Disaster Relief Project - Part 2

Michael Macfarlan (zxr5fv)

DS6030 Statistical Learning, Summer 2023

Introduction

This project uses a number of classification methods to tackle a real-world problem. The goal was to use pixel data from satellite imagery to identify the locations of people displaced by the 2010 earthquake in Haiti. One of the most challenging issues plagueing disaster relief services is locating displaced people in a timely and efficient manner. If it were possible to speed up and make more efficient the process of locating displaced Haitians, this would allow search and rescue workers to provide relief to those in need before they run out of food, water, and medicine.

A team from the Rochester Institute of Technology gathered geo-referenced imagery which was converted to pixel data. This pixel data is then used to train a number of classification models in this project to identify blue tarps, which were used as temporary shelters by Haitians displaced by the earthquake. Later, holdout data is used to evaluate the performance of the models.

The first part of this project focuses on exploratory data analysis and data visualization, exploring the relationships between the RGB values of each pixel, the classes of each pixel, and interesting insights uncovered.

This project then applies logistic regression, linear discriminant analysis, quadratic discriminant analysis, k-nearest neighbors, penalized logistic regression, random forest, and support vector machine models to the pixel data to predict which observations were blue tarps. Two different types of models are compared for each method, one using the raw RGB values of each pixel, and the other using a stepwise regression with multiple transformed variables. These models are then evaluated using 10-fold cross validation for the purpose of finding the best performing model.

Before assessing models on the holdout data, the data has to be prepared and ingested properly, then assessed to ensure that the holdout data is similar to the training data.

The reported results include a table of performance statistics (and qualities) during cross validation as well as during holdout testing. Results in both tables cover characteristics for each of the best performing models, including accuracy, true positive rate, false positive rate, precision, AUC, tuning, and more.

In its conclusion, this project discusses potential improvements to consider and possible shortcomings and limitations in the approach to model creation and evaluation. Additionally, the project hypothesized that the Support Vector Machine model (with a radial kernel) with no transformed variables would emerge as the model that would perform best in this scenario. This was hypothesized with support of performance metrics as well as a consideration of its balance of false positives and false negatives in particular.

Upon a fuller assessment of the models with holdout data, it was clear that model with the best performance was not the SVM model, but the Penalized Logistic Regression model with transformed variables.

Note: For the sake of brevity, many steps will be described but not shown. For the full code, please reference the R Markdown file.

Data Loading and Exploratory Data Analysis

Data Loading and Preparation

First, the necessary packages are loaded and the training data is read in from a CSV. Then, additional columns are appended to the dataframe to make different kinds of exploratory data analysis easier, enable the creation of different kinds of plots, and prepare it for the regression analysis. A column is added to note the class numerically and a column is added to note whether the pixel is a blue tarp or not.

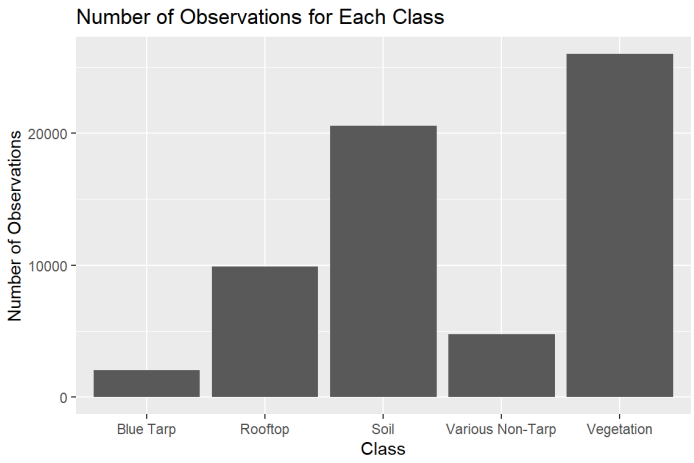
The final columns and data types of the columns in our dataframe include:

Variable	Description
Class	string description representing the class of the pixel
Red	integer representing the red value of the pixel
Green	integer representing the green value of the pixel
Blue	integer representing the blue value of the pixel
NumClass	number representing the class of the pixel
ClassTarp	factor representing whether the pixel is a blue tarp or not

Finally, the data is checked for missing values. There are none.

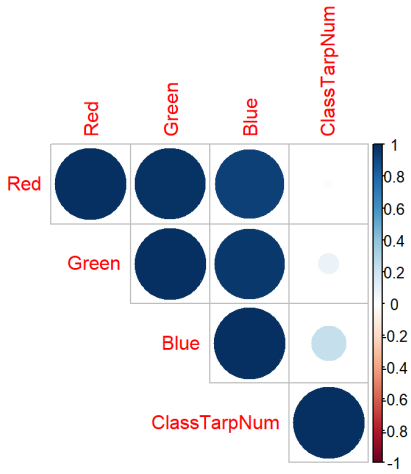
Exploratory Data Analysis

One of the first things to explore is the distribution of the classes in the data. One would assume that the minority of the observations will be blue tarps, as the area they take in relation to everything else is small.

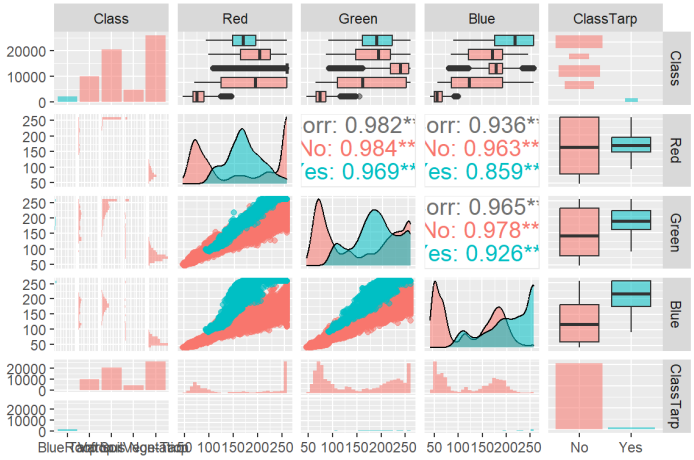


Next, we visually explore relationships between the RGB values and `ClassTarp`.

By exploring correlations between the RGB values and `ClassTarp` in the figure below, we can see that `Blue` has the highest correlation with `ClassTarp`, followed by `Green`, although slight. This is also expected, since the tarps we are targeting are blue in color.

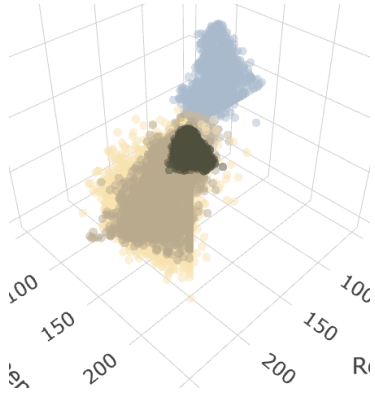


In the following figure, we continue to explore the scatterplots of the continuous variables against each other, `Class`, and `ClassTarp` and visualize the `ClassTarp` variable across all plots using color.



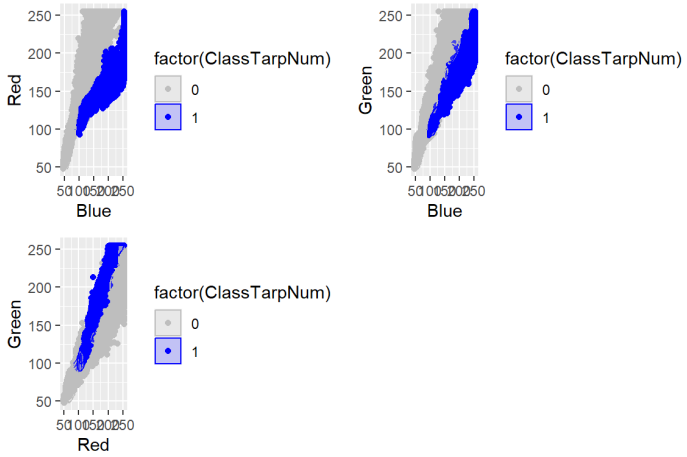
This figure's plots makes it much easier to see the bell curve shaped distribution of the pixel value against `ClassTarp`, unless the pixel value is blue, in which case as the value increases, so does the distribution of blue tarps. Another way to say this is the distribution of `ClassTarp` = Yes skews to the right the most for blue pixels.

To add a bit of realism to the next figure, the average RGB values for each class were calculated and converted to hex values to use as the color for each class. The following 3D plot was created using these average color for each class.



The previous figure is the easiest to see the separation of the classes across the three axes of RGB values. Although some classes like `Soil` and `Various Non-Tarp`, and `Rooftop` seem to particularly intermingle and overlap, one can start to see general clustered areas for each class in the 3D plot.

Finally, the following figures are the probability density plots of the RGB values plotted against each other and distinguished with the color of the points representing the `ClassTarp` variable.



This figure specifically highlights the location of the blue tarps in two dimensions across the scatterplots of the RGB values against each other. Here we see interesting cluster groupings. On the `Red` vs `Green` plot, we see a dense, tightly defined cluster, yet overlapping more with other classes. On the `Blue` vs `Red` and `Blue` vs `Green` plots, we see the blue tarps slightly more widely spread, yet not overlapping as much with the other classes. This is essentially showing us the same thing as the 3D plot, but specifically highlighting the blue tarps and shown in two dimensions.

