Disaster Relief Project: Part I

Karan Manwani

Aug 15, 2021

SYS 6018 | Summer 2021 | University of Virginia

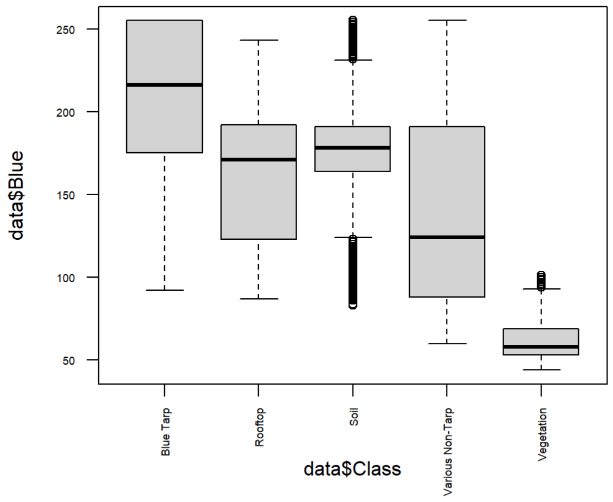
1 Introduction

#After the earthquake in Haiti in 2010, rescue workers were working in challenging conditions to deliver aid to displaced persons. One of the biggest challenges was locating people who had been displaced in conditions where communication systems had been destroyed and roads were impassable. A solution was to use high resolution geo-referenced imagery taken from airplanes to locate people. It was known that people who had been displaced were setting up blue tarps as temporary shelters, and the location of these blue tarps could be identified by the images in order to help rescue workers determine where to deliver aid to. However, given the number of images, it was difficult and inefficient to go through them manually. A more efficient way is to build a model that would use the RGB codes from the images to determine which images are blue tarps. In this project, I have built several models and calculated the accuracy rates of each to them to determine which model to use. A 10 fold cross-validation was used to evaluate each of the models.

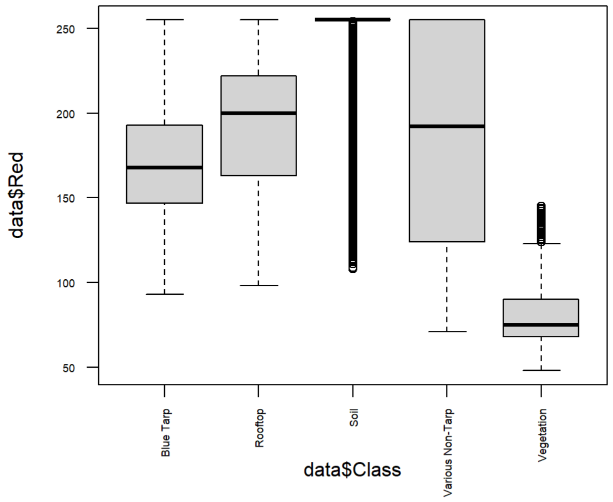
2 Training Data / EDA

#Load the data, and perform exploratory data analysis using bloxplots for each color for all the classes.

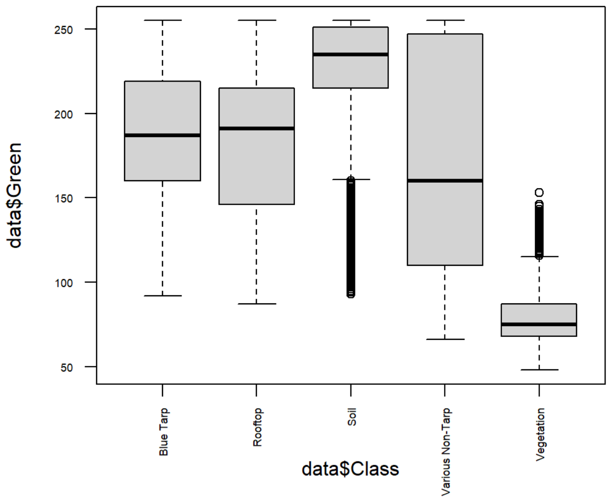
```r
data <- read.csv("HaitiPixels.csv")
par(cex.axis=0.55)
boxplot(data$Blue ~ data$Class,las=2)
```



```r
boxplot(data$Red ~ data$Class,las=2)
```



```r
boxplot(data$Green ~ data$Class,las=2)
```



#The boxplots show that the numbers for Blue are much higher for blue tarps than the other classes. Hence, we should be able to build a model that can identify blue tarps using the images. It also shows Red tends to be prevelant in Various Non-tarps and rooftops.

```r
# Load Required Packages
library(tidyverse)
library(boot)
library(caret)
library(ggplot2)
library(ROCR)
library(pROC)
library(glmnet)
library(e1071)
library(kernlab)
```

3 Model Training

#The models below were all built using a 10-fold cross validation, where by the training data would have 10 folds for building the models. The data was trained on all the folds except one fold, which was used for fitting the data, and this was repeated until all folds were used for training and each fold was then used for testing as part of the process. The Training data was made up of 70% of the entire data set and the remaining 30% of the data was used to test the models.

3.1 Set-up

#Create a new column to split class into 2 - Blue Tarp and Non-Blue Tarpc(for everything that is not a Blue Tarp)

```r
data <- transform(data, "BlueTarp"= ifelse(Class=="Blue Tarp", "Y", "N"))
data$BlueTarp<-as.factor(data$BlueTarp)
```

#split data into test and train for validation set cross validations by 70-30 split. Train data will be used for 10-fold validation. Test data will be used for predictions. Set up the specifications for the 10 fold cross-validation.

```
set.seed(1)
sample_data<-sample.int(nrow(data),floor(0.7*nrow(data)),replace=F)
train.data<-data[sample_data, ]
test.data<-data[-sample_data, ]

ctrlspecs<-trainControl(method="cv",number=10,savePredictions = "all",classProbs = TRUE)
```

3.2 Logistic Regression

#In this model, the response falls into two categories for if an image is a Blue Tarp or not i.e Y or N. We use logistic regression model to determine the probability that an image belongs to a Blue Tarp image or not.

```
model_l<-train(as.factor(BlueTarp)~Red+Green+Blue,data=train.data,method="glm",family=binomial,trC
         ontrol=ctrlspecs)
```

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

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

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

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

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

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

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

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

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

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

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

```
summary(model_l)
```

```
#> 
#> Call:
#> NULL
#> 
#> Deviance Residuals: 
#>     Min       1Q   Median       3Q      Max  
#> -3.4295  -0.0193  -0.0013   0.0000   3.7893  
#> 
#> Coefficients:
#>             Estimate Std. Error z value Pr(>|z|)    
#> (Intercept)  0.26070    0.22929   1.137    0.256    
#> Red         -0.28680    0.01696 -16.907   <2e-16 ***
#> Green       -0.19543    0.01604 -12.185   <2e-16 ***
#> Blue         0.47551    0.01917  24.807   <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: 12440  on 44267  degrees of freedom
#> Residual deviance:  1173  on 44264  degrees of freedom
#> AIC: 1181
#> 
#> Number of Fisher Scoring iterations: 12
```

#Logistic Regression Confusion Matrix

```
predictions<-predict(model_l,test.data)
confusionMatrix(predictions,test.data$BlueTarp,positive = "Y")
```

```
#> Confusion Matrix and Statistics
#> 
#>           Reference
#> Prediction     N     Y
#>          N 18333    77
#>          Y    20   543
#>                                          
#>                Accuracy : 0.9949         
#>                  95% CI : (0.9938, 0.9959)
#>     No Information Rate : 0.9673         
#>     P-Value [Acc > NIR] : < 2.2e-16      
#>                                          
#>                   Kappa : 0.9154         
#>                                          
#>  Mcnemar's Test P-Value : 1.301e-08      
#>                                          
#>             Sensitivity : 0.87581        
#>             Specificity : 0.99891        
#>          Pos Pred Value : 0.96448        
#>          Neg Pred Value : 0.99582        
#>              Prevalence : 0.03268        
#>          Detection Rate : 0.02862        
#>    Detection Prevalence : 0.02967        
#>       Balanced Accuracy : 0.93736        
#>                                          
#>        'Positive' Class : Y              
#> 
```

#Calculate the accuracy rate and sensitivity rate with 0.5 Threshold. This means anything with a 0.5 probability or higher will be classified as a Blue Tarp.

```
(18833+543)/(18833+543+77+20)
```

```
#> [1] 0.9950187
```

```
543/(543+77)
```

```
#> [1] 0.8758065
```

#Based on the caret package default threshold of 0.5, the accuracy rate is 99.5% and sensitivity rate is 87.5%. We are more concerned about sensitivity because we want to identify as many blue tarps as possible even if we get some wrongly classified as blue tarps and it turns out that they are not. #Decrease Threshold to 0.1 and calculate a new confusion matrix.

```
predict_threshold<-predict(model_l,newdata=test.data,type="prob")
alteredProb<-predict_threshold$Y
alteredProb<-factor(ifelse(alteredProb>=0.1,"Y","N"))
confusionMatrix(alteredProb,test.data$BlueTarp,positive = "Y")
```

```
#> Confusion Matrix and Statistics
#> 
#>           Reference
#> Prediction     N     Y
#>          N 18200    32
#>          Y   153   588
#> 
#>                Accuracy : 0.9902
#>                  95% CI : (0.9887, 0.9916)
#>     No Information Rate : 0.9673
#>     P-Value [Acc > NIR] : < 2.2e-16
#> 
#>                   Kappa : 0.8591
#> 
#>  Mcnemar's Test P-Value : < 2.2e-16
#> 
#>             Sensitivity : 0.94839
#>             Specificity : 0.99166
#>          Pos Pred Value : 0.79352
#>          Neg Pred Value : 0.99824
#>              Prevalence : 0.03268
#>          Detection Rate : 0.03099
#>    Detection Prevalence : 0.03906
#>       Balanced Accuracy : 0.97003
#> 
#>        'Positive' Class : Y
#> 
```

#The accuracy of the model is still around 99%, but the sensitivity has now gone up to 94.8%. The True negative rate is at 99.17%

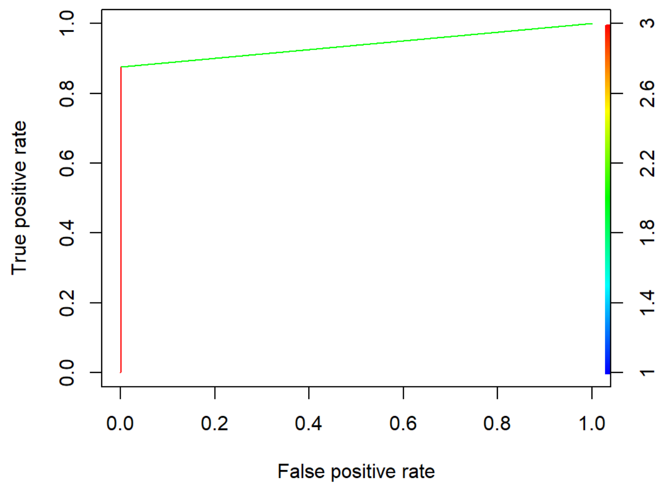
#ROC Curve for Logistic Model #The ROC curve is built from the sample/test data using the model built in the training data.

```
predict0 <- predict(model_l,test.data, type = 'raw')

ROCRpred0 <- prediction(as.numeric(predict0),as.numeric(test.data$BlueTarp))

ROCRperf0<- performance(ROCRpred0, 'tpr', 'fpr')

plot(ROCRperf0, colorize=TRUE, text.adj=c(-0.2,1.7))
```



#AUC for Logistic model #The higher/closer to 1, the better the performance of the model in differentiating between positive and negative classes.

```
auc_ROCR <- performance(ROCRpred0, measure = "auc")
AUC=auc_ROCR@y.values[[1]]
AUC
```

```
#> [1] 0.9373584
```

3.3 LDA

#This model is used to classify patterns between images that are Blue Tarps and those that are not. In this model, we model the distribution of the predictors (red, blue, green) separately in each of the response classes (Y or N for Blue Tarps), and then use Bayes theorem to flip these around into estimates for the probability that a point is a blue tarp given the predictor value.

```
model_lda<-train(BlueTarp~Red+Green+Blue,data=train.data,method="lda",trControl=ctrlspecs)
```

#LDA Confusion Matrix

```
predictions_lda<-predict(model_lda,newdata = test.data)
confusionMatrix(predictions_lda,test.data$BlueTarp,positive = "Y")
```

```
#> Confusion Matrix and Statistics
#> 
#>           Reference
#> Prediction     N     Y
#>          N 18160   137
#>          Y   193   483
#> 
#>                Accuracy : 0.9826
#>                  95% CI : (0.9806, 0.9844)
#>     No Information Rate : 0.9673
#>     P-Value [Acc > NIR] : < 2.2e-16
#> 
#>                   Kappa : 0.7364
#> 
#>  Mcnemar's Test P-Value : 0.002465
#> 
#>             Sensitivity : 0.77903
#>             Specificity : 0.98948
#>          Pos Pred Value : 0.71450
#>          Neg Pred Value : 0.99251
#>              Prevalence : 0.03268
#>          Detection Rate : 0.02546
#>    Detection Prevalence : 0.03563
#>       Balanced Accuracy : 0.88426
#> 
#>        'Positive' Class : Y
#> 
```

#Calculate the accuracy rate and sensitivity rate with 0.5 Threshold

```
(18160+483)/(18160+483+193+137)
```

```
#> [1] 0.9826069
```

```
483/(483+137)
```

```
#> [1] 0.7790323
```

