Lab6

Elke Windschitl

2023-03-01

```
library(readr)
library(tidyverse)
```

```
## ── Attaching packages ─────────────────────────────────────── tidyverse 1.3.2 ──
## ✔ ggplot2 3.4.0     ✔ dplyr   1.1.0
## ✔ tibble  3.1.8     ✔ stringr 1.5.0
## ✔ tidyr   1.3.0     ✔ forcats 1.0.0
## ✔ purrr   1.0.1
## ── Conflicts ────────────────────────────────────────── tidyverse_conflicts() ──
## ✖ dplyr::filter() masks stats::filter()
## ✖ dplyr::lag()    masks stats::lag()
```

```
library(tidymodels)
```

```
## ── Attaching packages ──────────────────────────────────────── tidymodels 1.0.0 ──
## ✔ broom        1.0.3     ✔ rsample      1.1.1
## ✔ dials        1.1.0     ✔ tune         1.0.1
## ✔ infer        1.0.4     ✔ workflows    1.1.2
## ✔ modeldata    1.1.0     ✔ workflowsets 1.0.0
## ✔ parsnip      1.0.3     ✔ yardstick    1.1.0
## ✔ recipes      1.0.4
## ── Conflicts ─────────────────────────────────────── tidymodels_conflicts() ──
## ✖ scales::discard() masks purrr::discard()
## ✖ dplyr::filter()   masks stats::filter()
## ✖ recipes::fixed()  masks stringr::fixed()
## ✖ dplyr::lag()      masks stats::lag()
## ✖ yardstick::spec() masks readr::spec()
## ✖ recipes::step()   masks stats::step()
## • Use tidymodels_prefer() to resolve common conflicts.
```

```
library(xgboost)
```

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

```
library(tictoc)
library(vip)
```

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

```
library(rsample)
library(recipes)
library(pROC)
```

```
## Type 'citation("pROC")' for a citation.
##
## Attaching package: 'pROC'
##
## The following objects are masked from 'package:stats':
##
##     cov, smooth, var
```

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

```
urlfile <- "https://raw.githubusercontent.com/MaRo406/eds-232-machine-learning/main/dat
a/eel.model.data.csv"

eel_data <- read_csv(url(urlfile)) %>%
  select(-Site)
```

```
## 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.
```

```
eel_data$Angaus <- as.factor(eel_data$Angaus)
```

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

```
# Stratified sampling with the rsample package
set.seed(123) #set a seed for reproducibility
split <- initial_split(data = eel_data,
                       prop = .7,
                       strata = "Angaus")
split
```

```
## <Training/Testing/Total>
## <699/301/1000>
```

```
eel_train <- training(split)
eel_test  <- testing(split)

# Set up cross validation
cv_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

```
eel_rec <- recipe(Angaus ~ ., data = eel_train) %>%
  step_dummy(all_nominal(), -all_outcomes(), one_hot = TRUE) %>%
  prep(training = eel_train, retain = TRUE)

# bake to check
baked_eel <- bake(eel_rec, 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()

```
eel_spec <- parsnip::boost_tree(mode = "classification",
                                engine = "xgboost",
                                trees = 3000,
                                learn_rate = 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().

```
tic()
eel_grid <- expand.grid(learn_rate = seq(0.0001, 0.3, length.out = 30))
eel_grid
```

```
##    learn_rate
## 1  0.00010000
## 2  0.01044138
## 3  0.02078276
## 4  0.03112414
## 5  0.04146552
## 6  0.05180690
## 7  0.06214828
## 8  0.07248966
## 9  0.08283103
## 10 0.09317241
## 11 0.10351379
## 12 0.11385517
## 13 0.12419655
## 14 0.13453793
## 15 0.14487931
## 16 0.15522069
## 17 0.16556207
## 18 0.17590345
## 19 0.18624483
## 20 0.19658621
## 21 0.20692759
## 22 0.21726897
## 23 0.22761034
## 24 0.23795172
## 25 0.24829310
## 26 0.25863448
## 27 0.26897586
## 28 0.27931724
## 29 0.28965862
## 30 0.30000000
```

