Disaster relief project - Part 1

Taeyoon Kim   Mauricio Mathey   Max Pearton

Introduction

In response to the devastating earthquake that struck Haiti in 2010, rescue efforts faced significant challenges in locating displaced persons residing in makeshift shelters amidst the widespread destruction. With communication networks and infrastructure severely disrupted, aid workers struggled to pinpoint the locations of those in need amidst the vast expanse of affected areas. The presence of blue tarps, used by displaced individuals to construct temporary shelters, served as a vital indicator for locating those in need. The problem now wasn't so much where to look as it was how fast these unique locations could be pinpointed amidst a wide array of geographical features and other man-made structures that are present in high resolution geo-referenced imagery taken from an aircraft. The sheer quantity of aerial footage collected on a daily basis deemed it an impossible task for the human eye. Aid workers would be unable to search through the thousands of images in time to correctly identify the tarps and subsequently coordinate rescue operations. To address this critical issue, data mining techniques emerged as a promising solution. Automated algorithms that are capable of efficiently and accurately identifying these shelters would be deployed on the collected data, enabling rescue workers to deliver essential aid in a timely manner. This project aims to evaluate various classification algorithms, along with ensemble methods and support vector machines to determine the most effective approach for identifying displaced persons within
the imagery data, with the ultimate goal of facilitating timely assistance to those affected by the calamity.

Our endeavor began with extensive exploratory data analyses to gain a better understanding of the underlying characteristics of the data collected by the team from the Rochester Institute of Technology. The discoveries made in this process informed our methodology in terms of how we manipulated the data, which performance metric best served our interests, and how we justify recommending a particular model given the task at hand.

Data

We are working with data collected from aerial images. From Table 1, we can see that the information that we have is class, red, green, blue. In our case, as what we are trying to predict is if it's vegetation or a makeshift, Class will be our dependent variable. Red, Green, and Blue are our independent variables.

From Table 1 we can see that there are 5 categories in the dataset. We are interested in predicting the "Blue Tarp" category. A first issue that we can observe is that it is an unbalanced set. Blue tarps represent 3% of the data only. This means that it would be very easy for any method to ignore this class and it could achieve a 97% accuracy. This suggests three initial strategies. The first one is that when splitting the data we will need to take a stratified sample. This will ensure that the distribution of both training and testing sets resemble the original one. The second strategy is reducing the number of classes from 5 to only 2. This will help us focus on identifying blue tarps and not other types of objects. Finally, is the metric we will optimize for. If we choose accuracy, it will be very easy for us to achieve an accuracy of 97%. But this would be at the expense of not correctly identifying any shelters. On the other hand if we are overly cautious and assign a greater amount of shelters, we could potentially dilute rescue efforts. A metric that balances the sensitivity and the specificity is the J-Index. This is the metric we will be using.

Table 1: Number of observations per class

Class	count	percentage
Blue Tarp	2022	3
Rooftop	9903	16
Soil	20566	33
Various Non-Tarp	4744	8
Vegetation	26006	41

Let's explore how does the presence of red, blue, or green determine each class. As previously mentioned, we will reclassify the data. Blue Tarp will be coded as Shelter and all the other categories as Other. This will allow us to focus on identifying only the shelters which is the problem we are trying to solve.

From figure 1 we can see that for red the shelters present most of the values between ~120 and 225 which allow to differentiate them from other objects. For blue we can see that shelters don't present low values of blue but they do present values above 200 which could help differentiate. Finally for greens we observe a similar behavior where shelter vs other exhibit differentiated behaviors.

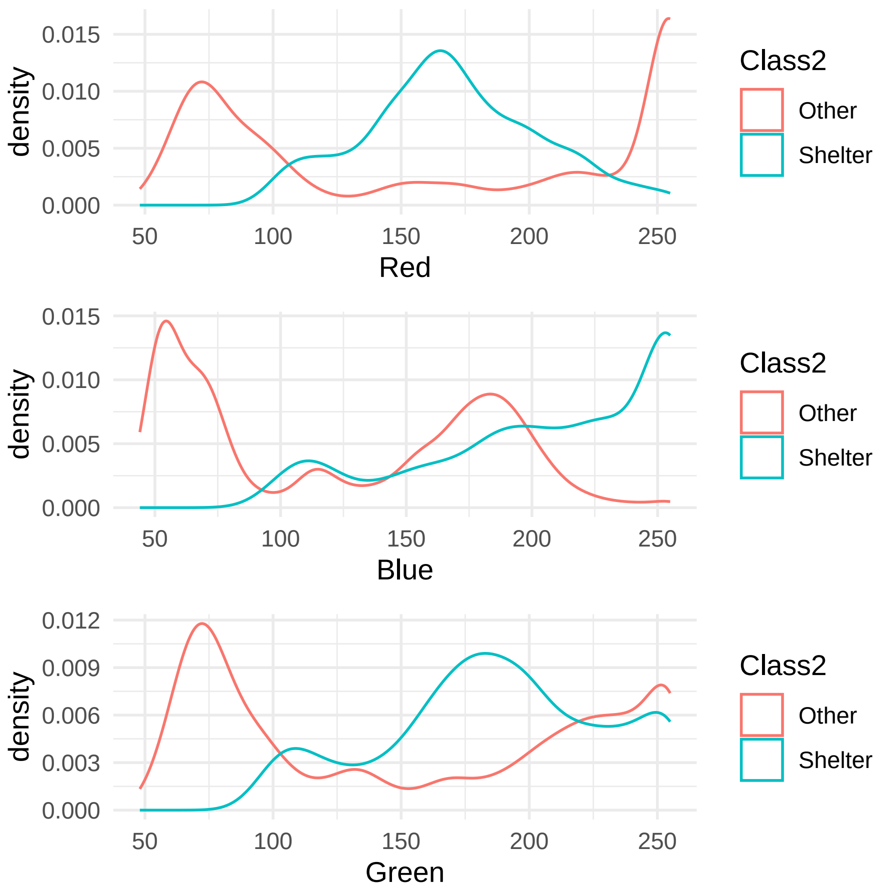


Figure 1: Distribution of color by class

Let's explore if there are any interactions between colors that could help us identify shelters better.

From figure 2 we can see that there seems to be an approximately linear separations between blue and red for shelter vs non shelter. This suggests that methods with linear boundaries might perform well.

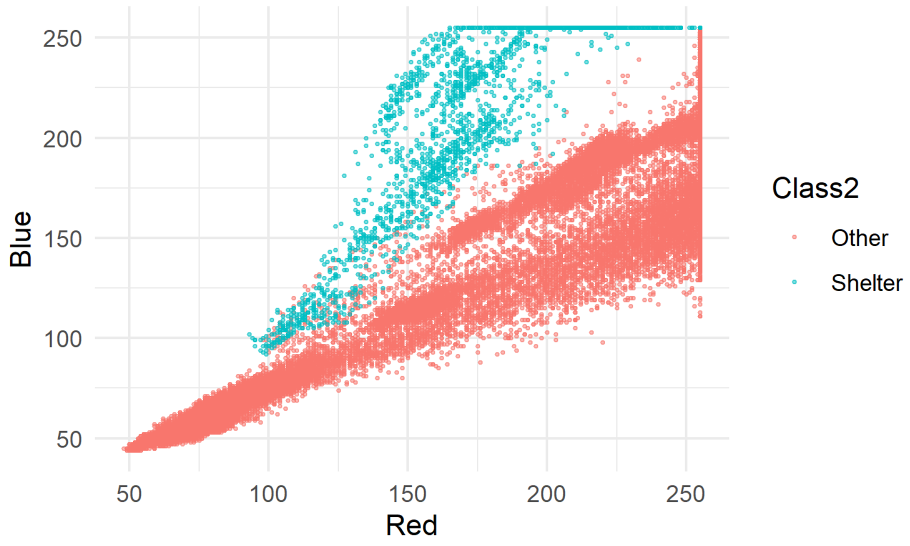


Figure 2: Blue vs Red separation

From figure 3 we can see that while there is a separation between green and red, it seems to be nonlinear. This because there is an overlaping area where you would need a U shape function to be able to separe them. And even with that we might not be able to do so.

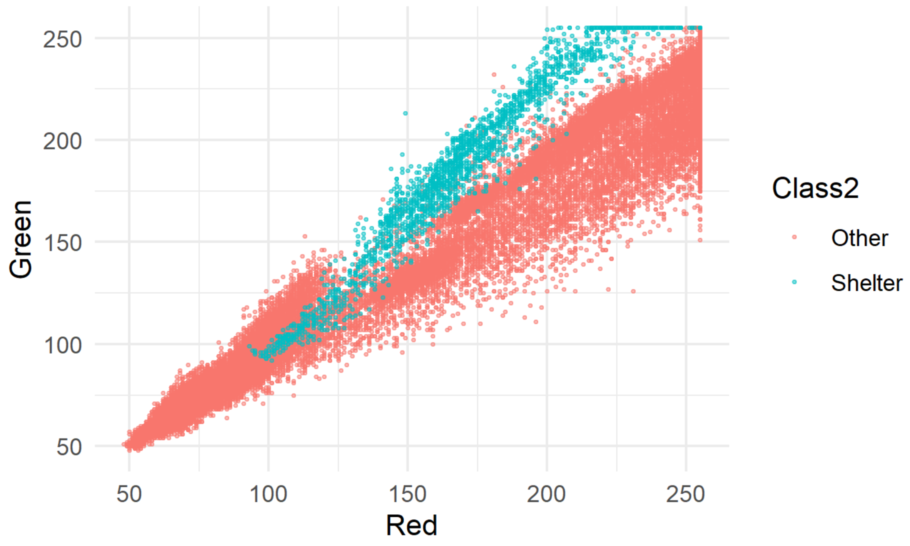


Figure 3: Green vs Red separation

From figure 4 we can see that there seems to be a separation between green and blue. While not exactly linear, it seems it could be approximated as one.

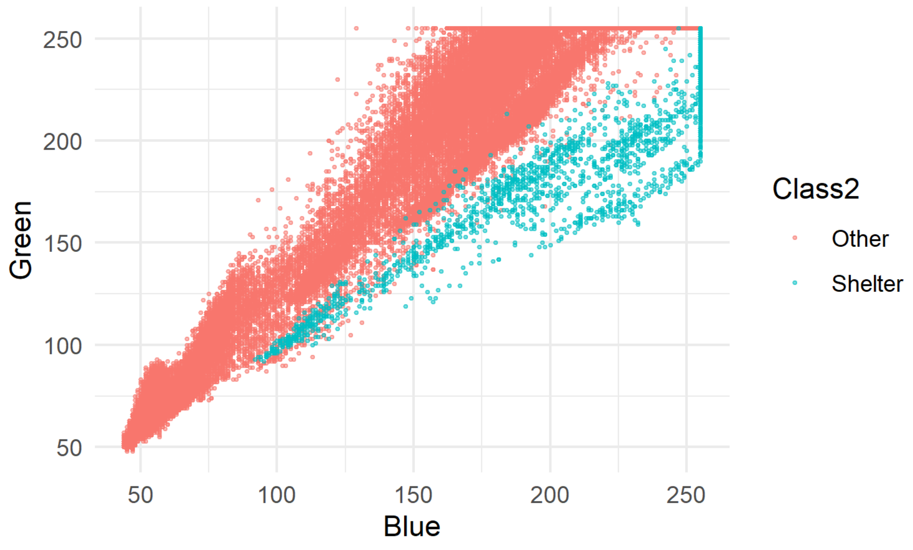


Figure 4: Green vs Blue separation

In comparison to Part 1 of this project, Part 2 includes a far greater quantity of holdout data. Whereas the training dataset included approximately 63,000 entries, the eight text files that comprise the holdout dataset consists of a bit over 2,000,000 entries. The newly presented data includes several additional levels of detail including ID, X & Y coordinates on a map, as well as longitude and latitude readings. Most notably, the holdout dataset does not explicitly label the three columns assigned to measure each respective color (Red, Green, and Blue).

Description of methodology

We will test 7 different models: logistic regression, linear discriminant analysis, quadratic discriminant analysis, K-nearest neighbor, penalized logistic regression, boosted trees, and support vector machine. We will use 10 fold cross-validation to assess the performance of each of the models and decide on hyperparameter tuning. we will tune the hyperparameters using ROC AUC. To determine the optima threshold we will use J-Index because it is an unbalanced dataset. By using J-Index instead of other metric we make sure there is a balance between sensitivity and specificity.

