Blue Tarp Final Project

Cavan Ingram

Introduction

In this project, we want to be able to classify and locate displaced persons living in makeshift shelters following the destruction of the earthquake in Haiti in 2010. To do this, we will first train different classification methods with the given training dataset to try and classify whether or not a given data point is a blue tarp. This dataset provided contains over 63,000 data points with four variables. Three of the variables (Red, Blue, and Green) are all forms of color data that provide numerical representations of the color. The other variable (Class) helps classify what the specific data point is (whether it is vegetation, soil, a rooftop, various non-tarp objects, or a blue tarp).

With this information, we wanted to find and utilize the best classification method to better identify the blue tarps on the training dataset. The more accurate the method used, the more lives we might be able to save or at least help. To do this, we needed to perform a couple of steps on the data before jumping right into the modeling process (such as creating a variable and performing cross-validation). These processes are talked about throughout the paper, as they would lead into the modeling process, where we used a total of nine models to try and find the best possible process to classify the blue tarps. These nine models (logistic regression, QDA, LDA, KNN, Ridge, Lasso, SVM (linear and radial), and random forests) were chosen for the analysis because they were all exceptional for classifying variables.

To compare these models on the training dataset, we chose to use various statistics to measure the differences in performance between the models (Accuracy, Precision, TPR (Sensitivity), false-negative rate (FNR), false-positive rate (FPR), and different threshold values). However, the three most important statistics in the analysis were Accuracy, FNR, and FPR (all of which go hand in hand). Accuracy was chosen because a higher accuracy would represent the best balance between the models by ensuring the FPR and FNR were both low. However, a low FNR was deemed to be important because the lower the FNR, the fewer blue tarps that would be missed due to being classified incorrectly. Similarly, a low FPR would represent more time and resources saved by incorrectly sending people/resources to blue tarps that were not there. A balance between these three statistics helped figure out what method should be used for this analysis (in both the training and testing data sets).

After these steps were completed, we also tested these models on a holdout dataset. The holdout dataset has over two million data points and even included some extra variables. However, for this specific analysis, the same variables were used in the holdout dataset as in the training dataset. We tested all nine models on this holdout dataset to see how they would perform and to help determine what method worked the best overall. All of these results and the ones listed above were compared and talked about in the conclusions to come up with an accurate recommendations and thoughts as to what method should be used and improvements that could be made.

Data Preparation (Training)

Before getting started with the exploration of the training data and our modeling, we need to set up the R environment by loading in the required packages, loading in the dataset, and creating a new variable within the existing dataset.

```
library(tidyverse)
library(dbplyr)
library(caret)
library(ROCR)
library(ggplot2)
library(GGally)
library(randomForest)

data <- read.csv("/Users/cavaningram/Downloads/HaitiPixels.csv")
data$BlueTarp <- ifelse(data$Class == 'Blue Tarp', 'yes', 'no')
```

In this process, a binary variable named BlueTarp was created based on the Class variable in our dataset. If the Class variable had a value of ‘Blue Tarp’, then the BlueTarp variable would take a value of ‘yes’ (and ‘no’ otherwise). This variable is very important in the modeling process as it helps us better determine the goal of the project to identify blue tarps.

Data Exploration (Training)

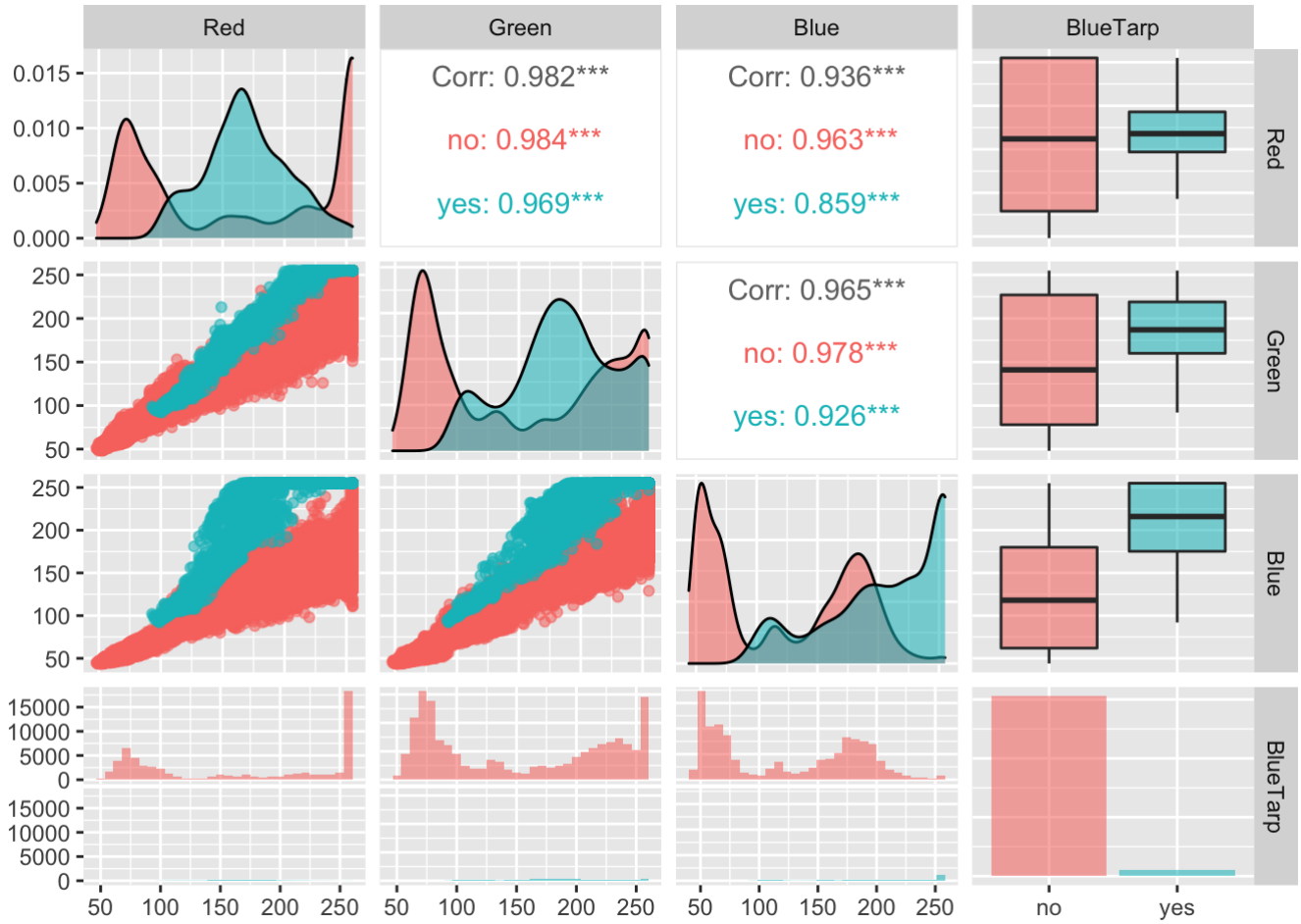
Now that we have set up the environment we need, we are able to begin to examine the training data itself to see how the variables are related to one another or if we need to scale the data for our analysis. First, we want to examine the correlation between the variables.

```
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
##    BlueTarp
##  Length:63241
##  Class :character
##  Mode  :character
##
##
##
```

In the analysis shown above, we can see that we successfully created a categorical BlueTarp variable and added it to the dataset. More importantly, we can see that the Red, Blue, and Green variables have very similar spreads. They all have the same minimum and maximum, and their means are somewhat the same (the blue value being less than the other two). Overall, this indicates that no scaling needs to be done for these variables. However, we want to analyze how each of those numerical predictor variables relates to our categorical response variable BlueTarp.

```
ggpairs(data[2:5], aes(color = BlueTarp, alpha = .5))
```



In the plot above, we can see that our three colors have a high, positive correlation with one another from the scatter plots. This indicates that there may be some multicollinearity present when we create our models, and it needs to be something we remain aware of when selecting the predictors. These scatter plots help show a more detailed representation of what is happening with the data. The correlation plots now show the difference in color between the Blue Tarp values and the other values. It helps show the separation in the two colors and can help show how trying to find the blue tarp variables will be successful.

In the box plots , we can see that the three colors do interact somewhat differently with the BlueTarp variable. We can see that the Blue and Green colors of the dataset have significantly different means between the no and yes response (indicating that these could be good predictors). The Red color, on the other hand, does not seem to have a significant difference in the mean, which might indicate it is not a variable that would be needed for the analysis (or might be the predictor to drop if multicollinearity is present within the model). These results will help us in determining the ideal model to use for the analysis.

We can also see some density plots and bar charts that can help further in our analysis. The bar chart in the bottom right of the figure above helps show how few blue tarps there are compared to the other variables. The density plots provide further explanation as to how the blue tarp variables differ from the non blue tarp variables.

Cross Validation and method used to set up model (Training)

To start our evaluation process, we first want to set up the cross-validation that we will use throughout the modeling process when training the models. This process can be shown in the code block below. We need to set up this cross-validation process to better examine our models' performance in training. In this scenario, the data

set was split into ten parts (folds) so that the cross-validation could split the folds into ten different pseudo train and test splits so that every fold would be used as a “test” set exactly once. Then the model chosen will run through each of the ten scenarios and average the performance metrics together to give us an accurate representation of the model performance. This method helps estimate how our model will perform on data it has not seen before (which will be seen later with the holdout data) rather than just randomly assigning a train and test split. Doing this 10 times rather than just a single split helps ensure that the entire dataset is taken into account for training. A single split might happen to miss some of the most important data for training and hence, negatively impact our results in testing.

```
set.seed(1)
ctrl <- trainControl(method='cv', number=10, classProbs = TRUE, savePredictions = TRUE)
```

Now that we have set up the cross validation, we need to see what predictors should be used for our BlueTarp response variable. We did this by running the code below.

```
trControl1 <- trainControl(method="none")
predictors <- train(BlueTarp ~ Blue + Red + Green, data=data, trControl=trControl1,
                    method="glmStepAIC", direction='both')
```

```
## Start:  AIC=1777.53
## .outcome ~ Blue + Red + Green
##
##         Df Deviance    AIC
## <none>       1769.5 1777.5
## - Green  1   2065.5 2071.5
## - Red    1   3007.1 3013.1
## - Blue   1   8313.1 8319.1
```

This output shows that Red, Blue, and Green should be used as predictors. Using step wise regression in caret, we can see that these three predictors create the best model and that we should not drop any of them. Using the interactions between these variables was thought about (as discussed in the conclusions) as it could produce slightly better results for certain models, but it makes the models much more difficult to interpret and understand. Not only that, but it does not improve results for all models, so for consistency in training and testing, we wanted to use the same variables for all models. Lastly, there was some fear that the models might be over fitting with all of the variables and their interactions, due to the fact that the variables are highly correlated with one another already.

We will now create a function that prints out all of the necessary information for each model to determine which model performs the best in training. The method below (model_creator) takes two parameters of m and n. M represents the method to be used for the model, and n (if needed) takes in a value of 0 or 1 to differentiate between the lasso and ridge regressions. After inputting those parameter(s), the model is created, and the results of that model are printed out. Not only that, but as the function progresses, we can also create a ROC curve and AUC value for each respective model. Lastly, using the thresholder function in caret, we print out the individual threshold for each model that has either the highest accuracy of all the thresholds or the lowest false-negative rate. This function will be essential as we proceed through the rest of the project.

```
model_creator <- function(m, n) {
  #model creation
  set.seed(1)
  if(missing(n)) {
    if(m == "knn"){
     a <- train(BlueTarp ~ Blue + Red + Green, data = data, method = "knn", tuneGrid = e
xpand.grid(k = 1:10), trControl = ctrl)
       print(a$results)
       saveRDS(a, file = paste(m, "_mod", sep=""), compress = TRUE)

    }

    else if(m=="svmLinear") {
      a <- train(BlueTarp ~ Red + Blue +Green, data=data,
         method='svmLinear',
         trControl=ctrl)
      print(a$results)
      saveRDS(a, file = paste(m, "_mod", sep=""), compress = TRUE)
    }

    else if (m=="svmRadial") {
      a <- train(BlueTarp ~ Red + Blue + Green, data=data,
                 method=m,
                 trControl=ctrl)
      print(a$results)
      saveRDS(a, file = paste(m, "_mod", sep=""), compress = TRUE)
    }

    else if(m== "rf") {
        a <- train(BlueTarp ~ Red + Blue + Green, data=data,
                   method=m,
                   ntree=500,
                   tuneGrid=expand.grid(mtry=c(1,2,3)),
                   trControl=ctrl)
       print(a$results)
       saveRDS(a, file = paste(m, "_mod", sep=""), compress = TRUE)
    }

      else {
          a <- train(BlueTarp ~  Red + Blue + Green, data = data, method = m,
                     trControl = ctrl)
          saveRDS(a, file = paste(m, "_mod", sep=""), compress = TRUE)
          print(a$results)

      }
  }

  else if(n==1) {
              a <- train(BlueTarp~ Red + Blue + Green, data=data,
                         preProcess = c("scale"), method=m,
                         tuneGrid=data.frame(alpha=rep(n,18),
                                             lambda=10^seq(.4,-3,by=-.2)),
```

```
                        trControl=ctrl)
            print(a$results)
            print(plot(a))
            saveRDS(a, file = "lasso_mod", compress = TRUE)


  }
  else {
            a <- train(BlueTarp~ Red + Blue + Green, data=data,
                        preProcess = c("scale"), method=m,
                        tuneGrid=data.frame(alpha=rep(n,18),
                                            lambda=10^seq(.4,-3,by=-.2)),
                        trControl=ctrl)
            print(a$results)
            saveRDS(a, file = "ridge_mod", compress = TRUE)
            print(plot(a))
    }

  #ROC Curve
  pred <- predict(a, data, type='prob')
  predob <- prediction(pred$yes, data$BlueTarp)
  model.roc <- performance(predob, measure='tpr', x.measure='fpr')
  plot(model.roc, colorize=T, print.cutoffs.at=c(0, 0.1, 0.9, 1.0))
  lines(x=c(0,1), y=c(0,1), col='grey')

  #AUC calculation
  auc_ROCR <- performance(predob, measure = "auc")
  auc_ROCR <- auc_ROCR@y.values[[1]]
  AUC <<- replace(AUC, i, auc_ROCR)
  print("AUC")
  print(auc_ROCR)

  #Threshold selection
  threshold.stats <- thresholder(a,
                        threshold = seq(0.05, 0.95, by = 0.05),
                        statistics = "all")

  threshold.stats$FNR <- 1 - threshold.stats$Sensitivity
  threshold.stats$FPR <- 1 - threshold.stats$Specificity

thresh <- (threshold.stats %>% slice_max(Accuracy) %>%
  select("prob_threshold", "Accuracy", "Precision", "Sensitivity", "FNR", "FPR"))

Threshold <<- replace(Threshold, i, thresh$prob_threshold)
Accuracy <<- replace(Accuracy, i, thresh$Accuracy)
Precision <<- replace(Precision, i, thresh$Precision)
TPR <<- replace(TPR, i, thresh$Sensitivity)
FNR <<- replace(FNR, i, thresh$FNR)
FPR <<- replace(FPR, i, thresh$FPR)

print(thresh)
```

```
print(threshold.stats %>% slice_min(FNR) %>%
  select("prob_threshold", "Accuracy", "Precision", "Sensitivity", "FNR", "FPR"))

Method = c("GLM","QDA","LDA","KNN","Ridge", "Lasso", "SVMlin", "SVMrad", "RF")

K = c("","","","5","", "", "", "", "")
Lamda = c("","","","",".001", ".001", "", "", "")
Parameters = c("","","","","", "", "", "", "3")

performance_table <<- data.frame(Method, K, Lamda, AUC = round(AUC, 4), Threshold = roun
d(Threshold, 4), Accuracy = round(Accuracy, 4), Precision = round(Precision, 4), TPR = r
ound(TPR, 4), FNR = round(FNR, 4), FPR = round(FPR, 4))
}
```