Formulas and Models

Analysis Preparation

Before getting started with model construction, the seed needs to be set for reproducibility. In addition, to properly conduct cross validation for the models, the folds and the trainControl object are created.

Furthermore, to make the analysis of the models simpler, two formulas are created to quickly calculate and append statistics in a dataframe and to plot ROC curves. The `thresh_test` function calculates the statistics for a model at multiple thresholds and returns a dataframe of the statistics.

```
# https://www.rdocumentation.org/packages/caret/versions/6.0-92/topics/thresholder
# sequence of thresholds to test
thresh <- seq(0.1, 0.9, 0.1)

# statistics to calculate
statsout <- c("Accuracy", "Kappa", "Sensitivity", "Specificity", "Precision")

# this is a function to test multiple thresholds for a model and return stats
# input: model to test
# output: df of stats
thresh_test <- function(model) {

  # calculate statistics for each threshold
  stats <- thresholder(model, threshold=thresh, statistics=statsout)

  # add false negative rate
  stats$falseNeg <- 1 - stats$Sensitivity

  # and false positive rate
  stats$falsePos <- 1 - stats$Specificity

  return(stats)
}
```

The `roc_auc_plot` function plots the ROC curve for a selected model and calculates the area under the curve (AUROC).

```
# https://www.geeksforgeeks.org/how-to-calculate-auc-area-under-curve-in-r/#
# this is a function to plot the ROC curve for a model and calculate AUROC
# input: model to plot, stats df, title of model
# output: stats df with AUROC appended
roc_auc_plot <- function(model, stat_df, model_title) {
      # calculate probabilities and TPR/FPR
      prob <- model$pred[order(model$pred$rowIndex),]

      # get rates
      rates <- prediction(prob$Yes, as.numeric(data$ClassTarp))

      # plot ROC curve with dotted diagonal line
      roc <- performance(rates, measure = "tpr", x.measure = "fpr")
      plot(roc, main=paste("ROC Curve:", model_title))
      abline(a = 0, b = 1, lty = 2, col = "red")

      # calculate AUROC
      auc <- performance(rates, "auc")

      # append AUROC to stats
      stat_df <- stat_df %>% mutate(AUROC = auc@y.values[[1]])
      return(stat_df)
}
```

Finally, before during some of the model constructions, caret's parallel computing functionality will be used. This takes advantage of the local machine's number of cores, minus a certain number so that we can continue working while the models are built.

Model Formula Selection

Next, the models are built. First, a test to see if there would be any reason to leave out any predictor variables, `Red`, `Green`, or `Blue` in the modeling. A simple logistic regression model was created including all three variables and the variables and AIC are inspected.

```
# create basic logistic regression model
mod_log <- glm(ClassTarp ~ Red + Green + Blue, data = data, family = "binomial")
summary(mod_log)
```

```
#>
#> Call:
#> glm(formula = ClassTarp ~ Red + Green + Blue, family = "binomial",
#>     data = data)
#>
#> Coefficients:
#>             Estimate Std. Error z value Pr(>|z|)
#> (Intercept)  0.20984    0.18455   1.137    0.256
#> Red         -0.26031    0.01262 -20.632   <2e-16 ***
#> Green       -0.21831    0.01330 -16.416   <2e-16 ***
#> Blue         0.47241    0.01548  30.511   <2e-16 ***
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> (Dispersion parameter for binomial family taken to be 1)
#>
#>     Null deviance: 17901.6  on 63240  degrees of freedom
#> Residual deviance:  1769.5  on 63237  degrees of freedom
#> AIC: 1777.5
#>
#> Number of Fisher Scoring iterations: 12
```

Results show that all of the variables are significant and the AIC is low. At first glance, we have a good looking model.

Next, we conduct a stepwise regression to see if any variables can be removed.

This stepwise regression did not remove any variables. Therefore, without transformations, we know that keeping all the variables is the best model.

The next step was to find out if there were any transformations that would improve the model, so a heavy-handed approach was taken by putting a number of variable transformations in the model, then doing a stepwise regression to calculate which variables to drop or keep. (For the sake of brevity, the code for the stepwise regression is not shown here, but can be found in the Rmd i.)

```
# Transformations: all pred variables ^2, ^3, ^4, log, sqrt
mod_log_t <- glm(ClassTarp ~ Red + Green + Blue +
I(Red^2) + I(Green^2) + I(Blue^2) +
    log(Red) + log(Green) + log(Blue) +
    sqrt(Red) + sqrt(Green) + sqrt(Blue) +
    I(Red^3) + I(Green^3) + I(Blue^3) +
    I(Red^4) + I(Green^4) + I(Blue^4),
    data = data, family = "binomial")
```

The stepwise regression of the model with transformations removed: `Green^2  Blue^2  Red^4`

In summary, four models were created and compared. A model with no transformations, a stepwise model with no transformations, a model with transformations, and a stepwise model with transformations.

The AIC and formulas of each are as follows:

Model	AIC	Formula
Model with no transformations	1777.53	ClassTarp ~ Red + Green + Blue
Stepwise model with no transformations	1777.53	ClassTarp ~ Red + Green + Blue
Model with transformations	1004.04	ClassTarp ~ Red + Green + Blue + I(Red^2) + I(Green^2) + I(Blue^2) + log(Red) + log(Green) + log(Blue) + sqrt(Red) + sqrt(Green) + sqrt(Blue) + I(Red^3) + I(Green^3) + I(Blue^3) + I(Red^4) + I(Green^4) + I(Blue^4)
Stepwise model with transformations	998.46	ClassTarp ~ Red + Green + Blue + I(Red^2) + log(Red) + log(Green) + log(Blue) + sqrt(Red) + sqrt(Green) + sqrt(Blue) + I(Red^3) + I(Green^3) + I(Blue^3) + I(Green^4) + I(Blue^4)

Interestingly, the stepwise model with transformations only removed 3 variables from the model with all transformations included.

What we see here are results that may indicate that the transformations may be useful in improving the accuracy of the model, however, we may also be able to predict that the additional variables may end up overfitting the model. I moved forward with testing the model with no transformations as well as the stepwise model with a selection of transformations.

Because it's always interesting to test different models, I decided to move forward with testing the model with no transformations against the stepwise model with transformations.

For the remainder of the report, I will refer to the model with no transformations as MWNT and the stepwise model with transformations as SMWT.

Logistic Regression

The first model tested is a logistic regression model. The models are built with caret's train function, the control object is set to 10-fold cross validation, and then the accuracy and kappa statistics are compared.

```
#>                                     Accuracy  Kappa
#> [1,] "Model with No Transformations"  0.9952721 0.9203991
#> [2,] "Stepwise Transformations Model" 0.9963315 0.9398715
```

These results show that the SMWT is slightly more accurate than the MWNT. For good measure, we will test each model with a threshold test to see if we can improve the accuracy of the model by adjusting the threshold. Then, the best performing thresholds of each model are compared.

```
# threshold test for glm_mod
glm_mod_thres <- thresh_test(glm_mod)

# print the row with the highest accuracy and remove the first column
glm_mod_thres[2:9] %>% slice_max(Accuracy)
```

```
#>   prob_threshold  Accuracy     Kappa Sensitivity Specificity Precision
#> 1            0.7 0.9956516 0.9282686   0.9984645   0.9104814 0.9970478
#>      falseNeg   falsePos
#> 1 0.001535459 0.08951861
```

```
# threshold test for glm_mod_t
glm_mod_t_thres <- thresh_test(glm_mod_t)

# print the row with the highest accuracy and remove the first column
glm_mod_t_thres[2:9] %>% slice_max(Accuracy)
```

```
#>   prob_threshold  Accuracy     Kappa Sensitivity Specificity Precision
#> 1            0.5 0.9963315 0.9398715   0.9985135   0.9302517 0.9976995
#>      falseNeg   falsePos
#> 1 0.001486453 0.06974833
```

Between the two models and each of their threshold tests, the most accurate model is the SMWT at a threshold of 0.5. The ROC curve is plotted and the AUROC is calculated for this model next.

```
#>   prob_threshold  Accuracy     Kappa Sensitivity Specificity Precision
#> 1            0.5 0.9963315 0.9398715   0.9985135   0.9302517 0.9976995
#>      falseNeg   falsePos     AUROC
#> 1 0.001486453 0.06974833 0.9855816
```

In general, the ROC curve tells us how well the model is able to distinguish between a pixel that is a tarp and a pixel that is not a tarp. The closer the curve is to the top left corner, the better the model is at distinguishing between the two classes, because this curve is showing us the true positive rate (sensitivity) against the false positive rate (1-specificity).

Here we have an excellent looking ROC curve and a very high AUROC of 0.9856. Next, we explore the performance of additional models.

