Disaster Relief Project Part 2

Mahin Ganesan, Wyatt Priddy, and Taylor Tucker

2024-04-22

Introduction

In the aftermath of a devastating earthquake that rattled Haiti, displaced people are living in makeshift shelters awaiting support from aid workers. The aid workers, mainly from the United States military, are trying to reach the dispersed camps. With communication lines being down and the terrain being impassable, there are challenges in providing relief in a time-sensitive manner.

It is known that the makeshift shelters are largely constructed with blue tarps, therefore the Rochester Institute of Technology deployed airplanes to collect high resolution geo-referenced imagery. This will help us identify where displaced persons are based on the tarps.

To determine where to allocate aid, we need to use data mining against the thousands of photos taken each day which human eyes can not efficiently filter through. Determining the location in a timely manner will be paramount to rendering aid successfully and saving human life.

The primary aim of this experiment is to evaluate the efficacy of various classification algorithms in accurately and promptly identifying the presence of makeshift shelters and, by extension, the displaced persons residing within them. By harnessing the power of machine learning and image analysis, our objective is to develop a robust algorithm capable of rapidly scanning through the imagery data, pinpointing areas of interest, and facilitating timely intervention by rescue teams.

To facilitate efficient rescue efforts, we will need to determine a threshold where the number of false positives is minimal. This may not necessarily be the model that has the highest performance in accuracy but rather the highest performance with precision, or the proportion of true positives that are correctly identified by the model out of all true positives and false positive. If there are additional aid resources after rescuing all persons identified from our models, we can loosen our model specifications to classify more objects as blue tarps in an attempt to expand our search efforts.

Data

Team members have assisted our mission by classifying training data consisting of 63,241 data points for our investigation. There are 5 classifications that have been assigned to the pixel level data:

1. Blue Tarp
2. Rooftop
3. Soil
4. Various Non-Tarp
5. Vegetation

As expected when trying to find a needle in a haystack, our representation of misplaced persons (blue tarps) makes up only a small portion of the data set. Just 3.2% of the total data set, or 2022 records, are classified

as `blue tarp`. After inspecting the average color of the classes, it becomes apparent that even though blue tarps are a small section of the data their distinction in color should set them apart.

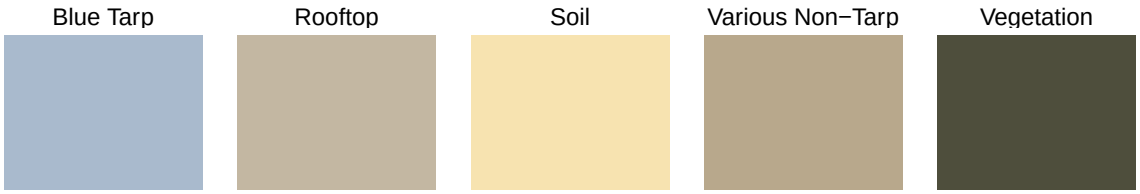

Figure 1: Average Color of Class

Vegetation and soil cover over 73% of the pictures as to be expected of pictures that largely will include countryside.

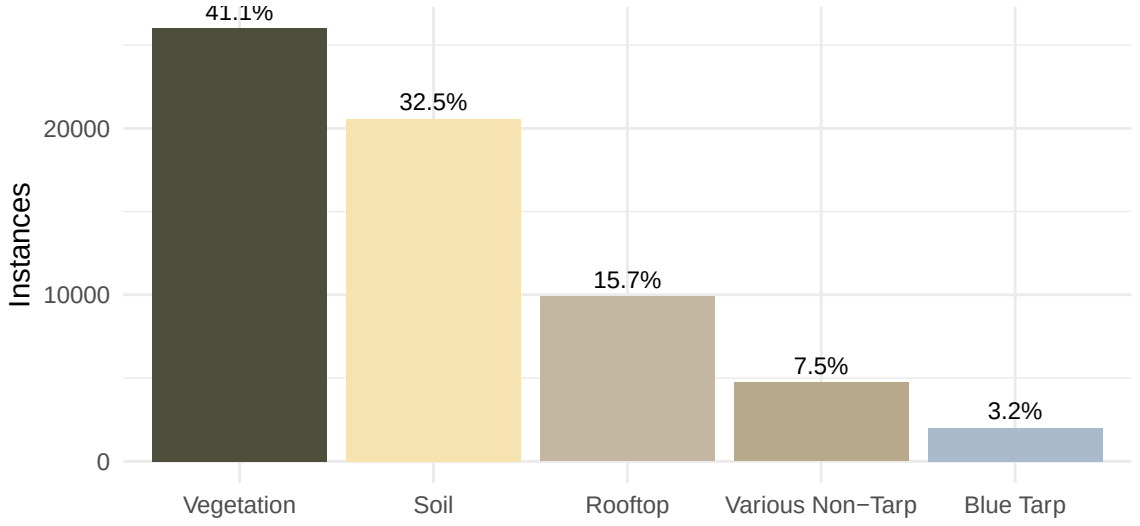

Figure 2: Distribution of Classifications in Training Set

When viewing the distribution of blue tarps within the training set, a clear pattern can be discerned distinguishing the blue tarps from the other classes in the data set. This leaves us optimistic that there will be an optimal machine learning algorithm that will help us determine the location of the blue tarps and save the lives of numerous refugees.

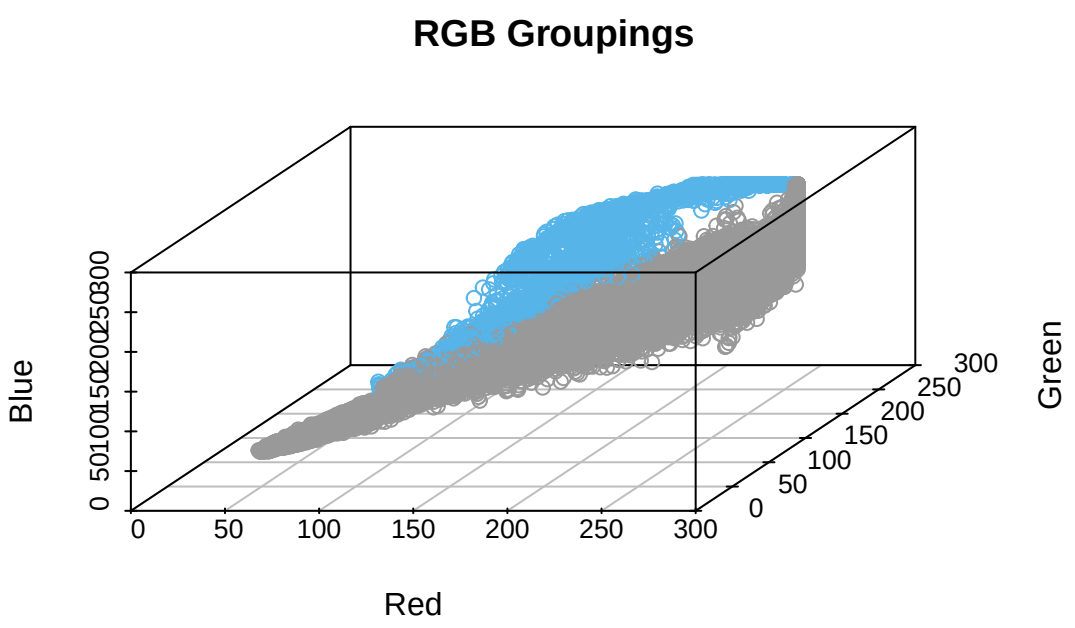

Figure 3: 3d Visualization of Blue Tarps

Description of Methodology

Each model is being trained with 10-fold cross-validation to ensure the models are generalizing well and stable over the data set rather than performing well on a single subsection of the train/test split.

The tunable parameters in the five models were number of neighbors in KNN, and penalty and mixture in the Penalized Logistic Regression models. For the Random Forest classifier, we tuned the number of features each tree would sample, the number of trees in the forest, and the minimum values for a split. Finally, for the linear SVM, we tuned the cost and margin hyperparameters; for the polynomial SVM, we tuned the cost, margin, degree, and scale factor hyperparameters; and for the RBF SVM, we tuned the cost, margin, and RBF sigma parameters. These hyperparameters were tuned using a random grid search and 10-fold cross validation. The grid boundaries were determined using a guess-and-check approach, utilizing the autoplot method to ensure that the optimal range of values was included in the boundaries. The optimal combination of hyperparameters was defined by the best metric scores.

The metrics we used are listed below, and are canonically used when measuring classification model performance.

The thresholds were selected by calculating metrics for a sequence of thresholds and selecting the threshold which created a model which performed best based on our metrics. We decided to not allow any threshold above 0.85, since that might make our model too sensitive and misclassify blue tarps as non-blue tarps leading to a pockets of refugees going undiscovered. This threshold limit also increased our confidence in the decisions our models made.

When given the holdout dataset, the color columns were unlabeled, and thus, the correct combination of red, green, and blue columns had to be determined. This was done manually, using the given blue tarp color values. We found, by trying the various permutations of the columns, one combination consistently produced a blue color, while the other permutations were various, clearly non-blue colors.

Libraries used:

- tidymodels: used for model creation, cross-validation, and determining model performance
- tidyverse: used for plotting, data manipulation, and chaining operationsx‘
- probably: used for assessing threshold performance on the models
- discrim: used for the LDA/QDA models
- doParallel: used for setting up parallel processing of the code to speed up performance.
- scatterplot3d: used to render 3d scatter plot of RGB classifications of Blue Tarps

Metrics utilized:

- specificity: $\frac{TN}{FP+TN}$
- sensitivity: $\frac{TP}{TP+FN}$
- accuracy: $\frac{TP+TN}{TP+TN+FP+FN}$
- precision: $\frac{TP}{TP+FP}$
- Receiver Operating Characteristic Area Under the Curve (ROC AUC)

Results

Logistic Regression Model

The logistic model was fit across a 10-fold cross-validation. The model seems to be generalizing well between the different folds with minimal variation throughout the training performance metrics as seen in *Figure 4*. The average performance metrics tell us that the initial model has very high sensitivity with little variation across the folds. Specificity is much lower with more variation in the results indicating is it not as stable. This can be interpreted as the model is performing well at classifying the blue tarps. However it is classifying many objects that are not blue tarps creating many false positives as seen in the lower specificity.

