Disaster Relief Project Part 1

Mahin Ganesan, Wyatt Priddy, and Taylor Tucker

2024-04-17

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.

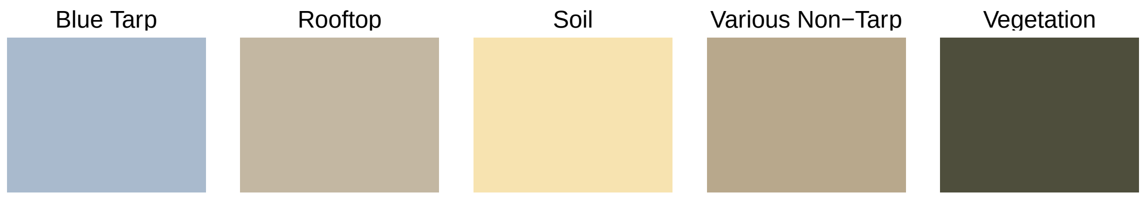


Figure 1: Average Color of Class

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

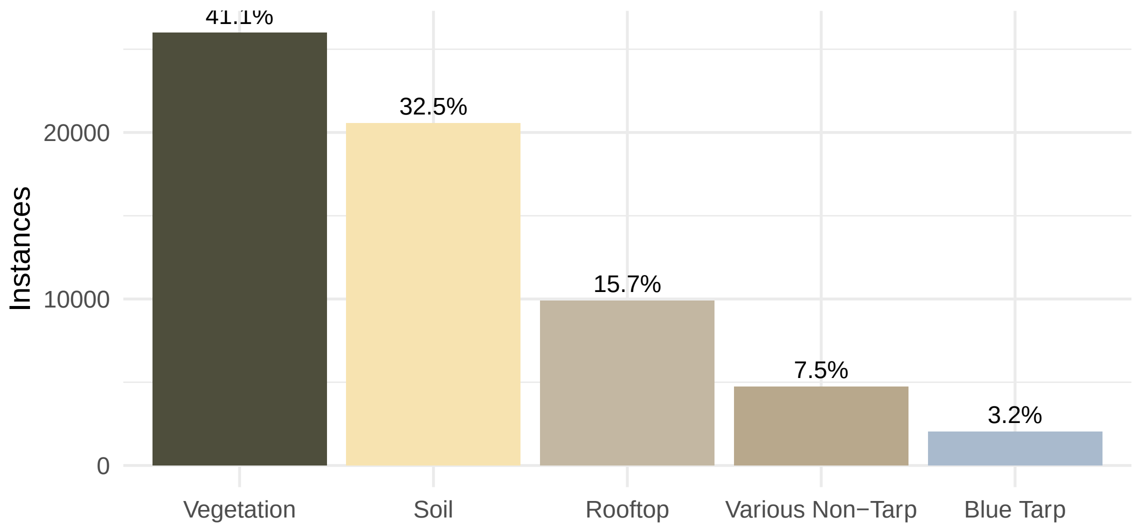


Figure 2: Distribution of Classifications in Training Set

```
## Warning: Unknown or uninitialised column: `color`.
```

