

EST-25134: Aprendizaje Estadístico

Profesor: Alfredo Garbuno Iñigo — Primavera, 2023 — Modelos en ensamble.

Objetivo: En esta sección estudiaremos una forma de incorporar varios modelos para crear un modelo predictivo mas fuerte que un modelo individual. La estrategia está basada en una técnica de remuestreo que ya hemos estudiado previamente.

Lectura recomendada: Sección 8.2 de James et al. [3]. Capítulo 15 de Hastie et al. [2]. Capítulo 5 (*bagging*) y 7 (*random forest*) de Greenwell [1].

1. INTRODUCCIÓN

Anteriormente, vimos los modelos predictivos basados en árboles de decisión (regresión y clasificación). En esta sección del curso estudiaremos distintas estrategias para combinarlos para obtener un mejor modelo predictivo al costo de interpretabilidad. Algunos de estos modelos continúan representando el estado del arte en competencias como [Kaggle](#) para datos tabulares.

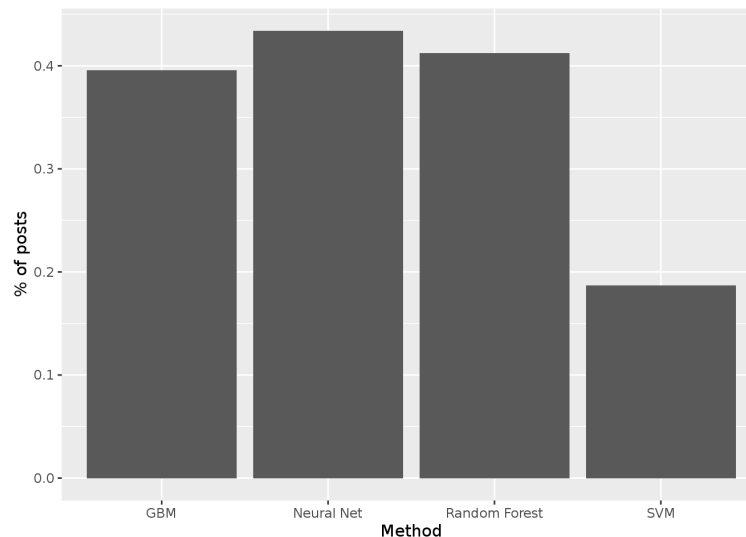


FIGURA 1. Algoritmos que tienden a quedar en los primeros lugares en las competencias de Kaggle. Tomado de [aquí](#).

De igual manera se han publicado algunos reportes donde se confirma que en particular para datos tabulares modelos de ensamble basados en árboles tienen mejor capacidad predictiva.

[Submitted on 18 Jul 2022]

Why do tree-based models still outperform deep learning on tabular data?

Léo Grinsztajn (SODA), Edouard Oyallon (ISIR, CNRS), Gaël Varoquaux (SODA)

While deep learning has enabled tremendous progress on text and image datasets, its superiority on tabular data is not clear. We contribute extensive benchmarks of standard and novel deep learning methods as well as tree-based models such as XGBoost and Random Forests, across a large number of datasets and hyperparameter combinations. We define a standard set of 45 datasets from varied domains with clear characteristics of tabular data and a benchmarking methodology accounting for both fitting models and finding good hyperparameters. Results show that tree-based models remain state-of-the-art on medium-sized data (~10K samples) even without accounting for their superior speed. To understand this gap, we conduct an empirical investigation into the differing inductive biases of tree-based models and Neural Networks (NNs). This leads to a series of challenges which should guide researchers aiming to build tabular-specific NNs: 1. be robust to uninformative features, 2. preserve the orientation of the data, and 3. be able to easily learn irregular functions. To stimulate research on tabular architectures, we contribute a standard benchmark and raw data for baselines: every point of a 20 000 compute hours hyperparameter search for each learner.

[Submitted on 6 Jun 2021 (v1), last revised 23 Nov 2021 (this version, v2)]

Tabular Data: Deep Learning is Not All You Need

Ravid Shwartz-Ziv, Amitai Armon

A key element in solving real-life data science problems is selecting the types of models to use. Tree ensemble models (such as XGBoost) are usually recommended for classification and regression problems with tabular data. However, several deep learning models for tabular data have recently been proposed, claiming to outperform XGBoost for some use cases. This paper explores whether these deep models should be a recommended option for tabular data by rigorously comparing the new deep models to XGBoost on various datasets. In addition to systematically comparing their performance, we consider the tuning and computation they require. Our study shows that XGBoost outperforms these deep models across the datasets, including the datasets used in the papers that proposed the deep models. We also demonstrate that XGBoost requires much less tuning. On the positive side, we show that an ensemble of deep models and XGBoost performs better on these datasets than XGBoost alone.

2. REMUESTREO O BOOTSTRAP

Utilizar técnicas de remuestreo nos permite cuantificar la variabilidad de un estimador estadístico sin necesidad de invocar un régimen asintótico para el procedimiento. Asimismo, nos permite controlar, hasta cierto punto, la variabilidad de nuestros estimadores.[†]

[†]: Mas información de esto en el curso de EST-24107: Simulación.

Por ejemplo, consideremos la situación en donde tenemos una muestra de n observaciones Z_1, \dots, Z_n las cuales provienen de una distribución con varianza σ^2 . Es fácil demostrar que la varianza de la media \bar{Z}_n tiene una varianza σ^2/n .

2.0.1. Importante: Esto quiere decir, que podemos 1) generar muestras, 2) promediar y, entonces, reducimos la varianza estimada!

2.0.2. Para pensar: Usualmente no tenemos acceso al proceso generador de datos (ya sea $\mathbb{P}_{X,Y}$ ó \mathbb{P}_X). ¿Qué estrategia podemos utilizar?

2.1. Bootstrap

Podemos utilizar la muestra $z_1, \dots, z_n \stackrel{\text{iid}}{\sim} \pi$ como un *proxy* de la población de la cual queremos generar observaciones. En este sentido, consideramos que la función de acumulación empírica (ECDF, por sus siglas en inglés) es un *buen* estimador de la función de probabilidad (ó CDF por sus siglas en inglés)

$$\pi[X \leq x] \approx \hat{\pi}_n[X \leq x] = \frac{1}{n} \sum_{i=1}^n I_{[z_i \leq x]}. \quad (1)$$

Con mi muestra, entonces, podemos calcular algún estimador de una característica poblacional de interés

$$\hat{\theta}_n = t(z_1, \dots, z_n). \quad (2)$$

Con este procedimiento podemos generar B conjuntos de datos

$$z_1^{(b)}, \dots, z_n^{(b)} \stackrel{\text{iid}}{\sim} \hat{\pi}_n, \quad b = 1, \dots, B, \quad (3)$$

para obtener una colección de estimadores $\hat{\theta}_n^{(b)} = t(z_1^{(b)}, \dots, z_n^{(b)})$ y, a través de un promedio, obtener un estimador

$$\bar{\theta}_{B,n}^{(\text{bag})} = \frac{1}{B} \sum_{b=1}^B \hat{\theta}_n^{(b)}, \quad (4)$$

con varianza que se reduce a una tasa $1/B$.