Linear Discriminant Analysis

The next model tested is a linear discriminant analysis model. Again, the models are built with caret's train function, the control object is set to 10-fold cross validation, and then the accuracy and kappa statistics are compared.

```
lda_mod <- train(ClassTarp ~ Red + Green + Blue, data = data,
                 method = "lda",
                 trControl = control)

lda_mod_t <- train(ClassTarp ~ Red + Green + Blue +
                       I(Red^2) + log(Red) + log(Green) +
                       log(Blue) + sqrt(Red) + sqrt(Green) +
                       sqrt(Blue) + I(Red^3) + I(Green^3) +
                       I(Blue^3) + I(Green^4) + I(Blue^4),
                 data = data,
                 method = "lda",
                 trControl = control)

rbind(c("Model with No Transformations", lda_mod$results[2], lda_mod$results[3]),
      c("Stepwise Transformations Model", lda_mod_t$results[2], lda_mod_t$results[3]))
```

```
#>                                    Accuracy  Kappa
#> [1,] "Model with No Transformations"  0.9839345 0.7528654
#> [2,] "Stepwise Transformations Model" 0.9882671 0.8311042
```

These results show that the SMWT again performs at a slightly higher accuracy than the MWNT. Next, threshold tests and comparisons.

```
lda_mod_thres <- thresh_test(lda_mod)
lda_mod_thres[2:9] %>% slice_max(Accuracy)
```

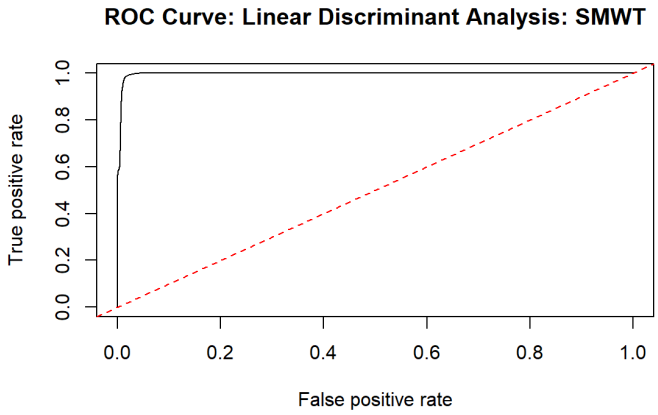
```
#>   prob_threshold Accuracy     Kappa Sensitivity Specificity Precision
#> 1            0.1 0.984646 0.7476611   0.9926493   0.7423353 0.9915003
#>      falseNeg  falsePos
#> 1 0.007350662 0.2576647
```

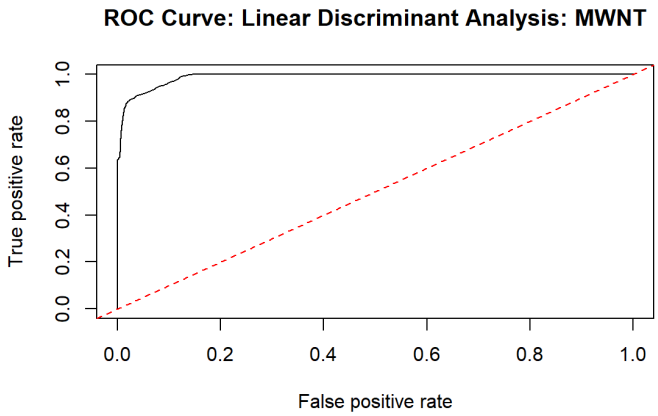
```
lda_mod_t_thres <- thresh_test(lda_mod_t)
lda_mod_t_thres[2:9] %>% slice_max(Accuracy)
```

```
#>   prob_threshold  Accuracy    Kappa Sensitivity Specificity Precision
#> 1            0.1 0.9888996 0.833544   0.9916693    0.905048 0.9968474
#>      falseNeg   falsePos
#> 1 0.008330736 0.09495196
```

Between the two models and each of their threshold tests, the most accurate model is the SMWT. This time, the threshold with the best performance is 0.1. This is a surprising feature of the LDA model, because setting the threshold lower means that the model is more likely to classify a pixel as a tarp, which by nature may increase the likelihood of false positives. We do see that the false positive rate does rise significantly for the model without transformations but is not as significant for the SMWT.

Next, the ROC curve and AUROC calculation for the SMWT. To illustrate the differences between ROC curves between models at this stage in comparison, both ROC curves are plotted.





As we can see from the plots, the ROC curve for the SMWT is much closer to the top left corner than the ROC curve for the MWNT. By nature, this means the area under the curve is larger for the SMWT. For the MWNT LDA, as TPR increases, FPR increases at a faster rate than the SMWT LDA.

In general, LDA models usually work well when classes are well separated. It's no surprise after the EDA that we have a good performing LDA model (for SMWT) with an excellent looking ROC curve that hugs the upper left corner of the plot and an AUROC of 0.9967.

Quadratic Discriminant Analysis

The third model tested is a quadratic discriminant analysis model. Again, the models are built with caret's train function, the control object is set to 10-fold cross validation, and then the accuracy and kappa statistics are compared.

```
qda_mod <- train(ClassTarp ~ Red + Green + Blue, data = data,
                 method = "qda",
                 trControl = control)

qda_mod_t <- train(ClassTarp ~ Red + Green + Blue +
                       I(Red^2) + log(Red) + log(Green) +
                       log(Blue) + sqrt(Red) + sqrt(Green) +
                       sqrt(Blue) + I(Red^3) + I(Green^3) +
                       I(Blue^3) + I(Green^4) + I(Blue^4),
                 data = data,
                 method = "qda",
                 trControl = control)

rbind(c("Model with No Transformations", qda_mod$results[2], qda_mod$results[3]),
      c("Stepwise Transformations Model", qda_mod_t$results[2], qda_mod_t$results[3]))
```

```
#>                                    Accuracy  Kappa
#> [1,] "Model with No Transformations"  0.9945605 0.9050503
#> [2,] "Stepwise Transformations Model" 0.9692447 0.6599745
```

This time, the results show that MWNT is slightly more accurate than the MWNT. Additional threshold tests and comparisons are performed next.

```
qda_mod_thres <- thresh_test(qda_mod)
qda_mod_thres[2:9] %>% slice_max(Accuracy)
```

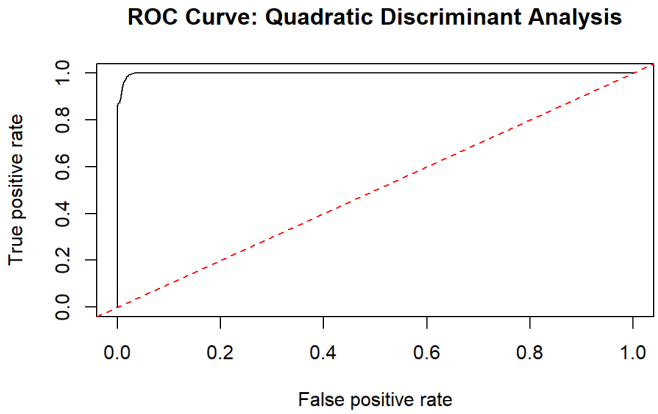
```
#>   prob_threshold  Accuracy     Kappa Sensitivity Specificity Precision
#> 1            0.7 0.9947502 0.9097009   0.9993303    0.856082 0.9952663
#>       falseNeg falsePos
#> 1 0.0006697211 0.143918
```

```
qda_mod_t_thres <- thresh_test(qda_mod_t)
qda_mod_t_thres[2:9] %>% slice_max(Accuracy)
```

```
#>   prob_threshold  Accuracy     Kappa Sensitivity Specificity Precision
#> 1            0.1 0.9742098 0.6990632   0.9735867   0.9930791  0.999765
#>     falseNeg    falsePos
#> 1 0.02641334 0.006920938
```

Between the two models and each of their threshold tests, the most accurate model is the model without transformations with a threshold of 0.7. This time, a model with a high threshold ends up being our most accurate, but it's important to take note of the false positive rate. This model's FPR ends up being the highest of all the final models we compare at 0.1439. The significance of this is discussed further in the conclusions of the report.

Next, the ROC curve and AUROC calculation.




```
#>   prob_threshold  Accuracy     Kappa Sensitivity Specificity Precision
#> 1            0.7 0.9947502 0.9097009   0.9993303    0.856082 0.9952663
#>       falseNeg falsePos     AUROC
#> 1 0.0006697211 0.143918 0.9981946
```

QDA models tend to work well on data with more complex decision boundaries and imbalanced classes. While the model does perform well with a healthy looking ROC curve and an AUROC of 0.9982, it is important to consider the cost of a high false positive rate.

K Nearest Neighbors

The fourth model tested is a k Nearest Neighbors model. The models are built with caret's train function, the control object is set to 10-fold cross validation, and then the accuracy and kappa statistics are compared. In addition, the k parameter is tuned to find the best performing model. Here, a collection of k values is tested based on experimentation and to show a plot later on. The k values of 125 and 250 were used to see if common practice of using a square root of the number of observations (or 1/2 the square root) would be a good choice for this model. Also, a plot will be produced to how show the the accuracy across values of k.

```
clust <- makeCluster(detectCores() - 6)
registerDoParallel(clust)

klist <- data.frame(k = c(1, 5, 25, 31, 51, 125, 250))

knn_mod <- train(ClassTarp  ~ Red + Green + Blue, data = data,
                 tuneGrid = klist,
                 method = "knn",
                 metric = "Accuracy",
                 trControl = control)


knn_mod_t <- train(ClassTarp ~ Red + Green + Blue +
                       I(Red^2) + log(Red) + log(Green) +
                       log(Blue) + sqrt(Red) + sqrt(Green) +
                       sqrt(Blue) + I(Red^3) + I(Green^3) +
                       I(Blue^3) + I(Green^4) + I(Blue^4),
                 data = data,
                 tuneGrid = klist,
                 method = "knn",
                 metric = "Accuracy",
                 trControl = control)

#stop cluster
stopCluster(clust)
```

First, we'll compare the most accurate k value for each model.

