitivity", "FPR", "Precision")%>%
    knitr::kable(digits=4) %>%
    kableExtra::kable_styling(full_width=FALSE)
log.FPR
```

prob_threshold	ERR.Rate	Accuracy	Sensitivity	FPR	Precision
0.05	0.0068	0.9932	0.9997	3e-04	0.9934

```
# Highest Precision
log.Prec = thresh.stats %>% dplyr::slice_max(Precision) %>%
    dplyr::select("prob_threshold", "ERR.Rate", "Accuracy", "Sensitivity", "FPR", "Precision")%>%
    knitr::kable(digits=4) %>%
    kableExtra::kable_styling(full_width=FALSE)
log.Prec
```

prob_threshold	ERR.Rate	Accuracy	Sensitivity	FPR	Precision
0.95	0.0131	0.9869	0.9872	0.0128	0.9992

```
# extract optimal sensitivity values for results table
log.sen.tab = thresh.stats %>% dplyr::slice_max(Sensitivity) %>%
     dplyr::select("prob_threshold", "ERR.Rate", "Accuracy", "Sensitivity", "FPR", "Precision")
log.sen.tab = log.sen.tab[1,]
row.names(log.sen.tab) = c("Logistic")
```

A lower threshold, near 0.05 is favored by Sensitivity (True Positive Rate) and False Positive Rate. Accuracy and Precision favor a threshold of 0.75. This logistic regression model is highly effective per these metrics. I choose a threshold of 0.05 since it has the highest Sensitivity of 0.9997. Sensitivity is the most important performance metric for threshold selection in this analysis. High Sensitivity translate to the accurate identification of survivors in this context. This is represented by the True Positive Rate (Sensitivity).

4.3 LDA Model

Now, I train my LDA model.

4.3.1 Training and Validation

I will use the same features recommended by the feature selection from the logistic model.

I train the LDA model using 10-fold cross validation with the training data set for reasons previously mentioned. Principally, the computational advantage that this has over LOOCV, and the hedge against over fitting that this method has compared to a simple validation split.

```
# train model using caret
set.seed(7)
trControl <- caret::trainControl(method='cv', number=10, savePredictions = TRUE,
                                 classProbs = TRUE)
lda = caret::train(Is.BTarp~., data=train ,trControl=trControl,
                   method="lda")

lda
```

```
#> Linear Discriminant Analysis
#>
#> 63241 samples
#>     3 predictor
#>     2 classes: 'Other', 'Tarp'
#>
#> No pre-processing
#> Resampling: Cross-Validated (10 fold)
#> Summary of sample sizes: 56916, 56917, 56917, 56917, 56917, 56917, ...
#> Resampling results:
#>
#>   Accuracy  Kappa
#>   0.983887  0.7521203
```

The output above shows that the LDA model has a high training accuracy of 0.983887. I am ultimately more interested in the test accuracy than the train accuracy.

I extract the components of the cross-validated model.

```
lda$finalModel
```

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

10-fold CV recommends a model with the following linear discriminant coefficients:
$Red: -0.02896984, Green: -0.02328544, Blue: 0.06923974.$

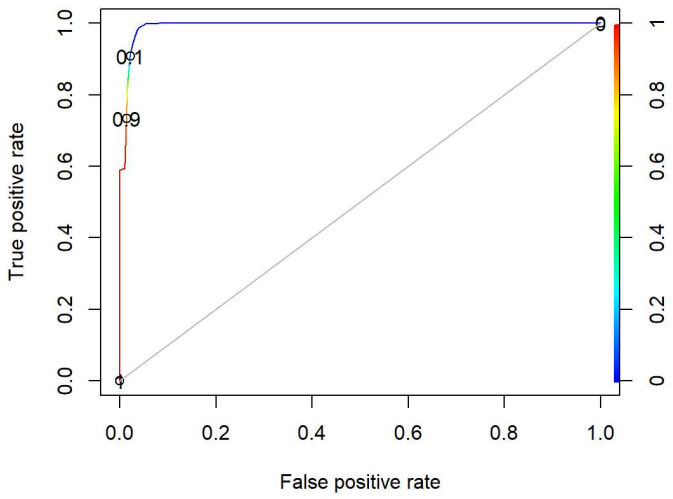
I test the final LDA model on the holdout set, again generating estimated probabilities.

```
# select best threshold value
pred.lda <- predict(lda, holdout, type='prob')

prelda <- ROCR::prediction(pred.lda$Tarp, holdout$Is.BTarp, label.ordering=c('Other', 'Tarp'))

model.roc <- ROCR::performance(prelda, measure='tpr', x.measure='fpr')

lda.ROC = plot(model.roc, colorize=T, print.cutoffs.at=c(0, 0.1, 0.9, 1.0), title="ROCR of LDA Performance on Hold
out Data ")+
  lines(x=c(0,1), y=c(0,1), col='grey')
```




```
lda.ROC
```

```
#> integer(0)
```

The shape of the ROC curve indicates that this is a highly accurate model, having a curve that approaches the point [0,1], although it is not as smooth as the logisitic regression ROC curve.

4.3.2 Results

```
auc.calc = performance(prelda, "auc")
auc.lda = auc.calc@y.values
auc.lda
```

```
#> [[1]]
#> [1] 0.9924212
```

The AUC calculation for the model prediction on the test data is 0.9924212. This is far more accurate than random guessing, similar to the logistic model. Now, I use the probabilities to evaluate the best prediction thresholds based on Accuracy, Sensitivity, False Positive Rate (FPR), and Precision.

```
# use thresholder function to analyze and select the best threshold value
# primary goal is to maximize TPR and minimize FNR
thresh.stats = thresholder(lda,
                           threshold = seq(0.05, 0.95, by = 0.05),
                           statistics = 'all')

thresh.stats$FPR = 1-thresh.stats$Sensitivity

thresh.stats$ERR.Rate = 1-thresh.stats$Accuracy
```

Next, I generate a table to that contains the Error Rate, Accuracy, Sensitivity, False Positive Rate (FPR), and Precision performance metrics by varying probability thresholds.

```
# create table to view various threshold metrics for threshold selection
lda.thresh = thresh.stats %>%
    dplyr::select("prob_threshold", "ERR.Rate", "Accuracy", "Sensitivity", "FPR", "Precision")%>%
    knitr::kable(digits=4) %>%
    kableExtra::kable_styling(full_width=FALSE)

# Highest Accuracy
lda.Acc = thresh.stats %>% dplyr::slice_max(Accuracy) %>%
    dplyr::select("prob_threshold", "ERR.Rate", "Accuracy", "Sensitivity", "FPR", "Precision")%>%
    knitr::kable(digits=4) %>%
    kableExtra::kable_styling(full_width=FALSE)
lda.Acc
```

prob_threshold	ERR.Rate	Accuracy	Sensitivity	FPR	Precision
0.15	0.0152	0.9848	0.9923	0.0077	0.992

```
# Highest TPR
lda.Sen = thresh.stats %>% dplyr::slice_max(Sensitivity) %>%
    dplyr::select("prob_threshold", "ERR.Rate", "Accuracy", "Sensitivity", "FPR", "Precision")%>%
    knitr::kable(digits=4) %>%
    kableExtra::kable_styling(full_width=FALSE)
lda.Sen
```

prob_threshold	ERR.Rate	Accuracy	Sensitivity	FPR	Precision
0.05	0.0154	0.9846	0.9936	0.0064	0.9905

