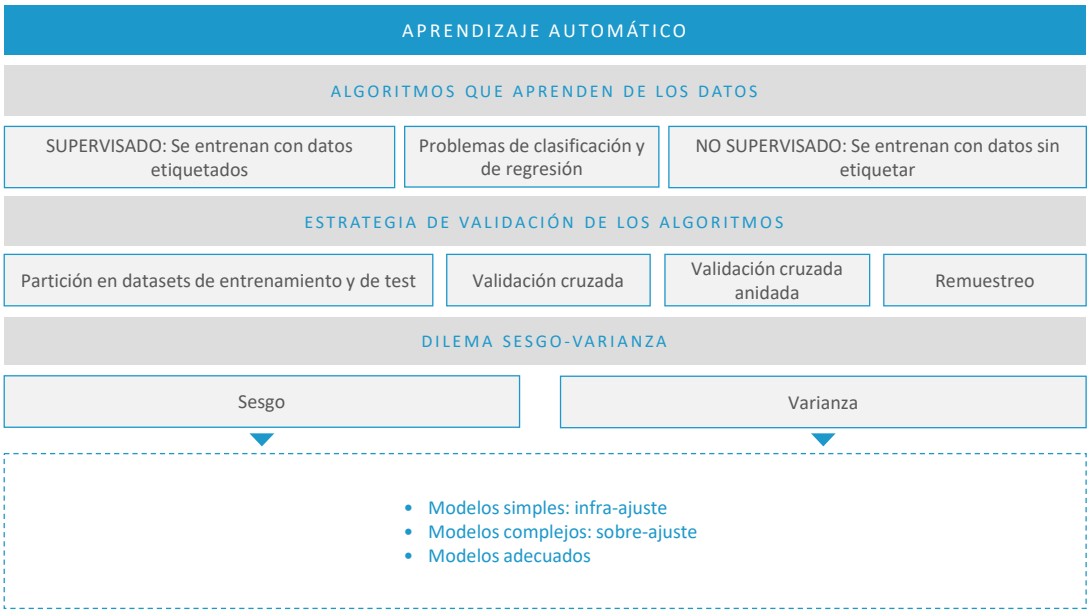
Técnicas Multivariantes

Introducción al aprendizaje automático

Índice

Esquema



Ideas clave

3.1 Introducción y objetivos

En este tema aprenderás qué es el aprendizaje automático (o machine learning), qué ventajas ofrece, cómo se utiliza y los distintos tipos en los que se puede clasificar. Además, se estudiarán las diferentes estrategias a la hora de distribuir los datos para crear el modelo y para comprobar su validez.

3.2 El aprendizaje automático

El aprendizaje automático, también referido como *machine learning*, se puede considerar como la ciencia (y el arte) de programar computadoras de modo que puedan *aprender* a través de los datos (Geron 2019). Aunque en los últimos años ha cogido un gran impulso, su conceptualización no es tan moderna, y ya en 1959 ya se definía el aprendizaje automático como: "el campo de estudio que dota a las computadoras de la habilidad de aprender sin ser programadas explíciitamente" (Samuel 1959).

Uno de los grandes hitos en el aprendizaje automático, fue la consecución de un filtro anti-spam para correos electrónicos mediante aprendizaje automático. Esta aplicación consigue entrenar a un programa mediante aprendizaje automático a través de ejemplos múltiples correos electrónicos marcados como *spam* por distintos usuarios y de ejemplos de correos que no son *spam* (también llamados *ham*).

La diferencia frente a la programación tradicional (Figura 1), es que en el aprendizaje automático no se programan unas características en concreto, sino que el programa

aprende mediante los ejemplos, es decir, el conjunto de datos de entrenamiento (Figura 2). Si se quisiera intentar programar de manera tradicional los posibles patrones que aparecen en un correo *spam*, el código necesario sería muy complejo (por ejemplo, si se detecta que en muchos correos que son *spam* se emplea la palabra "gratis" o "asombroso" se podría añadir esta regla, pero al poco tiempo los creadores de *spam* cambiarían "gratis" por "sin coste" y habría que añadir una nueva regla, etc.).

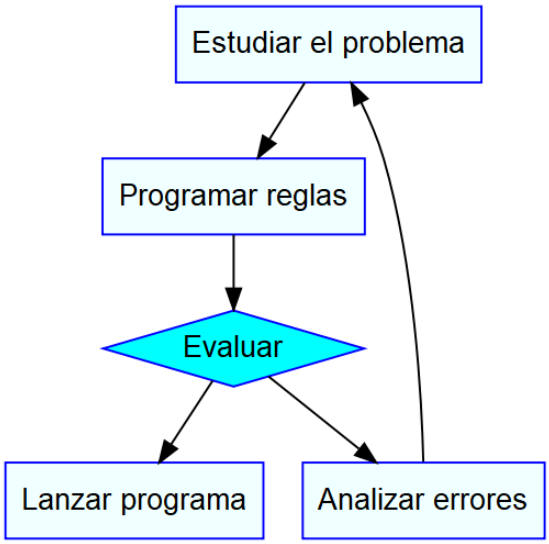


Figura 1: El enfoque tradicional

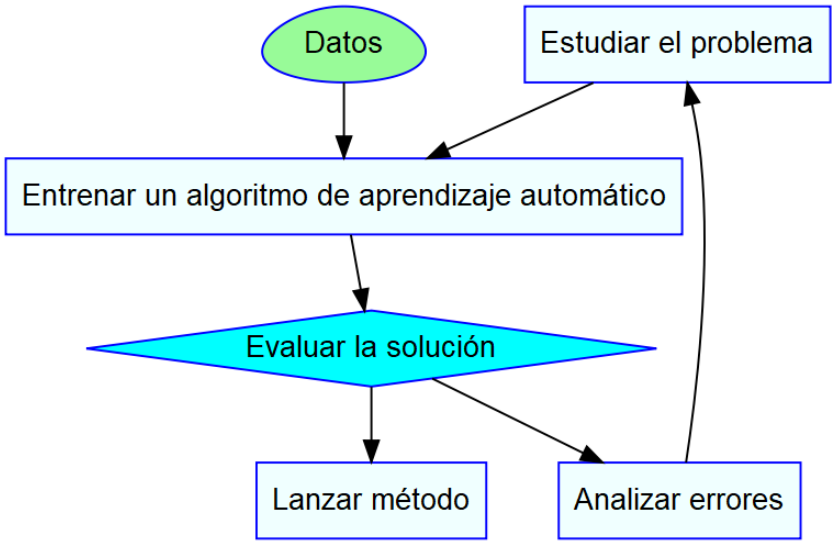


Figura 2: El enfoque del aprendizaje automático

Otro de los problemas en los que es mucho más eficiente abordarlo desde la perspectiva del aprendizaje automático es el reconocimiento de imágenes. Uno de los ejemplos más característicos es la clasificación entre gatos y perros. Si se quiere realizar un programa que cuando se cargue una imagen de un gato o de un perro detecte qué clase de animal es, sería muy complicado realizarlo mediante la definición de patrones. Se podría intentar programar: "si tiene orejas puntiagudas es un gato". Pero ante la aparición de un perro con orejas puntiagudas el programa clasificaría incorrectamente. En este caso, la solución también se consigue mediante aprendizaje automático.