#This gives us an accuracy of 98.2%, but a sensitivity of 77.9%. #Decrease Threshold to 0.1 and calculate a new confusion matrix since we are more concerned about sensitivity.

```
predict_threshold2<-predict(model_lda,newdata=test.data,type="prob")
alteredProb2<-predict_threshold2$Y
alteredProb2<-factor(ifelse(alteredProb2>=0.1,"Y","N"))
confusionMatrix(alteredProb2,test.data$BlueTarp,positive = "Y")
```

```
#> Confusion Matrix and Statistics
#> 
#>           Reference
#> Prediction     N     Y
#>          N 18104   107
#>          Y   249   513
#> 
#>                Accuracy : 0.9812          
#>                  95% CI : (0.9792, 0.9831)
#>     No Information Rate : 0.9673          
#>     P-Value [Acc > NIR] : < 2.2e-16       
#> 
#>                   Kappa : 0.7328          
#> 
#>  Mcnemar's Test P-Value : 7.84e-14        
#> 
#>             Sensitivity : 0.82742         
#>             Specificity : 0.98643         
#>          Pos Pred Value : 0.67323         
#>          Neg Pred Value : 0.99412         
#>              Prevalence : 0.03268         
#>          Detection Rate : 0.02704         
#>    Detection Prevalence : 0.04016         
#>       Balanced Accuracy : 0.90693         
#> 
#>        'Positive' Class : Y               
#> 
```

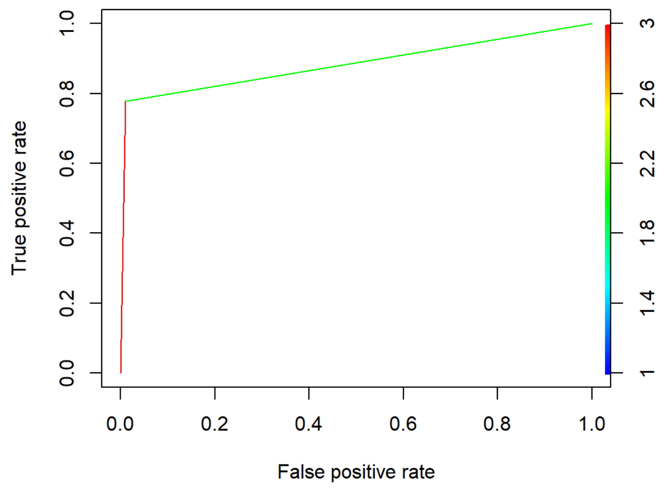
#The accuracy remains high at 98.12%, but sensitivity goes up to 82.7%. The True negative rate is at 98.64% #ROC Curve for LDA

```
predict1 <- predict(model_lda,test.data, type = 'raw')

ROCRpred1 <- prediction(as.numeric(predict1),as.numeric(test.data$BlueTarp))

ROCRperf1<- performance(ROCRpred1, 'tpr', 'fpr')

plot(ROCRperf1, colorize=TRUE, text.adj=c(-0.2,1.7))
```



#AUC for LDA

```
auc_ROCR1 <- performance(ROCRpred1, measure = "auc")
AUC1=auc_ROCR1@y.values[[1]]
AUC1
```

```
#> [1] 0.8842581
```

3.4 QDA

#This is like LDA except it assumes that each class has its own covariance matrix. This is a more flexible model than LDA and can do better when the training data set is large.

```
model_qda<-train(BlueTarp~Red+Green+Blue,data=train.data,method="qda",trControl=ctrlspecs)
```

#QDA Confusion Matrix

```
predictions_qda<-predict(model_qda,newdata = test.data)
confusionMatrix(predictions_qda,test.data$BlueTarp,positive = "Y")
```

```
#> Confusion Matrix and Statistics
#> 
#>           Reference
#> Prediction     N     Y
#>          N 18347   107
#>          Y     6   513
#> 
#>                Accuracy : 0.994
#>                  95% CI : (0.9928, 0.9951)
#>     No Information Rate : 0.9673
#>     P-Value [Acc > NIR] : < 2.2e-16
#> 
#>                   Kappa : 0.8977
#> 
#>  Mcnemar's Test P-Value : < 2.2e-16
#> 
#>             Sensitivity : 0.82742
#>             Specificity : 0.99967
#>          Pos Pred Value : 0.98844
#>          Neg Pred Value : 0.99420
#>              Prevalence : 0.03268
#>          Detection Rate : 0.02704
#>    Detection Prevalence : 0.02735
#>       Balanced Accuracy : 0.91355
#> 
#>        'Positive' Class : Y
#> 
```

#Calculate the accuracy rate and sensitivity rate with 0.5 Threshold

```
(18347+513)/(18347+513+107+6)
```

```
#> [1] 0.9940442
```

```
513/(620)
```

```
#> [1] 0.8274194
```

#This gives us an accuracy of 99.4%, but a sensitivity of 82.7%. #Decrease Threshold to 0.1 and calculate a new confusion matrix since are more concerned about sensitivity.

```
predict_threshold3<-predict(model_qda,newdata=test.data,type="prob")
alteredProb3<-predict_threshold3$Y
alteredProb3<-factor(ifelse(alteredProb3>=0.1,"Y","N"))
confusionMatrix(alteredProb3,test.data$BlueTarp,positive = "Y")
```

```
#> Confusion Matrix and Statistics
#> 
#>           Reference
#> Prediction     N     Y
#>          N 18226    69
#>          Y   127   551
#>                                           
#>                Accuracy : 0.9897          
#>                  95% CI : (0.9881, 0.9911)
#>     No Information Rate : 0.9673          
#>     P-Value [Acc > NIR] : < 2.2e-16       
#>                                           
#>                   Kappa : 0.8437          
#>                                           
#>  Mcnemar's Test P-Value : 4.673e-05       
#>                                           
#>             Sensitivity : 0.88871         
#>             Specificity : 0.99308         
#>          Pos Pred Value : 0.81268         
#>          Neg Pred Value : 0.99623         
#>              Prevalence : 0.03268         
#>          Detection Rate : 0.02904         
#>    Detection Prevalence : 0.03573         
#>       Balanced Accuracy : 0.94089         
#>                                           
#>        'Positive' Class : Y               
#> 
```

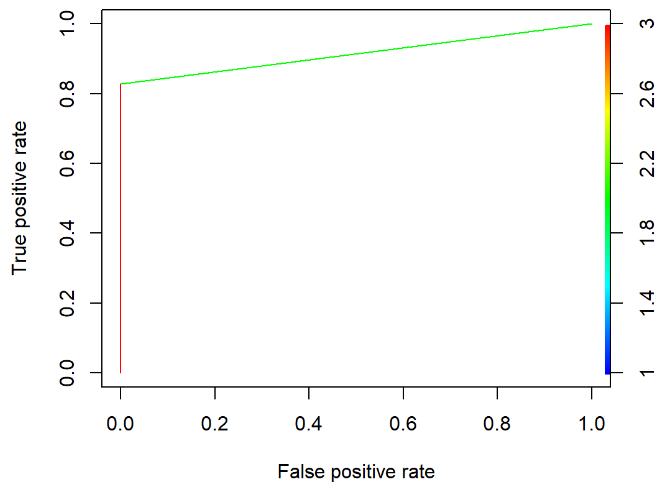
#The Accuracy remains high at 98.97%, but the sensitivity goes up to 88.87%. The True negative rate is at 99.3%. #ROC Curve for QDA Model

```
predict2 <- predict(model_qda,test.data, type = 'raw')

ROCRpred2 <- prediction(as.numeric(predict2),as.numeric(test.data$BlueTarp))

ROCRperf2<- performance(ROCRpred2, 'tpr', 'fpr')

plot(ROCRperf2, colorize=TRUE, text.adj=c(-0.2,1.7))
```



#AUC for QDA

```
auc_ROCR2 <- performance(ROCRpred2, measure = "auc")
AUC2=auc_ROCR2@y.values[[1]]
AUC2
```

```
#> [1] 0.9135462
```

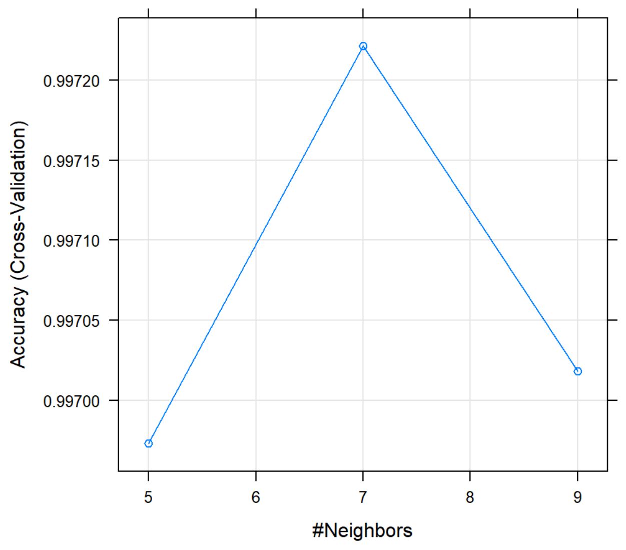
3.5 KNN

#This model makes a prediction based on the points that are closest to it, and uses them to estimate if an image is a Blue Tarp or not. The number of points used to make this determination is K. As we will see below, there are multiple options to choose from when selecting K, such as 5,7, etc.

```
model_knn<-train(BlueTarp~Red+Green+Blue,data=train.data,method="knn",trControl=ctrlspecs)
```

#Tuning Parameter k

```
plot(model_knn)
```



```
model_knn
```

```
#> k-Nearest Neighbors
#> 
#> 44268 samples
#>     3 predictor
#>     2 classes: 'N', 'Y'
#> 
#> No pre-processing
#> Resampling: Cross-Validated (10 fold)
#> Summary of sample sizes: 39841, 39840, 39841, 39841, 39842, 39842, ...
#> Resampling results across tuning parameters:
#> 
#>   k  Accuracy   Kappa
#>   5  0.9969730  0.9507780
#>   7  0.9972215  0.9547270
#>   9  0.9970182  0.9513612
#> 
#> Accuracy was used to select the optimal model using the largest value.
#> The final value used for the model was k = 7.
```

#The value of k used for the KNN model is 7. The plot and table show this produces the highest accuracy rate and it was used in the model.

#KNN Confusion Matrix

```
predictions_knn<-predict(model_knn,newdata = test.data)
confusionMatrix(predictions_knn,test.data$BlueTarp,positive = "Y")
```

```
#> Confusion Matrix and Statistics
#> 
#>           Reference
#> Prediction     N     Y
#>          N 18321    31
#>          Y    32   589
#>                                           
#>                Accuracy : 0.9967          
#>                  95% CI : (0.9958, 0.9974)
#>     No Information Rate : 0.9673          
#>     P-Value [Acc > NIR] : <2e-16          
#>                                           
#>                   Kappa : 0.9475          
#>                                           
#>  Mcnemar's Test P-Value : 1               
#>                                           
#>             Sensitivity : 0.95000         
#>             Specificity : 0.99826         
#>          Pos Pred Value : 0.94847         
#>          Neg Pred Value : 0.99831         
#>              Prevalence : 0.03268         
#>          Detection Rate : 0.03104         
#>    Detection Prevalence : 0.03273         
#>       Balanced Accuracy : 0.97413         
#>                                           
#>        'Positive' Class : Y               
#> 
```

#The KNN has an accuracy of 99.7%, and a sensitivity rate of 95% when k=7. The True negative rate is at 99.8%.

#Decrease Threshold to 0.1 and calculate a new confusion matrix since are more concerned about sensitivity.

```
predict_threshold4<-predict(model_knn,newdata=test.data,type="prob")
alteredProb4<-predict_threshold4$Y
alteredProb4<-factor(ifelse(alteredProb4>=0.1,"Y","N"))
confusionMatrix(alteredProb4,test.data$BlueTarp,positive = "Y")
```

```
#> Confusion Matrix and Statistics
#> 
#>           Reference
#> Prediction     N     Y
#>          N 18280    12
#>          Y    73   608
#> 
#>                Accuracy : 0.9955
#>                  95% CI : (0.9945, 0.9964)
#>     No Information Rate : 0.9673
#>     P-Value [Acc > NIR] : < 2.2e-16
#> 
#>                   Kappa : 0.9324
#> 
#>  Mcnemar's Test P-Value : 7.62e-11
#> 
#>             Sensitivity : 0.98065
#>             Specificity : 0.99602
#>          Pos Pred Value : 0.89280
#>          Neg Pred Value : 0.99934
#>              Prevalence : 0.03268
#>          Detection Rate : 0.03205
#>    Detection Prevalence : 0.03589
#>       Balanced Accuracy : 0.98833
#> 
#>        'Positive' Class : Y
#> 
```

#The Accuracy remains high at 99.55%, but the sensitivity goes up to 98.07%. The True negative rate is at 99.6%.

