penguins_project

2024-07-16

```r
#loading packages
pacman::p_load(palmerpenguins, tidyverse, tidymodels, skimr, mice, rpart.plot, ranger)
```

```r
#loading data
data('penguins')

#check for any leftover NA values
skim(penguins)
```

Table 1: Data summary

Name	penguins
Number of rows	344
Number of columns	7
Column type frequency:	
factor	3
numeric	4
Group variables	None

Variable type: factor

skim_variable	n_missing	complete_rate	ordered	n_unique	top_counts
species	0	1.00	FALSE	3	Ade: 152, Gen: 124, Chi: 68
island	0	1.00	FALSE	3	Bis: 168, Dre: 124, Tor: 52
sex	11	0.97	FALSE	2	mal: 168, fem: 165

Variable type: numeric

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100	hist
bill_length_mm	2	0.99	43.92	5.46	32.1	39.23	44.45	48.5	59.6	
bill_depth_mm	2	0.99	17.15	1.97	13.1	15.60	17.30	18.7	21.5	
flipper_length_mm	2	0.99	200.92	14.06	172.0	190.00	197.00	213.0	231.0	
body_mass_g	2	0.99	4201.75	801.95	2700.0	3550.00	4050.00	4750.0	6300.0	

```r
#replacing sex data
imputed_data = mice(penguins, method = 'pmm', m = 5, maxit = 50, seed = 500)
```

```r
#Inspect the imputed data
summary(imputed_data)
```

```
## Class: mids
## Number of multiple imputations:  5
## Imputation methods:
##           species            island    bill_length_mm     bill_depth_mm
##                ""                ""             "pmm"             "pmm"
## flipper_length_mm       body_mass_g               sex
##             "pmm"             "pmm"             "pmm"
## PredictorMatrix:
##                   species island bill_length_mm bill_depth_mm flipper_length_mm
## species                 0      1              1             1                 1
## island                  1      0              1             1                 1
## bill_length_mm          1      1              0             1                 1
## bill_depth_mm           1      1              1             0                 1
## flipper_length_mm       1      1              1             1                 0
## body_mass_g             1      1              1             1                 1
##                   body_mass_g sex
## species                     1   1
## island                      1   1
## bill_length_mm              1   1
## bill_depth_mm               1   1
## flipper_length_mm           1   1
## body_mass_g                 0   1
```

```r
# Create a complete dataset with imputed values
penguins_complete = complete(imputed_data)
skim(penguins_complete)
```

Table 4: Data summary

Name	penguins_complete
Number of rows	344
Number of columns	7
Column type frequency:	
factor	3
numeric	4
Group variables	None

Variable type: factor

skim_variable	n_missing	complete_rate	ordered	n_unique	top_counts
species	0	1	FALSE	3	Ade: 152, Gen: 124, Chi: 68
island	0	1	FALSE	3	Bis: 168, Dre: 124, Tor: 52
sex	0	1	FALSE	2	fem: 173, mal: 171