En resumen, el aprendizaje automático sirve para:

- Tratar con problemas cuya resoluición requiera de un ajuste muy fino o de largas listas de reglas y excepciones.
- Resolver problemas complejos que no tienen solución desde un enfoque tradicional.
- Entornos cambiantes: el aprendizaje automático se puede adaptar a nuevos datos.
- Proporcionar hallazgos en grandes volumenes de datos: el aprendizaje automático es una buena pareja de baile del *"big data"*.

Ejemplos de aplicaciones

Algunas de las aplicaciones concretas en las cuales se pueden aplicar las técnicas de aprendizaje automático y que vamos a desarrollar en la asignatura son:

- **Estimación de la demanda de un producto**, mediante técnicas de regresión, como las que se abordarán en los Temas 4 y 5 de la asignatura, o mediante modelos de ensamble (Tema 8).
- **Detección de fraude en tarjetas de crédito** mediante técnicas de clasificación (Tema

8).

- **Representación de conjuntos de datos multidimensionales complejos** mediante técnicas de reducción de la dimensión (Tema 9).

- **Segmentar clientes según sus patrones de compra** mediante técnicas de clustering (Tema 9).

Y algunas otras, que aunque queden fuera del alcance de la asignatura, también se resuelven mediante técnicas de aprendizaje automático:

- **Analizar imágenes de productos en una línea de producción y clasificarlas automáticamente** mediante redes neuronales convolucionales (CNNs por sus siglas en inglés).

- **Detección de tumores en un escáner cerebral**, también mediante CNNs.

- **Clasificación automática de nuevos artículos** mediante procesamiento del lenguaje natural (NLP, por sus siglas en inglés) y más específicamente, utilizando clasificación de textos empleando redes neuronales recurrentes (RNNs por sus siglas en inglés) o CNNs.

- **Detección de lenguaje ofensivo en blogs o redes sociales**, también mediante NLP.

- **Resumen de documentos**, también mediante NLP.

- **Crear un chatbot**, también mediante NLP.

Tipos de aprendizaje automático

Supervisado y no supervisado

Existen distintos tipos de aprendizaje automático. Por un lado tenemos el **aprendizaje supervisado**, en el cual se *etiquetan* los datos y se buscan relaciones concretas entre ellos. En cambio, en el **aprendizaje no supervisado**, el sistema trata de aprender con datos sin etiquetar y extraer conclusiones. Para que quede más claro, se van a emplear dos ejemplos de clasificación:

- En un primer caso, supongamos que tenemos las características socioeconómicas de tres tipos de clientes según el tipo de servicio que ha contratado en nuestra empresa: clientes A, clientes B y clientes C. En este caso tenemos cada uno de nuestros clientes perfectamente etiquetado, y el objetivo de realizar un modelo de aprendizaje supervisado consiste en que ante las características de un nuevo cliente potencial podemos identificar a que tipo de servicio esté más interesado.

- Un segundo problema, sería tener que clasificar a todos nuestros clientes, los cuales no tienen a priori ninguna etiqueta, y buscar grupos similares entre ellos. En este caso estaríamos ante un problema de aprendizaje automático no supervisado.

Entre las técnicas más empleadas para el análisis supervisado se encuentran:

- Técnicas de regresión.

- Técnicas de clasificación.

- Modelos de ensamblado.

En cambio, en los problemas de análisis no supervisado se emplean:

- Técnicas de reducción de la dimensión.

- Clustering.

Basado en casos o basado en un modelo

Otra manera de clasificar los métodos de aprendizaje automático es atendiendo cómo *generalizan*. La mayoría de los métodos de aprendizaje automático están pensados para realizar predicciones. Esto significa que dado un número de ejemplos que entrenen el método, éste será capaz de hacer buenas predicciones (para *generalizar*) ejemplos que no haya visto antes. Ajustar bien los ejemplos de entrenamiento es importante, pero insuficiente. El verdadero objetivo es predecir bien ante nuevos datos. Se pueden clasificar los distintos métodos de aprendizaje automático entre aquellos que están basados en casos o los que están basados en un modelo.

- **Basados en casos**: Los métodos basados en casos se suelen encontrar en problemas de clasificación. Se basan en definir el tipo de un nuevo ejemplo asignándole el valor del caso (o casos) más cercano (o cercanos). Un ejemplo de método basado en casos es el del *vecino más cercano*, que se tratará en el Tema 7 de la asignatura.

- **Basados en un modelo**: Otra manera de generalizar partiendo de los ejemplos de entrenamiento es construir un modelo y posteriormente emplear dicho modelo para hacer las predicciones. Un ejemplo de método basado en modelo es un modelo de regresión lineal múltiple, que se detallará en el Tema 5 de la asignatura.

3.3 Ajuste y predicción del modelo

En el apartado anterior se ha hecho referencia al ajuste y a la predicción de un método de aprendizaje automático. Cuando se aplica un método, es necesario realizar esta

distinción. Es decir, cómo el modelo es capaz de ajustar los datos con los cuales se está creando el modelo (**ajuste**) y de cómo es capaz de dar una **predicción** sobre unos datos los cuales no se han empleado para ajustar el modelo. Para evaluar la capacidad predictiva de un modelo, por tanto, es necesario no utilizar todos los datos disponibles en la creación del mismo, sino que hay que *guardar* una proporción de los datos.

3.4 El dilema sesgo-varianza

El sesgo

El sesgo estadístico consiste en una distorsión de los resultados en los que el valor calculado difiere del real. El sesgo se puede encontrar tanto en el método como en los datos. El sesgo en los datos lo podemos encontrar si la muestra obtenida de los datos no es representativa de la población: por muy bueno que sea nuestro modelo no se podrán extraer conclusiones correctas. Por ejemplo, estaríamos ante un problema de sesgo en el muestreo si para estimar la altura de la población general observásemos la altura de las personas que salen de entrenar de un pabellón de baloncesto. Por este motivo, es muy importante cerciorarse de cómo se han obtenido los datos, si se ha realizado algún tratamiento o preprocesado sobre ellos, etc.

