# Lab 6 - Eel Distribution with Boosted Trees

Colleen McCamy

2023-03-01

```
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.2 --
## v ggplot2 3.4.0      v purrr   0.3.5
## v tibble  3.1.8      v dplyr   1.0.10
## v tidyr   1.2.1      v stringr 1.5.0
## v readr   2.1.2      v forcats 0.5.1
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
library(tidymodels)
```

```
## -- Attaching packages --------------------------------------- tidymodels 1.0.0 --
## v broom        1.0.0     v rsample      1.1.0
## v dials        1.0.0     v tune         1.0.0
## v infer        1.0.3     v workflows    1.0.0
## v modeldata    1.0.1     v workflowsets 1.0.0
## v parsnip      1.0.1     v yardstick    1.1.0
## v recipes      1.0.1
## -- Conflicts ------------------------------------------ tidymodels_conflicts() --
## x scales::discard() masks purrr::discard()
## x dplyr::filter()   masks stats::filter()
## x recipes::fixed()  masks stringr::fixed()
## x dplyr::lag()      masks stats::lag()
## x yardstick::spec() masks readr::spec()
## x recipes::step()   masks stats::step()
## * Use suppressPackageStartupMessages() to eliminate package startup messages
```

```
library(ggplot2)
library(rsample)
library(baguette)
library(recipes)
library(parsnip)
library(tune)
library(dials)
library(workflows)
library(yardstick)
library(xgboost)
```

```
##
## Attaching package: 'xgboost'
##
## The following object is masked from 'package:dplyr':
##
##     slice
```

```
library(beepr)
library(tictoc)
library(tidyr)
library(caret)
```

```
## Loading required package: lattice
##
## Attaching package: 'caret'
##
## The following objects are masked from 'package:yardstick':
##
##     precision, recall, sensitivity, specificity
##
## The following object is masked from 'package:purrr':
##
##     lift
```

```
library(vip)
```

```
##
## Attaching package: 'vip'
##
## The following object is masked from 'package:utils':
##
##     vi
```

## Case Study Eel Species Distribution Modeling

This week's lab follows a modeling project described by Elith et al. (2008) (Supplementary Reading)

## Data

Grab the model training data set from the class Git:

data/eel.model.data.csv

```
# reading in the data
eel_dat <- read_csv("/Users/colleenmccamy/Documents/MEDS/classes/winter/eds_232_machine_learning/labs/e
  janitor::clean_names() |>
  dplyr::select(-c("site")) |>
  mutate("angaus" = as.factor(angaus))
```

```
## Rows: 1000 Columns: 14
## -- Column specification --------------------------------------------------------
```

```
## Delimiter: ","
## chr  (1): Method
## dbl (13): Site, Angaus, SegSumT, SegTSeas, SegLowFlow, DSDist, DSMaxSlope, U...
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

### Split and Resample

Split the joined data from above into a training and test set, stratified by outcome score. Use 10-fold CV to resample the training set, stratified by Angaus

```
set.seed(3619) # adding set.seed for reproducibility 369 Michelle's is

# conducting the initial split
eel_split <- initial_split(eel_dat,
                           prop = 0.70,
                           strata = "angaus")

# determining the training and test data for the model
eel_test <- testing(eel_split)
eel_train <- training(eel_split)

# creating cross-validation folds for the training data
eel_folds <- eel_train |> vfold_cv(v = 10,
                                   strata = "angaus")
```

### Preprocess

Create a recipe to prepare your data for the XGBoost model. We are interested in predicting the binary outcome variable Angaus which indicates presence or absence of the eel species Anguilla australis

```
# creating a recipe for our model
boost_rec <- recipe(angaus ~., data = eel_train) |>
  step_dummy(all_nominal(), -all_outcomes(), one_hot = TRUE) |>
  prep(eel_train)
```

## Tuning XGBoost

### Tune Learning Rate

Following the XGBoost tuning strategy outlined on Monday, first we conduct tuning on just the learn_rate parameter:

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

- Only specify one parameter to tune()

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))

- 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().

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

```
##########################################
############ Initiating the Model ##########
# ------------------------------------- #

# creating an XGBoost model specification
learn_spec <-
  boost_tree(
    mode = "classification",
    learn_rate = tune()) |>
    set_engine("xgboost")