Variable type: numeric

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100	hist
bill_length_mm	0	1	43.92	5.46	32.1	39.2	44.45	48.50	59.6	
bill_depth_mm	0	1	17.15	1.97	13.1	15.6	17.30	18.70	21.5	
flipper_length_mm	0	1	200.97	14.12	172.0	190.0	197.00	213.25	231.0	
body_mass_g	0	1	4203.71	805.81	2700.0	3550.0	4050.00	4756.25	6300.0	

```r
#starting from the top w/imputation using tidymodels
#defining a basic recipe
basic_recipe =
  recipe(species ~ ., data = penguins) |>
  step_impute_median(all_numeric_predictors()) |>
  step_impute_mode(all_factor_predictors())

#prepping data --> IE looking at data before imputation
basic_prep = basic_recipe |> prep()

#going through with steps --> baking up dataset
basic_bake = basic_prep |> bake(new_data = NULL)

#growing short tree by hand
table(penguins$island, penguins$species)
```

```
##
##             Adelie Chinstrap Gentoo
##   Biscoe        44         0    124
##   Dream         56        68      0
##   Torgersen     52         0      0
```

```r
#split the data based on the island
biscoe_counts = c(44, 0, 78)  # Counts for Adelie, Chinstrap, Gentoo on Biscoe
dream_torgersen_counts = c(56 + 52, 68, 0)  # Combined counts for Dream and Torgersen

#total counts for each node
n_biscoe = sum(biscoe_counts)
n_dream_torgersen = sum(dream_torgersen_counts)

#total penguins
n_total = n_biscoe + n_dream_torgersen

#proportions for each species in each node
p_adelie_biscoe = biscoe_counts[1] / n_biscoe
p_chinstrap_biscoe = biscoe_counts[2] / n_biscoe
p_gentoo_biscoe = biscoe_counts[3] / n_biscoe

p_adelie_dream_torgersen = dream_torgersen_counts[1] / n_dream_torgersen
p_chinstrap_dream_torgersen = dream_torgersen_counts[2] / n_dream_torgersen
p_gentoo_dream_torgersen = dream_torgersen_counts[3] / n_dream_torgersen

#gini impurity for each node
gini_biscoe = 1 - (p_adelie_biscoe^2 + p_chinstrap_biscoe^2 + p_gentoo_biscoe^2)
gini_dream_torgersen = 1 - (p_adelie_dream_torgersen^2 + p_chinstrap_dream_torgersen^2 + p_gentoo_dream_
#weighted Gini impurity for the split
weighted_gini_island = (n_biscoe * gini_biscoe + n_dream_torgersen * gini_dream_torgersen) / n_total

#print results
gini_biscoe
```

```
## [1] 0.4611664
```

```
gini_dream_torgersen
```

```
## [1] 0.4741736
```

```
weighted_gini_island
```

```
## [1] 0.4688485
```

```
#growing tree by hand using sex
table(penguins$sex, penguins$species)
```

```
##
##          Adelie Chinstrap Gentoo
##   female     73        34     58
##   male       73        34     61
```

```
#split the data based on the sex
female_counts = c(73, 34, 34)  # Counts for Adelie, Chinstrap, Gentoo for females
male_counts = c(73, 34, 44)  # Counts for Adelie, Chinstrap, Gentoo for males

# Total counts for each node
n_female = sum(female_counts)
n_male = sum(male_counts)

# Total penguins (excluding NA values)
n_total = n_female + n_male

# Proportions for each species in each node
p_adelie_female = female_counts[1] / n_female
p_chinstrap_female = female_counts[2] / n_female
p_gentoo_female = female_counts[3] / n_female

p_adelie_male = male_counts[1] / n_male
p_chinstrap_male = male_counts[2] / n_male
p_gentoo_male = male_counts[3] / n_male

# Gini impurity for each node
gini_female = 1 - (p_adelie_female^2 + p_chinstrap_female^2 + p_gentoo_female^2)
gini_male = 1 - (p_adelie_male^2 + p_chinstrap_male^2 + p_gentoo_male^2)

# Weighted Gini impurity for the split
weighted_gini_sex = (n_female * gini_female + n_male * gini_male) / n_total

# Print results
gini_female
```

```
## [1] 0.6156632
```

```
gini_male
```

```
## [1] 0.6306741
```

```
weighted_gini_sex
```

```
## [1] 0.6234257
```

```
#specify a decision tree model with a tuning parameter for cost complexity
tree_model = decision_tree(cost_complexity = tune()) %>%
  set_mode("classification") %>%
  set_engine("rpart")

#create a workflow
penguins_workflow = workflow() %>%
  add_recipe(basic_recipe) %>%
  add_model(tree_model)

#set up cross-validation with 5 folds
set.seed(123)
penguins_folds = vfold_cv(penguins, v = 5)

#set up a grid of values for cost complexity
cost_complexity_grid = grid_regular(cost_complexity(), levels = 10)

#tune the model using cross-validation
set.seed(123)
tune_results = tune_grid(
  penguins_workflow,
  resamples = penguins_folds,
  grid = cost_complexity_grid,
  metrics = metric_set(accuracy)
)

#collect the best tuning parameters
best_params = select_best(tune_results, metric = "accuracy")

#finalize the workflow with the best parameters
final_workflow = penguins_workflow %>%
  finalize_workflow(best_params)

#split the data into training and testing sets
set.seed(123)
penguins_split = initial_split(penguins, prop = 0.8)
penguins_train = training(penguins_split)
penguins_test = testing(penguins_split)

#fit the final workflow on the training data
final_fit = final_workflow %>%
  fit(data = penguins_train)
```

```r
#make predictions on the test data
penguins_predictions = predict(final_fit, new_data = penguins_test) %>%
  bind_cols(penguins_test)

#evaluate the model's performance
final_metrics = penguins_predictions %>%
  metrics(truth = species, estimate = .pred_class)

final_conf_mat = penguins_predictions %>%
  conf_mat(truth = species, estimate = .pred_class)

#print the metrics and confusion matrix
print(final_metrics)
```

