Lab6

Sujan Bhattarai

2023-03-01

Case Study: Eel Distribution Modeling

This week's lab follows a project modeling the eel species Anguilla australis described by Elith et al. (2008). There are two data sets for this lab. You'll use one for training and evaluating your model, and you'll use your model to make predictions predictions on the other. Then you'll compare your model's performance to the model used by Elith et al.

Data

Grab the training and evaluation data sets (eel.model.data.csv, eel.eval.data.csv) from github here: https://github.com/MaRo406/eds-232-machine-learning/blob/main/data

```
#load all libraries requiref for xgboost
library(tidymodels)
library(xgboost)
library(tictoc)
library(vip)
library(readr)
library(dplyr)
library(purrr)
library(ggplot2)
library(ranger)
library(yardstick)
library(workflows)
library(recipes)
library(modeldata)
library(parsnip)
library(dials)
```

```
#download the data from github using direct github link
training_data <- read_csv("https://raw.githubusercontent.com/MaRo406/eds-232-machine-learning/main/data/
evaluation_data <- read_csv("https://raw.githubusercontent.com/MaRo406/eds-232-machine-learning/main/dat
```

```
#take a look at the data, look at na values and nuermic, categorical columns
glimpse(training_data)
```

```
## Rows: 1,000
## Columns: 14
## $ Site       <dbl> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, ~
## $ Angaus     <dbl> 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0,~
## $ SegSumT    <dbl> 16.0, 18.7, 18.3, 16.7, 17.2, 15.1, 12.7, 18.1, 18.9, 18.2,~
## $ SegTSeas   <dbl> -0.10, 1.51, 0.37, -3.80, 0.33, 1.83, 2.17, 1.00, 1.59, 0.7~
## $ SegLowFlow <dbl> 1.036, 1.003, 1.001, 1.000, 1.005, 1.015, 1.001, 1.002, 1.0~
## $ DSDist     <dbl> 50.2000, 132.5300, 107.4400, 166.8200, 3.9500, 11.1700, 42.~
```

```
## $ DSMaxSlope <dbl> 0.57, 1.15, 0.57, 1.72, 1.15, 1.72, 2.86, 2.29, 0.40, 3.43,~
## $ USAvgT     <dbl> 0.09, 0.20, 0.49, 0.90, -1.20, -0.20, 1.45, 0.47, 0.25, 0.0~
## $ USRainDays <dbl> 2.470, 1.153, 0.847, 0.210, 1.980, 3.300, 0.430, 1.153, 0.8~
## $ USSlope    <dbl> 9.8, 8.3, 0.4, 0.4, 21.9, 25.7, 9.6, 4.9, 9.8, 20.5, 3.9, 6~
## $ USNative   <dbl> 0.81, 0.34, 0.00, 0.22, 0.96, 1.00, 0.09, 0.02, 0.74, 0.92,~
## $ DSDam      <dbl> 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0,~
## $ Method     <chr> "electric", "electric", "spo", "electric", "electric", "ele~
## $ LocSed     <dbl> 4.8, 2.0, 1.0, 4.0, 4.7, 4.5, 4.3, NA, NA, 3.6, 3.7, 1.0, 3~
```

```r
#conver the Angaus column to a factor
training_data$Angaus <- as.factor(training_data$Angaus)
```

Preprocess

Create a recipe to prepare your data for the XGBoost model

```r
# Create a recipe to prepare your data for the XGBoost model
eel_recipe <- recipe(Angaus ~ ., data = training_data) %>%
  step_naomit(all_predictors()) %>%
  step_dummy(all_nominal(), -all_outcomes()) %>%
  step_center(all_predictors(), -all_outcomes()) %>%
  step_scale(all_predictors(), -all_outcomes())
```

Split and Resample

Split the model data (eel.model.data.csv) into a training and test set, stratified by outcome score (Angaus). Use 10-fold CV to resample the training set.

```r
#split to 80-20 train and test set
set.seed(123)
eel_split <- initial_split(training_data, prop = 0.8, strata = Angaus)
train <- training(eel_split)
test <-  testing(eel_split)
```

```r
#use 10 fold cross validation to resample the training set
cv_set <- vfold_cv(train, v = 10, strata = Angaus)
```

Tuning XGBoost

Tune Learning Rate

Following the XGBoost tuning strategy outlined in lecture, first we conduct tuning on just the learning rate parameter:

