Disaster Relief Project: Part 2

Andres Izquierdo

May 10, 2022

Contents

SYS 6018 | Spring 2022 | University of Virginia

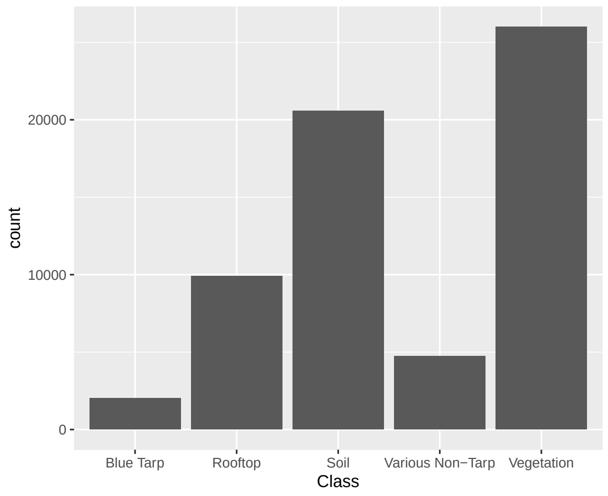
Introduction

This project will be implementing 4 models (Logistic Regression, KNN (K-nearest neighbor), Penalized Logistic Regression (elastic net penalty), and SVM (Support Vector Machines)) as a method to classify images. The images under question come from after natural disaster, in this case following the destruction of the earthquake in Haiti in 2010. Classification will be used to identify makeshift shelters using blue tarps following natural disasters to locate displaced people to get them food and water. Using data mining models they are able to classify pixels faster and accurately to get rescue workers to those locations in time.

Training Data / EDA

```r
# Load Required Packages
library(tidyverse)
library(ISLR)
library(caret)
library(FNN)
library(broom)
library(yardstick)
library(glmnet)
library(e1071)
library(gbm)
library(randomForest)
library(scales)
library(caTools)
library(MASS)
library(ROCR)

Training.Data <- read.csv("~/UVA SYS ME/SYS 6018 Data Mining/Project/HaitiTraining.csv")
calc_acc <- function(actual, predicted) {
  mean(actual == predicted)
}
```

```r
# plot to see count of all the classes
ggplot(data = Training.Data) +
  geom_bar(mapping = aes(x = Class))
```



```r
# Blue Tarp is the least amount of pixels of any of the classes.
Training.Data %>%
  count(Class)
```

```
#>              Class     n
#> 1        Blue Tarp  2022
#> 2          Rooftop  9903
#> 3             Soil 20566
#> 4 Various Non-Tarp  4744
#> 5       Vegetation 26006
```

```r
# Number of classes per each pixel.
by(Training.Data, Training.Data$Class, summary)
```

```
#> Training.Data$Class: Blue Tarp
#>     Class                Red             Green             Blue
#>  Length:2022        Min.   : 93.0   Min.   : 92.0   Min.   : 92.0
#>  Class :character   1st Qu.:147.0   1st Qu.:160.0   1st Qu.:175.2
#>  Mode  :character   Median :168.0   Median :187.0   Median :216.0
#>                     Mean   :169.7   Mean   :186.4   Mean   :205.0
#>                     3rd Qu.:193.0   3rd Qu.:219.0   3rd Qu.:255.0
#>                     Max.   :255.0   Max.   :255.0   Max.   :255.0
#> ------------------------------------------------------------
#> Training.Data$Class: Rooftop
#>     Class                Red            Green            Blue
#>  Length:9903        Min.   : 98.0   Min.   : 87.0   Min.   : 87.0
#>  Class :character   1st Qu.:163.0   1st Qu.:146.0   1st Qu.:123.0
#>  Mode  :character   Median :200.0   Median :191.0   Median :171.0
#>                     Mean   :195.4   Mean   :183.9   Mean   :162.8
#>                     3rd Qu.:222.0   3rd Qu.:215.0   3rd Qu.:192.0
#>                     Max.   :255.0   Max.   :255.0   Max.   :243.0
#> ------------------------------------------------------------
#> Training.Data$Class: Soil
#>     Class                Red            Green            Blue
#>  Length:20566       Min.   :108.0   Min.   : 93.0   Min.   : 83.0
#>  Class :character   1st Qu.:255.0   1st Qu.:215.0   1st Qu.:164.0
#>  Mode  :character   Median :255.0   Median :235.0   Median :178.0
#>                     Mean   :247.9   Mean   :227.5   Mean   :176.7
#>                     3rd Qu.:255.0   3rd Qu.:251.0   3rd Qu.:191.0
#>                     Max.   :255.0   Max.   :255.0   Max.   :255.0
#> ------------------------------------------------------------
#> Training.Data$Class: Various Non-Tarp
#>     Class                Red            Green            Blue
#>  Length:4744        Min.   : 71.0   Min.   : 66.0   Min.   : 60.0
#>  Class :character   1st Qu.:124.0   1st Qu.:110.0   1st Qu.: 88.0
#>  Mode  :character   Median :192.0   Median :160.0   Median :124.0
#>                     Mean   :184.8   Mean   :168.8   Mean   :140.9
#>                     3rd Qu.:255.0   3rd Qu.:247.0   3rd Qu.:191.0
#>                     Max.   :255.0   Max.   :255.0   Max.   :255.0
#> ------------------------------------------------------------
#> Training.Data$Class: Vegetation
#>     Class                Red          Green             Blue
#>  Length:26006       Min.   : 48   Min.   : 48.00   Min.   : 44.00
#>  Class :character   1st Qu.: 68   1st Qu.: 68.00   1st Qu.: 53.00
#>  Mode  :character   Median : 75   Median : 75.00   Median : 58.00
#>                     Mean   : 79   Mean   : 78.45   Mean   : 60.92
#>                     3rd Qu.: 90   3rd Qu.: 87.00   3rd Qu.: 69.00
#>                     Max.   :145   Max.   :153.00   Max.   :101.00
```

```
# looking at all the summaries of each Class.
```

Looking at the EDA done on this data set I can see now that the least amount of classified pixels in this dataset is blue tarps with vegetation and soil being the most classified pixels. This may present an imbalance when it comes to modeling the data as we want to specifically classify for blue tarps and there are not a lot of examples. Looking at the summary statistics of each class I can see that all the pixels are classified using the same scale so no standardization nor normalization will have to be made upon this data.

Model Training

Set-up

```
# Setting up folds for cross-validation.
Training.Data[Training.Data == "Blue Tarp"] <- "1"
Training.Data[Training.Data == "Rooftop"] <- "0"
Training.Data[Training.Data == "Soil"] <- "0"
Training.Data[Training.Data == "Various Non-Tarp"] <- "0"
Training.Data[Training.Data == "Vegetation"] <- "0"
Training.Data$Class <- as.numeric(Training.Data$Class)

CV_perf_table <- data.frame(matrix(ncol = 8, nrow = 9))
colnames(CV_perf_table) <- c("Model", "Tuning", "AUROC", "Threshold", "Accuracy", "TPR", "FPR", "Precis
CV_perf_table[1, 1] = "Log Reg"
CV_perf_table[2, 1] = "KNN"
CV_perf_table[3, 1] = "Penalized Log Reg"
CV_perf_table[4, 1] = "SVM"
CV_perf_table[5, 1] = "Naive Bayes"
CV_perf_table[6, 1] = "LDA"
CV_perf_table[7, 1] = "QDA"
CV_perf_table[8, 1] = "Random Forests"
CV_perf_table[9, 1] = "Boosted Trees"
```

Since in this project we are only interested in identifying Blue Tarp pixels I have transformed the Class column into a binary indicators with Blue Tarp being "1" and all other classes being "0" since we have no interest in identifying them.

Logistic Regression

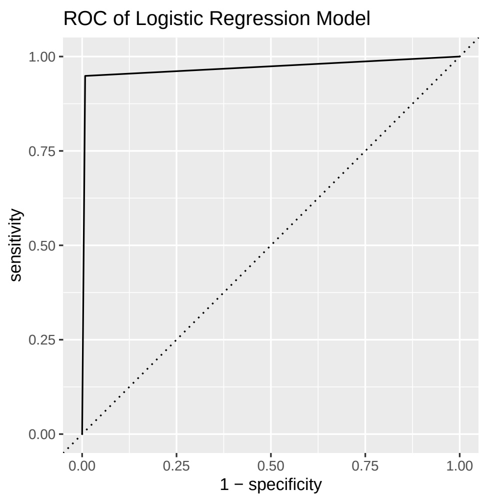
```
set.seed(1)
# Creating a training data and testing data split.
Train <- createDataPartition(Training.Data$Class, p=0.75, list=FALSE)
training.LR <- Training.Data[ Train, ]
testing.LR <- Training.Data[ -Train, ]

# Setting Cross-Validation to 10-folds
ctrl <- trainControl(method = "cv", number = 10, savePredictions = TRUE)

# Training Model with training dats using Logistic Regression.
mod_fitcv <- train(Class~., data=training.LR, method="glm", family="binomial", trControl = ctrl)
```

```
#> Warning in train.default(x, y, weights = w, ...): You are trying to do
#> regression and your outcome only has two possible values Are you trying to do
#> classification? If so, use a 2 level factor as your outcome column.
```

```
#> Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
#> Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
#> Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

#> Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

#> Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

#> Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

#> Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

#> Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

#> Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

#> Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

#> Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```r
# Using trained model to predict values.
pred.LR <- predict(mod_fitcv, newdata = testing.LR)
# Setting Threshhold for predicted values.
pred.LR <- ifelse(pred.LR > 0.1,1,0)
# Creating Confusion Matrix with predicted and test values.
conf_matrix.LR <- caret::confusionMatrix(table(pred.LR,(testing.LR$Class)))
# Creating ROC Curve built from out of sample data
ROC.LR = tibble(truth = factor(testing.LR$Class, levels =c(1,0)), pred.LR) %>%
  yardstick::roc_curve(truth, pred.LR)

ROC.LR %>%
ggplot(aes(1-specificity, sensitivity)) + geom_line() +
geom_abline(lty=3) +
coord_equal() + ggtitle("ROC of Logistic Regression Model")
```



```r
# Computing AUROC for the Predicted values for Logistic Regression.
AUROC.LR <- yardstick::roc_auc_vec(factor(testing.LR$Class, 1:0), pred.LR)
```

```
# Logistic Regression Accuracy.
acc.LR <- conf_matrix.LR$overall
acc.LR <- as.numeric(acc.LR[1])

# Logistic Regression TPR and FPR.
TPR.LR <- ROC.LR[3,3]
FPR.LR <- ROC.LR[3,2]

# Logistic Regression Precision.
precision.LR <- (conf_matrix.LR$table[4]) / (conf_matrix.LR$table[4]+conf_matrix.LR$table[2])

CV_perf_table[1, 2] = "N/A"
CV_perf_table[1, 3] = AUROC.LR
CV_perf_table[1, 4] = "> 0.1"
CV_perf_table[1, 5] = acc.LR
CV_perf_table[1, 6] = TPR.LR
CV_perf_table[1, 7] = FPR.LR
CV_perf_table[1, 8] = precision.LR
```

KNN

```
set.seed(2)
# Setting up number of folds for Cross-Validation.
n.folds = 10
n = nrow(Training.Data)
fold = sample(rep(1:n.folds, length=n))
K = 1:50
MSE = matrix(NA, length(K), n.folds)
n.val = numeric(n.folds)

# For loop to compute K neighbors for each fold and compute the MSE for each iteration.
for(j in 1:n.folds){

  #-- Set training/val data
  val = which(fold == j)
  train = which(fold != j)
  n.val[j] = length(val)

  #-- fit set of knn models
  for(i in 1:length(K)){
    k = K[i]
    knn = FNN::knn.reg(train = Training.Data[train,, drop=FALSE],
                       y = Training.Data$Class[train],
                       test = Training.Data[val,, drop=FALSE],
                       k = k)
    r.val = Training.Data$Class[val]-knn$pred      # residuals on val data
    MSE[i, j] = mean(r.val^2)
  }
}

# Calculating MSE to find which K neighbor gives us the least amount of MSE.
```

```r
CV = (MSE %*% n.val)/n
SE = apply(MSE, 1, sd)/sqrt(n.folds)
results.knn = tibble(k = K, MSE = CV[,1], SE)
results.knn
```

```
#> # A tibble: 50 x 3
#>         k     MSE       SE
#>     <int>   <dbl>    <dbl>
#>  1      1 0.00251 0.000271
#>  2      2 0.00193 0.000192
#>  3      3 0.00182 0.000162
#>  4      4 0.00181 0.000166
#>  5      5 0.00173 0.000157
#>  6      6 0.00172 0.000153
#>  7      7 0.00172 0.000155
#>  8      8 0.00171 0.000152
#>  9      9 0.00169 0.000155
#> 10     10 0.00165 0.000146
#> # ... with 40 more rows
```