##########################################
############ Tuning Learning Rate ##########
# ------------------------------------- #

# establishing the learn sequence
learn_seq <- seq(0.001, 0.3, length.out = 30)

# creating a grid with the different learning rate parameters
learn_seq_grid <-
  expand.grid(
    learn_rate = learn_seq)

# adding a workflow the learning rate tune
learn_wf <- workflow() |>
  add_model(learn_spec) |>
  add_recipe(boost_rec)

tic()
# tuning with the cross validation data to find the best learning rate
learn_tuned <- tune_grid(
  object = learn_wf,
  resamples = eel_folds,
  grid = learn_seq_grid,
  metrics = yardstick::metric_set(roc_auc, accuracy))

beep(8)
toc()
```

```
## 52.262 sec elapsed
```

```
print("72 seconds elapsed for this learning rate tuning with a sequence.")
```

```
## [1] "72 seconds elapsed for this learning rate tuning with a sequence."
```

```
##########################################
##### Finalizing & Evaluating the Model ####
# ------------------------------------- #

# selecting the best learning rate
best_learn <- select_best(learn_tuned)[1,1]
best_learn
```

```
## # A tibble: 1 x 1
##   learn_rate
##        <dbl>
## 1      0.238
```

```
# showing the learning rate accuracy
accuracy_best_learn <- as_tibble(show_best(learn_tuned,
                                           metric = "roc_auc",
                                           n = 1))[1,4]

# printing results
print(paste0("The area under the curve metric for these best learnig rate is ", round(accuracy_best_lea
```

```
## [1] "The area under the curve metric for these best learnig rate is 0.8447"
```

```
top_learn <- as_tibble(show_best(learn_tuned))
knitr::kable(top_learn)
```

| learn_rate | .metric | .estimator | mean | n | std_err | .config |
|---|---|---|---|---|---|---|
| 0.2381379 | roc_auc | binary | 0.8447093 | 10 | 0.0213223 | Preprocessor1_Model24 |
| 0.3000000 | roc_auc | binary | 0.8405032 | 10 | 0.0240042 | Preprocessor1_Model30 |
| 0.2896897 | roc_auc | binary | 0.8404445 | 10 | 0.0236151 | Preprocessor1_Model29 |
| 0.2793793 | roc_auc | binary | 0.8400773 | 10 | 0.0206681 | Preprocessor1_Model28 |
| 0.2484483 | roc_auc | binary | 0.8381586 | 10 | 0.0217091 | Preprocessor1_Model25 |

**Tune Tree Parameters**

1. Create a new specification where you set the learning rate (which you already optimized) and tune the tree parameters.
2. Set up a tuning grid. This time use grid_max_entropy() to get a representative sampling of the parameter space
3. Show the performance of the best models and the estimates for the tree parameter values associated with each.

```
##########################################
########### Initiating the Model ###########
# ------------------------------------- #

# creating an XGBoost model specification
tree_spec <-
```

```
  boost_tree(
    mode = "classification",
    learn_rate = best_learn,
    trees = 3000,
    tree_depth = tune(),
    min_n = tune(),
    loss_reduction = tune()) |>
    set_engine("xgboost")

# grid specification for the specification
tree_params <-
  dials::parameters(
    tree_depth(),
    min_n(),
    loss_reduction())

# adding a workflow the learning rate tune
tree_wf <- workflow() |>
  add_model(tree_spec) |>
  add_recipe(boost_rec)

tree_grid <- grid_max_entropy(tree_params, size = 20, iter = 100)

##############################################
### Tuning the Model for Tree Parameters ###
# ---------------------------------------- #

tic()
# tuning with the cross validation data to find the best learning rate
tree_tuned <- tune_grid(
  object = tree_wf,
  resamples = eel_folds,
  grid = tree_grid,
  metrics = yardstick::metric_set(roc_auc, accuracy))

beep(8)
toc()
```

```
## 367.762 sec elapsed
```

```
print("431 seconds elapsed when tuning for these tree specifications")
```

```
## [1] "431 seconds elapsed when tuning for these tree specifications"
```

```
##############################################
###### Finalizing & Fitting the Model ######
# ---------------------------------------- #