1. Create a model specification using {xgboost} for the estimation

- Only specify one parameter to tune()

```r
#Instantiate the xgboost model
xgboost_spec <- boost_tree(
  learn_rate = tune())%>%
  set_engine("xgboost") %>%
  set_mode("classification")
```

2. Set up a grid to tune your model by using a range of learning rate parameter values: expand.grid(learn_rate = seq(0.0001, 0.3, length.out = 30))

```r
#Set up a grid to tune your model by using a range of learning rate parameter values\
learn_rate_grid <- expand.grid(learn_rate = seq(0.0001, 0.3, length.out = 30))
```

- Use appropriate metrics argument(s) - Computational efficiency becomes a factor as models get more complex and data get larger. Record the time it takes to run. Do this for each tuning phase you run.You could use {tictoc} or Sys.time().

```
# tune the model with the learning rate grid
Sys.time()
```

```
## [1] "2024-03-06 20:13:13 PST"
```

```
learn_rate_tune <- tune_grid(
  xgboost_spec,
  eel_recipe,
  resamples = cv_set,
  grid = learn_rate_grid,
  metrics = metric_set(roc_auc),
  control = control_grid(save_pred = TRUE)
)

Sys.time()
```

```
## [1] "2024-03-06 20:13:42 PST"
```

3. Show the performance of the best models and the estimates for the learning rate parameter values associated with each.

```
#select the best model
best_learn_rate <- learn_rate_tune %>%
  select_best(metric = "roc_auc")

best_learn_rate
```

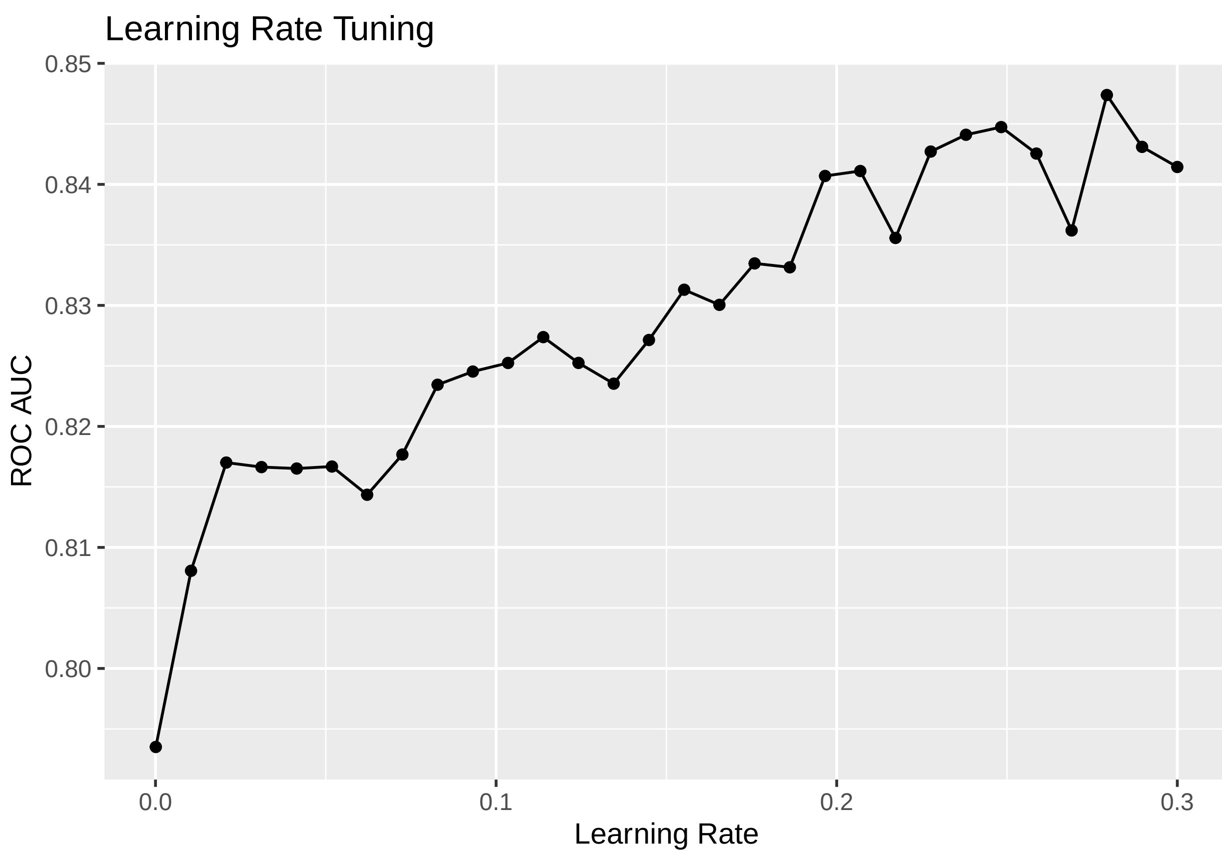
```
## # A tibble: 1 x 2
##   learn_rate .config
##        <dbl> <chr>
## 1      0.279 Preprocessor1_Model28
```

```
#show estimates in tables
learn_rate_tune %>%
  collect_metrics() %>%
  filter(.metric == "roc_auc")
```

```
## # A tibble: 30 x 7
##    learn_rate .metric .estimator  mean     n std_err .config
##         <dbl> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
##  1     0.0001 roc_auc binary     0.794    10  0.0198 Preprocessor1_Model01
##  2     0.0104 roc_auc binary     0.808    10  0.0170 Preprocessor1_Model02
##  3     0.0208 roc_auc binary     0.817    10  0.0177 Preprocessor1_Model03
##  4     0.0311 roc_auc binary     0.817    10  0.0171 Preprocessor1_Model04
##  5     0.0415 roc_auc binary     0.817    10  0.0168 Preprocessor1_Model05
##  6     0.0518 roc_auc binary     0.817    10  0.0151 Preprocessor1_Model06
##  7     0.0621 roc_auc binary     0.814    10  0.0161 Preprocessor1_Model07
##  8     0.0725 roc_auc binary     0.818    10  0.0154 Preprocessor1_Model08
##  9     0.0828 roc_auc binary     0.823    10  0.0135 Preprocessor1_Model09
## 10     0.0932 roc_auc binary     0.825    10  0.0109 Preprocessor1_Model10
## # i 20 more rows
```

```
# show the performance for each learning rate with the best model based on estimate vlaue
learn_rate_tune %>%
```

```r
  collect_metrics() %>%
  filter(.metric == "roc_auc") %>%
  ggplot(aes(x = learn_rate, y = mean)) +
  geom_point() +
  geom_line() +
  labs(title = "Learning Rate Tuning",
       x = "Learning Rate",
       y = "ROC AUC")
```