```
# Lowest FPR
lda.FPR = thresh.stats %>% dplyr::slice_min(FPR) %>%
    dplyr::select("prob_threshold", "ERR.Rate", "Accuracy", "Sensitivity", "FPR", "Precision")%>%
    knitr::kable(digits=4) %>%
    kableExtra::kable_styling(full_width=FALSE)
lda.FPR
```

prob_threshold	ERR.Rate	Accuracy	Sensitivity	FPR	Precision
0.05	0.0154	0.9846	0.9936	0.0064	0.9905

```
# Highest Precision
lda.Prec = thresh.stats %>% dplyr::slice_max(Precision) %>%
    dplyr::select("prob_threshold", "ERR.Rate", "Accuracy", "Sensitivity", "FPR", "Precision")%>%
    knitr::kable(digits=4) %>%
    kableExtra::kable_styling(full_width=FALSE)
lda.Prec
```

prob_threshold	ERR.Rate	Accuracy	Sensitivity	FPR	Precision
0.95	0.0191	0.9809	0.9849	0.0151	0.9953

```
# extract optimal sensitivity values for results table
lda.sen.tab = thresh.stats %>% dplyr::slice_max(Sensitivity) %>%
     dplyr::select("prob_threshold", "ERR.Rate", "Accuracy", "Sensitivity", "FPR", "Precision")
lda.sen.tab = lda.sen.tab[1,]
row.names(lda.sen.tab) = c("LDA")
```

Accuracy, Sensitivity, and FPR each favor a low threshold – between 0.05 and 0.15. Conversely, Precision favors a high threshold, 0.95 in this case. In this case, I choose a threshold of 0.05 since it has the highest Sensitivity of 0.9936. Sensitivity is the most important performance metric for threshold selection in this analysis. High Sensitivity translate to the accurate identification of survivors in this context. This is represented by the True Positive Rate (Sensitivity)

4.4 QDA Model

Now it is time to train a QDA model.

4.4.1 Training and Validation

I will continue to use the same features, Red, Green, and Blue, given their predictive efficacy thus far.

```
# train model using caret
set.seed(7)
trControl <- caret::trainControl(method='cv', number=10, savePredictions = TRUE,
                                 classProbs = TRUE)
qda = caret::train(Is.BTarp~., data=train ,trControl=trControl,
                     method="qda")

qda
```

```
#> Quadratic Discriminant Analysis
#>
#> 63241 samples
#>     3 predictor
#>     2 classes: 'Other', 'Tarp'
#>
#> No pre-processing
#> Resampling: Cross-Validated (10 fold)
#> Summary of sample sizes: 56916, 56917, 56917, 56917, 56917, 56917, ...
#> Resampling results:
#>
#>   Accuracy   Kappa
#>   0.9946079  0.9057966
```

The output above shows that the QDA model has a high training accuracy of 0.9946079. I am ultimately more interested in the test accuracy than the train accuracy.

The final QDA model is summarized in the R output directly below.

```
qda$finalModel
```

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

4.4.2 Results

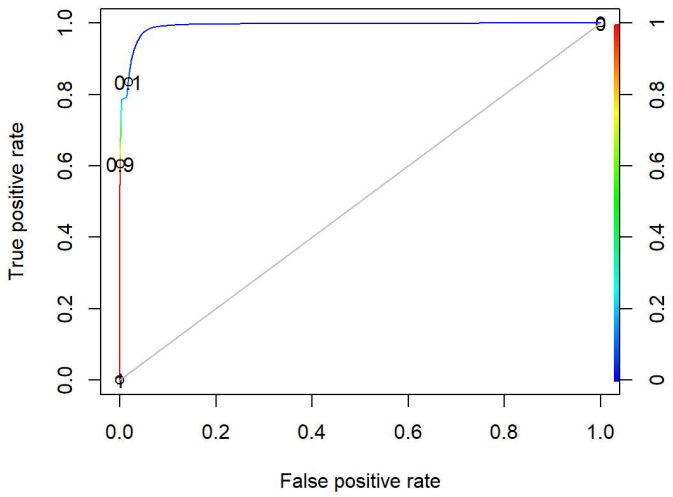
I test the final model using the holdout data set and produce a ROC curve to evaluate the form of the curve to determine testing performance.

```
# select best threshold value
pred.prob <- predict(qda, holdout, type='prob')

predob <- ROCR::prediction(pred.prob$Tarp, holdout$Is.BTarp, label.ordering=c('Other', 'Tarp'))

model.roc <- ROCR::performance(predob, measure='tpr', x.measure='fpr')

qda.ROC = plot(model.roc, colorize=T, print.cutoffs.at=c(0, 0.1, 0.9, 1.0))+
  lines(x=c(0,1), y=c(0,1), col='grey')
```



The QDA model is highly accurate per the ROC Curve. It also bends toward [0,1], which represents a perfect classification model.

```
# get AUC value for this curve
auc.calc = performance(predob, "auc")
auc.qda = auc.calc@y.values
auc.qda
```

```
#> [[1]]
#> [1] 0.9922366
```

QDA has an AUC value that nearly identical to the LDA model. The QDA model's prediction capability is almost perfect. Now, it is time to evaluate its optimal prediction thresholds.

```
# use thresholder function to analyze and select the best threshold value
# primary goal is to maximize TPR and minimize FNR
thresh.stats = thresholder(qda,
                           threshold = seq(0.05, 0.95, by = 0.05),
                           statistics = 'all')

thresh.stats$FPR = 1-thresh.stats$Sensitivity

thresh.stats$ERR.Rate = 1-thresh.stats$Accuracy
```

I generate the performance metric tables using the same process as earlier.

```
# create table to view various threshold metrics for threshold selection
qda.thresh = thresh.stats %>%
    dplyr::select("prob_threshold", "ERR.Rate", "Accuracy", "Sensitivity", "FPR", "Precision")%>%
    knitr::kable(digits=4) %>%
    kableExtra::kable_styling(full_width=FALSE)

# Highest Accuracy
qda.Acc = thresh.stats %>% dplyr::slice_max(Accuracy) %>%
    dplyr::select("prob_threshold", "ERR.Rate", "Accuracy", "Sensitivity", "FPR", "Precision")%>%
    knitr::kable(digits=4) %>%
    kableExtra::kable_styling(full_width=FALSE)
qda.Acc
```

prob_threshold	ERR.Rate	Accuracy	Sensitivity	FPR	Precision
0.75	0.0052	0.9948	0.9991	9e-04	0.9955

```
# Highest TPR
qda.Sen = thresh.stats %>% dplyr::slice_max(Sensitivity) %>%
    dplyr::select("prob_threshold", "ERR.Rate", "Accuracy", "Sensitivity", "FPR", "Precision")%>%
    knitr::kable(digits=4) %>%
    kableExtra::kable_styling(full_width=FALSE)
qda.Sen
```

prob_threshold	ERR.Rate	Accuracy	Sensitivity	FPR	Precision
0.05	0.0081	0.9919	1	0	0.9917