A seed with a value of 1 will be used across the models to ensure reproducibility. We will have a training set and a holdout set. We will get a stratified sample by the class variable. The split will have an 80% - 20% split for training and testing.

Logistic regression

The first model we are going to test is a logistic regression. From table 3 we can see that the AUC ROC is very high.

Table 2: Performance of logistic regression

.metric	.estimator	mean	n	std_err	.config
roc_auc	binary	0.9985003	10	0.0001734	Preprocessor1_Model1

The next step is figuring out the optimal threshold. As previously mentioned, we want to maximize j-index to prevent false positives.

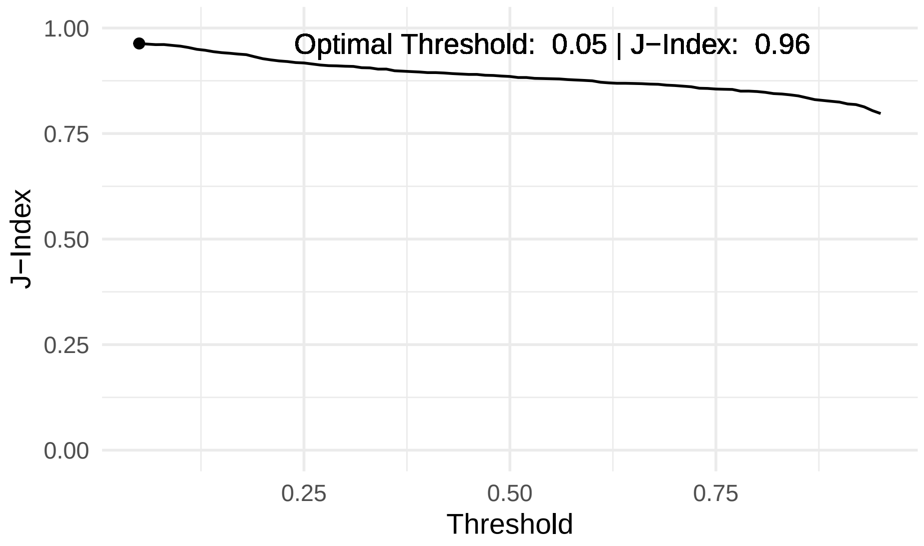


Figure 5: Optimal threshold for logistic regression

From figure 5 we can see that by reducing the threshold, the J-Index increases to 0.96.

Linear discriminant analysis

Next we are going to move to a linear discriminant analysis (LDA) model.

Before optimizing the threshold, we can see in table 4 a good performance. By finding the optimal threshold we find that LDA can achieve a J-Index of 0.86.

Table 3: Performance of LDA

.metric	.estimator	mean	n	std_err	.config
roc_auc	binary	0.9889001	10	0.0004169	Preprocessor1_Model1

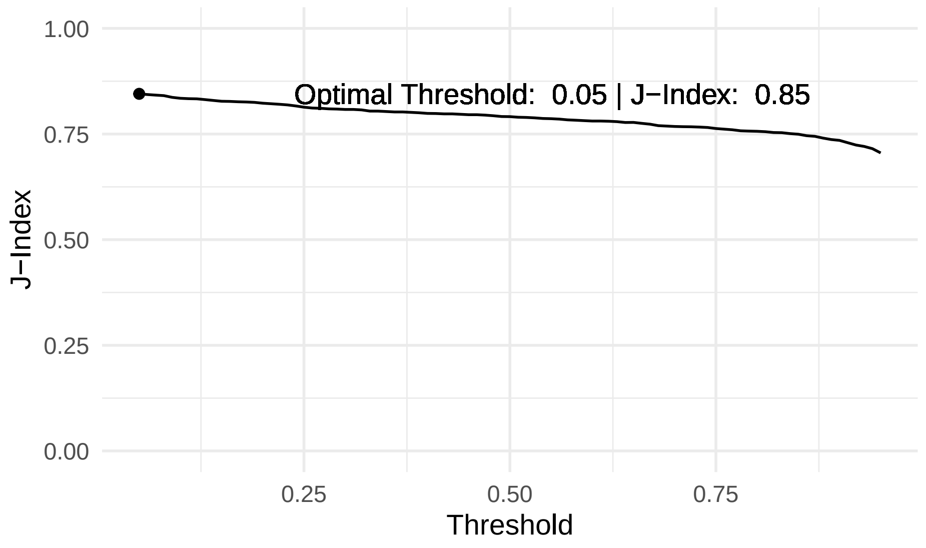


Figure 6: Optimal threshold for LDA

Quadratic discriminant analysis

Next we are going to move to a quadratic discriminant analysis (QDA) model.

Before optimizing the threshold, we can see in table 5 a good performance. By finding the optimal threshold we find that LDA can achieve a J-Index of 0.97.

Table 4: Performance of QDA

.metric	.estimator	mean	n	std_err	.config
roc_auc	binary	0.9982273	10	0.0003391	Preprocessor1_Model1

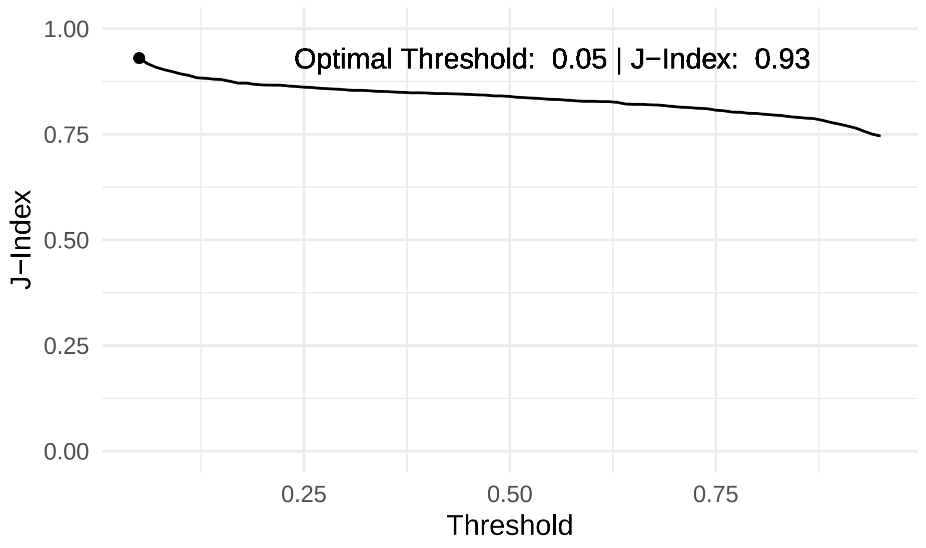


Figure 7: Optimal threshold for QDA

K-nearest-neighbor

Next we will explore K-nearest-neighbors. For KNN we will define the number of neighbors K as a tuneable parameter to understand which value would yield the highest J-Index. We evaluated between 2 and 20 neighbors.

From table 6, we can see that if we aim to maximize the ROC AUC, the optimal number of neighbors is 20. Other viable alternatives could be 16, 19, 15, or 18. All have very similar ROC AUC values.

Table 5: Performance based on number of neighbors

neighbors	.metric	.estimator	mean	n	std_err	.config
20	roc_auc	binary	0.9940224	10	0.0012195	Preprocessor1_Model19
16	roc_auc	binary	0.9940224	10	0.0012195	Preprocessor1_Model15
19	roc_auc	binary	0.9940220	10	0.0012196	Preprocessor1_Model18
15	roc_auc	binary	0.9940219	10	0.0012197	Preprocessor1_Model14
18	roc_auc	binary	0.9940218	10	0.0012195	Preprocessor1_Model17

We can see that the optimal threshold for KNN is 0.01, achieving a J-Index of 0.98.

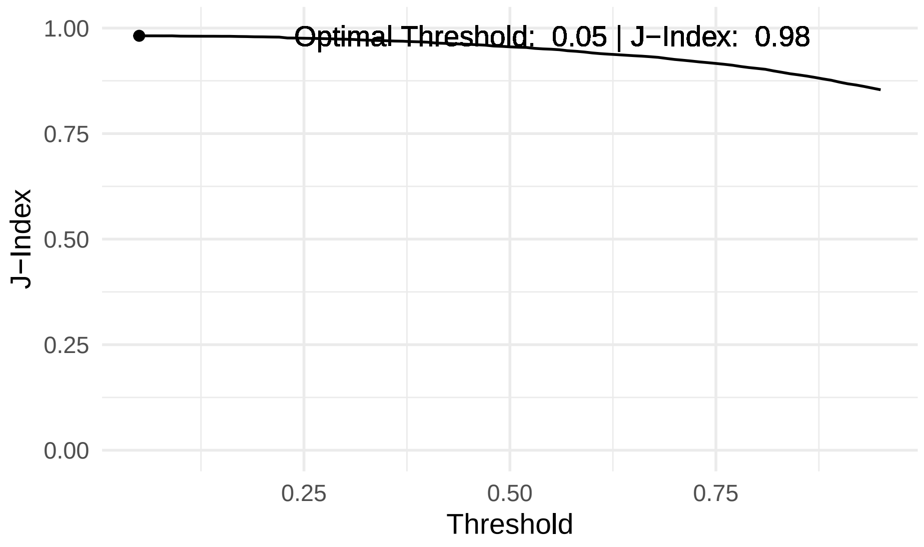


Figure 8: Optimal threshold for KNN

Penalized logistic regression

Finally, let's explore the results from penalized logistic regression. In this case we have 2 tuning parameters. For tuning we used a random grid. The first one is penalty, this determines the amount of penalty that we will introduce in the model. The second one is mixture. Which determines the proportion of Lasso vs Ridge penalization that we will apply. 0 means only Ridge penalization is being used while 1 that only Lasso. Anything in between is a mixture of both, with lower values having a higher proportion of Ridge and higher ones a higher proportion of Lasso.

Table 6: Penalty and mixture tuning

penalty	mixture	.metric	.estimator	mean	n	std_err	.config
0.00000000	0.2121425	roc_auc	binary	0.9985731	10	0.00011227	Preprocessor1_Model05
0.00000042	0.1255551	roc_auc	binary	0.9985695	10	0.00009295	Preprocessor1_Model03
0.00000000	0.1862176	roc_auc	binary	0.9985683	10	0.00010869	Preprocessor1_Model04
0.00000000	0.2672207	roc_auc	binary	0.9985624	10	0.00012085	Preprocessor1_Model06
0.00000756	0.1079436	roc_auc	binary	0.9985561	10	0.00008963	Preprocessor1_Model02

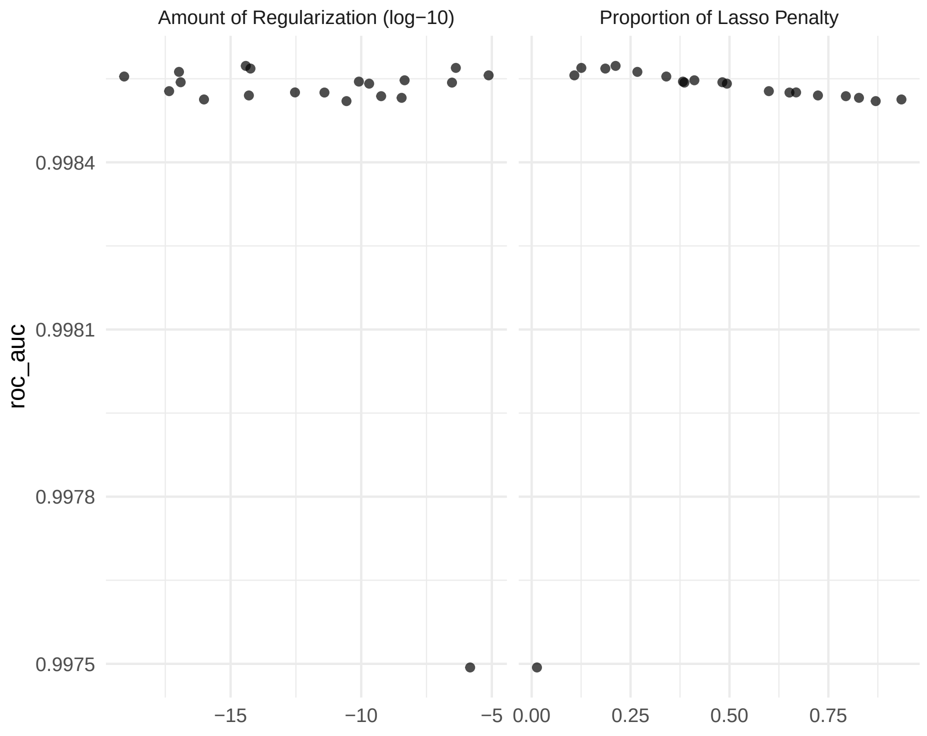


Figure 9: Model performance based on regulazation and proportion of Lasso penalty

From both table 7 and the figure 9 we can see that the amount of penalty applied is very small and that it is mostly Lasso.