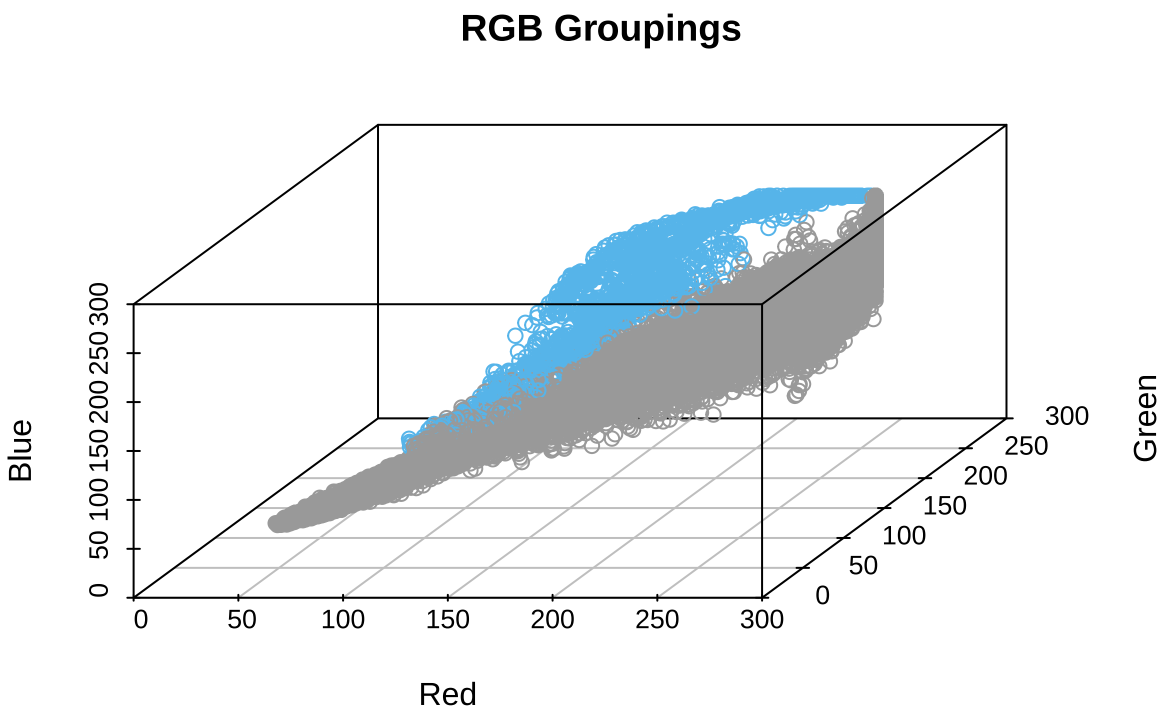


Figure 3: 3d Visualization of Blue Tarps

Description of Methodology

Given that our sample data set only has 3.2% of the target class, we will use stratification to account for the imbalance in our test and train split. We are looking to ensure that there is even representation between our two sets to ensure a robust analysis when training our models and subsequently predicting on the test data. We are using an 80/20 split on the training versus testing data. Since the data set is relatively large and we are using stratification to ensure an even population of blue tarps across the split, leaving 20% of the data for hold-out is appropriate.

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. 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 range 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 automatically using a stepwise function 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. This threshold limit also increased our confidence in the decisions our models made.

Libraries used:

- tidymodels: used for model creation, cross-validation, and determining model performance
- tidyverse: used for plotting, data manipulation, and chaining operations
- probably: used for assessing threshold performance on the models

- discrim: used for the LDA/QDA models
- patchwork: used for combining plots
- 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: $\frac{TP}{TP+FP}$
- ROC AUC

Results

Logistic Regression Model

The logistic model was fit across the 10-fold cross-validation. The model seems to be generalizing well across the different folds with minimal variation across the training performance metrics as seen in *Figure 3.* 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.

After performing threshold analysis, a threshold of 0.84 was selected to minimize the false positives while keeping a substantial portion of the true positives in the analysis. Seen in *Table 1*, Utilizing this threshold resulted in a precision rating of 0.996 meaning that 99.6% 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 87.7% 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 0.84, 342 of the 390 blue tarps in the testing set are correctly identified by the model. Of the 12,259 not blue tarps, only 4 are false positives. Of all the blue tarps identified by the model, there is a 98.8% rate of aid workers time and resources not being wasted on futile searches. While 48 blue tarps were unidentified by the model (12.3% of total tarps), once the aid workers can provide resources to the 342 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.

LDA Model

The LDA Model has similar variations in the cross-validation performance as the logistic regression model seen in *Figure 3.* 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.

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 1.* 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 299 of the 390 blue tarps in the testing set are correctly identified by the model. Of the 12,259 not blue tarps, 88 were false positives. This would represent an increase of 2100% more false positives and could equate to numerous man hours and resources being wasted if we were to go with this model. 91 blue tarps were false negatives also resulting in more pockets of refuges being missed by this model.

