



ARTIFICIAL INTELLIGENCE PROFESSIONAL CERTIFICATE



CertiProf®

ARTIFICIAL INTELLIGENCE PROFESSIONAL CERTIFICATE

CAIPC®



CAIPC® Versión 062021

CertiProf®

Objetivos de Aprendizaje

- Comprender los fundamentos de la Inteligencia Artificial y el Aprendizaje Automático
- Describir los métodos de aprendizaje automático: supervisado y no supervisado
- Utilizar el Análisis de Datos para la toma de decisiones
- Comprender los límites de los algoritmos
- Entender y comprender la programación en Python, los conocimientos matemáticos esenciales en IA y los métodos básicos de programación

¿Quién es CertiProf®?

CertiProf® es una entidad certificadora fundada en los Estados Unidos en 2015, ubicada actualmente en Sunrise, Florida.

Nuestra filosofía se basa en la creación de conocimiento en comunidad y para ello su red colaborativa está conformada por:

- Nuestros **Lifelong Learners (LLL)** se identifican como Aprendices Continuos, lo que demuestra su compromiso inquebrantable con el aprendizaje permanente, que es de vital importancia en el mundo digital en constante cambio y expansión de hoy. Independientemente de si ganan o no el examen.
- Las universidades, centros de formación, y facilitadores en todo el mundo forman parte de nuestra red de aliados **ATPs (Authorized Training Partners.)**
- Los autores (**co-creadores**) son expertos de la industria o practicantes que, con su conocimiento, desarrollan contenidos para la creación de nuevas certificaciones que respondan a las necesidades de la industria.
- Personal Interno: Nuestro equipo distribuido con operaciones en India, Brasil, Colombia y Estados Unidos está a cargo de superar obstáculos, encontrar soluciones y entregar resultados excepcionales.

Nuestras Acreditaciones y Afiliaciones

Memberships



Digital badges issued by

The logo for Credly is the word "Credly" written in a large, orange, cursive, handwritten-style font.

Agile Alliance

CertiProf® es un miembro corporativo de la Agile Alliance.

Al unirnos al programa corporativo Agile Alliance, continuamos empoderando a las personas ayudándolas a alcanzar su potencial a través de la educación. Cada día, brindamos más herramientas y recursos que permiten a nuestros socios capacitar a profesionales que buscan mejorar su desarrollo profesional y sus habilidades.

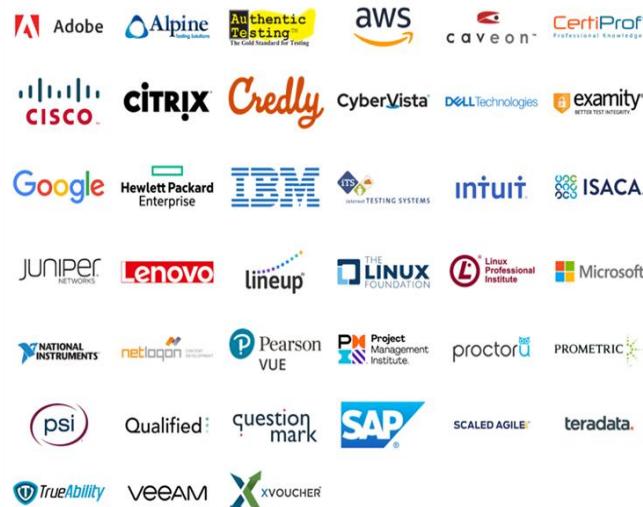
<https://www.agilealliance.org/organizations/certiprof/>



IT Certification Council - ITCC

CertiProf® es un miembro activo de la ITCC.

El propósito fundamental del ITCC es brindar apoyo a la industria y sus empresas miembros mediante la comercialización del valor de la certificación, la promoción de la seguridad de los exámenes, el fomento de la innovación y el establecimiento y el intercambio de las mejores prácticas de la industria.



Esta alianza permite que las personas y empresas certificadas o acreditadas con CertiProf® cuenten con una distinción a nivel mundial a través de un distintivo digital.

Credly es el repositorio de insignias más grande del mundo y empresas líderes en el área de tecnología como IBM, Microsoft, PMI, Nokia, la Universidad de Stanford, entre otras, emiten sus insignias con Credly.



