Practica 1

Guillermo Palomo y Miguel Díaz-Plaza

2022-12-04

Table of Contents

[Introducción 3](#_Toc120986913)

[1. EDA 4](#_Toc120986914)

[a. skim 4](#_Toc120986915)

[b. str 9](#_Toc120986916)

[c. Missing values (NA’s) 11](#_Toc120986917)

[d. Atributos constantes 13](#_Toc120986918)

[e. Plot de la variable de respuesta a lo largo del tiempo. 13](#_Toc120986919)

[2. RAE 14](#_Toc120986920)

[3. Buscando el mejor modelo 15](#_Toc120986921)

[3.1. Imputación y escalado 15](#_Toc120986922)

[3.2. Evaluación sin ajuste de hiper-parámetros 19](#_Toc120986923)

[3.3. Evaluación con ajuste de hiper-parámetros 21](#_Toc120986924)

[3.4. Métodos de ensembles con y sin ajuste de hiper-parámetros 25](#_Toc120986925)

[Sin ajuste 25](#_Toc120986926)

[Con ajuste 26](#_Toc120986927)

[4. Modelo final 27](#_Toc120986928)

[5. Hyperband 29](#_Toc120986929)

[Conclusión 31](#_Toc120986930)

[Bibliografía 32](#_Toc120986931)

library(mlr3data)  
library(skimr)  
library(corrplot)  
library(janitor)  
library(mlr3)  
library(mlr3learners)  
#remotes::install\_github("mlr-org/mlr3extralearners")  
library(mlr3extralearners)  
library(mlr3pipelines)  
library(kknn)  
library(rpart)  
library(BBmisc)  
library(Cubist)  
library(mlr3verse)  
library(xgboost)  
library(ranger)  
library(randomForest)  
library(gbm)  
library(bbotk)  
library(mlr3hyperband)

# Introducción

El objetivo de esta práctica será predecir la radiación solar diaria en una planta solar de Oklahoma a partir de predicciones de variables meteorológicas del día anterior. Para ello disponemos de una base de datos que explica el comportamiento de la radiación solar a partir de distintas variables meteorológicas, en una central solar, en nuestro caso la **número 9**.

Los atributos de entrada que tenemos son las predicciones del día siguiente de 15 variables meteorológicas. Los datos han sido generados por una simulación de ecuaciones de la atmósfera denominada NWP (Numerical Weather Prediction).

Dichas variables han sido generadas para cinco momentos del día siguiente: 12h, 15h, 18h, 21h, 24h UTC.

Como hay 15 variables, generadas para 5 momentos del día siguiente, en total tenemos 75 atributos de entrada.

La última columna de los datos se denomina “salida” y es la radiación solar acumulada durante todo el día.

Poseemos un conjunto de datos desde 1994-2003. Con ellos, haremos varias pruebas para obtener el mejor modelo posible. Esto incluye todo tipo de pruebas sobre actividades de la metodología: ¿cuál es el mejor método para preprocesar?, ¿cuál es el mejor método de construcción de modelos?, ¿cuáles son los mejores hiper-parámetros para cada método? etc. Obtendremos una estimación del comportamiento de lo que podría obtener el modelo y construiremos el modelo final.

En todo nuestro trabajo utilizaremos la semilla set.seed(100430523).

# 1. EDA

El primer paso que realizaremos será un Análisis Exploratorio de los Datos (EDA).

Cargamos las dos bases de datos que tenemos, en esta primera parte utilizaremos datos\_disp para crear nuestro modelo final y ese modelo final para calcular predicciones sobre el conjuntodatos\_compet.

datos\_disp = readRDS("disp\_9.rds")  
datos\_compet = readRDS("compet\_9.rds")

## a. skim

skim(datos\_disp)

Data summary

|  |  |
| --- | --- |
| Name | datos\_disp |
| Number of rows | 4380 |
| Number of columns | 76 |
| \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |  |
| Column type frequency: |  |
| character | 9 |
| factor | 43 |
| numeric | 24 |
| \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |  |
| Group variables | None |

**Variable type: character**

| skim\_variable | n\_missing | complete\_rate | min | max | empty | n\_unique | whitespace |
| --- | --- | --- | --- | --- | --- | --- | --- |
| dlwrf\_s2\_1 | 613 | 0.86 | 3 | 5 | 0 | 3 | 0 |
| dlwrf\_s3\_1 | 569 | 0.87 | 3 | 5 | 0 | 3 | 0 |
| dlwrf\_s5\_1 | 657 | 0.85 | 3 | 5 | 0 | 3 | 0 |
| tcdc\_ea3\_1 | 0 | 1.00 | 3 | 5 | 0 | 3 | 0 |
| tcolc\_e1\_1 | 0 | 1.00 | 3 | 5 | 0 | 3 | 0 |
| tmin\_2m4\_1 | 0 | 1.00 | 3 | 5 | 0 | 3 | 0 |
| tmp\_2m\_4\_1 | 438 | 0.90 | 3 | 5 | 0 | 3 | 0 |
| ulwrf\_t3\_1 | 0 | 1.00 | 3 | 5 | 0 | 3 | 0 |
| ulwrf\_t4\_1 | 569 | 0.87 | 3 | 5 | 0 | 3 | 0 |

**Variable type: factor**

| skim\_variable | n\_missing | complete\_rate | ordered | n\_unique | top\_counts |
| --- | --- | --- | --- | --- | --- |
| apcp\_sf1\_1 | 0 | 1 | TRUE | 3 | low: 4345, med: 29, hig: 6 |
| apcp\_sf2\_1 | 0 | 1 | TRUE | 3 | low: 4362, med: 14, hig: 4 |
| apcp\_sf3\_1 | 0 | 1 | TRUE | 3 | low: 4364, med: 14, hig: 2 |
| dlwrf\_s1\_1 | 0 | 1 | TRUE | 3 | med: 2017, hig: 1700, low: 663 |
| dlwrf\_s4\_1 | 0 | 1 | TRUE | 3 | med: 1998, hig: 1711, low: 671 |
| dswrf\_s1\_1 | 0 | 1 | TRUE | 3 | low: 4250, med: 97, hig: 33 |
| dswrf\_s2\_1 | 0 | 1 | TRUE | 3 | low: 2038, med: 1209, hig: 1133 |
| dswrf\_s4\_1 | 0 | 1 | TRUE | 3 | hig: 2330, med: 1581, low: 469 |
| pres\_ms1\_1 | 0 | 1 | TRUE | 3 | med: 3382, low: 578, hig: 420 |
| pres\_ms2\_1 | 0 | 1 | TRUE | 3 | med: 3468, hig: 488, low: 424 |
| pres\_ms3\_1 | 0 | 1 | TRUE | 3 | med: 3569, hig: 545, low: 266 |
| pres\_ms4\_1 | 0 | 1 | TRUE | 3 | med: 3625, hig: 499, low: 256 |
| pwat\_ea2\_1 | 0 | 1 | TRUE | 3 | low: 2466, med: 1567, hig: 347 |
| pwat\_ea4\_1 | 0 | 1 | TRUE | 3 | low: 2358, med: 1598, hig: 424 |
| pwat\_ea5\_1 | 0 | 1 | TRUE | 3 | low: 2362, med: 1576, hig: 442 |
| spfh\_2m1\_1 | 0 | 1 | TRUE | 3 | low: 2297, med: 1520, hig: 563 |
| spfh\_2m3\_1 | 0 | 1 | TRUE | 3 | low: 2099, med: 1430, hig: 851 |
| spfh\_2m5\_1 | 0 | 1 | TRUE | 3 | low: 2063, med: 1436, hig: 881 |
| tcdc\_ea5\_1 | 0 | 1 | TRUE | 3 | low: 4312, med: 63, hig: 5 |
| tcolc\_e2\_1 | 0 | 1 | TRUE | 3 | low: 4302, med: 67, hig: 11 |
| tcolc\_e3\_1 | 0 | 1 | TRUE | 3 | low: 4328, med: 45, hig: 7 |
| tmax\_2m1\_1 | 0 | 1 | TRUE | 3 | hig: 2268, med: 1964, low: 148 |
| tmax\_2m2\_1 | 0 | 1 | TRUE | 3 | hig: 2244, med: 1897, low: 239 |
| tmax\_2m4\_1 | 0 | 1 | TRUE | 3 | med: 2144, hig: 1869, low: 367 |
| tmax\_2m5\_1 | 0 | 1 | TRUE | 3 | med: 2135, hig: 1877, low: 368 |
| tmin\_2m2\_1 | 0 | 1 | TRUE | 3 | hig: 2289, med: 1940, low: 151 |
| tmin\_2m3\_1 | 0 | 1 | TRUE | 3 | hig: 2285, med: 1943, low: 152 |
| tmin\_2m5\_1 | 0 | 1 | TRUE | 3 | med: 2126, hig: 1978, low: 276 |
| tmp\_2m\_1\_1 | 0 | 1 | TRUE | 3 | hig: 2288, med: 1948, low: 144 |
| tmp\_2m\_2\_1 | 0 | 1 | TRUE | 3 | hig: 2237, med: 1899, low: 244 |
| tmp\_2m\_3\_1 | 0 | 1 | TRUE | 3 | med: 2054, hig: 2029, low: 297 |
| tmp\_sfc2\_1 | 0 | 1 | TRUE | 3 | hig: 2125, med: 1986, low: 269 |
| tmp\_sfc3\_1 | 0 | 1 | TRUE | 3 | med: 2206, hig: 1812, low: 362 |
| tmp\_sfc4\_1 | 0 | 1 | TRUE | 3 | med: 2254, hig: 1688, low: 438 |
| tmp\_sfc5\_1 | 0 | 1 | TRUE | 3 | med: 2092, hig: 2050, low: 238 |
| ulwrf\_s1\_1 | 0 | 1 | TRUE | 3 | med: 2267, hig: 1826, low: 287 |
| ulwrf\_s3\_1 | 0 | 1 | TRUE | 3 | med: 2108, hig: 1761, low: 511 |
| ulwrf\_s5\_1 | 0 | 1 | TRUE | 3 | med: 2293, hig: 1338, low: 749 |
| ulwrf\_t1\_1 | 0 | 1 | TRUE | 3 | hig: 2435, med: 1711, low: 234 |
| ulwrf\_t2\_1 | 0 | 1 | TRUE | 3 | hig: 2317, med: 1774, low: 289 |
| ulwrf\_t5\_1 | 0 | 1 | TRUE | 3 | hig: 2449, med: 1614, low: 317 |
| uswrf\_s4\_1 | 0 | 1 | TRUE | 3 | low: 2719, med: 1624, hig: 37 |
| uswrf\_s5\_1 | 0 | 1 | TRUE | 3 | low: 2398, med: 1962, hig: 20 |

