MODULO 4: MACHINE LEARNING

4.1 Explorando las técnicas de *machine learning*

En este módulo expondremos diversas técnicas de *machine learning* que, en la actualidad, son las herramientas más demandadas en las organizaciones en el marco de la ciencia de datos. Estas técnicas permiten identificar patrones dentro de un conjunto amplio de variables que un humano tardaría mucho tiempo en identificar.

Si bien hay gran cantidad técnicas, nos centraremos solo en algunas de ellas, de manera que pueda conocer el potencial de estas e, incluso, crear algunos modelos simples para que pueda comenzar a ponerlos en práctica en problemas cotidianos dentro del mundo del *marketing* y los negocios.

4.1.1 Introducción al machine learning

La minería de datos (*data mining*), dentro del marco de la ciencia de datos, es el estudio que se encarga de las tareas de recopilación, limpieza (*data munging*), procesamiento, análisis y obtención de información útil de los datos. Por lo tanto, es un término amplio que se utiliza para describir los diferentes aspectos del procesamiento de datos.

Machine learning consiste en aplicar algoritmos y modelos estadísticos para encontrar patrones de los datos y utilizar la inferencia para estimar el valor de una variable (*target*) que es de interés para la toma de decisiones.

En el módulo anterior hemos visto herramientas prácticas que nos permiten llevar a cabo muchas de las tareas que se realizan en las primeras etapas del proceso. Dentro de las fases finales, cuando los datos ya están dispuestos en una estructura que nos permite trabajar con ellos, y ya nos aseguramos de que son consistentes y de que contamos con una cantidad suficiente de datos relevantes, iniciamos la etapa puramente analítica de los datos. Esto nos va a permitir sacar los *insights* o los modelos que le darán sentido a todo el proceso. En esta fase entra el concepto de *machine learning* o aprendizaje automático (a partir de ahora usaremos ambos términos de manera indistinta).

El aprendizaje automático se ha convertido en uno de los temas más importantes y populares dentro de las organizaciones que buscan formas innovadoras de aprovechar los datos que poseen para alcanzar un nuevo nivel de comprensión del negocio.

¿Por qué y para qué agregar esta disciplina a la ecuación empresarial? Con los modelos de *machine learning* apropiados, las organizaciones tienen la capacidad de predecir continuamente

cambios en el negocio, de ese modo, son capaces de prepararse para lo que sigue. Si se usan las fuentes de datos más adecuadas y estas poseen una permanente actualización que acompañe los constantes movimientos del contexto de la organización, se tiene una oportunidad inigualable de predecir lo que se puede esperar en el futuro. Pero la clave de todo el modelo es, justamente, su continuo monitoreo, el sondeo de los cambios del negocio y la actualización e incorporación de todo dato relevante a nivel negocio que ayuden a que las predicciones de este sigan performando de la misma forma y no queden obsoletos.

A partir del momento en que surge la necesidad de resolver un problema o situación en el marco de la organización a través de los datos, una habilidad que entra en juego es saber descomponer este problema en partes, de modo que cada una pueda ser encasillada en un proceso o herramienta conocida. Reconocer los problemas y encontrar una solución conocida aplicada a situaciones similares, es una manera práctica e indispensable por parte de un científico de datos para comenzar a trabajar. De este modo, evita la pérdida de tiempo que implica "reinventar la rueda" para resolver cosas que otro ya solucionó con anterioridad. Además, le permite enfocar la atención a otras partes del proceso en las que es posible agregar valor haciendo uso de su creatividad e inteligencia.

4.1.2 Abordajes de machine learning

A pesar de que se desarrollaron gran cantidad de algoritmos de *machine learning* específicos a lo largo de los años, solo hay unos pocos tipos de tareas fundamentalmente distintas que abordan estos algoritmos. Vale la pena definir estas tareas con claridad para comprender el potencial de esta disciplina. Sin embargo, en la actualidad, la mayoría de los casos de negocio concentran sus soluciones en dos: clasificación y regresión, en los cuales nos vamos a enfocar más en este módulo.

En muchos proyectos de análisis de negocios queremos encontrar las correlaciones entre una variable particular que describe a un individuo, y otras variables. Por ejemplo, en una empresa de servicios, en datos históricos se puede saber qué clientes se fueron de la compañía. Con estos datos es posible averiguar qué otras variables se correlacionan con el *churn* (baja) de los clientes. Encontrar tales correlaciones son los ejemplos más básicos de tareas de clasificación y regresión.

La clasificación y el *scoring* intentan predecir, para cada individuo en una población, a qué conjunto de clases pertenece cada uno en particular. Generalmente, las clases son mutuamente excluyentes. Una pregunta de clasificación de ejemplo para un caso de negocio sería: "Entre todos los clientes en la base de datos de una empresa que se les ofrezca una determinada oferta, ¿cuáles son más propensos a responder y cuáles no?" En este ejemplo, las dos clases se podrían denominar: "responderán" y "no responderán".

Para una tarea de clasificación, un procedimiento de minería de datos produce un modelo que, dado un nuevo individuo, determina a qué clase pertenece este. Una tarea estrechamente relacionada es el scoring o estimación de probabilidad de clase. Un modelo de scoring aplicado a un individuo produce, en lugar de una predicción de "a qué clase pertenece", un score que representa la probabilidad de que ese individuo pertenezca a cada clase. En nuestro escenario de respuesta del cliente, un modelo de scoring podría evaluar cada cliente individual y producir un score de la probabilidad de que cada uno responda la oferta. La clasificación y el scoring están muy relacionados y son de los modelos más populares en ciencia de datos aplicados en empresas tradicionales.

La regresión (o estimación de valor) intenta estimar o predecir, para cada individuo, el valor numérico de alguna variable en particular para ese individuo. Un ejemplo de disparador de negocio para realizar una regresión sería: "¿Cuántos pesos gastará en X servicio un cliente

determinado?" En este caso, la variable que se va a predecir es el gasto en el servicio, y un modelo podría generarse a partir del gasto de otros individuos similares en la población y el uso histórico del servicio por parte del individuo

La regresión está bastante relacionada con la clasificación, pero ambas son diferentes. Mientras que un algoritmo de clasificación predice si algo va a suceder, la regresión predice en qué magnitud va a pasar.

El *matcheo* de similitudes (*similarity matching*) se usa para identificar individuos similares según datos conocidos acerca de ellos. Este recurso se puede usar directamente para encontrar entidades o individuos similares, y es la base de uno de los métodos más populares para hacer recomendaciones de productos (encontrar personas similares en términos de los productos que les han gustado o han comprado).

Al realizar un *clustering* lo que se busca es agrupar individuos de una población en grupos (*clusters*) por su similitud, pero no impulsado por ningún propósito específico. Una pregunta de agrupamiento de ejemplo sería: "¿Nuestros clientes forman segmentos naturales por su comportamiento?" El *clustering* es útil para los análisis exploratorios de mercado, para entender qué segmentos existen naturalmente, basados en su comportamiento. Este análisis puede servir de puntapié para otras tareas o enfoques de *data mining*. El *clustering* también se usa como entrada para los procesos de toma de decisiones que se centran en preguntas como: "¿Qué productos deberíamos desarrollar para cubrir las expectativas de nuestros clientes?" "¿Cómo deben ser estructurados nuestros equipos de atención al cliente?"

La agrupación por asociación (también conocida como "análisis del carrito de compras") intenta encontrar asociaciones entre entidades basadas en las transacciones que las involucran. Un ejemplo de pregunta disparadora sería: "¿Qué artículos se compran comúnmente juntos?"

Mientras que el *clustering* analiza la similitud entre los objetos según sus atributos, la agrupación por asociación considera la similitud de los objetos en función de su aparición conjunta en las transacciones. Por ejemplo, analizar los registros de compra de un supermercado puede revelar que los pañales se compran en conjunto con latas de cerveza con mucha más frecuencia de lo que podríamos esperar. Decidir cómo actuar sobre este descubrimiento podría requerir cierta creatividad: podría sugerir una promoción especial, exhibición de productos de manera conjunta en las góndolas o realizar ofertas combinadas.

La concurrencia de productos en compras es un tipo común de agrupación conocida como análisis de carrito de compra. Algunos sistemas de recomendación también realizan agrupación por afinidad al encontrar, por ejemplo, pares de películas que son vistas frecuentemente por las mismas personas ("las personas que vieron *Titanic* también vieron *El lobo de Wall Street*").