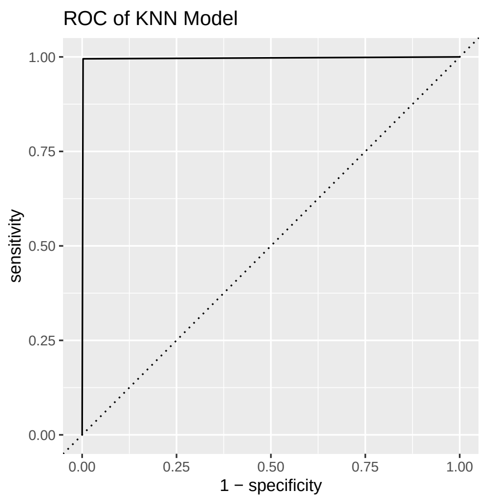
```r
opt.K <- results.knn %>%
  filter(min_rank(MSE) == 1)
opt.K
```

```
#> # A tibble: 1 x 3
#>       k     MSE       SE
#>   <int>   <dbl>    <dbl>
#> 1    10 0.00165 0.000146
```

```r
# 10 CV training Running model with optimal K neighbor.
K = 10 # folds
folds = rep(1:K, length=nrow(Training.Data)) %>% sample()
for(k in 1:K){
test = (folds == k)
train = (folds != k)
knn = FNN::knn.reg(train = Training.Data[train,],
                       y = Training.Data$Class[train],
                       test = Training.Data[test,],
                       k = opt.K$k)
}

# Setting Threshold to 0.1
pred.knn <- ifelse(knn$pred > 0.1,1,0)

# Creating Confusion Matrix
cm.knn <- table(Training.Data$Class[val],pred.knn)
cm.knn
```

```
#>    pred.knn
#>        0    1
#>   0 6109   15
#>   1    1  199
```

```r
# KNN Accuracy.
acc.knn <-calc_acc(actual = Training.Data$Class[val],
         pred.knn)

# Creating ROC Curve built from out of sample data
ROC.knn = tibble(truth = factor(Training.Data$Class[val], levels =c(1,0)), pred.knn) %>%
  yardstick::roc_curve(truth, pred.knn)

ROC.knn %>%
ggplot(aes(1-specificity, sensitivity)) + geom_line() +
geom_abline(lty=3) +
coord_equal() + ggtitle("ROC of KNN Model")
```



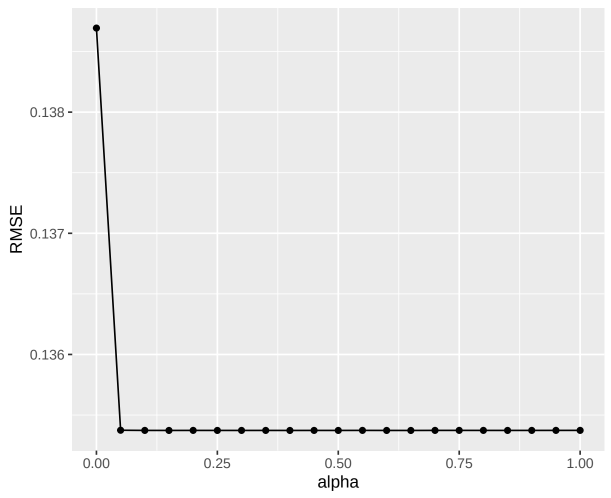
```r
Test.knn <- Training.Data[val,, drop=FALSE]

# Computing AUROC for the Predicted values for KNN.
AUROC.knn <- yardstick::roc_auc_vec(factor(Test.knn$Class, 1:0),  pred.knn)


# KNN Precision.
precision.knn <- (cm.knn[2,2]) / (cm.knn[2,1]+cm.knn[2,2])

# KNN TPR and FPR.
TPR.knn <- ROC.knn[3,3]
FPR.knn <- ROC.knn[3,2]

CV_perf_table[2, 2] = opt.K$k
CV_perf_table[2, 3] = AUROC.knn
CV_perf_table[2, 4] = "> 0.1"
CV_perf_table[2, 5] = acc.knn
CV_perf_table[2, 6] = TPR.knn
CV_perf_table[2, 7] = FPR.knn
CV_perf_table[2, 8] = precision.knn
```

Tuning Parameter k

Tuning parameter k neighbor for the KNN model was determined by running a for loop with 10-fold cross validation through a training model that cycled through different k values from 1 through 50 and testing the model on the validation set to calculate the MSE for the given k value. From the 50 k values that were tested the model with the lowest MSE was chosen as the most optimal k parameter.

Penalized Logistic Regression (ElasticNet)

```r
set.seed(3)
# Elastic Net
# Creating Testing and Training Data for Elastic Net.
Training.Data.EN <- Training.Data
n = nrow(Training.Data.EN)
n_hout = 500
hout = sample(n, size = n_hout)
hout_data = Training.Data.EN[hout,]
Training.Data.EN = Training.Data.EN[-hout,]
x.train = glmnet::makeX(Training.Data.EN[,2:4])
Y.train = Training.Data.EN$Class
hout_data_x = glmnet::makeX(hout_data[,2:4])
hout_data_y = hout_data$Class
alpha.seq = seq(0, 1, by=.05)

K = 10           # number of folds
folds = rep(1:K, length=nrow(x.train)) %>% sample() # make folds

est_rmse <- function(alpha, folds){
  min(cv.glmnet(x.train, Y.train, foldid = folds, alpha=alpha)$cvm) %>% sqrt()
}

RMSE2 = tibble()
for(i in 1:10){
  folds_new = sample(folds) # shuffle folds
  rmse = tibble(iter = i, alpha=alpha.seq,
                RMSE = sapply(alpha.seq, est_rmse, folds=folds_new))
  RMSE2 = bind_rows(RMSE2, rmse)
}

RMSE.mu = RMSE2 %>%
  group_by(alpha) %>% summarize(RMSE=mean(RMSE), RMSE.median = median(RMSE))

RMSE.mu %>%
  group_by(alpha) %>% summarize(RMSE=mean(RMSE)) %>%
  ggplot(aes(alpha, RMSE)) + geom_point() + geom_line()
```



```r
RMSE.mu %>% arrange(RMSE)
```

```
#> # A tibble: 21 x 3
#>     alpha  RMSE RMSE.median
#>     <dbl> <dbl>       <dbl>
#>  1  0.65 0.135       0.135
#>  2  0.4  0.135       0.135
#>  3  0.6  0.135       0.135
#>  4  0.15 0.135       0.135
#>  5  0.5  0.135       0.135
#>  6  0.3  0.135       0.135
#>  7  0.25 0.135       0.135
#>  8  0.85 0.135       0.135
#>  9  0.35 0.135       0.135
#> 10  0.45 0.135       0.135
#> # ... with 11 more rows
```

```r
(alpha.hat = RMSE.mu %>% slice_min(RMSE) %>% pull(alpha) )
```

```
#> [1] 0.65
```

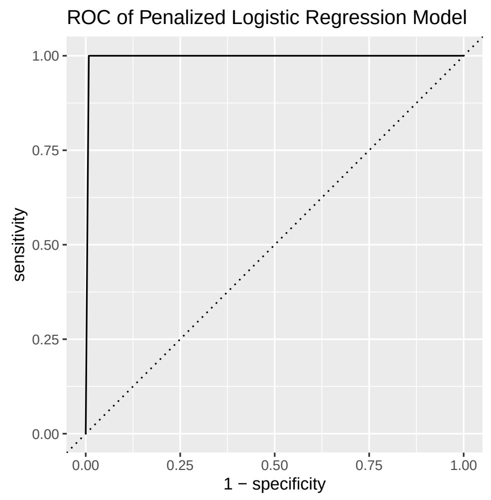
```r
alpha.hat # optimal alpha after tuning.
```

```
#> [1] 0.65
```

```r
# Setting up logistic regression model and training with training data.
a = alpha.hat  # setting alpha for elastic net
fit.enet = cv.glmnet(x = x.train, y=as.factor(Y.train), alpha=a, nfolds = 10, family="binomial")

(lambda.hat = fit.enet$lambda.min)
```

```
#> [1] 6.416034e-06
```

```
# Making predictions with model.
pred.EN = predict(fit.enet, s=lambda.hat, newx=hout_data_x, type="response")[,1]
# Threshold for predictions.
pred.EN <- ifelse(pred.EN > 0.1,1,0)

# Elasticnet Confusion Matrix
conf_matrix.EN <- confusionMatrix(as.factor(pred.EN),as.factor(hout_data_y))

# Elasticnet Accuracy
acc.EN <- calc_acc(actual = hout_data_y,
         pred.EN)

# Elasticnet ROC Curve built from out of sample data
ROC.EN = tibble(truth = factor(hout_data_y, levels =c(1,0)), pred.EN) %>%
  yardstick::roc_curve(truth, pred.EN)

ROC.EN %>%
ggplot(aes(1-specificity, sensitivity)) + geom_line() +
geom_abline(lty=3) +
coord_equal() + ggtitle("ROC of Penalized Logistic Regression Model")
```



```
# Elasticnet AUROC
AUROC.EN <- yardstick::roc_auc_vec(factor(hout_data_y, 1:0), pred.EN)
cm.EN <- table(hout_data_y,pred.EN)
cm.EN
```

```
#>            pred.EN
#> hout_data_y   0   1
#>           0 485   4
#>           1   0  11
```

```
# Elacticnet Precision
precision.EN <- (cm.EN[2,2]) / (cm.EN[2,1]+cm.EN[2,2])

# Elasticnet TPR and FPR
TPR.EN <- ROC.EN[3,3]
```

```r
FPR.EN <- ROC.EN[3,2]

CV_perf_table[3, 2] = "alpha = 0.65 and lambda = 6.416034e-06"
CV_perf_table[3, 3] = AUROC.EN
CV_perf_table[3, 4] = "> 0.1"
CV_perf_table[3, 5] = acc.EN
CV_perf_table[3, 6] = TPR.EN
CV_perf_table[3, 7] = FPR.EN
CV_perf_table[3, 8] = precision.EN
```

Tuning Parameters λ and α

The parameters that needed to be tuned for the Penalized Logistic Regression model are λ and α. The way I chose the optimal α was by runing a for loop for various iterations to test out alpha values incrementing at 0.05 steps and then calculating the RMSE of each alpha to find out which one gave the lowest RMSE value and is therefore my optimal α. Once finding the optimal α it was then plugged into the cv.glmnet() function and the optimal λ was then returned.