**Variable type: numeric**

| skim\_variable | n\_missing | complete\_rate | mean | sd | p0 | p25 | p50 | p75 | p100 | hist |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| apcp\_sf4\_1 | 876 | 0.80 | 0.26 | 0.99 | 0.00 | 0.00 | 0.00 | 0.02 | 12.21 | ▇▁▁▁▁ |
| apcp\_sf5\_1 | 745 | 0.83 | 0.54 | 1.99 | 0.00 | 0.00 | 0.00 | 0.09 | 28.97 | ▇▁▁▁▁ |
| dswrf\_s3\_1 | 657 | 0.85 | 369.14 | 162.40 | 14.55 | 224.55 | 373.64 | 518.00 | 641.91 | ▂▇▆▇▇ |
| dswrf\_s5\_1 | 0 | 1.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | ▁▁▇▁▁ |
| pres\_ms5\_1 | 745 | 0.83 | 101499.43 | 770.83 | 98276.55 | 101014.35 | 101432.93 | 101961.07 | 104460.88 | ▁▂▇▃▁ |
| pwat\_ea1\_1 | 0 | 1.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | ▁▁▇▁▁ |
| pwat\_ea3\_1 | 0 | 1.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | ▁▁▇▁▁ |
| spfh\_2m2\_1 | 701 | 0.84 | 0.01 | 0.00 | 0.00 | 0.00 | 0.01 | 0.01 | 0.02 | ▇▇▅▆▂ |
| spfh\_2m4\_1 | 0 | 1.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | ▁▁▇▁▁ |
| tcdc\_ea1\_1 | 0 | 1.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | ▁▁▇▁▁ |
| tcdc\_ea2\_1 | 701 | 0.84 | 0.07 | 0.16 | 0.00 | 0.00 | 0.01 | 0.06 | 1.79 | ▇▁▁▁▁ |
| tcdc\_ea4\_1 | 3942 | 0.10 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | ▁▁▇▁▁ |
| tcolc\_e4\_1 | 4030 | 0.08 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | ▁▁▇▁▁ |
| tcolc\_e5\_1 | 0 | 1.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | ▁▁▇▁▁ |
| tmax\_2m3\_1 | 482 | 0.89 | 291.41 | 10.64 | 259.27 | 283.05 | 292.28 | 300.57 | 311.89 | ▁▃▇▇▆ |
| tmin\_2m1\_1 | 613 | 0.86 | 282.99 | 9.46 | 247.84 | 275.80 | 283.53 | 291.27 | 300.74 | ▁▂▇▇▇ |
| tmp\_2m\_5\_1 | 788 | 0.82 | 289.80 | 10.92 | 252.90 | 281.43 | 290.45 | 299.29 | 311.76 | ▁▃▇▇▆ |
| tmp\_sfc1\_1 | 0 | 1.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | ▁▁▇▁▁ |
| ulwrf\_s2\_1 | 613 | 0.86 | 374.61 | 53.47 | 215.08 | 330.18 | 375.47 | 423.60 | 474.54 | ▁▅▇▇▇ |
| ulwrf\_s4\_1 | 832 | 0.81 | 432.34 | 58.66 | 269.12 | 386.79 | 433.85 | 482.39 | 567.73 | ▁▆▇▇▃ |
| uswrf\_s1\_1 | 482 | 0.89 | 0.12 | 0.32 | 0.00 | 0.00 | 0.00 | 0.00 | 1.00 | ▇▁▁▁▁ |
| uswrf\_s2\_1 | 526 | 0.88 | 38.52 | 26.84 | 0.00 | 13.02 | 34.68 | 62.00 | 93.09 | ▇▅▅▃▃ |
| uswrf\_s3\_1 | 569 | 0.87 | 76.78 | 32.00 | 2.09 | 51.91 | 79.73 | 103.09 | 249.55 | ▅▇▅▁▁ |
| salida | 0 | 1.00 | 16057114.92 | 7815054.49 | 36000.00 | 10432275.00 | 15844650.00 | 22977750.00 | 30771900.00 | ▅▇▇▇▆ |

Observamos un total de  **instancias** y  **atributos ( más la salida)**.

## b. str

str(datos\_disp)