QDA Model

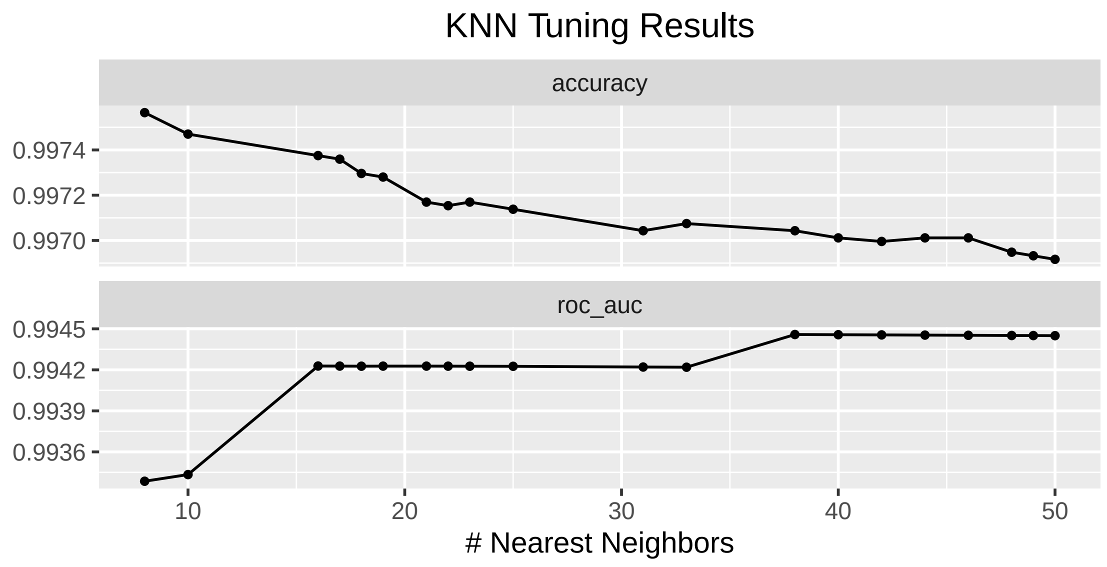
The QDA model performs better than the LDA model for the 10-fold cross-validation but worse than the logistic regression. The main difference between all the models thus far has been the variation in specificity as sensitivity, precision, and accuracy have been comparable across the models. The QDA model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

At the selected threshold of 0.85 on the testing data, the QDA model only has 2 false positives. However, the number of true positives predicted is 312 which results in 30 less pockets of refuges being rescued and the number of false negatives is 78 which is an increase of 62.5% more blue tarps being unidentified compared to the logistic regression model.

K-Nearest Neighbors Model

The KNN model performs better 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.

The KNN model has 5 false positives and 43 false negatives, and finds more true positive blue tarp locations than all of the other models so far, narrowly beating out logistic regression with 5 more true positives, which means 5 more pockets of displaced persons found. 5 more true positives has significant real world significance in this case, since every additional correct pocket found is possibly many lives saved. 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 a favorable option.



We tuned the neighbors parameter to get the optimal number of neighbors that maximizes the roc_auc metric, which ended up being 17. The roc_auc value for this number of neighbors is about 0.994.

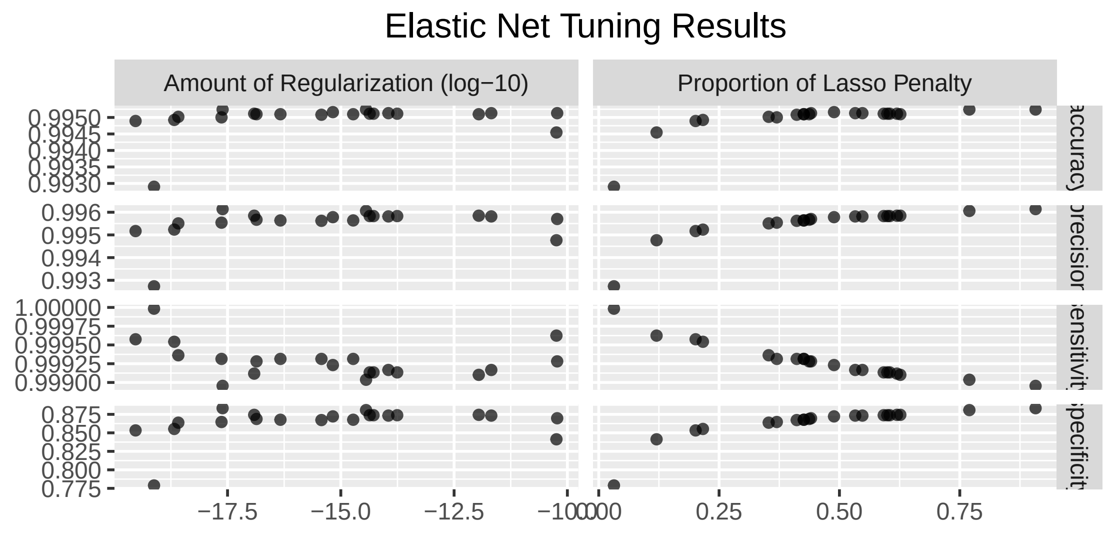
Penalized Logistic Regression Model

The penalized log model using elasticnet (L1-L2 regularization) performs better than LDA and QDA, but worse than 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.

