Wyatt Scott — Disaster Relief Project: Part II

Executive summary:

Tragedy struck in January 2010 when a magnitude 7.0 earthquake struck the Republic of Haiti.[1] According to the Haitian government, the earthquake left over 316,000 dead or missing, 300,000 injured, and over 1.3 million homeless and displaced.[2] Haiti already suffered from poor water quality, among other stresses, before the earthquake that all but rendered obsolete the Haitian government; the earthquake destroyed the presidential palace and 14 of 16 government ministries and claimed the lives of numerous government officials. The earthquake decimated the island nation's infrastructure, and on the day of the disaster, the President of Haiti declared a national emergency and requested immediate assistance from the United States and the international community.[3]

In the aftermath, many stayed in makeshift shelters, fearing the aftershocks would further destroy what structures remained; building codes are weak, and construction standards on the island are low. Well over a million displaced persons created temporary shelters, many using blue tarps, which could serve as good indicators for locating the displaced persons to provide relief and assistance. Finding these blue tarps was challenging for a team of researchers from various institutes and organizations that collected aerial imagery over Haiti. The issue was that aid workers could not quickly search the thousands of images to locate the tarps and communicate those locations to rescue workers on the ground.

Researchers converted the aerial images into machine-readable datasets of Red, Green, and Blue (RGB) values. This project uses one such dataset to find an optimal model out of seven statistical learning methods to classify pixels accurately as to whether they belong to a blue tarp. The seven statistical learning methods include logistic regression, linear discriminant analysis, quadratic discriminant analysis, K-nearest neighbors, penalized logistic regression, random forest, and support vector machine.

---

(1) Data:

We train the models using a dataset of 63,241 observations and test them using a holdout set of 2,004,177 observations.

The training data contains 63,241 observations with four columns:

- **`Class`** : Categorical variable for pixel classification, including Blue Tarp, Rooftop, Soil, Various Non-Tarp, and Vegetation.
- **`Red`** : Numeric variable for the pixel's Red color quantity.
- **`Green`** : Numeric variable for the pixel's Green color quantity.
- **`Blue`** : Numeric variable for the pixel's Blue color quantity.

Several steps were necessary to wrangle the data for modeling. First, we created a factor type variable, **`Tarp`** , to differentiate observations identified as Blue Tarps (e.g., **`Tarp=Yes`**) from those identified as Rooftop, Soil, Various Non-Tarp, or Vegetation (e.g., **`Tarp=No`**).

The holdout set required more wrangling to get it into a format comparable to the training data. Those providing the holdout set shared several text files that we cleaned and converted to CSV files before combining the separate files into one.

The holdout set uses different column names than the training data, and it's unclear which columns represent `Red`, `Green`, and `Blue`. Columns `B1`, `B2`, and `B3` in the holdout set seem to represent the RGB values, but which color corresponds to which column is unclear. So, we will perform some additional analysis and rename the columns appropriately.

(1.1) Exploratory Data Analysis:

Next, we use density plots to explore the color distributions for the training and holdout sets based on `Tarp` status. This allows us to compare the mean values for the `B1`, `B2`, and `B3` columns in the holdout set to those of the `Red`, `Green`, and `Blue` columns of the training data.

Figure 1.1: Holdout vs. Train Data - Color Distributions by Tarp Status

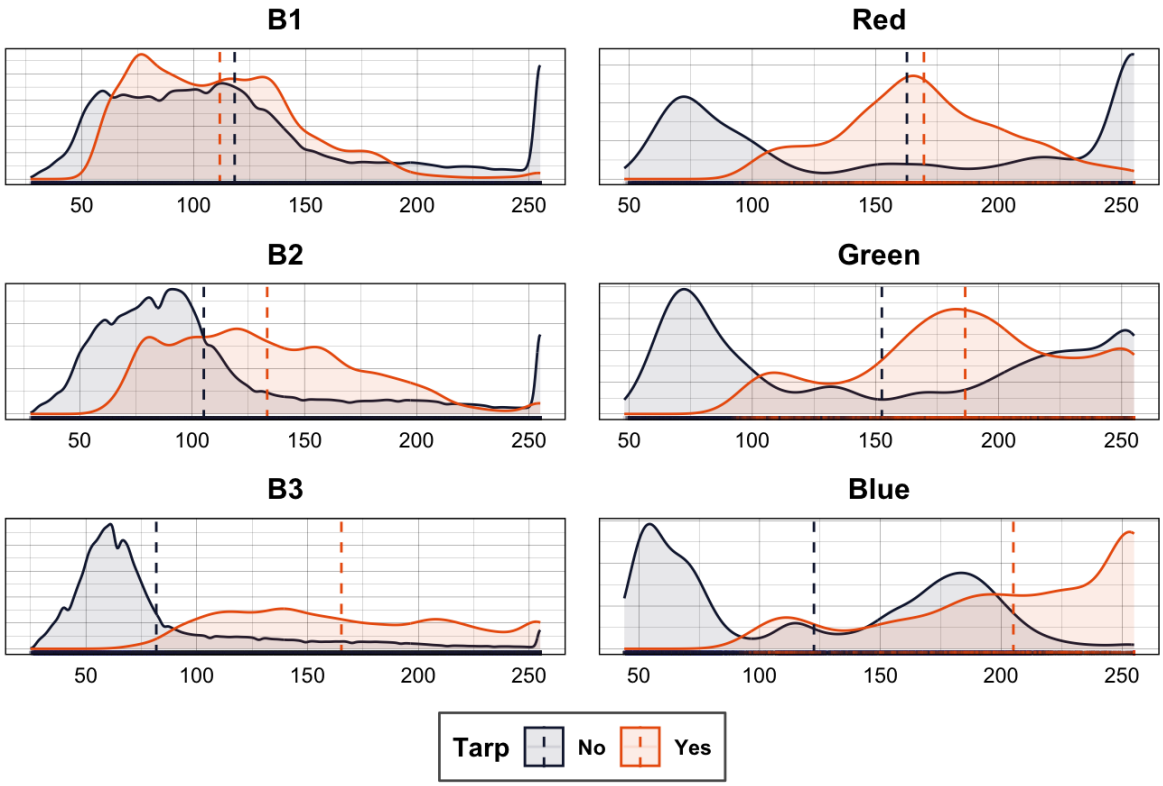


Figure 1.1 leads us to believe that `B1` represents `Red`, `B2` represents `Green`, and `B3` represents `Blue`. The vertical dashed lines represent the mean value for each color, for each class of `Tarp`. The spread is similar between the mean values for `B3` and `Blue` when `Tarp=No` versus when `Tarp=Yes`; the same goes for `B2` and `Green` and `B1` and `Red`. We rename those columns in the holdout data to appropriately label them for model testing.

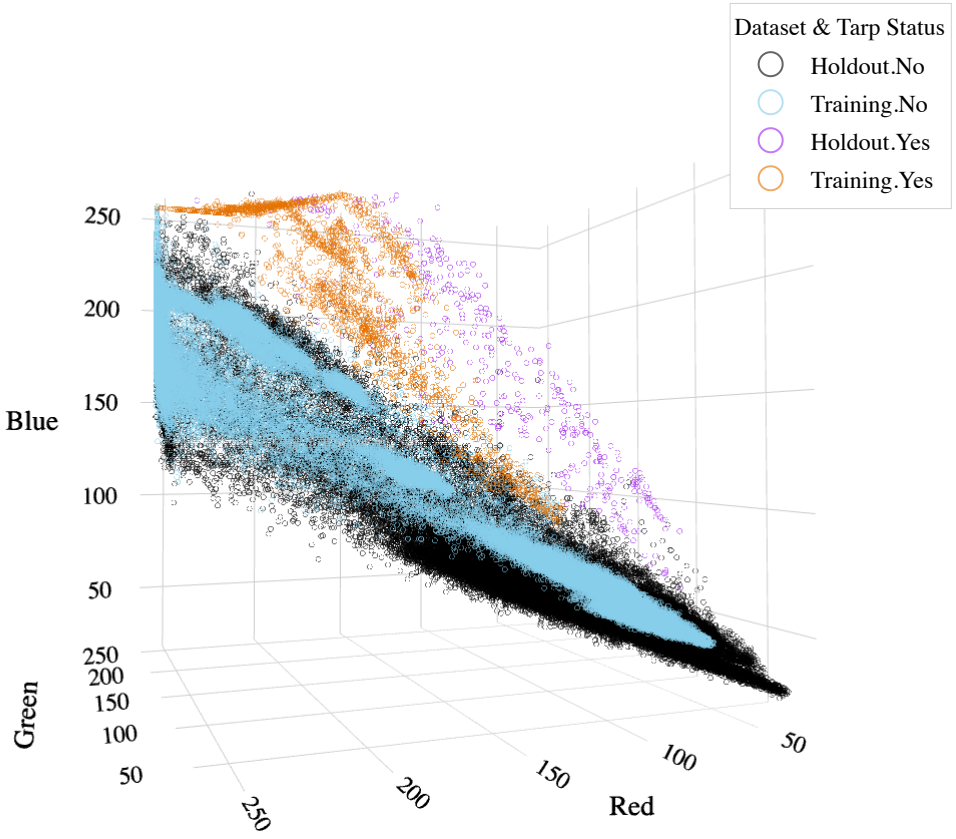
Table 1.1 below is a frequency table we use to check class imbalance in the training and holdout data. The frequency table shows significant class imbalance in both datasets.

Table 1.1: Class Imbalance

	Count		Percent	
Tarp Status	**Training**	**Holdout**	**Training**	**Holdout**
No	61,219	1,989,697	96.8	99.27
Yes	2,022	14,480	3.2	0.01

Next, we create a 3D scatterplot of all observations by `Tarp` status, using the RGB values for each of the three axes. We take a random subset of 60,000 observations from the training and holdout data and color them by `Tarp` status and dataset. Figure 1.2 below shows a clear class-separating hyperplane for both datasets, though the separation appears clearer for the holdout data; we should be able to accurately predict pixels' classes for the `Tarp` variable using RGB values as predictors.

Figure 1.2: RGB Values by Dataset and Tarp Status



(2) Methods:

(2.1) Parameter Optimization:

Using k-fold cross-validation with $k = 10$, we fit all models on the three-dimensional, two-class data. K-fold CV randomly splits the observations into k groups (or folds, 10, in this case) of approximately equal size. Several metrics are computed for each fold, including Accuracy, Error rate, etc., and the process is repeated 10 times with a different validation set each time. This process results in 10 estimates for each metric that are then averaged. Because the data in this project are simulated (i.e., we know the true classification of each pixel), we can compute the true test error and evaluate our CV results. Take Accuracy, for example, where $k = 10$ and accuracy_i represents the model's Accuracy on the ith fold:

$$\text{Accuracy} = \frac{1}{k}\sum_{i=1}^{k} \text{accuracy}_i \qquad \text{(Equation 2.1)}$$

10-fold CV allows for lower variability in the test error estimates, given how the formula averages across folds. For this project, however, we are interested in more than the test error estimates; we perform 10-fold CV on several statistical learning methods using different thresholds to identify the method that provides the best results. K-fold CV with $k = 10$ has "been shown empirically to yield test error rate estimates that suffer neither from excessively high bias nor from very high variance."[4]

While 10-fold CV provides less variability in the test error estimates, it leads to an intermediate level of bias as each training set only contains ~56,916.9 observations, derived from the following formula where $k = 10$ and $n = 63,241$:

$$(k-1)n/k \qquad \text{(Equation 2.2)}$$

(2.2) Train Control and Threshold Analysis:

We leverage the *trainControl* function to define the training control, specifying the method as "cv" and the number of folds as 10. For consistency, we define several objects and pass them to the *trainControl* function; for example, we pass "folds" to the index parameter to achieve identical stratified samples across models. The caret package, by default, selects the best model using Accuracy, but Accuracy can be misleading in the context of imbalanced data (see section 2.3), so we set the *summaryFunction* argument equal to caret's built-in *twoClassSummary* function, which forces caret to instead select the best model based on AUROC. We use the *train* function from the caret package to fit the models.