```
wf_eel_tune <- workflow() %>%
  add_recipe(eel_rec) %>%
  add_model(eel_spec)
toc()
```

```
## 0.008 sec elapsed
```

```
tic()
doParallel::registerDoParallel()

set.seed(123)

eel_rs <- tune_grid(
  wf_eel_tune,
  Angaus~.,
  resamples = cv_folds,
  grid = eel_grid
 # metrics = metric_set(accuracy, roc_auc) -- this wasn't working when I did this
)
```

```
## Warning: The `...` are not used in this function but one or more objects were
## passed: ''
```

```
toc()
```

```
## 136.053 sec elapsed
```

```
eel_rs
```

```
## # Tuning results
## # 10-fold cross-validation using stratification
## # A tibble: 10 × 4
##    splits           id     .metrics          .notes
##    <list>           <chr>  <list>            <list>
##  1 <split [628/71]> Fold01 <tibble [60 × 5]> <tibble [0 × 3]>
##  2 <split [629/70]> Fold02 <tibble [60 × 5]> <tibble [0 × 3]>
##  3 <split [629/70]> Fold03 <tibble [60 × 5]> <tibble [0 × 3]>
##  4 <split [629/70]> Fold04 <tibble [60 × 5]> <tibble [0 × 3]>
##  5 <split [629/70]> Fold05 <tibble [60 × 5]> <tibble [0 × 3]>
##  6 <split [629/70]> Fold06 <tibble [60 × 5]> <tibble [0 × 3]>
##  7 <split [629/70]> Fold07 <tibble [60 × 5]> <tibble [0 × 3]>
##  8 <split [629/70]> Fold08 <tibble [60 × 5]> <tibble [0 × 3]>
##  9 <split [630/69]> Fold09 <tibble [60 × 5]> <tibble [0 × 3]>
## 10 <split [630/69]> Fold10 <tibble [60 × 5]> <tibble [0 × 3]>
```

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

```
eel_rs %>%
  tune::show_best(metric = "accuracy") %>%
  knitr::kable()
```

learn_rate	.metric	.estimator	mean	n	std_err	.config
0.1655621	accuracy	binary	0.8312548	10	0.0102540	Preprocessor1_Model17
0.1035138	accuracy	binary	0.8312352	10	0.0091530	Preprocessor1_Model11
0.3000000	accuracy	binary	0.8298469	10	0.0129247	Preprocessor1_Model30
0.1448793	accuracy	binary	0.8298256	10	0.0110078	Preprocessor1_Model15
0.2379517	accuracy	binary	0.8298061	10	0.0110902	Preprocessor1_Model24

```
eel_best_learn <- eel_rs %>%
  tune::select_best("accuracy")

knitr::kable(eel_best_learn)
```

learn_rate	.config
0.1655621	Preprocessor1_Model17

```
eel_model <- eel_spec %>%
  finalize_model(eel_best_learn)
```

Tune Tree Parameters

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

```
eel_spec2 <- parsnip::boost_tree(mode = "classification",
                                 engine = "xgboost",
                                 trees = 3000,
                                 learn_rate = eel_best_learn$learn_rate,
                                 min_n = tune(),
                                 tree_depth = tune(),
                                 loss_reduction = tune()
                                 )
```

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

```
eel_params <- dials::parameters(
  min_n(),
  tree_depth(),
  loss_reduction()
)

eel_grid2 <- dials::grid_max_entropy(eel_params, size = 30)
knitr::kable(head(eel_grid2))
```

min_n	tree_depth	loss_reduction
36	11	3.9791655
9	5	0.0000000
31	14	0.0000719
6	2	0.0004576
27	15	27.7383166
22	4	14.8775302

```
wf_eel_tune2 <- workflow() %>%
  add_recipe(eel_rec) %>%
  add_model(eel_spec2)
```

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

```
set.seed(123)
tic()
doParallel::registerDoParallel()

eel_rs2 <- tune_grid(
  wf_eel_tune2,
  Angaus~.,
  resamples = cv_folds,
  grid = eel_grid2
)
```

```
## Warning: The `...` are not used in this function but one or more objects were
## passed: ''
```