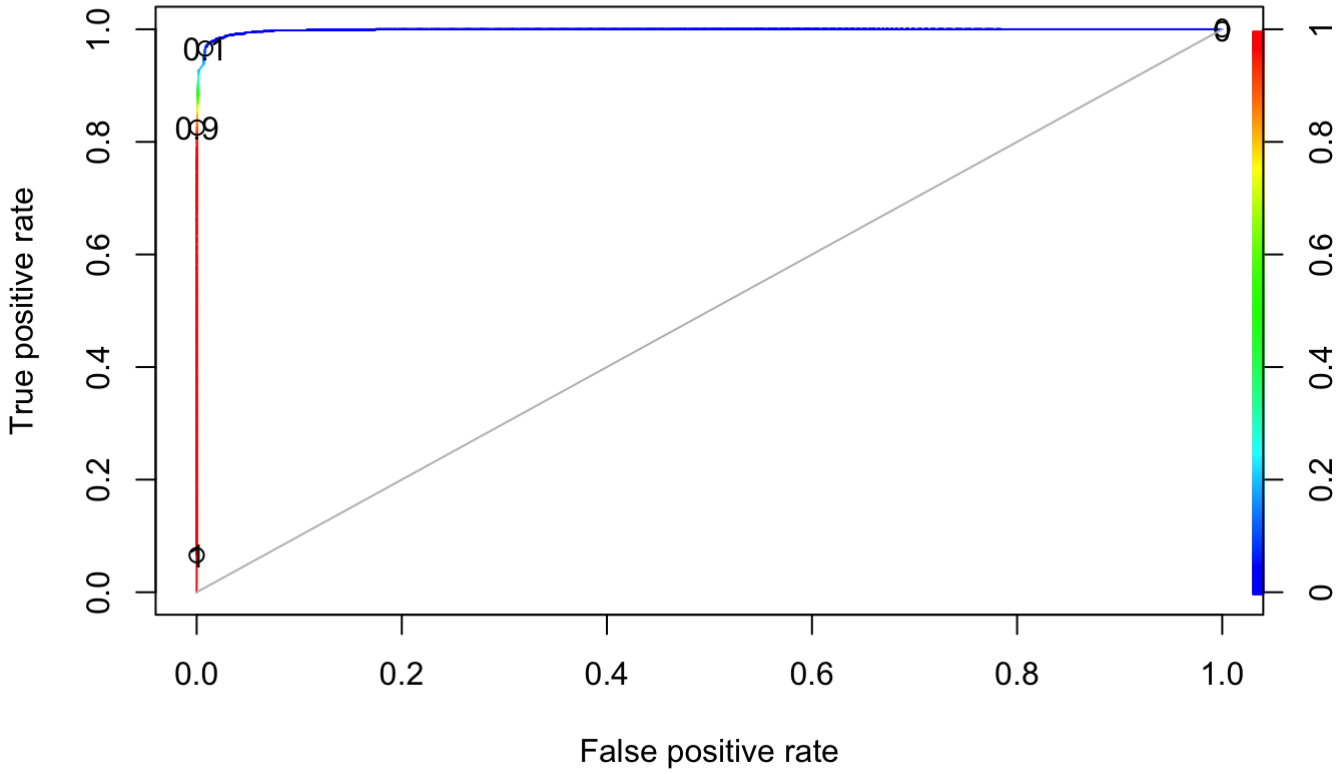
Models (Training)

Logistic Regression

The first model we chose to analyze in training was the logistic regression model. In this model, we analyze the relationship between Red, Green, and Blue to correctly classify our BlueTarp variable. The BlueTarp variable is binary, which is typically what logistic regression is used for (predict the outcome of a binary variable). Because this is a binary classification problem, the logistic regression model might perform better than some of the other methods that are better for response variables that have more than two classes. We were able to use the model_creator function seen above to analyze this model and its accuracy.

```
model_creator("glm")
```

```
##   parameter  Accuracy     Kappa  AccuracySD    KappaSD
## 1      none 0.9952404 0.9198727 0.001047351 0.01826815
```




```
## [1] "AUC"
## [1] 0.9985069
##   prob_threshold  Accuracy Precision Sensitivity         FNR        FPR
## 1           0.75 0.9957464 0.9973241   0.9982848 0.001715166 0.08110277
##   prob_threshold  Accuracy Precision Sensitivity          FNR       FPR
## 1           0.05 0.9932481 0.9933621    0.999706 0.0002940296 0.2022631
```

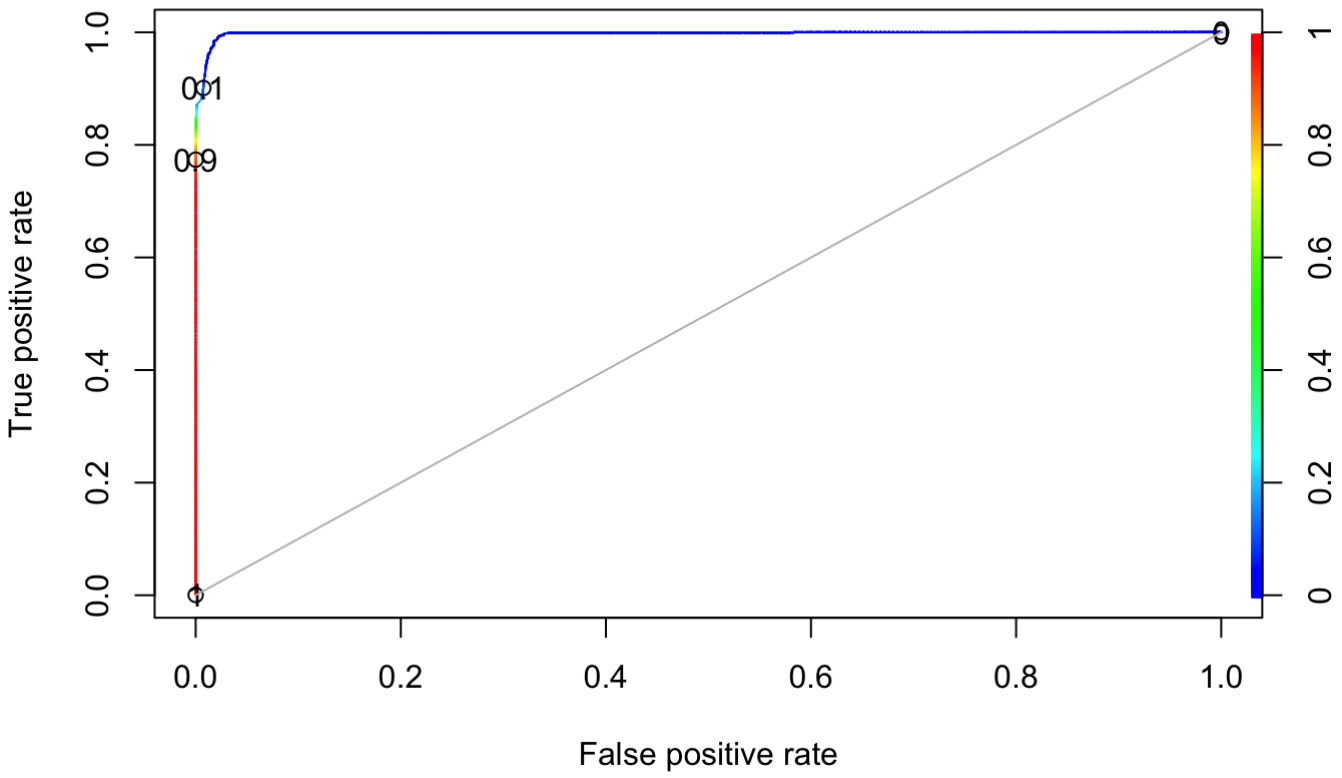
From the outputs above, we can see that the logistic regression model performed really well and was very accurate on the training data. In the cross-validation, the accuracy score averaged to be .9952. We can also see from the ROC curve and the AUC that the model classified everything very well with an AUC of .9985. This high value represents the ability of this model to correctly distinguish between the two different classes in our created BlueTarp variable. Lastly, we want to evaluate the different thresholds for our model. Throughout this analysis and the analysis for the other models, we take advantage of the caret library and have caret obtain for us two separate threshold values. One for the lowest possible FNR, and the other for the highest overall accuracy. For this model, we can see that the threshold that obtained the highest accuracy was at a threshold of .75 with a corresponding accuracy score of .9957. The threshold with the lowest FNR was at .05, with a corresponding FNR of .0003. When deciding which threshold to use for this particular model, we want to look at all the metrics and compare which one does the best overall job. For this specific scenario, we will choose a threshold value of .75. The reason for this is not only because it has the highest overall accuracy score, but it also has a low FNR and a significantly lower FPR. This threshold might miss a few blue tents that it incorrectly designated as not blue tents compared to the other threshold, but it will save a significant amount of time and resources by not going to blue tents that are not actually there. Since the FNR is still so low, this threshold is ideal because it will save lots of time and resources.

QDA

The second process we chose to analyze was the quadratic discriminant analysis. This model is very similar to the linear discriminant analysis which is talked about in detail in the next section. The difference in this analysis compared to the LDA is that it does not assume he classes have identical covariance matrices and is hence more flexible (but could have high variance). We were able to use the model_creator function seen above to analyze this model and its accuracy.

```
model_creator("qda")
```

```
##   parameter  Accuracy     Kappa  AccuracySD    KappaSD
## 1      none 0.9945605 0.9049272 0.001085673 0.02054495
```




```
## [1] "AUC"
## [1] 0.9982175
##   prob_threshold  Accuracy Precision Sensitivity          FNR       FPR
## 1            0.6 0.9947503 0.9950089   0.9995916 0.0004083686 0.1518144
##   prob_threshold  Accuracy Precision Sensitivity FNR       FPR
## 1           0.05 0.9918408 0.9916427           1   0 0.2551724
```

This model also performs very well, although not quite as good as the linear regression. The cross-validation shows that there is still a high accuracy score of .9946. We can also see from the ROC curve and AUC value (.9982) that the model performed exceptionally well but slightly worse than the logistic regression. For the thresholds, we can see that .6 was chosen for the highest accuracy, and .05 was chosen for the lowest FNR (just

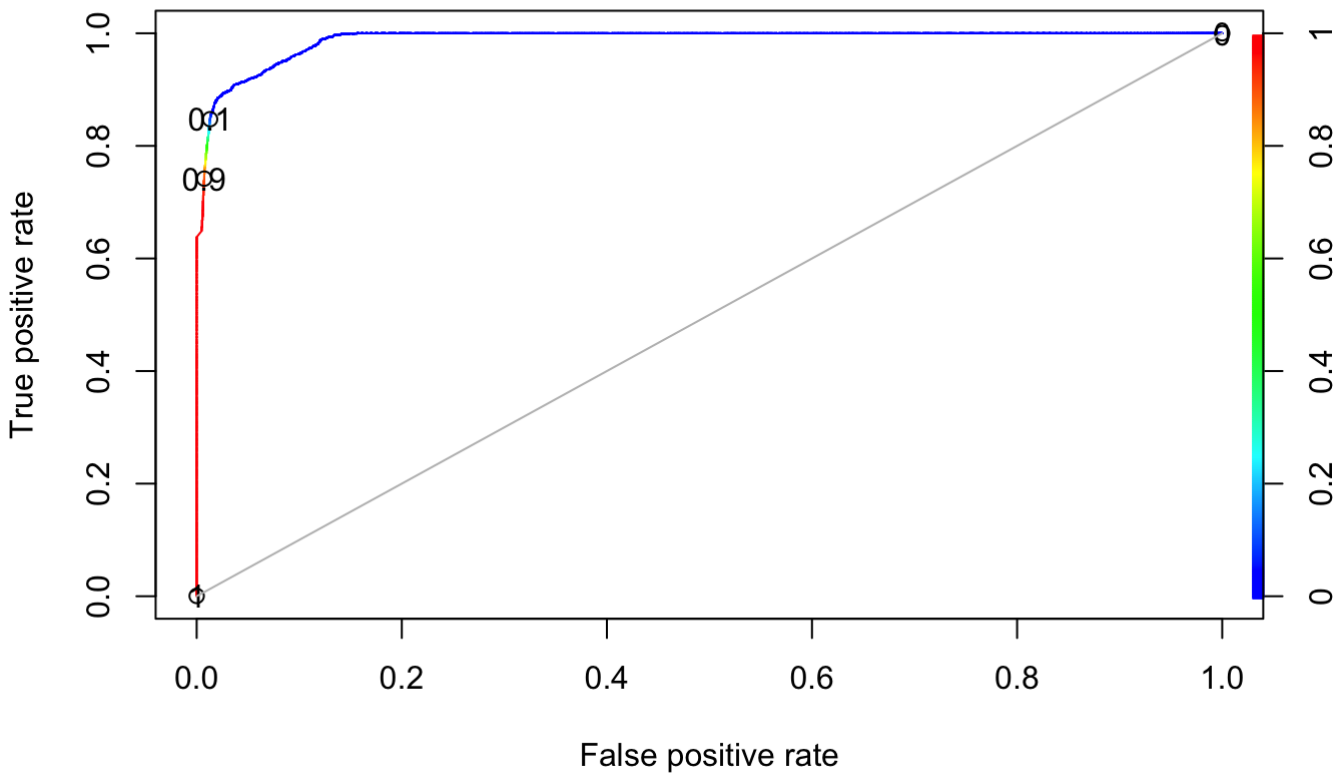
as it was in the logistic regression model). When comparing these two thresholds, we again look at all of the statistics listed as a whole. Like the logistic regression model, we can again see a similar pattern where .6 seems to be a better threshold because it has a higher overall accuracy score and also still has a low FNR and much lowers FPR, which means it is the ideal threshold. However, as stated earlier, this model did not perform as well as the logistic regression model. The only reason to consider this model is that it does have a lower FNR at the selected threshold. Still, it is lacking in the other categories to put it above using the logistic regression model at this point.

LDA

The third process we chose to analyze was the linear discriminant analysis. In this model, we assume that the data follows a Gaussian distribution and that each of the classes has identical covariance matrices, but it can still perform well if these assumptions are not met. This method is used for classification and dimensional reduction. It tries to classify by dividing the data into regions and then trying to assign data to those regions using the maximum likelihood (using the Bayesian rule). This method could, though, have a high bias. We were able to use the model_creator function seen above to analyze this model and its accuracy.

```
model_creator("lda")
```

```
##   parameter  Accuracy     Kappa  AccuracySD    KappaSD
## 1      none 0.9839187 0.7530147 0.001482914 0.02121463
```



```
## [1] "AUC"
## [1] 0.9888768
##   prob_threshold  Accuracy Precision Sensitivity         FNR       FPR
## 1           0.15 0.9848358 0.9919998    0.992339 0.007660994 0.2423109
##   prob_threshold  Accuracy Precision Sensitivity         FNR       FPR
## 1           0.05 0.9846145 0.9904919   0.9936458 0.006354215 0.2887919
```

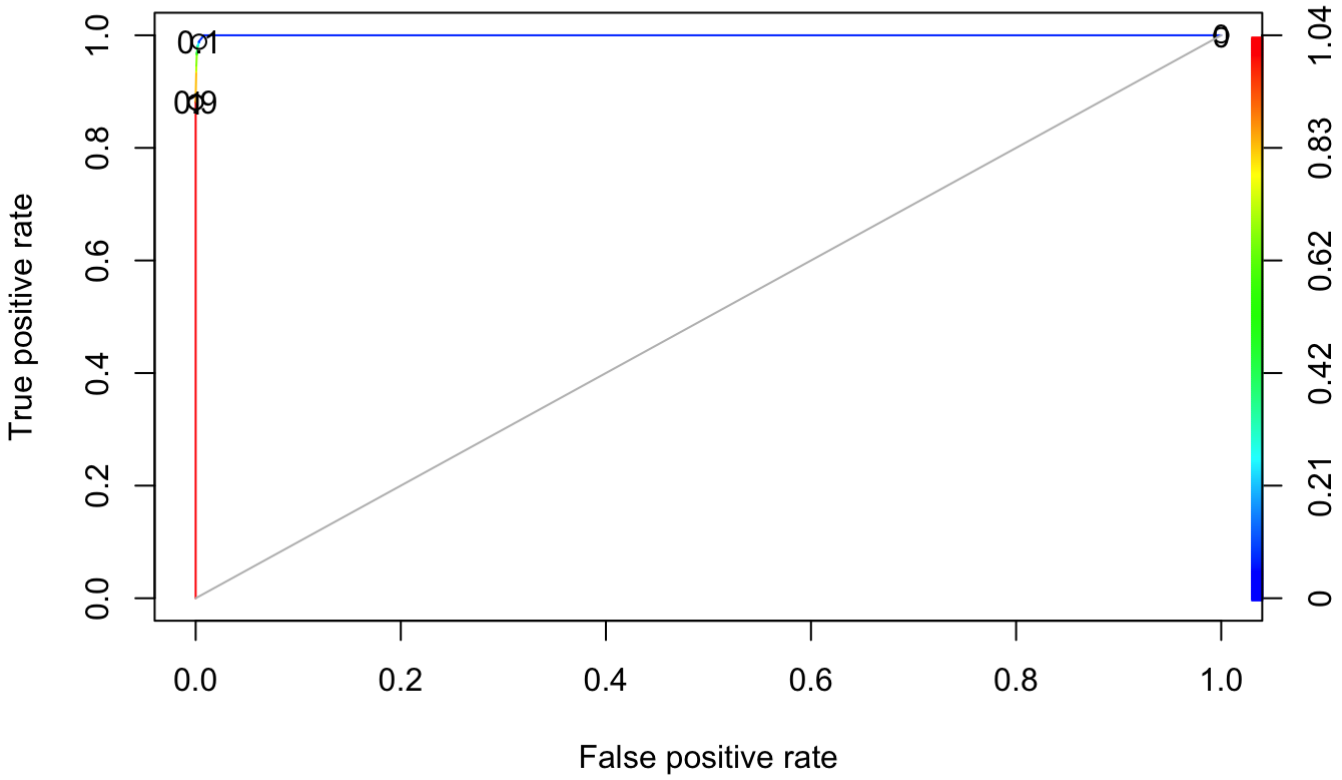
At first glance, this model seems to be the worst out of the ones we have seen thus far. Its overall accuracy scores are lower for the cross-validation (.9839), it has a worse ROC curve, and it has a lower AUC value (.9889). However, these scores are still excellent and show that the model is functioning at a high level. We again look to see what the desired threshold for this model is and it chose .05 again for the lowest FNR, but it chose .15 for the highest accuracy. However, these thresholds still have lower overall accuracy scores and a much higher FPR, which shows that this model should not be the choice to use for the analysis (even though the FNR is slightly lower). Since we have only a few parameters and LDA is less flexible, the QDA model was able to perform and fit the data better.