El *profiling* (también conocido como "descripción de comportamiento") intenta caracterizar el típico comportamiento de un individuo, grupo o población. Un ejemplo de pregunta disparadora de este método sería: "¿Cuál es el uso típico de los celulares de este segmento de clientes?" El comportamiento puede no tener una descripción simple. El perfil del uso del teléfono celular puede requerir una descripción detallada de los promedios de tiempo de uso de red de noche, horas pico y fines de semana, uso de *roaming*, cantidad de uso de datos, compra de *packs*, cargos por excedencia, cantidad de mensajes de texto, suscripción a contenidos móviles, etcétera. El comportamiento se puede describir, generalmente, sobre una población entera o hasta el nivel de pequeños grupos o, incluso, individuos.

La elaboración de perfiles se utiliza, a menudo, con el fin de establecer normas de comportamiento para la aplicación de detección de anomalías, como la detección de fraudes y el monitoreo de intrusiones en los sistemas informáticos (como *hackers*). Por ejemplo, si sabemos qué tipo de compras suele realizar una persona con una tarjeta de crédito, seremos capaces de determinar si un nuevo cargo en la tarjeta se ajusta a ese perfil o no. Entonces, podremos

usar este grado de no coincidencia como un *score* de fraude y emitir una alarma si el puntaje es lo suficientemente alto como para sospechar.

El *link prediction* (o predicción de enlaces) intenta predecir conexiones entre los distintos datos, por lo general, sugiriendo que debería existir un enlace y, posiblemente, también estimando la fuerza de ese enlace. Esto es muy común en redes sociales como *Facebook* o *Linkedin*: "Dado que usted y X comparten 10 contactos, ¿te gustaría conectar con Y?"

La reducción de dimensionalidad intenta tomar un gran conjunto de datos y reemplazarlo por uno más pequeño que contenga, principalmente, la información más importante del mayor. El conjunto de datos más pequeño puede ser más fácil de manejar y procesar. Además, puede, incluso, revelar mejor la información. Por ejemplo, un conjunto de datos masivo sobre las preferencias de visualización de películas de los consumidores puede reducirse a un conjunto de datos mucho más pequeño que revele las preferencias de gusto del consumidor que están latentes en los datos de visualización (por ejemplo, las preferencias del género del espectador). La reducción de dimensionalidad suele implicar pérdida de información, pero se compensa con una mejor comprensión en menor tiempo.

El modelado causal intenta ayudarnos a comprender qué eventos o acciones realmente influyen sobre los demás. Por ejemplo, considere que usamos modelos predictivos para seleccionar el *target* de consumidores al enviar un anuncio, y observamos que, de hecho, los consumidores objetivo compran a una tasa mayor después de haber sido seleccionados. ¿Esto se debió a que el anuncio influyó en la compra de dichos consumidores? ¿O el modelo predictivo simplemente hizo un gran trabajo identificando a los consumidores que habrían comprado de todas formas?

Las técnicas para el modelado causal incluyen aquellas que involucran inversión en datos, como experimentos controlados aleatorios (por ejemplo, los llamados *A/B testing*), así como métodos sofisticados para extraer conclusiones causales de datos observacionales. Los métodos –tanto experimentales como observacionales– para el modelado causal intentan comprender cuál sería la diferencia de comportamiento de un individuo ante la situación donde el evento de "tratamiento" (en este caso del ejemplo, mostrar un anuncio a un individuo en particular) efectivamente iba a suceder, y en las situaciones en las que no iba a suceder.

4.1.3 Modelos de predicción supervisados

Uno de los principios fundamentales del *data mining* es encontrar o seleccionar variables informativas importantes o "atributos" de ciertos datos de interés, con el fin de resolver alguna problemática de negocio que permita reducir la incertidumbre para la toma de decisiones. Llevar a cabo un modelo predictivo supervisado es una de las mejores opciones para lidiar con esta incertidumbre.

Para poner en práctica un modelo supervisado, uno de los primeros pasos es determinar cuál es la variable *target* que se quiere predecir o comprender mejor. Muchas veces, esta variable es desconocida en el momento en que se la requiere para tomar una decisión. Por ejemplo, si quiere anticiparse y hacerle una oferta a un cliente que se va a dar de baja, aunque aún no sepa si esto va a suceder. Saber cuál es la variable que quiere predecir (la baja o *churn* como se lo conoce en el mundo corporativo) lo ayuda a discernir cuáles son los atributos conocidos que contribuirían a que pueda tener un mejor panorama de por qué los clientes se dan de baja. El solo hecho de encontrar estas variables correlacionadas puede proporcionarle una información muy importante sobre el problema de negocios. Por otro lado, encontrar atributos informativos también es de ayuda para lidiar con volúmenes de datos cada vez más grandes.

Encontrar atributos informativos también es la base para un modelo predictivo llamado *árbol de decisión*. Esta técnica incorpora la idea de segmentación supervisada, selecciona

repetidamente atributos informativos y toma decisiones de acuerdo con el valor que adquiere cada uno de estos. De esta manera, cada rama del árbol se va abriendo cada vez más a medida que las distintas variables adoptan diferentes respuestas hasta llegar al recorrido crítico de valores que tienen que asumir dichos atributos, y que nos deriven a la mejor predicción para alcanzar el *target* buscado.

En términos generales, un modelo es una representación simplificada de la realidad creada para servir a un propósito. Se simplifica en función de algunos supuestos sobre lo que es importante para el propósito específico y lo que no es; en ocasiones se basa en restricciones de información o trazabilidad. Pero, ¿qué significa que sea *supervisado*? Se dice que se supervisa porque tiene un atributo *target* y algunos datos de "entrenamiento" que nos permiten conocer el valor que asumen las distintas variables asociadas cuando se cumple el *target* (en el caso de ser un problema de clasificación si/no), o nos permite conocer el mismo valor del *target* que necesitamos predecir (en caso de ser un problema de regresión). El modelo estima el valor de la variable *target* como una función (posiblemente una función probabilística) de las características observadas en el entrenamiento.

Ahora veremos algunos ejemplos para una mejor comprensión de estos conceptos.

- **Clasificación**

Tendríamos un problema de clasificación si contamos con información comportamental de un cliente en una empresa de telefonía celular (cantidad de reclamos, gastos de facturación, cantidad de productos adquiridos, antigüedad del cliente, total de consumo) y necesitamos estimar si este se va a dar de baja. El *target* que intentamos predecir es si el cliente se va a dar de baja o no, en determinado plazo. Tenemos la posibilidad de entrenar este modelo porque contamos con información histórica de los clientes que se dieron de baja y los distintos valores que asumieron esas variables que mencionamos anteriormente (también denominadas *features*) cuando otros clientes se dieron la baja, y también de aquellos que no lo hicieron.

- **Regresión**

Nos enfrentamos a un problema de regresión cuando lo que necesitamos estimar no es un valor previamente definido (si o no, 1 o 2, blanco o negro), sino un valor numérico desconocido. Este modelo se utiliza, con frecuencia, para hacer pronósticos o *forecasting*. Por ejemplo, necesitamos dimensionar cuál va a ser la demanda de llamadas en un *call center* y así prever la cantidad de operadores en atención telefónica. Para ello, podríamos mirar los datos históricos de la cantidad de llamadas, con datos de día, hora, mes, fechas especiales y otros datos relevantes que podrían tener que ver con la naturaleza del negocio al que se esté brindando la atención. Por ejemplo, si la atención que brindan los operadores es para una tarjeta de crédito, sumaría incorporar variables de hitos como promociones especiales, fecha de cierre y vencimiento del resumen de la tarjeta. De esta manera, se entrena el modelo y, de acuerdo con el valor que asuman dichas variables, se puede estimar con cierto grado de certeza la cantidad de llamadas a esperar por día y hasta por hora.

4.2 Creando un modelo con *Scikit-learn*

Scikit-learn es una de las herramientas de aprendizaje automático de Python más confiables y ampliamente utilizadas. Contiene una amplia selección de métodos de aprendizaje automático supervisados y no supervisados estándar con métodos para la selección y evaluación de modelos, transformación de datos, carga de datos y monitoreo de modelos. En esta librería encontramos algoritmos para hacer clasificación, *clustering* y predicción, entre otros.

Dentro de los algoritmos de aprendizaje supervisado encontramos dos de regresión, pero con distintas implementaciones:

- Regresión lineal:

> [Permite] obtener la relación entre unas variables independientes (X) y una variable dependiente (Y). Es decir, teniendo una serie de variables predictoras obtiene la relación con una variable cuantitativa a predecir. La regresión lineal explica la variable Y con las variables X, y obtiene la función lineal que mejor se ajusta o explica esta relación. (Álvaro, 2018, https://bit.ly/3eq4HxI).

