Lab6

AUTHOR
Guillermo Romero

PUBLISHED
March 1, 2023

```
library(tidyverse)
```

```
── Attaching core tidyverse packages ──────────────────────── tidyverse 2.0.0 ──
✓ dplyr     1.1.0     ✓ readr     2.1.4
✓ forcats   1.0.0     ✓ stringr   1.5.0
✓ ggplot2   3.4.1     ✓ tibble    3.1.8
✓ lubridate 1.9.2     ✓ tidyr     1.3.0
✓ purrr     1.0.1
── Conflicts ────────────────────────────────────────── tidyverse_conflicts() ──
✖ dplyr::filter() masks stats::filter()
✖ dplyr::lag()    masks stats::lag()
ℹ Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

```
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    1.1.3
✓ modeldata    1.1.0     ✓ workflowsets 1.0.0
✓ parsnip      1.0.4     ✓ yardstick    1.1.0
✓ recipes      1.0.5
── 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()
• Dig deeper into tidy modeling with R at https://www.tmwr.org
```

```
library(readr)
library(skimr)
library(ggpubr)
library(patchwork)
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(corrplot)
```

```
corrplot 0.92 loaded
```

```
library(flextable)
```

```

Attaching package: 'flextable'

The following objects are masked from 'package:ggpubr':

    border, font, rotate

The following object is masked from 'package:purrr':
```

```
    compose
```

```
library(baguette)
library(ranger)
library(ggplot2)
library(vip)
```

```
Attaching package: 'vip'

The following object is masked from 'package:utils':

    vi
```

```
library(here)
```

```
here() starts at C:/Users/bsf31/Documents/meds/eds232ml
```

```
library(tictoc)
library(xgboost)
```

```
Attaching package: 'xgboost'

The following object is masked from 'package:dplyr':

    slice
```

```
library(vip)
```

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

```
eel_data <- read_csv(here('eel.model.data.csv')) |>
  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.
```

```
names(eel_data)
```

```
 [1] "Angaus"     "SegSumT"    "SegTSeas"   "SegLowFlow" "DSDist"
 [6] "DSMaxSlope" "USAvgT"     "USRainDays" "USSlope"    "USNative"
[11] "DSDam"      "Method"     "LocSed"
```

