Disaster Relief Project - Part 1

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.

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, and penalized logistic regression 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.

The reported results include a table of performance statistics and 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 concludes that the Penalized Logistic Regression model with transformed variables emerged as the model that would perform best in this scenario. This conclusion is based on the strength of its performance statistics as well as a consideration of its balance of false positives and false negatives in particular.

Data Loading and Exploratory Data Analysis

Data Loading and Preparation

First, the necessary packages are loaded and the data is read in from a CSV file.

```
library(tidyverse)
library(dplyr)
library(htmlTable)
library(ggplot2)
library(GGally)
library(ROCR)
library(knitr)
library(caret)
library(plotly)
library(gridExtra)
library(glmnet)
library(pROC)
library(kableExtra)
library(corrplot)
library(ggcorrplot)
```

```
data <- read.csv("HaitiPixels.csv")
```

Next, 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

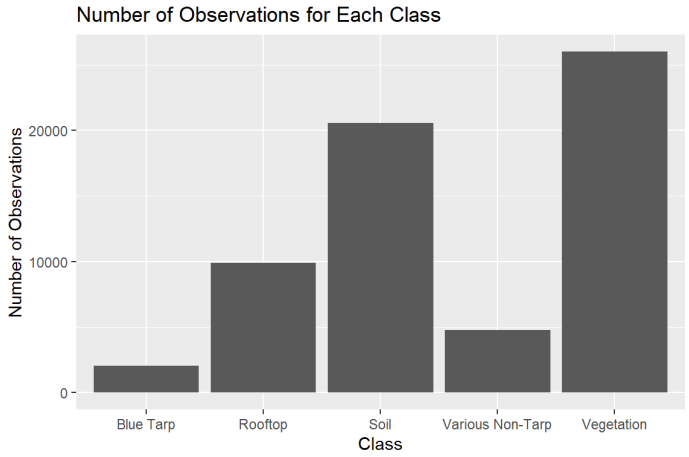
Finally, the data is checked for missing values. There are none.

```
sum(is.na(data))
```

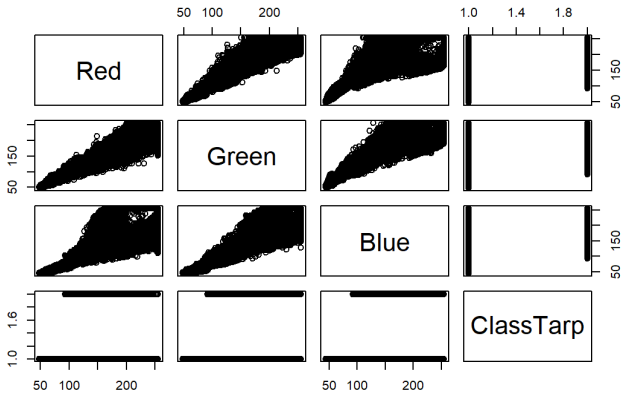
```
#> [1] 0
```

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.

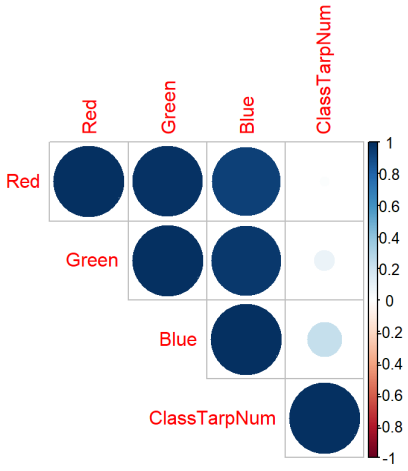


Next, the pairs plots are created to visually explore the relationships between the RGB values.



One interesting tidbit here is the plots of all pixel values against `ClassTarp` that may indicate a possible sigmoid relationship between the RGB values and `ClassTarp`, which is the classical shape of a logistic regression curve. This may indicate that the RGB values are significant in determining whether a pixel is a blue tarp or not.

