Disaster Relief Project: Part II

Thomas Butler, vra2cf

Aug 12, 2021

SYS 6018 | Spring 2021 | University of Virginia

Introduction

The 2010 Haiti earthquake was devastated Haiti and displaced many people. Rescue workers were trying to help get food and water to displaced people but there was a challenge. The earthquake destroyed communications and made roads impassable which made finding displaced people hard to find spread out across thousands of square miles. It was known that displaced people were using blue tarps to create temporary shelters so Rochester Institute of Technology were using aircraft to collect high resolution geo-referenced images. The next issue is that there were thousands of images collected every day and a limited amount of time and resources to find and bring people supplies. If some of that time and resources were spent on going through pictures to find people then rescue efforts would be less effective if they could even find these blue tarps, never mind communicating these results back to the rescue workers on the ground in a timely manner.

My goal is to create an algorithm that can effectively locate blue tarps from these images faster and ideally more accurately than a person could so people in Haiti can get the help they need. I will be testing 7 different models and choosing the best one. The models are Logistic Regression, Linear Discriminant Analysis (LDA), Quadratic Discriminate Analysis (QDA), K-nearest neighbor (KNN), Penalized Logistic Regression (elastic net penalty), Random Forest, and SVM (Linear, Radial, and Poly).

Training Data / EDA

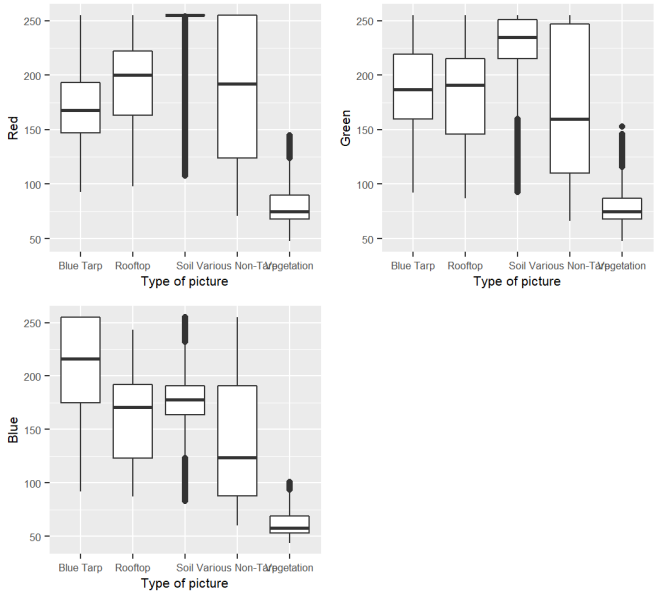
Load data, explore data, etc.

```
# Load Required Packages
library(tidyverse)
library(readr)
library(gridExtra)
library(caret)
library(glmnet)
library(e1071)
library(ROCR)
library(pROC)
library(plotly)
library(kernlab)
library(scatterplot3d)
library(mgcv)
```

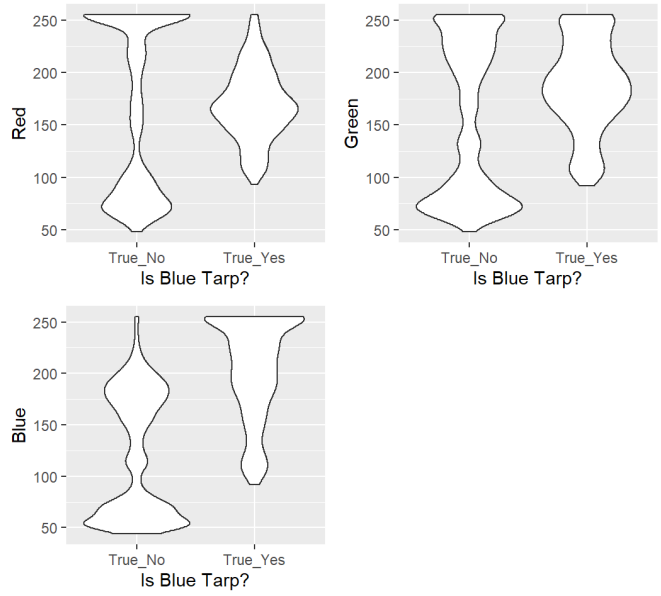
```
setwd("C:/Users/Tommy/OneDrive/Documents/R/University of Virginia/DS 6030")
data <- read_csv("HaitiPixels.csv")
data$Class <- as.factor(data$Class)
summary(data)
```

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

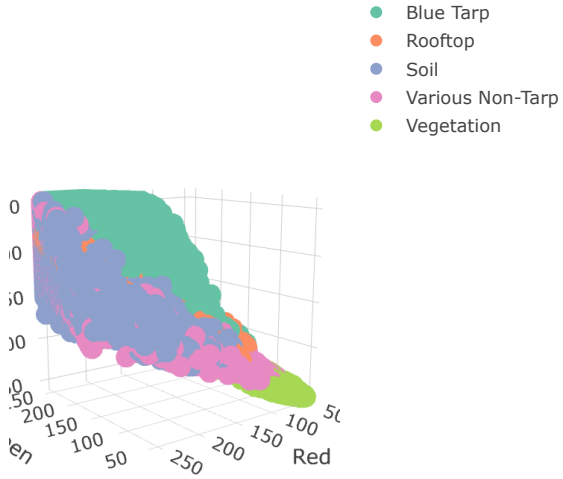
```
Red_graph <- ggplot(data, aes(x=Class, y = Red)) +
  geom_boxplot() +
  labs(x="Type of picture", y="Red") +
  theme(text = element_text(size=8))
Green_graph <- ggplot(data, aes(x=Class, y = Green)) +
  geom_boxplot() +
  labs(x="Type of picture", y="Green") +
  theme(text = element_text(size=8))
Blue_graph <- ggplot(data, aes(x=Class, y = Blue)) +
  geom_boxplot() +
  labs(x="Type of picture", y="Blue") +
  theme(text = element_text(size=8))
grid.arrange(Red_graph, Green_graph, Blue_graph, nrow=2)
```




```
data$Blue_Tarp <- "True_No"
for (num in c(1:nrow(data))){
  if (data$Class[num] == "Blue Tarp"){
    data$Blue_Tarp[num] <- "True_Yes"
  }
}
data$Blue_Tarp <- as.factor(data$Blue_Tarp)

Red_graph <- ggplot(data, aes(x=Blue_Tarp, y = Red)) +
  geom_violin() +
  labs(x="Is Blue Tarp?", y="Red")
Green_graph <- ggplot(data, aes(x=Blue_Tarp, y = Green)) +
  geom_violin() +
  labs(x="Is Blue Tarp?", y="Green")
Blue_graph <- ggplot(data, aes(x=Blue_Tarp, y = Blue)) +
  geom_violin() +
  labs(x="Is Blue Tarp?", y="Blue")
grid.arrange(Red_graph, Green_graph, Blue_graph, nrow=2)
```




```
#3D scatter plots
plot_ly(data, x=~Green, y=~Red, z=~Blue, type="scatter3d", color=~Class)
```



Looks like blue will be the most important factor in deciding accuracy of a picture the difficulty is the other data points close to the blue tarp class, looking at the other colors should help accuracy in the models.

Looking at the 3d scatterplot of the classes it looks like the Blue Tarp class is relatively separated but when the blue values are lower a rooftop could be mistaken for a Blue Tarp and to a lesser extent Soil and Various Non-Tarp

Model Training

Set-up

I have already created a factor column for Blue_Tarp. If factor is Yes then class is Blue Tarp otherwise not Blue Tarp.

```
class_row <- data[1]
data <- data[-1]
```

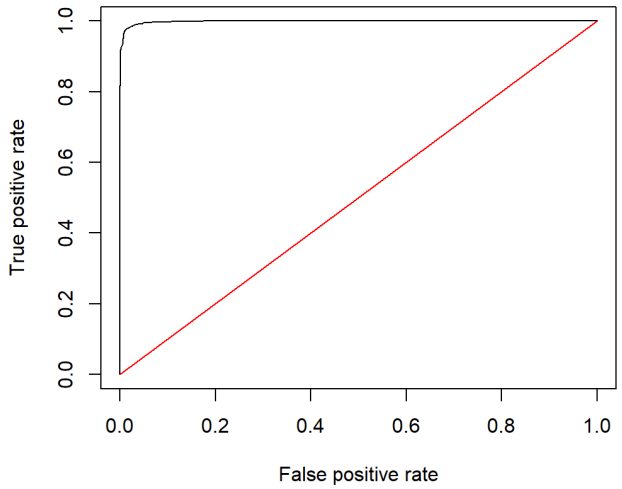
Dropped class row, keep separate just in case I need.

Logistic Regression

```
set.seed(3.14)
trControl = caret::trainControl(method='cv',
                                number=10,
                                savePredictions=TRUE, # required for thresholder
                                classProbs=TRUE,      # required for thresholder
                                allowParallel=TRUE)
modelFit_glm = caret::train(Blue_Tarp ~ ., data=data,
                 method='glm',
                 family='binomial',  # set the family for logistic regression
                 trControl=trControl)

modelFit_glm
```

```
#> Generalized Linear Model
#>
#> 63241 samples
#>     3 predictor
#>     2 classes: 'True_No', 'True_Yes'
#>
#> No pre-processing
#> Resampling: Cross-Validated (10 fold)
#> Summary of sample sizes: 56917, 56918, 56917, 56917, 56917, 56917, ...
#> Resampling results:
#>
#>   Accuracy   Kappa
#>   0.9952879  0.9206688
```

```
#get AUROC
preds<-predict(modelFit_glm$finalModel, newdata = data, type = "response")
rates<-prediction(preds, data$Blue_Tarp)
roc_result<-performance(rates,measure="tpr", x.measure="fpr")
plot(roc_result)
lines(x = c(0,1), y = c(0,1), col="red")
```




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

#Threshold
threshold_fun_glm <- thresholder(modelFit_glm, threshold = seq(0.05, 0.95, by=0.05), final = TRUE)
ind <- which.max(threshold_fun_glm$Accuracy)
threshold_glm <- threshold_fun_glm$prob_threshold[ind]

table(preds>threshold_glm, data$Blue_Tarp)
```

```
#>
#>         True_No True_Yes
#>   FALSE   61178      291
#>   TRUE       41     1731
```

```
#Accuracy
Accuracy_glm <- threshold_fun_glm$Accuracy[ind]

#TPR
TPR_glm <- threshold_fun_glm$Sensitivity[ind]

#FPR
FPR_glm <- 1-threshold_fun_glm$Specificity[ind]

#Precision
Precision_glm <- threshold_fun_glm$Precision[ind]

row_glm <- c(NA, AUROC_glm, threshold_glm, Accuracy_glm, TPR_glm, FPR_glm, Precision_glm)
```

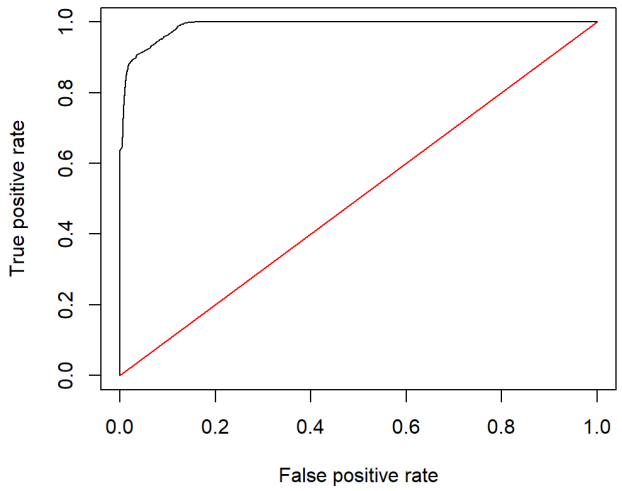
I run a 10-fold CV on a Logistic Regression with the caret package. I then used that model with the full data to predict what those values are and use the True values of the data to predict the AUC. I run a function to pick the threshold which maximizes accuracy and use the output of that function to find Accuracy, TPR, FPR, and Precision.

LDA

```
set.seed(3.14)
trControl = caret::trainControl(method='cv',
                                number=10,
                                savePredictions=TRUE, # required for thresholder
                                classProbs=TRUE,      # required for thresholder
                                allowParallel=TRUE)
modelFit_lda = caret::train(Blue_Tarp ~ ., data=data,
                method='lda',
                family='binomial',  # set the family for logistic regression
                trControl=trControl)

modelFit_lda
```

```
#> Linear Discriminant Analysis
#>
#> 63241 samples
#>     3 predictor
#>     2 classes: 'True_No', 'True_Yes'
#>
#> No pre-processing
#> Resampling: Cross-Validated (10 fold)
#> Summary of sample sizes: 56917, 56918, 56917, 56917, 56917, 56917, ...
#> Resampling results:
#>
#>   Accuracy   Kappa
#>   0.9839345  0.7528268
```

```
#get AUROC
preds<-predict(modelFit_lda$finalModel, newdata = data[-4], type = "response")$posterior[,2]
rates<-prediction(preds, data$Blue_Tarp)
roc_result<-performance(rates,measure="tpr", x.measure="fpr")
plot(roc_result)
lines(x = c(0,1), y = c(0,1), col="red")
```




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

#Threshold
threshold_fun_lda <- thresholder(modelFit_lda, threshold = seq(0.05, 0.95, by=0.05), final = TRUE)
ind <- which.max(threshold_fun_lda$Accuracy)
threshold_lda <- threshold_fun_lda$prob_threshold[ind]

table(preds>threshold_lda, data$Blue_Tarp)
```

```
#>
#>         True_No True_Yes
#>   FALSE   60438      322
#>   TRUE      781     1700
```

```
#Accuracy
Accuracy_lda <- threshold_fun_lda$Accuracy[ind]

#TPR
TPR_lda <- threshold_fun_lda$Sensitivity[ind]

#FPR
FPR_lda <- 1-threshold_fun_lda$Specificity[ind]

#Precision
Precision_lda <- threshold_fun_lda$Precision[ind]

row_lda <- c(NA, AUROC_lda, threshold_lda, Accuracy_lda, TPR_lda, FPR_lda, Precision_lda)
```

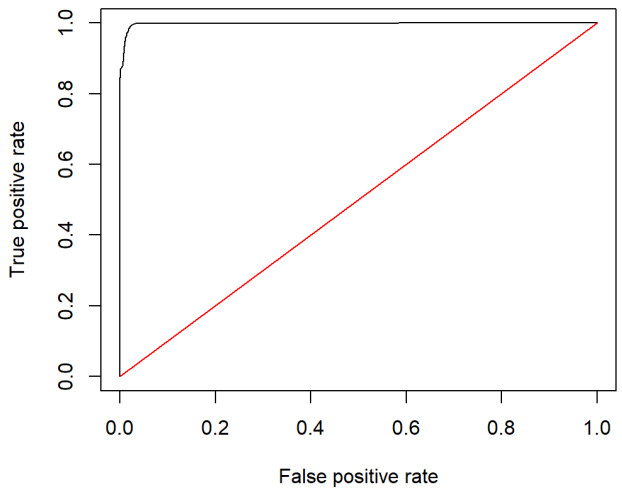
I run a 10-fold CV on LDA with the caret package. I then used that model with the full data to predict what those values are and use the True values of the data to predict the AUC. I run a function to pick the threshold which maximizes accuracy and use the output of that function to find Accuracy, TPR, FPR, and Precision.