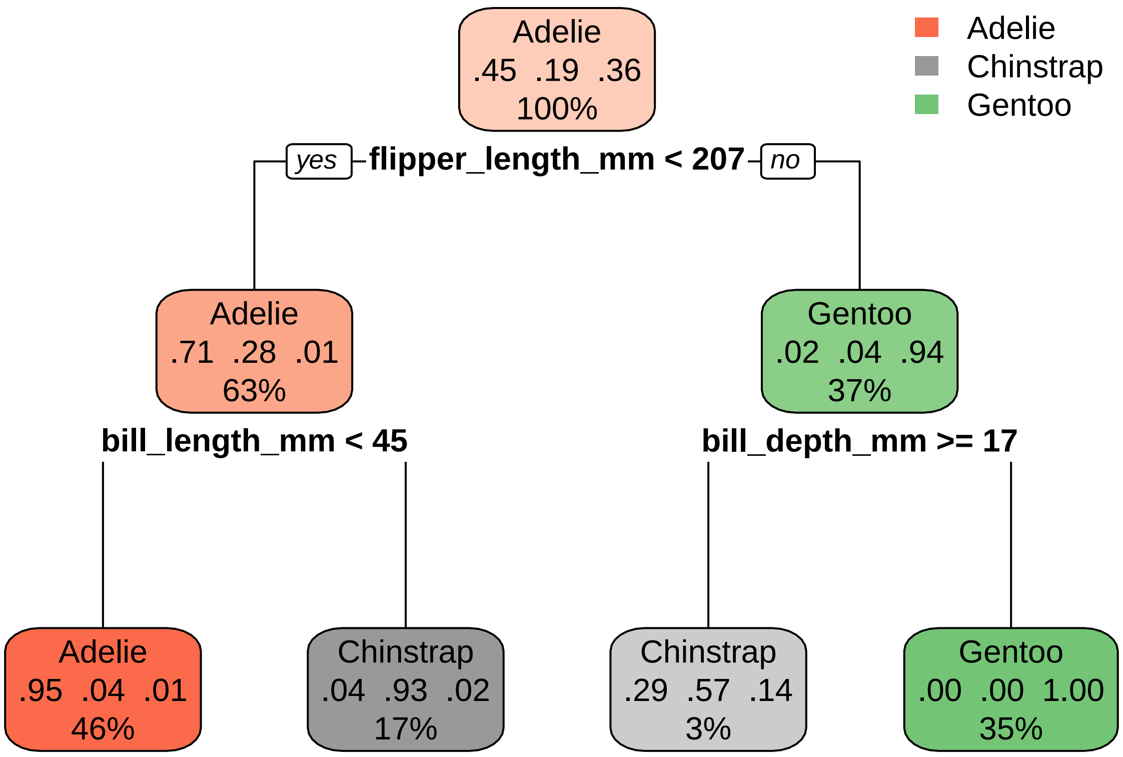
```
## # A tibble: 2 x 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy multiclass     0.957
## 2 kap      multiclass     0.933
```

```r
print(final_conf_mat)
```

```
##            Truth
## Prediction  Adelie Chinstrap Gentoo
##   Adelie        27         1      0
##   Chinstrap      1        15      1
##   Gentoo         0         0     24
```

```r
#visualize the final decision tree
final_tree = extract_fit_engine(final_fit)
rpart.plot(final_tree)
```

```
## Warning: Cannot retrieve the data used to build the model (so cannot determine roundint and is.binar
## To silence this warning:
##     Call rpart.plot with roundint=FALSE,
##     or rebuild the rpart model with model=TRUE.
```



```r
# Specify a random forest model with tuning parameters for mtry, min_n, and trees
rf_model = rand_forest(
  mtry = tune(),
  min_n = tune(),
  trees = tune()
) %>%
  set_mode("classification") %>%
  set_engine("ranger")

# Create a workflow
rf_workflow = workflow() %>%
  add_recipe(basic_recipe) %>%
  add_model(rf_model)

#using 5fold CV from last time

# Set up a grid of values for mtry, min_n, and trees
rf_grid = grid_regular(
  mtry(range = c(1, 5)),
  min_n(range = c(2, 10)),
  trees(range = c(50, 200)),
  levels = 5
)

#tuning model
set.seed(123)
```

```r
rf_results = tune_grid(
  rf_workflow,  # Use the correct workflow variable
  resamples = penguins_folds,
  grid = rf_grid,  # Use the correct grid variable
  metrics = metric_set(accuracy)
)

# Collect the best tuning parameters
best_params = select_best(rf_results, metric = "accuracy")

# Finalize the workflow with the best parameters
final_workflow = rf_workflow %>%
  finalize_workflow(best_params)

# Split the data into training and testing sets
set.seed(123)
penguins_split = initial_split(penguins, prop = 0.8)
penguins_train = training(penguins_split)
penguins_test = testing(penguins_split)

# Fit the final workflow on the training data
final_fit = final_workflow %>%
  fit(data = penguins_train)

# Make predictions on the test data
penguins_predictions = predict(final_fit, new_data = penguins_test) %>%
  bind_cols(penguins_test)

# Evaluate the model's performance
final_metrics = penguins_predictions %>%
  metrics(truth = species, estimate = .pred_class)

final_conf_mat = penguins_predictions %>%
  conf_mat(truth = species, estimate = .pred_class)

# Print the metrics and confusion matrix
print(final_metrics)
```