```
skim(eel_data)
```

Data summary

Name	eel_data
Number of rows	1000

Number of columns	13
Column type frequency:	
character	1
numeric	12
Group variables	None

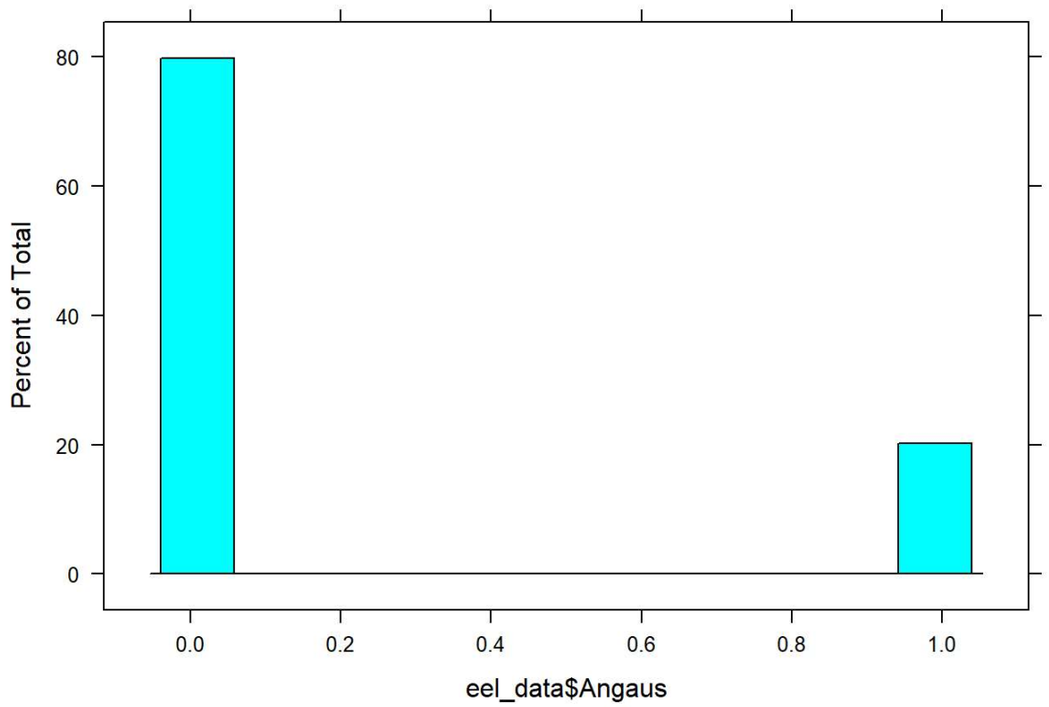
Variable type: character

skim_variable	n_missing	complete_rate	min	max	empty	n_unique	whitespace
Method	0	1	3	8	0	5	0

Variable type: numeric

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100	hist
Angaus	0	1.00	0.20	0.40	0.00	0.00	0.00	0.00	1.00	
SegSumT	0	1.00	16.42	1.80	10.10	15.20	16.50	18.10	19.60	
SegTSeas	0	1.00	0.31	1.40	-4.20	-0.34	0.54	1.30	4.09	
SegLowFlow	0	1.00	1.09	0.29	1.00	1.00	1.01	1.04	3.82	
DSDist	0	1.00	74.50	80.65	0.07	10.27	43.52	114.68	411.27	
DSMaxSlope	0	1.00	3.02	3.10	0.00	0.57	1.72	4.57	20.30	
USAvgT	0	1.00	-0.41	1.24	-6.00	-1.00	-0.10	0.39	1.89	
USRainDays	0	1.00	1.22	0.86	0.21	0.62	1.03	1.51	3.30	
USSlope	0	1.00	14.49	7.95	0.00	8.50	13.90	20.52	33.50	
USNative	0	1.00	0.57	0.37	0.00	0.17	0.66	0.96	1.00	
DSDam	0	1.00	0.17	0.37	0.00	0.00	0.00	0.00	1.00	
LocSed	176	0.82	3.79	1.35	1.00	3.00	4.00	4.80	7.00	

```
histogram(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

```
# Specify the outcome variable as a factor
eel_data$Angaus <- factor(eel_data$Angaus)
# Create training (70%) and test (30%) sets for the
set.seed(123)  # for reproducibility (random sample)
data_split <- initial_split(eel_data, prop = 0.70, strata = Angaus)
data_train <- training(data_split)
data_test  <- testing(data_split)


# 10-fold CV on the training dataset
cv_folds <- data_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

```
# Create a recipe for the data
eel_recipe <- recipe(Angaus ~ ., data = data_train) %>%
  step_normalize(all_numeric(), -all_outcomes()) %>%
  step_dummy(all_nominal(), -all_outcomes())

# Print the recipe
eel_recipe
```

```
── Recipe ──────────────────────────────────────────────────────────────────────


── Inputs

Number of variables by role

outcome:    1
predictor: 12


── Operations

• Centering and scaling for: all_numeric(), -all_outcomes()

• Dummy variables from: all_nominal(), -all_outcomes()
```

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()

```
xgb_spec <- boost_tree(
  mtry = 12, # Fixed value for mtry
  trees = 1000,  # A large number of trees to start with
  tree_depth = 4,  # Maximum depth of each tree
  min_n = 10,  # Minimum number of observations in each terminal node
  loss_reduction = 0,  # Minimum loss reduction required to make a further partition on a leaf node
  sample_size = 1,  #The size of the data set used for modeling within an iteration of the modeling algorithm
  stop_iter = 10,  # Early stopping parameter
  # update the learning rate parameter to be tuned
  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))

```
# Set up a grid for tuning the learning rate parameter
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().