QDA

```
set.seed(3.14)
trControl = caret::trainControl(method='cv',
                                number=10,
                                savePredictions=TRUE, # required for thresholder
                                classProbs=TRUE,      # required for thresholder
                                allowParallel=TRUE)
modelFit_qda = caret::train(Blue_Tarp ~ ., data=data,
                 method='qda',
                 family='binomial',  # set the family for logistic regression
                 trControl=trControl)

modelFit_qda
```

```
#> Quadratic Discriminant Analysis
#>
#> 63241 samples
#>     3 predictor
#>     2 classes: 'True_No', 'True_Yes'
#>
#> No pre-processing
#> Resampling: Cross-Validated (10 fold)
#> Summary of sample sizes: 56917, 56918, 56917, 56917, 56917, 56917, ...
#> Resampling results:
#>
#>   Accuracy   Kappa
#>   0.9945763  0.9052424
```

```
#get AUROC
preds<-predict(modelFit_qda$finalModel, newdata = data[-4], type = "response")$posterior[,2]
rates<-prediction(preds, data$Blue_Tarp)
roc_result<-performance(rates,measure="tpr", x.measure="fpr")
plot(roc_result)
lines(x = c(0,1), y = c(0,1), col="red")
```




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

#Threshold
threshold_fun_qda <- thresholder(modelFit_qda, threshold = seq(0.05, 0.95, by=0.05), final = TRUE)
ind <- which.max(threshold_fun_qda$Accuracy)
threshold_qda <- threshold_fun_qda$prob_threshold[ind]

table(preds>threshold_qda, data$Blue_Tarp)
```

```
#>
#>         True_No True_Yes
#>   FALSE   61207      347
#>   TRUE       12     1675
```

```
#Accuracy
Accuracy_qda <- threshold_fun_qda$Accuracy[ind]

#TPR
TPR_qda <- threshold_fun_qda$Sensitivity[ind]

#FPR
FPR_qda <- 1-threshold_fun_qda$Specificity[ind]

#Precision
Precision_qda <- threshold_fun_qda$Precision[ind]

row_qda <- c(NA, AUROC_qda, threshold_qda, Accuracy_qda, TPR_qda, FPR_qda, Precision_qda)
```

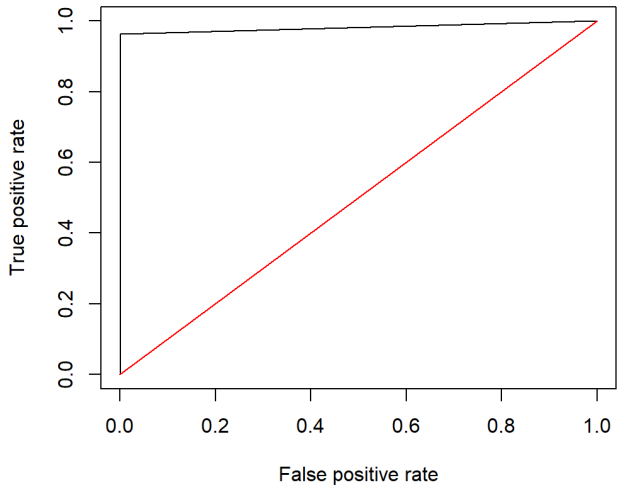
I run a 10-fold CV on QDA with the caret package. I then used that model with the full data to predict what those values are and use the True values of the data to predict the AUC. I run a function to pick the threshold which maximizes accuracy and use the output of that function to find Accuracy, TPR, FPR, and Precision.

KNN

```
set.seed(3.14)
trControl = caret::trainControl(method='cv',
                                number=10,
                                savePredictions=TRUE, # required for thresholder
                                classProbs=TRUE,      # required for thresholder
                                allowParallel=TRUE)

if (file.exists("modelFit_knn.rda")){
  load(file = "modelFit_knn.rda")
} else {
  set.seed(3.14)
  tuneGrid = expand.grid(k = c(3,5,7,9,12))
  modelFit_knn = caret::train(Blue_Tarp ~ ., data=data,
                              method='knn',
                              trControl=trControl,
                              tuneGrid = tuneGrid)
  save(modelFit_knn, file = "modelFit_knn.rda")
}

modelFit_knn
```

```
#> k-Nearest Neighbors
#>
#> 63241 samples
#>     3 predictor
#>     2 classes: 'True_No', 'True_Yes'
#>
#> No pre-processing
#> Resampling: Cross-Validated (10 fold)
#> Summary of sample sizes: 56917, 56918, 56917, 56917, 56917, 56917, ...
#> Resampling results across tuning parameters:
#>
#>   k   Accuracy   Kappa
#>    3  0.9972644  0.9557673
#>    5  0.9972170  0.9551514
#>    7  0.9973119  0.9567908
#>    9  0.9972328  0.9554510
#>   12  0.9971538  0.9541336
#>
#> Accuracy was used to select the optimal model using the largest value.
#> The final value used for the model was k = 7.
```

```
#get AUROC
preds<-modelFit_knn$pred[modelFit_knn$pred$k == modelFit_knn$bestTune$k,] %>% arrange(rowIndex) %>% select(pred)
preds <- ifelse(preds == "True_Yes", 1,0)
rates<-prediction(preds, data$Blue_Tarp)
roc_result<-performance(rates,measure="tpr", x.measure="fpr")
plot(roc_result)
lines(x = c(0,1), y = c(0,1), col="red")
```




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

#Threshold
threshold_knn <- NA

table(preds, data$Blue_Tarp)
```

```
#>
#> preds True_No True_Yes
#>     0   61123       74
#>     1      96     1948
```

```
#Accuracy
Accuracy_knn <- (table(preds, data$Blue_Tarp)[1] + table(preds, data$Blue_Tarp)[4]) / sum(table(preds, data$Blue_Tarp)[1:4])

#TPR
TPR_knn <- table(preds, data$Blue_Tarp)[4]/ sum(table(preds, data$Blue_Tarp)[3:4])

#FPR
FPR_knn <- table(preds, data$Blue_Tarp)[2]/ sum(table(preds, data$Blue_Tarp)[1:2])

#Precision
Precision_knn <- table(preds, data$Blue_Tarp)[4]/ (table(preds, data$Blue_Tarp)[2] + table(preds, data$Blue_Tarp)[4])

row_knn <- c(paste0("k = ", modelFit_knn$bestTune$k), AUROC_knn,
             threshold_knn, Accuracy_knn, TPR_knn, FPR_knn, Precision_knn)
```

I run a 10-fold CV on KNN with the caret package and pick the K which maximizes accuracy. I then used that model with the full data to predict what those values are and use the True values of the data to predict the AUC. I create a confusion matrix to calculate the Accuracy, TPR, FPR, and Precision.

This is the first time I save a model in different file with the mgcv package after I ran it. I will use this for every model after this.

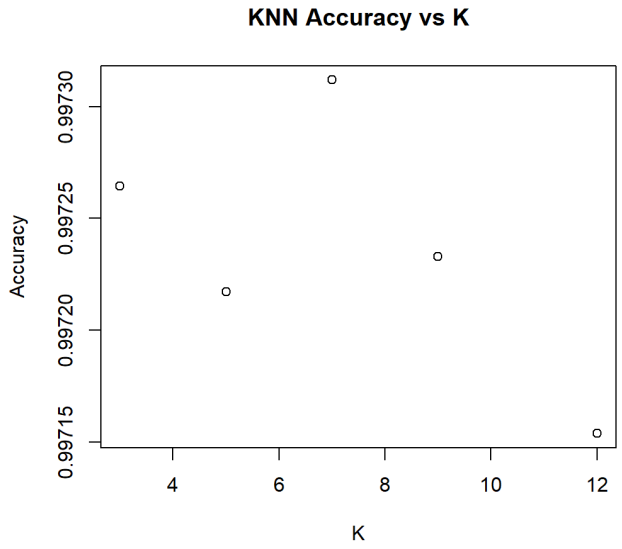
Tuning Parameter k

How were tuning parameter(s) selected? What value is used? Plots/Tables/etc.

```
modelFit_knn
```

```
#> k-Nearest Neighbors
#>
#> 63241 samples
#>     3 predictor
#>     2 classes: 'True_No', 'True_Yes'
#>
#> No pre-processing
#> Resampling: Cross-Validated (10 fold)
#> Summary of sample sizes: 56917, 56918, 56917, 56917, 56917, 56917, ...
#> Resampling results across tuning parameters:
#>
#>   k   Accuracy   Kappa
#>    3  0.9972644  0.9557673
#>    5  0.9972170  0.9551514
#>    7  0.9973119  0.9567908
#>    9  0.9972328  0.9554510
#>   12  0.9971538  0.9541336
#>
#> Accuracy was used to select the optimal model using the largest value.
#> The final value used for the model was k = 7.
```

```
plot(x = modelFit_knn$results$k, y= modelFit_knn$results$Accuracy, main = "KNN Accuracy vs K", xlab="K", ylab="Accuracy")
```



I checked a few values of K with a 10 fold CV and picked the value with the highest accuracy. K=7. Overall the accuracies were very similar and if a different metric was more important that could be explored to determine if there was a better KNN model than K=7.