```
toc()
```

```
## 104.479 sec elapsed
```

```
eel_rs2
```

```
## # Tuning results
## # 10-fold cross-validation using stratification
## # A tibble: 10 × 4
##    splits           id     .metrics          .notes
##    <list>           <chr>  <list>            <list>
##  1 <split [628/71]> Fold01 <tibble [60 × 7]> <tibble [0 × 3]>
##  2 <split [629/70]> Fold02 <tibble [60 × 7]> <tibble [0 × 3]>
##  3 <split [629/70]> Fold03 <tibble [60 × 7]> <tibble [0 × 3]>
##  4 <split [629/70]> Fold04 <tibble [60 × 7]> <tibble [0 × 3]>
##  5 <split [629/70]> Fold05 <tibble [60 × 7]> <tibble [0 × 3]>
##  6 <split [629/70]> Fold06 <tibble [60 × 7]> <tibble [0 × 3]>
##  7 <split [629/70]> Fold07 <tibble [60 × 7]> <tibble [0 × 3]>
##  8 <split [629/70]> Fold08 <tibble [60 × 7]> <tibble [0 × 3]>
##  9 <split [630/69]> Fold09 <tibble [60 × 7]> <tibble [0 × 3]>
## 10 <split [630/69]> Fold10 <tibble [60 × 7]> <tibble [0 × 3]>
```

```
eel_rs2 %>%
  tune::show_best(metric = "accuracy") %>%
  knitr::kable()
```

min_n	tree_depth	loss_reduction	.metric	.estimator	mean	n	std_err	.config
18	7	0.1439623	accuracy	binary	0.8412772	10	0.0102077	Preprocessor1_Model15
3	11	0.0000000	accuracy	binary	0.8384592	10	0.0134773	Preprocessor1_Model08
6	15	0.0000015	accuracy	binary	0.8342350	10	0.0135843	Preprocessor1_Model14
3	8	0.0000019	accuracy	binary	0.8341734	10	0.0128348	Preprocessor1_Model18
4	5	0.2273498	accuracy	binary	0.8313991	10	0.0135351	Preprocessor1_Model12

```
eel_best_trees <- eel_rs2 %>%
  tune::select_best("accuracy")

knitr::kable(eel_best_trees)
```

min_n	tree_depth	loss_reduction	.config
18	7	0.1439623	Preprocessor1_Model15

```
eel_model2 <- eel_spec2 %>%
  finalize_model(eel_best_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.

```
eel_spec3 <- parsnip::boost_tree(mode = "classification",
                                 engine = "xgboost",
                                 trees = 3000,
                                 learn_rate = eel_best_learn$learn_rate,
                                 min_n = eel_best_trees$min_n,
                                 tree_depth = eel_best_trees$tree_depth,
                                 mtry = tune(),
                                 loss_reduction = eel_best_trees$loss_reduction,
                                 sample_size = tune(),
                                 stop_iter = tune()
                                 )
```

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

```
eel_params2 <- dials::parameters(
  finalize(mtry(),select(baked_eel,-Angaus)),
  sample_size = sample_prop(c(.4, .9)),
  stop_iter())

eel_grid3 <- dials::grid_max_entropy(eel_params2, size = 30)
knitr::kable(head(eel_grid3))
```

mtry	sample_size	stop_iter
14	0.7549716	19
4	0.5533700	7
13	0.8496896	12
3	0.4478160	13
11	0.8950759	20
9	0.5073687	20

```
wf_eel_tune3 <- workflow() %>%
  add_recipe(eel_rec) %>%
  add_model(eel_spec3)
```