```
# Lowest FPR
qda.FPR = thresh.stats %>% dplyr::slice_min(FPR) %>%
    dplyr::select("prob_threshold", "ERR.Rate", "Accuracy", "Sensitivity", "FPR", "Precision")%>%
    knitr::kable(digits=4) %>%
    kableExtra::kable_styling(full_width=FALSE)
qda.FPR
```

prob_threshold	ERR.Rate	Accuracy	Sensitivity	FPR	Precision
0.05	0.0081	0.9919	1	0	0.9917

```
# Highest Precision
qda.Prec = thresh.stats %>% dplyr::slice_max(Precision) %>%
    dplyr::select("prob_threshold", "ERR.Rate", "Accuracy", "Sensitivity", "FPR", "Precision")%>%
    knitr::kable(digits=4) %>%
    kableExtra::kable_styling(full_width=FALSE)
qda.Prec
```

prob_threshold	ERR.Rate	Accuracy	Sensitivity	FPR	Precision
0.95	0.0117	0.9883	0.9898	0.0102	0.9981

```
qda.sen.tab = thresh.stats %>% dplyr::slice_max(Sensitivity) %>%
     dplyr::select("prob_threshold", "ERR.Rate", "Accuracy", "Sensitivity", "FPR", "Precision")
qda.sen.tab = qda.sen.tab[1,]
row.names(qda.sen.tab) = c("QDA")
```

Accuracy and Precision favor a higher threshold, ranging between 0.7 and 0.95. Sensitivity and FPR measures recommend a lower threshold near 0.05. In this case, I choose a threshold of 0.05 as I believe Sensitivity is the most important performance metric for threshold selection. Accurate identification of survivors is the most important objective for this model, and that is reflected by the True Positive Rate (Sensitivity)

4.5 KNN

Next, I build a KNN model.

4.5.1 Training and Validation

I train a KNN model, again using the same features and processes that have previously been leveraged.

```
# train model using caret
set.seed(7)

trControl <- caret::trainControl(method='cv', number=10, savePredictions = TRUE,
                                 classProbs = TRUE)
knn = caret::train(Is.BTarp~., data=train ,trControl=trControl,
                   method="knn" )
```

I extract the optimal number for K, as well as the results to determine the model accuracy on the cross validation data.

```
knn$finalModel
```

```
#> 5-nearest neighbor model
#> Training set outcome distribution:
#>
#> Other  Tarp
#> 61219  2022
```

```
# get training accuracy
knn$results
```

```
#>   k  Accuracy     Kappa   AccuracySD     KappaSD
#> 1 5 0.9972644 0.9558333 0.0005966112 0.009787899
#> 2 7 0.9972486 0.9556680 0.0007462108 0.011996334
#> 3 9 0.9972328 0.9555314 0.0006158702 0.009893115
```

The final KNN model has a k-value of 5 and a training accuracy of 0.9972644. While the final model display the inputs, it is important to clarify that the model includes the following inputs: $Predictors : Red, Green, Blue | k = 5$.

4.5.2 Results

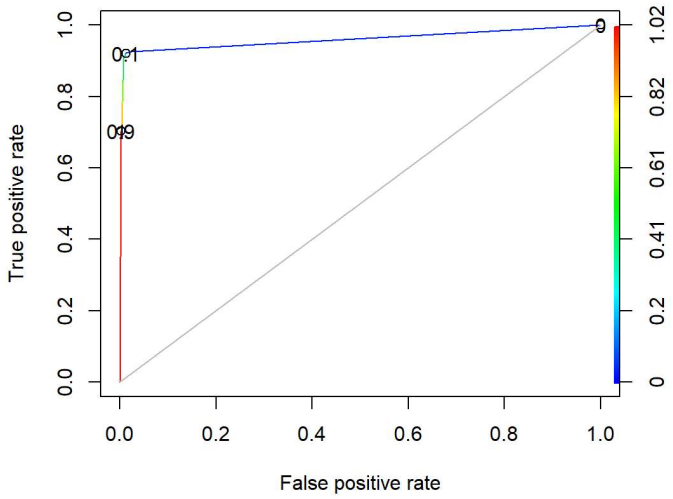
I test the model using the holdout data to generate estimated probabilities, and then produce

```
# select best threshold value
pred.prob <- predict(knn, holdout, type='prob')

predob <- ROCR::prediction(pred.prob$Tarp, holdout$Is.BTarp, label.ordering=c('Other', 'Tarp'))

model.roc <- ROCR::performance(predob, measure='tpr', x.measure='fpr')

knn.ROC = plot(model.roc, colorize=T, print.cutoffs.at=c(0, 0.1, 0.9, 1.0))+
  lines(x=c(0,1), y=c(0,1), col='grey')
```



Inspection of the KNN ROC Curve suggests that the KNN is sufficient to provide accurate predictions of where blue tarps are located. However, the curve does not ascend as smoothly along the x-axis as the other ROC curves I have produced thus far.

```
# get AUC value for this curve
auc.calc = performance(predob, "auc")
auc.knn = auc.calc@y.values
auc.knn
```

```
#> [[1]]
#> [1] 0.9598593
```

The AUC value for KNN is the first of the models tested that drops below 0.99; however this is still a very accurate model with as 0.9598593 is a considerably high AUC value. This is another highly accurate model.

Now, I generate the optimal KNN probability thresholds.

```
# use thresholder function to analyze and select the best threshold value
# primary goal is to maximize TPR and minimize FNR
thresh.stats = thresholder(knn,
                           threshold = seq(0.05, 0.95, by = 0.05),
                           statistics = 'all')

thresh.stats$FPR = 1-thresh.stats$Sensitivity

thresh.stats$ERR.Rate = 1-thresh.stats$Accuracy
```

Now I create the performance metric tables by probability threshold.

```
# create table to view various threshold metrics for threshold selection
knn.thresh = thresh.stats %>%
    dplyr::select("prob_threshold", "ERR.Rate", "Accuracy", "Sensitivity", "FPR", "Precision")%>%
    knitr::kable(digits=4) %>%
    kableExtra::kable_styling(full_width=FALSE)

# Highest Accuracy
knn.Acc = thresh.stats %>% dplyr::slice_max(Accuracy) %>%
    dplyr::select("prob_threshold", "ERR.Rate", "Accuracy", "Sensitivity", "FPR", "Precision")%>%
    knitr::kable(digits=4) %>%
    kableExtra::kable_styling(full_width=FALSE)
knn.Acc
```

prob_threshold	ERR.Rate	Accuracy	Sensitivity	FPR	Precision
0.50	0.0027	0.9973	0.9985	0.0015	0.9987
0.55	0.0027	0.9973	0.9985	0.0015	0.9987

```
# Highest TPR
knn.Sen = thresh.stats %>% dplyr::slice_max(Sensitivity) %>%
    dplyr::select("prob_threshold", "ERR.Rate", "Accuracy", "Sensitivity", "FPR", "Precision")%>%
    knitr::kable(digits=4) %>%
    kableExtra::kable_styling(full_width=FALSE)
knn.Sen
```

prob_threshold	ERR.Rate	Accuracy	Sensitivity	FPR	Precision
0.05	0.0045	0.9955	0.9997	3e-04	0.9957