# selecting the best learning rate
accuracy_best_tree <- as_tibble(show_best(tree_tuned, n = 1))[1,6]
best_tree <- select_best(tree_tuned, metric = "roc_auc")

print(paste0("The area under the curve metric for these best tree parameters is ", round(accuracy_best_
```

```
## [1] "The area under the curve metric for these best tree parameters is 0.8513"
```

```
top_tree <- show_best(tree_tuned)
knitr::kable(top_tree)
```

| min_n | tree_depth | loss_reduction | .metric | .estimator | mean | n | std_err | .config |
|---|---|---|---|---|---|---|---|---|
| 16 | 12 | 1.2946401 | roc_auc | binary | 0.8512562 | 10 | 0.0213447 | Preprocessor1_Model05 |
| 15 | 1 | 1.8216643 | roc_auc | binary | 0.8508507 | 10 | 0.0207769 | Preprocessor1_Model02 |
| 4 | 3 | 0.8849202 | roc_auc | binary | 0.8464873 | 10 | 0.0217249 | Preprocessor1_Model07 |
| 4 | 8 | 1.7392168 | roc_auc | binary | 0.8410977 | 10 | 0.0219633 | Preprocessor1_Model03 |
| 13 | 10 | 0.0490642 | roc_auc | binary | 0.8388953 | 10 | 0.0237028 | Preprocessor1_Model15 |

**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.
2. Set up a tuning grid. Use grid_max_entropy() again.
3. Show the performance of the best models and the estimates for the tree parameter values associated with each.

```
##############################################
############ Initiating the Model ############
# ------------------------------------- #

# saving the different tree parameters
best_min <- best_tree$min_n[1]
best_depth <- best_tree$tree_depth[1]
best_loss <- best_tree$loss_reduction[1]

# creating an XGBoost model specification
model_spec <-
  boost_tree(
    mode = "classification",
    learn_rate = best_learn,
    trees = 3000,
    tree_depth = best_depth,
    min_n = best_min,
    loss_reduction = best_loss,
    mtry = tune(),
    sample_size = tune()) |>
    set_engine("xgboost")

###TEST 2
model_grid <- model_spec |>
  parameters() |>
  finalize(select(eel_test, -angaus))  |>
  grid_max_entropy(size = 20,
                   iter = 100)

# grid specification for the specification
```

```
model_params <-
   dials::parameters(finalize(mtry(), eel_train |> select(-angaus)),
     sample_size = sample_prop(c(0.1, 0.9)))

# using the grid paramters to get the optimal stochastic parameters
model_grid <-
  dials::grid_max_entropy(
    model_params,
    size = 40,
    iter = 30)

# adding a workflow the learning rate tune
model_wf <- workflow() |>
  add_model(model_spec) |>
  add_recipe(boost_rec)

##############################################
# Tuning the Model for Stochastic Parameters
# ------------------------------------------ #

tic()
# tuning with the cross validation data to find the best learning rate
model_tuned <- tune_grid(
  object = model_wf,
  resamples = eel_folds,
  grid = model_grid)

beep(8)
toc()
```

```
## 1510.554 sec elapsed
```

```
print("528 seconds elapsed in tuning the Stochastic parameters.")
```

```
## [1] "528 seconds elapsed in tuning the Stochastic parameters."
```

```
##############################################
##### Finalizing & Evaluating the Model ####
# ------------------------------------------ #

# selecting the best learning rate
best_model <- select_best(model_tuned, metric = "roc_auc")
best_model
```

```
## # A tibble: 1 x 3
##    mtry sample_size .config
##   <int>       <dbl> <chr>
## 1     5       0.877 Preprocessor1_Model11
```

```
# selecting the best accuracy
accuracy_best_model <- as_tibble(show_best(model_tuned, n = 1))[1,5]
```

```
print(paste0("The area under the curve metric for these best Stochastic parameters is ",round(accuracy_
```

```
## [1] "The area under the curve metric for these best Stochastic parameters is 0.8464"
```

