DS6030 Disaster Relief Project 2

Group 4: Alanna Hazlett, Etienne Jimenez, Venkat Viswanathan

Introduction

In 2010 Haiti was hit by a devastating earthquake. Many homes were destroyed and people were making shelters utilizing blue tarps. In an effort to provide assistance with food and water, the Rochester Institute of Technology was flying over the country capturing images to locate people utilizing these blue tarps. They did not have enough time or manpower to sift through these photos by hand to locate people, as they wanted to get help to them as quickly as possible. They utilized the pixel colors and the amounts of those pixels to create a model to identify where the blue tarps are. Here, we too are building a model to identify the blue tarps based on pixel data in an effort to quickly find survivors who may need assistance. Once the model is built, we could utilize it on future data, say from another natural disaster where we know people are using blue tarps, to make these same predictions. This will help speed up recovery efforts and provide support to survivors.

Data

The data we utilized to train our model was in a csv format, called HaitiPixels.csv, and had the following variables.

- **Class**, a categorical variable describing the landscape/architecture identified in the photograph. Comprised of Vegetation, Soil, Rooftop, Various Non-Tarp, and Blue Tarp. Blue Tarp is the category of interest for predictions.
- **Red**, a numeric variable for the amount of red pixels in the photograph.
- **Green**, a numeric variable for the amount of green pixels in the photograph.
- **Blue**, a numeric variable for the amount of blue pixels in the photograph.
- For building our model we created a dataframe called **binary_training_data**, which is comprised of the same columns as training_data, however Class was mutated into a binary category, with NotBlueTarp and BlueTarp. BlueTarp is the category of interest for predictions.

The data provided for us to make predictions on was separated into multiple txt files.

- orthovnir057_ROI_NON_Blue_Tarps.txt
- orthovnir078_ROI_Blue_Tarps.txt
- orthovnir069_ROI_NOT_Blue_Tarps.txt
- orthovnir069_ROI_Blue_Tarps.txt

- orthovnir067_ROI_NOT_Blue_Tarps.txt
- orthovnir067_ROI_Blue_Tarps.txt
- orthovnir078_ROI_NON_Blue_Tarps.txt
- orthovnir067_ROI_Blue_Tarps_data.txt

All these files except orthovnir067_ROI_Blue_Tarps_data.txt were used to create our holdout dataframe, as this file is a repeat of orthovnir067_ROI_Blue_Tarps.txt. The first 8 lines of the .txt files are metadata we did not need, so they were skipped. We only retained the relevant columns we needed to make our predictions, B1, B2, and B3. These columns correspond to Red, Green, and Blue in our training data, and we used techniques to determine which column in the holdout corresponded with which column in the training data. For the holdout dataset the class label comes from the individual file names. For files containing “ROI_NON_Blue_Tarps” or “ROI_NOT_Blue_Tarps” in the file name we added a column called Class, where every row was “**NotBlueTarp**”. For files containing “ROI_Blue_Tarps” we added a column called Class, where every row was “**BlueTarp**”. Finally, all of the individual dataframes were merged row by row to create our Holdout dataset.

In total, the seven files totaled over 2 million observations, and within each we find nine columns describing distinct characteristics of pictures taken above the affected areas in Haiti. We noticed that the first six columns contain an ID marker and descriptive information of each picture's location, while the last three columns contained three numbers which highly resembled the three colors from our training data set.

Description of Methodology

To perform our calculations and analysis, we employed RStudio's libraries **Tidyverse, Tidymodels, discrim, patchwork, probably, reshape2, ggcorrplot, vip, ranger , bonsai, GGally, kknn, kernlab, and doParallel**. All models were trained using Tidymodels' functions, such as recipe, workflow, as well as their pre-built functions to carry out specifications, tuning, training, and fitting for Logistic Regression (Log Reg), Linear Discriminant Analysis (LDA), Quadratic Discriminant Analysis (QDA), K Nearest Neighbors (KNN), Tuned Logistic Regression (Tuned Log Reg), Random Forest (RF), Linear Support Vector Machine (SVM Linear), Polynomial Support Vector Machine (SVM Polynomial), and Radial Basis Function Support Vector Machine (SVM RBF).

We now describe our approach from starting to check out the data's properties, to interesting visualizations and proportions we found in the data, to eventually adding model specifications and considering the training data into the model fitting. We started by carrying out some exploratory data analysis to get an initial feeling of the structure of our training data.

From the start, we knew that the training data was made up of four columns: three describing the colors found in the pixels from each picture, and one column classifying the terrain of the pictures. Possible options for the Class column include Vegetation, Soil, Rooftop, Various Non-Tarp, and Blue Tarp. What we were most interested in was to have an idea of the distribution and proportions of the Blue Tarps in this data, because those are the places where we need to send aid resources. We start by plotting a bar chart with the proportions of the different Class categories:

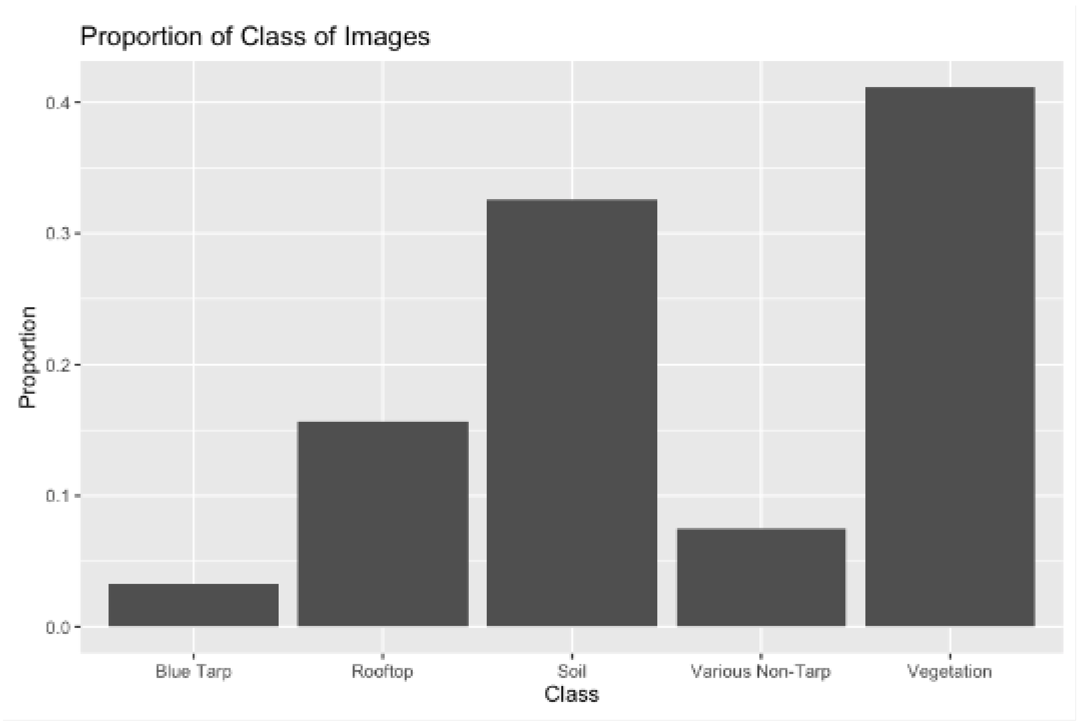


Figure 1

In Figure 1, we observe that the class for most observations is different from Blue Tarp. Indeed, from the results of a proportions table, we see that the Blue Tarp observations comprise only 3% of the training data set.

We found that an overwhelming number of observations in this dataset are of the Vegetation and Soil types, and together they make up more than 70% of the observations. The proportion of observations categorized as Blue Tarp is exceedingly small, making up only 3% of the total data. We now want to give ourselves an idea of how the three color variables are distributed among each Class category and see whether there exists any kind of pattern between colors and Class. To do that, we created the following visualization:

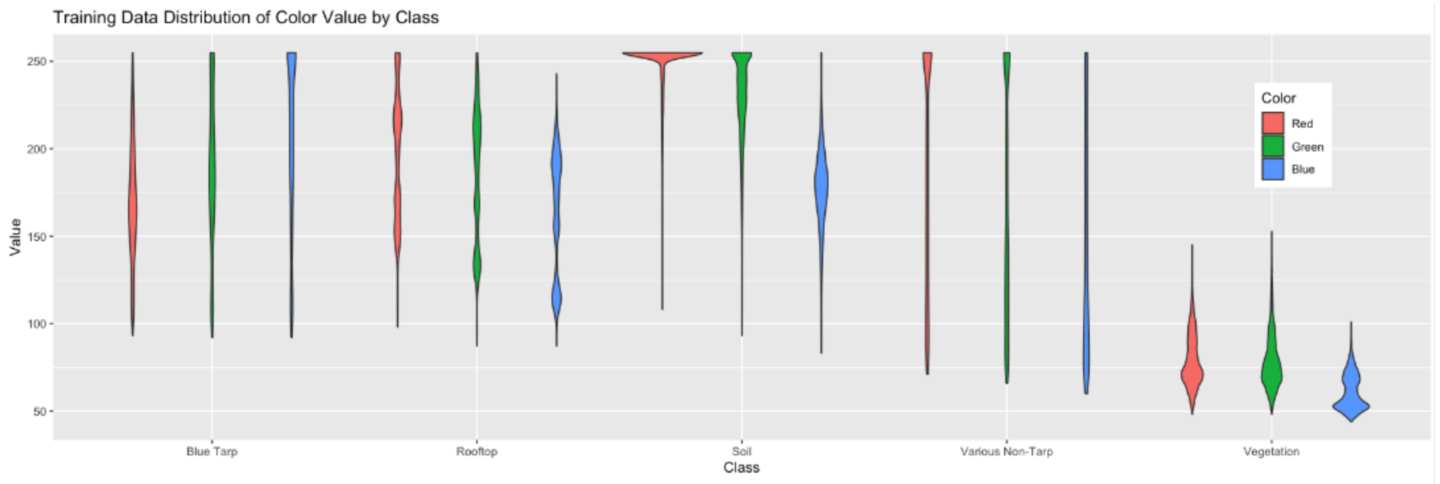
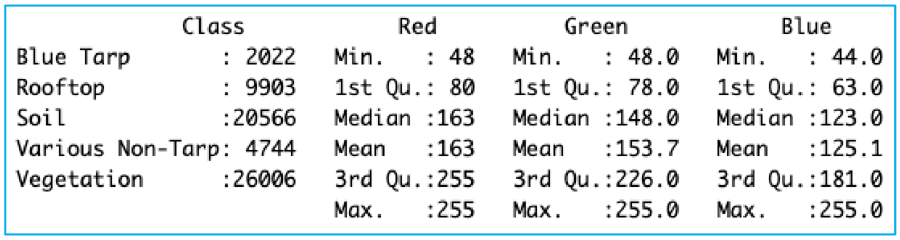


Figure 2



```
                  Class          Red          Green            Blue
Blue Tarp        : 2022   Min.   : 48   Min.   : 48.0   Min.   : 44.0
Rooftop          : 9903   1st Qu.: 80   1st Qu.: 78.0   1st Qu.: 63.0
Soil             :20566   Median :163   Median :148.0   Median :123.0
Various Non-Tarp: 4744    Mean   :163   Mean   :153.7   Mean   :125.1
Vegetation       :26006   3rd Qu.:255   3rd Qu.:226.0   3rd Qu.:181.0
                          Max.   :255   Max.   :255.0   Max.   :255.0
```

Table 1

From Figure 2 and Table 1, we can draw a few remarks. First, the most abundant category in the Class variable, Vegetation, averages lower numbers than all other four categories. Second, the categories of Soil, Rooftop, and Various Non-Tarp all have higher numbers of Red and Green color pixels, and Blue has the smallest average for all three of them. Lastly, the Blue variable has the highest average and concentration for the Blue Tarp category, strongly implying that there may exist a strong correlation between a BlueTarp observation and a high number of the Blue variable.