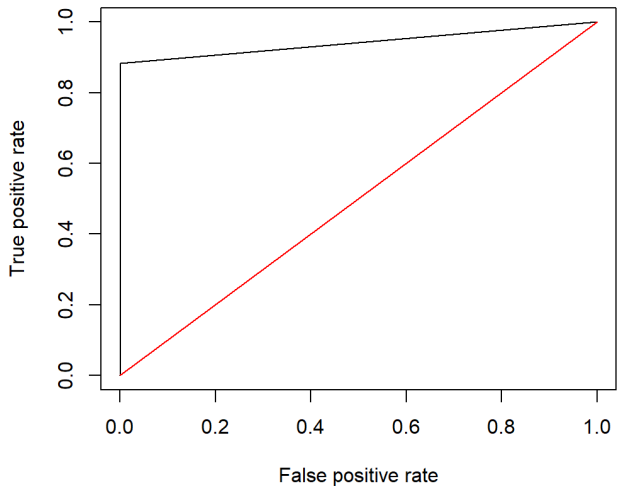
Penalized Logistic Regression (ElasticNet)

```
set.seed(3.14)
lambdas = 10^seq(-1, -5, by=-0.1)
trControl = caret::trainControl(method='cv',
                                number=10,
                                savePredictions=TRUE, # required for thresholder
                                classProbs=TRUE,      # required for thresholder
                                allowParallel=TRUE)

if (file.exists("modelFit_mix.rda")){
  load(file = "modelFit_mix.rda")
} else {
  set.seed(3.14)
  tuneGrid = expand.grid(lambda=lambdas, alpha=c(0,0.5,1))
  modelFit_mix = caret::train(Blue_Tarp ~ ., data=data,
                              method='glmnet',
                              family='binomial',  # set the family for logistic regression
                              trControl=trControl,
                              tuneGrid=tuneGrid)
  save(modelFit_mix, file = "modelFit_mix.rda")
  }

modelFit_mix
```

```
#> glmnet
#>
#> 63241 samples
#>     3 predictor
#>     2 classes: 'True_No', 'True_Yes'
#>
#> No pre-processing
#> Resampling: Cross-Validated (10 fold)
#> Summary of sample sizes: 56917, 56918, 56917, 56917, 56917, 56917, ...
#> Resampling results across tuning parameters:
#>
#>   alpha  lambda        Accuracy   Kappa
#>   0.0    1.000000e-05  0.9778783  0.462649602
#>   0.0    1.258925e-05  0.9778783  0.462649602
#>   0.0    1.584893e-05  0.9778783  0.462649602
#>   0.0    1.995262e-05  0.9778783  0.462649602
#>   0.0    2.511886e-05  0.9778783  0.462649602
#>   0.0    3.162278e-05  0.9778783  0.462649602
#>   0.0    3.981072e-05  0.9778783  0.462649602
#>   0.0    5.011872e-05  0.9778783  0.462649602
#>   0.0    6.309573e-05  0.9778783  0.462649602
#>   0.0    7.943282e-05  0.9778783  0.462649602
#>   0.0    1.000000e-04  0.9778783  0.462649602
#>   0.0    1.258925e-04  0.9778783  0.462649602
#>   0.0    1.584893e-04  0.9778783  0.462649602
#>   0.0    1.995262e-04  0.9778783  0.462649602
#>   0.0    2.511886e-04  0.9778783  0.462649602
#>   0.0    3.162278e-04  0.9778783  0.462649602
#>   0.0    3.981072e-04  0.9778783  0.462649602
#>   0.0    5.011872e-04  0.9778783  0.462649602
#>   0.0    6.309573e-04  0.9778783  0.462649602
#>   0.0    7.943282e-04  0.9778783  0.462649602
#>   0.0    1.000000e-03  0.9778783  0.462649602
#>   0.0    1.258925e-03  0.9778783  0.462649602
#>   0.0    1.584893e-03  0.9778783  0.462649602
#>   0.0    1.995262e-03  0.9778783  0.462649602
#>   0.0    2.511886e-03  0.9778783  0.462649602
#>   0.0    3.162278e-03  0.9778783  0.462649602
#>   0.0    3.981072e-03  0.9778783  0.462649602
#>   0.0    5.011872e-03  0.9746684  0.336357634
#>   0.0    6.309573e-03  0.9708733  0.158832044
#>   0.0    7.943282e-03  0.9687228  0.041174118
#>   0.0    1.000000e-02  0.9680271  0.000000000
#>   0.0    1.258925e-02  0.9680271  0.000000000
#>   0.0    1.584893e-02  0.9680271  0.000000000
#>   0.0    1.995262e-02  0.9680271  0.000000000
#>   0.0    2.511886e-02  0.9680271  0.000000000
#>   0.0    3.162278e-02  0.9680271  0.000000000
#>   0.0    3.981072e-02  0.9680271  0.000000000
#>   0.0    5.011872e-02  0.9680271  0.000000000
#>   0.0    6.309573e-02  0.9680271  0.000000000
#>   0.0    7.943282e-02  0.9680271  0.000000000
#>   0.0    1.000000e-01  0.9680271  0.000000000
#>   0.5    1.000000e-05  0.9951772  0.917774206
#>   0.5    1.258925e-05  0.9950348  0.915056967
#>   0.5    1.584893e-05  0.9949874  0.914100790
#>   0.5    1.995262e-05  0.9949558  0.913395562
#>   0.5    2.511886e-05  0.9949400  0.912870792
#>   0.5    3.162278e-05  0.9948925  0.911667729
#>   0.5    3.981072e-05  0.9947818  0.909534433
#>   0.5    5.011872e-05  0.9946079  0.906161383
#>   0.5    6.309573e-05  0.9944498  0.903093856
#>   0.5    7.943282e-05  0.9941652  0.897511062
#>   0.5    1.000000e-04  0.9939438  0.892939568
#>   0.5    1.258925e-04  0.9937857  0.889566633
#>   0.5    1.584893e-04  0.9934852  0.883516855
#>   0.5    1.995262e-04  0.9932164  0.877888571
#>   0.5    2.511886e-04  0.9925997  0.865296000
#>   0.5    3.162278e-04  0.9921570  0.856133691
#>   0.5    3.981072e-04  0.9915245  0.842824239
#>   0.5    5.011872e-04  0.9910185  0.831888012
#>   0.5    6.309573e-04  0.9902437  0.814873876
#>   0.5    7.943282e-04  0.9893740  0.795276797
#>   0.5    1.000000e-03  0.9883145  0.770538001
#>   0.5    1.258925e-03  0.9873026  0.745971721
#>   0.5    1.584893e-03  0.9861641  0.717164996
```

```
#>   0.5    1.995262e-03  0.9850730  0.688200962
#>   0.5    2.511886e-03  0.9841401  0.662252078
#>   0.5    3.162278e-03  0.9830965  0.632672063
#>   0.5    3.981072e-03  0.9819738  0.599081091
#>   0.5    5.011872e-03  0.9798865  0.532709383
#>   0.5    6.309573e-03  0.9758543  0.385392069
#>   0.5    7.943282e-03  0.9707626  0.153254015
#>   0.5    1.000000e-02  0.9681694  0.008555349
#>   0.5    1.258925e-02  0.9680271  0.000000000
#>   0.5    1.584893e-02  0.9680271  0.000000000
#>   0.5    1.995262e-02  0.9680271  0.000000000
#>   0.5    2.511886e-02  0.9680271  0.000000000
#>   0.5    3.162278e-02  0.9680271  0.000000000
#>   0.5    3.981072e-02  0.9680271  0.000000000
#>   0.5    5.011872e-02  0.9680271  0.000000000
#>   0.5    6.309573e-02  0.9680271  0.000000000
#>   0.5    7.943282e-02  0.9680271  0.000000000
#>   0.5    1.000000e-01  0.9680271  0.000000000
#>   1.0    1.000000e-05  0.9952246  0.919465807
#>   1.0    1.258925e-05  0.9952404  0.919716499
#>   1.0    1.584893e-05  0.9952562  0.919966773
#>   1.0    1.995262e-05  0.9952879  0.920431876
#>   1.0    2.511886e-05  0.9952562  0.919860244
#>   1.0    3.162278e-05  0.9952088  0.918942911
#>   1.0    3.981072e-05  0.9952246  0.919078773
#>   1.0    5.011872e-05  0.9951455  0.917628745
#>   1.0    6.309573e-05  0.9951613  0.917702413
#>   1.0    7.943282e-05  0.9951772  0.917902260
#>   1.0    1.000000e-04  0.9950348  0.915106284
#>   1.0    1.258925e-04  0.9950507  0.915223050
#>   1.0    1.584893e-04  0.9950507  0.914948852
#>   1.0    1.995262e-04  0.9949558  0.912904792
#>   1.0    2.511886e-04  0.9948293  0.910321175
#>   1.0    3.162278e-04  0.9946395  0.906749462
#>   1.0    3.981072e-04  0.9942442  0.898960923
#>   1.0    5.011872e-04  0.9938964  0.892106232
#>   1.0    6.309573e-04  0.9937066  0.888194345
#>   1.0    7.943282e-04  0.9933903  0.881438396
#>   1.0    1.000000e-03  0.9928369  0.870211955
#>   1.0    1.258925e-03  0.9921570  0.856098920
#>   1.0    1.584893e-03  0.9914138  0.840475418
#>   1.0    1.995262e-03  0.9904018  0.818362683
#>   1.0    2.511886e-03  0.9893740  0.795342731
#>   1.0    3.162278e-03  0.9877453  0.756814882
#>   1.0    3.981072e-03  0.9861957  0.717840193
#>   1.0    5.011872e-03  0.9846935  0.677759369
#>   1.0    6.309573e-03  0.9832546  0.637233672
#>   1.0    7.943282e-03  0.9813887  0.581124111
#>   1.0    1.000000e-02  0.9753641  0.365256533
#>   1.0    1.258925e-02  0.9682643  0.014218810
#>   1.0    1.584893e-02  0.9680271  0.000000000
#>   1.0    1.995262e-02  0.9680271  0.000000000
#>   1.0    2.511886e-02  0.9680271  0.000000000
#>   1.0    3.162278e-02  0.9680271  0.000000000
#>   1.0    3.981072e-02  0.9680271  0.000000000
#>   1.0    5.011872e-02  0.9680271  0.000000000
#>   1.0    6.309573e-02  0.9680271  0.000000000
#>   1.0    7.943282e-02  0.9680271  0.000000000
#>   1.0    1.000000e-01  0.9680271  0.000000000
#>
#> Accuracy was used to select the optimal model using the largest value.
#> The final values used for the model were alpha = 1 and lambda = 1.995262e-05.
```

```
#get AUROC
preds<-na.omit(modelFit_mix$pred[modelFit_mix$pred$alpha == modelFit_mix$bestTune$alpha,]
               [modelFit_mix$pred$lambda == modelFit_mix$bestTune$lambda,]) %>%
  arrange(rowIndex) %>% select(pred, rowIndex)
rowIndex <- preds$rowIndex
preds <- ifelse(preds$pred == "True_Yes", 1,0)
rates<-prediction(preds, data$Blue_Tarp[rowIndex])
roc_result<-performance(rates,measure="tpr", x.measure="fpr")
plot(roc_result)
lines(x = c(0,1), y = c(0,1), col="red")
```




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

#Threshold
threshold_fun_mix <- thresholder(modelFit_mix, threshold = seq(0.05, 0.95, by=0.05), final = TRUE)
ind <- which.max(threshold_fun_mix$Accuracy)
threshold_mix <- threshold_fun_mix$prob_threshold[ind]

table(preds, data$Blue_Tarp[rowIndex])
```

```
#>
#> preds True_No True_Yes
#>     0   61160      237
#>     1      59     1781
```

```
#Accuracy
Accuracy_mix <- threshold_fun_mix$Accuracy[ind]

#TPR
TPR_mix <- threshold_fun_mix$Sensitivity[ind]

#FPR
FPR_mix <- 1-threshold_fun_mix$Specificity[ind]

#Precision
Precision_mix <- threshold_fun_mix$Precision[ind]

row_mix <- c(paste0("alpha = ", modelFit_mix$bestTune$alpha, ", lambda = ", modelFit_mix$bestTune$lambda),
             AUROC_mix, threshold_mix, Accuracy_mix, TPR_mix, FPR_mix, Precision_mix)
```

I run a 10-fold CV on a penalized logistic regression model with the caret package picking the alpha and lambda that maximizes accuracy. I then used that model with the full data to predict what those values are and use the True values of the data to predict the AUC. I run a function to pick the threshold which maximizes accuracy and use the output of that function to find Accuracy, TPR, FPR, and Precision.

Seems like I lost 4 points of data somehow. I don't know how it occurred but I just removed them from my calculation. They were 3 True Yes values and 1 True No value.

Tuning Parameters

```
modelFit_mix
```

```
#> glmnet
#>
#> 63241 samples
#>     3 predictor
#>     2 classes: 'True_No', 'True_Yes'
#>
#> No pre-processing
#> Resampling: Cross-Validated (10 fold)
#> Summary of sample sizes: 56917, 56918, 56917, 56917, 56917, 56917, ...
#> Resampling results across tuning parameters:
#>
#>   alpha  lambda        Accuracy   Kappa
#>   0.0    1.000000e-05  0.9778783  0.462649602
#>   0.0    1.258925e-05  0.9778783  0.462649602
#>   0.0    1.584893e-05  0.9778783  0.462649602
#>   0.0    1.995262e-05  0.9778783  0.462649602
#>   0.0    2.511886e-05  0.9778783  0.462649602
#>   0.0    3.162278e-05  0.9778783  0.462649602
#>   0.0    3.981072e-05  0.9778783  0.462649602
#>   0.0    5.011872e-05  0.9778783  0.462649602
#>   0.0    6.309573e-05  0.9778783  0.462649602
#>   0.0    7.943282e-05  0.9778783  0.462649602
#>   0.0    1.000000e-04  0.9778783  0.462649602
#>   0.0    1.258925e-04  0.9778783  0.462649602
#>   0.0    1.584893e-04  0.9778783  0.462649602
#>   0.0    1.995262e-04  0.9778783  0.462649602
#>   0.0    2.511886e-04  0.9778783  0.462649602
#>   0.0    3.162278e-04  0.9778783  0.462649602
#>   0.0    3.981072e-04  0.9778783  0.462649602
#>   0.0    5.011872e-04  0.9778783  0.462649602
#>   0.0    6.309573e-04  0.9778783  0.462649602
#>   0.0    7.943282e-04  0.9778783  0.462649602
#>   0.0    1.000000e-03  0.9778783  0.462649602
#>   0.0    1.258925e-03  0.9778783  0.462649602
#>   0.0    1.584893e-03  0.9778783  0.462649602
#>   0.0    1.995262e-03  0.9778783  0.462649602
#>   0.0    2.511886e-03  0.9778783  0.462649602
#>   0.0    3.162278e-03  0.9778783  0.462649602
#>   0.0    3.981072e-03  0.9778783  0.462649602
#>   0.0    5.011872e-03  0.9746684  0.336357634
#>   0.0    6.309573e-03  0.9708733  0.158832044
#>   0.0    7.943282e-03  0.9687228  0.041174118
#>   0.0    1.000000e-02  0.9680271  0.000000000
#>   0.0    1.258925e-02  0.9680271  0.000000000
#>   0.0    1.584893e-02  0.9680271  0.000000000
#>   0.0    1.995262e-02  0.9680271  0.000000000
#>   0.0    2.511886e-02  0.9680271  0.000000000
#>   0.0    3.162278e-02  0.9680271  0.000000000
#>   0.0    3.981072e-02  0.9680271  0.000000000
#>   0.0    5.011872e-02  0.9680271  0.000000000
#>   0.0    6.309573e-02  0.9680271  0.000000000
#>   0.0    7.943282e-02  0.9680271  0.000000000
#>   0.0    1.000000e-01  0.9680271  0.000000000
#>   0.5    1.000000e-05  0.9951772  0.917774206
#>   0.5    1.258925e-05  0.9950348  0.915056967
#>   0.5    1.584893e-05  0.9949874  0.914100790
#>   0.5    1.995262e-05  0.9949558  0.913395562
#>   0.5    2.511886e-05  0.9949400  0.912870792
#>   0.5    3.162278e-05  0.9948925  0.911667729
#>   0.5    3.981072e-05  0.9947818  0.909534433
#>   0.5    5.011872e-05  0.9946079  0.906161383
#>   0.5    6.309573e-05  0.9944498  0.903093856
#>   0.5    7.943282e-05  0.9941652  0.897511062
#>   0.5    1.000000e-04  0.9939438  0.892939568
#>   0.5    1.258925e-04  0.9937857  0.889566633
#>   0.5    1.584893e-04  0.9934852  0.883516855
#>   0.5    1.995262e-04  0.9932164  0.877888571
#>   0.5    2.511886e-04  0.9925997  0.865296000
#>   0.5    3.162278e-04  0.9921570  0.856133691
#>   0.5    3.981072e-04  0.9915245  0.842824239
#>   0.5    5.011872e-04  0.9910185  0.831888012
#>   0.5    6.309573e-04  0.9902437  0.814873876
#>   0.5    7.943282e-04  0.9893740  0.795276797
#>   0.5    1.000000e-03  0.9883145  0.770538001
#>   0.5    1.258925e-03  0.9873026  0.745971721
#>   0.5    1.584893e-03  0.9861641  0.717164996
```