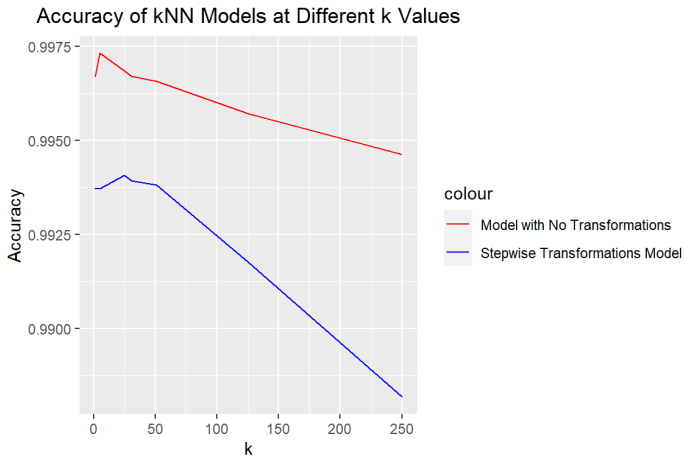
```
# most accurate k value for no transformation model
knn_mod$results %>% slice_max(Accuracy)
```

```
#>   k  Accuracy     Kappa   AccuracySD     KappaSD
#> 1 5 0.9973277 0.9570074 0.0006121549 0.009914947
```

```
# most accurate k value for stepwise transformation model
knn_mod_t$results %>% slice_max(Accuracy)
```

```
#>    k  Accuracy     Kappa   AccuracySD    KappaSD
#> 1 25 0.9940703 0.9048098 0.0008036978 0.01291266
```

We can explore accuracy across multiple kNN models by visualizing accuracy across k values. Below, we see that the model with the highest accuracy across k values is the MWNT at k = 5.

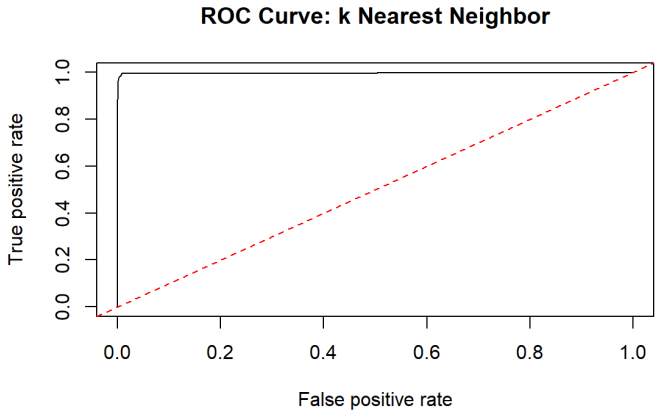


Interestingly, the SMWT consistently underperforms the MWNT across all k values. This is a good example of the curse of dimensionality. The SMWT model has 15 predictors, while the MWNT model only has 3. The additional predictors in the SMWT model are not adding any value to the model, and are in fact making the model less accurate. Hence, a good example of why it is important to be careful when adding predictors depending on the model used. For this situation, MWNT outperforms SMWT and k values in a lower range seem to perform better than higher k values.

Next, we conduct a threshold test on the most accurate kNN model.

```
#>   k prob_threshold  Accuracy     Kappa Sensitivity Specificity Precision
#> 1 5            0.4 0.9972961 0.9563593   0.9985626   0.9589475 0.9986445
#>      falseNeg   falsePos
#> 1 0.001437447 0.04105253
```

The threshold with the highest accuracy for the MWNT and k = 5 is 0.4.




```
#>   k prob_threshold  Accuracy     Kappa Sensitivity Specificity Precision
#> 1 5            0.4 0.9972961 0.9563593   0.9985626   0.9589475 0.9986445
#>      falseNeg   falsePos     AUROC
#> 1 0.001437447 0.04105253 0.9971263
```

kNN models tend to work well when the data has localized patterns considering that this type of model relies on the idea that data points that are near one another tend to be of the same class. This is generally the case with our tarp data points.

The kNN model performs extremely well. The ROC curve is close to perfect and the AUROC is 0.9971.

Penalized Logistic Regression (Elastic Net Penalty)

The fifth model tested is a penalized logistic regression. The models are built with caret's train function, the control object is set to 10-fold cross validation, and then the accuracy and kappa statistics are compared.

Additionally, PLR model hyperparameters, alpha and lambda, must be tuned with the tuneGrid argument.

Alpha is the elastic net mixing parameter, and is a value between 0 and 1. This sets the penaly term combination between L1 (lasso regression) and L2 regularization (ridge regression). Alpha is about penalty balance.

Lambda is the regularization parameter that controls the strength of the penalty, and is a value between 0 and 1. The higher lambda is, the more features shrink toward a simpler model. Lambda is about penalty strength.

Considering the concern of overfitting stated when constructing the model with a large number of transformed variables, one may hypothesize that this will control for potential overfitting and be quite a good model.

To tune alpha and lambda, a grid of values is created to test for each hyperparameter. Then, caret's train function tunes the hyperparameters and accuracy is compared.

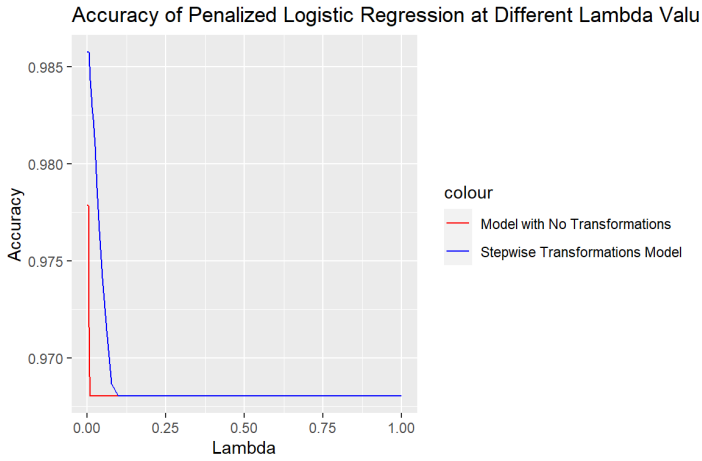
```
clust <- makeCluster(detectCores() - 6)
registerDoParallel(clust)
l_grid <- expand.grid(alpha = 0, lambda = 10^seq(-10, 0, length.out = 100))

plr_mod <- train(ClassTarp ~ Red + Green + Blue, data = data,
                 method = "glmnet",
                 tuneGrid = l_grid,
                 trControl = control)


stopCluster(clust)
```

MWNT:

```
clust <- makeCluster(detectCores() - 6)
registerDoParallel(clust)
plr_mod_t <- train(ClassTarp ~ Red + Green + Blue +
                        I(Red^2) + log(Red) + log(Green) +
                        log(Blue) + sqrt(Red) + sqrt(Green) +
                        sqrt(Blue) + I(Red^3) + I(Green^3) +
                        I(Blue^3) + I(Green^4) + I(Blue^4),
                        data=data,
                        method="glmnet",
                        tuneGrid=l_grid,
                        trControl=control)

stopCluster(clust)
```

SMWT:

We'll explore accuracy across multiple lambda values with the following figure.



Here we see that both models have multiple values of lambda near zero that are tied for highest accuracy. This is expected considering that the models are penalized logistic regressions, and the lower the lambda value, the less the penalty.

Here, the SMWT outperforms the MWNT across the board.

Next, we'll test the alpha hyperparameter for the SMWT with a locked in lambda value of 1e-10, which was at the top of the list.

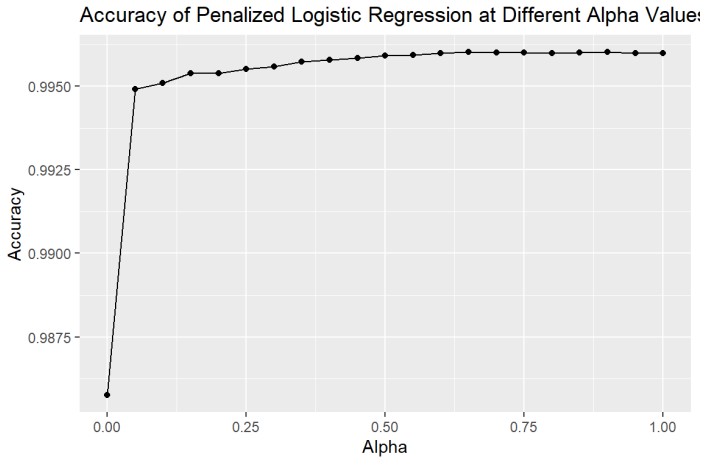
```
clust <- makeCluster(detectCores() - 6)
registerDoParallel(clust)
a_grid <- expand.grid(alpha = seq(0, 1, 0.05), lambda = 1.000000e-10)

plr_mod_z <- train(ClassTarp ~ Red + Green + Blue +
                        I(Red^2) + log(Red) + log(Green) +
                        log(Blue) + sqrt(Red) + sqrt(Green) +
                        sqrt(Blue) + I(Red^3) + I(Green^3) +
                        I(Blue^3) + I(Green^4) + I(Blue^4),
                        data = data,
                        method = "glmnet",
                        tuneGrid = a_grid,
                        trControl = control)
stopCluster(clust)
```

```
plr_mod_z$results %>% slice_max(Accuracy)
```

```
#>   alpha lambda  Accuracy     Kappa   AccuracySD    KappaSD
#> 1  0.65  1e-10 0.9960311 0.9349648 0.0009081827 0.01494441
#> 2  0.90  1e-10 0.9960311 0.9350494 0.0009020436 0.01487137
```

With a lambda of 1e-10, models with two different alpha values tie as the most accurate at alpha = 0.65 and alpha = 0.90. Considering the minimal difference beetween the two and to keep it simple, we'll choose alpha = 0.65.

