EST-25134: Aprendizaje Estadístico

Profesor: Alfredo Garbuno Iñigo — Primavera, 2023 — Flujo de diagnóstico.
Objetivo: Que veremos.
Lectura recomendada: Capítulo 10 de [1] y capítulo 15 de [2].

1. INTRODUCCIÓN

Hemos discutido ya sobre distintos modelos y cómo cada modelo tiene distintas necesidades para pre-procesar los datos antes de realizarse el ajuste. En el capítulo de 10 de Kuhn and Johnson [1] se ajustan varios modelos para predecir la capacidad de compresión de mezclas de concreto en función de los ingredientes que se utiliza para cada mezcla. Las preguntas en contreto que resolveremos en esta sección son:

> ¿Cómo podemos comparar distintos modelos entre si? ¿Cómo podemos utilizar un flujo de trabajo que nos ayude a hacerlo de manera eficiente?

Los datos que usaremos para ilustrar estos conceptos son los mismos que usan [1] donde lo que nos interesa es predecir `compressive_strength` y las unidades son kilogramos por metro cúbico.

```
library(tidymodels)
data(concrete, package = "modeldata")
concrete |> print(n = 3, width = 70)
```

```
# A tibble: 1,030 × 9
  cement blast_...¹f fly_ash water ...²super ...³coars fine_...   age ...compr
   <dbl>      <dbl>   <dbl> <dbl>    <dbl>    <dbl>    <dbl> <int>   <dbl>
1   540          0        0   162      2.5     1040      676    28    80.0
2   540          0        0   162      2.5     1055      676    28    61.9
3   332.       142.       0   228      0        932      594   270    40.3
# ... with 1,027 more rows, and abbreviated variable names
# ¹ blast_furnace_slag, ²superplasticizer, ³coarse_aggregate,
#   fine_aggregate, compressive_strength
#  Use `print(n = ...)` to see more rows
```

En particular para estos datos tenemos mezclas que se probaron varias veces por lo tanto reduciremos un poco esta multiplicidad.

```
concrete <-
  concrete %>%
  group_by(across(-compressive_strength)) %>%
  summarize(compressive_strength = mean(compressive_strength),
            .groups = "drop")
```

Prepararemos nuestros conjuntos de entrenamiento y prueba

```
set.seed(1501)
concrete_split <- initial_split(concrete, strata = compressive_strength)
concrete_train <- training(concrete_split)
concrete_test  <- testing(concrete_split)

set.seed(1502)
concrete_folds <-
  vfold_cv(concrete_train, strata = compressive_strength, repeats = 5)
```

Usaremos algunas preparaciones de datos, pues hay modelos (no todos) que las requieren

```
normalized_rec <-
  recipe(compressive_strength ~ ., data = concrete_train) %>%
  step_normalize(all_predictors())

poly_recipe <-
  normalized_rec %>%
  step_poly(all_predictors()) %>%
  step_interact(~ all_predictors():all_predictors())
```

Preparemos nuestras especificaciones de modelos

```
library(rules)
library(baguette)

linear_reg_spec <-
  linear_reg(penalty = tune(), mixture = tune()) %>%
  set_engine("glmnet")

mars_spec <-
  mars(prod_degree = tune()) %>%  #<- use GCV to choose terms
  set_engine("earth") %>%
  set_mode("regression")
```

```
cart_spec <-
  decision_tree(cost_complexity = tune(), min_n = tune()) %>%
  set_engine("rpart") %>%
  set_mode("regression")

bag_cart_spec <-
  bag_tree() %>%
  set_engine("rpart", times = 50L) %>%
  set_mode("regression")
```

```
knn_spec <-
  nearest_neighbor(neighbors = tune(),
                   dist_power = tune(),
                   weight_func = tune()) %>%
  set_engine("kknn") %>%
  set_mode("regression")
```

```
rf_spec ←
  rand_forest(mtry = tune(), min_n = tune(), trees = 1000) %>%
  set_engine("ranger") %>%
  set_mode("regression")

xgb_spec ←
  boost_tree(tree_depth = tune(), learn_rate = tune(),
             loss_reduction = tune(),
             min_n = tune(), sample_size = tune(),
             trees = tune()) %>%
  set_engine("xgboost") %>%
  set_mode("regression")

cubist_spec ←
  cubist_rules(committees = tune(), neighbors = tune()) %>%
  set_engine("Cubist")
```