```
# Lowest FPR
knn.FPR = thresh.stats %>% dplyr::slice_min(FPR) %>%
    dplyr::select("prob_threshold", "ERR.Rate", "Accuracy", "Sensitivity", "FPR", "Precision")%>%
    knitr::kable(digits=4) %>%
    kableExtra::kable_styling(full_width=FALSE)
knn.FPR
```

prob_threshold	ERR.Rate	Accuracy	Sensitivity	FPR	Precision
0.05	0.0045	0.9955	0.9997	3e-04	0.9957

```
# Highest Precision
knn.Prec = thresh.stats %>% dplyr::slice_max(Precision) %>%
    dplyr::select("prob_threshold", "ERR.Rate", "Accuracy", "Sensitivity", "FPR", "Precision")%>%
    knitr::kable(digits=4) %>%
    kableExtra::kable_styling(full_width=FALSE)
knn.Prec
```

prob_threshold	ERR.Rate	Accuracy	Sensitivity	FPR	Precision
0.95	0.0082	0.9918	0.9916	0.0084	0.9999

```
# extract optimal threshold via Sensitivity
knn.sen.tab = thresh.stats %>% dplyr::slice_max(Sensitivity) %>%
     dplyr::select("prob_threshold", "ERR.Rate", "Accuracy", "Sensitivity", "FPR", "Precision")
knn.sen.tab = knn.sen.tab[1,]
row.names(knn.sen.tab) = c("KNN")
```

The Accuracy metric favors a mid-ranged threshold between 0.5-0.55. Sensitivity and FPR favor a low threshold near 0.05, while Precision favors a very high threshold around 0.95. I choose a threshold of 0.05 as it translates to the highest Sensitivity of 0.9997, since I am using Sensitivity as the metric of most importance in this analysis.

4.6 Lasso Regression

Now it is time to create a penalized logistic regression model. I choose to train a Lasso Regression.

4.6.1 Training, and Validation

Unlike the previous models, the Lasso Regression has a tuning parameter, represented by lambda. This controls the strength of the penalty term of Lasso Regression. Thus, I need to add a tuning grid with specific parameters to find the optimal lambda value in addition to other optimal model parameters during cross validation. I expand the tuning grid to evaluate a large range of lambda values in hopes of creating a broad enough range to capture the optimal value. I maintain the same train control features, using 10-fold cross validation.

```
set.seed(7)
# Define the grid of tuning parameters to explore
lambdas <- 10^seq(1, -15, by=-0.2)  # Note the exponential scale!
tuneGrid <- expand.grid(alpha=1, lambda=tail(lambdas, -15))

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

lasso <- train(Is.BTarp ~ ., data=train, method='glmnet',
               trControl=trControl, tuneGrid=tuneGrid)
lasso
```

```
#> glmnet
#>
#> 63241 samples
#>     3 predictor
#>     2 classes: 'Other', 'Tarp'
#>
#> No pre-processing
#> Resampling: Cross-Validated (10 fold)
#> Summary of sample sizes: 56916, 56917, 56917, 56917, 56917, 56917, ...
#> Resampling results across tuning parameters:
#>
#>   lambda        Accuracy   Kappa
#>   1.000000e-15  0.9952404  0.9196935
#>   1.584893e-15  0.9952404  0.9196935
#>   2.511886e-15  0.9952404  0.9196935
#>   3.981072e-15  0.9952404  0.9196935
#>   6.309573e-15  0.9952404  0.9196935
#>   1.000000e-14  0.9952404  0.9196935
#>   1.584893e-14  0.9952404  0.9196935
#>   2.511886e-14  0.9952404  0.9196935
#>   3.981072e-14  0.9952404  0.9196935
#>   6.309573e-14  0.9952404  0.9196935
#>   1.000000e-13  0.9952404  0.9196935
#>   1.584893e-13  0.9952404  0.9196935
#>   2.511886e-13  0.9952404  0.9196935
#>   3.981072e-13  0.9952404  0.9196935
#>   6.309573e-13  0.9952404  0.9196935
#>   1.000000e-12  0.9952404  0.9196935
#>   1.584893e-12  0.9952404  0.9196935
#>   2.511886e-12  0.9952404  0.9196935
#>   3.981072e-12  0.9952404  0.9196935
#>   6.309573e-12  0.9952404  0.9196935
#>   1.000000e-11  0.9952404  0.9196935
#>   1.584893e-11  0.9952404  0.9196935
#>   2.511886e-11  0.9952404  0.9196935
#>   3.981072e-11  0.9952404  0.9196935
#>   6.309573e-11  0.9952404  0.9196935
#>   1.000000e-10  0.9952404  0.9196935
#>   1.584893e-10  0.9952404  0.9196935
#>   2.511886e-10  0.9952404  0.9196935
#>   3.981072e-10  0.9952404  0.9196935
#>   6.309573e-10  0.9952404  0.9196935
#>   1.000000e-09  0.9952404  0.9196935
#>   1.584893e-09  0.9952404  0.9196935
#>   2.511886e-09  0.9952404  0.9196935
#>   3.981072e-09  0.9952404  0.9196935
#>   6.309573e-09  0.9952404  0.9196935
#>   1.000000e-08  0.9952404  0.9196935
#>   1.584893e-08  0.9952404  0.9196935
#>   2.511886e-08  0.9952404  0.9196935
#>   3.981072e-08  0.9952404  0.9196935
#>   6.309573e-08  0.9952404  0.9196935
#>   1.000000e-07  0.9952404  0.9196935
#>   1.584893e-07  0.9952404  0.9196935
#>   2.511886e-07  0.9952404  0.9196935
#>   3.981072e-07  0.9952404  0.9196935
#>   6.309573e-07  0.9952404  0.9196935
#>   1.000000e-06  0.9952404  0.9196935
#>   1.584893e-06  0.9952404  0.9196935
#>   2.511886e-06  0.9952404  0.9196935
#>   3.981072e-06  0.9952404  0.9196935
#>   6.309573e-06  0.9952404  0.9196935
#>   1.000000e-05  0.9952404  0.9196935
#>   1.584893e-05  0.9952088  0.9190378
```

```
#>   2.511886e-05  0.9952562  0.9197413
#>   3.981072e-05  0.9951297  0.9172786
#>   6.309573e-05  0.9951772  0.9177721
#>   1.000000e-04  0.9950823  0.9157747
#>   1.584893e-04  0.9950032  0.9139437
#>   2.511886e-04  0.9948451  0.9105198
#>   3.981072e-04  0.9942284  0.8985014
#>   6.309573e-04  0.9936908  0.8875188
#>   1.000000e-03  0.9928369  0.8697187
#>   1.584893e-03  0.9914138  0.8400304
#>   2.511886e-03  0.9893740  0.7947263
#>   3.981072e-03  0.9862115  0.7177919
#>   6.309573e-03  0.9832229  0.6358503
#>   1.000000e-02  0.9753641  0.3656302
#>
#> Tuning parameter 'alpha' was held constant at a value of 1
#> Accuracy was used to select the optimal model using the largest value.
#> The final values used for the model were alpha = 1 and lambda = 2.511886e-05.
```

I extract the optimal tuning parameter produced from cross validation.