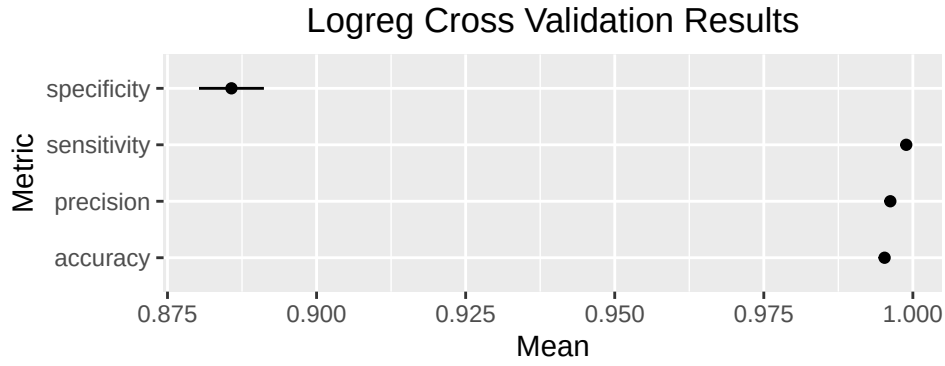

Figure 4: Logistic Regression Cross Validation Results

After performing threshold analysis, a threshold of 0.85 was selected to minimize the false positives while keeping a substantial portion of the true positives in the analysis. Seen in *Table 3*, Utilizing this threshold resulted in a precision rating of 0.998 meaning that 99.8% of the positive predictions in the model are true positives. The AUC score of 0.999 means that the model has near perfect classification abilities and is highly reliable in determining whether a data point is a blue tarp or not. Similar to the results from the training

data, specificity is lower at 97.7% on the tesing data. The performance on the testing data in relation to the cross-validation of the training set indicates that the model is not overfitting on the training data

At the threshold of 0.85, 10,930 of the 14,480 blue tarps in the testing set are correctly identified by the model. Of the 1,989,697 not blue tarps, only 29,133 are false positives. Of all the blue tarps identified by the model, there is a 69.1% rate of aid workers time and resources not being wasted on futile searches. While 326 blue tarps were unidentified by the model (2.2% of total tarps), once the aid workers can provide resources to the 14,154 correctly identified refuges we can then expand the search. This will help us maximize a timely and precise search of refugees accounting for early and easy wins before expanding the search areas to get the last pockets of refugees.

Figure 5: Logistic Regression ROC Curve

Penalized Logistic Regression Model

The penalized logistic model using elasticnet (L1-L2 regularization) performs better than LDA and QDA, but worse than the untuned logistic regression and KNN overall when looking at the performance metrics. The sensitivity, accuracy, and precision are pretty similar to the other models and are all above 0.995, indicating the model does very well on the test data. However, the specificity is around 0.87, which is more than 2% less than the KNN model. This is a very important distinction because even though it is only a couple percentage points lower, it means a lower rate of true positives, or displaced persons found accurately using this model, so using the KNN model over this would likely result in saving more lives and less mistakes of military going to the wrong spots.

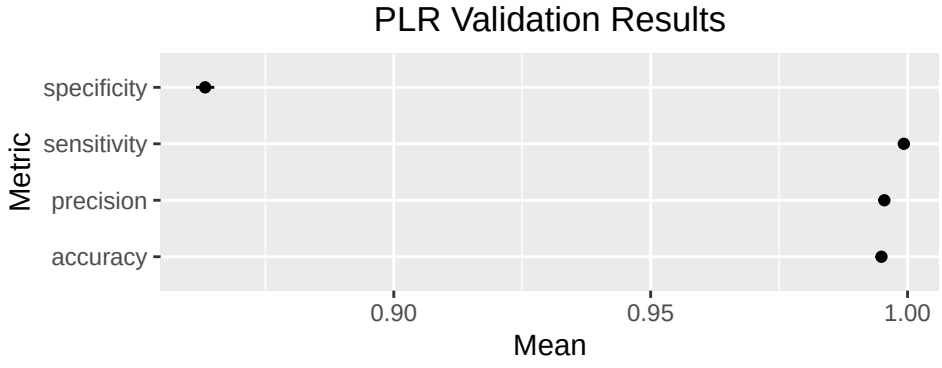

Figure 6: Penalized Logistic Regression Cross Validation Results

The penalized logistic model has 5,288 false positives and 360 false negatives. While the model finds 6,121

more true positive blue tarp locations than the previous best model, KNN, it also has 1,270 more false positives. This is a big deterrent to the model since a higher false positive rate means a higher chance of military going to a wrong pocket, costing valuable time that could end up meaning less lives saved. This makes the KNN model more attractive as a model choice compared to the penalized logistic regression model.

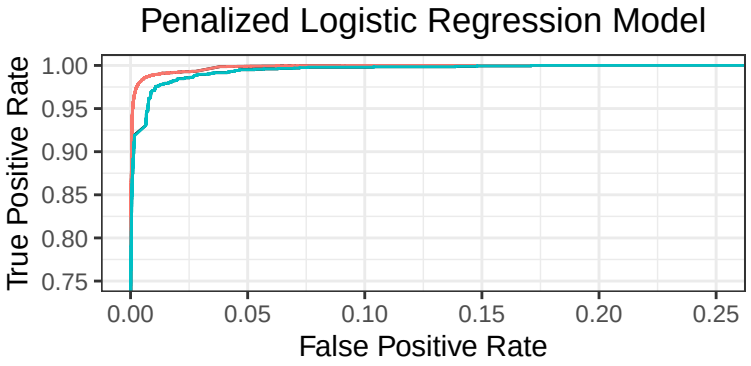

Figure 7: Penalized Logistic Regression ROC Curve

We tuned the penalty and mixture parameter using a random grid to get the optimal number that maximizes accuracy of the model. The tuning parameters had a range of (-20,-10) for penalty and (0,10) for mixture and a paramater space of 200 to encompass the variations.

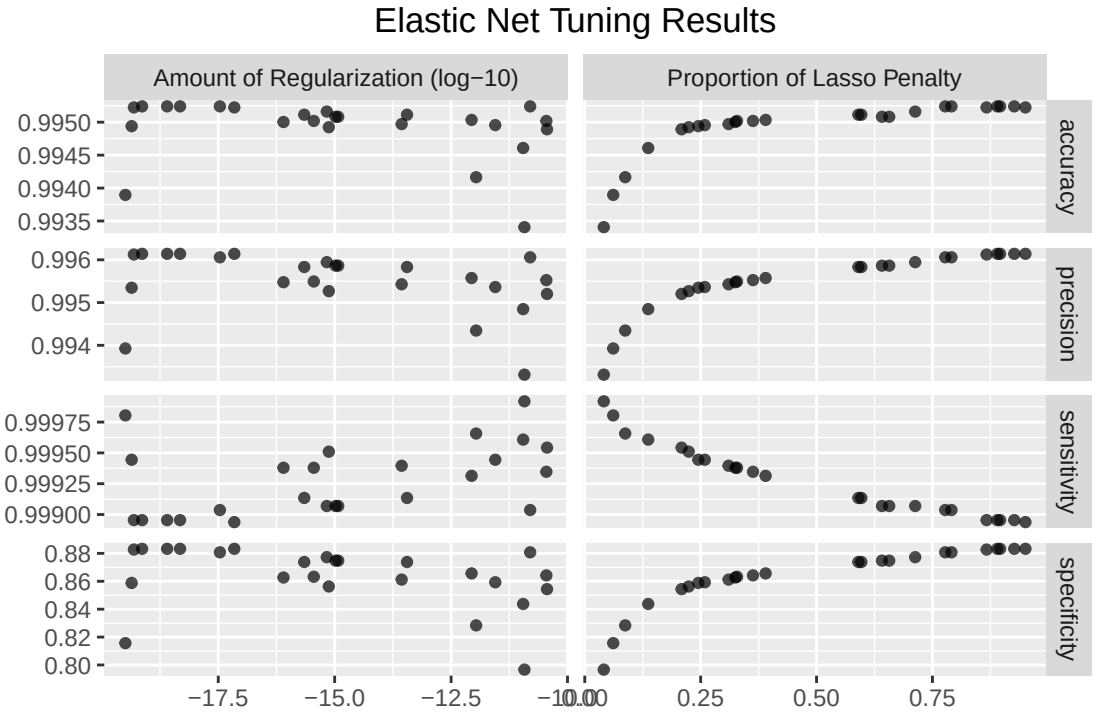

Figure 8: Tune Results

LDA Model

The LDA Model has similar variations in the cross-validation performance as the logistic regression model seen in *Figure 9*. Specificity seems to be the only metric not performing at above 97%. The specificity across the 10-fold cross-validation of the training set is around 80% with slightly less variation within one standard error than the logistic regression.

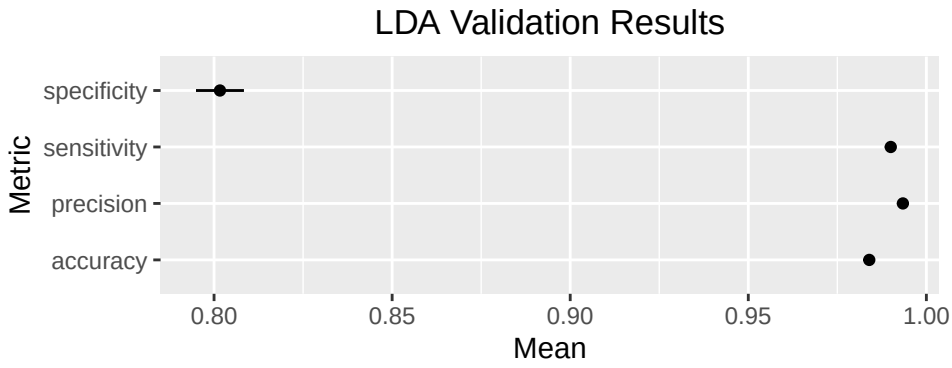

Figure 9: Linear Discriminate Analysis Cross Validation Results

The threshold selected was also 0.84 to keep the number of false positives at a minimum. At this threshold, the LDA model performs worse across all performance metrics and than logistic regression model, seen in *Table 3*. This seems to indicate that a linear decision boundary may not be the best fit for the data set being utilized.

At the selected threshold of 0.84, only 10,930 of the 14,480 blue tarps in the testing set are correctly identified by the model. Of the 1,989,697 not blue tarps, 29,133 were false positives. This would represent an increase of 600% more false positives than the penalized logistic regression and could equate to numerous man hours and resources being wasted if we were to go with this model. 3,550 blue tarps were false negatives also resulting in more pockets of refuges being missed by this model.