```
#>   0.5    1.995262e-03  0.9850730  0.688200962
#>   0.5    2.511886e-03  0.9841401  0.662252078
#>   0.5    3.162278e-03  0.9830965  0.632672063
#>   0.5    3.981072e-03  0.9819738  0.599081091
#>   0.5    5.011872e-03  0.9798865  0.532709383
#>   0.5    6.309573e-03  0.9758543  0.385392069
#>   0.5    7.943282e-03  0.9707626  0.153254015
#>   0.5    1.000000e-02  0.9681694  0.008555349
#>   0.5    1.258925e-02  0.9680271  0.000000000
#>   0.5    1.584893e-02  0.9680271  0.000000000
#>   0.5    1.995262e-02  0.9680271  0.000000000
#>   0.5    2.511886e-02  0.9680271  0.000000000
#>   0.5    3.162278e-02  0.9680271  0.000000000
#>   0.5    3.981072e-02  0.9680271  0.000000000
#>   0.5    5.011872e-02  0.9680271  0.000000000
#>   0.5    6.309573e-02  0.9680271  0.000000000
#>   0.5    7.943282e-02  0.9680271  0.000000000
#>   0.5    1.000000e-01  0.9680271  0.000000000
#>   1.0    1.000000e-05  0.9952246  0.919465807
#>   1.0    1.258925e-05  0.9952404  0.919716499
#>   1.0    1.584893e-05  0.9952562  0.919966773
#>   1.0    1.995262e-05  0.9952879  0.920431876
#>   1.0    2.511886e-05  0.9952562  0.919860244
#>   1.0    3.162278e-05  0.9952088  0.918942911
#>   1.0    3.981072e-05  0.9952246  0.919078773
#>   1.0    5.011872e-05  0.9951455  0.917628745
#>   1.0    6.309573e-05  0.9951613  0.917702413
#>   1.0    7.943282e-05  0.9951772  0.917902260
#>   1.0    1.000000e-04  0.9950348  0.915106284
#>   1.0    1.258925e-04  0.9950507  0.915223050
#>   1.0    1.584893e-04  0.9950507  0.914948852
#>   1.0    1.995262e-04  0.9949558  0.912904792
#>   1.0    2.511886e-04  0.9948293  0.910321175
#>   1.0    3.162278e-04  0.9946395  0.906749462
#>   1.0    3.981072e-04  0.9942442  0.898960923
#>   1.0    5.011872e-04  0.9938964  0.892106232
#>   1.0    6.309573e-04  0.9937066  0.888194345
#>   1.0    7.943282e-04  0.9933903  0.881438396
#>   1.0    1.000000e-03  0.9928369  0.870211955
#>   1.0    1.258925e-03  0.9921570  0.856098920
#>   1.0    1.584893e-03  0.9914138  0.840475418
#>   1.0    1.995262e-03  0.9904018  0.818362683
#>   1.0    2.511886e-03  0.9893740  0.795342731
#>   1.0    3.162278e-03  0.9877453  0.756814882
#>   1.0    3.981072e-03  0.9861957  0.717840193
#>   1.0    5.011872e-03  0.9846935  0.677759369
#>   1.0    6.309573e-03  0.9832546  0.637233672
#>   1.0    7.943282e-03  0.9813887  0.581124111
#>   1.0    1.000000e-02  0.9753641  0.365256533
#>   1.0    1.258925e-02  0.9682643  0.014218810
#>   1.0    1.584893e-02  0.9680271  0.000000000
#>   1.0    1.995262e-02  0.9680271  0.000000000
#>   1.0    2.511886e-02  0.9680271  0.000000000
#>   1.0    3.162278e-02  0.9680271  0.000000000
#>   1.0    3.981072e-02  0.9680271  0.000000000
#>   1.0    5.011872e-02  0.9680271  0.000000000
#>   1.0    6.309573e-02  0.9680271  0.000000000
#>   1.0    7.943282e-02  0.9680271  0.000000000
#>   1.0    1.000000e-01  0.9680271  0.000000000
#>
#> Accuracy was used to select the optimal model using the largest value.
#> The final values used for the model were alpha = 1 and lambda = 1.995262e-05.
```

```
modelFit_mix$bestTune
```

	alpha	lambda
86	1	2e-05

I tested a few alphas and picked a lambda around the point where accuracy didn't increase anymore. I picked a alpha = 1 or a lasso penalty and the lambda = 1.99526231496888e-05.

Random Forest

```
set.seed(3.14)
#sampsize <- seq(10, 100, by=10)
#ntree <- seq(100, 1000, by=100)
trControl = caret::trainControl(method='cv',
                                number=10,
                                savePredictions=TRUE, # required for thresholder
                                classProbs=TRUE,      # required for thresholder
                                allowParallel=TRUE)
tuneGrid = expand.grid(.mtry = c(2,3))

if (file.exists("df_sampsize.rda")){
  load(file = "df_sampsize.rda")
} else {
  df_sampsize <- data.frame()
  for (sampsize in seq(10, 100, by=10)){
    set.seed(3.14)
    modelFit_rf = caret::train(Blue_Tarp ~ ., data=data,
                               method='rf',
                               metric="Accuracy",
                               ntree = 200,
                               sampsize = sampsize,
                               trControl=trControl,
                               tuneGrid = tuneGrid)
    df_sampsize <- rbind(df_sampsize, c(modelFit_rf$results[which(modelFit_rf$results$mtry == modelFit_rf$bestTune$mtry),]
                                        ,sampsize = sampsize))
    save(df_sampsize, file = "df_sampsize.rda")
  }
}

df_sampsize
```

mtry	Accuracy	Kappa	AccuracySD	KappaSD	sampsize
2	0.9752377	0.4128914	0.0032741	0.0826203	10
3	0.9781789	0.5126607	0.0036070	0.0532444	20
3	0.9787007	0.5026609	0.0024708	0.0649502	30
3	0.9799656	0.5347604	0.0011969	0.0404152	40
3	0.9799814	0.5350778	0.0013396	0.0442785	50
3	0.9802660	0.5446600	0.0012491	0.0402868	60
3	0.9808036	0.5621314	0.0012645	0.0400240	70
3	0.9819896	0.5996335	0.0010000	0.0300818	80
3	0.9821635	0.6046104	0.0012437	0.0377266	90
3	0.9828593	0.6257685	0.0009655	0.0281029	100

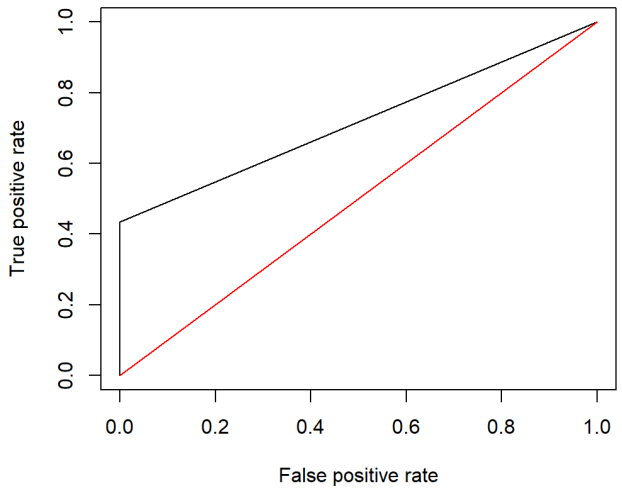
```
if (file.exists("df_ntree.rda")){
  load(file = "df_ntree.rda")
} else {
  df_ntree <- data.frame()
  for (ntree in seq(100, 1000, by=100)){
    set.seed(3.14)
    modelFit_rf = caret::train(Blue_Tarp ~ ., data=data,
                               method='rf',
                               metric="Accuracy",
                               ntree = ntree,
                               sampsize = 50,
                               trControl=trControl,
                               tuneGrid = tuneGrid)
    df_ntree <- rbind(df_ntree, c(modelFit_rf$results[which(modelFit_rf$results$mtry == modelFit_rf$bestTune$mtry),]
                                  ,ntree = ntree))
  save(df_ntree, file = "df_ntree.rda")
  }
}

df_ntree
```

mtry	Accuracy	Kappa	AccuracySD	KappaSD	ntree
3	0.9791432	0.5256820	0.0020287	0.0459806	100
3	0.9799814	0.5350778	0.0013396	0.0442785	200
3	0.9798865	0.5320867	0.0012573	0.0411404	300
3	0.9800763	0.5383454	0.0012816	0.0426086	400
3	0.9802028	0.5426655	0.0012007	0.0390854	500
3	0.9800604	0.5378333	0.0012764	0.0416368	600
3	0.9800762	0.5383791	0.0012667	0.0410913	700
3	0.9801711	0.5414490	0.0013059	0.0425929	800
3	0.9800604	0.5377289	0.0013420	0.0437244	900
3	0.9802344	0.5435221	0.0013223	0.0426983	1000

```
set.seed(3.14)
modelFit_rf = caret::train(Blue_Tarp ~ ., data=data,
                 method='rf',
                 metric="Accuracy",
                 ntree = 200,
                 sampsize = 100,
                 trControl=trControl,
                 tuneGrid = tuneGrid)

if (file.exists("modelFit_rf.rda")){
  load(file = "modelFit_rf.rda")
} else {
  set.seed(3.14)
  modelFit_rf = caret::train(Blue_Tarp ~ ., data=data,
                             method='rf',
                             metric="Accuracy",
                             ntree = 200,
                             sampsize = 100,
                             trControl=trControl,
                             tuneGrid = tuneGrid)
  save(modelFit_rf, file = "modelFit_rf.rda")
}

modelFit_rf
```

```
#> Random Forest
#>
#> 63241 samples
#>     3 predictor
#>     2 classes: 'True_No', 'True_Yes'
#>
#> No pre-processing
#> Resampling: Cross-Validated (10 fold)
#> Summary of sample sizes: 56917, 56918, 56917, 56917, 56917, 56917, ...
#> Resampling results across tuning parameters:
#>
#>   mtry  Accuracy   Kappa
#>   2     0.9811673  0.5744041
#>   3     0.9828593  0.6257685
#>
#> Accuracy was used to select the optimal model using the largest value.
#> The final value used for the model was mtry = 3.
```

```
#get AUROC
preds<-predict(modelFit_rf$finalModel, newdata = data, type="response")
preds <- ifelse(preds == "True_Yes", 1,0)
rates<-prediction(preds, data$Blue_Tarp)
roc_result<-performance(rates,measure="tpr", x.measure="fpr")
plot(roc_result)
lines(x = c(0,1), y = c(0,1), col="red")
```




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

#Threshold
threshold_fun_rf <- thresholder(modelFit_rf, threshold = seq(0.1, 0.95, by=0.05), final = TRUE)
ind <- which.max(threshold_fun_rf$Accuracy)
threshold_rf <- threshold_fun_rf$prob_threshold[ind]

table(preds>threshold_rf, data$Blue_Tarp)
```

```
#>
#>         True_No True_Yes
#>   FALSE   61219     1142
#>   TRUE        0      880
```

```
#Accuracy
Accuracy_rf <- threshold_fun_rf$Accuracy[ind]

#TPR
TPR_rf <- threshold_fun_rf$Sensitivity[ind]

#FPR
FPR_rf <- 1-threshold_fun_rf$Specificity[ind]

#Precision
Precision_rf <- threshold_fun_rf$Precision[ind]

row_rf <- c(paste0("mtry = ", modelFit_rf$bestTune$mtry, ", sampsize = 100, ntree=200"),
            AUROC_rf, threshold_rf, Accuracy_rf, TPR_rf, FPR_rf, Precision_rf)
```

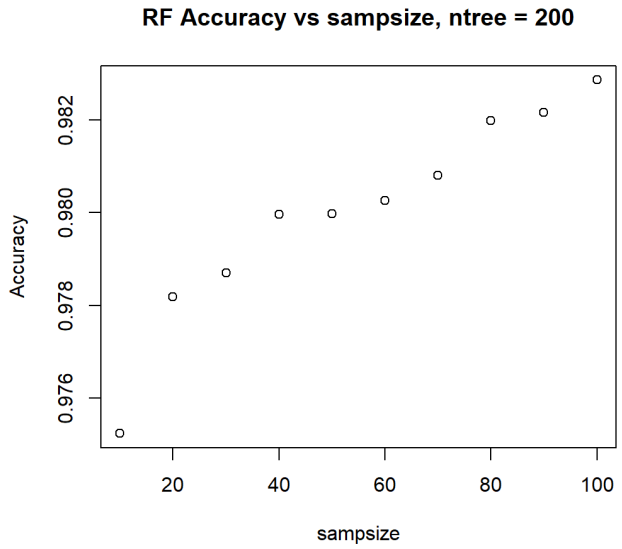
I run a 10-fold CV on a Random FOrest model with the caret package picking the mtry and sampsize that maximizes accuracy. I picked ntree that had a good accuracy while balancing computation cost. I then used that model with the full data to predict what those values are and use the True values of the data to predict the AUC. I run a function to pick the threshold which maximizes accuracy and use the output of that function to find Accuracy, TPR, FPR, and Precision.

Tuning Parameters

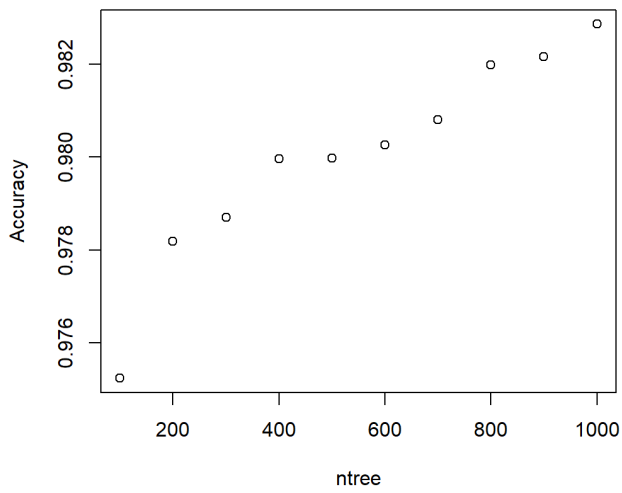
```
modelFit_rf
```

```
#> Random Forest
#>
#> 63241 samples
#>     3 predictor
#>     2 classes: 'True_No', 'True_Yes'
#>
#> No pre-processing
#> Resampling: Cross-Validated (10 fold)
#> Summary of sample sizes: 56917, 56918, 56917, 56917, 56917, 56917, ...
#> Resampling results across tuning parameters:
#>
#>   mtry  Accuracy   Kappa
#>   2     0.9811673  0.5744041
#>   3     0.9828593  0.6257685
#>
#> Accuracy was used to select the optimal model using the largest value.
#> The final value used for the model was mtry = 3.
```

```
plot(x = df_sampsize$sampsize, y= df_sampsize$Accuracy, main = "RF Accuracy vs sampsize, ntree = 200",
     xlab="sampsize", ylab="Accuracy")
```




```
plot(x = df_ntree$ntree, y= df_sampsize$Accuracy, main = "RF Accuracy vs ntree, sampsize=50", xlab="ntree", ylab="Accuracy")
```



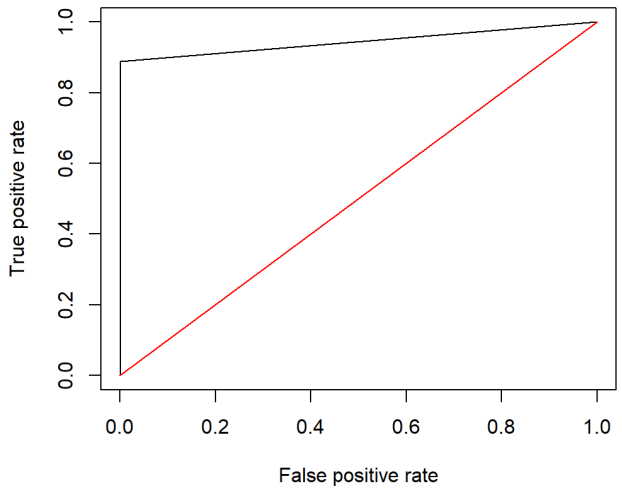
I manually picked my sampsize and ntree values. I ran 10 to 100 in intervals of 10 for sampsize and 100 seems to be the best. I picked the best sampsize of 100 and tested it on ntree of 100-1000 in 100 intervals but there was very little accuracy change between these values so I picked sampsize=50 and ntree 100-1000 and the accuracy difference between increasing ntree from 200 to >200 doesn't match the computation cost so I picked ntree = 200 as the optimal value. The model picked the optimal value for mtry = 3.