## 'data.frame': 4380 obs. of 76 variables:  
## $ apcp\_sf1\_1: Ord.factor w/ 3 levels "low"<"medium"<..: 1 1 1 1 1 1 1 1 1 1 ...  
## $ apcp\_sf2\_1: Ord.factor w/ 3 levels "low"<"medium"<..: 1 1 1 1 1 1 1 1 1 1 ...  
## $ apcp\_sf3\_1: Ord.factor w/ 3 levels "low"<"medium"<..: 1 1 1 1 1 1 1 1 1 1 ...  
## $ apcp\_sf4\_1: num 0 0.394 0 0 0 ...  
## $ apcp\_sf5\_1: num 0 0.812 NA 0 0 ...  
## $ dlwrf\_s1\_1: Ord.factor w/ 3 levels "low"<"medium"<..: 2 2 1 1 1 2 1 1 1 2 ...  
## $ dlwrf\_s2\_1: chr "red" "blue" "red" "blue" ...  
## $ dlwrf\_s3\_1: chr "red" "blue" "red" NA ...  
## $ dlwrf\_s4\_1: Ord.factor w/ 3 levels "low"<"medium"<..: 1 2 1 1 2 1 1 1 1 2 ...  
## $ dlwrf\_s5\_1: chr "red" NA "red" "red" ...  
## $ dswrf\_s1\_1: Ord.factor w/ 3 levels "low"<"medium"<..: 1 1 1 1 1 1 1 1 1 1 ...  
## $ dswrf\_s2\_1: Ord.factor w/ 3 levels "low"<"medium"<..: 1 1 1 1 1 1 1 1 1 1 ...  
## $ dswrf\_s3\_1: num 191 151 187 172 NA ...  
## $ dswrf\_s4\_1: Ord.factor w/ 3 levels "low"<"medium"<..: 2 1 2 2 2 2 2 2 2 1 ...  
## $ dswrf\_s5\_1: num 0 0 0 0 0 0 0 0 0 0 ...  
## $ pres\_ms1\_1: Ord.factor w/ 3 levels "low"<"medium"<..: 2 2 2 2 2 1 3 3 2 2 ...  
## $ pres\_ms2\_1: Ord.factor w/ 3 levels "low"<"medium"<..: 2 2 2 2 2 1 3 3 2 2 ...  
## $ pres\_ms3\_1: Ord.factor w/ 3 levels "low"<"medium"<..: 2 2 2 3 2 2 3 3 2 2 ...  
## $ pres\_ms4\_1: Ord.factor w/ 3 levels "low"<"medium"<..: 2 2 2 3 1 2 3 3 2 2 ...  
## $ pres\_ms5\_1: num 102054 101438 101425 102470 100030 ...  
## $ pwat\_ea1\_1: num 0 0 0 0 0 0 0 0 0 0 ...  
## $ pwat\_ea2\_1: Ord.factor w/ 3 levels "low"<"medium"<..: 1 1 1 1 1 1 1 1 1 1 ...  
## $ pwat\_ea3\_1: num 0 0 0 0 0 0 0 0 0 0 ...  
## $ pwat\_ea4\_1: Ord.factor w/ 3 levels "low"<"medium"<..: 1 1 1 1 1 1 1 1 1 1 ...  
## $ pwat\_ea5\_1: Ord.factor w/ 3 levels "low"<"medium"<..: 1 1 1 1 1 1 1 1 1 1 ...  
## $ spfh\_2m1\_1: Ord.factor w/ 3 levels "low"<"medium"<..: 1 1 1 1 1 1 1 1 1 1 ...  
## $ spfh\_2m2\_1: num 0.00357 0.00381 0.00178 0.00253 0.00304 ...  
## $ spfh\_2m3\_1: Ord.factor w/ 3 levels "low"<"medium"<..: 1 1 1 1 1 1 1 1 1 1 ...  
## $ spfh\_2m4\_1: num 0 0 0 0 0 0 0 0 0 0 ...  
## $ spfh\_2m5\_1: Ord.factor w/ 3 levels "low"<"medium"<..: 1 1 1 1 1 1 1 1 1 1 ...  
## $ tcdc\_ea1\_1: num 0 0 0 0 0 0 0 0 0 0 ...  
## $ tcdc\_ea2\_1: num NA 0.1373 NA 0.0191 NA ...  
## $ tcdc\_ea3\_1: chr "red" "red" "red" "red" ...  
## $ tcdc\_ea4\_1: num NA NA NA NA 0 NA NA NA NA NA ...  
## $ tcdc\_ea5\_1: Ord.factor w/ 3 levels "low"<"medium"<..: 1 1 1 1 1 1 1 1 1 1 ...  
## $ tcolc\_e1\_1: chr "red" "red" "red" "red" ...  
## $ tcolc\_e2\_1: Ord.factor w/ 3 levels "low"<"medium"<..: 1 1 1 1 1 1 1 1 1 1 ...  
## $ tcolc\_e3\_1: Ord.factor w/ 3 levels "low"<"medium"<..: 1 1 1 1 1 1 1 1 1 1 ...  
## $ tcolc\_e4\_1: num NA NA NA NA NA NA NA NA NA NA ...  
## $ tcolc\_e5\_1: num 0 0 0 0 0 0 0 0 0 0 ...  
## $ tmax\_2m1\_1: Ord.factor w/ 3 levels "low"<"medium"<..: 2 2 2 2 2 2 1 1 2 2 ...  
## $ tmax\_2m2\_1: Ord.factor w/ 3 levels "low"<"medium"<..: 2 2 1 2 2 2 1 1 2 2 ...  
## $ tmax\_2m3\_1: num 281 283 273 275 281 ...  
## $ tmax\_2m4\_1: Ord.factor w/ 3 levels "low"<"medium"<..: 2 2 1 1 2 1 1 1 2 2 ...  
## $ tmax\_2m5\_1: Ord.factor w/ 3 levels "low"<"medium"<..: 2 2 1 1 2 1 1 1 2 2 ...  
## $ tmin\_2m1\_1: num 277 275 267 NA 271 ...  
## $ tmin\_2m2\_1: Ord.factor w/ 3 levels "low"<"medium"<..: 2 2 2 2 2 2 1 1 2 2 ...  
## $ tmin\_2m3\_1: Ord.factor w/ 3 levels "low"<"medium"<..: 2 2 2 2 2 2 1 1 2 2 ...  
## $ tmin\_2m4\_1: chr "blue" "blue" "red" "blue" ...  
## $ tmin\_2m5\_1: Ord.factor w/ 3 levels "low"<"medium"<..: 2 2 2 1 2 1 1 1 2 2 ...  
## $ tmp\_2m\_1\_1: Ord.factor w/ 3 levels "low"<"medium"<..: 2 2 2 2 2 2 1 1 2 2 ...  
## $ tmp\_2m\_2\_1: Ord.factor w/ 3 levels "low"<"medium"<..: 2 2 1 2 2 2 1 1 2 2 ...  
## $ tmp\_2m\_3\_1: Ord.factor w/ 3 levels "low"<"medium"<..: 2 2 1 2 2 2 1 1 2 2 ...  
## $ tmp\_2m\_4\_1: chr "blue" "blue" "red" "red" ...  
## $ tmp\_2m\_5\_1: num 277 280 274 270 281 ...  
## $ tmp\_sfc1\_1: num 0 0 0 0 0 0 0 0 0 0 ...  
## $ tmp\_sfc2\_1: Ord.factor w/ 3 levels "low"<"medium"<..: 2 2 1 2 2 2 1 1 1 2 ...  
## $ tmp\_sfc3\_1: Ord.factor w/ 3 levels "low"<"medium"<..: 2 2 2 2 2 1 1 2 1 2 ...  
## $ tmp\_sfc4\_1: Ord.factor w/ 3 levels "low"<"medium"<..: 2 2 1 1 2 1 1 1 1 2 ...  
## $ tmp\_sfc5\_1: Ord.factor w/ 3 levels "low"<"medium"<..: 2 2 2 1 2 1 1 1 2 2 ...  
## $ ulwrf\_s1\_1: Ord.factor w/ 3 levels "low"<"medium"<..: 2 2 1 2 2 2 1 1 1 2 ...  
## $ ulwrf\_s2\_1: num 328 329 289 305 NA ...  
## $ ulwrf\_s3\_1: Ord.factor w/ 3 levels "low"<"medium"<..: 2 2 1 1 1 2 1 1 1 2 ...  
## $ ulwrf\_s4\_1: num 377 380 355 352 375 ...  
## $ ulwrf\_s5\_1: Ord.factor w/ 3 levels "low"<"medium"<..: 2 2 1 1 2 1 1 1 1 2 ...  
## $ ulwrf\_t1\_1: Ord.factor w/ 3 levels "low"<"medium"<..: 2 2 2 2 2 3 2 2 2 2 ...  
## $ ulwrf\_t2\_1: Ord.factor w/ 3 levels "low"<"medium"<..: 2 2 2 2 2 2 2 2 2 2 ...  
## $ ulwrf\_t3\_1: chr "blue" "blue" "blue" "blue" ...  
## $ ulwrf\_t4\_1: chr "blue" "blue" "blue" "blue" ...  
## $ ulwrf\_t5\_1: Ord.factor w/ 3 levels "low"<"medium"<..: 2 2 2 2 2 2 2 2 2 2 ...  
## $ uswrf\_s1\_1: num 0 0 0 NA 0 0 0 0 0 0 ...  
## $ uswrf\_s2\_1: num 7 2.18 8.18 5.27 6.45 ...  
## $ uswrf\_s3\_1: num 45 29.2 45.6 37 42.3 ...  
## $ uswrf\_s4\_1: Ord.factor w/ 3 levels "low"<"medium"<..: 1 1 1 1 1 1 1 1 1 1 ...  
## $ uswrf\_s5\_1: Ord.factor w/ 3 levels "low"<"medium"<..: 1 1 1 1 1 1 1 1 1 1 ...  
## $ salida : num 10848300 3318300 5266500 8270100 8009400 ...