KNN

The fourth model we chose to analyze was the k nearest neighbors model. In this model, we look to classify a data point as a blue tarp or not by the surrounding points that are near to it. This number of points can vary based on how many “neighbors” work best. We were able to use the model_creator function seen above to analyze this model and its accuracy. In this process, we created a k nearest neighbor model for all k’s 1 through 10 to get the ideal number of neighbors. The process showed (as seen in the output below) that the number of neighbors with the highest accuracy was 5.

```
model_creator("knn")
```

```
##     k  Accuracy     Kappa   AccuracySD     KappaSD
## 1   1 0.9968059 0.9480064 0.0006402757 0.010806539
## 2   2 0.9968691 0.9492846 0.0007022723 0.011471150
## 3   3 0.9972486 0.9554020 0.0006063956 0.010298947
## 4   4 0.9972170 0.9549109 0.0007644340 0.012769293
## 5   5 0.9972961 0.9563522 0.0007204716 0.011964718
## 6   6 0.9972012 0.9549487 0.0005775919 0.009667717
## 7   7 0.9971063 0.9534254 0.0006456574 0.010810114
## 8   8 0.9971854 0.9547157 0.0006531471 0.010637007
## 9   9 0.9972328 0.9554017 0.0006550601 0.010868946
## 10 10 0.9971379 0.9539269 0.0006889515 0.011369701
```




```
## [1] "AUC"
## [1] 0.9998582
##   prob_threshold  Accuracy Precision Sensitivity         FNR        FPR
## 1           0.50 0.9972961 0.9987585   0.9984482 0.001551804 0.03757743
## 2           0.55 0.9972961 0.9987585   0.9984482 0.001551804 0.03757743
##   prob_threshold  Accuracy Precision Sensitivity          FNR       FPR
## 1           0.05 0.9955251 0.9957862    0.999608 0.0003920341 0.1280983
```

In the output above, we can see that this model performed very well. It had the highest accuracy score so far of .9972, and had a great ROC curve with a nearly perfect AUC value of .9999, which was also the best so far. When looking at the selected thresholds, .05 had the lowest FNR score, but the highest overall accuracy scores happened at .5 and .55 because they both had the highest scores. Not only that, but all the statistics for those two thresholds were identical. When deciding which threshold is better for the model, we again saw that the threshold with the highest accuracy was better overall due to the FPR being so much lower in comparison to the .05 threshold. In this scenario, .5 or .55 are the ideal threshold values, but for simplicity, we will just call .5 the ideal threshold from here on out for this particular model. This model is currently the ideal choice for the analysis because of how high the accuracy is to go along with relatively low FNR and FPR rates.

Penalized Logistic Regression

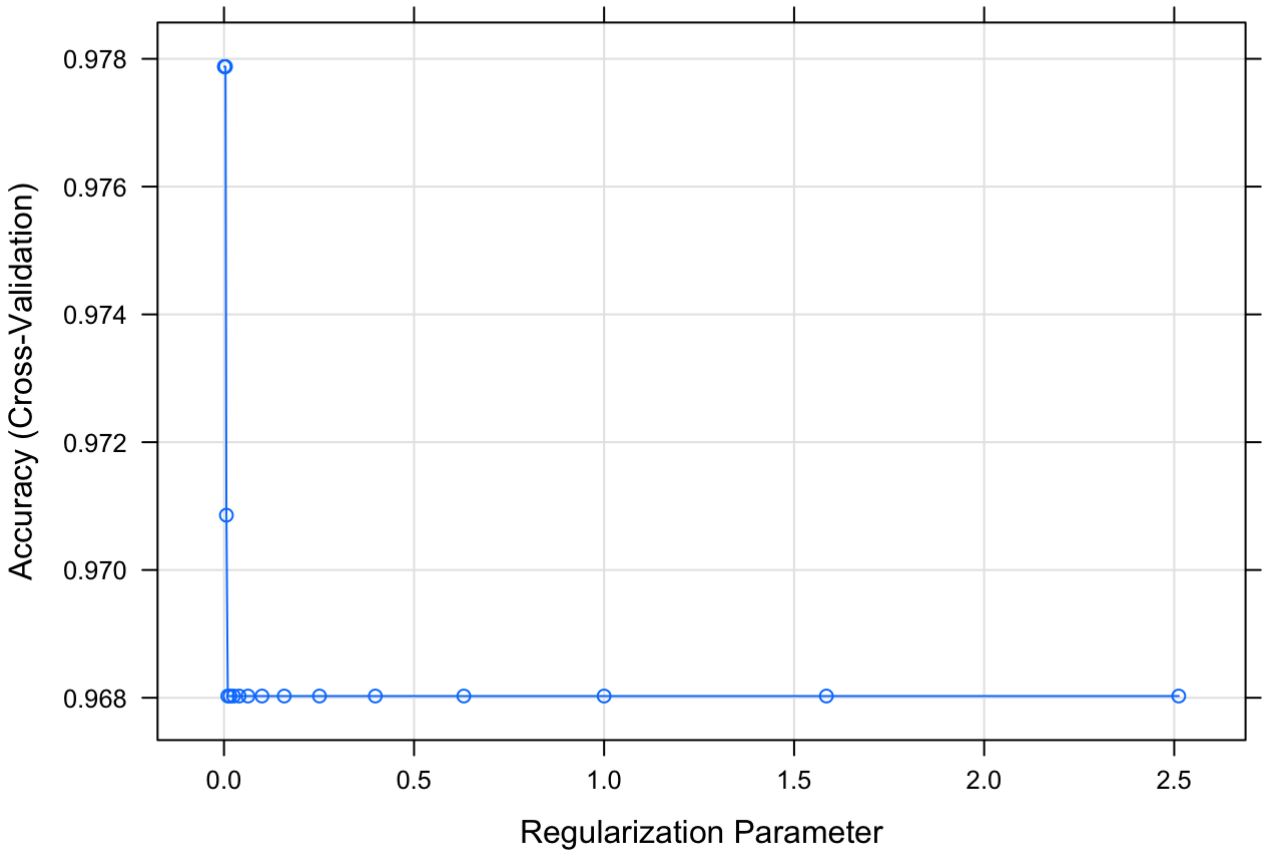
Ridge Logistic Regression

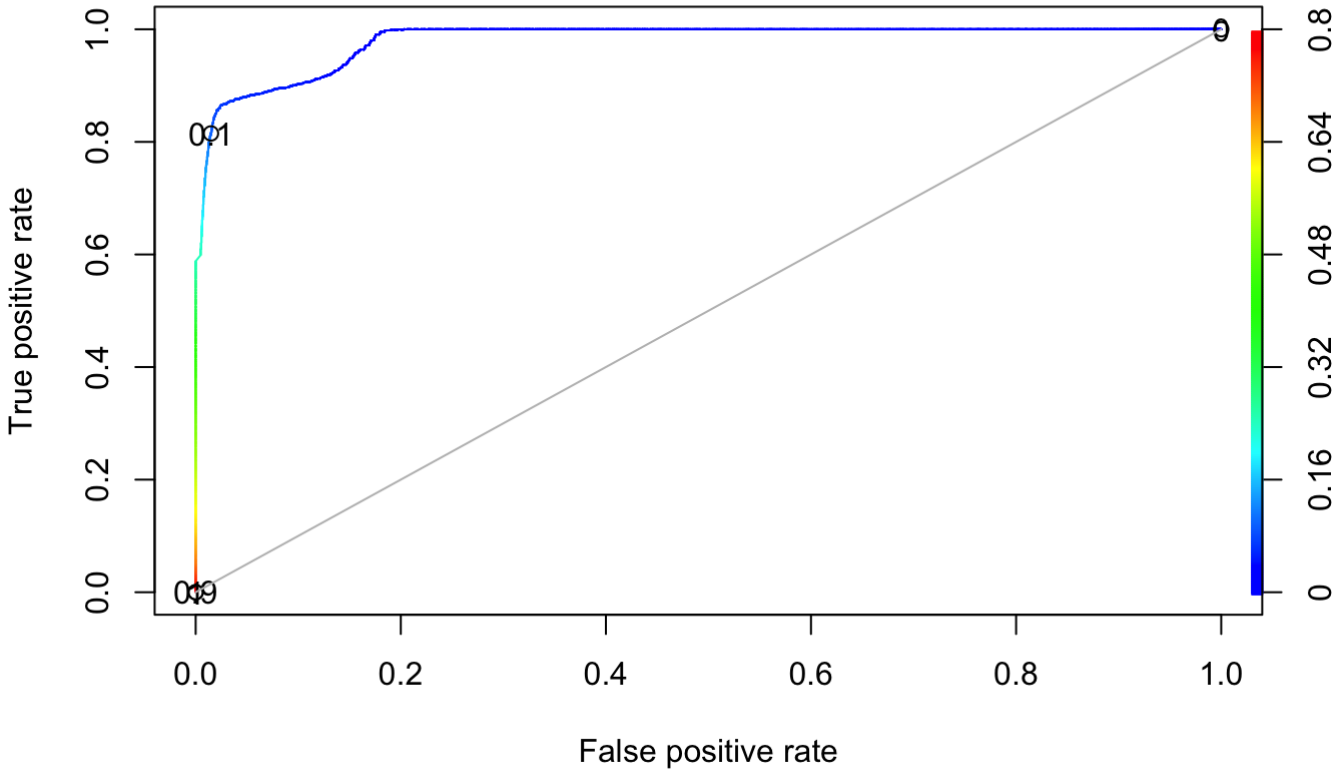
The fifth model we chose to analyze was the ridge regression model. This model is one example of a penalized regression model. This model is similar to the lasso regression below and also least squares. The ridge regression penalty (taking square of the coefficients) will shrink all the coefficients to zero, but none will actually be zero and

be eliminated (like with lasso). We were able to use the model_creator function seen above to analyze this model and its accuracy.

```
model_creator("glmnet", 0)
```

```
##    alpha      lambda  Accuracy     Kappa   AccuracySD    KappaSD
## 1      0 0.001000000 0.9778783 0.4624899 8.698447e-04 0.03052600
## 2      0 0.001584893 0.9778783 0.4624899 8.698447e-04 0.03052600
## 3      0 0.002511886 0.9778783 0.4624899 8.698447e-04 0.03052600
## 4      0 0.003981072 0.9778783 0.4624899 8.698447e-04 0.03052600
## 5      0 0.006309573 0.9708575 0.1581568 3.246825e-04 0.01618341
## 6      0 0.010000000 0.9680271 0.0000000 6.428573e-05 0.00000000
## 7      0 0.015848932 0.9680271 0.0000000 6.428573e-05 0.00000000
## 8      0 0.025118864 0.9680271 0.0000000 6.428573e-05 0.00000000
## 9      0 0.039810717 0.9680271 0.0000000 6.428573e-05 0.00000000
## 10     0 0.063095734 0.9680271 0.0000000 6.428573e-05 0.00000000
## 11     0 0.100000000 0.9680271 0.0000000 6.428573e-05 0.00000000
## 12     0 0.158489319 0.9680271 0.0000000 6.428573e-05 0.00000000
## 13     0 0.251188643 0.9680271 0.0000000 6.428573e-05 0.00000000
## 14     0 0.398107171 0.9680271 0.0000000 6.428573e-05 0.00000000
## 15     0 0.630957344 0.9680271 0.0000000 6.428573e-05 0.00000000
## 16     0 1.000000000 0.9680271 0.0000000 6.428573e-05 0.00000000
## 17     0 1.584893192 0.9680271 0.0000000 6.428573e-05 0.00000000
## 18     0 2.511886432 0.9680271 0.0000000 6.428573e-05 0.00000000
```


Accuracy (Cross-Validation)
0.978
0.976
0.974
0.972
0.970
0.968
0.0
0.5
1.0
1.5
2.0
2.5
Regularization Parameter


True positive rate
0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0
False positive rate
0.8
0.64
0.48
0.32
0.16
0

```
## [1] "AUC"
## [1] 0.9801936
##   prob_threshold  Accuracy Precision Sensitivity          FNR       FPR
## 1           0.75 0.9866384 0.9864016   0.9999837 1.633453e-05 0.4173828
##    prob_threshold  Accuracy Precision Sensitivity FNR       FPR
## 1            0.05 0.9680271 0.9680271           1   0 1.0000000
## 2            0.10 0.9680271 0.9680271           1   0 1.0000000
## 3            0.15 0.9680271 0.9680271           1   0 1.0000000
## 4            0.20 0.9680271 0.9680271           1   0 1.0000000
## 5            0.25 0.9686121 0.9685938           1   0 0.9817002
## 6            0.30 0.9694818 0.9694375           1   0 0.9544993
## 7            0.35 0.9706994 0.9706210           1   0 0.9164122
## 8            0.40 0.9726760 0.9725484           1   0 0.8545993
## 9            0.45 0.9750004 0.9748250           1   0 0.7819002
## 10           0.50 0.9778783 0.9776589           1   0 0.6918646
## 11           0.55 0.9803451 0.9801004           1   0 0.6147100
## 12           0.60 0.9821477 0.9818927           1   0 0.5583378
## 13           0.65 0.9833336 0.9830753           1   0 0.5212579
## 14           0.70 0.9848990 0.9846408           1   0 0.4722772
```

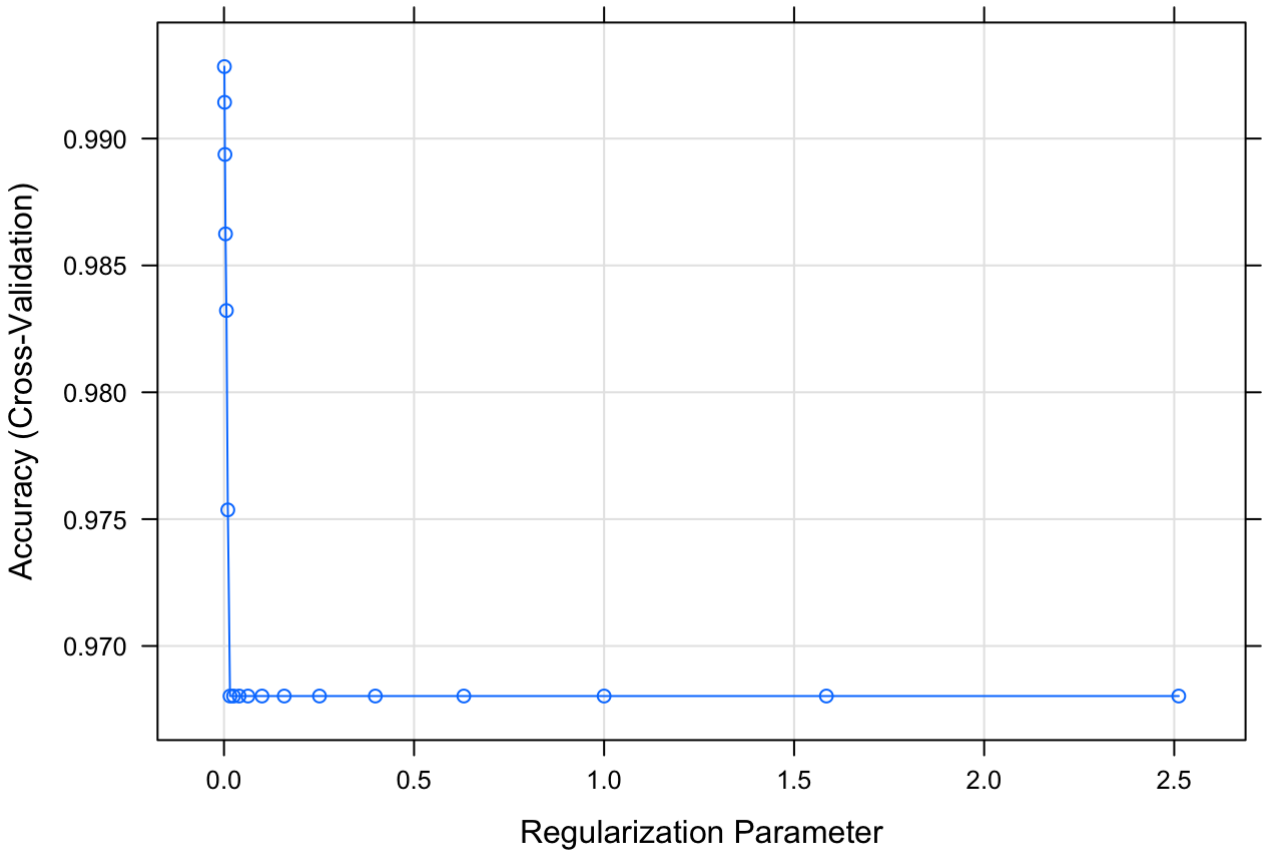
From the output shown above, we can see that we have a decent-looking ROC curve with a good AUC value of .9801. These are very good, but the regression does not seem to be performing as well as some of the other models. Looking at the threshold values where we have the lowest FNR values, we can see that we have a zero FNR for 14 different thresholds (.05 to .7). For the threshold value with the highest accuracy, we can see that a value of .75 was chosen with a corresponding accuracy of .9866. This threshold has a nearly nonexistent FNR, but it has a higher FPR than some other methods. However, we will choose .75 for this threshold because it has a very low FNR and high accuracy, but it does not perform as well overall as some of the other models. We can also see from the output that the optimal tuning parameter (lambda) for this method is .001, as it produces the highest overall accuracy.