Quién Debería Asistir a Este Taller de Certificación

- Cualquier persona interesada en ampliar sus conocimientos en Inteligencia Artificial y Aprendizaje Automático
- Ingenieros, analistas, directores de marketing
- Analistas de datos, científicos de datos, administradores de datos
- Cualquier persona interesada en técnicas de minería de datos y aprendizaje automático

Presentación

¡Bienvenidos!

Reporte en el siguiente formato:

- Nombre
- Compañía
- Nombre del cargo y experiencia
- Expectativas de este curso



Artificial Intelligence Professional Certificate - CAIPC®

Issued by [CertiProf](#)

The holders of this badge have validated their skills and knowledge in Artificial Intelligence Professional, understanding key principles and concepts such as machine learning, essential mathematics knowledge in AI, and basic programming methods. They have demonstrated how to use the data analysis for decision-making and understand the limits of algorithms.

[Learn more](#)

Certification

Paid

Skills

AI

Algorithms

Data Analysts

Data Scientists

Data Stewards

Machine Learning

<https://www.credly.com/org/certiprof/badge/artificial-intelligence-professional-certificate-ca>

Lifelong Learning

Los portadores de esta insignia en particular han demostrado su compromiso inquebrantable con el aprendizaje permanente, que es de vital importancia en el mundo digital actual, en constante cambio y expansión. También identifica las cualidades de una mente abierta, disciplinada y en constante evolución, capaz de utilizar y contribuir con sus conocimientos al desarrollo de un mundo más igualitario y mejor.

Criterios de Adquisición:

- Ser candidato a una certificación de CertiProf
- Ser un aprendiz continuo y enfocado
- Identificarse con el concepto de aprendizaje permanente
- Creer e identificarse realmente con el concepto de que el conocimiento y la educación pueden y deben cambiar el mundo
- Querer impulsar su crecimiento profesional



COMPARTE Y VERIFICA

TUS LOGROS DE APRENDIZAJE FÁCILMENTE

#CAIPC #CertiProf



CertiProf®
certiprof.com

Fundamentos del Aprendizaje Automático



Fundamentos del Aprendizaje Automático - ML

En 1959, Arthur Samuel, informático pionero en el estudio de la inteligencia artificial, describió el aprendizaje automático - ML- como "el estudio que da a los ordenadores la capacidad de aprender sin ser programados explícitamente". El artículo seminal de Alan Turing (Turing, 1950) introdujo una norma de referencia para demostrar la inteligencia de las máquinas, de manera que una máquina tiene que ser inteligente y responder de una manera que no pueda diferenciarse de la de un ser humano.

El aprendizaje automático es una aplicación de la inteligencia artificial en la que un ordenador/máquina aprende de las experiencias pasadas (datos de entrada) y hace predicciones futuras. El rendimiento de un sistema de este tipo debería estar, como mínimo, a la altura del ser humano.

En este material, nos centraremos en los problemas de clustering para el aprendizaje automático no supervisado con el algoritmo K-Means. Para el aprendizaje automático supervisado describiremos el problema de clasificación con una demostración del algoritmo de árboles de diseño y el de regresión con un ejemplo de regresión lineal. A continuación se presenta un resumen que representa los tipos de aprendizaje automático y algunos algoritmos como ejemplos en la siguiente figura:

Fundamentos del Aprendizaje Automático - ML

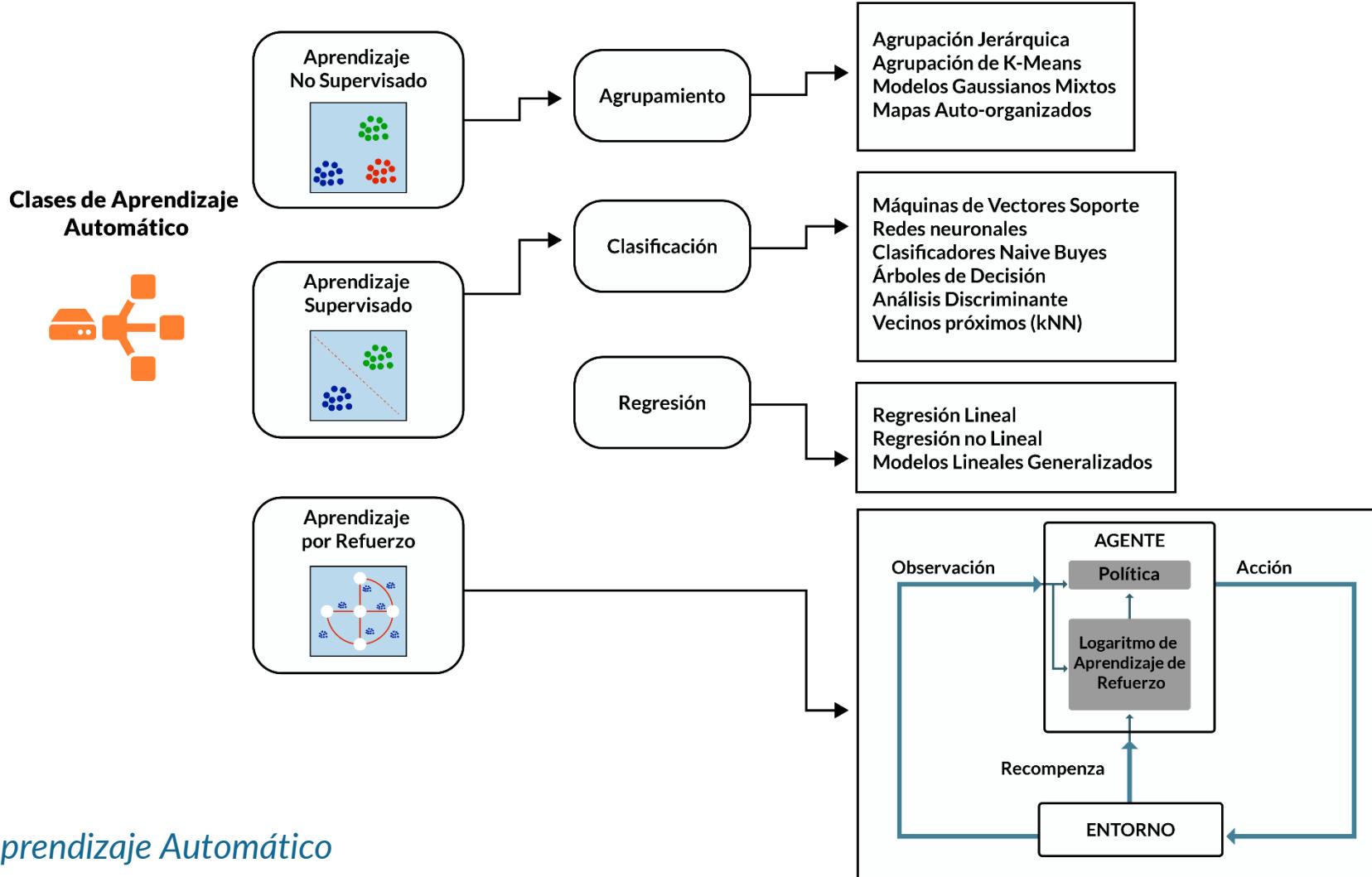


Figura 1. Tipos de Aprendizaje Automático

I.1 Puntos Clave



Aprendizaje Automático Supervisado

El Aprendizaje Supervisado es un enfoque para crear inteligencia artificial (IA), en el que un algoritmo informático se entrena con datos de entrada que han sido etiquetados para un resultado concreto. El modelo se entrena hasta que puede detectar los patrones subyacentes y las relaciones entre los datos de entrada y las etiquetas de salida, lo que le permite obtener resultados de etiquetado precisos cuando se le presentan datos nunca antes vistos.

Aprendizaje Automático Supervisado

Clasificación

Un algoritmo de clasificación tiene como objetivo clasificar las entradas en un número determinado de categorías o clases, basándose en los datos etiquetados con los que fue entrenado. Los algoritmos de clasificación pueden utilizarse para clasificaciones binarias, como filtrar el correo electrónico en spam o no spam y clasificar los comentarios de los clientes como positivos o negativos. El reconocimiento de características, como el reconocimiento de letras y números escritos a mano o la clasificación de medicamentos en muchas categorías diferentes, es otro problema de clasificación que se resuelve con el aprendizaje supervisado.

Regresión

El análisis de regresión consiste en un conjunto de métodos de aprendizaje automático que permiten predecir una variable de resultado continua (y) en función del valor de una o varias variables predictoras (x). En resumen, el objetivo del modelo de regresión es construir una ecuación matemática que defina y como una función de las variables x .