```
# final model
lasso$bestTune
```

```
#>    alpha       lambda
#> 53     1 2.511886e-05
```

The final lasso model includes all three predictors with a tuning parameter of 2.511886e-05, which results in a training accuracy of 0.9952562. The model inputs are represented by the following notation:
$features: Red, Green, Blue | lambda = 2.511886e-05$

4.6.2 Results

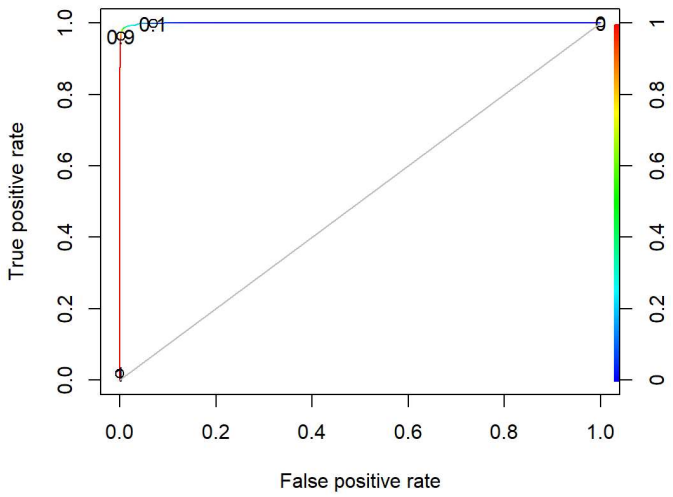
Finally, I test the lasso model on the holdout set and generate the estimated probabilities. Additionally, I generate an ROC curve.

```
# select best threshold value
pred.prob <- predict(lasso, holdout, type='prob')

predob <- ROCR::prediction(pred.prob$Tarp, holdout$Is.BTarp, label.ordering=c('Other', 'Tarp'))

model.roc <- ROCR::performance(predob, measure='tpr', x.measure='fpr')

lasso.ROC = plot(model.roc, colorize=T, print.cutoffs.at=c(0, 0.1, 0.9, 1.0))+
  lines(x=c(0,1), y=c(0,1), col='grey')
```



The lasso regression model has a ROCR curve that bends very close to the point [0,1]. This is a very accurate classification model.

```
# get AUC value for this curve
auc.calc = performance(predob, "auc")
auc.lasso = auc.calc@y.values
auc.lasso
```

```
#> [[1]]
#> [1] 0.9994511
```

The AUC of 0.9994511 indicates a very accurate model, supporting the interpretation of the ROCR Curve. I now generate the optimal probability thresholds for the lasso model.

```
# use thresholder function to analyze and select the best threshold value
# primary goal is to maximize TPR and minimize FNR
thresh.stats = thresholder(lasso,
                           threshold = seq(0.05, 0.95, by = 0.05),
                           statistics = 'all')

thresh.stats$FPR = 1-thresh.stats$Sensitivity

thresh.stats$ERR.Rate = 1-thresh.stats$Accuracy
```

Next, I create the performance metrics in table format according to probability threshold.

```
# create table to view various threshold metrics for threshold selection
lasso.thresh = thresh.stats %>%
    dplyr::select("prob_threshold", "ERR.Rate", "Accuracy", "Sensitivity", "FPR", "Precision")%>%
    knitr::kable(digits=4) %>%
    kableExtra::kable_styling(full_width=FALSE)

# Highest Accuracy
lasso.Acc = thresh.stats %>% dplyr::slice_max(Accuracy) %>%
    dplyr::select("prob_threshold", "ERR.Rate", "Accuracy", "Sensitivity", "FPR", "Precision")%>%
    knitr::kable(digits=4) %>%
    kableExtra::kable_styling(full_width=FALSE)
lasso.Acc
```

prob_threshold	ERR.Rate	Accuracy	Sensitivity	FPR	Precision
0.75	0.0043	0.9957	0.9984	0.0016	0.9971

```
# Highest TPR
lasso.Sen = thresh.stats %>% dplyr::slice_max(Sensitivity) %>%
    dplyr::select("prob_threshold", "ERR.Rate", "Accuracy", "Sensitivity", "FPR", "Precision")%>%
    knitr::kable(digits=4) %>%
    kableExtra::kable_styling(full_width=FALSE)
lasso.Sen
```

prob_threshold	ERR.Rate	Accuracy	Sensitivity	FPR	Precision
0.05	0.0074	0.9926	0.9998	2e-04	0.9927

```
# Lowest FPR
lasso.FPR = thresh.stats %>% dplyr::slice_min(FPR) %>%
    dplyr::select("prob_threshold", "ERR.Rate", "Accuracy", "Sensitivity", "FPR", "Precision")%>%
    knitr::kable(digits=4) %>%
    kableExtra::kable_styling(full_width=FALSE)
lasso.FPR
```

prob_threshold	ERR.Rate	Accuracy	Sensitivity	FPR	Precision
0.05	0.0074	0.9926	0.9998	2e-04	0.9927

```
# Highest Precision
lasso.Prec = thresh.stats %>% dplyr::slice_max(Precision) %>%
    dplyr::select("prob_threshold", "ERR.Rate", "Accuracy", "Sensitivity", "FPR", "Precision")%>%
    knitr::kable(digits=4) %>%
    kableExtra::kable_styling(full_width=FALSE)
lasso.Prec
```

prob_threshold	ERR.Rate	Accuracy	Sensitivity	FPR	Precision
0.95	0.0132	0.9868	0.9871	0.0129	0.9992

```
# extract optimal sensitivity values for results table
las.sen.tab = thresh.stats %>% dplyr::slice_max(Sensitivity) %>%
     dplyr::select("prob_threshold", "ERR.Rate", "Accuracy", "Sensitivity", "FPR", "Precision")
las.sen.tab = las.sen.tab[1,]
row.names(las.sen.tab) = c("Lasso")
```

Accuracy favors a higher threshold of 0.75. Precision favors a very high threshold of 0.95, whereas Sensitivity and FPR favor a lower threshold between 0.05. I choose the 0.05 threshold as it has the best Sensitivity value of 0.9998.

4.7 Support Vector Machine Model

Now, I build a Support Vector Machine model (SVM).

4.7.1 Training and Validation

Given the shape of the data based on inference from the exploratory data analysis and the superior performance of the Logistic and LDA models, I will train an SVM using linear kernels.

I will use the Caret package to build this model. I will need to specify a tune grid since this model will have a tuning parameter, cost. I will have the Caret package training a variety of values for cost, ranging from 1-10.

```
set.seed(7)
# Define the grid of tuning parameters to explore

tuneGrid <- expand.grid(cost=c(1,2,3,4,5,6,7,8,9,10))

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

svm <- train(Is.BTarp ~ ., data=train, method='svmLinear2',
               trControl=trControl, tuneGrid=tuneGrid)
svm
```

```
#> Support Vector Machines with Linear Kernel
#>
#> 63241 samples
#>     3 predictor
#>     2 classes: 'Other', 'Tarp'
#>
#> No pre-processing
#> Resampling: Cross-Validated (10 fold)
#> Summary of sample sizes: 56916, 56917, 56917, 56917, 56917, 56917, ...
#> Resampling results across tuning parameters:
#>
#>   cost  Accuracy   Kappa
#>    1    0.9950032  0.9143405
#>    2    0.9951297  0.9168894
#>    3    0.9951930  0.9181135
#>    4    0.9952246  0.9187810
#>    5    0.9952720  0.9197115
#>    6    0.9952879  0.9199981
#>    7    0.9953195  0.9205834
#>    8    0.9952879  0.9201261
#>    9    0.9952720  0.9198807
#>   10    0.9952720  0.9198774
#>
#> Accuracy was used to select the optimal model using the largest value.
#> The final value used for the model was cost = 7.
```

I extract the tuned parameters from the model.