The penalized log model has 11 false positives and 37 false negatives. While the model finds 5 more true positive blue tarp locations than the previous best model, KNN, it also has 6 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.

Table 1: 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	0.5523	0.9980	0.9948	0.9967	0.9641	KNN
0.83	0.9749	0.9974	0.9972	0.9998	0.9995	Penalized LR

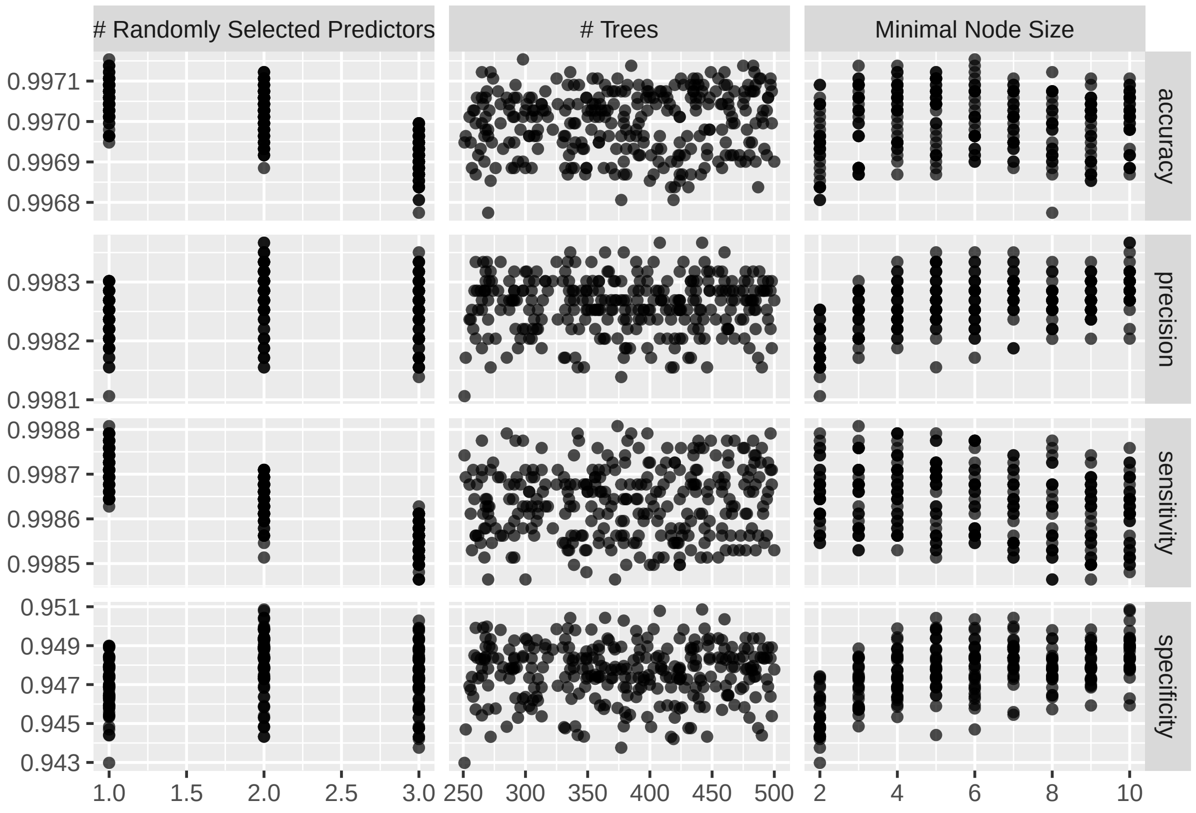


We tuned the penalty and mixture parameter to get the optimal number that maximizes accuracy of the model using a range of (-20,-10) for penalty and (0,10) for mixture.