As mentioned in the Data Section above, our training data consisted of four variables: the Class column and the three colors considered in the EDA section. However, since the goal of our analysis is to identify and predict where an observation could potentially contain a Blue Tarp, we mutated the Class column to point out whether the Class was BlueTarp or NotBlueTarp, converting this binary column into a factor before we continue carrying out our model training. The three color variables will remain numeric, and they will serve as the three predictors for our models.

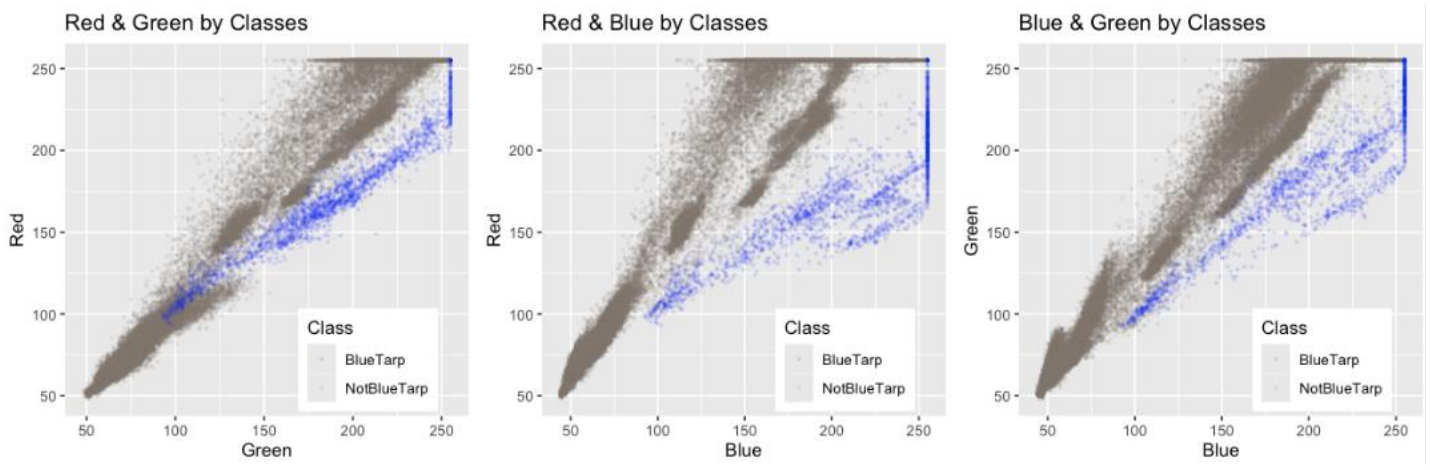


Figure 3

From Figure 3, we see that all trends of the distribution of BlueTarp follows a distinctive path from observations marked otherwise, with the separation slightly less distinct in Red against Green. For the plots with Blue as a value, BlueTarp observations start after the Blue predictor is around 95, all the way to the max possible value of 256.

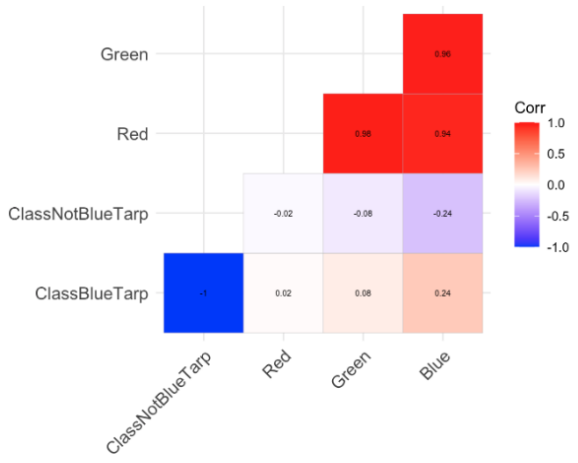


Figure 4: Correlation Matrix of Predictors and Response Outcomes

On this correlation matrix in Figure 4, we note that all three predictor variables appear to be highly positively correlated, with all three pairs of variables returning a 0.94 correlation or higher. As expected, the Blue variable is the one most correlated with the presence of Blue Tarps at 0.24 correlation.

In the holdout set, the columns present have the names B1, B2, and B3. Thus, we set out to find statistical patterns between the three color variables in the training set and these three columns in the holdout set, to determine which color corresponds to which column.

We started by figuring out how many of the training data's rows had Red as the highest number out of the three colors, and doing the same for the Green variable. After dividing each number by the total number of rows, we found that 67.5% of the training data's observations had Red as (strictly) the highest number out of the three, with 17.2% having Green as the highest number out of the three. Blue was the highest number in only 2.5% of those observations, which is consistent with the proportion of BlueTarp observations in the Class column.

Moving into the holdout set, we ran identical code on the three columns B1, B2, and B3. Interestingly, we found that the proportion of observations where B1 was strictly the highest out of the three was 76.04%, for B2 this number was 15.3%, while for B3 it was only 1.2%.

In addition to the proportions, we utilize the distribution of the pixel colors. B1 appears similar in distribution to Red, as they share a large count near 250 and have the most density around 50-125 pixels. B2 appears similar in distribution to Green. B2 compared to B3 has a more significant count near 250. We propose green is more likely to retain higher pixel values than blue due to its presence in nature. B2's highest density is a higher pixel value than that of B3, which aligns with Green and Blue respectively. B3 appears similar in distribution to Blue. For Blue, most density is lower than Red or Green, and this holds true for B3 compared to B1 or B2.

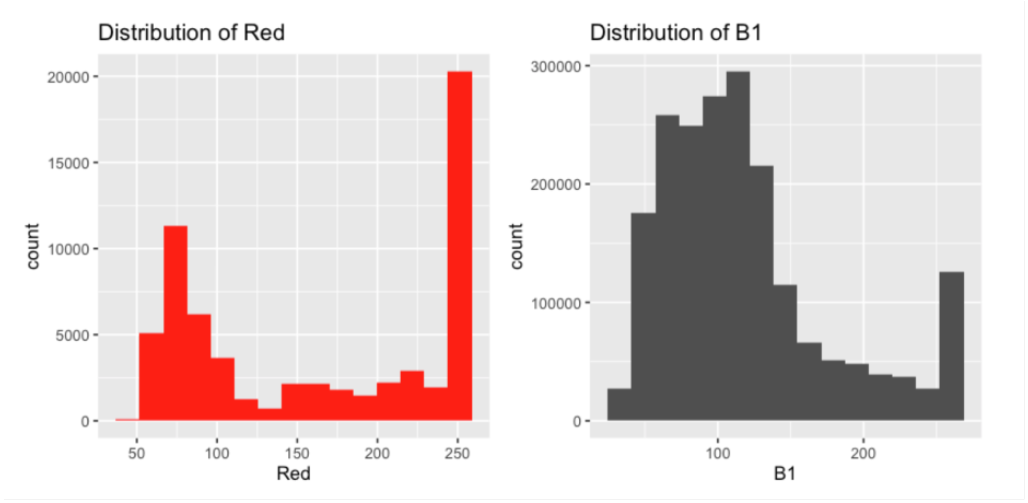


Figure 5 (a)

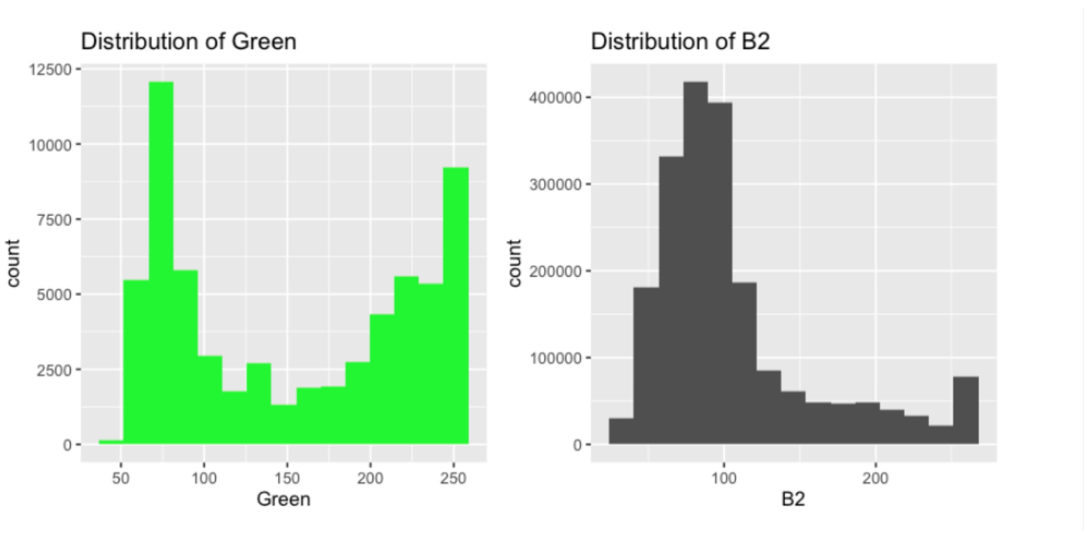


Figure 5 (b)