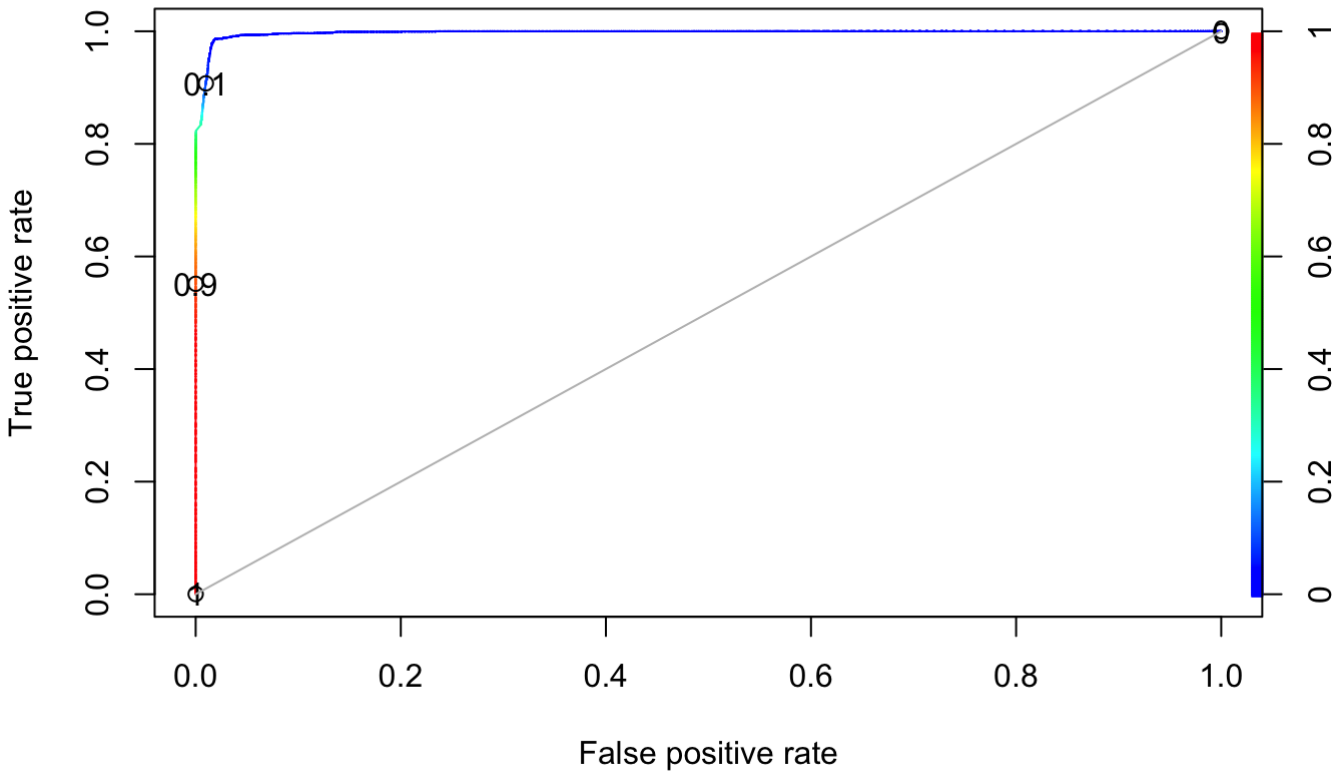
Lasso Logistic Regression

The sixth model we chose to analyze was the lasso regression model. This is another form of a penalized regression model. However, in the lasso, the coefficients are shrunk towards zero, meaning the less important features may be eliminated. This method helps reduce the complexity of the model. We were able to use the model_creator function seen above to analyze this model and its accuracy.

```
model_creator("glmnet", 1)
```

```
##    alpha      lambda  Accuracy     Kappa   AccuracySD    KappaSD
## 1      1 0.001000000 0.9928370 0.8700755 1.013774e-03 0.02040513
## 2      1 0.001584893 0.9914296 0.8406116 1.013015e-03 0.02134669
## 3      1 0.002511886 0.9893740 0.7949949 1.187176e-03 0.02663037
## 4      1 0.003981072 0.9862431 0.7188242 1.157343e-03 0.02926055
## 5      1 0.006309573 0.9832229 0.6364826 8.384911e-04 0.02393065
## 6      1 0.010000000 0.9753641 0.3655069 5.096418e-04 0.02038515
## 7      1 0.015848932 0.9680271 0.0000000 6.428573e-05 0.00000000
## 8      1 0.025118864 0.9680271 0.0000000 6.428573e-05 0.00000000
## 9      1 0.039810717 0.9680271 0.0000000 6.428573e-05 0.00000000
## 10     1 0.063095734 0.9680271 0.0000000 6.428573e-05 0.00000000
## 11     1 0.100000000 0.9680271 0.0000000 6.428573e-05 0.00000000
## 12     1 0.158489319 0.9680271 0.0000000 6.428573e-05 0.00000000
## 13     1 0.251188643 0.9680271 0.0000000 6.428573e-05 0.00000000
## 14     1 0.398107171 0.9680271 0.0000000 6.428573e-05 0.00000000
## 15     1 0.630957344 0.9680271 0.0000000 6.428573e-05 0.00000000
## 16     1 1.000000000 0.9680271 0.0000000 6.428573e-05 0.00000000
## 17     1 1.584893192 0.9680271 0.0000000 6.428573e-05 0.00000000
## 18     1 2.511886432 0.9680271 0.0000000 6.428573e-05 0.00000000
```


Accuracy (Cross-Validation)
0.990
0.985
0.980
0.975
0.970
0.0
0.5
1.0
1.5
2.0
2.5
Regularization Parameter


True positive rate
0.0
0.2
0.4
0.6
0.8
1.0
0.1
0.9
0
False positive rate

```
## [1] "AUC"
## [1] 0.9973879
##   prob_threshold  Accuracy Precision Sensitivity          FNR       FPR
## 1           0.65 0.9938964  0.994072    0.999657 0.0003430305 0.1805053
##   prob_threshold  Accuracy Precision Sensitivity FNR       FPR
## 1           0.05 0.9835550 0.9832963           1   0 0.5143125
## 2           0.10 0.9856422 0.9853852           1   0 0.4490392
## 3           0.15 0.9873500 0.9871018           1   0 0.3956202
## 4           0.20 0.9885043 0.9882653           1   0 0.3595132
## 5           0.25 0.9895638 0.9893359           1   0 0.3263644
## 6           0.30 0.9902753 0.9900555           1   0 0.3041287
## 7           0.35 0.9911766 0.9909687           1   0 0.2759401
```

From the beginning, we can see that this model appears to perform better than the ridge regression model. We seem to have a better ROC curve and a high AUC value of .9985. Going further into the analysis, we can see that the threshold with the lowest FNR was anything from .05 to .35, with a corresponding FNR of 0. However, the threshold with the highest accuracy was .65, with a corresponding accuracy of .9939. These performance metrics seem to be on par with the best-performing methods (KNN and logistic regression). It also has a low FNR and a low FPR. We can also see from the outputs that the optimal tuning parameter is at .001 (like with ridge). Overall, this method performed very well but was still slightly worse than the k-nearest neighbors model (especially regarding the FPR rate).

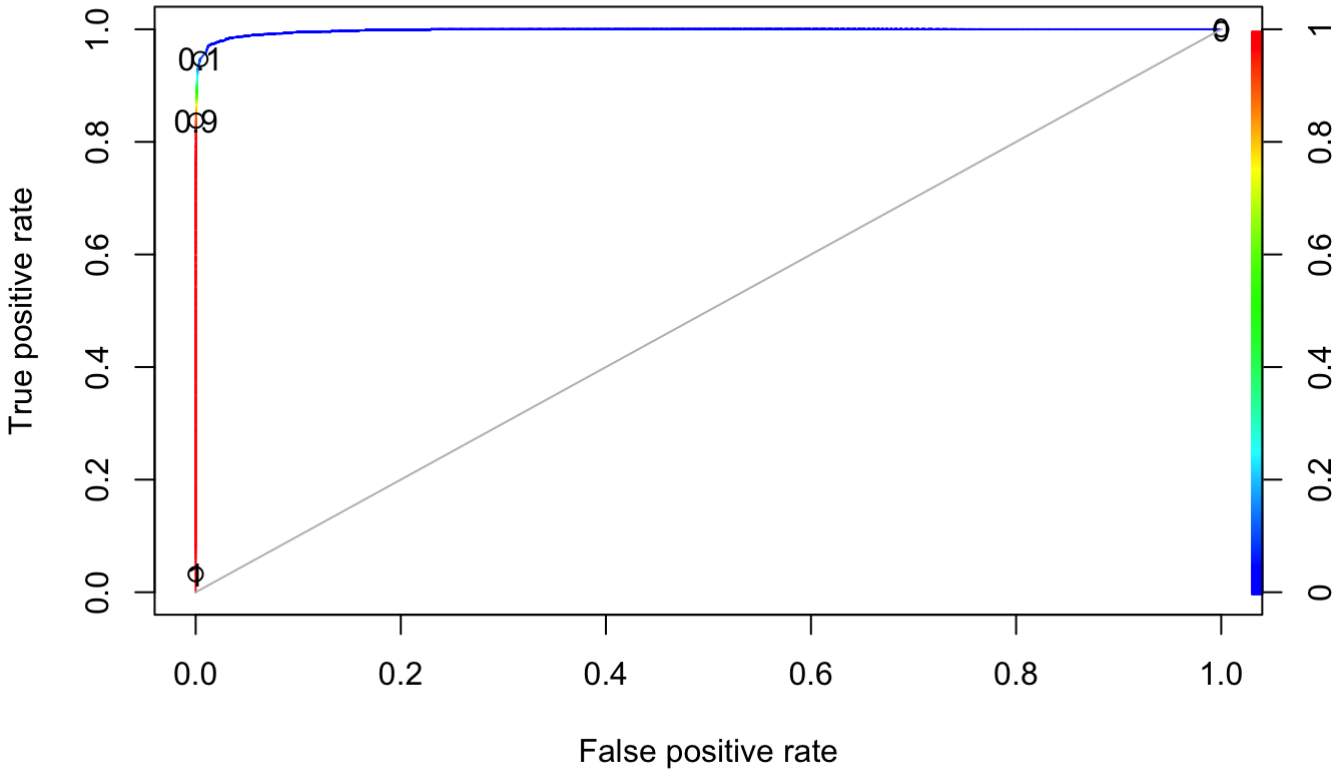
SVM

SVM (Linear)

We then wanted to analyze Support Vector Machines (SVM) for our seventh and eighth classification models. For the seventh model we analyzed the SVM with a linear kernel and for the eighth model we analyzed SVM with a radial kernel. The linear kernel is used when the data can be accurately classified using a single line to separate the data. The radial kernel is used for data that is not linear and works more similarly to k-nearest neighbors in its classification process. SVM models can be good at binary classification and basically creates a line/hyper plane to separate the data into classes. We can now evaluate the performance of the two SVM models.

```
model_creator("svmLinear")
```

```
##   C  Accuracy     Kappa   AccuracySD    KappaSD
## 1 1 0.9953986 0.9225159 0.0009172839 0.01626153
```



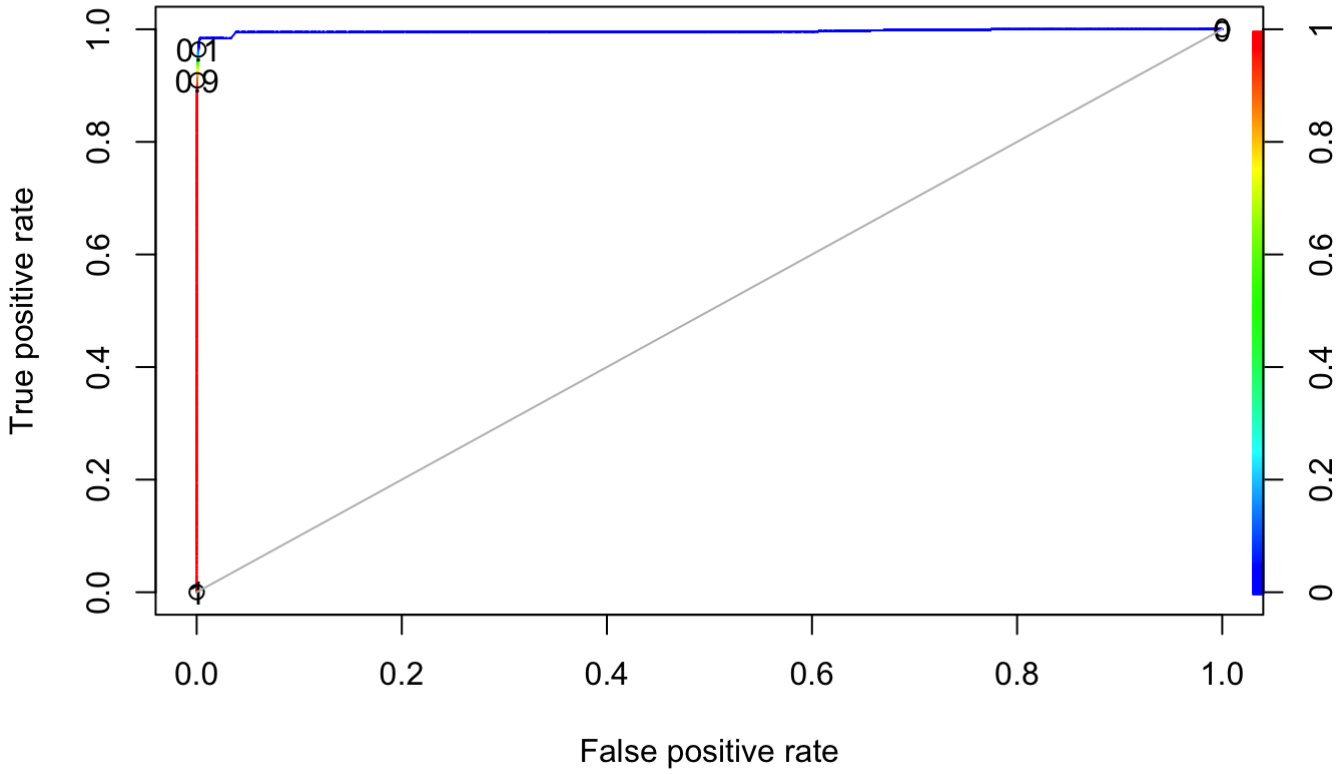

```
## [1] "AUC"
## [1] 0.9978488
##   prob_threshold  Accuracy Precision Sensitivity         FNR        FPR
## 1           0.75 0.9957939  0.997243   0.9984155 0.001584482 0.08357314
##   prob_threshold  Accuracy Precision Sensitivity          FNR      FPR
## 1           0.05 0.9937541 0.9939265    0.999657 0.0003430358 0.184951
```

In the output shown above, we can see the results of the SVM model with a linear kernel. First and foremost, we can see that we obtain an accuracy of about .9953 for the model. We also obtain a good-looking ROC curve with a high AUC of .9978. These statistics are very good, but still not quite as good as what we obtained for knn. Lastly, we begin to look at the parameter values obtained for this model. At a threshold of .05, we obtain the lowest possible FNR and an accuracy of .9938. With a threshold of .75, we obtain the highest possible accuracy of .9958. This produces the best overall model due to the fact that the FNR and FPR scores are still low. However, this model is still not quite as good overall compared to knn.