Determining the appropriate decision rule cutoff (i.e., threshold) without substantive domain expertise is difficult. We do not wield expertise on this particular topic, so we rely on statistical methods. To evaluate the models at various thresholds, we use the caret package's *thresholder* function, which "uses the resampling results from a train object to generate performance statistics over a set of probability thresholds for two-class problems."[5] The *thresholder* function takes several parameter inputs; one of the arguments we pass to the *thresholder* function is "stats," which we define as a list of model performance metrics we want it to return, including Balanced Accuracy, Kappa, Sensitivity, Specificity, Precision, F1, and False Positive Rate (FPR) calculated at each threshold between 0.1 and 0.9 in increments of 0.1. We define a function, *thresh*, within which we use the *thresholder* function to call the same function on multiple models and reduce code replication.

We believe it's important to focus on FPR and Balanced Accuracy instead of False Negative Rate (FNR) and Accuracy because the cost is high in terms of time and, potentially, life if the model incorrectly predicts a pixel to belong to a blue tarp; a team of aid providers using this information could arrive at locations where there are no displaced persons because the model incorrectly predicted blue tarps in those locations. Further, Accuracy is a misleading measure given class imbalance; we could have high Accuracy, yet the FPR or the FNR could be high. In this context, it seems more informative to look at the FPR and Balanced Accuracy.

(2.3) Evaluating Model Performance:

(2.3.1) Balanced Accuracy and Cohen's Kappa (Kappa):

The unbalanced nature of the classes leads us to consider Balanced Accuracy and Kappa over Accuracy.[6]

Balanced Accuracy is the average between sensitivity and specificity, measuring the average Accuracy obtained from the minority and majority classes. Balanced Accuracy is calculated using the formula below.

$$\frac{(\text{sensitivity} + \text{specificity})}{2} \qquad \text{(Equation 2.3)}$$

To calculate Kappa, we must first calculate *random accuracy* via the following formula:

$$\text{random accuracy} = \left(\frac{(TP + FN)(TP + FP) + (TN + FP)(TN + FN)}{(TP + TN + FP + FN)^2}\right) \quad \text{(Equation 2.4)}$$

From here, we calculate Kappa from the following formula:

$$\text{Kappa} = \left(\frac{\text{accuracy - random accuracy}}{1 - \text{random accuracy}}\right) \quad \text{(Equation 2.5)}$$

Kappa indicates how the model performs compared to random guessing, with values ranging from -1 (the model performs worse than random guessing) to +1 (a perfect model). A Kappa of 0 would mean the model performs as well as random guessing.[7]

We categorize Kappa values based on the work of Richard Landis and Gary Koch.[8]

(2.3.2) F1 Score:

We also evaluate F1 scores at each threshold to gauge performance. Rather than use precision or recall separately, the F1 statistic effectively combines their respective trade-offs into a single statistic reflecting "the 'goodness' of a classifier in the presence of rare classes."[9] Before describing the formula for F1, we must first consider the formulas for precision and recall:

Precision is calculated using the following formula:

$$\text{precision} = \frac{TP}{TP + FP} \quad \text{(Equation 2.6)}$$

Recall is calculated as follows:

$$\text{recall} = \frac{TP}{TP + FN} \quad \text{(Equation 2.7)}$$

Precision and recall measure a model's predictive performance but in different ways; precision measures how much the False Positives (FPs) influence error, whereas recall measures how much False Negatives (FNs) influence the error. We evaluate FPRs separately, as described above, but FNs are still important for this project.

The FNs are important in that if a model's FNR is high and disaster relief efforts use that model, then the relief groups on the ground could bypass certain locations that the model wrongly predicted not to have blue tarps, thereby overlooking displaced persons.

Combining precision and recall provides the formula for F1, where β denotes the relative importance of precision vs. recall (usually set to 1):[10]

$$F1 = \frac{(1 + \beta^2) * \text{recall} * \text{precision}}{\beta^2 * \text{recall} + \text{precision}} \quad \text{(Equation 2.8)}$$

The F1 score is the harmonic mean of precision and recall and is well-suited for this dataset; many studies note the usefulness of F1 scores for evaluating classifier model performance with imbalanced classes.[11] In this dataset, for example, a model that predicts every pixel (observation) not to belong to blue tarps (e.g., `Tarp=No`) would be 96.8% accurate on the training data and over 99% accurate on the holdout data.

(2.3.3) ROC Curve and AUC:

We use Receiver Operating Characteristic (ROC) curves and the Area Under the Curve (AUC) to evaluate and compare model performance. The ROC curve plots have two dimensions; a y-axis representing the Sensitivity (TPR) and an x-axis representing $1 - \text{Specificity}$ (FPR). The ROC curves show the TPR and FPR for every

possible threshold. A model that classifies at random, for example, will have a ROC curve lying along the diagonal, whereas a perfect model will have a sensitivity and specificity of 1 and would lie along the (0,1) position. The further the ROC curve is from the diagonal and closer to (0,1), the better the model is at correctly classifying observations.

We generate the ROC curves for each model using the out-of-fold (OOF) predicted probabilities returned from `model$pred` ; this approach generates a less biased performance metric using CV results instead of the data used to train the models. We also include the ROC curves for each fold.

For AUC, a value of 1.0 indicates a model that correctly classifies every observation. An AUC of 0.5 would indicate the model randomly guesses.

Note on ROC curves in this report: The dark-blue line represents the ROC curve using the OOF sample from CV; the rainbow-colored lines represent the ROC curve for each of the ten folds. The plots are zoomed in on both axes to provide a better view of the ROC curve differences. The ROC curves in section 10.2 are slightly different; these plots include a dark-blue line as before and one orange line representing the ROC curve for the model tested against the holdout set.

(2.3.4) Run Time:

We use the tictoc package and the *sys.time* function to time each model's train and test duration. We use parallel processing via the doParallel package to spread computations across multiple cores to increase computational efficiency. While the test and train time metrics are useful, they will likely vary across machines depending on specifications. Below are the machine specifications for the models trained in this report:

- OS: Mac - Ventura 13.5
- Processor: Apple M2 Pro, 10-Core CPU
- RAM: 16.0 GB

---

(3) Logistic Regression:

We fit an additive logistic regression model with all three predictors:

$$\text{Tarp} \sim \text{Red}(x_1) + \text{Blue}(x_2) + \text{Green}(x_2) \qquad \text{(Equation 3.1)}$$

(3.1) Training and Parameter/Hyperparameter Optimization:

Table 3.1 includes the results from the *train* function output for the logistic model.

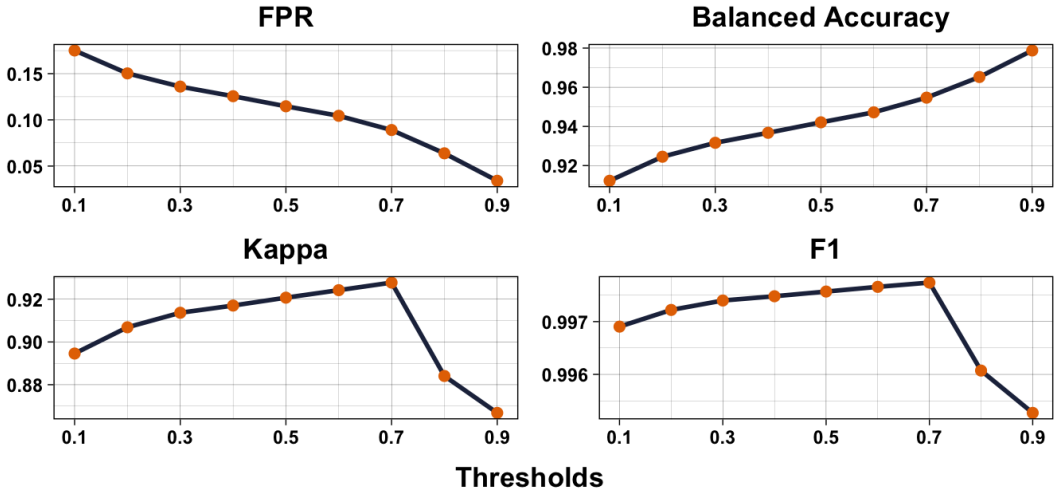
Table 3.1: Logistic Results

ROC	Sensitivity	Specificity
99.85	99.89	88.53

(3.2) Threshold Analysis and ROC Curve:

Figure 3.1 below shows the fluctuation in FPR, Balanced Accuracy, Kappa, and F1 at each threshold.

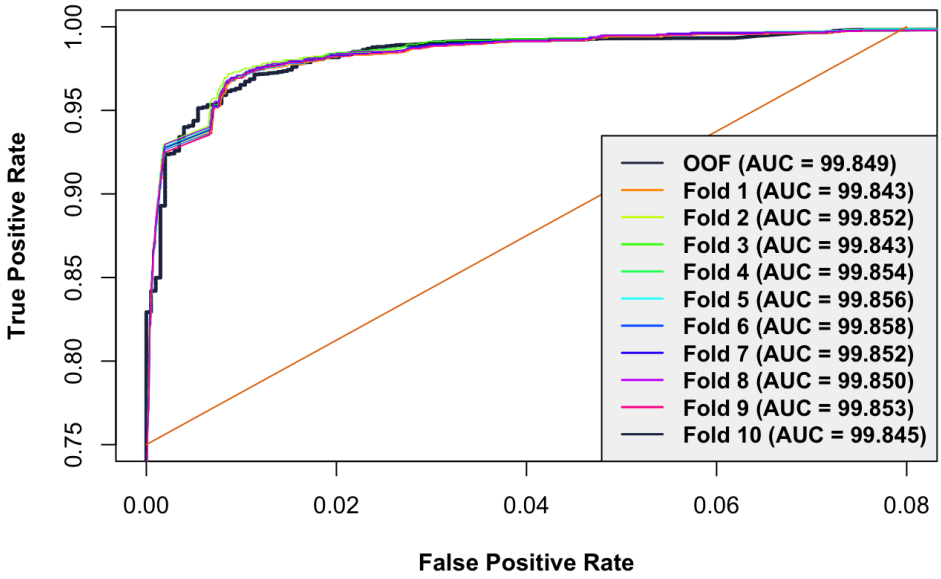
Figure 3.1: Performance Metrics by Threshold (Logistic)



Comparing the performance metrics for the logistic regression model at various thresholds, we find that the best overall performance occurs at a threshold of 0.9, where FPR is the lowest. The trade-off between a Kappa of 0.928 at a threshold of 0.7 and 0.867 at a threshold of 0.9 is insufficient to warrant going with the former, whereas the change in FPR from 0.0890 to 0.0341 between those same thresholds is significant. The Balanced Accuracy, Kappa, and F1 values barely fluctuate between a threshold of 0.7 and 0.9, so the decision to choose a threshold of 0.9 for this logistic regression model came down to the change in FPR.

Next, we use ROC curve plots to confirm the previous results and further evaluate the model's predictive performance.

Figure 3.2: ROC Curves (Logistic)



The ROC curves and AUC values for the logistic model show that it is accurate and far better than random guessing. The ROC curve for the OOF sample is similar to the fold curves, so the model seems not to overfit the training data.

(3.3) Final Model:

Table 3.2 below contains the `$finalModel` component of the logistic regression model:

Table 3.2: Logistic finalModel

Betas	Coefficients
β_0 **(Intercept)**	0.2098
β_1 **(Red)**	-0.2603
β_2 **(Blue)**	0.4724
β_3 **(Green)**	-0.2183

Degrees of Freedom: Total = 63,240 | Residual = 63,237.
Deviance: Null = 17,900 | Residual = 1,770.
AIC = 1,778.