El sesgo en el método puede aparecer por el método en sí que se aplique o por la manera en la que se obtienen los parámetros (o hiperparámetros) del mismo. En algunas ocasiones, ese sesgo será inevitable, pero tendremos que tratar de reducirlo. Cabe destacar que un hiperparámetro es un valor de la configuración del algoritmo de aprendizaje que no se ajusta con los datos y, por tanto, no se puede obtener su valor óptimo con los métodos de validación tradicionales. Sin embargo, el analista puede fijar, *a priori*, basándose en su experiencia y conocimientos, el valor de los hiperparámetros para un determinado problema, o bien puede emplear técnicas de validación más avanzadas, como la validación cruzada anidada, que veremos más adelante.

La varianza

En el apartado anterior se ha comentado qué es el sesgo, pero existe otro aspecto a tener en cuenta, que es la varianza del modelo. La varianza del modelo es la variabilidad en los parámetros del modelo ante un conjunto de datos distinto (otra muestra distinta) de la misma población. En muchas ocasiones, para disminuir esta varianza del modelo, es necesario introducir un sesgo en el mismo. Es lo que se conoce como el dilema o la solución de compromiso sesgo-varianza o *bias-variance tradeoff*.

La Solución de compromiso

En esta solución de compromiso es importante tener en cuenta que si el modelo con el cual se va a ajustar los datos es demasiado sencillo, se puede producir un infra-ajuste del modelo y obtener un modelo con un sesgo alto. Por el contrario, si acudimos a modelos muy complejos, es posible estar incurriendo en un sobre-ajuste de los datos y, por lo tanto, se obtendrá un modelo con una gran varianza. En la Figura 3 se puede observar este fenómeno. En el primer panel, se han ajustado los datos (los puntos rojos) mediante un modelo de regresión lineal simple:

$$Y = \beta_0 + \beta_1 \cdot X,$$

que puede presentar un sesgo elevado debido al infraajuste.

En el segundo panel, se han ajustado los datos mediante un modelo de regresión polinómico de grado 2, es decir:

$$Y = \beta_0 + \beta_1 \cdot X + \beta_2 \cdot X^2.$$

En este caso, el modelo parece que tiene un ajuste adecuado.

Por último, en el tercer panel se han ajustado los datos mediante un modelo de regresión polinómico de grado 6, con lo cual, se está obteniendo un sobreajuste del modelo y se tendrá una gran varianza de los parámetros de ajuste (β_i con $i = 0, ..., 6$) frente a nuevos datos.

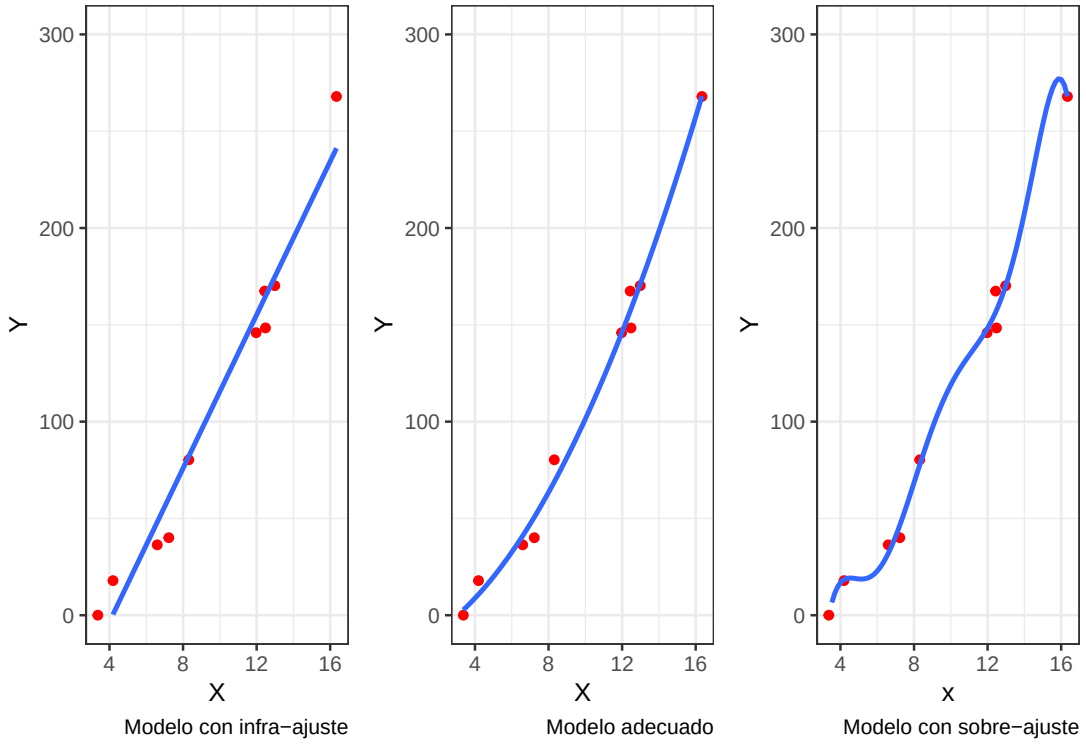


Figura 3: Diferentes sesgo-varianza en los modelos

3.5 Técnicas de validación

Medida de la bondad del ajuste

Para evaluar la capacidad de un modelo de ajustar un conjunto de datos o de predecir el valor ante un dato no visto es necesario definir una medida de la bondad del ajuste. Según el tipo de problema (regresión o clasificación) y según el tipo de algoritmo empleado se emplearán unas u otras medidas. En los problemas de regresión, una medida habitual para evaluar la bondad del ajuste es la raíz del error cuadrático medio:

$$rmse = \sqrt{\frac{\sum_i^n (\hat{y}_i - y_i)^2}{n}}.$$

Es importante hacer notar que en los próximos apartados, para definir las distintas técnicas de validación se empleará, en muchos casos, por simplicidad, el término error en vez del término más preciso de medida de bondad del ajuste.

