# Final Report - Disaster Relief

Alex Bass

4/28/2022

## Foreword About General Approach

My general approach goes as follows:

1. Import training data and conduct EDA.
2. Train all models using 10 fold cross-validation and hyperparameter tuning with `caret` in parallel:

i. Here is a list of models I will train : Logistic Regression, Ridge Regression, Linear Dicriminant Analysis, Quadriatic Discriminant Analysis, Random Forest, and Support Vector Machines
ii. Using optimal model for each method (obtained from 10fold cv) train that model on entire training data set and record results in a table.
iii. Obtain ROC curve from this model.

3. Import and clean test data
4. Using optimal model, predict using test data and record results.
5. Conclusion: Select best model and justification.

## 1. Import training data and conduct EDA

```r
data <- read.csv('Disaster Relief Part 1/HaitiPixels.csv')

summary(data)
```

```
##     Class                Red           Green            Blue      
##  Length:63241       Min.   : 48   Min.   : 48.0   Min.   : 44.0  
##  Class :character   1st Qu.: 80   1st Qu.: 78.0   1st Qu.: 63.0  
##  Mode  :character   Median :163   Median :148.0   Median :123.0  
##                     Mean   :163   Mean   :153.7   Mean   :125.1  
##                     3rd Qu.:255   3rd Qu.:226.0   3rd Qu.:181.0  
##                     Max.   :255   Max.   :255.0   Max.   :255.0  
```

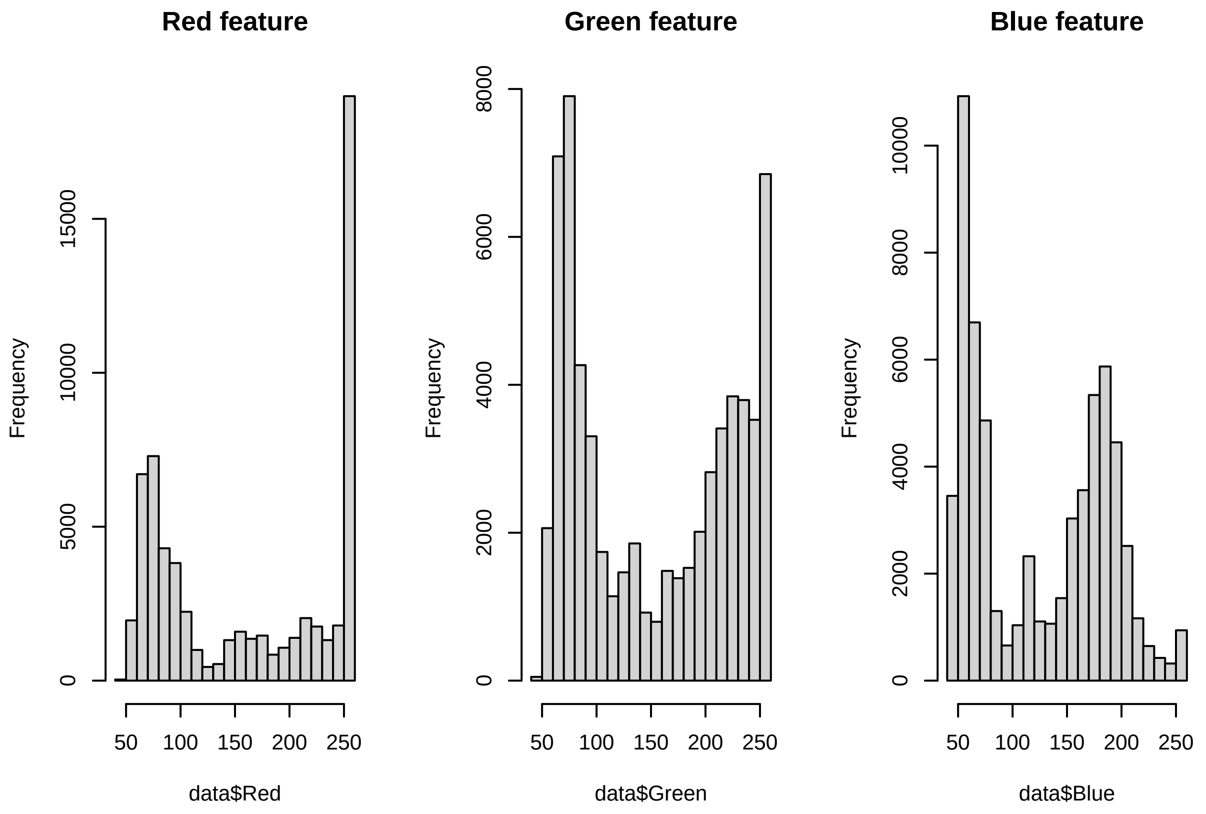
```r
table(data$Class)
```

```
## 
##        Blue Tarp          Rooftop             Soil Various Non-Tarp 
##             2022             9903            20566             4744 
##       Vegetation 
##            26006 
```

A few notes here:

- When looking at all of the pixels, More pixels are more intense in Red and Green vs. Blue. This can be seen because The mean and median for Blue pixels is lower for blue than the other colors.
- Also, it is clear to see that the data set is unbalanced. There are only 2000 blue tarps while there are almost 60000 other observations. This may cause problems in our modeling efforts.

```r
par(mfrow = c(1, 3))

hist(data$Red, main = 'Red feature')
hist(data$Green, main = 'Green feature')
hist(data$Blue, main = 'Blue feature')
```



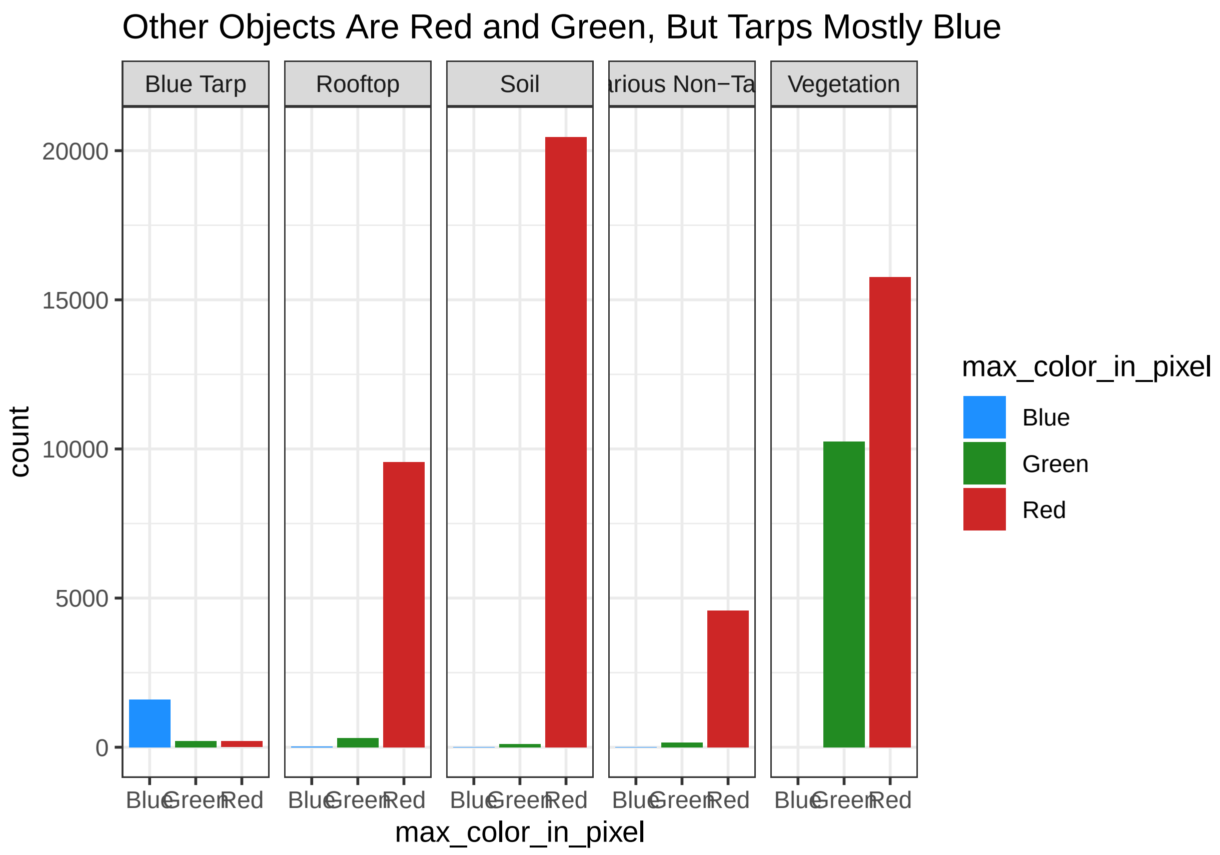
```r
cor(data[,2:4])
```

```
##             Red     Green      Blue
## Red   1.0000000 0.9815987 0.9355294
## Green 0.9815987 1.0000000 0.9648024
## Blue  0.9355294 0.9648024 1.0000000
```

There are a few important things here…

1. From this chart, we can see that there are *a lot* of red 255 and red in general! We will keep these distributions in mind as we move forward. We may want to try and normalize some of these features. The appear to be in bimodal distributions around 50 and around 250.
2. Also, it seems that all of the variables are *strongly* correlated with each other which may have implications for some model assumptions.

```r
#finding max color in each pixel
max_color <- sapply(1:nrow(data), function(x){
  values <- as.numeric(data[x,c('Red', "Green", "Blue")])
  names(values) <- c('Red', "Green", "Blue")
  names(which.max(values))
})

#creating dataframe for visualization
data_for_vis <- data.frame(
  max_color_in_pixel = max_color,
  class = data$Class
)

#plotting max color in pixel by object
data_for_vis %>%
  ggplot() +
  geom_bar(aes(x = max_color_in_pixel, fill = max_color_in_pixel)) +
  facet_grid(~ class) +
  theme_bw() +
  labs(title = "Other Objects Are Red and Green, But Tarps Mostly Blue") +
  scale_fill_manual(values = c("dodgerblue", "forestgreen", "firebrick3"))
```



This is an important chart because it shows us that while the other classifiable objects are mostly Red and Green, Blue is predominantly seen in the blue tarp object. Because of this, I expect Blue to be a very strong predictor in our classification efforts of the blue tarp. I can see some blue in other objects like Rooftop for example, but it is mostly seen in the blue tarp.

```
data$Class <- factor(ifelse(data$Class == 'Blue Tarp', "Blue_Tarp", "Other"),
                     levels = c("Other", "Blue_Tarp"))
table(data$Class)/length(data$Class)
```

### Creating a Binary Response Variable

```
##
##      Other  Blue_Tarp
## 0.96802707 0.03197293
```

Because we are only interested in predicting Blue tarps, I condensed the response variable for ease in analysis to either Blue Tarp or Other.

As seen here, this data set is imbalanced and contains mostly Other objects besides blue tarps which may affect the quality of our results.

## Model Training - Logistic Regression

```
exploreTransformations <- function (df, x, y) {
  fm.linear <- glue('{y} ~ {x}', x=x, y=y)
  fm.quadratic <- glue('{y} ~ poly({x}, 2)', x=x, y=y)
  fm.sqrt <- glue('{y} ~ sqrt({x})', x=x, y=y)
  fm.log <- glue('{y} ~ log({x})', x=x, y=y)

  model.linear <- glm(fm.linear, df, family = "binomial")
  model.quadratic <- glm(fm.quadratic, df, family = "binomial")
  model.sqrt <- glm(fm.sqrt, df, family = "binomial")
  model.log <- glm(fm.log, df, family = "binomial")

  results <- c(
    predictor = x,
    linear=summary(model.linear)$aic,
    quadratic=summary(model.quadratic)$aic,
    sqrt=summary(model.sqrt)$aic,
    log=summary(model.log)$aic)

  return (results)
}

purrr::map_dfr(
  names(data)[2:4],
  ~{exploreTransformations(df = data, x = .x, y = 'Class')}
)
```

```
## # A tibble: 3 x 5
##   predictor linear           quadratic        sqrt             log
##   <chr>     <chr>            <chr>            <chr>            <chr>
## 1 Red       17890.5933901228 12817.9669672475 17831.0329317045 17725.9574303576
## 2 Green     17473.5045415182 15739.3729509791 17345.0959219273 17202.4651980848
## 3 Blue      13623.1070623655 13220.0260987259 13853.2876440047 14111.8752102746
```

For this logistic regression model, I tested a number of transformations that may be influential in this regression. I found that the Red Quadratic term is a transformation that I want to test out.

```
summary(glm(Class ~ Red + Green + Blue + I(Red^2),
            data = data, family = "binomial"))
```

```
##
## Call:
## glm(formula = Class ~ Red + Green + Blue + I(Red^2), family = "binomial",
##     data = data)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -3.4358  -0.0243  -0.0066   0.0000   2.9014
##
```

```
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept) -1.034e+01  6.676e-01 -15.486  < 2e-16 ***
## Red         -6.687e-02  1.340e-02  -4.989 6.07e-07 ***
## Green       -1.689e-01  1.156e-02 -14.610  < 2e-16 ***
## Blue         3.808e-01  1.263e-02  30.166  < 2e-16 ***
## I(Red^2)    -4.482e-04  2.508e-05 -17.874  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 17901.6  on 63240  degrees of freedom
## Residual deviance:  1338.8  on 63236  degrees of freedom
## AIC: 1348.8
##
## Number of Fisher Scoring iterations: 12
```

```
summary(glm(Class~ Red + Blue + Green,
            data = data, family = "binomial"))$aic
```

```
## [1] 1777.526
```

```
summary(glm(Class ~ Red + Green + Blue + I(Red^2),
            data = data, family = "binomial"))$aic
```

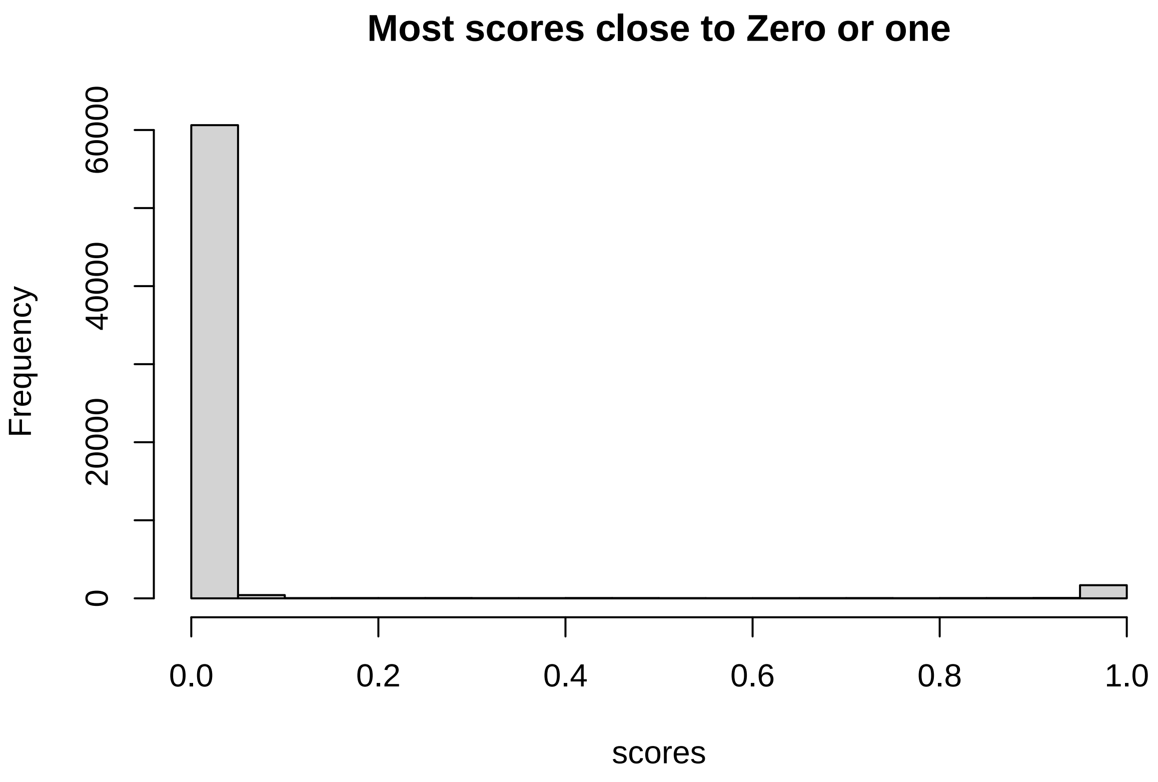
```
## [1] 1348.813
```

After adding this to our logistic regression model, We can see that this significantly reduced the AIC which is a good indicator that we should keep it in the model as it improves the fit. It is also statistically significant signifying it is useful in classification!

```

#setting a seed for selecting a threshold
set.seed(2)

fitControl <- trainControl(method = "cv",
                           number = 10,
                           classProbs = T,
                           allowParallel = T,
                           savePredictions = "final")

LRdata <- data

LRdata$red_sq <- data$Red^2

unregister_dopar()
Mycluster <-  makeCluster(detectCores()-2)
registerDoParallel(Mycluster)

mod_1 <- train(Class~.,
      data = LRdata,
      method = 'glm',
      trControl = fitControl,
      family = 'binomial',
      metric = "ROC"
      )

stopCluster(Mycluster)
```