Tune Tree Parameters

1. Create a new specification where you set the learning rate (which you already optimized) and tune the tree parameters.

```r
# tune only tree parameters
xgboost_tree_spec <- boost_tree(
  trees = tune(),
  learn_rate = best_learn_rate[[1]]) %>%
  set_engine("xgboost") %>%
  set_mode("classification")
```

2. Set up a tuning grid. This time use grid_latin_hypercube() to get a representative sampling of the parameter space

```r
#Set up a tuning grid. This time use grid_latin_hypercube() to get a representative sampling of the par
tree_grid <- grid_latin_hypercube(
  trees() %>% finalize(mtry = NULL),
```

```
  size = 20)
```

3. Show the performance of the best models and the estimates for the tree parameter values associated with each.

```
#use the tuning grid to tune the tree parameters
Sys.time()
```

```
## [1] "2024-03-06 20:23:23 PST"
```

```
tree_tune <- tune_grid(
  xgboost_tree_spec,
  eel_recipe,
  resamples = cv_set,
  grid = tree_grid,
  metrics = metric_set(roc_auc),
  control = control_grid(save_pred = TRUE)
)
Sys.time()
```

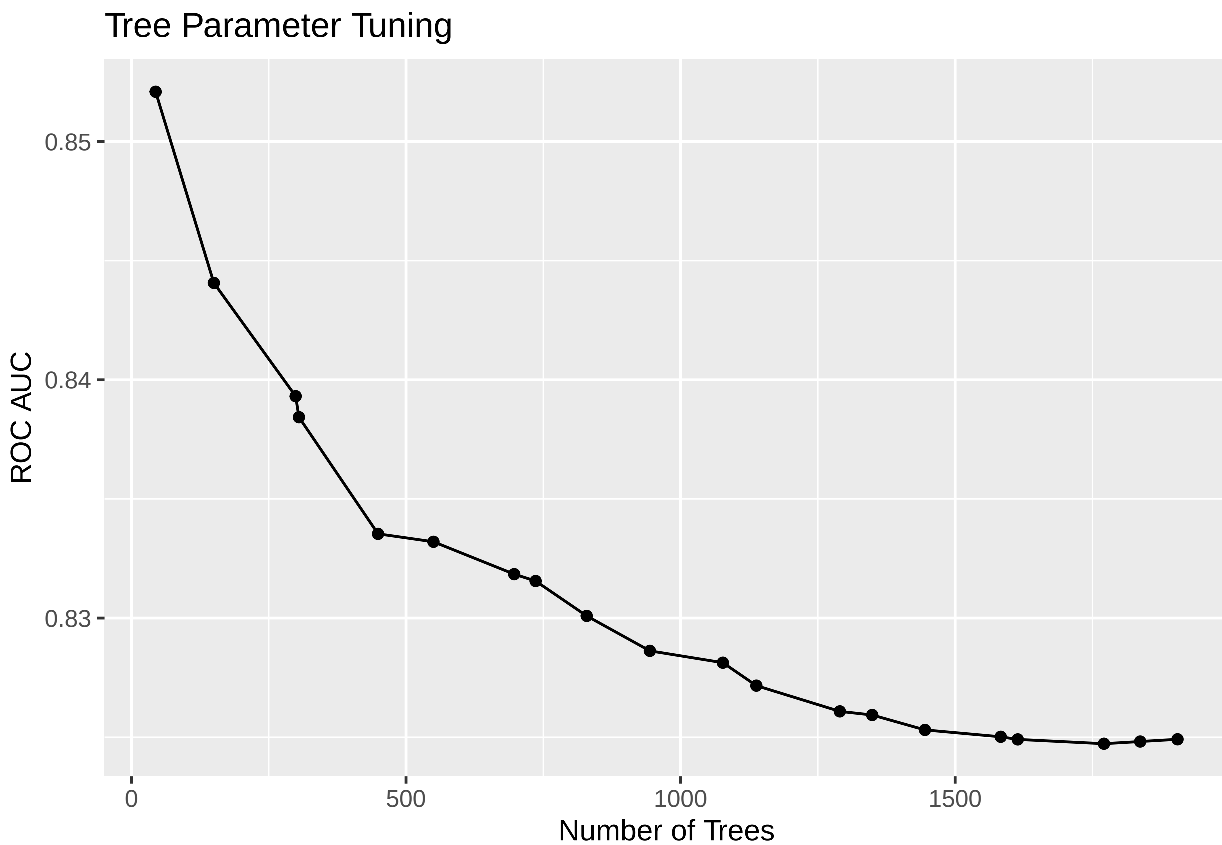
```
## [1] "2024-03-06 20:23:41 PST"
```

```
#show estimates in table
tree_tune %>%
  collect_metrics() %>%
  filter(.metric == "roc_auc")
```

```
## # A tibble: 20 x 7
##    trees .metric .estimator  mean     n std_err .config
##    <int> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
##  1    44 roc_auc binary     0.852    10 0.00998 Preprocessor1_Model01
##  2   150 roc_auc binary     0.844    10 0.0137  Preprocessor1_Model02
##  3   299 roc_auc binary     0.839    10 0.0145  Preprocessor1_Model03
##  4   305 roc_auc binary     0.838    10 0.0145  