Aprendizaje Automático no Supervisado

El aprendizaje no supervisado es una técnica de aprendizaje automático en la que los usuarios no necesitan supervisar el modelo. En su lugar, permite que el modelo trabaje por sí mismo para descubrir patrones e información que antes no se detectaba. Se ocupa principalmente de los datos no etiquetados.

Agrupamiento - (Clustering)

Este método de clasificación no supervisada reúne un conjunto de algoritmos de aprendizaje cuyo objetivo es agrupar datos no etiquetados con propiedades similares. Aislar patrones o familias de esta manera también prepara el terreno para la posterior aplicación de algoritmos de aprendizaje supervisado (como el KNN).

Aprendizaje Automático por Refuerzo

El Aprendizaje por Refuerzo (RL) es un área del aprendizaje automático que se ocupa de cómo los agentes inteligentes deben realizar acciones en un entorno para maximizar la noción de recompensa acumulada. El aprendizaje por refuerzo es uno de los tres paradigmas básicos del aprendizaje automático, junto con el aprendizaje supervisado y el aprendizaje no supervisado.

I.2 Introducción K-Nearest Neighbors



Introducción

En esencia, la ciencia de los datos nos ayuda a dar sentido al enorme mundo de información que nos rodea, un mundo demasiado complejo para estudiarlo directamente por nosotros mismos. Los datos son el registro de todo lo que ocurre y lo que debemos aprender de ello. El verdadero valor de toda esta información es su significado. El aprendizaje automático nos ayuda a descubrir patrones en los datos, que es donde vive el significado. Cuando podemos ver el significado de los datos, podemos hacer predicciones sobre el futuro. En esta lección, exploraremos el aprendizaje automático con una técnica llamada "**K vecinos más próximos**" o "**K-Nearest Neighbors**". Utilizaremos un conjunto de datos de tarifas de alquiler de AirBnB para identificar tarifas similares en una zona para unidades de AirBnB que compiten entre sí y hacer predicciones sobre las tarifas ideales para maximizar los beneficios. *Tendrás que sentirte cómodo programando en Python, y tendrás que estar familiarizado con las librerías NumPy y pandas. Estos son algunos de los puntos que puedes esperar de esta lección:*

Los fundamentos del flujo de trabajo del aprendizaje automático:

- Cómo funciona el algoritmo K-Nearest Neighbors
- El papel de la distancia euclídea en el aprendizaje automático

Ahora, vamos a conocer nuestro conjunto de datos.

Introducción a los Datos

Aunque AirBnB no publica ningún dato sobre los anuncios de su mercado, un grupo independiente llamado Inside AirBnB ha extraído datos sobre una muestra de los anuncios de muchas de las principales ciudades del sitio web. En esta lección se hará uso de los datos extraídos pertenecientes a la ciudad de Washington.

Cada fila del conjunto de datos es un anuncio específico que estaba disponible para alquilar en AirBnB en la zona de Washington, D.C. el conjunto de datos se almacena en un archivo con extensión csv (valores separados por comas) con el nombre de dc_airbnb.csv. Podrás darle click para descargar y visualizar el archivo