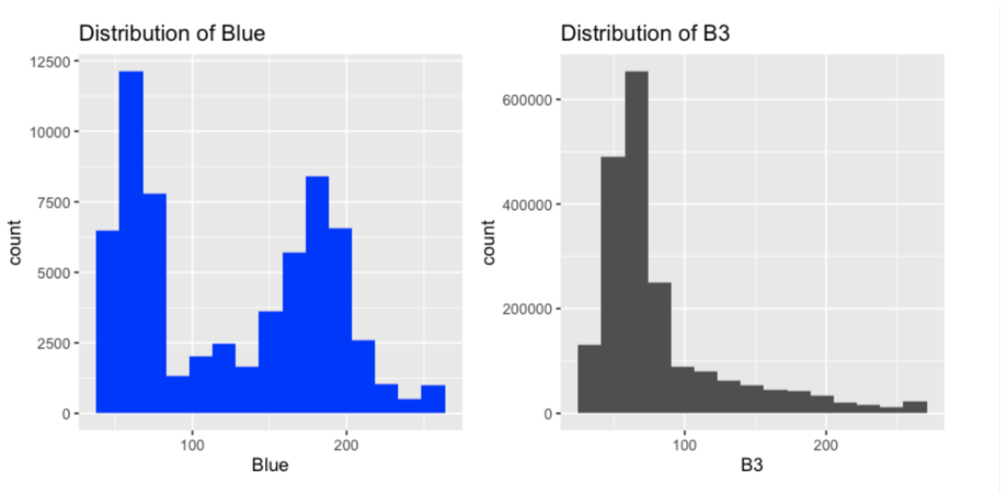


Figure 5 (c)

We compare the distributions of the individual variables from the holdout set with those closest in cumulative distribution to them.

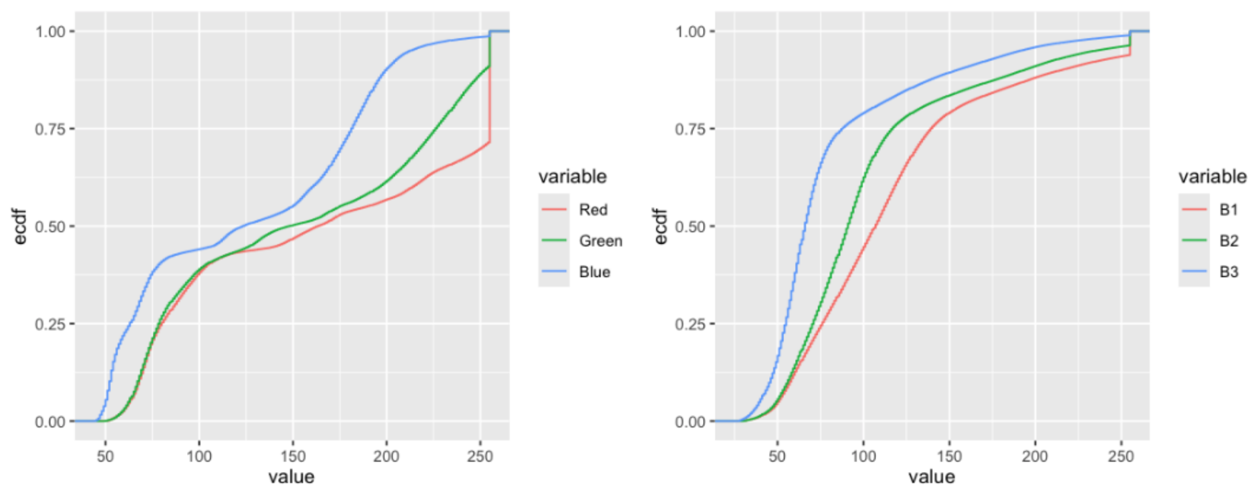


Figure 6: Cumulative Distribution of Pixel Colors

In Figure 6, we find the cumulative distributions for the three predictor variables on the training set (left) and the three unlabeled variables of the holdout set (right).

With this statistical data in hand, we are confident to mark the value of B1 as that of Red, as it was by far the largest out of the three and with a highly similar proportion as the biggest number in the training data set. We mark the value of B2 as Green, also with an almost identical proportion as in the training set and higher than the last number. This implies we mark the final value of B3 as Blue.

The classification models for all nine models were trained following the workflow process from the **Tidymodels** package: we prepare the regression formula, and we pass it together with the training data set to the recipe function. In this step, we normalize all our numeric predictors. We define the specifications for all models before finally defining the workflows for each, by adding the recipe defined above as well as the specifications for each model. This sets the environment to carry out model tuning, which allows us to target specific characteristics to identify the best version of our models ahead of finding their final fit.

Tuning

As we prepare to tune our models, we start by clearly defining the recipe, the model specifications, and the parameters for tuning. Each model will have different specifications and parameters to be tuned, but they all share the same recipe, consisting of the regression formula, the training data, and normalized predictors.

Another thing that remains the same across every model we are tuning is the **tune_bayes()** function, which we choose in this case over the **tune_grid()** function given that

tune_bayes() is more efficient and effective as we work with bigger data sets, and despite the additional computational time it takes to run, will provide more accurate results for all models. The parameters found inside the **tune_bayes()** function include the defined workflow, resamples from 10-fold cross validation, the param_info containing each model's tunable parameters, metrics the metrics we wish to evaluate, all of them run for 25 iterations. The **autoplot()** function displays the results from the latest tuning action, depending on the model parameters.

For KNN, we make sure to tune the number of neighbors, a parameter that controls the flexibility of the model. The default values are from 1 to 15. We notice an increase in ROC-AUC until about the eighth neighbor, and values start leveling off. We tune the number of neighbors from 1 to 8 to find the most ideal value of ROC-AUC, and now all of them appear to be relatively on the same level.

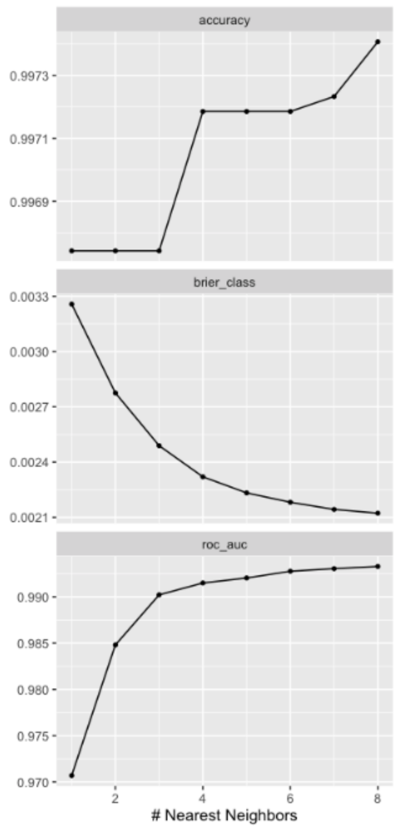


Figure 7: KNN Tuning Plot

For Tuned Log Reg, we pre-process the model in almost the exact same way as with the untuned version of logistic regression. The only exception is that this time we carry out both L1 and L2 regularization by tuning the penalty and mixture hyperparameters inside the model's specifications. The default range for them starts from -10 to 0 for the penalty, and from 0 to 1 for the mixture. After our first round of tuning, we realize we may remove some of the higher values for the penalty parameter, and reduce this interval from [-10,0] to [-10, -4]. For the mixture, the first quartile displays lower results than those on the rest of the interval. We slightly altered this interval from [0,1] to [0.25,1], and now obtain the following plot results.

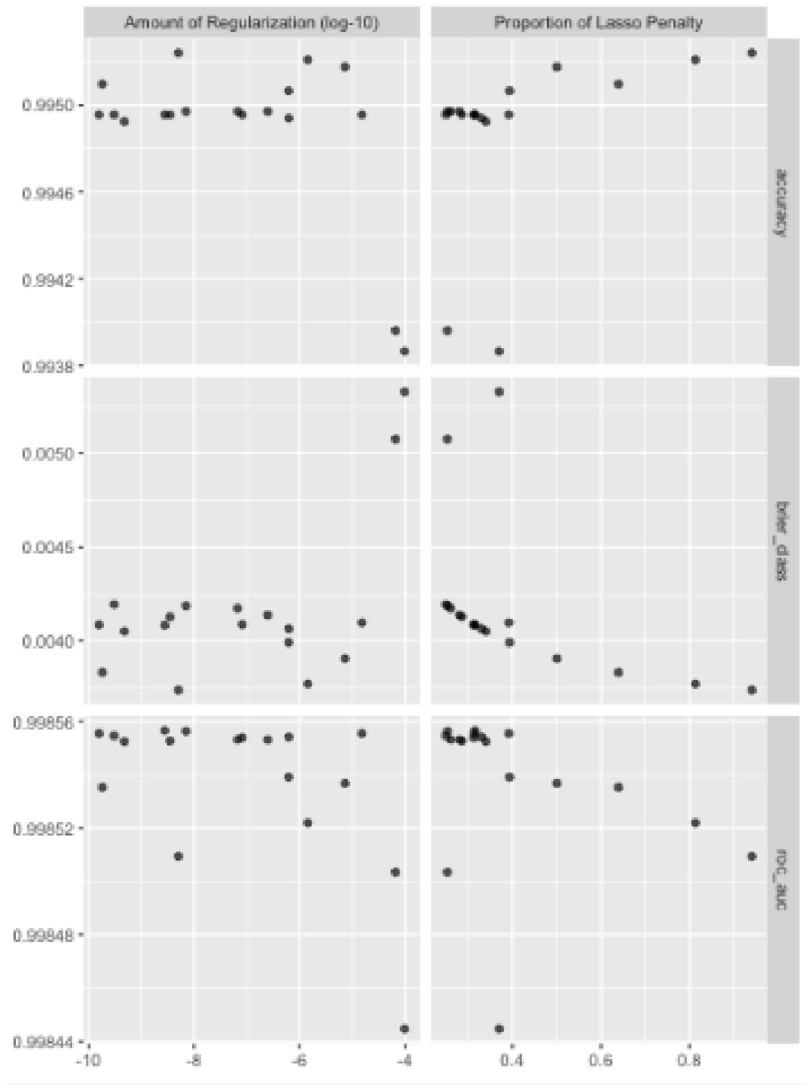


Figure 8: Penalized Logistic Regression Tuning Plot

We follow next with the RF model. The specifications for this model come from the function **rand_forest(),** which allows us to define the mode (classification in this case), as well as the mtry and min_n parameters. The mtry parameter states how many predictors will be considered at each split when creating the tree branches, while the min_n parameter defines the minimum number of observations required before we can further split a node. By default, RF models assign mtry as [1, 3], and min_n to the interval [2, 40]. We pass all this information to our **tune_bayes()** function.