Este algoritmo, entonces, permite predecir valores continuos. Por ejemplo, si tuviéramos el histórico de consumos de MB de datos de una base de clientes, podríamos aplicar una regresión lineal para predecir el consumo de datos de un cliente para un momento determinado.

- Regresión logística: algoritmo de aprendizaje supervisado de clasificación que permite estimar valores discretos. Por ejemplo, si un cliente va a realizar o no una compra frente al estímulo de una oferta. “Los eventos de clasificación pueden ser representados por la variable Y, la cual puede tomar dos valores 0 y 1. El valor 1 habitualmente es representado como un evento exitoso, mientras que el valor 0 como un evento fallido” (Tello, 2020, https://bit.ly/2zuP2hZ).

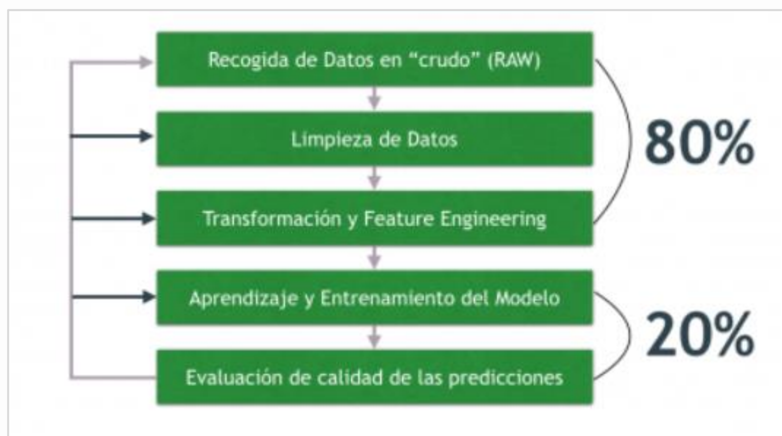
4.2.1 Preparación de los datos

En *machine learning* los algoritmos se ajustan a los cambios del entorno, adaptándose a las nuevas situaciones según se lo vaya alimentando con nuevos datos. Sin embargo, es importante entender que el análisis predictivo no es más que un algoritmo lógico-matemático, y aunque este aprende, solo puede funcionar y extraer sentido de los datos que ingresan al modelo. Los algoritmos no tienen la capacidad de razonar como los humanos, por lo que el éxito de estos depende, principalmente, de los datos de entrada.

El proceso completo de generación de un modelo predictivo, desde la captura de los datos hasta la predicción, equivale en este enfoque a cocinar un plato. Los ingredientes serían los datos, y la receta, el algoritmo: si los ingredientes están en mal estado, por muy buena que sea la receta, el plato no saldrá bien. De forma equivalente, si los datos no son de calidad (es decir, no están bien seleccionados, limpios y transformados), incluso el mejor algoritmo nos dará unas predicciones de baja calidad. (Gonzáles, s. f., https://bit.ly/2XatRuD).

El proceso de preparación de los datos –la obtención, limpieza y transformación de datos– es de suma importancia para la creación de un modelo. Estas etapas de preparación de los datos insumen un 80% del tiempo de la producción de un modelo. El 20% restante corresponde al entrenamiento y generación de métricas para la evaluación de las predicciones.

Figura 1: Cocinar una predicción



Fuente: Gonzáles, s. f., https://bit.ly/2XatRuD

La fase de limpieza de datos comprende, entre otras tareas:

- Igualar formatos [por ejemplo, campos de fecha y hora]
- Descartar campos [por ejemplo, que vienen vacíos, sin datos]
- Corregir errores ortográficos…
- Eliminar columnas duplicadas
- Borrar registros no útiles

Con los datos "limpios" ya se puede empezar a hacer una selección de los que serán útiles para hacer las predicciones. En esta fase hay que descartar todos aquellos datos que no aporten valor al modelo y solo aporten ruido para el mismo. (Gonzáles, s. f., https://bit.ly/2XatRuD).

Este trabajo se suele llamar *feature engineering* y comprende, entre otras, algunas de las siguientes tareas:

- Descartar las variables o *features* con contenido aleatorio.

- Descartar las variables que poseen una alta tasa de registros vacíos.
- Generar nuevas variables de campos categóricos (en caso de modelos de clasificación).
- Generar nuevas variables a partir de estadísticas de otras variables (consumo de los últimos 3 meses, compras en el último año, etcétera).
- Seleccionar las features que son predictoras.

Para aclarar el tema y pensarlo de manera práctica, brindaremos un ejemplo de un modelo de regresión logística en el cual contamos con múltiples parámetros para poder predecir un valor discreto. Para ello, utilizaremos un dataset bastante popular dentro del mundo del machine learning con información de los pasajeros del Titanic, el evento a predecir será si sobrevivió o no al hundimiento.

Para empezar, el primer paso de obtención de los datos es levantar dicho dataset con el comando de pandas read_csv de la siguiente ubicación: https://raw.githubusercontent.com/florenciaortega/ML/master/titanic_original.csv

Lo primero que haremos es un análisis exploratorio de los datos, observaremos el *dataframe* resultante y todas sus variables. Para ello, ejecutaremos el comando de pandas *pd.head ()* con el cual visualizaremos las primeras líneas del *dataset*, o, como en este caso, con el comando *pd.sample ()* que saca una muestra aleatoria de diferentes posiciones de fila del *dataset*.

Figura 2

In [1]:
```
import pandas as pd
import numpy as np
```

In [18]:
```
# Obtenemos el dataset de mi repositorio de github
URL = 'https://raw.githubusercontent.com/florenciaortega/ML/master/titanic_original.csv'
df = pd.read_csv(URL)
```

In [19]:
```
df.sample(5)
```

Out[19]:

	pclass	survived	name	sex	age	sibsp	parch	ticket	fare	cabin	embarked	boat	body	home.dest
174	1.0	0.0	Kent, Mr. Edward Austin	male	58.0	0.0	0.0	11771	29.7000	B37	C	NaN	258.0	Buffalo, NY
1077	3.0	1.0	O'Driscoll, Miss. Bridget	female	NaN	0.0	0.0	14311	7.7500	NaN	Q	D	NaN	NaN
295	1.0	1.0	Thayer, Mr. John Borland Jr	male	17.0	0.0	2.0	17421	110.8833	C70	C	B	NaN	Haverford, PA
598	2.0	1.0	Wright, Miss. Marion	female	26.0	0.0	0.0	220844	13.5000	NaN	S	9	NaN	Yoevil, England / Cottage Grove, OR
639	3.0	0.0	Asplund, Master. Carl Edgar	male	5.0	4.0	2.0	347077	31.3875	NaN	S	NaN	NaN	Sweden Worcester, MA

Las bibliotecas, como *statsmodels* y *scikit-learn*, generalmente no se pueden alimentar con datos faltantes (valores NaN), por lo que miramos las columnas para ver si hay alguno que contenga datos faltantes.

Figura 3

```
In [97]: df.isnull().sum()
Out[97]: pclass          1
         survived        1
         name            1
         sex             1
         age           264
         sibsp           1
         parch           1
         ticket          1
         fare            2
         cabin        1015
         embarked        3
         boat          824
         body         1189
         home.dest     565
         dtype: int64
```

Vemos que todas las columnas tienen algún dato faltante. Pero el que haya varias con 1 solo dato faltante da la pauta de que hay una fila completa que vino vacía. Estos casos en que todos los registros de una fila tienen valores NaN no son útiles para el modelo y los eliminamos.

Para el resto de los registros que tengan solo algunos valores NaN de columnas, es donde conviene imputar datos a esos valores nulos para que todo el registro de esa fila sí ingrese al modelo.

