Lab6

Charlie Curtin

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 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 packages
library(tidyverse)
library(here)
library(parsnip)
library(caret)
library(tidymodels)
library(vip)
```

```
# read in the data
eel_model <- read_csv(here("labs", "data", "eel.model.data.csv")) %>%
  mutate(Angaus = as.factor(Angaus)) %>%
  select(-Site)
```

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.

```
set.seed(345)

# split the model data
eel_split <- initial_split(eel_model, strata = Angaus)
eel_train <- training(eel_split)
eel_test <- testing(eel_split)

# specify k-folds resampling with 10 folds
cv_folds <- vfold_cv(eel_train, v = 10)
```

Preprocess

Create a recipe to prepare your data for the XGBoost model

```
# specify a recipe
eel_rec <- recipe(Angaus ~ ., data = eel_train) %>%
  # turn nominal variables into dummy variables
  step_dummy(all_nominal_predictors(), one_hot = TRUE) %>%
  step_normalize(all_numeric_predictors())
```

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
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.

```
# specify an xgboost model with learn_rate set to tune
xgb_spec <- boost_tree(learn_rate = tune()) %>%
  set_mode("classification") %>%
  set_engine("xgboost")

# create a workflow with our model
xgb_wf <- workflow() %>%
  add_model(xgb_spec) %>%
  add_recipe(eel_rec)

# set up grid to tune our model
system.time(
  xgb_tune <- tune_grid(
  xgb_wf,
  resamples = cv_folds,
  grid = expand.grid(learn_rate = seq(0.0001, 0.3, length.out = 30))
))
```

```
## Warning: package 'xgboost' was built under R version 4.3.2
```

```
##    user  system elapsed
##  25.193   0.561  33.014
```

```r
# show performance of the top 3 models in terms of accuracy
show_best(xgb_tune, metric = "accuracy", n = 3)
```

```
## # A tibble: 3 x 7
##   learn_rate .metric  .estimator  mean     n std_err .config
##        <dbl> <chr>    <chr>      <dbl> <int>   <dbl> <chr>
## 1      0.145 accuracy binary     0.841    10 0.0130  Preprocessor1_Model15
## 2      0.279 accuracy binary     0.839    10 0.0130  Preprocessor1_Model28
## 3      0.217 accuracy binary     0.837    10 0.00897 Preprocessor1_Model22
```

```r
# select the best learn rate
learn_rate <- select_best(xgb_tune, metric = "accuracy")$learn_rate
```

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_latin_hypercube() 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.

```r
# create a new model specification to tune the tree parameter
xgb_spec2 <- boost_tree(learn_rate = learn_rate,
                        trees = tune()) %>%
  set_mode("classification") %>%
  set_engine("xgboost")

# add our model to a workflow
xgb_wf2 <- workflow() %>%
  add_model(xgb_spec2) %>%
  add_recipe(eel_rec)

# set up a tuning grid using a latin hypercube
lh_grid <- grid_latin_hypercube(x = parameters(xgb_wf2), size = 20)
```

```
## Warning: ‘parameters.workflow()‘ was deprecated in tune 0.1.6.9003.
## i Please use ‘hardhat::extract_parameter_set_dials()‘ instead.
## This warning is displayed once every 8 hours.
## Call ‘lifecycle::last_lifecycle_warnings()‘ to see where this warning was
## generated.
```

```r
system.time(
  xgb_tune2 <- tune_grid(
  xgb_wf2,
  resamples = cv_folds,
  grid = lh_grid)
)
```

```
##    user  system elapsed
##  19.146   0.406  25.065
```

```
# show top 3 models by accuracy
show_best(xgb_tune2, metric = "accuracy", n = 3)
```

```
## # A tibble: 3 x 7
##   trees .metric  .estimator  mean     n std_err .config
##   <int> <chr>    <chr>      <dbl> <int>   <dbl> <chr>
## 1   366 accuracy binary     0.835    10  0.0105 Preprocessor1_Model04
## 2  1703 accuracy binary     0.833    10  0.0103 Preprocessor1_Model18
## 3  1895 accuracy binary     0.833    10  0.0103 Preprocessor1_Model19
```

```
# select best number of trees
trees <- select_best(xgb_tune2, metric = "accuracy")$trees
```

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

```
# create a new model specification to tune the tree parameter
xgb_spec3 <- boost_tree(learn_rate = learn_rate,
                        trees = trees,
                        mtry = tune(),
                        sample_size = tune()) %>%
  set_mode("classification") %>%
  set_engine("xgboost")

# add our model to a workflow
xgb_wf3 <- workflow() %>%
  add_model(xgb_spec3) %>%
  add_recipe(eel_rec)

# set up a tuning grid using a latin hypercube
lh_grid3 <- grid_latin_hypercube(x = finalize(parameters(xgb_wf3), x = eel_train),
                                 size = 20)

system.time(
  xgb_tune3 <- tune_grid(
  xgb_wf3,
  resamples = cv_folds,
  grid = lh_grid3)
)
```