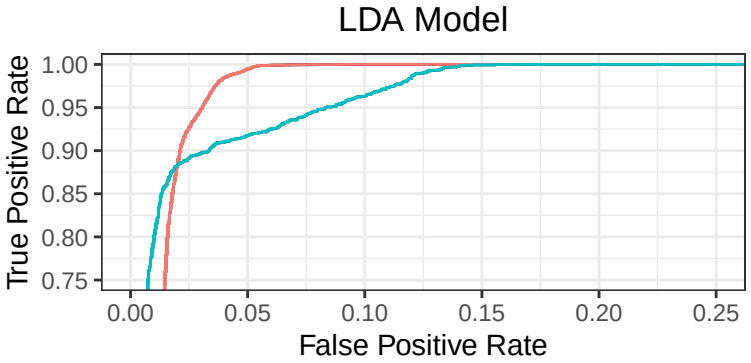

Figure 10: Linear Discriminate Analysis ROC Curve

QDA Model

The QDA model performs better than the LDA model for the 10-fold cross-validation but worse than the logistic regression. The main difference between all the models thus far has been the variation in specificity as sensitivity, precision, and accuracy have been comparable across the models. The QDA model also has

smaller variation within one standard error for specificity than the logistic regression but has a higher average specificity than the LDA model.

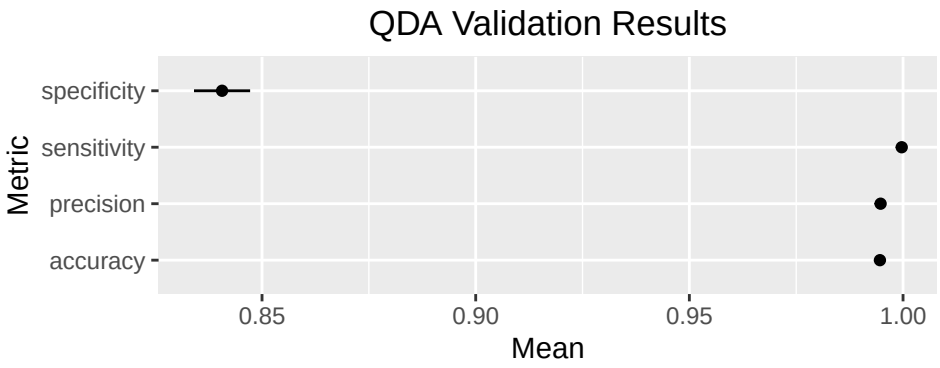

Figure 11: Quadratic Discriminate Analysis Cross Validation Results

At the selected threshold of 0.85 on the testing data, the QDA model only has 2,003 false positives. However, the number of true positives predicted is 9,007 which results in 5,147 less pockets of refuges being rescued and the number of false negatives is 5,473 which is an increase of 1,578% more blue tarps being unidentified compared to the logistic regression model.

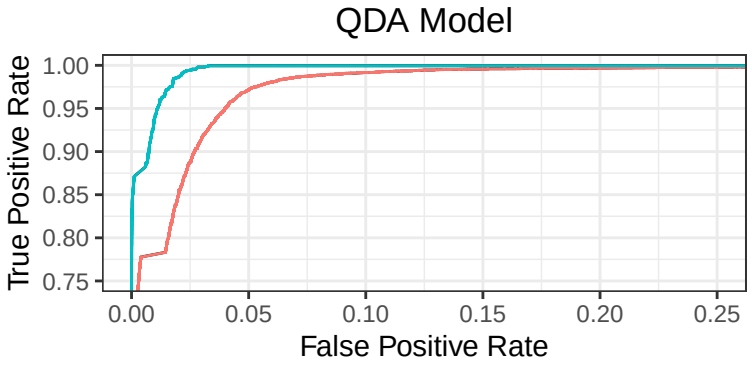

Figure 12: Quadratic Discriminate Analysis ROC Curve

K-Nearest Neighbors Model

The KNN model performs somewhat worse than the other models looking at the 10-fold cross validation. The sensitivity, accuracy, and precision are pretty similar to the other models, but the specificity performs better than LDA and QDA by a wide margin and performs better than logistic regression by 1%, which can make a pretty large impact in practice, since a higher specificity indicates more true positives, which means a higher chance of US military troops going to the correct location and saving valuable time when trying to find blue tarps with displaced persons.

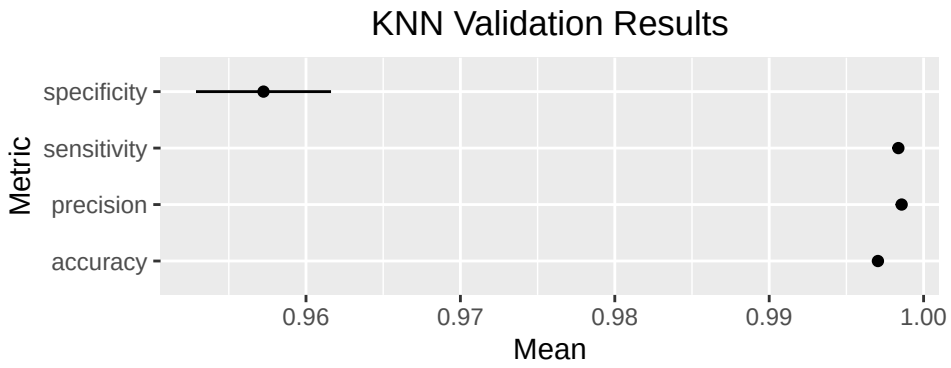

Figure 13: KNN Cross Validation Results

The KNN model has 4,018 false positives and 6,481 false negatives, and finds more true positive blue tarp locations than all of the other models so far, performing substantially worse than the logistic regression with 6,155 less true positives, which means thousands less pockets of displaced persons found. 6,155 less true positives has significant real world significance in this case, since every additional correct pocket missed is possibly many lives lost. So, finding more true blue tarp locations is the most important factor, since it means there will be more people saved and less errors made in the search, making the 20 neighbor KNN model an unfavorable option.

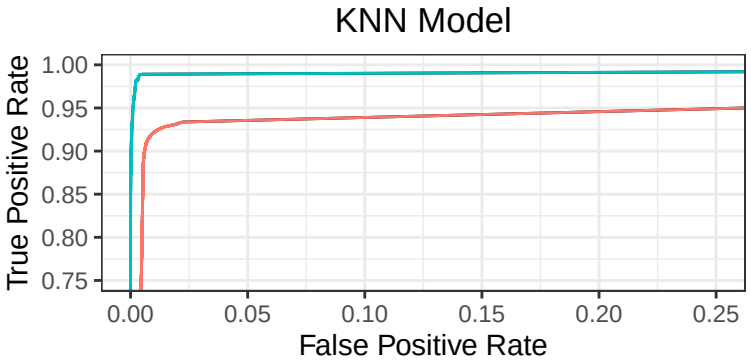

Figure 14: KNN ROC Curve

We tuned the neighbors parameter using a random grid and a size of 22 with a range of 5 to 50 for the neighbors. We found the optimal number of neighbors that maximizes the roc_auc metric, which ended up being 20. The roc_auc value for this number of neighbors is about 0.994.

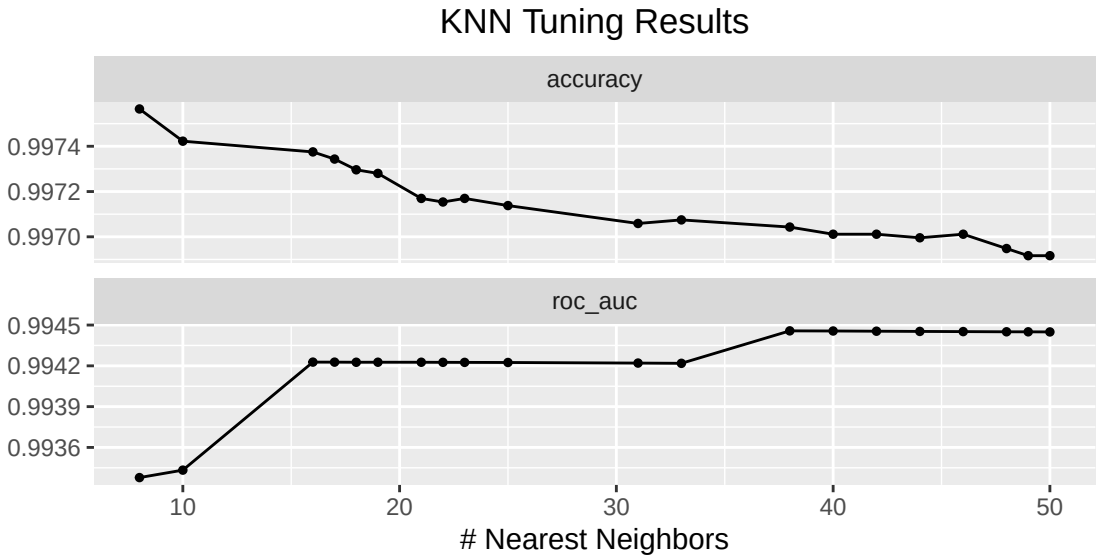

Figure 15: Tune Results

Random Forest

The Random Forest Classifier model performed very well on this data, and indeed performed well among all the classifiers. The model was trained and tuned using 10-fold cross validation. The parameters that were tuned were `mtry`, `trees`, and `min_n`, which represent the number of predictors which will be used at each split, the number of trees in the forest, and the minimum number of datapoints required for splitting a node, respectively. The optimal values for the aforementioned hyperparameters were `mtry=1`, `trees=355`, and `min_n = 6`, which gave a cross-validated training accuracy of 0.9972.

Table 1: Best RF Model

mtry	trees	min_n	.metric	mean
1	355	6	accuracy	0.9971854

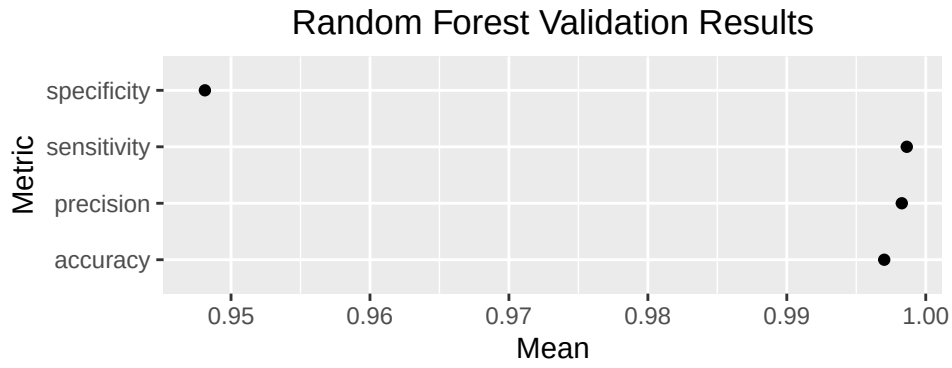

Figure 16: Random Forest Cross Validation Results

At this threshold of 0.85, we found that the model had 1,695 false positives and 7,880 false negatives, and a testing ROC AUC score of 0.976.