```
# Create a time tracker
tic()

# Tune the model using cross-validation
xgb_res <- tune_grid(
  xgb_spec,
  resamples = cv_folds,
  grid = learn_rate_grid,
  metrics = metric_set(roc_auc, pr_auc, accuracy),
  control = control_grid(verbose = FALSE),
  preprocessor = eel_recipe
)

# Record the elapsed time
toc()
```

```
300.14 sec elapsed
```

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

```
# Show the best models and their associated tree parameter values
best_rocauc <- show_best(xgb_res, "roc_auc", n = 1)
best_pr_auc <- show_best(xgb_res, "pr_auc", n = 1)
best_acc <- show_best(xgb_res, "accuracy", n = 1)

best <- bind_rows(best_rocauc,best_pr_auc, best_acc)

theme_box(flextable(best))
```

learn_rate	.metric	.estimator	mean	n	std_err	.config
0.28965862	roc_auc	binary	0.8422070	10	0.014784903	Preprocessor1_Model29
0.01044138	pr_auc	binary	0.9585236	10	0.002940721	Preprocessor1_Model02
0.20692759	accuracy	binary	0.8127023	10	0.011724095	Preprocessor1_Model21

Tune Tree Parameters

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

```
# Create a new specification with the optimal learning rate
xgb_spec2 <- boost_tree(
  mtry = tune(),
  trees = tune(),  # Tune the number of trees
  tree_depth = tune(),  # Tune the maximum depth of each tree
  min_n = tune(),  # Tune the minimum number of observations in each terminal node
  loss_reduction = tune(),  # Tune the minimum loss reduction required to make a further partition on a leaf
  sample_size = 1,  # The size of the data set used for modeling within an iteration of the modeling algorith
  stop_iter = 10,  # Early stopping parameter
  learn_rate = 0.01044138  # Set the optimal learning rate
) %>%
```

```
  set_engine("xgboost") %>%
  set_mode("classification")
```

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

```
# Set up a grid to tune the tree parameters
tree_grid <- grid_max_entropy(
  mtry(range = c(1, 12)),
  trees(range = c(100, 2000)),
  tree_depth(range = c(3, 10)),
  min_n(range = c(1, 20)),
  loss_reduction(range = c(0, 0.1)),
  size = 100
)

# Tune the model using cross-validation
tic()
xgb_res2 <- tune_grid(
  xgb_spec2,
  resamples = cv_folds,
  grid = tree_grid,
  metrics = metric_set(roc_auc, pr_auc, accuracy),
  control = control_grid(verbose = FALSE),
  preprocessor = eel_recipe
)
toc()
```

```
982.03 sec elapsed
```

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

```
# Show the best models and their associated tree parameter values
best_rocauc2 <- show_best(xgb_res2, "roc_auc", n = 1)
best_pr_auc2 <- show_best(xgb_res2, "pr_auc", n = 1)
best_acc2 <- show_best(xgb_res2, "accuracy", n = 1)

best2 <- bind_rows(best_rocauc2,best_pr_auc2, best_acc2)

theme_box(flextable(best2))
```

mtry	trees	min_n	tree_depth	loss_reduction	.metric	.estimator	mean	n	std_err	.confi
2	574	1	7	1.119450	roc_auc	binary	0.8711348	10	0.011583911	Prepr
2	574	1	7	1.119450	pr_auc	binary	0.9654967	10	0.003299161	Prepr
10	394	3	7	1.012903	accuracy	binary	0.8456239	10	0.010184144	Prepr

```
theme_box(flextable(best_pr_auc2))
```

mtry	trees	min_n	tree_depth	loss_reduction	.metric	.estimator	mean	n	std_err	.config
2	574	1	7	1.11945	pr_auc	binary	0.9654967	10	0.003299161	Preproc

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.

