# 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")
```

```
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. FLUJO DE PROCESAMIENTO

## 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