2. SELECCIÓN Y COMPARACIÓN

2.1. Flujo de procesamiento

```
normalized ←
  workflow_set(
    preproc = list(normalized = normalized_rec),
    models = list(KNN = knn_spec)
  )
normalized
```

```
# A workflow set/tibble: 1 × 4
  wflow_id       info             option    result
  <chr>          <list>           <list>    <list>
1 normalized_KNN <tibble [1 × 4]> <opts[0]> <list [0]>
```

Podemos corroborar que tenemos lo usual

```
normalized %>% extract_workflow(id = "normalized_KNN")
```

```
== Workflow ==========================================================
Preprocessor: Recipe
Model: nearest_neighbor()

-- Preprocessor ------------------------------------------------------
1 Recipe Step

- step_normalize()

-- Model -------------------------------------------------------------
K-Nearest Neighbor Model Specification (regression)

Main Arguments:
  neighbors = tune()
  weight_func = tune()
  dist_power = tune()

Computational engine: kknn
```

Para los demás modelos podemos utilizar `dplyr` para definir `respuesta` y `atributos`.

```r
model_vars <-
  workflow_variables(outcomes = compressive_strength,
                     predictors = everything())
no_pre_proc <-
  workflow_set(
    preproc = list(simple = model_vars),
    models = list(MARS = mars_spec, CART = cart_spec,
                  CART_bagged = bag_cart_spec,
                  RF = rf_spec, boosting = xgb_spec,
                  Cubist = cubist_spec)
  )
no_pre_proc
```

```
# A workflow set/tibble: 6 × 4
  wflow_id           info             option    result
  <chr>              <list>           <list>    <list>
1 simple_MARS        <tibble [1 × 4]> <opts[0]> <list [0]>
2 simple_CART        <tibble [1 × 4]> <opts[0]> <list [0]>
3 simple_CART_bagged <tibble [1 × 4]> <opts[0]> <list [0]>
4 simple_RF          <tibble [1 × 4]> <opts[0]> <list [0]>
5 simple_boosting    <tibble [1 × 4]> <opts[0]> <list [0]>
6 simple_Cubist      <tibble [1 × 4]> <opts[0]> <list [0]>
```

Agregamos el conjunto de modelos usan términos no lineales e interacciones.

```r
with_features <-
  workflow_set(
    preproc = list(full_quad = poly_recipe),
    models = list(linear_reg = linear_reg_spec, KNN = knn_spec)
  )
```

Finalmente, creamos el conjunto completo de procesamiento

```r
all_workflows <-
  bind_rows(no_pre_proc, normalized, with_features) %>%
  ## Make the workflow ID's a little more simple:
  mutate(wflow_id = gsub("(simple_)|(normalized_)", "", wflow_id))
all_workflows
```

```
# A workflow set/tibble: 9 × 4
  wflow_id             info             option    result
  <chr>                <list>           <list>    <list>
1 MARS                 <tibble [1 × 4]> <opts[0]> <list [0]>
2 CART                 <tibble [1 × 4]> <opts[0]> <list [0]>
3 CART_bagged          <tibble [1 × 4]> <opts[0]> <list [0]>
4 RF                   <tibble [1 × 4]> <opts[0]> <list [0]>
5 boosting             <tibble [1 × 4]> <opts[0]> <list [0]>
6 Cubist               <tibble [1 × 4]> <opts[0]> <list [0]>
7 KNN                  <tibble [1 × 4]> <opts[0]> <list [0]>
8 full_quad_linear_reg <tibble [1 × 4]> <opts[0]> <list [0]>
9 full_quad_KNN        <tibble [1 × 4]> <opts[0]> <list [0]>
```

2.2. Ajuste y evaluación de modelos

Casi todos los modelos tienen parámetros que se tienen que ajustar. Podemos utilizar los métodos de ajuste que ya hemos visto (`tune_grid()`, etc.). Con la función `workflow_map()` se aplica la misma función para **todos** los flujos de entrenamiento.