```
# Create a new specification with the optimal learning rate and tree parameters
#
xgb_spec3 <- boost_tree(
  mtry = 2,
  trees = 574,  #  the number of trees
  tree_depth = 7,  #  the maximum depth of each tree
  min_n = 1,  # the minimum number of observations in each terminal node
  loss_reduction = 1.119450,  #  the minimum loss reduction required to make a further partition on a leaf no
  sample_size = tune(),  # The size of the data set used for modeling within an iteration of the modeling alg
```

```
  stop_iter = tune(),  # Early stopping parameter
  learn_rate = 0.01044138  # Set the optimal learning rate
) %>%
  set_engine("xgboost") %>%
  set_mode("classification")
```

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

```
# Set up a grid to tune the stochastic parameters
#

stochastic_grid <- grid_max_entropy(
  sample_size( range = c(0, 1)),
  stop_iter(c(5, 50)
))
```

```
# Tune the model using cross-validation
tic()
xgb_res3 <- tune_grid(
  xgb_spec3,
  resamples = cv_folds,
  grid = stochastic_grid ,
  metrics = metric_set(roc_auc, pr_auc, accuracy),
  control = control_grid(verbose = FALSE),
  preprocessor = eel_recipe
)
toc()
```

```
13.8 sec elapsed
```

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

```
# Show the best models and their associated tree parameter values
best_rocauc3 <- show_best(xgb_res3, "roc_auc", n = 1)
best_pr_auc3 <- show_best(xgb_res3, "pr_auc", n = 1)
best_acc3 <- show_best(xgb_res3, "accuracy", n = 1)

best3 <- bind_rows(best_rocauc3,best_pr_auc3, best_acc3)

theme_box(flextable(best3))
```

sample_size	stop_iter	.metric	.estimator	mean	n	std_err	.config
1	15	roc_auc	binary	0.8664657	10	0.011865011	Preprocessor1_Model1
1	15	pr_auc	binary	0.9638896	10	0.003282888	Preprocessor1_Model1
1	15	accuracy	binary	0.8427259	10	0.009127297	Preprocessor1_Model1

Finalize workflow and make final prediction

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

```
xgb_spec_final <- boost_tree(
  mtry = 2,
  trees = 574,  #  the number of trees
  tree_depth = 7,  #  the maximum depth of each tree
  min_n = 1,  # the minimum number of observations in each terminal node
  loss_reduction = 1.119450,  #  the minimum loss reduction required to make a further partition on a leaf no
  sample_size = 1,  # The size of the data set used for modeling within an iteration of the modeling algorith
  stop_iter = 15,  # Early stopping parameter
  learn_rate = 0.01044138  # Set the optimal learning rate
) %>%
  set_engine("xgboost") %>%
  set_mode("classification")
```

```
# Set up the final workflow
xgb_wf_final <- workflow() %>%
```

```
  add_model(xgb_spec_final) %>%
  add_recipe(eel_recipe)

# Fit the model using 10-fold cross-validation

tic()
final_fit <- xgb_wf_final %>%
  fit_resamples(
    resamples = cv_folds,
    metrics = metric_set(roc_auc,pr_auc, accuracy),
    control = control_resamples(save_pred = TRUE)
  )
toc()
```

```
8.16 sec elapsed
```

```
# Get the average performance across all folds
final_perf <- final_fit %>%
  collect_metrics()

theme_box(flextable(final_perf) )
```

.metric	.estimator	mean	n	std_err	.config
accuracy	binary	0.8412973	10	0.009945459	Preprocessor1_Model1
pr_auc	binary	0.9639506	10	0.003385449	Preprocessor1_Model1
roc_auc	binary	0.8662291	10	0.011864232	Preprocessor1_Model1