La mayoría () son variables categóricas ordinales con niveles, es decir, factores “low”, “medium” y “high”. También hay algunas numéricas () y otras categóricas () que str detecta como characters que son las que tienen colores (red, blue y green) como observaciones, y debería detectarlas como factores, paso que realizaremos más adelante en la imputación .

## c. Missing values (NA’s)

Aparecen valores faltantes para todo tipo de variables en la base de datos. Un elevado número de NA’s nos puede dar indicios de eliminar dicha variable, ya que habría que predecir una gran proporción de datos de dicha variable en caso de querer utilizarla.

noms <- colnames(datos\_disp)  
Nas <- round(1 - skim(datos\_disp)$complete\_rate,3)  
df\_NA <- data.frame(noms,Nas)  
df\_NA

## noms Nas  
## 1 apcp\_sf1\_1 0.14  
## 2 apcp\_sf2\_1 0.13  
## 3 apcp\_sf3\_1 0.15  
## 4 apcp\_sf4\_1 0.00  
## 5 apcp\_sf5\_1 0.00  
## 6 dlwrf\_s1\_1 0.00  
## 7 dlwrf\_s2\_1 0.10  
## 8 dlwrf\_s3\_1 0.00  
## 9 dlwrf\_s4\_1 0.13  
## 10 dlwrf\_s5\_1 0.00  
## 11 dswrf\_s1\_1 0.00  
## 12 dswrf\_s2\_1 0.00  
## 13 dswrf\_s3\_1 0.00  
## 14 dswrf\_s4\_1 0.00  
## 15 dswrf\_s5\_1 0.00  
## 16 pres\_ms1\_1 0.00  
## 17 pres\_ms2\_1 0.00  
## 18 pres\_ms3\_1 0.00  
## 19 pres\_ms4\_1 0.00  
## 20 pres\_ms5\_1 0.00  
## 21 pwat\_ea1\_1 0.00  
## 22 pwat\_ea2\_1 0.00  
## 23 pwat\_ea3\_1 0.00  
## 24 pwat\_ea4\_1 0.00  
## 25 pwat\_ea5\_1 0.00  
## 26 spfh\_2m1\_1 0.00  
## 27 spfh\_2m2\_1 0.00  
## 28 spfh\_2m3\_1 0.00  
## 29 spfh\_2m4\_1 0.00  
## 30 spfh\_2m5\_1 0.00  
## 31 tcdc\_ea1\_1 0.00  
## 32 tcdc\_ea2\_1 0.00  
## 33 tcdc\_ea3\_1 0.00  
## 34 tcdc\_ea4\_1 0.00  
## 35 tcdc\_ea5\_1 0.00  
## 36 tcolc\_e1\_1 0.00  
## 37 tcolc\_e2\_1 0.00  
## 38 tcolc\_e3\_1 0.00  
## 39 tcolc\_e4\_1 0.00  
## 40 tcolc\_e5\_1 0.00  
## 41 tmax\_2m1\_1 0.00  
## 42 tmax\_2m2\_1 0.00  
## 43 tmax\_2m3\_1 0.00  
## 44 tmax\_2m4\_1 0.00  
## 45 tmax\_2m5\_1 0.00  
## 46 tmin\_2m1\_1 0.00  
## 47 tmin\_2m2\_1 0.00  
## 48 tmin\_2m3\_1 0.00  
## 49 tmin\_2m4\_1 0.00  
## 50 tmin\_2m5\_1 0.00  
## 51 tmp\_2m\_1\_1 0.00  
## 52 tmp\_2m\_2\_1 0.00  
## 53 tmp\_2m\_3\_1 0.20  
## 54 tmp\_2m\_4\_1 0.17  
## 55 tmp\_2m\_5\_1 0.15  
## 56 tmp\_sfc1\_1 0.00  
## 57 tmp\_sfc2\_1 0.17  
## 58 tmp\_sfc3\_1 0.00  
## 59 tmp\_sfc4\_1 0.00  
## 60 tmp\_sfc5\_1 0.16  
## 61 ulwrf\_s1\_1 0.00  
## 62 ulwrf\_s2\_1 0.00  
## 63 ulwrf\_s3\_1 0.16  
## 64 ulwrf\_s4\_1 0.90  
## 65 ulwrf\_s5\_1 0.92  
## 66 ulwrf\_t1\_1 0.00  
## 67 ulwrf\_t2\_1 0.11  
## 68 ulwrf\_t3\_1 0.14  
## 69 ulwrf\_t4\_1 0.18  
## 70 ulwrf\_t5\_1 0.00  
## 71 uswrf\_s1\_1 0.14  
## 72 uswrf\_s2\_1 0.19  
## 73 uswrf\_s3\_1 0.11  
## 74 uswrf\_s4\_1 0.12  
## 75 uswrf\_s5\_1 0.13  
## 76 salida 0.00

La tabla anterior muestra la proporción de NAs. Como vemos la mayoría de atributos están completos pero llaman la atención **“ulwrf\_s4\_1”** y **“ulwrf\_s5\_1”** con **más de un 90% de NA’s**

## d. Atributos constantes

En ocasiones nos encontramos con atributos que solo toman un valor para toda la población. Esto hace que sean irrelevantes por lo que es conveniente eliminarlos.

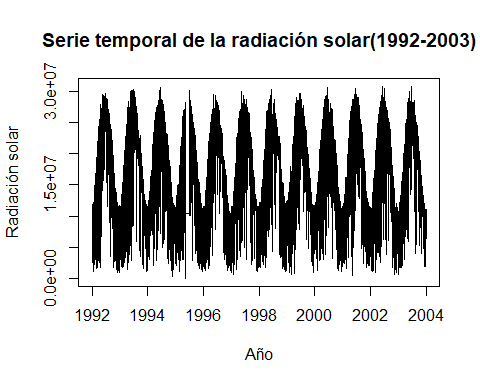
Como vemos en el skim del apartado a, hay algunas variables constantes que son todo ceros, por lo que las eliminaremos después, en el preproceso, junto con las que tengan más del 80% de NAs.

También el skim nos ofrece histogramas que nos pueden ser útiles para localizar estos atributos constantes.

Los eliminaremos más adelante con la función remove\_constant() de la librería janitor.

## e. Plot de la variable de respuesta a lo largo del tiempo.

serie <- ts(datos\_disp$salida, freq=365, start = c(1992,1))  
plot(serie, main="Serie temporal de la radiación solar(1992-2003)",  
xlab="Año", ylab="Radiación solar")



Vemos que la radiación solar oscila estacionalmente. En los meses de verano encontramos los valores más altos, porque hay radiación más alta, y en invierno los más bajos, porque en los meses de invierno hay radiación más baja.

Observamos además 12 distintos ciclos, que corresponden al número de años en los que se recogen los datos de la base de datos.

# 2. RAE

La métrica del “Relative absolute error” (RAE) se basa en comparar el rendimiento de un modelo trivial, por ejemplo, la media de la salida, con el modelo que estemos utilizando.

De otra forma, el RAE es una medida de evaluación de modelos predictivos y es expresada como las predicciones de nuestro modelo menos las observaciones, entre la media de los datos menos las observaciones.

Su fórmula es:

Un modelo razonable tendrá un **RAE** menor que y serán mejores aquellos modelos cuyo **RAE** esté más próximo a .

En MLR3, se simplifica bastante su cálculo y se obtiene utilizando:

medida <- msr("regr.rae")

# 3. Buscando el mejor modelo

## 3.1. Imputación y escalado

A continuación, vamos a comparar diferentes métodos de imputación de NA’s y escalado de los datos, y vamos a determinar cuál es la mejor combinación.

Existen un total de  **combinaciones posibles**, ya que vamos a tener en cuenta:

* 3 tipos de imputación para variables categóricas: con la moda (imputemode), con un valor al azar (imputesample) y con el learner de clasificación de rpart (classif\_learner\_rpart)
* 3 tipos de imputación para variables numéricas o cuantitativas: con la media (imputemean), con la mediana (imputemedian) y con el learner de regresión de rpart (imputehist. regr\_learner\_rpart)
* 3 tipos de escalado: scale, scalerange y scalemaxabs.

Escogeremos un tipo de imputación para variables categóricas, otro tipo de imputación para variables cuantitativas y un tipo de escalado. Elegiremos aquella combinación de las tres con menor RAE.