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

```
set.seed(123)
tic()
doParallel::registerDoParallel()

eel_rs3 <- tune_grid(
  wf_eel_tune3,
  Angaus~.,
  resamples = cv_folds,
  grid = eel_grid3
)
```

```
## Warning: The `...` are not used in this function but one or more objects were
## passed: ''
```

```
toc()
```

```
## 50.336 sec elapsed
```

```
eel_rs3
```

```
## # Tuning results
## # 10-fold cross-validation using stratification
## # A tibble: 10 × 4
##    splits           id     .metrics          .notes
##    <list>           <chr>  <list>            <list>
##  1 <split [628/71]> Fold01 <tibble [60 × 7]> <tibble [0 × 3]>
##  2 <split [629/70]> Fold02 <tibble [60 × 7]> <tibble [0 × 3]>
##  3 <split [629/70]> Fold03 <tibble [60 × 7]> <tibble [0 × 3]>
##  4 <split [629/70]> Fold04 <tibble [60 × 7]> <tibble [0 × 3]>
##  5 <split [629/70]> Fold05 <tibble [60 × 7]> <tibble [0 × 3]>
##  6 <split [629/70]> Fold06 <tibble [60 × 7]> <tibble [0 × 3]>
##  7 <split [629/70]> Fold07 <tibble [60 × 7]> <tibble [0 × 3]>
##  8 <split [629/70]> Fold08 <tibble [60 × 7]> <tibble [0 × 3]>
##  9 <split [630/69]> Fold09 <tibble [60 × 7]> <tibble [0 × 3]>
## 10 <split [630/69]> Fold10 <tibble [60 × 7]> <tibble [0 × 3]>
```

```
eel_rs3 %>%
  tune::show_best(metric = "accuracy") %>%
  knitr::kable()
```

mtry	sample_size	stop_iter	.metric	.estimator	mean	n	std_err	.config
14	0.7263510	13	accuracy	binary	0.8170697	10	0.0121699	Preprocessor1_Model20
9	0.7459334	18	accuracy	binary	0.8141297	10	0.0125233	Preprocessor1_Model26
16	0.8406406	7	accuracy	binary	0.8127644	10	0.0118827	Preprocessor1_Model17

mtry	sample_size	stop_iter	.metric	.estimator	mean	n	std_err	.config
6	0.7097034	4	accuracy	binary	0.8127017	10	0.0106736	Preprocessor1_Model09
5	0.8536800	4	accuracy	binary	0.8113347	10	0.0101579	Preprocessor1_Model22

```
eel_best_stoch <- eel_rs3 %>%
  tune::select_best("accuracy")

knitr::kable(eel_best_stoch)
```

mtry	sample_size	stop_iter	.config
14	0.726351	13	Preprocessor1_Model20

```
eel_model3 <- eel_spec3 %>%
  finalize_model(eel_best_stoch)
```

Finalize workflow and make final prediction

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

```
eel_final_spec <- parsnip::boost_tree(mode = "classification",
                                engine = "xgboost",
                                trees = 3000,
                                learn_rate = eel_best_learn$learn_rate,
                                min_n = eel_best_trees$min_n,
                                tree_depth = eel_best_trees$tree_depth,
                                mtry = eel_best_stoch$mtry,
                                loss_reduction = eel_best_trees$loss_reduction,
                                stop_iter = eel_best_stoch$stop_iter,
                                sample_size = eel_best_stoch$sample_size
                                )

wf_eel_final <- workflow() %>%
  add_recipe(eel_rec) %>%
  add_model(eel_final_spec)

final_simple_fit <- wf_eel_final %>% # fit to just training data (need for later)
  fit(data = eel_train)

final_eel_fit <- last_fit(eel_final_spec, Angaus~., split) # does training fit then fina
l prediction as well
final_eel_fit$.predictions
```