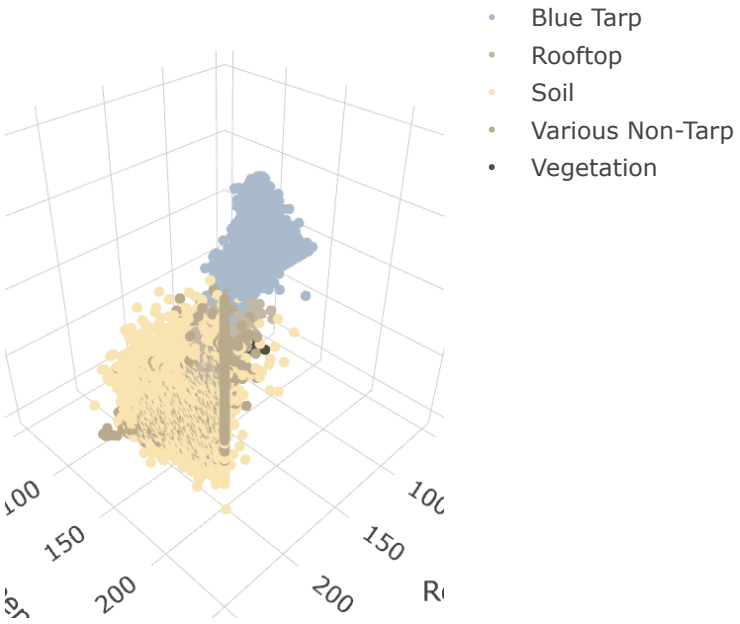
By exploring correlations between the RGB values and `ClassTarp` in the figure below, we can see that `Blue` has the highest correlation with `ClassTarp`, followed by `Green`, although slight. This is also expected, since the tarps we are targeting are blue in color.



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.

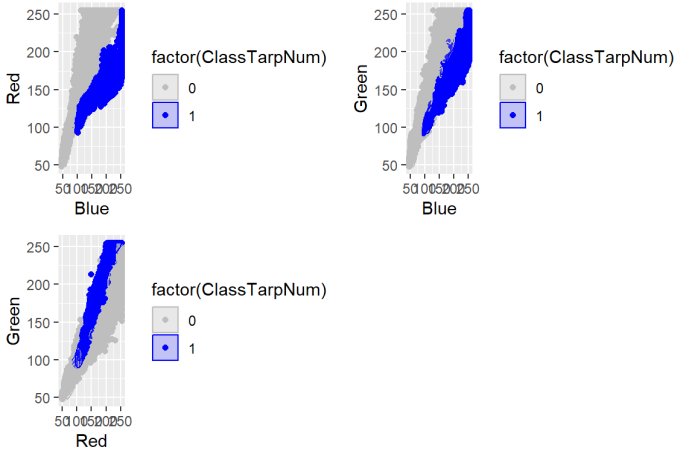
This figure's plots makes it much easier to see the bell curve shaped distribution of the pixel value against `ClassTarp`, unless the pixel value is blue, in which case as the value increases, so does the distribution of blue tarps. Another way to say this is the distribution of `ClassTarp` = Yes skews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

```
# set seed for reproducibility
seed <- 73
set.seed(seed)

# create folds for 10-fold cross validation
folds <- createFolds(data$ClassTarp, k = 10, list = TRUE, returnTrain = TRUE)

# create trainControl object for 10-fold cross validation
control <- trainControl(method = "cv",
                        number = 10,
                        index = folds,
                        savePredictions = TRUE,
                        classProbs = TRUE)
```

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).

```r
# 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)
}
```

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.

```r
# 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
```