We'll also plot this model's accuracy across multiple alpha values.



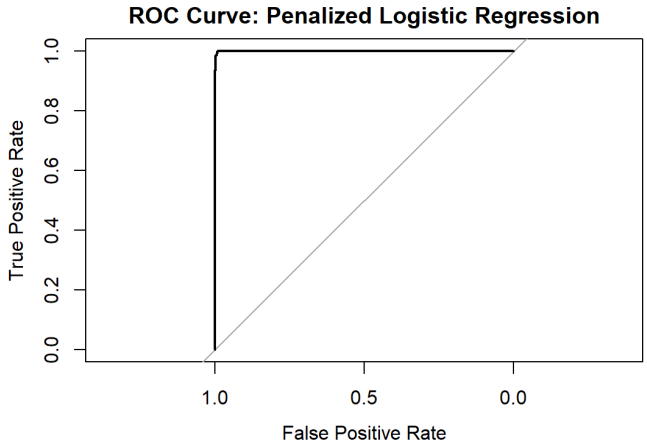
Next, we'll test the threshold for classification for the SMWT with a locked in lambda value of 1e-10 and alpha value of 0.65.

```
plr_mod_z_thres <- thresh_test(plr_mod_z)
plr_mod_z_thres %>% slice_max(Accuracy)
```

```
#>   alpha lambda prob_threshold  Accuracy     Kappa Sensitivity Specificity
#> 1  0.65  1e-10           0.7 0.9963631 0.9423115   0.9975335   0.9609252
#>   Precision    falseNeg   falsePos
#> 1 0.9987082 0.002466544 0.03907477
```

The most accurate threshold for this model ended up being 0.7, which matches the threshold for QDA as well. In comparison with QDA, though, we don't notice the same high false positve rate.

Finally, we'll generate the ROC curve and AUROC for this model with the alpha value set at 0.65, lambda value set at 1e-10, and threshold set at 0.7.



It's important to note for PLR, that with a lambda value of 1e-10, the model experiences essentially no regularization, essentially making this model a standard logistic regression. Even still, we have an excellent looking ROC curve with an AUROC of 0.9996, the highest of all the models we've test.

Typically PLR tends to work well with data that includes numerous predictor variables. The benefit of using PLR is the protection that it provides against possible overfitting and multicollinearity issues, which is exactly the problem we knew we needed to avoid with all the transformations used.

Random Forest

The sixth model tested is a random forest. The models are built with caret's train function, the control object is set to 10-fold cross validation, and then the accuracy and kappa statistics are compared.

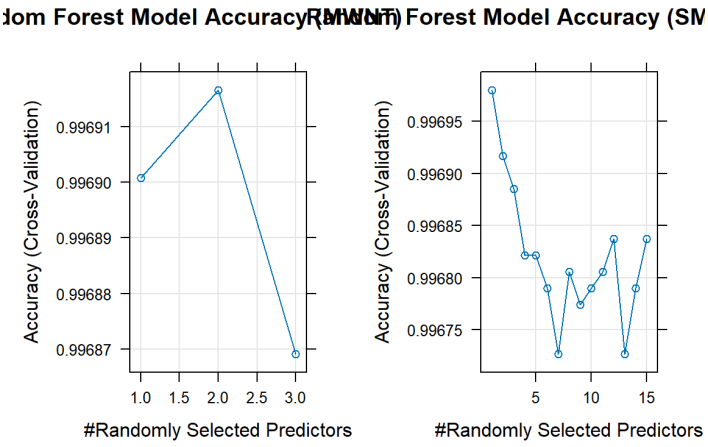
With the random forest model, we'll also have to decide on the number of feature variables we will try for each node. Typically, the square root of the total number of variables is used.

Here, we'll use the total amount of variables for each model, which is a bit over the top, but we'll will be able to plot our findings later.

```
clust <- makeCluster(detectCores() - 8)
registerDoParallel(clust)

tuneGrid <- expand.grid(.mtry = c(1,2,3))

rf_mod <- train(ClassTarp ~ Red + Green + Blue, data = data,
                  method = "rf",
                  trControl = control,
                  tuneGrid = tuneGrid)

stopCluster(clust)
```

```
clust <- makeCluster(detectCores() - 8)
registerDoParallel(clust)

tuneGrid <- expand.grid(.mtry = c(1,2,3,4,5,6,7,8,9,10,11,12,13,14,15))
rf_mod_t <- train(ClassTarp ~ Red + Green + Blue +
                        I(Red^2) + log(Red) + log(Green) +
                        log(Blue) + sqrt(Red) + sqrt(Green) +
                        sqrt(Blue) + I(Red^3) + I(Green^3) +
                        I(Blue^3) + I(Green^4) + I(Blue^4),
                  data = data,
                  method = "rf",
                  trControl = control,
                  tuneGrid = tuneGrid)

stopCluster(clust)
```

```
# for caching purposes
clust <- makeCluster(detectCores() - 8)
registerDoParallel(clust)

tuneGrid <- expand.grid(.mtry = 1)

rf_mod_final <- train(ClassTarp ~ Red + Green + Blue, data = data,
                  method = "rf",
                  trControl = control,
                  tuneGrid = tuneGrid)

stopCluster(clust)
```



The MWNT performs best with an mtry value of 1, with in accuracy of 0.99695. The SMWT performs just below this accuracy rate with an mtry value of 1 at an accuracy of 0.99693. Now for threshold tests and further comparisons.

```
rf_mod_thres <- thresh_test(rf_mod)
rf_mod_thres[2:9] %>% slice_max(Accuracy)
```

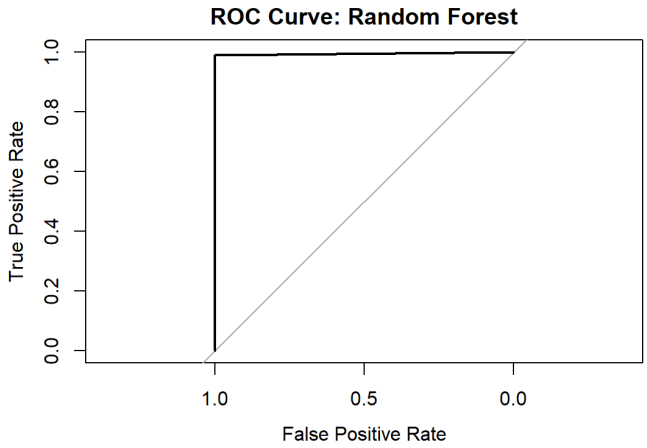
```
#>   prob_threshold  Accuracy     Kappa Sensitivity Specificity Precision
#> 1            0.6 0.9969324 0.9507921   0.9982032   0.9584549 0.9986275
#>     falseNeg   falsePos
#> 1 0.00179682 0.04154514
```

```
rf_mod_t_thres <- thresh_test(rf_mod_t)
rf_mod_t_thres[2:9] %>% slice_max(Accuracy)
```

```
#>   prob_threshold  Accuracy     Kappa Sensitivity Specificity Precision
#> 1            0.6 0.9970272 0.9521735   0.9983175   0.9579525 0.9986115
#>     falseNeg   falsePos
#> 1 0.00168247 0.04204751
```

Between the two models and each of their threshold tests, the MWNT sneaks ahead in accuracy by a pinch at a threshold of 0.6, and slightly outperforms on FPR as well.

Next, the ROC curve and AUROC calculation.



Random forest models tend to work well on data with outliers and non-linear relationships. It is also a good model to use for high dimensional data.

The model performs extremely well. The ROC curve is near flawless and the AUROC measures in at 0.9999.

Support Vector Machine

The final model tested is a support vector machine. The models are built with caret's train function, the control object is set to 10-fold cross validation, and then the accuracy and kappa statistics are compared.

First, we must decide on what type of kernal to use. To make our decision, we'll train the two models with different kernel types with default parameters, then fine-tune the best performing model.

Notedly, according to `caret` 's documentation, the sigmoid kernel is not an available model. For the purpose of this project, we will continue with testing linear, polynomial, and radial.

The accuracies were:

SVM linear, MWNT: 0.9969324; SVM poly, MWNT: 0.9959678; SVM radial, MWNT: 0.9969482

SVM linear, SMWT: 0.9961734; SVM poly, SMWT: 0.9962999; SVM radial, SMWT: 0.9969324

The most accurate model is the MWNT model with a radial kernel.

For SVM, the main parameters to tune are cost (C) and gamma (sigma). C is a cost parameter and it defines how heavily we penalize datapoints that are classified incorrectly. A small C value will result in a smoother decision boundary and a large C value will more tightly fit the data. Gamma determines the reach of a single training instance. A small gamma value will result in each training instance having a large influence on the decision boundary, and high gamma values will result in each training instance having a smaller influence on the decision boundary.