Support Vector Machines (SVM)

```
setwd("C:/Users/Tommy/OneDrive/Documents/R/University of Virginia/DS 6030")
set.seed(3.14)
trControl = caret::trainControl(method='cv',
                                number=10,
                                savePredictions=TRUE, # required for thresholder
                                classProbs=TRUE,      # required for thresholder
                                allowParallel=TRUE)

#linear
if (file.exists("modelFit_svm_linear.rda")){
  load(file = "modelFit_svm_linear.rda")
} else {
  set.seed(3.14)
  tuneGrid = expand.grid(C=c(0.01,0.1,1,5,10))
  modelFit_svm_linear = caret::train(Blue_Tarp ~ ., data=data,
                                     method='svmLinear',
                                     trControl=trControl,
                                     tuneGrid = tuneGrid)
  save(modelFit_svm_linear, file = "modelFit_svm_linear.rda")
}

modelFit_svm_linear
```

```
#> Support Vector Machines with Linear Kernel
#>
#> 63241 samples
#>     3 predictor
#>     2 classes: 'True_No', 'True_Yes'
#>
#> No pre-processing
#> Resampling: Cross-Validated (10 fold)
#> Summary of sample sizes: 56917, 56918, 56917, 56917, 56917, 56917, ...
#> Resampling results across tuning parameters:
#>
#>   C      Accuracy   Kappa
#>    0.01  0.9913822  0.8597934
#>    0.10  0.9952246  0.9190986
#>    1.00  0.9954143  0.9228772
#>    5.00  0.9953827  0.9222954
#>   10.00  0.9953827  0.9222954
#>
#> Accuracy was used to select the optimal model using the largest value.
#> The final value used for the model was C = 1.
```

```
#get AUROC
preds <- modelFit_svm_linear$pred[modelFit_svm_linear$pred$C == modelFit_svm_linear$bestTune$C,] %>%
  arrange(rowIndex) %>%
  select(pred, rowIndex)
rowIndex <- preds$rowIndex
preds <- ifelse(preds$pred == "True_Yes", 1,0)
rates<-prediction(preds, data$Blue_Tarp[rowIndex])
roc_result<-performance(rates,measure="tpr", x.measure="fpr")
plot(roc_result)
lines(x = c(0,1), y = c(0,1), col="red")
```



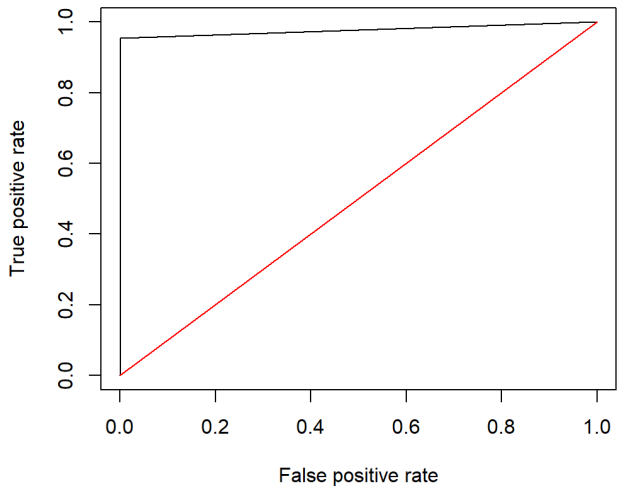

```
auc<-performance(rates, measure = "auc")
AUROC_svm_linear <- auc@y.values[[1]]

#Threshold
threshold_fun_svm <- thresholder(modelFit_svm_linear, threshold = seq(0.05, 0.95, by=0.05), final = TRUE)
ind <- which.max(threshold_fun_svm$Accuracy)
threshold_svm_linear <- threshold_fun_svm$prob_threshold[ind]

table(preds>threshold_svm_linear, data$Blue_Tarp)
```

```
#>
#>         True_No True_Yes
#>   FALSE   61156      227
#>   TRUE       63     1795
```

```
#Accuracy
Accuracy_svm_linear <- threshold_fun_svm$Accuracy[ind]

#TPR
TPR_svm_linear <- threshold_fun_svm$Sensitivity[ind]

#FPR
FPR_svm_linear <- 1-threshold_fun_svm$Specificity[ind]

#Precision
Precision_svm_linear <- threshold_fun_svm$Precision[ind]

row_svm_linear <- c(paste0("C = ", modelFit_svm_linear$bestTune$C), AUROC_svm_linear, threshold_svm_linear,
                    Accuracy_svm_linear, TPR_svm_linear, FPR_svm_linear, Precision_svm_linear)

#radial
if (file.exists("modelFit_svm_radial.rda")){
  load(file = "modelFit_svm_radial.rda")
} else {
  set.seed(3.14)
  tuneGrid = expand.grid(C=c(1, 10, 50, 100, 250), sigma=c(1, 2, 4, 8, 16, 32, 64))
  modelFit_svm_radial = caret::train(Blue_Tarp ~ ., data=data,
                                     method='svmRadial',
                                     trControl=trControl,
                                     tuneGrid = tuneGrid)
  save(modelFit_svm_radial, file = "modelFit_svm_radial.rda")
}

modelFit_svm_radial
```

```
#> Support Vector Machines with Radial Basis Function Kernel
#>
#> 63241 samples
#>     3 predictor
#>     2 classes: 'True_No', 'True_Yes'
#>
#> No pre-processing
#> Resampling: Cross-Validated (10 fold)
#> Summary of sample sizes: 56917, 56918, 56917, 56917, 56917, 56917, ...
#> Resampling results across tuning parameters:
#>
#>   C    sigma  Accuracy   Kappa
#>     1   1     0.9964105  0.9414927
#>     1   2     0.9964896  0.9424790
#>     1   4     0.9966952  0.9456881
#>     1   8     0.9969798  0.9505345
#>     1  16     0.9970747  0.9521487
#>     1  32     0.9971696  0.9537344
#>     1  64     0.9970747  0.9522412
#>    10   1     0.9968691  0.9489701
#>    10   2     0.9969324  0.9500128
#>    10   4     0.9970905  0.9526358
#>    10   8     0.9970114  0.9511014
#>    10  16     0.9974226  0.9579718
#>    10  32     0.9972486  0.9552977
#>    10  64     0.9971538  0.9536172
#>    50   1     0.9969640  0.9505381
#>    50   2     0.9970589  0.9522066
#>    50   4     0.9971696  0.9541100
#>    50   8     0.9973909  0.9575676
#>    50  16     0.9973593  0.9569754
#>    50  32     0.9973119  0.9562429
#>    50  64     0.9967110  0.9461486
#>   100   1     0.9969640  0.9505857
#>   100   2     0.9971537  0.9537597
#>   100   4     0.9972644  0.9557249
#>   100   8     0.9973277  0.9567097
#>   100  16     0.9972328  0.9549279
#>   100  32     0.9970905  0.9524695
#>   100  64     0.9964264  0.9413656
#>   250   1     0.9970272  0.9515795
#>   250   2     0.9972961  0.9561115
#>   250   4     0.9973119  0.9564845
#>   250   8     0.9973119  0.9564953
#>   250  16     0.9972328  0.9550227
#>   250  32     0.9969956  0.9508268
#>   250  64     0.9963315  0.9395983
#>
#> Accuracy was used to select the optimal model using the largest value.
#> The final values used for the model were sigma = 16 and C = 10.
```

```
#get AUROC
preds <- na.omit(modelFit_svm_radial$pred[modelFit_svm_radial$pred$C == modelFit_svm_radial$bestTune$C,]
                 [modelFit_svm_radial$pred$sigma == modelFit_svm_radial$bestTune$sigma,])%>%
  arrange(rowIndex) %>%
  select(pred, rowIndex)
rowIndex <- preds$rowIndex
preds <- ifelse(preds$pred == "True_Yes", 1,0)
rates<-prediction(preds, data$Blue_Tarp[rowIndex])
roc_result<-performance(rates,measure="tpr", x.measure="fpr")
plot(roc_result)
lines(x = c(0,1), y = c(0,1), col="red")
```



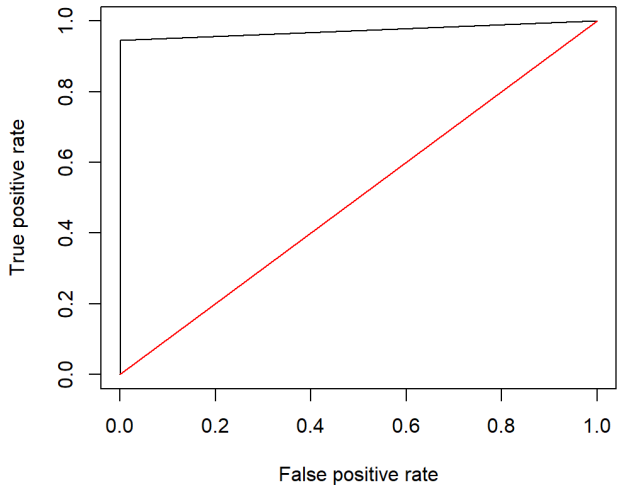

```
auc<-performance(rates, measure = "auc")
AUROC_svm_radial <- auc@y.values[[1]]

#Threshold
threshold_fun_svm <- thresholder(modelFit_svm_radial, threshold = seq(0.05, 0.95, by=0.05), final = TRUE)
ind <- which.max(threshold_fun_svm$Accuracy)
threshold_svm_radial <- threshold_fun_svm$prob_threshold[ind]

table(preds>threshold_svm_radial, data$Blue_Tarp[rowIndex])
```

```
#>
#>         True_No True_Yes
#>   FALSE   61158       93
#>   TRUE       62     1922
```

```
#Accuracy
Accuracy_svm_radial <- threshold_fun_svm$Accuracy[ind]

#TPR
TPR_svm_radial <- threshold_fun_svm$Sensitivity[ind]

#FPR
FPR_svm_radial <- 1-threshold_fun_svm$Specificity[ind]

#Precision
Precision_svm_radial <- threshold_fun_svm$Precision[ind]

row_svm_radial <- c(paste0("C = ", modelFit_svm_radial$bestTune$C, ", sigma = ", modelFit_svm_radial$bestTune$sigma),
                    AUROC_svm_radial, threshold_svm_radial, Accuracy_svm_radial, TPR_svm_radial,
                    FPR_svm_radial, Precision_svm_radial)

#Poly
if (file.exists("modelFit_svm_poly.rda")){
  load(file = "modelFit_svm_poly.rda")
} else {
  set.seed(3.14)
  tuneGrid = expand.grid(C=c(1, 10, 50, 100, 250), scale=1, degree = c(2,3))
  modelFit_svm_poly = caret::train(Blue_Tarp ~ ., data=data,
                                   method='svmPoly',
                                   trControl=trControl,
                                   tuneGrid = tuneGrid)
  save(modelFit_svm_poly, file = "modelFit_svm_poly.rda")
}

modelFit_svm_poly
```

```
#> Support Vector Machines with Polynomial Kernel
#>
#> 63241 samples
#>     3 predictor
#>     2 classes: 'True_No', 'True_Yes'
#>
#> No pre-processing
#> Resampling: Cross-Validated (10 fold)
#> Summary of sample sizes: 56917, 56918, 56917, 56917, 56917, 56917, ...
#> Resampling results across tuning parameters:
#>
#>   C    degree  Accuracy   Kappa
#>     1  2       0.9956832  0.9285942
#>     1  3       0.9963789  0.9406803
#>    10  2       0.9957306  0.9292269
#>    10  3       0.9967584  0.9472267
#>    50  2       0.9962366  0.9378296
#>    50  3       0.9966793  0.9459652
#>   100  2       0.9963473  0.9397612
#>   100  3       0.9968375  0.9485395
#>   250  2       0.9963947  0.9405258
#>   250  3       0.9968375  0.9484323
#>
#> Tuning parameter 'scale' 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 degree = 3, scale = 1 and C = 100.
```

```
#get AUROC
preds <- na.omit(modelFit_svm_poly$pred[modelFit_svm_poly$pred$C == modelFit_svm_poly$bestTune$C,]
                 [modelFit_svm_poly$pred$degree == modelFit_svm_poly$bestTune$degree,])%>%
  arrange(rowIndex) %>%
  select(pred, rowIndex)
rowIndex <- preds$rowIndex
preds <- ifelse(preds$pred == "True_Yes", 1,0)
rates<-prediction(preds, data$Blue_Tarp[rowIndex])
roc_result<-performance(rates,measure="tpr", x.measure="fpr")
plot(roc_result)
lines(x = c(0,1), y = c(0,1), col="red")
```




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

#Threshold
threshold_fun_svm <- thresholder(modelFit_svm_poly, threshold = seq(0.05, 0.95, by=0.05), final = TRUE)
ind <- which.max(threshold_fun_svm$Accuracy)
threshold_svm_poly <- threshold_fun_svm$prob_threshold[ind]

table(preds>threshold_svm_poly, data$Blue_Tarp[rowIndex])
```

```
#>
#>         True_No True_Yes
#>   FALSE   61145      108
#>   TRUE       75     1907
```