SVM (Radial)

```
model_creator("svmRadial")
```

```
##      sigma    C  Accuracy     Kappa   AccuracySD    KappaSD
## 1 8.297721 0.25 0.9966319 0.9446450 0.0007033949 0.01220305
## 2 8.297721 0.50 0.9968691 0.9485279 0.0008293945 0.01413992
## 3 8.297721 1.00 0.9969956 0.9506588 0.0007782316 0.01324788
```




```
## [1] "AUC"
## [1] 0.9964331
##   prob_threshold  Accuracy Precision Sensitivity         FNR        FPR
## 1           0.95 0.9970905 0.9989864   0.9980072 0.001992847 0.03066137
##   prob_threshold  Accuracy Precision Sensitivity          FNR       FPR
## 1           0.05 0.9958887 0.9962558     0.99951 0.0004900493 0.1137468
```

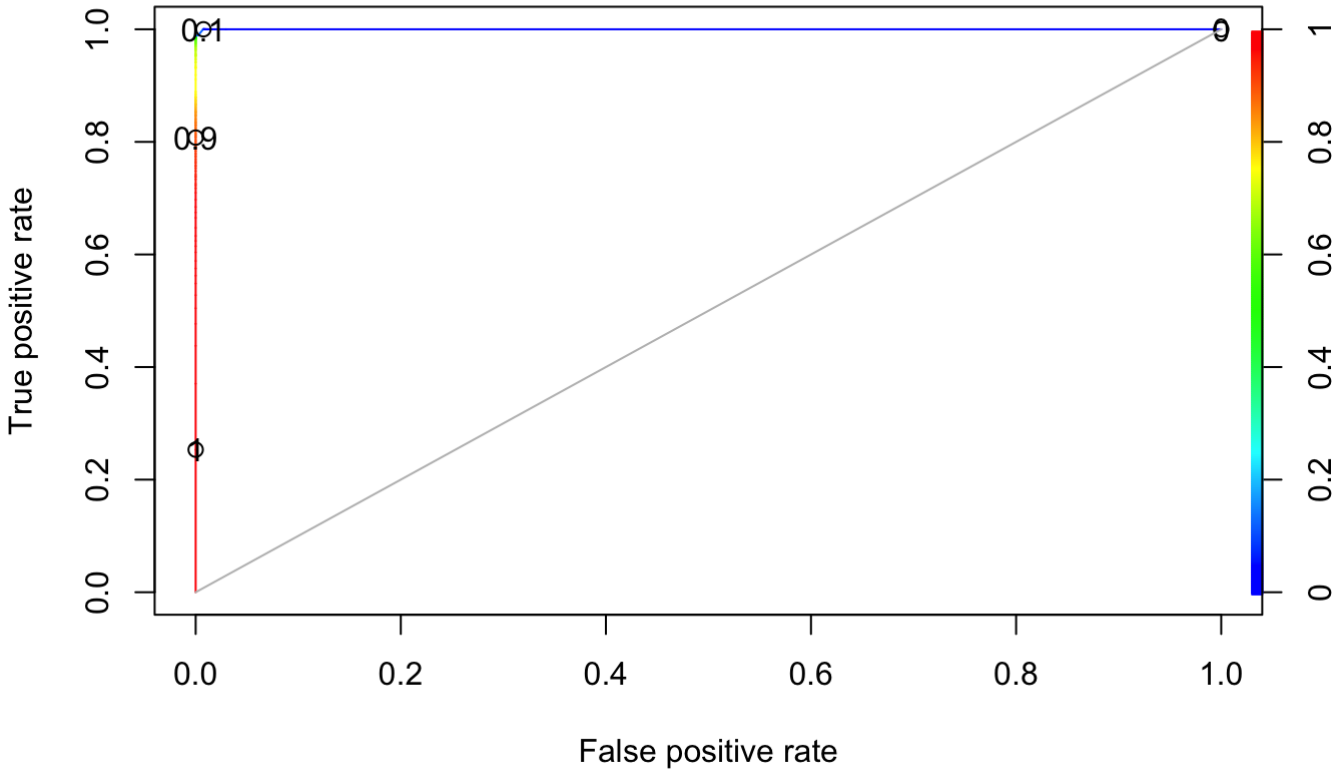
In this section, we can see the output for the SVM model with a radial kernel. With a C value of 1, we obtain an overall accuracy of about .997. Our AUC is .9964 with a good looking ROC curve. The overall accuracy appears to be slightly higher, but the AUC values is slightly lower when compared to the SVM model with a linear kernel. Looking at the threshold values, we can see that the lowest FNR corresponds to a threshold value of .05 again. However, with a threshold of .95, we obtain a higher overall accuracy (.9971) and still have low FNR and FPR values. These values are very close to that of knn and will be discussed in greater detail when looking at the performance table.

Random Forests

The ninth and final model we will train and evaluate is a random forests model. This model is also known for its accuracy in classification due to the fact that it creates an ensemble of decision trees to evaluate the data. Random forests have low correlation among the trees because it generates a random subset of features (unlike simple decision trees). For this particular model, we used 500 trees and and three different “mtry” values (1, 2, and 3) representing the three columns used. We can now evaluate the performance for this model.

```
model_creator("rf")
```

```
##   mtry  Accuracy     Kappa   AccuracySD     KappaSD
## 1    1 0.9969956 0.9509827 0.0006412222 0.010662522
## 2    2 0.9969166 0.9497695 0.0006247550 0.010453005
## 3    3 0.9967901 0.9477420 0.0004717232 0.007974458
```




```
## [1] "AUC"
## [1] 0.9999465
##   prob_threshold  Accuracy Precision Sensitivity         FNR        FPR
## 1            0.6 0.9970589 0.9986605   0.9983012 0.001698823 0.04055016
##   prob_threshold  Accuracy Precision Sensitivity          FNR       FPR
## 1           0.05 0.9895005 0.9893184    0.999951 4.900359e-05 0.3268766
```

With our final model in training, we can see that the random forests model also performed very well on the training dataset. All of the mtry values performed similarly, with a mtry value of 1 having a slightly higher accuracy (.9969). From there, we see that we obtain a nearly perfect ROC curve with an AUC value of .99995. Moving on to the threshold values, a threshold value of .05 again produces the lowest possible FNR score. However, a threshold value of .6 gives us a better overall model with low FNR and FPR scores with an accuracy of .997. This (along with SVM with a radial kernel and knn) seems to be performing very well on the training dataset and will be further discussed with the performance table.

Performance Table (Training)

Below we can see the results of all out models on the training data.

```
performance_table
```

Method	K	Lam…	AUC	Threshold	Accuracy	Precision	TPR	FNR	FPR
<chr>	<chr>	<chr>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
GLM			0.9985	0.75	0.9957	0.9973	0.9983	0.0017	0.0811
QDA			0.9982	0.60	0.9948	0.9950	0.9996	0.0004	0.1518
LDA			0.9889	0.15	0.9848	0.9920	0.9923	0.0077	0.2423
KNN	5		0.9999	0.50	0.9973	0.9988	0.9984	0.0016	0.0376
Ridge		.001	0.9802	0.75	0.9866	0.9864	1.0000	0.0000	0.4174
Lasso		.001	0.9974	0.65	0.9939	0.9941	0.9997	0.0003	0.1805
SVMlin			0.9978	0.75	0.9958	0.9972	0.9984	0.0016	0.0836
SVMrad			0.9964	0.95	0.9971	0.9990	0.9980	0.0020	0.0307
RF			0.9999	0.60	0.9971	0.9987	0.9983	0.0017	0.0406

9 rows

In our analysis for training, we could see that all eight models performed very well. All models were more than 98 percent accurate in predicting whether or not there was a blue tarp. There were slight variations from model to model, but overall, we can safely say any of these models would be adequate in predicting the existence of a blue tarp. Some of that can be attributed to the dataset in that the more “blue” color there is, it would be easier to predict the existence that a tarp would be there. Nonetheless, we can be confident in the results of these models since they all had a similar and extremely high accuracy. Not only that, but since we used cross-validation, we can be more sure that the models are not performing well because of chance or over fitting. The cross-validation helps show how the model will perform on a never before seen set of data. Not only that but all the statistics and charts shown in this paper help demonstrate the validity and outstanding performance of the models. From the performance table above, three methods seem to stand out above the rest. KNN, SVM with a radial kernel, and the random forests models all performed very well. All were similar in their statistics and had extremely high ac curacies. LDA and the ridge models on the other hand, performed the worst (but still performed well overall). These results will be discussed further in the conclusions.

Data Preparation (Testing)

After training all of our models and comparing their performance, we now move on to assembling our holdout dataset for testing. We were given seven different txt files that required some preprocessing and then were all combined into a single dataframe. We also created a BlueTarp variable for the data points that were deemed to have a blue tarp in the image. These steps can be seen below and resulted in a testing dataset with slightly over two million data points for us to evaluate the eight models with. For our analysis, we only evaluated the color value (red, blue, and green) columns to predict whether or not there was a blue tarp present in the image. To determine which values were red, blue, and green, we had to analyze the raw data and compare it to the training data (especially where blue tarps were present) and were able to determine that B1 = red, B2 = green, and B3 = blue. Dataframe y was excluded from this analysis (commented out) because it contained duplicated data.

```
myvars <- c("Red", "Green", "Blue", "BlueTarp")

z <- read.fwf("/Users/cavaningram/Downloads/Hold+Out+Data (1)/orthovnir057_ROI_NON_Blue_
Tarps.txt", skip = 8, widths = c(10, 7, 7, 11, 12, 11, 12, 5, 5, 5), header = FALSE)
colnames(z) <- c('ID','X','Y', 'Map X', 'Map Y', 'Lat', 'Lon', 'Red', 'Green', 'Blue')
z$BlueTarp <- "no"
z <- z[myvars]

#y <- read.fwf("/Users/cavaningram/Downloads/Hold+Out+Data #(1)/orthovnir067_ROI_Blue_Ta
rps_data.txt", widths = c(10), skip = 1, header = #FALSE)
#colnames(y) <- c('none', 'B1', 'B2', 'B3')
#y$BlueTarp <- "yes"
#y <- y[myvars]

x <- read.fwf("/Users/cavaningram/Downloads/Hold+Out+Data (1)/orthovnir067_ROI_Blue_Tarp
s.txt", skip = 8, widths = c(9, 5, 7, 11, 12, 11, 12, 5, 5, 5), header = FALSE)
colnames(x) <- c('ID','X','Y', 'Map X', 'Map Y', 'Lat', 'Lon', 'Red', 'Green', 'Blue')
x$BlueTarp <- "yes"
x <- x[myvars]

w <- read.fwf("/Users/cavaningram/Downloads/Hold+Out+Data (1)/orthovnir067_ROI_NOT_Blue_
Tarps.txt", skip = 8, widths = c(9, 7, 7, 11, 12, 11, 12, 5, 5, 5), header = FALSE)
colnames(w) <- c('ID','X','Y', 'Map X', 'Map Y', 'Lat', 'Lon', 'Red', 'Green', 'Blue')
w$BlueTarp <- "no"
w <- w[myvars]

v <- read.fwf("/Users/cavaningram/Downloads/Hold+Out+Data (1)/orthovnir069_ROI_Blue_Tarp
s.txt", skip = 8, widths = c(9, 5, 7, 11, 12, 11, 12, 5, 5, 5), header = FALSE)
colnames(v) <- c('ID','X','Y', 'Map X', 'Map Y', 'Lat', 'Lon', 'Red', 'Green', 'Blue')
v$BlueTarp <- "yes"
v <- v[myvars]

u <- read.fwf("/Users/cavaningram/Downloads/Hold+Out+Data (1)/orthovnir069_ROI_NOT_Blue_
Tarps.txt", skip = 8, widths = c(9, 7, 7, 11, 12, 11, 12, 5, 5, 5), header = FALSE)
colnames(u) <- c('ID','X','Y', 'Map X', 'Map Y', 'Lat', 'Lon', 'Red', 'Green', 'Blue')
u$BlueTarp <- "no"
u <- u[myvars]

t <- read.fwf("/Users/cavaningram/Downloads/Hold+Out+Data (1)/orthovnir078_ROI_Blue_Tarp
s.txt", skip = 8, widths = c(9, 5, 7, 11, 12, 11, 12, 5, 5, 5), header = FALSE)
colnames(t) <- c('ID','X','Y', 'Map X', 'Map Y', 'Lat', 'Lon', 'Red', 'Green', 'Blue')
t$BlueTarp <- "yes"
t <- t[myvars]

s <- read.fwf("/Users/cavaningram/Downloads/Hold+Out+Data (1)/orthovnir078_ROI_NON_Blue_
Tarps.txt", skip = 8, widths = c(9, 7, 7, 11, 12, 11, 12, 5, 5, 5), header = FALSE)
colnames(s) <- c('ID','X','Y', 'Map X', 'Map Y', 'Lat', 'Lon', 'Red', 'Green', 'Blue')
s$BlueTarp <- "no"
s <- s[myvars]

data2 <- rbind(z, x, w, v, u, t, s)
```

EDA (Testing)

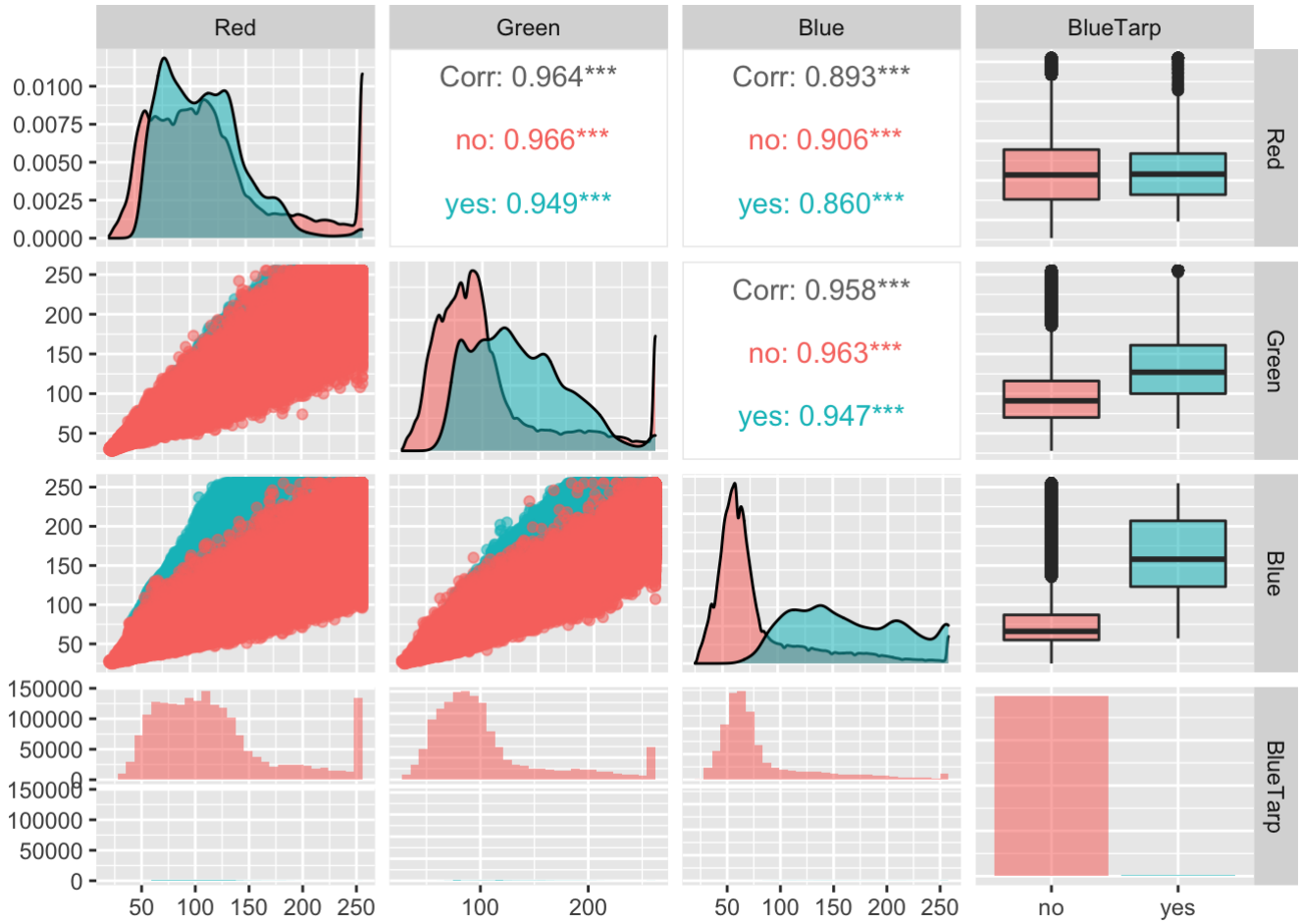
Below we will do some exploratory data analysis on the new testing data that was created for further analysis.

```
summary(data2)
```

```
##       Red              Green             Blue          BlueTarp
##  Min.   : 27.0   Min.   : 28.0   Min.   : 25.00   Length:2004177
##  1st Qu.: 76.0   1st Qu.: 71.0   1st Qu.: 55.00   Class :character
##  Median :107.0   Median : 91.0   Median : 66.00   Mode  :character
##  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
```

In this summary table, we can see that the spread of the variables (min and max values) is roughly similar (the min values for each variable are lower, but the max is the same). One major difference is the mean and the median values for each variable are lower. However, that could be due to the fact that we have many more data points in the dataset and it is giving us a more accurate representation of the dataset.

```
ggpairs(data2, aes(color = BlueTarp, alpha = .5))
```



In the plot above, we can see similar results to the training dataset. The box plots again show that blue and green should be great predictors due to their difference in values for blue tarps and non-blue tarps (whereas red is more similar between the two variables). We also can again see strong correlations between the variables. The main

difference between the testing and training dataset is that there seems to be more overlap between the variables. In both the scatter plots and the density plots, it seems harder to distinguish between the blue tarp and non blue tarp variables than with the training dataset. However, overall, this analysis shows that the dataset is still similar enough to use for testing and should produce good results.