Con ayuda del código de Carlos Morales y unos ligeros cambios que lo hacen más sencillo y automático encontramos la mejor forma de imputar y escalar.

En el preproceso sin info leakage, lo que hemos hecho es suprimir las columnas constantes, y aquellas con un porcentaje de NA’s superior al .

Por su parte, en el preproceso con info leakage, ya si es cuando tratamos de convertir predictores, y elegimos las formas de imputado y escalado óptimas para nuestros datos.

# Preproceso no info leakage  
  
datos\_disp\_RC <- remove\_constant(datos\_disp, na.rm = T, quiet = F)

## Removing 9 constant columns of 76 columns total (Removed: dswrf\_s5\_1, pwat\_ea1\_1, pwat\_ea3\_1, spfh\_2m4\_1, tcdc\_ea1\_1, tcdc\_ea4\_1, tcolc\_e4\_1, tcolc\_e5\_1, tmp\_sfc1\_1).

datos\_disp\_RC\_RNA <- datos\_disp\_RC[,!(names(datos\_disp\_RC) %in% noms[Nas >= 0.8])]  
  
df <- datos\_disp\_RC\_RNA  
  
charcol <- c(colnames(df[, sapply(df, class) == 'character']))  
  
for (i in charcol){  
 df[[i]] <- as.factor(df[[i]])  
}  
  
c(colnames(df[, sapply(df, class) == 'character'])) #Ya no hay columnas que sea 'character', solo factores

## character(0)

source("extras\_from\_mlr.R")  
df <- createDummyFeatures(df, target = "salida")  
  
  
# Preproceso info leakage  
  
imputcat <- c( "imputemode", "imputesample", "imputelearner")  
imputnum <- c("imputemean", "imputemedian", "imputehist", "imputelearner")  
esc <- c("scale", "scalerange", "scalemaxabs")  
  
eshiper <- expand.grid(imputcat,imputnum, esc)  
  
df\_task <- as\_task\_regr(df, target = "salida")  
  
  
knn\_lrn <- lrn("regr.kknn")  
  
res\_desc <- rsmp("custom")  
res\_desc$instantiate(df\_task,  
 train = list(1:(6\*365)), # 6 a?os  
 test = list((6\*365+1):(9\*365))) # 3 a?os  
  
medida <- msr("regr.rae")  
  
totalerror <- c()  
set.seed(100430523)  
  
for (i in 1:dim(eshiper)[1]) {  
 if (as.character(eshiper[i,1]) == "imputelearner" & as.character(eshiper[i,2]) != "imputelearner"){  
 graph =  
 po(as.character(eshiper[i,1]), lrn("classif.rpart")) %>>%  
 po(as.character(eshiper[i,2])) %>>%  
 po(as.character(eshiper[i,3])) %>>%  
 po(knn\_lrn)  
 }  
 else if (as.character(eshiper[i,2]) == "imputelearner" & as.character(eshiper[i,1]) != "imputelearner"){  
 graph =  
 po(as.character(eshiper[i,1])) %>>%  
 po(as.character(eshiper[i,2]), lrn("regr.rpart")) %>>%  
 po(as.character(eshiper[i,3])) %>>%  
 po(knn\_lrn)  
 }  
 else if (as.character(eshiper[i,2]) == "imputelearner" & as.character(eshiper[i,1]) == "imputelearner"){  
 graph =  
 po(as.character(eshiper[i,1]), lrn("classif.rpart"), id = "impute\_clrpart") %>>%  
 po(as.character(eshiper[i,2]), lrn("regr.rpart"), id = "impute\_regrpart") %>>%  
 po(as.character(eshiper[i,3])) %>>%  
 po(knn\_lrn)  
 }  
 else{  
 graph =  
 po(as.character(eshiper[i,1])) %>>%  
 po(as.character(eshiper[i,2])) %>>%  
 po(as.character(eshiper[i,3])) %>>%  
 po(knn\_lrn)  
 }  
 graph\_lrn <- as\_learner(graph)  
 knn\_resample <- resample(task = df\_task,  
 learner = graph\_lrn,  
 resampling = res\_desc)  
 knn\_rae <- knn\_resample$aggregate(medida)  
 totalerror <- append(totalerror, knn\_rae)  
}

## INFO [18:57:23.663] [mlr3] Applying learner 'imputemode.imputemean.scale.regr.kknn' on task 'df' (iter 1/1)  
## INFO [18:57:28.949] [mlr3] Applying learner 'imputesample.imputemean.scale.regr.kknn' on task 'df' (iter 1/1)  
## INFO [18:57:32.893] [mlr3] Applying learner 'imputelearner.imputemean.scale.regr.kknn' on task 'df' (iter 1/1)  
## INFO [18:57:34.777] [mlr3] Applying learner 'imputemode.imputemedian.scale.regr.kknn' on task 'df' (iter 1/1)  
## INFO [18:57:38.593] [mlr3] Applying learner 'imputesample.imputemedian.scale.regr.kknn' on task 'df' (iter 1/1)  
## INFO [18:57:42.541] [mlr3] Applying learner 'imputelearner.imputemedian.scale.regr.kknn' on task 'df' (iter 1/1)  
## INFO [18:57:44.774] [mlr3] Applying learner 'imputemode.imputehist.scale.regr.kknn' on task 'df' (iter 1/1)  
## INFO [18:57:48.750] [mlr3] Applying learner 'imputesample.imputehist.scale.regr.kknn' on task 'df' (iter 1/1)  
## INFO [18:57:52.244] [mlr3] Applying learner 'imputelearner.imputehist.scale.regr.kknn' on task 'df' (iter 1/1)  
## INFO [18:57:55.239] [mlr3] Applying learner 'imputemode.imputelearner.scale.regr.kknn' on task 'df' (iter 1/1)  
## INFO [18:58:19.144] [mlr3] Applying learner 'imputesample.imputelearner.scale.regr.kknn' on task 'df' (iter 1/1)  
## INFO [18:58:41.641] [mlr3] Applying learner 'impute\_clrpart.impute\_regrpart.scale.regr.kknn' on task 'df' (iter 1/1)  
## INFO [18:59:02.880] [mlr3] Applying learner 'imputemode.imputemean.scalerange.regr.kknn' on task 'df' (iter 1/1)  
## INFO [18:59:05.917] [mlr3] Applying learner 'imputesample.imputemean.scalerange.regr.kknn' on task 'df' (iter 1/1)  
## INFO [18:59:09.484] [mlr3] Applying learner 'imputelearner.imputemean.scalerange.regr.kknn' on task 'df' (iter 1/1)  
## INFO [18:59:11.174] [mlr3] Applying learner 'imputemode.imputemedian.scalerange.regr.kknn' on task 'df' (iter 1/1)  
## INFO [18:59:14.107] [mlr3] Applying learner 'imputesample.imputemedian.scalerange.regr.kknn' on task 'df' (iter 1/1)  
## INFO [18:59:17.220] [mlr3] Applying learner 'imputelearner.imputemedian.scalerange.regr.kknn' on task 'df' (iter 1/1)  
## INFO [18:59:18.920] [mlr3] Applying learner 'imputemode.imputehist.scalerange.regr.kknn' on task 'df' (iter 1/1)  
## INFO [18:59:21.867] [mlr3] Applying learner 'imputesample.imputehist.scalerange.regr.kknn' on task 'df' (iter 1/1)  
## INFO [18:59:25.891] [mlr3] Applying learner 'imputelearner.imputehist.scalerange.regr.kknn' on task 'df' (iter 1/1)  
## INFO [18:59:28.807] [mlr3] Applying learner 'imputemode.imputelearner.scalerange.regr.kknn' on task 'df' (iter 1/1)  
## INFO [18:59:51.212] [mlr3] Applying learner 'imputesample.imputelearner.scalerange.regr.kknn' on task 'df' (iter 1/1)  
## INFO [19:00:13.021] [mlr3] Applying learner 'impute\_clrpart.impute\_regrpart.scalerange.regr.kknn' on task 'df' (iter 1/1)  
## INFO [19:00:32.849] [mlr3] Applying learner 'imputemode.imputemean.scalemaxabs.regr.kknn' on task 'df' (iter 1/1)  
## INFO [19:00:35.673] [mlr3] Applying learner 'imputesample.imputemean.scalemaxabs.regr.kknn' on task 'df' (iter 1/1)  
## INFO [19:00:39.293] [mlr3] Applying learner 'imputelearner.imputemean.scalemaxabs.regr.kknn' on task 'df' (iter 1/1)  
## INFO [19:00:41.259] [mlr3] Applying learner 'imputemode.imputemedian.scalemaxabs.regr.kknn' on task 'df' (iter 1/1)  
## INFO [19:00:44.634] [mlr3] Applying learner 'imputesample.imputemedian.scalemaxabs.regr.kknn' on task 'df' (iter 1/1)  
## INFO [19:00:48.211] [mlr3] Applying learner 'imputelearner.imputemedian.scalemaxabs.regr.kknn' on task 'df' (iter 1/1)  
## INFO [19:00:50.318] [mlr3] Applying learner 'imputemode.imputehist.scalemaxabs.regr.kknn' on task 'df' (iter 1/1)  
## INFO [19:00:54.362] [mlr3] Applying learner 'imputesample.imputehist.scalemaxabs.regr.kknn' on task 'df' (iter 1/1)  
## INFO [19:00:58.660] [mlr3] Applying learner 'imputelearner.imputehist.scalemaxabs.regr.kknn' on task 'df' (iter 1/1)  
## INFO [19:01:00.751] [mlr3] Applying learner 'imputemode.imputelearner.scalemaxabs.regr.kknn' on task 'df' (iter 1/1)  
## INFO [19:01:22.128] [mlr3] Applying learner 'imputesample.imputelearner.scalemaxabs.regr.kknn' on task 'df' (iter 1/1)  
## INFO [19:01:43.536] [mlr3] Applying learner 'impute\_clrpart.impute\_regrpart.scalemaxabs.regr.kknn' on task 'df' (iter 1/1)