```
##    user  system elapsed
##  61.916   1.266  78.344
```

```
# show top 3 models by accuracy
show_best(xgb_tune3, metric = "accuracy", n = 3)
```

```
## # A tibble: 3 x 8
##    mtry sample_size .metric  .estimator  mean     n std_err .config
##   <int>       <dbl> <chr>    <chr>      <dbl> <int>   <dbl> <chr>
## 1     3       0.493 accuracy binary     0.841    10 0.0121  Preprocessor1_Model~
## 2     8       0.742 accuracy binary     0.839    10 0.0105  Preprocessor1_Model~
## 3     2       0.822 accuracy binary     0.838    10 0.00638 Preprocessor1_Model~
```

```
mtry <- select_best(xgb_tune3, metric = "accuracy")$mtry

sample_size <- select_best(xgb_tune3, metric = "accuracy")$sample_size
```

Finalize workflow and make final prediction

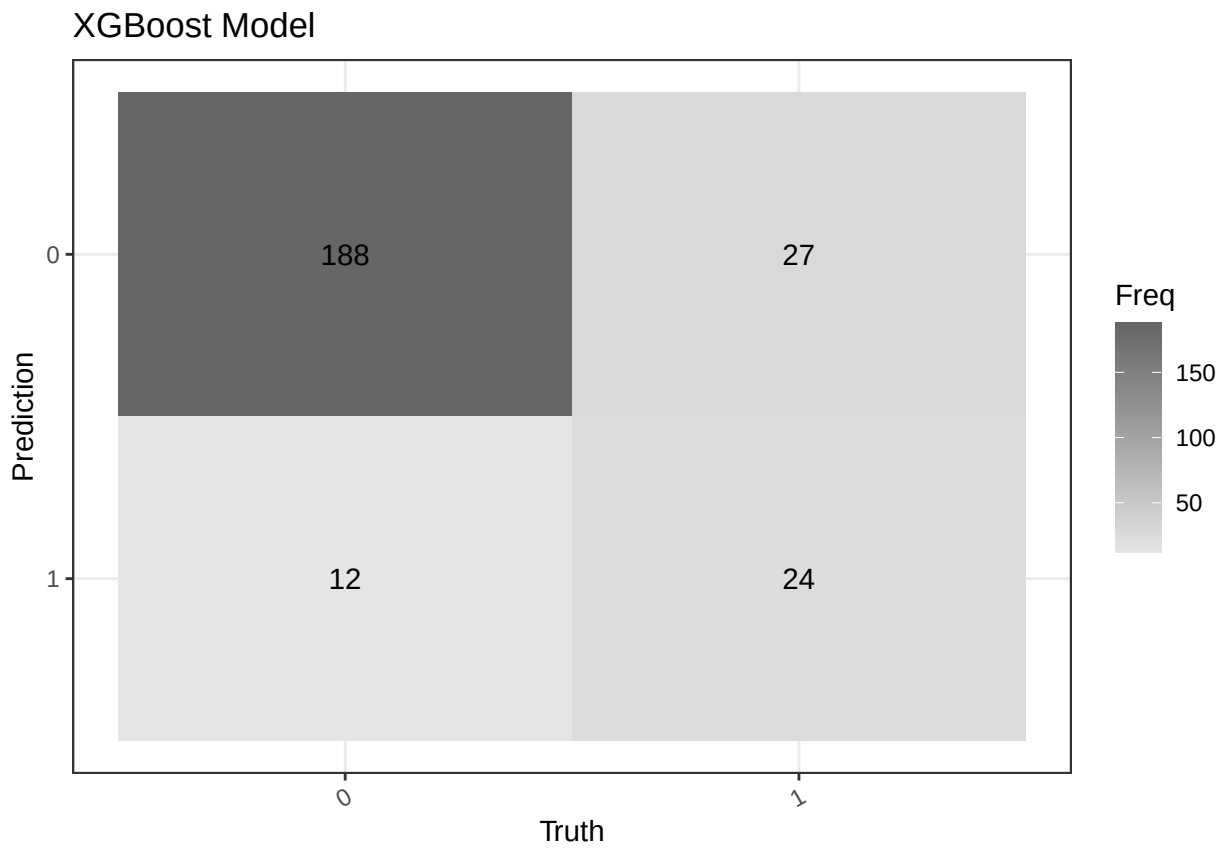
1. How well did your model perform? What types of errors did it make?

```
# finalize our workflow with our tuned parameters
xgb_final <- finalize_workflow(xgb_wf3,
                               select_best(xgb_tune3, metric = "accuracy"))

# fit our model
fit_xgb = fit(xgb_final, eel_train)

# make predictions on the test set
predict_xgb = predict(fit_xgb, eel_test) %>% # get testing predictions
  bind_cols(eel_test) %>%   #bind to testing column
  mutate(Angaus = as.factor(Angaus))

# assess the accuracy of our final model
accuracy(predict_xgb, truth = Angaus, estimate = .pred_class)
```

```
## # A tibble: 1 x 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy binary         0.845
```

- Our model correctly predicts the presence or absence of the eel species with 84% accuracy.