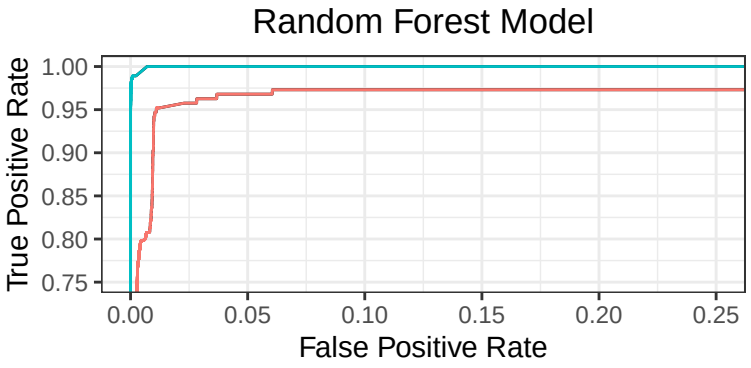

Figure 17: Random Forest ROC Curve

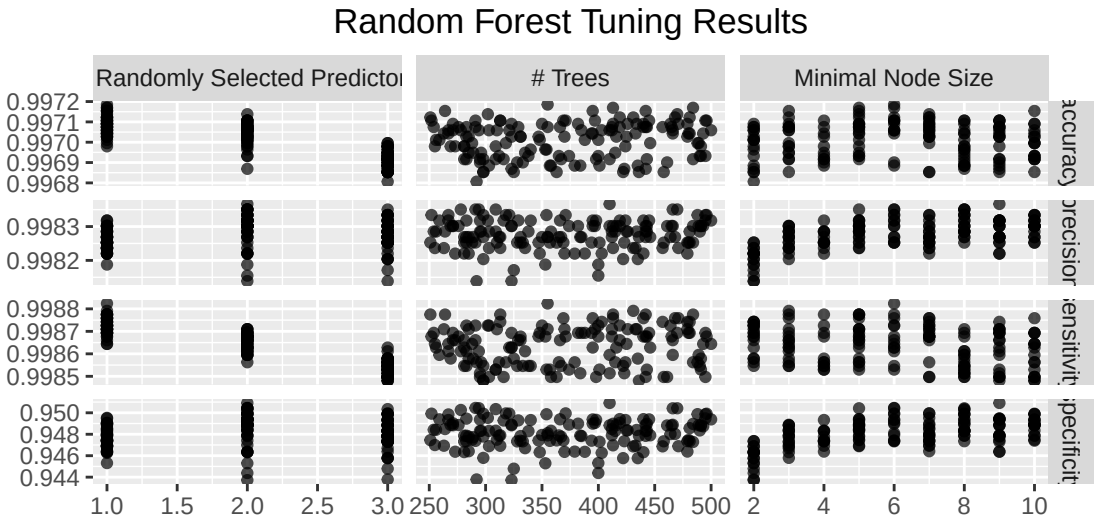

Figure 18: Tune Results

Linear Support Vector Machine

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

The Linear Support Vector Machine model performs better than all of the previous models. Similar to the other models, the cross-validation results are very strong, close to 100% on all metrics outside of specificity. The model also performs much better on the test set than the previous models, showing its ability to protect against overfitting better than the other models.

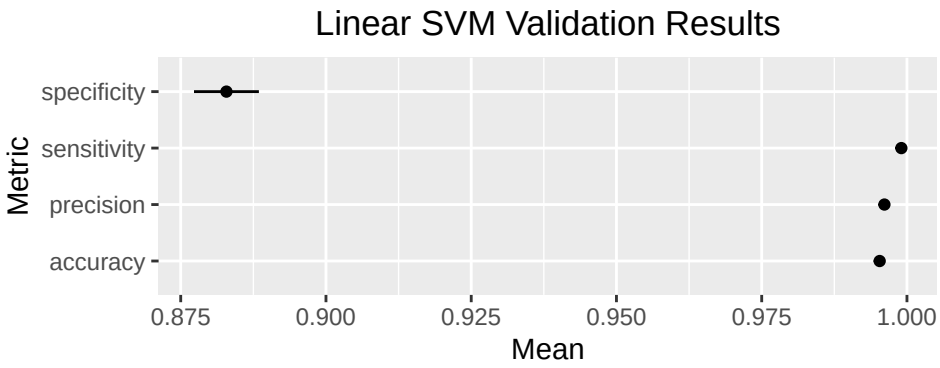

Figure 19: Linear Support Vector Machine Cross Validation Results

The model has 8,287 false positives and 315 false negatives. This is a drastic improvement over the test results of the other models, specifically relating to the false negatives, which have a smaller count, meaning a higher percent impact with each less false negative. This means more correctly identified blue tarps and therefore more lives saved. Later, we will see that the Polynomial SVM performs even better than this model, meaning we will not end up using this model as our final choice

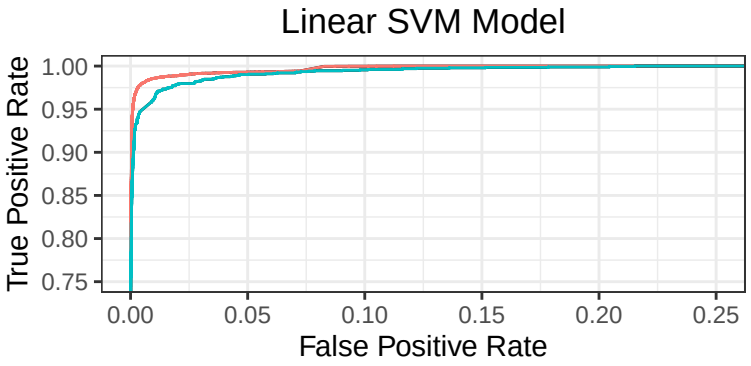

Figure 20: Linear Support Vector Machine ROC Curve

We tuned the cost and margin parameters to maximize roc_auc of the model using the default ranges for the parameters. We used Bayesian hyperparameter tuning for the linear SVM model, with ten iterations.

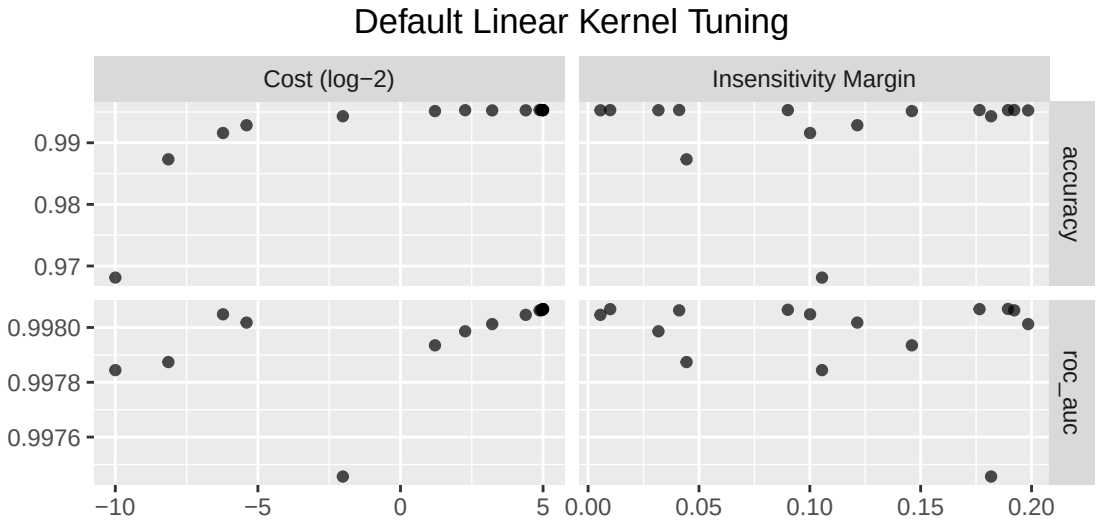

Figure 21: Tune Results

Polynomial Support Vector Machine

The Polynomial SVM performs well relative to all of the previous models in terms of the ROC-AUC curve based on the test data results. The model has very high accuracy and precision, matching all of the other models. While the CV results from this model aren't the best out of all of the models from the perspective of specificity, the results from the test metrics are the best out of any model, demonstrating the importance of having a training and test set to watch for overfitting, which the Polynomial SVM does the best to nullify.

```
## unique notes:
## ----------------------------------------------
## Error: cannot allocate vector of size 22.6 Gb
```

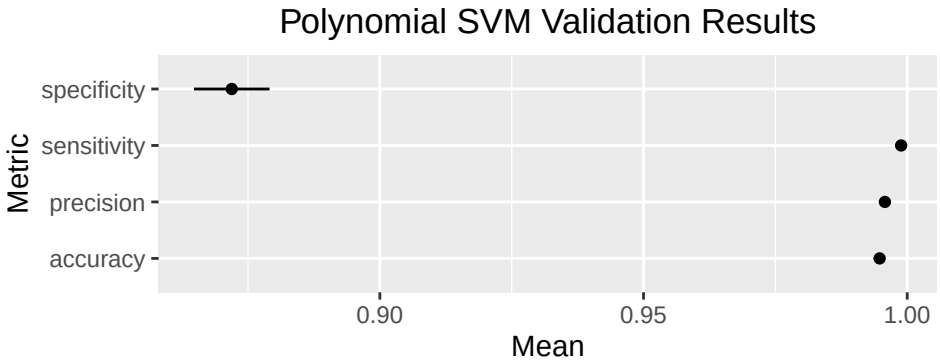

Figure 22: Polynomial Support Vector Machine Cross Validation Results

The model has 6,671 false positives and 195 false negatives. This compares well to all of the other models, proving to be the best model so far, indicated by the ROC curve below. So, choosing this model would take us in the right direction in terms of saving the most displaced persons as fast as possible.

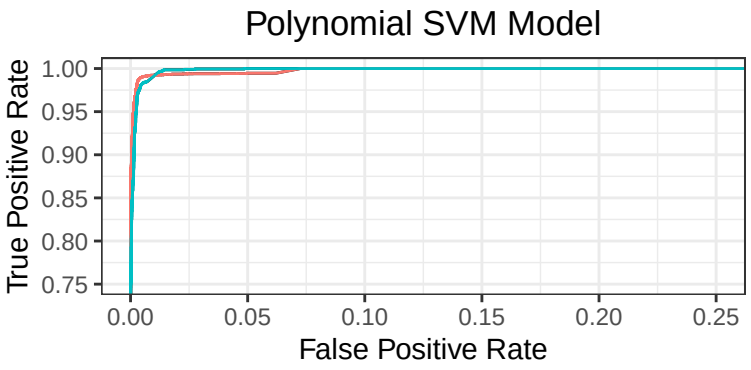

Figure 23: Polynomial Support Vector Machine ROC Curve

For the Polynomial SVM model, we tuned the cost, margin, degree, and scale factor. Degree determines which degree the polynomial should be, based on the one with the best roc_auc from our model. We decided to use Bayesian hyperparameter tuning for our model, with ten iterations.