Preprocessor1_Model04
##  5   449 roc_auc binary     0.834    10 0.0160  Preprocessor1_Model05
##  6   550 roc_auc binary     0.833    10 0.0168  Preprocessor1_Model06
##  7   697 roc_auc binary     0.832    10 0.0166  Preprocessor1_Model07
##  8   736 roc_auc binary     0.832    10 0.0167  Preprocessor1_Model08
##  9   829 roc_auc binary     0.830    10 0.0168  Preprocessor1_Model09
## 10   944 roc_auc binary     0.829    10 0.0175  Preprocessor1_Model10
## 11  1077 roc_auc binary     0.828    10 0.0177  Preprocessor1_Model11
## 12  1138 roc_auc binary     0.827    10 0.0177  Preprocessor1_Model12
## 13  1290 roc_auc binary     0.826    10 0.0176  Preprocessor1_Model13
## 14  1349 roc_auc binary     0.826    10 0.0178  Preprocessor1_Model14
## 15  1445 roc_auc binary     0.825    10 0.0178  Preprocessor1_Model15
## 16  1583 roc_auc binary     0.825    10 0.0179  Preprocessor1_Model16
## 17  1614 roc_auc binary     0.825    10 0.0179  Preprocessor1_Model17
## 18  1771 roc_auc binary     0.825    10 0.0176  Preprocessor1_Model18
## 19  1837 roc_auc binary     0.825    10 0.0176  Preprocessor1_Model19
## 20  1905 roc_auc binary     0.825    10 0.0179  Preprocessor1_Model20
```

```
#show the estimates of each tree parameter
tree_tune %>%
  collect_metrics() %>%
  filter(.metric == "roc_auc") %>%
  ggplot(aes(x = trees, y = mean)) +
  geom_point() +
```

```
  geom_line() +
  labs(title = "Tree Parameter Tuning",
       x = "Number of Trees",
       y = "ROC AUC")
```




```
#select the best model
best_tree <- tree_tune %>%
  select_best(metric = "roc_auc")

best_tree
```

```
## # A tibble: 1 x 2
##   trees .config
##   <int> <chr>
## 1    44 Preprocessor1_Model01
```