Method used (Testing)

For our analysis of the models using the holdout data, we saved the models we found in training and applied the holdout data to them to determine just how well the models would perform on unseen data. For the ROC curves, we used a smaller dataset (25000 rows) to produce the plot. This random sample was enough to get us an accurate representation of the entire dataset for the ROC curve.

```
sample_rows <- sample(1:nrow(data2), 25000)
data_subs<- data2[sample_rows, ]
```

Models (Testing)

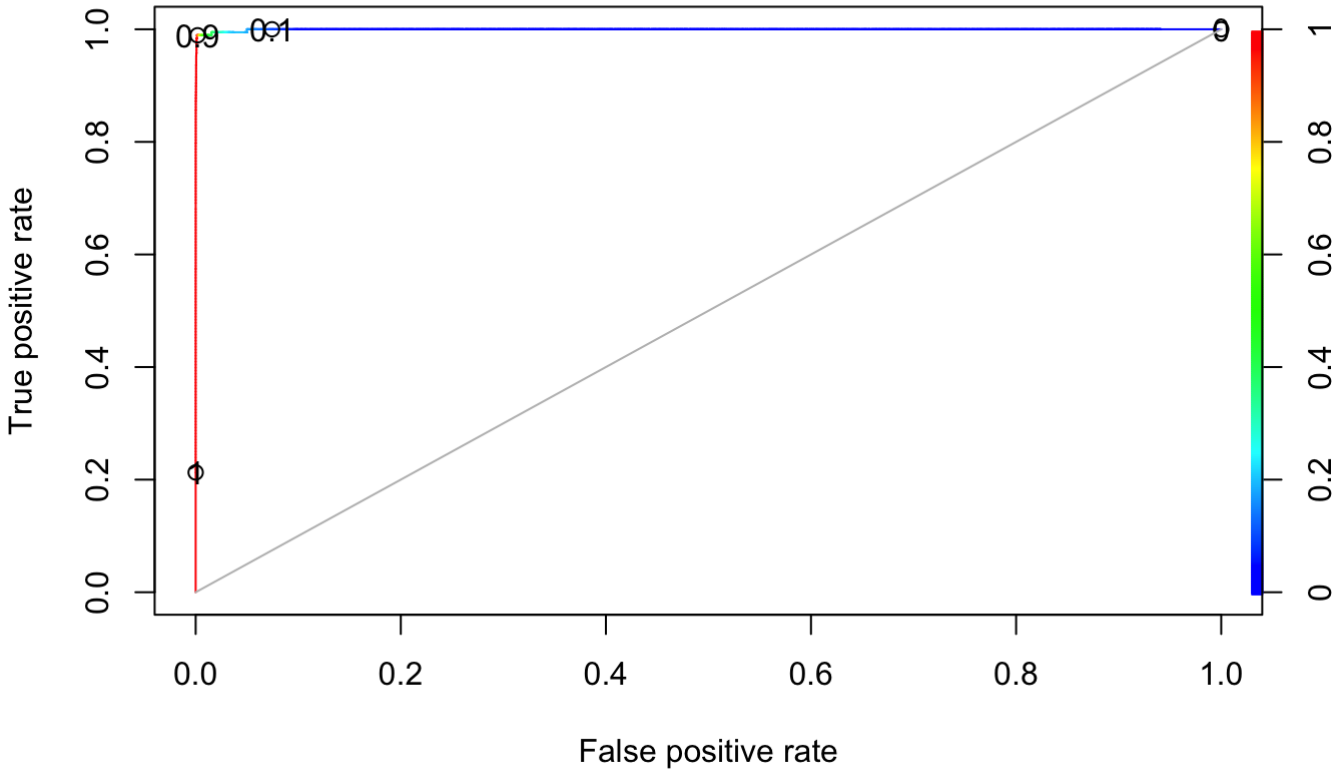
Now we can evaluate how our models performed on the testing data using our saved models from training. Rather than talk about each model individually as they are tested (as in the training portion of the project), we will talk about all of the models performance when looking at the performance table that was created.

Logistic Regression

```
lr <- readRDS("glm_mod")
predictions <- predict(lr, newdata=data2)
cm <- confusionMatrix(predictions, as.factor(data2$BlueTarp))
cm
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction      no     yes
##        no  1969383     170
##        yes   20314   14310
##
##                Accuracy : 0.9898
##                  95% CI : (0.9896, 0.9899)
##     No Information Rate : 0.9928
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.5786
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.9898
##             Specificity : 0.9883
##          Pos Pred Value : 0.9999
##          Neg Pred Value : 0.4133
##              Prevalence : 0.9928
##          Detection Rate : 0.9826
##    Detection Prevalence : 0.9827
##       Balanced Accuracy : 0.9890
##
##        'Positive' Class : no
##
```

```
pred <- predict(lr, data_subs, type='prob')
predob <- prediction(pred$yes, data_subs$BlueTarp)
model.roc <- performance(predob, measure='tpr', x.measure='fpr')
plot(model.roc, colorize=T, print.cutoffs.at=c(0, 0.1, 0.9, 1.0))
lines(x=c(0,1), y=c(0,1), col='grey')
```




```
#AUC calculation
auc_ROCR <- performance(predob, measure = "auc")
auc_ROCR <- auc_ROCR@y.values[[1]]
AUC <- replace(AUC, i, auc_ROCR)
print("AUC")
```

```
## [1] "AUC"
```

```
print(auc_ROCR)
```

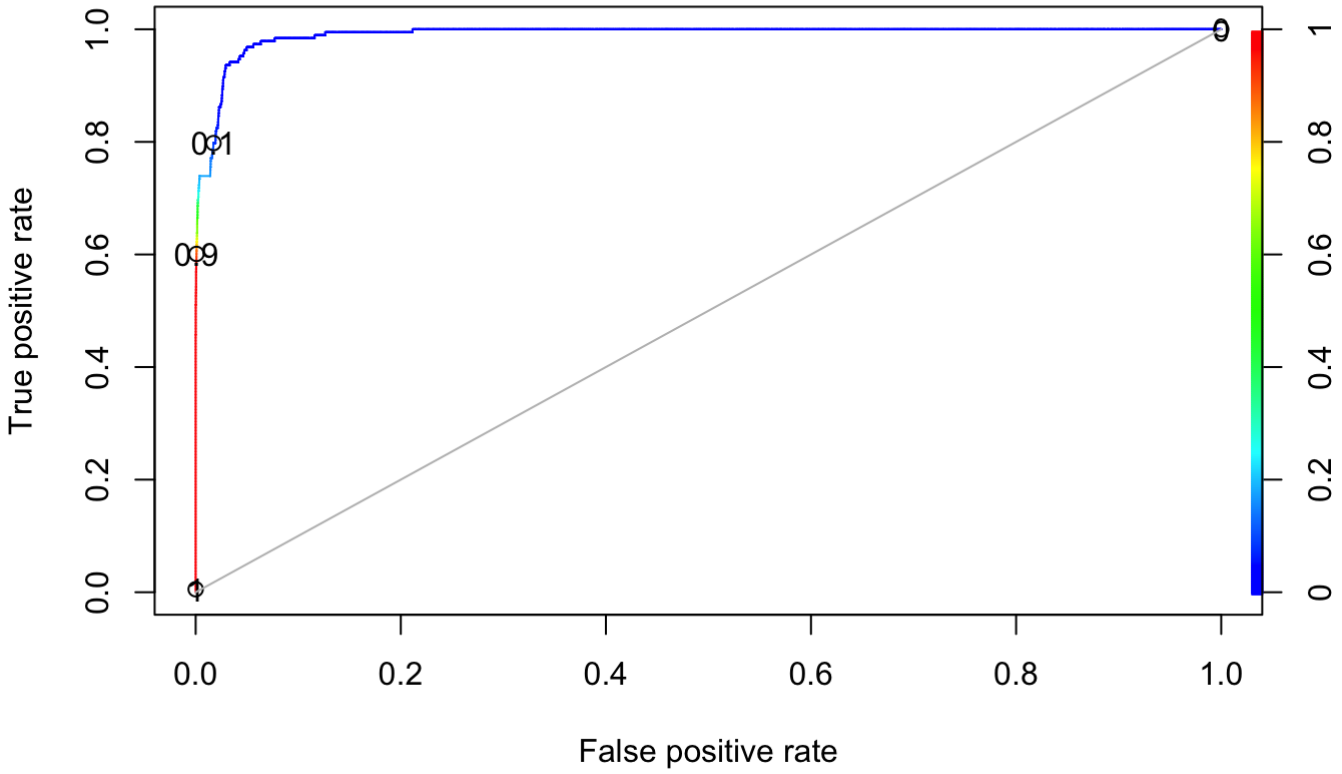
```
## [1] 0.9995753
```

QDA

```
qda <- readRDS("qda_mod")
predictions <- predict(qda, newdata=data2)
cm <- confusionMatrix(predictions, as.factor(data2$BlueTarp))
cm
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction      no     yes
##        no  1986046    4422
##        yes    3651   10058
##
##                Accuracy : 0.996
##                  95% CI : (0.9959, 0.9961)
##     No Information Rate : 0.9928
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.7116
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.9982
##             Specificity : 0.6946
##          Pos Pred Value : 0.9978
##          Neg Pred Value : 0.7337
##              Prevalence : 0.9928
##          Detection Rate : 0.9910
##    Detection Prevalence : 0.9932
##       Balanced Accuracy : 0.8464
##
##        'Positive' Class : no
##
```

```
pred <- predict(qda, data_subs, type='prob')
predob <- prediction(pred$yes, data_subs$BlueTarp)
model.roc <- performance(predob, measure='tpr', x.measure='fpr')
plot(model.roc, colorize=T, print.cutoffs.at=c(0, 0.1, 0.9, 1.0))
lines(x=c(0,1), y=c(0,1), col='grey')
```




```
#AUC calculation
auc_ROCR <- performance(predob, measure = "auc")
auc_ROCR <- auc_ROCR@y.values[[1]]
AUC <- replace(AUC, i, auc_ROCR)
print("AUC")
```

```
## [1] "AUC"
```

```
print(auc_ROCR)
```

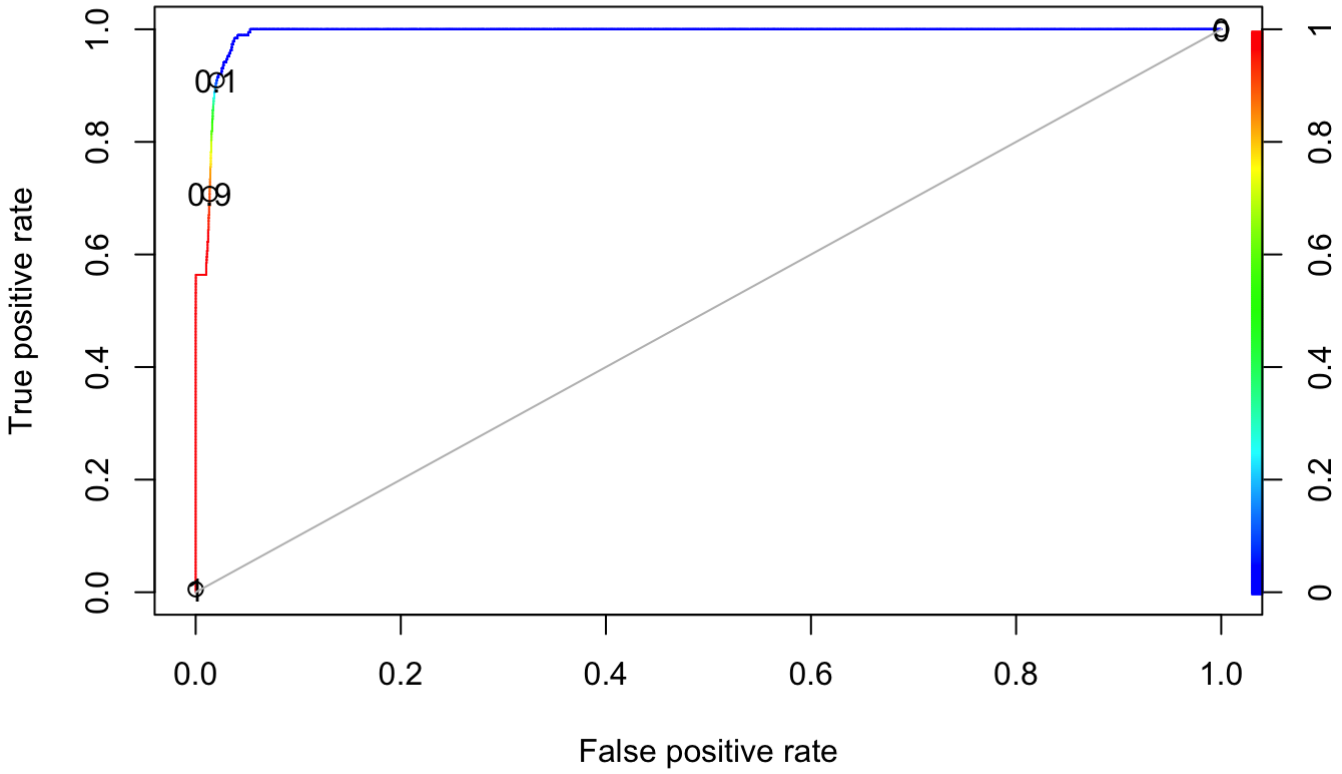
```
## [1] 0.9905931
```

LDA

```
lda <- readRDS("lda_mod")
predictions <- predict(lda, newdata=data2)
cm <- confusionMatrix(predictions, as.factor(data2$BlueTarp))
cm
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction      no     yes
##        no  1955452    2332
##        yes   34245   12148
##
##                Accuracy : 0.9817
##                  95% CI : (0.9816, 0.9819)
##     No Information Rate : 0.9928
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.3924
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.9828
##             Specificity : 0.8390
##          Pos Pred Value : 0.9988
##          Neg Pred Value : 0.2618
##              Prevalence : 0.9928
##          Detection Rate : 0.9757
##    Detection Prevalence : 0.9769
##       Balanced Accuracy : 0.9109
##
##        'Positive' Class : no
##
```

```
pred <- predict(lda, data_subs, type='prob')
predob <- prediction(pred$yes, data_subs$BlueTarp)
model.roc <- performance(predob, measure='tpr', x.measure='fpr')
plot(model.roc, colorize=T, print.cutoffs.at=c(0, 0.1, 0.9, 1.0))
lines(x=c(0,1), y=c(0,1), col='grey')
```




```
#AUC calculation
auc_ROCR <- performance(predob, measure = "auc")
auc_ROCR <- auc_ROCR@y.values[[1]]
AUC <- replace(AUC, i, auc_ROCR)
print("AUC")
```

```
## [1] "AUC"
```

```
print(auc_ROCR)
```

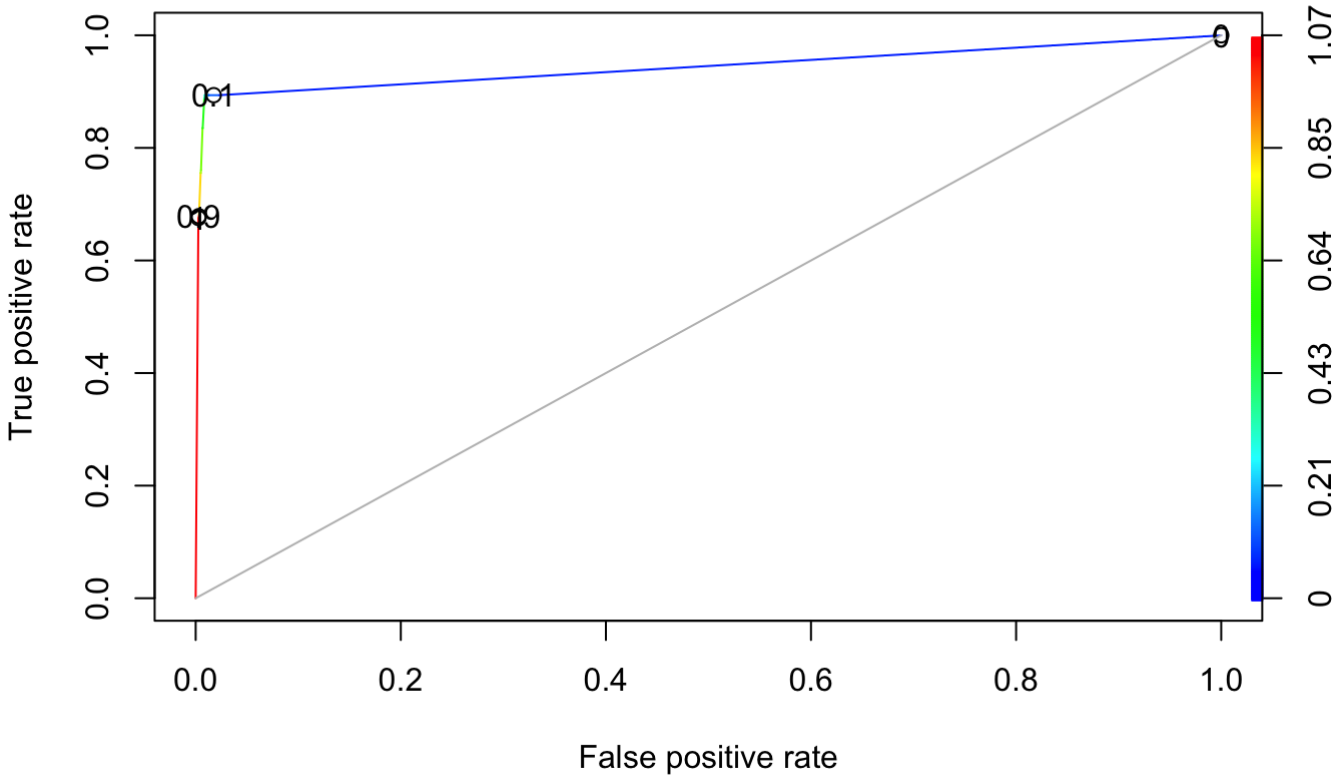
```
## [1] 0.9920356
```

KNN

```
knn <- readRDS("knn_mod")
predictions <- predict(knn, newdata=data2)
cm <- confusionMatrix(predictions, as.factor(data2$BlueTarp))
cm
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction      no     yes
##        no  1976896    2636
##        yes   12801   11844
##
##                Accuracy : 0.9923
##                  95% CI : (0.9922, 0.9924)
##     No Information Rate : 0.9928
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.6018
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.9936
##             Specificity : 0.8180
##          Pos Pred Value : 0.9987
##          Neg Pred Value : 0.4806
##              Prevalence : 0.9928
##          Detection Rate : 0.9864
##    Detection Prevalence : 0.9877
##       Balanced Accuracy : 0.9058
##
##        'Positive' Class : no
##
```

```
pred <- predict(knn, data_subs, type='prob')
predob <- prediction(pred$yes, data_subs$BlueTarp)
model.roc <- performance(predob, measure='tpr', x.measure='fpr')
plot(model.roc, colorize=T, print.cutoffs.at=c(0, 0.1, 0.9, 1.0))
lines(x=c(0,1), y=c(0,1), col='grey')
```




```
#AUC calculation
auc_ROCR <- performance(predob, measure = "auc")
auc_ROCR <- auc_ROCR@y.values[[1]]
AUC <- replace(AUC, i, auc_ROCR)
print("AUC")
```

```
## [1] "AUC"
```

```
print(auc_ROCR)
```