Usaremos las mismas opciones para cada uno. Es decir, 25 candidatos en cada modelo para validación cruzada, utilizando la misma separación en bloques.

```
grid_ctrl <-
  control_grid(
    save_pred = TRUE,
    parallel_over = "everything",
    save_workflow = TRUE
  )
```

```
all_cores <- parallel::detectCores(logical = TRUE) - 3
library(doParallel)
cl <- makePSOCKcluster(all_cores)
registerDoParallel(cl)
```

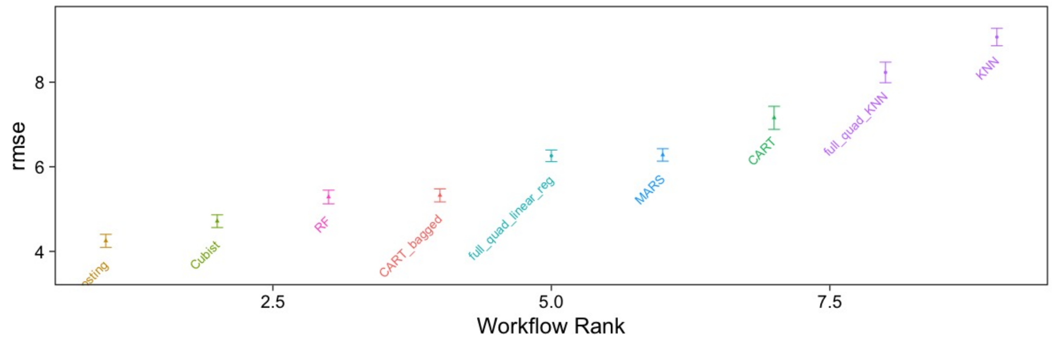
```
system.time(
  grid_results <-
    all_workflows %>%
    workflow_map(
      seed = 1503,
      resamples = concrete_folds,
      grid = 25,
      control = grid_ctrl
    )
)
```

```
i Creating pre-processing data to finalize unknown parameter: mtry
    user  system  elapsed
  26.698   5.506 2083.210
```

```
grid_results %>%
 rank_results() %>%
 filter(.metric == "rmse") %>%
 select(model, .config, rmse = mean, rank)
```

```
# A tibble: 177 × 4
   model        .config                 rmse  rank
   <chr>        <chr>                  <dbl> <int>
 1 boost_tree   Preprocessor1_Model04   4.25     1
 2 boost_tree   Preprocessor1_Model06   4.29     2
 3 boost_tree   Preprocessor1_Model13   4.31     3
 4 boost_tree   Preprocessor1_Model14   4.39     4
 5 boost_tree   Preprocessor1_Model16   4.46     5
 6 boost_tree   Preprocessor1_Model03   4.47     6
 7 boost_tree   Preprocessor1_Model15   4.48     7
 8 boost_tree   Preprocessor1_Model05   4.55     8
 9 boost_tree   Preprocessor1_Model20   4.71     9
```

```
10 cubist_rules Preprocessor1_Model24  4.71    10
# ... with 167 more rows
#  Use ‘print(n = ...)‘ to see more rows
```




```
library(finetune)

race_ctrl <-
  control_race(
    save_pred = TRUE,
    parallel_over = "everything",
    save_workflow = TRUE
  )
```

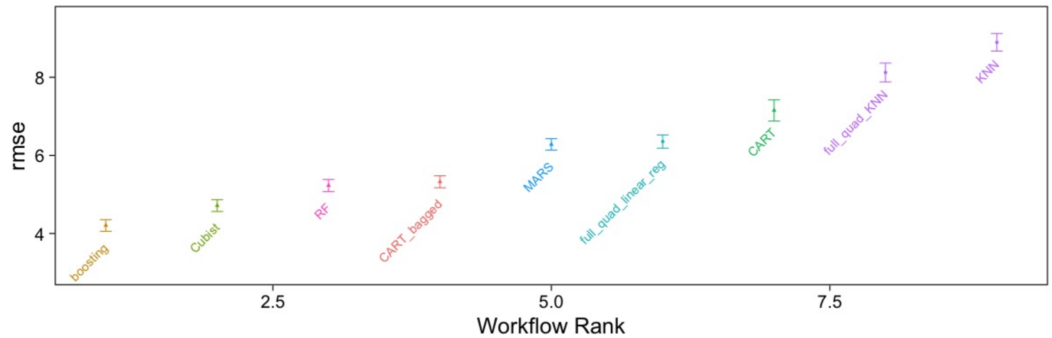
```
system.time(
  race_results <-
    all_workflows %>%
    workflow_map(
      "tune_race_anova",
      seed = 1503,
      resamples = concrete_folds,
      grid = 25,
      control = race_ctrl
    ))
```