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 file.)

```
# 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

Model	AIC	Formula
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.

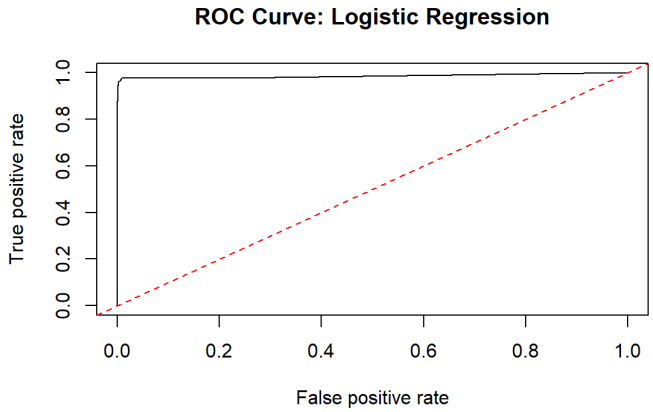
```
# threshold test for glm_mod
glm_mod_thres <- thresh_test(glm_mod)

# 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)
```

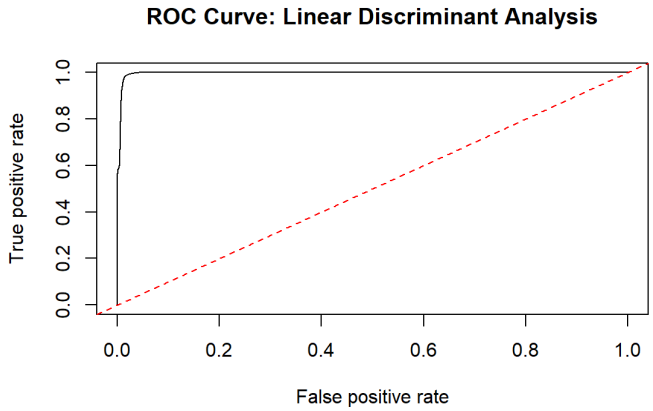
```
#>   prob_threshold Accuracy     Kappa Sensitivity Specificity Precision
#> 1            0.1 0.984646 0.7476611   0.9926493   0.7423353 0.9915003
#>      falseNeg  falsePos
#> 1 0.007350662 0.2576647
```

```
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.




```
#>   prob_threshold  Accuracy    Kappa Sensitivity Specificity Precision
#> 1            0.1 0.9888996 0.833544   0.9916693    0.905048 0.9968474
#>      falseNeg   falsePos     AUROC
#> 1 0.008330736 0.09495196 0.9966999
```

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 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)
```

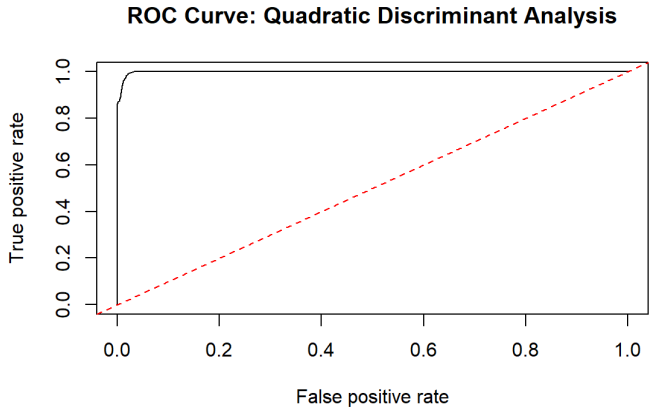
```
#>   prob_threshold  Accuracy     Kappa Sensitivity Specificity Precision
#> 1            0.7 0.9947502 0.9097009   0.9993303    0.856082 0.9952663
#>       falseNeg falsePos
#> 1 0.0006697211 0.143918
```

```
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, k is sequenced from 0 to 50 in intervals of 5.

```
klist <- data.frame(k = seq(0, 50, 5))
klist[1,] <- 1

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)
```