In figure 10, we can see that just as with logistic regression, the optimal threshold is 0.04. which would yield a J-Index of 0.97.

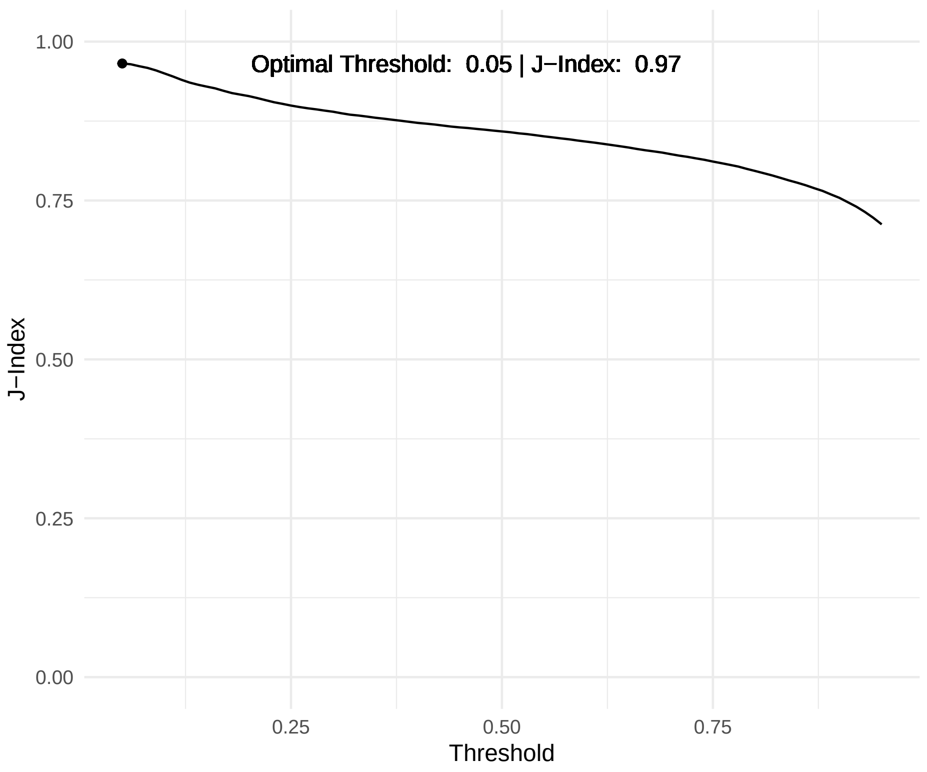


Figure 10: Optimal threshold for penalized logistic regression

Boosted trees

Moving onto boosted trees. We have 2 parameters to tune, tree depth and learning rate. To figure out the optimal parameters, we used random grid search.

We can see that the optimal tree depth is 9 with a learning rate of 0.0258.

Table 7: Tree depth and learn rate tuning

tree_depth	learn_rate	.metric	.estimator	mean	n	std_err	.config
9	0.0258432	roc_auc	binary	0.9993506	10	0.0003318	Preprocessor1_Model01
7	0.0000733	roc_auc	binary	0.9990459	10	0.0002914	Preprocessor1_Model03
6	0.0014066	roc_auc	binary	0.9989807	10	0.0002473	Preprocessor1_Model17
5	0.0027949	roc_auc	binary	0.9989106	10	0.0004386	Preprocessor1_Model15
14	0.0067177	roc_auc	binary	0.9987731	10	0.0006533	Preprocessor1_Model09

In terms of optimal threshold, we see that it is at 0.04 with a J-Index of 0.36. We know this value is not correct as show in the following graph where we can see that for a threshold of 0.04 it is 0.9922. We tried debugging the error but could not find the root cause.

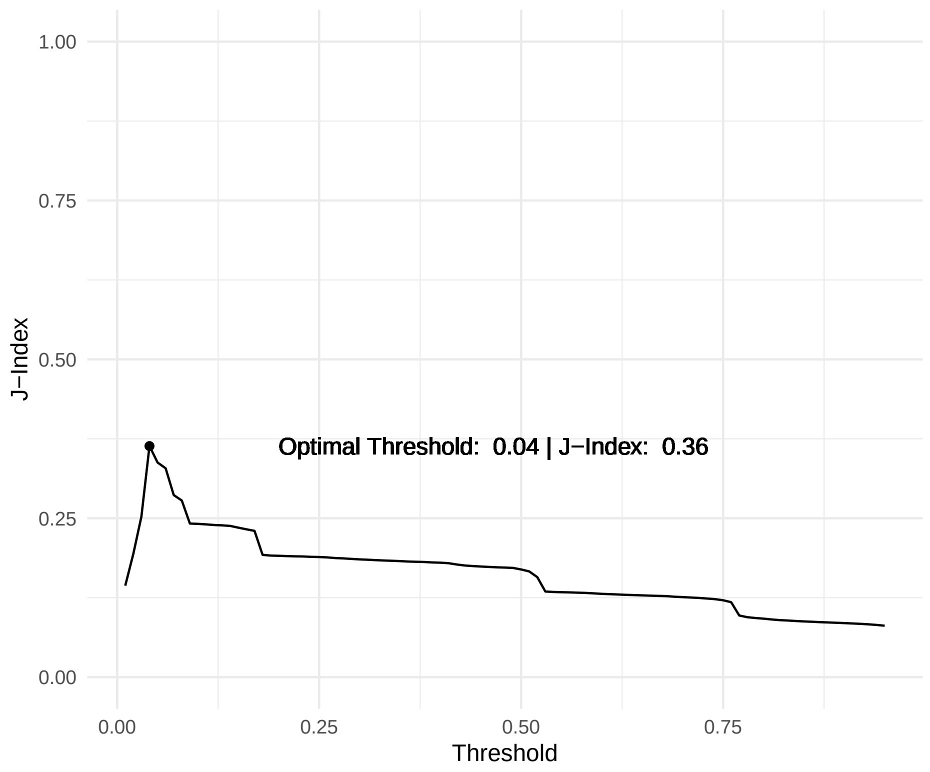


Figure 11: Optimal threshold for boosted trees

```
## # A tibble: 4 x 4
##   Model   .metric     .estimator .estimate
##   <chr>   <chr>       <chr>          <dbl>
## 1 Boosted j_index     binary         0.992
## 2 Boosted accuracy    binary         0.992
## 3 Boosted sensitivity binary         0.992
## 4 Boosted specificity binary         1
```

SVM

Moving onto SVM, we are going to test 3 different kernels: linear, polynomial, and RBF. The tuning parameters for linear will be cost and margin. For polynomial will be cost, margin, and degree. Finally for RBF will be cost, margin, and RBF sigma. The tuning for all three kernels will be done using Bayesian tuning.

For the linear kernel we can see that with a cost of 0.014 and a margin of 0.198 we get an AUC ROC of 0.998.

Table 8: Tuning for linear kernel

cost	margin	.metric	.estimator	mean	n	std_err	.config	.iter
0.0140703	0.1977642	roc_auc	binary	0.9980728	10	0.0001642	Iter7	7
0.0141168	0.1469972	roc_auc	binary	0.9980727	10	0.0001652	Iter17	17
0.0148066	0.1954280	roc_auc	binary	0.9980715	10	0.0001671	Iter2	2
0.0143270	0.1825255	roc_auc	binary	0.9980715	10	0.0001658	Iter11	11

cost	margin	.metric	.estimator	mean	n	std_err	.config	.iter
0.0138520	0.1968109	roc_auc	binary	0.9980702	10	0.0001624	Iter10	10

For the polynomial kernel the optimal hyperparameters yield an ROC AUC of 0.999.

Table 9: Tuning for polynomial kernel

cost	degree	margin	.metric	.estimator	mean	n	std_err	.config	.iter
0.0921505	3	0.0298426	roc_auc	binary	0.9995986	10	7.61e-05	Iter20	20
0.0953232	3	0.1015786	roc_auc	binary	0.9995985	10	7.64e-05	Iter14	14
0.0937893	3	0.1106684	roc_auc	binary	0.9995982	10	7.68e-05	Iter24	24
0.0857325	3	0.1458598	roc_auc	binary	0.9995947	10	7.86e-05	Iter15	15
0.0370012	3	0.0242762	roc_auc	binary	0.9995937	10	6.80e-05	Iter5	5

Finally for the radial kernel we can achieve 0.999 as well of AUC.

Table 10: Tuning for radial kernel

cost	rbf_sigma	margin	.metric	.estimator	mean	n	std_err	.config	.iter
30.0540331	0.9899998	0.0656745	roc_auc	binary	0.9996986	10	2.55e-05	Iter21	21
15.2189239	0.9109387	0.0194579	roc_auc	binary	0.9996930	10	2.29e-05	Iter24	24
7.0208498	0.7634778	0.1723675	roc_auc	binary	0.9996615	10	2.56e-05	Iter22	22
16.7055809	0.4078784	0.1846589	roc_auc	binary	0.9996535	10	3.04e-05	Iter23	23
0.6517671	0.1550576	0.1386978	roc_auc	binary	0.9995630	10	3.30e-05	Iter25	25

In terms of the optimal threshold we can see that for the linear kernel it is 0.02, for the polynomial kernel it is 0.02, and for the radial one 0.02.