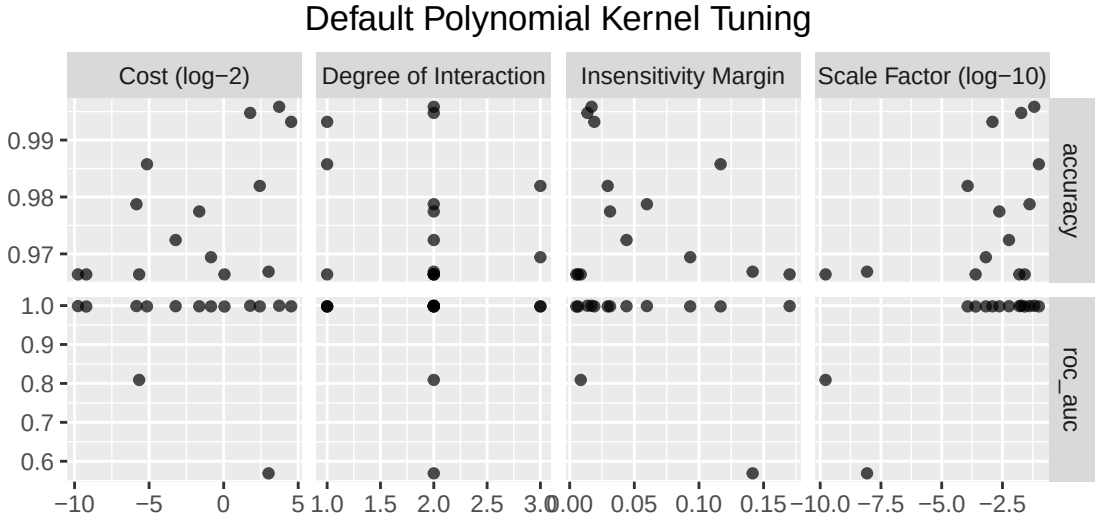

Figure 24: Tune Results

Radial Basis Function Support Vector Machine

The Radial Basis Function (RBF) Support Vector Machine model does not perform well compared to some of the previous models, especially the other SVM models. The RBF SVM model performs well overall in terms of CV metrics, but shows a much worse ROC curve than some of the other models, with a lower true positive rate on the test set. The difference is a few percent below than the other SVM models, which would have a very important real-world impact, with less displaced persons possibly being saved.

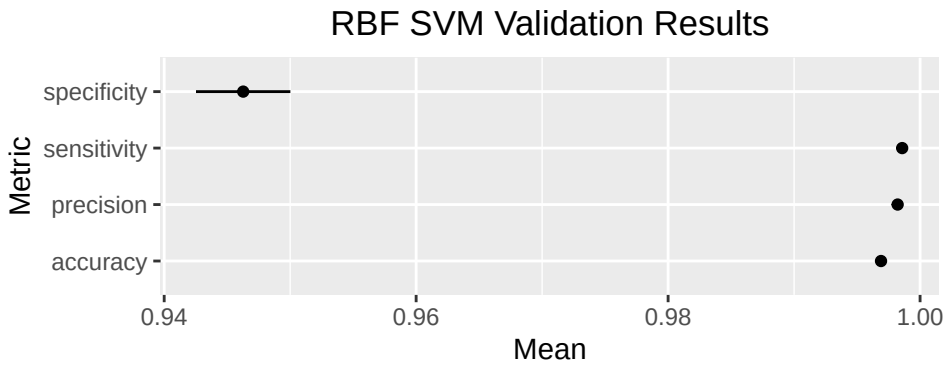

Figure 25: Radial Basis Function Support Vector Machine Cross Validation Results

The model has 6,334 false positives and 5,069 false negatives. The number of false negatives is drastically higher than the other SVM models, leading to many incorrect classifications of blue tarps as non-blue tarps. This could cause many displaced persons to not be saved, leading us to not choose this as our final model.

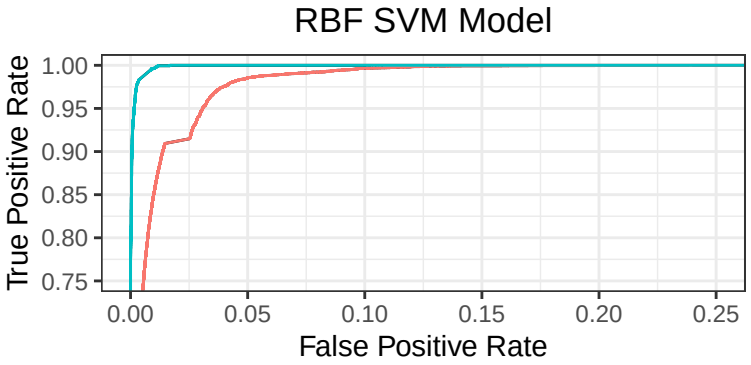

Figure 26: Radial Basis Function Support Vector Machine ROC Curve

For this model, we tuned the cost, margin, and the sigma. Sigma refers to how non-linear the model will be, affecting the shape of the decision boundary. We used default ranges for the parameters and tuned the parameters using Bayesian hyperparameter optimization, with ten iterations.

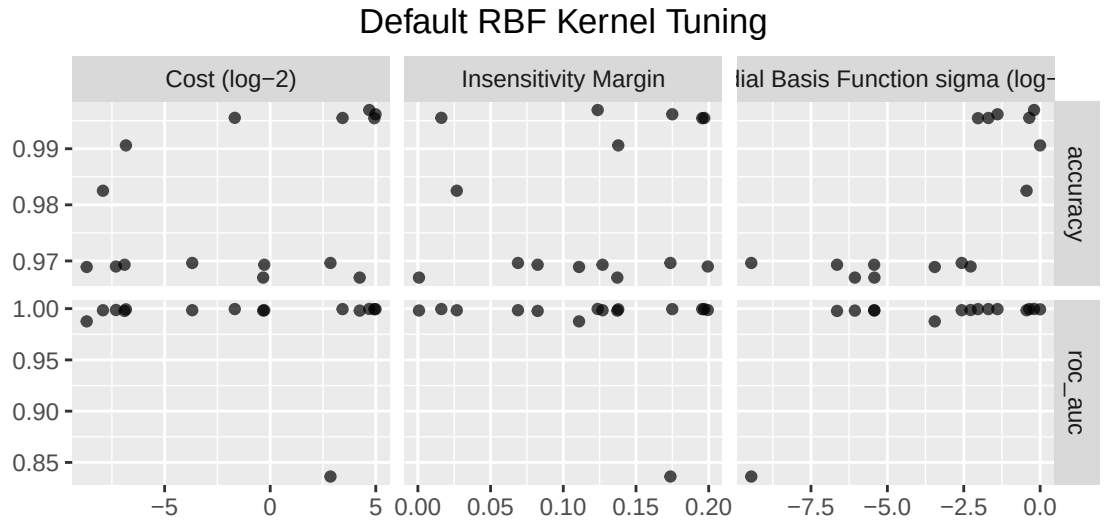

Figure 27: Tune Results

Table 2: Test Performance Metrics

Threshold	specificity	sensitivity	accuracy	precision	roc_auc	model
0.85	0.9775	0.9968	0.9967	0.9998	0.9994	logreg
0.84	0.7548	0.9854	0.9837	0.9982	0.9921	LDA
0.85	0.6220	0.9990	0.9963	0.9973	0.9915	QDA
0.85	0.5524	0.9980	0.9948	0.9967	0.9641	KNN
0.85	0.9782	0.9958	0.9956	0.9998	0.9991	Linear SVM
0.85	0.9795	0.9968	0.9967	0.9999	0.9996	Polynomial SVM
0.85	0.6499	0.9968	0.9943	0.9975	0.9940	RBF SVM
0.85	0.4558	0.9991	0.9952	0.9961	0.9758	Random Forest
0.83	0.9751	0.9973	0.9972	0.9998	0.9995	Penalized LR

Conclusion

- Based on our analyses, we found that the Polynomial SVM Model performs the best out of the nine models explored. Based on our performance metrics of specificity, sensitivity, accuracy, precision, and AUC, this model consistently scored either higher than, or similarly to, the other models. In particular, the Polynomial SVM Model had the highest testing specificity, accuracy, and precision out of all the models we tried, with values of 0.9795, 0.9967, and 0.9999, respectively. This model incorrectly classified 6,671 non-tarps as tarps, and incorrectly classified 195 tarps as non-tarps. We found this to be the best combination of false-positives and true-negatives in order to efficiently allocate resources in a rescue situation. In particular, the 195 non-tarps which were classified as tarps was relatively low compared to the size of the holdout set, which would allow rescue teams to avoid performing operations at non-tarp locations.

The numerous models tested throughout this analysis all yielded better results, albeit at varying degrees, than if we were to classify everything as a non-blue tarp based on the ROC curves. By maintaining a high threshold, we were able to be precise regarding our classification of blue tarps to minimize the false positives saving valuable time and resources. Robustly testing through cross validation and tuning of these

nine models, we can be confident the best method is being applied to find refugees after this horrific natural disaster. If we were to explore a different strategy to identify the refugees, it could be very possible that a different model would be more optimal. While the Polynomial SVM was best for our needs now, other models may shine in a different light.

- The rationale for the using the Polynomial SVM to detect blue tarps is not only that it performed the best based on our analysis, but also based on results found in further exploratory data analysis. In *Figure 3*, we see that the blue tarp dataset, in the 3D space of the red, green, and blue values, appears to be separable by some hyperplane. Thus, it makes sense that a support vector machine, especially one as flexible as a Polynomial SVM, might be able to separate those classes. Further, we found that the Polynomial SVM performed best on the holdout set when it came to our most important metrics, mentioned below. The most important metrics to our analysis were precision, specificity, and ROC AUC. In all of these metrics, the Polynomial SVM performed the best out of all of the models we made, with a specificity of 0.9833, a precision of 0.9999, and a ROC AUC of 0.9994. Thus, our recommendation is to use the Polynomial SVM for the detection of blue tarps.

- The metrics calculated in the tables above are highly relevant to this application context, especially when considering that a real and tangible allocation of resources may be informed by, and human lives saved by, the metrics in this report. The metrics we used were accuracy, precision, sensitivity, specificity, and ROC AUC. The accuracy of the model is not entirely as important as some of the other metrics, as the class distribution is highly skewed towards non-blue-tarps. Precision is more important, as it measures the proportion of true positives to all of the positive classifications. High precision, therefore, is important to the allocation of resources. Specificity is the ratio of true negatives to the true negatives and false positive, and so this matters quite a bit in the context of this problem, as we want to be sure that the resources are being allocated to specific sites - in other words, we would not want to send resources towards some areas which do not indeed have blue tarps. The ROC AUC metric is an important metric for our problem, as it allows us to glimpse the effectiveness of the model by looking at the ratio of specificity and sensitivity. Finally, the sensitivity is the ratio of the true positives to true positives and false negatives, which, in the context of our problem, is not so important as we do not want our model to be so sensitive as to divert real resources away towards false negatives.