We minimized the range of min_n down to between 2 and 20 as this is where the variance in ROC-AUC values occurs. We selected the best-performing model based on ROC-AUC. In this case, our ideal number for mtry is 2, and for min_n it is 10.

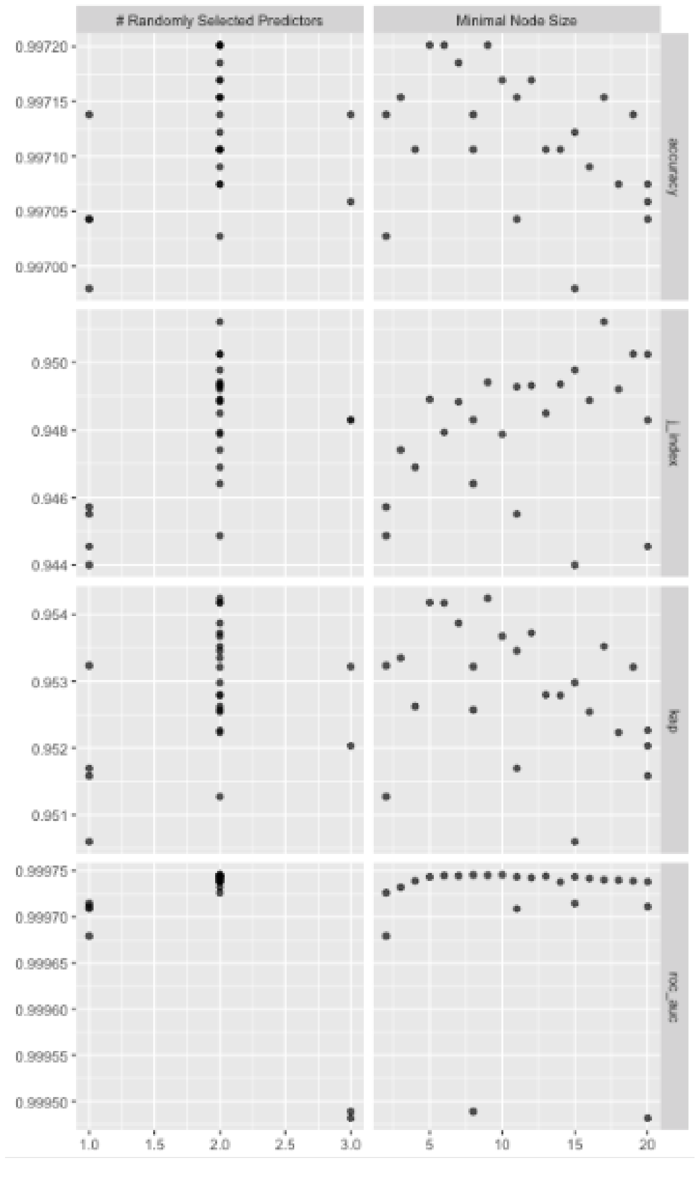


Figure 9: Random Forest Tuning Plot

We now go into the last three models, all of which are different versions of the main Support Vector Machine (SVM) model. We start by considering the SVM Linear model, which has cost and margin as its tunable parameters. Here, cost refers to the cost of a violation of the allowed margin, and by default has the interval [-10, 5]. The margin has default values of 0 to 0.2. The **tune_bayes()** function now returns the following plot, which shows that the original default parameters are doing a good-enough job at calculating our model metrics, and we do not need to update any more of the workflow's parameters. The ideal cost value is 0.0452 and margin of 0.0299.

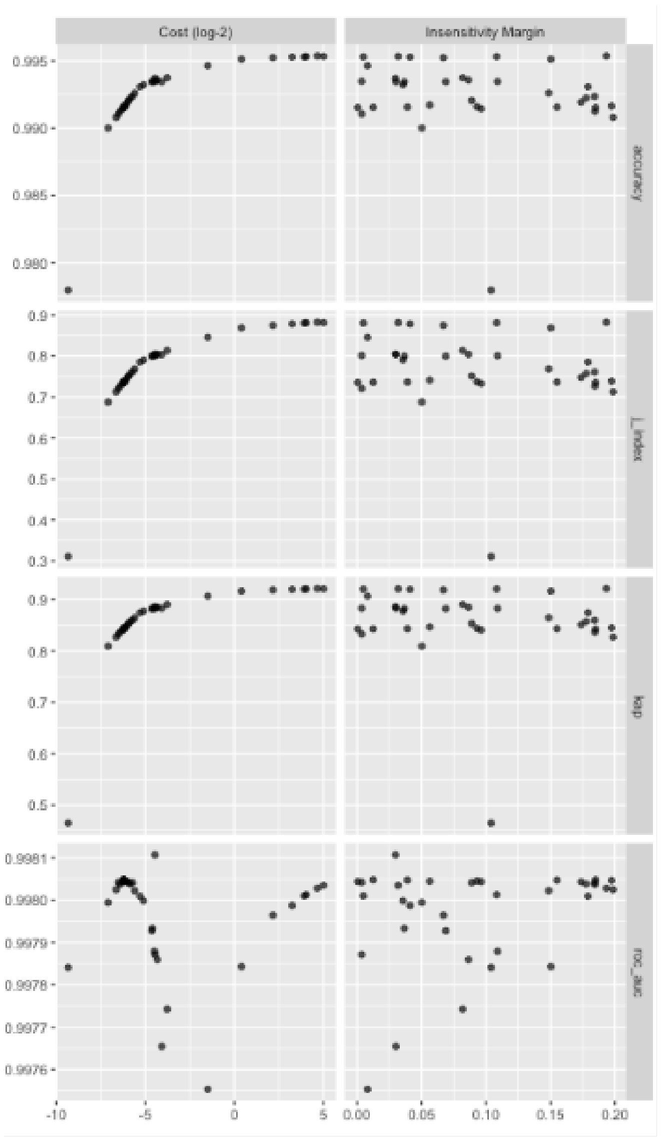


Figure 10: SVM Linear Tuning Plot

The next model will be SVM Polynomial, which in its specifications now contains the cost, margin, and degree parameters. We also want to tune the degree and find the ideal polynomial curve that will best fit this type of model. The default for degree is the interval [1,3] , cost defaults to the interval [-10, 5], and margin defaults to [0,0.2]. From this plot's results, we see that values stay relatively normal after cost is reaches about 5. We obtained the following plot, which we now consider has satisfactory results for this tuning phase. As it is given here, the ideal degree for our SVM Polynomial model is a cubic expression with cost given by 0.1575 and margin of 0.0995.

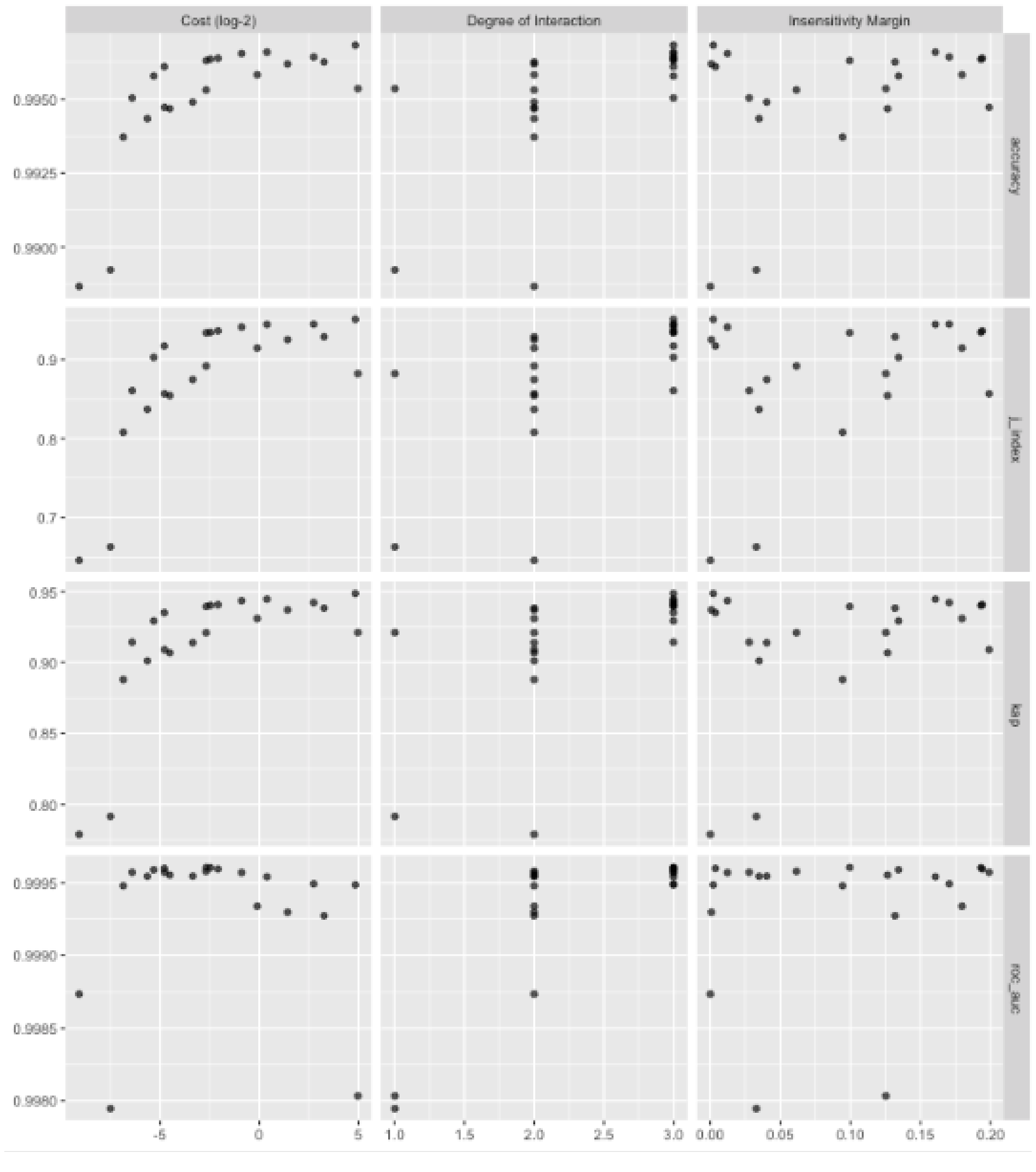


Figure 11: SVM Polynomial Tuning Plot

The last model we are considering here is the SVM RBF. This model will contain three tunable parameters, which are cost, margin, and rbf_sigma. The latter one defines the number passed to the radial basis function, which aims to maximize the width of the margin between classes using a nonlinear boundary. By default, the value of rbf_sigma is 0.2, and we will leave it this way for our first run of results. After passing everything to our tuning function, we update cost to be between –1 and 5 and rbf_sigma to be between –2.5 and 0 and achieve the results in Figure 12.