nomconf <- eshiper[match(min(totalerror), totalerror),]  
cat(paste("El menor error producido es: ", min(totalerror),  
 "La mejor configuración de preprocesado será:" ,  
 as.character(eshiper[match(min(totalerror), totalerror),1]),  
 as.character(eshiper[match(min(totalerror), totalerror),2]),  
 as.character(eshiper[match(min(totalerror), totalerror),3]),  
 sep = "\n"))

## El menor error producido es:   
## 0.446936555517012  
## La mejor configuración de preprocesado será:  
## imputesample  
## imputemedian  
## scalerange

La combinación ganadora es: **imputesample, imputemedian y scalerange**; con un error, que es el menor de entre las 36 combinaciones, de .

Por tanto, el mejor preproceso de nuestros datos se consigue con una imputación de los NA’s de las variables categóricas según datos al azar, una imputación en las variables numéricas según la mediana, y un escalado según el rango.

## 3.2. Evaluación sin ajuste de hiper-parámetros

En este siguiente apartado compararemos varios métodos SIN ajuste de hiper-parámetros. Estos métodos son los siguientes: **regr.lm, rpart, vecino más cercano, cubist, y SVM lineal y radial (kernel gausiano)**.

Para todos los métodos citados, emplearemos la combinación ganadora de preproceso de los datos que vimos en el apartado anterior que era: **imputesample, imputemedian y scalerange**.

set.seed(100430523)  
  
t <- matrix(0,1,2)  
colnames(t)=c("Learner","Rae")  
  
metodos <- c("regr.lm","regr.rpart","regr.kknn","regr.cubist","regr.svm")   
  
df\_task <- as\_task\_regr(df, target="salida")  
  
res\_desc=rsmp("custom")  
res\_desc$instantiate(df\_task,   
 train=list(1:(6\*365)),   
 test=list((6\*365+1):(9\*365)))  
  
preproceso <- po("imputesample") %>>% po("imputemedian") %>>% po("scalerange")   
  
  
for (i in metodos) {  
 learner = lrn(i)  
 graph = preproceso %>>% po(learner)  
 secuencia = as\_learner(graph)  
 metodo\_resample = resample(task=df\_task,   
 learner=secuencia,   
 resampling=res\_desc)  
 rae = metodo\_resample$aggregate(msr("regr.rae"))  
 t <- rbind(t, c(i,rae))  
}

## INFO [19:02:04.161] [mlr3] Applying learner 'imputesample.imputemedian.scalerange.regr.lm' on task 'df' (iter 1/1)  
## INFO [19:02:07.509] [mlr3] Applying learner 'imputesample.imputemedian.scalerange.regr.rpart' on task 'df' (iter 1/1)  
## INFO [19:02:10.275] [mlr3] Applying learner 'imputesample.imputemedian.scalerange.regr.kknn' on task 'df' (iter 1/1)  
## INFO [19:02:13.272] [mlr3] Applying learner 'imputesample.imputemedian.scalerange.regr.cubist' on task 'df' (iter 1/1)  
## INFO [19:02:18.023] [mlr3] Applying learner 'imputesample.imputemedian.scalerange.regr.svm' on task 'df' (iter 1/1)

graph=preproceso %>>%  
 po(lrn("regr.svm",kernel="radial"))  
secuencia = as\_learner(graph)  
metodo\_resample = resample(task=df\_task,   
 learner=secuencia,   
 resampling=res\_desc)

## INFO [19:02:26.471] [mlr3] Applying learner 'imputesample.imputemedian.scalerange.regr.svm' on task 'df' (iter 1/1)

rae = metodo\_resample$aggregate(msr("regr.rae"))  
t <- rbind(t, c("radial.svm",rae))  
t <- data.frame(t[-1,], row.names = NULL)   
t$Rae <- round(as.numeric(t$Rae),3)  
t

## Learner Rae  
## 1 regr.lm 0.387  
## 2 regr.rpart 0.447  
## 3 regr.kknn 0.454  
## 4 regr.cubist 0.361  
## 5 regr.svm 0.376  
## 6 radial.svm 0.379

En esta tabla observamos que el mejor método es el **cubist**, ya que es el que tiene menor RAE con . Aunque, todos los modelos parecen válidos (unos mejores que otros, por ejemplo, los peores parecen rpart y KNN) ya que tienen un error parecido y cercano al de **cubist**.

## 3.3. Evaluación con ajuste de hiper-parámetros

A continuación, en este apartado, repetiremos el proceso del apartado anterior, pero en esta ocasión, en lugar de poner en omisión o default los hiper-parámetros, vamos a ajustarlos.

Los métodos que utilizaremos para calcular los mejores hiper-parámetros serán:

* Para el vecino más cercano (**KNN**), vamos a ajustar sólo el número de vecinos con **Grid Search**.
* Para **rpart** y **SVM** usad **Random Search**.
* De cubist y de regr.lm no vamos a hacer ajuste de hiper-parámetros.

En este apartado, respecto al tipo de resample, hemos tenido que cambiar de “custom”, que es el que estábamos usando hasta el momento a “holdoutorder” (manteniendo los 9 años de entrenamiento y los 3 de test). Nos hemos visto obligados a hacer este cambio porque de la otra manera no funcionaba el AutoTuner.

* KNN con Grid Search

set.seed(100430523)  
df\_task <- as\_task\_regr(df, target="salida")  
  
desc\_outer = rsmp("custom")  
desc\_outer$instantiate(df\_task, train=list(1:(6\*365)), test=list((6\*365+1):(9\*365)))  
  
source("ResamplingHoldoutOrder.R")  
desc\_inner = rsmp("holdoutorder", ratio=6/9)  
  
kknn\_space = ps(  
 regr.kknn.k = p\_fct(levels = seq(1,50,2))  
)  
  
generate\_design\_grid(kknn\_space, param\_resolutions = c(regr.kknn.k=10))

## <Design> with 25 rows:  
## regr.kknn.k  
## 1: 1  
## 2: 3  
## 3: 5  
## 4: 7  
## 5: 9  
## 6: 11  
## 7: 13  
## 8: 15  
## 9: 17  
## 10: 19  
## 11: 21  
## 12: 23  
## 13: 25  
## 14: 27  
## 15: 29  
## 16: 31  
## 17: 33  
## 18: 35  
## 19: 37  
## 20: 39  
## 21: 41  
## 22: 43  
## 23: 45  
## 24: 47  
## 25: 49  
## regr.kknn.k

terminator = trm("none")  
  
tuner = tnr("grid\_search", param\_resolutions=c(regr.kknn.k=10))  
  
kknn\_learner=preproceso %>>% po(lrn("regr.kknn"))  
  
kknn\_ajuste = AutoTuner$new(  
 learner = kknn\_learner,  
 resampling = desc\_inner,  
 measure = msr("regr.rae"),  
 search\_space = kknn\_space,  
 terminator = terminator,  
 tuner = tuner,  
 store\_tuning\_instance = TRUE  
)  
  
lgr::get\_logger("mlr3")$set\_threshold("warn")  
lgr::get\_logger("bbotk")$set\_threshold("warn")  
  
kknn\_ajuste\_resample = resample(df\_task, kknn\_ajuste, desc\_outer, store\_models = TRUE )  
  
kknn\_ajuste\_rae = kknn\_ajuste\_resample$aggregate(msr("regr.rae"))  
print(round(kknn\_ajuste\_rae,3))

## regr.rae   
## 0.431

Vemos como mejora el método KNN con el ajuste de hiper-parámetros, consiguiendo un RAE de , frente al que obtuvimos sin el ajuste de hiper-parámetros.