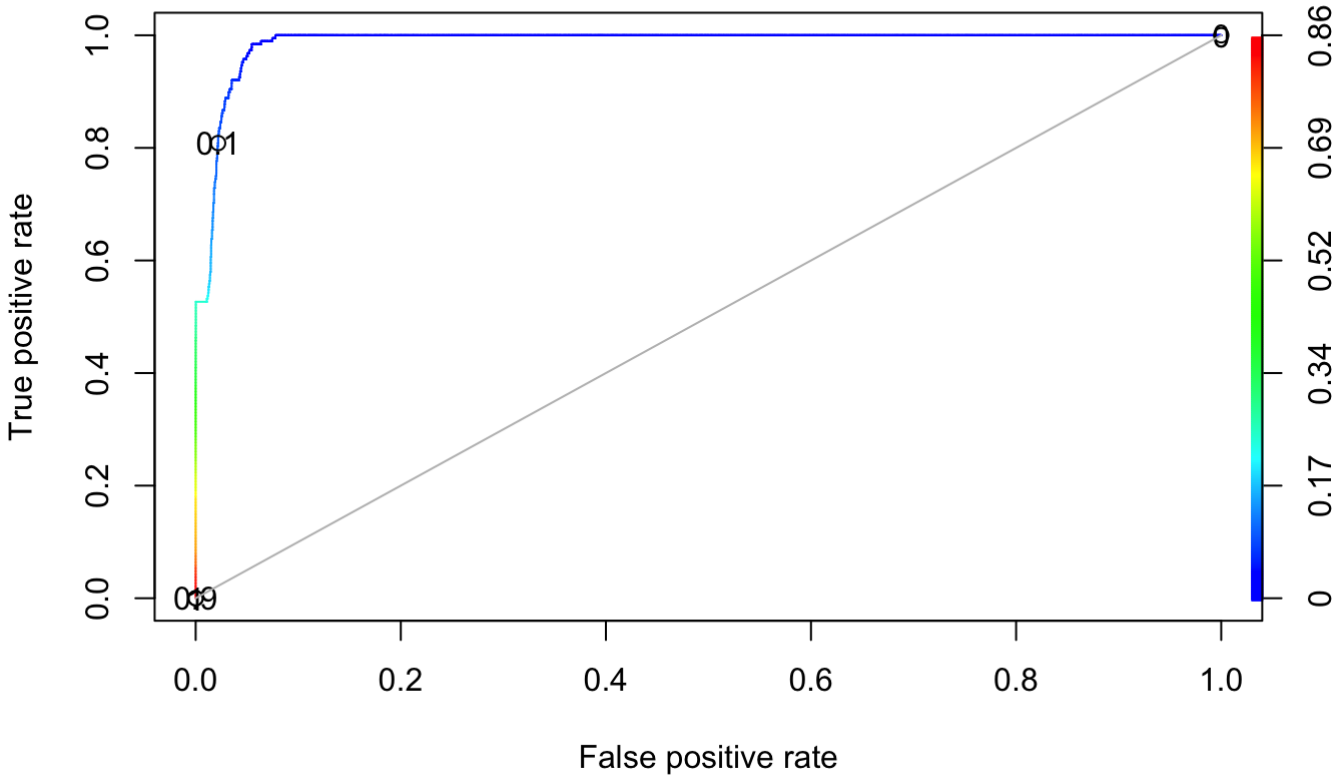
```
## [1] 0.9434313
```

Ridge

```
ridge <- readRDS("ridge_mod")
predictions <- predict(ridge, newdata=data2)
cm <- confusionMatrix(predictions, as.factor(data2$BlueTarp))
cm
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction      no     yes
##        no  1989688   10552
##        yes       9    3928
##
##                Accuracy : 0.9947
##                  95% CI : (0.9946, 0.9948)
##     No Information Rate : 0.9928
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.4248
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 1.0000
##             Specificity : 0.2713
##          Pos Pred Value : 0.9947
##          Neg Pred Value : 0.9977
##              Prevalence : 0.9928
##          Detection Rate : 0.9928
##    Detection Prevalence : 0.9980
##       Balanced Accuracy : 0.6356
##
##        'Positive' Class : no
##
```

```
pred <- predict(ridge, data_subs, type='prob')
predob <- prediction(pred$yes, data_subs$BlueTarp)
model.roc <- performance(predob, measure='tpr', x.measure='fpr')
plot(model.roc, colorize=T, print.cutoffs.at=c(0, 0.1, 0.9, 1.0))
lines(x=c(0,1), y=c(0,1), col='grey')
```




```
#AUC calculation
auc_ROCR <- performance(predob, measure = "auc")
auc_ROCR <- auc_ROCR@y.values[[1]]
AUC <- replace(AUC, i, auc_ROCR)
print("AUC")
```

```
## [1] "AUC"
```

```
print(auc_ROCR)
```

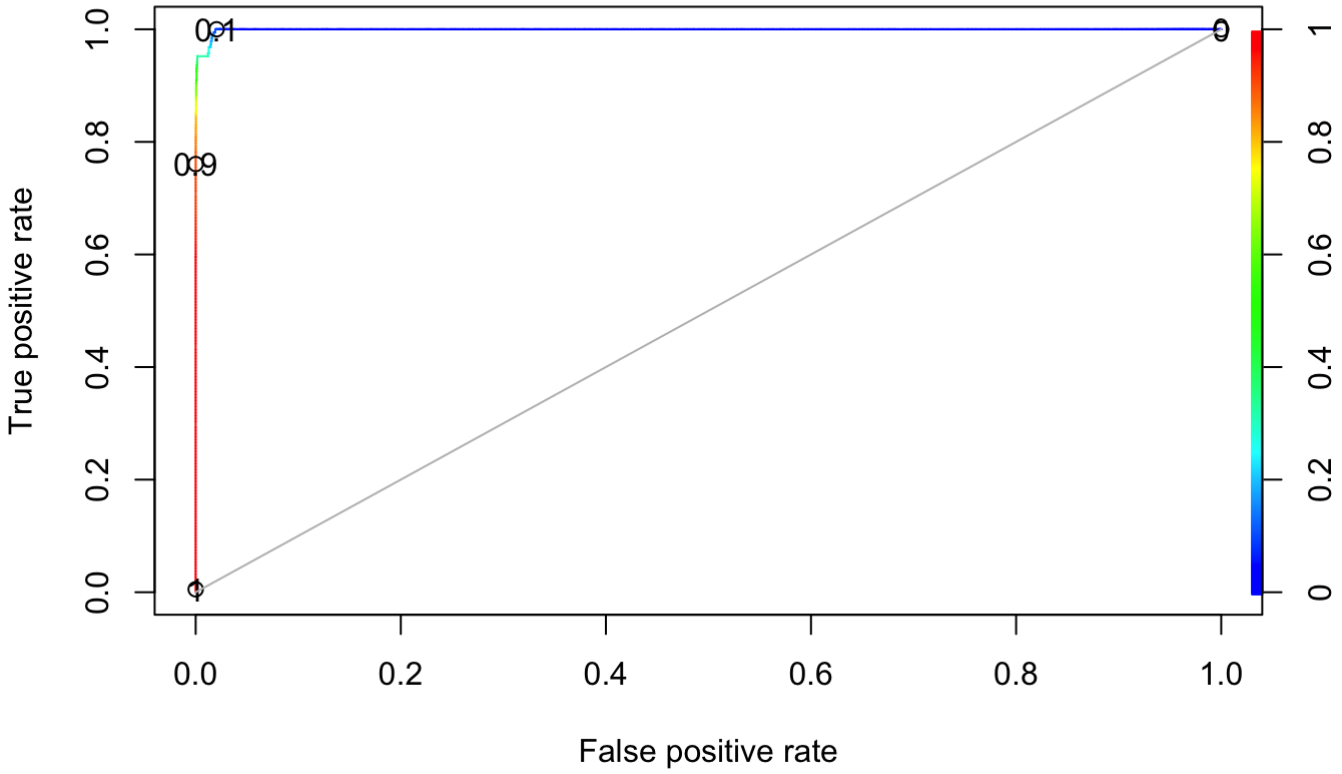
```
## [1] 0.9880182
```

Lasso

```
lasso <- readRDS("lasso_mod")
predictions <- predict(lr, newdata=data2)
cm <- confusionMatrix(predictions, as.factor(data2$BlueTarp))
cm
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction      no     yes
##        no  1969383     170
##        yes   20314   14310
##
##                Accuracy : 0.9898
##                  95% CI : (0.9896, 0.9899)
##     No Information Rate : 0.9928
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.5786
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.9898
##             Specificity : 0.9883
##          Pos Pred Value : 0.9999
##          Neg Pred Value : 0.4133
##              Prevalence : 0.9928
##          Detection Rate : 0.9826
##    Detection Prevalence : 0.9827
##       Balanced Accuracy : 0.9890
##
##        'Positive' Class : no
##
```

```
pred <- predict(lasso, data_subs, type='prob')
predob <- prediction(pred$yes, data_subs$BlueTarp)
model.roc <- performance(predob, measure='tpr', x.measure='fpr')
plot(model.roc, colorize=T, print.cutoffs.at=c(0, 0.1, 0.9, 1.0))
lines(x=c(0,1), y=c(0,1), col='grey')
```




```
#AUC calculation
auc_ROCR <- performance(predob, measure = "auc")
auc_ROCR <- auc_ROCR@y.values[[1]]
AUC <- replace(AUC, i, auc_ROCR)
print("AUC")
```

```
## [1] "AUC"
```

```
print(auc_ROCR)
```

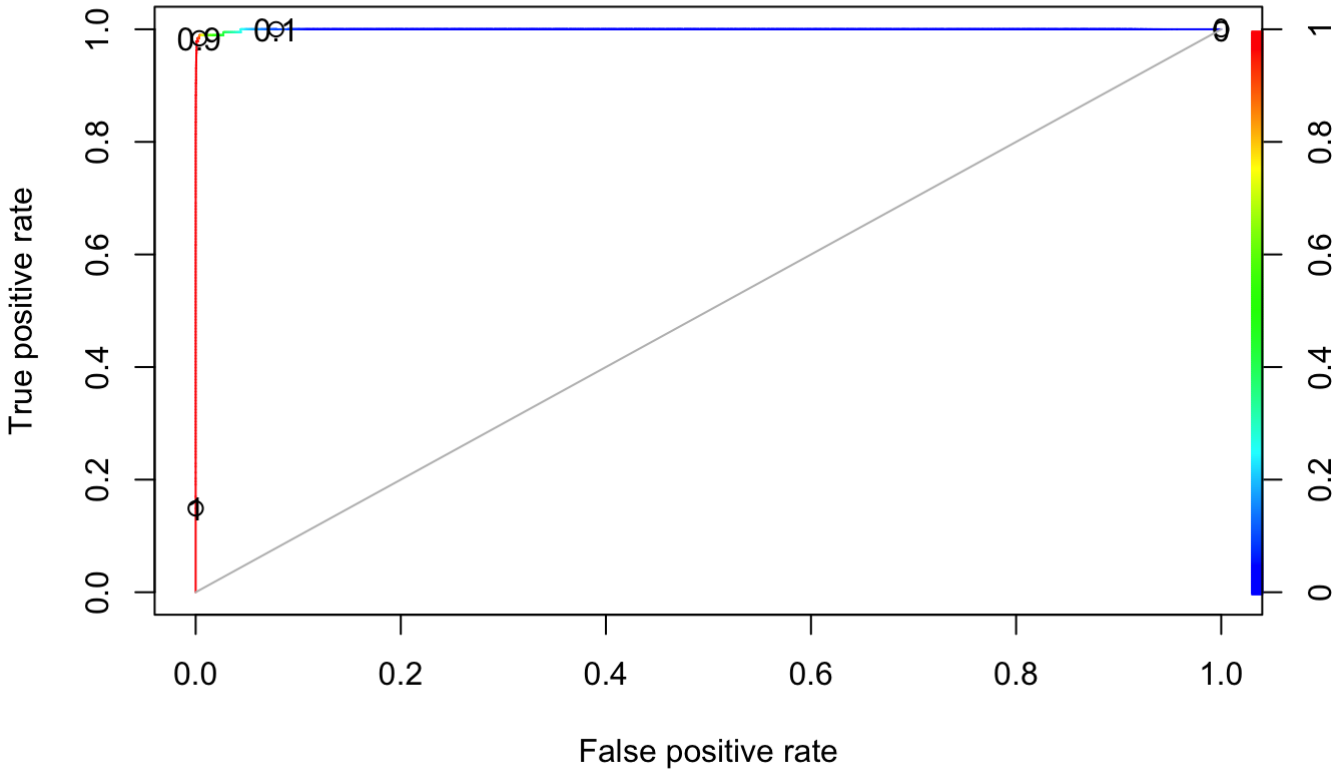
```
## [1] 0.9991894
```

SVM (Linear kernel)

```
svml <- readRDS("svmLinear_mod")
predictions <- predict(svml, newdata=data2)
cm <- confusionMatrix(predictions, as.factor(data2$BlueTarp))
cm
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction      no     yes
##        no  1949087     162
##        yes   40610   14318
##
##                Accuracy : 0.9797
##                  95% CI : (0.9795, 0.9799)
##     No Information Rate : 0.9928
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.4058
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.9796
##             Specificity : 0.9888
##          Pos Pred Value : 0.9999
##          Neg Pred Value : 0.2607
##              Prevalence : 0.9928
##          Detection Rate : 0.9725
##    Detection Prevalence : 0.9726
##       Balanced Accuracy : 0.9842
##
##        'Positive' Class : no
##
```

```
pred <- predict(svml, data_subs, type='prob')
predob <- prediction(pred$yes, data_subs$BlueTarp)
model.roc <- performance(predob, measure='tpr', x.measure='fpr')
plot(model.roc, colorize=T, print.cutoffs.at=c(0, 0.1, 0.9, 1.0))
lines(x=c(0,1), y=c(0,1), col='grey')
```




```
#AUC calculation
auc_ROCR <- performance(predob, measure = "auc")
auc_ROCR <- auc_ROCR@y.values[[1]]
AUC <- replace(AUC, i, auc_ROCR)
print("AUC")
```

```
## [1] "AUC"
```

```
print(auc_ROCR)
```

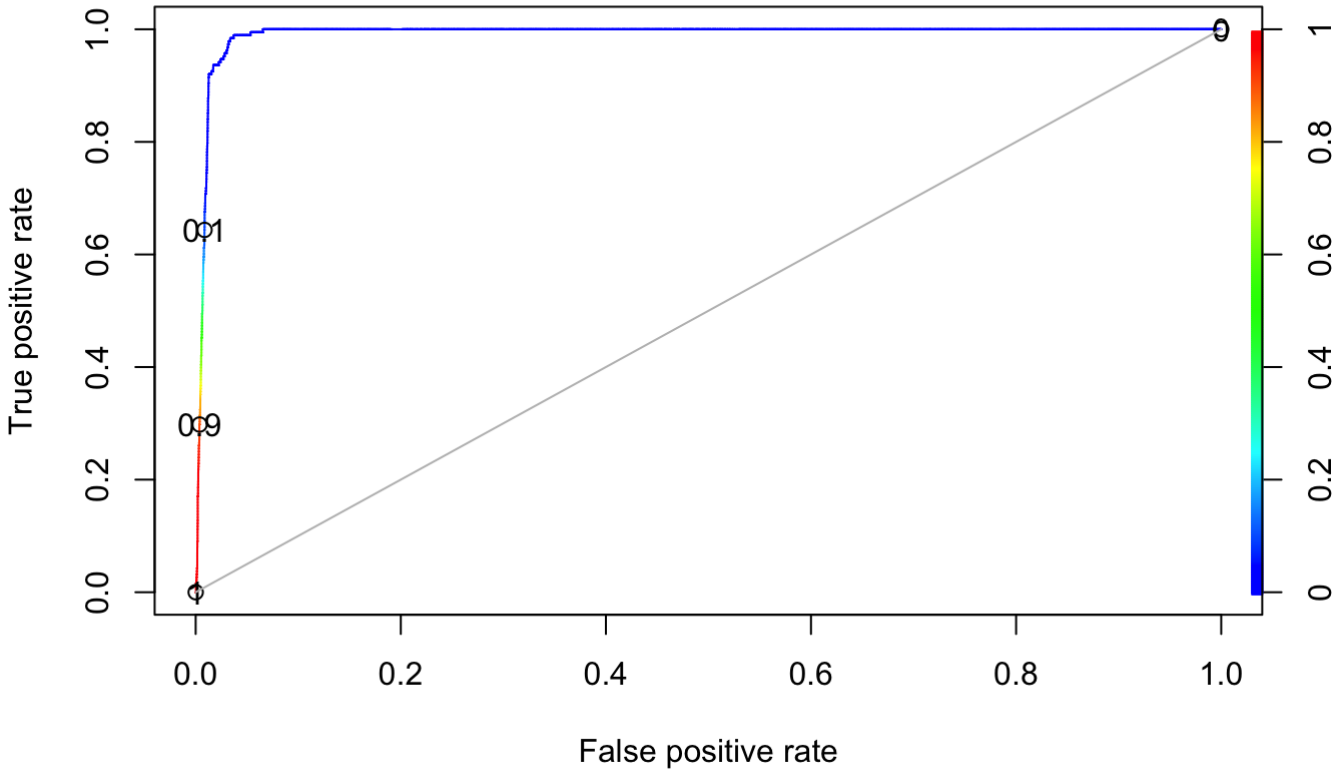
```
## [1] 0.9995292
```

SVM (radial kernel)

```
svmr <- readRDS("svmRadial_mod")
predictions <- predict(svmr, newdata=data2)
cm <- confusionMatrix(predictions, as.factor(data2$BlueTarp))
cm
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction      no     yes
##        no  1978171    8115
##        yes   11526    6365
##
##                Accuracy : 0.9902
##                  95% CI : (0.9901, 0.9903)
##     No Information Rate : 0.9928
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.3884
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.9942
##             Specificity : 0.4396
##          Pos Pred Value : 0.9959
##          Neg Pred Value : 0.3558
##              Prevalence : 0.9928
##          Detection Rate : 0.9870
##    Detection Prevalence : 0.9911
##       Balanced Accuracy : 0.7169
##
##        'Positive' Class : no
##
```

```
pred <- predict(svmr, data_subs, type='prob')
predob <- prediction(pred$yes, data_subs$BlueTarp)
model.roc <- performance(predob, measure='tpr', x.measure='fpr')
plot(model.roc, colorize=T, print.cutoffs.at=c(0, 0.1, 0.9, 1.0))
lines(x=c(0,1), y=c(0,1), col='grey')
```




```
#AUC calculation
auc_ROCR <- performance(predob, measure = "auc")
auc_ROCR <- auc_ROCR@y.values[[1]]
AUC <- replace(AUC, i, auc_ROCR)
print("AUC")
```

```
## [1] "AUC"
```

```
print(auc_ROCR)
```