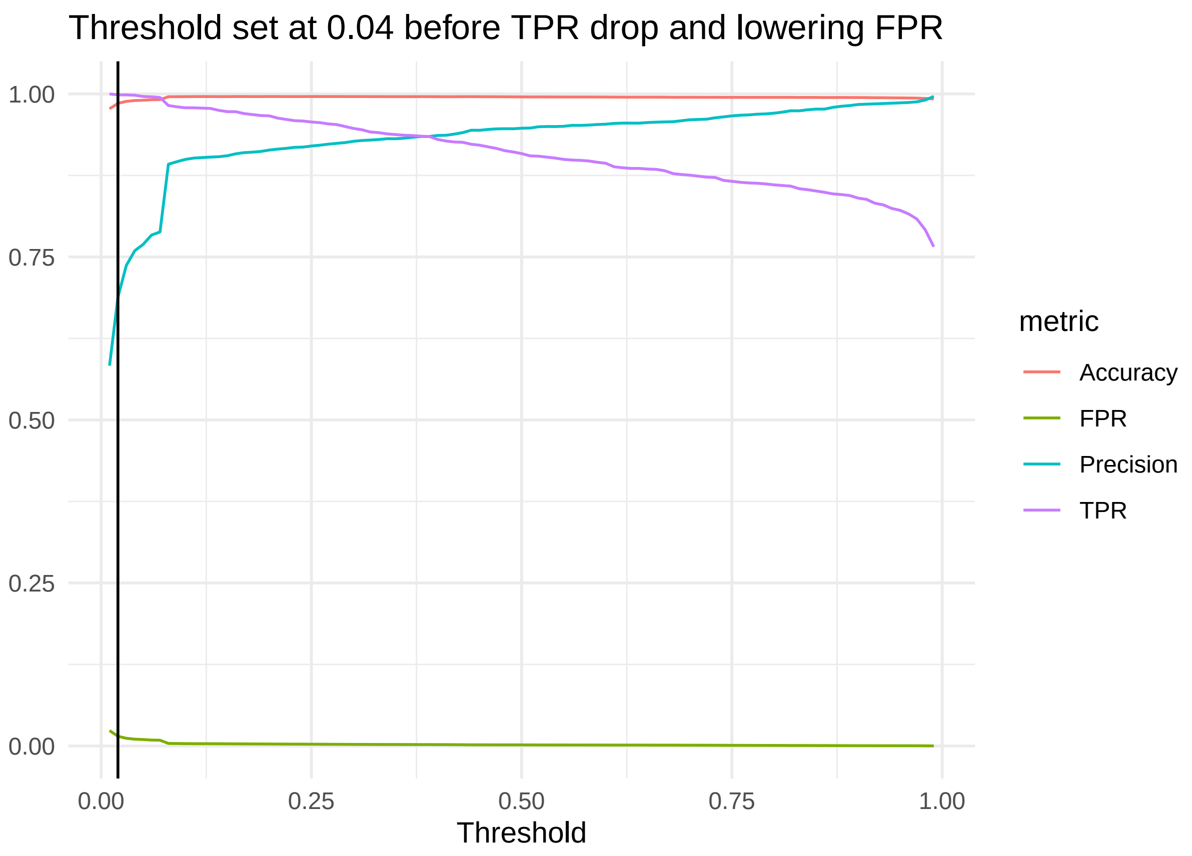
**Logistic Regression : Threshold Selection + Justification** This model didn't take to long to run. Other models may take more computational resources, but I'll hold off caching this particular model.

```
fitControl <- caret::trainControl(method='none',
                                  classProbs = T,
                                  savePredictions = TRUE)

x <- data.matrix(data[,2:4])
y <- data$Class

LRdata <- data

LRdata$red_sq <- data$Red^2

LR_1 <- train(Class~.,
      data = LRdata,
      method = 'glm',
      trControl = fitControl,
      family = 'binomial',
      metric = "ROC"
      )

predicted <- predict(LR_1, newdata=LRdata, type="prob")[,2]

hist(predicted, main = "Most scores close to Zero or one", xlab = "scores")
```



One quick note here... As you can see most of the scores are around 0 or 1. I am going to plot the different metrics over different threshold, but we likely won't see much change because of this fact. This is a pattern we will see throughout this project, the scores are very often close to zero or one making threshold selection not of huge importance.

```r
range_of_thresholds <- seq(0.01,0.99, 0.01)

threshold_selection <- purrr::map_dfr(range_of_thresholds, ~{
  data.frame(
    Threshold = .x,
    TPR = ModelMetrics::sensitivity(as.numeric(y)-1, predicted = predicted, cutoff = .x),
    FPR = 1 - ModelMetrics::specificity(as.numeric(y)-1, predicted = predicted, cutoff = .x),
    Precision = ModelMetrics::precision(as.numeric(y)-1, predicted = predicted, cutoff = .x),
    Accuracy = sum(ifelse(predicted>.x, "Blue_Tarp", "Other") == y)/length(y)
  )
})

threshold_selection %>%
  pivot_longer(cols = 2:5, names_to = 'metric') %>%
  ggplot(aes(x = Threshold, y = value)) +
  geom_line(aes(group = metric, color = metric)) +
  geom_vline(xintercept = 0.02) +
  labs(title = "Threshold set at 0.04 before TPR drop and lowering FPR", y = "") +
  theme_minimal()
```



```r
results <- threshold_selection[threshold_selection$Threshold == 0.02,]

rates <- prediction(predicted, data$Class)
```

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

(cv_results_LR <- data.frame(
  model = "Logistic Regression",
  tuning = "N/A",
  auroc = 1-round(auc,4),
  threshold = 0.02,
  accuracy = results$Accuracy,
  TPR = results$TPR,
  FPR = results$FPR,
  Precision = results$Precision
))
```

```
##                 model tuning  auroc threshold  accuracy       TPR        FPR
## 1 Logistic Regression    N/A 0.9996      0.02 0.9854999 0.9985163 0.01493001
##   Precision
## 1 0.6883737
```

```
roc_df <- data.frame(
  truth = as.numeric(LRdata$Class)-1,
  predicted = predicted
)

#Storing ROC curve
P1 <- roc_df %>%
  ggplot(aes(d = truth, m = predicted)) +
  geom_roc(n.cuts = 0) +
  theme_bw() +
  labs(title = "Best Logistic Regression Model",
       caption = glue("AUC is {round(auc, 4)}"))
```

Here are a several metrics by threshold for our logistic regression model. In this case, it is more important to identify people who may need help to deliver food and supplies than to check a few more areas where people are not. So, raising the true positive rate to as close to 100% is important and while we don't want the FPR to be too high the cost is lower if we take supplies to a few places that are actually False Positives. For these reasons, I am selecting the threshold of 0.98 because we will likely identify most all people who need help (with a very high TPR of 99%) and only have about 1% (FPR of 1.49%) of images that are false positives.

## Ridge - testing and cross validation

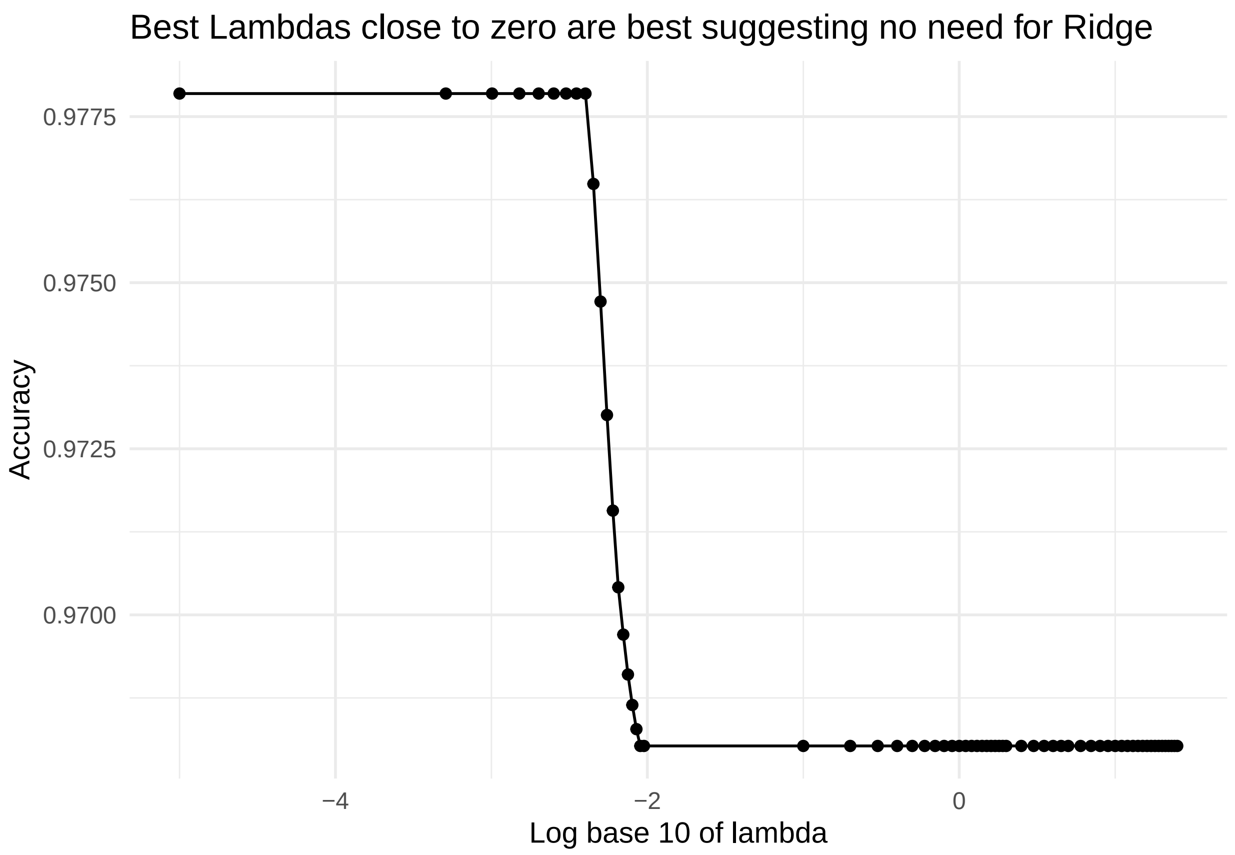
```
tic()
fitControl <- trainControl(method = "cv",
                           number = 10,
                           classProbs = T,
                           savePredictions = "final",
                           allowParallel = T)

parameters <- c(seq(0.00001, 0.01, by =0.0005) ,seq(0.1, 2, by =0.1) ,  seq(2, 5, 0.5) , seq(5, 25, 1))

y <- data$Class
x <- data.matrix(data[2:4])
```

```r
unregister_dopar()
Mycluster <-  makeCluster(detectCores()-2)
registerDoParallel(Mycluster)

ridge <- train(
  y = y,
  x = x,
  method = 'glmnet',
  tuneGrid = expand.grid(alpha = 0, lambda = parameters),
  trControl = fitControl,
  metric =  "ROC")

stopCluster(Mycluster)

toc()
```

```
## 23.719 sec elapsed
```

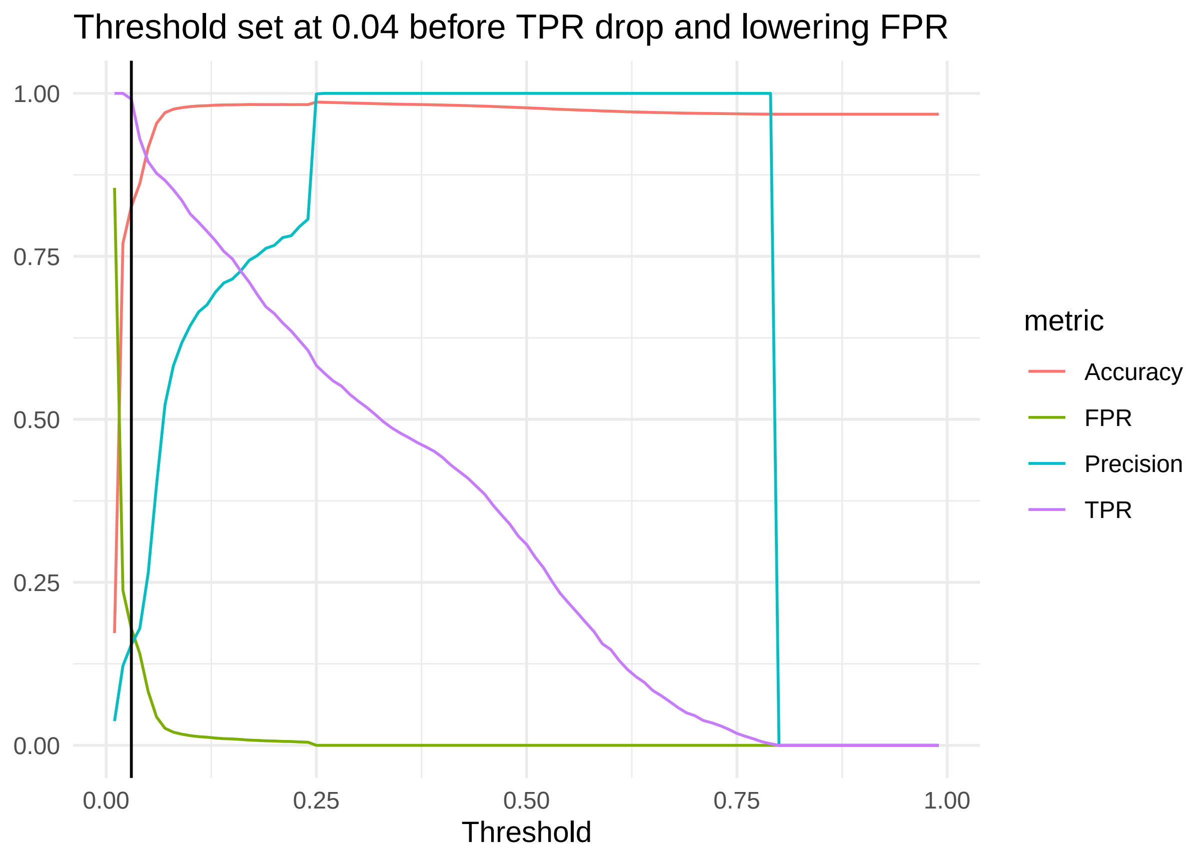
This model took longer to train than the Logistic Regression model, but still not a huge constraint on resources.

```r
ridge$results %>%
  mutate(lambda = log10(lambda)) %>%
  ggplot(aes(x = lambda, y = Accuracy, group = 1)) +
  geom_point() +
  geom_line() +
  labs(title = "Best Lambdas close to zero are best suggesting no need for Ridge",
       x = "Log base 10 of lambda") +
  theme_minimal()
```



As we can see here, the highest performing lambdas are close to zero which suggests that we do not need this model.

```
fitControl <- caret::trainControl(method='none',
                                  classProbs = T,
                                  savePredictions = TRUE)

x <- data.matrix(x)

ridge_1 <- train(
  y = y,
  x = x,
  method = 'glmnet',
  tuneGrid = data.frame(alpha = 0, lambda = 0.00401),
  trControl = fitControl,
  metric =  "ROC")

predicted <- predict(ridge_1$finalModel, newx = x, s = 0.00401, type = "response")

range_of_thresholds <- seq(0.01,0.99, 0.01)

threshold_selection <- purrr::map_dfr(range_of_thresholds, ~{
  data.frame(
    Threshold = .x,
    TPR = ModelMetrics::sensitivity(as.numeric(y)-1, predicted = predicted, cutoff = .x),
    FPR = 1 - ModelMetrics::specificity(as.numeric(y)-1, predicted = predicted, cutoff = .x),
    Precision = ModelMetrics::precision(as.numeric(y)-1, predicted = predicted, cutoff = .x),
    Accuracy = sum(ifelse(predicted>.x, "Blue_Tarp", "Other") == y)/length(y)
  )
})

threshold_selection %>%
  pivot_longer(cols = 2:5, names_to = 'metric') %>%
  ggplot(aes(x = Threshold, y = value)) +
  geom_line(aes(group = metric, color = metric)) +
  geom_vline(xintercept = 0.03) +
  labs(title = "Threshold set at 0.04 before TPR drop and lowering FPR", y = "") +
  theme_minimal()
```



```
results <- threshold_selection[threshold_selection$Threshold == 0.04,]

rates <- prediction(predicted, data$Class)
auc <- performance(rates, measure = "auc")
auc <- auc@y.values[[1]]

(cv_results_Ridge <- data.frame(
  model = "Ridge Regression",
  tuning = "Lambda = 0.00401",
  auroc = 1-round(auc,4),
  threshold = 0.04,
  accuracy = results$Accuracy,
  TPR = results$TPR,
  FPR = results$FPR,
  Precision = results$Precision
))
```

```
##                model           tuning  auroc threshold accuracy       TPR
## 1 Ridge Regression Lambda = 0.00401 0.9802      0.04 0.861419 0.9302671
##         FPR Precision
## 1 0.140855 0.1790746
```

```
roc_df <- data.frame(
  truth = as.numeric(data$Class)-1,
  predicted = predicted
)