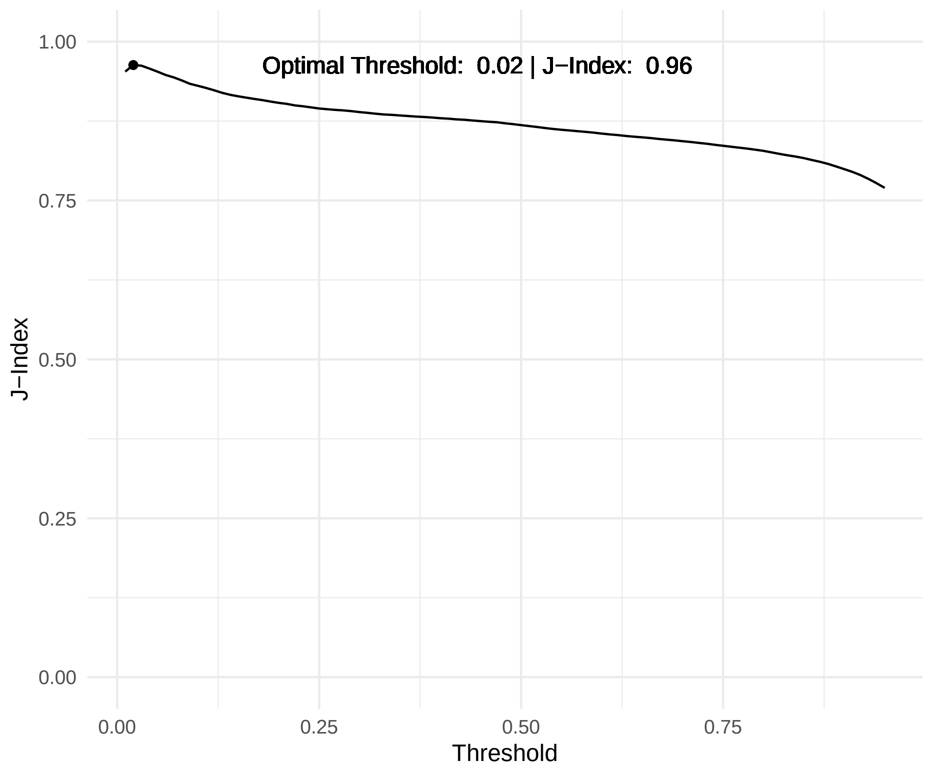


Figure 12: Optimal threshold for linear kernel

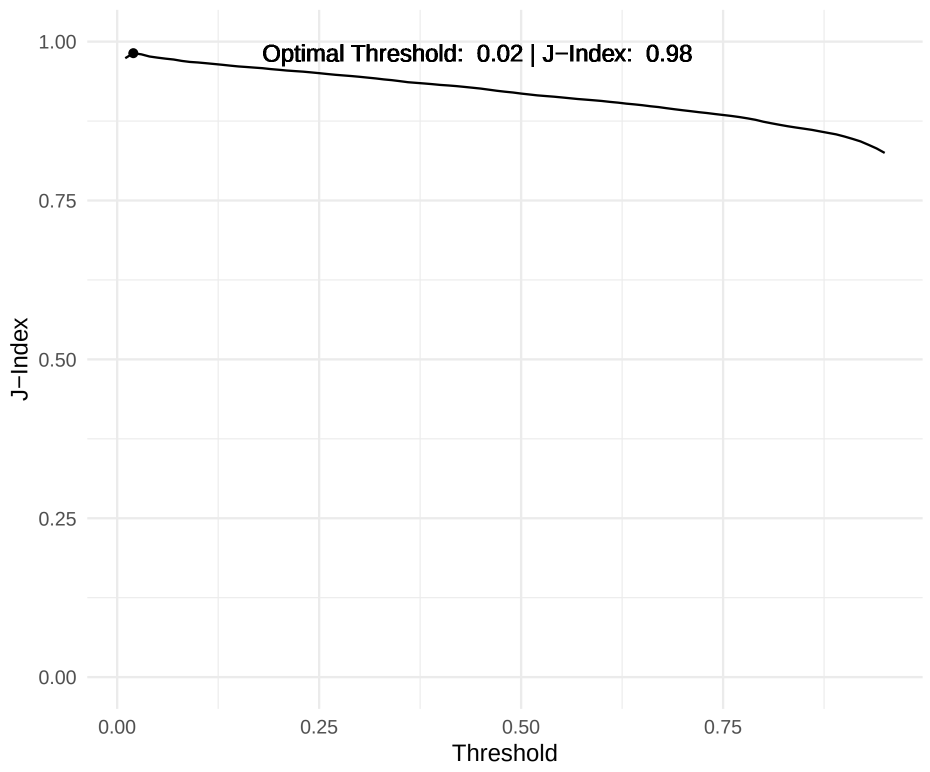


Figure 13: Optimal threshold for polynomial kernel

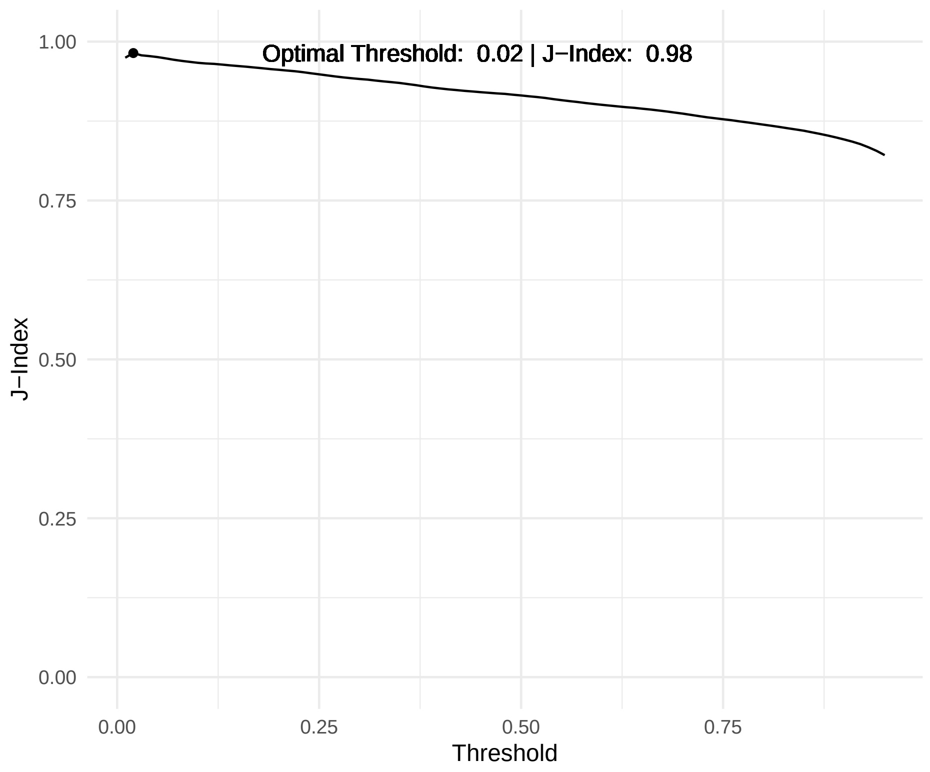


Figure 14: Optimal threshold for radial kernel

```
## Setting default kernel parameters
```

Results

Performance of models on training set

In figure 11 we can observe the ROC curves. As we can see, the only model that has a slightly worse ROC is LDA. The other models are very similar. Having said this, we will now move to evaluating the performance of the models on the holdout set to assess how they perform on unseen data.

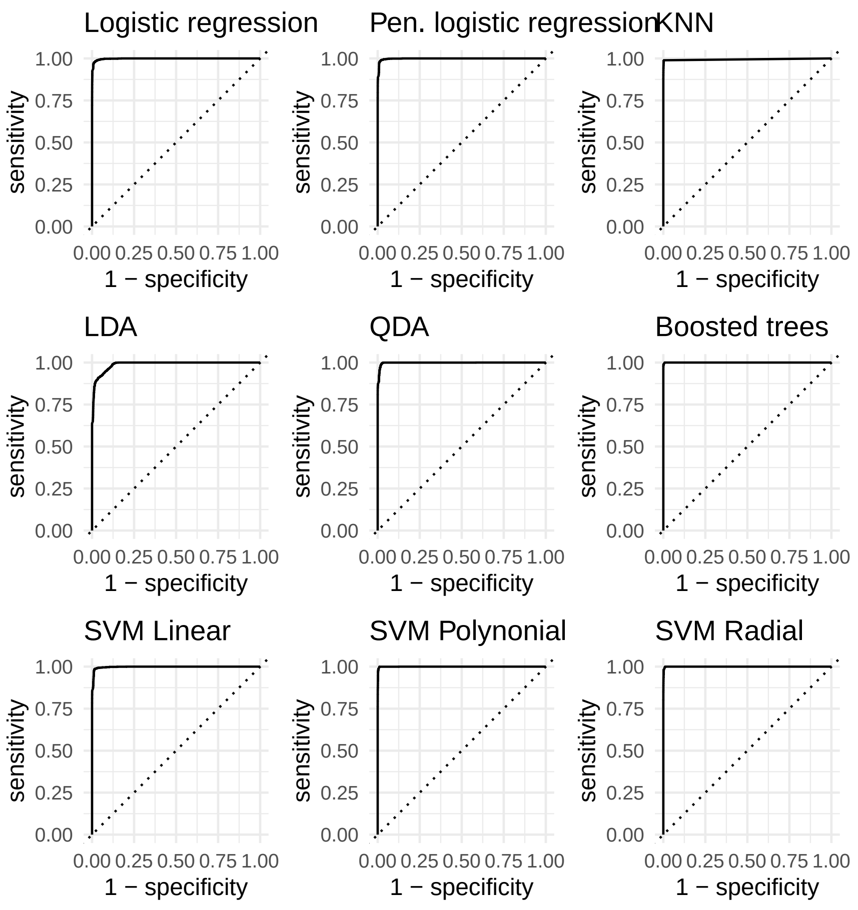


Figure 15: ROC for each model using training data

Performance on holdout set

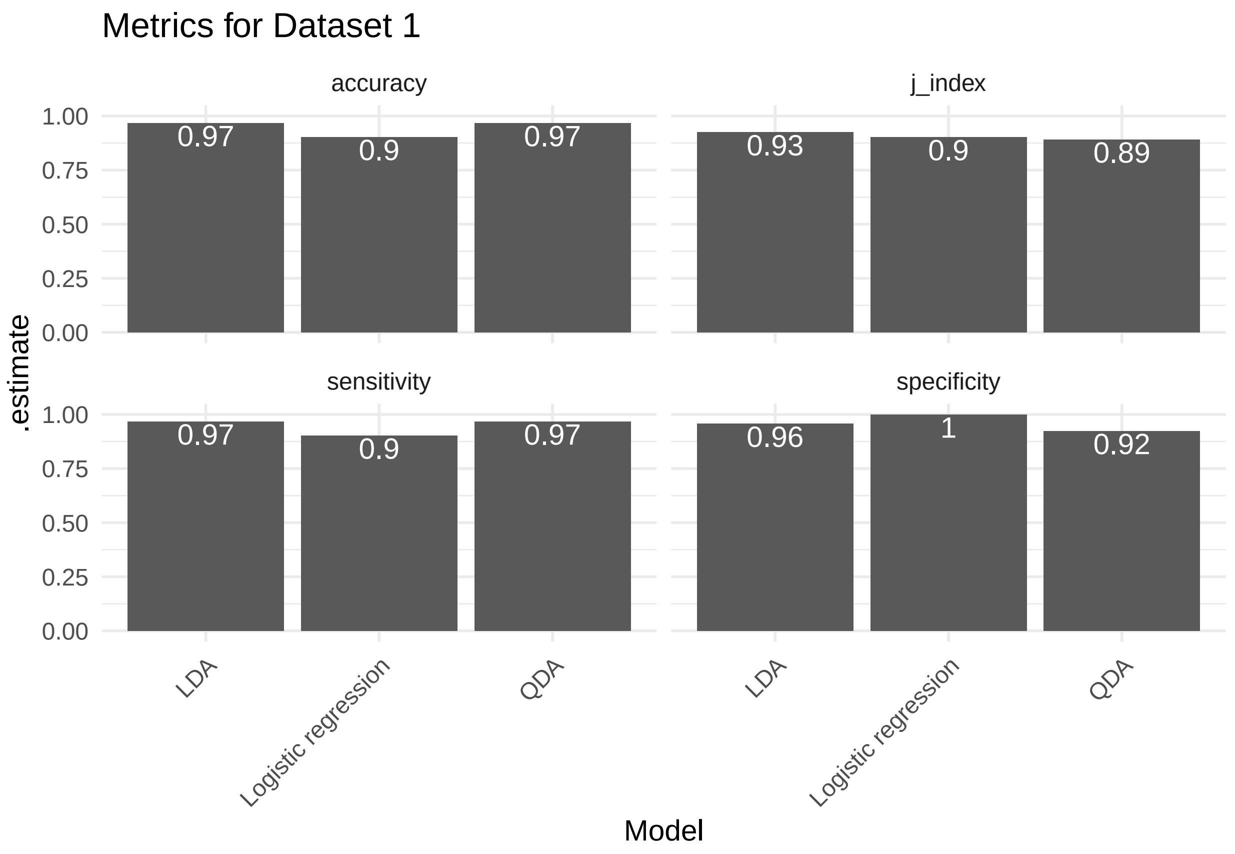
Determining the correct order of the columns