Support Vector Machines (SVM)

```r
set.seed(4)
# Creating training data for tuning with tune.svm()
Training.Data.SVM.tune <- Training.Data

n = nrow(Training.Data.SVM.tune)

# Tuning data size
n_tune = 10000

tune = sample(n, size = n_tune)

tune_data = Training.Data.SVM.tune[tune,]

# Setting folds to 10
tc <- tune.control(cross = 10)
# Tuning Polynomial model parameters.
poly.tune.poly = tune.svm(factor(Class) ~., data=tune_data, kernel="polynomial",
degree=c(1,2,3), cost = c(0.01,0.1,1,10,100), tunecontrol = tc)
best.poly <- poly.tune.poly$best.parameters
# From poly.tune.poly we get that the best tuning parameters are degree = 1 with cost = 100.

# Tuning Linear model parameters.
poly.tune.line = tune.svm(factor(Class)~., data=tune_data, kernel="linear", cost = c(0.01,0.1,1,10,100)
best.line <- poly.tune.line$best.parameters
# From poly.tune.line we get that the best tuning parameter is cost = 100.

# Tuning Radial model parameters.
poly.tune.radi = tune.svm(factor(Class)~., data=tune_data, kernel="radial", cost = c(0.01,0.1,1,10,100)
best.radi <- poly.tune.radi$best.parameters
# From poly.tune.poly we get that the best tuning parameters are gamma = 0.1 with cost = 10.
# Saving best parameters per tune.svm()
best.poly <- poly.tune.poly$best.parameters
best.line <- poly.tune.line$best.parameters
best.radi <- poly.tune.radi$best.parameters

# Only going to run two models since both the poly tuning parameter gives me a degree = 1 linear model.

# Creating testing data.
Training.Data.SVM <- Training.Data

n = nrow(Training.Data.SVM)

# Testing data size
n_test = 1000

test = sample(n, size = n_test)

test_data_SVM = Training.Data.SVM[test,]

train_data_SVM = Training.Data.SVM[-test,]

fit.SVM.line = svm(factor(Class) ~.,
    data = train_data_SVM,
    #: tuning parameters
    kernel = "linear",
    cost = best.line$cost, # best
    decision.values = TRUE,
    tunecontrol = tc
    )

fit.SVM.radi = svm(factor(Class) ~.,
    data = train_data_SVM,
    #: tuning parameters
    kernel = "radial",
    gamma = best.radi$gamma, # best
    cost = best.radi$cost, # best
    decision.values = TRUE,
    tunecontrol = tc
    )

# Making predictions on Testing Data for SVM Linear and radial model.
pred.SVM.line = predict(fit.SVM.line, test_data_SVM, decision.values = TRUE)

pred.SVM.radi = predict(fit.SVM.radi, test_data_SVM, decision.values = TRUE)

# Creating Confusion matrix for SVM Linear and radial model predictions.
conf_matrix.SVM.line <- confusionMatrix(as.factor(c(pred.SVM.line)),as.factor(test_data_SVM$Class))

conf_matrix.SVM.radi <- confusionMatrix(as.factor(c(pred.SVM.radi)),as.factor(test_data_SVM$Class))

# SVM Linear and radial accuracies
acc_line <- calc_acc(actual = test_data_SVM$Class, c(pred.SVM.line))
acc_radi <- calc_acc(actual = test_data_SVM$Class, c(pred.SVM.radi))
```

```r
# SVM Linear and radial TPR and FPR calculation.
svm_conf_line <- conf_matrix.SVM.line$table
svm_conf_radi <- conf_matrix.SVM.radi$table

TPR_radi <- svm_conf_radi[2,2]/(svm_conf_radi[2,2]+svm_conf_radi[1,2])
TPR_line <- svm_conf_line[2,2]/(svm_conf_line[2,2]+svm_conf_line[1,2])

FPR_radi <- svm_conf_radi[1,1]/(svm_conf_radi[1,2]+svm_conf_radi[1,1])
FPR_line <- svm_conf_line[1,1]/(svm_conf_line[1,2]+svm_conf_line[1,1])

# SVM Linear and radial precision calculation.
precision.radi <- (svm_conf_radi[2,2]) / (svm_conf_radi[2,1]+svm_conf_radi[2,2])

precision.line <- (svm_conf_line[2,2]) / (svm_conf_line[2,1]+svm_conf_line[2,2])

library(pROC)
```

```
#> Type 'citation("pROC")' for a citation.
```

```
#>
#> Attaching package: 'pROC'
```

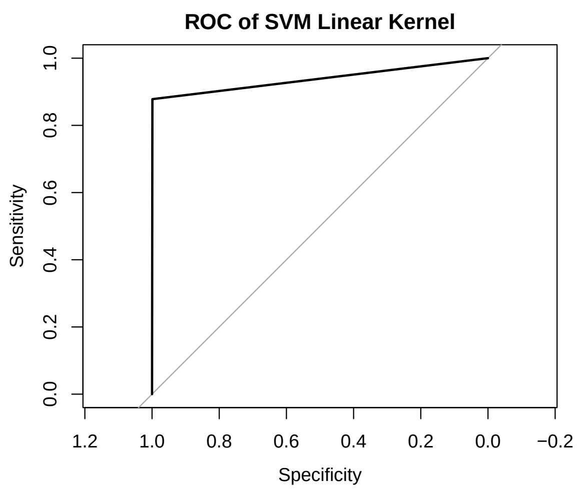
```
#> The following objects are masked from 'package:stats':
#>
#>     cov, smooth, var
```

```r
library(ROCR)

# SVM Linear and radial ROC Curves built from out of sample data.
line_prob <- predict(fit.SVM.line, newdata = test_data_SVM[,2:4], type = "prob")

line_pROC <- roc(response = test_data_SVM$Class, predictor = as.numeric(line_prob))
```

```
#> Setting levels: control = 0, case = 1
```

```
#> Setting direction: controls < cases
```

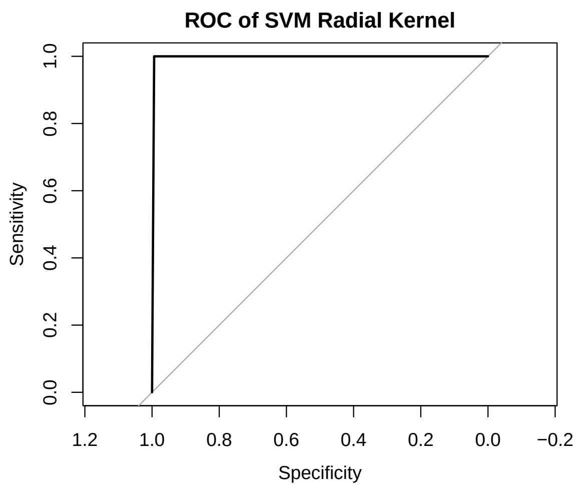
```r
ROC_line <- plot(line_pROC, main = "ROC of SVM Linear Kernel")
```



```
radi_prob <- predict(fit.SVM.radi, newdata = test_data_SVM[,2:4], type = "prob")

radi_pROC <- roc(response = test_data_SVM$Class, predictor = as.numeric(radi_prob))
```

```
#> Setting levels: control = 0, case = 1
#> Setting direction: controls < cases
```

```
ROC_radi <- plot(radi_pROC, main = "ROC of SVM Radial Kernel")
```



```
CV_perf_table[4, 2] = "gamma = 4 cost = 100"
CV_perf_table[4, 3] = radi_pROC$auc
CV_perf_table[4, 4] = "N/A"
CV_perf_table[4, 5] = acc_radi
CV_perf_table[4, 6] = TPR_radi
CV_perf_table[4, 7] = FPR_radi
CV_perf_table[4, 8] = precision.radi
```

Tuning Parameters and Methods.

For the SVM models above the tune.svm() function was run on a limited tuning data set using 10-fold cross validation to determine the best parameters for three kernels: polynomial (cost), linear (degree and cost), and radial (cost and γ). From the outputs of the tune.svm() the best parameters for the polynomial model were degree equals to 1 and cost equals to 100 which was the same thing as my linear recommendation so only two models were developed moving forward using the linear and radial kernel. Using the best parameters for each kernel the models were trained,using 10-fold cross validation, and then tested to see which performed best. The best performing model out of the linear and radial was the radial, acheiving better numbers of accuracy and a better ROC plot.

Final Conclusions

```
CV_perf_table
```

```
#>               Model                                 Tuning     AUROC Threshold
#> 1           Log Reg                                    N/A 0.9703728     > 0.1
#> 2               KNN                                     10 0.9962753     > 0.1
#> 3 Penalized Log Reg alpha = 0.65 and lambda = 6.416034e-06 0.9959100     > 0.1
#> 4               SVM                    gamma = 4 cost = 100 0.9968717       N/A
#> 5       Naive Bayes                                   <NA>        NA      <NA>
#> 6               LDA                                   <NA>        NA      <NA>
#> 7               QDA                                   <NA>        NA      <NA>
#> 8    Random Forests                                   <NA>        NA      <NA>
#> 9     Boosted Trees                                   <NA>        NA      <NA>
#>    Accuracy       TPR       FPR Precision
#> 1 0.9906388 0.9487179 0.9920277 0.7976783
#> 2 0.9974700 0.9950000 0.9975506 0.9950000
#> 3 0.9920000 1.0000000 0.9918200 1.0000000
#> 4 0.9940000 1.0000000 1.0000000 0.8723404
#> 5        NA        NA        NA        NA
#> 6        NA        NA        NA        NA
#> 7        NA        NA        NA        NA
#> 8        NA        NA        NA        NA
#> 9        NA        NA        NA        NA
```

Threshold Justification

The Threshold chosen for the logistic regression, KNN, and penalized logistic regression models was chosen to be > 0.1. This threshold was chosen because the out put of all of my predictors were all estimated probabilities with positive probabilities scoring the highest and negative probabilities scoring the lowest so having a probability of 0.1 made sure that I was setting a threshold that could collect the most true positives possible.

Conclusion #1

Running the models and going over the results all models perform exceptionally well and are all comparable to each other with accuracies all over 99% with few errors in the way the classified the test data. The one that works the best is the Penalized Logistic Regression (ElasticNet) model, this is due to the fact that that the ElasticNet model goes through a finer tuning process than the other methods as it uses the entire data set through k folds to determine the optimal parameters. The SVM model is tuned through a limited tuning data set that is smaller due to the time it would take to process the entire data set to determine the best parameters. The KNN model does offer a better accuracy but its precision is the lower than the ElasticNet model. The logistic regression scores the worst in accuracy and AUROC.

Conclusion #2

The additional recommendations to improve results would be to work on the tuning parameters to see if they can be improved further than they already have. For the SVM parameter increasing the size of the tuning data size should lead to better parameters for each of the kernel models that could lead to a different kernel model being chosen due to performance. For the ElasticNet parameters running more iterations of the RMSE for loop could verify if the correct k neighbor value was chosen or not. The Same can be said for the ElasticNet tuning parameter of running more iterations of for loop for calculating the MSE can verify if the correct α value was chosen.

Conclusion #3

The work being done here is very effective, all models score very well in all categories and are very comparable to each other. All four of these models are more than capable of classifying if a pixel is a blue tarp out of thousands of pixels in a matter of minutes in order to go through images fast locate where humanitarian aid should go to with little error. The data gone through here seems to be well suited for a one class prediction method because we have a lot of data that we can use to train with observations of the desired class "Blue Tarp". This gives the model a lot of examples that help further identify a "Blue Tarp" pixel.

PART II

Model Training

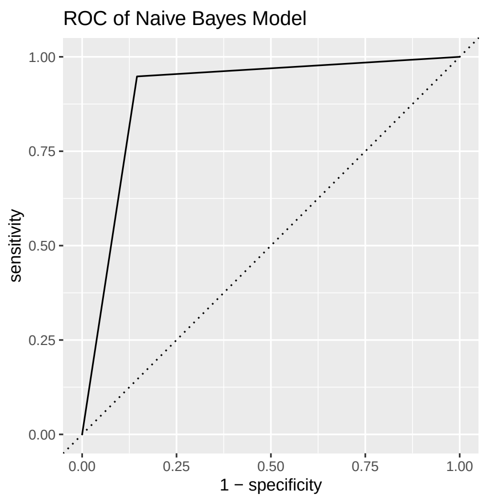
Naive Bayes

```r
# Setting Seed and Creating partitioned data
set.seed(5)
Train.nb <- createDataPartition(Training.Data$Class, p=0.75, list=FALSE)
training.nb <- Training.Data[ Train.nb, ]
testing.nb <- Training.Data[ -Train.nb, ]

# Naive Bayes
train_control <- trainControl(method = "cv", number = 10)

fit.nb.tune <- train(
   x = training.nb[,2:4],
   y = as.factor(training.nb$Class), method = "nb",
   trControl = train_control)

fit.nb.tune
```

```
#> Naive Bayes
#>
```

```
#> 47431 samples
#>     3 predictor
#>     2 classes: '0', '1'
#>
#> No pre-processing
#> Resampling: Cross-Validated (10 fold)
#> Summary of sample sizes: 42688, 42688, 42688, 42688, 42688, 42687, ...
#> Resampling results across tuning parameters:
#>
#>   usekernel  Accuracy   Kappa
#>   FALSE      0.9702726  0.1317966
#>    TRUE      0.9782000  0.5861417
#>
#> Tuning parameter 'fL' was held constant at a value of 0
#> Tuning
#>  parameter 'adjust' 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 fL = 0, usekernel = TRUE and adjust
#>  = 1.
```

```r
pred.nb.tune <- predict(fit.nb.tune, testing.nb[,-1], type="prob")
pred.nb.1.prob <- pred.nb.tune[,2]

# Setting Threshhold for predicted values.
pred.nb.1 <- ifelse(pred.nb.1.prob > 0.1,1,0)
# Creating Confusion Matrix with predicted and test values.
conf_matrix.nb <- caret::confusionMatrix(table(pred.nb.1,(testing.nb$Class)))
# Creating ROC Curve built from out of sample data
ROC.nb = tibble(truth = factor(testing.nb$Class, levels =c(1,0)), pred.nb.1) %>%
  yardstick::roc_curve(truth, pred.nb.1)

ROC.nb %>%
ggplot(aes(1-specificity, sensitivity)) + geom_line() +
geom_abline(lty=3) +
coord_equal() + ggtitle("ROC of Naive Bayes Model")
```