The following equation represents the logistic regression model, where x_1, x_2, and x_3 denote **`Red`**, **`Blue`**, and **`Green`**, respectively:

$$\log\left(\frac{\hat{\pi}}{1-\hat{\pi}}\right) = 0.2098 - 0.2603(x_1) + 0.4724(x_2) - 0.2183(x_3) \qquad \text{(Equation 3.2)}$$

Below, we provide context behind the estimated regression coefficients.

- $\hat{\beta}_0 = 0.2098$: This indicates the estimated log odds of the probability that the pixel belongs to a blue tarp (e.g., **`Tarp=Yes`**) when $x_1 = x_2 = x_3 = 0$.
- $\hat{\beta}_1 = -0.2603$: This means that, for an additional one unit increase in **`Red`**, the estimated odds of the pixel belonging to a blue tarp gets multiplied by a factor of $\exp(-0.2603) = 0.7708126$. Put differently, for an additional one unit increase in **`Red`** (on average), the estimated log odds of a pixel belonging to a blue tarp decreases by 0.26031 while controlling for **`Blue`** and **`Green`**.
- $\hat{\beta}_2 = 0.4724$: This means that, for an additional one unit increase in **`Blue`**, the estimated odds of the pixel belonging to a blue tarp gets multiplied by a factor of $\exp(0.4724) = 1.603855$. Put differently, for an additional one unit increase in **`Blue`** (on average), the estimated log odds of the pixel belonging to a blue tarp decreases by 0.47241 while controlling for **`Red`** and **`Green`**.
- $\hat{\beta}_3 = -0.2183$: This means that, for an additional one unit increase in **`Green`**, the estimated odds of the pixel belonging to a blue tarp gets multiplied by a factor of $\exp(-0.2183) = 0.8038762$. Put differently, for an additional one unit increase in **`Green`** (on average), the estimated log odds of the pixel belonging to a blue tarp decreases by 0.21831 while controlling for **`Red`** and **`Blue`**.

(3.4) Assumptions and Limitations:

The additive logistic regression model in this section assumes that the relationship between the predictors and response is linear; it assumes that the effect of each color value on the log odds is constant and additive. It also assumes an absence of multicollinearity, which does not hold for this dataset. Table 3.3 below shows the correlation matrix for the training data and the Variance Inflation Factors (VIFs) for the logistic regression model. The VIFs are very high, suggesting the model assumption of the absence of multicollinearity does not hold.

Table 3.3: Correlation Matrix and VIFs

Predictor	VIFs
Red	224.006
Green	259.507
Blue	371.083

Predictor	Red	Green	Blue
Red	1.00	0.98	0.94
Green	0.98	1.00	0.97
Blue	0.94	0.97	1.00

(4) Linear Discriminant Analysis (LDA):

We now fit an LDA model using the *train* function and the same *trainControl* object for consistency.

We log-transform the predictors before training the LDA model to make the data more homoscedastic (i.e., to stabilize the variances, bringing them closer to equal) because LDA assumes that the class-conditional distributions have equal variances.

The LDA model equation is as follows:

$$\text{LD} = \text{Red}(\log(x_1)) + \text{Blue}(\log(x_2)) + \text{Green}(\log(x_3)) \qquad \text{(Equation 4.1)}$$

(4.1) Training and Parameter/Hyperparameter Optimization:

Table 4.1 includes the results from the *train* function output for the LDA model.

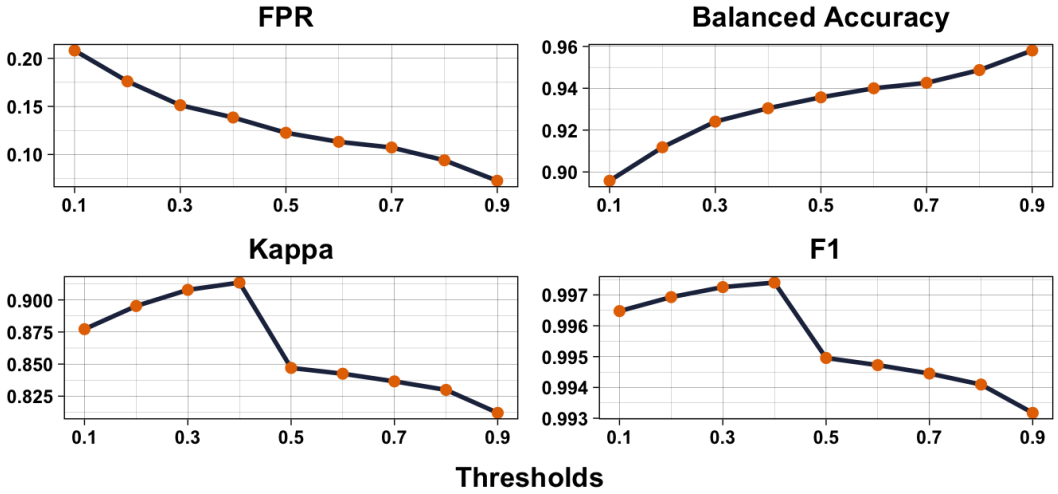
Table 4.1: LDA Results

ROC	Sensitivity	Specificity
99.76	99.4	87.74

(4.2) Threshold Analysis and ROC Curve:

Figure 4.1 below shows the fluctuation in FPR, Balanced Accuracy, Kappa, and F1 at each threshold.

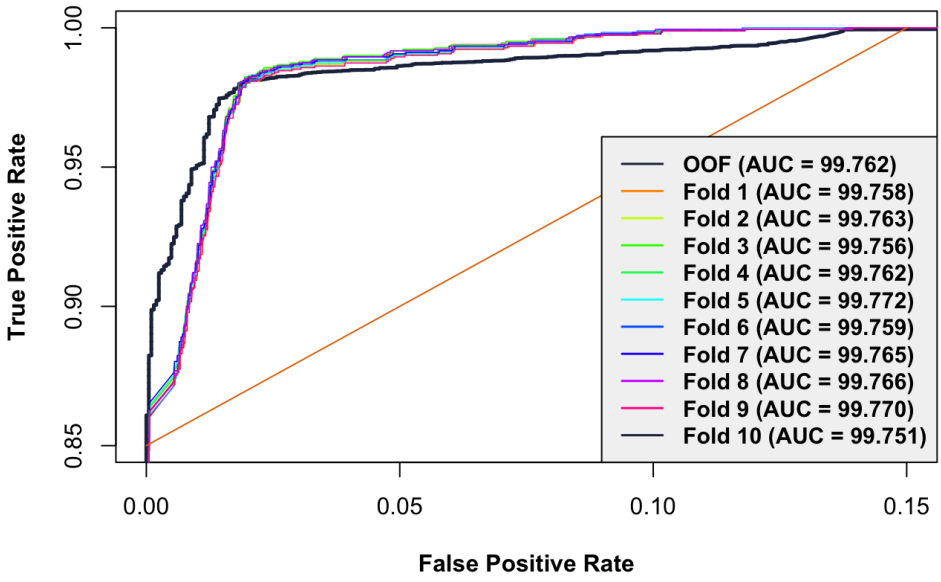
Figure 4.1: Performance Metrics by Threshold (LDA)



Comparing the performance metrics for the LDA model at various thresholds, the best overall performance appears to occur at a threshold of 0.9. The FPR is lowest and Balanced Accuracy is highest at this threshold. Kappa and the F1 score do not vary much.

Next, we use ROC curve plots to confirm the previous results and further evaluate the model's predictive performance.

Figure 4.2: ROC Curves (LDA)



The ROC curves and AUC values for the LDA model show that it is highly accurate and far better than random guessing. The ROC curve for the OOF sample is only slightly different than the fold curves, so the model seems not to overfit the training data.

(4.3) Final Model:

Table 4.2 below contains the `$finalModel` component of the LDA model:

Table 4.2: LDA finalModel

Predictor	LD1
log(Red)	-4.527
log(Blue)	5.566
log(Green)	-0.777

		Group Means		
Tarp Status	**Prior Probs**	**log(Red)**	**log(Blue)**	**log(Green)**
No	96.80	-0.009	-0.0363	-0.018
Yes	3.19	0.288	1.0990	0.556

The equation for the final LDA model is included below, where LD represents the linear discriminant and the coefficients represent the weights assigned to each predictor. x_1, x_2, and x_3 denote **`Red`**, **`Blue`**, and **`Green`**, respectively:

$$\mathrm{LD1} = -4.5271(x_1) + 5.5658(x_2) - 0.7768(x_3) \qquad \text{(Equation 4.2)}$$

In the context of this analysis, the first linear discriminant function $(\mathrm{LD1})$ is a linear combination of the estimated coefficients of the linear discriminants for each log-transformed predictor.

(4.4) Assumptions and Limitations:

The LDA model assumes that each class has a Gaussian distribution with the same covariance.

---

(5) Quadratic Discriminant Analysis (QDA):

We now fit a QDA model using the *train* function and the same *trainControl* object for consistency.

(5.1) Training and Parameter/Hyperparameter Optimization:

Table 5.1 includes the results from the *train* function output for the QDA model.

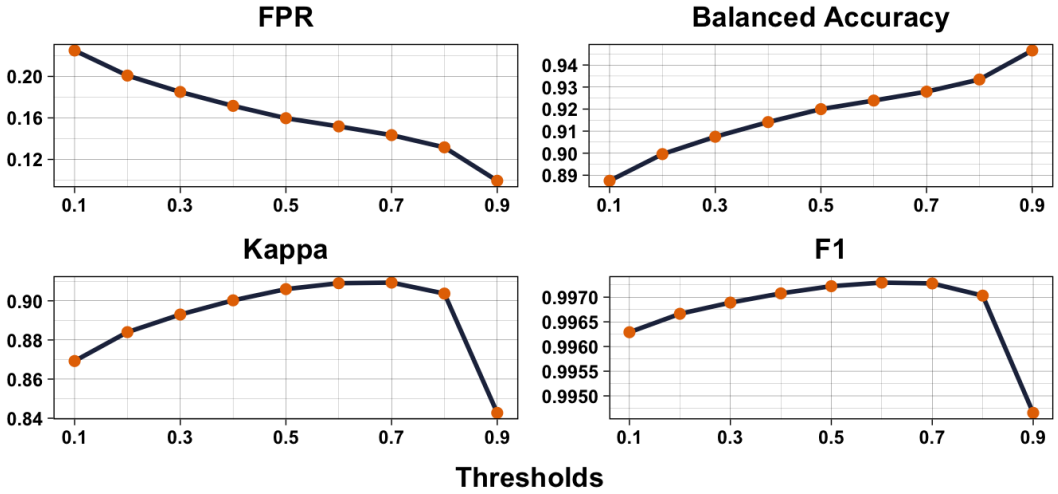
Table 5.1: QDA Results

ROC	Sensitivity	Specificity
99.82	99.97	84.03

(5.2) Threshold Analysis and ROC Curve:

Figure 5.1 below visualizes the fluctuation in FPR, Balanced Accuracy, Kappa, and F1 at each threshold.