```
eel_recipe_test <- recipe(Angaus ~ ., data = data_test) %>%
  step_normalize(all_numeric(), -all_outcomes()) %>%
  step_dummy(all_nominal(), -all_outcomes())


xgb_wf_final_test <- workflow() %>%
  add_model(xgb_spec_final) %>%
  add_recipe(eel_recipe_test)

tic()
final_fit_test <- xgb_wf_final_test %>%
  fit_resamples(
    resamples = vfold_cv(data_test, v = 10, strata = Angaus),
    metrics = metric_set(roc_auc,pr_auc, accuracy),
    control = control_resamples(save_pred = TRUE)
  )
toc()
```

```
8.92 sec elapsed
```

```
# Get the average performance across all folds
final_perf_test <- final_fit_test %>%
  collect_metrics()

theme_box(flextable(final_perf_test) )
```

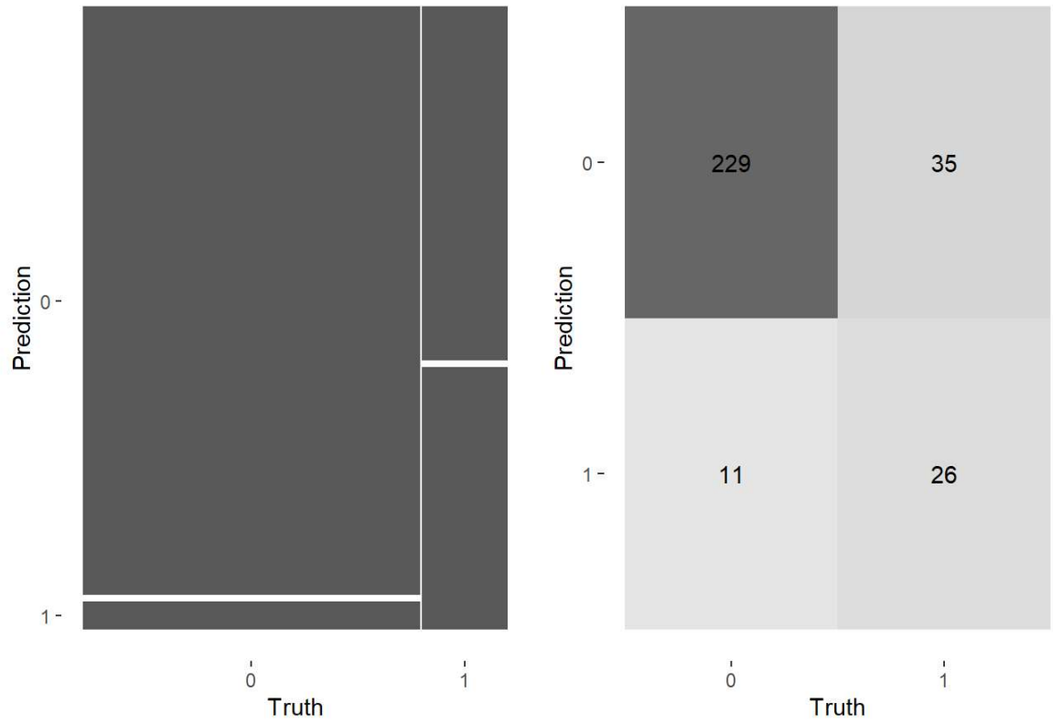
.metric	.estimator	mean	n	std_err	.config
accuracy	binary	0.8470968	10	0.015141238	Preprocessor1_Model1
pr_auc	binary	0.9676901	10	0.006031429	Preprocessor1_Model1
roc_auc	binary	0.8818452	10	0.019236512	Preprocessor1_Model1

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