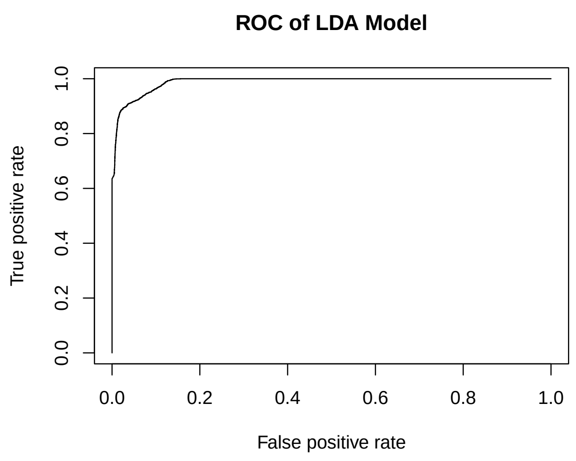


```r
# Computing AUROC for the Predicted values for Logistic Regression.
AUROC.nb <- yardstick::roc_auc_vec(factor(testing.nb$Class, 1:0), pred.nb.1)

# Naive Bayes Accuracy.
acc.nb <- conf_matrix.nb$overall
acc.nb <- as.numeric(acc.nb[1])

# Naive Bayes TPR and FPR.
TPR.nb <- ROC.nb[3,3]
FPR.nb <- ROC.nb[3,2]

# Naive Bayes Precision.
precision.nb <- (conf_matrix.nb$table[4]) / (conf_matrix.nb$table[4]+conf_matrix.nb$table[2])

CV_perf_table[5, 2] = "kernel = TRUE"
CV_perf_table[5, 3] = AUROC.nb
CV_perf_table[5, 4] = "> 0.1"
CV_perf_table[5, 5] = acc.nb
CV_perf_table[5, 6] = TPR.nb
CV_perf_table[5, 7] = FPR.nb
CV_perf_table[5, 8] = precision.nb
```

LDA (Linear Discriminant Analysis)

```r
set.seed(2020)
K = 10
eval <- tibble()
folds = rep(1:K, length=nrow(Training.Data)) %>% sample()

for(k in 1:K){
test = (folds == k)
train = (folds != k)
fit.LDA <- lda(Class ~., data = Training.Data[train,])
fit.LDA.predict <- predict(fit.LDA, Training.Data[test, ], type = "prob")
output = tibble(truth = Training.Data[test,1],
                predictions = fit.LDA.predict$posterior)
eval <- rbind(eval, output)
}

# Predicting Results
pred.LDA <- eval$predictions
pred.LDA.thresh <- ifelse(eval$predictions > 0.1,1,0)
conf_matrix.LDA <- caret::confusionMatrix(table((pred.LDA.thresh[,2]), as.factor(eval$truth)))

roc.LDA <- prediction(pred.LDA[,2], eval$truth)

perf.LDA <- performance(roc.LDA,"tpr","fpr")

AUROC.LDA <- performance(roc.LDA, measure = "auc")

AUROC.LDA <- AUROC.LDA@y.values
```

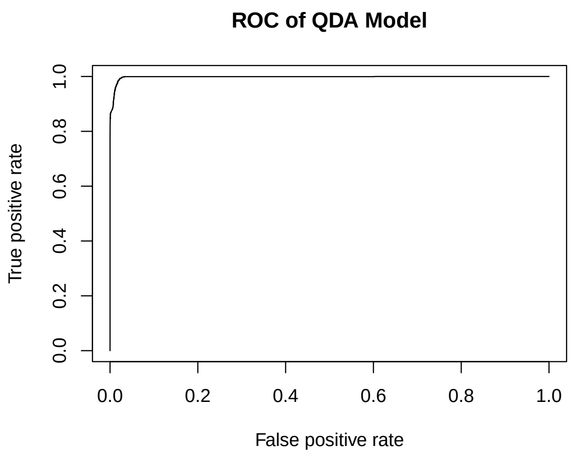
```r
plot(perf.LDA, main = "ROC of LDA Model")
```



```r
acc.LDA <- conf_matrix.LDA$overall
acc.LDA <- as.numeric(acc.LDA[1])

TPR.LDA <- (conf_matrix.LDA$table[4])/(conf_matrix.LDA$table[4]+conf_matrix.LDA$table[3])
FPR.LDA <- (conf_matrix.LDA$table[2])/(conf_matrix.LDA$table[1]+conf_matrix.LDA$table[2])

precision.LDA <- (conf_matrix.LDA$table[4]) / (conf_matrix.LDA$table[4]+conf_matrix.LDA$table[2])

CV_perf_table[6, 2] = "N/A"
CV_perf_table[6, 3] = AUROC.LDA
CV_perf_table[6, 4] = "> 0.1"
CV_perf_table[6, 5] = acc.LDA
CV_perf_table[6, 6] = TPR.LDA
CV_perf_table[6, 7] = FPR.LDA
CV_perf_table[6, 8] = precision.LDA
```

QDA (Quadratic Discriminant Analysis)

```r
set.seed(2020)
K = 10
eval <- tibble()
folds = rep(1:K, length=nrow(Training.Data)) %>% sample()

for(k in 1:K){
test = (folds == k)
train = (folds != k)
fit.QDA <- qda(Class ~., data = Training.Data[train,])
fit.QDA.predict <- predict(fit.QDA, Training.Data[test, ], type = "prob")
output = tibble(truth = Training.Data[test,1],
                 predictions = fit.QDA.predict$posterior)
eval <- rbind(eval, output)
}

# Predicting Results
pred.QDA <- eval$predictions
pred.QDA.thresh <- ifelse(eval$predictions > 0.1,1,0)
conf_matrix.QDA <- caret::confusionMatrix(table((pred.QDA.thresh[,2]), as.factor(eval$truth)))

roc.QDA <- prediction(pred.QDA[,2], eval$truth)

perf.QDA <- performance(roc.QDA,"tpr","fpr")

AUROC.QDA <- performance(roc.QDA, measure = "auc")

AUROC.QDA <- AUROC.QDA@y.values

plot(perf.QDA, main = "ROC of QDA Model")
```



```
acc.QDA <- conf_matrix.QDA$overall
acc.QDA <- as.numeric(acc.QDA[1])

TPR.QDA <- (conf_matrix.QDA$table[4])/(conf_matrix.QDA$table[4]+conf_matrix.QDA$table[3])
FPR.QDA <- (conf_matrix.QDA$table[2])/(conf_matrix.QDA$table[1]+conf_matrix.QDA$table[2])

precision.QDA <- (conf_matrix.QDA$table[4]) / (conf_matrix.QDA$table[4]+conf_matrix.QDA$table[2])

CV_perf_table[7, 2] = "N/A"
CV_perf_table[7, 3] = AUROC.QDA
CV_perf_table[7, 4] = "> 0.1"
CV_perf_table[7, 5] = acc.QDA
CV_perf_table[7, 6] = TPR.QDA
CV_perf_table[7, 7] = FPR.QDA
CV_perf_table[7, 8] = precision.QDA
```

Random Forests

```
if(.Platform$OS.type == "windows") withAutoprint({
memory.size()
memory.size(TRUE)
memory.limit()
})
```

```
#> > memory.size()
#> [1] 410.53
#> > memory.size(TRUE)
#> [1] 803
#> > memory.limit()
#> [1] 7937
```

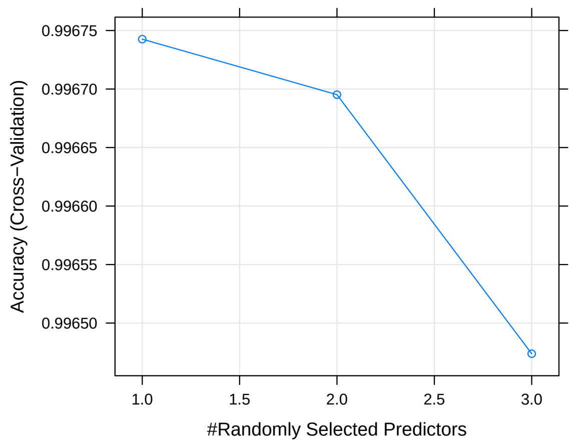
```
memory.limit(size=56000)
```

```
#> [1] 56000
```

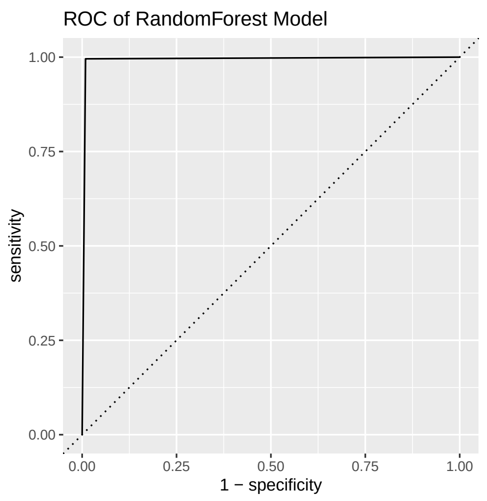
```
# Random Forest
x <- Training.Data[,2:4]
y <- as.factor(Training.Data[,1])
ntree=1000
metric <- "Accuracy"

# Tuning for optimal mtry
control <- trainControl(method="cv", number=2, search="random")
tunegrid <- expand.grid(.mtry=c(1:3))
rf <- train(x = x, y = y, method="rf", tuneLength=1, metric = metric, tuneGrid = tunegrid, trControl=co
print(rf)
```

```
#> Random Forest
#>
#> 63241 samples
#>     3 predictor
#>     2 classes: '0', '1'
#>
#> No pre-processing
#> Resampling: Cross-Validated (2 fold)
#> Summary of sample sizes: 31621, 31620
#> Resampling results across tuning parameters:
#>
#>   mtry  Accuracy   Kappa
#>   1     0.9967426  0.9469743
#>   2     0.9966952  0.9464968
#>   3     0.9964738  0.9430498
#>
#> Accuracy was used to select the optimal model using the largest value.
#> The final value used for the model was mtry = 1.
```

```r
plot(rf)
```



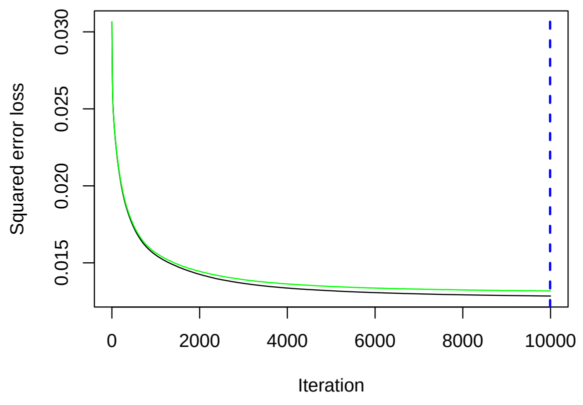
```r
opt.mtry <- rf$bestTune
opt.mtry <- opt.mtry$mtry

# Setting up 10-fold Cross-Validation method Using optimal mtry
set.seed(2020)
K = 10
eval <- tibble()
folds = rep(1:K, length=nrow(Training.Data)) %>% sample()

for(k in 1:K){
test = (folds == k)
train = (folds != k)
rf.opt <- randomForest(as.factor(Class) ~ ., data=Training.Data[train, ],
                       ntree=ntree,
                       mtry = opt.mtry)
rf.opt.predict <- predict(rf.opt, Training.Data[test, ], type = "prob")
output = tibble(truth = Training.Data[test,1],
                predictions = rf.opt.predict[,2])
eval <- rbind(eval, output)
}

# Predicting Results
pred.rf <- ifelse(eval$predictions > 0.1,1,0)
conf_matrix.rf <- caret::confusionMatrix(table(pred.rf,(eval$truth)))

ROC.rf = tibble(truth = factor(eval$truth, levels =c(1,0)), pred.rf) %>%
  yardstick::roc_curve(truth, pred.rf)

ROC.rf %>%
ggplot(aes(1-specificity, sensitivity)) + geom_line() +
geom_abline(lty=3) +
coord_equal() + ggtitle("ROC of RandomForest Model")
```