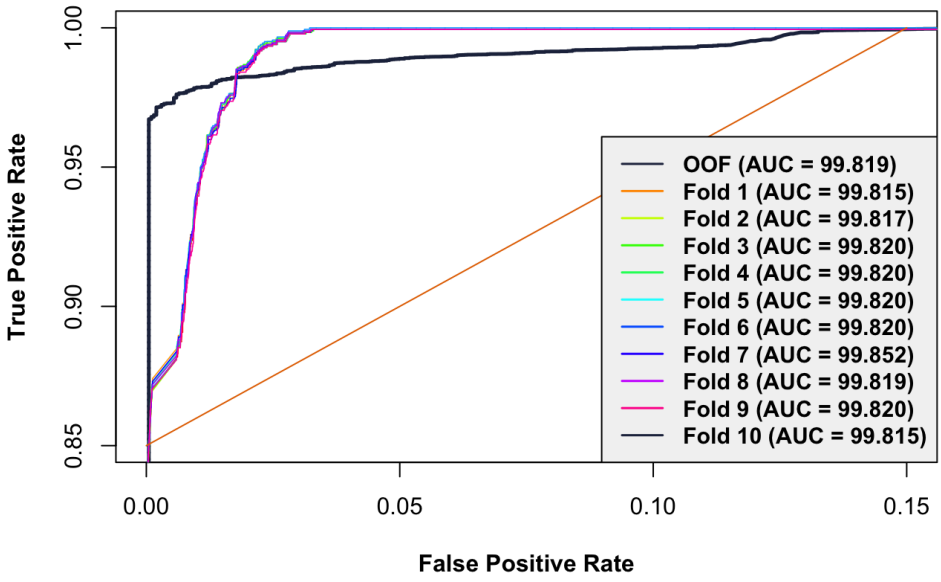
Figure 5.1: Performance Metrics by Threshold (QDA)



Comparing the performance metrics for the QDA model at various thresholds, the best overall performance appears to occur at a threshold of 0.8; the FPR is second-lowest, and Balanced Accuracy is second-highest at this threshold, while Kappa and F1 remain high. The F1 score is also high at each.

Next, we use ROC curve plots to confirm the previous results and further evaluate the model's predictive performance.

Figure 5.2: ROC Curves (QDA)



The ROC curves and AUC values for the QDA model show that it is highly accurate and far better than random guessing. The ROC curve for the OOF sample is quite different than the fold curves, so the model may overfit the training data, which could indicate poor performance on the holdout set.

(5.3) Final Model:

Table 5.2 below contains the `$finalModel` component of the QDA model:

Table 5.2: QDA finalModel

		Group Means		
Tarp Status	**Prior Probs**	**Red**	**Blue**	**Green**
No	96.80	162.76	122.49	152.58
Yes	3.19	169.66	205.04	186.42

Table 5.2 above shows us the prior probability of the groups, which isn't incredibly informative given we already know this from the exploratory data analysis. What is informative, however, are the group means:

For observations that the model classifies as belonging to a blue tarp, the mean **`Blue`** value is much higher than the observations the model classified as not belonging to a blue tarp. On the other hand, the mean value for **`Red`** barely changes from class to class, and the mean for **`Green`** changes only slightly.

(5.4) Assumptions and Limitations:

The QDA model assumes that each class has a Gaussian distribution but potentially different covariances.

---

(6) K-Nearest Neighbor (KNN):

We now fit a KNN model using the *train* function and the same *trainControl* object for consistency. We customize the tuning process by adding the *tuneLength* parameter to the *train* function and passing an integer (10) to it, which tells the *train* function to try 10 different values for k.

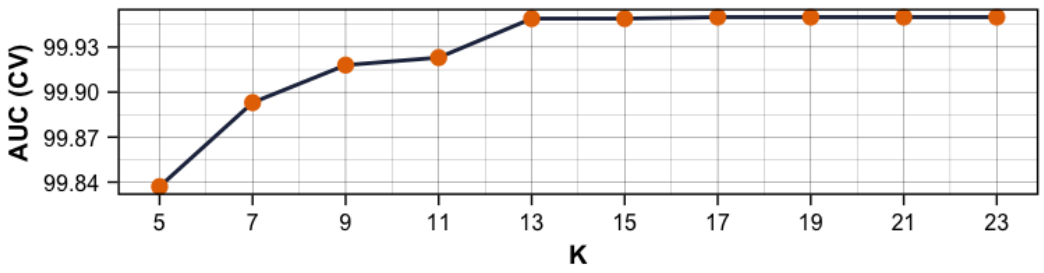
We train the KNN model using the following formula, where x_0 represents the test observation to be classified, and $\mathcal{N}_0$ represents the K points in the training data that are closest to x_0. The KNN model estimates the conditional probability for class j as the fraction of points in $\mathcal{N}_0$ whose response values equal j and then classifies x_0 to the class with the largest probability from:[12]

$$\Pr(\mathrm{Y} = \mathrm{j} | \mathrm{X} = \mathrm{x}_0) = \frac{1}{\mathrm{K}} \sum_{\mathrm{i} \in \mathcal{N}_0} \mathrm{I}(\mathrm{y}_\mathrm{i} = \mathrm{j}) \qquad \text{(Equation 5.1)}$$

(6.1) Training and Parameter/Hyperparameter Optimization:

Figure 6.1 shows the AUC results from the *train* function output for the KNN model.

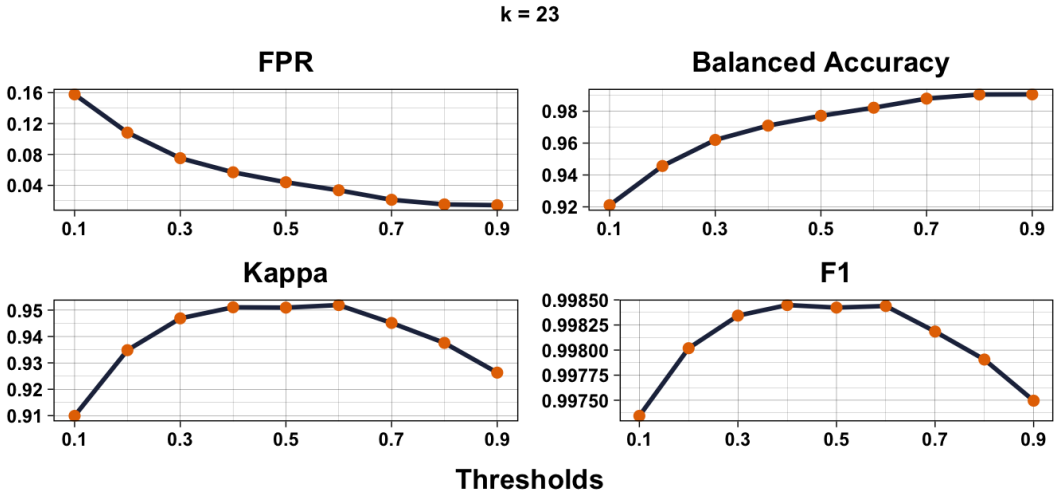
Figure 6.1: AUC by K Values



(6.2) Threshold Analysis and ROC Curve:

Figure 6.2 below shows the change in FPR, Balanced Accuracy, Kappa, and F1 at each threshold.

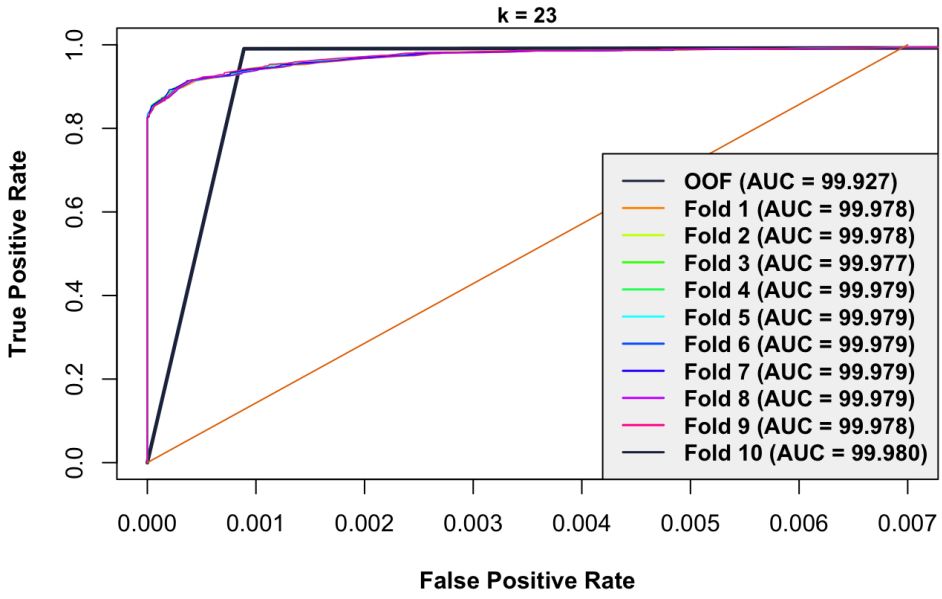
Figure 6.2: Performance Metrics by Threshold (KNN)



Comparing the performance metrics for the KNN model where $k = 23$ at various thresholds, the best overall performance appears to occur at a threshold of 0.6. This threshold strikes a tough balance between FPR and the other performance metrics. The FPR is still quite low at a threshold of 0.6 (compared to other models), and this is the threshold at which Kappa and F1 are the highest.

Next, we use ROC curve plots to confirm the previous results and further evaluate the model's predictive performance.

Figure 6.3: ROC Curves (KNN)



The ROC curves and AUC values for the KNN model show that it is highly accurate and far better than random guessing. The ROC curve for the OOF sample is quite different than the fold curves, so the model may overfit the training data, which could indicate poor performance on the holdout set.

(6.3) Final Model:

Table 6.2 below includes the training outcome distribution from the `$finalModel` component of the KNN model:

Table 6.2: KNN finalModel

Tarp Status	Outcome
No	61,219
Yes	2,022

k = 23.

(6.4) Assumptions and Limitations:

The KNN model is a non-parametric statistical learning approach with few underlying assumptions and flexible decision boundaries. The KNN model has some limitations; for example, it treats all instances equally regardless of class, and the dataset for this project is highly imbalanced, so the majority class may dominate the classification and create misleading and/or biased results. Another limitation, or at least something to be aware of with this model, is that it is sensitive to the scale of the parameters; KNN calculates the distance between instances by parameter values, and the predictors are within the same range (0 to 255).

---

(7) Penalized Logistic Regression (ElasticNet Penalty):

We now fit a Penalized Logistic Regression (ElasticNet penalty) model using the *train* function and the same *trainControl* object for consistency. First, we alter a few *trainControl* parameters.

The ElasticNet penalty combines the ridge regression penalty and the lasso regression penalty; it combines L1 and L2 regularization to achieve feature selection and regularization.

We must define two hyperparameters (α and λ) given that we defined the method parameter for the *train* function as "glmnet." α can be a value between 0 and 1; when $\alpha = 0$, the lasso penalty is 0, and only the ridge regression penalty remains. Below, we describe the method used to build the PLR model with ElasticNet penalty.

Rather than pass sequences of λ and α values to the *expand.grid* function from the caret package, we set *tuneLength* to 20 and let the caret package choose the optimal hyperparameters.

(7.1) Training and Parameter/Hyperparameter Optimization:

Table 7.1 below contains the *bestTune* object of the PLR model instead of the output for the *train* function, as the latter would be long given that we set *tuneLength* to 20.