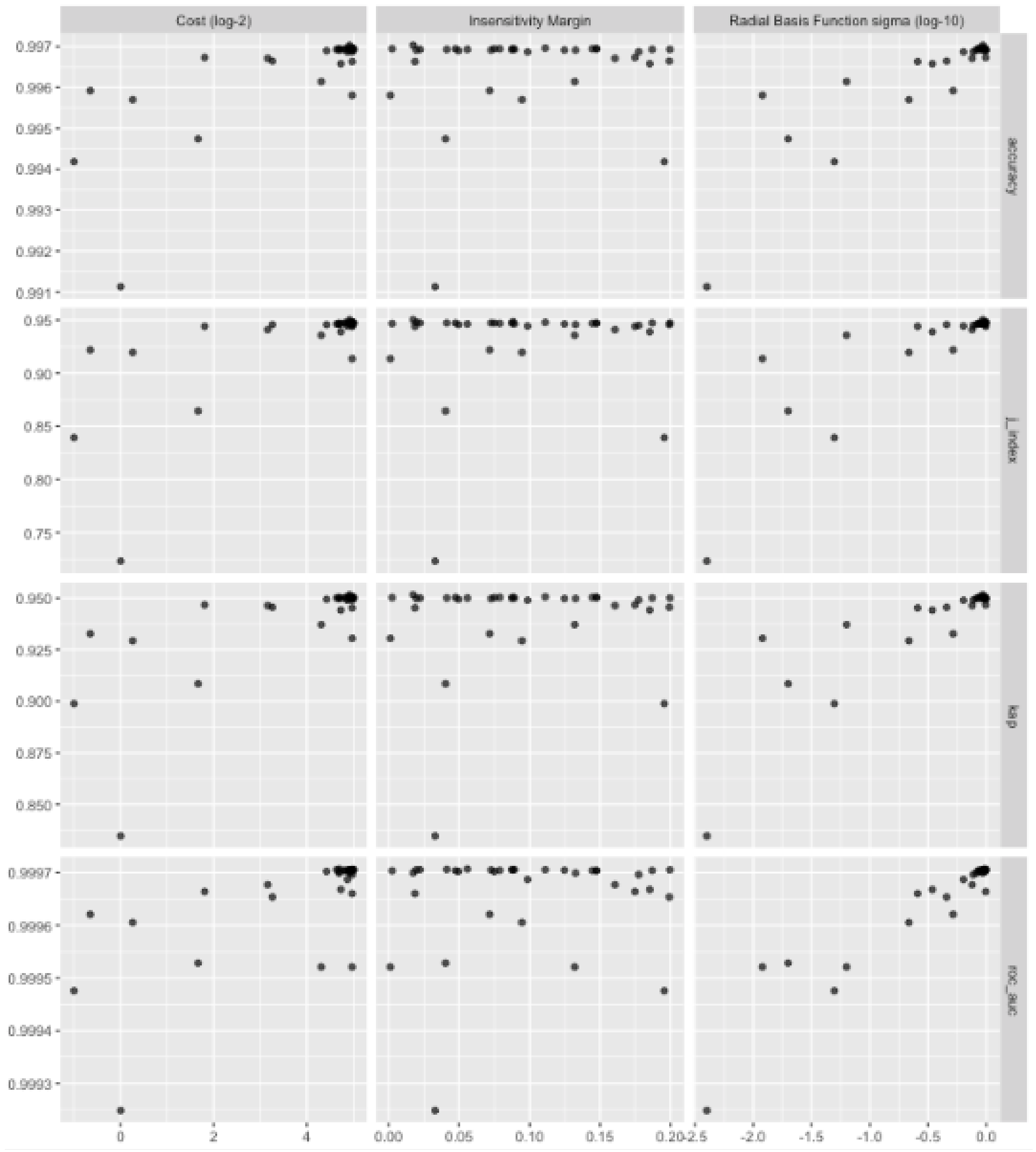


Figure 12: SVM RBF Tuning Plot

Tuning Summary for Various models

Model	Tuning Parameter	Value Range	Selected value to build model
KNN	Number of neighbors	1-15	8
Tuned Logistic Regression	Penalty Mixture	-10 to –4 0.25 to 1	-4 0.25
Random Forest	Mtry min_n	2-20 1-3	2 10
SVM Linear	Cost Margin	-10 to 5 0 to 0.2	0.0452 0.0299
SVM Poly	Cost Margin	-10 to 5 0 to 0.2	0.1575 0.0995
SVM Radial	Cost Margin RBF_Sigma	-2 to 32 0 to 0.2 log10(-2.5 to 0)	25.9816 0.0559 0.9806

Table 2

Cross Validation

As we prepare for cross-validation, we also determine the number of subsets on which to perform the cross-validation (10) and stratified sampling on the response variable, to ensure an even distribution on all of the subsets. This is all done using the **vfold_cv()** function. We also prepare the metrics we want to determine for the cross-validation, which for this analysis include the **ROC- AUC, accuracy, Kappa, and J-index.** Lastly, we use the **control_resamples()** function to save the predictions we draw from cross-validation.

We now complete the cross-validation step by initializing the **fit_resamples()** function utilizing each model's workflow. Besides the workflows, we also need to pass the above-defined resamples, metrics, and control functions to them, and now our cross-validation results are ready. We now display the metrics using the **collect_metrics()** function.

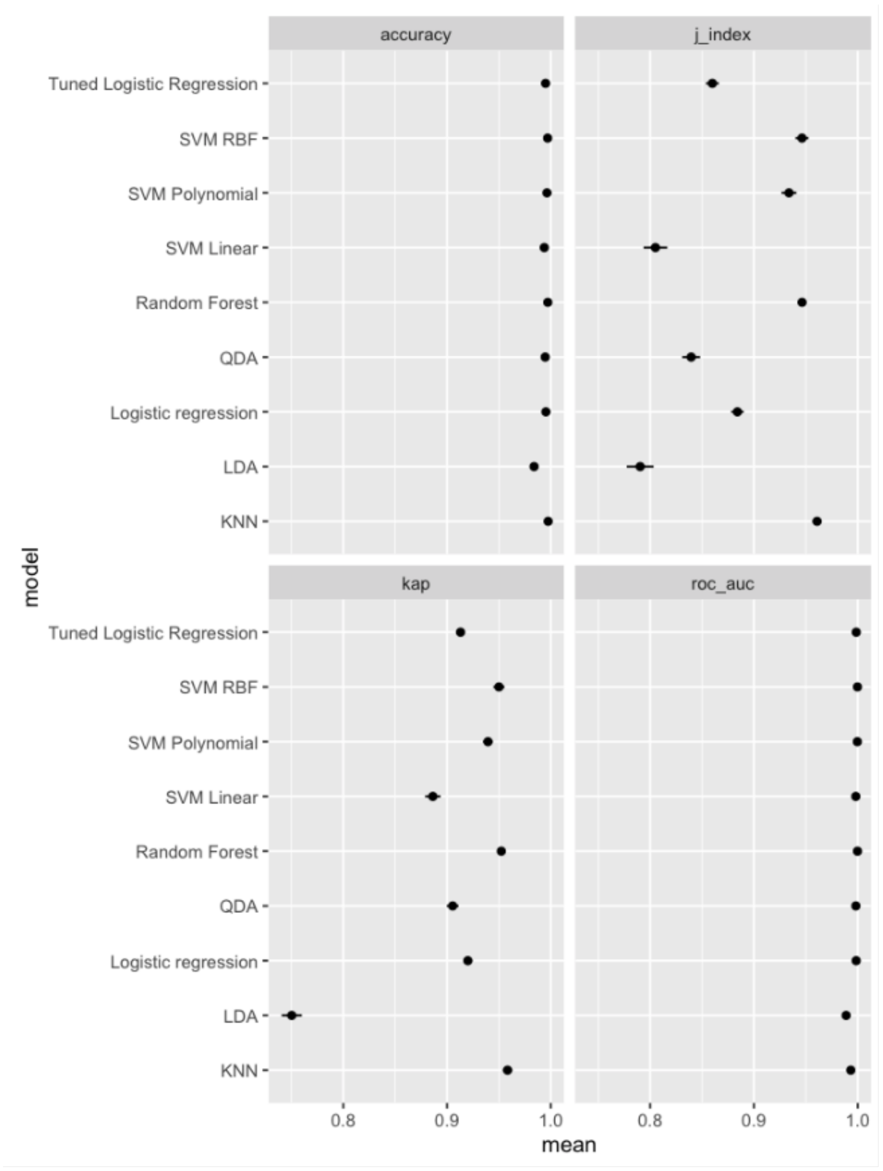
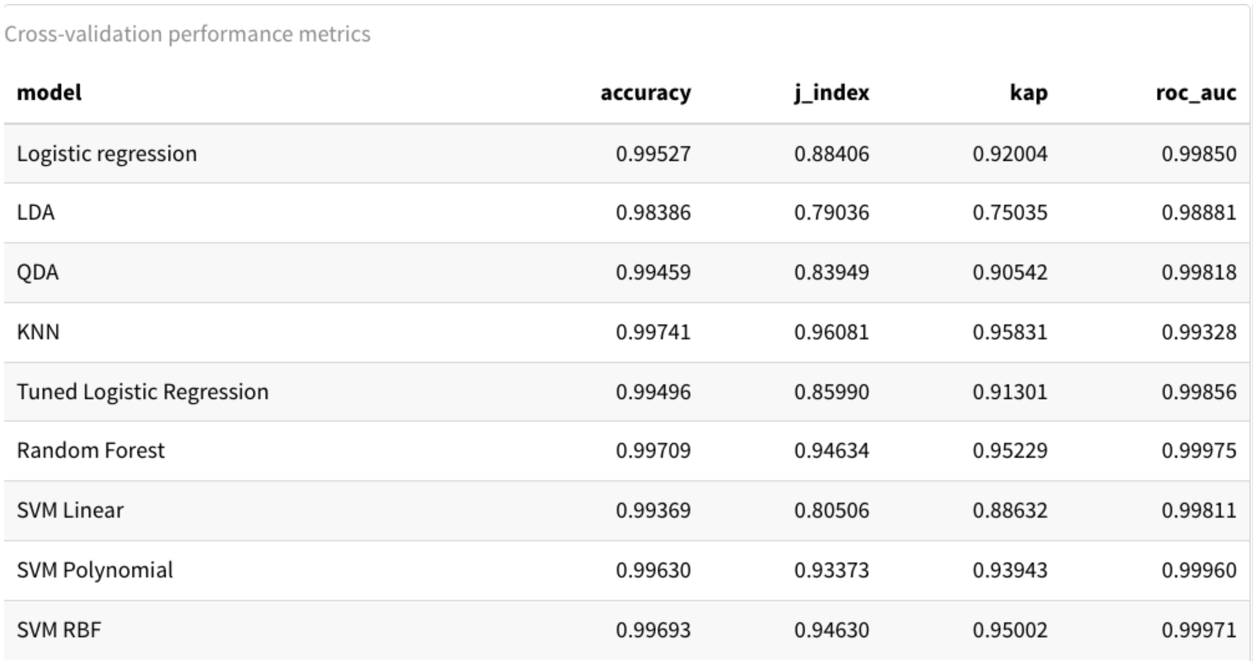