- That being said, most of the models performed very well, as can be seen by the test performance metrics. There were no models which significantly outperformed, or underperformed, based on our performance metrics and qualitative analysis. We have high confidence in our results especially considering the amount of exploration of models which we undertook. Each model was 10-fold cross-validated, and any tunable parameter was tuned as well. Thus, we have high confidence that our results are an accurate depiction of model performance, and since all models performed relatively well, we have high confidence in our results.

- Based on this confidence, we believe that our work would be effective in terms of helping saving human lives. In particular, our high threshold reduces the false-positive rate - the rate at which non-tarps are classified as tarps. By reducing this quantity, we can reduce the number of misinformed rescuers who may be sent to areas where there are no people to be rescued. This is great, but comes with the trade off in that the true-negative rate is higher. However, this can be remedied in later rescue attempts by lowering the threshold. This allows for an initial efficient use of rescue resources. Therefore, we believe that these well-performing models would be effective in saving human lives by efficiently pointing rescue resources towards those who need it.

- To get to these conclusions, numerous models were run and tweaked. Graph sizes were changed and chunk-headers updated. Each run resulting in hours of knitting time. Some lessons learned when developing this project were; cache early and often, parallelism is your friend, test your code on small data before going big, and be systematic in your code design and updating. An ounce of preparation with system design beforehand is well worth the pound of trade off in time at the end.