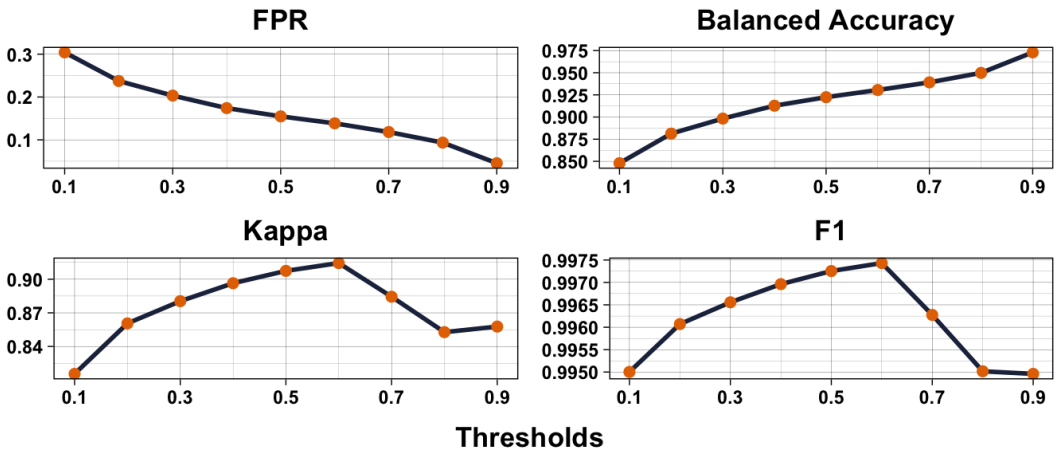
Table 7.1: PLR bestTune

Hyperparameter	Values
α	0.1474
λ	2e−05

(7.2) Threshold Analysis and ROC Curve:

Figure 7.1 below visualizes the fluctuation in FPR, Balanced Accuracy, Kappa, and F1 at each threshold.

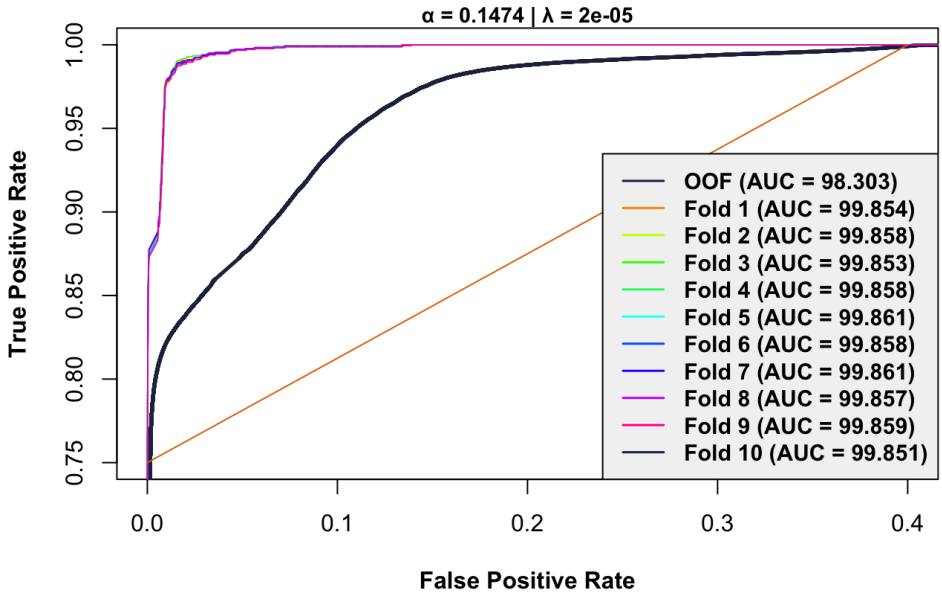
Figure 7.1: Performance Metrics by Threshold (PLR)



Comparing the performance metrics for the PLR model (where $\lambda = 2e-05$ and $\alpha = 0.1474$) at various thresholds, the best overall performance appears to occur at 0.6. Balanced Accuracy is relatively high and the FPR is relatively low at a threshold of 0.6. The drop in Kappa and F1 from 0.6 to 0.9 warrants going with the lower threshold.

Next, we use ROC curve plots to confirm the previous results and further evaluate the model's predictive performance.

Figure 7.2: ROC Curves (PLR)



The ROC curves and AUC values for the PLR model show that it is highly accurate and far better than random guessing. The ROC curve for the OOF sample is quite different than the fold curves, so the model may overfit the training data, which could indicate poor performance on the holdout set.

(7.3) Final Model:

Table 7.2 below shows the PLR model's performance metrics at a threshold of 0.6:

Table 7.2: PLR Performance Metrics

α	λ	Threshold	Balanced Accuracy	Kappa	Sensitivity	Specificity	Precision	F1	FPR	AUROC
0.1474	2e-05	0.6	93.05	91.45	99.94	86.15	99.54	99.74	13.85	98.303

(7.4) Assumptions and Limitations:

The ElasticNet penalty for logistic regression assumes a linear relationship between the predictor variables and the log odds of the response variable.

---

(8) Random Forest (RF):

A weak learner classification decision tree predicts that each observation belongs to the most commonly occurring class of training observations in the region to which it belongs. These weak learners, however, suffer from high variance as they're prone to produce very different results if trained on different subsets of the training data. Since we use the bagging method, B trees are built using random subsets **with replacement** from the training data. We leverage an ensemble learning algorithm (the RF model) to overcome the high variance that the bagging process can produce. RF uses bootstrapping to take repeated samples from the training data, train a model on the bth bootstrapped training set to get $\hat{f}^{*}b(x)$, and then average all the predictions to find the ensemble predictor:[13]

$$\hat{f}_{bag}(x) = \frac{1}{B}\sum_{b=1}^{B} \hat{f}^{\cdot}b(x) \qquad \text{(Equation 8.1)}$$

As we're running a binary classification, the model we use takes a "majority vote," that is, if most of the trees classified a given pixel as belonging to a blue tarp, then the ensemble predictor classifies it as such.

RF decorrelates the trees by using a random subset of m predictors as candidates for each node (or split), taking a different subset of m predictors (although it could be the same subset) for each subsequent node, and then repeating this process for each node.

We now fit the RF model using the *train* function and the same *trainControl* object for consistency. First, we alter a few *trainControl* parameters. The "rf" method in the caret package *train* function only takes one parameter, mtry. We define a list of mtry values as either 1 or 2, given that we only have 3 predictors and using $m = p$ amounts to bagging.

(8.1) Training and Parameter/Hyperparameter Optimization:

Table 8.1 includes the results from the *train* function output for the RF model.

Table 8.1: RF Results

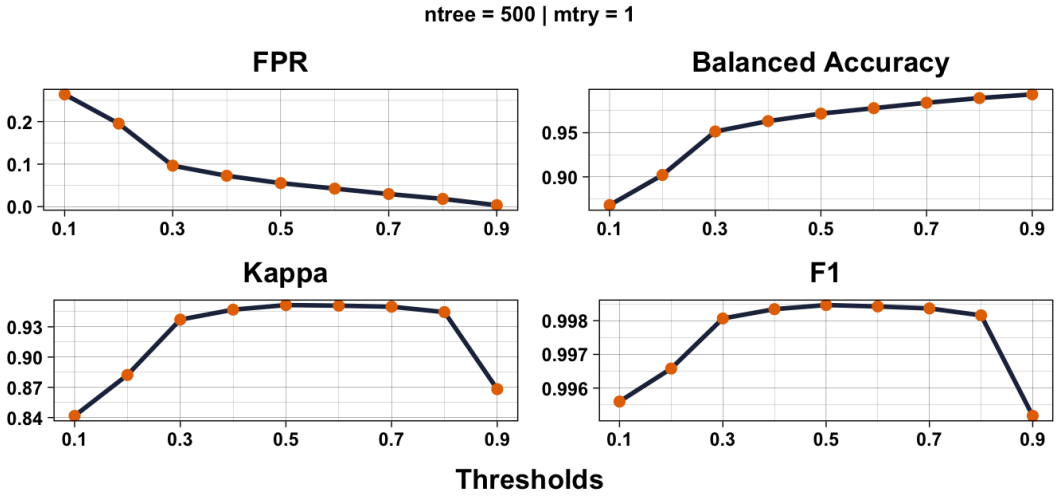
mtry	ROC	Sensitivity	Specificity
★1	99.97	99.88	94.41
2	99.43	99.87	94.51

ROC was used to select optimal model.
Optimal hyperparameter value is mtry = 1.

(8.2) Threshold Analysis and ROC Curve:

Figure 8.1 below visualizes the fluctuation in FPR, Balanced Accuracy, Kappa, and F1 at each threshold.

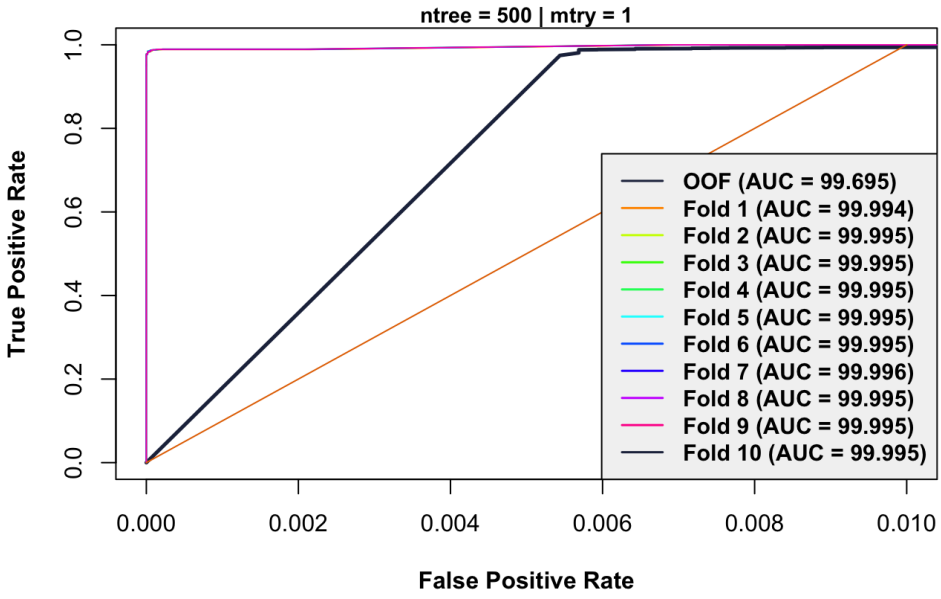
Figure 8.1: Performance Metrics by Threshold (RF)



Comparing the performance metrics for the RF model (where $\text{mtry} = 1$ and $\text{ntree} = 500$) at various thresholds, we choose a threshold of 0.5. The FPR is very low at a threshold of 0.5 and only decreases slightly thereafter, whereas Kappa and F1 scores are highest at a threshold of 0.5, and the Balanced Accuracy remains very high.

Next, we use ROC curve plots to confirm the previous results and further evaluate the model's predictive performance.

Figure 8.2: ROC Curves (RF)



The ROC curves and AUC values for the RF model show that it is highly accurate and far better than random guessing. The ROC curve for the OOF sample is quite different than the fold curves, so the model may overfit the training data, which could indicate poor performance on the holdout set.

(8.3) Final Model:

Table 8.2 below contains the `$finalModel` component of the RF model:

Table 8.2: RF finalModel

	Reference		
Predicted	**No**	**Yes**	**Class Error**
No	61,149	70	0.001143
Yes	118	1,904	0.058358

ntree = 500 | mtry = 1.

OOB estimate of error rate: 0.3%.

The `$finalModel` component of the RF model shows us that only one variable is considered at each split during tree construction (e.g., $\text{mtry} = 1$). The output also tells us that the OOB estimate of the error rate is very low, only 0.3% and that the class error is higher for the "Yes" class.

(8.4) Assumptions and Limitations:

Because RF involves bootstrapping and the training data is highly imbalanced, there's a decent risk that a bootstrapped sample contains few (if any) observations of the minority class. This can result in a tree that performs poorly on the holdout data.

---

(9) Support Vector Machine (SVM):

SVMs are an extension of support vector classifiers, which are an extension of maximal margin classifiers. Maximal margin classifiers look for a separating hyperplane (subspace) that separates the classes to maximize the distance (e.g., *margin*) between the separation surface and the nearest data points.