#Storing ROC curve
P2 <- roc_df %>%
  ggplot(aes(d = truth, m = predicted)) +
  geom_roc(n.cuts = 0) +
  theme_bw() +
  labs(title = "Ridge Regression ROC",
       caption = glue("AUC is {round(1-auc, 4)}"))
```

The threshold is set at 0.04 in this case because the TPR is still quite high at 93% while the FPR is not too high. This would mean that 14% of the images would not have blue tarps, but as discussed previously, It is more important to find people who are displaced vs. having to look through extra images. This is still a good model with accuracy at 0.86, but simply not as good as our Logistic Regression we first ran.

## K - Nearest Neighbors

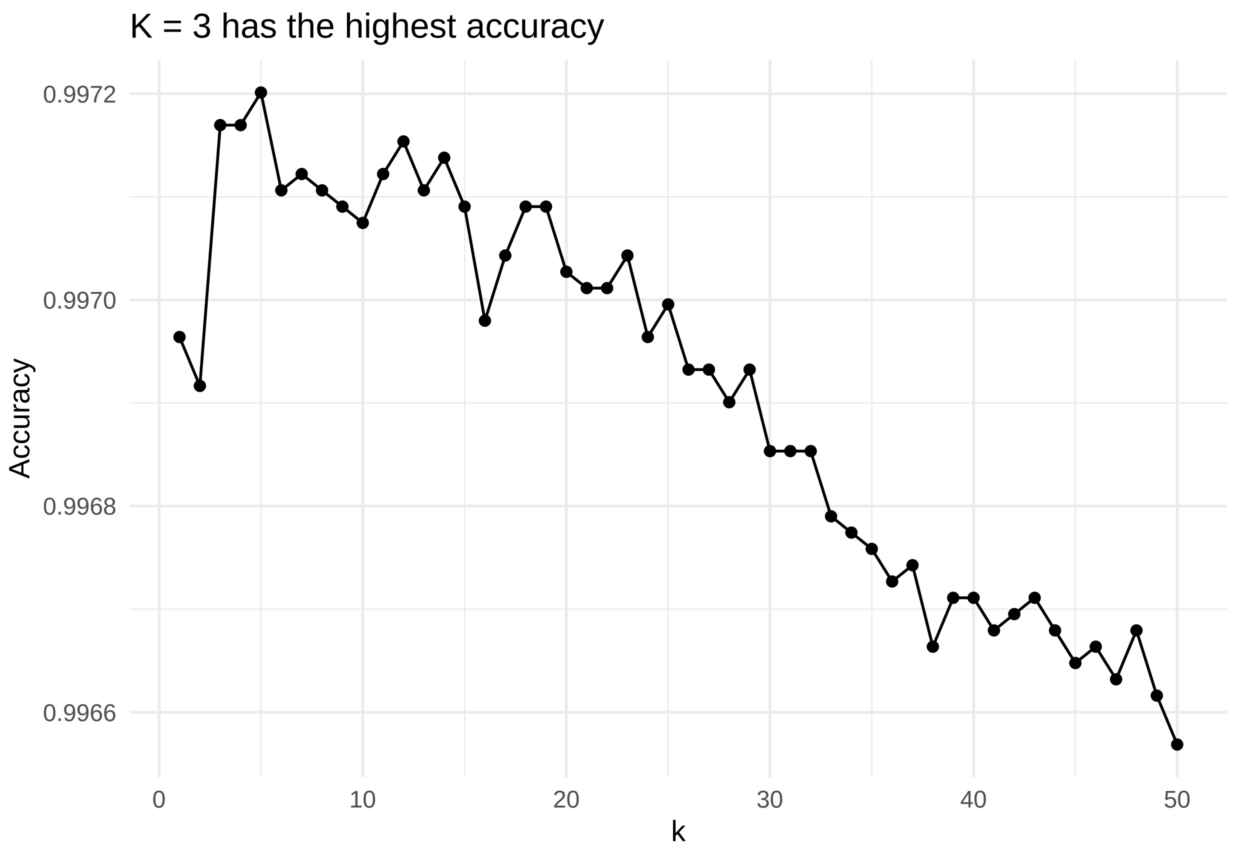
```
tic()
fitControl <- trainControl(method = "cv",
                           number = 10,
                           classProbs = T,
                           savePredictions = TRUE,
                           allowParallel = T)

parameters <- c(seq(1, 50, by =1))

y <- data$Class
x <- data.matrix(data[2:4])

unregister_dopar()
Mycluster <-  makeCluster(detectCores()-2)
registerDoParallel(Mycluster)

knn <- train(
      y=y,
      x = x,
      method = "knn",
      trControl = fitControl,
      tuneGrid = expand.grid(k = parameters),
      metric = "ROC"
    )

stopCluster(Mycluster)
```

```
toc()
```

```
## 404.839 sec elapsed
```

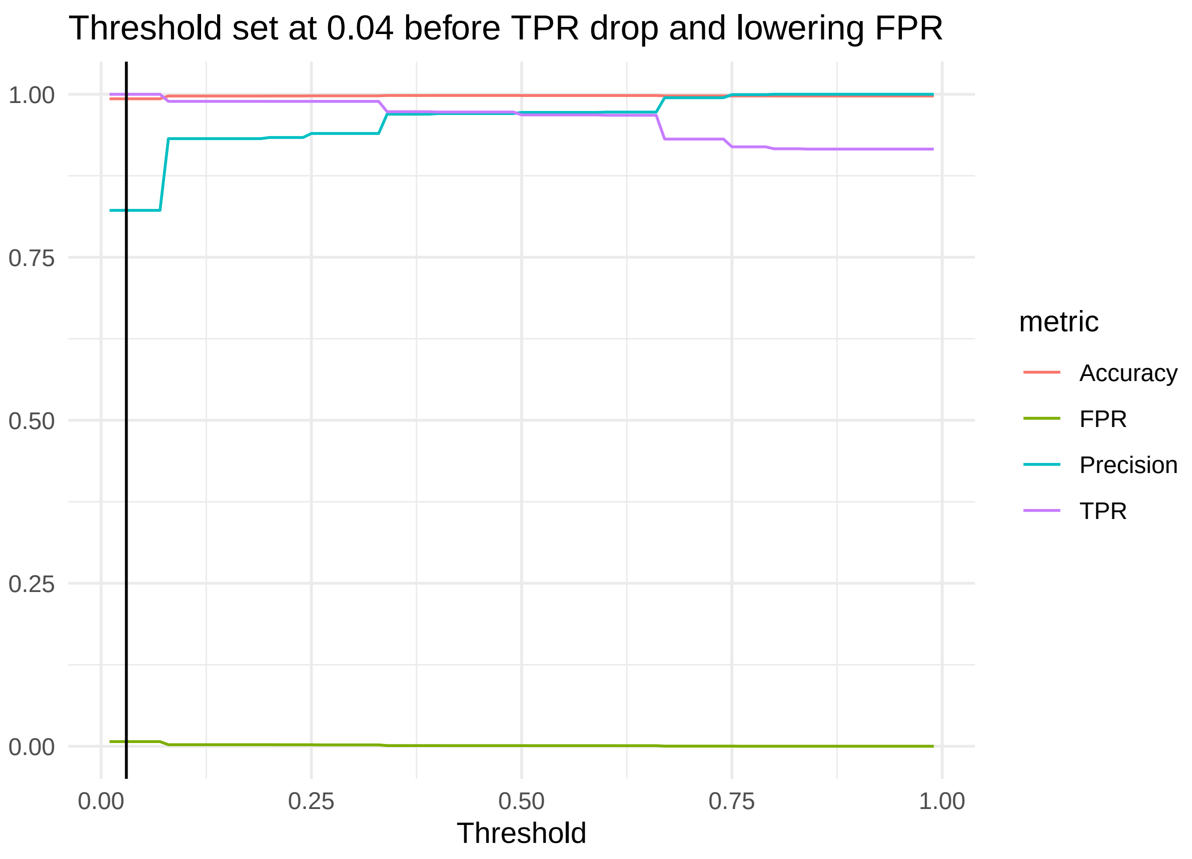
This run took up significant time, but still manageable. This is using 6 of my 8 cores.

```
knn$results %>%
  ggplot(aes(x = k, y = Accuracy, group = 1)) +
  geom_point() +
  geom_line() +
  labs(title = "K = 3 has the highest accuracy",
       x = "k") +
  theme_minimal()
```



Looking at the chart, we can see that when K is 3 the CV model has the highest accuracy. I would have expanded k, but it appears that the accuracy slowly decreases as k increases afterwards. So, I will just assume that it continues in this direction.

```
fitControl <- caret::trainControl(method='none',
                                  classProbs = T,
                                  savePredictions = TRUE)

x <- data.matrix(x)

knn_1 <- train(
  y = y,
  x = x,
  method = 'knn',
  tuneGrid = data.frame(k = 3),
  trControl = fitControl,
  metric =  "ROC")

predicted <- predict(knn_1$finalModel, x, type = "prob")[,2]

range_of_thresholds <- seq(0.01,0.99, 0.01)

threshold_selection <- purrr::map_dfr(range_of_thresholds, ~{
  data.frame(
    Threshold = .x,
    TPR = ModelMetrics::sensitivity(as.numeric(y)-1, predicted = predicted, cutoff = .x),
    FPR = 1 - ModelMetrics::specificity(as.numeric(y)-1, predicted = predicted, cutoff = .x),
    Precision = ModelMetrics::precision(as.numeric(y)-1, predicted = predicted, cutoff = .x),
    Accuracy = sum(ifelse(predicted>.x, "Blue_Tarp", "Other") == y)/length(y)
  )
})

threshold_selection %>%
  pivot_longer(cols = 2:5, names_to = 'metric') %>%
  ggplot(aes(x = Threshold, y = value)) +
  geom_line(aes(group = metric, color = metric)) +
  geom_vline(xintercept = 0.03) +
  labs(title = "Threshold set at 0.04 before TPR drop and lowering FPR", y = "") +
  theme_minimal()
```



```
results <- threshold_selection[threshold_selection$Threshold == 0.04,]
```

```r
rates <- prediction(predicted, data$Class)
auc <- performance(rates, measure = "auc")
auc <- auc@y.values[[1]]

(cv_results_knn <- data.frame(
  model = "KNN",
  tuning = "k=3",
  auroc = 1-round(auc,4),
  threshold = 0.04,
  accuracy = results$Accuracy,
  TPR = results$TPR,
  FPR = results$FPR,
  Precision = results$Precision
))
```

```
##   model tuning  auroc threshold  accuracy TPR         FPR Precision
## 1   KNN    k=3 0.9999      0.04 0.9930741   1 0.007154642 0.8219512
```

```r
roc_df <- data.frame(
  truth = as.numeric(data$Class)-1,
  predicted = predicted
)

#Storing ROC curve
P3 <- roc_df %>%
  ggplot(aes(d = truth, m = predicted)) +
  geom_roc(n.cuts = 0) +
  theme_bw() +
  labs(title = "KNN ROC",
       caption = glue("AUC is {round(1-auc, 4)}"))
```

I choose this model because the TPR is 100% while the FPR is less than 1% making it an optimal model. We will see on the hold out data whether this model is actually over fit in the end because I remain fairly skeptical of this one.

## Linear Discriminant Analysis

```r
tic()
fitControl <- trainControl(method = "cv",
                           number = 10,
                           classProbs = T,
                           savePredictions = TRUE,
                           allowParallel = T)

y <- data$Class
x <- data.matrix(data[2:4])

unregister_dopar()
Mycluster <-  makeCluster(detectCores()-2)
registerDoParallel(Mycluster)

lda <- train(
      y=y,
      x = x,
      method = "lda",
      trControl = fitControl,
      metric = "ROC"
    )

stopCluster(Mycluster)

toc()
```

```
## 12.774 sec elapsed
```

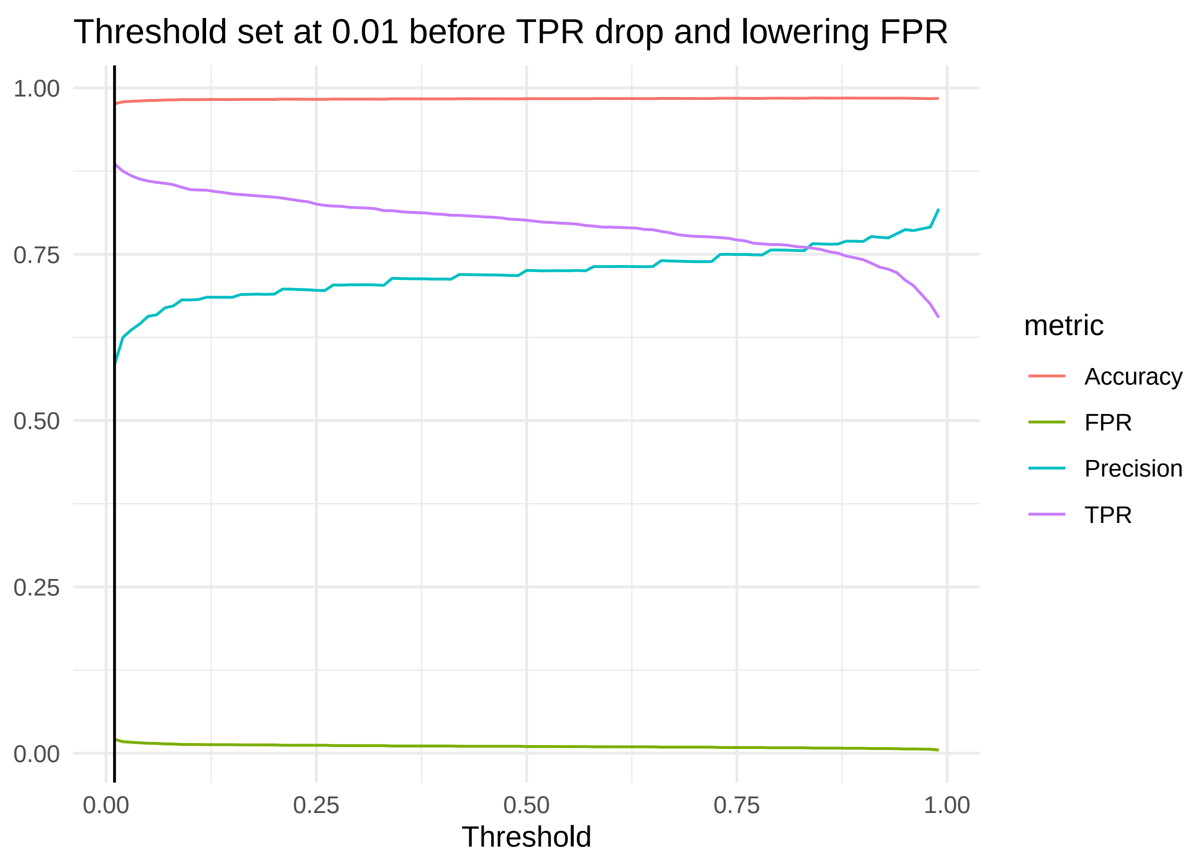
Because there are no tuning parameters, this was a relatively light weight run. Lets look at some performance metrics.

```
lda$results
```

```
##   parameter  Accuracy    Kappa  AccuracySD    KappaSD
## 1      none 0.9839028 0.752686 0.001983461 0.02862634
```

Accuracy is high, but lower than some models we've seen. Let's train our model on the entire set and look at some more metrics.