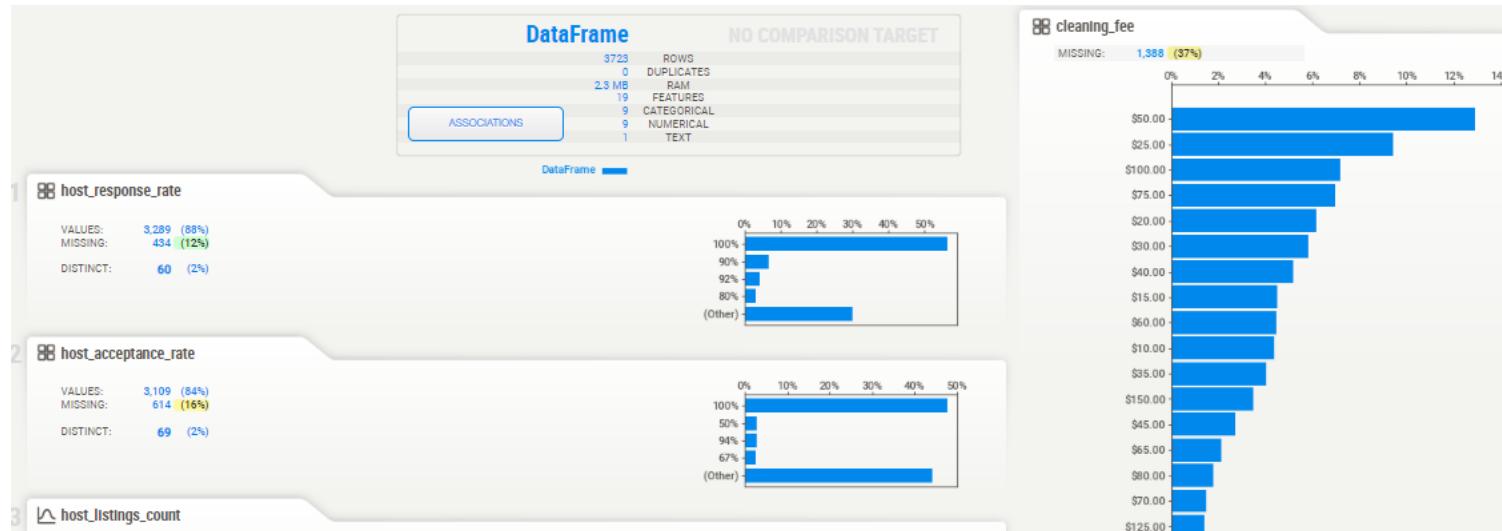
Introducción a los Datos

- **host_response_rate**: tasa de respuesta del host
- **host_acceptance_rate**: número de solicitudes al host que se convierten en rentas
- **host_listings_count**: número de otros listados del host
- **latitude**: latitud de las coordenadas geográficas
- **longitude**: longitud de las coordenadas geográficas
- **city**: la ciudad de la renta
- **zipcode**: código postal de la renta
- **state**: el estado de la renta
- **accommodates**: el número de invitados que la renta puede acomodar
- **room_type**: tipo de renta (Cuarto privado, compartido o casa/apartamento entero)
- **bedrooms**: número de cuartos incluidos en la renta
- **bathrooms**: número de baños incluidos en la renta
- **beds**: número de camas incluidas en la renta
- **price**: precio por noche para la renta
- **cleaning_fee**: tarifa adicional por limpiar la renta después de que el huésped se vaya
- **security_deposit**: depósito de garantía reembolsable, en caso de daños
- **minimum_nights**: número mínimo de noches que un huésped puede permanecer en la renta
- **maximum_nights**: número máximo de noches que un huésped puede permanecer en la renta
- **number_of_reviews**: número de reseñas que han dejado huéspedes anteriores

Introducción a los Datos

A continuación se presentara un análisis exploratorio de datos (EDA). En el cual se podrá visualizar de una manera mas detallada y con graficos los datos de cada columna como lo pueden llegar a ser, los datos del numero máximo y mínimo de una columna numérica, el valor que mas se repite en una columna, la cantidad de valores nulos y no nulos, entre otros. Se aconseja descargar el archivo desde el siguiente [link](#) y abrirlo desde el ordenador, con el objetivo de interactuar desde un navegador web.

Ejemplo de visualización:



Introducción a los Datos

Vamos a leer el conjunto de datos en Pandas y a familiarizarnos con él.

Instrucciones

1. En el editor de código de la derecha, escribe un código que haga lo siguiente :

- Leer **dc_airbnb.csv** en un DataFrame llamado **dc_listings**
- Utilice la función **print** para mostrar la primera fila de **dc_listings**

Soluciones

```
1 Import pandas as pd  
2 dc_listing = pd.read_csv( 'dc_airbnb.csv' )  
3 print (dc_listings. iloc [0] )
```

K-nearest Neighbors

Esta es la estrategia que queríamos utilizar:

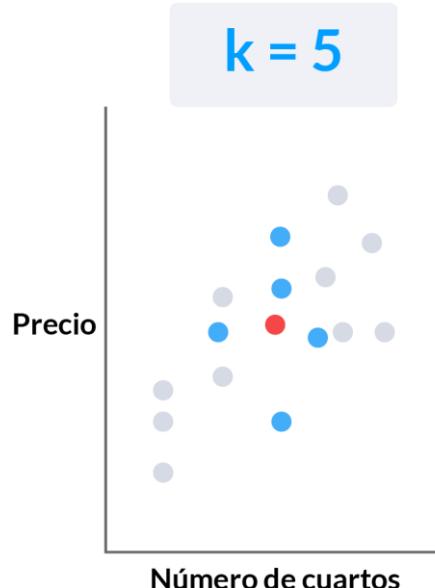
- Encontrar unos cuantos listados similares
- Calcular el precio medio de alquiler por noche de estos anuncios
- Establecer el precio medio como el precio de nuestro anuncio

El algoritmo de k-próximos es similar a esta estrategia. Aquí tiene una visión general:

K-nearest Neighbors

Paso 1

k= el número de listados similares con los que se quiere comparar



Paso 2

Por cada anuncio, calcule la similitud con nuestro anuncio sin precio

Cuartos	Precio
1	160
3	350
1	60
1	95
1	50

Más similar
0
↓
Menos similar

Cuartos	Precio
1	?

Figura 2. Pasos de K-Nearest Neighbors

K-nearest Neighbors

Paso 3

Clasificar cada listado según la métrica de similitud y seleccionar los primeros k listados

Cuartos	Precio	Similitud
1	160	0
1	60	0
1	95	0
1	50	0
3	350	2

Precio medio de la lista

Paso 4

Calcular el precio medio de venta de los k anuncios similares y utilizarlo como precio de venta

Cuartos	Precio
1	105

K-nearest Neighbors

Hay dos cosas que debemos desgranar con más detalle:

- La métrica de similitud
- Cómo elegir el valor de **k**

En esta lección, definiremos la métrica de similitud que vamos a utilizar. A continuación, implementaremos el algoritmo de **k** vecinos más cercanos y lo utilizaremos para sugerir un precio para un nuevo anuncio sin precio. En esta lección utilizaremos un valor k de **5**.

Distancia Euclíadiana

La métrica de similitud funciona **comparando un conjunto fijo de características numéricas (otra palabra para atributos) entre dos observaciones**, o espacios vitales en nuestro caso. Cuando se trata de predecir un valor continuo, como el precio, la principal métrica de similitud es la **distancia Euclíadiana**. Esta es la fórmula general de la distancia euclíadiana:

$$d = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2 + \dots + (q_n - p_n)^2}$$

Donde **q1** a **qn** representan los valores de las características de una observación y **p1** a **pn** representan los valores de las características de la otra observación. A continuación se muestra un diagrama que desglosa la distancia euclíadiana entre las dos primeras observaciones del conjunto de datos utilizando únicamente las columnas **host_listing_count, accommodates, bedrooms, bathrooms y bed**.

Distancia Euclídea

Diferencias \downarrow $(q_1 - p_1) + (q_2 - p_2) + \dots + (q_n - p_n)$

Diferencias al cuadrado \downarrow $(q_1 - p_1)^2 + (q_2 - p_2)^2 + \dots + (q_n - p_n)^2$

Distancia euclídea $= \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2 + \dots + (q_n - p_n)^2}$

index	host_listings_count	accommodates	bedrooms	bathrooms	beds
0	26	4	1	1	2
1	1	6	3	3	3

Diferencias \downarrow $(26 - 1) + (4 - 6) + (1 - 3) + (1 - 3) + (2 - 3)$

Diferencias al cuadrado \downarrow $(25)^2 + (-2)^2 + (-2)^2 + (-2)^2 + (-1)^2$

Distancia euclídea $= \sqrt{625 + 4 + 4 + 4 + 1}$

$= \sqrt{638}$

$= 25.258661$

Distancia Euclíadiana

En esta lección, utilizaremos sólo una característica para mantener las cosas simples mientras se familiariza con el flujo de trabajo del aprendizaje automático. Dado que sólo estamos utilizando una característica, este caso se denomina **caso univariante**. La fórmula para el caso univariante es:

$$d = \sqrt{(q_1 - p_1)^2}.$$

La raíz cuadrada y la potencia al cuadrado se cancelan, y la fórmula se simplifica a:

$$d = |q_1 - p_1|.$$

La vivienda que queremos alquilar tiene capacidad para tres personas. En primer lugar, calculemos la distancia entre la primera vivienda del conjunto de datos y la nuestra, utilizando únicamente la función de **accommodates**.