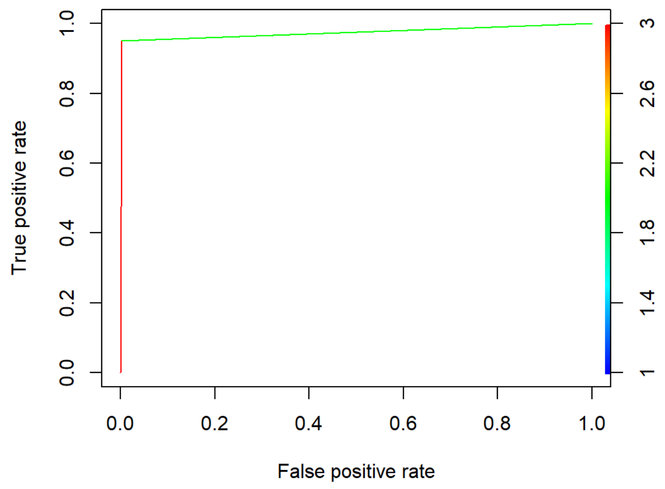
#ROC Curve for KNN Model

```
predict3 <- predict(model_knn,test.data, type = 'raw')

ROCRpred3 <- prediction(as.numeric(predict3),as.numeric(test.data$BlueTarp))

ROCRperf3<- performance(ROCRpred3, 'tpr', 'fpr')

plot(ROCRperf3, colorize=TRUE, text.adj=c(-0.2,1.7))
```



#AUC for KNN

```
auc_ROCR3 <- performance(ROCRpred3, measure = "auc")
AUC3=auc_ROCR3@y.values[[1]]
AUC3
```

```
#> [1] 0.9741282
```

3.6 Penalized Logistic Regression (Ridge)

#The Penalized Logistic Regression

```
x.train=model.matrix(BlueTarp~-1+Red+Blue+Green,data=train.data)
y.train=train.data$BlueTarp
x.test=model.matrix(BlueTarp~-1+Red+Blue+Green,data=test.data)
y.test=test.data$BlueTarp
ridge.fit <- cv.glmnet(x.train, y.train, type.measure="class",
  alpha=0, family="binomial",nlambda=100)
```

```
ridge.predicted <- predict(ridge.fit, s=ridge.fit$lambda.min, newx=x.test)
```

3.7 Penalized Logistic Regression (lasso)

```
lasso.fit <- cv.glmnet(x.train, y.train, type.measure="class",
  alpha=1, family="binomial",nlambda=100)
```

```
lasso.predicted <- predict(lasso.fit, s=lasso.fit$lambda.min, newx=x.test)
```

#Calculate the accuracy rate and sensitivity rate with 0.1 Threshold using lasso model

```
predict_thresholdls<-predict(lasso.fit, s=lasso.fit$lambda.min, newx=x.test)
alteredProbls<-predict_thresholdls[,1]
alteredProbls<-factor(ifelse(alteredProbls>=0.1,"Y","N"))
confusionMatrix(alteredProbls,test.data$BlueTarp,positive = "Y")
```

```
#> Confusion Matrix and Statistics
#> 
#>           Reference
#> Prediction     N     Y
#>          N 18335    78
#>          Y    18   542
#> 
#>                Accuracy : 0.9949
#>                  95% CI : (0.9938, 0.9959)
#>     No Information Rate : 0.9673
#>     P-Value [Acc > NIR] : < 2.2e-16
#> 
#>                   Kappa : 0.916
#> 
#>  Mcnemar's Test P-Value : 1.726e-09
#> 
#>             Sensitivity : 0.87419
#>             Specificity : 0.99902
#>          Pos Pred Value : 0.96786
#>          Neg Pred Value : 0.99576
#>              Prevalence : 0.03268
#>          Detection Rate : 0.02857
#>    Detection Prevalence : 0.02952
#>       Balanced Accuracy : 0.93661
#> 
#>        'Positive' Class : Y
#> 
```

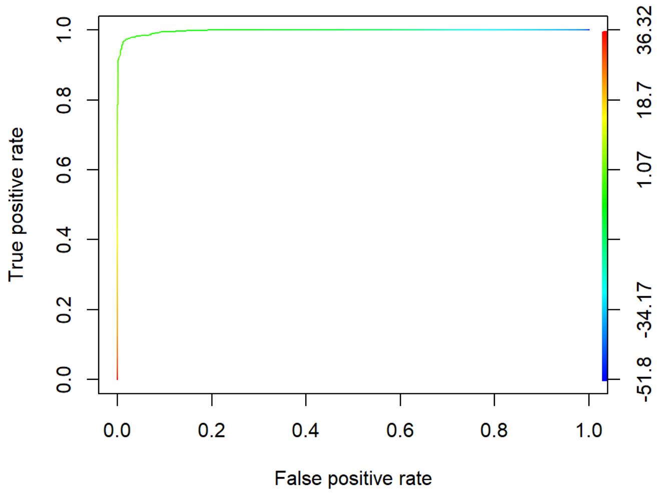
#This shows a sensitivity rate of 87.4% and overall accuracy of 99.49% #ROC Curve and AUROC for Lasso Model

```
predictls0 <- predict(lasso.fit, s=lasso.fit$lambda.min, newx=x.test)

ROCRpredls0 <- prediction(as.numeric(predictls0),as.numeric(test.data$BlueTarp))

ROCRperfls0<- performance(ROCRpredls0, 'tpr', 'fpr')

plot(ROCRperfls0, colorize=TRUE, text.adj=c(-0.2,1.7))
```



```
auc_ROCRls0 <- performance(ROCRpredls0, measure = "auc")
AUCls0=auc_ROCRls@y.values[[1]]
```

```
#> Error in eval(expr, envir, enclos): object 'auc_ROCRls' not found
```

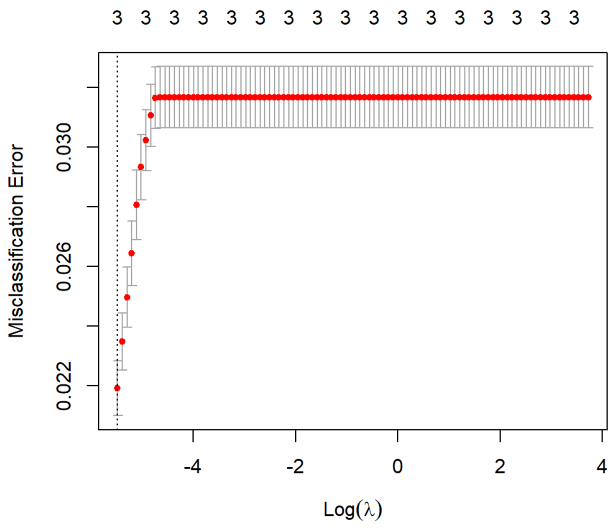
```
AUCls0
```

```
#> Error in eval(expr, envir, enclos): object 'AUCls0' not found
```

3.7.1 Tuning Parameters

#Tuning Parameter Ridge

```
plot(ridge.fit)
```



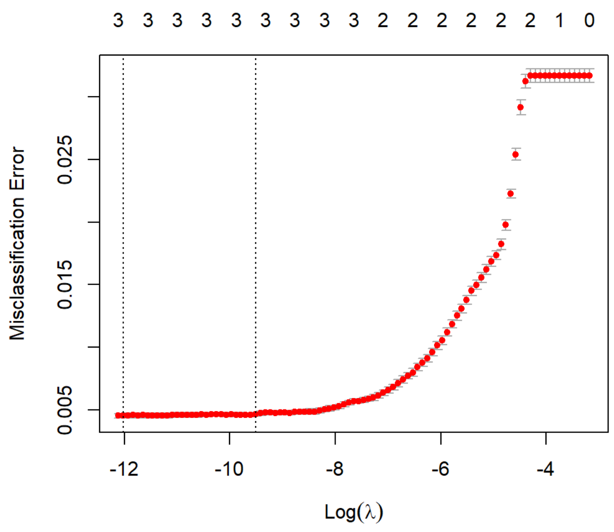
```
log(ridge.fit$lambda.min)
```

```
#> [1] -5.483995
```

###The plot shows that the minimum value of lamda is -5.483995, with a misclassification rate of 0.022. Perform Lasso tuning in order to determine if the misclassification rate can be lower as the ridge did not give a significant improvement.

#Tuning Parameter Lasso

```
plot(lasso.fit)
```



```
log(lasso.fit$lambda.min)
```

```
#> [1] -12.01962
```

###The minimum lambda in lasso is -12.02 with a misclassification rate of 0.005. The lasso reduces the misclassification rate and improves the accuracy more than the ridge regression does.

3.8 Threshold Selection

#Initially the models were run with the default threshold of 0.5, which would classify any data point with an estimated probability of 0.5 or above as a Blue Tarp. All the models had high accuracy levels with this threshold, but I felt it was more important to reduce this threshold so that even if the probability of an image being a Blue Tarp was lower, it would get picked up. In order words, I was more concerned with having the sensitivity rate higher so we could reach as many people as possible even if some were not Blue Tarps as long as the model still maintained a high accuracy. I reduced the threshold down to 0.1 and this still resulted in high accuracy rates for the models, while bringing up the sensitivity rates. This approach was carried out for all models and resulted in using the 0.1 threshold for all models that required a threshold setting.

4 Results (Cross-Validation)

** CV Performance Table **

```
Model<-c("Log Reg","LDA", "QDA", "KNN", "Penalized Log Reg")
Tuning<-c("-","-","-","k=7","lambda=-12.02, alpha=1")
AUROC<-c("0.937","0.884","0.9135","0.974","0.999")
Threshold<-c("0.1","0.1","0.1","0.1","0.1")
Accuracy<-c("0.9902","0.9812","0.9897","0.9955","0.995")
TPR<-c("0.948","0.82742","0.889","0.9806","0.874")
FPR<-c("0.008","0.014","0.007","0.004","0.001")
Precision<-c("0.7935","0.6732","0.81268","0.8928","0.9695")
CV.Performance<-data.frame(Model,Tuning,AUROC,Threshold,Accuracy,TPR,FPR,Precision)
knitr::kable(CV.Performance,"pipe")
```

Model	Tuning	AUROC	Threshold	Accuracy	TPR	FPR	Precision
Log Reg	-	0.937	0.1	0.9902	0.948	0.008	0.7935
LDA	-	0.884	0.1	0.9812	0.82742	0.014	0.6732
QDA	-	0.9135	0.1	0.9897	0.889	0.007	0.81268
KNN	k=7	0.974	0.1	0.9955	0.9806	0.004	0.8928
Penalized Log Reg	lambda=-12.02, alpha=1	0.999	0.1	0.995	0.874	0.001	0.9695

```
CV.Performance
```

```
#>               Model                 Tuning  AUROC Threshold Accuracy     TPR
#> 1           Log Reg                      -  0.937       0.1   0.9902   0.948
#> 2               LDA                      -  0.884       0.1   0.9812 0.82742
#> 3               QDA                      - 0.9135       0.1   0.9897   0.889
#> 4               KNN                    k=7  0.974       0.1   0.9955  0.9806
#> 5 Penalized Log Reg lambda=-12.02, alpha=1  0.999       0.1    0.995   0.874
#>     FPR Precision
#> 1 0.008    0.7935
#> 2 0.014    0.6732
#> 3 0.007   0.81268
#> 4 0.004    0.8928
#> 5 0.001    0.9695
```

#These metrics were calculated using the testing data set, which is 30% of the entire data set that was set up at the start. The models were built using a 10 -fold cross validation on the training dataset. These models were then used for prediction on the testing data.

5 Conclusions

5.0.1 Conclusion #1

#The 5 models present very high accuracy rates of over 98%. The range is a little wider for the sensitivity rates, and the reason sensitivity rate is an important factor in this situation is because we are concerned about reaching as many people as possible even though if we get some false positives in the process as long as it doesn't decrease the accuracy rates by much. I reduced the thresholds to 0.1 as described in the above section on thresholds. From these models, I would select the KNN model as this gives us a high sensitivity rate of 98.06% with an accuracy rate of 99.55% and a false positive rate of 0.4%. The K in the KNN model that produced the best results based on accuracy was 7. This model also has the highest AUROC and precision rates.

5.0.2 Conclusion #2

#Given that KNN model did very well in this dataset, it appears that the dataset is non-parametric i.e it is an approach where no decisions are made about the shape of the boundary. In other words, it is very flexible. This is why it performs better than the LDA model especially when we look at precision and TPR rates. This is because LDA assumes a linear decision boundary but in this instance the decision boundary seems to be highly non-linear.

5.0.3 Conclusion #3

#I believe the work here is helpful in saving human lives. Given the high rates of accuracy and sensitivity, I think these models bring in objectiveness in selecting an image that is a Blue Tarp and hence where aid can be delivered. It not only brings in objectiveness to the decision making, but also speed and precision to making the decision which is much harder to match with a manual process using the human eye. It would be much harder to match a 98% accuracy rate just with the human eye and not to mention the time it would take to manually go through each image. Such time and precision in a disaster relief makes all the difference in the world, which is why such algorithms/models should be implemented in these scenarios.

NOTE: PART II same as above plus add Random Forest and SVM to Model Training.

#Part 2

6 Random Forests

#This is a tree-based ensemble model, whereby when each split is created in the tree, the variables selected are from a random subset of the total available features. This leads to a greater decorrelation of the trees and can produce very robust models. So, in this case the features would be the RGB colors and each split will have a random subset of this.

#Built the random forest with number of variables as the tuning parameter.