```
# Compute the confusion matrix for the final model training data
final_pred <- final_fit %>%
  collect_predictions()

conf_mat(final_pred, truth = Angaus, estimate = .pred_class)
```

```
          Truth
Prediction   0   1
         0 526  79
         1  32  62
```

```
# Compute the confusion matrix for the final model test data
final_pred_test <- final_fit_test %>%
  collect_predictions()

cm_test <- conf_mat(final_pred_test, truth = Angaus, estimate = .pred_class)

autoplot(cm_test, type = "mosaic") +
autoplot(cm_test, type = "heatmap")
```



Running the model on the test data

**The confusion matrix shows the classification results of the final model. There were 26 true negatives (species not present and correctly classified), 11 false positives (species present but wrongly classified as absent), 35 false negatives (species absent but wrongly classified as present), and 229 true positives (species present and correctly classified) in the test dataset.

For the accuracy metric, the mean value is
0.8470968, which means that on average, the model correctly predicts the presence or absence of the eel species in 84.7% of cases. The standard error is 0.015141238, indicating some variation in performance across the 10 folds.

For the ROC AUC metric, the mean value is 0.8818452, which means that the model has good discriminatory power to distinguish between positive and negative cases. The standard error is 0.019236512, indicating relatively low variability in performance across the 10 folds.**

For the PR AUC metric, the mean value is 0.9676901, which means that the model has very good precision-recall tradeoff, and can achieve high precision while maintaining high recall or vice versa. The standard error is 0.006031429, indicating relatively low variability in performance across the 10 folds. This suggests that the model is consistently able to achieve high precision-recall tradeoff across different test sets.

Fit your model the evaluation data and compare performance

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

```
eel_eval_data <- read_csv(here('eel.eval.data.csv'))
```

```
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.
```

```
eel_eval_data$Angaus_obs <- factor(eel_eval_data$Angaus_obs)
```

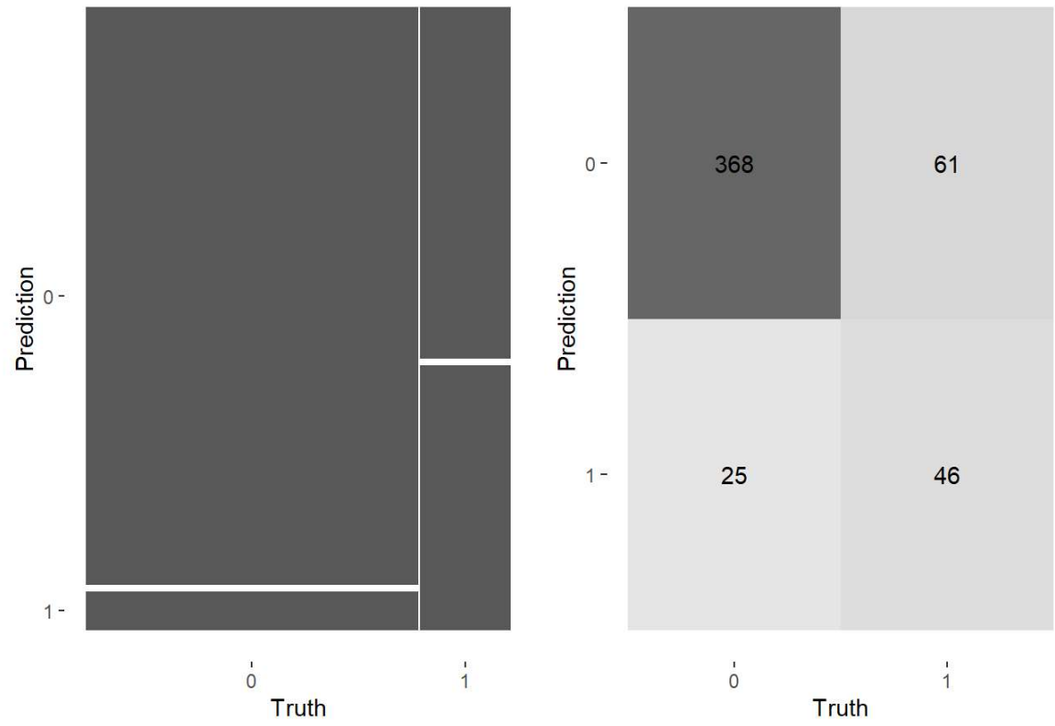
```
# Create a recipe for the data
eel_recipe_eval <- recipe(Angaus_obs ~ ., data = eel_eval_data) %>%
  step_normalize(all_numeric(), -all_outcomes()) %>%
  step_dummy(all_nominal(), -all_outcomes())