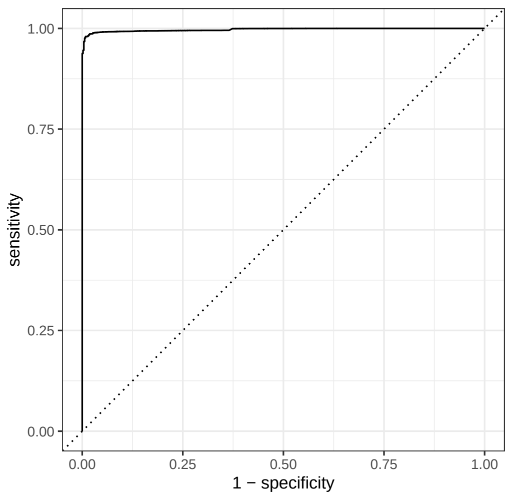
```
AUROC.rf <- yardstick::roc_auc_vec(factor(eval$truth, 1:0), pred.rf)

acc.rf <- conf_matrix.rf$overall
acc.rf <- as.numeric(acc.rf[1])

TPR.rf <- ROC.rf[3,3]
FPR.rf <- ROC.rf[3,2]

precision.rf <- (conf_matrix.rf$table[4]) / (conf_matrix.rf$table[4]+conf_matrix.rf$table[2])

CV_perf_table[8, 2] = "opt.mtry = 1"
CV_perf_table[8, 3] = AUROC.rf
CV_perf_table[8, 4] = "> 0.1"
CV_perf_table[8, 5] = acc.rf
CV_perf_table[8, 6] = TPR.rf
CV_perf_table[8, 7] = FPR.rf
CV_perf_table[8, 8] = precision.rf
```

Boosted Trees

```
# Boosted Trees
set.seed(12)
shrink = seq(.001, 0.1, by = 0.049)
MSE = matrix(NA, 3, 2)
MSE[,1] = seq(.001, 0.1, by = 0.049)

Train <- createDataPartition(Training.Data$Class, p=0.75, list=FALSE)
training.bt <- Training.Data[ Train, ]
testing.bt <- Training.Data[ -Train, ]


# train GBM model

for (i in 1:length(shrink)){
gbm.fit.1 <- gbm(
  formula = as.factor(Class) ~ .,
  distribution = "gaussian",
  data = training.bt,
  n.trees = 1000,
  interaction.depth = 1,
  shrinkage = shrink,
  cv.folds = 3,
  n.cores = NULL,
  verbose = FALSE
  )
min_MSE <- which.min(gbm.fit.1$cv.error)
MSE[i,2] <- sqrt(gbm.fit.1$cv.error[min_MSE])
}

MSE <- as.data.frame(MSE)
row <- which.min(MSE$V2)
shrinkage <- MSE[row,1]

gbm.fit.2 <- gbm(
  formula = as.factor(Class) ~ .,
  distribution = "gaussian",
  data = training.bt,
  n.trees = 10000,
  interaction.depth = 1,
  shrinkage = shrinkage,
  cv.folds = 3,
  n.cores = NULL,
  verbose = FALSE
  )

n.trees <- gbm.perf(gbm.fit.2, method = "cv")
```



```
gbm.fit.3 <- gbm(
  formula = as.factor(Class) ~ .,
  distribution = "gaussian",
  data = training.bt,
  n.trees = n.trees,
  interaction.depth = 1,
  shrinkage = shrinkage,
  cv.folds = 10,
  n.cores = NULL,
  verbose = FALSE
  )

# Predicting Results
pred.bt <- predict(gbm.fit.3, testing.bt, n.trees=n.trees, type = "link")
pred.bt <- rescale(pred.bt)
pred.bt <- as.data.frame(pred.bt)
pred.bt <- (1 - pred.bt)
pred.bt.thresh <- ifelse(pred.bt > 0.1,1,0)
pred.boost <- tibble(testing.bt$Class, pred.bt)
names(pred.boost)[1] <- "Truth"

conf_matrix.bt <- caret::confusionMatrix(table(pred.bt.thresh,(testing.bt$Class)))

autoplot(roc_curve(pred.boost, truth = factor(Truth), predictions = pred.bt), title = "Boosted Tree ROC
```



```
AUROC.bt <- yardstick::roc_auc_vec(factor(pred.boost$Truth), pred.boost$pred.bt)

acc.bt <- (conf_matrix.bt$table[4]+conf_matrix.bt$table[1])/(conf_matrix.bt$table[1]+conf_matrix.bt$tabl
acc.bt <- (1 - acc.bt)

TPR.bt <- (conf_matrix.bt$table[4])/(conf_matrix.bt$table[4]+conf_matrix.bt$table[3])
FPR.bt <- (conf_matrix.bt$table[2])/(conf_matrix.bt$table[1]+conf_matrix.bt$table[2])

precision.bt <- (conf_matrix.bt$table[4]) / (conf_matrix.bt$table[4]+conf_matrix.bt$table[3])

CV_perf_table[9, 2] = "n.trees = 9990 & shrinkage = 0.05"
CV_perf_table[9, 3] = AUROC.bt
CV_perf_table[9, 4] = "> 0.1"
CV_perf_table[9, 5] = acc.bt
CV_perf_table[9, 6] = TPR.bt
```

```
CV_perf_table[9, 7] = FPR.bt
CV_perf_table[9, 8] = precision.bt
```

Threshold Selection

The Thresholds were all set to > 0.1, the reasoning for this was because I would rather have more false more false positives then false negatives if it meant making sure that all the make shift shelters were located and that the displaced people can be delivered food and water as quickly as possible.

Results (Cross-Validation)

Performance Table

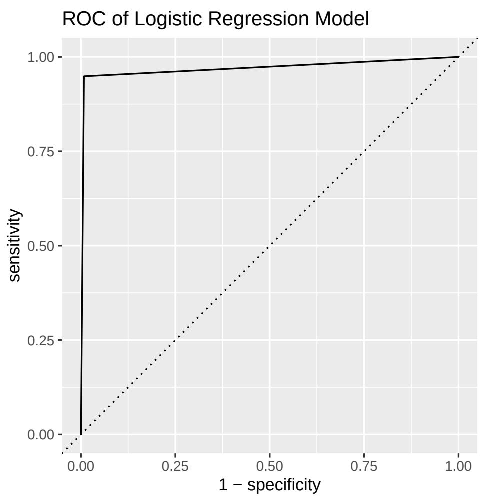
```
CV_perf_table
```

```
#>               Model                               Tuning     AUROC Threshold
#> 1           Log Reg                                  N/A 0.9703728     > 0.1
#> 2               KNN                                   10 0.9962753     > 0.1
#> 3 Penalized Log Reg alpha = 0.65 and lambda = 6.416034e-06 0.9959100     > 0.1
#> 4               SVM                  gamma = 4 cost = 100 0.9968717       N/A
#> 5       Naive Bayes                        kernel = TRUE 0.9014149     > 0.1
#> 6               LDA                                  N/A 0.9888501     > 0.1
#> 7               QDA                                  N/A 0.9981938     > 0.1
#> 8    Random Forests                         opt.mtry = 1 0.9932661     > 0.1
#> 9     Boosted Trees      n.trees = 9990 & shrinkage = 0.05 0.9971182     > 0.1
#>    Accuracy       TPR         FPR Precision
#> 1 0.9906388 0.9487179 0.992027707 0.7976783
#> 2 0.9974700 0.9950000 0.997550621 0.9950000
#> 3 0.9920000 1.0000000 0.991820041 1.0000000
#> 4 0.9940000 1.0000000 1.000000000 0.8723404
#> 5 0.8576850 0.9481038 0.854725978 0.1759911
#> 6 0.9824639 0.8471810 0.013067838 0.6816554
#> 7 0.9895795 0.9005935 0.007481337 0.7990347
#> 8 0.9911292 0.9955490 0.990983191 0.7847953
#> 9 0.9702087 0.9553753 1.000000000 0.9553753
```

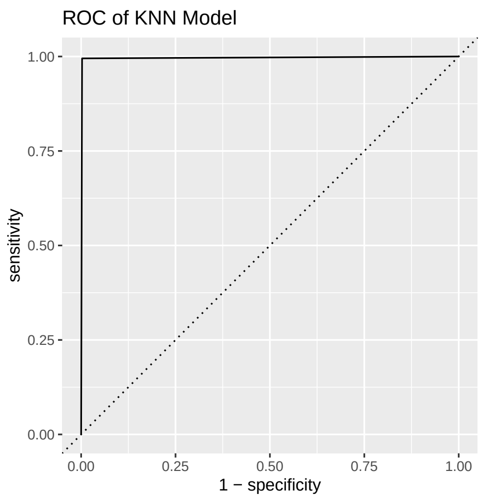
The metrics here were calculated by using parameters (where applicable) that maximized the performance of these models.

ROC Curves

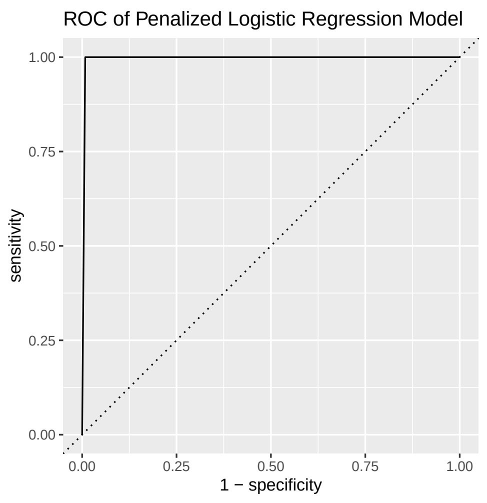
```
# Logistic Regression ROC Curves
ROC.LR %>%
ggplot(aes(1-specificity, sensitivity)) + geom_line() +
geom_abline(lty=3) +
coord_equal() + ggtitle("ROC of Logistic Regression Model")
```



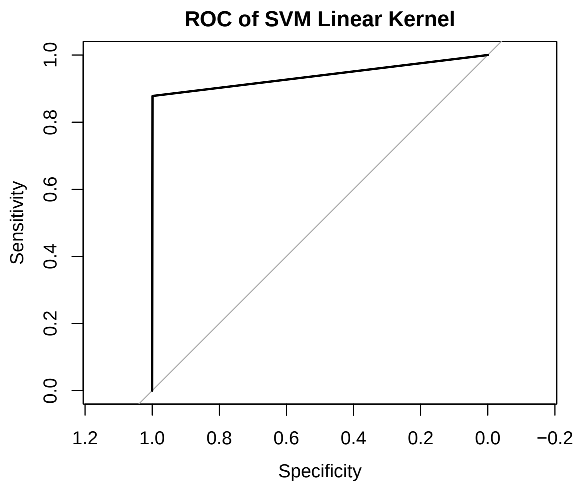
```
# KNN   ROC Curves
ROC.knn %>%
ggplot(aes(1-specificity, sensitivity)) + geom_line() +
geom_abline(lty=3) +
coord_equal() + ggtitle("ROC of KNN Model")
```



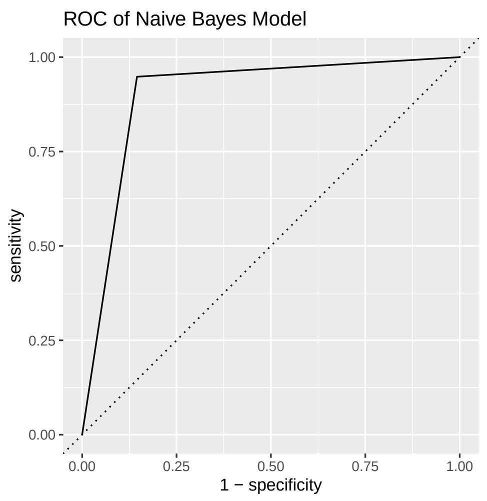
```
# Penalized Logistic Regression (elastic net penalty) ROC Curve
ROC.EN %>%
ggplot(aes(1-specificity, sensitivity)) + geom_line() +
geom_abline(lty=3) +
coord_equal() + ggtitle("ROC of Penalized Logistic Regression Model")
```



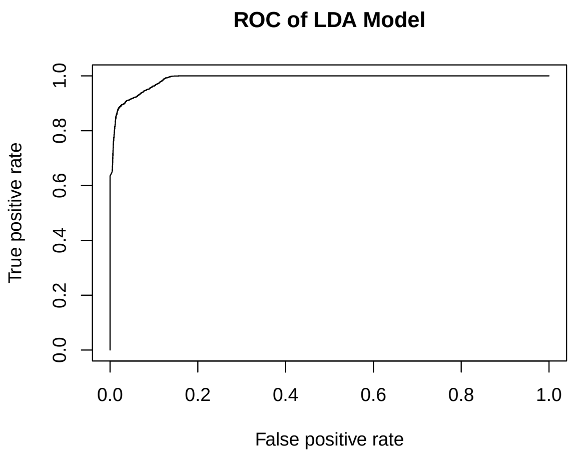
```
# Support Vector Machine ROC Curve
ROC_line <- plot(line_pROC, main = "ROC of SVM Linear Kernel")
```