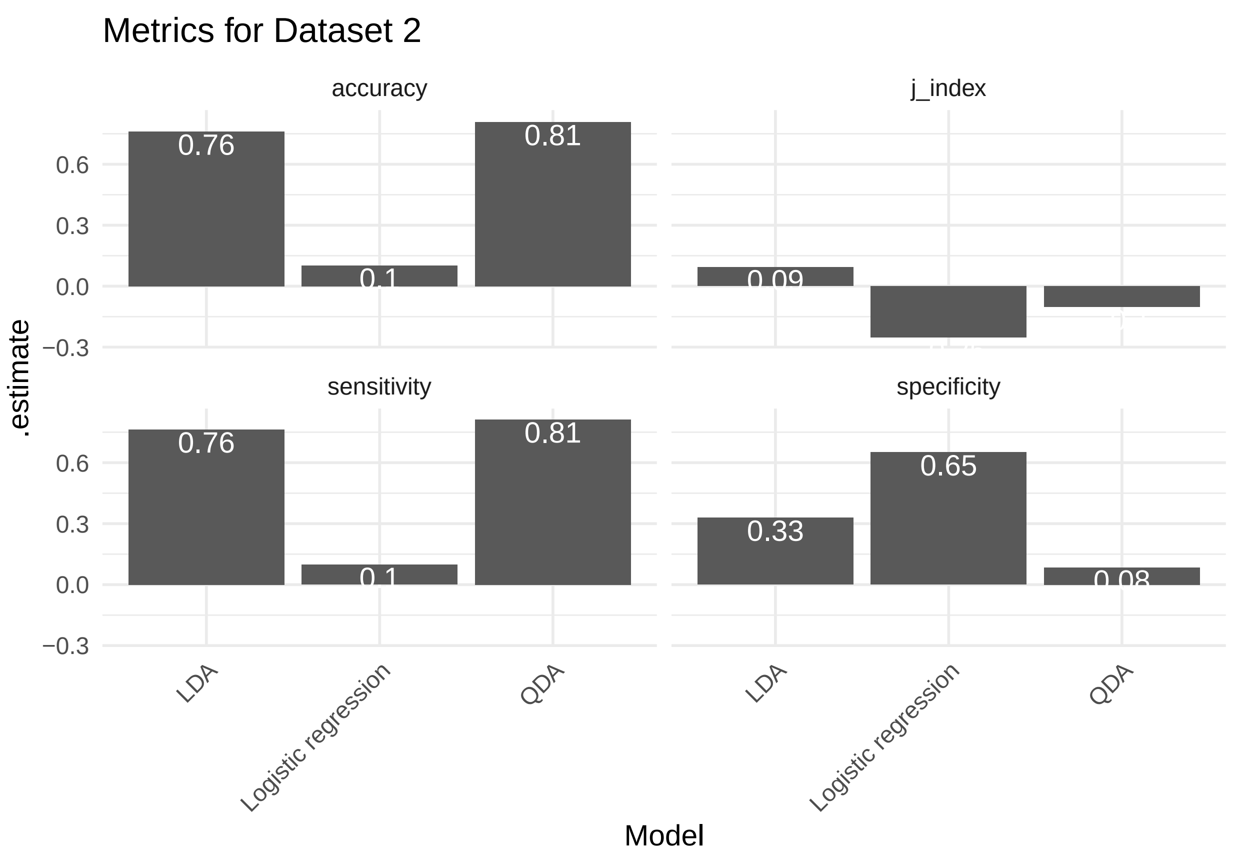
Now that each of the seven models were built and trained on the initial dataset, we needed to identify the correct validation set. The conundrum here was that we'd been given a holdout set without properly labeled columns corresponding to the red, green, and blue identifiers. As such, we decided to test each of the permutations (six in total) and analyze performance metrics to identify which matchup of B1 through B3 with red, green, and blue is correct. A loop was constructed to streamline the six iterations. Initially, we started off with five of the base metrics to distinguish the correct configuration. It didn't take long to realize that this was more than enough, as the respective metric graphs clearly indicated which color configuration was correct and those that weren't.

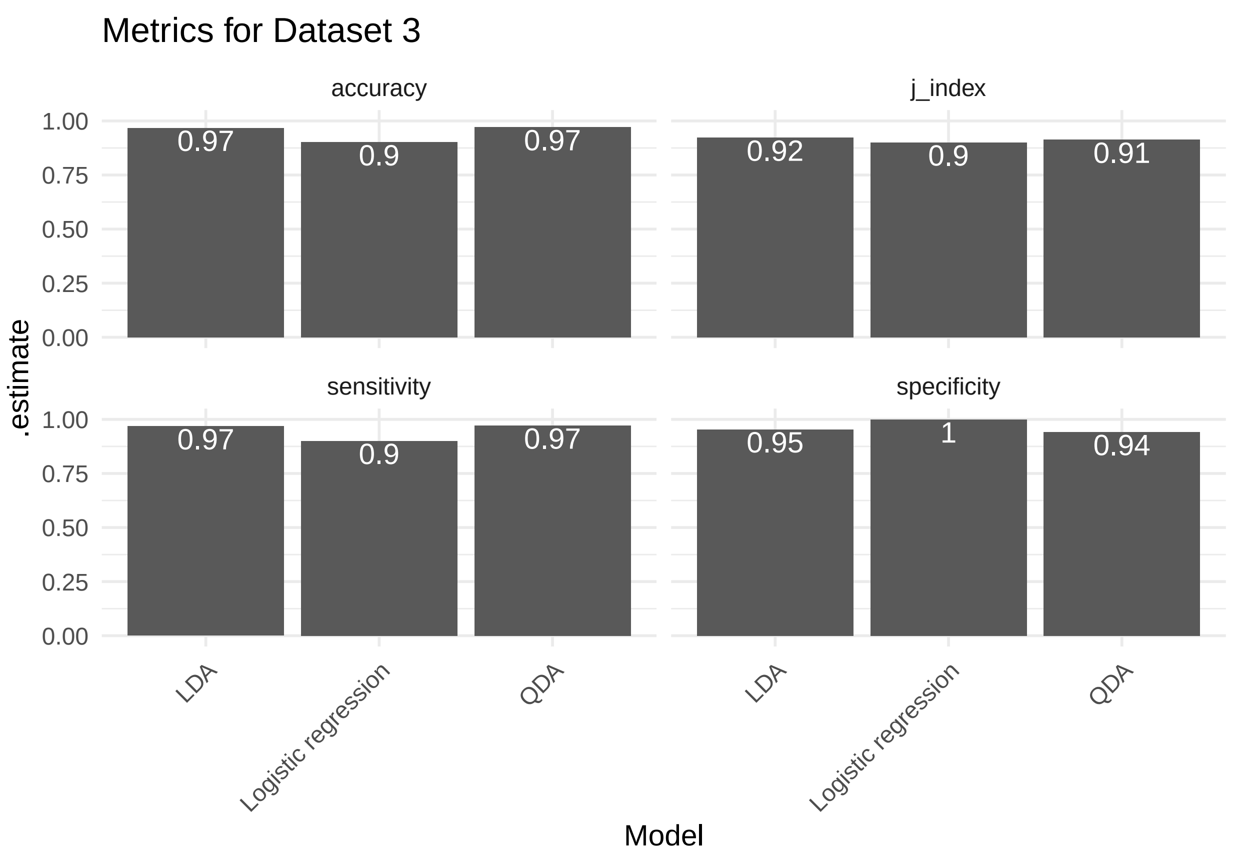
```
## [1] "Error: Insufficient columns after splitting."
```

Assessing which is the correct order

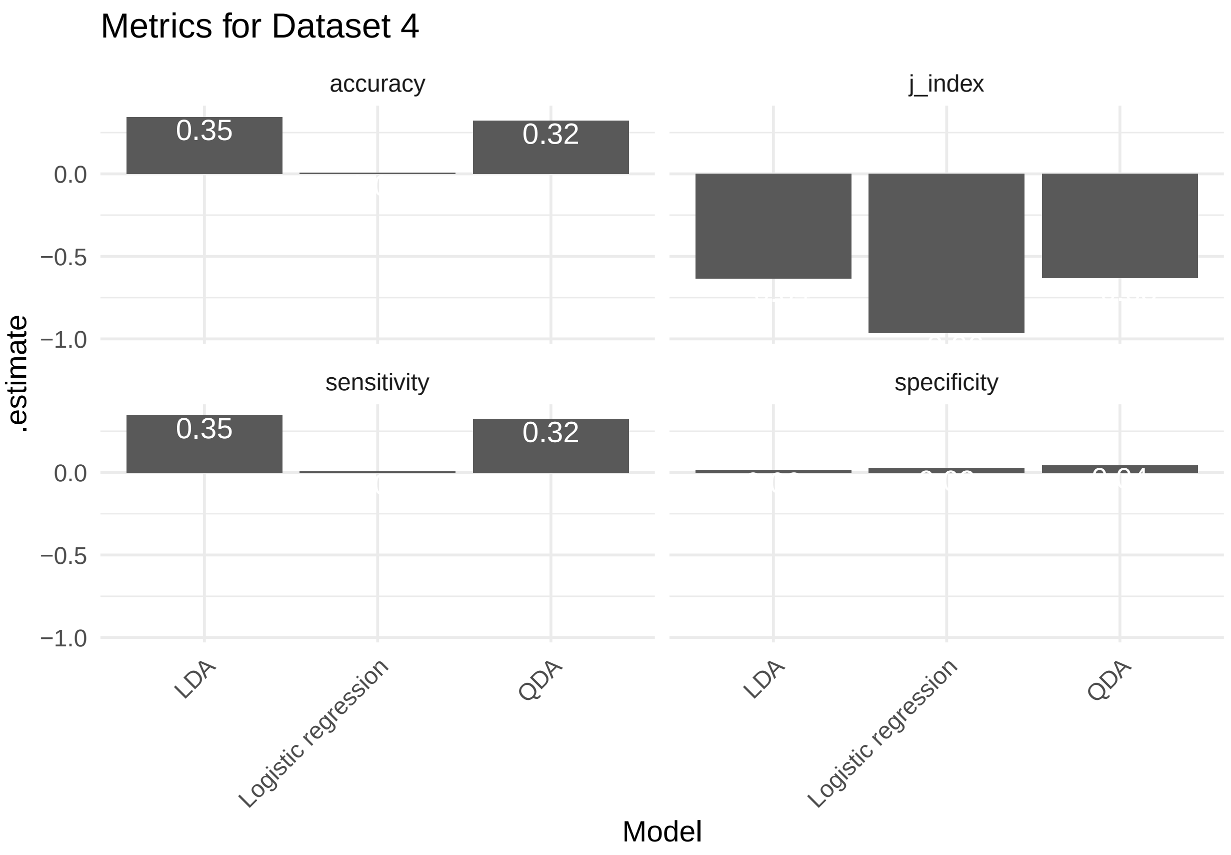
Of the six datasets tested, datasets 2, 4, 5, and 6 had conspicuous irregularities across the metrics measured and were quickly deemed inaccurate. On the other hand, datasets 1 and 3 seemed to indicate a high level of predictive performance from the respective models used. A very interesting observation we'd like to make here is that the primary difference between datasets 1 and 3 were that Red and Green were swapped between B1 and B2. The variable B3 remained consistently tied to the color Blue. Despite this, the two datasets displayed similar levels of performance, with dataset 1 holding an incremental advantage over dataset 3. What this may indicate is that the models are heavily indexing on the blue color in a given image file to classify whether the given location is a blue tarp or a non-blue tarp (shelter or other in our scenario). This makes sense as the entire premise of this project was that shelter locations would most often be indicated by the use of blue tarp rather than any other color.