xgb_wf_final_eval <- workflow() %>%
  add_model(xgb_spec_final) %>%
  add_recipe(eel_recipe_eval)

tic()
final_fit_eval <- xgb_wf_final_eval %>%
  fit_resamples(
    resamples = vfold_cv(eel_eval_data, v = 10, strata = Angaus_obs),
    metrics = metric_set(roc_auc,pr_auc, accuracy),
    control = control_resamples(save_pred = TRUE)
  )
toc()
```

```
8.72 sec elapsed
```

```
# Get the average performance across all folds
final_perf_eval <- final_fit_eval %>%
  collect_metrics()

theme_box(flextable(final_perf_eval) )
```

.metric	.estimator	mean	n	std_err	.config
accuracy	binary	0.8280432	10	0.011533081	Preprocessor1_Model1
pr_auc	binary	0.9605975	10	0.005515549	Preprocessor1_Model1
roc_auc	binary	0.8706177	10	0.014172529	Preprocessor1_Model1

```
# Compute the confusion matrix for the final model test data
final_pred_eval <- final_fit_eval %>%
  collect_predictions()

cm_eval <- conf_mat(final_pred_eval, truth = Angaus_obs, estimate = .pred_class)

autoplot(cm_eval, type = "mosaic") +
autoplot(cm_eval, type = "heatmap")
```



Running the model on the test data

**The confusion matrix shows the classification results of the final model. There were 46 true negatives (species not present and correctly classified), 25 false positives (species present but wrongly classified as absent), 61 false negatives (species absent but wrongly classified as present), and 368 true positives (species present and correctly classified) in the test dataset.

For the accuracy metric, the mean value is
0.8280432, which means that on average, the model correctly predicts the presence or absence of the eel species in 82.8% of cases. The standard error is 0.011533081, indicating some variation in performance across the 10 folds.

For the ROC AUC metric, the mean value is 0.8706177, which means that the model has good discriminatory power to distinguish between positive and negative cases. The standard error is 0.014172529, indicating relatively low variability in performance across the 10 folds.**

For the PR AUC metric, the mean value is 0.9605975, which means that the model has very good precision-recall tradeoff, and can achieve high precision while maintaining high recall or vice versa. The standard error is 0.005515549, indicating relatively low variability in performance across the 10 folds. This suggests that the model is consistently able to achieve high precision-recall tradeoff across different test sets.

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

- Use {vip} to compare variable importance

```
# Run your final model
final_tuned <- tune_grid(
  object = xgb_wf_final_eval,
  resamples = vfold_cv(eel_eval_data, v = 10, strata = Angaus_obs),
  metrics   = metric_set(pr_auc))
```

```
Warning: No tuning parameters have been detected, performance will be evaluated
using the resamples with no tuning. Did you want to [tune()] parameters?
```

```
final_eval_final <- xgb_wf_final_eval %>%
    finalize_workflow(select_best(final_tuned, metric = "pr_auc"))
# Fitting our final workflow
tic()
final_fit = final_eval_final %>% fit(data = eel_eval_data)
toc()
```