Para localizar qué columna tiene el valor NaN, basta con escribir el siguiente código (ver Figura 4):

Figura 4

```
In [87]: nan_rows = df[df.isnull().any(1)]
         nan_rows
Out[87]:
```

	pclass	name	sex	age	sibsp	parch	ticket	fare	cabin	embarked	boat	body	home.dest
0	1.0	Allen, Miss. Elisabeth Walton	female	29.0000	0.0	0.0	24160	211.3375	B5	S	2	NaN	St Louis, MO
1	1.0	Allison, Master. Hudson Trevor	male	0.9167	1.0	2.0	113781	151.5500	C22 C26	S	11	NaN	Montreal, PQ / Chesterville, ON
2	1.0	Allison, Miss. Helen Loraine	female	2.0000	1.0	2.0	113781	151.5500	C22 C26	S	NaN	NaN	Montreal, PQ / Chesterville, ON
3	1.0	Allison, Mr. Hudson Joshua Creighton	male	30.0000	1.0	2.0	113781	151.5500	C22 C26	S	NaN	135.0	Montreal, PQ / Chesterville, ON
4	1.0	Allison, Mrs. Hudson J C (Bessie Waldo Daniels)	female	25.0000	1.0	2.0	113781	151.5500	C22 C26	S	NaN	NaN	Montreal, PQ / Chesterville, ON
...	...	...	...	...	...	...	...	...	...	...	...	...	...
1305	3.0	Zabour, Miss. Thamine	female	NaN	1.0	0.0	2665	14.4542	NaN	C	NaN	NaN	NaN
1306	3.0	Zakarian, Mr. Mapriededer	male	26.5000	0.0	0.0	2656	7.2250	NaN	C	NaN	304.0	NaN
1307	3.0	Zakarian, Mr. Ortin	male	27.0000	0.0	0.0	2670	7.2250	NaN	C	NaN	NaN	NaN
1308	3.0	Zimmerman, Mr. Leo	male	29.0000	0.0	0.0	315082	7.8750	NaN	S	NaN	NaN	NaN
1309	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

1310 rows × 13 columns

Creamos un *dataset* nuevo llamado "*nan_rows*", donde podemos ver todas las filas que contienen al menos un valor NaN. Como se puede observar, la hipótesis era correcta y la última fila (*index* 1309) vino vacía, seguramente por un error de importación del csv. Vamos a eliminar ese registro con *dropna* y en los parámetros vamos a colocar "*all*" que significa que se tienen que

eliminar los registros donde todos los valores sean NaN. Si pusiéramos "*any*", se eliminarían todas las filas que contengan al menos un NaN y perderíamos gran cantidad de registros, dado que hay muchos *features* que no venían completos. Una vez hecho esto, vemos nuevamente cuántos valores NaN nos quedan para imputar.

Figura 5

```
In [94]: df=df.dropna(how='all')
         df
```

Out[94]:

	pclass	name	sex	age	sibsp	parch	ticket	fare	cabin	embarked	boat	body	home.dest
0	1.0	Allen, Miss. Elisabeth Walton	female	29.0000	0.0	0.0	24160	211.3375	B5	S	2	NaN	St Louis, MO
1	1.0	Allison, Master. Hudson Trevor	male	0.9167	1.0	2.0	113781	151.5500	C22 C26	S	11	NaN	Montreal, PQ / Chesterville, ON
2	1.0	Allison, Miss. Helen Loraine	female	2.0000	1.0	2.0	113781	151.5500	C22 C26	S	NaN	NaN	Montreal, PQ / Chesterville, ON
3	1.0	Allison, Mr. Hudson Joshua Creighton	male	30.0000	1.0	2.0	113781	151.5500	C22 C26	S	NaN	135.0	Montreal, PQ / Chesterville, ON
4	1.0	Allison, Mrs. Hudson J C (Bessie Waldo Daniels)	female	25.0000	1.0	2.0	113781	151.5500	C22 C26	S	NaN	NaN	Montreal, PQ / Chesterville, ON
...	...	...	...	...	...	...	...	...	...	...	...	...	...
1304	3.0	Zabour, Miss. Hileni	female	14.5000	1.0	0.0	2665	14.4542	NaN	C	NaN	328.0	NaN
1305	3.0	Zabour, Miss. Thamine	female	NaN	1.0	0.0	2665	14.4542	NaN	C	NaN	NaN	NaN
1306	3.0	Zakarian, Mr. Mapriededer	male	26.5000	0.0	0.0	2656	7.2250	NaN	C	NaN	304.0	NaN
1307	3.0	Zakarian, Mr. Ortin	male	27.0000	0.0	0.0	2670	7.2250	NaN	C	NaN	NaN	NaN
1308	3.0	Zimmerman, Mr. Leo	male	29.0000	0.0	0.0	315082	7.8750	NaN	S	NaN	NaN	NaN

1309 rows × 13 columns

Figura 6

```
In [99]: df.isnull().sum()

Out[99]: pclass          0
         survived        0
         name            0
         sex             0
         age           263
         sibsp           0
         parch           0
         ticket          0
         fare            1
         cabin        1014
         embarked        2
         boat          823
         body         1188
         home.dest     564
         dtype: int64
```

Si quisiéramos usar "*age*" como predictor, nos encontraríamos igualmente con el problema de que faltan datos. Hay varias maneras de hacer la imputación de datos faltantes, pero usaremos una simple, tomando la mediana del conjunto de datos de entrenamiento para completar los valores nulos del *dataset*.

Figura 7

```
In [102]: impute_value = df['age'].median()
          print(impute_value)

          28.0

In [104]: df['age'] = df['age'].fillna(impute_value)
          df.head(5)

Out[104]:
```

	pclass	survived	name	sex	age	sibsp	parch	ticket	fare	cabin	embarked	boat	body	home.dest
0	1.0	1.0	Allen, Miss. Elisabeth Walton	female	29.0000	0.0	0.0	24160	211.3375	B5	S	2	NaN	St Louis, MO
1	1.0	1.0	Allison, Master. Hudson Trevor	male	0.9167	1.0	2.0	113781	151.5500	C22 C26	S	11	NaN	Montreal, PQ / Chesterville, ON
2	1.0	0.0	Allison, Miss. Helen Loraine	female	2.0000	1.0	2.0	113781	151.5500	C22 C26	S	NaN	NaN	Montreal, PQ / Chesterville, ON
3	1.0	0.0	Allison, Mr. Hudson Joshua Creighton	male	30.0000	1.0	2.0	113781	151.5500	C22 C26	S	NaN	135.0	Montreal, PQ / Chesterville, ON
4	1.0	0.0	Allison, Mrs. Hudson J C (Bessie Waldo Daniels)	female	25.0000	1.0	2.0	113781	151.5500	C22 C26	S	NaN	NaN	Montreal, PQ / Chesterville, ON

Hacemos nuevamente la verificación para la columna "*age*" y verificamos que no quedan valores faltantes (ver Figura 8):

Figura 8

```
In [109]: df['age'].isnull().sum()
Out[109]: 0
```

El paso siguiente para preparar los datos para una regresión es codificar las variables categóricas, como es el caso de la columna "*sex*" que posee dos categorías: "*female*" y "*male*".

Figura 9

```
In [106]: df['IsFemale'] = (df['sex'] == 'female').astype(int)
          df['IsMale'] = (df['sex'] == 'male').astype(int)
          df.head(5)

Out[106]:
```

	pclass	survived	name	sex	age	sibsp	parch	ticket	fare	cabin	embarked	boat	body	home.dest	IsFemale	IsMale
0	1.0	1.0	Allen, Miss. Elisabeth Walton	female	29.0000	0.0	0.0	24160	211.3375	B5	S	2	NaN	St Louis, MO	1	0
1	1.0	1.0	Allison, Master. Hudson Trevor	male	0.9167	1.0	2.0	113781	151.5500	C22 C26	S	11	NaN	Montreal, PQ / Chesterville, ON	0	1
2	1.0	0.0	Allison, Miss. Helen Loraine	female	2.0000	1.0	2.0	113781	151.5500	C22 C26	S	NaN	NaN	Montreal, PQ / Chesterville, ON	1	0
3	1.0	0.0	Allison, Mr. Hudson Joshua Creighton	male	30.0000	1.0	2.0	113781	151.5500	C22 C26	S	NaN	135.0	Montreal, PQ / Chesterville, ON	0	1
4	1.0	0.0	Allison, Mrs. Hudson J C (Bessie Waldo Daniels)	female	25.0000	1.0	2.0	113781	151.5500	C22 C26	S	NaN	NaN	Montreal, PQ / Chesterville, ON	1	0

Entonces, las variables nuevas de sexo y de edad están listas para ingresar como predictoras en la regresión.

Luego tenemos el número de cabina, el cual podría ser un buen *feature* que determine la localización en el barco y, por ende, la cercanía a las salidas de los botes. Veamos la Figura 10 cuántos valores *missing* tiene.

Figura 10

```
In [112]: print('Percent of missing "Cabin" records is %.2f%%' %((df['cabin'].isnull().sum()/df.shape[0])*100))

Percent of missing "Cabin" records is 77.46%
```

Como el porcentaje de valores faltantes es muy elevado, no podemos considerarlo propicio para la imputación. Por ello, descartamos esta variable para ingresarla al modelo.