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

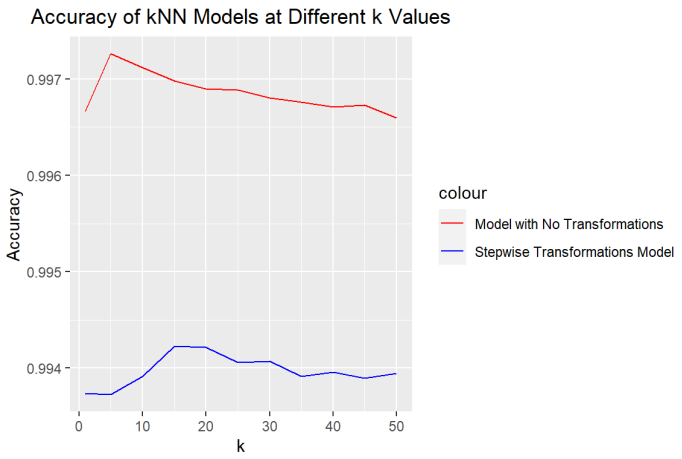
```
# most accurate k value for no transformation model
knn_mod$results %>% slice_max(Accuracy)
```

```
#>   k  Accuracy     Kappa   AccuracySD     KappaSD
#> 1 5 0.9972644 0.9559932 0.0005629855 0.009105698
```

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

```
#>    k  Accuracy     Kappa   AccuracySD    KappaSD
#> 1 15 0.9942284 0.9072502 0.0007898072 0.01284056
```

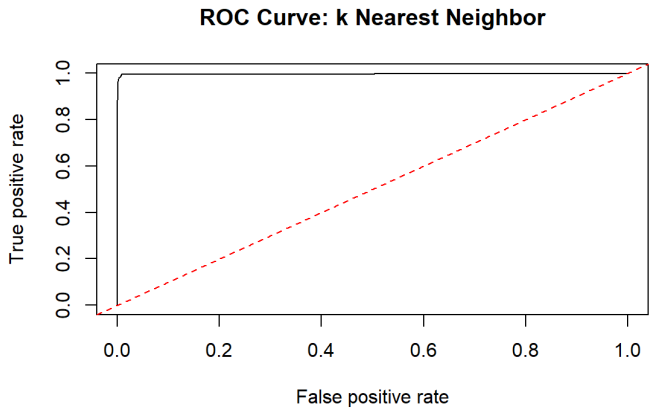
We can explore accuracy across multiple kNN models by visualizing accuracy across k values. Below, we see that the model with the highest accuracy across k values is the MWNT at k = 5.



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 fourth 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.

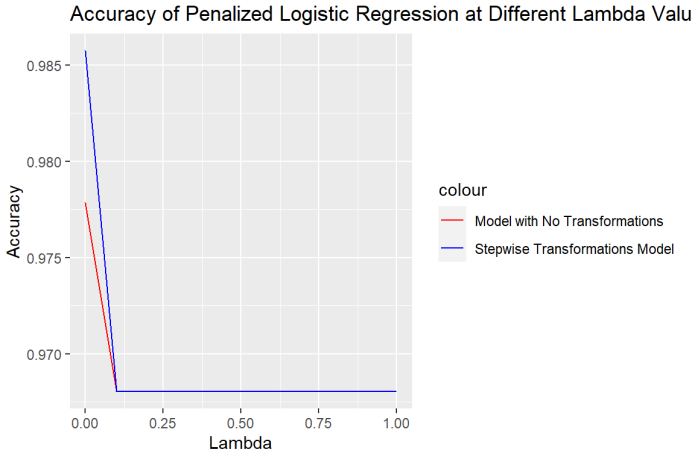
```
l_grid <- expand.grid(alpha = 0, lambda = seq(0, 1, 0.1))

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

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

```
#>   alpha lambda  Accuracy     Kappa  AccuracySD    KappaSD
#> 1     0      0 0.9778625 0.4608508 0.001471737 0.05282817
```

```
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)

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

```
#>   alpha lambda  Accuracy     Kappa  AccuracySD    KappaSD
#> 1     0      0 0.9857529 0.7057049 0.001403216 0.03823686
```

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



Here we see that both models are most accurate at lambda = 0, and the model with transformations has a higher accuracy than the model without transformations.