Random Forest

WRITEUP TODO

mtry	trees	min_n	.metric	.estimator	mean	n	std_err	.config
1	298	6	accuracy	binary	0.9971537	10	0.0002041	Preprocessor1_Model209

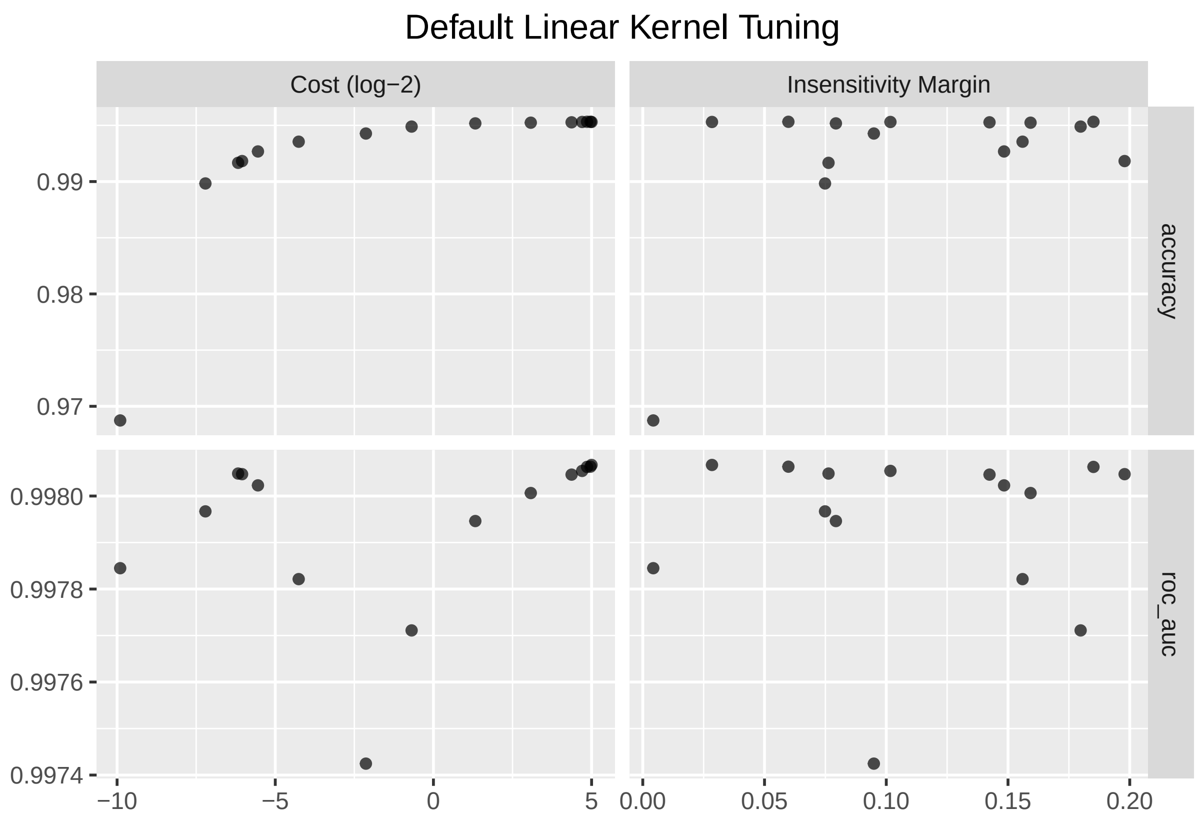


mtry	trees	min_n	.metric	.estimator	mean	n	std_err
1	298	6	accuracy	binary	0.9971537	10	0.0002041
1	475	3	accuracy	binary	0.9971379	10	0.0002242
1	483	6	accuracy	binary	0.9971379	10	0.0002179
1	385	4	accuracy	binary	0.9971379	10	0.0002127
2	336	5	accuracy	binary	0.9971221	10	0.0002145