Tune Stochastic Parameters

1. Create a new specification where you set the learning rate and tree parameters (which you already optimized) and tune the stochastic parameters.

```
# tune only stochastic parameters
xgboost_stoch_spec <- boost_tree(
  mtry = tune(),
  min_n = tune(),
  sample_size = tune(),
  tree_depth = tune(),
```

```r
  trees = best_tree[[1]],
  learn_rate = best_learn_rate[[1]]) %>%
  set_engine("xgboost") %>%
  set_mode("classification")
```

2. Set up a tuning grid. Use grid_latin_hypercube() again.

```r
# finalize mtry range based on the number of predictors
mtry_finalized <- finalize(mtry(), train)

#tuning grid for the stochastic parameters in the xgboost model
stoch_grid <- grid_latin_hypercube(mtry_finalized,
                                   sample_size = sample_prop(),
                                   min_n(),
                                   tree_depth(),
                                   size = 20)
```

3. Show the performance of the best models and the estimates for the tree parameter values associated with each.

```r
#use the tuning grid to tune the stochastic parameters
Sys.time()
```

```
## [1] "2024-03-06 20:15:09 PST"
```

```r
stoch_tune <- tune_grid(
  xgboost_stoch_spec,
  eel_recipe,
  resamples = cv_set,
  grid = stoch_grid,
  metrics = metric_set(roc_auc),
  control = control_grid(save_pred = TRUE)
)

Sys.time()
```

```
## [1] "2024-03-06 20:15:29 PST"
```

```r
#show the estimates of each stochastic parameter in table
stoch_tune %>%
  collect_metrics() %>%
  filter(.metric == "roc_auc")
```

```
## # A tibble: 20 x 10
##     mtry min_n tree_depth sample_size .metric .estimator  mean     n std_err
##    <int> <int>      <int>       <dbl> <chr>   <chr>      <dbl> <int>   <dbl>
##  1    11    31          9       0.756 roc_auc binary     0.838    10 0.00581
##  2     9    37          5       0.271 roc_auc binary     0.5      10 0
##  3     4    34          1       0.829 roc_auc binary     0.824    10 0.00986
##  4    10    19         12       0.610 roc_auc binary     0.828    10 0.00790
##  5     2    19          2       0.701 roc_auc binary     0.830    10 0.00931
##  6    12     4         13       0.938 roc_auc binary     0.834    10 0.0146
##  7     7    12          3       0.992 roc_auc binary     0.834    10 0.0101
##  8     7     4          7       0.197 roc_auc binary     0.819    10 0.0130
##  9     8    25         10       0.173 roc_auc binary     0.5      10 0
## 10    13    17         11       0.466 roc_auc binary     0.829    10 0.00731
## 11     2    22          6       0.399 roc_auc binary     0.779    10 0.0191
```

```
## 12    14    23          10       0.527 roc_auc binary     0.834     10 0.00611
## 13     5    30           4       0.347 roc_auc binary     0.5       10 0
## 14     9     6          14       0.777 roc_auc binary     0.834     10 0.0101
## 15     1    11           9       0.895 roc_auc binary     0.835     10 0.0104
## 16     4    40           7       0.583 roc_auc binary     0.727     10 0.0229
## 17    10    28          14       0.310 roc_auc binary     0.5       10 0
## 18     6    36           3       0.665 roc_auc binary     0.799     10 0.00928
## 19    12    15          12       0.129 roc_auc binary     0.5       10 0
## 20     5     9           6       0.434 roc_auc binary     0.832     10 0.0129
## # i 1 more variable: .config <chr>
```

```r
#select the best model
best_stoch <- stoch_tune %>%
  select_best(metric = "roc_auc")
```