Here, we use a grid search to find the best combination of these two parameters. We will be using different values of C and gamma on a logarithmic scale to get a better idea of the best combination of parameters.

```
clust <- makeCluster(detectCores() - 6)
registerDoParallel(clust)
start <- Sys.time()
svm_mod_final2 <- train(ClassTarp ~ Red + Green + Blue, data = data,
                  method = "svmRadial",
                  trControl = control,
                  tuneGrid = expand.grid(sigma = 10,
                                         C = 100))
end <- Sys.time()
time <- end - start
stopCluster(clust)
time
```

```
#> Time difference of 44.30459 secs
```

```
svm_mod_final2$results %>% slice_max(Accuracy)
```

```
#>   sigma   C  Accuracy     Kappa   AccuracySD    KappaSD
#> 1    10 100 0.9973277 0.9566144 0.0007243727 0.01174378
```

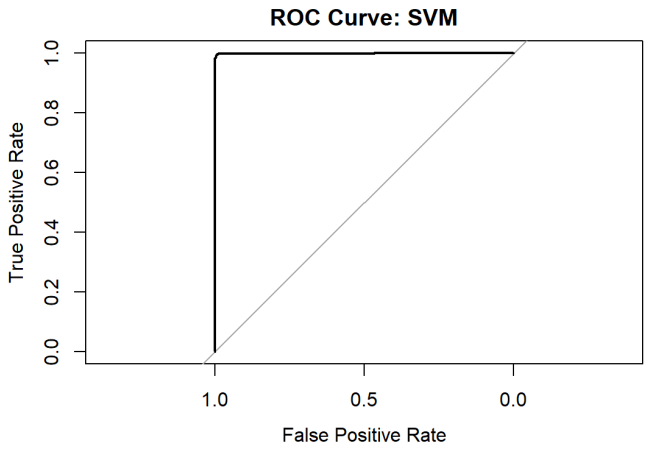
According to the grid search, the best combination of cost and gamma are 100 and 10, respectively, with in accuracy of 0.9973. Now for threshold tests and further comparisons.

```
svm_mod_thres <- thresh_test(svm_mod_final2)
svm_mod_thres[2:9] %>% slice_max(Accuracy)
```

```
#>     C prob_threshold  Accuracy     Kappa Sensitivity Specificity Precision
#> 1 100            0.6 0.9974226 0.9582965   0.9987096   0.9584549 0.9986283
#>      falseNeg
#> 1 0.001290436
```

Our threshold test shows that the best threshold is 0.9, with an accuracy of 0.9974. As discussed previously, this is a very high threshold which seems like it would risk an increase the number of false positives, but the threshold test shows a very accurate performance.

Next, the ROC curve and AUROC calculation.




```
#> Area under the curve: 0.9987
```

SVM models can work well on data that has complex decision boundaries, yet clear separation margins. As we see here, the SVM model works well on our dataset with an accuracy of 0.9974 and an AUROC of 0.9987.

Cross Validation Model Performance Table Results

A table summarizing key characteristics and performance metrics for each model with the initial data can be found below.

Cross Validation Model Performance Table

Model	Transformed	Tuning	Threshold	Accuracy	AUROC	TPR	FPR	FNR	Precision
Logistic Regression	Yes		0.5	0.9968	0.9856	0.9985	0.0697	0.0015	0.9977
LDA	Yes		0.1	0.9889	0.9967	0.9917	0.0950	0.0083	0.9968
QDA	No		0.7	0.9948	0.9982	0.9993	0.1439	0.0007	0.9953
KNN	No	k = 5	0.4	0.9973	0.9971	0.9986	0.0411	0.0014	0.9986
PLR	Yes	alpha = 0.65, lambda = 1e-10	0.7	0.9964	0.9996	0.9975	0.0390	0.0025	0.9609
RF	No	mtry = 1	0.6	0.9970	0.9999	0.9983	0.0425	0.0016	0.9986
SVM (Radial)	No	sigma = 10, C = 100	0.9	0.9974	0.9987	0.9983	0.0312	0.0017	0.9990

Hold Out Data Ingestion

The data used for the hold out tests is formatted a bit differently than the data used for the cross validation tests. First, it needs to be consolidated, files need to be checked as duplicates, then the dataframe needs to be transformed in the same way as the cross validation data. Finally, the data is checked for missing values. There are none.

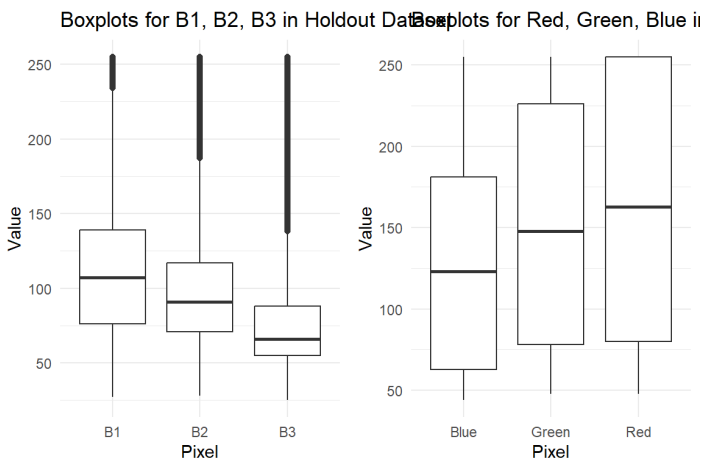
The new dataset includes coordinates for pixel locations as well as pixel values. Unfortunately, the pixel value columns are unlabeled, so we perform a bit of EDA to figure out which columns are the pixel values `Red`, `Green` and `Blue`.

Hold Out Data EDA

We start by comparing the summary statistics and boxplots of the columns between the datasets to get a hint at which of the columns is `Red`, `Green` and `Blue`.

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

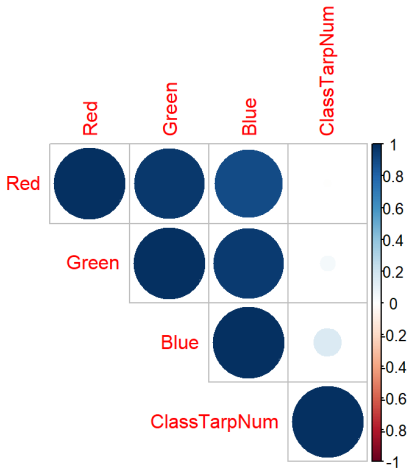
```
#>        B1              B2              B3       
#>  Min.   : 27.0   Min.   : 28.0   Min.   : 25.00  
#>  1st Qu.: 76.0   1st Qu.: 71.0   1st Qu.: 55.00  
#>  Median :107.0   Median : 91.0   Median : 66.00  
#>  Mean   :118.3   Mean   :105.4   Mean   : 82.36  
#>  3rd Qu.:139.0   3rd Qu.:117.0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



Based on similarities in the summary statistics and boxplots, we can assume that `B1` is `Red`, `B2` is `Green`, and `B3` is `Blue` by following the logic that `B1` and `Red` have the highest mean, median, boxplot, `B2` and `Green` have the second highest, and `B3` and `Blue` have the lowest.

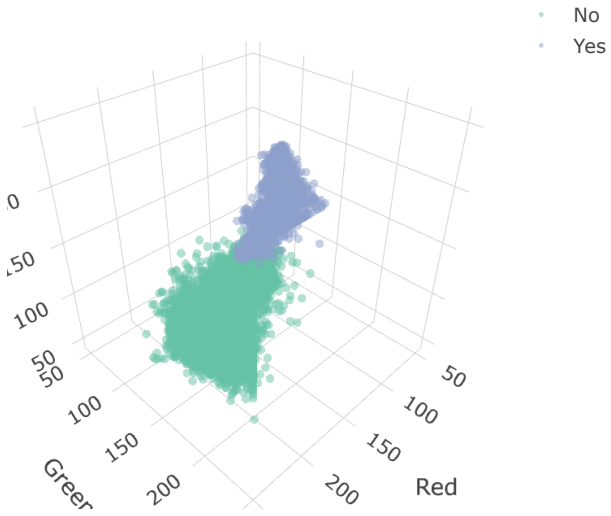
Now we know that `B1` is `Red`, `B2` is `Green`, and `B3` is `Blue`. We will rename the columns in the holdout dataset to reflect this. We're assuming that the imagery from both datasets is similar in nature, otherwise these models would be useless. For example, if the imagery from the holdout dataset was taken in the late evening, early morning, or at night, the spectrum of RGB values would be quite different between datasets and therefore render the models ineffective.

To verify, we can explore the corrplot of the RGB values against `ClassTarp` and compare them to the original dataset.



This looks exactly like the original dataset, with the strongest correlation between `ClassTarp` and `Blue`, then `Green`, then `Red`.

Although we do not see the 3D plot categorized further than whether or not it is a tarp, we can see that the plot below is extremely similar to the original dataset.



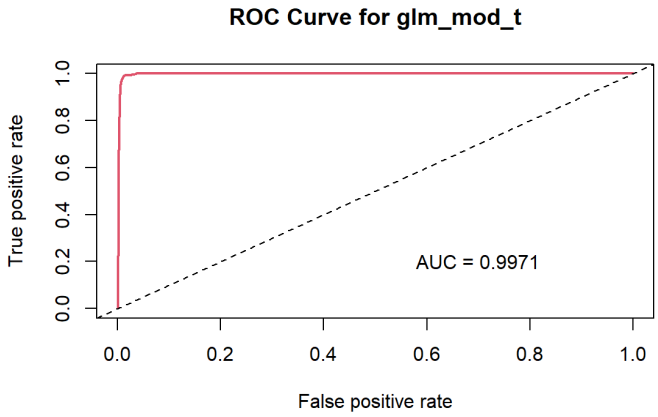
Hold Out Data Model Performance Results

With holdout data now in hand, we can assess the performance of these models and see how their performance fairs on new data. First, a function is created to assess the performance of the model on the holdout data by calculating the performance metrics as well as plotting the ROC curve. A list of tuning parameters is also set up to be used in the function.