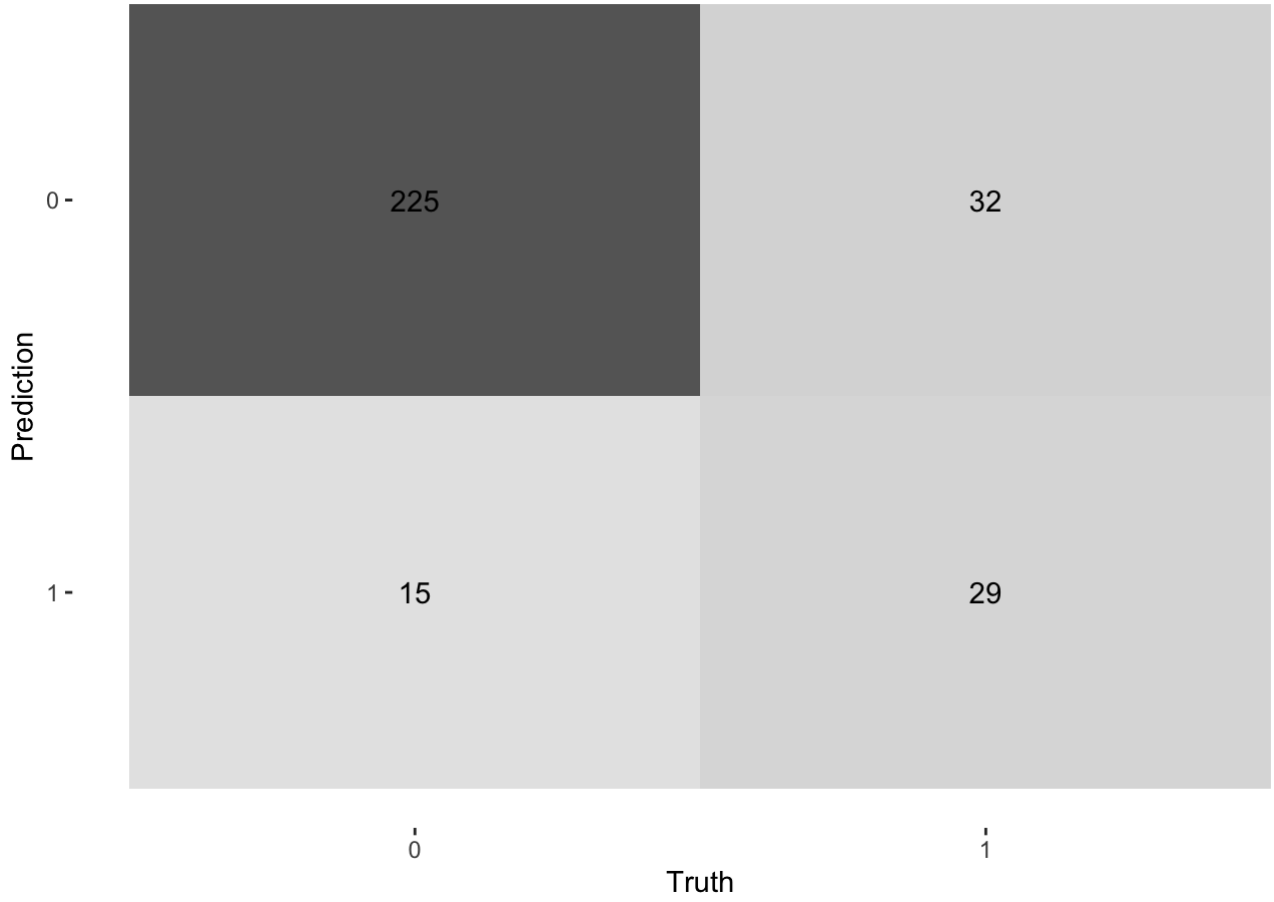
```
## [[1]]
## # A tibble: 301 × 6
##    .pred_0 .pred_1  .row .pred_class Angaus .config
##      <dbl>   <dbl> <int> <fct>       <fct>  <chr>
##  1  0.995  0.00502     1 0           0      Preprocessor1_Model1
##  2  0.0830 0.917       2 1           1      Preprocessor1_Model1
##  3  0.629  0.371       3 0           0      Preprocessor1_Model1
##  4  0.670  0.330       4 0           0      Preprocessor1_Model1
##  5  0.796  0.204       9 0           0      Preprocessor1_Model1
##  6  0.296  0.704      10 1           1      Preprocessor1_Model1
##  7  0.0991 0.901      13 1           1      Preprocessor1_Model1
##  8  0.992  0.00834    14 0           0      Preprocessor1_Model1
##  9  0.972  0.0282     21 0           0      Preprocessor1_Model1
## 10  0.827  0.173      24 0           0      Preprocessor1_Model1
## # … with 291 more rows
```

```
final_eel_fit$.metrics
```

```
## [[1]]
## # A tibble: 2 × 4
##   .metric  .estimator .estimate .config
##   <chr>    <chr>          <dbl> <chr>
## 1 accuracy binary         0.844 Preprocessor1_Model1
## 2 roc_auc  binary         0.866 Preprocessor1_Model1
```

```
eel_test_rs <- cbind(eel_test, final_eel_fit$.predictions)
eel_test_rs <- eel_test_rs[,-1]

cm<- eel_test_rs %>% yardstick::conf_mat(truth = Angaus, estimate = .pred_class)
autoplot(cm, type = "heatmap")
```




```
tibble <- final_eel_fit %>% collect_metrics()
tibble
```

```
## # A tibble: 2 × 4
##   .metric  .estimator .estimate .config
##   <chr>    <chr>          <dbl> <chr>
## 1 accuracy binary         0.844 Preprocessor1_Model1
## 2 roc_auc  binary         0.866 Preprocessor1_Model1
```

```
final_eel_accuracy <- tibble %>%
  filter(.metric == "accuracy") %>%
  pull(.estimate)

final_eel_auc <- tibble %>%
  filter(.metric == "roc_auc") %>%
  pull(.estimate)
```