Finalize workflow and make final prediction

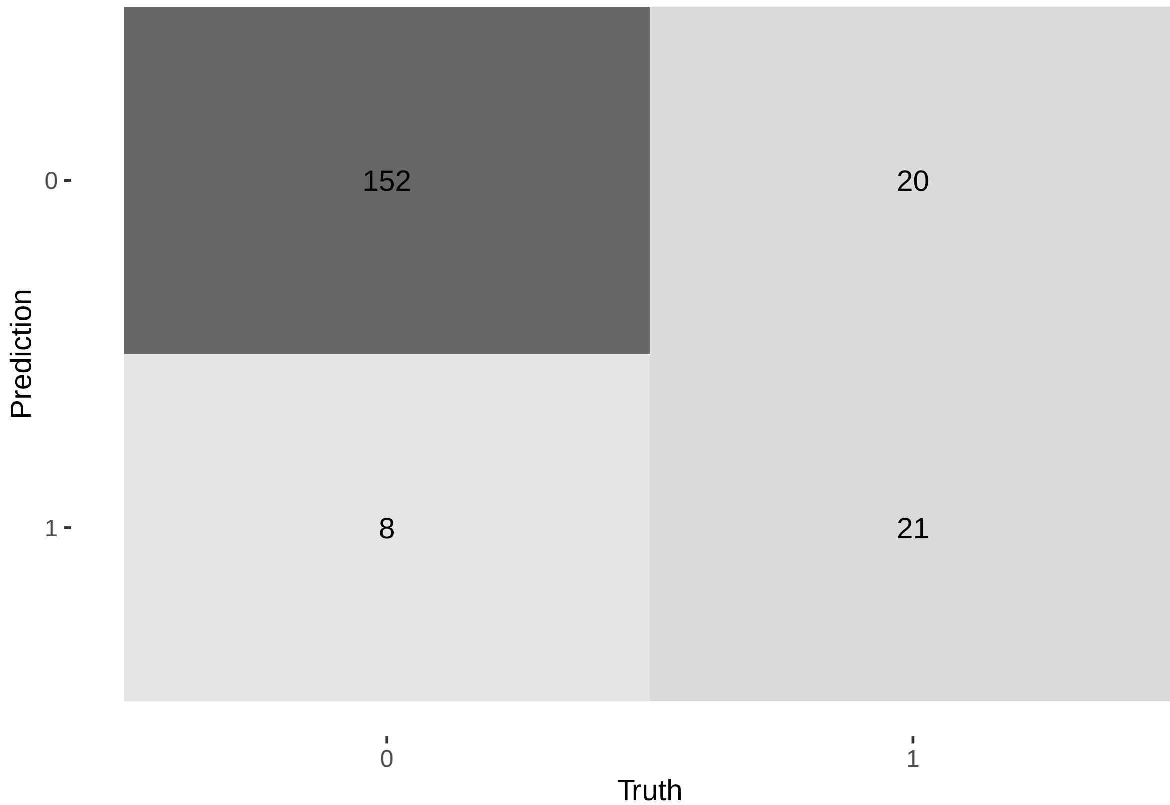
```r
#finalize the model with best parameters
final_xgboost <- boost_tree(
  trees = best_tree[[1]],
  min_n = best_stoch[[1]],
  sample_size = best_stoch[[4]],
  tree_depth = best_stoch[[2]],
  learn_rate = best_learn_rate[[1]]) %>%
  set_engine("xgboost") %>%
  set_mode("classification")

#fit the model on the data
final_fit <- fit(final_xgboost, Angaus ~ ., train)

# use last fit to predict on the test data
eel_predictions <- last_fit(final_xgboost,
                            Angaus ~ .,
                            eel_split) %>%
                   collect_predictions()
```

1. How well did your model perform? What types of errors did it make?

```r
#use the yardstick package to evaluate the model
eel_predictions %>%
  metrics(truth = Angaus, estimate = .pred_class)
```

```
## # A tibble: 2 x 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy binary         0.861
## 2 kap      binary         0.519
```

```r
#plot the confusion matrix as image
conf_mat <- eel_predictions %>%
  conf_mat(truth = Angaus, estimate = .pred_class)

conf_mat %>%
  autoplot(type = "heatmap")
```



This is one of the most misleading model accuracy I have ever worked with. I counted total number of non-Anguas in the test set and there are 160 non Anguas. and if we look at the confusion matrix, there are 151 model predictions that were correct with truth. it is not because model performed well, its becuase there were too many non-anguas values, which led to the high accuracy. I don't think this model is any better than a dummy model that randomly picks anguas values and predict it. My Kap value clearly shows that model is only 45-50% efficient.