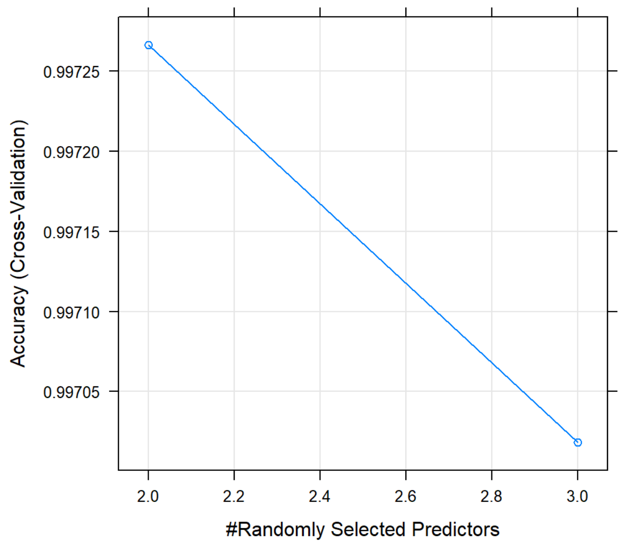
```
set.seed(1)
model_rf<-train(BlueTarp~Red+Green+Blue,data=train.data,method="rf",trControl=ctrlspecs,tuneGrid=e
         xpand.grid(mtry=c(2,3)))
```

#Print and plot the random forest model to show accuracy by number of variables (mtry)

```
print(model_rf)
```

```
#> Random Forest
#>
#> 44268 samples
#>     3 predictor
#>     2 classes: 'N', 'Y'
#>
#> No pre-processing
#> Resampling: Cross-Validated (10 fold)
#> Summary of sample sizes: 39842, 39840, 39842, 39841, 39841, 39842, ...
#> Resampling results across tuning parameters:
#>
#>   mtry  Accuracy   Kappa
#>   2     0.9972666  0.9550384
#>   3     0.9970181  0.9511920
#>
#> Accuracy was used to select the optimal model using the largest value.
#> The final value used for the model was mtry = 2.
```

```
plot(model_rf)
```



#This shows that the model with 2 splits at each node is more accurate and will produce a more optimal model than with 3 splits. Note we cannot do more splits given that we only have 3 variables (RGB).

#Show summary of final random forest model

```
model_rf$finalModel
```

```
#>
#> Call:
#>  randomForest(x = x, y = y, mtry = min(param$mtry, ncol(x)))
#>                Type of random forest: classification
#>                      Number of trees: 500
#> No. of variables tried at each split: 2
#>
#>         OOB estimate of  error rate: 0.29%
#> Confusion matrix:
#>       N    Y class.error
#> N 42815   51 0.001189754
#> Y    77 1325 0.054921541
```

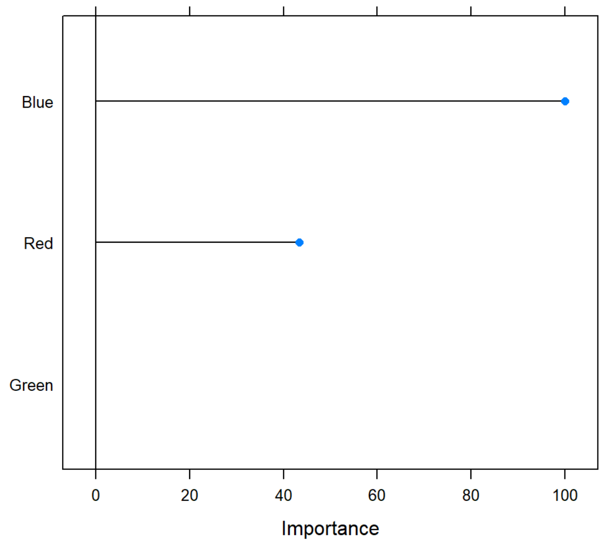
#The final random forest model has 500 trees with mtry of 2 and this yields an error estimate of 0.29% on the training data.

#Determine the most important variables in the random forest models

```
print(varImp(model_rf))
```

```
#> rf variable importance
#>
#>       Overall
#> Blue   100.00
#> Red     43.33
#> Green    0.00
```

```
plot(varImp(model_rf))
```



#As can be seen from the table and plot, the most important variable is Blue followed by Red. This makes sense since Blue would be the color determine if an image contains a Blue tarp. Red is also important in ruling out Blue Tarps given how distinct it is from the color Blue. Green is the least important variable.

#RF Confusion Matrix on test data (30% of original set). This is similar to what was done with the initial models.

```
predictions_rf<-predict(model_rf,newdata = test.data)
confusionMatrix(predictions_rf,test.data$BlueTarp,positive = "Y")
```

```
#> Confusion Matrix and Statistics
#> 
#>           Reference
#> Prediction     N     Y
#>          N 18332    43
#>          Y    21   577
#> 
#>                Accuracy : 0.9966
#>                  95% CI : (0.9957, 0.9974)
#>     No Information Rate : 0.9673
#>     P-Value [Acc > NIR] : < 2.2e-16
#> 
#>                   Kappa : 0.9457
#> 
#>  Mcnemar's Test P-Value : 0.008665
#> 
#>             Sensitivity : 0.93065
#>             Specificity : 0.99886
#>          Pos Pred Value : 0.96488
#>          Neg Pred Value : 0.99766
#>              Prevalence : 0.03268
#>          Detection Rate : 0.03041
#>    Detection Prevalence : 0.03152
#>       Balanced Accuracy : 0.96475
#> 
#>        'Positive' Class : Y
#> 
```

#This shows that a 0.5 threshold the accuracy is 99.66%, and the sensitivity is 93%.

#Decrease Threshold to 0.1 and calculate a new confusion matrix since are more concerned about sensitivity.

```
predict_threshold5<-predict(model_rf,newdata=test.data,type="prob")
alteredProb5<-predict_threshold5$Y
alteredProb5<-factor(ifelse(alteredProb5>=0.1,"Y","N"))
confusionMatrix(alteredProb5,test.data$BlueTarp,positive = "Y")
```

```
#> Confusion Matrix and Statistics
#> 
#>           Reference
#> Prediction     N     Y
#>          N 18279    11
#>          Y    74   609
#> 
#>                Accuracy : 0.9955
#>                  95% CI : (0.9945, 0.9964)
#>     No Information Rate : 0.9673
#>     P-Value [Acc > NIR] : < 2.2e-16
#> 
#>                   Kappa : 0.9325
#> 
#>  Mcnemar's Test P-Value : 1.758e-11
#> 
#>             Sensitivity : 0.98226
#>             Specificity : 0.99597
#>          Pos Pred Value : 0.89165
#>          Neg Pred Value : 0.99940
#>              Prevalence : 0.03268
#>          Detection Rate : 0.03210
#>    Detection Prevalence : 0.03600
#>       Balanced Accuracy : 0.98911
#> 
#>        'Positive' Class : Y
#> 
```

#Based on a threshold of 0.1, the accuracy remains high at 99.55%, and the sensitivity goes up to 98.226%

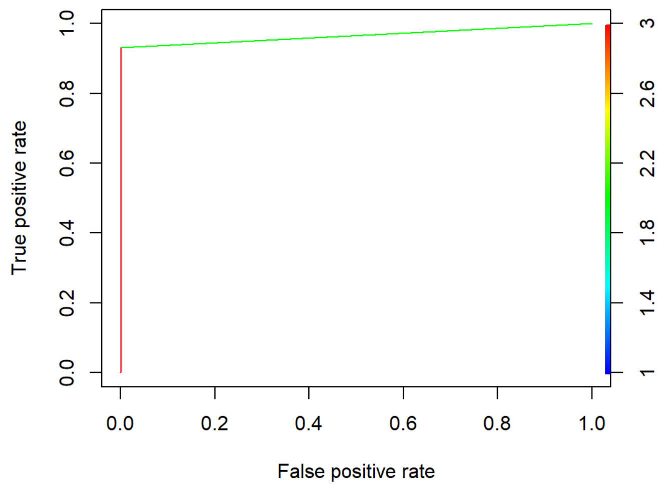
#ROC Curve for RF Model

```
predictrf <- predict(model_rf,test.data, type = 'raw')

ROCRpredrf <- prediction(as.numeric(predictrf),as.numeric(test.data$BlueTarp))

ROCRperfrf<- performance(ROCRpredrf, 'tpr', 'fpr')

plot(ROCRperfrf, colorize=TRUE, text.adj=c(-0.2,1.7))
```



#AUC for Random forest #The higher/closer to 1, the better the performance of the model in differentiating between positive and negative classes.

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

```
#> [1] 0.9647505
```

7 SVM

#This model uses a function to put the data in a high dimensional space and uses a support vector classifier in this higher dimensional space to make the classifications. The Kernel handles the mapping into the higher dimensional space. We will examine Linear, Polynomial, and Radial kernels for building the SVM model.

#Run an SVM with Linear kernal using 10 fold CV with costs ranging from 0.5 to 5 to find the optimal model. Higher cost mean fewer observations will violate the margin, so this would be more flexible but with a risk of overtraining.

```
set.seed(1)
model_svm_linear<- train(BlueTarp~ Red+Green+Blue, data=train.data, method = "svmLinear", trContro
         l = ctrlspecs,tuneGrid=expand.grid(C=seq(0.5,5,0.5)))
```

#show the tuning parameter and final SVM linear model from 10 fold CV.

```
model_svm_linear$bestTune
```

```
#>     C
#> 5 2.5
```

```
model_svm_linear
```

```
#> Support Vector Machines with Linear Kernel
#> 
#> 44268 samples
#>     3 predictor
#>     2 classes: 'N', 'Y'
#> 
#> No pre-processing
#> Resampling: Cross-Validated (10 fold)
#> Summary of sample sizes: 39842, 39840, 39842, 39841, 39841, 39842, ...
#> Resampling results across tuning parameters:
#> 
#>   C    Accuracy   Kappa
#>   0.5  0.9954368  0.9223763
#>   1.0  0.9954594  0.9229287
#>   1.5  0.9955272  0.9242003
#>   2.0  0.9955498  0.9246049
#>   2.5  0.9955724  0.9250096
#>   3.0  0.9954594  0.9229518
#>   3.5  0.9954820  0.9233507
#>   4.0  0.9954369  0.9224641
#>   4.5  0.9954594  0.9229004
#>   5.0  0.9954143  0.9220245
#> 
#> Accuracy was used to select the optimal model using the largest value.
#> The final value used for the model was C = 2.5.
```

#Based on the accuracy, the cost of 2.5 produces the optimal model with an accuracy of 0.9955.

#Training error with Optimized SVM Linear Kernel model

```
predictions_svlinear<-predict(model_svm_linear,train.data)
confusionMatrix(predictions_svlinear,train.data$BlueTarp,positive = "Y")
```

```
#> Confusion Matrix and Statistics
#> 
#>           Reference
#> Prediction     N     Y
#>          N 42820   153
#>          Y    46  1249
#> 
#>                Accuracy : 0.9955          
#>                  95% CI : (0.9948, 0.9961)
#>     No Information Rate : 0.9683          
#>     P-Value [Acc > NIR] : < 2.2e-16       
#> 
#>                   Kappa : 0.9239          
#> 
#>  Mcnemar's Test P-Value : 5.729e-14       
#> 
#>             Sensitivity : 0.89087         
#>             Specificity : 0.99893         
#>          Pos Pred Value : 0.96448         
#>          Neg Pred Value : 0.99644         
#>              Prevalence : 0.03167         
#>          Detection Rate : 0.02821         
#>    Detection Prevalence : 0.02925         
#>       Balanced Accuracy : 0.94490         
#> 
#>        'Positive' Class : Y               
#> 
```

#SVM Linear Confusion Matrix on test data (30% of original set)

```
predictions_svlinear<-predict(model_svm_linear,newdata = test.data)
confusionMatrix(predictions_svlinear,test.data$BlueTarp,positive = "Y")
```

```
#> Confusion Matrix and Statistics
#> 
#>           Reference
#> Prediction     N     Y
#>          N 18334    74
#>          Y    19   546
#> 
#>                Accuracy : 0.9951
#>                  95% CI : (0.994, 0.996)
#>     No Information Rate : 0.9673
#>     P-Value [Acc > NIR] : < 2.2e-16
#> 
#>                   Kappa : 0.919
#> 
#>  Mcnemar's Test P-Value : 2.149e-08
#> 
#>             Sensitivity : 0.88065
#>             Specificity : 0.99896
#>          Pos Pred Value : 0.96637
#>          Neg Pred Value : 0.99598
#>              Prevalence : 0.03268
#>          Detection Rate : 0.02878
#>    Detection Prevalence : 0.02978
#>       Balanced Accuracy : 0.93980
#> 
#>        'Positive' Class : Y
#> 
```

#The accuracy rate 99.5% and the sensitivity is 88% using this model.

#Decrease Threshold to 0.1 and calculate a new confusion matrix since are more concerned about sensitivity.

```
predict_threshold6<-predict(model_svm_linear,newdata=test.data,type="prob")
alteredProb6<-factor(ifelse(predict_threshold6$Y>=0.1,"Y","N"))
confusionMatrix(alteredProb6,test.data$BlueTarp,positive = "Y")
```

```
#> Confusion Matrix and Statistics
#> 
#>           Reference
#> Prediction     N     Y
#>          N 18266    39
#>          Y    87   581
#>                                           
#>                Accuracy : 0.9934          
#>                  95% CI : (0.9921, 0.9945)
#>     No Information Rate : 0.9673          
#>     P-Value [Acc > NIR] : < 2.2e-16       
#>                                           
#>                   Kappa : 0.8987          
#>                                           
#>  Mcnemar's Test P-Value : 2.826e-05       
#>                                           
#>             Sensitivity : 0.93710         
#>             Specificity : 0.99526         
#>          Pos Pred Value : 0.86976         
#>          Neg Pred Value : 0.99787         
#>              Prevalence : 0.03268         
#>          Detection Rate : 0.03062         
#>    Detection Prevalence : 0.03521         
#>       Balanced Accuracy : 0.96618         
#>                                           
#>        'Positive' Class : Y               
#> 
```

#The accuracy remains high at 0.993, and the sensitivity goes up to 0.937.