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

```
print(paste0("The model had an accuracy of ", round(final_eel_accuracy,2),
". The ROC area under the curve was ", round(final_eel_auc, 2), ". The rate of false neg
atives was ", round(cm$table[3]/nrow(eel_test), 2), ", and the rate of false positives w
as ", round(cm$table[2]/nrow(eel_test),2), "."))
```

```
## [1] "The model had an accuracy of 0.84. The ROC area under the curve was 0.87. The ra
te of false negatives was 0.11, and the rate of false positives was 0.05."
```

Fit your model the evaluation data and compare performance

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

```
# Read in eval data
eval_dat <- read_csv("eel.eval.data.csv") %>%
  rename(Angaus = Angaus_obs) %>% # rename to match previous data
  mutate(Angaus = as_factor(Angaus)) # make outcome a factor
```

```
## 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.
```

```
prediction <- final_simple_fit %>% predict(new_data = eval_dat) # generate predictions
eval_dat_pred <- cbind(eval_dat, prediction)

# Compare predicted classes to actual classes
correct_predictions <- sum(eval_dat_pred$.pred_class == eval_dat_pred$Angaus)

# Calculate accuracy
accuracy <- correct_predictions / nrow(eval_dat_pred)

# Calculate auc
eval_dat_pred$pred_num <- as.numeric(eval_dat_pred$.pred_class)
auc <- auc(eval_dat_pred$Angaus, eval_dat_pred$pred_num)
```

```
## Setting levels: control = 0, case = 1
## Setting direction: controls < cases
```

2. How does your model perform on this data?

```
print(paste0("The model had an accuracy of ", accuracy ," on these data, which isn't qui
te as good as the accuracy when applying the model to the testing data. However the diff
erence is not too extreme and seems pretty good given that the dummy classifier would be
0.744. The model had an AUC of ", round(auc[1], 2), ". This does not seem great."))
```

```
## [1] "The model had an accuracy of 0.81 on these data, which isn't quite as good as th
e accuracy when applying the model to the testing data. However the difference is not to
o extreme and seems pretty good given that the dummy classifier would be 0.744. The mode
l had an AUC of 0.66. This does not seem great."
```

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

```
print(paste0("The model here does not do as well as the model in Elith et al. which foun
d a model AUC of 0.858. My AUC of ", round(auc[1], 2) , " is fairly far off. I would gue
ss that Elith et al. did more tuning to find the optimal values, where as I was more lim
ited by computing power."))
```