Figura 11

```
In [124]: df.drop('cabin', axis=1, inplace=True)
```

Hacemos lo mismo con la variable "*embarked*".

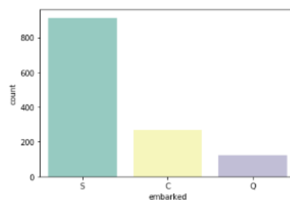
Figura 12

```
In [115]: # percent of missing "Embarked"
          print('Percent of missing "Embarked" records is %.2f%%' %((df['embarked'].isnull().sum()/df.shape[0])*100))

Percent of missing "Embarked" records is 0.15%
```

Como la cantidad de valores faltante es muy poca, podemos imputar esos valores nulos con el puerto de embarque donde más gente se ha subido al barco. A esto lo podemos ver con un gráfico de la librería de *Seaborn*. Para ello importamos dicha librería.

Figura 13

```
In [122]: import seaborn as sns
          print('Pasajeros por puerto de embarcación (C = Cherbourg, Q = Queenstown, S = Southampton):')
          print(df['embarked'].value_counts())
          sns.countplot(x='embarked', data=df, palette='Set3')
          plt.show()

Pasajeros por puerto de embarcación (C = Cherbourg, Q = Queenstown, S = Southampton):
S    914
C    270
Q    123
Name: embarked, dtype: int64
```



Como podemos ver, el puerto de *Southampton* es donde más gente abordó al barco, con lo cual, imputamos los valores faltantes asignando este puerto.

Figura 14

```
In [123]: df["embarked"].fillna(df['embarked'].value_counts().idxmax(), inplace=True)
```

Por otro lado, tenemos las variables “*sibsp*” y “*parch*”, que refieren a la cantidad de hijos y padres que contenía la familia en caso de viajaran en conjunto. Para simplificar, vamos a crear una variable binaria que indique si el pasajero viajaba solo o no, utilizando el comando que vimos en el módulo anterior: “*numpy.where*”.

Figura 15

```
In [128]: df['TravelAlone']=np.where((df["sibsp"]+df["parch"])>0, 0, 1)
          df.head(3)
```

Out[128]:

	pclass	survived	name	sex	age	sibsp	parch	ticket	fare	embarked	boat	body	home.dest	IsFemale	IsMale	TravelAlone
0	1.0	1.0	Allen, Miss. Elisabeth Walton	female	29.0000	0.0	0.0	24160	211.3375	S	2	NaN	St Louis, MO	1	0	1
1	1.0	1.0	Allison, Master. Hudson Trevor	male	0.9167	1.0	2.0	113781	151.5500	S	11	NaN	Montreal, PQ / Chesterville, ON	0	1	0
2	1.0	0.0	Allison, Miss. Helen Loraine	female	2.0000	1.0	2.0	113781	151.5500	S	NaN	NaN	Montreal, PQ / Chesterville, ON	1	0	0

Luego, procedemos a descartar las originales.

Figura 16

```
In [129]: df.drop('parch', axis=1, inplace=True)
          df.drop('sibsp', axis=1, inplace=True)
```

Por otro lado, nos quedan las variables “*pclass*” y “*embarked*” (que imputamos anteriormente), y las vamos a transformar en variables binarias utilizando un método distinto de pandas llamado *get_dummies*. Este toma las variables categóricas y las transforma en numéricas, generando tantas columnas con datos binarios como categorías tienen las variables originales. En la siguiente figura (17) veremos cómo funciona.

Figura 17

```
In [130]: df=pd.get_dummies(df, columns=["pclass","embarked"])
          df.head(3)
```

Out[130]:

age	ticket	fare	boat	body	home.dest	IsFemale	IsMale	TravelAlone	pclass_1.0	pclass_2.0	pclass_3.0	embarked_C	embarked_Q	embarked_S
29.0000	24160	211.3375	2	NaN	St Louis, MO	1	0	1	1	0	0	0	0	1
0.9167	113781	151.5500	11	NaN	Montreal, PQ / Chesterville, ON	0	1	0	1	0	0	0	0	1
2.0000	113781	151.5500	NaN	NaN	Montreal, PQ / Chesterville, ON	1	0	0	1	0	0	0	0	1

Finalmente, como último paso de la limpieza de los datos, descartaremos la variable “*name*”, “*boat*” y “*body*”, que no suman a la regresión, y “*home.dest*” y “*ticket*” para simplificar el análisis, aunque no descartamos que estas pudieran ser *features* de utilidad si son correctamente tratadas para ingresar al modelo.

Figura 18

```
In [132]: df.drop('body', axis=1, inplace=True)
          df.drop('name', axis=1, inplace=True)
          df.drop('ticket', axis=1, inplace=True)
          df.drop('boat', axis=1, inplace=True)
          df.drop('home.dest', axis=1, inplace=True)
```

Veamos cómo quedó el *dataframe* final con la limpieza y el tratamiento hecho en la Figura 19.

Figura 19

```
In [136]: df.sample(5)
Out[136]:
```

	survived	age	fare	IsFemale	IsMale	TravelAlone	pclass_1.0	pclass_2.0	pclass_3.0	embarked_C	embarked_Q	embarked_S
563	0.0	35.0	10.5000	0	1	1	0	1	0	0	0	1
739	0.0	44.0	16.1000	0	1	0	0	0	1	0	0	1
451	0.0	44.0	26.0000	0	1	0	0	1	0	0	0	1
742	1.0	45.0	8.0500	0	1	1	0	0	1	0	0	1
1186	0.0	28.0	21.6792	0	1	0	0	0	1	1	0	0

4.2.2 Entrenando el modelo

La única forma de saber qué tan bien generaliza un modelo, es decir, qué tan bien funciona, es testeándolo con nuevos casos. Entonces, podríamos entrenar nuestro modelo con el dataset completo, ponerlo en producción para que empiece a correr y, recién en esa instancia, ver qué tan bien funciona. Sin embargo, si bien esto puede hacerse, no se lleva a cabo en la práctica, ya que se corre el riesgo de que el modelo haya estado sesgado por alguna razón, o no haya contado con todas las features adecuadas para tener un modelo robusto, o caímos ante el sesgo del overfiting.

El accuracy es la medida de precisión de un algoritmo de clasificación de aprendizaje automático. Es una forma de medir con qué frecuencia el algoritmo predice correctamente. La precisión es el número de casos pronosticados correctamente de todos los casos de un dataset. Más formalmente, se define como el número de verdaderos positivos y verdaderos negativos dividido por el número total de verdaderos positivos, verdaderos negativos, falsos positivos y falsos negativos. Un verdadero positivo o verdadero negativo se da cuando el algoritmo clasificó correctamente como verdadero o falso respectivamente. Un falso positivo o falso negativo, por otro lado, se da cuando el algoritmo clasificó incorrectamente un caso. Por ejemplo, volviendo al algoritmo del hospital, si clasifica un caso de cáncer negativo como positivo, sería un falso negativo y viceversa.

Imaginemos el caso en el que creamos un sistema que diagnostique enfermedades en pacientes según sus historias clínicas, si se pone a andar ese sistema cuando ya está en el hospital funcionando, puede traer conclusiones catastróficas para las personas. Por ejemplo, falsos diagnósticos. Por eso, a la hora de entrenar un modelo se separa el *dataset* en dos partes: por un lado, vamos a tener el conjunto de entrenamiento (*training set*); por otro, vamos a tener el conjunto de test (*testing set*). Entonces, se entrena con este conjunto *train* y, una vez que ya conseguimos un modelo predictivo, vamos a probarlo con las instancias del conjunto de *test*. Solo

una vez superadas estas dos instancias y verificado que el *accuracy* del modelo es razonable, se pasa a publicar el modelo para "deployarlo".

Una vez que tenemos todos los datos listos, procedemos a separar el *dataset* en conjunto de entrenamiento y testeo, y luego identificamos las variables independientes (X) y la dependiente (y), que es nuestro target. En el caso del *dataset* del Titanic, lo que buscamos es predecir la supervivencia de los pasajeros tras su hundimiento, con lo cual, la variable target es "*survived*".