Next, we'll test the alpha hyperparameter for the SMWT with a locked in lambda value of 0.

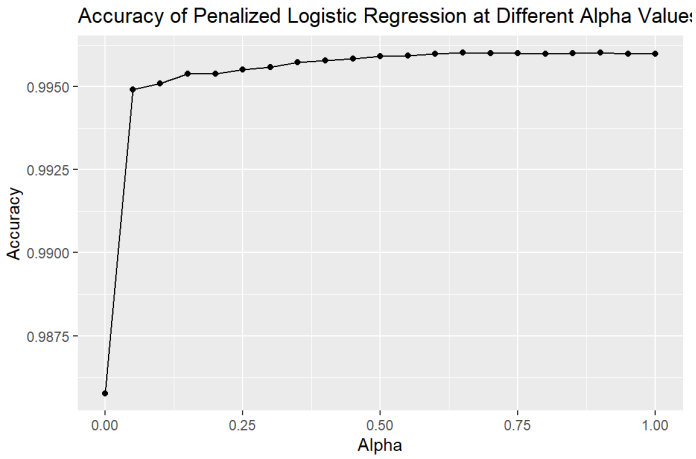
```
a_grid <- expand.grid(alpha = seq(0, 1, 0.05), lambda = 0)

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,
                        savePredictions = FALSE)  # not saving predictions to fix roc_auc_plot er
ror?
```

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

```
#>   alpha lambda  Accuracy     Kappa   AccuracySD    KappaSD
#> 1  0.65      0 0.9960311 0.9349648 0.0009081827 0.01494441
#> 2  0.90      0 0.9960311 0.9350494 0.0009020436 0.01487137
```

With a lambda of 0, 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.



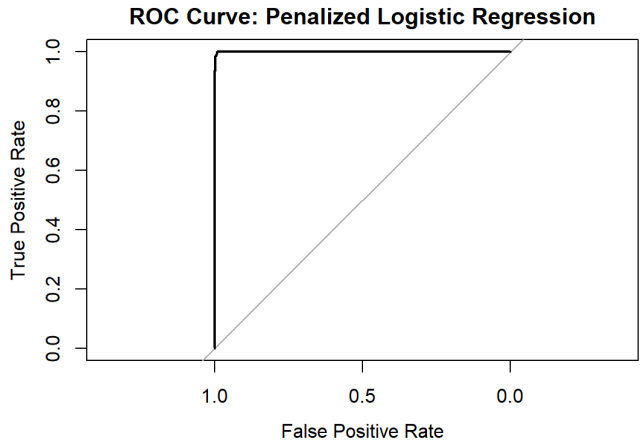
Next, we'll test the threshold for classification for the SMWT with a locked in lambda value of 0 and alpha value of 0.65.

```
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      0            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 0, and threshold set at 0.7.



Here we have another excellent looking ROC curve with an AUROC of 0.9996, the highest of all the models we've test.

PLR tends to work well with data that includes numerous predictor variables, which is why we're seeing it perform especially well here with our model that includes transformations. 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.

Model Performance Table Results

A table summarizing key characteristics and performance metrics can be found below.

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 = 0	0.7	0.9964	0.9996	0.9975	0.0390	0.0025	0.9609

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.

The first conclusion of this report is pointing out one of its weaknesses. 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. 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 transformations. Although beyond the scope of this project, it may be worthwhile to explore methods like recursive feature elimination, principal component analysis, or tree-based methods.

The second conclusion is that 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."

As a third conclusion, 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. This would enable us to use spatial statistics 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.

Which brings us to this project's determination of the best model: of the models compared here, the best performing model for this scenario is the logistic regression of the stepwise model with transformations. This model has 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 is acceptable in comparison to the cost of many more false positives in models with similar or better FNRs, like QDA and KNN.

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.