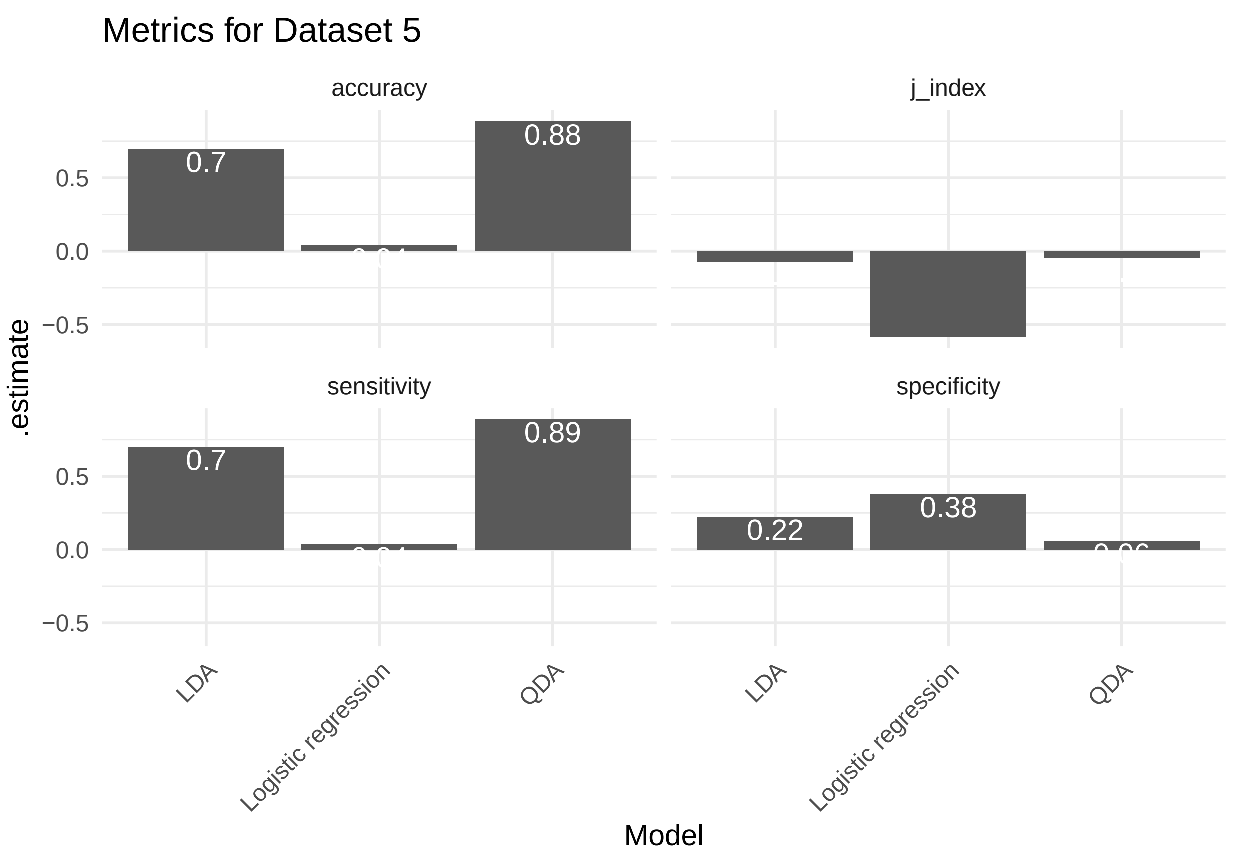

Metrics for Dataset 2
accuracy
j_index
sensitivity
specificity
.estimate
0.6
0.3
0.0
−0.3
0.76
0.1
0.81
0.09
0.76
0.1
0.81
0.33
0.65
0.08
LDA
Logistic regression
QDA
Model



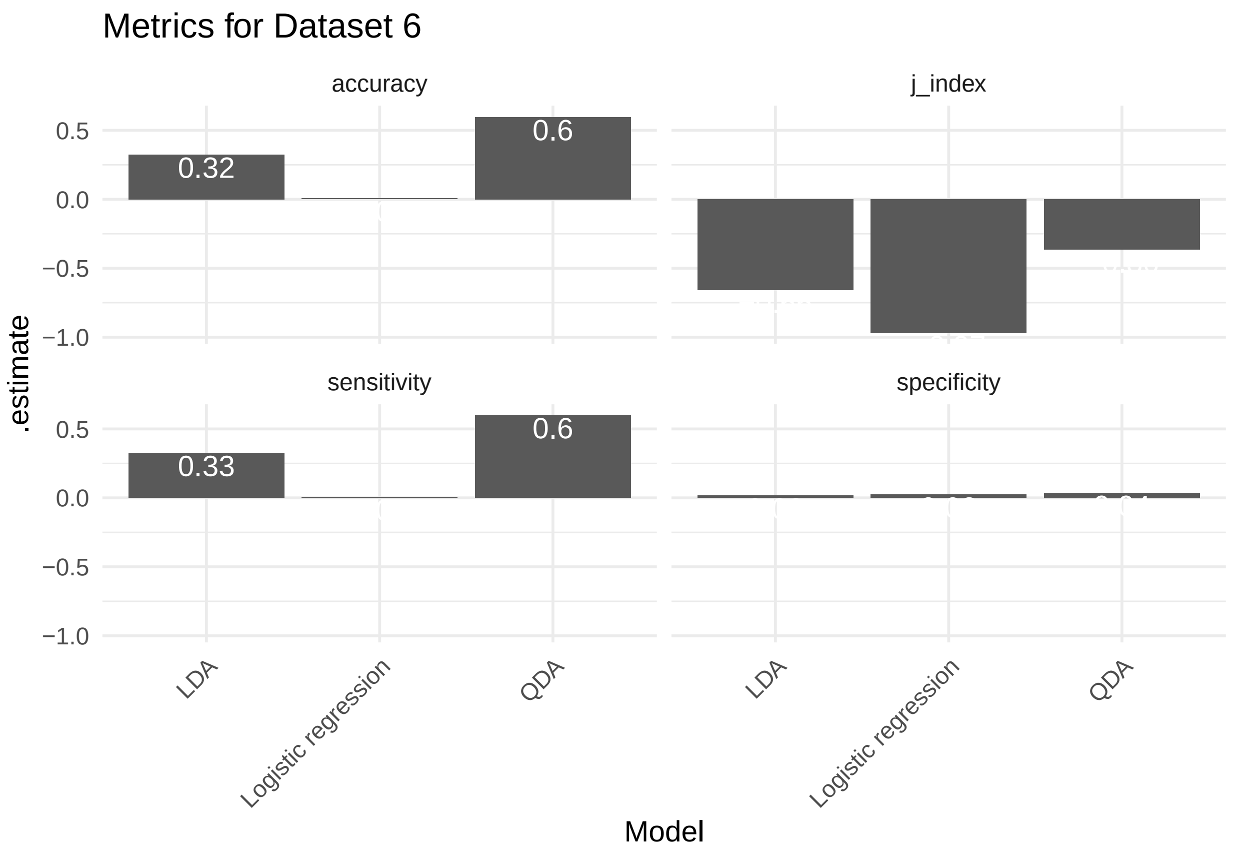
Metrics for Dataset 3
accuracy
j_index
sensitivity
specificity
.estimate
1.00
0.75
0.50
0.25
0.00
0.97
0.9
0.97
0.92
0.9
0.91
0.97
0.9
0.97
0.95
1
0.94
LDA
Logistic regression
QDA
Model



Metrics for Dataset 4
accuracy
j_index
sensitivity
specificity
0.35
0.32
0.35
0.32
0.0
−0.5
−1.0
.estimate
LDA
Logistic regression
QDA
Model



Metrics for Dataset 5
accuracy
j_index
sensitivity
specificity
0.7
0.88
0.7
0.89
0.22
0.38
0.5
0.0
−0.5
.estimate
LDA
Logistic regression
QDA
Model



Evaluating all models

We first start by evaluating thw models using the ROC of the models. Overall, we can say that all models have strong performance on both the training and holdout set. LDA in particular is interesting as it was the worse performing model but with the holdout set in improves significantly. Boosted trees is a model that drops its performance on the holdout set.

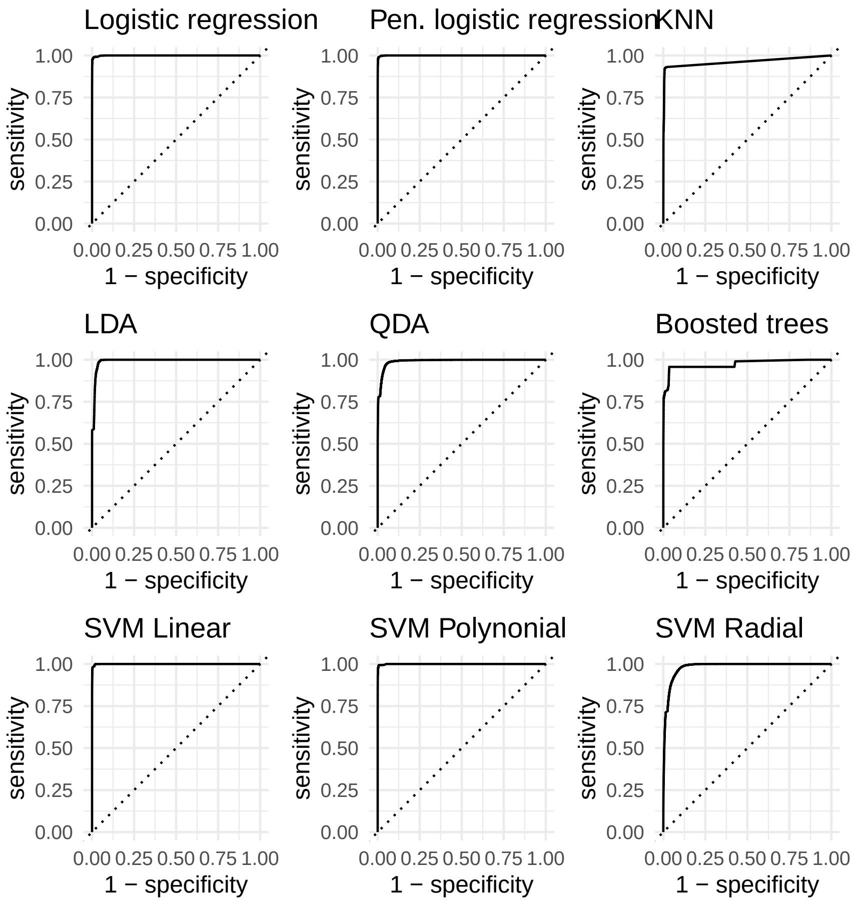


Figure 16: ROC for each model using the holdout set

Looking at the performance metrics. As expected, boosted trees has the lowest J-Index despite having the highest accuracy. This is because we are working with an unbalance dataset, which means that boosted trees are predicting the most common class to achieve its high accuracy. QDA is the model that performs the best on the J-Index. Which is the metric that we prefer to use because of the nature of the project and the unbalanced dataset.

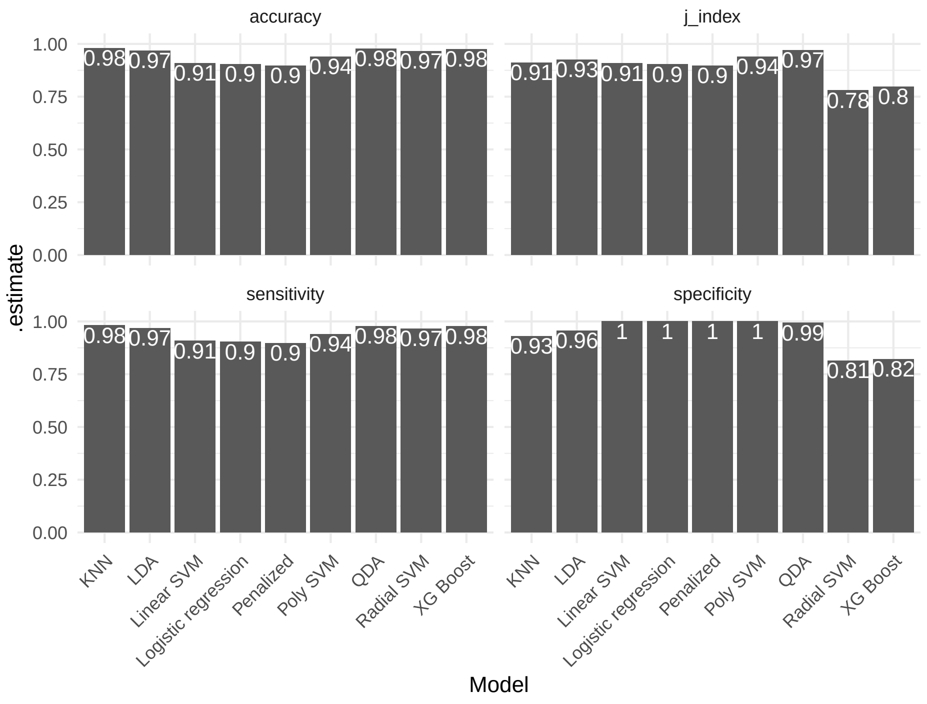


Figure 17: Performance metrics for each model using testing data

Conclusions

1. Best Performing Algorithms

After conducting cross-validation and analyzing the holdout dataset, it was found that the k-nearest neighbors (KNN), linear discriminant analysis (LDA), quadratic discriminant analysis (QDA) support vector machines (SVMs), and XG Boost algorithms performed exceptionally well in accurately identifying displaced persons living in makeshift shelters following the Haiti Earthquake. Each of these models exhibited high accuracy and produced fine ROC curves in the training and holdout set. The QDA model is one to note with high performance, performing well in all 4 metric categories, and having the highest accuracy, j_index and sensitivity. The KNN model and XG Boost rivaled the QDA in terms of accuracy, with all three models having an accuracy of 0.98. The ROC curves for all the models showed a little bit of variability between the holdout set and the training set. Overall, the ROC curves for all models on the training set were superb which indicates strong performance. The ROC curves for the holdout set did not perform quite as well as the curves on the training set. All the models produced concrete and solid results.