Distancia Euclídea

Instrucciones

1. Calcule la distancia euclídea entre nuestro espacio vital, que tiene capacidad para tres personas, y el primer espacio vital del DataFrame **dc_listings**
2. Asigna el resultado a **first_distance** y muestra el valor con la función **print**

Soluciones

```
1 import numpy as np
2 our_acc_value = 3
3 first_living_space_value = dc_listings.iloc[0]
4     ['accommodates']
5 first_distance = np.abs(first_living_space_value -
our_acc_value)
6 print(first_distance)
```

Calcular la Distancia para Todas las Observaciones

La distancia euclídea entre la primera fila del DataFrame `dc_listings` y nuestro propio espacio vital es **1**.

¿Cómo sabemos si esto es alto o bajo? Si nos fijamos en la propia ecuación de la distancia euclídea, el valor más bajo que podemos alcanzar es **0**. Esto sucede cuando el valor de la característica es exactamente el mismo para las dos observaciones que estamos comparando. Si $p_1=q_1$, entonces $d=|q_1-p_1|$, lo que da como resultado $d=0$. Cuanto más se acerque a **0** la distancia, más similares son los espacios vitales.

Si queremos calcular la distancia euclídea entre cada espacio habitable del conjunto de datos y un espacio habitable con capacidad para **8** personas, he aquí una vista previa de cómo sería.

our listing	dc_listings		
accommodates	index	accommodates	distance
8	0	4	$ 8 - 4 $
	1	6	$ 8 - 6 $
	2	1	$ 8 - 1 $
	3	2	$ 8 - 2 $

Calcular la Distancia para Todas las Observaciones

A continuación, podemos clasificar los espacios de vida existentes por valores de distancia ascendentes, el sustituto de la similitud.

Instrucciones

1. Calcule la distancia entre cada valor de la columna de **accommodates** de **dc_listings** y el valor **3**, que es el número de personas que aloja nuestro listado:
 - Utiliza el método **apply** para calcular el valor absoluto entre cada valor de los **accommodates** y **3**, y devuelve una nueva Serie que contiene los valores de la distancia
2. Asignar los valores de la distancia a la **columna** de la distancia
3. Utilice el método de la serie **value_counts** y la función de **print** para mostrar los recuentos de valores únicos para la columna de la distancia

Soluciones

```
1 new_listing = 3
2 dc_listings['distance'] =
3     dc_listings['accommodates'].apply(
4         lambda x: np.abs(x - new_listing)
5 )
5 print(dc_listings['distance'].value_counts())
```

Aleatoriedad y Clasificación

Parece que hay bastantes espacios habitables (461, para ser exactos) que pueden albergar a tres personas como la nuestra. Esto significa que los cinco "vecinos más cercanos" que seleccionemos después de la clasificación tendrán un valor de distancia de cero.

Si ordenamos por la columna de la **distance** y luego seleccionamos los 5 primeros espacios habitables, estaríamos sesgando el resultado a la ordenación del conjunto de datos.

En su lugar, vamos a ordenar el conjunto de datos de forma aleatoria y luego ordenamos el DataFrame por la columna de la **distance**. De este modo, todos los espacios vitales que albergan el mismo número de personas seguirán estando en la parte superior del DataFrame, pero estarán en orden aleatorio en las primeras 461 filas.

```
print(dc_listings[dc_listings["distance"] == 0]  
      ["accommodates"])
```

26	3
34	3
36	3
40	3
44	3
45	3
48	3
65	3
66	3
71	3
75	3
86	3
...	

Aleatoriedad y Clasificación

Instrucciones

1. Ordenar aleatoriamente las filas de **dc_listings**:
 - Use la función **np.random.permutation()** para devolver una matriz NumPy de valores de índice mezclados
 - Utilice el método DataFrame **loc[]** para devolver un nuevo DataFrame que contenga el orden aleatorio
 - Asigne el nuevo DataFrame de nuevo a **dc_listings**
2. Después de la aleatorización, ordenar **dc_listings** por **column** de distancia, y asignar de nuevo a **dc_listings**
3. Mostrar los 10 primeros valores de la columna de **price** mediante la función **print**

Soluciones

```
1 import numpy as np
2 np.random.seed(1)
3 dc_listings =
4 dc_listings.loc[np.random.permutation(len(dc_listings))]
5 dc_listings = dc_listings.sort_values('distance')
5 print(dc_listings.iloc[0:10]['price'])
```

Precio Promedio

Antes de poder seleccionar los cinco espacios vitales más similares y calcular el precio medio, debemos limpiar la columna de **price**.