```
# visualize the confusion matrix of our model
predict_xgb %>%
  conf_mat(truth = Angaus, estimate = .pred_class) %>% # create confusion matrix
  autoplot(type = "heatmap") + # plot confusion matrix with heatmap
  theme_bw() + #change theme
  theme(axis.text.x = element_text(angle = 30, hjust = 1)) +
  #rotate axis labels
  labs(title = "XGBoost Model")
```



- By visualizing the confusion matrix, we can see that our model has made 11 false positives and 29 false negatives.

Fit your model the evaluation data and compare performance

1. Now used your final model to predict on the other dataset (eval.data.csv)
2. How does your model perform on this data?
3. How do your results compare to those of Elith et al.?

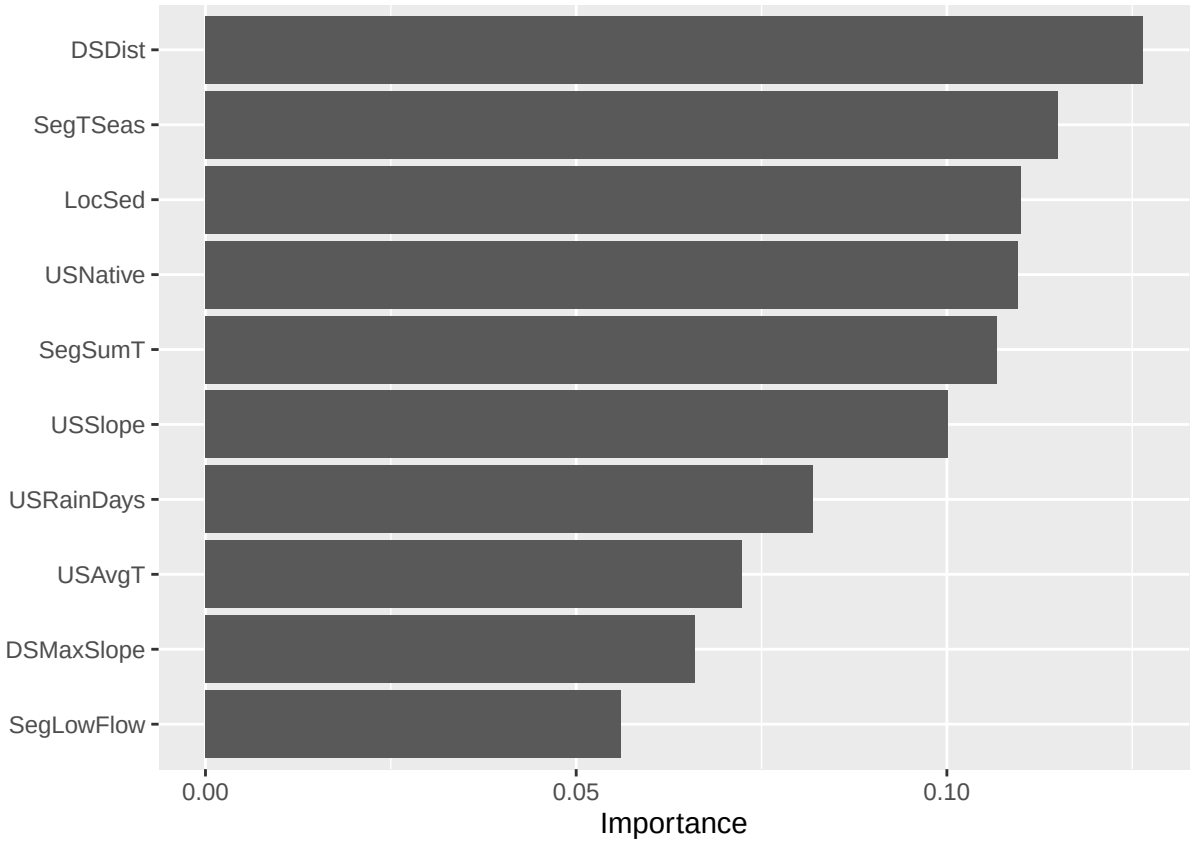
- Use {vip} to compare variable importance
- What do your variable importance results tell you about the distribution of this eel species?

```
# read in the other dataset
eel_eval <- read_csv(here("labs", "data", "eel.eval.data.csv")) %>%
  mutate(Angaus = as.factor(Angaus_obs)) %>%
  select(-Angaus_obs)

# make predictions on the new data
predict_xgb_eval = predict(fit_xgb, eel_eval) %>% # get testing predictions
  bind_cols(eel_eval)

# assess the accuracy
accuracy(predict_xgb_eval, truth = Angaus, estimate = .pred_class)
```

```
## # A tibble: 1 x 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy binary         0.818
```

- Our model predicts fish species occurrence on this new data with almost 82% accuracy.

```
vip(fit_xgb)
```



- Our variable importance plot tells us that "DSDist", which is distance to coast, is the most important variable in the data for predicting occurrence of a fish.