El muestreo $z_1^{(b)}, \dots, z_n^{(b)} \stackrel{\text{iid}}{\sim} \hat{\pi}_n$ implica tomar muestras **con** reemplazo del conjunto de datos observado. Nota que las remuestras son del mismo tamaño que la muestra original. Es decir, cada remuestra b tiene n observaciones. Como el procedimiento es con reemplazo, esto puede ocasionar que pueda haber algunas observaciones que se repitan en la remuestra.

2.2. Ejemplo: Suavizadores

La estrategia de remuestreo nos puede ayudar a cuantificar la estabilidad de ciertos estimadores. Por ejemplo, consideremos los datos que teníamos sobre el ingreso para un conjunto de 150 observaciones. El interés es construir un suavizador que relacione **Edad** con **Ingreso**.

```

1 # A tibble: 150 × 5
2   year   age wage education      hi.income
3   <int> <dbl> <dbl> <fct>         <dbl>
4 1  2003    53  81.6 4. College Grad          0
5 2  2008    50  82.7 4. College Grad          0
6 3  2006    35 155. 4. College Grad          0
7 # ... with 147 more rows
8 # Use 'print(n = ...)' to see more rows

```

Utilizaremos un suavizador de *splines* con 15 grados de libertad, ver Fig. 2.

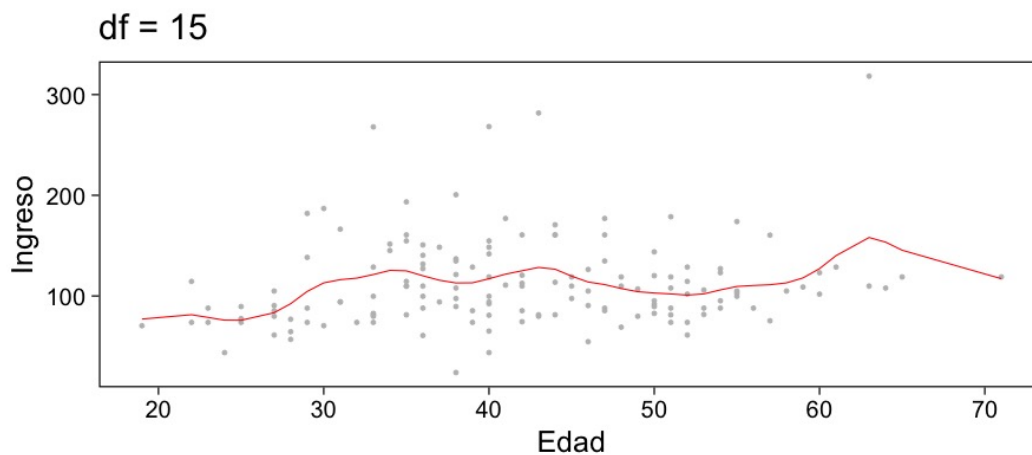


FIGURA 2. Suavizador por splines con 15 grados de libertad.

A través de remuestreo podemos cuantificar la estabilidad de dicha estimación, ver Fig. 3. La estabilidad también la podemos graficar por medio de intervalos de confianza. Ver Fig. 4.

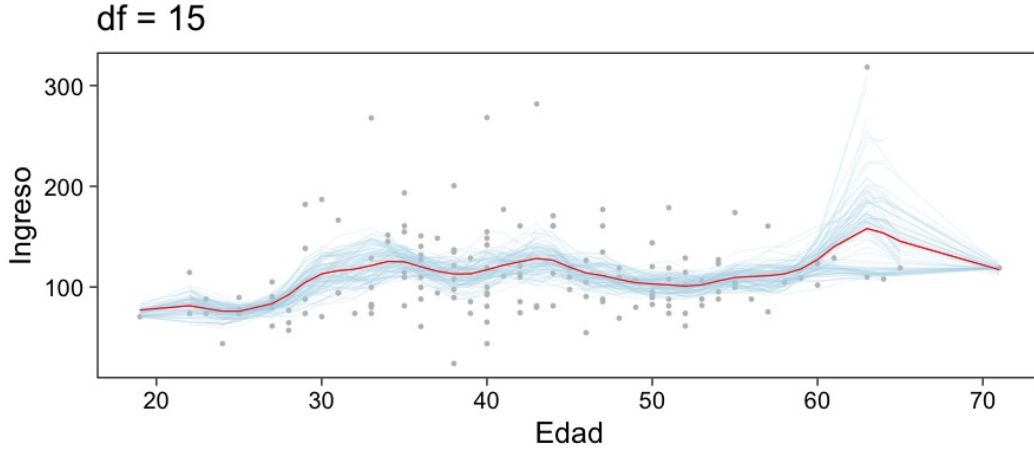


FIGURA 3. Suavizador por splines con 15 grados de libertad, réplicas con remuestreo.

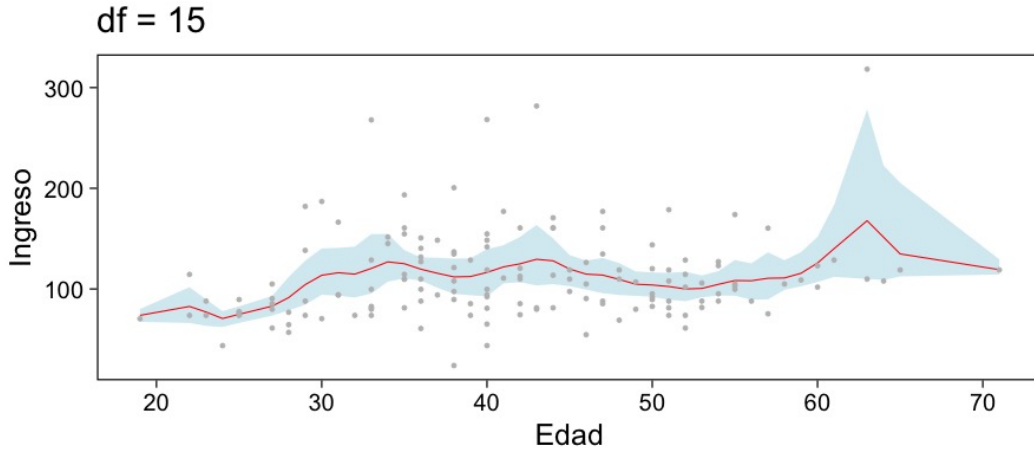


FIGURA 4. Suavizador por splines con 15 grados de libertad, réplicas con remuestreo.