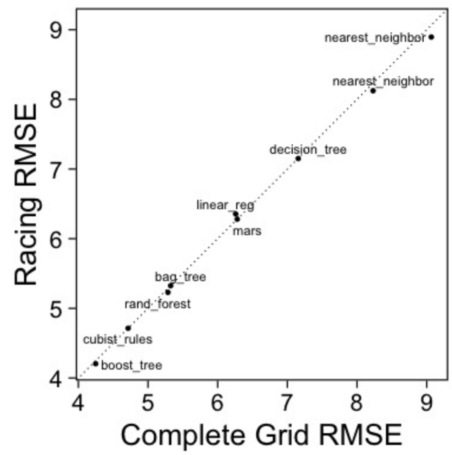
```
i Creating pre-processing data to finalize unknown parameter: mtry
   user  system elapsed
157.602   6.237 678.471
```

```
race_results
```

```
# A workflow set/tibble: 9 × 4
  wflow_id             info             option     result
  <chr>                <list>           <list>     <list>
1 MARS                 <tibble [1 × 4]> <opts[3]> <race[+]>
2 CART                 <tibble [1 × 4]> <opts[3]> <race[+]>
3 CART_bagged          <tibble [1 × 4]> <opts[3]> <rsmp[+]>
4 RF                   <tibble [1 × 4]> <opts[3]> <race[+]>
5 boosting             <tibble [1 × 4]> <opts[3]> <race[+]>
6 Cubist               <tibble [1 × 4]> <opts[3]> <race[+]>
```

```
7 KNN                    <tibble [1 × 4]> <opts[3]> <race[+]>
8 full_quad_linear_reg <tibble [1 × 4]> <opts[3]> <race[+]>
9 full_quad_KNN          <tibble [1 × 4]> <opts[3]> <race[+]>
```





2.3. Finalizar modelo

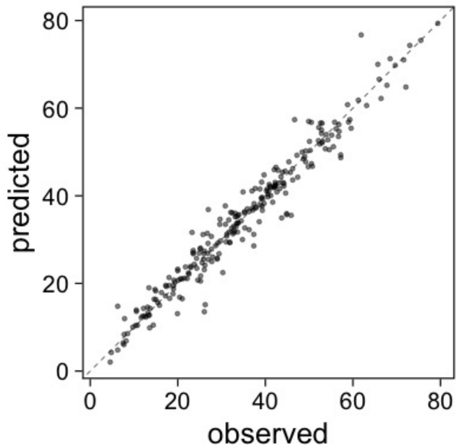
```
best_results <-
  race_results %>%
  extract_workflow_set_result("boosting") %>%
  select_best(metric = "rmse")
best_results
```

```
# A tibble: 1 × 7
  trees min_n tree_depth learn_rate loss_reduction sample_size .config
  <int> <int>      <int>      <dbl>          <dbl>       <dbl> <chr>
1  1957     8          7     0.0756    0.000000145       0.679 Preprocessor1_
   Model04
```

```
boosting_test_results <-
   race_results %>%
   extract_workflow("boosting") %>%
   finalize_workflow(best_results) %>%
   last_fit(split = concrete_split)
```

```
collect_metrics(boosting_test_results)
```

```
# A tibble: 2 × 4
  .metric .estimator .estimate .config
  <chr>   <chr>          <dbl> <chr>
1 rmse    standard       3.43  Preprocessor1_Model1
2 rsq     standard       0.953 Preprocessor1_Model1
```



3. MODELO DE ENSAMBLE

REFERENCIAS

[1] M. Kuhn and K. Johnson. *Applied Predictive Modeling*. Springer New York, New York, NY, 2013. ISBN 978-1-4614-6848-6 978-1-4614-6849-3. . 1

[2] M. Kuhn and J. Silge. *Tidy Modeling with R*. O'Reilly Media, Inc., 2022. 1