* rpart con Random Search

set.seed(100430523)  
  
desc\_outer = rsmp("custom")  
desc\_outer$instantiate(df\_task, train=list(1:(6\*365)), test=list((6\*365+1):(9\*365)))  
desc\_inner = rsmp("holdoutorder", ratio=6/9)  
rpart\_space = ps(  
 regr.rpart.minsplit = p\_int(lower=10, upper=20),   
 regr.rpart.maxdepth = p\_int(lower = 2,upper = 6)  
)  
terminator = trm("evals", n\_evals = 20)  
tuner = tnr("random\_search")  
secuencia = preproceso %>>% po(lrn("regr.rpart"))  
  
rpart\_ajuste = AutoTuner$new(  
 learner = secuencia,  
 resampling = desc\_inner,  
 measure = msr("regr.rae"),  
 search\_space = rpart\_space,  
 terminator = terminator,  
 tuner = tuner  
)  
rpart\_ajuste\_resample = resample(df\_task, rpart\_ajuste, desc\_outer, store\_models = T )  
  
rae=rpart\_ajuste\_resample$aggregate(msr("regr.rae"))  
print(round(rae,3))

## regr.rae   
## 0.448

En esta ocasión, el ajuste de hiper-parámetros no sirve para mejorar la precisión de rpart. Con ajuste de hiper-parámetros el RAE es de y sin ajuste era de

* SVM con Random Search

set.seed(100430523)  
  
secuencia = preproceso %>>% po(lrn("regr.svm", kernel="radial", type = "eps-regression"))  
svm\_space = ps(  
 regr.svm.cost = p\_dbl(lower=-3, upper=3, trafo=function(x)10^x),   
 regr.svm.gamma = p\_dbl(lower=-3, upper=3, trafo=function(x)10^x)  
)  
generate\_design\_random(svm\_space, 50)

## <Design> with 50 rows:  
## regr.svm.cost regr.svm.gamma  
## 1: -2.9153445 -1.83254074  
## 2: -0.2701687 -2.80301032  
## 3: 1.1107157 -0.26267832  
## 4: 2.9750087 -0.92522955  
## 5: -2.3809552 1.77654618  
## 6: -0.5303984 -2.96078547  
## 7: 1.6721896 -0.92105382  
## 8: 2.3537237 -0.20003137  
## 9: -2.1635871 -2.53212678  
## 10: 0.6233513 2.65643814  
## 11: -1.9157116 -1.92249955  
## 12: 1.8551091 2.85439517  
## 13: 1.6010648 2.94275523  
## 14: 0.8425719 -2.11336215  
## 15: -2.0291448 2.73334289  
## 16: 1.6688596 -0.03116505  
## 17: -1.3344798 0.91231936  
## 18: 2.7539581 -1.84630043  
## 19: 0.3316641 -1.08523395  
## 20: -0.4364394 2.99473168  
## 21: 1.3946994 -2.60355642  
## 22: -0.3313310 1.72763554  
## 23: 1.2337795 0.61330387  
## 24: -2.6783574 -2.38212860  
## 25: -2.2693785 -1.27066715  
## 26: -0.1651306 0.18445258  
## 27: -1.7740386 -1.04945869  
## 28: 1.9109478 0.13668798  
## 29: 2.7965119 2.01159190  
## 30: 1.4422947 1.17734919  
## 31: 2.3260438 -0.15644802  
## 32: -1.8693057 -2.27965212  
## 33: 1.9081737 -0.64587111  
## 34: -0.6729063 -2.81993918  
## 35: -1.7535507 -2.98876099  
## 36: 1.8730255 0.44699877  
## 37: -0.3392484 -0.92088537  
## 38: -1.9164986 2.30907022  
## 39: -2.5111247 -1.09645120  
## 40: -2.7396833 -1.24562727  
## 41: -0.9688626 2.04799109  
## 42: -2.5487669 1.30241563  
## 43: -1.5407977 2.81691473  
## 44: 2.6013589 2.19773493  
## 45: 2.6961758 2.45533024  
## 46: -1.5341794 1.89411511  
## 47: -0.7446686 -2.16348896  
## 48: -2.6855168 1.74841826  
## 49: -2.5895437 0.34953024  
## 50: -2.0825479 -1.81925306  
## regr.svm.cost regr.svm.gamma

terminator = trm("evals", n\_evals = 5)  
tuner = tnr("random\_search")  
svm\_ajuste = AutoTuner$new(  
 learner = secuencia,  
 resampling = desc\_inner,  
 measure = msr("regr.rae"),  
 search\_space = svm\_space,  
 terminator = terminator,  
 tuner = tuner,  
 store\_tuning\_instance = TRUE  
)  
svm\_ajuste\_resample = resample(df\_task, svm\_ajuste, desc\_outer, store\_models = TRUE )  
  
rae <- svm\_ajuste\_resample$aggregate(msr("regr.rae"))  
print(round(rae,3))

## regr.rae   
## 0.742

El RAE para el caso del SVM es de con hiper-parámetros, un error demasiado alto comparado con el caso sin ajuste de hiper-parámetros.

## 3.4. Métodos de ensembles con y sin ajuste de hiper-parámetros

Los métodos de ensambles que vamos a estudiar son Random Forest y Gradient Boosting.

Como hemos encontrado dos tipos de learner para cada método y vamos a calcular el RAE con ambos. Nos quedaremos con ranger y xgboost para ajustar hiper-parámetos como pide la práctica.

### Sin ajuste

set.seed(100430523)  
  
t <- matrix(0,1,2)  
colnames(t)=c("Learner","Rae")  
  
metodos <- c("regr.ranger","regr.randomForest","regr.xgboost","regr.gbm" )   
  
df\_task <- as\_task\_regr(df, target="salida")  
  
res\_desc=rsmp("custom")  
res\_desc$instantiate(df\_task,   
 train=list(1:(6\*365)),   
 test=list((6\*365+1):(9\*365)))  
  
preproceso <- po("imputesample") %>>% po("imputemedian") %>>% po("scalerange")   
  
for (i in metodos) {  
 learner = lrn(i)  
 graph = preproceso %>>% po(learner)  
 secuencia = as\_learner(graph)  
 metodos\_resample = resample(task=df\_task,   
 learner=secuencia,   
 resampling=res\_desc)  
 rae = metodos\_resample$aggregate(msr("regr.rae"))  
 t <- rbind(t, c(i,rae))  
}

## Distribution not specified, assuming gaussian ...

t <- data.frame(t[-1,], row.names = NULL)   
t$Rae <- round(as.numeric(t$Rae),3)  
t

## Learner Rae  
## 1 regr.ranger 0.374  
## 2 regr.randomForest 0.364  
## 3 regr.xgboost 1.745  
## 4 regr.gbm 0.385

### Con ajuste

Vamos a ajustar solo los dos hiper-parámetros más importantes de cada modelo.

Random Forest con ranger:

set.seed(100430523)  
  
ranger\_space = ps(  
 regr.ranger.mtry = p\_int(lower=1, upper=10),  
 regr.ranger.max.depth = p\_int(lower=2, upper=10)  
   
)  
terminator = trm("evals", n\_evals = 10)  
tuner = tnr("random\_search")  
secuencia = preproceso %>>% po(lrn("regr.ranger"))  
  
ranger\_ajuste = AutoTuner$new(  
 learner = secuencia,  
 resampling = desc\_inner,  
 measure = msr("regr.rae"),  
 search\_space = ranger\_space,  
 terminator = terminator,  
 tuner = tuner  
)  
ranger\_ajuste\_resample = resample(df\_task, ranger\_ajuste, desc\_outer )  
  
rae=ranger\_ajuste\_resample$aggregate(msr("regr.rae"))  
print(rae)

## regr.rae   
## 0.3832021

El RAE con ranger para Random Forest es de: .