Validación I: Entrenamiento y test

Para evitar los posibles problemas de sobreajuste, es necesario evaluar la bondad del modelo sobre unos datos que no se hayan empleado para la realización del mismo. Existen numerosos métodos para realizar las particiones entre el conjunto de datos de entrenamiento y el conjunto de datos de validación y/o test. El método más sencillo para construir esta partición consiste en dejar una proporción de los datos para obtener los parámetros del modelo, y luego comprobar con la proporción de datos restante cómo de bueno es ese modelo. En este caso, el conjunto de datos (o *dataset*) se dividirá entre el *dataset* de entrenamiento y el *dataset* de test. En la Figura 4 se muestra una partición donde el 0.7 del conjunto de datos se emplea para entrenamiento y el 0.3 restante para test.

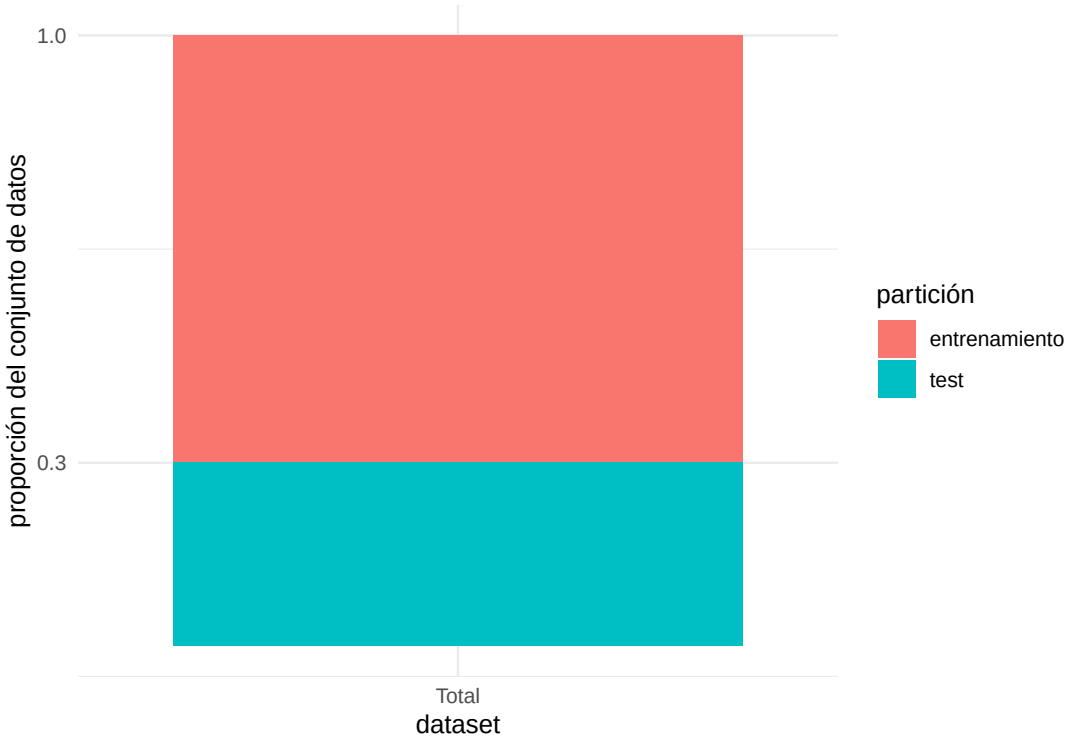


Figura 4: Partición entrenamiento-test básica

En este caso se obtiene el método de aprendizaje automático mediante el dataset de entrenamiento y se valida mediante el dataset de test. En este caso, por tanto, se pueden obtener el error del dataset de entrenamiento y el dataset de test. En la mayoría de los casos, este error será menor en el dataset de entrenamiento (error en el ajuste) que en el dataset de test (estimación del error de generalización), ya que no se han empleado los datos del dataset del test para la construcción del modelo. Sin embargo, si la diferencia entre estos errores es muy grande, es un indicador de que esta produciéndose sobreajuste en el método y puede ser necesario modificar alguno de los hiperparámetros del mismo o, incluso, cambiar de método.

La ventaja de realizar este método de validación es su sencillez, mientras que el principal inconveniente es que se evalua un modelo con una potencia predictiva restringida, ya que los datos del dataset de test no aportan información a la realización del método de aprendizaje automático.

Validación II: Validación cruzada

Para poder incluir información de todos los datos del conjunto de la muestra en la evaluación del modelo es posible realizar la estimación del modelo mediante validación cruzada (CV por sus siglas en inglés). La CV consiste en dividir la muestra en k particiones, entrenar al modelo usando $k-1$ de las k particiones y después medir la bondad de la predicción en la partición restante. Se realiza este proceso para las k posibles combinaciones de $k-1$ particiones que existen y se calculan la media de las bondades estimadas. En la Figura 5 se muestra un ejemplo de CV utilizando 5 particiones. En la primera comprobación se usará la primera partición para el test y las 4 restantes para el entrenamiento. En la segunda comprobación se usará la segunda partición para el test, y así sucesivamente hasta la quinta comprobación, donde será la quinta partición, la empleada para el test.