#Run an SVM with radial kernel using 10 fold CV with costs ranging from 1 to 5 to find the optimal model. Tune gamma from 0.5 to 2. Gamma defines how far the influence of a single training point reaches (low values are far reaching and high values are close reaching). With high values of gamma the model is more flexible and could have high variance.

```
set.seed(1)
model_svm_radial<- train(BlueTarp~ Red+Green+Blue, data=train.data, method = "svmRadial", trContro
         l = ctrlspecs,tuneGrid=expand.grid(C=seq(1,5,1),sigma=seq(0.5,2,0.5)))
```

#show the tuning parameter and final SVM Radial model from 10 fold CV.

```
model_svm_radial$bestTune
```

```
#>    sigma C
#> 20     2 5
```

```
model_svm_radial
```

```
#> Support Vector Machines with Radial Basis Function Kernel
#> 
#> 44268 samples
#>     3 predictor
#>     2 classes: 'N', 'Y' 
#> 
#> No pre-processing
#> Resampling: Cross-Validated (10 fold) 
#> Summary of sample sizes: 39842, 39840, 39842, 39841, 39841, 39842, ... 
#> Resampling results across tuning parameters:
#> 
#>   C  sigma  Accuracy   Kappa    
#>   1  0.5    0.9959790  0.9340455
#>   1  1.0    0.9962953  0.9392134
#>   1  1.5    0.9964535  0.9416319
#>   1  2.0    0.9966567  0.9447285
#>   2  0.5    0.9962501  0.9384431
#>   2  1.0    0.9963405  0.9398641
#>   2  1.5    0.9966342  0.9445580
#>   2  2.0    0.9969052  0.9488823
#>   3  0.5    0.9962953  0.9392369
#>   3  1.0    0.9964308  0.9412556
#>   3  1.5    0.9968149  0.9474831
#>   3  2.0    0.9969052  0.9489294
#>   4  0.5    0.9962953  0.9391563
#>   4  1.0    0.9965438  0.9430006
#>   4  1.5    0.9968826  0.9485698
#>   4  2.0    0.9969956  0.9504042
#>   5  0.5    0.9963179  0.9395579
#>   5  1.0    0.9966342  0.9445560
#>   5  1.5    0.9968600  0.9482117
#>   5  2.0    0.9970182  0.9508213
#> 
#> Accuracy was used to select the optimal model using the largest value.
#> The final values used for the model were sigma = 2 and C = 5.
```

#The model with a cost of 5 and Gamma of 2 produced the highest accuracy of 0.997.

#Training error with Optimized SVM Radial Kernel model

```
predictions_svmrad<-predict(model_svm_radial,train.data)
confusionMatrix(predictions_svmrad,train.data$BlueTarp,positive = "Y")
```

```
#> Confusion Matrix and Statistics
#> 
#>           Reference
#> Prediction     N     Y
#>          N 42820    72
#>          Y    46  1330
#> 
#>                Accuracy : 0.9973
#>                  95% CI : (0.9968, 0.9978)
#>     No Information Rate : 0.9683
#>     P-Value [Acc > NIR] : < 2e-16
#> 
#>                   Kappa : 0.9561
#> 
#>  Mcnemar's Test P-Value : 0.02137
#> 
#>             Sensitivity : 0.94864
#>             Specificity : 0.99893
#>          Pos Pred Value : 0.96657
#>          Neg Pred Value : 0.99832
#>              Prevalence : 0.03167
#>          Detection Rate : 0.03004
#>    Detection Prevalence : 0.03108
#>       Balanced Accuracy : 0.97379
#> 
#>        'Positive' Class : Y
#> 
```

#SVM Radial Confusion Matrix on test data (30% of original set)

```
predictions_svmrad<-predict(model_svm_radial,newdata = test.data)
confusionMatrix(predictions_svmrad,test.data$BlueTarp,positive = "Y")
```

```
#> Confusion Matrix and Statistics
#> 
#>           Reference
#> Prediction     N     Y
#>          N 18327    44
#>          Y    26   576
#> 
#>                Accuracy : 0.9963
#>                  95% CI : (0.9953, 0.9971)
#>     No Information Rate : 0.9673
#>     P-Value [Acc > NIR] : < 2e-16
#> 
#>                   Kappa : 0.9408
#> 
#>  Mcnemar's Test P-Value : 0.04216
#> 
#>             Sensitivity : 0.92903
#>             Specificity : 0.99858
#>          Pos Pred Value : 0.95681
#>          Neg Pred Value : 0.99760
#>              Prevalence : 0.03268
#>          Detection Rate : 0.03036
#>    Detection Prevalence : 0.03173
#>       Balanced Accuracy : 0.96381
#> 
#>        'Positive' Class : Y
#> 
```

#The accuracy is 0.9963 and sensitivity is 0.92903 with a 0.5 threshold.

#Decrease Threshold to 0.1 and calculate a new confusion matrix since are more concerned about sensitivity.

```
predict_threshold7<-predict(model_svm_radial,newdata=test.data,type="prob")
alteredProb7<-factor(ifelse(predict_threshold7$Y>=0.1,"Y","N"))
confusionMatrix(alteredProb7,test.data$BlueTarp,positive = "Y")
```

```
#> Confusion Matrix and Statistics
#> 
#>           Reference
#> Prediction     N     Y
#>          N 18308    20
#>          Y    45   600
#> 
#>                Accuracy : 0.9966
#>                  95% CI : (0.9956, 0.9974)
#>     No Information Rate : 0.9673
#>     P-Value [Acc > NIR] : < 2.2e-16
#> 
#>                   Kappa : 0.9468
#> 
#>  Mcnemar's Test P-Value : 0.002912
#> 
#>             Sensitivity : 0.96774
#>             Specificity : 0.99755
#>          Pos Pred Value : 0.93023
#>          Neg Pred Value : 0.99891
#>              Prevalence : 0.03268
#>          Detection Rate : 0.03162
#>    Detection Prevalence : 0.03400
#>       Balanced Accuracy : 0.98265
#> 
#>        'Positive' Class : Y
#> 
```

#The accuracy remains high at 0.9966, but sensitivity goes up to 0.96744 with a threshold of 0.1.

#Run an SVM with polinomial kernal using 10 fold CV with costs ranging from 1 to 5 to find the optimal model.

```
set.seed(1)
model_svm_poly<- train(BlueTarp~ Red+Green+Blue, data=train.data, method = "svmPoly", trControl =
          ctrlspecs,tuneGrid=expand.grid(C=seq(1,5,1),degree=seq(2,10,1),scale=1))
```

```
#> line search fails -0.419145 1.275405 2.772567e-05 -3.671232e-06 -5.899181e-09 8.90312e-10 -1.66
8273e-13
```

```
#> Warning in method$predict(modelFit = modelFit, newdata = newdata, submodels =
#> param): kernlab class prediction calculations failed; returning NAs
```

```
#> Warning in method$prob(modelFit = modelFit, newdata = newdata, submodels =
#> param): kernlab class probability calculations failed; returning NAs
```

```
#> Warning in data.frame(..., check.names = FALSE): row names were found from a
#> short variable and have been discarded
#> line search fails -0.5468251 1.258535 1.370732e-05 -2.069137e-06 -4.637905e-09 1.257097e-09 -6.
617435e-14
```

```
#> Warning in method$predict(modelFit = modelFit, newdata = newdata, submodels =
#> param): kernlab class prediction calculations failed; returning NAs
```

```
#> Warning in method$prob(modelFit = modelFit, newdata = newdata, submodels =
#> param): kernlab class probability calculations failed; returning NAs
```

```
#> Warning in data.frame(..., check.names = FALSE): row names were found from a
#> short variable and have been discarded
```

```
#> line search fails -0.5406041 1.265553 1.221677e-05 -1.536578e-06 -3.774665e-09 -1.871293e-10 -
4.582666e-14
```

```
#> Warning in method$predict(modelFit = modelFit, newdata = newdata, submodels =
#> param): kernlab class prediction calculations failed; returning NAs
```

```
#> Warning in method$prob(modelFit = modelFit, newdata = newdata, submodels =
#> param): kernlab class probability calculations failed; returning NAs
```

```
#> Warning in data.frame(..., check.names = FALSE): row names were found from a
#> short variable and have been discarded
```

```
#> line search fails -0.5026347 1.168369 2.316626e-05 -3.5005e-06 -6.334456e-09 2.098648e-09 -1.54
092e-13
```

```
#> Warning in method$predict(modelFit = modelFit, newdata = newdata, submodels =
#> param): kernlab class prediction calculations failed; returning NAs
```

```
#> Warning in method$prob(modelFit = modelFit, newdata = newdata, submodels =
#> param): kernlab class probability calculations failed; returning NAs
```

```
#> Warning in data.frame(..., check.names = FALSE): row names were found from a
#> short variable and have been discarded
```

```
#> line search fails -0.6473826 1.0397 1.624187e-05 -2.846766e-06 -6.494904e-09 2.663817e-09 -1.13
0726e-13
```

```
#> Warning in method$predict(modelFit = modelFit, newdata = newdata, submodels =
#> param): kernlab class prediction calculations failed; returning NAs
#> Warning in method$prob(modelFit = modelFit, newdata = newdata, submodels =
#> param): kernlab class probability calculations failed; returning NAs
```

```
#> Warning in data.frame(..., check.names = FALSE): row names were found from a
#> short variable and have been discarded
```

```
#> line search fails -0.4310193 1.492083 3.191922e-05 -2.688709e-06 -7.667049e-09 -3.902584e-09 -
2.342333e-13
```

```
#> Warning in method$predict(modelFit = modelFit, newdata = newdata, submodels =
#> param): kernlab class prediction calculations failed; returning NAs
```

```
#> Warning in method$prob(modelFit = modelFit, newdata = newdata, submodels =
#> param): kernlab class probability calculations failed; returning NAs
```

```
#> Warning in data.frame(..., check.names = FALSE): row names were found from a
#> short variable and have been discarded
```

```
#> line search fails -0.3959643 1.281632 3.734713e-05 -4.311983e-06 -7.166498e-09 3.111938e-10 -2.
6899e-13
```

```
#> Warning in method$predict(modelFit = modelFit, newdata = newdata, submodels =
#> param): kernlab class prediction calculations failed; returning NAs
```

```
#> Warning in method$prob(modelFit = modelFit, newdata = newdata, submodels =
#> param): kernlab class probability calculations failed; returning NAs
```

```
#> Warning in data.frame(..., check.names = FALSE): row names were found from a
#> short variable and have been discarded
```

```
#> line search fails -0.4523165 1.388406 3.30598e-05 -4.5016e-06 -8.607587e-09 1.171353e-10 -2.850
924e-13
```

```
#> Warning in method$predict(modelFit = modelFit, newdata = newdata, submodels =
#> param): kernlab class prediction calculations failed; returning NAs
```

```
#> Warning in method$prob(modelFit = modelFit, newdata = newdata, submodels =
#> param): kernlab class probability calculations failed; returning NAs
```

```
#> Warning in data.frame(..., check.names = FALSE): row names were found from a
#> short variable and have been discarded
#> Warning in nominalTrainWorkflow(x = x, y = y, wts = weights, info = trainInfo, :
#> There were missing values in resampled performance measures.
```

#show the tuning parameter and final SVM Polynomial model from 10 fold CV.

```
model_svm_poly$bestTune
```

```
#>    degree scale C
#> 41      6     1 5
```

```
model_svm_poly
```

```
#> Support Vector Machines with Polynomial Kernel
#> 
#> 44268 samples
#>     3 predictor
#>     2 classes: 'N', 'Y'
#> 
#> No pre-processing
#> Resampling: Cross-Validated (10 fold)
#> Summary of sample sizes: 39842, 39840, 39842, 39841, 39841, 39842, ...
#> Resampling results across tuning parameters:
#> 
#>   C  degree  Accuracy   Kappa
#>   1   2      0.9958435  0.9308077
#>   1   3      0.9965889  0.9433617
#>   1   4      0.9966793  0.9448995
#>   1   5      0.9969955  0.9501167
#>   1   6      0.9972892  0.9549141
#>   1   7      0.9972440  0.9539321
#>   1   8      0.9964308  0.9380674
#>   1   9      0.9956175  0.9234302
#>   1  10      0.9926131  0.8625224
#>   2   2      0.9958887  0.9314598
#>   2   3      0.9965889  0.9434019
#>   2   4      0.9968600  0.9480956
#>   2   5      0.9971762  0.9532411
#>   2   6      0.9972892  0.9548665
#>   2   7      0.9972892  0.9544990
#>   2   8      0.9964759  0.9397690
#>   2   9      0.9945785  0.9028693
#>   2  10      0.9910092  0.8297053
#>   3   2      0.9958435  0.9305303
#>   3   3      0.9967019  0.9452339
#>   3   4      0.9969730  0.9499691
#>   3   5      0.9971762  0.9532753
#>   3   6      0.9972440  0.9542668
#>   3   7      0.9971537  0.9524287
#>   3   8      0.9958084  0.9266967
#>   3   9      0.9931552  0.8738324
#>   3  10      0.9908386  0.8234085
#>   4   2      0.9959113  0.9316749
#>   4   3      0.9967922  0.9467328
#>   4   4      0.9969956  0.9503683
#>   4   5      0.9972214  0.9540046
#>   4   6      0.9971988  0.9535510
#>   4   7      0.9972440  0.9537264
#>   4   8      0.9957305  0.9253623
#>   4   9      0.9931978  0.8752302
#>   4  10      0.9902300  0.8122697
#>   5   2      0.9958661  0.9309541
#>   5   3      0.9968374  0.9475689
#>   5   4      0.9970859  0.9519412
#>   5   5      0.9971311  0.9525538
#>   5   6      0.9973344  0.9556440
#>   5   7      0.9971537  0.9521116
#>   5   8      0.9947364  0.9055569
#>   5   9      0.9931979  0.8762335
#>   5  10      0.9888182  0.7795664
```

```
#>
#> Tuning parameter 'scale' was held constant at a value of 1
#> Accuracy was used to select the optimal model using the largest value.
#> The final values used for the model were degree = 6, scale = 1 and C = 5.
```

#The optimal model with polynomial kernel is with a cost of 5 and degree of 6.