However, we can say the model performed well with an accuracy of 0.86. The model made more false positive errors than false negative errors. The model misclassified 8 of the non-anguas as Anguas. While the model also misclassified 18 Anguas as non Anguas.

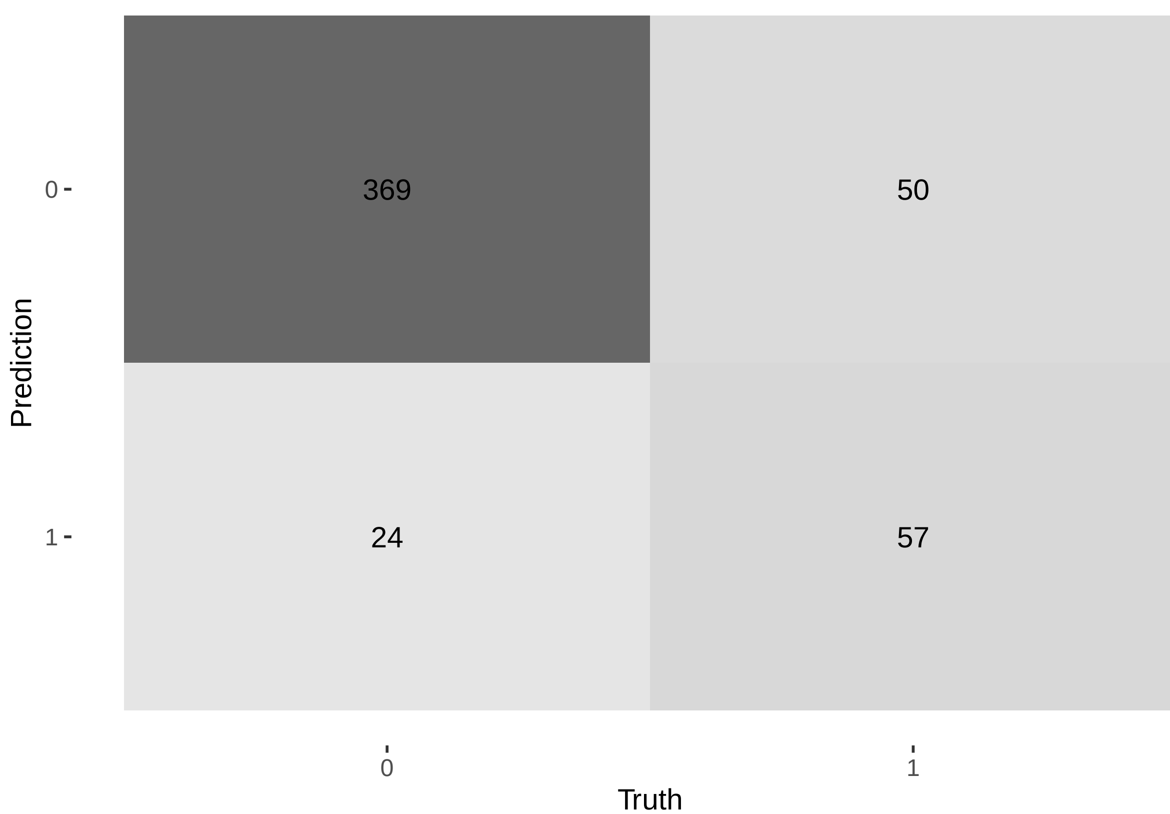
Fit your model the evaluation data and compare performance

1. Now used your final model to predict on the other dataset (eval.data.csv)

```
#change the Angaus column to factor
evaluation_data$Angaus_obs <- as.factor(evaluation_data$Angaus_obs)

# fit with the evaluation data
final_fit <- fit(final_xgboost, Angaus_obs ~ ., evaluation_data)


#check the accuracy of the model
final_fit %>%
  predict(evaluation_data) %>%
  bind_cols(evaluation_data) %>%
  metrics(truth = Angaus_obs, estimate = .pred_class)
```

```
## # A tibble: 2 x 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy binary         0.852
## 2 kap      binary         0.517
```

```
#plot confusion matrix
conf_mat <- final_fit %>%
  predict(evaluation_data) %>%
  bind_cols(evaluation_data) %>%
  conf_mat(truth = Angaus_obs, estimate = .pred_class)

conf_mat %>%
  autoplot(type = "heatmap")
```