```r
# this is a function to do multiple model assessments with the new holdout data
# input: model to plot, model tuning parameters
# output: model performance stats and ROC plot
assess_mod <- function(model, mod_tuning) {
      # predict on holdout data
      prob <- predict(model, newdata=data_ho, type="prob")
      pred <- as.factor(ifelse(prob$Yes > mod_tuning$prob_threshold, "Yes", "No"))
      rate <- prediction(prob[,2], data_ho$ClassTarp)
      auc <- performance(rate, "auc")
      conf <- confusionMatrix(pred, data_ho$ClassTarp, positive = "Yes")
      Tuning <- NaN
      Tuning2 <- NaN

      # Extract tuning parameters
      if("alpha" %in% names(mod_tuning)) Tuning <- mod_tuning[["alpha"]]
      else if("k" %in% names(mod_tuning)) Tuning <- mod_tuning[["k"]]
      else if("mtry" %in% names(mod_tuning)) Tuning <- mod_tuning[["mtry"]]
      else if("C" %in% names(mod_tuning)) Tuning <- mod_tuning[["C"]]

      if("lambda" %in% names(mod_tuning)) Tuning2 <- mod_tuning[["lambda"]]
      else if("sigma" %in% names(mod_tuning)) Tuning2 <- mod_tuning[["sigma"]]

      # f1 score
      Precision <- conf$byClass[["Precision"]]
      Recall <- conf$byClass[["Sensitivity"]]
      F1 <- 2 * (Precision * Recall) / (Precision + Recall)

      stats <- data.frame(Tuning = Tuning,
                     Tuning2 = Tuning2,
                     AUROC = auc@y.values[[1]],
                     Threshold = mod_tuning$prob_threshold,
                     Accuracy = conf$overall[["Accuracy"]],
                     TPR = conf$byClass[["Sensitivity"]],
                     FPR = 1 - conf$byClass[["Specificity"]],
                     Precision = conf$byClass[["Precision"]],
                     F1 = F1)

      # roc plot
      roc_perf <- performance(rate, measure = "tpr", x.measure = "fpr")
      plot(roc_perf, main=paste("ROC Curve for", deparse(substitute(model))), col=2, lwd=2)
      abline(a=0, b=1, lty=2)
      text(0.7, 0.2, paste("AUC =", round(auc@y.values[[1]], 4)))

      return(stats)
}
```

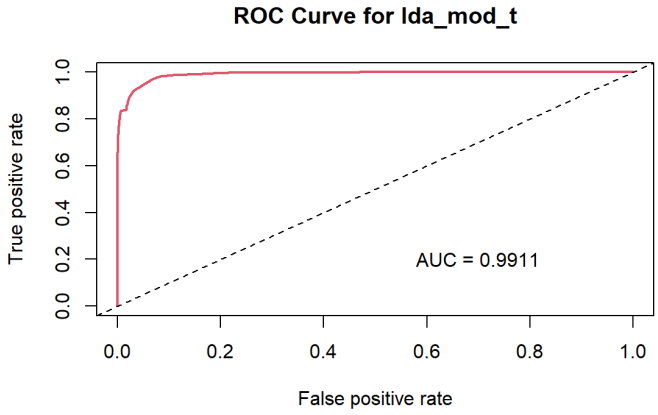
Findings are printed first, then discussed below.

Logistic Regression



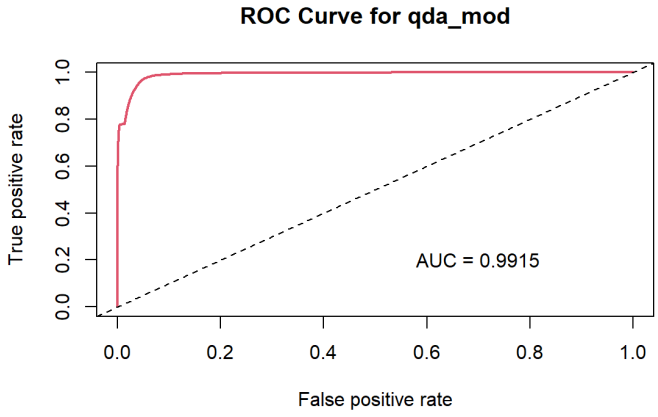

```
#>   Tuning Tuning2     AUROC Threshold  Accuracy       TPR        FPR Precision
#> 1    NaN     NaN 0.9970766       0.5 0.9872212 0.9907459 0.01280446 0.3602441
#>          F1
#> 1 0.5283686
```

Linear Discriminant Analysis



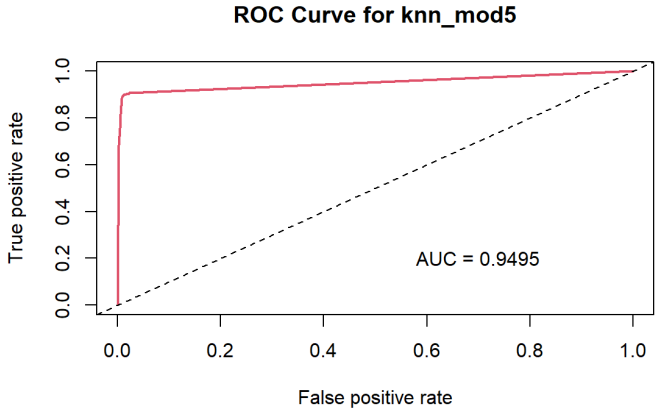

```
#>   Tuning Tuning2     AUROC Threshold  Accuracy       TPR        FPR Precision
#> 1    NaN     NaN 0.9911449       0.1 0.9383423 0.9611878 0.06182399 0.1016439
#>          F1
#> 1 0.1838464
```

Quadratic Discriminant Analysis



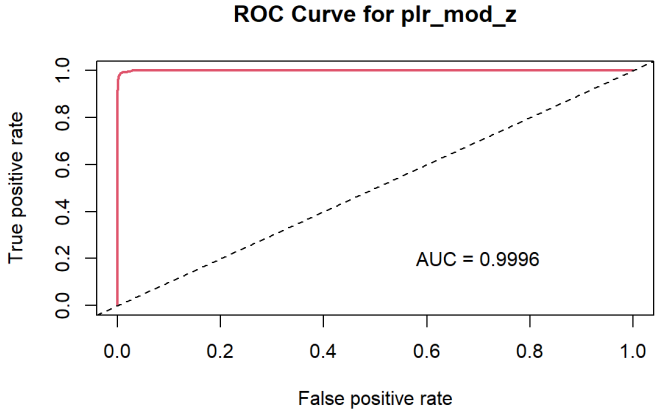
```
#>   Tuning Tuning2     AUROC Threshold  Accuracy       TPR         FPR Precision
#> 1    NaN     NaN 0.9915001       0.7 0.9961595 0.6540055 0.001350457 0.7789751
#>         F1
#> 1 0.711041
```

K-Nearest Neighbors



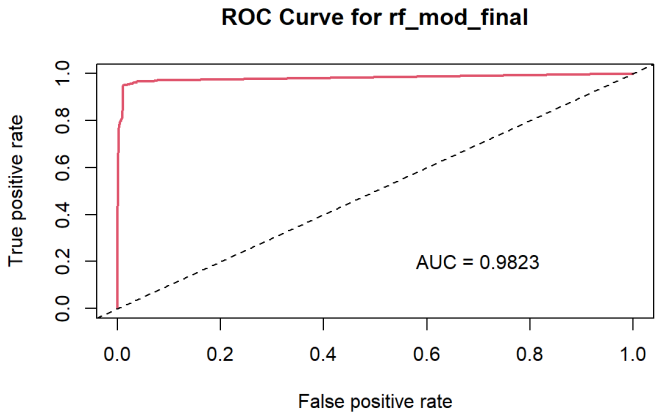

```
#>   Tuning Tuning2     AUROC Threshold  Accuracy      TPR         FPR Precision
#> 1      5     NaN 0.9494872       0.4 0.9921384 0.819337 0.006604021 0.4744841
#>          F1
#> 1 0.6009523
```

Penalized Logistic Regression



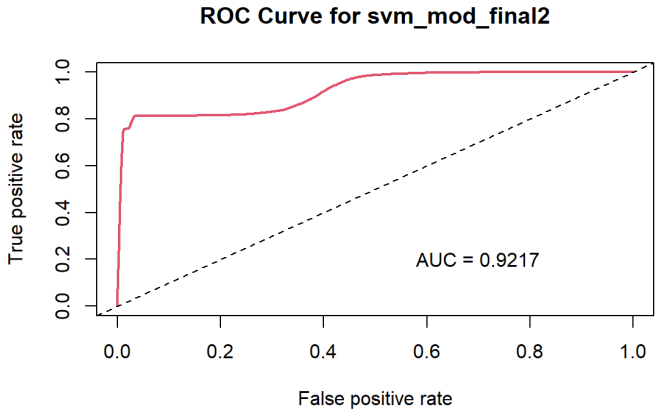