#Training error with Optimized SVM Polynomial Kernel model

```
predictions_svmpoly<-predict(model_svm_poly,train.data)
confusionMatrix(predictions_svmpoly,train.data$BlueTarp,positive = "Y")
```

```
#> Confusion Matrix and Statistics
#>
#>           Reference
#> Prediction     N     Y
#>          N 42845    83
#>          Y    21  1319
#>
#>                Accuracy : 0.9977
#>                  95% CI : (0.9972, 0.9981)
#>     No Information Rate : 0.9683
#>     P-Value [Acc > NIR] : < 2.2e-16
#>
#>                   Kappa : 0.9609
#>
#>  Mcnemar's Test P-Value : 2.21e-09
#>
#>             Sensitivity : 0.94080
#>             Specificity : 0.99951
#>          Pos Pred Value : 0.98433
#>          Neg Pred Value : 0.99807
#>              Prevalence : 0.03167
#>          Detection Rate : 0.02980
#>    Detection Prevalence : 0.03027
#>       Balanced Accuracy : 0.97015
#>
#>        'Positive' Class : Y
#>
```

#SVM Polynomial Confusion Matrix on test data (30% of original set)

```
predictions_svmpoly<-predict(model_svm_poly,newdata = test.data)
confusionMatrix(predictions_svmpoly,test.data$BlueTarp,positive = "Y")
```

```
#> Confusion Matrix and Statistics
#> 
#>           Reference
#> Prediction     N     Y
#>          N 18329    50
#>          Y    24   570
#> 
#>                Accuracy : 0.9961
#>                  95% CI : (0.9951, 0.9969)
#>     No Information Rate : 0.9673
#>     P-Value [Acc > NIR] : < 2.2e-16
#> 
#>                   Kappa : 0.937
#> 
#>  Mcnemar's Test P-Value : 0.003659
#> 
#>             Sensitivity : 0.91935
#>             Specificity : 0.99869
#>          Pos Pred Value : 0.95960
#>          Neg Pred Value : 0.99728
#>              Prevalence : 0.03268
#>          Detection Rate : 0.03004
#>    Detection Prevalence : 0.03131
#>       Balanced Accuracy : 0.95902
#> 
#>        'Positive' Class : Y
#> 
```

#With a 0.5 threshold the accuracy is 0.9961 and sensitivity is 0.91935.

#Decrease Threshold to 0.1 and calculate a new confusion matrix since are more concerned about sensitivity.

```
predict_threshold8<-predict(model_svm_poly,newdata=test.data,type="prob")
alteredProb8<-factor(ifelse(predict_threshold8$Y>=0.1,"Y","N"))
confusionMatrix(alteredProb8,test.data$BlueTarp,positive = "Y")
```

```
#> Confusion Matrix and Statistics
#> 
#>           Reference
#> Prediction     N     Y
#>          N 18209     9
#>          Y   144   611
#> 
#>                Accuracy : 0.9919
#>                  95% CI : (0.9906, 0.9932)
#>     No Information Rate : 0.9673
#>     P-Value [Acc > NIR] : < 2.2e-16
#> 
#>                   Kappa : 0.8846
#> 
#>  Mcnemar's Test P-Value : < 2.2e-16
#> 
#>             Sensitivity : 0.98548
#>             Specificity : 0.99215
#>          Pos Pred Value : 0.80927
#>          Neg Pred Value : 0.99951
#>              Prevalence : 0.03268
#>          Detection Rate : 0.03220
#>    Detection Prevalence : 0.03979
#>       Balanced Accuracy : 0.98882
#> 
#>        'Positive' Class : Y
#> 
```

#After lowering the threshold to 0.1, the accuracy remains high at 0.9919, and the sensitivity goes up to 0.985.

#From the three svm models, it seems like the polynomial kernel produces the best results on the training data. We will see if this holds on the holdout data.

8 Hold-out Data / EDA - Load data, explore data, etc.

#Load data and add a column to classify as Blue Tarp or non Blue Tarp

```
village1.tarp<-read.table("orthovnir067_ROI_Blue_Tarps.txt",skip=8,header=FALSE, sep="")
village1.tarp$BlueTarp<-'Y'

village1.notarp<-read.table("orthovnir067_ROI_NOT_Blue_Tarps.txt",skip=8,header=FALSE, sep="")
village1.notarp$BlueTarp<-'N'

village2.tarp<-read.table("orthovnir069_ROI_Blue_Tarps.txt",skip=8,header=FALSE,sep="")
village2.tarp$BlueTarp<-'Y'

village2.notarp<-read.table("orthovnir069_ROI_NOT_Blue_Tarps.txt",skip=8,header=FALSE,sep="")
village2.notarp$BlueTarp<-'N'

village3.tarp<-read.table("orthovnir078_ROI_Blue_Tarps.txt",skip=8,header=F,sep="")
village3.tarp$BlueTarp<-'Y'

village3.notarp<-read.table("orthovnir078_ROI_NON_Blue_Tarps.txt",skip=8,header=F,sep="")
village3.notarp$BlueTarp<-'N'

village4.notarp<-read.table("orthovnir057_ROI_NON_Blue_Tarps.txt",skip=8,header=F,sep="")
village4.notarp$BlueTarp<-'N'
```

#Combine the files

```
village.holdout<-rbind(village1.tarp,village1.notarp,village2.tarp,village2.notarp,village3.tarp,v
         illage3.notarp,village4.notarp)
```

#insert column header names and set BlueTarp to factor variable

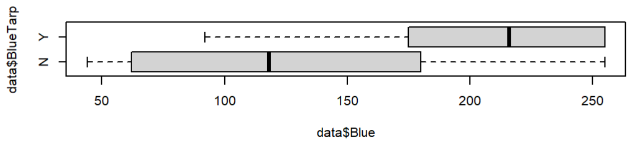
```
colnames(village.holdout)[8]<-"Red"
colnames(village.holdout)[9]<-"Green"
colnames(village.holdout)[10]<-"Blue"
village.holdout$BlueTarp<-as.factor(village.holdout$BlueTarp)
```

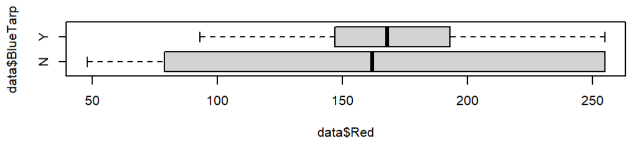
#select columns for analysis

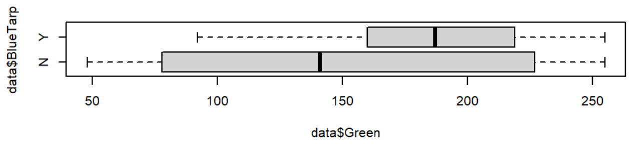
```
village.holdout<-village.holdout[8:11]
```

#EDA using boxplots to compare the training datasets and the holdout dataset.

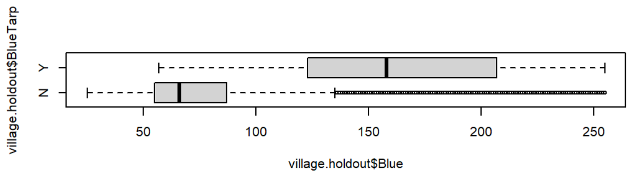
```
par(mfrow=c(3,1))
boxplot(data$Blue ~ data$BlueTarp, horizontal = T)
boxplot(data$Red ~ data$BlueTarp, horizontal = T)
boxplot(data$Green ~ data$BlueTarp, horizontal = T)
```

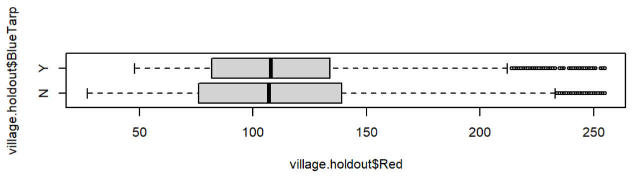


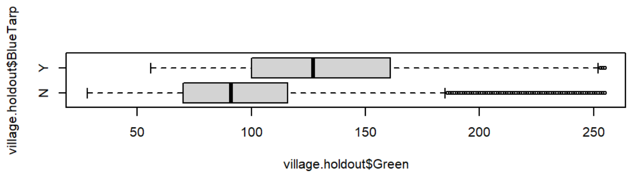




```r
par(mfrow=c(3,1))
boxplot(village.holdout$Blue ~ village.holdout$BlueTarp, horizontal = T)
boxplot(village.holdout$Red ~ village.holdout$BlueTarp, horizontal = T)
boxplot(village.holdout$Green ~ village.holdout$BlueTarp, horizontal = T)
```







#Using the box plots of the training data and the holdout data, we look to see if there are any similar patterns. We can see that in the training data the values for Blue in Blue tarps are higher and lower for non-blue tarps, i.e., the medians are further apart, the q1 to q3 quartiles are higher for blue tarps vs. non-blue tarps. A very similar behavior is also observed in the hold out data. Looking at Red next, in the training data the medians for blue tarps and non-blue tarps are close to each other, and the Red q1 to q3 quartiles are much wider for non-blue tarps than blue tarps. A similar pattern can be seen in the hold out data with respect to the medians being close to each other. The Red q1 to q3 quartiles are wider for non-blue tarps than blue tarps, but not as much as the training data. For Green, we see the medians are lower for non-blue tarps and the q1 to q3 quartiles are lower too, but they are much wider in range for non-blue tarps. It is somewhat similar in the holdout data where the median is lower for non-blue tarps and so is the q1 to q3 quartiles. However, the range for q1 to q3 quartiles are not as wide for non-blue tarps as they were for the training data. The similarities that were identifiable at least show that we have labeled the RGB columns correctly in the holdout data set since this was not given.

9 Results (Hold-Out)

#Logistic Regression predictions on holdout data using 0.1 threshold.

```
predict_holdout_lr<-predict(model_l,newdata = village.holdout,type="prob")
predict_holdout_lr_0.1<-factor(ifelse(predict_holdout_lr$Y>=0.1,"Y","N"))
confusionMatrix(predict_holdout_lr_0.1,village.holdout$BlueTarp,positive = "Y")
```

```
#> Confusion Matrix and Statistics
#> 
#>           Reference
#> Prediction       N       Y
#>          N 1846029       3
#>          Y  143668   14477
#> 
#>                Accuracy : 0.9283
#>                  95% CI : (0.928, 0.9287)
#>     No Information Rate : 0.9928
#>     P-Value [Acc > NIR] : 1
#> 
#>                   Kappa : 0.1566
#> 
#>  Mcnemar's Test P-Value : <2e-16
#> 
#>             Sensitivity : 0.999793
#>             Specificity : 0.927794
#>          Pos Pred Value : 0.091543
#>          Neg Pred Value : 0.999998
#>              Prevalence : 0.007225
#>          Detection Rate : 0.007223
#>    Detection Prevalence : 0.078908
#>       Balanced Accuracy : 0.963793
#> 
#>        'Positive' Class : Y
#> 
```

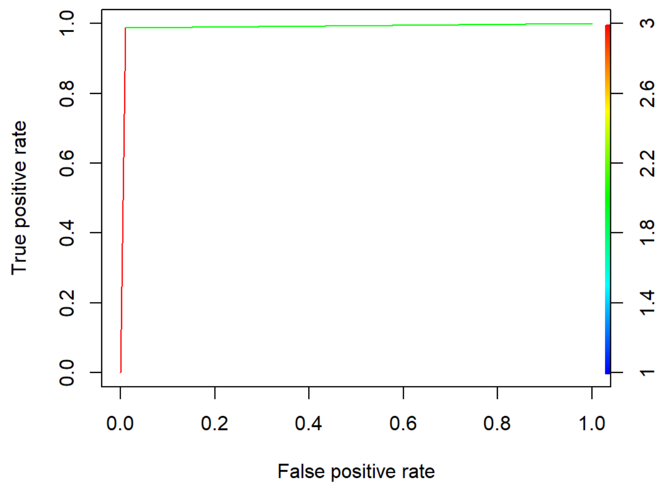
#ROC Curve and AUROC for Logistic Model

```
predictlg <- predict(model_l,village.holdout, type = 'raw')

ROCRpredlg <- prediction(as.numeric(predictlg),as.numeric(village.holdout$BlueTarp))

ROCRperflg<- performance(ROCRpredlg, 'tpr', 'fpr')

plot(ROCRperflg, colorize=TRUE, text.adj=c(-0.2,1.7))
```