```
#Accuracy
Accuracy_svm_poly <- threshold_fun_svm$Accuracy[ind]

#TPR
TPR_svm_poly <- threshold_fun_svm$Sensitivity[ind]

#FPR
FPR_svm_poly <- 1-threshold_fun_svm$Specificity[ind]

#Precision
Precision_svm_poly <- threshold_fun_svm$Precision[ind]

row_svm_poly <- c(paste0("C = ", modelFit_svm_poly$bestTune$C, ", degree = ", modelFit_svm_poly$bestTune$degree),
                  AUROC_svm_poly, threshold_svm_poly, Accuracy_svm_poly, TPR_svm_poly,
                  FPR_svm_poly, Precision_svm_poly)
```

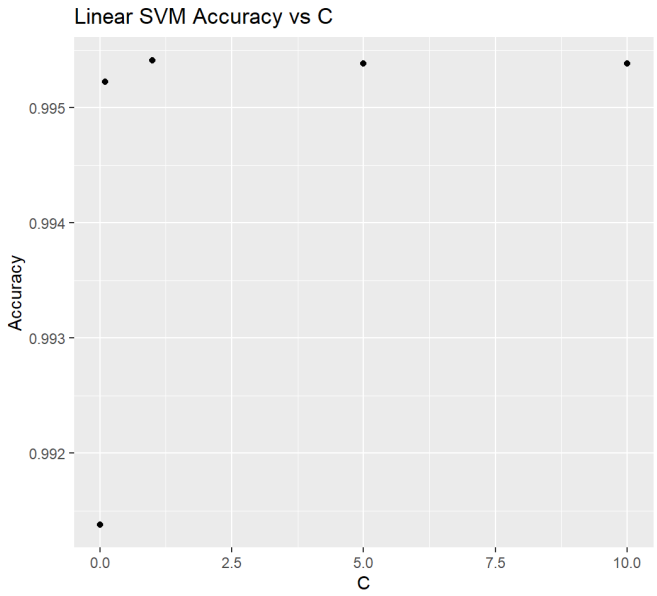
I can't figure out how to plot SVM with kernlab. In theory "kernlab::plot(modelFit_svm_linear$finalModel, data = data)" should work but doesn't.

I ran a 10-fold CV on three SVM models (Linear, radial, and polynomial) with parameters optimized for accuracy. I then used the classes the model predicted with the True values of the data to predict the AUC. I ran a function to pick the threshold which maximizes accuracy and use the output of that function to find Accuracy, TPR, FPR, and Precision.

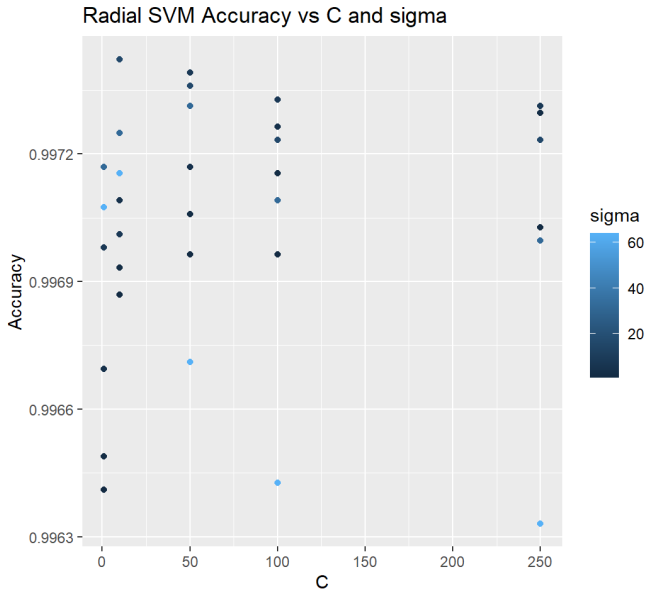
I seemed to have lost 6 points on Radial SVM and Poly SVM. I don't know why.

Tuning Parameters

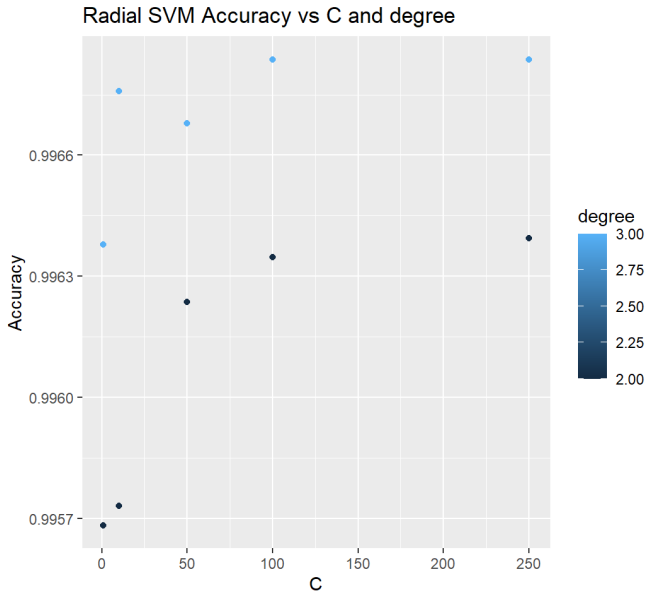
```
ggplot(modelFit_svm_linear$results, aes(x=C, y=Accuracy)) +
  geom_point() +
  ggtitle("Linear SVM Accuracy vs C")
```




```
ggplot(modelFit_svm_radial$results, aes(x=C, y=Accuracy, color=sigma)) +
  geom_point() +
  ggtitle("Radial SVM Accuracy vs C and sigma")
```




```
ggplot(modelFit_svm_poly$results, aes(x=C, y=Accuracy, color=degree)) +
  geom_point() +
  ggtitle("Radial SVM Accuracy vs C and degree")
```



The optimal parameters for linear: C=1, radial: C=10 sigma=16, and poly: C=100 degree=3 Scale=1.

Threshold Selection

I used thresholder from the caret package looking at a range of thresholds from 0.05-0.95 in 0.05 intervals and picked the threshold which had the highest accuracy.

I picked the threshold which would give the highest accuracy but in reality if a different metric was more important then that could be used to determine the best threshold. A good example of this is assuming that there simply aren't enough resources to get to everyone who is predicted to have a blue tarp then Precision would be the most important metric to maximize for as that would lead to the least amount of rescue workers going on trips where there is no blue tarp. Another further example is when there are more rescue workers than people under blue tarps who need help. In this scenario it is best to maximize True Positive Rate (TPR) as this reduces the chance that any blue tarp gets missed. With this understanding on scenarios, it is best to pick a threshold which maximizes Precision and TPR as a ratio. Example: ratio Precision + (1-ratio TPR). I didn't do this as I believe information is lacking and the model might not be remade if another metric is found to be more important. If the model is not remade I believe maximizing the models for Accuracy is the best option as it is the most general metric.

ROC Curves

All Models used in sample data to build their predicted data in the ROC curve and out of sample data for the true data in the ROC curve.

Results (Cross-Validation)

```
Performance_Table <- as.data.frame(matrix(data = c(row_glm, row_lda, row_qda, row_knn, row_mix, row_rf, row_svm_linear,
                                                   row_svm_radial, row_svm_poly), nrow = 9, ncol = 7, byrow = TRUE,
                                          dimnames = list(c("Log Reg", "LDA", "QDA", "KNN", "Penalized Log Reg",
                                                            "Random Forest", "SVM Linear", "SVM Radial",
                                                            "SVM Poly 3"),c("Tuning", "AUROC", "Threshold",
                                                                            "Accuracy", "TPR", "FPR", "Precision"))))
Performance_Table
```

	Tuning	AUROC	Threshold	Accuracy	TPR	FPR	Precision
Log Reg	NA	0.998506949374036	0.75	0.995714792487518	0.998252186481477	0.0811125201190069	0.9973238117799
LDA	NA	0.988876770009067	0.15	0.984851612331256	0.992355335323294	0.242340145344584	0.9919993715264
QDA	NA	0.998217543931769	0.6	0.994750228099733	0.999591634051421	0.151833877969078	0.9950090273664
KNN	k = 7	0.980917215550618	NA	0.997311870463781	0.963402571711177	0.00156814060994136	0.9530332681017
Penalized Log Reg	alpha = 1, lambda = 1.99526231496888e-05	0.940796617016382	0.75	0.995635728601528	0.998382862731068	0.0875408476808271	0.9971127246127
Random Forest	mtry = 3, sampsize = 100, ntree=200	0.717606330365974	0.85	0.984519521517295	0.98750391817929	0.105823538018826	0.9964733628871
SVM Linear	C = 1	0.943352911824776	0.75	0.995746415538549	0.998382857393858	0.0840779398136858	0.9972265074921
SVM Radial	C = 10, sigma = 16	0.976416706455909	0.7	0.997501638769189	0.998856566804443	0.0435180217529142	0.9985631103626
SVM Poly 3	C = 100, degree = 3	0.97258844763587	0.65	0.996900715744338	0.998170495145247	0.0415548944056967	0.9986273060616

Each of the Tuning values were selected on maximizing accuracy. I picked the threshold which maximized accuracy for all relevant models. Every other metric was calculated by the thresholder function. There are only slight differences in how exactly the models ran. Logistic regression, LDA, QDA, and Random Forest predicted metrics based on the predicted data run into the final model again, or in sample data. KNN, Penalized Logistic Regression (Lasso Logistic Regression), and all SVM models took the predicted values of their optimal tuning parameters already calculated by the model for Cross-Validation, certainly in sample data.

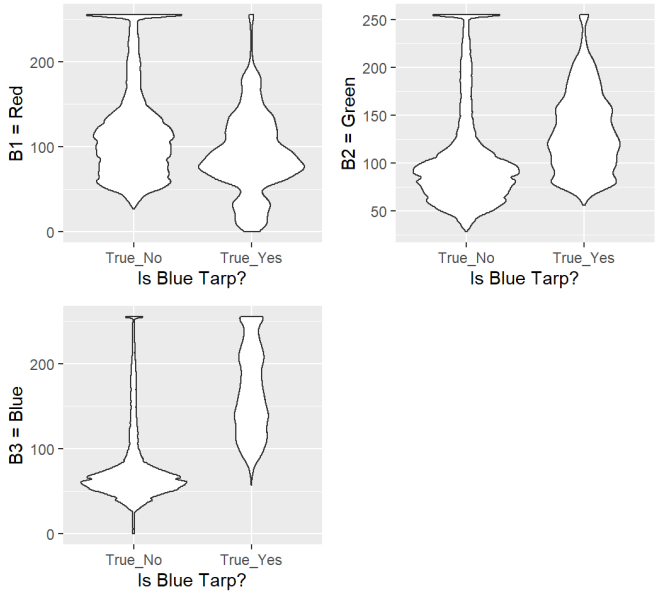
Hold-out Data / EDA

Load data, explore data, etc.

```
setwd("C:/Users/Tommy/OneDrive/Documents/R/University of Virginia/DS 6030")

temp <- data.frame()
holdout <- data.frame()

#Non Blue Tarp Data
test <- read_table("Hold Out Data/orthovnir057_ROI_NON_Blue_Tarps.txt", skip = 8, col_names = FALSE)
temp <- rbind(temp, test)
test <- read_table("Hold Out Data/orthovnir067_ROI_NOT_Blue_Tarps.txt", skip = 8, col_names = FALSE)
temp <- rbind(temp, test)
test <- read_table("Hold Out Data/orthovnir069_ROI_NOT_Blue_Tarps.txt", skip = 8, col_names = FALSE)
temp <- rbind(temp, test)
test <- read_table("Hold Out Data/orthovnir078_ROI_NON_Blue_Tarps.txt", skip = 8, col_names = FALSE)
temp <- rbind(temp, test)

temp <- temp[,8:10]
colnames(temp) <- c("Red", "Green", "Blue")
temp$Blue_Tarp <- "True_No"

#Blue Tarp Data
test <- read_table("Hold Out Data/orthovnir067_ROI_Blue_Tarps.txt", skip = 8, col_names = FALSE)
holdout <- rbind(holdout, test)
test <- read_table("Hold Out Data/orthovnir069_ROI_Blue_Tarps.txt", skip = 8, col_names = FALSE)
holdout <- rbind(holdout, test)
test <- read_table("Hold Out Data/orthovnir078_ROI_Blue_Tarps.txt", skip = 8, col_names = FALSE)
holdout <- rbind(holdout, test)

holdout <- holdout[,8:10]
colnames(holdout) <- c("Red", "Green", "Blue")
holdout$Blue_Tarp <- "True_Yes"

holdout <- rbind(holdout, temp)
holdout$Blue_Tarp <- as.factor(holdout$Blue_Tarp)