Gradient Boosting:

set.seed(100430523)  
  
xgboost\_space = ps(  
 regr.xgboost.eta = p\_dbl(lower=0.001, upper=0.6),  
 regr.xgboost.max\_depth = p\_int(lower=1, upper=8),  
 regr.xgboost.nrounds = p\_int(lower=1, upper=8)  
)  
terminator = trm("evals", n\_evals = 10)  
tuner = tnr("random\_search")  
secuencia = preproceso %>>% po(lrn("regr.xgboost"))  
  
xgboost\_ajuste = AutoTuner$new(  
 learner = secuencia,  
 resampling = desc\_inner,  
 measure = msr("regr.rae"),  
 search\_space = xgboost\_space,  
 terminator = terminator,  
 tuner = tuner  
)  
xgboost\_ajuste\_resample = resample(df\_task, xgboost\_ajuste, desc\_outer )  
  
rae=xgboost\_ajuste\_resample$aggregate(msr("regr.rae"))  
print(rae)

## regr.rae   
## 0.3926469

El RAE con Gradient Boosting es de: .

# 4. Modelo final

Como hemos demostrado en todos los apartados anteriores, los dos métodos con menor RAE son **cubist** (entre los métodos no ensambles) y randomforest sin ajuste de hiper-parámetros (entre los métodos de ensambles).

Entre todos los modelos, en el que obtenemos un menor RAE y por tanto es el mejor modelo para procesar nuestros datos es: **cubist**. Con un RAE de

Una vez escogido este mejor modelo, vamos a estimar el error de dicho modelo con los años que habíamos reservado para test.

set.seed(100430523)  
df\_task=as\_task\_regr(df, target="salida")  
  
res\_desc=rsmp("custom")  
res\_desc$instantiate(df\_task, train=list(1:(9\*365)), test=list((9\*365+1):(12\*365)))  
  
secuencia= preproceso %>>% po(lrn("regr.cubist"))  
secuencia = as\_learner(secuencia)  
  
final\_resample = resample(task=df\_task,   
 learner=secuencia,   
 resampling=res\_desc)  
rae=final\_resample$aggregate(msr("regr.rae"))  
print(rae)

## regr.rae   
## 0.3836791

El error estimado es de .

modelo\_final <- secuencia$train(df\_task)  
saveRDS(modelo\_final, "modelo\_final.rds")

Por último usaremos este modelo para predecir los datos de los próximos años para los datos de competición.

noms\_compet <- colnames(datos\_compet)  
Nas\_compet <- round(1 - skim(datos\_compet)$complete\_rate,3)  
datos\_compet\_RC <- remove\_constant(datos\_compet, na.rm = T, quiet = F)

## Removing 9 constant columns of 75 columns total (Removed: dswrf\_s5\_1, pwat\_ea1\_1, pwat\_ea3\_1, spfh\_2m4\_1, tcdc\_ea1\_1, tcdc\_ea4\_1, tcolc\_e4\_1, tcolc\_e5\_1, tmp\_sfc1\_1).

datos\_compet\_RC\_RNA <- datos\_compet\_RC[,!(names(datos\_compet\_RC) %in% noms\_compet[Nas\_compet >= 0.8])]  
  
df2 <- datos\_compet\_RC\_RNA  
  
charcol <- c(colnames(df2[, sapply(df2, class) == 'character']))  
  
for (i in charcol){  
 df2[[i]] <- as.factor(df2[[i]])  
}  
  
c(colnames(df2[, sapply(df2, class) == 'character'])) #Ya no hay columnas que sea 'character', solo factores

## character(0)

source("extras\_from\_mlr.R")  
df2 <- createDummyFeatures(df2, target = "salida")  
  
set.seed(100430523)  
  
predicciones\_compet <- modelo\_final$predict\_newdata(df2)  
write.csv(predicciones\_compet$response,"predicciones\_compet\_9.csv", row.names = FALSE)

# 5. Hyperband

Tanto las técnicas vistas anteriormente como Grid Search y Random Search, como ésta de la que estamos hablando, que es Hyperband, se tratan de métodos de optimización de hiper-parámetros. El ajuste de hiper-parámetros lo planteamos como un problema de optimización, y consiste en encontrar una configuración de hiper-parámetros para un determinado modelo, que permita obtener el mayor rendimiento posible de la función que modeliza un conjunto de validación, a través precisamente de esos hiper-parámtros.

La diferencia fundamental entre Grid y Random Search con Hyperband, es que los dos primeros modelos buscan a ciegas entre el espacio de búsqueda de hipér-parámetros. Grid discretiza la búsqueda obteniendo todas las configuraciones posibles de hiper-parámetros y eligiendo la mejor, lo que hace que pueda ser muy elevado el número de configuraciones que tiene que calcular; mientras que con Random Search eliges aleatoriamente el número de configuraciones que quieres estudiar, lo que compensa el anterior problema. Ambos métodos son considerados dentro del grupo Black-Box Optimization.

Por su parte, Hyperband, la ventaja que tiene respecto a los dos anteriores métodos, es que se desempeña relativamente bien en conjuntos de datos con alta dimensionalidad, como es nuestro caso. Hyperband es como si llevara a cabo Grid Search, pero para varios valores de n. Para ello, Hyperband emplea un presupuesto mínimo asociado con cada valor de n, que se asignará a todas las configuraciones antes de que se descarten algunas. Por lo que, un valor de n mayor, implica una r menor y, un early stopping más agresivo (se descartarán más configuraciones en menor tiempo).

Vamos a utilizar Hyperband para ajustar los hiper-parámetros del método xgboost y lo compararemos con el realizado en la práctica anteriormente con Random Search.

set.seed(100430523)  
  
learner = lrn("regr.xgboost",  
 nrounds = to\_tune(p\_int(27, 243, tags = "budget")),  
 eta = to\_tune(1e-4, 1, logscale = TRUE),  
 max\_depth = to\_tune(1, 20),  
 colsample\_bytree = to\_tune(1e-1, 1),  
 colsample\_bylevel = to\_tune(1e-1, 1),  
 lambda = to\_tune(1e-3, 1e3, logscale = TRUE),  
 alpha = to\_tune(1e-3, 1e3, logscale = TRUE),  
 subsample = to\_tune(1e-1, 1)  
)  
secuencia = preproceso %>>% learner  
  
instance = tune(  
 method = "hyperband",  
 task = df\_task,  
 learner = secuencia,  
 resampling = desc\_inner,  
 measures = msr("regr.rae"),  
 eta = 3  
)  
  
instance$result

## regr.xgboost.nrounds regr.xgboost.eta regr.xgboost.max\_depth  
## 1: 81 -0.9920708 6  
## regr.xgboost.colsample\_bytree regr.xgboost.colsample\_bylevel  
## 1: 0.5252304 0.2869674  
## regr.xgboost.lambda regr.xgboost.alpha regr.xgboost.subsample  
## 1: 5.98985 -0.6048392 0.9781593  
## learner\_param\_vals x\_domain regr.rae  
## 1: <list[12]> <list[8]> 0.3845171

El error estimado para Hyperband es de: , que mejora el anterior ajuste con Random Search para el Gradient Boosting.

# Conclusión

Como hemos visto anteriormente, de entre todos los modelos estudiados y analizados, el mejor modelo para procesar nuestros datos, que es el modelo final elegido para predecir los datos de los próximos años (los datos de competición), ha sido el **Cubist** con **imputesample**, **imputemedian** y **scalerange**, porque con estos obteníamos el menor RAE, que era de . Para los datos de test el RAE final es de .

# Bibliografía

(n.d.). Hyperband for mlr3 • mlr3hyperband. Retrieved December 3, 2022, from <https://mlr3hyperband.mlr-org.com/>

*Hyperparameter optimization con Hyperband*. (2021, April 13). Peep data. Retrieved December 3, 2022, from <https://peepdata.github.io/hyperparameter-optimization-hyperband/>