The linear support vector classifier can be represented as

$$f(x) = \beta_0 + \sum_{i=1}^{n} \alpha_i \langle x, x_i \rangle \qquad \text{(Equation 9.1)}$$

where:

- $f(x)$ is the decision function, taking an input vector x and outputting a scalar value that is used to predict the class label for input data point x.
- β_0 is the intercept; a constant representing the offset of the decision boundary from the origin along the y-axis.
- $\sum_{i=1}^{n}$ denotes the summation over all observations in the training data.
- α_i denotes each data point x_i s respective coefficient; observations with non-zero α_i s are support vectors that lie on or within the margin.
- n is the number of training observations.
- $\langle x, x_i \rangle$ represents the inner product (i.e., dot product) between the input vector x and the observation x_i; this represents the similarity between the x and x_i inside the feature space.

In summing over all n (from $i = 1$ to n), the equation considers all training data points in defining the separation hyperplane. This could be a massive undertaking for larger datasets; however, since $\alpha_i \neq 0$ only for support vectors, we can rewrite the previous equation as

$$f(x) = \beta_0 + \sum_{i \in S} \alpha_i \langle x, x_i \rangle \qquad \text{(Equation 9.2)}$$

where S is the collection of support vector indices. This equation is more computationally efficient as we only consider the support vectors rather than all training observations.

The kernel functions used by SVMs replace representations of the inner product with a generalization

$$K(x_i, x_{i'}) \qquad \text{(Equation 9.3)}$$

where K is a kernel function that quantifies the similarity of two observations.

The EDA in section 1.3.1 showed us that the class-separating plane is close to, but not exactly, linear. This leads us to consider two SVMs; one with a linear kernel and one with a radial kernel (i.e., Radial Basis Function (RBF) kernel).

The equation for the linear kernel is

$$K(x_i, x_{i'}) = \sum_{j=1}^{p} x_{ij} x_{i'j} \qquad \text{(Equation 9.4)}$$

where p denotes the number of features (or dimensions). The linear kernel is denoted as $K(x_i, x_{i'})$, which takes two feature vectors, x_i and $x_{i'}$ each containing p features. The kernel value is calculated using the dot product of the feature vectors in the p-dimensional feature space. The dot product between the two feature vectors is the sum of the element-wise product of their corresponding feature values, x_{ij} and x'_{ij} for $j = 1$ to p.

The radial kernel is

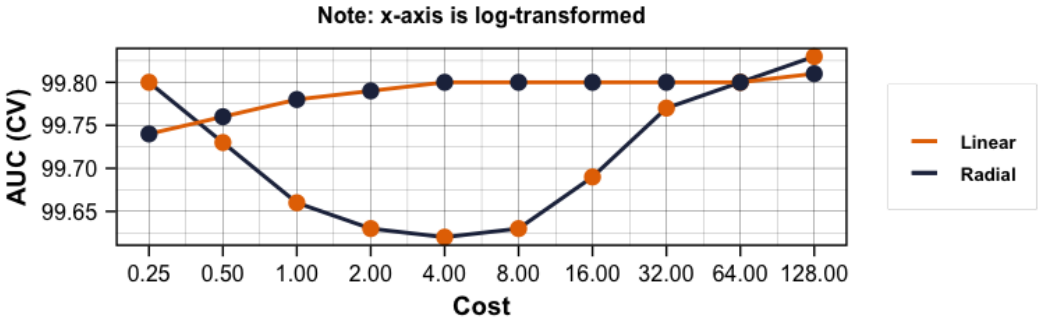
$$K(x_i, x_{i'}) = \exp(-\gamma \sum_{j=1}^{p} (x_i - x_{i'j})^2) \qquad \text{(Equation 9.5)}$$

where γ is a positive constant. The RBF finds support vector classifiers in infinite dimensions.

We determine whether to use a linear or radial kernel by training two separate models using the *train* function and the same *trainControl* object for consistency; however, we set *tuneLength* to 10 for the SVM with a radial kernel and pass the same series of cost values to the *tuneGrid* for SVM with a linear kernel. The cost argument specifies the cost of violating the margin; the margin is wider for lower cost values and narrows as cost increases. For now, we let caret choose γ for the SVM with the radial kernel; γ acts as a regularization hyperparameter, defining the influence of training observations on determining the decision boundary.

Figure 9.1 below displays the AUROC values as cost increases for both kernels.

Figure 9.1: AUC — Linear vs. Radial Kernel



Caret identifies $\text{cost} = 128$ as the optimal cost value for both kernels. A cost value of 128 seems high and may result in overfitting the training data and poor performance on the holdout set. Further, as Figure 9.1 shows, increasing cost from 0.25 to 128 only slightly increases AUC. Such a high cost value could produce too many

support vectors and too narrow a margin, and, thus, poor performance on the holdout set.

For the SVM with the radial kernel, caret identifies $\gamma = 8.2977$ as the optimal γ value, which also seems high and could result in an irregular decision boundary overfit to the training data.

Figure 9.2 below shows the training data colored by `Tarp` status, with separate colors for the support vectors for each kernel where $\text{cost} = 128$ for both kernels and $\gamma = 8.2977$ for the radial kernel.

Figure 9.2: Linear and Radial Support Vectors

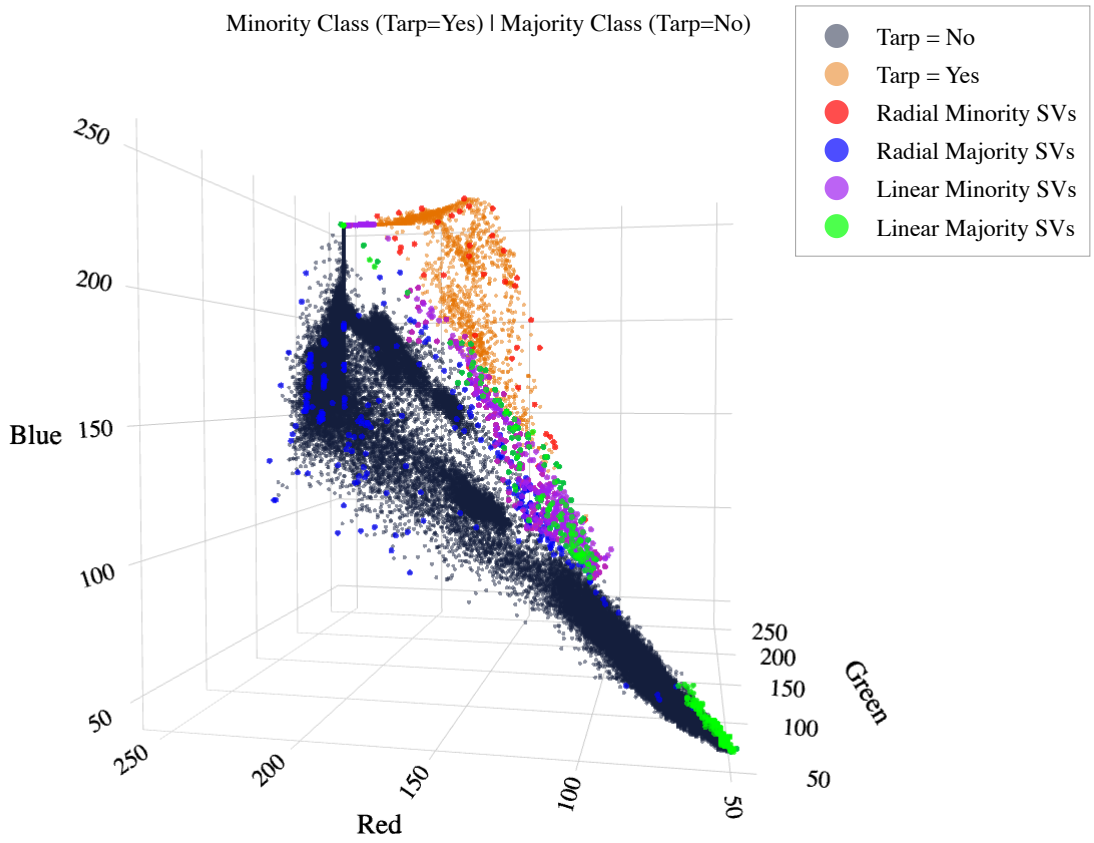


Figure 9.2 shows that the support vectors for the SVM with the linear kernel are tightly grouped for both classes. We also see that the support vectors for the SVM with the radial kernel are much more dispersed.

Since we saw that the model performance did not change much as cost increased, we will train another set of SVMs, this time defining cost as a constant (0.25) and γ as a sequence from 0.25 to 2.00 in increments of 0.25, which we pass to the *tuneGrid* parameter.

Table 9.1 below shows the performance of the SVM with the radial kernel where $\text{cost} = 0.25$ and γ changes from 0.25 to 2.00 in increments of 0.25.

Table 9.1: SVM Radial Results

γ	1.00	★1.25	1.50	1.75	2.00	2.25	2.50	2.75	3.00
ROC	99.960	99.962	99.961	99.960	99.958	99.954	99.949	99.944	99.940
Sensitivity	99.819	99.824	99.838	99.846	99.851	99.856	99.864	99.860	99.863

cost held constant at 0.25.

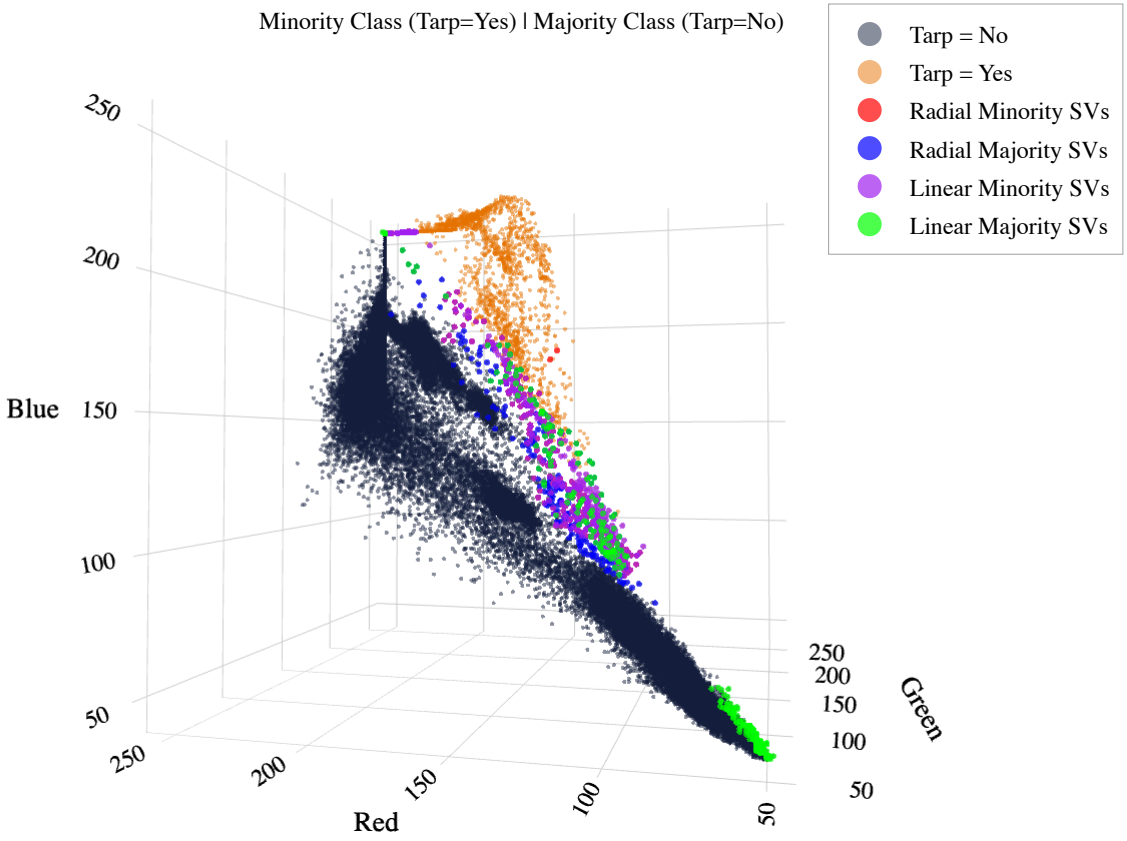
ROC was used to select optimal model.