```
fitControl <- caret::trainControl(method='none',
                                  classProbs = T,
                                  savePredictions = TRUE)

x <- data.matrix(x)

lda_1 <- train(
  y = y,
  x = x,
  method = 'lda',
  trControl = fitControl,
  metric =  "ROC")

predicted <- predict(lda_1$finalModel, x, type = "prob")$posterior[,2]

range_of_thresholds <- seq(0.01,0.99, 0.01)

threshold_selection <- purrr::map_dfr(range_of_thresholds, ~{
  data.frame(
    Threshold = .x,
    TPR = ModelMetrics::sensitivity(as.numeric(y)-1, predicted = predicted, cutoff = .x),
    FPR = 1 - ModelMetrics::specificity(as.numeric(y)-1, predicted = predicted, cutoff = .x),
    Precision = ModelMetrics::precision(as.numeric(y)-1, predicted = predicted, cutoff = .x),
    Accuracy = sum(ifelse(predicted>.x, "Blue_Tarp", "Other") == y)/length(y)
  )
})

threshold_selection %>%
  pivot_longer(cols = 2:5, names_to = 'metric') %>%
  ggplot(aes(x = Threshold, y = value)) +
  geom_line(aes(group = metric, color = metric)) +
  geom_vline(xintercept = 0.01) +
  labs(title = "Threshold set at 0.01 before TPR drop and lowering FPR", y = "") +
  theme_minimal()
```



```
results <- threshold_selection[threshold_selection$Threshold == 0.01,]

rates <- prediction(predicted, data$Class)
auc <- performance(rates, measure = "auc")
auc <- auc@y.values[[1]]

(cv_results_lda <- data.frame(
  model = "LDA",
  tuning = "N/A",
  auroc = 1-round(auc,4),
  threshold = 0.01,
  accuracy = results$Accuracy,
  TPR = results$TPR,
  FPR = results$FPR,
  Precision = results$Precision
))
```

```
##   model tuning  auroc threshold  accuracy       TPR        FPR Precision
## 1   LDA    N/A 0.9889      0.01 0.9761863 0.8857567 0.02082687 0.5841487
```

```
roc_df <- data.frame(
  truth = as.numeric(data$Class)-1,
  predicted = predicted
)

#Storing ROC curve
P4 <- roc_df %>%
  ggplot(aes(d = truth, m = predicted)) +
  geom_roc(n.cuts = 0) +
  theme_bw() +
  labs(title = "LDA ROC",
       caption = glue("AUC is {round(1-auc, 4)}"))
```

It seems like the LDA while retaining a high accuracy is missing some of the BLUE TARP classifications which is costing its true positive rate. I probably wouldn't stick with this model because the TPR is so important for this problem! But, time will tell. We need to predict the test data here and evaluate to be sure.

## Quadriatic Discriminant Analysis

```
tic()
fitControl <- trainControl(method = "cv",
                           number = 10,
                           classProbs = T,
                           savePredictions = TRUE,
                           allowParallel = T)

y <- data$Class
x <- data.matrix(data[2:4])

unregister_dopar()
Mycluster <-  makeCluster(detectCores()-2)
registerDoParallel(Mycluster)

qda <- train(
      y=y,
      x = x,
      method = "qda",
      trControl = fitControl,
      metric = "ROC"
    )

stopCluster(Mycluster)

toc()
```

```
## 13.04 sec elapsed
```

This run time was similar to LDA. Will not cache this model. There is no tuning parameter for the QDA model, so moving to train model on the entire sample.

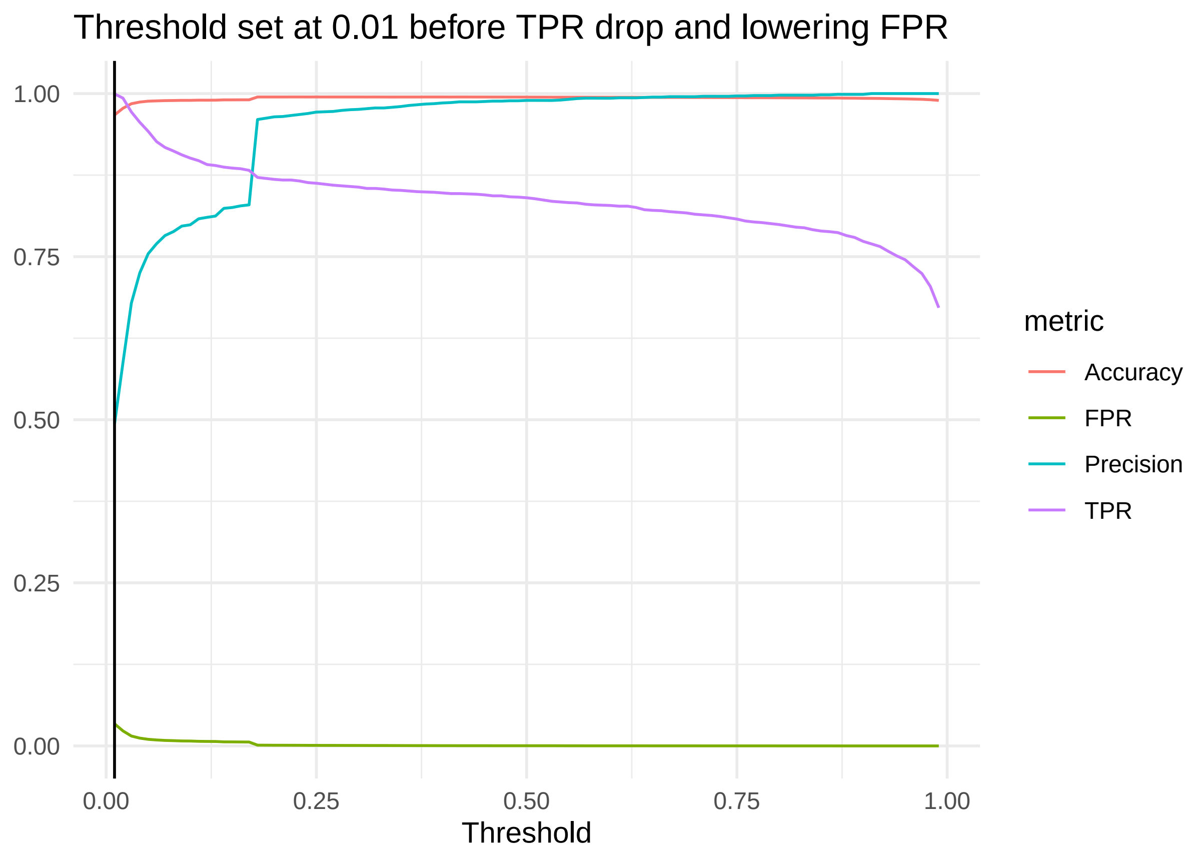
```
qda$results
```

```
##   parameter Accuracy     Kappa   AccuracySD    KappaSD
## 1      none 0.994608 0.9058222 0.0009171906 0.01746779
```

Looking at some preliminary results here, CV10 Accuracy is pretty high, but is similar to other models

```
fitControl <- caret::trainControl(method='none',
                                  classProbs = T,
                                  savePredictions = TRUE)

x <- data.matrix(x)
```

```r
qda_1 <- train(
  y = y,
  x = x,
  method = 'qda',
  trControl = fitControl,
  metric =  "ROC")

predicted <- predict(qda_1$finalModel, x, type = "prob")$posterior[,2]

range_of_thresholds <- seq(0.01,0.99, 0.01)

threshold_selection <- purrr::map_dfr(range_of_thresholds, ~{
  data.frame(
    Threshold = .x,
    TPR = ModelMetrics::sensitivity(as.numeric(y)-1, predicted = predicted, cutoff = .x),
    FPR = 1 - ModelMetrics::specificity(as.numeric(y)-1, predicted = predicted, cutoff = .x),
    Precision = ModelMetrics::precision(as.numeric(y)-1, predicted = predicted, cutoff = .x),
    Accuracy = sum(ifelse(predicted>.x, "Blue_Tarp", "Other") == y)/length(y)
  )
})

threshold_selection %>%
  pivot_longer(cols = 2:5, names_to = 'metric') %>%
  ggplot(aes(x = Threshold, y = value)) +
  geom_line(aes(group = metric, color = metric)) +
  geom_vline(xintercept = 0.01) +
  labs(title = "Threshold set at 0.01 before TPR drop and lowering FPR", y = "") +
  theme_minimal()
```



```r
results <- threshold_selection[threshold_selection$Threshold == 0.01,]

rates <- prediction(predicted, data$Class)
auc <- performance(rates, measure = "auc")
auc <- auc@y.values[[1]]

(cv_results_qda <- data.frame(
  model = "QDA",
  tuning = "N/A",
  auroc = 1-round(auc,4),
  threshold = 0.01,
  accuracy = results$Accuracy,
  TPR = results$TPR,
  FPR = results$FPR,
  Precision = results$Precision
))
```

```
##   model tuning  auroc threshold  accuracy       TPR        FPR Precision
## 1   QDA    N/A 0.9982      0.01 0.9671574 0.9995054 0.03391104 0.4932878
```

```r
roc_df <- data.frame(
  truth = as.numeric(data$Class)-1,
  predicted = predicted
)

#Storing ROC curve
P5 <- roc_df %>%
  ggplot(aes(d = truth, m = predicted)) +
  geom_roc(n.cuts = 0) +
  theme_bw() +
  labs(title = "QDA ROC",
       caption = glue("AUC is {round(1-auc, 4)}"))
```

I choose to set the threshold for this model at 0.01. I chose this threshold because the model obtains almost 100% TPR and only 3% FPR. This is exceptional performance and will rival some of the other models we have. We have now two more models to look at.

## Random Forest

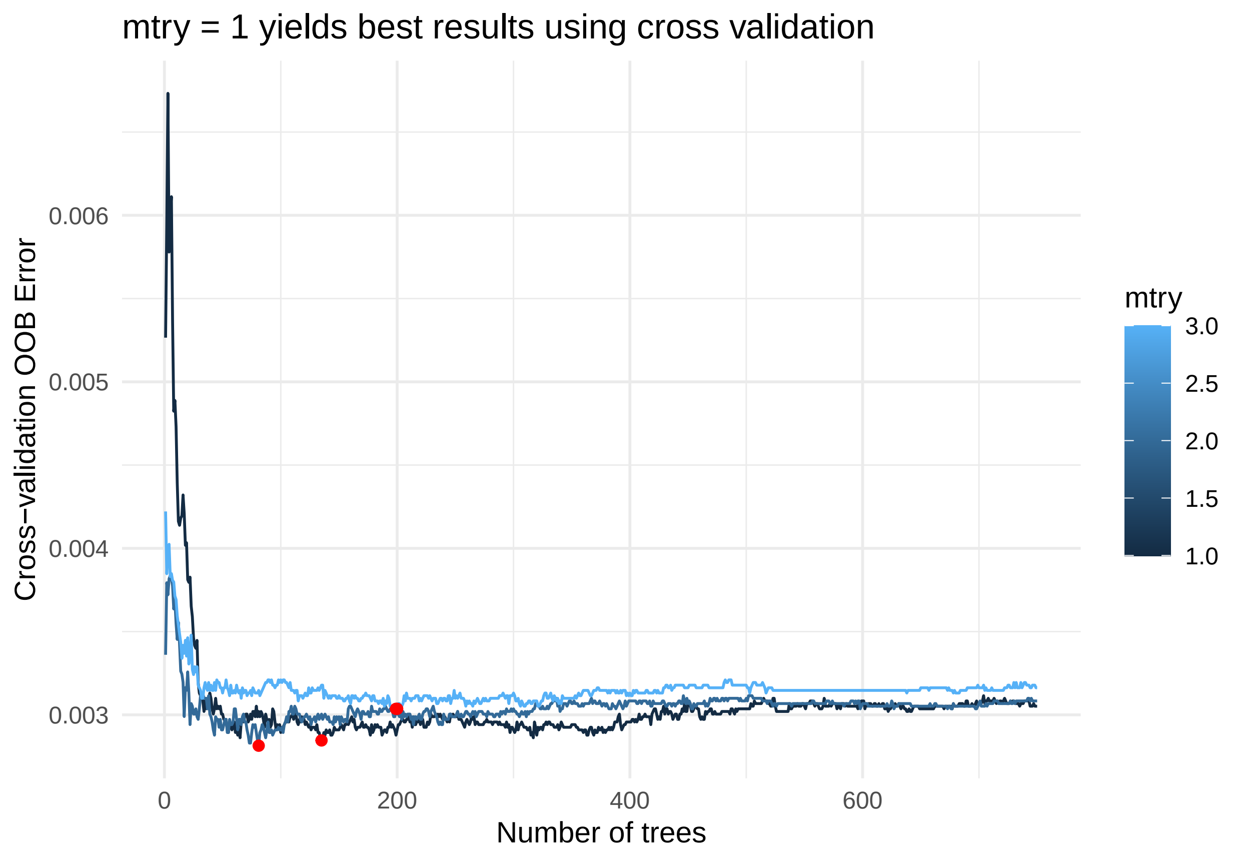
```
tic()
fitControl <- trainControl(method = "cv",
                           number = 10,
                           classProbs = T,
                           savePredictions = TRUE,
                           allowParallel = T)

y <- data$Class
x <- data.matrix(data[2:4])

OOB <- list()

for (i in 1:3){
  unregister_dopar()
  Mycluster <-  makeCluster(detectCores()-2)
  registerDoParallel(Mycluster)

  rf <- train(
        y=y,
        x = x,
        method = "rf",
        ntree = 750,
        trControl = fitControl,
        tuneGrid = expand.grid(data.frame(mtry = i)),
        metric = "ROC"
      )

  stopCluster(Mycluster)

  OOB[[i]] <- rf$finalModel$err.rate[,1]
}

toc()
```

```
## 328.538 sec elapsed
```

Running this took several minutes, but running parallel cut times significantly it would seem.

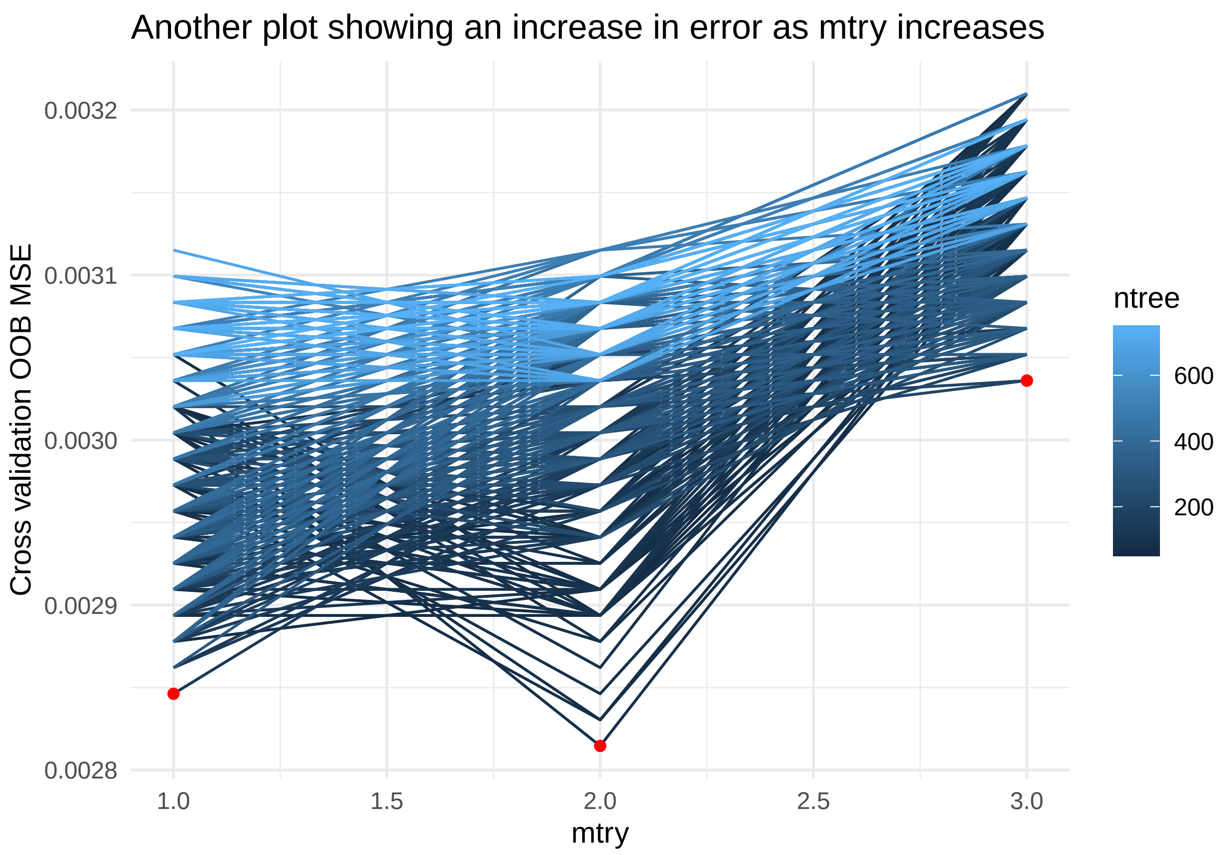
```
results <- data.frame(ntree = 1:750,
           mtry_1 = OOB[[1]],
           mtry_2 = OOB[[2]],
           mtry_3 = OOB[[3]])

results <- results %>%
```

```r
  pivot_longer(cols = 2:4, names_to = "mtry") %>%
  mutate(mtry = as.numeric(str_remove(string = mtry, pattern = "mtry_"))) %>%
  rename(OOB_error = value)

minima <- results %>%
  group_by(mtry) %>%
  filter(OOB_error == min(OOB_error))

results %>%
  ggplot(aes(x = ntree, y = OOB_error, color = mtry, group = mtry)) +
  geom_line() +
  geom_point(data=minima, color='red') +
  labs(title = "mtry = 1 yields best results using cross validation",
       y = "Cross-validation OOB Error",
       x = "Number of trees") +
  theme_minimal()
```



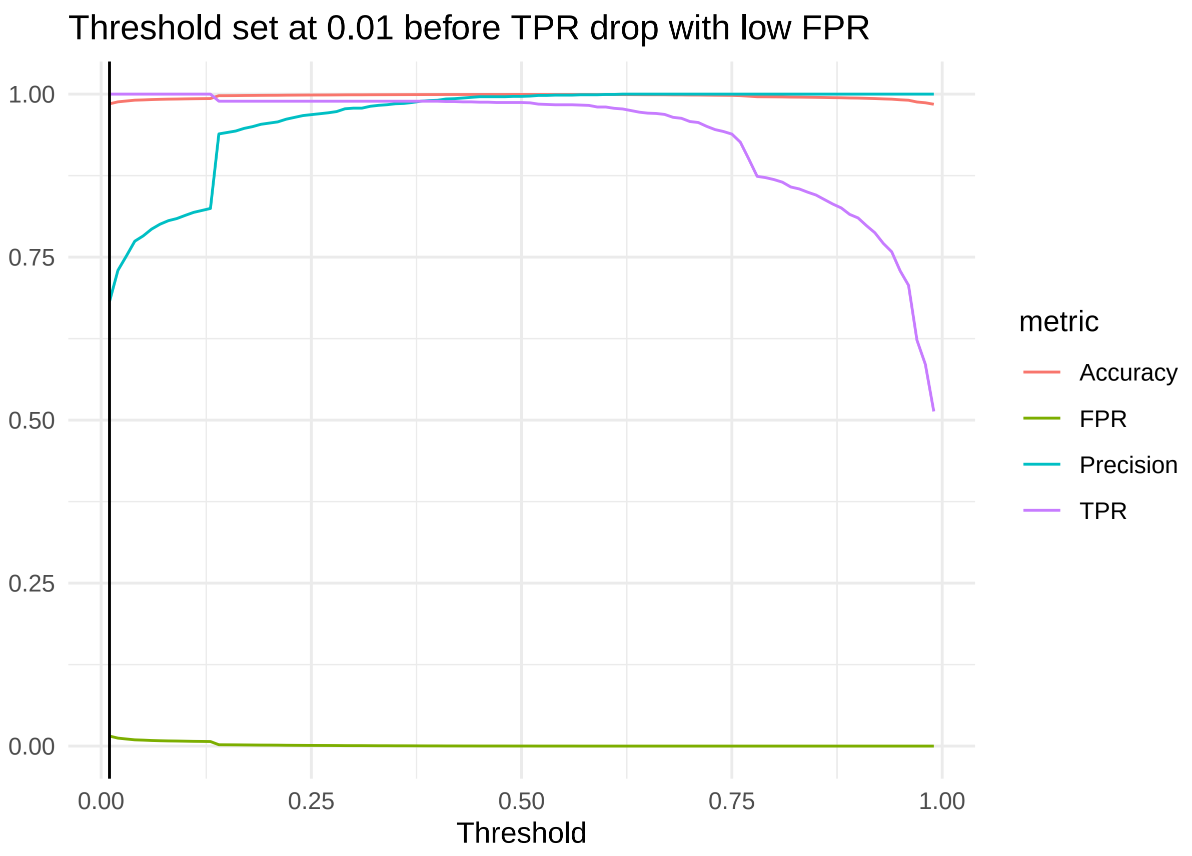
Originally, I ran this only with `ntree = 500`, but after plotting, I saw movement and it looked like the error might decrease. So, I ran it with `ntree = 750` instead. After doing this, I realized that the lines look pretty stable here. The lowest OOB MSE is using around 250 trees with `mtry = 1`.