Appendix

```
knitr::opts_chunk$set(echo=FALSE)
knitr::opts_chunk$set(cache=TRUE, autodep=TRUE)
knitr::opts_chunk$set(fig.align="center", fig.pos="H")
# Set up Parallel Processing
library(doParallel)

cl <- makePSOCKcluster(parallel::detectCores(logical = FALSE))
registerDoParallel(cl)

# Load Libraries
library(tidyverse)
library(tidymodels)
library(probably)
library(discrim)
library(patchwork)

# Read in Data
haiti_train <- read_csv('https://gedeck.github.io/DS-6030/project/HaitiPixels.csv', show_col_types=FALSE
            mutate(BlueTarp= factor(ifelse(Class=="Blue Tarp", "Yes", "No"),labels=c("No", "Yes")))

holdout <- read_delim('holdout.txt', delim='\t') %>%
                mutate(BlueTarp = factor(BlueTarp)) %>%
                rename(Red = V8,
                       Green = V9,
                       Blue = V10)

# View Average Color of Each Class
haiti_train %>%
  group_by(Class) %>%
    summarize(R = mean(Red),
              G = mean(Green),
              B = mean(Blue))%>%
    mutate(hex = rgb(R, G, B, maxColorValue = 255))%>%
    ggplot(aes(x = 1, y = 1, fill = hex)) +
  geom_tile() +
  scale_fill_identity() +
  theme_void() +
  facet_wrap(~Class, ncol=5)
# Show different classifications in data set
haiti_train %>%
  group_by(Class) %>%
  summarize(count=n()) %>%
  mutate(percent_of_total = sprintf('%.1f%%', count / sum(count) * 100)) %>%
    ggplot(aes(x = reorder(Class, -count), y = count, fill=Class)) +
    geom_bar(stat = "identity") +
    scale_fill_manual(values=c('#A9BACD', '#C3B7A2' , '#F7E3B0','#B8A88C', '#4E4E3C')) +
    geom_text(aes(label = percent_of_total), vjust = -0.5, size = 3) +
    labs(x = "", y = "Instances") +
    theme_minimal()+
    guides(fill = "none")
```

```r
library(scatterplot3d)

colors <- c("#999999", "#56B4E9")
colors <- colors[as.numeric(haiti_train$BlueTarp)]

scatterplot3d(haiti_train %>% select(c(Red, Green, Blue)),
              color=colors,
              angle=55,
              main="RGB Groupings")


# Set seed
set.seed(81718)

# Create training data set
train_data <- haiti_train

# Create test data set
test_data <- holdout

# Set up 10-fold cross-validation
resamples <- vfold_cv(train_data, v=10, strata=BlueTarp)

# Set settings for control resamples
cv_control <- control_resamples(save_pred=TRUE)

# Define performance metrics
performance_metrics <- metric_set(specificity, sensitivity, accuracy, precision)


get_ROC_plot <- function(model, train_data, test_data, model_name){
  # Augment train and test data with predicted probabilities
  roc_train <- augment(model, train_data) %>%
    roc_curve(truth = BlueTarp, .pred_Yes, event_level = "second") %>%
    mutate(Dataset = "Train")

  roc_test <- augment(model, test_data) %>%
    roc_curve(truth = BlueTarp, .pred_Yes, event_level = "second") %>%
    mutate(Dataset = "Test")

  # Combine train and test ROC curve data
  roc_data <- bind_rows(roc_train, roc_test)

    # Plot ROC curves for train and test data with different colors
  autoplot(roc_data) +
    geom_line(aes(x = 1 - specificity, y = sensitivity, color = Dataset))+
    labs(title = model_name, x = "False Positive Rate", y = "True Positive Rate")+
    theme(plot.title = element_text(hjust = 0.5))
}

# Create Function to Visual Train Metrics
visualize_training <- function(fit_resample_results, title){
```

```
    aggregate_metrics <-  bind_rows(fit_resample_results$.metrics) %>%
          group_by(.metric) %>%
          summarize(Mean = mean(.estimate),
                    std_err = sd(.estimate) / sqrt(n()))%>%
          rename(Metric=.metric)

    aggregate_metrics %>%
      ggplot(aes(x=Mean, y=Metric, xmin=Mean-std_err, xmax=Mean+std_err)) +
      geom_point() +
      geom_linerange() +
      ggtitle(title) +
      theme(plot.title = element_text(hjust = 0.5))
}

# Create Function to visualize distributions
distribution_graph <- function(model, data, model_name) {
                           model %>%
                                augment(data) %>%
                                   ggplot(aes(x=.pred_Yes, color=BlueTarp)) +
                                   geom_density(bw=0.07) +
                                   labs(x='p(BlueTarp)', title=model_name) +
                                   theme(plot.title = element_text(hjust = 0.5))
}

# Test Thresholds
performance_func_1 <- function(model, data){
                     threshold_perf(model %>% augment(train_data),
                                    BlueTarp,
                                    .pred_Yes,
                                    thresholds = seq(0.01, 0.85, 0.01), event_level="second",
                                    metrics=performance_metrics)
}

# Pick best precision as Threshold
max_precision <- function(performance_data){
          performance_data %>%
              filter(.metric == 'precision') %>%
              filter(.estimate == max(.estimate))
}


# Create Formula
formula <- BlueTarp ~ Red + Green + Blue

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

# Create Log Model
logreg_model <- logistic_reg() %>%
    set_engine("glm") %>%
    set_mode("classification")
```

```r
# define and execute the cross-validation workflow
logreg_wf <- workflow() %>%
    add_model(logreg_model) %>%
    add_recipe(rec)

# Cross Validate Model
logreg_fit_cv <- logreg_wf %>%
                    fit_resamples(resamples=resamples, control=cv_control, metrics=performance_metrics)

# Visualize logreg Fit
logreg_cv_viz <- visualize_training(logreg_fit_cv, "Logreg Cross Validation Results")
logreg_cv_viz

# Fit Model
logreg_model_fit <- logreg_wf %>% fit(train_data)

# Get Performance Thresholds
logreg_threshold_performance <- performance_func_1(logreg_model_fit)

# Run Model on Test Data
logreg_results <- logreg_model_fit %>% augment(test_data)

# Change Pred Class metric based on threshold testing
logreg_results$.threshold_pred_class <- as.factor(ifelse(logreg_results$.pred_Yes >= max_precision(logre

# View results before and after threshold picking
performance_table <-  performance_metrics(logreg_results, truth=BlueTarp, estimate=.threshold_pred_class
                                bind_rows(roc_auc(logreg_results, truth=BlueTarp, .pred_Yes, event_le
                                mutate(Threshold = max_precision(logreg_threshold_performance)$.thresh
                                  dplyr::select(c(Threshold, .metric, .estimate)) %>%
                                    pivot_wider(names_from = .metric, values_from = .estimate, id_col
                                    mutate(model="logreg")


get_ROC_plot(logreg_model_fit, train_data, test_data, "Logistic Regression Model") +
              coord_cartesian(xlim=c(0, 0.25), ylim=c(0.75, 1)) +
              guides(colour="none")

# Create Penalized Log Model
elasticnet_spec <- logistic_reg(engine="glmnet", mode="classification",
                                penalty=tune(), mixture=tune())

# Create Workflow
elasticnet_wf <- workflow() %>%
    add_model(elasticnet_spec) %>%
    add_recipe(rec)

# Testing a range of parameter values
parameters <- extract_parameter_set_dials(elasticnet_wf) %>%
    update(
        penalty=penalty(c(-20, -10)),
        mixture=mixture(c(0, 10))
    )
```

```r
# Identify best model based on searching the parameter space
tune_results <- tune_grid(elasticnet_wf,
                          resamples=resamples,
                          grid=grid_random(parameters, size=200),
                          metrics=performance_metrics)

# show_best(tune_results, metric='accuracy', n=1) %>%
#     knitr::kable()
#
# autoplot(tune_results)

# Finalize workflow with best model and fit model
elasticnet_model_tuned <- elasticnet_wf %>%
    finalize_workflow(select_best(tune_results, metric="accuracy")) %>%
  fit(train_data)

# Get preds with test data
elastic_results <- elasticnet_model_tuned %>% augment(test_data)



# Visualize Penalized Log Fit
elastic_cv_viz <- visualize_training(tune_results,
                                "PLR Validation Results")

# Run Model on Test Data
# elastic_model_fit <- elasticnet_wf %>% fit(train_data)



# Create Validation Metric Set
# elasticnet_wf_fit_cv <- elasticnet_model_tuned %>%
#     fit_resamples(resamples, control=control_resamples(save_pred=TRUE),metrics=performance_metrics)


# Fit EN Model
# elasticnet_model_final_fit <- elasticnet_model_tuned %>% fit(train_data)

# Get threshold performance
en_threshold_performance <- performance_func_1(elasticnet_model_tuned)

# Change Pred Class metric based on threshold testing
elastic_results$.threshold_pred_class <- as.factor(ifelse(elastic_results$.pred_Yes >= max_precision(en

# tune_results %>%
#     show_best(metric='accuracy') %>%
#     select(-.config) %>%
#     knitr::kable()
#
# test_predictions <- elasticnet_model_tuned %>%
#   predict(test_data) %>%
#   bind_cols(test_data)
```

```r
#
# test_metrics <- test_predictions %>%
#   performance_metrics(truth = BlueTarp, estimate = .pred_class)

# View results before and after threshold picking
performance_table <-  bind_rows(performance_table,
                                performance_metrics(elastic_results, truth=BlueTarp, estimate=.threshold
                                  bind_rows(roc_auc(elastic_results, truth=BlueTarp, .pred_Yes, event_l
                                  mutate(Threshold = max_precision(en_threshold_performance)$.threshold
                                  dplyr::select(c(Threshold,
                                                  .metric, .estimate)) %>%
                                      pivot_wider(names_from = .metric, values_from = .estimate, id_col
                                      mutate(model="Penalized LR")
                                )

elastic_cv_viz


get_ROC_plot(elasticnet_model_tuned, train_data, test_data, "Penalized Logistic Regression Model") +
              coord_cartesian(xlim=c(0, 0.25), ylim=c(0.75, 1)) +
              guides(colour="none")


autoplot(tune_results) +
  ggtitle('Elastic Net Tuning Results')+
   theme(plot.title = element_text(hjust = 0.5))

# Create LDA Model
lda_model <- discrim_linear(mode="classification") %>%
               set_engine("MASS")

# Create Workflow
lda_wf <- workflow()%>%
            add_model(lda_model)%>%
            add_recipe(rec)

# Create Validation Metric Set
lda_wf_fit_cv <- lda_wf %>%
                    fit_resamples(resamples=resamples, control=cv_control, metrics=performance_metrics

# Visualize CV Results Fit
lda_cv_viz <- visualize_training(lda_wf_fit_cv, "LDA Validation Results")

# Fit LDA Model
lda_model_fit <- lda_wf %>% fit(train_data)

# Run Model on Test Data
lda_results <- lda_model_fit %>% augment(test_data)

# Get Performance Thresholds
lda_threshold_performance <- performance_func_1(lda_model_fit)

# Change Pred Class metric based on threshold testing
```

```r
lda_results$.threshold_pred_class <- as.factor(ifelse(lda_results$.pred_Yes >= max_precision(lda_thresh

# View results before and after threshold picking
performance_table <-  bind_rows(performance_table,
                                performance_metrics(lda_results, truth=BlueTarp, estimate=.threshold_pr
                                  bind_rows(roc_auc(lda_results, truth=BlueTarp, .pred_Yes, event_level
                                  mutate(Threshold = max_precision(lda_threshold_performance)$.threshol
                                  dplyr::select(c(Threshold, .metric, .estimate)) %>%
                                      pivot_wider(names_from = .metric, values_from = .estimate, id_col
                                      mutate(model="LDA")
                                )

# Create QDA Model
qda_model <- discrim_quad(mode="classification") %>%
               set_engine("MASS")

# Create Workflow
qda_wf <- workflow()%>%
            add_model(qda_model)%>%
            add_recipe(rec)

# Create Validation Metric Set
qda_wf_fit_cv <- qda_wf %>%
                    fit_resamples(resamples=resamples, control=cv_control, metrics=performance_metrics

# Visualize QDA Fit
qda_cv_viz <- visualize_training(qda_wf_fit_cv, "QDA Validation Results")

# Fit QDA Model
qda_model_fit <- qda_wf %>% fit(train_data)

# Run Model on Test Data
qda_results <- qda_model_fit %>% augment(test_data)

# Get Performance Thresholds
qda_threshold_performance <- performance_func_1(qda_model_fit)

# Change Pred Class metric based on threshold testing
qda_results$.threshold_pred_class <- as.factor(ifelse(qda_results$.pred_Yes >= max_precision(qda_thresh


# View results before and after threshold picking
performance_table <-  bind_rows(performance_table,
                                performance_metrics(qda_results, truth=BlueTarp, estimate=.threshold_pr
                                  bind_rows(roc_auc(qda_results, truth=BlueTarp, .pred_Yes, event_level
                                  mutate(Threshold = max_precision(qda_threshold_performance)$.threshol
                                  dplyr::select(c(Threshold, .metric, .estimate)) %>%
                                      pivot_wider(names_from = .metric, values_from = .estimate, id_col
                                      mutate(model="QDA")
                                )

# performance_table %>%
#       knitr::kable(digits=4, caption='Test Performance Metrics')
```

```
lda_cv_viz

get_ROC_plot(lda_model_fit, train_data, test_data, "LDA Model") +
              coord_cartesian(xlim=c(0, 0.25), ylim=c(0.75, 1)) +
              guides(colour="none")

qda_cv_viz

 get_ROC_plot(qda_model_fit, train_data, test_data, "QDA Model") +
              coord_cartesian(xlim=c(0, 0.25), ylim=c(0.75, 1)) +
              guides(colour="none")


# Create KNN Model
knn_model <- nearest_neighbor(neighbors=tune()) %>%
    set_mode("classification") %>%
    set_engine("kknn")


# Create Workflow
knn_wf <- workflow() %>%
    add_model(knn_model) %>%
    add_recipe(rec)

# Set neighbor range
knn_model_params <- extract_parameter_set_dials(knn_wf) %>%
  update(neighbors=neighbors(range=c(5, 50)))

# Tune neighbors param
knn_tune_results <- tune_grid(knn_wf, resamples=resamples,
                              control=cv_control,
                              grid=grid_random(knn_model_params, size=22))

# show_best(knn_tune_results, metric="accuracy", n=1)
# show_best(knn_tune_results, metric="roc_auc", n=1)

# Tune the model based on the best grid search result
knn_tuned_model <- knn_wf %>% finalize_workflow(select_best(knn_tune_results, metric="roc_auc"))

# Create Validation Metric Set
knn_wf_fit_cv <- knn_tuned_model %>%
    fit_resamples(resamples, control=control_resamples(save_pred=TRUE),metrics=performance_metrics)


# Visualize KNN Fit
knn_cv_viz <- visualize_training(knn_wf_fit_cv, "KNN Validation Results")


# Fit KNN Model
knn_model_final_fit <- knn_tuned_model %>% fit(train_data)

# Run Model on Test Data
knn_results <- knn_model_final_fit %>% augment(test_data)
```

```r
# Get Performance Thresholds
knn_threshold_performance <- performance_func_1(knn_model_final_fit)

# Change Pred Class metric based on threshold testing
knn_results$.threshold_pred_class <- as.factor(ifelse(knn_results$.pred_Yes >= max_precision(knn_thresh

# View results before and after threshold picking
performance_table <-  bind_rows(performance_table,