```
0.56 sec elapsed
```

```
final_fit %>%
extract_fit_parsnip() %>%
```

```
vip(geom = "col", num_features = 13)
```

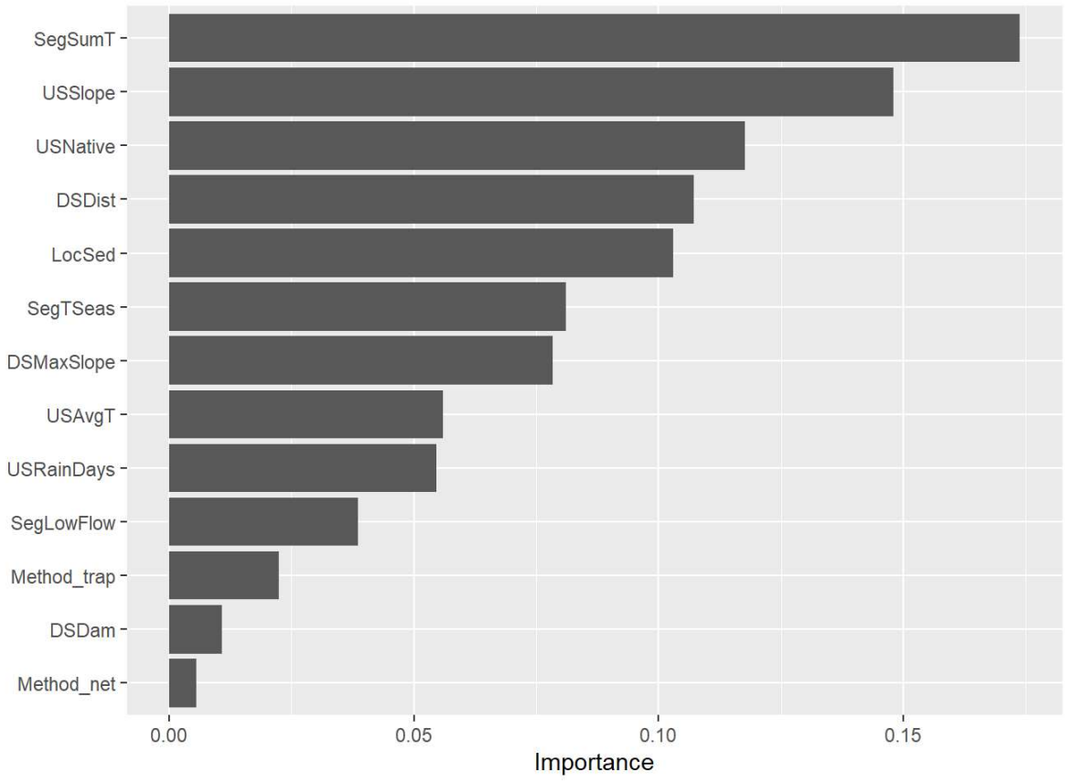


Table 2. Summary of the relative contributions (%) of predictor variables for a boosted regression tree model developed with cross-validation on data from 1000 sites using tree complexity of 5 and learning rate of 0·005

Predictor	Relative contribution (%)
SegSumT	24·7
USNative	11·3
Method	11·1
DSDist	9·7
LocSed	8·0
DSMaxSlope	7·3
USSlope	6·9
USRainDays	6·5
USAvgT	5·7
SegTSeas	5·7
SegLowFlow	2·9
DSDam	0·1

Elith, J., et al. "A Working Guide to Boosted Regression Trees." Journal of Animal Ecology, vol. 77, no. 4, July 2008, pp. 802–13, https://doi.org/10.1111/j.1365-2656.2008.01390.x.

Summer air Temperature is the greatest variable of importance in both models. There are some similarities and some drastic differences. My model places US slope as the next highest at 17.5% where they only have it at 6-9%. I then have US Native which was their second most important but the values are similar. The combined method does not reach the same level of importance in my model but the remaining variables have similar values.

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

That the distribution is largely dependent on the Summer air temperature, in areas with indigenous forest(proportion) and dependent on the average slope in the upstream. This follows the papers description

> 'The models developed for A. australis are consistent with the known ecology of the species, and accurately describe a species occurring in warm, lowland rivers in agricultural landscapes, often close to the coast but also penetrating inland, and preferring reaches with fine sediments. The modelled interactions highlight the suitability of habitats combining low flood frequencies and warm temperatures'