Linear Support Sector Machine

Random words to fill in later

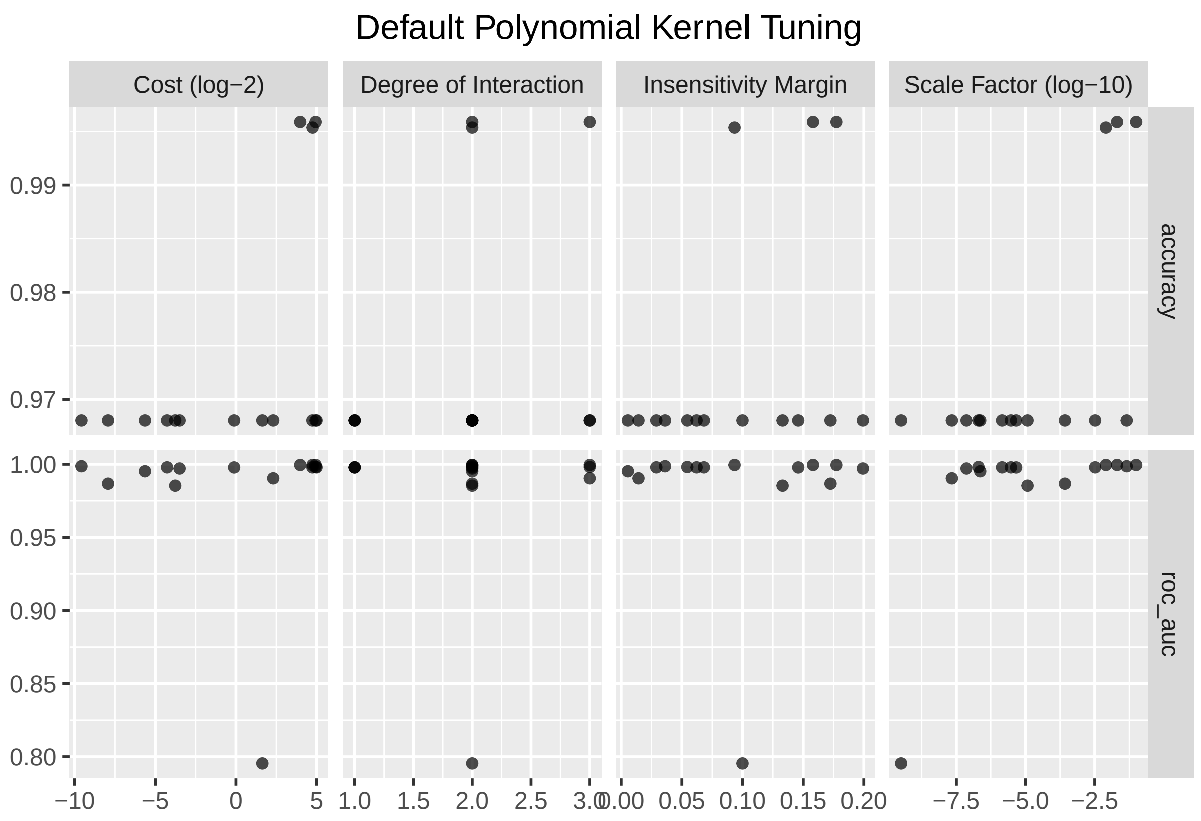
```
##  Setting default kernel parameters
```



Polynomial Support Sector Machine

Random words to fill in later

```
## ! The Gaussian process model is being fit using 4 features but only has 5
##   data points to do so. This may cause errors or a poor model fit.
```