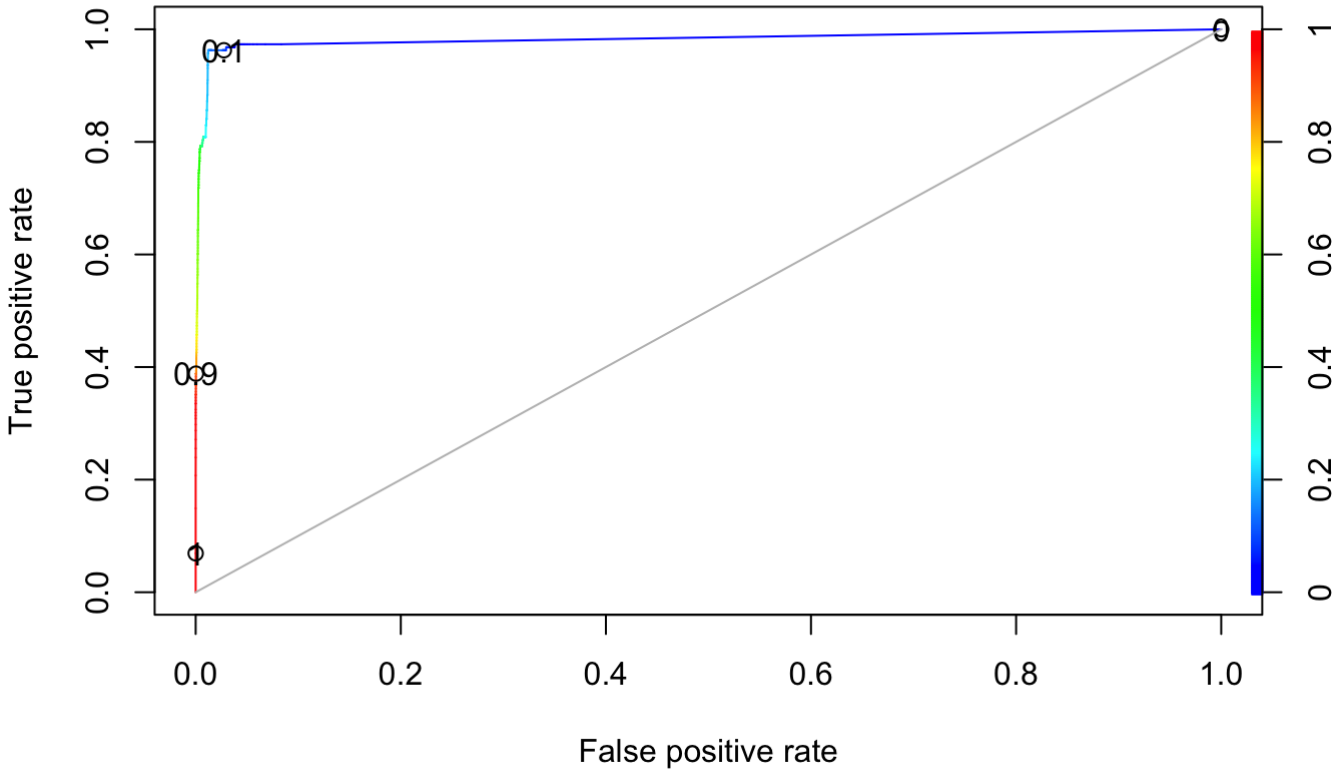
```
## [1] 0.9917224
```

Random Forests

```
rf <- readRDS("rf_mod")
predictions <- predict(rf, newdata=data2)
cm <- confusionMatrix(predictions, as.factor(data2$BlueTarp))
cm
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction      no     yes
##        no  1982367    3183
##        yes    7330   11297
##
##                Accuracy : 0.9948
##                  95% CI : (0.9947, 0.9949)
##     No Information Rate : 0.9928
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.6799
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.9963
##             Specificity : 0.7802
##          Pos Pred Value : 0.9984
##          Neg Pred Value : 0.6065
##              Prevalence : 0.9928
##          Detection Rate : 0.9891
##    Detection Prevalence : 0.9907
##       Balanced Accuracy : 0.8882
##
##        'Positive' Class : no
##
```

```
pred <- predict(rf, data_subs, type='prob')
predob <- prediction(pred$yes, data_subs$BlueTarp)
model.roc <- performance(predob, measure='tpr', x.measure='fpr')
plot(model.roc, colorize=T, print.cutoffs.at=c(0, 0.1, 0.9, 1.0))
lines(x=c(0,1), y=c(0,1), col='grey')
```




```
#AUC calculation
auc_ROCR <- performance(predob, measure = "auc")
auc_ROCR <- auc_ROCR@y.values[[1]]
AUC <- replace(AUC, i, auc_ROCR)
print("AUC")
```

```
## [1] "AUC"
```

```
print(auc_ROCR)
```

```
## [1] 0.9825719
```

Performance Table (Testing)

Now that we have out holdout dataset assembled, we were able to test the models created in training on this new testing dataset. The performance of the models with the holdout dataset can be seen in the table below.

```
AUC <- c(AUC_lr, AUC_qda, AUC_lda, AUC_knn, AUC_rid, AUC_las, AUC_svml, AUC_svmr, AUC_r
f)

Accuracy <- c(accuracy_lr, accuracy_qda, accuracy_lda, accuracy_knn, accuracy_rid, accur
acy_las, accuracy_svml, accuracy_svmr, accuracy_rf)

FPR <- c(FPR_lr, FPR_qda, FPR_lda, FPR_knn, FPR_rid, FPR_las, FPR_svml, FPR_svmr, FPR_r
f)

FNR <- c(FNR_lr, FNR_qda, FNR_lda, FNR_knn, FNR_rid, FNR_las, FNR_svml, FNR_svmr, FNR_r
f)

Balanced_Acc <- c(BA_lr, BA_qda, BA_lda, BA_knn, BA_rid, BA_las, BA_svml, BA_svmr, BA_r
f)

Precision <- c(precision_lr, precision_qda, precision_lda, precision_knn, precision_rid,
precision_las, precision_svml, precision_svmr, precision_rf)

TPR <- c(TPR_lr, TPR_qda, TPR_lda, TPR_knn, TPR_rid, TPR_las, TPR_svml, TPR_svmr, TPR_r
f)

Method = c("GLM","QDA","LDA","KNN","Ridge", "Lasso", "SVMlin", "SVMrad", "RF")

K = c("","","","5","", "", "", "", "")
Lamda = c("","","","",".001", ".001", "", "", "")
Parameters = c("","","","","", "", "", "", "3")

performance_table_test <- data.frame(Method, K, Lamda, AUC = round(AUC, 4), Accuracy = r
ound(Accuracy, 4), FNR = round(FNR, 4), FPR = round(FPR, 4), Precision = round(Precisio
n, 4), TPR = round(TPR, 4))
```

```
performance_table_test
```

Method	K	Lamda	AUC	Accuracy	FNR	FPR	Precision	TPR
<chr>	<chr>	<chr>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
GLM			0.9996	0.9898	0.0102	0.0117	0.9999	0.9898
QDA			0.9906	0.9960	0.0018	0.3054	0.9978	0.9982
LDA			0.9920	0.9817	0.0172	0.1610	0.9988	0.9828
KNN	5		0.9434	0.9923	0.0064	0.1820	0.9987	0.9936
Ridge		.001	0.9880	0.9947	0.0000	0.7287	0.9947	1.0000
Lasso		.001	0.9992	0.9898	0.0102	0.0117	0.9999	0.9898
SVMlin			0.9995	0.9797	0.0204	0.0112	0.9999	0.9796
SVMrad			0.9917	0.9902	0.0058	0.5604	0.9959	0.9942
RF			0.9826	0.9948	0.0037	0.2198	0.9984	0.9963

9 rows

When evaluating the models performance in the performance table above, a couple of things stand out. First, all the models again performed very well overall when just examining their accuracy scores. However, when looking at the FPR values, we can see that random forests, ridge, SVM (radial kernel), and QDA all have high FPR scores. Even though this metric is not as important as the overall accuracy or FNR, these methods stand out because they have FPR scores higher than 20 percent. Even with the high FPR score, QDA does obtain the highest overall accuracy. The reason this seems to have occurred is because these models seemed to predict “no” more than they did “yes” (when looking at the outputs in each of their respective sections). This caused a high accuracy, but high FPR because there were more “no” data points than there were “yes”.

The next two models we wanted to evaluate were SVM (linear kernel) and LDA. Both of these methods performed similarly (which makes sense since they are assuming the data has some sort of linearity). These two methods have the highest overall FNR scores (even though they are both around two percent). They also produce similar AUC values and accuracy scores. Overall, these models seemed to be average in their performance when comparing them to some of the other models.

The best performing models that we might want to take a closer look at are knn, random forests, lasso, and logistic regression. When looking at the logistic regression and lasso models, we seem to have nearly identical scores, but logistic regression is slightly better in some areas. When comparing knn and random forests, we again see similar overall metrics, but since the random forests model has a higher overall accuracy and lower FNR score (albeit with a high FPR) we will choose that model for analysis. Not only that, but the knn model does appear to take the longest time to run consistently. The best model between these two will be discussed further in the conclusion section.

When simply comparing this performance table for the holdout data to that of the training data, their are some differences. One main difference is that all of the models appeared to perform slightly worse on the holdout data. This makes sense because we are using a much larger dataset with more points/outliers that can occur. In particular though, the SVM models did not perform as well on the testing dataset than the training. As talked about before, we can also see the high FPR values that some models experienced, showing that they are simply predicting “no” more often than “yes” which is skewing their accuracy scores slightly since there are more “no” variables overall.

Conclusions

Conclusion 1 (Which algorithm works best in the cross-validation data)

Even though all the models performed well in the training data with cross-validation, one model performed better than the rest. We can see in the performance table for the training data that the k nearest neighbors model (using five neighbors at a threshold level of .5) obtained the highest overall accuracy (.9973) of any other model at any threshold. Not only that, but it also had a very low FNR (.0016) and a low FPR (.0376). What this means is that the model is extremely accurate and will lower the amount of tarps that are missed as well as lowering the amount of time and resources lost from sending people to “blue tarps” that really are not there. This model has the best balance of any model in helping the most people possible in the most efficient manner.

Conclusion 2 (Which algorithm works best on the Hold-Out data)

Two models stood out above the rest when deciding which model was the best on the holdout data. The logistic regression model performed well all the way around in each metric, whereas the random forests model performed very well in some metrics (accuracy and FNR), but not as good in others (FPR). When simply determining which model performed the best (not necessarily which one to choose for the analysis), I would say the logistic regression model performed the best. There are a couple of reasons I would say this even though the logistic regression model did not have one of the best accuracies. The most important reason is because it has a balance between the FPR and FNR scores (both being below two percent). This shows that it was actually predicting the outcomes of the holdout dataset well, whereas some of the other models were simply predicting “no” a lot more which resulted in a higher accuracy because it had more data points. It was also one of the quickest models to run and produced great results.

Conclusion 3 (Justification of findings between training and testing)

Our best model for the training dataset was knn, whereas the best model for the testing dataset was logistic regression. The main reason for this has been discussed throughout this paper already, but the reason knn did not perform as well on the testing data was because it was simply predicting more “no” outcomes than “yes”. With a smaller dataset in training, this difference was harder to see because there were less points to classify, however, as the number of points increased, we could see that logistic regression was doing a better job at correctly predicting an outcome and deciphering between “yes” and “no” than knn and the other models were doing. Both of these models were towards the top of their class for testing and training, but at the end of the day, one performed slightly better on the holdout data (logistic regression) than the other (knn).

Conclusion 4 (Which algorithm you recommend for use in detection of blue tarps)

Now, the most important question in the project is trying to figure out what algorithm should be used for the detection of blue tarps. At first glance, the obvious and easy answer would seem to be logistic regression as it performed the best on the holdout data and was the best at actually predicting the actual outcomes. Not only that, but it also performed very well on the training data. However, I would argue that logistic regression would not be the best choice for this project. Even though it performed the best, it has a “higher” FNR value of around 1 percent. As discussed earlier, the FNR is important because the lower the FNR value, the fewer number of blue tarps that would be missed. In the output for logistic regression in testing seen above, we can see that the model missed around 20,000 blue tarp points. However, a model such as random forests only missed around 7,000 blue tarp points. Because of that, I would argue random forests should be used for the detection of blue tarps.

The random forests model is definitely not the obvious choice in this analysis, but because it has such a high accuracy and low FNR on the holdout data, it is the one that should be used. The only argument against the random forests model is that it has a high FPR on the holdout data of around twenty percent. While that is a cause for some concern, the most important thing for this analysis is to accurately predict the existence of a blue tarp so that those people can receive aid who need it. There might be more time and resources wasted going to places that were thought to have blue tarps that do not, but the random forests model provides a great overall accuracy in predicting the existence of the blue tarps with around a three times lower FNR score compared to logistic regression.The random forests model should be used for this analysis because it provides a great FNR score and overall accuracy while also not having too high of a FPR score.

Conclusion 5 (Effectiveness of Saving Life)

With all that being said, it is essential to try and determine how helpful this analysis could be in saving human life. We have already seen that the models used are very effective in predicting the existence of a blue tarp with the given data, which is a great start. However, more steps need to be taken to help save human life. As it stands, all the model is able to do is predict the existence of a blue tarp from the given color data. The most critical information that should be used in the analysis is some sort of location data for the data points. With the location data provided in the holdout set, we might be able to better determine with the models where the blue tarps are located and where to send help to. Not only that, but we might be able to find clusters of blue tarps and be sure to help send resources to their first to help a large portion of people fast. Another data point that would be useful would be either a cluster of pixels (by using their location) or simply the size of overall “objects” to help better determine what is happening. However, the analysis of the accuracy of identifying the blue tarps is already very high, which is a great place to start. This analysis is a great stepping stone in helping to save human life, but more information could be used to help more people.

Conclusion 6 (Alternative methods)

Even though the models chosen in the analysis performed well, we might be able to better improve the accuracy by using a different method or different predictors. Using the interactions between the predictors could produce a slightly more accurate model for specific models (especially for something like ridge and lasso), so it might be helpful to use those as predictors (even though it makes the model more complex) to help improve results. Another thought might be to try and utilize an active learner in combination with a neural network where we can help teach the model precisely what a blue tarp is (almost like when Google prompts you with images to prove you are not a robot). This method (along with others) might help improve this analysis but also might assist in performing other useful tasks.

Conclusion 7 (Other Applications)

Overall, this is a great place to start. However, it begs the question if this sort of analysis can be applied in any other form or fashion to other problems. Assuming we can obtain similar data points for color and the classifications of what things are, we might also be able to locate additional points of interest. For one, maybe we would be able to use this to help determine areas that have been covered by water by either analyzing the color of the water or how the color that was once there is not longer there. We might also be able to utilize this to determine areas with high smoke/fire and areas of need (but you would need active data that is constantly updating, so it would be challenging). This analysis could be utilized in many different ways, but we have already seen that this process is good at identifying specific objects.

References and Appendix

For a majority of the project, I utilized the class notes and book (Introduction to Statistical Learning) for help when needed.

The code below helps give an example of a different set of predictors that could be used for a model.

```
# c <- train(BlueTarp ~ Red + Blue + Green + Red:Blue + Red:Green + Red:Blue:Green, data
= data, method = "glm",
#            trControl = ctrl)
```