3. BOOTSTRAPPED AGGREGATION: BAGGING

En el contexto de modelado predictivo nos interesa **estimar la relación** que existe entre atributos x y una respuesta de interés y por medio de una función $f : \mathcal{X} \mapsto \mathcal{Y}$. Dicho estimador, lo denotamos por \hat{f}_n haciendo énfasis en que ha sido construido con una muestra de tamaño n . Recordemos que este estimador en el contexto de modelado predictivo es resultado de un problema de optimización con una función de pérdida adecuada.

Si utilizamos *bootstrap*, para cada uno de los B conjuntos de entrenamiento estimamos $\hat{f}_n^{(b)}$ con $b = 1, \dots, B$ para poder hacer predicciones por medio de

$$\hat{f}_{B,n}^{(\text{bag})}(x) = \frac{1}{B} \sum_{b=1}^B \hat{f}_n^{(b)}(x), \quad (5)$$

esto lo llamamos **bootstrap aggregation** o **bagging**.

3.1. En problemas de clasificación

Para problemas de clasificación podemos considerar las predicciones de cada uno de los modelos $\hat{f}_n^{(b)}$ y tomar la clase con **más votos** dentro del conjunto de B predictores.

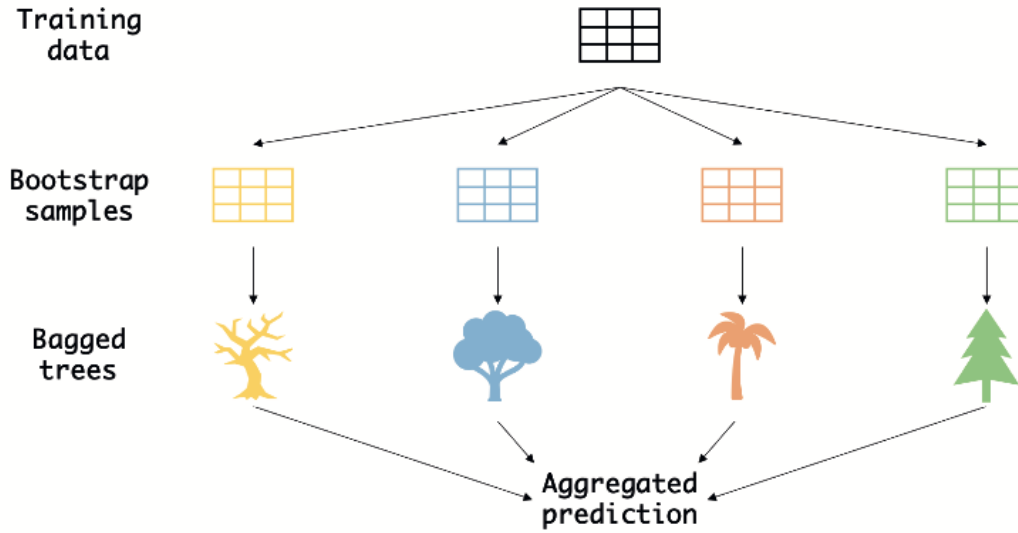


FIGURA 5. Ilustración esquemática de bagging por medio de árboles de decisión. Tomada de [1].

3.2. Error de generalización

- Usando *bootstrap* entrenamos con cada uno de los conjuntos de datos remuestreados.
- Cada conjunto remuestreado utiliza, en promedio, $2/3$ de los datos originales.
- El conjunto no utilizado lo llamamos **conjunto fuera de bolsa** (*out-of-bag*, OOB).
- Podemos obtener predicciones para cada observación $i = 1, \dots, n$ cuando se encuentra en algún conjunto OOB. En promedio, tenemos $B/3$ predicciones para cada observación, las cuales podemos promediar para obtener la predicción final.
- Esto es un estimador de LOO-CV utilizando **bagging**.

3.3. Observaciones

- El estimador $\hat{f}_{B,n}^{(\text{bag})}$ es un estimador Monte Carlo. ¿De qué?
- El estimador $\hat{f}_{B,n}^{(\text{bag})} \rightarrow \hat{f}_n$ con $B \rightarrow \infty$ en cada uno de los puntos a evaluar x .
- Cuando los atributos están altamente correlacionados los árboles de decisión pueden presentar varianza alta. **Ventaja:** en esta situación **bagging** puede suavizar la varianza y reducir el error de generalización.

3.4. Bagging, regresión y MSE

- Si estamos en tareas de regresión y medimos el error de generalización por medio de pérdida cuadrática obtenemos lo siguiente

$$\mathbb{E} \left(y - \hat{f}^*(x) \right)^2 \geq \mathbb{E} \left(y - \mathbb{E} \hat{f}^*(x) \right)^2, \quad (6)$$

donde \hat{f}^* es una estimación por medio de una remuestra y $\mathbb{E} \hat{f}^*$ es el valor esperado de las estimaciones de f utilizando remuestras.

- Por lo tanto, *bagging* podrá disminuir el MSE.

3.5. Bagging y clasificación

- En problemas de clasificación, no tenemos descomposición aditiva de sesgo y varianza. A menos que ...

- El uso de **bagging** puede hacer de un mal clasificador, algo todavía peor. Consideremos el caso siguiente.

3.5.1. Bagging y clasificadores: Supongamos que tenemos un clasificador binario que asigna $Y = 1$ para todo x con probabilidad 0.4. ¿Cuál es la tasa de error de clasificación de este modelo? ¿Cuál sería la tasa de error de clasificación de un consenso con este modelo?