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

Metrics and Graphics

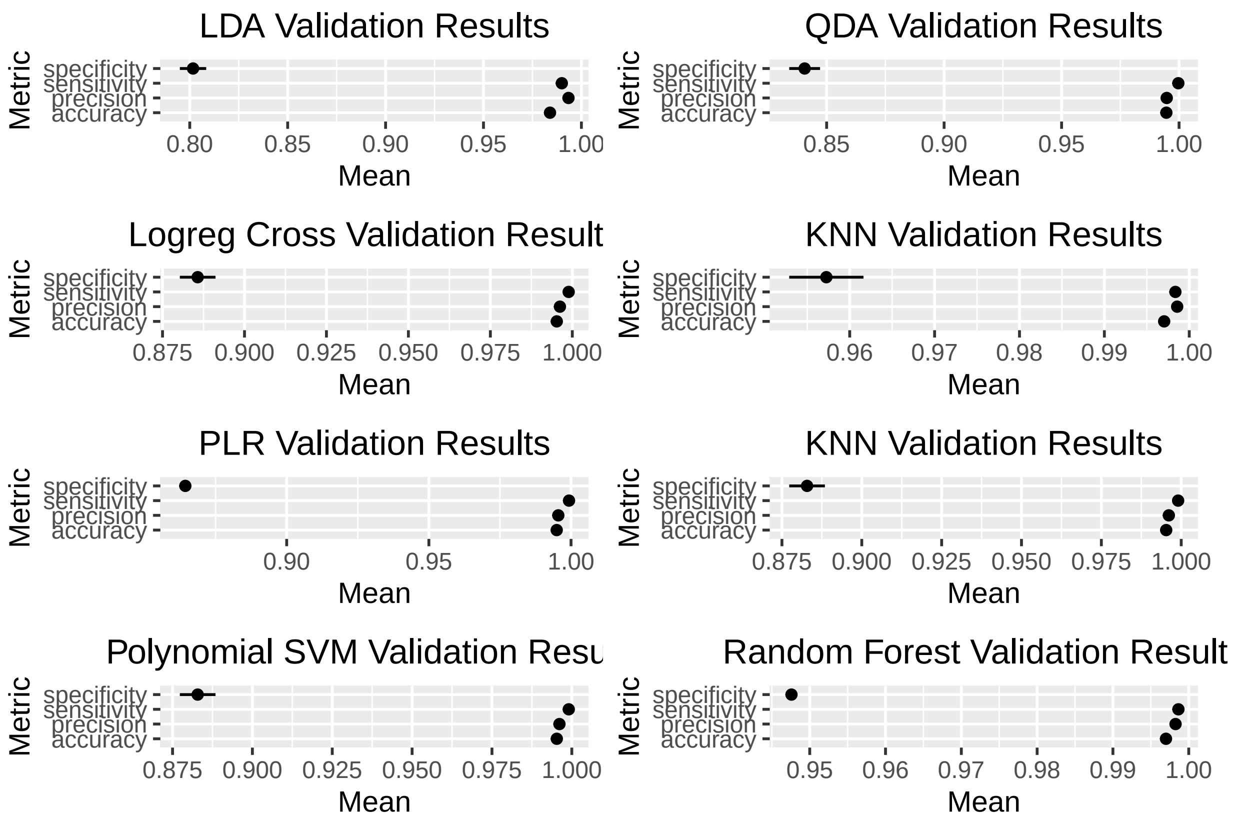


Figure 4: Cross Validation Metrics

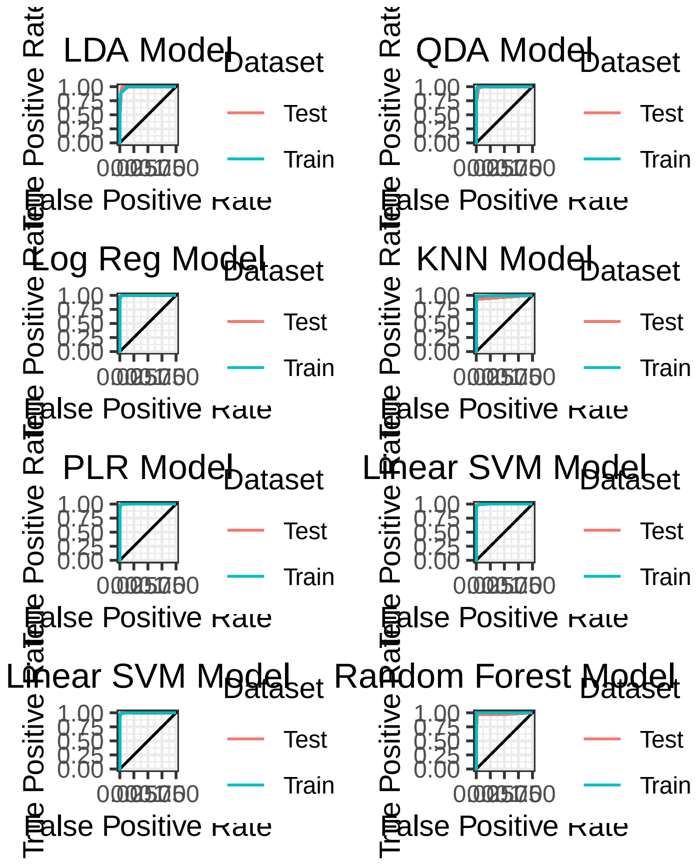


Figure 5: ROC Curves

Conclusion

Based on our analyses, we found that the K-Nearest Neighbors Model performs the best out of the five 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 KNN Model had the highest specificity, accuracy, and precision out of all the models we tried, with values of 0.8923, 0.9963, and 0.9966, respectively. This model incorrectly classified 5 non-tarps as tarps, and incorrectly classified 42 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 5 non-tarps which were classified as tarps was relatively low, which would allow rescue teams to avoid performing operations at non-tarp locations.