```r
ggplot(results %>% filter(ntree >50), aes(x=mtry, y=OOB_error, color=ntree, group=ntree)) +
  geom_line() +
  geom_point(data=minima, color='red') +
  labs(title = "Another plot showing an increase in error as mtry increases",
       y = "Cross validation OOB MSE") +
  theme_minimal()
```



Here we see more clearly a trend in the previous plot… As `mtry` increases, so does error. We also see that among the 3 values of `mtry` there is a sweet spot of `ntree` in the 200s. The middle of the plot is lighter blue and the top the plot is very dark blue meaning that values of `ntree` above or below the 200s seem to have more error.

```
fitControl <- caret::trainControl(method='none',
                                  classProbs = T,
                                  savePredictions = TRUE)

rf_1 <- train(
        y=y,
        x = x,
        method = "rf",
        ntree = 254,
        trControl = fitControl,
        tuneGrid = expand.grid(data.frame(mtry = 1)),
        metric = "ROC"
      )

predicted <- predict(rf_1$finalModel, x, type = "prob")[,2]

range_of_thresholds <- seq(0.01,0.99, 0.01)

threshold_selection <- purrr::map_dfr(range_of_thresholds, ~{
  data.frame(
    Threshold = .x,
    TPR = ModelMetrics::sensitivity(as.numeric(y)-1, predicted = predicted, cutoff = .x),
    FPR = 1 - ModelMetrics::specificity(as.numeric(y)-1, predicted = predicted, cutoff = .x),
    Precision = ModelMetrics::precision(as.numeric(y)-1, predicted = predicted, cutoff = .x),
    Accuracy = sum(ifelse(predicted>.x, "Blue_Tarp", "Other") == y)/length(y)
  )
})

threshold_selection %>%
  pivot_longer(cols = 2:5, names_to = 'metric') %>%
  ggplot(aes(x = Threshold, y = value)) +
  geom_line(aes(group = metric, color = metric)) +
  geom_vline(xintercept = 0.01) +
  labs(title = "Threshold set at 0.01 before TPR drop with low FPR", y = "") +
  theme_minimal()
```



```
results <- threshold_selection[threshold_selection$Threshold == 0.01,]

rates <- prediction(predicted, data$Class)
auc <- performance(rates, measure = "auc")
auc <- auc@y.values[[1]]

(cv_results_rf <- data.frame(
  model = "Random Forest",
  tuning = "ntree = 254; mtry = 1",
  auroc = 1-round(auc,4),
  threshold = 0.01,
  accuracy = results$Accuracy,
  TPR = results$TPR,
  FPR = results$FPR,
  Precision = results$Precision
))
```

```
##            model                tuning  auroc threshold  accuracy TPR        FPR
## 1 Random Forest ntree = 254; mtry = 1 0.9999      0.01 0.9850888   1 0.01540371
##   Precision
## 1 0.6819562
```

```
roc_df <- data.frame(
  truth = as.numeric(data$Class)-1,
  predicted = predicted
)

#Storing ROC curve
P6 <- roc_df %>%
  ggplot(aes(d = truth, m = predicted)) +
  geom_roc(n.cuts = 0) +
  theme_bw() +
  labs(title = "Random Forest ROC",
       caption = glue("AUC is {round(1-auc, 4)}"))
```

I am selecting the threshold values of 0.01 which will keep my TPR very high while my FPR is less than 2% (1.5%). Threshold selection for this model is not extremely influential (the typical 0.5 threshold would perform almost as well.), but selecting this threshold will improve our metrics even if only by a little. This model also seems to perform quite well - like others we have seen before.

## Support Vector Machines

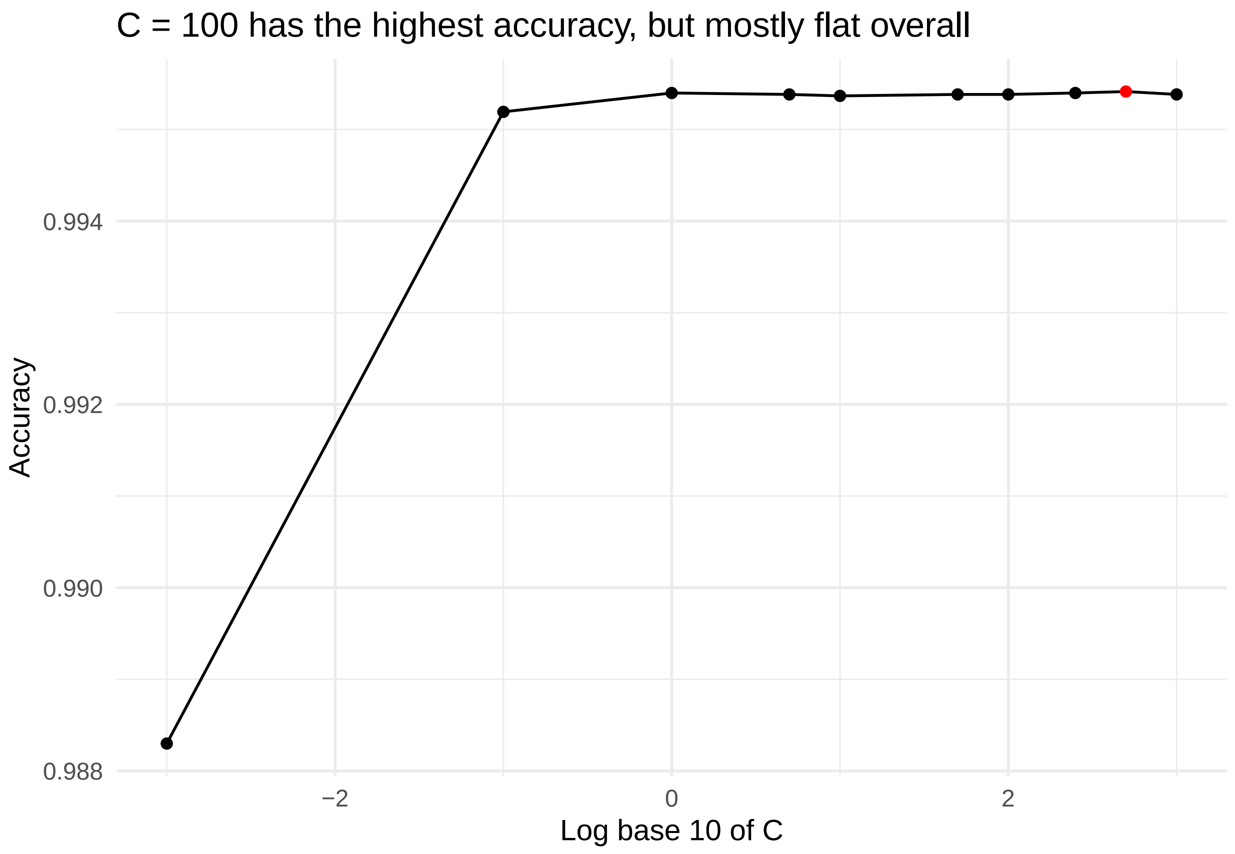
```

tic()
fitControl <- trainControl(method = "cv",
                           number = 10,
                           classProbs = T,
                           savePredictions = TRUE,
                           allowParallel = T)

y <- data$Class
x <- data.matrix(data[2:4])

unregister_dopar()
Mycluster <-  makeCluster(detectCores()-2)
registerDoParallel(Mycluster)

svm_linear <- train(
      y=y,
      x = x,
      method = "svmLinear",
      trControl = fitControl,
      tuneGrid = expand.grid(data.frame(C = c(0.001,0.1,1,5,10,50,100,250, 500, 1000))),
      metric = "ROC"
    )

stopCluster(Mycluster)
```

```
toc()
```

**Linear Kernel**

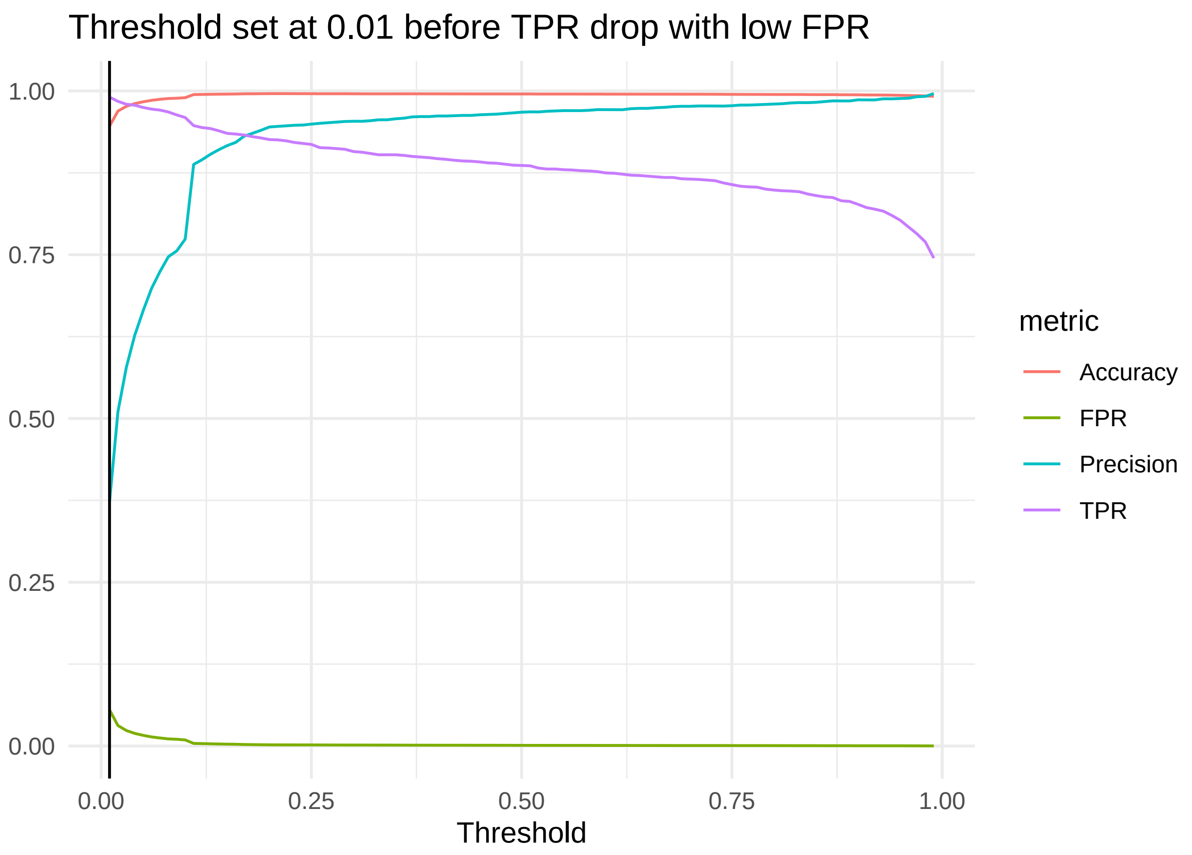
```
## 286.946 sec elapsed
```

```
svm_linear$results %>%
  mutate(C = log10(C),
         point_color = ifelse(Accuracy == max(Accuracy), "red", "black")) %>%
  ggplot(aes(x = C, y = Accuracy, group = 1)) +
  geom_line() +
    geom_point(aes(color = point_color)) +
  labs(title = "C = 100 has the highest accuracy, but mostly flat overall",
       x = "Log base 10 of C") +
  theme_minimal() +
  theme(legend.position = "none") +
  scale_color_manual(values = c("black", "red"))
```



Different values of C in this case don't seem to affect the model that much. Even before C passes zero, the accuracy is about the same in the model overall with values of C after that. I am going to choose the value of C with the highest accuracy which in this case is C = 100.

```
fitControl <- caret::trainControl(method='none',
                                  classProbs = T,
                                  savePredictions = TRUE)

svm_linear_1 <- train(
      y=y,
      x = x,
      method = "svmLinear",
      trControl = fitControl,
      tuneGrid = expand.grid(data.frame(C = 100)),
      metric = "ROC"
    )

predicted <- predict(svm_linear_1, x, type = "prob")[,2]

range_of_thresholds <- seq(0.01,0.99, 0.01)

threshold_selection <- purrr::map_dfr(range_of_thresholds, ~{
  data.frame(
    Threshold = .x,
    TPR = ModelMetrics::sensitivity(as.numeric(y)-1, predicted = predicted, cutoff = .x),
    FPR = 1 - ModelMetrics::specificity(as.numeric(y)-1, predicted = predicted, cutoff = .x),
    Precision = ModelMetrics::precision(as.numeric(y)-1, predicted = predicted, cutoff = .x),
    Accuracy = sum(ifelse(predicted>.x, "Blue_Tarp", "Other") == y)/length(y)
  )
})

threshold_selection %>%
  pivot_longer(cols = 2:5, names_to = 'metric') %>%
  ggplot(aes(x = Threshold, y = value)) +
  geom_line(aes(group = metric, color = metric)) +
  geom_vline(xintercept = 0.01) +
  labs(title = "Threshold set at 0.01 before TPR drop with low FPR", y = "") +
  theme_minimal()
```



```r
results <- threshold_selection[threshold_selection$Threshold == 0.01,]

rates <- prediction(predicted, data$Class)
auc <- performance(rates, measure = "auc")
auc <- auc@y.values[[1]]

(cv_results_svm_linear <- data.frame(
  model = "SVM - Linear",
  tuning = "C=100",
  auroc = 1-round(auc,4),
  threshold = 0.01,
  accuracy = results$Accuracy,
  TPR = results$TPR,
  FPR = results$FPR,
  Precision = results$Precision
))
```

```
##          model tuning  auroc threshold  accuracy       TPR        FPR Precision
## 1 SVM - Linear  C=100 0.9981      0.01 0.9463323 0.9906034 0.05512994 0.3724433
```

```r
roc_df <- data.frame(
  truth = as.numeric(data$Class)-1,
  predicted = predicted
)

#Storing ROC curve
P7 <- roc_df %>%
  ggplot(aes(d = truth, m = predicted)) +
  geom_roc(n.cuts = 0) +
  theme_bw() +
  labs(title = "Linear SVM ROC",
       caption = glue("AUC is {round(1-auc, 4)}"))
```

This model does not seem to be as good as previous ones because when the threshold is at 0.01, the TPR is 0.99 and the FPR is almost 6%. In other models, this was much lower. I am once again going to choose the threshold of 0.01 as having a high TPR is fine even if it results in error in FPR. It is more important to identify people in need than it is to check a few extra images.