Figure 13: Cross Validation Metrics Graph



Cross-validation performance metrics

model	accuracy	j_index	kap	roc_auc
Logistic regression	0.99527	0.88406	0.92004	0.99850
LDA	0.98386	0.79036	0.75035	0.98881
QDA	0.99459	0.83949	0.90542	0.99818
KNN	0.99741	0.96081	0.95831	0.99328
Tuned Logistic Regression	0.99496	0.85990	0.91301	0.99856
Random Forest	0.99709	0.94634	0.95229	0.99975
SVM Linear	0.99369	0.80506	0.88632	0.99811
SVM Polynomial	0.99630	0.93373	0.93943	0.99960
SVM RBF	0.99693	0.94630	0.95002	0.99971

Table 3

From Figure 13 and Table 3, we start by pointing out that the accuracy and ROC-AUC for all models is already very high, with all models except LDA being above 0.99 for both metrics. The models' metrics only start diverging once we consider the kappa index and the J-index, both of which show four models above all the rest: KNN, RF, SVM Polynomial and SVM RBF. Out of the four, we note that KNN performs slightly better than the other three across all metrics.

We now turn our attention to the ROC and associated AUC. This is a crucial metrics and, as mentioned above, the values for all nine models are very close to one another, so we give ourselves a better idea of the differences between them by creating a visualization containing the above values for all three models.

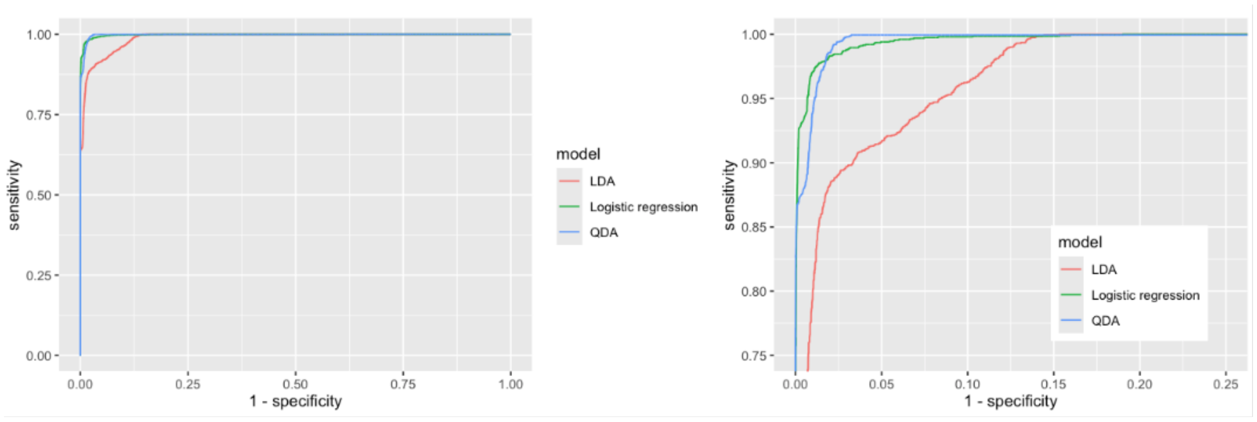


Figure 14 (a)

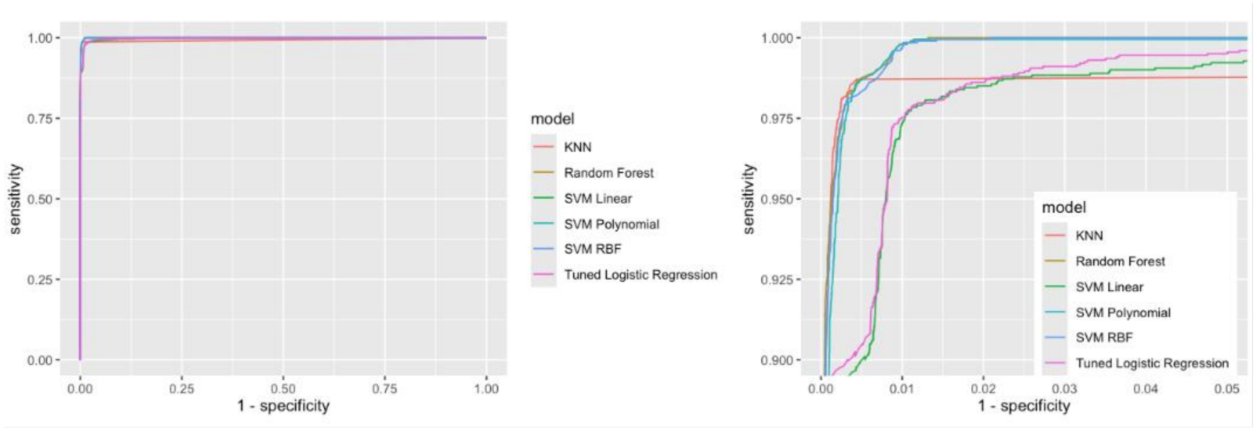


Figure 14 (b)

We split the nine models into two separate plots: the three untuned models on Figure 14 (a), and the six tuned models on Figure 14 (b).

We can better understand how our models are predicting data by running some preliminary tests on their performance on the holdout data. Before moving on, we employ the **augment()** function to the models we have trained so far and visualize how well they are able to predict BlueTarp in the following plot in Figure 15:

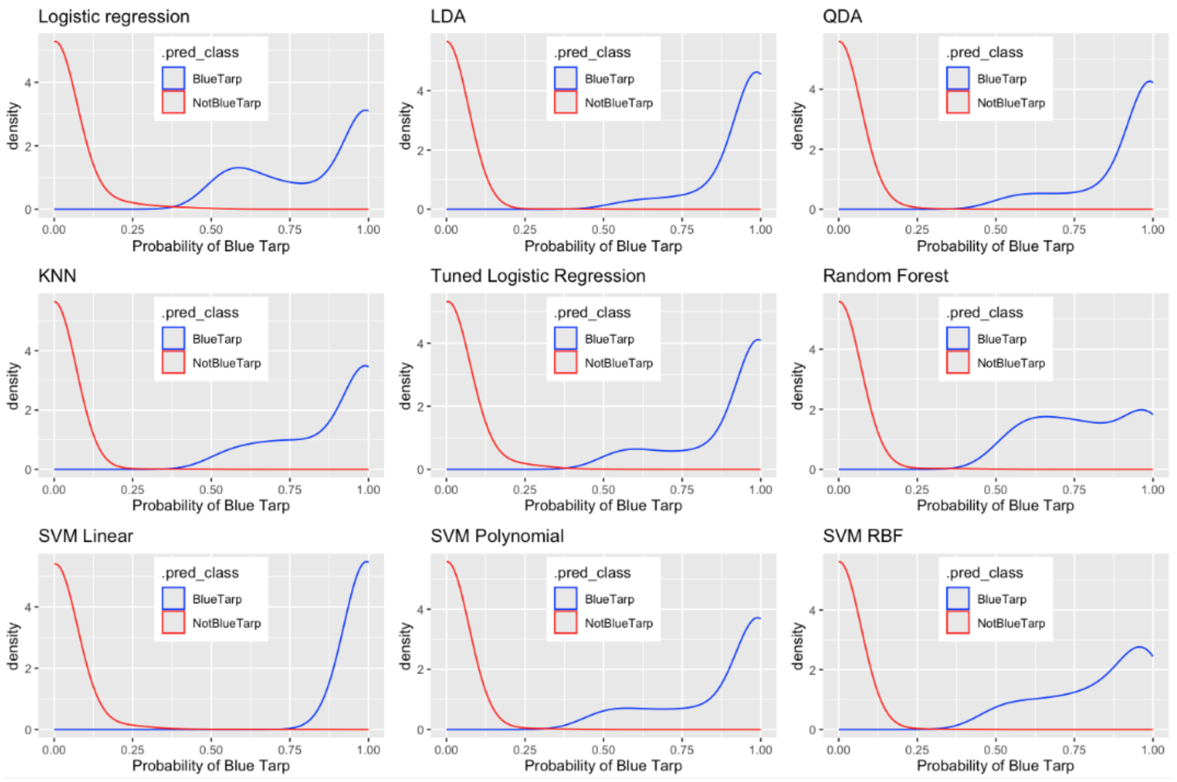


Figure 15

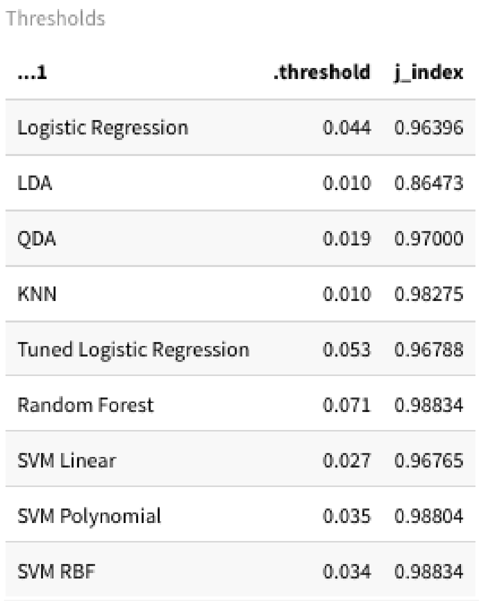
From Figure 15, we note that all nine models have a very high density of points assigned as NotBlueTarp when their probabilities of this classification are less than 25%. A majority of models start classifying observations as Blue Tarp for those who have probabilities as low as 50%. We identify a higher density of points in the range between 50% and 75% for Log Reg, RF, and SVM RBF than in all others. For the other models, the points positively identified as Blue Tarp had remarkably high probabilities, with the vast majority having over 75% to 80% chance.