3.5.2. Bagging y la sabiduría de las masas: Supongamos que tenemos una colección de clasificadores independientes donde cada uno tiene una tasa de error de $\varepsilon < 0.5$, y sea

$$S_1(x) = \sum_{b=1}^B I[G^{(b)}(x) = 1], \quad (7)$$

el voto por consenso de que la instancia x pertenezca a la clase 1. Dado que los clasificadores son independientes entonces

$$S_1(x) \sim \text{Binomial}(B, 1 - \varepsilon), \quad (8)$$

donde

$$\mathbb{P}(\text{clasificación correcta}) = \mathbb{P}(S_1 > B/2) \approx 1, \quad (9)$$

con B suficientemente grande.

El resultado anterior se conoce como **Sabiduría de las masas** en donde se asume que cada clasificador es un clasificador **débil**. Con tasa de error ligeramente menor al azar. Para que el consenso de dichos clasificadores tenga buenos resultados se necesita, además, que los clasificadores sean **independientes**.

3.6. Observaciones

- Utilizar **bagging** en un problema de clasificación con árboles no es un procedimiento que utilice árboles independientes. Por lo tanto no hay garantía de que el consenso mejore el error de clasificación.
- En general para **bagging** estamos dispuestos a usar modelos de **alta varianza** puesto que el remuestreo se encarga de ayudarnos a contenerla.

4. APLICACIÓN: MISIONES DE ASTRONAUTAS

Ejemplo tomado de: [Bagging with tidymodels and #TidyTuesday astronaut missions](#). Cargamos la información y exploramos cuál es la nave (*spacecraft* ?) que mas se utiliza en misiones espaciales.

```
1 astronauts >
2   count(in_orbit, sort = TRUE) >
3   print(n = 5)
```

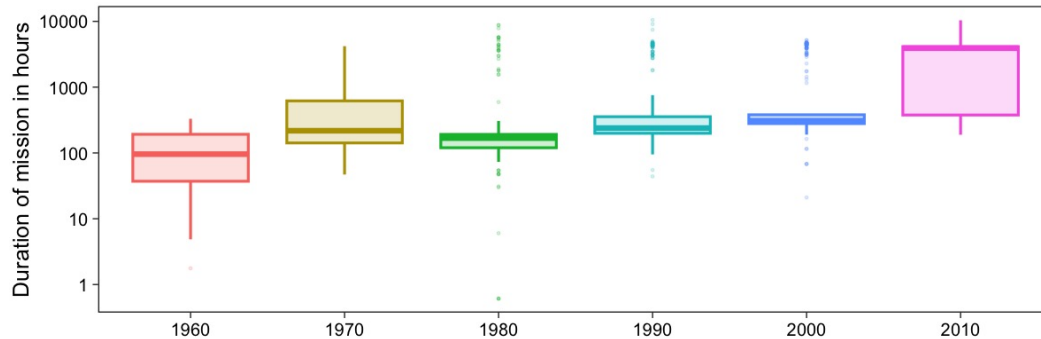
```
1 # A tibble: 289 × 2
2   in_orbit      n
3   <chr>      <int>
4 1 ISS          174
5 2 Mir           71
6 3 Salyut 6      24
```

```

7 4 Salyut 7      24
8 5 STS-42       8
9 # ... with 284 more rows
10 # Use 'print(n = ...)' to see more rows

```

Podemos explorar cómo ha cambiado la duración de las misiones a lo largo del tiempo.



Objetivo: predecir la duración de una misión espacial en función de algunas características de la misión.

```

1 # A tibble: 1,270 × 5
2   hours_mission military_civilian occupation year_of_mission in_orbit
3   <dbl> <chr> <chr> <dbl> <chr>
4 1     0.571 military pilot      1961 Vostok 2
5 2     3.22 military pilot      1961 Vostok 2
6 3     1.61 military pilot      1962 MA-6
7 4     5.36 military psp       1998 STS
8 5     1.61 military pilot      1962 Mercury-...Atlas
9 # ... with 1,265 more rows
10 # Use 'print(n = ...)' to see more rows

```

4.1. Proceso de modelado

```

1 set.seed(123)
2 astro_split <- initial_split(astronauts_df, strata = hours_mission)
3 astro_train <- training(astro_split)
4 astro_test <- testing(astro_split)

```

```

1 astro_recipe <- recipe(hours_mission ~ ., data = astro_train) >
2   step_other(occupation, in_orbit,
3             threshold = 0.005, other = "Other"
4             ) >
5   step_dummy(all_nominal_predictors())

```

```

1 astro_wf <- workflow() %>%
2   add_recipe(astro_recipe)
3
4 astro_wf

```

```
1 Workflow
2 Preprocessor: Recipe
3 Model: None
4
5 Preprocessor
6 2 Recipe Steps•
7
8 step_other()•
9 step_dummy()
```

4.2. Ajuste de bagging con árboles de clasificación

Especificamos el modelo que utilizaremos

```
1 library(baguette)
2
3 tree_spec <- bag_tree() %>%
4   set_engine("rpart", times = 25) %>%
5   set_mode("regression")
6
7 tree_spec
```

```
1 Bagged Decision Tree Model Specification (regression)
2
3 Main Arguments:
4   cost_complexity = 0
5   min_n = 2
6
7 Engine-Specific Arguments:
8   times = 25
9
10 Computational engine: rpart
```

Ajustamos el modelo a nuestros de datos de entrenamiento.

```
1 tree_rs <- astro_wf %>%
2   add_model(tree_spec) %>%
3   fit(astro_train)
4
5 tree_rs
```

```
1 == Workflow [trained] =====
2 Preprocessor: Recipe
3 Model: bag_tree()
4
5 -- Preprocessor -----
6 2 Recipe Steps
7
8 - step_other()
9 - step_dummy()
10
11 -- Model-----
12 Bagged CART (regression with 25 members)
13
14 Variable importance scores include:
```



```

15
16 # A tibble: 13 × 4
17   term                                value std.error  used
18   <chr>                                <dbl>   <dbl> <int>
19 1 year_of_mission                    799.    28.9    25
20 2 in_orbit_Other                     439.    48.0    25
21 3 occupation_flight.engineer         284.    30.8    25
22 4 in_orbit_STS                       265.    23.4    25
23 5 in_orbit_Mir                       161.    16.4    25
24 6 in_orbit_Salyut                     91.9     7.99    25
25 7 occupation_pilot                    73.5    16.3    25
26 8 occupation_msp                      71.6     7.11    25
27 9 occupation_other..space.tourist..  44.1     4.04    25
28 10 military_civilian_military         37.6     3.29    25
29 11 occupation_psp                     19.5     4.39    25
30 12 occupation_Other                   18.9     2.58    21
31 13 in_orbit_Mir.EP                    11.5     2.47    25