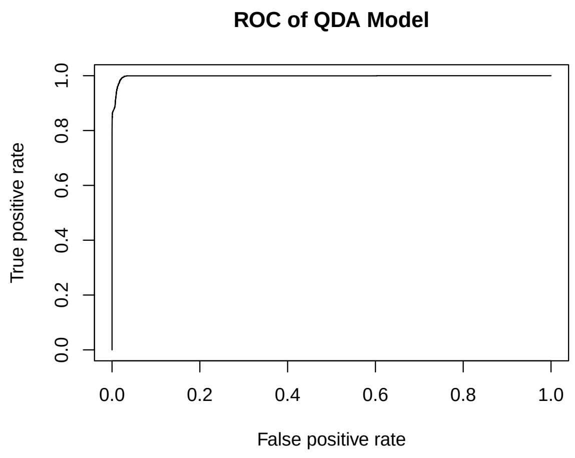
```
# Naive Bayes ROC Curve
ROC.nb %>%
ggplot(aes(1-specificity, sensitivity)) + geom_line() +
geom_abline(lty=3) +
coord_equal() + ggtitle("ROC of Naive Bayes Model")
```



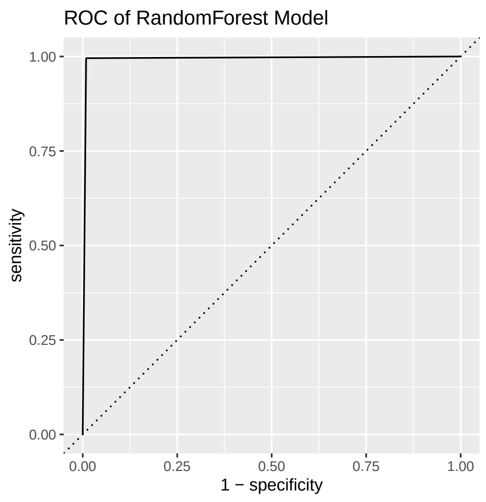
```
# Linear Discriminant Analysis ROC Curve
plot(perf.LDA, main = "ROC of LDA Model")
```



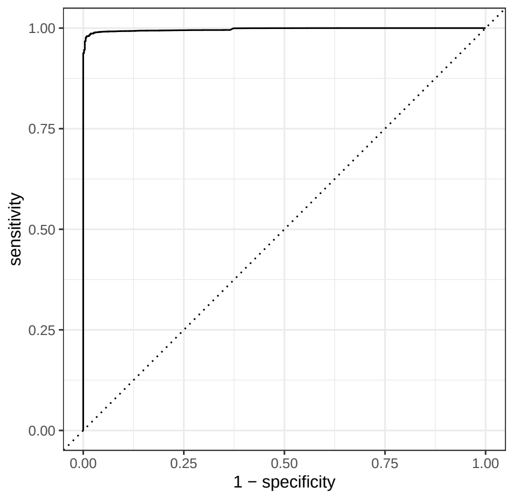
```
# Quadratic Discriminant Analysis ROC Curve
plot(perf.QDA, main = "ROC of QDA Model")
```



```
# Random Forests ROC
ROC.rf %>%
ggplot(aes(1-specificity, sensitivity)) + geom_line() +
geom_abline(lty=3) +
coord_equal() + ggtitle("ROC of RandomForest Model")
```



```
# Boosted Trees
autoplot(roc_curve(pred.boost, truth = factor(Truth), predictions = pred.bt), title = "Boosted Tree ROC
```



ADDITIONAL SECTIONS FOR PART II:

Hold-out Data / EDA

Hold-out Data

```
proj_dir = "~/UVA SYS ME/SYS 6018 Data Mining/Project"

ortho057_noblue = read_table(file.path(proj_dir, "orthovnir057_ROI_NON_Blue_Tarps.txt"), skip = 8, col_r
```

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

```
ortho057_noblue = ortho057_noblue[,8:10]
colnames(ortho057_noblue) <- c("Red", "Green", "Blue")
ortho057_noblue = cbind(bluetarp= 0, ortho057_noblue, file = "057_N")

ortho067_blue = read_table(file.path(proj_dir, "orthovnir067_ROI_Blue_Tarps.txt"), skip = 8, col_names =
```

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

```
ortho067_blue = ortho067_blue[,8:10]
colnames(ortho067_blue) <- c("Red", "Green", "Blue")
ortho067_blue = cbind(bluetarp= 1, ortho067_blue, file = "067_blue")


ortho067_blue_data = read_table(file.path(proj_dir, "orthovnir067_ROI_Blue_Tarps_data.txt"), skip = 8, 
```

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

```
ortho067_blue_data = ortho067_blue_data[,1:3]
colnames(ortho067_blue_data) <- c("Red", "Green", "Blue")
ortho067_blue_data = cbind(bluetarp= 1, ortho067_blue_data, file = "067_blue_data")

ortho067_noblue = read_table(file.path(proj_dir, "orthovnir067_ROI_NOT_Blue_Tarps.txt"), skip = 8, col_
```

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

```
ortho067_noblue = ortho067_noblue[,8:10]
colnames(ortho067_noblue) <- c("Red", "Green", "Blue")
ortho067_noblue = cbind(bluetarp= 0, ortho067_noblue, file = "067_N")
```

```
ortho069_blue = read_table(file.path(proj_dir, "orthovnir069_ROI_Blue_Tarps.txt"), skip = 8, col_names =
```

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

```
ortho069_blue = ortho069_blue[,8:10]
colnames(ortho069_blue) <- c("Red", "Green", "Blue")
ortho069_blue = cbind(bluetarp= 1, ortho069_blue, file = "069_blue")

ortho069_noblue = read_table(file.path(proj_dir, "orthovnir069_ROI_NOT_Blue_Tarps.txt"), skip = 8, col_
```

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

```
ortho069_noblue = ortho069_noblue[,8:10]
colnames(ortho069_noblue) <- c("Red", "Green", "Blue")
ortho069_noblue = cbind(bluetarp= 0, ortho069_noblue, file = "069_N")

ortho078_blue = read_table(file.path(proj_dir, "orthovnir078_ROI_Blue_Tarps.txt"), skip = 8, col_names =
```

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

```
ortho078_blue = ortho078_blue[,8:10]
colnames(ortho078_blue) <- c("Red", "Green", "Blue")
ortho078_blue = cbind(bluetarp= 1, ortho078_blue, file = "078_blue")

ortho078_noblue = read_table(file.path(proj_dir, "orthovnir078_ROI_NON_Blue_Tarps.txt"), skip = 8, col_
```

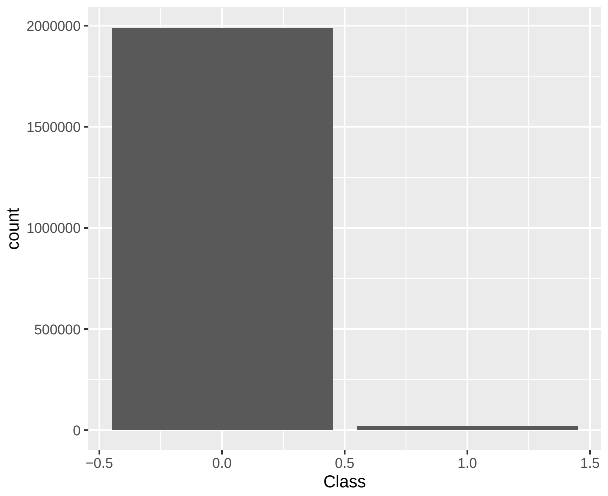
```
#>
#> -- Column specification --------------------------------------------------------
#> cols(
#>   X1 = col_double(),
#>   X2 = col_double(),
#>   X3 = col_double(),
#>   X4 = col_double(),
#>   X5 = col_double(),
#>   X6 = col_double(),
#>   X7 = col_double(),
#>   X8 = col_double(),
#>   X9 =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

```
ortho078_noblue = ortho078_noblue[,8:10]
colnames(ortho078_noblue) <- c("Red", "Green", "Blue")
ortho078_noblue = cbind(bluetarp= 0, ortho078_noblue, file = "078_N")

df_list <- list(
  ortho057_noblue,
  ortho067_blue,
  ortho067_blue_data,
  ortho067_noblue,
  ortho069_blue,
  ortho069_noblue,
  ortho078_blue,
  ortho078_noblue
  )

HaitiHoldOut_complete <- Reduce(function(x, y) merge(x, y, all=TRUE), df_list)

HaitiHoldOut <- HaitiHoldOut_complete

HaitiHoldOut <- HaitiHoldOut %>%
  relocate(bluetarp, .before = Red) %>%
  subset(select = -c(5))

names(HaitiHoldOut)[1] <- 'Class'
```

```
CV_HOO_perf_table <- data.frame(matrix(ncol = 8, nrow = 9))
colnames(CV_HOO_perf_table) <- c("Model", "Tuning", "AUROC", "Threshold", "Accuracy", "TPR", "FPR", "Pr
CV_HOO_perf_table[1, 1] = "Log Reg"
CV_HOO_perf_table[2, 1] = "KNN"
CV_HOO_perf_table[3, 1] = "Penalized Log Reg"
CV_HOO_perf_table[4, 1] = "SVM"
CV_HOO_perf_table[5, 1] = "Naive Bayes"
CV_HOO_perf_table[6, 1] = "LDA"
CV_HOO_perf_table[7, 1] = "QDA"
CV_HOO_perf_table[8, 1] = "Random Forests"
CV_HOO_perf_table[9, 1] = "Boosted Trees"
```

EDA

```
# plot to see count of all the classes
ggplot(data = HaitiHoldOut) +
  geom_bar(mapping = aes(x = Class))
```



```
# Blue Tarp is the least amount of pixels of any of the classes.
HaitiHoldOut %>%
  count(Class)
```

```
#>   Class       n
#> 1     0 1989697
#> 2     1   18919
```

```
# Number of classes per each pixel.
by(HaitiHoldOut, HaitiHoldOut$Class, summary)
```

```
#> HaitiHoldOut$Class: 0
#>      Class        Red             Green             Blue
#>  Min.   :0   Min.   : 27.0   Min.   : 28.0   Min.   : 25.00
#>  1st Qu.:0   1st Qu.: 76.0   1st Qu.: 70.0   1st Qu.: 55.00
#>  Median :0   Median :107.0   Median : 91.0   Median : 66.00
#>  Mean   :0   Mean   :118.3   Mean   :105.2   Mean   : 81.76
#>  3rd Qu.:0   3rd Qu.:139.0   3rd Qu.:116.0   3rd Qu.: 87.00
#>  Max.   :0   Max.   :255.0   Max.   :255.0   Max.   :255.00
#> ------------------------------------------------------------
#> HaitiHoldOut$Class: 1
#>      Class        Red             Green            Blue
#>  Min.   :1   Min.   : 48.0   Min.   : 56.0   Min.   : 57
#>  1st Qu.:1   1st Qu.: 82.0   1st Qu.:102.0   1st Qu.:126
#>  Median :1   Median :110.0   Median :131.0   Median :162
#>  Mean   :1   Mean   :112.2   Mean   :135.8   Mean   :168
#>  3rd Qu.:1   3rd Qu.:134.0   3rd Qu.:163.0   3rd Qu.:210
#>  Max.   :1   Max.   :255.0   Max.   :255.0   Max.   :255
```

```
# looking at all the summaries of each Class.
```

Results (Hold-Out)

Logistic Regression Hold-Out

```
# Using trained model to predict values on Hold-Out (HOO) Data.
pred.LR.HOO <- predict(mod_fitcv, newdata = HaitiHoldOut[, 2:4])
# Setting Threshhold for predicted values.
pred.LR.HOO <- ifelse(pred.LR.HOO > 0.1,1,0)
# Creating Confusion Matrix with predicted and test values.
conf_matrix.LR.HOO <- caret::confusionMatrix(table(pred.LR.HOO,(HaitiHoldOut$Class)))
# Creating ROC Curve built from out of sample data
ROC.LR.HOO = tibble(truth = factor(HaitiHoldOut$Class, levels =c(1,0)), pred.LR.HOO) %>%
  yardstick::roc_curve(truth, pred.LR.HOO)

# Computing AUROC for the Predicted values for Logistic Regression.
AUROC.LR.HOO <- yardstick::roc_auc_vec(factor(HaitiHoldOut$Class, 1:0), pred.LR.HOO)