```
# final model
svm$finalModel
```

```
#>
#> Call:
#> svm.default(x = as.matrix(x), y = y, kernel = "linear", cost = param$cost,
#>     probability = classProbs)
#>
#>
#> Parameters:
#>    SVM-Type:  C-classification
#>  SVM-Kernel:  linear
#>        cost:  7
#>
#> Number of Support Vectors:  743
```

The final SVM model produced by cross validation, according to the output, used 743 support vectors and a cost of 7. This results in a training accuracy of 0.9953195. Now it is time to test the model against the holdout data and produce an ROC curve to evaluate.

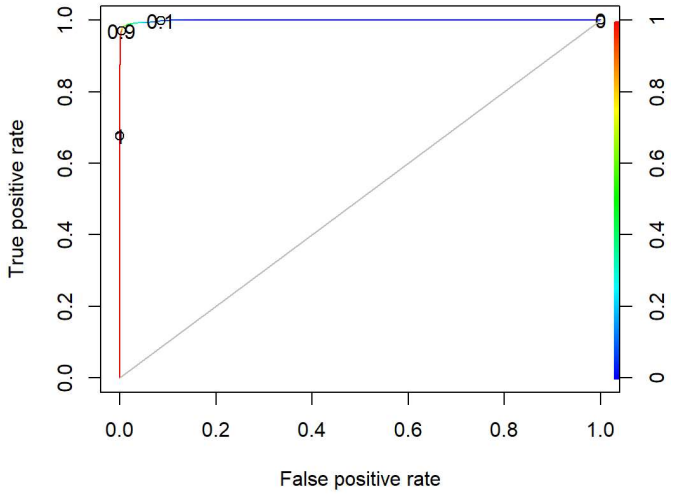
4.7.2 Results

```
# select best threshold value
pred.prob <- predict(svm, holdout, type='prob')

predob <- ROCR::prediction(pred.prob$Tarp, holdout$Is.BTarp, label.ordering=c('Other', 'Tarp'))

model.roc <- ROCR::performance(predob, measure='tpr', x.measure='fpr')

svm.ROC = plot(model.roc, colorize=T, print.cutoffs.at=c(0, 0.1, 0.9, 1.0))+
  lines(x=c(0,1), y=c(0,1), col='grey')
```



The ROC curve bends toward the optimal point, [0,1]. This SVM model is very accurate based on inspection of this graph.

```
# get AUC value for this curve
auc.calc = performance(predob, "auc")
auc.svm = auc.calc@y.values
auc.svm
```

```
#> [[1]]
#> [1] 0.9991305
```

SVM has an AUC of 0.9991305, extraordinarily close to 1. This is the second best model thus far in the analysis according to AUC. Next, I produce the tables of performance metrics to help determine the optimal probability threshold.

```
# use thresholder function to analyze and select the best threshold value
# primary goal is to maximize TPR and minimize FNR
thresh.stats = thresholder(svm,
                           threshold = seq(0.05, 0.95, by = 0.05),
                           statistics = 'all')

thresh.stats$FPR = 1-thresh.stats$Sensitivity

thresh.stats$ERR.Rate = 1-thresh.stats$Accuracy
```

```
# create table to view various threshold metrics for threshold selection
svm.thresh = thresh.stats %>%
    dplyr::select("prob_threshold", "ERR.Rate", "Accuracy", "Sensitivity", "FPR", "Precision")%>%
    knitr::kable(digits=4) %>%
    kableExtra::kable_styling(full_width=FALSE)

# Highest Accuracy
svm.Acc = thresh.stats %>% dplyr::slice_max(Accuracy) %>%
    dplyr::select("prob_threshold", "ERR.Rate", "Accuracy", "Sensitivity", "FPR", "Precision")%>%
    knitr::kable(digits=4) %>%
    kableExtra::kable_styling(full_width=FALSE)
svm.Acc
```

prob_threshold	ERR.Rate	Accuracy	Sensitivity	FPR	Precision
0.8	0.0041	0.9959	0.9982	0.0018	0.9976

```
# Highest TPR
svm.Sen = thresh.stats %>% dplyr::slice_max(Sensitivity) %>%
    dplyr::select("prob_threshold", "ERR.Rate", "Accuracy", "Sensitivity", "FPR", "Precision")%>%
    knitr::kable(digits=4) %>%
    kableExtra::kable_styling(full_width=FALSE)
svm.Sen
```

prob_threshold	ERR.Rate	Accuracy	Sensitivity	FPR	Precision
0.05	0.0065	0.9935	0.9997	3e-04	0.9936

```
# Lowest FPR
svm.FPR = thresh.stats %>% dplyr::slice_min(FPR) %>%
    dplyr::select("prob_threshold", "ERR.Rate", "Accuracy", "Sensitivity", "FPR", "Precision")%>%
    knitr::kable(digits=4) %>%
    kableExtra::kable_styling(full_width=FALSE)
svm.FPR
```

prob_threshold	ERR.Rate	Accuracy	Sensitivity	FPR	Precision
0.05	0.0065	0.9935	0.9997	3e-04	0.9936

```
# Highest Precision
svm.Prec = thresh.stats %>% dplyr::slice_max(Precision) %>%
    dplyr::select("prob_threshold", "ERR.Rate", "Accuracy", "Sensitivity", "FPR", "Precision")%>%
    knitr::kable(digits=4) %>%
    kableExtra::kable_styling(full_width=FALSE)
svm.Prec
```

prob_threshold	ERR.Rate	Accuracy	Sensitivity	FPR	Precision
0.95	0.0165	0.9835	0.9838	0.0162	0.9991

```
# extract optimal sensitivity values for results table
svm.sen.tab = thresh.stats %>% dplyr::slice_max(Sensitivity) %>%
     dplyr::select("prob_threshold", "ERR.Rate", "Accuracy", "Sensitivity", "FPR", "Precision")
svm.sen.tab = svm.sen.tab[1,]
row.names(svm.sen.tab) = c("SVM")
```

Error Rate and Accuracy favor a probability threshold of 0.8. Precision favors a probability threshold of 0.95. However, I choose a probability threshold of 0.05, which has the highest Sensitivity of 0.9997. FPR also favors 0.05.

4.8 Random Forest

The final step in the model-building process is to create a random forest model.

4.8.1 Training and Validation

I will again use the Caret package to fit a model. The tuning parameter for the random forest model is “mtry,” which is the number of features considered at each decision split in the decision tree. Since the data only contain three parameters, I will simply set the mtry parameter to 3. If the data had many features, e.g. 12, it would be advised to select a proper mtry value through cross validation. Then, I extract the performance information from cross validation.