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

```r
print(final_conf_mat)
```

```
##            Truth
## Prediction  Adelie Chinstrap Gentoo
##   Adelie        28         0      0
##   Chinstrap      0        16      0
##   Gentoo         0         0     25
```

```r
#one more time RF with 'fancy' approaches to imputation
#RF fancy recipe
fancy_recipe =
  recipe(species ~ ., data = penguins) |>
  step_impute_mean(all_numeric_predictors()) |>
  step_impute_knn(all_factor_predictors())

#RF fancy workflow
rf_fancy_workflow =  workflow() %>%
  add_recipe(fancy_recipe) %>%
  add_model(rf_model)

#tuning model
set.seed(123)
rf_fancy_results = tune_grid(
  rf_fancy_workflow,  # Use the correct workflow variable
  resamples = penguins_folds,
  grid = rf_grid,  # Use the correct grid variable
  metrics = metric_set(accuracy)
)

#grabbing best tuning parameters
best_fancy_params = select_best(rf_fancy_results, metric = "accuracy")

#final workflow w/best parameters
final_fancy_workflow = rf_fancy_workflow %>%
  finalize_workflow(best_fancy_params)

#fit workflow on training data
final_fancy_fit = final_fancy_workflow %>%
  fit(data = penguins_train)

#make predictions on the test data
penguins_fancy_predictions = predict(final_fancy_fit, new_data = penguins_test) %>%
  bind_cols(penguins_test)

#evaluate the model's performance
final_fancy_metrics = penguins_predictions %>%
  metrics(truth = species, estimate = .pred_class)

final_fancy_conf_mat = penguins_predictions %>%
  conf_mat(truth = species, estimate = .pred_class)

#print results
print(final_fancy_metrics)
```

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

```r
print(final_fancy_conf_mat)
```

```
##             Truth
## Prediction  Adelie Chinstrap Gentoo
##   Adelie        28         0      0
##   Chinstrap      0        16      0
##   Gentoo         0         0     25
```