```

Realizamos predicciones en nuestro conjunto de prueba para poder reportar capacidad predictiva.

```

1 test_rs <- astro_test %>%
2   bind_cols(predict(tree_rs, astro_test)) %>%
3   rename(.pred_tree = .pred)
4
5 test_rs > print(n = 5, width = 73)

```

```

1 # A tibble: 318 × 6
2   hours_mission military_civilian occupation year_of_...1mi in_...2or ....3pred
3           <dbl> <chr>           <chr>           <dbl> <chr>       <dbl>
4 1         3.22 military         pilot           1961 ...Vostok    1.31
5 2         1.61 military         pilot           1962 MA-6        2.94
6 3         5.36 military         psp             1998 STS         5.55
7 4         6.05 military         pilot           1970 Soyuz 9     5.30
8 5         5.93 military         commander       1974 Soyuz ...   5.93
9 # ... with 313 more rows, and abbreviated variable names 1year_of_mission,
10 # 2in_orbit, 3.pred_tree
11 # Use 'print(n = ...)' to see more rows

```

Por ejemplo, podemos reportar un conjunto de métricas

```

1 test_rs %>%
2   metrics(hours_mission, .pred_tree)

```

```

1 # A tibble: 3 × 3
2   .metric .estimator .estimate
3   <chr>   <chr>         <dbl>
4 1 rmse    standard       0.663
5 2 rsq     standard       0.769
6 3 mae     standard       0.356

```

Podemos jugar con el modelo para tratar de entender la respuesta del modelo a situaciones de interés.

```

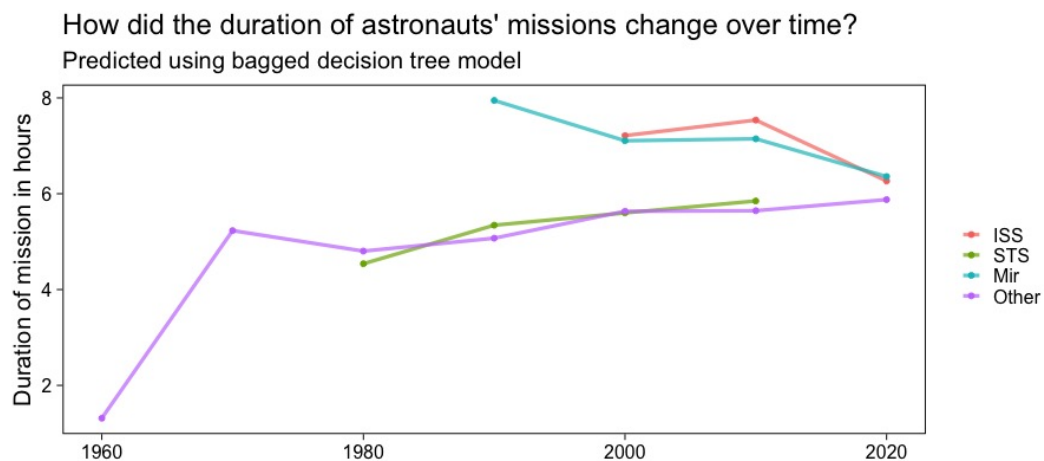
1 new_astronauts <- crossing(
2   in_orbit = fct_inorder(c("ISS", "STS", "Mir", "Other")),
3   military_civilian = "civilian",
4   occupation = "Other",
5   year_of_mission = seq(1960, 2020, by = 10)
6 ) %>%
7   filter(
8     !(in_orbit == "ISS" & year_of_mission < 2000),
9     !(in_orbit == "Mir" & year_of_mission < 1990),
10    !(in_orbit == "STS" & year_of_mission > 2010),
11    !(in_orbit == "STS" & year_of_mission < 1980)
12  )
13
14 new_astronauts

```

```

1 # A tibble: 18 × 4
2   in_orbit military_civilian occupation year_of_mission
3   <fct>      <chr>          <chr>      <dbl>
4 1 ISS      civilian          Other      2000
5 2 ISS      civilian          Other      2010
6 3 ISS      civilian          Other      2020
7 4 STS      civilian          Other      1980
8 5 STS      civilian          Other      1990
9 6 STS      civilian          Other      2000
10 7 STS      civilian          Other      2010
11 8 Mir      civilian          Other      1990
12 9 Mir      civilian          Other      2000
13 10 Mir     civilian          Other      2010
14 11 Mir     civilian          Other      2020
15 12 Other   civilian          Other      1960
16 13 Other   civilian          Other      1970
17 14 Other   civilian          Other      1980
18 15 Other   civilian          Other      1990
19 16 Other   civilian          Other      2000
20 17 Other   civilian          Other      2010
21 18 Other   civilian          Other      2020

```



5. BOSQUES ALEATORIOS

El modelo propuesto de Bosques aleatorios (RF por sus siglas en inglés) ayuda a de-correlacionar un conjunto de árboles. Para lograr esto seguimos utilizando remuestreo para seleccionar conjuntos de datos de entrenamiento. Al mismo tiempo, con cada conjunto de remuestras, utilizamos un conjunto de m predictores al azar para entrenar. Esto es, utilizamos para cada remuestra, un subconjunto distinto de predictores para entrenar un árbol.

Usualmente consideramos $m \approx \sqrt{p}$. Esto permite restringir el espacio de búsqueda y dejar de utilizar consistentemente los mismos predictores en cada remuestra.

5.1. Motivación

Si consideramos la situación donde tenemos B variables iid cada una con varianza σ^2 entonces el promedio tendrá varianza igual σ^2/B . Si las variables son sólo id (números aleatorios de la misma población) con correlación positiva ρ , entonces el promedio tendrá varianza igual a

$$\rho\sigma^2 + \frac{1-\rho}{B}\sigma^2. \quad (10)$$

Incluso aunque tomemos un número suficiente de árboles para controlar el segundo término, el primer término no desvanece con $B \rightarrow \infty$. Es por esto que bosques aleatorios busca reducir la correlación entre árboles al permitir que se ajusten a conjuntos aleatorios (en observaciones y predictores) por medio de remuestreo.

5.2. Sobre-ajuste

- El consenso de votos tiende a ser robusto contra sobre-ajustar y si utilizamos una B (el número de árboles) suficientemente grande estabilizamos la variabilidad del error de generalización.
- Usualmente tenemos problemas de sobre-ajuste cuando el número de predictores es alto y el número de predictores relevantes para la predicción es pequeño.