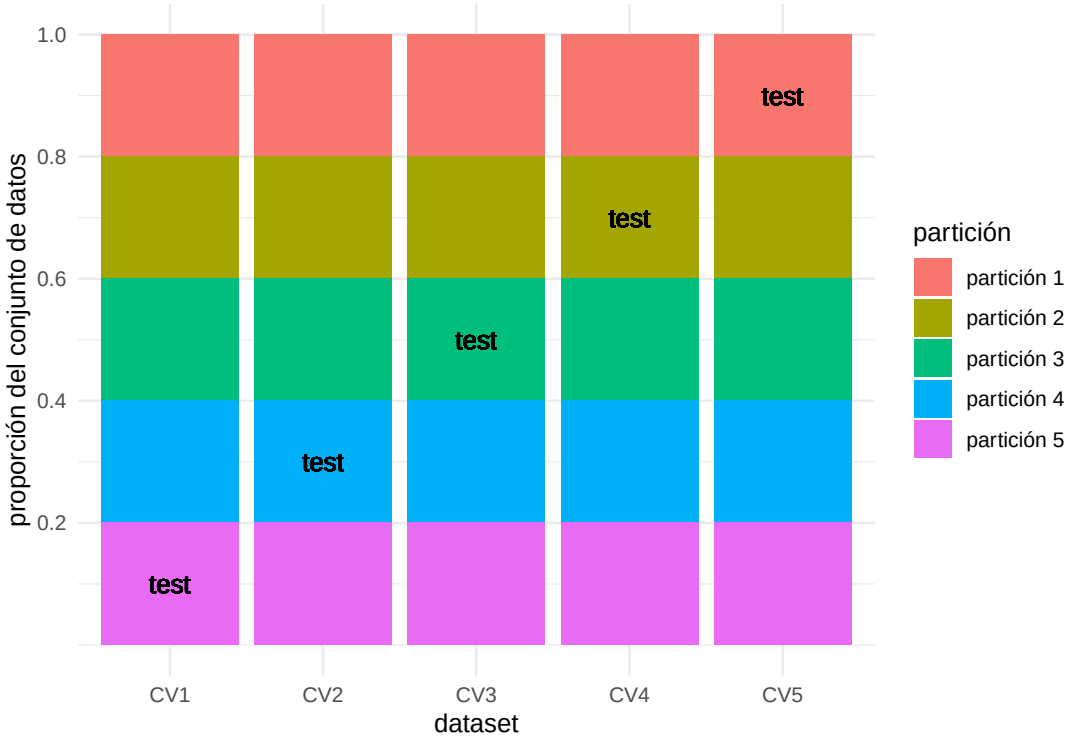


Figura 5: Validación cruzada con 5 particiones

Con la CV se consigue incluir toda la información del conjunto de datos a la hora de obtener una estimación del error de generalización del algoritmo de aprendizaje au-

tomático a través de la media de los errores de los dataset de test para cada una de las comprobaciones. Además, se puede obtener una estimación del error de entrenamiento realizando la media de los errores del dataset de entrenamiento para cada una de las comprobaciones. En este caso, es necesario comprobar que los errores en el dataset de test no son mucho mayores a los errores en el dataset de entrenamiento, lo cual nos podría estar indicando un sobreajuste del modelo.

En algunas ocasiones, la CV puede verse influenciada por la manera en la que se han realizado las particiones, influyendo en los resultados del entrenamiento o del test para algunas de las comprobaciones. Para minimizar este efecto, una buena práctica es repetir el cálculo de la CV r veces y promediar los resultados obtenidos. Así, se consiguen evitar posibles errores en el ajuste debidos a una partición del conjunto de datos en concreto.

Un caso especial de validación cruzada es cuando se realizan tantas particiones como número de datos se tienen. Este método se conoce como LOOCV (de las siglas en inglés *Leave One Out Cross Validation*) y consiste en entrenar el modelo cada vez con todos los datos menos uno y comprobar su validez con el dato restante. Este método tiene como ventajas:

- Al realizar una partición para cada dato, desaparece la incertidumbre a la hora de distribuir los datos entre las distintas particiones y no es necesario realizar múltiples repeticiones de la validación cruzada.

- Podemos identificar medidas influyentes: si un resultado de la validación es muy diferente al resto, es posible que el dato correspondiente a esa partición, sea una medida influyente.

Sin embargo, tiene los inconvenientes:

- Puede ser computacionalmente costoso si el número de datos es elevado.

- Puede presentar problemas de sobreajuste, ya que está empleando casi todos los datos ($n-1$) para el entrenamiento del modelo.

Validación III: Validación cruzada anidada

La validación cruzada anidada (NCV por sus siglas en inglés) se utiliza cuando además de querer obtener una estimación del error de generalización es necesario ajustar algún hiperparámetro del algoritmo.

La NCV permite estimar el error de generalización una vez se ha fijado el valor óptimo de los hiperparámetros con un conjunto de datos que no se han empleado para la busqueda de estos hiperparámetros. Si se escogieran los hiperparámetros a la vez que se calcula el error, se estaría dando lugar a un resultado del error de generalización demasiado optimista. Así pues, el realizar una selección de hiperparámetros con validación cruzada no anidada utilizando los mismos datos para ajustar los hiperparámetros y para evaluar el error de generalización del modelo produce un sobreajuste del modelo. (Cawley and Talbot 2010).

Para evitar este problema, la NCV utiliza 2 bucles a la hora de realizar la validación. En el bucle interior, se obtiene los valores óptimos de los hiperparámetros empleando de manera efectiva la CV. En el bucle exterior, se realiza la estimación del error de generalización. En la Figura 6 se muestra un ejemplo de validación cruzada anidada con 5 particiones para el bucle externo y 4 particiones para el interno. Esto quiere decir que se emplea el 4/5 de los datos para la obtención de los valores óptimos de los hiperparámetros y 1/5 para la validación de los datos. Dentro del bucle interior, se divide el conjunto de datos en 3/4 para el entrenamiento del modelo y 1/4 para la validación (y selección de los hiperparámetros).

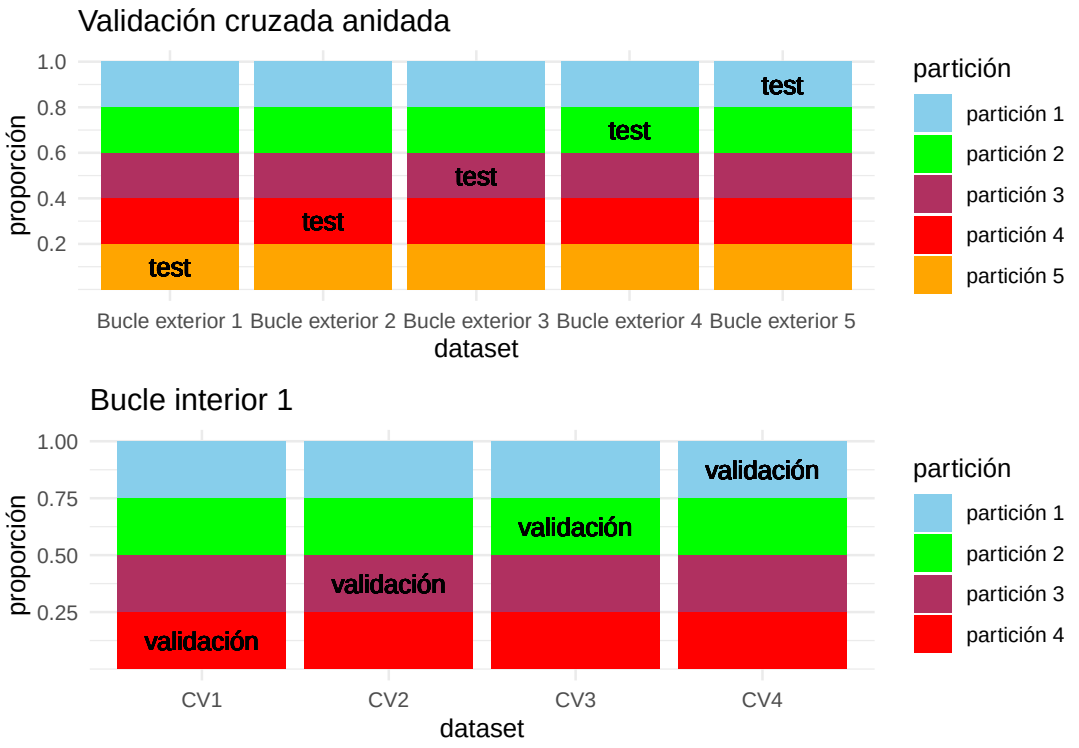


Figura 6: Validación cruzada anidada con 5 particiones

La manera de proceder en este caso es realizar el bucle interior 1 para seleccionar el valor de los hiperparámetros óptimos. Con estos valores se entrena un modelo con las 4 particiones del bucle interior 1. Por último, se estima el error de test en la partición que no se ha utilizado para la obtención de los hiperparámetros del bucle exterior (la partición 5). Este proceso se repetirá para cada uno de los bucles interiores.