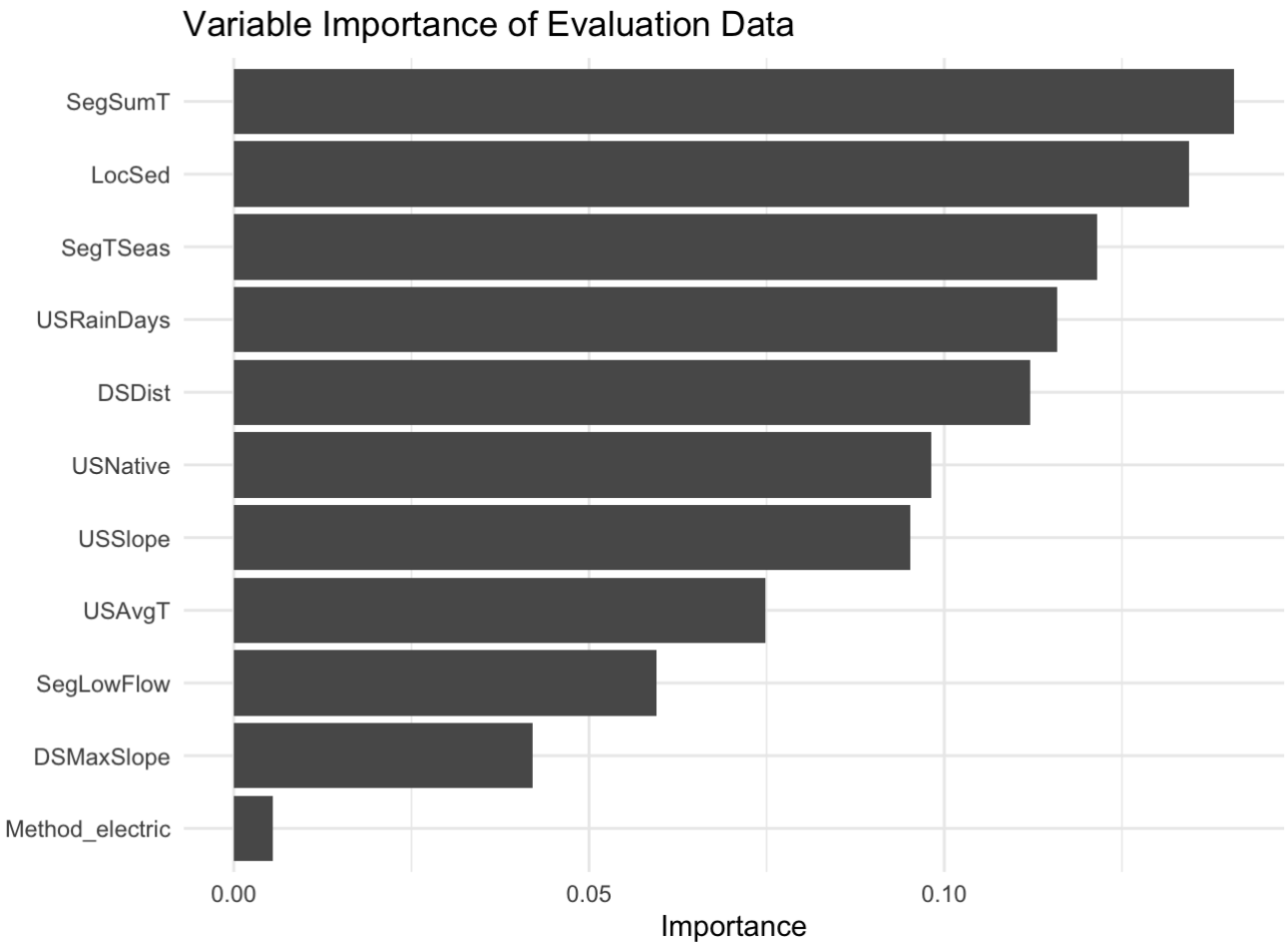
```
## [1] "The model here does not do as well as the model in Elith et al. which found a mo
del AUC of 0.858. My AUC of 0.66 is fairly far off. I would guess that Elith et al. did
more tuning to find the optimal values, where as I was more limited by computing power."
```

- Use {vip} to compare variable importance

```
set.seed(123)
orig_var_imp <- final_simple_fit %>%
  fit(data = eval_dat) %>%
  pull_workflow_fit() %>%
  vip(geom = "col", num_features = 16) +
  theme_minimal() +
  labs(title = "Variable Importance of Evaluation Data")
```

```
## Warning: `pull_workflow_fit()` was deprecated in workflows 0.2.3.
## ℹ Please use `extract_fit_parsnip()` instead.
```

```
orig_var_imp
```



- What do your variable importance results tell you about the distribution of this eel species? **The variable importance results tell us that the distribution of this eel species in the environment is highly driven by summer air temperature, consistent with Elith et al. The distribution is less driven by fishing methods.**