```

tic()
fitControl <- trainControl(method = "cv",
                           number = 10,
                           classProbs = T,
                           savePredictions = TRUE,
                           allowParallel = T)

y <- data$Class
x <- data.matrix(data[2:4])

unregister_dopar()
Mycluster <-  makeCluster(detectCores()-2)
registerDoParallel(Mycluster)

svm_poly <- train(
      y=y,
      x = x,
      method = "svmPoly",
      trControl = fitControl,
      tuneGrid = expand.grid(data.frame(C = c(0.001,1,10,100, 1000),
                                        degree = 1:5,
                                        scale = c(0.01, 0.1, 0.25, 0.5, 0.8))),
      metric = "ROC"
    )

stopCluster(Mycluster)

toc()
```

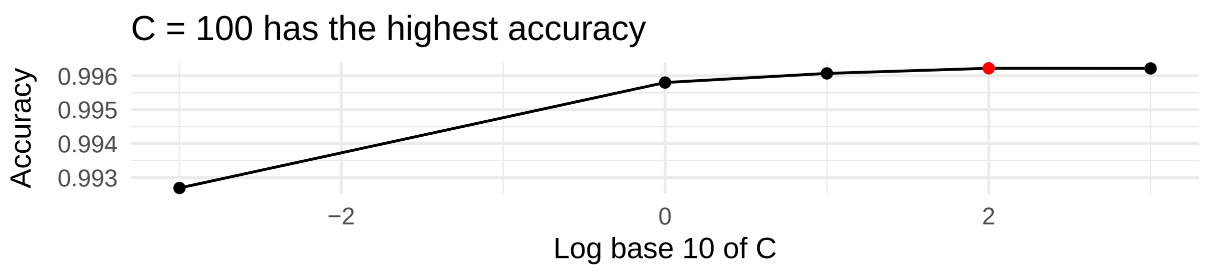
**Polynomial Kernel**

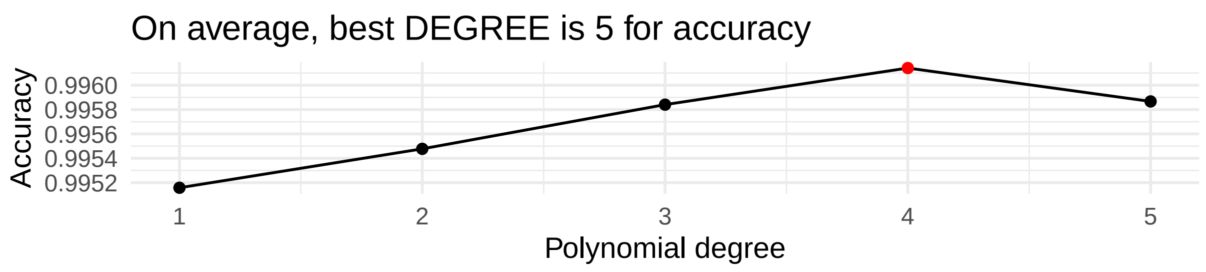
```
## 2861.993 sec elapsed
```

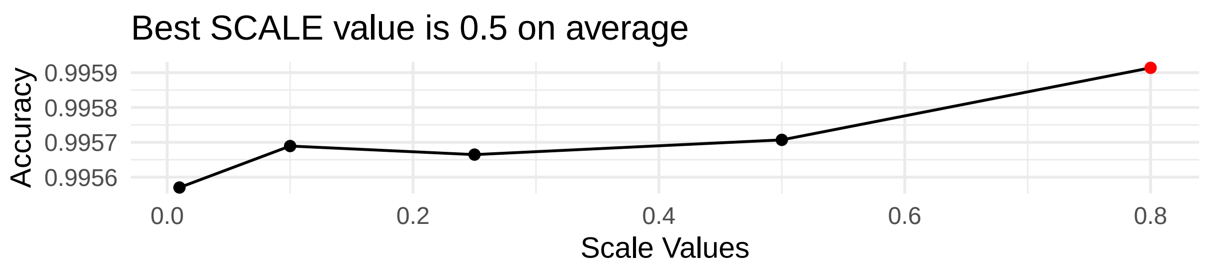
This model took up significant memory and time to run. I cached this and will hopefully not see any problems knitting here.

```
poly_results <- svm_poly$results %>%
  filter(!is.na(KappaSD))

plot_1 <- poly_results %>%
  group_by(C) %>%
  summarise(Accuracy = mean(Accuracy)) %>%
  mutate(C = log10(C),
         point_color = ifelse(Accuracy == max(Accuracy), "red", "black")) %>%
```

```r
  ggplot(aes(x = C, y = Accuracy, group = 1)) +
  geom_line() +
    geom_point(aes(color = point_color)) +
  labs(title = "C = 100 has the highest accuracy",
       x = "Log base 10 of C") +
  theme_minimal() +
  theme(legend.position = "none") +
  scale_color_manual(values = c("black", "red"))

plot_2 <- poly_results %>%
  group_by(degree) %>%
  summarise(Accuracy = mean(Accuracy)) %>%
  mutate(
         point_color = ifelse(Accuracy == max(Accuracy), "red", "black")) %>%
  ggplot(aes(x = degree, y = Accuracy, group = 1)) +
  geom_line() +
    geom_point(aes(color = point_color)) +
  labs(title = "On average, best DEGREE is 5 for accuracy",
       x = "Polynomial degree") +
  theme_minimal() +
  theme(legend.position = "none") +
  scale_color_manual(values = c("black", "red"))

plot_3 <- poly_results %>%
  group_by(scale) %>%
  summarise(Accuracy = mean(Accuracy)) %>%
  mutate(
         point_color = ifelse(Accuracy == max(Accuracy), "red", "black")) %>%
  ggplot(aes(x = scale, y = Accuracy, group = 1)) +
  geom_line() +
    geom_point(aes(color = point_color)) +
  labs(title = "Best SCALE value is 0.5 on average",
       x = "Scale Values") +
  theme_minimal() +
  theme(legend.position = "none") +
  scale_color_manual(values = c("black", "red"))

plot_1/plot_2/plot_3
```

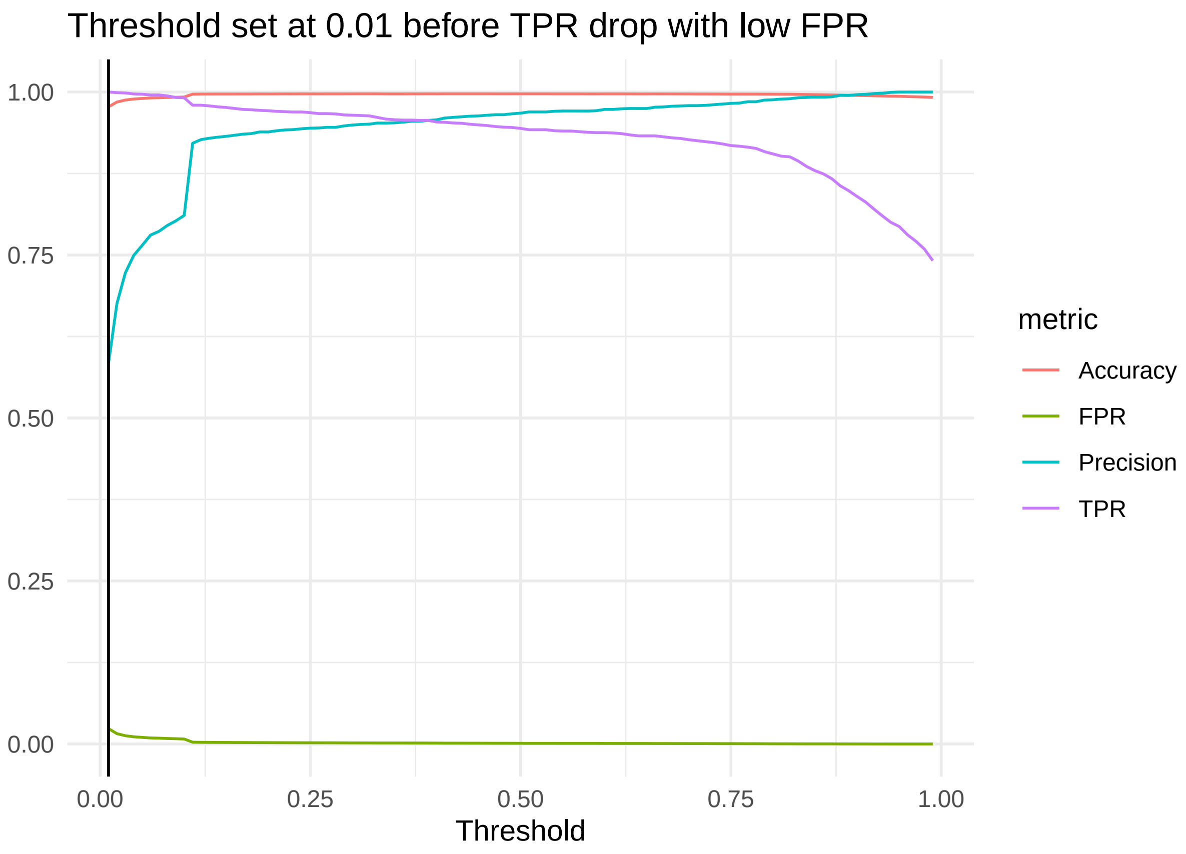






Why these are the best average values overall the best combination is found in the model output, and this model actually doesn't have any of the best average values for the tuning parameters. This may suggest that if a models other parameters are inaccurate or bad, that this value in whatever hyperparameter may make up for the error. But, when finding the best combination of parameters, it requires some parameters to have values with lower average accuracy, but will actually result in the best combination of parameters.

```
fitControl <- caret::trainControl(method='none',
                                  classProbs = T,
                                  savePredictions = TRUE)

svm_poly_1 <- train(
      y=y,
      x = x,
      method = "svmPoly",
      trControl = fitControl,
      tuneGrid = expand.grid(data.frame(C = 1000,
                                        degree = 5,
                                        scale = 0.25)),
      metric = "ROC"
    )

predicted <- predict(svm_poly_1, x, type = "prob")[,2]

range_of_thresholds <- seq(0.01,0.99, 0.01)

threshold_selection <- purrr::map_dfr(range_of_thresholds, ~{
  data.frame(
    Threshold = .x,
    TPR = ModelMetrics::sensitivity(as.numeric(y)-1, predicted = predicted, cutoff = .x),
    FPR = 1 - ModelMetrics::specificity(as.numeric(y)-1, predicted = predicted, cutoff = .x),
    Precision = ModelMetrics::precision(as.numeric(y)-1, predicted = predicted, cutoff = .x),
    Accuracy = sum(ifelse(predicted>.x, "Blue_Tarp", "Other") == y)/length(y)
  )
})

threshold_selection %>%
  pivot_longer(cols = 2:5, names_to = 'metric') %>%
  ggplot(aes(x = Threshold, y = value)) +
  geom_line(aes(group = metric, color = metric)) +
  geom_vline(xintercept = 0.01) +
  labs(title = "Threshold set at 0.01 before TPR drop with low FPR", y = "") +
  theme_minimal()
```



```
results <- threshold_selection[threshold_selection$Threshold == 0.01,]

rates <- prediction(predicted, data$Class)
auc <- performance(rates, measure = "auc")
auc <- auc@y.values[[1]]

(cv_results_svm_poly <- data.frame(
  model = "SVM - Poly",
  tuning = "C=1000; Scale=0.25; degree=5",
  auroc = 1-round(auc,4),
  threshold = 0.01,
  accuracy = results$Accuracy,
```

```r
  TPR = results$TPR,
  FPR = results$FPR,
  Precision = results$Precision
))
```

```
##        model                       tuning  auroc threshold  accuracy TPR
## 1 SVM - Poly C=1000; Scale=0.25; degree=5 0.9997      0.01 0.9771983   1
##          FPR Precision
## 1 0.02355478 0.5837182
```

```r
roc_df <- data.frame(
  truth = as.numeric(data$Class)-1,
  predicted = predicted
)

#Storing ROC curve
P8 <- roc_df %>%
  ggplot(aes(d = truth, m = predicted)) +
  geom_roc(n.cuts = 0) +
  theme_bw() +
  labs(title = "Polynomial SVM ROC",
       caption = glue("AUC is {round(1-auc, 4)}"))
```

For our polynomial kernel SVM, it seems to perform better than the linear kernel as the TPR is 100% and FPR is about at 2%. There seems to be a similar pattern with cutoffs and SVM models where the cutoff are very low - usually at 0.01, and so it is with this one. The optimal cutoff for this model is 0.01 that yields the best results we are looking for.

```r

tic()
fitControl <- trainControl(method = "cv",
                           number = 10,
                           classProbs = T,
                           savePredictions = TRUE,
                           allowParallel = T)

y <- data$Class
x <- data.matrix(data[2:4])

unregister_dopar()
Mycluster <-  makeCluster(detectCores()-2)
registerDoParallel(Mycluster)

svm_radial <- train(
      y=y,
      x = x,
      method = "svmRadial",
      trControl = fitControl,
      tuneGrid = expand.grid(data.frame(C = c(0.001,1,10,100, 1000),
                                        sigma = c(0.01, 1, 10,100, 1000))),
      metric = "ROC"
    )

stopCluster(Mycluster)
```

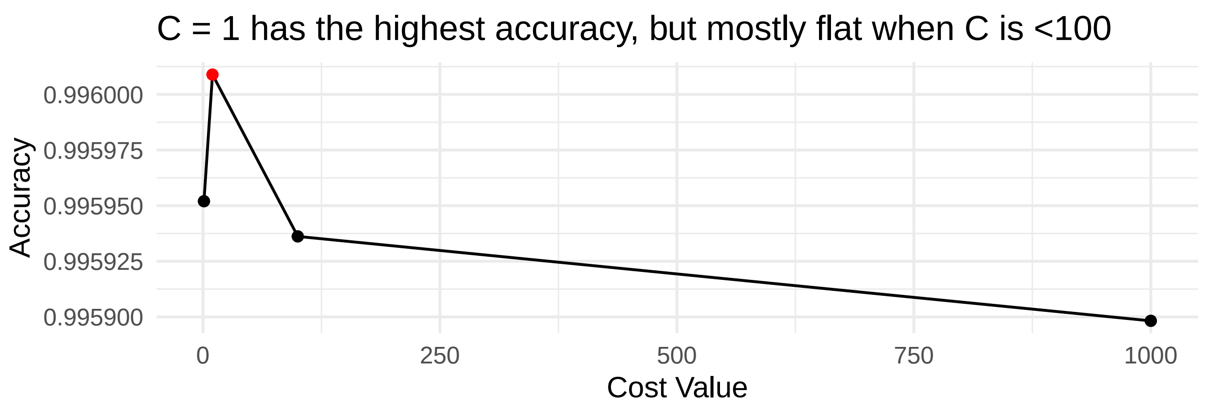
```
toc()
```

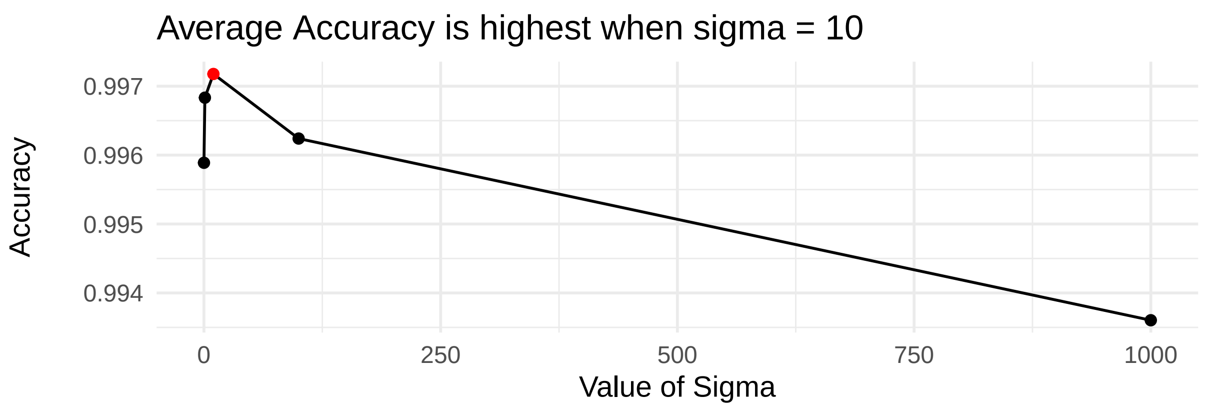
**Radial Kernel**

```
## 5852.829 sec elapsed
```

This took an hour and a half to run in parallel. I am a little surprised by this because the lab didn't seem to take this long.