2. Why our findings are reconcilable

The congruence in findings across different algorithms underscores the reliability and consistency of the analysis. Despite variations in modeling techniques and underlying assumptions, the overarching goal of identifying blue tarps and locating displaced persons remained consistent. This compatibilit reinforces the notion that multiple approaches can be employed synergistically to tackle complex real-world problems, providing a safety net against model bias or overreliance on a single methodology.

3. Algorithm Recommendation – Detection of Blue Tarps

Drawing upon the strengths observed in the performance evaluations, it is recommended to deploy a hybrid approach leveraging the SVMs and boosted trees for blue tarp detection. SVMs excel in capturing complex relationships within the data and handling non-linear decision boundaries, while boosted trees offer robustness against overfitting and can effectively handle large-scale datasets. By combining these complimentary techniques, a more holistic and accurate detection mechanism can be established, enhancing the efficacy of disaster response efforts. An argument can be made for the boosted tree model to take priority in this context due to its higher sensitivity than the SVM models. Representing the true positive rate, sensitivity would be one of the most critical metrics to focus on because it ensures that no survivors are missed. This assumption is made possible by the fact that the scope of this project does not outline resource limitations for deployable rescue packages.

4. Relating metrics to this application context

There are several reasons why the ROC AUC metric is the preferred pathway for quantifying the model's predictive performance. First, this metric gives a holistic view of the model's performance regardless of any threshold. We can objectively assess the model's ability to distinguish between the classes across all possible classification thresholds. More importantly, however, ROC AUC is generally considered robust against class imbalance. As we pointed out in Part 1 of this analysis, only 3% of the data pertains to positive blue tarp identifiers. Since our primary objective is to properly identify emergency shelters set up by people in need of aid, it is critical to use a metric that evaluates a model's ability to discriminate between classes without being affected by the weight of class distribution. Realistically, data pipelines in emergency scenarios cannot supply high-quality aerial images with ideal class distributions. A resilient metric in the face of varying data volume and quality would be most suitable for this application context.

5. Over-indexing on Blue

The fact that the models performed essentially identically across datasets 1 and 3, indicates several interesting possibilities regarding our model and the nature of the data given. For context, the only difference between datasets 1 and 3 was that B1 and B2 were Red and Green respectively in dataset 1 while the opposite was true in dataset 3. Column B3 was consistently assigned to the blue color. This may be indicative of the fact that our model is not heavily dependent on the specific color information provided by the red and green channels. The blue channel, on the other hand, carries most of the discriminative information needed for identifying the blue-tarp shelters. It's also very possible that the model is using other features derived from the image data such as shape, size, or texture to distinguish the classes – features that are not sensitive to the specific color details provided by red and green. Again, a large part of this analysis relies on the assumption that blue tarp really is the most abundant resource for displaced victims of the Haiti earthquake relative to tarp of any other color or a material of different qualities.

6. Further Research and Improvement

While the current analysis yielded promising results, avenues for further research and improvement abound in the realm of automated image analysis for disaster response. Future endeavors could focus on exploring more advanced machine learning techniques, such as convolutional neural networks (CNNs), which might be able to extract and leverage hierarchical features that are less dependent on specific color channels. CNNs are specifically designed to process data with grid-like topology (such as images). It eliminates the need for manual feature extraction since it is highly effective at learning and generalizing from visual data. Additionally, integrating real-time sensor data and leveraging geospatial analytics for enhanced situational awareness could significantly augment the effectiveness of disaster response efforts. By fostering interdisciplinary collaboration and embracing innovative technologies, we can continue to advance the frontier of humanitarian aid and disaster relief.