Figura 20

```
In [138]: from sklearn.model_selection import train_test_split
          train, test = train_test_split(df, random_state = 1234)

In [139]: print(train.shape)
          print(test.shape)

          (981, 12)
          (328, 12)

In [140]: y = df.pop('survived')
          X = df

In [141]: X_train, y_train, X_test, y_test = train_test_split(X, y)

In [142]: print(X_train.shape)
          print(X_test.shape)
          print(y_train.shape)
          print(y_test.shape)

          (981, 11)
          (981,)
          (328, 11)
          (328,)

In [144]: X_train.head(3)
Out[144]:
```

	age	fare	IsFemale	IsMale	TravelAlone	pclass_1.0	pclass_2.0	pclass_3.0	embarked_C	embarked_Q	embarked_S
983	28.0	7.5500	0	1	1	0	0	1	0	0	1
498	32.0	13.5000	0	1	1	0	1	0	0	0	1
1022	28.0	7.8958	0	1	1	0	0	1	0	0	1

El siguiente paso es convertir en arreglos de *numpy* los conjuntos de testeo y entrenamiento. Primero, identificamos las variables independientes dentro del arreglo que vamos a llamar "*predictors*"; segundo, creamos los arreglos *X_train* y *X_test* sobre la base de los *dataframes* de *train* y test, pero tomando únicamente estas variables que llamamos *predictoras*. Luego, dejamos en los arreglos *y_train* e *y_test* la variable "*survived*", que es nuestra variable *target*.

Figura 21

```
In [240]: predictors = ['pclass_1.0','pclass_2.0','pclass_3.0', 'IsFemale', 'IsMale',
                        'age','embarked_C','embarked_Q','embarked_S','TravelAlone','fare']

In [264]: X_train = train[predictors].values
          print(X_train[:2])

          [[ 0.      0.      1.      1.      0.      0.75    1.      0.      0.
             0.     19.2583]
           [ 1.      0.      0.      0.      1.     14.4542  1.      0.      0.
             1.     39.6   ]]

In [265]: X_test = test[predictors].values
          print(X_test[:2])

          [[ 0.      0.      1.      0.      1.     14.4542  0.      0.      1.
             1.      8.05  ]
           [ 1.      0.      0.      0.      1.     36.      0.      0.      1.
             0.     78.85  ]]

In [266]: y_train = train['survived'].values
          print(y_train[:10])

          [1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]

In [267]: y_test = test['survived'].values
          print(y_test[:10])

          [0. 0. 0. 0. 0. 1. 1. 1. 0. 1.]
```

El paso siguiente es entrenar nuestro modelo con una regresión logística, considerando nuestros conjuntos de entrenamiento *X_train* e *y_train* utilizando el método *fit*.

Figura 22

```
In [255]: from sklearn.linear_model import LogisticRegression
          model = LogisticRegression()
          model.fit(X_train, y_train)

Out[255]: LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                             intercept_scaling=1, l1_ratio=None, max_iter=100,
                             multi_class='auto', n_jobs=None, penalty='l2',
                             random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                             warm_start=False)
```

Finalmente, testeamos con el conjunto de *testing* para poder comparar los resultados y medir el *accuracy* de nuestro modelo de regresión logística.

Figura 23

```
In [257]: y_predict = model.predict(X_test)
          y_predict[:20]

Out[257]: array([0., 0., 0., 0., 0., 1., 0., 0., 0., 1., 1., 0., 0., 0., 0., 1., 0.,
                 0., 0., 0.])

In [258]: from sklearn.metrics import accuracy_score
          metrics.accuracy_score(y_test, y_predict, normalize=True, sample_weight=None)

Out[258]: 0.7804878048780488
```

El *accuracy* de nuestro modelo dio 78%, lo que indica que del entrenamiento que hizo (con el conjunto *train*: *X_train* e y_train), al aplicarlo al conjunto de testeo (*X_test*) nos arrojó un 78% de predicciones correctas, al compararla con los valores de *y* reales para ese conjunto de testeo (*y_test*).

Todo resultado por encima del 50% es bueno en el sentido de que supera la aleatoriedad, y si el *accuracy* da muy cercano al 100%, hay que prestar atención porque nos podemos encontrar ante un caso de *overfitting*.

Igualmente, los modelos tienen parámetros de ajuste que nos permiten ir optimizando este valor de precisión del modelo. Uno de los métodos para hacerlo es lo que llamamos *cross validation*.

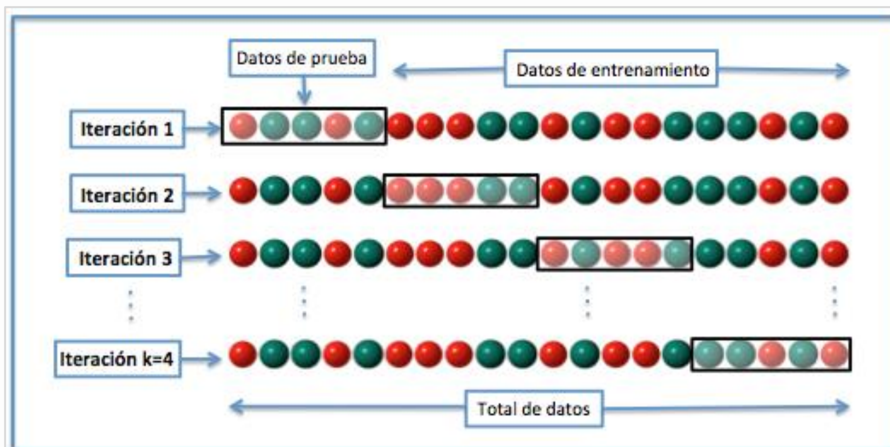
4.2.3 Cross validation

Cross validation es una técnica que, de un mismo conjunto de entrenamiento, hace subdivisiones e iteraciones de entrenamiento y testeo, de esta forma se evita generalizar y sobre ajustar el modelo a un único *dataset*. Esto es así porque, si entrenamos primero con un subconjunto y lo testeamos con una partecita del mismo, pero después hacemos una iteración cambiando esos subconjuntos de datos, va a pasar que nuestro algoritmo conozca otras instancias y reciba otras instancias de testeo que no conocía antes y, de esta forma, va a poder adaptarse a la mayor cantidad de casos posibles.

Existen diferentes formas de utilizar *cross validation* en nuestros conjuntos de entrenamientos. Entre ellas, lo que cambia es la forma de dividirlos y crear subconjuntos.

- *Kfold cross validation*: divide en cada subconjunto de testeo nuestro conjunto de entrenamiento, esto va a ir iterando K veces, entrenando y testeando con cada uno de estos subconjuntos.

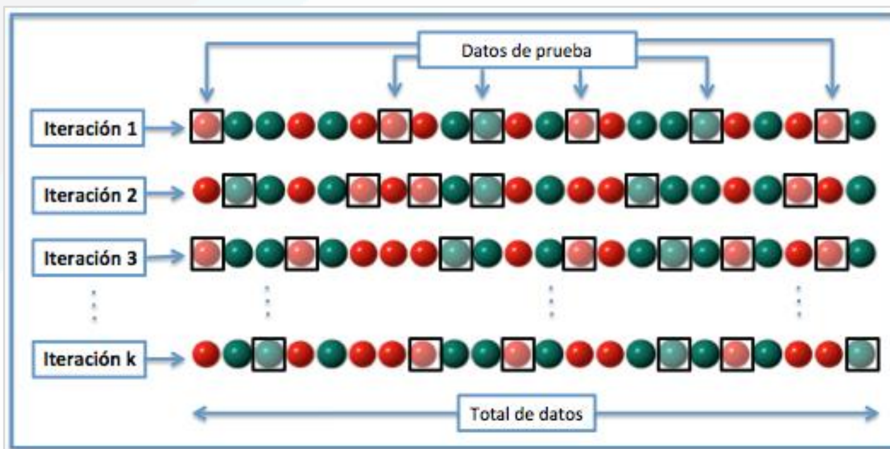
Figura 24: K-fold cross validation



Fuente: Creative Commons, 2011, https://bit.ly/2BcCHPR

- Después tenemos el *cross validation* aleatorio, que a estos subconjuntos los va a tomar al azar, es decir, en cada iteración va a tomar un subconjunto al azar y así hacer una iteración.