γ	1.00	★1.25	1.50	1.75	2.00	2.25	2.50	2.75	3.00
Specificty	92.637	92.582	92.333	92.432	92.383	92.482	92.797	92.630	92.580

cost held constant at 0.25.
ROC was used to select optimal model.

We see that the optimal tuning, according to caret, occurs when $\gamma = 1.25$ while holding cost constant at 0.25. Now we examine changes in the support vectors for the different kernels with the new hyperparameter values.

Figure 9.3: Linear and Radial Support Vectors



The radial support vectors are not as widely dispersed and the linear support vectors appear almost unchanged.

Table 9.2 below compares the SVM performance for each kernel.

Table 9.2: SVM Results, Radial vs. Linear Kernel

Statistics	Radial	Linear
ROC	99.962	99.743
Sensitivity	99.824	99.897
Specificity	92.582	88.477

cost = 0.25.
γ = 1.25.

Because the class separation is not quite linear and the radial kernel performs better, we will move forward using the SVM with the radial kernel.

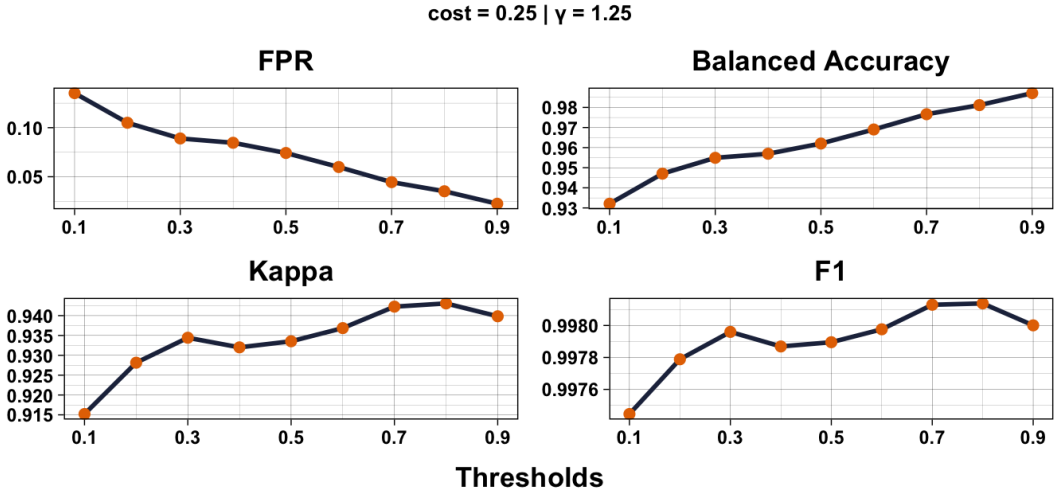
(9.1) Training and Parameter/Hyperparameter Optimization:

Table 9.2 above contains the results from the *train* function output for the SVM model with the radial kernel and $\text{cost} = 0.25$ and $\gamma = 1.25$.

(9.2) Threshold Analysis and ROC Curve:

Figure 9.4 below visualizes the fluctuation in FPR, Balanced Accuracy, Kappa, and F1 at each threshold.

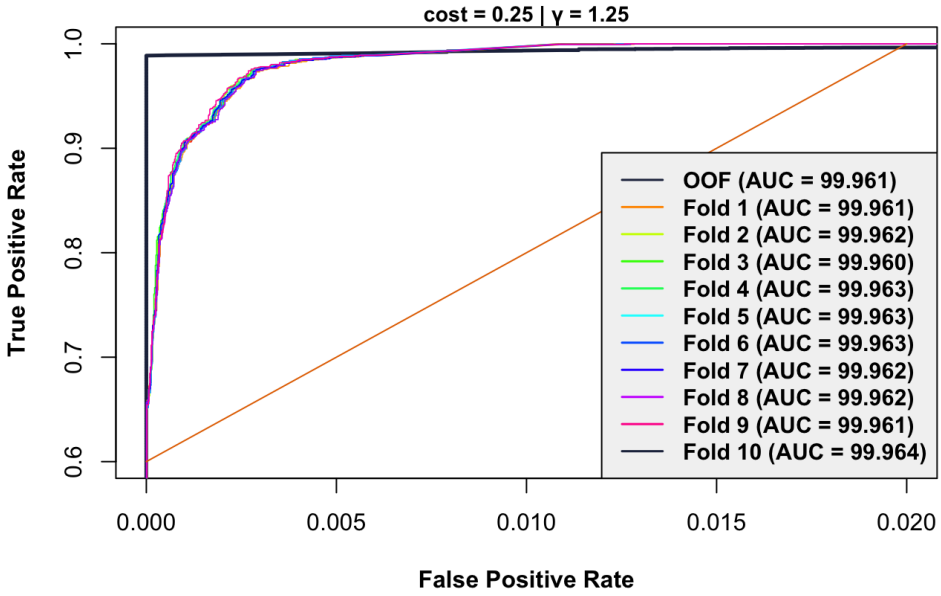
Figure 9.4: Performance Metrics by Threshold (SVM)



Comparing the performance metrics for the SVM with the radial kernel, we believe the optimal threshold is 0.9, as this is where the FPR is lowest and the Balanced Accuracy is highest; the Kappa value and F1 score are second-highest at 0.9 and only slightly less than when the threshold is 0.8.

Next, we use ROC curve plots to confirm the previous results and further evaluate the model's predictive performance.

Figure 9.5: ROC Curves (SVM, Radial)



The ROC curves and AUC values for the SVM show that it is highly accurate and far better than random guessing. The ROC curve for the OOF sample is quite different (albeit better) than the fold curves, so the model may overfit the training data, which could indicate poor performance on the holdout set.

(9.3) Final Model:

Table 9.3 below contains the SVM model's performance metrics at a threshold of 0.9:

Table 9.3: SVM Model Statistics

γ	cost	Threshold	Balanced Accuracy	Kappa	Sensitivity	Specificity	Precision	F1	FPR	AUROC
1.25	0.25	0.9	98.71	93.98	99.68	97.75	99.93	99.8	2.25	99.961

(9.4) Assumptions and Limitations:

SVMs tend to be one of the most accurate classifiers and work well with classes with nonlinear separation, and the classes in the training data for this project don't appear to be separated by a perfect linear plane; however, SVM can take a long time to train. SVMs can also be prone to overfitting training data; for example, the caret package first recommended $\text{cost} = 128$, but that seemed too high, so we adjusted accordingly.

(10) Model Performance Metrics:

Tables 10.1 and 10.2 below contain the CV and holdout test performance metrics for each model. The final row in the tables shows the range of values for each performance metric.

Table 10.1: Performance Metrics (Cross-Validation)

Model	Threshold	AUROC	Balanced Accuracy	Kappa	F1	FPR	Train Time (secs)
Logistic	0.9	99.848	97.88	86.68	99.53	3.41	10.67
LDA	0.9	99.761	95.81	81.2	99.32	7.27	0.77
QDA	0.8	99.82	93.34	90.37	99.7	13.16	0.68
KNN*	0.6	99.95	98.22	95.19	99.84	3.36	26.66
PLR†	0.6	98.303	93.05	91.45	99.74	13.85	46.43
RF‡	0.5	99.968	97.17	95.16	99.85	5.54	50.46
SVM§	0.9	99.961	98.71	93.98	99.8	2.25	59.64
Range	NA	1.67	5.63	13.96	0.53	11.53	55.62

Table 10.2: Performance Metrics (Holdout Data)

Model	Threshold	AUROC	Balanced Accuracy	Kappa	F1	FPR	Test Time (secs)

* k = 23.
† α = 0.1474 | λ = 2e-05.
‡ mtry = 1 | ntree = 500.
§ cost = 0.25 | γ = 1.25.

Model	Threshold	AUROC	Balanced Accuracy	Kappa	F1	FPR	Test Time (secs)
Logistic	0.9	99.94	98.59	83.55	99.86	2.57	2.72
LDA	0.9	99.96	98.17	85.71	99.88	3.45	3.82
QDA	0.8	99.15	81.78	70.88	99.81	36.33	3.83
KNN*	0.6	95.48	91.9	63.72	99.65	15.61	299.36
PLR†	0.6	99.97	98.43	87.13	99.9	2.95	1.31
RF‡	0.5	98.23	88.66	67.98	99.74	22.32	25.77
SVM§	0.9	99.83	95.12	79.89	99.84	9.51	52.58
Range	NA	3.28	16.81	32.21	0.44	33.76	332.33

* k = 23.

† $\alpha = 0.1474$ | $\lambda = 2e\text{-}05$.

‡ mtry = 1 | ntree = 500.

§ cost = 0.25 | $\gamma = 1.25$.

(10.2.1) ROC Curves for Holdout Data:

Figure 10.1 below shows the ROC curves for the models that performed better (in terms of AUC) on the holdout data than on the OOF sample from the 10-fold CV.

Figure 10.1: ROC Curves (Holdout and Training Data)