summary(holdout)
```

```
#>       Red            Green            Blue           Blue_Tarp
#>  Min.   :  0.0   Min.   : 28.0   Min.   :  0.00   True_No :1989697
#>  1st Qu.: 76.0   1st Qu.: 71.0   1st Qu.: 54.00   True_Yes:  14480
#>  Median :107.0   Median : 91.0   Median : 65.00
#>  Mean   :118.1   Mean   :105.4   Mean   : 79.81
#>  3rd Qu.:139.0   3rd Qu.:117.0   3rd Qu.: 83.00
#>  Max.   :255.0   Max.   :255.0   Max.   :255.00
```

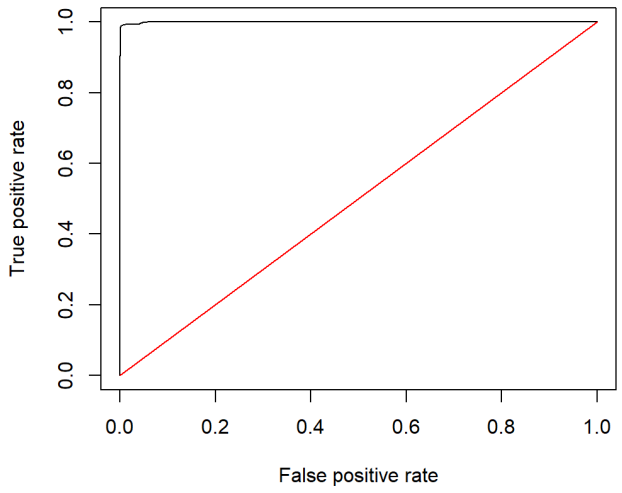
```
Red_graph <- ggplot(holdout, aes(x=Blue_Tarp, y = Red)) +
  geom_violin() +
  labs(x="Is Blue Tarp?", y="B1 = Red")
Green_graph <- ggplot(holdout, aes(x=Blue_Tarp, y = Green)) +
  geom_violin() +
  labs(x="Is Blue Tarp?", y="B2 = Green")
Blue_graph <- ggplot(holdout, aes(x=Blue_Tarp, y = Blue)) +
  geom_violin() +
  labs(x="Is Blue Tarp?", y="B3 = Blue")
grid.arrange(Red_graph, Green_graph, Blue_graph, nrow=2)
```



I assigned what I thought Red, Green and Blue were in the data. Blue is very obvious, Red and Green are alittle more nebulous but based on Green being alittle more bottom heavy on True_No and more centered on True_Yes it looks like B2 = Green. Also looking at the summary on test and training data, Red seems to have higher values on average in the quartiles which supports my observations. To this end the data labels B1, B2, and B3 as colors and says it is rgb which would logically make sense to match. B1 = red, B2=green, B3=Blue.

Results (Hold-Out)

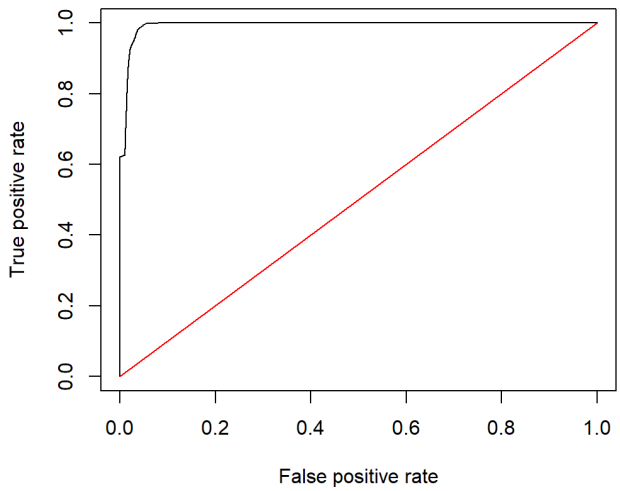
```
#GLM
preds<-predict(modelFit_glm$finalModel, newdata = holdout, type = "response")
rates<-prediction(preds, holdout$Blue_Tarp)
roc_result<-performance(rates,measure="tpr", x.measure="fpr")
plot(roc_result)
lines(x = c(0,1), y = c(0,1), col="red")
```




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

tab <- table(preds>threshold_glm, holdout$Blue_Tarp)
tab
```

```
#>
#>         True_No True_Yes
#>   FALSE 1980882      132
#>   TRUE     8815    14348
```

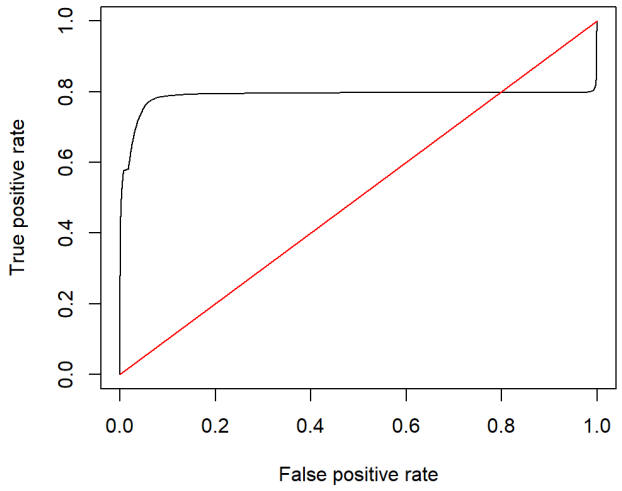
```
#Accuracy
Accuracy_glm <- (tab[1] + tab[4]) / sum(tab[1:4])

#TPR
TPR_glm <- tab[4]/ sum(tab[3:4])

#FPR
FPR_glm <- tab[2]/ sum(tab[1:2])

#Precision
Precision_glm <- tab[4]/ (tab[2] + tab[4])

row_glm <- c(NA, AUROC_glm, threshold_glm, Accuracy_glm, TPR_glm, FPR_glm, Precision_glm)



#LDA
#get AUROC
preds<-predict(modelFit_lda$finalModel, newdata = holdout[-4], type = "response")$posterior[,2]
rates<-prediction(preds, holdout$Blue_Tarp)
roc_result<-performance(rates,measure="tpr", x.measure="fpr")
plot(roc_result)
lines(x = c(0,1), y = c(0,1), col="red")
```




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

tab <- table(preds>threshold_lda, holdout$Blue_Tarp)
tab
```

```
#>
#>         True_No True_Yes
#>   FALSE 1949070     1246
#>   TRUE    40627    13234
```

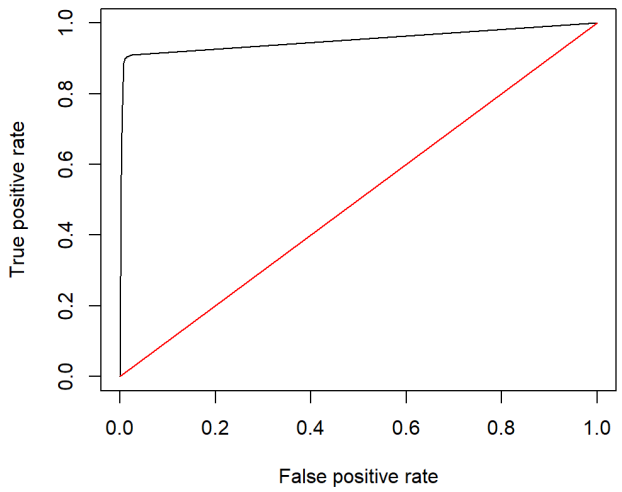
```
#Accuracy
Accuracy_lda <- (tab[1] + tab[4]) / sum(tab[1:4])

#TPR
TPR_lda <- tab[4]/ sum(tab[3:4])

#FPR
FPR_lda <- tab[2]/ sum(tab[1:2])

#Precision
Precision_lda <- tab[4]/ (tab[2] + tab[4])

row_lda <- c(NA, AUROC_lda, threshold_lda, Accuracy_lda, TPR_lda, FPR_lda, Precision_lda)



#QDA
#get AUROC
preds<-predict(modelFit_qda$finalModel, newdata = holdout[-4], type = "response")$posterior[,2]
rates<-prediction(preds, holdout$Blue_Tarp)
roc_result<-performance(rates,measure="tpr", x.measure="fpr")
plot(roc_result)
lines(x = c(0,1), y = c(0,1), col="red")
```




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

tab <- table(preds>threshold_qda, holdout$Blue_Tarp)
tab
```

```
#>
#>         True_No True_Yes
#>   FALSE 1984056     7350
#>   TRUE     5641     7130
```

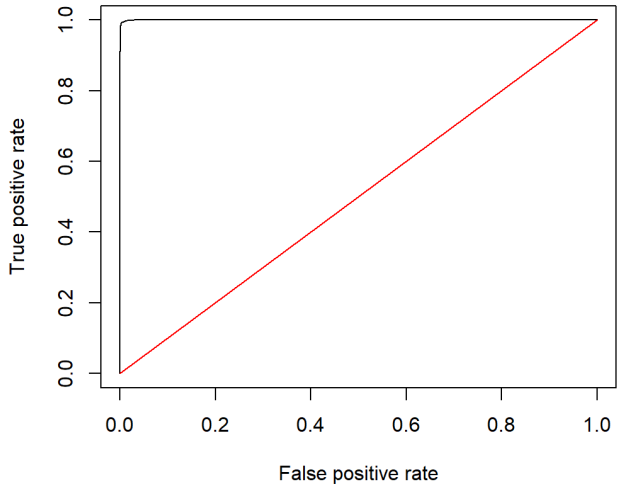
```
#Accuracy
Accuracy_qda <- (tab[1] + tab[4]) / sum(tab[1:4])

#TPR
TPR_qda <- tab[4]/ sum(tab[3:4])

#FPR
FPR_qda <- tab[2]/ sum(tab[1:2])

#Precision
Precision_qda <- tab[4]/ (tab[2] + tab[4])

row_qda <- c(NA, AUROC_qda, threshold_qda, Accuracy_qda, TPR_qda, FPR_qda, Precision_qda)



#KNN
#get AUROC
if (file.exists("preds_knn.rda")){
  load(file = "preds_knn.rda")
} else {
  preds<-predict(modelFit_knn$finalModel, newdata = holdout[-4])[,2]
  save(preds, file = "preds_knn.rda")
}
rates<-prediction(preds, holdout$Blue_Tarp)
roc_result<-performance(rates,measure="tpr", x.measure="fpr")
plot(roc_result)
lines(x = c(0,1), y = c(0,1), col="red")
```




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

tab <- table(preds>0.5, holdout$Blue_Tarp)
tab
```

```
#>
#>         True_No True_Yes
#>   FALSE 1977003     2449
#>   TRUE    12694    12031
```

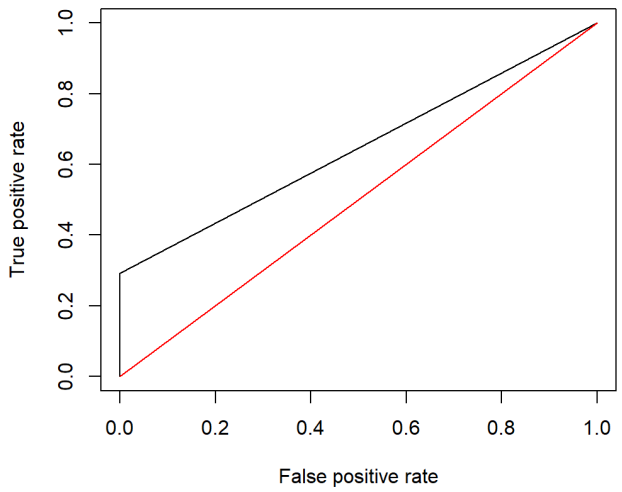
```
#Accuracy
Accuracy_knn <- (tab[1] + tab[4]) / sum(tab[1:4])

#TPR
TPR_knn <- tab[4]/ sum(tab[3:4])

#FPR
FPR_knn <- tab[2]/ sum(tab[1:2])

#Precision
Precision_knn <- tab[4]/ (tab[2] + tab[4])

row_knn <- c(paste0("k = ", modelFit_knn$bestTune$k), AUROC_knn, threshold_knn, Accuracy_knn, TPR_knn,
             FPR_knn, Precision_knn)



#Penalized Logistic Regression
#get AUROC
preds<-rowMeans(predict(modelFit_mix$finalModel, newx = model.matrix(~.-1, data=holdout[-4])))
rates<-prediction(preds, holdout$Blue_Tarp)
roc_result<-performance(rates,measure="tpr", x.measure="fpr")
plot(roc_result)
lines(x = c(0,1), y = c(0,1), col="red")
```




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

tab <- table(preds>threshold_mix, holdout$Blue_Tarp)
tab
```

```
#>
#>         True_No True_Yes
#>   FALSE 1988014      517
#>   TRUE     1683    13963
```

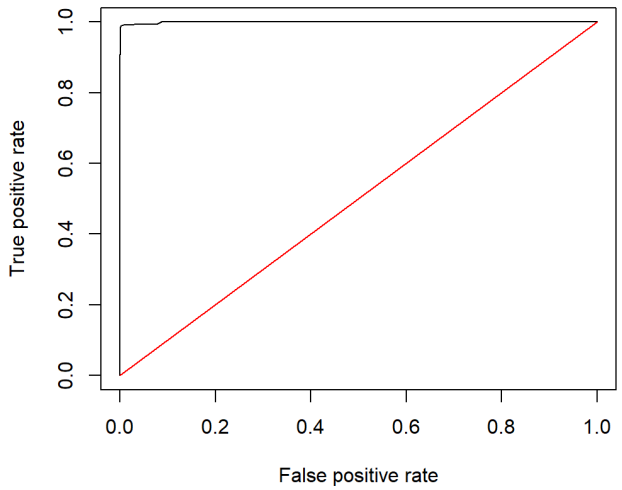
```
#Accuracy
Accuracy_mix <- (tab[1] + tab[4]) / sum(tab[1:4])

#TPR
TPR_mix <- tab[4]/ sum(tab[3:4])

#FPR
FPR_mix <- tab[2]/ sum(tab[1:2])

#Precision
Precision_mix <- tab[4]/ (tab[2] + tab[4])

row_mix <- c(paste0("alpha = ", modelFit_mix$bestTune$alpha, ", lambda = ", modelFit_mix$bestTune$lambda),
             AUROC_mix, threshold_mix, Accuracy_mix, TPR_mix, FPR_mix, Precision_mix)



#Random Forest
#get AUROC
preds<-predict(modelFit_rf$finalModel, newdata = holdout, type="response")
preds <- ifelse(preds == "True_Yes", 1,0)
rates<-prediction(preds, holdout$Blue_Tarp)
roc_result<-performance(rates,measure="tpr", x.measure="fpr")
plot(roc_result)
lines(x = c(0,1), y = c(0,1), col="red")
```




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

tab<- table(preds>threshold_rf, holdout$Blue_Tarp)
tab
```

```
#>
#>         True_No True_Yes
#>   FALSE 1989663    10238
#>   TRUE       34     4242
```

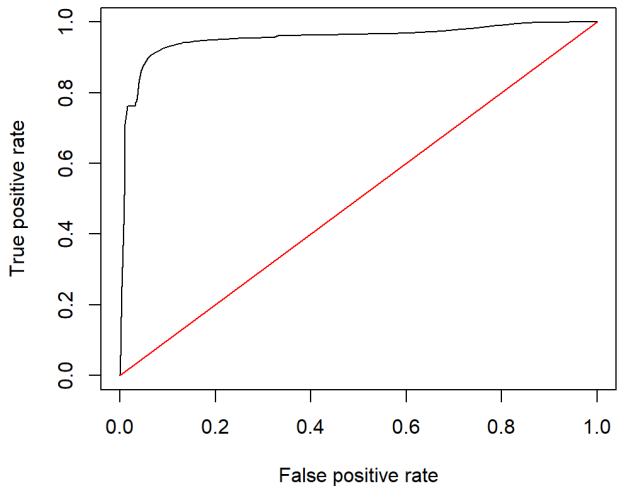
```
#Accuracy
Accuracy_rf <- (tab[1] + tab[4]) / sum(tab[1:4])