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

```
#> [1] 0.9889889
```

#LDA predictions on holdout data using 0.1 threshold.

```
predict_holdout_lda<-predict(model_lda,newdata = village.holdout,type="prob")
predict_holdout_lda_0.1<-factor(ifelse(predict_holdout_lda$Y>=0.1,"Y","N"))
confusionMatrix(predict_holdout_lda_0.1,village.holdout$BlueTarp,positive = "Y")
```

```
#> Confusion Matrix and Statistics
#> 
#>           Reference
#> Prediction       N       Y
#>          N 1947138    1402
#>          Y   42559   13078
#> 
#>                Accuracy : 0.9781          
#>                  95% CI : (0.9779, 0.9783)
#>     No Information Rate : 0.9928          
#>     P-Value [Acc > NIR] : 1               
#> 
#>                   Kappa : 0.3658          
#> 
#>  Mcnemar's Test P-Value : <2e-16          
#> 
#>             Sensitivity : 0.903177        
#>             Specificity : 0.978610        
#>          Pos Pred Value : 0.235059        
#>          Neg Pred Value : 0.999280        
#>              Prevalence : 0.007225        
#>          Detection Rate : 0.006525        
#>    Detection Prevalence : 0.027761        
#>       Balanced Accuracy : 0.940894        
#> 
#>        'Positive' Class : Y               
#> 
```

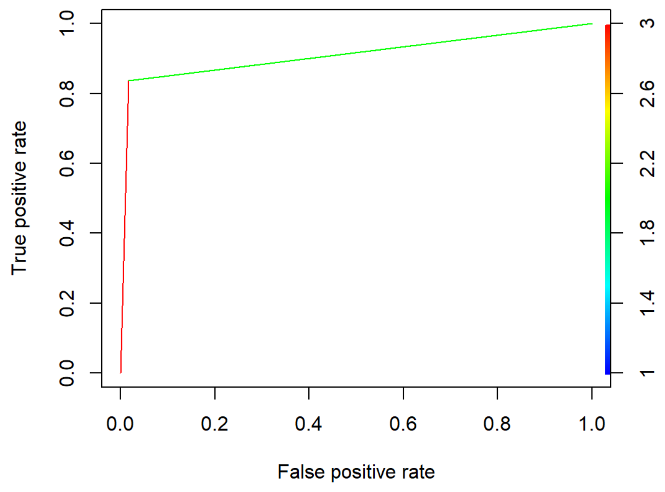
#ROC Curve and AUROC for LDA Model

```
predictlda <- predict(model_lda,village.holdout, type = 'raw')

ROCRpredlda <- prediction(as.numeric(predictlda),as.numeric(village.holdout$BlueTarp))

ROCRperflda<- performance(ROCRpredlda, 'tpr', 'fpr')

plot(ROCRperflda, colorize=TRUE, text.adj=c(-0.2,1.7))
```



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

```
#> [1] 0.9098517
```

#QDA predictions on holdout data using 0.1 threshold.

```
predict_holdout_qda<-predict(model_qda,newdata = village.holdout,type="prob")
predict_holdout_qda_0.1<-factor(ifelse(predict_holdout_qda$Y>=0.1,"Y","N"))
confusionMatrix(predict_holdout_qda_0.1,village.holdout$BlueTarp,positive = "Y")
```

```
#> Confusion Matrix and Statistics
#> 
#>           Reference
#> Prediction       N       Y
#>          N 1955479    2647
#>          Y   34218   11833
#> 
#>                Accuracy : 0.9816
#>                  95% CI : (0.9814, 0.9818)
#>     No Information Rate : 0.9928
#>     P-Value [Acc > NIR] : 1
#> 
#>                   Kappa : 0.3842
#> 
#>  Mcnemar's Test P-Value : <2e-16
#> 
#>             Sensitivity : 0.817196
#>             Specificity : 0.982802
#>          Pos Pred Value : 0.256954
#>          Neg Pred Value : 0.998648
#>              Prevalence : 0.007225
#>          Detection Rate : 0.005904
#>    Detection Prevalence : 0.022978
#>       Balanced Accuracy : 0.899999
#> 
#>        'Positive' Class : Y
#> 
```

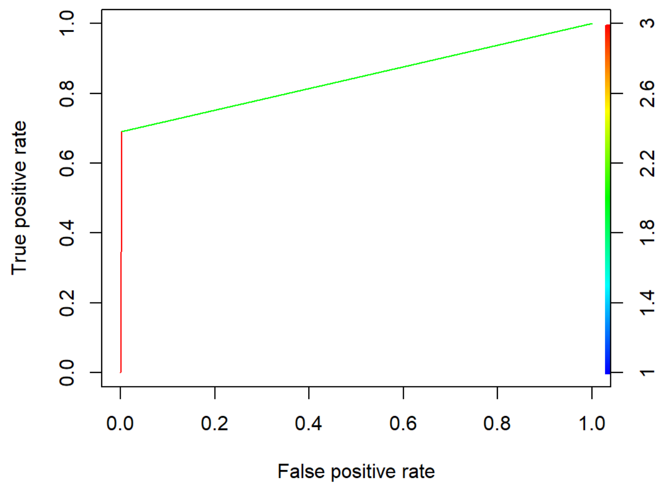
#ROC Curve and AUROC for QDA Model

```
predictqda <- predict(model_qda,village.holdout, type = 'raw')

ROCRpredqda <- prediction(as.numeric(predictqda),as.numeric(village.holdout$BlueTarp))

ROCRperfqda<- performance(ROCRpredqda, 'tpr', 'fpr')

plot(ROCRperfqda, colorize=TRUE, text.adj=c(-0.2,1.7))
```



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

```
#> [1] 0.8445653
```

#KNN predictions on holdout data using 0.1 threshold.

```
predict_holdout_knn<-predict(model_knn,newdata = village.holdout,type="prob")
predict_holdout_knn_0.1<-factor(ifelse(predict_holdout_knn$Y>=0.1,"Y","N"))
confusionMatrix(predict_holdout_knn_0.1,village.holdout$BlueTarp,positive = "Y")
```

```
#> Confusion Matrix and Statistics
#> 
#>           Reference
#> Prediction       N       Y
#>          N 1960915    1370
#>          Y   28782   13110
#> 
#>                Accuracy : 0.985
#>                  95% CI : (0.9848, 0.9851)
#>     No Information Rate : 0.9928
#>     P-Value [Acc > NIR] : 1
#> 
#>                   Kappa : 0.4593
#> 
#>  Mcnemar's Test P-Value : <2e-16
#> 
#>             Sensitivity : 0.905387
#>             Specificity : 0.985534
#>          Pos Pred Value : 0.312948
#>          Neg Pred Value : 0.999302
#>              Prevalence : 0.007225
#>          Detection Rate : 0.006541
#>    Detection Prevalence : 0.020902
#>       Balanced Accuracy : 0.945461
#> 
#>        'Positive' Class : Y
#> 
```

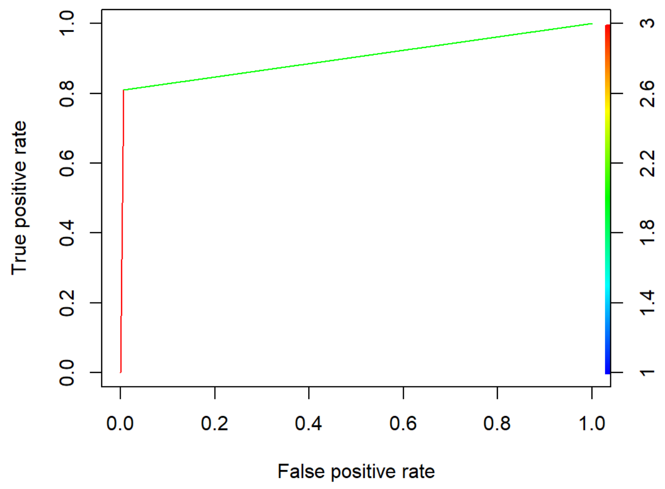
#ROC Curve and AUROC for KNN Model

```
predictknn <- predict(model_knn,village.holdout, type = 'raw')

ROCRpredknn <- prediction(as.numeric(predictknn),as.numeric(village.holdout$BlueTarp))

ROCRperfknn<- performance(ROCRpredknn, 'tpr', 'fpr')

plot(ROCRperfknn, colorize=TRUE, text.adj=c(-0.2,1.7))
```



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

```
#> [1] 0.9019009
```

#Lasso predictions on holdout data using 0.1 threshold.

```
x.holdout=model.matrix(BlueTarp~-1+Red+Blue+Green,data=village.holdout)
predict_holdout_ls<-predict(lasso.fit, s=lasso.fit$lambda.min, newx=x.holdout)
alteredProb_ls_0.1<-predict_holdout_ls[,1]
alteredProb_ls_0.1<-factor(ifelse(alteredProb_ls_0.1>=0.1,"Y","N"))
confusionMatrix(alteredProb_ls_0.1,village.holdout$BlueTarp,positive = "Y")
```

```
#> Confusion Matrix and Statistics
#> 
#>           Reference
#> Prediction       N       Y
#>          N 1972676     177
#>          Y   17021   14303
#> 
#>                Accuracy : 0.9914
#>                  95% CI : (0.9913, 0.9915)
#>     No Information Rate : 0.9928
#>     P-Value [Acc > NIR] : 1
#> 
#>                   Kappa : 0.6208
#> 
#>  Mcnemar's Test P-Value : <2e-16
#> 
#>             Sensitivity : 0.987776
#>             Specificity : 0.991445
#>          Pos Pred Value : 0.456615
#>          Neg Pred Value : 0.999910
#>              Prevalence : 0.007225
#>          Detection Rate : 0.007137
#>    Detection Prevalence : 0.015629
#>       Balanced Accuracy : 0.989611
#> 
#>        'Positive' Class : Y
#> 
```

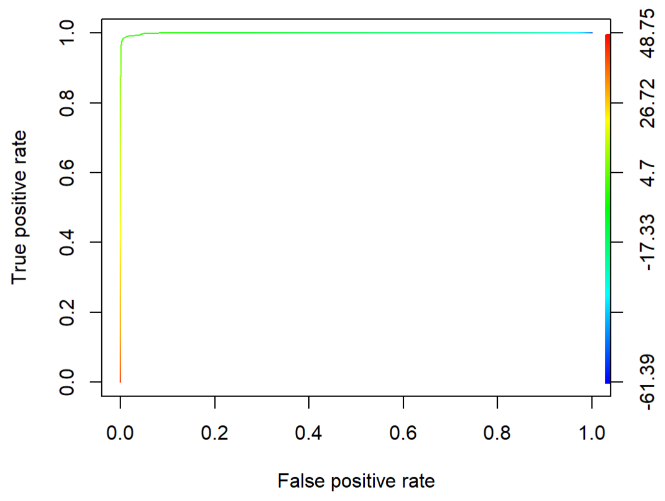
#ROC Curve and AUROC for Lasso Model

```
predictls <- predict(lasso.fit, s=lasso.fit$lambda.min, newx=x.holdout)

ROCRpredls <- prediction(as.numeric(predictls),as.numeric(village.holdout$BlueTarp))

ROCRperfls<- performance(ROCRpredls, 'tpr', 'fpr')

plot(ROCRperfls, colorize=TRUE, text.adj=c(-0.2,1.7))
```



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

```
#> [1] 0.99943
```

#Random forest predictions on holdout data using 0.1 threshold.

```
predict_holdout_rf<-predict(model_rf,newdata = village.holdout,type="prob")
predict_holdout_rf_0.1<-factor(ifelse(predict_holdout_rf$Y>=0.1,"Y","N"))
confusionMatrix(predict_holdout_rf_0.1,village.holdout$BlueTarp,positive = "Y")
```

```
#> Confusion Matrix and Statistics
#> 
#>           Reference
#> Prediction       N       Y
#>          N 1959830     695
#>          Y   29867   13785
#> 
#>                Accuracy : 0.9848
#>                  95% CI : (0.9846, 0.9849)
#>     No Information Rate : 0.9928
#>     P-Value [Acc > NIR] : 1
#> 
#>                   Kappa : 0.4685
#> 
#>  Mcnemar's Test P-Value : <2e-16
#> 
#>             Sensitivity : 0.952003
#>             Specificity : 0.984989
#>          Pos Pred Value : 0.315793
#>          Neg Pred Value : 0.999646
#>              Prevalence : 0.007225
#>          Detection Rate : 0.006878
#>    Detection Prevalence : 0.021781
#>       Balanced Accuracy : 0.968496
#> 
#>        'Positive' Class : Y
#> 
```

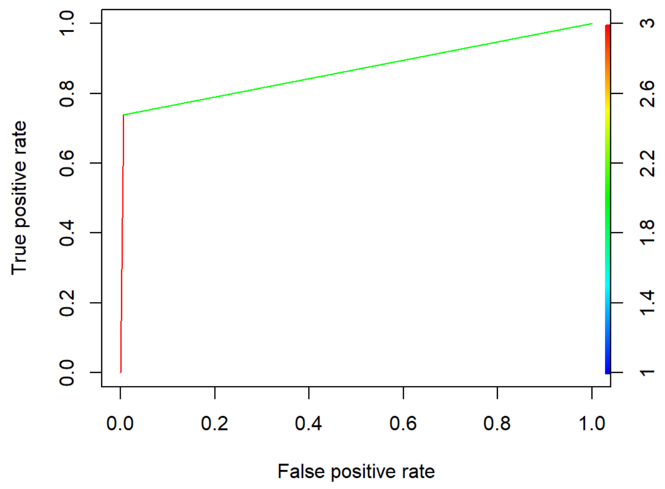
#ROC Curve and AUROC for Random Forest Model

```
predictrf <- predict(model_rf,village.holdout, type = 'raw')

ROCRpredrf <- prediction(as.numeric(predictrf),as.numeric(village.holdout$BlueTarp))

ROCRperfrf<- performance(ROCRpredrf, 'tpr', 'fpr')

plot(ROCRperfrf, colorize=TRUE, text.adj=c(-0.2,1.7))
```