5.3. Análisis de ajuste

La predicción de un bosque aleatorio se realiza por medio de

$$\hat{f}_{\text{RF}}(x) = \frac{1}{B} \sum_{b=1}^B \hat{f}^{(b)}(x) = \frac{1}{B} \sum_{b=1}^B T\left(x; \Theta(\mathcal{D}_n^{(b)})\right), \quad (11)$$

donde $T(x; \Theta)$ denota la predicción de un árbol utilizando los parámetros (variables de selección, puntos de corte) Θ . La notación $\Theta(\mathcal{D}_n)$ hace énfasis en que los parámetros que gobiernan el árbol fueron escogidos utilizando el conjunto de datos \mathcal{D}_n . El término $\mathcal{D}_n^{(b)}$ hace énfasis en que el conjunto de entrenamiento es una remuestra del conjunto original.

El predictor tiende a satisfacer la siguiente igualdad (ley de los grandes números, $B \rightarrow \infty$)

$$\hat{f}_{\text{RF}}(x) = \mathbb{E}_{\Theta|\mathcal{D}_n} T(x; \Theta(\mathcal{D}_n)), \quad (12)$$

donde hacemos énfasis en que es un valor esperado condicional en los datos de entrenamiento.

Nos interesa evaluar el **error estándar** de dicho estimador. Lo cual escribimos como

$$\text{SE}\left(\hat{f}_{\text{RF}}(x)\right)^2 = \mathbb{V}\left(\hat{f}_{\text{RF}}(x)\right) = \rho(x) \cdot \sigma^2(x), \quad (13)$$

donde:

- $\rho(x)$ es la correlación entre dos árboles

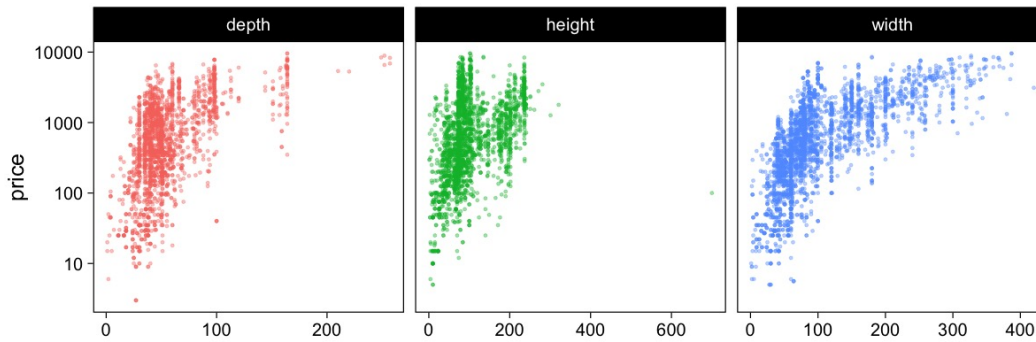
$$\rho(x) = \text{Corr}[T(x; \Theta_i(\mathcal{D}_n)), T(x; \Theta_j(\mathcal{D}_n))] . \quad (14)$$

- $\sigma^2(x)$ es la varianza de cualquier árbol

$$\sigma^2(x) = \mathbb{V}(T(x; \Theta(\mathcal{D}_n))) . \quad (15)$$

6. APLICACIÓN: PREDICCIÓN DE PRECIOS IKEA

Ejemplo tomado de: [Tune random forests for #TidyTuesday IKEA prices](#).



```
1 ikea_df <- ikea >
2   select(price, name, category, depth, height, width) >
3   mutate(price = log10(price)) >
4   mutate_if(is.character, factor)
5
6 ikea_df > print(n = 5)
```

```
1 # A tibble: 3,694 × 6
2   price name                category    depth height width
3   <dbl> <fct>                <fct>    <dbl>  <dbl> <dbl>
4 1  2.42 FREKVENS            Bar furniture    NA     99    51
5 2  3.00 NORDVIKEN          Bar furniture    NA    105    80
6 3  3.32 NORDVIKEN / NORDVIKEN Bar furniture    NA     NA    NA
7 4  1.84 STIG                Bar furniture    50    100    60
8 5  2.35 NORBERG            Bar furniture    60     43    74
9 # ... with 3,689 more rows
10 # Use 'print(n = ...)' to see more rows
```

```
1 set.seed(123)
2 ikea_split <- initial_split(ikea_df, strata = price)
3 ikea_train <- training(ikea_split)
4 ikea_test <- testing(ikea_split)
5
6 set.seed(234)
7 ikea_folds <- vfold_cv(ikea_train, strata = price)
```

```
1 library(textrecipes)
2 ranger_recipe <-
3   recipe(formula = price ~ ., data = ikea_train) >
```

```

4   step_other(name, category, threshold = 0.01) ▷
5   step_clean_levels(name, category) ▷
6   step_impute_knn(depth, height, width)
7
8   ranger_spec ←
9     rand_forest(mtry = tune(), min_n = tune(), trees = 1000) ▷
10    set_mode("regression") ▷
11    set_engine("ranger")
12
13   ranger_workflow ←
14     workflow() ▷
15     add_recipe(ranger_recipe) ▷
16     add_model(ranger_spec)
17
18   set.seed(8577)
19   ## Create a cluster object and then register:
20   cl ← makePSOCKcluster(6)
21   doParallel::registerDoParallel(cl)
22
23   ranger_tune ←
24     tune_grid(ranger_workflow,
25               resamples = ikea_folds,
26               grid = 11,
27               control = control_grid(parallel_over = "resamples", verbose = TRUE
28                                     )
29             )

```

```

1   show_best(ranger_tune, metric = "rmse")

```

```

1   # A tibble: 5 × 8
2     mtry min_n .metric .estimator   mean     n std_err .config
3   <int> <int> <chr>   <chr>     <dbl> <int>   <dbl> <chr>
4 1     2     4 rmse    standard  0.323    10  0.00602 Preprocessor1_Model10
5 2     5     6 rmse    standard  0.331    10  0.00562 Preprocessor1_Model06
6 3     4    10 rmse    standard  0.332    10  0.00570 Preprocessor1_Model05
7 4     3    18 rmse    standard  0.339    10  0.00569 Preprocessor1_Model01
8 5     2    21 rmse    standard  0.343    10  0.00561 Preprocessor1_Model08

```

```

1   show_best(ranger_tune, metric = "rsq")