3.6 Técnicas de remuestreo

En algunas ocasiones, cuando el tamaño de muestra para realizar el modelo o para aplicar el algoritmo de aprendizaje automático no es muy grande, es necesario acudir a otras técnicas de validación en las cuales no se realice una partición de los datos. En estos casos, se puede obtener una estimación del error de generalización apoyándose en métodos que requieren de la utilización de técnicas de remuestreo. Estas técnicas de remuestreo consisten en obtener una muestra similar a la original de nuestros datos

de manera artificial. Los métodos de remuestreo más empleados son el *jackknife* y el *bootstrap*.

- *Jackknife*: El Jacknife consiste en obtener las n-muestras derivadas de la muestra original a partir de eliminar una observación.

- *Bootstrap*: El *Bootstraping* es la técnica de remuestreo en el cual se obtiene una muestra del mismo tamaño que la original haciendo muestreo con reemplazamiento. Éste método es uno de los más utilizados en los modelos de ensamble, tal y como veremos en el Tema 8.

Cálculo del optimismo

Otra manera de obtener una estimación de la bondad del ajuste (o del error de generalización) consiste en:

- Entrenar el modelo o algoritmo con todos los datos disponibles.
- Obtener el error del entrenamiento.
- Corregir este error mediante el *optimismo*.

El *optimismo* se puede definir como la diferencia entre la medida de la bondad del ajuste en el dataset de entrenamiento y la medida de la bondad del ajuste en la generalización. Así pues, el cálculo del optimismo (Harrell Jr, Lee, and Mark 1996) se presenta como una técnica muy útil que permite corregir la bondad del algoritmo, sin perder muestra.

El cálculo del optimismo, para un modelo dado, consta de los siguientes pasos:

1. Con el modelo ajustado con toda la muestra (la muestra original), se obtiene la medida del bondad del ajuste, (p.e. el rmse).

2. Se muestrea una réplica bootstrap, en la cual se ajusta un nuevo modelo del mismo

estilo que el original. En el modelo bootstrap pueden cambiar el valor de los coeficientes β obtenidos e, incluso, las variables que se seleccionan, respecto al modelo original. Se calcula el rmse sobre esa réplica: $rmse_{boot}$, donde $rmse_{boot}$ puede ser mayor o menor que $rmse_{orig}$, dependiendo de los datos.

3. Con el modelo bootstrap se calcula el rmse sobre la muestra original $rmse_{orig}^{B*}$. $rmse_{orig}^{B*}$ normalmente será mayor que $rmse_{boot}$, ya que parte de las observaciones con las cuales se está calculando el rmse no han sido utilizadas para el entrenamiento del modelo.

4. Se repiten los pasos 2-3 B-veces. El optimismo se calcula como se muestra en la siguiente expresión:

$$Opt = \frac{1}{B}\sum_{i=1}^{B}\left(rmse_{orig}^{B*} - rmse_{boot}\right).$$

Una vez que se ha calculado el optimismo, se define el rmse corregido $rmse_{corregido}$ como: $rmse_{corregido} = rmse_{orig} + Opt$.

Material audiovisual

Accede al vídeo: Optimismo

3.7 Referencias bibliográficas

Cawley, G. C. and Talbot, N. L. C. (2010). On over-fitting in model selection and subsequent selection bias in performance evaluation. 11:2079–2107.

Geron, A. (2019). *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems.* 2 edition.

Harrell Jr, F. E., Lee, K. L., and Mark, D. B. (1996). Multivariable prognostic models: issues in developing models, evaluating assumptions and adequacy, and measuring and reducing errors. *Statistics in Medicine*, 15(4):361–387.

Samuel, A. L. (1959). Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, 3(3):210–229.

3.8 Ejercicios resueltos

Ejercicio 1.

Con este ejercicio practicarás con el cargado de los datos a través de una fuente externa (una url) y practicarás algunas de las funciones básicas del paquete *pandas*.

a) Crea un script que cargue la última versión del dataset **California housing** que se encuentra en:

https://github.com/ageron/handson-ml2/blob/master/datasets/housing/housing.csv

b) Muestra las 7 primeras observaciones de la base descargada.

c) Describe la base de datos (número y tipo de las variables que la conforman y si existen datos faltantes en algunas de las variables).

Solución

a) El primer paso a realizar cuando trabajamos en Python es cargar los paquetes y funciones que se van a utilizar (recordar que deben haberse instalado previamente). Una vez cargadas las librerías, se crea un script que carga la última versión del dataset **California housing** accediendo al repositorio de github donde se encuentra el contenido. Primero, se define una función que descargue los datos y los descomprima en csv (están comprimidos en .tgz). Después, se define otra función que cargue estos datos en un dataframe.

```
# librerias que se van a necesitar------------------------------
import os
import tarfile
import urllib.request
import pandas as pd

# definir las rutas y caminos donde se encuentran los datos------
DOWNLOAD_ROOT = "https://raw.githubusercontent.com/ageron/
handson-ml2/master/"
# path que se va a crear en nuestro sistema---------------------
HOUSING_PATH = os.path.join("datasets", "housing")
# lugar de descarga del dataset---------------------------------
HOUSING_URL = DOWNLOAD_ROOT + "datasets/housing/housing.tgz"

# definir una funcion que obtenga los datos y los descargue-----
def fetch_housing_data(housing_url=HOUSING_URL,
housing_path=HOUSING_PATH):
    if not os.path.isdir(housing_path):
        os.makedirs(housing_path)
```

```
    tgz_path = os.path.join(housing_path, "housing.tgz")
    urllib.request.urlretrieve(housing_url, tgz_path)
    housing_tgz = tarfile.open(tgz_path)
    housing_tgz.extractall(path=housing_path)
    housing_tgz.close()

fetch_housing_data()

# definir una funcion que cargue el csv en un dataframe----------
def load_housing_data(housing_path=HOUSING_PATH):
    csv_path = os.path.join(housing_path, "housing.csv")
    return pd.read_csv(csv_path)


housing = load_housing_data()
```