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

```
#> [1] 0.867136
```

#SVM Linear predictions on holdout data using 0.1 threshold.

```
predict_holdout_svmL<-predict(model_svm_linear,newdata = village.holdout,type="prob")
predict_holdout_svmL_0.1<-factor(ifelse(predict_holdout_svmL$Y>=0.1,"Y","N"))
confusionMatrix(predict_holdout_svmL_0.1,village.holdout$BlueTarp,positive = "Y")
```

```
#> Confusion Matrix and Statistics
#> 
#>           Reference
#> Prediction       N       Y
#>          N 1841976      84
#>          Y  147721   14396
#>                                           
#>                Accuracy : 0.9263          
#>                  95% CI : (0.9259, 0.9266)
#>     No Information Rate : 0.9928          
#>     P-Value [Acc > NIR] : 1               
#>                                           
#>                   Kappa : 0.1518          
#>                                           
#>  Mcnemar's Test P-Value : <2e-16          
#>                                           
#>             Sensitivity : 0.994199        
#>             Specificity : 0.925757        
#>          Pos Pred Value : 0.088800        
#>          Neg Pred Value : 0.999954        
#>              Prevalence : 0.007225        
#>          Detection Rate : 0.007183        
#>    Detection Prevalence : 0.080890        
#>       Balanced Accuracy : 0.959978        
#>                                           
#>        'Positive' Class : Y               
#> 
```

#SVM Radial predictions on holdout data using 0.1 threshold.

```
predict_holdout_svmR<-predict(model_svm_radial,newdata = village.holdout,type="prob")
predict_holdout_svmR_0.1<-factor(ifelse(predict_holdout_svmR$Y>=0.1,"Y","N"))
confusionMatrix(predict_holdout_svmR_0.1,village.holdout$BlueTarp,positive = "Y")
```

```
#> Confusion Matrix and Statistics
#> 
#>           Reference
#> Prediction       N       Y
#>          N 1966331    3103
#>          Y   23366   11377
#> 
#>                Accuracy : 0.9868
#>                  95% CI : (0.9866, 0.987)
#>     No Information Rate : 0.9928
#>     P-Value [Acc > NIR] : 1
#> 
#>                   Kappa : 0.4567
#> 
#>  Mcnemar's Test P-Value : <2e-16
#> 
#>             Sensitivity : 0.785704
#>             Specificity : 0.988257
#>          Pos Pred Value : 0.327462
#>          Neg Pred Value : 0.998424
#>              Prevalence : 0.007225
#>          Detection Rate : 0.005677
#>    Detection Prevalence : 0.017335
#>       Balanced Accuracy : 0.886980
#> 
#>        'Positive' Class : Y
#> 
```

#SVM Polynomial predictions on holdout data using 0.1 threshold.

```
predict_holdout_svmP<-predict(model_svm_poly,newdata = village.holdout,type="prob")
predict_holdout_svmP_0.1<-factor(ifelse(predict_holdout_svmP$Y>=0.1,"Y","N"))
confusionMatrix(predict_holdout_svmP_0.1,village.holdout$BlueTarp,positive = "Y")
```

```
#> Confusion Matrix and Statistics
#> 
#>           Reference
#> Prediction       N       Y
#>          N 1938791    7314
#>          Y   50906    7166
#>                                           
#>                Accuracy : 0.971           
#>                  95% CI : (0.9707, 0.9712)
#>     No Information Rate : 0.9928          
#>     P-Value [Acc > NIR] : 1               
#>                                           
#>                   Kappa : 0.1882          
#>                                           
#>  Mcnemar's Test P-Value : <2e-16          
#>                                           
#>             Sensitivity : 0.494890        
#>             Specificity : 0.974415        
#>          Pos Pred Value : 0.123399        
#>          Neg Pred Value : 0.996242        
#>              Prevalence : 0.007225        
#>          Detection Rate : 0.003576        
#>    Detection Prevalence : 0.028975        
#>       Balanced Accuracy : 0.734652        
#>                                           
#>        'Positive' Class : Y               
#> 
```

#Based on the results of the three SVM kernels on the holdout data, SVM linear produces the highest sensitivity and 99%, Radial is at 78.5%, and Polynomial 49.4%. The overall accuracy rates are about the same (in the 90 plus percentage range). Given that we are more concerned about sensitivity, we will select the SVM Linear kernel from the three SVM models.

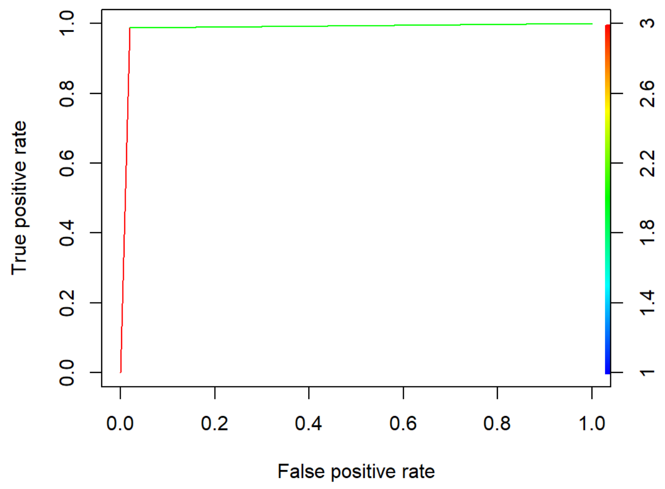
#ROC Curve and AUROC for SVM Model

```
predictsvm <- predict(model_svm_linear,village.holdout, type = 'raw')

ROCRpredsvm <- prediction(as.numeric(predictsvm),as.numeric(village.holdout$BlueTarp))

ROCRperfsvm<- performance(ROCRpredsvm, 'tpr', 'fpr')

plot(ROCRperfsvm, colorize=TRUE, text.adj=c(-0.2,1.7))
```



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

```
#> [1] 0.9846463
```

Hold-Out Performance Table Here

```
Model<-c("Log Reg","LDA", "QDA", "KNN", "Penalized Log Reg","Random Forest","SVM")
Tuning<-c("-","-","-","k=7","lambda=-12.02, alpha=1","mtry=2","kernel=linear, cost=2.5")
AUROC<-c("0.9889","0.9099","0.8446","0.9019","0.999","0.867","0.9846")
Threshold<-c("0.1","0.1","0.1","0.1","0.1","0.1","0.1")
Accuracy<-c("0.9283","0.9781","0.9816","0.985","0.9918","0.9848","0.926")
TPR<-c("0.9998","0.9032","0.817","0.9054","0.987","0.952","0.994")
FPR<-c("0.0722","0.022","0.018","0.015","0.01","0.015","0.075")
Precision<-c("0.0915","0.235","0.257","0.313","0.468","0.3158","0.089")
CV.Performance2<-data.frame(Model,Tuning,AUROC,Threshold,Accuracy,TPR,FPR,Precision)
knitr::kable(CV.Performance2,"pipe")
```

Model	Tuning	AUROC	Threshold	Accuracy	TPR	FPR	Precision
Log Reg	-	0.9889	0.1	0.9283	0.9998	0.0722	0.0915
LDA	-	0.9099	0.1	0.9781	0.9032	0.022	0.235
QDA	-	0.8446	0.1	0.9816	0.817	0.018	0.257
KNN	k=7	0.9019	0.1	0.985	0.9054	0.015	0.313
Penalized Log Reg	lambda=-12.02, alpha=1	0.999	0.1	0.9918	0.987	0.01	0.468
Random Forest	mtry=2	0.867	0.1	0.9848	0.952	0.015	0.3158
SVM	kernel=linear, cost=2.5	0.9846	0.1	0.926	0.994	0.075	0.089

10 Final Conclusions

10.0.1 Conclusion #1

#All the models showed a high accuracy rate of over 98% with the training data. The range was a little wider for sensitivity rates. The reason sensitivity rate is an important factor in this situation is because we are concerned about reaching as many people as possible even if we get some false positives in the process as long as it doesn't decrease the accuracy rates by much. I reduced the thresholds to 0.1 as described in the above section on thresholds. I had gone with KNN as my best model with the training data as it had a high sensitivity rate of 98%, high precision rate of 89%, and low false positive of 0.4%. But when I ran the models on the hold-out data, I preferred the penalized logistic model that used a lasso penalty. This had a high accuracy of 99%, sensitivity rate of 98.7% and a precision rate of 46.8%. The other models also had high accuracy rates in the hold-out data with results of over 92%, and the false positive ranged from 82% in the QDA to 99% in SVM linear and Penalized logistic regression. As I will discuss in one of my other conclusions, I did not select SVM Linear due to its low precision rate. While the KNN also had good results, the penalized logistic results came out slightly on top on all measures with the biggest difference being in precision (46.8% vs. 31.3%). It appeared that a logistic model would be best here as it did not overtrain the model and when adding the lasso penalty it improved results. The KNN might have slighly overtrained the model.

10.0.2 Conclusion #2

#The results are compatible because the penalized logistic regression performed well in training data with high accuracy and precision rates. It did not have the best sensitivity rate on the training data, but when used on the hold-out data, the sensitivity rate was the highest, which means it did not overtrain on the training dataset. Another reason this is compatible is because the logistic regression model performed well on the training and hold out data, but it was improved when we added the lasso penalty to make the model less flexible and even less prone to overfitting.

10.0.3 Conclusion #3

#The models that I prefer for solving this problem are Penalized Logistic regression and Random Forest. But in the end, I would select Penalized logistic regression with the lasso penalty. While a few models had comparable accuracy rates and sensitivity rates, the biggest difference came in the precision rate. Penalized logistic regression precision came in at a 46.8% and the next best was Random Forest at 31.58%. To me this is important as it saves time because it means we will have less false positives when we compare it to true positives from our predictions. This in turn means we would reach our true positives i.e blue tarps a lot quicker. I felt this is an important factor to consider in a disaster relief as time is of the essence. So, while many models had comparably high sensitivity rates, the difference came down to differences in the precision rates as models such as SVM, Random Forest, and KNN did not match up to the penalized logistic regression model's precision rates.

10.0.4 Conclusion #4

#The metrics in the table were relevant for this problem because they helped in determining what factors were important in selecting a model in this scenario. Starting with accuracy, we want a model to get the predictions right as many times as possible. TPR is important because in this model sensitivity is an important factor as we are concerned about reaching as many people as possible even if we get some false positives in the process. The threshold plays an important role in this because this tells us our goal/intention, which in this case was to get a high sensitivity rate. The AUROC tells us our ability to distinguish between classes (blue tarps and non-blue tarps in this case). In other words, it tells us if our model will do better than a flip of a coin, and the models here did that because they came in at over 0.90 in almost all cases. The precision rate is important here as it tells us from our positive predictions how many were actually positive, and as mentioned above, this can be a huge time save in a disaster recovery as we will waste less time with incorrect predictions. The tuning parameters show what we did to get the best model where applicable.

10.0.5 Conclusion #5

#When building the models with the different SVM kernels (linear, radial, polynomial), I noticed that the linear kernel did not do well with sensitivity on the training set when compared to the other two kernels. It came in at 93%, whereas radial came in at 96.7%, and polynomial came in even higher at 98.5%. The overall accuracy rates were comparable for all 3 kernels. But when I applied the three models to the hold-out the data, the linear kernel did much better with regards to sensitivity. It came in at 99.4%, which was one of the highest across all models in the project. Radial and Polynomial on the other hand were the lowest performers for sensitivity with 78.5% and 49% respectively. This made me conclude the decision boundary for this dataset is more on the linear side which is not what I had initially thought. The reason I was mistaken is because the Radial and especially the Polynomial models were overfitting on the training data. This is probably also why models such as penalized logistic regression did well and models such as QDA did not. After thinking about it further it would make sense that it is more linear given that the Blue is more prevalent in blue tarps and these points be closer to each other if we were to plot this on a plane.

10.0.6 Conclusion #6

#When running the models on the hold out data, the parametric models ended up doing better than when they were run on the training data. Models such as Logistic regression, SVM Linear, and LDA all improved their sensitivity rates from training to hold out data set. For instance, LDA sensitivity rates went from 82.7% to 90.32% from training to hold-out results. The non-parametric models which make no assumptions on the shape of the boundary decreased in performance when it came to sensitivity rates. For instance, KNN went down from 98.06% to 90.54%. Same applies as for SVM Linear versus SVM polynomial and SVM radial as explained above. Random forest also dropped although not as significantly and still maintained good overall sensitivity rates. The hold-out data set was much larger than the training data set and perhaps with the more data points it showed that this data set actually has more of normal distribution, which is why the parametric models did well in the end. With the smaller subset in the training dataset this might have not been as obvious or prevalent, which is why the non-parametric models did better in training results but they might have ended up overfitting.