```

```

1   # A tibble: 5 × 8
2     mtry min_n .metric .estimator   mean     n std_err .config
3   <int> <int> <chr>   <chr>     <dbl> <int>   <dbl> <chr>
4 1     2     4 rsq     standard  0.752    10  0.0106 Preprocessor1_Model10
5 2     5     6 rsq     standard  0.740    10  0.0100 Preprocessor1_Model06
6 3     4    10 rsq     standard  0.738    10  0.0101 Preprocessor1_Model05
7 4     3    18 rsq     standard  0.728    10  0.0104 Preprocessor1_Model01
8 5     2    21 rsq     standard  0.723    10  0.0107 Preprocessor1_Model08

```

```

1   final_rf ← ranger_workflow ▷
2     finalize_workflow(select_best(ranger_tune))
3
4   final_rf

```

```

1 Warning message:
2 No value of 'metric' was given; metric 'rmse' will be used.
3 == Workflow =====
4 Preprocessor: Recipe
5 Model: rand_forest()
6
7 -- Preprocessor -----
8 3 Recipe Steps
9
10 - step_other()
11 - step_clean_levels()
12 - step_impute_knn()
13
14 -- Model -----
15 Random Forest Model Specification (regression)
16
17 Main Arguments:
18   mtry = 2
19   trees = 1000
20   min_n = 4
21
22 Computational engine: ranger

```

```

1 ikea_fit ← last_fit(final_rf, ikea_split)
2 ikea_fit

```

```

1 # Resampling results
2 # Manual resampling
3 # A tibble: 1 × 6
4   splits          id      .metrics .notes   .predictions .workflow
5   <list>         <chr>    <list>  <list>  <list>       <list>
6 1 <split [2770/924]> train/test split <tibble> <tibble> <tibble>   <workflow>
  >

```

```

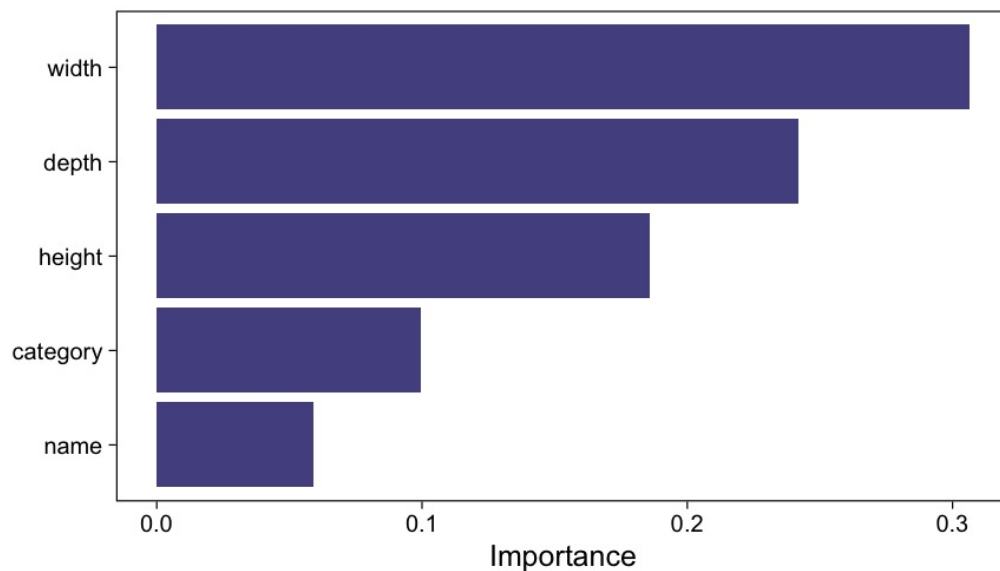
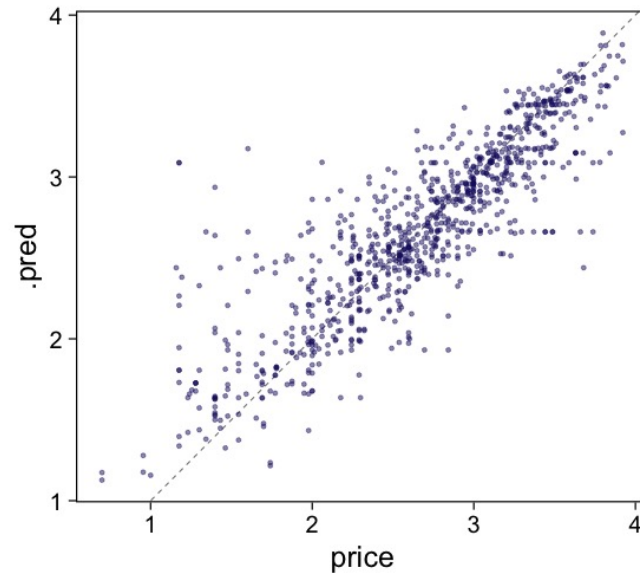
1 collect_metrics(ikea_fit)

```

```

1 # A tibble: 2 × 4
2   .metric .estimator .estimate .config
3   <chr>   <chr>       <dbl> <chr>
4 1 rmse    standard      0.318 Preprocessor1_Model11
5 2 rsq     standard      0.752 Preprocessor1_Model11

```



6.1. Post-procesando el bosque

```

1 ranger_prep <- prep(ranger_recipe, training = ikea_train)
2 rf_model <- randomForest::randomForest(
3   price ~., bake(ranger_prep, ikea_train),
4   mtry = 2, ntree = 1000, nodesize = 4)

1 collect_forest_predictions <- function(model, data){
2   predictions <- predict(model, bake(ranger_prep, data), predict.all = TRUE)
3   predictions$individual >
4     as_tibble() >
5     mutate(observation = 1:nrow(data),
6            truth = data$price) >
7     pivot_longer(cols = 1:1000,
8                  values_to = ".prediction", names_to = ".tree") >
9     group_by(observation) >
10    mutate(.estimate = cummean(.prediction)) >

```

```

11 ungroup() > select(c(-.prediction)) >
12 nest(data = c(observation, truth, .estimate)) >
13 mutate(results = map(data, function(x) { x > rmse(truth, .estimate) }))
14 }

```

```

1 predictions_train <- collect_forest_predictions(rf_model, ikea_train)
2 predictions_test  <- collect_forest_predictions(rf_model, ikea_test)