```
# saving best Stochastic parameters
best_mtry <- best_model$mtry[1]
best_sample <- best_model$sample_size[1]

show_best(model_tuned)
```

```
## # A tibble: 5 x 8
##    mtry sample_size .metric .estimator  mean     n std_err .config
##   <int>       <dbl> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1     5       0.877 roc_auc binary     0.846    10  0.0240 Preprocessor1_Model11
## 2     6       0.858 roc_auc binary     0.846    10  0.0267 Preprocessor1_Model21
## 3    10       0.832 roc_auc binary     0.844    10  0.0230 Preprocessor1_Model19
## 4     1       0.767 roc_auc binary     0.843    10  0.0260 Preprocessor1_Model33
## 5     2       0.863 roc_auc binary     0.842    10  0.0267 Preprocessor1_Model14
```

```
top_learn <- as_tibble(show_best(model_tuned))
knitr::kable(top_learn)
```

| mtry | sample_size | .metric | .estimator | mean | n | std_err | .config |
|---|---|---|---|---|---|---|---|
| 5 | 0.8768933 | roc_auc | binary | 0.8464100 | 10 | 0.0240014 | Preprocessor1_Model11 |
| 6 | 0.8577280 | roc_auc | binary | 0.8460359 | 10 | 0.0266904 | Preprocessor1_Model21 |
| 10 | 0.8320133 | roc_auc | binary | 0.8438026 | 10 | 0.0229903 | Preprocessor1_Model19 |
| 1 | 0.7674055 | roc_auc | binary | 0.8431517 | 10 | 0.0260425 | Preprocessor1_Model33 |
| 2 | 0.8632364 | roc_auc | binary | 0.8422032 | 10 | 0.0267197 | Preprocessor1_Model14 |

## Finalize workflow and make final prediction

1. Assemble your final workflow will all of your optimized parameters and do a final fit.

```
# creating an XGBoost model specification
final_spec <-
  boost_tree(
    mode = "classification",
    learn_rate = best_learn,
    trees = 3000,
    tree_depth = best_depth,
    min_n = best_min,
    loss_reduction = best_loss,
    mtry = best_mtry,
    sample_size = best_sample) |>
    set_engine("xgboost")

# final workflow
final_wf <- workflow() |>
```

```
  add_model(final_spec) |>
  add_recipe(boost_rec)

# running a final fit
final_fit <- last_fit(final_wf,
                      split = eel_split)

# collecting the predictions from the final fit
final_pred <- final_fit |> collect_predictions()

# collecting metrics from the final fit
angaus_metrics <- final_fit |> collect_metrics()
angaus_metrics
```

```
## # A tibble: 2 x 4
##   .metric  .estimator .estimate .config
##   <chr>    <chr>          <dbl> <chr>
## 1 accuracy binary         0.831 Preprocessor1_Model1
## 2 roc_auc  binary         0.843 Preprocessor1_Model1
```

```
# creating a confusion matrix
confusion_matrix <- confusionMatrix(final_pred$.pred_class,
                                    final_pred$angaus,
                                    positive = "1")