The final question to answer is how we determine the threshold with which we will train our final models. We do this by utilizing the **threshold_perf()** function from the **probably** package to get the relationship between the threshold values and performance. We determine the best J-index metric for each model and evaluate the model's performance for this assigned threshold value. We decided to implement the J-index rather than the accuracy because we are working with an imbalanced data set, and we see that the high accuracy levels for all the models across all threshold levels may not singularly point out where the best performance is found.

We can find the highest J-index value by augmenting the training set, that is adding columns containing the values and probabilities predicted by the model for each observation, adding a sequence of thresholds from 0.01 to 0.99 in increments of 0.001 and extracting the best J-index performance out of all the selected thresholds.

Results

The following table summarizes our findings for the best J-Index and related threshold selection information.

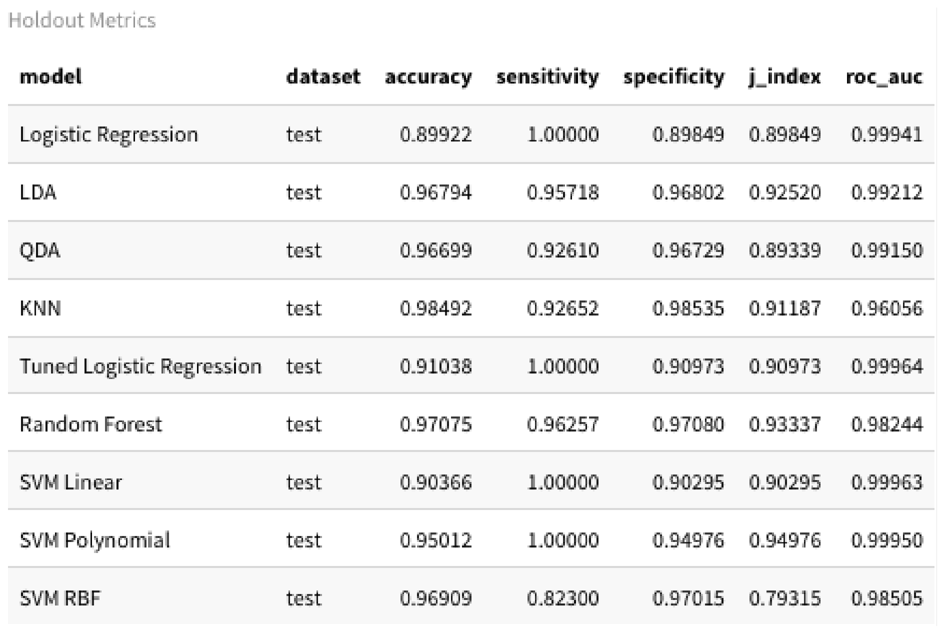


Thresholds

...1	.threshold	j_index
Logistic Regression	0.044	0.96396
LDA	0.010	0.86473
QDA	0.019	0.97000
KNN	0.010	0.98275
Tuned Logistic Regression	0.053	0.96788
Random Forest	0.071	0.98834
SVM Linear	0.027	0.96765
SVM Polynomial	0.035	0.98804
SVM RBF	0.034	0.98834

Table 4

As we see in Table 4, the model with the best J-Index value is the one for RF among the 9 models followed closely by SVM Polynomial. LDA remains the worst-performing model out of the nine. We then proceed to determine the accuracy, sensitivity, specificity, J-Index and ROC-AUC for each model at the determined thresholds.


Holdout Metrics

model	dataset	accuracy	sensitivity	specificity	j_index	roc_auc
Logistic Regression	test	0.89922	1.00000	0.89849	0.89849	0.99941
LDA	test	0.96794	0.95718	0.96802	0.92520	0.99212
QDA	test	0.96699	0.92610	0.96729	0.89339	0.99150
KNN	test	0.98492	0.92652	0.98535	0.91187	0.96056
Tuned Logistic Regression	test	0.91038	1.00000	0.90973	0.90973	0.99964
Random Forest	test	0.97075	0.96257	0.97080	0.93337	0.98244
SVM Linear	test	0.90366	1.00000	0.90295	0.90295	0.99963
SVM Polynomial	test	0.95012	1.00000	0.94976	0.94976	0.99950
SVM RBF	test	0.96909	0.82300	0.97015	0.79315	0.98505

Table 5

Table 5, the holdout metrics table shows expanded metrics from the holdout dataset: **Accuracy, Sensitivity, Specificity, J-Index, and ROC-AUC.**

We can single out that, by the ROC-AUC metric, Tuned Log Reg is the best-performing model out of all nine, with SVM Linear following closely behind. It is worth noting, however, that all models have outstanding ROC-AUC metrics, since all except for KNN have values above 0.98, making most differences noticeable only after the thousandth decimal place.

Focusing now on accuracy, KNN is clearly the superior model, with only RF following closely behind. Every model has an accuracy of at least 0.90 except for Log Reg. Accuracy proves to be an important metric where all models can be distinctively ranked from best to worst, but given the imbalanced data sets which we are working on, it may not prove to be the most trustworthy metric on which to base our final decision. It would be wise to now consider the J-Index as well.

Looking closely at the J-Index column, we see that SVM Polynomial ranks the highest out of all models, with RF being ranked second-highest. KNN, which had the highest accuracy value, is slightly below these two. Finally, we consider the specificity-sensitivity trade-off, before we finally can decide on which model ranks best overall.

Starting by sensitivity, for this metric we evaluate the proportion of True Positives to the total number of positives for each model. It is interesting to note that there are four models which have a perfect value of 1.0, and these are SVM Linear, SVM Polynomial, Log Reg, and Tuned Log Reg. Unfortunately, except for SVM Polynomial, the other three models are lacking on the other metrics to really consider them strong candidates as the best model. For RF and KNN, we report sensitivity of 0.96 and 0.92, respectively.

For the specificity metric, we are measuring the proportion of True Negatives to the total number of negatives for each model. This means how good a model is at correctly classifying observations as NotBlueTarp. The highest such value corresponds to KNN at 0.985, with RF coming second at 0.97.

After basing our results on the holdout dataset, we can conclude that RF is the best model overall on the various metrics, as its results are in the Top 2 of most metrics. KNN is a close second, with slightly better specificity, accuracy, and cross-validation metrics than RF.

We can also see Tuned Log Reg has performed the worst based on accuracy, sensitivity, specificity, and J-Index metrics. Neither this nor the Log Reg reported anywhere near ideal results for this data.

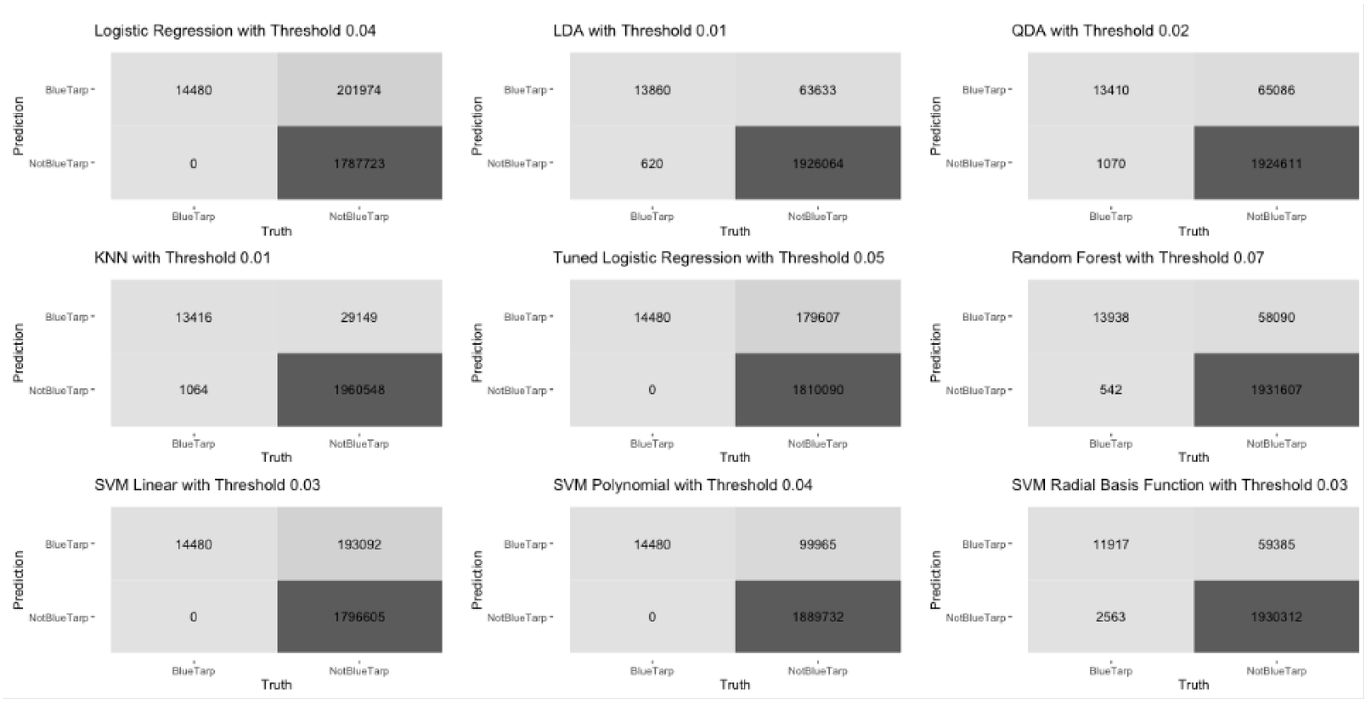


Figure 16

We also look at the proportion of Blue Tarp identified in Holdout. This will also help us identify the performance of each of the algorithms. From our EDA, we calculated the proportions of the possible number of observations that could potentially be classified as BlueTarp, and further compared those instances with the results obtained by testing our refined models in the holdout set. If we can identify how many blue tarps are identified by each, we would know how they perform and what algorithm we can recommend for such real-life scenarios.