```

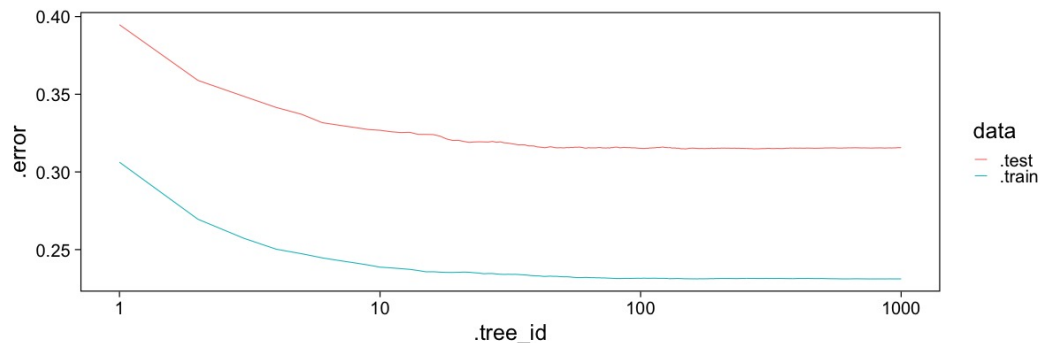


FIGURA 6. Error predictivo utilizando bosque aleatorio.

```

1 lasso_spec <- linear_reg(penalty = tune(), mixture = 1) >
2   set_engine("glmnet") >
3   set_mode("regression")

```

```

1 lasso_rec <- recipe(price ~ ., data = trees_train)

```

```

1 set.seed(108727)
2 forest_boot <- vfold_cv(trees_train, v = 10)

```

```

1 lasso_wf <- workflow() >
2   add_recipe(lasso_rec) >
3   add_model(lasso_spec)
4
5 lasso_grid <- lasso_wf >
6   tune_grid(
7     resamples = forest_boot,
8     grid = 50,
9     control = control_grid(verbose = FALSE)
10  )

```

```

1 lasso_grid >
2   collect_metrics()

```



```

1 # A tibble: 100 × 7
2   penalty .metric .estimator mean      n std_err .config
3   <dbl> <chr> <chr> <dbl> <int> <dbl> <chr>
4 1 1.26e-10 rmse standard 0.307    10 0.0132 Preprocessor1_Model01
5 2 1.26e-10 rsq standard 0.770    10 0.00920 Preprocessor1_Model01
6 3 1.90e-10 rmse standard 0.307    10 0.0132 Preprocessor1_Model02
7 4 1.90e-10 rsq standard 0.770    10 0.00920 Preprocessor1_Model02
8 5 3.28e-10 rmse standard 0.307    10 0.0132 Preprocessor1_Model03
9 6 3.28e-10 rsq standard 0.770    10 0.00920 Preprocessor1_Model03
10 7 4.34e-10 rmse standard 0.307    10 0.0132 Preprocessor1_Model04
11 8 4.34e-10 rsq standard 0.770    10 0.00920 Preprocessor1_Model04
12 9 6.71e-10 rmse standard 0.307    10 0.0132 Preprocessor1_Model05
13 10 6.71e-10 rsq standard 0.770    10 0.00920 Preprocessor1_Model05
14 # ... with 90 more rows
15 # Use 'print(n = ...)' to see more rows

```

```

1 lowest_rmse <- lasso_grid >
2   select_best("rmse")
3
4 final_lasso <- finalize_workflow(
5   lasso_wf,
6   lowest_rmse
7 )

```

```

1 active_trees <- final_lasso >
2   fit(trees_train) >
3   broom::tidy() >
4   filter(estimate != 0) >
5   mutate(.tree = term, beta = estimate) >
6   select(c(.tree, beta))
7
8 active_trees > print(n = 5)

```

```

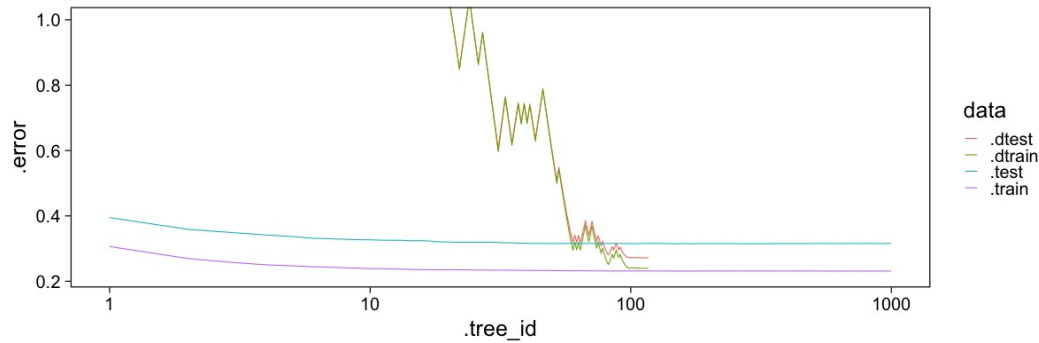
1 # A tibble: 118 × 2
2   .tree      beta
3   <chr>    <dbl>
4 1 (Intercept) -0.0396
5 2 V17          0.0195
6 3 V18          0.0463
7 4 V30          0.0121
8 5 V34          0.0176
9 # ... with 113 more rows
10 # Use 'print(n = ...)' to see more rows

```

```

1 dpredictions_train <- collect_dforest_predictions(rf_model, ikea_train, active_
  trees)
2 dpredictions_test <- collect_dforest_predictions(rf_model, ikea_test, active_
  trees)

```



7. CONCLUSIONES

- Los bosques aleatorios son uno de los métodos más generales de predicción.
- Son fáciles de entrenar, usualmente ajustando dos parámetros por validación cruzada.
- Heredan ventajas de los árboles. Por ejemplo, las predicciones siempre se encuentran en el rango de las observaciones.
- Pueden ser lentos en predicción.
- Tienen capacidad de extrapolación limitada.

REFERENCIAS

- [1] B. M. Greenwell. *Tree-Based Methods for Statistical Learning in R*. Chapman and Hall/CRC, Boca Raton, first edition, may 2022. ISBN 978-1-00-308903-2. . [1](#), [5](#)
- [2] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning*. Springer Series in Statistics. Springer New York, New York, NY, 2009. ISBN 978-0-387-84857-0 978-0-387-84858-7. . [1](#)
- [3] G. James, D. Witten, T. Hastie, and R. Tibshirani. *An Introduction to Statistical Learning: With Applications in R*. Springer Texts in Statistics. Springer US, New York, NY, 2021. [1](#)