Appendix

```r
knitr::opts_chunk$set(echo=FALSE)
library(tidymodels)
library(tidyverse)
library(patchwork)
library(discrim)
library(kableExtra)
library(bonsai)
library(kernlab)
library(MASS)
library(doParallel)
cl <- makePSOCKcluster(parallel::detectCores(logical = FALSE))
registerDoParallel(cl)

train <- read.csv("~/MSDS/Statistical learning/Project/Part 1/HaitiPixels.csv")

table2 <- train %>% group_by(Class) %>% summarize(count = n()) %>%
  mutate(percentage = round(count/sum(count)*100,0))

kable(table2, caption = "Number of observations per class")
train$Class2 <- ifelse(train$Class == 'Blue Tarp','Shelter','Other')
reds <- ggplot(train, aes(x = Red, color = Class2)) + geom_density() + theme_minimal()
blues <- ggplot(train, aes(color = Class2, x = Blue)) + geom_density() + theme_minimal()
greens <- ggplot(train, aes(color = Class2, x = Green)) + geom_density() + theme_minimal()

reds / blues / greens
ggplot(train, aes(x = Red, y = Blue, color = Class2)) +
  geom_point(size = 0.5, alpha = 0.5) +
  theme_minimal()
ggplot(train, aes(x = Red, y = Green, color = Class2)) +
  geom_point(size = 0.5, alpha = 0.5) + theme_minimal()
ggplot(train, aes(x = Blue, y = Green, color = Class2)) +
  geom_point(size = 0.5, alpha = 0.5) + theme_minimal()
set.seed(1)

# Set class as factor
train$Class2 <- as.factor(train$Class2)

resamples <- vfold_cv(train, v=10, strata=Class2)
cv_control <- control_resamples(save_pred=TRUE)
# Formula
formula <- Class2 ~ Red + Blue + Green

# Recipe
rec <- recipe(formula, data=train) %>%
  step_normalize(all_numeric_predictors())

# Logistic regression specification
logreg_spec <- logistic_reg(engine="glm", mode="classification")

# Logistic regression workflow
logreg_wf <- workflow() %>%
  add_recipe(rec) %>%
  add_model(logreg_spec)
# Define metrics
metrics =  metric_set(roc_auc)

# Cross-validation
logreg_cv <- fit_resamples(logreg_wf, resamples,
                           metrics=metrics, control=cv_control)

kable(collect_metrics(logreg_cv), caption = "Performance of logistic regression")
# Create function to scan thresholds
threshold_graph <- function(model_cv, model_name) {
  performance <- probably::threshold_perf(collect_predictions(model_cv),
                                          Class2, .pred_Shelter,
                                          thresholds=seq(0.01, 0.95, 0.01),
                                          event_level="second",
                                          metrics=metric_set(j_index))

  max_metrics <- performance %>%
    group_by(.metric) %>%
    filter(.estimate == max(.estimate))

  optimal_threshold <- max_metrics$.threshold[1]

  ggplot(performance, aes(x=.threshold, y=.estimate)) +
    geom_line() +
    geom_point(data=max_metrics, color="black") +
    labs(x="Threshold", y="J-Index") +
    geom_text(x = optimal_threshold, y = max_metrics$.estimate[1],
              label = paste("Optimal Threshold: ", round(optimal_threshold, 2),
                            "|","J-Index: ", round(max_metrics$.estimate[1], 2)),
              vjust = 0.5, hjust = -0.3, color = "black", size = 4) +
    coord_cartesian(ylim=c(0, 1))
}
threshold_graph(logreg_cv, "Logistic regression") + theme_minimal()
logreg_model <- fit(logreg_wf, data = train)
# Define LDA specifications
lda_spec <- discrim_linear(mode="classification") %>%
set_engine('MASS')

# LDA model specification
lda_wf <- workflow() %>%
  add_recipe(rec) %>%
  add_model(lda_spec)

# Cross validation
lda_cv <- fit_resamples(lda_wf, resamples, metrics=metrics, control=cv_control)
kable(collect_metrics(lda_cv), caption = "Performance of LDA")
threshold_graph(lda_cv, "LDA") + theme_minimal()
lda_model <- fit(lda_wf, data = train)
# Define QDA specifications
qda_spec <- discrim_quad(mode="classification") %>%
set_engine('MASS')
```

```r
# QDA model specification
qda_wf <- workflow() %>%
  add_recipe(rec) %>%
  add_model(qda_spec)

# Cross validation
qda_cv <- fit_resamples(qda_wf, resamples, metrics=metrics, control=cv_control)
kable(collect_metrics(qda_cv), caption = "Performance of QDA")
threshold_graph(qda_cv, "QDA") + theme_minimal()
qda_model <- fit(qda_wf, data = train)
# Define KNN specifications
knn_spec <- nearest_neighbor(engine='kknn',
                             mode="classification",
                             neighbors=tune())

# KNN model specification
knn_wf <- workflow() %>%
  add_recipe(rec) %>%
  add_model(knn_spec)

# Define neighbors parameters
nn_params <- extract_parameter_set_dials(knn_wf) %>%
  update(neighbors=neighbors(c(2, 20)))


# Cross validation
knn_cv <- tune_grid(knn_wf,
                    metrics=metrics,
                    resamples=resamples,
                    control=cv_control,
                    grid=grid_regular(nn_params, levels=20))
kable(show_best(knn_cv, metric='roc_auc'),
      caption = "Performance based on number of neighbors")
threshold_graph(knn_cv, "KNN") + theme_minimal()
knn_model <- knn_wf %>%
  finalize_workflow(select_best(knn_cv, metric="roc_auc")) %>%
  fit(train)
# Set seed to ensure reproducibility
set.seed(1)

# Define penalized logistic regression specifications
penlog_spec <- logistic_reg(engine='glmnet',
                            mode="classification",
                            penalty=tune(),
                            mixture=tune())

# Penalized logistic regression model specification
penlog_wf <- workflow() %>%
  add_recipe(rec) %>%
  add_model(penlog_spec)

# Define parameters
penlog_params <- extract_parameter_set_dials(penlog_wf) %>%
  update(penalty=penalty(c(-20, -5)),
         mixture=mixture(c(0, 1)))


# Cross validation
penlog_cv <- tune_grid(penlog_wf,
                    metrics=metrics,
                    resamples=resamples,
                    control=cv_control,
                    grid=grid_random(penlog_params, size=20))
kable(show_best(penlog_cv, metric='roc_auc'),
      caption = "Penalty and mixture tuning",
      digits = 8,
      format.args = list(scientific = FALSE))
autoplot(penlog_cv) + theme_minimal()
penlog_model <- penlog_wf %>%
  finalize_workflow(select_best(penlog_cv, metric="roc_auc")) %>%
  fit(train)
threshold_graph(penlog_cv, 'Penalized') + theme_minimal()
# Set seed to ensure reproducibility
set.seed(1)

# Define penalized logistic regression specifications
boosted_spec <- boost_tree(mode="classification", engine="lightgbm",
                       trees=500,
                       tree_depth=tune(), learn_rate=tune())

# Penalized logistic regression model specification
boosted_wf <- workflow() %>%
  add_recipe(rec) %>%
  add_model(boosted_spec)

# Define parameters
boosted_params <- extract_parameter_set_dials(boosted_wf)


# Cross validation
boosted_cv <- tune_grid(boosted_wf,
                    metrics=metric_set(roc_auc,j_index),
                    resamples=resamples,
                    control=cv_control,
                    grid=grid_random(boosted_params, size=20))
kable(show_best(boosted_cv, metric='roc_auc'),
      caption = "Tree depth and learn rate tuning")
threshold_graph(boosted_cv, 'Boosted trees') + theme_minimal()
calculate_metrics_probs <- function(model_name, model, data, threshold){
  results <- augment(model, data)
  results$prediction <- as.factor(ifelse(results$.pred_Shelter>threshold,'Shelter','Other'))
  metrics <- bind_rows(
   bind_cols(Model = model_name,j_index(results, Class2, prediction)),
   bind_cols(Model = model_name,accuracy(results, Class2, prediction)),
   bind_cols(Model = model_name,sensitivity(results, Class2, prediction)),
   bind_cols(Model = model_name,specificity(results, Class2, prediction)),
  )
  return(metrics)
}
boosted_model <- boosted_wf %>%
  finalize_workflow(select_best(boosted_cv, metric="roc_auc")) %>%
  fit(train)
calculate_metrics_probs('Boosted', boosted_model, train, 0.04)
svm_recipe <- recipe(formula, data=train)

#SVM with Linear Kernel
svm_linear_spec <- svm_linear(mode="classification", engine="kernlab",
                              cost = tune(), margin = tune())

svm_linear_wf <- workflow() %>%
  add_recipe(svm_recipe) %>%
  add_model(svm_linear_spec)

#SVM with Polynomial Kernel
svm_poly_spec <- svm_poly(mode="classification", engine="kernlab",
                          cost = tune(), margin = tune(), degree = tune())

svm_poly_wf <- workflow() %>%
  add_recipe(svm_recipe) %>%
  add_model(svm_poly_spec)

#SVM with Radial
svm_radial_spec <- svm_rbf(mode="classification", engine="kernlab",
                           cost = tune(), margin = tune(), rbf_sigma = tune())

svm_radial_wf <- workflow() %>%
  add_recipe(svm_recipe) %>%
  add_model(svm_radial_spec)

#Linear SVM
parameters <- extract_parameter_set_dials(svm_linear_spec)

linear_svm_tune <- tune_bayes(
  svm_linear_wf,
  metrics = metrics,
  resamples = resamples,
  param_info = parameters,
  control = control_bayes(save_pred = TRUE),
  iter = 25
)

#Polynomial SVM
parameters <- extract_parameter_set_dials(svm_poly_spec)

poly_svm_tune <- tune_bayes(
  svm_poly_wf,
  metrics = metrics,
  resamples = resamples,
  param_info = parameters,
  control = control_bayes(save_pred = TRUE),
  iter = 25
)

#Radial SVM
parameters <- extract_parameter_set_dials(svm_radial_spec)

parameters <- parameters %>%
  update(
    rbf_sigma = rbf_sigma(range=c(-4, 0), trans=log10_trans())
  )

radial_svm_tune <- tune_bayes(
  svm_radial_wf,
  metrics = metrics,
  resamples = resamples,
  param_info = parameters,
  control = control_bayes(save_pred = TRUE),
  iter = 25
)

kable(show_best(linear_svm_tune, metric='roc_auc'),
      caption = "Tuning for linear kernel")
kable(show_best(poly_svm_tune, metric='roc_auc'),
      caption = "Tuning for polynomial kernel")
kable(show_best(radial_svm_tune, metric='roc_auc'),
      caption = "Tuning for radial kernel")
threshold_graph(linear_svm_tune, 'Linear kernel') + theme_minimal()
threshold_graph(poly_svm_tune, 'Polynomial kernel') + theme_minimal()
threshold_graph(radial_svm_tune, 'Radial kernel') + theme_minimal()
svm_linear_model <- svm_linear_wf %>%
  finalize_workflow(select_best(linear_svm_tune, metric="roc_auc")) %>%
  fit(train)

svm_poly_model <- svm_poly_wf %>%
  finalize_workflow(select_best(poly_svm_tune, metric="roc_auc")) %>%
  fit(train)

svm_radial_model <- svm_radial_wf %>%
  finalize_workflow(select_best(radial_svm_tune, metric="roc_auc")) %>%
  fit(train)
get_ROC_plot <- function(model, data, model_name) {
  model %>%
    augment(data) %>%
    roc_curve(truth=Class2, .pred_Shelter, event_level="second") %>%
    autoplot() + theme_minimal() +
    labs(title=model_name)
}
logreg_roc <- get_ROC_plot(logreg_model, train, "Logistic regression")
lda_roc <- get_ROC_plot(lda_model, train, "LDA")
qda_roc <- get_ROC_plot(qda_model, train, "QDA")
knn_roc <- get_ROC_plot(knn_model, train, "KNN")
penlog_roc <- get_ROC_plot(penlog_model, train, "Pen. logistic regression")
```

```r
boosted_roc <- get_ROC_plot(boosted_model, train, "Boosted trees")
linear_svm_roc <- get_ROC_plot(svm_linear_model, train, "SVM Linear")
poly_svm_roc <- get_ROC_plot(svm_poly_model, train, "SVM Polynonial")
radial_svm_roc <- get_ROC_plot(svm_radial_model, train, "SVM Radial")
(logreg_roc + penlog_roc + knn_roc) / (lda_roc + qda_roc + boosted_roc) /
  (linear_svm_roc + poly_svm_roc + radial_svm_roc)
# Import the datasets
blue_tarp_files <- c(
  "orthovnir078_ROI_Blue_Tarps.txt",
  "orthovnir069_ROI_Blue_Tarps.txt",
  "orthovnir067_ROI_Blue_Tarps.txt",
  "orthovnir067_ROI_Blue_Tarps_data.txt"
)
data_frames <- list()
for (file in blue_tarp_files) {
  file_path <- file.path("C:/Users/Mauricio/Documents/MSDS/Statistical learning/Project/Part 1", file)
  data1 <- read.delim(file_path, skip = 10, header = FALSE, stringsAsFactors = FALSE)
  data1$V1 <- as.character(data1$V1)

  # Split the single column into multiple columns based on whitespace separator
  data1 <- data.frame(do.call(rbind, strsplit(data1$V1, "\\s+")))
  # Ensure that there are at least 11 columns after splitting
  if (ncol(data1) >= 11) {
    # Select the last three columns (B1, B2, B3)
    data1 <- data1[, c(9:11), drop = FALSE]
    # Rename the columns
    colnames(data1) <- c("B1", "B2", "B3")
    # Convert character columns to numeric
    data1[] <- lapply(data1, as.numeric)
    # Add a column to indicate whether it has a blue tarp
    data1$Class2 <- "Shelter"

    data_frames[[length(data_frames) + 1]] <- data1
  } else {
    print("Error: Insufficient columns after splitting.")
  }
}
non_blue_tarp_files <- c(
  "orthovnir078_ROI_NON_Blue_Tarps.txt",
  "orthovnir069_ROI_NOT_Blue_Tarps.txt",
  "orthovnir067_ROI_NOT_Blue_Tarps.txt",
  "orthovnir057_ROI_NON_Blue_Tarps.txt"
)

for (file in non_blue_tarp_files) {
  file_path <- file.path("C:/Users/Mauricio/Documents/MSDS/Statistical learning/Project/Part 1", file)
  data2 <- read.delim(file_path, skip = 10, header = FALSE, stringsAsFactors = FALSE)
  data2$V1 <- as.character(data2$V1)
  # Split the single column into multiple columns based on whitespace separator
  data2 <- data.frame(do.call(rbind, strsplit(data2$V1, "\\s+")))
  # Ensure that there are at least 11 columns after splitting
  if (ncol(data2) >= 11) {
    # Select the last three columns (B1, B2, B3)
    data2 <- data2[, c(9:11), drop = FALSE]
    # Rename the columns
    colnames(data2) <- c("B1", "B2", "B3")
    # Convert character columns to numeric
    data2[] <- lapply(data2, as.numeric)
    data2$Class2 <- factor(rep(FALSE, nrow(data2)), levels = c(TRUE, FALSE), labels = c("Shelter", "Othe


    data_frames[[length(data_frames) + 1]] <- data2
  } else {
    print("Error: Insufficient columns after splitting.")
  }
}

# Combine all data frames into one dataset
holdout_data <- do.call(rbind, data_frames)
library(gtools)

# Original dataset
# holdout_data <- data.frame(B1 = ..., B2 = ..., B3 = ..., Class2 = ...)

holdout_data$Class2 <- as.factor(holdout_data$Class2)

# Get all permutations of B1, B2, B3
color_permutations <- permutations(n = 3, r = 3, v = c("B1", "B2", "B3"))

# Loop through each permutation and create a new data frame
for (i in 1:nrow(color_permutations)) {
  # Extract the permutation
  permutation <- color_permutations[i, ]
  # Create a new data frame with reordered columns
  new_data <- holdout_data[, c(permutation, "Class2")]
  # Rename the columns to Red, Green, Blue, and Class2
  names(new_data) <- c("Red", "Green", "Blue", "Class2")
  # Assign to a unique variable
  assign(paste("dataset", i, sep = "_"), new_data, envir = .GlobalEnv)
}

dataset_list <- list(dataset_1, dataset_2, dataset_3, dataset_4, dataset_5, dataset_6)
plot_list <- list()
total_metrics_list <- list()

for (i in seq_along(dataset_list)) {
  current_dataset <- dataset_list[[i]]

  logreg_metrics <- calculate_metrics_probs('Logistic regression', logreg_model, current_dataset, 0.05)
  lda_metrics <- calculate_metrics_probs('LDA', lda_model, current_dataset, 0.01)
  qda_metrics <- calculate_metrics_probs('QDA', qda_model, current_dataset, 0.02)

  total_metrics <- bind_rows(logreg_metrics, lda_metrics, qda_metrics)
  total_metrics_list[[paste("total_metrics_dataset", i, sep = "_")]] <- total_metrics

  # Create a ggplot for the current set of metrics
  plot <- ggplot(total_metrics, aes(x = Model, y = .estimate)) +
    geom_col() +
    facet_wrap(~ .metric) +
    theme_minimal() +
    theme(axis.text.x = element_text(angle = 45, hjust = 1)) +
    geom_text(aes(label = round(.estimate, 2)), vjust = 1.1,
              position = position_dodge(width = 0.9), color = "white") +
    ggtitle(paste("Metrics for Dataset", i))
  plot_list[[paste("plot_dataset", i, sep = "_")]] <- plot
}

for (plot in plot_list) {
  print(plot)
}
logreg_roc <- get_ROC_plot(logreg_model, dataset_1, "Logistic regression")
lda_roc <- get_ROC_plot(lda_model, dataset_1, "LDA")
qda_roc <- get_ROC_plot(qda_model, dataset_1, "QDA")
knn_roc <- get_ROC_plot(knn_model, dataset_1, "KNN")
penlog_roc <- get_ROC_plot(penlog_model, dataset_1, "Pen. logistic regression")
boosted_roc <- get_ROC_plot(boosted_model, dataset_1, "Boosted trees")
linear_svm_roc <- get_ROC_plot(svm_linear_model, dataset_1, "SVM Linear")
poly_svm_roc <- get_ROC_plot(svm_poly_model, dataset_1, "SVM Polynonial")
radial_svm_roc <- get_ROC_plot(svm_radial_model, dataset_1, "SVM Radial")
(logreg_roc + penlog_roc + knn_roc) / (lda_roc + qda_roc + boosted_roc) /
  (linear_svm_roc + poly_svm_roc + radial_svm_roc)
logreg_metrics <- calculate_metrics_probs('Logistic regression', logreg_model, dataset_1, 0.05)
lda_metrics <- calculate_metrics_probs('LDA', lda_model, dataset_1, 0.01)
qda_metrics <- calculate_metrics_probs('QDA', qda_model, train, 0.02)
penlog_metrics <- calculate_metrics_probs('Penalized', penlog_model, dataset_1, 0.04)
knn_metrics <- calculate_metrics_probs('KNN', knn_model, dataset_1, 0.01)
boost_metrics <- calculate_metrics_probs('XG Boost', boosted_model, dataset_1, 0.04)
linear_svm_metrics <- calculate_metrics_probs('Linear SVM', svm_linear_model, dataset_1, 0.02)
poly_svm_metrics <- calculate_metrics_probs('Poly SVM', svm_poly_model, dataset_1, 0.02)
radial_svm_metrics <- calculate_metrics_probs('Radial SVM', svm_radial_model, dataset_1, 0.02)
total_metrics <- bind_rows(logreg_metrics, lda_metrics,
                           qda_metrics, knn_metrics, penlog_metrics,
                           boost_metrics, linear_svm_metrics,
                           poly_svm_metrics, radial_svm_metrics)
ggplot(total_metrics, aes(x = Model, y = .estimate)) + geom_col() +
  facet_wrap(~ .metric) + theme_minimal() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1)) +
  geom_text(aes(label = round(.estimate,2)), vjust = 1.1,
          position = position_dodge(width = 0.9), color = "white")
```