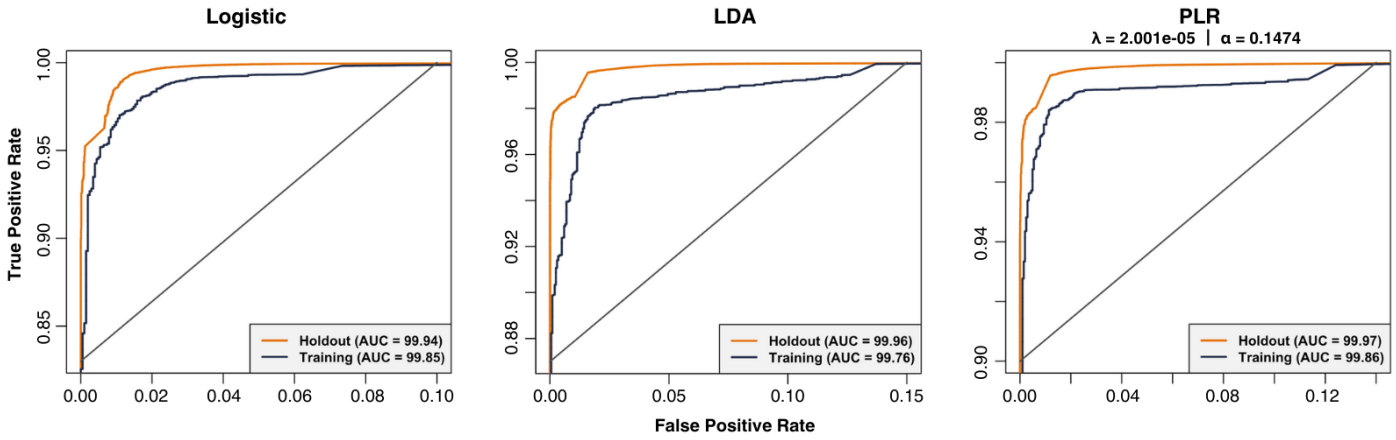
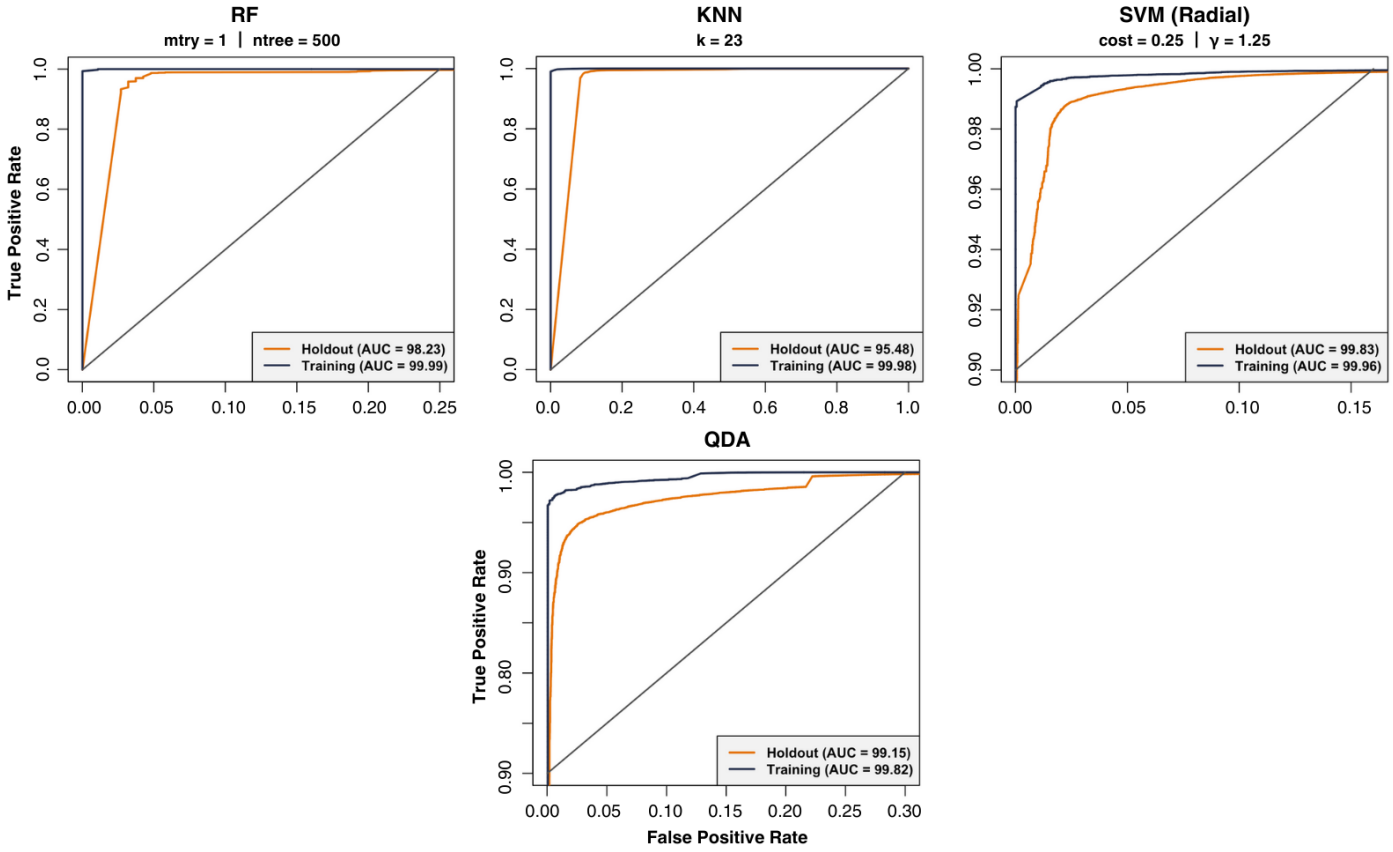


Figure 10.2 below shows the ROC curves for the models that performed worse (in terms of AUC) on the holdout data than on the OOF sample from the 10-fold CV.

Figure 10.2: ROC Curves (Holdout and Training Data)



(11) Conclusions:

(11.1) Best Performing Algorithm:

(11.1.1) Cross-Validation:

The SVM with an RBF kernel performed best on the training data in the CV. While the RF had the highest AUC, the SVM had the second-highest AUC but the lowest FPR and highest Balanced Accuracy. That said, the SVM also had the longest train duration, at 56.69 seconds, which isn't all that long but is roughly five times the duration of the Logistic, LDA, and QDA models. We were also careful in our hyperparameter tuning for the SVM; the recommended γ value may have forced the SVM to perform even better in the CV, but we knew this would likely result in poor performance on the holdout set.

(11.1.2) Holdout:

The PLR model performed best on the holdout set; it has the second-lowest FPR and Balanced Accuracy, the highest AUC, Kappa, and F1 score and is the fastest to deploy. The traditional Logistic and LDA models are close contenders. In fact, we could choose either of these three models as performing best on the holdout set, but PLR strikes a reasonable balance between the performance metrics we consider. Interpretability and communicating model performance are incredibly important, so we could just as easily choose the simpler Logistic model.

(11.2) Discussing Best Performing Algorithm:

The optimal model differs for the CV (SVM) and the holdout (PLR). On the surface, this may seem incompatible, but it is, however, reconcilable; the performance metrics for all models don't fluctuate too much between the CV and holdout testing except for the FPR for QDA, KNN, and RF, and the test duration for KNN. For QDA, KNN, and RF, we believe the models overfit the training data. For KNN, it's possible that $k = 23$ is too high and results in a

less flexible, nearly linear decision boundary. It's also possible the $k = 23$ is too low, resulting in a classifier with high variance and an overly flexible decision boundary. For the RF, as mentioned in section 8.4, it's possible the bootstrapped samples contain few (if any) observations of the minority class, which may partially explain the difference in performance. The performance differences could also have more to do with differences between the training and holdout data.

(11.3) Recommended Model:

All models perform relatively well on the holdout set, but a few stand out. We recommend the PLR model as the best option for detecting the presence of blue tarps. Tested against the holdout set, the PLR model has the highest AUC, Kappa, and F1 score, the second-highest Balanced Accuracy, the second-lowest FPR, and the fastest test time. The Logistic model is a close second, followed by LDA and SVM. The Logistic and PLR models are similar, but the PLR model may have performed slightly better because of the combination of L1 (Lasso) and L2 (Ridge) penalties encouraging sparse coefficient estimates that *might* then minimize the effects of multicollinearity, whereas the traditional Logistic model may overfit the training data in the presence of multicollinearity.

(11.4) Relevance of Performance Metrics:

Throughout this analysis, we emphasized the importance of AUC, Balanced Accuracy, Kappa, F1, and FPR. While training the models and selecting thresholds, we focused intently on the FPR, which may have resulted in biased threshold selection; choosing the threshold with more weight in our consideration given to FPR may result in the models performing poorly on unseen data. This is part of the reason we also chose to include the other performance metrics, but it made selecting the threshold a bit of a trade-off, as the FPR was the only performance metric to fluctuate much between thresholds. Given the degree of class imbalance in both datasets, two of our decisions regarding performance metrics seem appropriate and relevant; those are (1) the use of AUC instead of Accuracy as the metric choice for caret's *train* function and (2) the consideration of Balanced Accuracy, Kappa, and F1 scores as the main performance metrics (alongside FPR in the larger project context of disaster relief).

(11.5) Limitations and Considerations for Future Modeling:

The author's lack of subject matter expertise is a significant limitation of the methods used here. In that light, these models could serve as preliminary algorithms for further refining, specifically in terms of which performance metrics to consider in model evaluation and parameter tuning; future research could achieve this via collaboration with data scientists and non-data workers from involved government entities and non-governmental organizations, including but not limited to the Red Cross, the United Nations, and other humanitarian aid and disaster relief organizations.

Another limitation in the model development and parameter optimization presented here is that we train the models on a limited dataset. The models are accurate, but we would be remiss not to advise further model training on new data from different locations. It may even be that one model performs better on data from urban areas, and another performs better on data from rural environments.

On the data-level limitations, the RGB color scheme is not the most useful for such important use cases. Given more time, we would have benefited from researching deeper into feature engineering and transforming the RGB values into alternative color models that can decouple intensity (HSL, for example) and chromaticity (CIELUV, for example).

On the algorithmic-level limitations, various other approaches to adapt the learning algorithms beyond what we leverage in this report may result in improved model performance; for example, perhaps using the Synthetic Minority Over-sampling Technique (SMOTE) with the RF could improve that algorithms performance.

Each algorithm used in this project also has several underlying assumptions and limitations; we list these in the subsections “Assumptions and Limitations.”

(11.6) Real-World Application:

A wealth of subject matter expertise and related contextual information went untapped for this project. The dataset provided does not tell us where the data were captured; for example, if the data is from photographs taken over Port-au-Prince, then running the predictive models may not be all that useful given that many displaced persons in Port-au-Prince were sheltering under tarps provided by the Red Cross, so their location is assumed to be already known.[14]

There’s much to consider beyond strictly model development in the context of this project. We could find the optimal model for classifying pixel values and identifying blue tarps, but that knowledge is only useful if communicated in such a way that the aid providers on the ground can quickly act on it. It’s one thing to say, “There’s a blue tarp here,” and another to quickly and accurately identify a blue tarp’s location, know the terrain features around it, and navigate to it to provide the necessary relief. A vast chasm exists between the model output and the relief practitioners on the ground, but with communication, iteration, and practice, perhaps the next disaster can apply lessons learned and reduce loss of life and suffering for displaced persons.

---

(12) Sources:

---

1. Reginald DesRoches et al., “Overview of the 2010 Haiti Earthquake,” United States Geological Survey (Washington, DC: 23 Jan. 2011), available at https://escweb.wr.usgs.gov/share/mooney/142.pdf (https://escweb.wr.usgs.gov/share/mooney/142.pdf)↩
2. Government of the Republic of Haiti, “Action Plan for National Recovery and Development of Haiti,” (Port-au-Prince, Haiti: 2010).↩
3. Gary Cecchine et al., “The U.S. Military Response to the 2010 Haiti Earthquake: Considerations for Army Leaders,” The Rand Corporation (Washington, DC: 2013).↩
4. James, G. et al., *An Introduction to Statistical Learning*, (Springer, 2021) pp. 198-208.↩
5. <Kuhn, M., “Building Predictive Models in R Using the caret Package,” *Journal of Statistical Software*, vol. 28, no. 5 (2008) pp. 1–26.>↩
6. Jacob Cohen, “A Coefficient of Agreement for Nominal Scales,” *Journal of Educational and Psychological Measurement*, vol. 20, no. 1 (1960) pp. 37-46.↩
7. Esposito, C. et al, “GHOST: Adjusting the Decision Threshold to Handle Imbalanced Data in Machine Learning,” *Journal of Chemical Information and Modeling*, no. 61 (2021) pp. 2623-2640.↩
8. J. Richard Landis and Gary G. Koch, “The Measurement of Observer Agreement for Categorical Data,” *Biometrics* vol. 33, no. 1 (1977) pp. 159-174.↩
9. Chawla, N.V., *Data Mining for Imbalanced Datasets: An Overview*, in Maimon, O., Rokach, L. (eds) “Data Mining and Knowledge Discovery Handbook,” Springer, (Boston, MA: 2005).↩
10. Ibid, Chawla, N.V. (pp. 857).↩
11. Meng Liu et al., “Cost-Sensitive Feature Selection by Optimizing F-Measures,” *IEEE Trans Image Process* vol. 27, no. 3, (2018), pp. 1323-1335.↩

12. Ibid, James, G. et al. (pp. 39).↩

13. Ibid, James, G. et al. (pp. 341).↩

14. American Red Cross, “Haiti Earthquake Relief: One-Year Report,” (2011), available at https://www.redcross.org/content/dam/redcross/atg/PDF_s/HaitiEarthquake_OneYearReport.pdf (https://www.redcross.org/content/dam/redcross/atg/PDF_s/HaitiEarthquake_OneYearReport.pdf).↩