2. How does your model perform on this data?

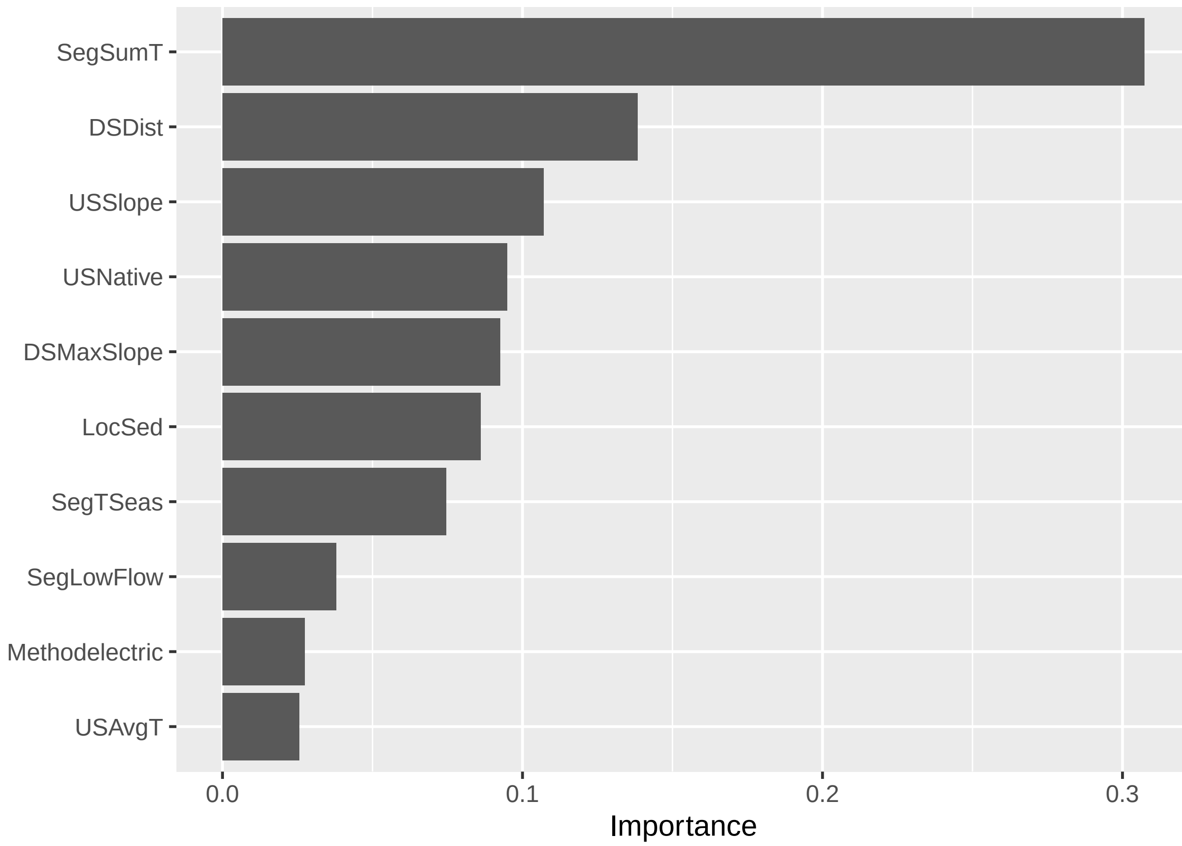
The model performed well with this data with similar high accuracy as with training set. However, the kap accuracy is still 50%, which means this model prediction is just 50% efficient. The model made lots of error in misclasifying true anguas. It misclassified 50 of the true anguas as non-anguas. I consider this as serious false negative issue.

3. How do your results compare to those of Elith et al.?

The model performed similar to that of Elith et al. The model used by Elith et al. had an accuracy of 0.86 for validation set, which is similar to my model. For training set, Elith had 0.95 for the training sets while my model had an accuracy of 0.85 only.

- Use {vip} to compare variable importance

```r
#use vip to plot the variable importance
final_fit %>%
  vip::vip(num_features = 10)
```



- What do your variable importance results tell you about the distribution of this eel species?

The variable of importance is in different order compared to Elith paper except for the top most variable. In my model, the most important variable is Summer Temperature, which is same as Elith paper. While the order changed for the rest of the variables. This tells me that the distribution of this eel species is highly dependent on the summer temperature. The other variables are also important but not as important as summer temperature. The changes in order can be acccounted with learning rate chosen by elith and me. He chose 0.005 and number of trees as 1000, becuase there were 1000 sites. But, I use tuning to find the best estimate for those hyperparameters. Since 1000 trees make more sense becuase there were 1000 sites, using niche knowledge is more helpful in this case and I believe Elith result is more true represenation of the anguas species distribution.