Conclusions

A discussion of the best performing algorithm(s) in the cross-validation and hold-out data. (they may not be the same)

Evaluating the performance of each model based on cross validation results only, we see in Table 2 that the model with the highest ROC AUC is RF, which is quickly followed up by SVM RBF. The model that performs the best on the rest of the metrics, accuracy, J index, and Kappa, is KNN. Given that KNN performs the best in several aspects, particularly J index as we know that metric is useful when dealing with imbalanced data, we conclude that it is the best model purely based on cross validation results alone.

Taking a look at the holdout metrics in Table 4, we see that Tuned Log Reg has the highest ROC AUC and is closely followed by SVM Linear and SVM Polynomial. KNN has the highest value for accuracy, which we know isn't as very reliable metric on its own due to the imbalance of the data. SVM Polynomial produces the best J index on the holdout dataset; this combined with its high performance on ROC AUC is what supports our conclusion that SVM Polynomial is the model that performs the best on the holdout.

A discussion or analysis justifying why your findings above are compatible or reconcilable. How sensitive we believe the model is to a shift in the decision boundary.

For our team, it took a considerable amount of discussion before we came up with the best overall model, and it was a topic that we analyzed from a statistical and practical viewpoint. RF is not the most robust model for any specific metric, as we found other models that were the strongest at one or more aspects from the cross-validation or holdout data performance. However, we still find faults in those models, which add up against the more robust profile brought on by RF. Ultimately, the confusion matrices gave us a complete idea of the actual number of people that could be negatively affected by our models' results and made us commit to RF as the top choice.

We can make assumptions for some of the models where we could estimate how a change in initial conditions affects overall output. In the case of our model of choice, which is RF, we can affect how it predicts by running a more robust model containing more trees; similarly, it is also beneficial to increase the minimal number of data points before further splitting up a node (referring to the min_n parameter). This could, in general, make for a more powerful model, which may improve accuracy to a point where it stands out as the best possible model without reasonable doubt.

A recommendation and rationale regarding which algorithm to use for detection of blue tarps. Which model runs faster (and well enough) so help can be sent quickly.

In some scenarios it may be better to choose a model that builds and performs quickly at the cost of it not producing "ideal" results, but rather results that are good enough to get the job done. If we were set with this task based on the nine models that we have built so far, we would start by eliminating the models that take the longest to train, which are RF, SVM Linear, SVM Polynomial, and SVM RBF.

The next criteria for elimination regarding this decision would be models that have very high false positive rates, because they will produce too many predicted blue tarp pixels that are actually not blue tarp pixels, resulting in more time wasted located people at these locations when they are not there. In this case that would be Log Reg with a false positive rate of 0.1015 and Tuned Log Reg with a false positive rate of 0.903. This leads us to evaluate the results from three models: LDA, QDA, and KNN. The decisions made at this point can be a little less clear cut, as we take into account metrics from both the cross validation and the holdout.

Looking at the cross-validation metrics (Table 2) we see that LDA underperforms compared to the other two models on all metrics we collected, accuracy, J index, and Kappa, and AUC. QDA has the best AUC and KNN performs the best on accuracy, J index, and Kappa. Moving onto the holdout metrics (Table 5) we see that QDA falls short for all metrics, accuracy, sensitivity, specificity, J index, and AUC. LDA performs best for sensitivity and J index and KNN performs best for accuracy and specificity. Given that LDA does not perform well on cross validation and that QDA falls short on the holdout, it seems that KNN may be the best choice in this scenario.

A discussion of the relevance of the metrics calculated in the tables to this application context.

The choice of classifier varies based on different criteria. There is no one size fits all approach. Imbalance in both training and holdout datasets and threshold selection could lead to this result. From the holdout metrics we can also get a good idea of the specificity – sensitivity trade off. Specificity refers to the number of NotBlueTarp that are correctly identified by the model. Sensitivity refers to the number of BlueTarp that are correctly identified by the model. We see that RF has the best specificity and sensitivity combination at 0.96 and 0.97, respectively. KNN has the second best specificity and sensitivity combination at 0.92 and 0.98, respectively. We also notice that 4 models (Log Reg and tuned Log Reg, SVM Linear and SVM Polynomial) have a sensitivity of 1 which means there are no false negatives. This also gives us a side effect of lower specificity. This is also captured at the confusion matrix grid (Figure 16).

What additional recommended actions can be taken to improve results?

The main recommendations to potentially improve results will vary by model and by training parameters. In the case of LDA and QDA, both models could see a noticeable advantage by applying model tuning techniques on it, such as L1 and L2 regularization. This is a way in which the high False Positive rate could be reduced, and not return such a gap between their performance and the tuned models' performance in cross-validation and holdout.
Another important consideration is that we are training our data on a highly imbalanced data set. This does not mean we cannot draw significant conclusions from the given variables, but it affects the metrics by which we can judge our variables. If the data provided were less imbalanced and included a higher ratio of Blue Tarp observations, we would take the accuracy metric into higher consideration, or construct our threshold selection based on accuracy, which has the potential to improve results.
The above consideration goes hand in hand with the use of J-Index. One way in which we could also optimize our models is by doing a general threshold scan to find the impact on performance.

What is your level of confidence in the results?

We are reasonably confident in our results for the RF Model, as it generally has good metrics for cross validation and the holdout dataset, which we go into more detail in the last section of this report. Also, because it has a low number of BlueTarp pixels not found, which we believe will not be an issue as these pixels are likely the borders of the blue tarps and the middle parts of the blue tarps will be identified. It does produce some false positives, but it is significantly less than most of the other models and we would rather ensure that all of the blue tarps are identified, so everyone can receive help.

Were there multiple adequately performing methods, or just one clear best method?

Of our nine models there are two that appear to be adequately performing models, depending on a few factors. As we discussed earlier KNN is a good model if you need to build and run the model on a limited time frame, but RF is a good choice if you are not under a tight time restriction. If we could determine roughly how many (a proportion) pixels make up the borders of the blue tarps, we may be able to better understand the allowable amount of true blue tarp pixels could falsely be classified as not blue tarps. This would be helpful because we would be able to have a higher confidence in the model results despite a decent number of false negatives. We see in the confusion matrices (Figure 16), that KNN has about twice the number of false negatives as RF. Meanwhile RF has about double the number of false positives that KNN has. The choice between these models results in KNN having a higher chance of missing blue tarps, due to missing the blue tarp pixels. This could potentially mean leaving people without help, whereas RF has a higher chance of utilizing more resources than necessary to ensure all blue tarps/people are found. The choice of model between these two is based on what kind of decisions you make; we stand by utilizing RF as it would better ensure we help everyone after the earthquake.

What other models can be explored that might improve results ?

We could consider boosting models to improve the results. In Boosting, the trees are grown successively, using a slow learner approach. The popular boosting models such as Adaboost or Gradient boosting could be tried to combine weak learners into a single strong learner. Adaboost (aka adaptable boosting) can combine several weak classifiers and has a knack for allocating increased weights to underrepresented classes (see footnote 1 for more details on Adaboost). Also, boosting algorithms can perform better on imbalanced datasets as we are increasing the importance of misclassified instances in boosting.

What is it about this data formulation that allows us to address it with predictive modeling tools?

We first consider how this data formulation enables us to address this problem using predictive modeling tools. From the beginning, we know that this is a classification problem. We are interested in solving this problem by predicting just for the small subset of the response variable within the enormous size of the dataset. Also, we notice from the violin plot (Figure 2), the pixel colors can show some patterns or correlation between the landscape of Haiti and the differently colored pixels found in our data sets. These aspects make it so we can apply our three classification models.

The provided training data is also optimal for modeling in the aspect that there are a lot of observations in comparison to the number of predictors. One thing to keep in mind is the imbalance that has happened in both datasets may influence the way we create our models and the conclusions we arrive at.

How effective do you think your work here could actually be in terms of helping to save human life?

After careful consideration in training our models and determining which one works the best, we are highly confident that the RF model we chose will help effectively save human lives. We know that our RF model has a high degree of correctly predicting those places where help is needed. It has a high true positive rate/sensitivity of 0.9626, as we can see in the confusion matrices in Figure 16 and the holdout metrics in Table 4. Doing so will ensure that the number of people who may unescapably be left unattended is as low as possible. It also significantly decreases the number of false positives, falsely identified blue tarps, which reduces the waste of resources. This ensures that time is not wasted checking locations that have a lower probability of not having blue tarps/people.

Every day that passes matters, and every location is just as important as all the others. This is why we argue our model will benefit the highest number of people and save the lives of those in these times of distress. This is one step into providing help to survivors, given that there are other logistics goals and questions to solve, such as the path of helicopters to follow and the most optimal way of distributing resources to the people in need.

Determination and justification of which algorithm works best

Based on the combined results from the cross-validation metrics and the holdout performance of all nine models, we have determined that RF is the best model to select for the prediction of blue tarp pixels. This conclusion arises not only from the fact that RF is one of the top-performing models on the holdout data set metrics but also from its similarity in presenting solid results during the cross-validation process. We can confirm this statement since RF displayed the second-highest accuracy, J-Index, and kappa while having the highest ROC-AUC value. For the holdout data, we see that it was only slightly outperformed by SVM Polynomial on three metrics, but particularly returns a better specificity. This is significant because RF ensures we spend as little resources as possible in places where they are not required.

One of the key reasons for favoring RF over KNN is the potential impact on human lives. When we examine the confusion matrix data, we find that KNN has twice the number of observations classified as NotBlueTarp, when in fact these places do have pixels corresponding to BlueTarp, indicating that there are people there that require aid. While KNN may have a better False Positive Rate, the risk of sending extra help to places where no help may be needed is a risk we can afford to take, rather than risking more human lives and disregarding the urgent call for help.

Foot note 1: Citation

Adaboost and class imbalance	https://www.sciencedirect.com/topics/engineering/adaboost	Last access Date: 08/04/2024