# Logistic Regression Accuracy.
acc.LR.HOO <- conf_matrix.LR.HOO$overall
acc.LR.HOO <- as.numeric(acc.LR.HOO[1])

# Logistic Regression TPR and FPR.
TPR.LR.HOO <- ROC.LR.HOO[3,3]
FPR.LR.HOO <- ROC.LR.HOO[3,2]

# Logistic Regression Precision.
precision.LR.HOO <- (conf_matrix.LR.HOO$table[4]) / (conf_matrix.LR.HOO$table[4]+conf_matrix.LR.HOO$tab

# Adding values to Performance Table
CV_HOO_perf_table[1, 2] = "N/A"
CV_HOO_perf_table[1, 3] = AUROC.LR.HOO
CV_HOO_perf_table[1, 4] = "> 0.1"
CV_HOO_perf_table[1, 5] = acc.LR.HOO
CV_HOO_perf_table[1, 6] = TPR.LR.HOO
```

```
CV_HOO_perf_table[1, 7] = FPR.LR.HOO
CV_HOO_perf_table[1, 8] = precision.LR.HOO
```

KNN Hold-Out

```
# 10 CV training Running model with optimal K neighbor and Testing with Hold Out Data.
K = 10 # folds
folds = rep(1:K, length=nrow(Training.Data)) %>% sample()
for(k in 1:K){
train = (folds != k)
knn.HOO = FNN::knn.reg(train = Training.Data[train,],
                       y = Training.Data$Class[train],
                       test = HaitiHoldOut,
                       k = opt.K$k)
}

# Setting Threshold to 0.1
pred.knn.HOO <- ifelse(knn.HOO$pred > 0.1,1,0)

# Creating Confusion Matrix
conf_matrix.knn.HOO <- caret::confusionMatrix(table(pred.knn.HOO,(HaitiHoldOut$Class)))
cm.knn.HOO <- table(HaitiHoldOut$Class,pred.knn.HOO)
cm.knn.HOO
```

```
#>    pred.knn.HOO
#>           0       1
#>   0 1965060   24637
#>   1    1295   17624
```

```
# KNN Accuracy.
acc.knn.HOO <-calc_acc(actual = HaitiHoldOut$Class,
         pred.knn.HOO)

# Computing AUROC for the Predicted values for KNN.
AUROC.knn.HOO <- yardstick::roc_auc_vec(factor(HaitiHoldOut$Class, 1:0),  pred.knn.HOO)

# KNN Precision.
precision.knn.HOO <- (cm.knn.HOO[2,2]) / (cm.knn.HOO[2,1]+cm.knn.HOO[2,2])

# KNN TPR and FPR.
TPR.knn.HOO <- (conf_matrix.knn.HOO$table[4])/(conf_matrix.knn.HOO$table[4]+conf_matrix.knn.HOO$table[3
FPR.knn.HOO <- (conf_matrix.knn.HOO$table[2])/(conf_matrix.knn.HOO$table[1]+conf_matrix.knn.HOO$table[2

# Adding values to Performance Table
CV_HOO_perf_table[2, 2] = opt.K$k
CV_HOO_perf_table[2, 3] = AUROC.knn.HOO
CV_HOO_perf_table[2, 4] = "> 0.1"
CV_HOO_perf_table[2, 5] = acc.knn.HOO
CV_HOO_perf_table[2, 6] = TPR.knn.HOO
CV_HOO_perf_table[2, 7] = FPR.knn.HOO
CV_HOO_perf_table[2, 8] = precision.knn.HOO
```

Penalized Logistic Regression (elastic net penalty) Hold-Out

```r
# Setting up Hold out (HOO) Data
HaitiHoldOut_x = glmnet::makeX(HaitiHoldOut[,2:4])
# Making predictions with model.
pred.EN.HOO = predict(fit.enet, s=lambda.hat, newx=HaitiHoldOut_x, type="response")[,1]
# Threshold for predictions.
pred.EN.HOO <- ifelse(pred.EN.HOO > 0.1,1,0)

# Elasticnet Confusion Matrix
conf_matrix.EN.HOO <- confusionMatrix(as.factor(pred.EN.HOO),as.factor(HaitiHoldOut$Class))

# Elasticnet Accuracy
acc.EN.HOO <- calc_acc(actual = HaitiHoldOut$Class,
         pred.EN.HOO)

# Elasticnet ROC Curve built from out of sample data
ROC.EN.HOO = tibble(truth = factor(HaitiHoldOut$Class, levels =c(1,0)), pred.EN.HOO) %>%
  yardstick::roc_curve(truth, pred.EN.HOO)

# Elasticnet AUROC
AUROC.EN.HOO <- yardstick::roc_auc_vec(factor(HaitiHoldOut$Class, levels =c(1,0)), pred.EN.HOO)

# Elasticnet Accuracy.
acc.EN.HOO <- conf_matrix.EN.HOO$overall
acc.EN.HOO <- as.numeric(acc.EN.HOO[1])

# Elasticnet TPR and FPR.
TPR.EN.HOO <- ROC.EN.HOO[3,3]
FPR.EN.HOO <- ROC.EN.HOO[3,2]

# Elasticnet Precision.
precision.EN.HOO <- (conf_matrix.EN.HOO$table[4]) / (conf_matrix.EN.HOO$table[4]+conf_matrix.EN.HOO$tabl

# Adding values to Performance Table
CV_HOO_perf_table[3, 2] = "alpha = 0.65 and lambda = 6.416034e-06"
CV_HOO_perf_table[3, 3] = AUROC.EN.HOO
CV_HOO_perf_table[3, 4] = "> 0.1"
CV_HOO_perf_table[3, 5] = acc.EN.HOO
CV_HOO_perf_table[3, 6] = TPR.EN.HOO
CV_HOO_perf_table[3, 7] = FPR.EN.HOO
CV_HOO_perf_table[3, 8] = precision.EN.HOO
```

SVM (Support Vector Machines) Hold-Out

```r
# Making predictions on Testing Data for SVM Linear and radial model.
pred.SVM.radi.HOO = predict(fit.SVM.radi, HaitiHoldOut, decision.values = TRUE)

# Creating Confusion matrix for SVM Linear and radial model predictions.
conf_matrix.SVM.radi.HOO <- confusionMatrix(as.factor(c(pred.SVM.radi.HOO)),as.factor(HaitiHoldOut$Clas
# SVM Linear and radial accuracies
acc_radi.HOO <- calc_acc(actual = HaitiHoldOut$Class, c(pred.SVM.radi.HOO))

# SVM Linear and radial TPR and FPR calculation.
svm_conf_radi.HOO <- conf_matrix.SVM.radi.HOO$table

TPR_radi.HOO <- svm_conf_radi.HOO[2,2]/(svm_conf_radi.HOO[2,2]+svm_conf_radi.HOO[1,2])

FPR_radi.HOO <- svm_conf_radi.HOO[1,1]/(svm_conf_radi.HOO[1,2]+svm_conf_radi.HOO[1,1])

# SVM Linear and radial precision calculation.
precision.radi.HOO <- (svm_conf_radi.HOO[2,2]) / (svm_conf_radi.HOO[2,1]+svm_conf_radi.HOO[2,2])


library(pROC)
library(ROCR)

# SVM Linear and radial ROC Curves built from out of sample data.
radi_prob.HOO <- predict(fit.SVM.radi, newdata = HaitiHoldOut, type = "prob")

radi_pROC.HOO <- roc(response = HaitiHoldOut$Class, predictor = as.numeric(radi_prob.HOO))
```

```
#> Setting levels: control = 0, case = 1
```

```
#> Setting direction: controls < cases
```

```r
# Adding values to Performance Table
CV_HOO_perf_table[4, 2] = "gamma = 4 cost = 100"
CV_HOO_perf_table[4, 3] = radi_pROC.HOO$auc
CV_HOO_perf_table[4, 4] = "N/A"
CV_HOO_perf_table[4, 5] = acc_radi.HOO
CV_HOO_perf_table[4, 6] = TPR_radi.HOO
CV_HOO_perf_table[4, 7] = FPR_radi.HOO
CV_HOO_perf_table[4, 8] = precision.radi.HOO
```

Naive Bayes Hold-Out

```r
# Predicting
nb.HOO.1 <- HaitiHoldOut[1:1004308,]
nb.HOO.2 <- HaitiHoldOut[1004309:2008616,]

pred.nb.tune.HOO.1 <- predict(fit.nb.tune, nb.HOO.1[,-1], type="prob")
pred.nb.tune.HOO.2 <- predict(fit.nb.tune, nb.HOO.2[,-1], type="prob")
pred.nb.tune.HOO <- rbind(pred.nb.tune.HOO.1, pred.nb.tune.HOO.2)
pred.nb.1.prob.HOO <- pred.nb.tune.HOO[,2]

# Setting Threshhold for predicted values.
pred.nb.1.HOO <- ifelse(pred.nb.1.prob.HOO > 0.1,1,0)
# Creating Confusion Matrix with predicted and test values.
conf_matrix.nb.HOO <- caret::confusionMatrix(table(pred.nb.1.HOO,(HaitiHoldOut$Class)))
```

```r
# Creating ROC Curve built from out of sample data
ROC.nb.HOO = tibble(truth = factor(HaitiHoldOut$Class, levels =c(1,0)), pred.nb.1.HOO) %>%
  yardstick::roc_curve(truth, pred.nb.1.HOO)

# Computing AUROC for the Predicted values for Naive Bayes.
AUROC.nb.HOO <- yardstick::roc_auc_vec(factor(HaitiHoldOut$Class, 1:0), pred.nb.1.HOO)

# Naive Bayes Accuracy.
acc.nb.HOO <- conf_matrix.nb.HOO$overall
acc.nb.HOO <- as.numeric(acc.nb.HOO[1])

# Naive Bayes TPR and FPR.
TPR.nb.HOO <- ROC.nb.HOO[3,3]
FPR.nb.HOO <- ROC.nb.HOO[3,2]

# Naive Bayes Precision.
precision.nb.HOO <- (conf_matrix.nb.HOO$table[4]) / (conf_matrix.nb.HOO$table[4]+conf_matrix.nb.HOO$tabl

# Adding values to Performance Table
CV_HOO_perf_table[5, 2] = "kernel = TRUE"
CV_HOO_perf_table[5, 3] = AUROC.nb.HOO
CV_HOO_perf_table[5, 4] = "> 0.1"
CV_HOO_perf_table[5, 5] = acc.nb.HOO
CV_HOO_perf_table[5, 6] = TPR.nb.HOO
CV_HOO_perf_table[5, 7] = FPR.nb.HOO
CV_HOO_perf_table[5, 8] = precision.nb.HOO
```

LDA (Linear Discriminant Analysis) Hold-Out

```r
# Predicting Results
fit.LDA.predict.HOO <- predict(fit.LDA, HaitiHoldOut, type = "prob")
pred.LDA.HOO <- fit.LDA.predict.HOO$posterior
pred.LDA.thresh.HOO <- ifelse(pred.LDA.HOO > 0.1,1,0)
conf_matrix.LDA.HOO <- caret::confusionMatrix(table((pred.LDA.thresh.HOO[,2]), as.factor(HaitiHoldOut$Cl

roc.LDA.HOO <- prediction(pred.LDA.HOO[,2], HaitiHoldOut$Class)

perf.LDA.HOO <- performance(roc.LDA.HOO,"tpr","fpr")

# Computing AUROC for the Predicted values for Linear Discriminant Analysis.
AUROC.LDA.HOO <- performance(roc.LDA.HOO, measure = "auc")

AUROC.LDA.HOO <- AUROC.LDA.HOO@y.values

# LDA Accuracy.
acc.LDA.HOO <- conf_matrix.LDA.HOO$overall
acc.LDA.HOO <- as.numeric(acc.LDA.HOO[1])

# LDA TPR AND FPR.
TPR.LDA.HOO <- (conf_matrix.LDA.HOO$table[4])/(conf_matrix.LDA.HOO$table[4]+conf_matrix.LDA.HOO$table[3
FPR.LDA.HOO <- (conf_matrix.LDA.HOO$table[2])/(conf_matrix.LDA.HOO$table[1]+conf_matrix.LDA.HOO$table[2
# LDA Precision
precision.LDA.HOO <- (conf_matrix.LDA.HOO$table[4]) / (conf_matrix.LDA.HOO$table[4]+conf_matrix.LDA.HOO$

# Adding values to Performance Table
CV_HOO_perf_table[6, 2] = "N/A"
CV_HOO_perf_table[6, 3] = AUROC.LDA.HOO
CV_HOO_perf_table[6, 4] = "> 0.1"
CV_HOO_perf_table[6, 5] = acc.LDA.HOO
CV_HOO_perf_table[6, 6] = TPR.LDA.HOO
CV_HOO_perf_table[6, 7] = FPR.LDA.HOO
CV_HOO_perf_table[6, 8] = precision.LDA.HOO
```

QDA (Quadratic Discriminant Analysis) Hold-Out