```
results_svm_rad <- svm_radial$results %>%
  filter(!is.na(AccuracySD))

plot_1 <- results_svm_rad %>%
  group_by(C) %>%
  summarise(Accuracy = mean(Accuracy)) %>%
  mutate(
         point_color = ifelse(Accuracy == max(Accuracy), "red", "black")) %>%
  ggplot(aes(x = C, y = Accuracy, group = 1)) +
  geom_line() +
    geom_point(aes(color = point_color)) +
  labs(title = "C = 1 has the highest accuracy, but mostly flat when C is <100",
       x = "Cost Value") +
  theme_minimal() +
  theme(legend.position = "none") +
  scale_color_manual(values = c("black", "red"))

plot_2 <- results_svm_rad %>%
  group_by(sigma) %>%
  summarise(Accuracy = mean(Accuracy)) %>%
  mutate(point_color = ifelse(Accuracy == max(Accuracy), "red", "black")) %>%
  ggplot(aes(x = sigma, y = Accuracy, group = 1)) +
  geom_line() +
    geom_point(aes(color = point_color)) +
  labs(title = "Average Accuracy is highest when sigma = 10",
       x = "Value of Sigma") +
  theme_minimal() +
  theme(legend.position = "none") +
  scale_color_manual(values = c("black", "red"))

plot_1/plot_2
```



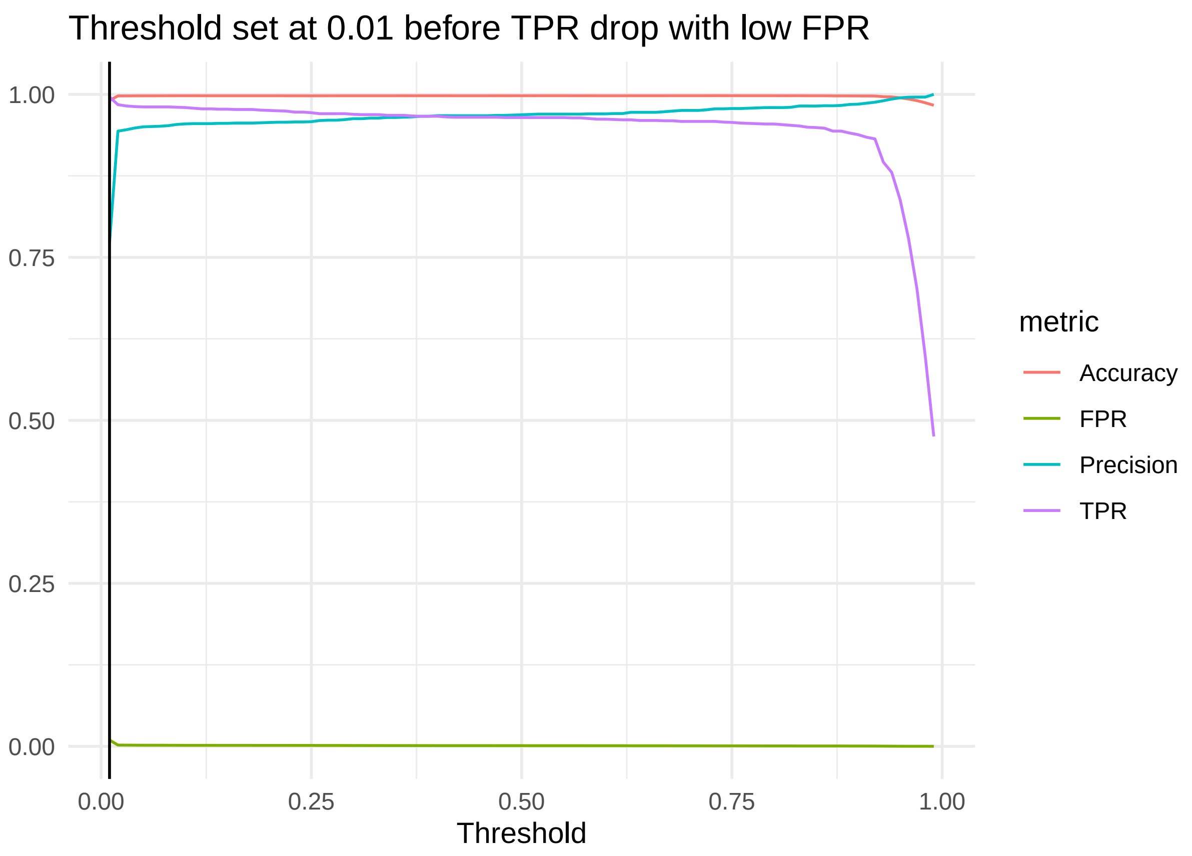


```
svm_radial$bestTune
```

```
##    sigma    C
## 23    10 1000
```

The accuracy remains fairly constant around different values of cost; however, when sigma = 10, it seems to perform better than other values of sigma. The best tune is also printed out which uses the sigma value of 10 which we also found to be better and then C=100 which our chart shows a high mean there.

```
fitControl <- caret::trainControl(method='none',
                                  classProbs = T,
                                  savePredictions = TRUE)

svm_radial_1 <- train(
      y=y,
      x = x,
      method = "svmRadial",
      trControl = fitControl,
      tuneGrid = expand.grid(data.frame(C = 100,
                                        sigma = 10)),
      metric = "ROC"
    )

predicted <- predict(svm_radial_1, x, type = "prob")[,2]

range_of_thresholds <- seq(0.01,0.99, 0.01)

threshold_selection <- purrr::map_dfr(range_of_thresholds, ~{
  data.frame(
    Threshold = .x,
    TPR = ModelMetrics::sensitivity(as.numeric(y)-1, predicted = predicted, cutoff = .x),
    FPR = 1 - ModelMetrics::specificity(as.numeric(y)-1, predicted = predicted, cutoff = .x),
    Precision = ModelMetrics::precision(as.numeric(y)-1, predicted = predicted, cutoff = .x),
    Accuracy = sum(ifelse(predicted>.x, "Blue_Tarp", "Other") == y)/length(y)
  )
})

threshold_selection %>%
  pivot_longer(cols = 2:5, names_to = 'metric') %>%
  ggplot(aes(x = Threshold, y = value)) +
  geom_line(aes(group = metric, color = metric)) +
  geom_vline(xintercept = 0.01) +
  labs(title = "Threshold set at 0.01 before TPR drop with low FPR", y = "") +
  theme_minimal()
```



```
results <- threshold_selection[threshold_selection$Threshold == 0.01,]

rates <- prediction(predicted, data$Class)
auc <- performance(rates, measure = "auc")
auc <- auc@y.values[[1]]

(cv_results_svm_radial <- data.frame(
  model = "SVM - Radial",
  tuning = "C=100; Sigma =10",
  auroc = 1-round(auc,4),
  threshold = 0.01,
  accuracy = results$Accuracy,
  TPR = results$TPR,
  FPR = results$FPR,
  Precision = results$Precision
))
```

```
##          model           tuning  auroc threshold  accuracy       TPR
## 1 SVM - Radial C=100; Sigma =10 0.9987      0.01 0.9904967 0.9960435
##           FPR Precision
## 1 0.009686535 0.7725355
```

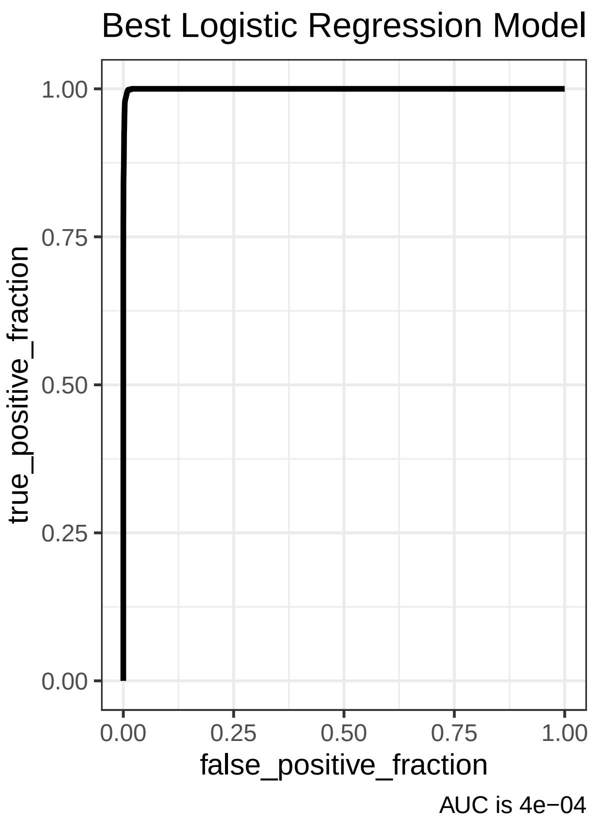
```
roc_df <- data.frame(
  truth = as.numeric(data$Class)-1,
  predicted = predicted
)

#Storing ROC curve
P9 <- roc_df %>%
  ggplot(aes(d = truth, m = predicted)) +
  geom_roc(n.cuts = 0) +
  theme_bw() +
  labs(title = "Radial SVM ROC",
       caption = glue("AUC is {round(1-auc, 4)}"))
```

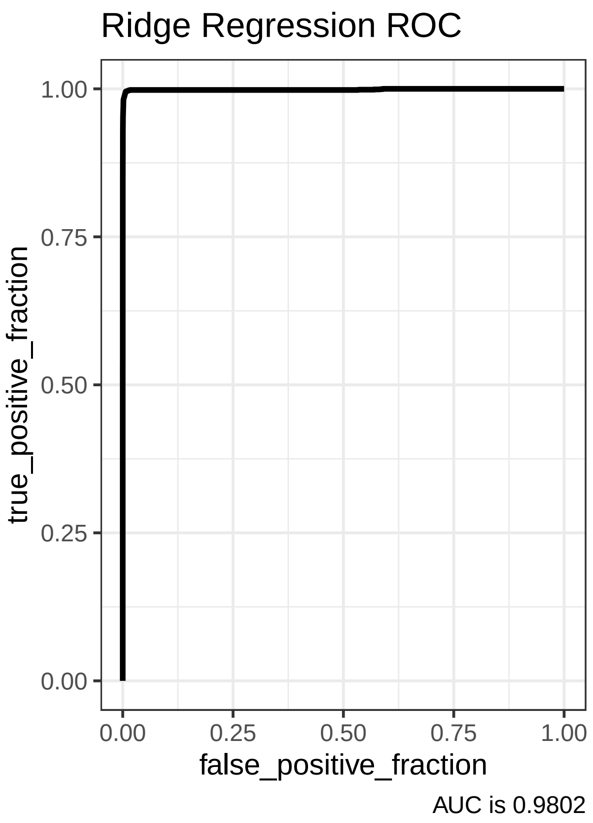
The radial kernel SVM didn't seem to perform as well as the polynomial kernel. Although, notably, in this model the FPR is VERY low, but we are missing a few classifications in the TPR which some of the other models correctly classified. I won't make any decisions about this model at this point, but will wait until we have the holdout data to test our model.

This threshold was selected maximizing the TPR while keeping the FPR reasonably low. In this case, the FPR stayed low for almost every threshold.

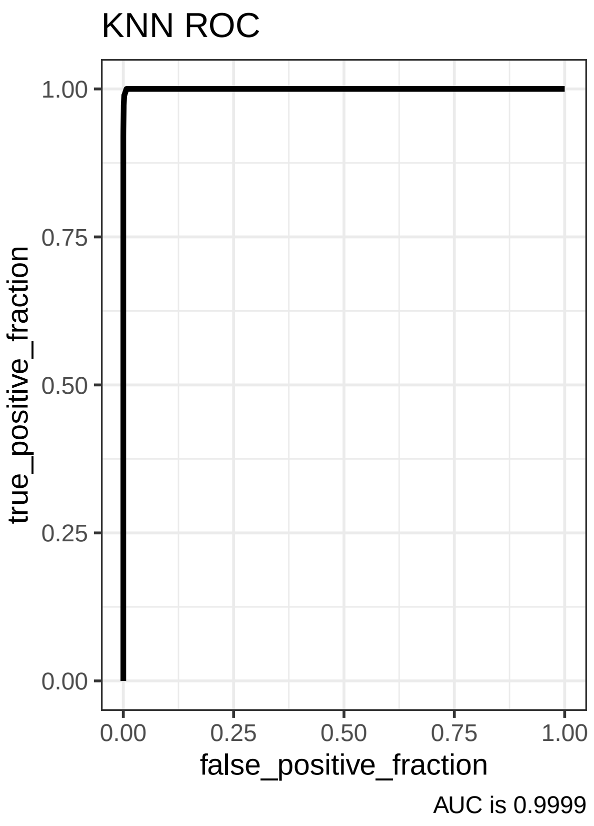
Next, we will look at all of our model's ROC curves

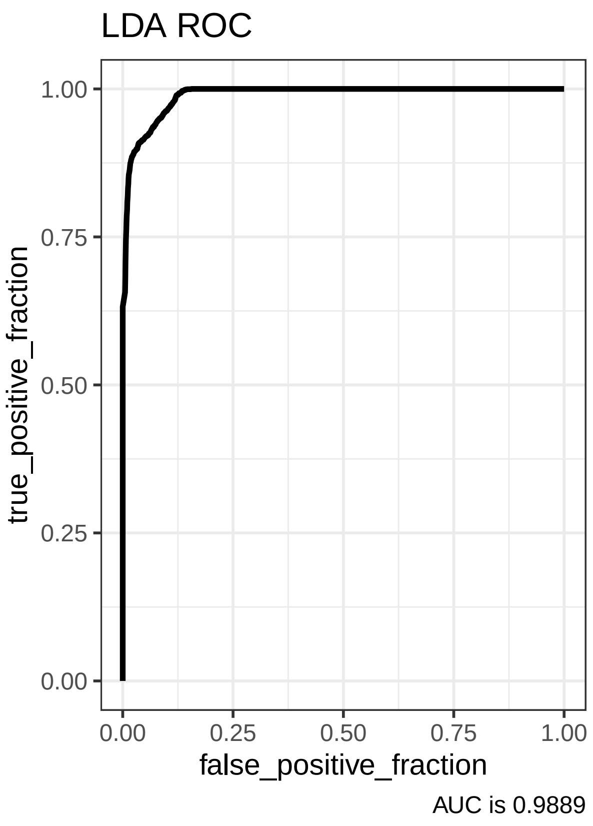
```
P1 + P2
```


Best Logistic Regression Model
true_positive_fraction
false_positive_fraction
1.00
0.75
0.50
0.25
0.00
0.00
0.25
0.50
0.75
1.00
AUC is 4e−04

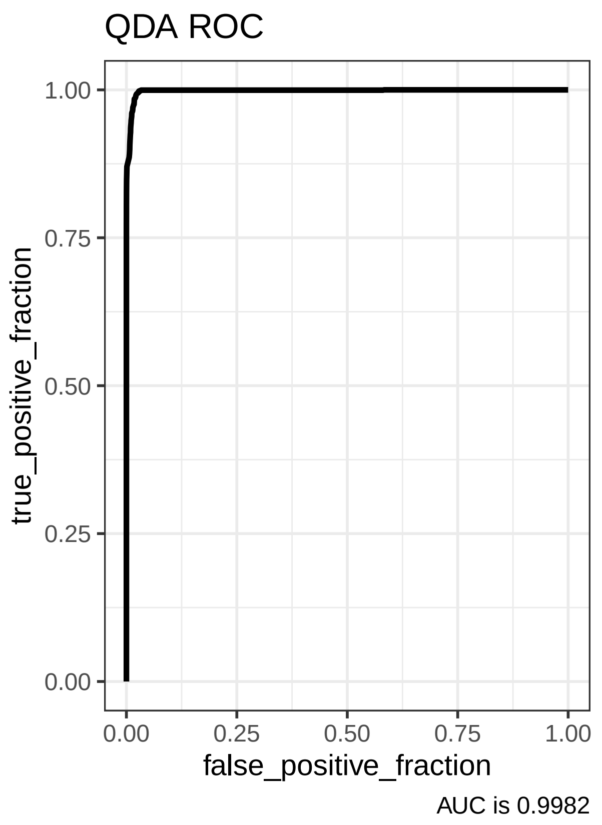

Ridge Regression ROC
true_positive_fraction
false_positive_fraction
1.00
0.75
0.50
0.25
0.00
0.00
0.25
0.50
0.75
1.00
AUC is 0.9802

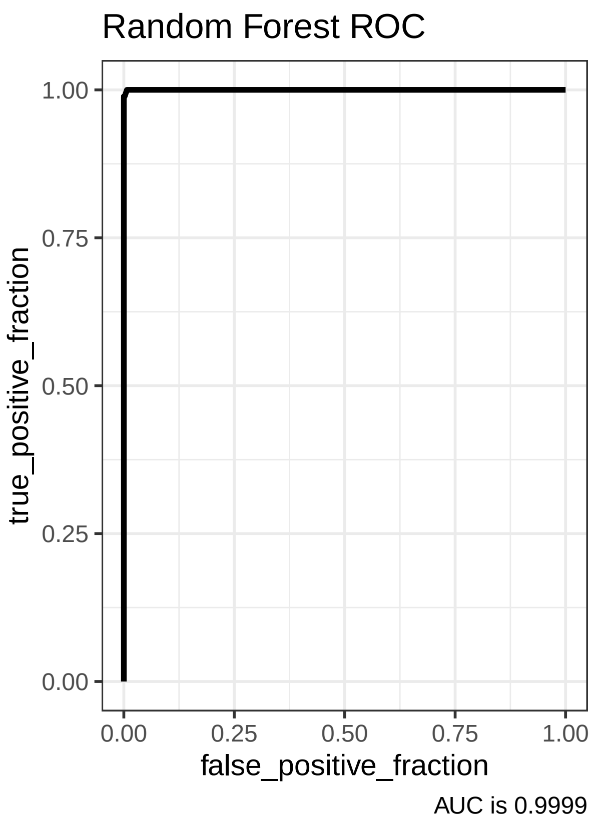
```
P3 + P4
```


KNN ROC
true_positive_fraction
false_positive_fraction
1.00
0.75
0.50
0.25
0.00
0.00
0.25
0.50
0.75
1.00
AUC is 0.9999

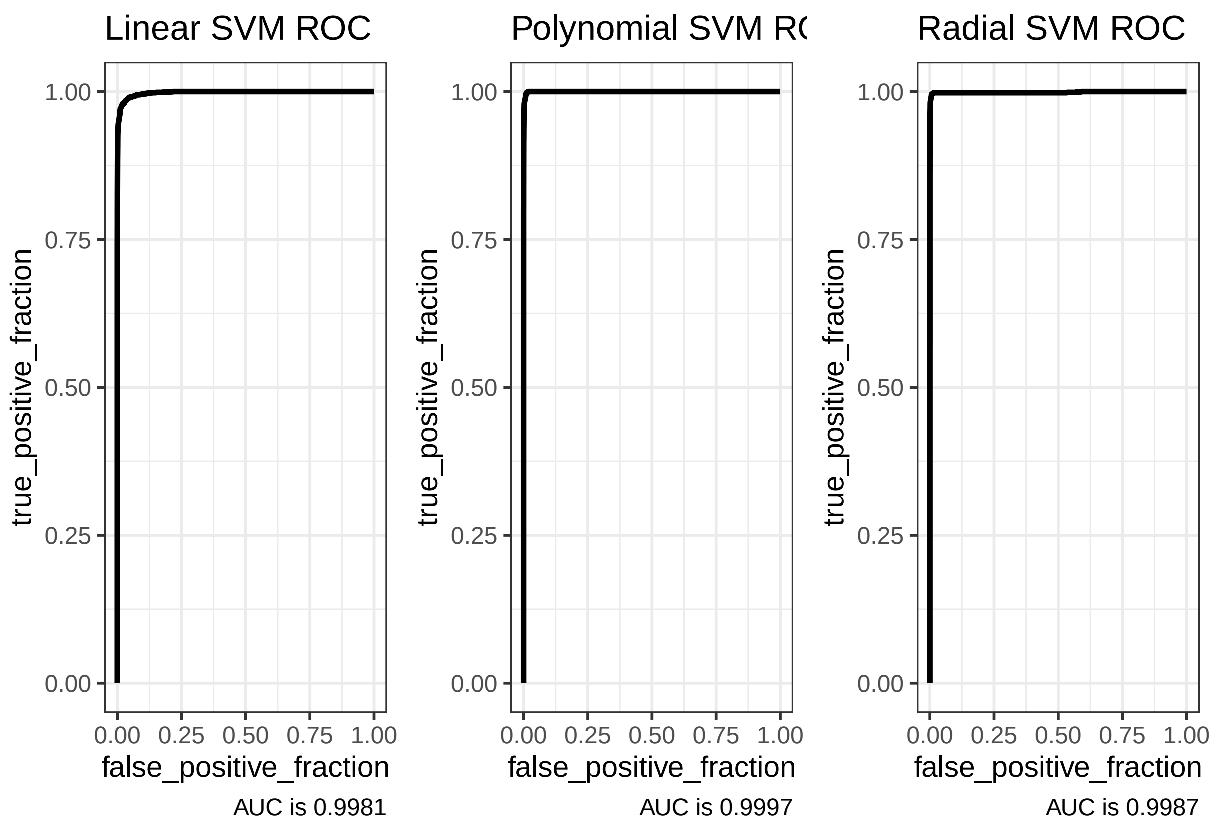

LDA ROC
true_positive_fraction
false_positive_fraction
1.00
0.75
0.50
0.25
0.00
0.00
0.25
0.50
0.75
1.00
AUC is 0.9889