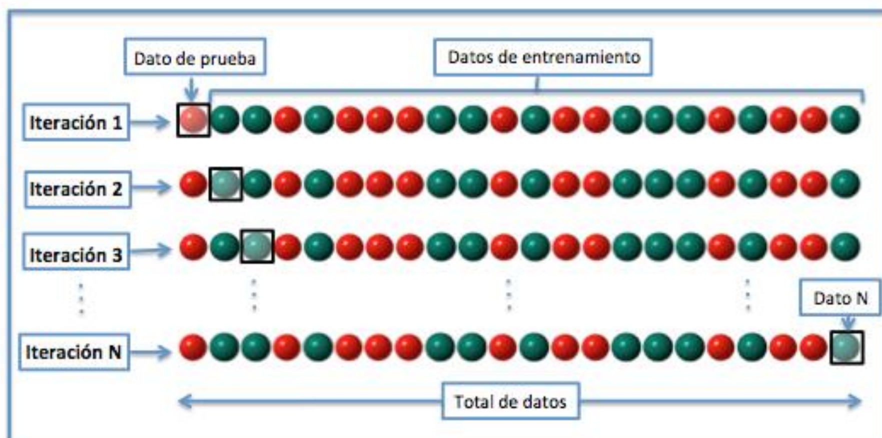
Figura 25: Cross validation aleatorio



Fuente: Creative Commons, 2011), https://bit.ly/3gwVYvG

- Después, el *leave one out* toma todas las instancias del entrenamiento, menos una, y solo la testea con esa instancia extra. Esta lo va a iterar por N veces, (N instancias), es decir, tomamos un subconjunto, dejamos uno afuera, entrenamos y testeamos; en el próximo, lo mismo, toma un subconjunto, lo saca para afuera, entrenamos y testeamos. De esta forma lo que conseguimos es que nuestro modelo pueda predecir de la mejor manera posible, es decir, que pueda generalizar ante la llegada de nuevos datos.

Figura 26



Fuente: Creative Commons, 2011, https://bit.ly/2ZlCrCp

Una vez que hicimos este entrenamiento con *cross validation* y ya tenemos una buena métrica de cómo se comporta, entonces vamos a traer nuestro conjunto de test que dejamos separado (y que nunca se tocó) y ahora sí se testea con este conjunto.

Lo que vamos a ver ahora es cómo utilizar *cross validation* desde *scikit-learn*. Esto es, con los datos cargados vamos a separar en un conjunto de entrenamiento y en un conjunto de testeo, sobre el conjunto de entrenamiento vamos a hacer las diferentes iteraciones utilizando *cross val score* de *scikit-learn*.

Figura 27

```
## cross validation

In [422]: from sklearn.model_selection import cross_val_score

          model = LogisticRegression(C=10)

In [424]: scores = cross_val_score(model, X_train, y_train, cv=4)
          print(scores)
          print(scores.mean())

          [0.79268293 0.80816327 0.76734694 0.78367347]
          0.787966650074664
```

Acá podemos ver que con nuestro conjunto de entrenamiento hicimos 4 iteraciones (cv=4). El *accuracy* para cada iteración se puede ver en el primer arreglo, y el promedio de todas las iteraciones arrojó 78,79%.

4.2.4 Grid search

Grid search es el proceso de escanear datos para configurar parámetros óptimos para un modelo. Dependiendo del tipo de modelo utilizado, ciertos parámetros son necesarios. Se puede aplicar a través del aprendizaje automático con el fin de calcular los mejores parámetros que se utilizarán para distintos modelos y nos ayudará a decidir cuál es el modelo más adecuado con la mejor predicción para el trabajo que estemos realizando.

Es importante tener en cuenta que *Grid search* puede ser extremadamente costoso, desde el punto de vista computacional, y puede demorar mucho tiempo en ejecutar. La mecánica de trabajo por detrás de esta herramienta es que crea un modelo con cada combinación de parámetros posible otorgada en el listado de parámetros de entrada. Recorre cada combinación de parámetros y almacena un modelo para cada combinación.

Veamos un ejemplo de implementación.

Si bien *Grid search* podría funcionar perfectamente con la regresión logística que hicimos anteriormente, para poder apreciar mejor sus bondades, utilizaremos un árbol de decisión que nos permita jugar un poco más con los parámetros de entrada de modo que se pueda apreciar más fácilmente.

El fundamento de *Grid search* es crear un diccionario con los parámetros de ajuste del modelo que queremos entrenar. En cada parámetro podemos colocar varias opciones de ajuste. Por ejemplo, en el árbol de decisión (*decision tree classifier*) de *scikit-learn* encontramos varios parámetros de ajuste como: *criterion* (criterio para selección de variables), *splitter* (estrategia usada para hacer el *split* de las ramas), *max_depth* (máxima profundidad del árbol), *max_features* (máximas variables tenidas en cuenta para el *split* de las ramas), etcétera.

Para saber más sobre los parámetros de este modelo, se recomienda consultar la documentación oficial en: https://scikit-learn.org/0.15/modules/generated/sklearn.tree.DecisionTreeClassifier.html

Figura 28

```
## Grid Search _DecisionTreeClassifier

In [444]: from sklearn.model_selection import GridSearchCV
          from sklearn.tree import DecisionTreeClassifier

In [445]: tree_model=DecisionTreeClassifier()

In [446]: param_grid= { 'max_depth':np.arange(3,6),
                        'splitter': ['best', 'random'],
                        'criterion':['gini','entropy']}

          grid = GridSearchCV(tree_model, param_grid, scoring='precision')

          grid.fit(X_train,y_train)

          print("Forest best parameters :",grid.best_params_)
          print("Forest best estimator :",grid.best_estimator_ )
          print("Forest best score :",grid.best_score_ )

          Forest best parameters : {'criterion': 'gini', 'max_depth': 4, 'splitter': 'best'}
          Forest best estimator : DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                                 max_depth=4, max_features=None, max_leaf_nodes=None,
                                 min_impurity_decrease=0.0, min_impurity_split=None,
                                 min_samples_leaf=1, min_samples_split=2,
                                 min_weight_fraction_leaf=0.0, presort='deprecated',
                                 random_state=None, splitter='best')
          Forest best score : 0.8286343181348832
```

Cuando “seteamos” los parámetros del algoritmo, en vez de ejecutar varias veces de manera manual con distintas combinaciones de parámetros, entrenar el modelo y luego calcular las métricas; cargamos el diccionario *param_grid* que creamos donde tenemos todos los posibles valores por parámetro que quisiéramos probar. Luego *Grid Search* generará tantos árboles como combinaciones posibles haya de esos parámetros, hasta encontrar la mejor combinación que maximice la precisión del modelo. Luego de que reentrenamos el modelo con esta mejor combinación de parámetros, testeamos con nuestro conjunto test para ver cómo nos dio el *accuracy*.

Figura 29

```
In [448]: y_predict_GS = grid.predict(X_test)

In [449]: metrics.accuracy_score(y_test, y_predict_GS)
Out[449]: 0.823170731707317
```

Podemos ver que aumentamos el *accuracy* de 78% a 82%, lo cual implica una gran mejora en términos de predicción y, por otro lado, al haber implementado *Grid Search* que utiliza técnicas de *cross validation*, nos asegura la robustez de este modelo.

4.2.5 Matriz de confusión

Una matriz de confusión es un resumen de los resultados de predicción sobre un problema de clasificación. El número de predicciones correctas e incorrectas se resume con valores sumarizados y se desglosa por clase. Esta es la clave de la matriz de confusión.

La matriz de confusión muestra las formas en que su modelo de clasificación erra cuando hace predicciones. Nos da una idea no solo de los errores que está cometiendo el algoritmo de clasificación, sino, más importante aún, de los tipos de errores que se están cometiendo.

Figura 30: Matrices de confusión

		Predicción	
		Positivos	**Negativos**
Observación	**Positivos**	Verdaderos Positivos (VP)	Falsos Negativos (FN)
	Negativos	Falsos Positivos (FP)	Verdaderos Negativos (VN)

Fuente: Ecured.cu, 2019, https://bit.ly/2M3XaZo

Donde:

- Verdaderos positivos (VP): la observación real y la predicción fueron ambas positivas.
- Falsos negativos (FN): la observación es positiva, pero la predicción es negativa.
- Verdaderos negativos (VN): observación negativa y predicción negativa.
- Falsos positivos (FP): observación negativa, pero la predicción es positiva.

El *accuracy* (la exactitud del modelo) se calcula como:

Figura 31: Cálculo del *accuracy*

$$\text{Accuracy} = \frac{VP + VN}{VP + VN + FP + FN}$$

Fuente: Elaboración propia.

El *recall* se puede definir como la relación entre el número total de casos positivos correctamente clasificados (VP) y el número total de ejemplos positivos (VP + FN). Un alto *recall* indica que la clase se reconoce correctamente (hay un pequeño número de FN).

Figura 32: Cálculo del *recall*

$$\text{Recall} = \frac{VP}{VP + FN}$$

Fuente: Elaboración propia.

Para obtener el valor de precisión, dividimos el número total de ejemplos positivos correctamente clasificados (VP) por el número total de ejemplos positivos predichos (VP+FP). Alta precisión indica que un ejemplo etiquetado como positivo es realmente positivo (hay un pequeño número de FP).