```
#>   Tuning Tuning2     AUROC Threshold  Accuracy      TPR         FPR Precision
#> 1   0.65       0 0.9996153       0.7 0.9970177 0.976174 0.002830582 0.7150807
#>          F1
#> 1 0.8254738
```

Random Forest




```
#>   Tuning Tuning2    AUROC Threshold  Accuracy       TPR         FPR Precision
#> 1      1     NaN 0.982338       0.6 0.9955742 0.7074586 0.002328998 0.6885334
#>          F1
#> 1 0.6978677
```

Support Vector Machine




```
#>   Tuning Tuning2     AUROC Threshold  Accuracy       TPR         FPR Precision
#> 1    100      10 0.9217022       0.9 0.9909609 0.3475138 0.004356442 0.3672993
#>          F1
#> 1 0.3571327
```

Holdout Data Model Performance Table

A table summarizing key characteristics and performance metrics with the holdout data for each model can be found below.

Model	Transformed	Tuning	Threshold	Accuracy	AUROC	TPR	FPR	Precision	F1
Logistic Regression	Yes		0.5	0.9872212	0.9970766	0.9907459	0.0128045	0.3602441	0.528368598419977
LDA	Yes		0.1	0.9383423	0.9911449	0.9611878	0.0618240	0.1016439	0.183846402789795
QDA	No		0.7	0.9961595	0.9915001	0.6540055	0.0013505	0.7789751	0.711041033149379
KNN	No	k = 5	0.4	0.9921384	0.9494872	0.8193370	0.0066040	0.4744841	0.600952284469659

Model	Transformed	Tuning	Threshold	Accuracy	AUROC	TPR	FPR	Precision	F1
PLR	Yes	alpha = 0.65, lambda = 1e-10	0.7	0.9970177	0.9996153	0.9761740	0.0028306	0.7150807	0.825473764125325
RF	No	mtry = 1	0.6	0.9955742	0.9823380	0.7074586	0.0023290	0.6885334	0.697867702159548
SVM (Radial)	No	sigma = 10, C = 100	0.9	0.9909609	0.9217022	0.3475138	0.0043564	0.3672993	0.357132718239886

Conclusions

Throughout the exploration of our data and the models used to predict the presence of tarps, we can begin to understand the many factors to consider and decisions to make when making trying to decide on which model to implement in a real world scenario.

Model Construction

Although the method for model construction for this report was straightforward, it's important to note the potential for improvements in the approach. For every method, two different models were tested. The idea of testing these two models in particular, one with stepwise transformations and one with no transformations, was to see if the transformations would improve the model's performance. This was a very basic and broad approach to feature engineering.

It must be noted that while the shotgun approach of adding numerous transformed variables then performing a stepwise regression was effective, there are certainly other analyses that could be performed to find and fit models with more targeted adjustments.

Other feature engineering techniques would be interesting to explore and may potentially improve model performance. In fact, different models may benefit from very different feature engineering techniques. For example, what would happen to the performance of these models if we introduced binning or interaction terms? It may prove beneficial to approaching each method with carte blanche and experimenting from scratch as opposed to testing the same two models across the board.

This would allow for the use of a more targeted approach to find the best transformations and may potentially improve the performance of the models even further.

Model Assessment

Optimization for Accuracy

The methods for determining best model performance are an area that would benefit from further exploration and stakeholder input. In this project, the models were primarily judged on their accuracy, but there are other important metrics and factors to consider when determining the best model. These factors can also depend heavily on the context of the use case for the model. Often, making judgements and determinations on model performance will not be as simple as the approach taken here. One must balance a number of details.

For example, in this scenario, we must consider the cost of false positives and false negatives. A false positive (incorrectly predicting the presence of a tarp where there is none) could potentially result in wasted time and resources when both are in low supply. A false negative (incorrectly predicting the absence of a tarp where there is one) could result in a person or persons in need of assistance not receiving it - a potentially tragic outcome. This is an incredibly difficult balance to maintain and requires us to balance the cost of false positives and false negatives and weigh them against each other. This is where statistical learning takes us from prediction to judgement. A very difficult task, and one that would certainly benefit from input from stakeholders and folks with "boots on the ground."

Order of Assessment

Another consideration is the order in which the models are assessed. This project assessed the models in the order in which they were constructed, which was done to avoid bias, understand how each model performed across different methods and make the best use of time.

However, it is possible that this order may have impacted the results. For example, when assessing the Support Vector Machine model, we assessed six models initially (MWNT and SMWT both with each of the three kernels), then assessed the best performing model of that group with different values of the cost and gamma parameters. This approach was taken to avoid assessing too many models, but it is possible that the best performing model was thrown out in the first step. Adjusted gamma and cost parameters may have bumped performance from behind the others in step one to the top of the list in the final step.

Again, the approach used here, while not perfect, was done to make the best use of time and avoid assessing too many models. However, it is possible that the ideal model was thrown out at some point, simply due to the order in which the models were assessed. The solution to this problem is simply time and compute power.

Model Implementation

It must be said that these models may not be perfect (and none are), but they do offer real world value. With the models from this project, we have the ability to predict the presence of a tarp with an extremely high degree of accuracy. This is a valuable tool that could potentially be improved even further with more data, like where the pixels are in relation to one another.

For example, to potentially improve the models' performance, we could consider advanced feature engineering techniques. For instance, we could calculate summary statistics such as the mean, median, or range of tarp sizes in our data. Then, we could create new binary features indicating whether a particular observation is near at least a certain number of pixels (lower limit, median, etc) with a specific range of pixel values for the blue color channel. This would allow the models to consider not

just the individual pixel values, but also their relationships and interactions within a broader context. This may help throw out large or small patches of color in the imagery that are tarp-colored, but are not tarps.

This may also open up methods to determine if the presence of a tarp in one area is correlated with the presence of a tarp in another area, or help us to understand the size of tarps and see if pixels are clustered in ways that correspond to this size or not. But improvements aside, the models from this project are still extremely useful in a real world scenario that may have the potential to improve outcomes from catastophic disasters like the earthquake in Haiti in 2010.

The Best Model

During Cross Validation

Which brings us to this project's determination of the best model: of the models compared here, the assessment approach taken during cross validation determined the the preferred model for this scenario to be the support vector machine (radial kernel) with no additional transformations (MWNT).

This model had the highest AUROC, extremely high accuracy, and the lowest false positive rate. The balance that must be struck between false positives and false negatives is certainly worth noting as discussed further below. It is of the view in this report that the cost of a slightly higher false negative rate was acceptable in comparison to the cost of many more false positives in models with similar or better FNRs, like QDA and KNN.

Balancing Judgement with Logistics

This opinion is also held while considering that there may be efforts in disaster relief that can greatly influence or lower the cost of false negatives in particular. While this may seem counter-intuitive, one may want to consider that it could potentially be possible that even if this model were successful in predicting the presence of all tarps, there almost certainly will not be enough aid to make it to all of the locations. In addition, it is worth considering that aid must logistically be delivered to areas with clusters of tarps in a hub-and-spoke approach, which emphasizes the importance of knowing where the tarps are as opposed to knowing where they are not. The likely scenario is that this model will inform logistical decisions, and Haitians will do a great deal of the work in moving out from their tarps to receive aid.

As mentioned above, the most powerful method for improving any of these models would be to involve the stakeholders and "boots on the ground" in the decisionmaking processes around balancing the cost of false positives and false negatives. In the end, the processes that occur after the model is built are where the real work occurs, and having the input from those who will be performing this work is invaluable.

Upon Testing with Holdout Data

With new data from the holdout set, each of the models was able to be assessed on real performance. These fresh assessments on new data provided very interesting insights. The model with the best performance across a number of factors was the penalized logistic regression with the transformed model (SMWT).

Holdout testing showed that this model had the highest accuracy, the highest AUROC, the second-highest TPR, and the 4th lowest false positive rate. This model also had an impressive F1 score of 0.83, far above the other models.

F1 score is a metric that integrates precision and recall in a single metric and is especially helpful in situations like the current scenario where the cost of false positive and false positives is important to consider. A high F1 score indicates that the model is performing well in both precision and recall, and a low F1 score indicates that the model is performing poorly in one or both of these areas.

Many metrics from the final PLR model stood out as impressive, but it is the opinion of this report that it strikes the balance between false positives and false negatives the best.

All that said, as stated before, stakeholder input will always be crucial in determining next steps. New data tends to come along with new insights about the scenario, and it is important to consider that the model that performs the best will be subject to the input of those who will be using it.

Optimization Alternative

In further considering potential improvements to the methods used here, findings from holdout testing warrants consideration of the most effective method for optimizing the models.

What is clear is that the model with the highest accuracy is not necessarily the best model. Therefore, improvements to our model construction and selection may potentially be made by focusing on different, or a more nuanced mix of performance metrics. For example, what would our results look like with more attention paid to the F1 score? Using the F1 score as a metric for determining the best model may have led to a different model being selected.

Conclusion

In conclusion, many of the models assessed here performed well, but as shown by the holdout testing, the best performing model was the penalized logistic regression with the transformed model (SMWT). Further, the model that was hypothesized to be the best performing model during cross validation (SVM) was actually one of the worst performing models with holdout data.

Here we have an excellent example of why it is important to develop robust methods, have a healthy dose of skepticism, test models on new data, learn from new findings, adjust accordingly, always learn and partner with stakeholder input, and never stop improving.