```
P5 + P6
```


QDA ROC
true_positive_fraction
false_positive_fraction
0.00
0.25
0.50
0.75
1.00
AUC is 0.9982


Random Forest ROC
true_positive_fraction
false_positive_fraction
0.00
0.25
0.50
0.75
1.00
AUC is 0.9999

```
P7 + P8 + P9
```



These represent the ROC curves from the selected models from cross-validation then trained on the whole data set. A few notes about these figures..

First, the area under the curve is written in the bottom right hand corner for comparison if desired.

When analyzing these various charts, most of the ROC curves look very similar; however, there are a few differences. For example, LDA, Ridge, and QDA ROC curves look as good as some of the other ones. Also, the linear kernel SVM doesn't look as good as the other SVMs for this data. We will take this information and use it when evaluating the tables.

### Training CV Model Table

```
CV_Error_table <- bind_rows(
  cv_results_LR,
  cv_results_Ridge,
  cv_results_lda,
  cv_results_qda,
  cv_results_knn,
  cv_results_rf,
  cv_results_svm_linear,
  cv_results_svm_poly,
  cv_results_svm_radial
) %>%
  arrange(desc(TPR))

knitr::kable(CV_Error_table)
```

| model | tuning | auroc | threshold | accuracy | TPR | FPR | Precision |
|---|---|---|---|---|---|---|---|
| KNN | k=3 | 0.9999 | 0.04 | 0.9930741 | 1.0000000 | 0.0071546 | 0.8219512 |
| Random Forest | ntree = 254; mtry = 1 | 0.9999 | 0.01 | 0.9850888 | 1.0000000 | 0.0154037 | 0.6819562 |
| SVM - Poly | C=1000; Scale=0.25; degree=5 | 0.9997 | 0.01 | 0.9771983 | 1.0000000 | 0.0235548 | 0.5837182 |
| QDA | N/A | 0.9982 | 0.01 | 0.9671574 | 0.9995054 | 0.0339110 | 0.4932878 |
| Logistic Regression | N/A | 0.9996 | 0.02 | 0.9854999 | 0.9985163 | 0.0149300 | 0.6883737 |
| SVM - Radial | C=100; Sigma =10 | 0.9987 | 0.01 | 0.9904967 | 0.9960435 | 0.0096865 | 0.7725355 |
| SVM - Linear | C=100 | 0.9981 | 0.01 | 0.9463323 | 0.9906034 | 0.0551299 | 0.3724433 |
| Ridge Regression | Lambda = 0.00401 | 0.9802 | 0.04 | 0.8614190 | 0.9302671 | 0.1408550 | 0.1790746 |
| LDA | N/A | 0.9889 | 0.01 | 0.9761863 | 0.8857567 | 0.0208269 | 0.5841487 |

To review, the process I used to generate this table (refer to the very beginning of this paper for more info on entire process):

1. Using 10 - Fold Cross Validation, train and model and tune

From our training errors, we can see that overall we've developed some models that are pretty good at predicting on the training data. From just this table, our top model is the KNN which perfectly predicts all Blue Tarps and has a less than 1% False positive rate. Its hard to beat this model! Close behind is the Random Forest model which also perfectly predicts all Blue Tarps, but has a slightly higher FPR which is at about 1.5%.

Our next step is an important one, we are going to use the models we trained on to predict on test data. This should give us good insight into some models that are overfit. Correctly classifying every Blue tarp might be overfitting, so we will see how the models hold up to real data. But first, we need to import and clean that data and calculate the relevant statistics!

## Importing and cleaning TEST HOLD OUT data

By observation, I deleted one file that is duplicate of another one: `orthovnir067_ROI_Blue_Tarps_data.txt`. Since this is only prediction data, the extra data (LAT, LONG, etc.) was not needed as it was not in any of the models, so those columns are filtered out.

```r
path <- "Disaster Relief Part 1/Hold+Out+Data/"

files <- list.files(path)

files <- paste0(path, files)

BlueTarps <- !grepl("NOT|NON", files)

final_test_data <- purrr::map2(files, BlueTarps, ~{
  table <- suppressWarnings(suppressMessages(read_table(.x, skip = 7))) %>%
    dplyr::select(dplyr::all_of(c("Y_1", "Lat", "Lon")))

  names(table) <- c("B1", "B2", "B3")

  if (isTRUE(.y)) table$Class <- "Blue_Tarp" else table$Class <- "Other"

  table
})
```

```
final_test_data <- data.table::rbindlist(final_test_data) %>% relocate(Class)

final_test_data$Class <- factor(final_test_data$Class,
                                levels = c( "Other","Blue_Tarp"))

head(final_test_data)
```

```
##    Class  B1 B2 B3
## 1: Other 104 89 63
## 2: Other 101 80 60
## 3: Other 103 87 69
## 4: Other 107 93 72
## 5: Other 109 99 68
## 6: Other 103 73 53
```

```
print("Summary of known data")
```

```
## [1] "Summary of known data"
```

```
summary(data)
```

```
##        Class            Red          Green            Blue
##  Other    :61219   Min.   : 48   Min.   : 48.0   Min.   : 44.0
##  Blue_Tarp: 2022   1st Qu.: 80   1st Qu.: 78.0   1st Qu.: 63.0
##                    Median :163   Median :148.0   Median :123.0
##                    Mean   :163   Mean   :153.7   Mean   :125.1
##                    3rd Qu.:255   3rd Qu.:226.0   3rd Qu.:181.0
##                    Max.   :255   Max.   :255.0   Max.   :255.0
```

```
print("Summary of new data")
```

```
## [1] "Summary of new data"
```

```
summary(final_test_data)
```

```
##        Class               B1              B2              B3
##  Other    :1989697   Min.   : 27.0   Min.   : 28.0   Min.   : 25.00
##  Blue_Tarp:  14480   1st Qu.: 76.0   1st Qu.: 71.0   1st Qu.: 55.00
##                      Median :107.0   Median : 91.0   Median : 66.00
##                      Mean   :118.3   Mean   :105.4   Mean   : 82.36
##                      3rd Qu.:139.0   3rd Qu.:117.0   3rd Qu.: 88.00
##                      Max.   :255.0   Max.   :255.0   Max.   :255.00
```

From our summaries, it seems to match up quite well as b1=red, b2=green, b3=blue. If you look at the median and mean values, while they are different between the data sets, both decrease at a similar rate as you go across colors with the highest median/mean being red, then green, and blue.

```
print("Summary of known data")
```

```
## [1] "Summary of known data"
```

```
summary(data[data$Class=="Blue_Tarp",])
```

```
##        Class           Red             Green             Blue
##  Other    :   0   Min.   : 93.0   Min.   : 92.0   Min.   : 92.0
##  Blue_Tarp:2022   1st Qu.:147.0   1st Qu.:160.0   1st Qu.:175.2
##                   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
```

```
print("Summary of new data")
```

```
## [1] "Summary of new data"
```

```
summary(final_test_data[final_test_data$Class=="Blue_Tarp",])
```

```
##         Class              B1               B2               B3       
##  Other    :    0   Min.   : 48.0   Min.   : 56.0   Min.   : 57.0  
##  Blue_Tarp:14480   1st Qu.: 82.0   1st Qu.:100.0   1st Qu.:123.0  
##                    Median :108.0   Median :127.0   Median :158.0  
##                    Mean   :111.7   Mean   :133.4   Mean   :165.3  
##                    3rd Qu.:134.0   3rd Qu.:161.0   3rd Qu.:207.0  
##                    Max.   :255.0   Max.   :255.0   Max.   :255.0
```

```
names(final_test_data)[2:4] <- c("Red", "Green", "Blue")
final_test_data <- as.data.frame(final_test_data)
```

It is also revealing when we filter by color == BLUE, B3 has the highest mean and since those particular images are images of BLUE tarps, it confirms our previous hypothesis of the names.

One last note about this, i think it makes sense for the columns to be Red, Green, and Blue because conventionally when talking about pixels we naturally order with our acronyms and language that way (e.g. rgb), so it would also make sense for the columns to be in that order.

## Predicting and Calculating Metrics

```
predicted <- list()

#LR data
LRdata <- final_test_data

LRdata$red_sq <- LRdata$Red^2

predicted[["Logistic Regression"]] <- predict(LR_1, newdata=LRdata, type="prob")[,2]

y <- final_test_data$Class
x <- data.matrix(final_test_data[,2:4])

predicted[["Ridge Regression"]] <- predict(ridge_1$finalModel, newx = x, s = 0.00401, type = "response"
predicted[["LDA"]] <- predict(lda_1$finalModel, x, type = "prob")$posterior[,2]
predicted[["QDA"]] <- predict(qda_1$finalModel, x, type = "prob")$posterior[,2]
predicted[["Random Forest"]] <- predict(rf_1$finalModel, x, type = "prob")[,2]
predicted[["KNN"]] <- predict(knn_1$finalModel, x, type = "prob")[,2]
predicted[["SVM - Linear"]] <- predict(svm_linear_1, x, type = "prob")[,2]
predicted[["SVM - Poly"]] <- predict(svm_poly_1, x, type = "prob")[,2]
predicted[["SVM - Radial"]] <- predict(svm_radial_1, x, type = "prob")[,2]
```

I would have just completely iterated over this process, but some of the models required special attention; for example, using the Red^2 feature in the logistic regression and the special parameters used in the Ridge Regression. They all just seem to have slight differences, so writing each one out.

```
Test_Results <- purrr::imap(
  predicted, ~{
    model_name <- .y
    threshold <- CV_Error_table[CV_Error_table$model == model_name,]$threshold
    tuning <- CV_Error_table[CV_Error_table$model == model_name,]$tuning

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

    data.frame(
      model_name = model_name,
      tuning = tuning,
      auroc = 1-round(auc,4),
      Threshold = threshold,
      TPR = ModelMetrics::sensitivity(as.numeric(y)-1, predicted = .x, cutoff = threshold),
      FPR = 1 - ModelMetrics::specificity(as.numeric(y)-1, predicted = .x, cutoff = threshold),
      Precision = ModelMetrics::precision(as.numeric(y)-1, predicted = .x, cutoff = threshold),
      Accuracy = sum(ifelse(.x>threshold, "Blue_Tarp", "Other") == y)/length(y)
    )
  }, y, CV_Error_table
)

Test_Results <- rbindlist(Test_Results)
```

### Final Table

```
knitr::kable(Test_Results %>% arrange(desc(TPR)))
```

| model_name | tuning | auroc | Threshold | TPR | FPR | Precision | Accuracy |
|---|---|---|---|---|---|---|---|
| Logistic Regression | N/A | 0.9997 | 0.02 | 1.0000000 | 0.0466533 | 0.1349412 | 0.9536837 |
| SVM - Linear | C=100 | 0.9991 | 0.01 | 1.0000000 | 0.1581276 | 0.0439980 | 0.8430149 |
| Ridge Regression | Lambda = 0.00401 | 0.9875 | 0.04 | 0.9866022 | 0.0623693 | 0.1032360 | 0.9379845 |
| Random Forest | ntree = 254; mtry = 1 | 0.9843 | 0.01 | 0.9730663 | 0.0511415 | 0.1216270 | 0.9490334 |
| QDA | N/A | 0.9915 | 0.01 | 0.9580110 | 0.0424095 | 0.1411851 | 0.9575936 |
| LDA | N/A | 0.9921 | 0.01 | 0.9571823 | 0.0319813 | 0.1788549 | 0.9679405 |
| KNN | k=3 | 0.9387 | 0.04 | 0.8841851 | 0.0205901 | 0.2381023 | 0.9787219 |
| SVM - Radial | C=100; Sigma =10 | 0.9217 | 0.01 | 0.7754834 | 0.0249209 | 0.1846450 | 0.9736371 |
| SVM - Poly | C=1000; Scale=0.25; degree=5 | 0.8812 | 0.01 | 0.5584945 | 0.0403685 | 0.0914736 | 0.9567334 |

Wow - we came up with some really good models!

### Conclusions

- As seen from our initial EDA, while these models are useful in predicting a blue tarp in images, it would be helpful if we had more observations in our training set of blue tarps. As currently from our training set, we only have about 3% classified as blue tarps.
- One general take away from this problem is that the separation boundaries for this classification problem are likely to be more simple than complex. For example, our highest performing models were simpler models like Logistic regression, Ridge regression, and linear SVM that more or less draw a line through the data, and models that specialize in complex, non-linear decision boundaries (e.g. radial SVM, degree - 5 polynomial SVM, and KNN) performed the lowest on our test data. These more

flexible models may also have suffered from overfitting which could be explored in future analysis, but even more likely the decision boundaries for this problem are probably just more linear than not.

- For this problem, I decided to use logistic regression as my model of choice. As we can see from the ROC curve, there was very little error from cross validation. It also performed better than the other models in the test data. If what we saw in the test data holds for new data, the Logistic Regression would identify 99% of Blue tarps correctly while incorrectly identify about 4%. This is an amazing result! It is very likely that this could be useful in saving human lives!
- An interesting finding was that while Red was a very popular color in the test data set, I found that there was a non-linear relationship between the intensity of the color Red and whether an image had a blue tarp images that were really red were significantly less likely to be blue tarps. This proved very useful in building our logistic regression model as I added a quadratic component for the predictor Red (which was significant).
- All of the predictors were correlated with each other which may affect the logistic regression model assumption of each predictor having an independent effect on Y. This model seems to perform well even in the presence of this possible violation of assumptions which is why I still decided to keep it. Initially, it seemed like perhaps Ridge regression might perform better because of the variable correlation; however, the optimal lambda was nearly zero with low coefficient estimates across the board which seems to suggest that it wasn't a useful model in this case. However, looking at the final results the ridge regression would be my second model of choice as it has a very high True Positive Rate and still a low False Positive Rate.
- One might be concerned because the average Precision across the models from the cross-validation table is much higher than the precision in the test data table. To this I would say two things: First, the test set and the training set were different in key ways - the means, medians, and standard deviations of the data were different between the two data sets. I think that in the future I might consider performing some pre-processing on my data before predicting and modeling. While the scales of the data are the same (all in pixels), the means were not the same and that could have led to some of this drop in precision. Secondly, I would say that precision isn't the most important metric in our analysis. While precision is meaningful in the context (better precision means fewer images to check), I argue that TPR is the most important metric while keeping the FPR low. In the test set, we correctly predicted 100% of the blue tarps while the False Positive Rate was at 4%. Its fine if we have to check a few more images, to me, its more important to identify all those in need. And, additionally, if only 4% of all positives are False Positives is a manageable amount of images to sort through from a feasibility standpoint.
- For further analysis, while the models here have done a good job of maximizing the TPR, the FPR could still be lowered and work could be done to reduce this number. To accomplish this, one could experiment with more features such as maybe an interaction variable or dummy variable if one or two colors reaches above a certain threshold (say red and green; if there is a lot of red and green, perhaps there will not be as much blue). Another approach would be to change the thresholds to better fit for precision instead of focusing on solely TPR and FPR.