#TPR
TPR_rf <- tab[4]/ sum(tab[3:4])

#FPR
FPR_rf <- tab[2]/ sum(tab[1:2])

#Precision
Precision_rf <- tab[4]/ (tab[2] + tab[4])

row_rf <- c(paste0("mtry = ", modelFit_rf$bestTune$mtry, ", sampsize = 100, ntree=200"),
            AUROC_rf, threshold_rf, Accuracy_rf, TPR_rf, FPR_rf, Precision_rf)



#Linear SVM
#get AUROC
if (file.exists("preds_linear_SVM.rda")){
  load(file = "preds_linear_SVM.rda")
} else {
  preds<-predict(modelFit_svm_linear$finalModel, newdata = holdout[-4], type="probabilities")[,2]
  save(preds, file = "preds_linear_SVM.rda")
}

rates<-prediction(preds, holdout$Blue_Tarp)
roc_result<-performance(rates,measure="tpr", x.measure="fpr")
plot(roc_result)
lines(x = c(0,1), y = c(0,1), col="red")
```




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

tab <- table(preds>threshold_svm_linear, holdout$Blue_Tarp)
tab
```

```
#>
#>         True_No True_Yes
#>   FALSE 1974501      129
#>   TRUE    15196    14351
```

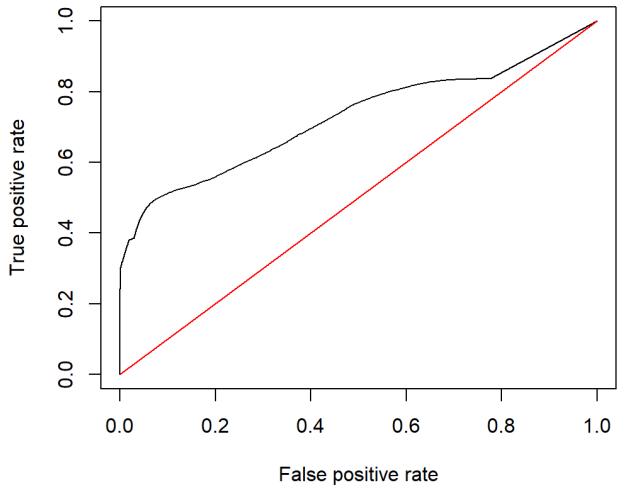
```
#Accuracy
Accuracy_svm_linear <- (tab[1] + tab[4]) / sum(tab[1:4])

#TPR
TPR_svm_linear <- tab[4]/ sum(tab[3:4])

#FPR
FPR_svm_linear <- tab[2]/ sum(tab[1:2])

#Precision
Precision_svm_linear <- tab[4]/ (tab[2] + tab[4])

row_svm_linear <- c(paste0("C = ", modelFit_svm_linear$bestTune$C), AUROC_svm_linear, threshold_svm_linear,
                    Accuracy_svm_linear, TPR_svm_linear, FPR_svm_linear, Precision_svm_linear)



#Radial SVM
#get AUROC
if (file.exists("preds_radial_SVM.rda")){
  load(file = "preds_radial_SVM.rda")
} else {
  preds<-predict(modelFit_svm_radial$finalModel, newdata = holdout[-4], type="probabilities")[,2]
  save(preds, file = "preds_radial_SVM.rda")
}

rates<-prediction(preds, holdout$Blue_Tarp)
roc_result<-performance(rates,measure="tpr", x.measure="fpr")
plot(roc_result)
lines(x = c(0,1), y = c(0,1), col="red")
```




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

tab <- table(preds>threshold_svm_radial, holdout$Blue_Tarp)
tab
```

```
#>
#>         True_No True_Yes
#>   FALSE 1981164    10852
#>   TRUE     8533     3628
```

```
#Accuracy
Accuracy_svm_radial <- (tab[1] + tab[4]) / sum(tab[1:4])

#TPR
TPR_svm_radial <- tab[4]/ sum(tab[3:4])

#FPR
FPR_svm_radial <- tab[2]/ sum(tab[1:2])

#Precision
Precision_svm_radial <- tab[4]/ (tab[2] + tab[4])

row_svm_radial <- c(paste0("C = ", modelFit_svm_radial$bestTune$C, ", sigma = ", modelFit_svm_radial$bestTune$sigma),
                    AUROC_svm_radial, threshold_svm_radial, Accuracy_svm_radial, TPR_svm_radial,
                    FPR_svm_radial, Precision_svm_radial)



#Poly SVM
#get AUROC
if (file.exists("preds_poly_SVM.rda")){
  load(file = "preds_poly_SVM.rda")
} else {
  preds<-predict(modelFit_svm_poly$finalModel, newdata = holdout[-4], type="probabilities")[,2]
  save(preds, file = "preds_poly_SVM.rda")
}

rates<-prediction(preds, holdout$Blue_Tarp)
roc_result<-performance(rates,measure="tpr", x.measure="fpr")
plot(roc_result)
lines(x = c(0,1), y = c(0,1), col="red")
```




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

tab <- table(preds>threshold_svm_poly, holdout$Blue_Tarp)
tab
```

```
#>
#>         True_No True_Yes
#>   FALSE 1981892     9918
#>   TRUE     7805     4562
```

```
#Accuracy
Accuracy_svm_poly <- (tab[1] + tab[4]) / sum(tab[1:4])

#TPR
TPR_svm_poly <- tab[4]/ sum(tab[3:4])

#FPR
FPR_svm_poly <- tab[2]/ sum(tab[1:2])

#Precision
Precision_svm_poly <- tab[4]/ (tab[2] + tab[4])

row_svm_poly <- c(paste0("C = ", modelFit_svm_poly$bestTune$C, ", degree = ", modelFit_svm_poly$bestTune$degree),
                  AUROC_svm_poly, threshold_svm_poly, Accuracy_svm_poly, TPR_svm_poly,
                  FPR_svm_poly, Precision_svm_poly)
```

I took the model I created with the training data and ran the test data into it. Keep the same threshold as in training data. I have put what looks like a threshold on KNN but I just classified the largest probability class as True (Blue_Tarp) or False (not a Blue_Tarp).

```
Performance_Table <- as.data.frame(matrix(data = c(row_glm, row_lda, row_qda, row_knn, row_mix, row_rf, row_svm_linear,
                                                   row_svm_radial, row_svm_poly), nrow = 9, ncol = 7, byrow = TRUE,
                                          dimnames = list(c("Log Reg", "LDA", "QDA", "KNN", "Penalized Log Reg",
                                                            "Random Forest", "SVM Linear", "SVM Radial",
                                                            "SVM Poly"),c("Tuning", "AUROC", "Threshold",
                                                                          "Accuracy", "TPR", "FPR", "Precision"))))
Performance_Table
```

	Tuning	AUROC	Threshold	Accuracy	TPR	FPR	Precision
Log Reg	NA	0.999619137971996	0.75	0.995535823432761	0.990883977900552	0.00443032280794513	0.619436169753
LDA	NA	0.993069428427794	0.15	0.979107134749077	0.913950276243094	0.020418686865387	0.245706540910
QDA	NA	0.787710303215911	0.6	0.993518037578517	0.492403314917127	0.00283510504363227	0.558296139691
KNN	k = 7	0.951343358137484	NA	0.992444280120967	0.830870165745856	0.00637986587907606	0.486592517694
Penalized Log Reg	alpha = 1, lambda = 1.99526231496888e-05	0.999822214438095	0.75	0.998902292561984	0.964295580110497	0.000845857434574209	0.892432570625
Random Forest	mtry = 3, sampsize = 100, ntree=200	0.646469356537996	0.85	0.994874704180319	0.292955801104972	1.70880289812972e-05	0.992048643592
SVM Linear	C = 1	0.999346677815464	0.75	0.992353469778368	0.991091160220995	0.00763734377646446	0.485700747960
SVM Radial	C = 10, sigma = 16	0.953618680410063	0.7	0.990327700597303	0.250552486187845	0.0042885926852179	0.298330729380
SVM Poly	C = 100, degree = 3	0.726755379127003	0.65	0.991156968670931	0.315055248618785	0.00392270782938307	0.368884935716

Tuning values and Threshold were kept the same from the Cross-Validated (CV) data. All other metrics were calculated by running the training model on the holdout data.

Final Conclusions

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

For the CV (Cross-Validation) algorithms, the Accuracy, TPR (True Positive Rate) and Precision are all really high for all the algorithms with >95% for all the algorithms and >99% for most so the most important metric ends up being FPR (False Positive Rate) by process of elimination. The standout model is KNN (k=7) as it has the lowest FPR of 0.16% but it also has the lowest TPR (96%) and Precision (95%). The next lowest model is SVM Radial and SVM Poly 3. They are both very similar with a FPR about 4% and TPR and Precision > 99.8%. The next models to be considered are SVM linear, Logistic Regression and Penalized Logistic Regression. These all have FPR about 8% and high TPR and Precision. The rest of the algorithms have too high FPR to be considered.

For the Hold-out data the best algorithm is one that has a high TPR, a low FPR and a High Precision. There is only really one algorithm that fits these requirements Penalized Logistic Regression. The next algorithms to consider are the ones which have a High TPR which are Logistic Regression and SVM Linear which both have a TPR of >99%. Logistic Regression is better since the Precision is higher than SVM Linear.

This leads the best algorithm as Penalized Logistic Regression followed by Logistic Regression and then SVM Linear. This is because the holdout data is more indicative of a good model, one that is not overtrained, something that CV data doesn't do as well.

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

Having CV models that have such high metrics on the training data means one of two things. Either all the models are actually amazing and will have those metrics on the test data or most/all of the models are overtrained. With having metrics as high as they were on all models I suspected that my models were overtrained but didn't know how to fix them without manually picking parameters for all my models and even then I wasn't sure I could get it right. Understanding this, I didn't have high expectations on my metrics on the test data, and for the most part I was right. I was shocked to find that Penalized Logistic Regression (PLR) had slightly lower but fairly similar metrics to the training data. My guess is that it was compatible because the tuning parameters and logistic regression create the perfect balance of the bias-variance trade off.

I also noticed that all my test models had high accuracy, a metric that I optimized for. It makes me wonder if I optimized for a different metric or weighted multiple metrics would I get similar results? I suspect because I optimized for accuracy on both training the models and threshold prediction this is why the accuracy on all the models is so high.

The fact I trained for high accuracy justifies the similarities in high accuracy in both the CV and holdout models. PLR seems to have been so similar in metric between CV and holdout methods because of the nature of PLR compared to the other models, the fact that it naturally creates a good bias-variance trade off and the model characterizes numeric data well.

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

So my recommendation changes depending on the situation of the rescue workers at the time assuming that the algorithms change. If there is more work than a rescue worker can reasonably do and we want to waste as little of a rescue worker's time as possible, Random Forest is the best model for this as it has a >99% accuracy on locations where blue tarps are actually hold blue tarps.

Now when there are more rescue workers on site and Random Forest isn't finding enough people then Penalized Logistic Regression (PLR) is the best algorithm. PLR has a ~89% accuracy on locations where a blue tarp is actually holds a Blue tarp but it finds 96% of all Blue Tarps out there.

Now if there are even more rescue workers on site and workers are really struggling to find people to help I would reluctantly recommend the Logistic Regression Model. It will find 99% of all Blue Tarps but of all the Blue Tarps predicted only 62% of them will actually be Blue Tarps.

My overall recommendation is for the early days when rescue workers are really struggling to get in Random Forest is best, then when there are enough rescue workers to mount a meaningful effort, I would switch the model to PLR. If the model will never be switch, PLR is the recommended model.

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

Accuracy is important because it is the metric that determines how well a model classifies a blue tarp overall, it is a ratio of all the values that are right over all values.

TPR represents how well the model predicts that a Blue Tarp that actually is a Blue Tarp over all true Blue Tarps. All true Blue Tarps means the blue tarps that the model correctly predicts + the points that the model doesn't predict as a Blue Tarp but actually is. TPR is the rate for the model to detect all true Blue Tarps.

FPR represents the rate at which the model predicts that a location is a Blue Tarp but gets it wrong. FPR is the probability of a false alarm.

Precision is the rate that a predicted Blue Tarp is actually a Blue tarp and not wrongly predicted.

Conclusion #5 What additional recommend actions can be taken to improve results?

Well with all this data there is a lack of data on the current situation of when this data was taken in Haiti. Context is needed to chose the best model and provide the most help, a difference in a few percentage points in these models can be the reason why 1 more person is saved or 10,000.

Having a constant flow of information can allow the model to change to best suit the needs of the rescue workers at the time. If the rescue workers feel the algorithm is working against them then this would be the worst case situation. There would need to be proper staff who could interpret the results of this model and give the rescue workers the best support possible so they can properly focus on their work. Even if many actions are taken to improve results if the rescue workers don't have proper support and training on how to use these results then no additional action would truly improve results.

There is only so much this project can cover and only so much knowledge I have but in theory taking this model and building further on it with an additional model would save more time and resources which would in turn save more lives. I believe that the optimal solution would be some pathfinding function for weighting the chance of finding a blue tarp in one location with a combination of the distance to other blue tarps locations with the other blue tarp probabilities. Basically a function that rewards/penalizes certain locations based on proximity to other blue tarps so as many people can be saved as possible/allow more optimal distribution of resources to places in need.

Conclusion #6 How effective do you think your work here could actually be in terms of helping to save human life?

I believe that these models can save a lot of lives as these models will save time and resources of the rescue workers at location. They don't have to use their limited on-site staff to find where their aid is needed, they can focus on aid, transportation, and supply management. Off-site staff can interpret the models and provide support to the rescue workers. That being said I am sure rescue workers have their own method and while pictures show where someone was, it doesn't show where they are going if they move around so these models need a constant supply of recent pictures (which may be challenging depending on the situation as infrastructure was damaged in the earthquake) and rescue workers will need to communicate with the locals on where people are congregating to supply aid to those in need. In the end these models help save lives but they are not and should not be the only method use to locate refuges.

It should be mentioned again that these models are only as good as the support staff behind them. If rescue workers don't trust the support staff giving them the information on these models or the models themselves then they can provide no help. Training is needed on both on and off-site workers to give proper support to rescue workers.