b) con el método *.head(7)* se muestran las primeras 7 observaciones. Es interesante hacer notar, que por defecto, *.head()* muestra 5 observaciones.

```
# se muestran las primeras 7 observaciones-----------------------
print(housing.head(7))
# el metodo por defecto muestra 5 observaciones------------------
```

```
##    longitude  latitude  ...  median_house_value  ocean_proximity
## 0    -122.23     37.88  ...            452600.0         NEAR BAY
## 1    -122.22     37.86  ...            358500.0         NEAR BAY
## 2    -122.24     37.85  ...            352100.0         NEAR BAY
## 3    -122.25     37.85  ...            341300.0         NEAR BAY
## 4    -122.25     37.85  ...            342200.0         NEAR BAY
## 5    -122.25     37.85  ...            269700.0         NEAR BAY
## 6    -122.25     37.84  ...            299200.0         NEAR BAY
##
```

```
## [7 rows x 10 columns]
```

```
print(housing.head())
```

```
##    longitude  latitude  ...  median_house_value  ocean_proximity
## 0    -122.23     37.88  ...            452600.0         NEAR BAY
## 1    -122.22     37.86  ...            358500.0         NEAR BAY
## 2    -122.24     37.85  ...            352100.0         NEAR BAY
## 3    -122.25     37.85  ...            341300.0         NEAR BAY
## 4    -122.25     37.85  ...            342200.0         NEAR BAY
##
## [5 rows x 10 columns]
```

c) Para describir la base de datos se utiliza el método *.info()*, el cual va a indicar el número y tipo de las variables que conforman el dataset, así como el número de observaciones no nulas para cada variable.

```
# descriptivo de la base de datos--------------------------------
print(housing.info())
```

```
## <class 'pandas.core.frame.DataFrame'>
## RangeIndex: 20640 entries, 0 to 20639
## Data columns (total 10 columns):
##  #   Column              Non-Null Count  Dtype
## ---  ------              --------------  -----
##  0   longitude           20640 non-null  float64
##  1   latitude            20640 non-null  float64
##  2   housing_median_age  20640 non-null  float64
##  3   total_rooms         20640 non-null  float64
##  4   total_bedrooms      20433 non-null  float64
##  5   population          20640 non-null  float64
##  6   households          20640 non-null  float64
##  7   median_income       20640 non-null  float64
```

```
##  8   median_house_value  20640 non-null  float64
##  9   ocean_proximity     20640 non-null  object
## dtypes: float64(9), object(1)
## memory usage: 1.6+ MB
## None
```

Del descriptivo se observa que se tienen 10 variables, de las cuales 9 son numéricas y 1 es de tipo objeto. Además, existen 20640 observaciones. También se observa que la variable **total_bed_rooms** únicamente tiene 20433 observaciones no nulas, y por lo tanto, tiene $20640 - 20433 = 207$ datos faltantes.

Ejercicio 2

a) En la base de datos **California housing** existe una variable que no es del tipo numérico. Muestra los diferentes valores que toma dicha variable.

b) Describe las variables numéricas (máximo, mínimo, media, desviación típica, cuartiles, etc.) y realiza un histograma de cada una de ellas.

Solución

a) Para obtener los diferentes valores de una variable se debe seleccionar la variable de interés y se puede emplear el método *.value_counts()*.

```
# descriptivo de la variable ocean_proximity----------------------
print(housing["ocean_proximity"].value_counts())
```

```
## <1H OCEAN     9136
## INLAND        6551
## NEAR OCEAN    2658
```

```
## NEAR BAY      2290
## ISLAND           5
## Name: ocean_proximity, dtype: int64
```

Se observa que existen 5 categorías distintas que puede tomar la variable **ocean_proximity**.

b)

Para obtener un buen descriptivo de las variables numéricas se emplea el método *.describe()*.

```
# descriptivo de la base de datos--------------------------------
print(housing.describe())
```

```
##           longitude      latitude  ...  median_income  median_house_value
## count  20640.000000  20640.000000  ...   20640.000000        20640.000000
## mean    -119.569704     35.631861  ...       3.870671       206855.816909
## std        2.003532      2.135952  ...       1.899822       115395.615874
## min     -124.350000     32.540000  ...       0.499900        14999.000000
## 25%     -121.800000     33.930000  ...       2.563400       119600.000000
## 50%     -118.490000     34.260000  ...       3.534800       179700.000000
## 75%     -118.010000     37.710000  ...       4.743250       264725.000000
## max     -114.310000     41.950000  ...      15.000100       500001.000000
##
## [8 rows x 9 columns]
```

Para dibujar los histogramas de las variables numéricas se puede emplear el método *.hist()*, donde se el número de barras que aparece en el histograma se define con el argumento *bins*. En la Figura 7 se muestran los histogramas de las 9 variables con 50 barras para cada variable.

```
# descriptivo de la base de datos-------------------------------
import matplotlib
import matplotlib.pyplot as plt
import numpy as np
# dividir los histogramas numericos-----------------------------
fig, axes = plt.subplots(nrows = 3, ncols = 3, figsize = (30,30))
# definir nombres ejes------------------------------------------
# primera fila
axes[0,0].set_xlabel("x")
axes[0,0].set_ylabel("n")
axes[0,1].set_xlabel("x")
axes[0,1].set_ylabel("n")
axes[0,2].set_xlabel("años")
axes[0,2].set_ylabel("n")
# segunda fila
axes[1,0].set_xlabel("x")
axes[1,0].set_ylabel("n")
axes[1,1].set_xlabel("x")
axes[1,1].set_ylabel("n")
axes[1,2].set_xlabel("habitantes")
axes[1,2].set_ylabel("n")
# tercera fila
axes[2,0].set_xlabel("x")
axes[2,0].set_ylabel("n")
axes[2,1].set_xlabel("10000 dólares")
axes[2,1].set_ylabel("n")
axes[2,2].set_xlabel("dólares")
axes[2,2].set_ylabel("n")
housing.hist(bins = 50, ax = axes);
plt.xlabel("x");
plt.ylabel("residuos");
plt.show();
```