# creating true positive/false positive, etc. values
tp <- 26
fn <- 35
fp <- 20
tn <- 220
true_positive <- tp / (tp+fn)
false_positive <- fp / (fp + tn)
true_negative <- tn / (tn+fp)
false_negative <- fn / (fn + tn)
```

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

   **RESPONSE: The model performed predictions with an 81.7% accuracy and an 0.849 area under the Receiver Operating Characteristic curve. Looking at the confusion matrix, we can see that the model performs well in predicting no angaus presence with a true negative rate of 0.90. However, the model does not perform as well in predicting when angaus is present as there is a true positive rate of 0.557 and false negative rate of 0.443. This could provide evidence that the the model could do best at predicting angaus presence versus angaus absence.**

## Fit your model the evaluation data and compare performance

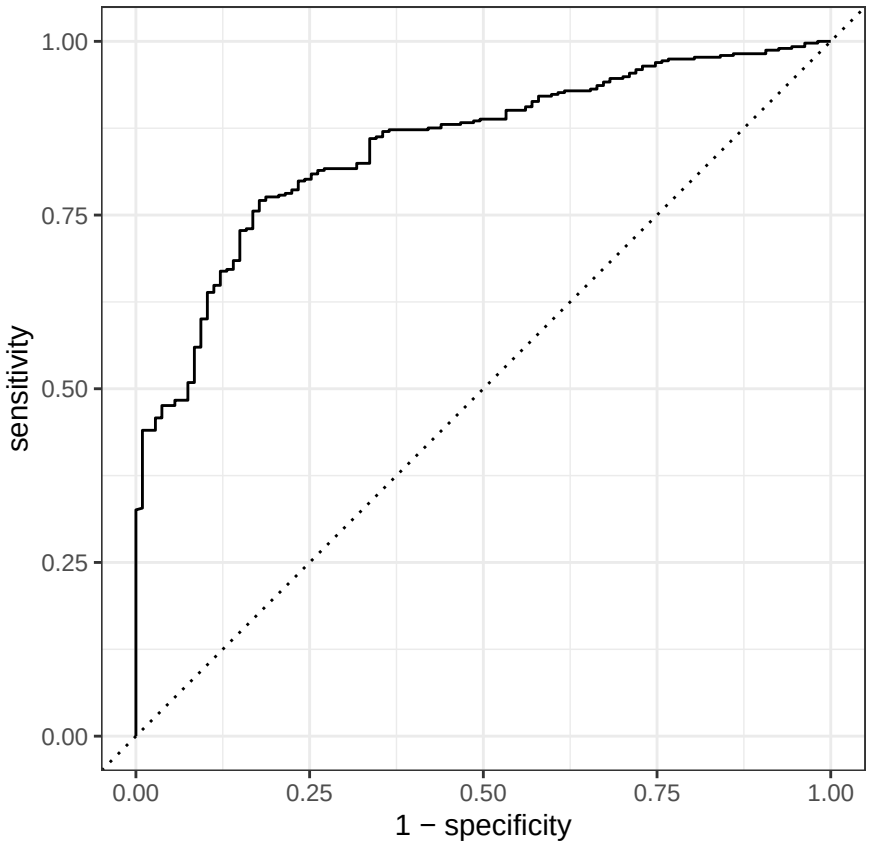
1. Now fit your final model to the big dataset: data/eval.data.csv

```
# reading in eel evaluation data
eel_eval_dat <- read_csv("/Users/colleenmccamy/Documents/MEDS/classes/winter/eds_232_machine_learning/la
  janitor::clean_names() |>
```

```
  rename("angaus" = "angaus_obs") |>
  mutate("angaus" = as.factor(angaus))
```

```
## Rows: 500 Columns: 13
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr  (1): Method
## dbl (12): Angaus_obs, SegSumT, SegTSeas, SegLowFlow, DSDist, DSMaxSlope, USA...
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
##########################################
######### Prepossessing the Data ###########
# ------------------------------------- #

##########################################
## Fitting the Data & Evaluating the Model #
# ------------------------------------- #
fit_final_wf <- final_wf |>
  fit(eel_train)

# running a final fit
eval_fit <- fit_final_wf |>
  predict(new_data = eel_eval_dat)

# changing outcome to level in the dataframe
eel_eval_level <- levels(eel_eval_dat$angaus)

# change predicted to the a factor with the same levels as the outcome
eval_fit$.pred_class <- factor(eval_fit$.pred_class, levels = eel_eval_level)

# adding new prediction column to the dataframe
eel_eval_dat <- cbind(eval_fit$.pred_class, eel_eval_dat) |>
  bind_cols(fit_final_wf |>
              predict(new_data = eel_eval_dat, type = "prob"))

# creating a confusion matrix
confusion_eval <- confusionMatrix(eel_eval_dat$"eval_fit$.pred_class",
                                  eel_eval_dat$angaus)