```
set.seed(7)
# Define the grid of tuning parameters to explore
tuneGrid <- data.frame(mtry=3)

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

rforest <- train(Is.BTarp ~ ., data=train, method='rf',
                 trControl=trControl, tuneGrid=tuneGrid)
rforest
```

```
#> Random Forest
#>
#> 63241 samples
#>     3 predictor
#>     2 classes: 'Other', 'Tarp'
#>
#> No pre-processing
#> Resampling: Cross-Validated (10 fold)
#> Summary of sample sizes: 56916, 56917, 56917, 56917, 56917, 56917, ...
#> Resampling results:
#>
#>   Accuracy   Kappa
#>   0.9969482  0.9502371
#>
#> Tuning parameter 'mtry' was held constant at a value of 3
```

The random forest model correctly classified 0.9969482 proportion of the data. This is a great start, but ultimatley the testing accruacy will be more important. The optimal tuning parameter is 3 since I tuned the model to hold mtry constant at this value.

4.8.2 Results

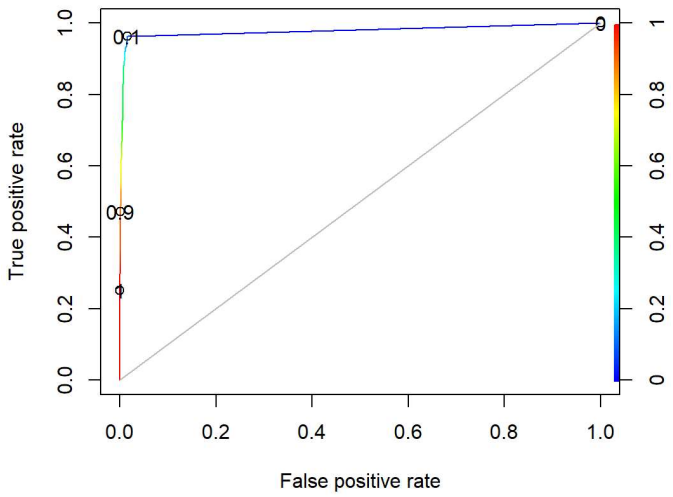
Finally, I test the random forest model on the holdout set and generate the estimated probabilities.

```
# select best threshold value
pred.prob <- predict(rforest, newdata=holdout, type='prob')

predob <- ROCR::prediction(pred.prob$Tarp, holdout$Is.BTarp, label.ordering=c('Other', 'Tarp'))

model.roc <- ROCR::performance(predob, measure='tpr', x.measure='fpr')

rforest.ROC = plot(model.roc, colorize=T, print.cutoffs.at=c(0, 0.1, 0.9, 1.0))+
  lines(x=c(0,1), y=c(0,1), col='grey')
```



The random forest model, joining its peers, has a ROCR curve that bends near the point [0,1]. This is a very accurate classification model.

```
# get AUC value for this curve
auc.calc = performance(predob, "auc")
auc.rforest = auc.calc@y.values
auc.rforest
```

```
#> [[1]]
#> [1] 0.9777202
```

The AUC of 0.977720 indicates a very accurate model, supporting the interpretation of the ROCR Curve. I now will generate the optimal probability thresholds for the random forest model.

```
# use thresholder function to analyze and select the best threshold value
# primary goal is to maximize TPR and minimize FNR
thresh.stats = thresholder(rforest,
                           threshold = seq(0.05, 0.95, by = 0.05),
                           statistics = 'all')

thresh.stats$FPR = 1-thresh.stats$Sensitivity

thresh.stats$ERR.Rate = 1-thresh.stats$Accuracy
```

Now I create tables with relevant performance metrics by probability thresholds.

```
# create table to view various threshold metrics for threshold selection
rforest.thresh = thresh.stats %>%
    dplyr::select("prob_threshold", "ERR.Rate", "Accuracy", "Sensitivity", "FPR", "Precision")%>%
    knitr::kable(digits=4) %>%
    kableExtra::kable_styling(full_width=FALSE)

# Highest Accuracy
rforest.Acc = thresh.stats %>% dplyr::slice_max(Accuracy) %>%
    dplyr::select("prob_threshold", "ERR.Rate", "Accuracy", "Sensitivity", "FPR", "Precision")%>%
    knitr::kable(digits=4) %>%
    kableExtra::kable_styling(full_width=FALSE)
rforest.Acc
```

prob_threshold	ERR.Rate	Accuracy	Sensitivity	FPR	Precision

prob_threshold	ERR.Rate	Accuracy	Sensitivity	FPR	Precision
0.6	0.003	0.997	0.9983	0.0017	0.9986

```
# Highest TPR
rforest.Sen = thresh.stats %>% dplyr::slice_max(Sensitivity) %>%
    dplyr::select("prob_threshold", "ERR.Rate", "Accuracy", "Sensitivity", "FPR", "Precision")%>%
    knitr::kable(digits=4) %>%
    kableExtra::kable_styling(full_width=FALSE)
rforest.Sen
```

prob_threshold	ERR.Rate	Accuracy	Sensitivity	FPR	Precision
0.05	0.0053	0.9947	0.9998	2e-04	0.9947

```
# Lowest FPR
rforest.FPR = thresh.stats %>% dplyr::slice_min(FPR) %>%
    dplyr::select("prob_threshold", "ERR.Rate", "Accuracy", "Sensitivity", "FPR", "Precision")%>%
    knitr::kable(digits=4) %>%
    kableExtra::kable_styling(full_width=FALSE)
rforest.FPR
```

prob_threshold	ERR.Rate	Accuracy	Sensitivity	FPR	Precision
0.05	0.0053	0.9947	0.9998	2e-04	0.9947

```
# Highest Precision
rforest.Prec = thresh.stats %>% dplyr::slice_max(Precision) %>%
    dplyr::select("prob_threshold", "ERR.Rate", "Accuracy", "Sensitivity", "FPR", "Precision")%>%
    knitr::kable(digits=4) %>%
    kableExtra::kable_styling(full_width=FALSE)
rforest.Prec
```

prob_threshold	ERR.Rate	Accuracy	Sensitivity	FPR	Precision
0.95	0.0049	0.9951	0.9954	0.0046	0.9995

```
# extract optimal sensitivity values for results table
rf.sen.tab = thresh.stats %>% dplyr::slice_max(Sensitivity) %>%
     dplyr::select("prob_threshold", "ERR.Rate", "Accuracy", "Sensitivity", "FPR", "Precision")
rf.sen.tab = rf.sen.tab[1,]
row.names(rf.sen.tab) = c("Random Forest")
```

The Error Rate and Accuracy favor a threshold of 0.6. Precision favors a threshold of 0.95. I choose a threshold of 0.05 as it has the highest Sensitivity of 0.9997.

5 Table of Results

5.1 Final Threshold Selection and Table

I base the final threshold selection on the respective Sensitivity values based on model testing. I contend that the proportion of true positives in this analysis is the most important threshold metric because the rescue workers must locate as many disaster survivors as humanly possible. A high Sensitivity means that the survivor locations identified by the algorithm are almost certain to matriculate into real survivor locations. This level of certainty is corroborated by a series of Sensitivity values that are either 1, or nearly 1, as seen in the performance table below.