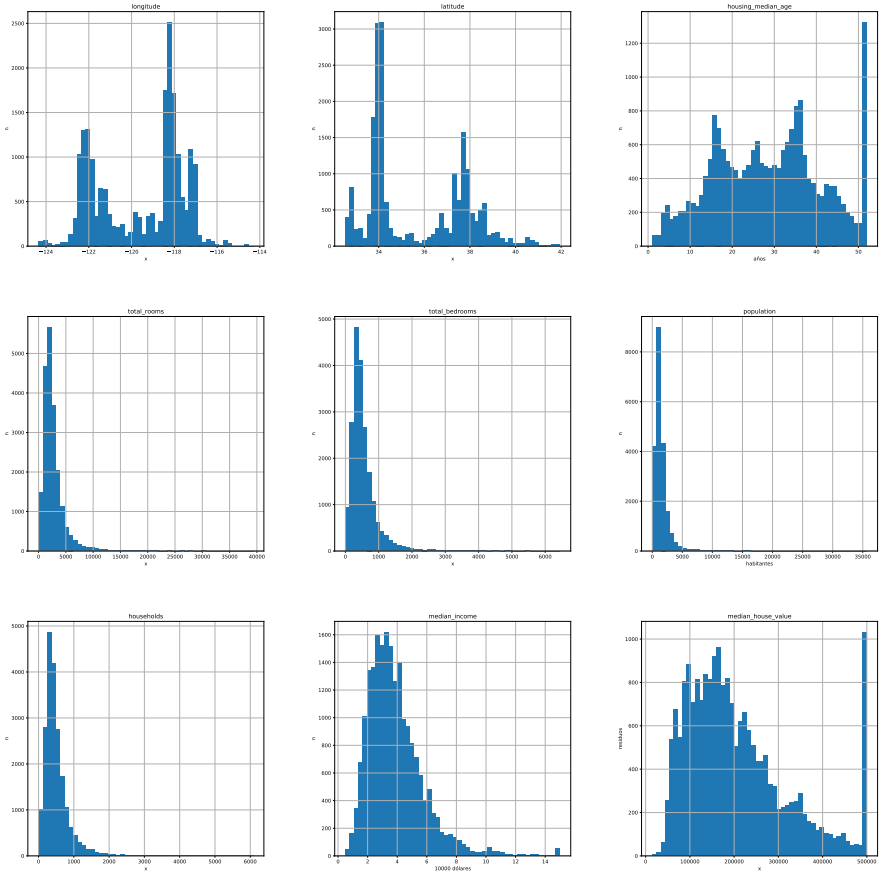


Figura 7: Histogramas de las variables numéricas de California housing.

Del análisis de las variables, podemos observar como la mediana de los ingresos viene escalada en 10000 dólares. Además, tanto la edad, como la mediana de los ingresos, como el precio de las casas están capados en sus valores máximos (parece que se han **winsorizado**).

Ejercicio 3

a) Divide el dataset **California housing** en un dataset de entrenamiento (0.7) y en un dataset de test (0.3).

b) Divide el dataset de **California housing** en 10 particiones listas para poder realizar validación cruzada.

Solución

a) Para poder obtener el error de generalización de un algoritmo de aprendizaje automático es necesario emplear alguna técnica de validación. Una de las más sencillas consiste en dividir el conjunto de datos en dataset de entrenamiento y de test.

Para dividir el conjunto de datos en las particiones de entrenamiento y de test en python se define la función *particiones* con argumentos el conjunto de datos sobre el cual se van a realizar las particiones y la proporción destinada a la partición de test (dataset, test_part). En está función se obtiene el entero más cercano del resultado de multiplicar el número de observaciones del conjunto de datos por la proporción destinada para el test (*test_part_size*). Por otro lado, se barajan los índices del dataset de manera aleatoria usando la función *np.random.permutation*. Por último, se asignan los *test_part_size* índices a *test_indices* y los índices restantes a *train_indices*.

```
# definir semilla para que la particion sea la misma-------------
np.random.seed(3)
# definir funcion particiones-----------------------------------
def particiones(dataset, test_part):
    test_part_size = int(len(dataset) * test_part)
    mezclar_indices = np.random.permutation(len(dataset))
    test_indices = mezclar_indices[:test_part_size]
    train_indices = mezclar_indices[test_part_size:]
    return dataset.iloc[train_indices], dataset.iloc[test_indices]
# usar funcion particiones con test_part 0.3--------------------
train_set, test_set = particiones(housing, 0.3)
# comprobar longitudes de los set de entrenamiento y de test-----
print(round(len(housing) * 0.7, 1))
```

```
## 14448.0
```

```
print(len(train_set))
```

```
## 14448
```

```
print(len(housing) * 0.3)
```

```
## 6192.0
```

```
print(len(test_set))
```

```
## 6192
```

Se comprueba que cuadra el número de observaciones en cada partición con las que deberían ser.

b) Para dividir el conjunto de datos en 10 particiones se procede de manera similar, pero en este caso se va a emplear un bucle con el número de particiones y se va a definir cada partición en la lista part_cv.

```
# definir numero de particiones-----------------------------------
n = 10
# crear lista del 1 al 10-----------------------------------------
n_list = list(range(1,n+1))
# definir longitud de cada particion------------------------------
part_size = int(len(housing) / n)
# barajar indices-------------------------------------------------
mezclar_indices = np.random.permutation(len(housing))
# crear lista vacia-----------------------------------------------
part_cv = {}
# asignar conjunto de indices-------------------------------------
for i in n_list:
  part_i = mezclar_indices[part_size * (i-1):part_size * i]
```

```
  part_cv["part_{0}".format(i)] = housing.iloc[part_i]
# comprobar algunas longitudes de las particiones----------------
print(len(part_cv["part_1"]))
```

```
## 2064
```

```
print(len(part_cv["part_2"]))
```

```
## 2064
```

```
print(len(part_cv["part_10"]))
# comprobar cabeceros-------------------------------------------
```

```
## 2064
```

```
print(part_cv["part_10"].head())
```

```
##        longitude  latitude  ...  median_house_value  ocean_proximity
## 16480    -121.28     38.17  ...            244200.0           INLAND
## 20591    -121.58     39.15  ...             55500.0           INLAND
## 20358    -118.95     34.18  ...            254900.0        <1H OCEAN
## 20615    -121.54     39.08  ...             57500.0           INLAND
## 10250    -117.86     33.88  ...            201400.0        <1H OCEAN
##
## [5 rows x 10 columns]
```

En los próximos temas, no tendremos que definir las particiones “a mano”, si no que vienen implementadas en las clases de la librería Scikit-Learn.