                                performance_metrics(knn_results, truth=BlueTarp, estimate=.threshold_pr
                                  bind_rows(roc_auc(knn_results, truth=BlueTarp, .pred_Yes, event_level
                                  mutate(Threshold = max_precision(knn_threshold_performance)$.threshol
                                  dplyr::select(c(Threshold,
                                                  .metric, .estimate)) %>%
                                      pivot_wider(names_from = .metric, values_from = .estimate, id_col
                                      mutate(model="KNN")
                                )
#
# performance_table %>%
#       knitr::kable(digits=4, caption='Test Performance Metrics')

knn_cv_viz


get_ROC_plot(knn_model_final_fit, train_data, test_data, "KNN Model") +
              coord_cartesian(xlim=c(0, 0.25), ylim=c(0.75, 1)) +
              guides(colour="none")

autoplot(knn_tune_results) +
  ggtitle('KNN Tuning Results')+
   theme(plot.title = element_text(hjust = 0.5))

# Create RF Model
rf_spec <- rand_forest(engine="ranger",
                       mode="classification",
                       mtry=tune(),
                       trees=tune(),
                       min_n=tune())

# Create Workflow
rf_wf <- workflow() %>%
    add_model(rf_spec) %>%
    add_recipe(rec)

# Testing a range of parameter values
parameters <- extract_parameter_set_dials(rf_wf) %>%
    update(
        mtry=mtry(c(1, 3)),
        trees=trees(c(250, 500)),
        min_n=min_n(c(2, 10))
    )
```

```r
# Identify best model based on searching the parameter space
tune_results <- tune_grid(rf_wf,
                          resamples=resamples,
                          grid=grid_random(parameters, size=150),
                          metrics=performance_metrics)

show_best(tune_results, metric='accuracy', n=1) %>%
    select(c(mtry, trees, min_n, .metric, mean)) %>%
    knitr::kable(caption="Best RF Model")



# Finalize workflow with best model and fit model
rf_model_tuned <- rf_wf %>%
    finalize_workflow(select_best(tune_results, metric="accuracy")) %>%
  fit(train_data)

# Get preds with test data
rf_results <- rf_model_tuned %>% augment(test_data)

# Get Performance Thresholds
rf_threshold_performance <- performance_func_1(rf_model_tuned)

# Change Pred Class metric based on threshold testing
rf_results$.threshold_pred_class <- as.factor(ifelse(rf_results$.pred_Yes >= max_precision(rf_threshold_


# View results before and after threshold picking
performance_table <-  bind_rows(performance_table,
                                performance_metrics(rf_results, truth=BlueTarp, estimate=.threshold_pred
                                  bind_rows(roc_auc(rf_results, truth=BlueTarp, .pred_Yes, event_level =
                                  mutate(Threshold = max_precision(rf_threshold_performance)$.threshold
                                  dplyr::select(c(Threshold,
                                                  .metric, .estimate)) %>%
                                      pivot_wider(names_from = .metric, values_from = .estimate, id_cols
                                      mutate(model="Random Forest")
                                )


# Visualize RF Fit
rf_cv_viz <- visualize_training(tune_results,
                                "Random Forest Validation Results")

# tune_results %>%
#     show_best(metric='accuracy') %>%
#     select(-.config) %>%
#     knitr::kable()

test_predictions <- rf_model_tuned %>%
  predict(test_data) %>%
  bind_cols(test_data)
```

```r
test_metrics <- test_predictions %>%
  performance_metrics(truth = BlueTarp, estimate = .pred_class)
```

```r
rf_cv_viz
```

```r
get_ROC_plot(rf_model_tuned, train_data, test_data, "Random Forest Model") +
              coord_cartesian(xlim=c(0, 0.25), ylim=c(0.75, 1)) +
              guides(colour="none")

autoplot(tune_results) +
  ggtitle('Random Forest Tuning Results')+
   theme(plot.title = element_text(hjust = 0.5))
```

```r
# Create Model
linear_svm_model <- svm_linear(mode='classification',
                                    engine='kernlab',
                                    cost=tune(),
                                    margin=tune()
                                    )
# Create workflow
linear_svm_wf <- workflow() %>%
                    add_model(linear_svm_model) %>%
                    add_recipe(rec)
```

```r
# extract tuning parameters
p <- extract_parameter_set_dials(linear_svm_wf)

# Tune the model
linear_svm_tune_grid_results <- tune_bayes(linear_svm_wf,
                               resamples = resamples,
                               control = control_bayes(save_pred=TRUE),
                               param_info=p,
                               iter=10
                               )

# Select best model
best_svm_linear <- select_best(linear_svm_tune_grid_results, metric="roc_auc")

linear_svm_model_tuned <- linear_svm_wf %>%
                              finalize_workflow(best_svm_linear)

# Create Validation Metric Set
linear_svm_wf_fit_cv <- linear_svm_model_tuned %>%
              fit_resamples(resamples, control=cv_control,metrics=performance_metrics)

# Visualize Linear SVM Fit
linear_svm_cv_viz <- visualize_training(linear_svm_wf_fit_cv, "Linear SVM Validation Results")
```

```r
# Fit Linear SVM
linear_svm_final_fit <- linear_svm_model_tuned %>% fit(train_data)

 # Run Model on Test Data
linear_svm_results <- linear_svm_final_fit %>% augment(test_data)

# Get Performance Thresholds
linear_svm_threshold_performance <- performance_func_1(linear_svm_final_fit)

# Change Pred Class metric based on threshold testing
linear_svm_results$.threshold_pred_class <- as.factor(ifelse(linear_svm_results$.pred_Yes >= max_precisi

 # View results before and after threshold picking
performance_table <-  bind_rows(performance_table,
                                performance_metrics(linear_svm_results, truth=BlueTarp, estimate=.thresh
                                  bind_rows(roc_auc(linear_svm_results, truth=BlueTarp, .pred_Yes, event
                                  mutate(Threshold = max_precision(linear_svm_threshold_performance)$.th
                                  dplyr::select(c(Threshold, .metric, .estimate)) %>%
                                      pivot_wider(names_from = .metric, values_from = .estimate, id_cols
                                      mutate(model="Linear SVM")
                                )



linear_svm_cv_viz

get_ROC_plot(linear_svm_final_fit, train_data, test_data, "Linear SVM Model") +
              coord_cartesian(xlim=c(0, 0.25), ylim=c(0.75, 1)) +
              guides(colour="none")


#Visualize Training
autoplot(linear_svm_tune_grid_results) + ggtitle('Default Linear Kernel Tuning')+
   theme(plot.title = element_text(hjust = 0.5))


# Create Model
poly_svm_model <- svm_poly(mode='classification',
                                  engine='kernlab',
                                  cost=tune(),
                                  margin=tune(),
                                  degree=tune(),
                                  scale_factor=tune()
                                  )

# Define Workflow
poly_svm_wf <- workflow() %>%
                add_recipe(rec) %>%
                add_model(poly_svm_model)

# extract tuning parameters
p <- extract_parameter_set_dials(poly_svm_wf)
```

```r
# Tune the model
poly_svm_tune_grid_results <- tune_bayes(poly_svm_wf,
                                resamples = resamples,
                                control = control_bayes(save_pred=TRUE),
                                param_info=p,
                                iter=10
                                )


# Select best model
best_svm_poly <- select_best(poly_svm_tune_grid_results, metric="roc_auc")

poly_svm_model_tuned <- poly_svm_wf %>%
                          finalize_workflow(best_svm_poly)
show_notes(.Last.tune.result)
# Create Validation Metric Set
poly_svm_wf_fit_cv <- poly_svm_model_tuned %>%
              fit_resamples(resamples, control=cv_control, metrics=performance_metrics)

# Visualize Poly SVM Fit
poly_svm_cv_viz <- visualize_training(poly_svm_wf_fit_cv, "Polynomial SVM Validation Results")

# Fit Poly SVM
poly_svm_final_fit <- poly_svm_model_tuned %>% fit(train_data)

 # Run Model on Test Data
poly_svm_results <- poly_svm_final_fit %>% augment(test_data)

# Get Performance Thresholds
poly_svm_threshold_performance <- performance_func_1(poly_svm_final_fit)

# Change Pred Class metric based on threshold testing
poly_svm_results$.threshold_pred_class <- as.factor(ifelse(poly_svm_results$.pred_Yes >= max_precision(

 # View results before and after threshold picking
performance_table <-  bind_rows(performance_table,
                                performance_metrics(poly_svm_results, truth=BlueTarp, estimate=.thresho
                                  bind_rows(roc_auc(poly_svm_results, truth=BlueTarp, .pred_Yes, event_
                                  mutate(Threshold = max_precision(poly_svm_threshold_performance)$.thr
                                  dplyr::select(c(Threshold, .metric, .estimate)) %>%
                                      pivot_wider(names_from = .metric, values_from = .estimate, id_col
                                      mutate(model="Polynomial SVM")
                                )


poly_svm_cv_viz


get_ROC_plot(poly_svm_final_fit, train_data, test_data, "Polynomial SVM Model") +
              coord_cartesian(xlim=c(0, 0.25), ylim=c(0.75, 1)) +
              guides(colour="none")
```

```
# Visualize Training
autoplot(poly_svm_tune_grid_results) + ggtitle('Default Polynomial Kernel Tuning')+
   theme(plot.title = element_text(hjust = 0.5))


# Create Model
rbf_svm_model <- svm_rbf(mode='classification',
                                    engine='kernlab',
                                    cost=tune(),
                                    margin=tune(),
                                    rbf_sigma=tune()
                                    )

# Define Workflow
rbf_svm_wf <- workflow() %>%
                add_recipe(rec) %>%
                add_model(rbf_svm_model)

# extract tuning parameters
p <- extract_parameter_set_dials(rbf_svm_wf)

# Tune the model
rbf_svm_tune_grid_results <- tune_bayes(rbf_svm_wf,
                               resamples = resamples,
                               control = control_bayes(save_pred=TRUE),
                               param_info=p,
                               iter=10
                               )



# Select best model
best_svm_rbf <- select_best(rbf_svm_tune_grid_results, metric="roc_auc")

rbf_svm_model_tuned <- rbf_svm_wf %>%
                           finalize_workflow(best_svm_rbf)

# Create Validation Metric Set
rbf_svm_wf_fit_cv <- rbf_svm_model_tuned %>%
              fit_resamples(resamples, control=cv_control, metrics=performance_metrics)

# Visualize RBF SVM Fit
rbf_svm_cv_viz <- visualize_training(rbf_svm_wf_fit_cv, "RBF SVM Validation Results")

# Fit RBF SVM
rbf_svm_final_fit <- rbf_svm_model_tuned %>% fit(train_data)

 # Run Model on Test Data
rbf_svm_results <- rbf_svm_final_fit %>% augment(test_data)

# Get Performance Thresholds
rbf_svm_threshold_performance <- performance_func_1(rbf_svm_final_fit)
```

```r
# Change Pred Class metric based on threshold testing
rbf_svm_results$.threshold_pred_class <- as.factor(ifelse(rbf_svm_results$.pred_Yes >= max_precision(rbf

 # View results before and after threshold picking
performance_table <-  bind_rows(performance_table,
                                performance_metrics(rbf_svm_results, truth=BlueTarp, estimate=.threshold
                                  bind_rows(roc_auc(rbf_svm_results, truth=BlueTarp, .pred_Yes, event_le
                                  mutate(Threshold = max_precision(rbf_svm_threshold_performance)$.thres
                                  dplyr::select(c(Threshold, .metric, .estimate)) %>%
                                      pivot_wider(names_from = .metric, values_from = .estimate, id_cols
                                      mutate(model="RBF SVM")
                                )

rbf_svm_cv_viz


get_ROC_plot(rbf_svm_final_fit, train_data, test_data, "RBF SVM Model") +
              coord_cartesian(xlim=c(0, 0.25), ylim=c(0.75, 1)) +
              guides(colour="none")


# Visualize Training
autoplot(rbf_svm_tune_grid_results) + ggtitle('Default RBF Kernel Tuning')+
   theme(plot.title = element_text(hjust = 0.5))

# Create Confusion Matrixes to reference TP, FP,TN, FP in discussion of results
logreg_conf_matrix <- conf_mat(logreg_results, estimate=.threshold_pred_class, truth=BlueTarp)
lda_conf_matrix <- conf_mat(lda_results, estimate=.threshold_pred_class, truth=BlueTarp)
qda_conf_matrix <- conf_mat(qda_results, estimate=.threshold_pred_class, truth=BlueTarp)
knn_conf_matrix <- conf_mat(knn_results, estimate=.threshold_pred_class, truth=BlueTarp)
en_conf_matrix <- conf_mat(elastic_results, estimate=.threshold_pred_class, truth=BlueTarp)
lsvm_conf_matrix <- conf_mat(linear_svm_results, estimate=.threshold_pred_class, truth=BlueTarp)
psvm_conf_matrix <- conf_mat(poly_svm_results, estimate=.threshold_pred_class, truth=BlueTarp)
rbfsvm_conf_matrix <- conf_mat(rbf_svm_results, estimate=.threshold_pred_class, truth=BlueTarp)
rf_conf_matrix <- conf_mat(rf_results, estimate=.threshold_pred_class, truth=BlueTarp)


# knn_conf_matrix
# qda_conf_matrix
# paste("log reg")
# logreg_conf_matrix
#
# paste("lda")
# lda_conf_matrix
#
# paste("qda")
# qda_conf_matrix
#
# paste('knn')
# knn_conf_matrix
#
# paste('en')
# en_conf_matrix
```

```
#
# paste("lsvm")
# lsvm_conf_matrix
#
# paste("psvm")
# psvm_conf_matrix
#
# paste("rbfsvm")
# rbfsvm_conf_matrix
#
# paste("random forest")
# rf_conf_matrix

# View results before and after threshold picking
performance_table <-  bind_rows(performance_table,
                                performance_metrics(rf_results, truth=BlueTarp, estimate=.threshold_pred
                                  bind_rows(roc_auc(rf_results, truth=BlueTarp, .pred_Yes, event_level =
                                  mutate(Threshold = max_precision(rf_threshold_performance)$.threshold
                                  dplyr::select(c(Threshold,
                                                  .metric, .estimate)) %>%
                                      pivot_wider(names_from = .metric, values_from = .estimate, id_cols
                                      mutate(model="Random Forest")
                                )

performance_table <-  bind_rows(performance_table,
                                performance_metrics(elastic_results, truth=BlueTarp, estimate=.threshold
                                  bind_rows(roc_auc(elastic_results, truth=BlueTarp, .pred_Yes, event_le
                                  mutate(Threshold = max_precision(en_threshold_performance)$.threshold)
                                  dplyr::select(c(Threshold,
                                                  .metric, .estimate)) %>%
                                      pivot_wider(names_from = .metric, values_from = .estimate, id_cols
                                      mutate(model="Penalized LR")
                                )

performance_table %>%
      knitr::kable(digits=4, caption='Test Performance Metrics')
```