```
# merge the data frames
log.lda.tab = merge(log.sen.tab,lda.sen.tab,by=c("prob_threshold","ERR.Rate","Accuracy","Sensitivity","FPR","Precision"),all.x=TRUE, all.y=TRUE)

build.tab2 = merge(log.lda.tab,qda.sen.tab, by=c("prob_threshold","ERR.Rate","Accuracy","Sensitivity","FPR","Precision"),all.x=TRUE, all.y=TRUE)

build.tab3 = merge(build.tab2,knn.sen.tab, by=c("prob_threshold","ERR.Rate","Accuracy","Sensitivity","FPR","Precision"),all.x=TRUE, all.y=TRUE)

build.tab4 = merge(build.tab3,las.sen.tab, by=c("prob_threshold","ERR.Rate","Accuracy","Sensitivity","FPR","Precision"),all.x=TRUE, all.y=TRUE)

build.tab5 = merge(build.tab4,svm.sen.tab, by=c("prob_threshold","ERR.Rate","Accuracy","Sensitivity","FPR","Precision"),all.x=TRUE, all.y=TRUE)

all.sen.tab = merge(build.tab5,rf.sen.tab, by=c("prob_threshold","ERR.Rate","Accuracy","Sensitivity","FPR","Precision"),all.x=TRUE, all.y=TRUE)

row.names(all.sen.tab) = c("Logistic","LDA","QDA","KNN; K=5","Lasso","SVM; Kernel=Linear","RForest")

final.tab = all.sen.tab %>%
  kable(digits=6) %>%
  kableExtra::kable_styling(full_width=FALSE)

final.tab
```

	prob_threshold	ERR.Rate	Accuracy	Sensitivity	FPR	Precision
Logistic	0.05	0.004538	0.995462	0.999673	0.000327	0.995658
LDA	0.05	0.005297	0.994703	0.999837	0.000163	0.994720
QDA	0.05	0.006515	0.993485	0.999673	0.000327	0.993637
KNN; K=5	0.05	0.006752	0.993248	0.999706	0.000294	0.993363
Lasso	0.05	0.007353	0.992647	0.999755	0.000245	0.992703
SVM; Kernel=Linear	0.05	0.008143	0.991857	1.000000	0.000000	0.991660
RForest	0.05	0.015401	0.984599	0.993629	0.006371	0.990492

5.2 AUC Comparison

I concatenate all of the AUC values into a table below for simpler comparison. The logistic classification model has the highest AUC, as seen in the “auc.log” column. You can see below that the lasso regression has the highest AUC value of ~0.99945, indicated in the ‘auc.lasso’ row

```
# grid these density plots
auc.tab = rbind(auc.log, auc.lda, auc.qda, auc.knn, auc.lasso, auc.svm, auc.rforest)

fin.auc = auc.tab %>%
  kable(digits=5) %>%
  kableExtra::kable_styling(full_width=FALSE)

fin.auc
```

auc.log	0.999387651587227
auc.lda	0.992421154674878

auc.qda	0.992236582695111
auc.knn	0.95985933686015
auc.lasso	0.999451145284436
auc.svm	0.999130543702735
auc.rforest	0.977720212704172

6 Conclusions

6.1 1. The KNN Model Performs Best on Cross Validation Data

I use accuracy to determine which model performs best on the cross validation during cross validation. The KNN model with K=5 had the highest cross validation accuracy of 0.9972644. Frankly, accuracy was the best available performance metric based on how the models were trained by the Caret package. While it is possible to direct Caret to produce a final model based on AUC, all of the model classes chosen by accuracy performed well in testing, so I did not see the need to retroactively change how final models were selected. Perhaps this is something to consider in future analyses. I do concede that this could improve comparative consistency. Yet again, the final models were all sufficient upon testing.

6.2 2. The Lasso Model Performs Best on Holdout Data

The Lasso model with a lambda value of 2.511886e-05 had the best performance on the holdout data. I base this conclusion on the Lasso's AUC of ~0.99945, which was the highest of all the models. I used AUC to make this decision as it measures the overall predictive ability of each model across all probability thresholds.I contend that AUC is the most comprehensive method of comparing classification models after testing.

6.3 3. The Lasso Model Should be Used to Locate Survivors

Ultimately, a model's performance on testing data is more important than its performance on cross validation data. Thus, the model I recommend to identify survivors' locations is the Lasso regression model with a probability threshold of 0.05. The Lasso model produced the highest AUC value of ~0.99945, although, all seven model families were highly accurate – not one having an AUC less than 0.95. As I previously mentioned, I use AUC as the final determinant of the most effective model since it evaluates the overall predictive ability of each model across probability thresholds.

I used Sensitivity to determine the appropriate probability threshold once I selected the best model class. As I stated throughout this analysis, Sensitivity is the most important of the probability threshold metrics in this context. It is absolutely essential that the model algorithm accurately identifies survivor locations, and having a True Positive Rate of 0.9997 indicates that there is almost a 100% chance that the model correctly identifies survivors, which could save lives.

6.4 4. Large Data Sets with Balanced Subsets Yield Accurate Results

This analysis confirms much of what most modern statistics tells us: larger data sets tend to yield more accurate results. My previous experience in modeling was primarily for purposes of statistical inference. I saw first hand how large data sets yielded more accurate conclusions on relationships between variables in the data. This analysis provided first-hand experience in how a very large, well-balanced data set can improve classification and predictive abilities. Each class of model that I generated yielded extraordinarily accurate predictions – having AUC values close to 1. I attribute this to robust cross-validation, which was bolstered by use of more than 60,000 training observations, and more than 2 million testing observations.

Additionally, the accuracy transcended the model family, further supporting the fact that there was enough data to establish performance parity between the models. For example, the LDA and QDA models have a difference in AUC of 0.0001846. The predictive difference between these models is functionally zero, in spite of the fact that LDA assumes that the covariance of each class is the same, whereas QDA makes no covariance assumptions. In many cases, this can lead to differences in model performance depending on the form and amount of the data. In smaller data sets, it is likely that there would be a greater contrast in the predictive ability, where one of the methods is conspicuously superior. In this case, there was enough data to mitigate the models' fundamental differences.

6.5 5. Simpler Models Should be Leveraged Where Practicable For Computational Efficiency

This analysis required a considerably large amount of data, with a training set containing 63,241 observations, and a holdout data set with 2,008,623 observations. Training and testing the various models required significant computational resources. Some models, expressly the SVM, Random Forest, and KNN models, took roughly 15-20 minutes of run time to train and test. While the overall processing speed depends on the type of device the researcher uses, it is evident that more complex models require more computing than simpler ones.

Sometimes, this additional computation that comes with increased model complexity is unnecessary when it is possible to build a simpler model that performs just as well as a more complex one. For example, in this analysis, the Logistic Regression had a higher AUC than the Random Forest model, despite taking less than half of the time to validate and test. A key takeaway from this analysis is that simpler models should be preferred to more complex ones to improve computational efficiency, especially when simpler models outperform more complex ones. Evaluating the data's form during Exploratory Data Analysis can help researchers identify whether a complex model is necessary, and if not, this can preserve computational resources.

6.6 6. These Models Could Literally Save Lives

Perhaps the most important conclusion is that any of the models from my analysis could save lives. Each has the ability to rapidly evaluate data taken from aerial imaging and make a determination of survivor location in relatively little time. In a situation as we witnessed in Haiti, every moment is critical. Each passing hour without access to food and water literally means life and death for survivors, especially for those who sustained injuries. If rescue entities had to rely solely on humans to manually search the image data, many people would die. Instead, rescuers could leverage my logistic regression model to locate survivors with 99% accuracy, and this is encouraging. While the technical and conceptual foundations of statistical learning are important, it is the capability to preserve, procure, and further humanity that really gives this field its significance. This analysis – with the goal of locating disaster victims in record time – serves as an excellent example of the power of statistical learning.