```r
# Predicting Results
fit.QDA.predict.HOO <- predict(fit.QDA, HaitiHoldOut, type = "prob")
pred.QDA.HOO <- fit.QDA.predict.HOO$posterior
pred.QDA.thresh.HOO <- ifelse(pred.QDA.HOO > 0.1,1,0)
conf_matrix.QDA.HOO <- caret::confusionMatrix(table((pred.QDA.thresh.HOO[,2]), as.factor(HaitiHoldOut$Cl

# Developing Metrics
roc.QDA.HOO <- prediction(pred.QDA.HOO[,2], HaitiHoldOut$Class)

perf.QDA.HOO <- performance(roc.QDA.HOO,"tpr","fpr")

# Computing AUROC for the Predicted values for Quadratic Discriminant Analysis.
AUROC.QDA.HOO <- performance(roc.QDA.HOO, measure = "auc")

AUROC.QDA.HOO <- AUROC.QDA.HOO@y.values

# QDA Accuracy.
acc.QDA.HOO <- conf_matrix.QDA.HOO$overall
acc.QDA.HOO <- as.numeric(acc.QDA.HOO[1])

# QDA TPR AND FPR.
TPR.QDA.HOO <- (conf_matrix.QDA.HOO$table[4])/(conf_matrix.QDA.HOO$table[4]+conf_matrix.QDA.HOO$table[3
FPR.QDA.HOO <- (conf_matrix.QDA.HOO$table[2])/(conf_matrix.QDA.HOO$table[1]+conf_matrix.QDA.HOO$table[2

# QDA Precision
precision.QDA.HOO <- (conf_matrix.QDA.HOO$table[4]) / (conf_matrix.QDA.HOO$table[4]+conf_matrix.QDA.HOO$

# Adding values to Performance Table
CV_HOO_perf_table[7, 2] = "N/A"
CV_HOO_perf_table[7, 3] = AUROC.QDA.HOO
CV_HOO_perf_table[7, 4] = "> 0.1"
CV_HOO_perf_table[7, 5] = acc.QDA.HOO
CV_HOO_perf_table[7, 6] = TPR.QDA.HOO
CV_HOO_perf_table[7, 7] = FPR.QDA.HOO
CV_HOO_perf_table[7, 8] = precision.QDA.HOO
```

Random Forests Hold-Out

```r
# Predicting Results
rf.opt.predict.HOO <- predict(rf.opt, HaitiHoldOut, type = "prob")
rf.opt.predict.HOO <- rf.opt.predict.HOO[,2]
pred.rf.HOO <- ifelse(rf.opt.predict.HOO > 0.1,1,0)

# Developing Metrics
conf_matrix.rf.HOO <- caret::confusionMatrix(table(pred.rf.HOO,(HaitiHoldOut$Class)))

ROC.rf.HOO = tibble(truth = factor(HaitiHoldOut$Class, levels =c(1,0)), pred.rf.HOO) %>%
  yardstick::roc_curve(truth, pred.rf.HOO)

# Random Forest AUROC
AUROC.rf.HOO <- yardstick::roc_auc_vec(factor(HaitiHoldOut$Class, 1:0), pred.rf.HOO)

# Random Forest Accuracy
acc.rf.HOO <- conf_matrix.rf.HOO$overall
acc.rf.HOO <- as.numeric(acc.rf.HOO[1])

# Random Forest TPR AND FPR
TPR.rf.HOO <- ROC.rf.HOO[3,3]
FPR.rf.HOO <- ROC.rf.HOO[3,2]

# Random Forest PRECISION
precision.rf.HOO <- (conf_matrix.rf.HOO$table[4]) / (conf_matrix.rf.HOO$table[4]+conf_matrix.rf.HOO$tabl

# Adding values to Performance Table
CV_HOO_perf_table[8, 2] = "opt.mtry = 1"
CV_HOO_perf_table[8, 3] = AUROC.rf.HOO
CV_HOO_perf_table[8, 4] = "> 0.1"
CV_HOO_perf_table[8, 5] = acc.rf.HOO
CV_HOO_perf_table[8, 6] = TPR.rf.HOO
CV_HOO_perf_table[8, 7] = FPR.rf.HOO
CV_HOO_perf_table[8, 8] = precision.rf.HOO
```

Boosted Trees Hold-Out

```r
# Predicting Results
pred.bt.HOO <- predict(gbm.fit.3, HaitiHoldOut, n.trees=n.trees, type = "link")
pred.bt.HOO <- rescale(pred.bt.HOO)
pred.bt.HOO <- as.data.frame(pred.bt.HOO)
pred.bt.HOO <- (1 - pred.bt.HOO)
pred.bt.thresh.HOO <- ifelse(pred.bt.HOO > 0.1,1,0)
pred.boost.HOO <- tibble(HaitiHoldOut$Class, pred.bt.HOO)
names(pred.boost.HOO)[1] <- "Truth"

conf_matrix.bt.HOO <- caret::confusionMatrix(table(pred.bt.thresh.HOO,(HaitiHoldOut$Class)))

# Boosted Trees AUROC
AUROC.bt.HOO <- yardstick::roc_auc_vec(factor(pred.boost.HOO$Truth), pred.boost.HOO$pred.bt.HOO)
```

```
# Boosted Trees Accuracy
acc.bt.HOO <- (conf_matrix.bt.HOO$table[4]+conf_matrix.bt.HOO$table[1])/(conf_matrix.bt.HOO$table[1]+co
acc.bt.HOO <- (1 - acc.bt.HOO)

# Boosted Trees TPR AND FPR
TPR.bt.HOO <- (conf_matrix.bt.HOO$table[4])/(conf_matrix.bt.HOO$table[4]+conf_matrix.bt.HOO$table[3])
FPR.bt.HOO <- (conf_matrix.bt.HOO$table[2])/(conf_matrix.bt.HOO$table[1]+conf_matrix.bt.HOO$table[2])

# Boosted Trees Precision
precision.bt.HOO <- (conf_matrix.bt.HOO$table[4]) / (conf_matrix.bt.HOO$table[4]+conf_matrix.bt.HOO$tab

# Adding values to Performance Table
CV_HOO_perf_table[9, 2] = "n.trees = 9990 & shrinkage = 0.05"
CV_HOO_perf_table[9, 3] = AUROC.bt.HOO
CV_HOO_perf_table[9, 4] = "> 0.1"
CV_HOO_perf_table[9, 5] = acc.bt.HOO
CV_HOO_perf_table[9, 6] = TPR.bt.HOO
CV_HOO_perf_table[9, 7] = FPR.bt.HOO
CV_HOO_perf_table[9, 8] = precision.bt.HOO
```

Cross-Validation Performance Table

```
CV_perf_table
```

```
#>                Model                            Tuning     AUROC Threshold
#> 1            Log Reg                               N/A 0.9703728     > 0.1
#> 2                KNN                                10 0.9962753     > 0.1
#> 3 Penalized Log Reg alpha = 0.65 and lambda = 6.416034e-06 0.9959100     > 0.1
#> 4                SVM                  gamma = 4 cost = 100 0.9968717       N/A
#> 5        Naive Bayes                         kernel = TRUE 0.9014149     > 0.1
#> 6                LDA                               N/A 0.9888501     > 0.1
#> 7                QDA                               N/A 0.9981938     > 0.1
#> 8     Random Forests                         opt.mtry = 1 0.9932661     > 0.1
#> 9      Boosted Trees      n.trees = 9990 & shrinkage = 0.05 0.9971182     > 0.1
#>    Accuracy       TPR         FPR Precision
#> 1 0.9906388 0.9487179 0.992027707 0.7976783
#> 2 0.9974700 0.9950000 0.997550621 0.9950000
#> 3 0.9920000 1.0000000 0.991820041 1.0000000
#> 4 0.9940000 1.0000000 1.000000000 0.8723404
#> 5 0.8576850 0.9481038 0.854725978 0.1759911
#> 6 0.9824639 0.8471810 0.013067838 0.6816554
#> 7 0.9895795 0.9005935 0.007481337 0.7990347
#> 8 0.9911292 0.9955490 0.990983191 0.7847953
#> 9 0.9702087 0.9553753 1.000000000 0.9553753
```

The metrics for calculated under the cross-validation framework were done using the average results of the 10-CV model.

Hold-out Performance Table

```
CV_HOO_perf_table
```

```
#>               Model                                  Tuning     AUROC Threshold
#> 1           Log Reg                                     N/A 0.9642899     > 0.1
#> 2               KNN                                      10 0.9595840     > 0.1
#> 3 Penalized Log Reg alpha = 0.65 and lambda = 6.416034e-06 0.9660632     > 0.1
#> 4               SVM                     gamma = 4 cost = 100 0.7379462       N/A
#> 5       Naive Bayes                            kernel = TRUE 0.6538929     > 0.1
#> 6               LDA                                     N/A 0.9924008     > 0.1
#> 7               QDA                                     N/A 0.9923053     > 0.1
#> 8    Random Forests                           opt.mtry = 1 0.9698620     > 0.1
#> 9     Boosted Trees      n.trees = 9990 & shrinkage = 0.05 0.9875966     > 0.1
#>    Accuracy       TPR        FPR  Precision
#> 1 0.9297193 0.9995243 0.92905553 0.11813729
#> 2 0.9870896 0.9315503 0.01238229 0.93155029
#> 3 0.9330768 0.9996829 0.93244348 0.12334833
#> 4 0.9880316 0.4830594 0.99507359 0.39057225
#> 5 0.8555812 0.4483324 0.85945347 0.02943841
#> 6 0.9778534 0.9067604 0.02147061 0.28651357
#> 7 0.9811761 0.8346636 0.01743079 0.31286034
#> 8 0.9727051 0.9669644 0.97275967 0.25235192
#> 9 0.9907105 0.9862572 1.00000000 0.98625720
```

Final Conclusions

Conclusion #1

The best performing models in the Cross-Validation data were the Random Forests, KNN, and Penalized Regression. the true positive rate at which the models score at are the highest amongst all the models with Penalized Regression having the best precision to be the best of the three mentioned. The best performing models in the hold-out data section are KNN and Boosted trees these models score the best in all the metrics recorded with the best of those two being KNN as the false positive rate on the model is really low but all other metrics score high.

Conclusion #2

The findings above are compatible because the models were developed using 10 fold cross-validation meaning that predictions were made on all the data to average out the results and obtain a final score. All of these models used the same data in their 10 CV folds and made predictions on an out of sample data set that had not been seen before by them. They are all also evaluated on the sema metrics in order to determine which model had the best performance out of the nine.

Conclusion #3

The model I would recommend to use for the detection of the blue tarps would be the KNN model this model performs well in the CV training and performs the best in the HOO data testing, this model give a great score in accuracy and the lowest FPR amongst the 9 tested to meaning that of the pixels selected there were not very many that were selected as blue tarp but in reality were not. This would allow the responding organization to this disaster to not have to go through Many False positives to decide where to send resources to which will save time in getting the displaced people what they need.

Conclusion #4

These metrics were all relevant and each contributed to the determination of the best model. The AUROC score gave us an understanding of how much the model is capable of distinguishing between the two classes. The Accuracy showed how well the model predicted correctly, TPR also helped with this. FPR showed of the predictions made how much were labeled as true positive but were actually not. And lastly Precision helped show that the model was able to identify only the relevant data points. Using all these metircs it was determined that KNN was the best of the nine models when testing for the Hold-Out data.

Conclusion #5

When setting up the Hold-Out data it seems safe to assume that the correct values for Red Green and Blue from the HOO data were selected as the performance metrics for the hold-out data were good. Exploratory Data Analysis was used to determine how to assign the RGB values for the Hold-Out data. Future work might include changes to the RGB values to verify this is true. Another possible area in which future work could be done is the scaling of data to see the effects if it helps or not. As discussed in the earlier portion of this project scaling was not done was all the color values followed the same scale and there were not sharp differences in the range of values used.

Conclusion #6

The worst model of the nine is the Naive Bayes model. This is both true in with the CV data and the hold-out data. for both we get the lowest AUROC score of the 9 models with the hold-out data being 0.65, slightly above the threshold of just picking randomly. The model also scored the lowest accuracies of the nine with the lowest precision as well. This could be due to the fact that the Naive Bayes classifier only focuses on the presence of a particular feature in a class and is unrelated to the presence of any other feature.