# plotting the ROC
eel_eval_dat |>
  roc_curve(truth = angaus, .pred_0) |>
  autoplot()
```




```
# determining the Area Under the Curve
final_auc <- eel_eval_dat |>
  roc_auc(angaus, .pred_0)
```

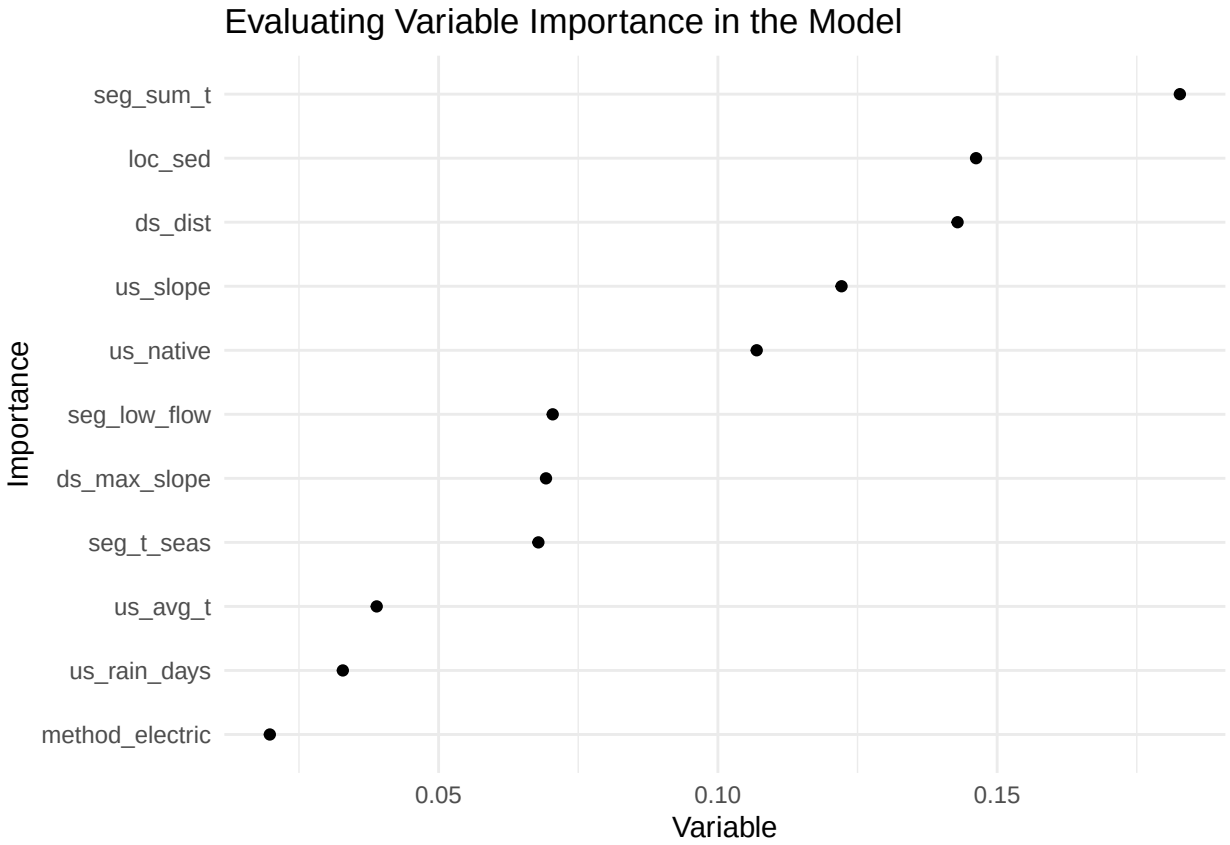
2. How does your model perform on this data?

   **Response: The final metric for the area under the curve is 0.860. This indicates that the model performed better on this new data than it did for the training and testing data. The accuracy of this model is also 0.836 and the no information rate of 0.786. We can also see that this data had a high occurrence of the majority class. The accuracy of the model is higher than the no information rate at the 5% significance level.**

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

- Use {vip} to compare variable importance
- What do your variable importance results tell you about the distribution of this eel species?

```
# conducting VIP on the final mod
final_wf |>
  fit(data = eel_eval_dat) |>
  extract_fit_parsnip() |>
  vip(geom = "point",
      num_features = 11) + theme_minimal() +
  labs(x = "Importance",
       y = "Variable",
       title = "Evaluating Variable Importance in the Model")
```




**RESPONSE: The area under the receiver operating characteristic curve (AUC ROC) for this model using the eel evaluation data was 0.86. This is similar to the AUC ROC metric for the independent model on 12,369 sites without cross validation or other training factors which was 0.858.**

**However the AUC ROC for the for looking at 1,000 sites and using additional training methods yielded 0.958. This is higher than the created model's performance. This could be due to the fact that the data used to train this model had a higher occurrence of no eel presence versus eel presence. While using a stratified split and cross validation to try to account for this, this could still be affecting our AUC ROC results.**

**The variable importance graph informs which variables are most influential in determining eel presence from the created model. Looking at the graph it appears that the summer air temperature in Celsius was most important followed by if the area fell within indigenous forest and the distance to the coast in kilometers.**