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.

One additional action our team could take to improve model results is to use ensemble methods. Two types of ensemble methods are bagging and boosting, and these methods would possibly improve our results by combining multiple model predictions rather that what we are currently doing of looking at individual models. Doing this could reduce variance and improve accuracy and other performance metrics. To do this, we would use stacking, where a meta-model would be trained on the predictions of multiple base models that we felt performed the best, such as KNN and logistic regression, and combine the predictions of these models to make a final prediction.

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")

# 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_cols
                                      mutate(model="logreg")


# 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
```

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

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


# View results before and after threshold picking
performance_table <-  bind_rows(performance_table,
                                performance_metrics(qda_results, truth=BlueTarp, estimate=.threshold_pre
                                  bind_rows(roc_auc(qda_results, truth=BlueTarp, .pred_Yes, event_level
                                  mutate(Threshold = max_precision(qda_threshold_performance)$.threshold
                                  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')

# 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)


# 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')

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


# 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))
    )


# 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")) %>%
```

```r
  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)
#
# 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=.threshol
                                  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")
                                )
```

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

autoplot(tune_results) +
  ggtitle('Elastic Net 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))
    )



# Identify best model based on searching the parameter space
tune_results <- tune_grid(rf_wf,
                          resamples=resamples,
                          grid=grid_random(parameters, size=300),
                          metrics=performance_metrics)

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

autoplot(tune_results)

# 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)



# Visualize Penalized Log 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)

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

# 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)


# 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, "KNN Validation Results")

# 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)
```

```r
# 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")
                                )


#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)

# 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
                               )

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

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

```r
poly_svm_model_tuned <- linear_svm_wf %>%
                          finalize_workflow(best_svm_poly)

# 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 Linear 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=.threshol
                                  bind_rows(roc_auc(poly_svm_results, truth=BlueTarp, .pred_Yes, event_
                                  mutate(Threshold = max_precision(linear_svm_threshold_performance)$.t
                                  dplyr::select(c(Threshold, .metric, .estimate)) %>%
                                      pivot_wider(names_from = .metric, values_from = .estimate, id_col
                                      mutate(model="Polynomial SVM")
                                )



# Visualize Cross validation Metrics
lda_cv_viz + qda_cv_viz + logreg_cv_viz + knn_cv_viz + elastic_cv_viz + linear_svm_cv_viz + poly_svm_cv

log_roc <- get_ROC_plot(logreg_model_fit, train_data, test_data, "Log Reg Model")
lda_roc <- get_ROC_plot(lda_model_fit, train_data, test_data, "LDA Model")
qda_roc <- get_ROC_plot(qda_model_fit, train_data, test_data, "QDA Model")
knn_roc <- get_ROC_plot(knn_model_final_fit, train_data, test_data, "KNN Model")
elastic_roc <- get_ROC_plot(elasticnet_model_tuned, train_data, test_data, "PLR Model")
linear_svm_roc <- get_ROC_plot(linear_svm_final_fit, train_data, test_data, "Linear SVM Model")
poly_svm_roc <- get_ROC_plot(poly_svm_final_fit, train_data, test_data, "Linear SVM Model")
rf_roc <- get_ROC_plot(rf_model_tuned, train_data, test_data, "Random Forest Model")

# Visualize ROC plots
lda_roc + qda_roc + log_roc + knn_roc + elastic_roc + linear_svm_roc + poly_svm_roc + rf_roc + plot_lay


# 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)
```

```r
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=.pred_class, truth=BlueTarp)
lsvm_conf_matrix <- conf_mat(linear_svm_results, estimate=.pred_class, truth=BlueTarp)
psvm_conf_matrix <- conf_mat(poly_svm_results, estimate=.pred_class, truth=BlueTarp)
rf_conf_matrix <- conf_mat(rf_results, estimate=.pred_class, truth=BlueTarp)


# knn_conf_matrix
# qda_conf_matrix
```