Regression

2024-09-26

library(readr)  
library(tidymodels)  
library(usemodels)

# Data

honey <- read\_csv("Honey.csv")

# Explore the data

honey  
unique(honey$Pollen\_analysis)  
  
str(honey)  
summary(honey)  
skimr::skim(honey)  
  
honey |>   
 group\_by(Pollen\_analysis) |>   
 count()

We will filter the first three Pollen to reduce the computing time since the data is very large

honey <- honey |>  
 select(-Price) |> # drop the price variable  
 filter(Pollen\_analysis %in% c("Acacia","Alfalfa","Avocado"))

# Check the distribution of the target variable

ggplot(honey, aes(x = Purity, y=after\_stat(density))) +   
 geom\_histogram() +  
 geom\_density()

It is obvious the target variable is not normally distributed. This a signal that we have to be careful how we partition the data into training and test sets

# Correlation of the variables

honey |>   
 select(-Pollen\_analysis) |>  
 correlations() |> round(digits = 2)

Set the random number stream using set.seed() so that the results can be reproduced later. Split a balanced data (80/20). The data split is done using stratified sampling

set.seed(202)  
honey\_split <- initial\_split(honey, prop = 0.80, strata = Purity)  
honey\_train <- training(honey\_split)  
honey\_test <- testing(honey\_split)

dim(honey\_train)  
dim(honey\_test)

The function strata divides the strata variable by four (by default) and samples within each stratum. The default number (four) can be changed using the breaks function.The change is necessary if the distribution of the test set is different from the training set

# Perform histogram with the split

train <- ggplot(honey\_train, aes(x = Purity, y=after\_stat(density))) +   
 geom\_histogram() +  
 geom\_density()+  
 labs(title = "Train")  
  
test <- ggplot(honey\_test, aes(x = Purity, y=after\_stat(density))) +   
 geom\_histogram() +  
 geom\_density()+  
 labs(title = "Test")  
gridExtra::grid.arrange(train, test, ncol=2)

# re-sampling the training set using 10 fold CV (stratified sampling)

set.seed(200)  
  
honey\_CV <- vfold\_cv(honey\_train, v = 10,   
 strata = Purity, repeats = 1)  
honey\_CV

For a more stable and reliable model, repeated cross validation is recommended. For the number of times to repeat the k-fold, there is no one size fit it all.Up to 10 repeats is usually recommended for a stable model. This configuration is used in order to reduce computing time.

# Preprocessing the data using the recipe function

The follow steps are necessary for this data

honey\_recipe <-   
 recipe(Purity ~., data = honey\_train) |>   
 step\_scale(all\_numeric\_predictors()) |>   
 step\_center(all\_numeric\_predictors()) |>   
 step\_dummy(all\_nominal\_predictors()) |>   
 step\_zv(all\_predictors())

step\_zv`Deletes any zero-variance predictors that have a single unique value. In case there is a factor level that was never observed in the training data (resulting in a column of all 0s).

summary(honey\_recipe)

# Specifying the model

STEPS

1. Choose a model
2. Specify an engine
3. Set the mode

Some regression algorithms present in tidymodels algorithms are random forest,ordinary least squares, Linear support vector machines (SVMs),Boosted trees, etc. We are going to demonstrate with the random forest algorithm

# Specify Engine

rf\_model <- rand\_forest(trees = 2000, min\_n = tune(),mtry = tune()) |>   
 set\_engine("ranger", verbose = TRUE) |>   
 set\_mode("regression")

The hyperparameters in random forest are trees,mtry and min\_n

# Setting the workflow   
rf\_wflow <-   
 workflow() |>   
 add\_model(rf\_model) |>   
 add\_recipe(honey\_recipe)

# Setting evaluation metrics

Set evaluation metrics for regression \* Root Mean square Error (RMSE) \* R-Square \* Mean Absolute Error (MAE)

rf.reg\_metric <- metric\_set(rmse, rsq, mae)

# Tune hyperparameters

rf\_grid <- grid\_regular(  
 mtry(range = c(1, 5)),  
 min\_n(range = c(2, 10)),  
 levels = 5  
)

# Control aspects of the grid search process

ctrl <- control\_resamples(save\_pred = TRUE, verbose = TRUE)

# Search grid

set.seed(203)  
start.time <- Sys.time()  
honey\_rf\_model <-   
 tune\_grid(  
 rf\_wflow,  
 resamples = honey\_CV,  
 control = ctrl,   
 metrics = rf.reg\_metric  
 )  
end.time <- Sys.time()  
time.taken <- end.time - start.time  
time.taken

The best hyperparameter configuration using rmse is 2000 tress, 7 mtry, and 10 min\_n. These parameters will be used to fit the final model. The run run time is 2.743102 hours for 10 fold repeated once. Therefore, it would take about 27.43102 hours to repeat the 10 fold ten times on my machine.32GBRAM, 1.5GB SSD, 2.1GHz Core i7

# View results

collect\_metrics(honey\_rf\_model)  
  
collect\_predictions(honey\_rf\_model)

# Find the best hyperparameter configuration to fit the final model

honey\_rf\_model |>   
 show\_best(metric = "rsq")   
  
honey\_rf\_model |>   
 select\_best(metric = "rsq")  
  
honey\_rf\_model |>   
 select\_best(metric = "mae")  
  
best\_params <- honey\_rf\_model |>   
 select\_best(metric = "rmse")

The best hyperparameters configuration based on rmse is 2000 trees, 7 mtry and 10 min\_n. These hyperparameters are used to fit the final model.

# Finalising the model

#Work flow method  
final\_workflow <- rf\_wflow |>   
 finalize\_workflow(best\_params)  
  
final\_workflow  
  
set.seed(204)

# Train the final model with the best hyperparameters

final\_model <- fit(final\_workflow, data = honey\_train)

# Parameter estimate - Train set

augment(final\_model, honey\_train) |>   
 rf.reg\_metric(truth = Purity, estimate = .pred)

rmse is 0.0110 r squared is 0.994 mae is 0.00288

# Predict with test set

set.seed(205)  
result <- final\_workflow |>   
 last\_fit(honey\_split, metrics = rf.reg\_metric)  
  
result

Note that honest\_split contains both training and test sets

# plot the actual and the predicted

library(probably)  
result |>   
 collect\_predictions() |>   
 cal\_plot\_regression(  
 truth = Purity,   
 estimate = .pred)

# Collect the evaluation metrics

result |> collect\_metrics()

# Collect the predictions

mod\_pred <- collect\_predictions(result)

Alternatively, Extract model to make prediction with the test set

pred <- augment(final\_model, honey\_test)  
  
augment(final\_model, honey\_test) |>   
 rf.reg\_metric(truth = Purity, estimate = .pred)

rmse is 0.0197 r squared is 0.980 mae is 0.00536