Figura 33: Cálculo del valor de precisión

$$\text{Precision} = \frac{VP}{VP + FP}$$

Fuente: Elaboración propia.

Dependiendo de cuál sea el *target* del modelo que estemos evaluando, las métricas de *recall* y precisión se vuelven muy importantes. Retomando el ejemplo anterior del algoritmo de diagnóstico de una enfermedad (llamémosle “coronavirus”) en un hospital. Evaluemos dos posibles matrices de confusión y analicemos cuál sería la más apropiada.

Figura 34: Matrices de confusión

A	pred _ P	pred _ N
real _ p	83	7
real _ n	6	4

Accuracy	87%
Recall	92%
Precision	93%

B	pred _ P	pred _ N
real _ p	80	1
real _ n	17	2

Accuracy	82%
Recall	99%
Precision	82%

Fuente: Elaboración propia.

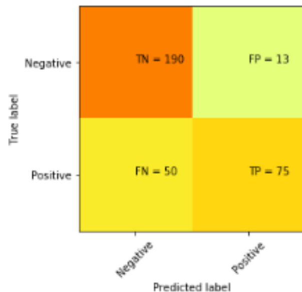
Si bien, a simple vista, observando las métricas de ambas uno tendería a elegir un modelo que arroje un *accuracy* de 87% versus uno de 82%, cuando miramos el *recall* del modelo (cantidad de valores positivos bien clasificados sobre el total de observaciones reales positivas), vemos que el modelo B tiene 99%, con lo cual nos daría la pauta de que solo el 1% de los casos que fueron pronosticados como saludables en realidad tenía coronavirus. Cuando entra en juego algo tan delicado como la salud de las personas, muchas veces es preferible ceder *accuracy* o precisión del modelo, pero ganar en *recall*.

En el modelo B, seguramente habrá más falsos diagnosticados con coronavirus, que quizás deban cumplir una cuarentena obligatoria o someterse a estudios. Sin duda, esto es preferible a que haya más personas con diagnósticos negativos que en realidad tienen la enfermedad y, sin embargo, hacen vida normal poniendo en riesgo tanto a sus allegados y como a la sociedad.

Volviendo al modelo del “Titanic”, veamos cómo armar una matriz de confusión en Python tomando como base nuestro último modelo de árbol de decisión.

Figura 35

```
In [524]: plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Wistia)
          classNames = ['Negative','Positive']
          plt.ylabel('True label')
          plt.xlabel('Predicted label')
          tick_marks = np.arange(len(classNames))
          plt.xticks(tick_marks, classNames, rotation=45)
          plt.yticks(tick_marks, classNames)
          s = [['TN','FP'], ['FN', 'TP']]
          for i in range(2):
              for j in range(2):
                  plt.text(j,i, str(s[i][j])+" = "+str(cm[i][j]))
          plt.show()
```



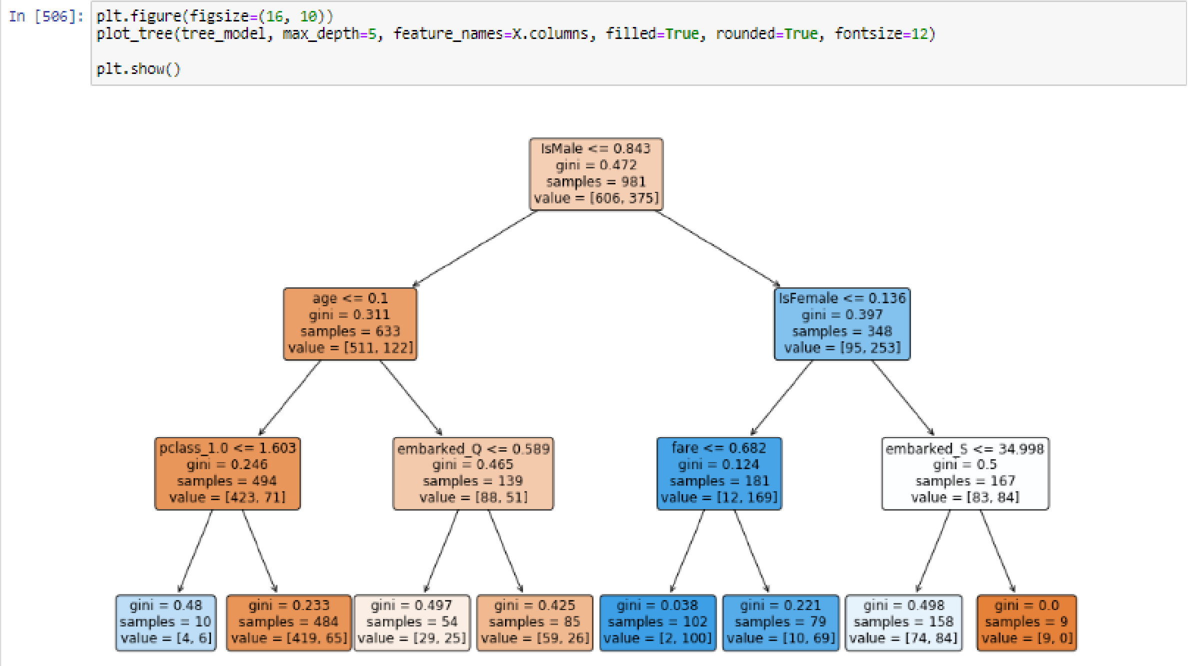
4.2.6 Graficando un árbol de decisión

Los árboles de decisión son una técnica de aprendizaje automático supervisado muy utilizada en muchos negocios. Como su nombre indica, esta técnica de machine learning toma una serie de decisiones en forma de árbol. Los nodos intermedios (las ramas) representan soluciones. Los nodos finales (las hojas) nos dan la predicción que vamos buscando. (Martínez Heras, s. fi, https://bit.ly/2Xe8GYB).

Una vez que tenemos nuestro modelo entrenado, muchas veces, si la profundidad del árbol lo permite, es bastante útil observar cómo se fueron desglosando las decisiones en el árbol para poder comprender mejor la manera en que incidieron los *features* en la predicción final y, así, comprender mejor y sacar *insights* del problema que se esté evaluando.

Observemos cómo se vería en Python nuestro árbol de decisión del Titanic.

Figura 36



Gini: medida de impureza

Gini es una medida de impureza. Cuando gini vale 0 [como en el último nodo a la derecha abajo del árbol, donde hay 9 casos], significa que ese nodo es totalmente puro. La impureza se refiere a cómo de mezcladas están las clases en cada nodo. Para calcular la impureza de gini, usamos la siguiente fórmula:

Figura 37: Valor del coeficiente de gini

$$gini = 1 - \sum_{k=1}^{n} p_c^2$$

p_c se refiere a la probabilidad de cada clase. Podemos calcularla dividiendo el número de muestras [*samples*] de cada clase en cada nodo, por el número de muestras totales por nodo. (Martínez Heras, s. f., https://bit.ly/2Xe8GYB).

Referencias

Álvaro, G. (15 de junio de 2018). Regresión lineal en Python. *Machine learning para todos*. Recuperado de https://machinelearningparatodos.com/regresion-lineal-en-python/

Figura 24. Creative Commons. (8 de diciembre de 2011). K-fold cross validation. Recuperado de https://commons.wikimedia.org/wiki/File:K-fold_cross_validation.jpg

Figura 25. Creative Commons. (8 de diciembre de 2011). Random cross validation. Recuperado de https://commons.wikimedia.org/wiki/File:Leave-one-out.jpg

Figura 26. Creative Commons. (8 de diciembre de 2011). Leave-one-out. Recuperado de https://commons.wikimedia.org/wiki/File:Random_cross_validation.jpg

Figura 30. Ecured.cu. (9 de septiembre de 2019). Matrices de confusion. Recuperado de https://www.ecured.cu/Archivo:Matrices_de_confusi%C3%B3n.png

Gonzáles, A. (s. f.). La importancia de limpiar, seleccionar y transformer los datos. *Cleverdata.io*. Recuperado de https://cleverdata.io/limpiar-seleccionar-transformar-datos/

Martínez Heras, J. (s. f.). Árboles de decisión con ejemplos en Python. IArtificial.net. Recuperado de https://iartificial.net/arboles-de-decision-con-ejemplos-en-python/

Tello, D. (29 de febrero de 2020). Regresión Logística en Python y R. Machine learning #02. *Medium.com*. Recuperado de https://medium.com/@dtellogaete/regresi%C3%B3n-log%C3%ADstica-en-python-y-r-machine-learning-02-fa066b3add09