En este momento, la columna de **price** contiene caracteres de coma (,) y signos de dólar y es una columna de texto en lugar de una columna numérica. Tenemos que eliminar estos valores y convertir toda la columna al tipo de datos **float**. Entonces, podremos calcular el precio medio.

Precio Promedio

Instrucciones

1. Elimine las comas (,) y el signo de dólar (\$) de la columna de **price**:
 - Utiliza el accesorio **str** para que podamos aplicar métodos de cadena a cada valor de la columna seguido del método de cadena **replace** para sustituir todos los caracteres de coma por el carácter vacío: **stripped_commas = dc_listings['price'].str.replace(',', '')**
 - Repita la operación para eliminar los caracteres del signo de dólar
2. Convierte el nuevo objeto Serie que contiene los valores limpiados al tipo de datos **float** y lo asigna de nuevo a la columna de **price** en **dc_listings**
3. Calcular la media de los cinco primeros valores de la columna de **price** y asignarla a **mean_price**
4. Utilice la función de **print** o la variable **inspector** que aparece a continuación para mostrar el **mean_price**

Soluciones

```
1 stripped_commas = dc_listings['price'].str.replace(',','')
2 stripped_dollars = stripped_commas.str.replace('$', '')
3 dc_listings['price'] = stripped_dollars.astype('float')
4 mean_price = dc_listings.iloc[0:5]['price'].mean()
5 print(mean_price)
```

Funciones de Predicción

¡Felicitaciones! ¡Acabas de hacer tu primera predicción! Basándonos en el precio medio de otros listados que alojan a tres personas, deberíamos cobrar **158,6** dólares por noche para que un huésped se aloje en nuestro espacio vital.

Escribamos una función más general que pueda sugerir el precio óptimo para otros valores de la columna **accommodates**.

El DataFrame **dc_listings** tiene información específica de nuestro espacio vital (por ejemplo, la columna **distance**).

Para ahorrar tiempo, hemos vuelto a poner el DataFrame **dc_listings** a cero y sólo hemos mantenido la limpieza de datos y la aleatorización que hicimos, ya que no eran exclusivas de la predicción que estábamos haciendo para nuestro espacio vital.

Funciones de Predicción

Instrucciones

1. Escriba una función llamada **predict_price** que pueda utilizar la técnica de aprendizaje automático k-nearest neighbors para calcular el precio sugerido para cualquier valor de los **accommodates**. Esta función debería hacer lo siguiente:
 - Toma un único parámetro, **new_listing**, que describe el número de habitaciones
 - (Hemos añadido código que asigna **dc_listings** a un nuevo DataFrame llamado **temp_df**. Hemos utilizado el método **pandas.DataFrame.copy()**, para que el DataFrame subyacente se asigne a **temp_df**, en lugar de ser sólo una referencia a **dc_listings**)
 - Calcular la distancia entre cada valor de la columna **accommodates** y el valor de **new_listing** que se pasó. Asignar el objeto Serie resultante a la columna **distance** en **temp_df**
 - Ordene **temp_df** por la columna **distance** y seleccione los cinco primeros valores de la columna **price**. No ordene **temp_df** de forma aleatoria
 - Calcule la media de estos cinco valores y utilícela como valor de retorno para toda la función **predict_price**

Funciones de Predicción

Instrucciones

2. Utilice la función **predict_price** para sugerir un precio para un espacio vital que haga lo siguiente:
 - Si tiene capacidad para **1** persona, asigna el precio sugerido a **acc_one**
 - Si tiene capacidad para **2** personas, asigna el precio sugerido a **acc_two**
 - Si tiene capacidad para **4** personas, asigne el precio sugerido a **acc_four**

Funciones de Predicción

Soluciones

```
1 # Brought along the changes we made to the `dc_listings`  
2 # Dataframe.  
3 dc_listings = pd.read_csv('dc_airbnb.csv')  
4 stripped_commas = dc_listings['price'].str.replace(',', '')  
5 stripped_dollars = stripped_commas.str.replace('$', '')  
6 dc_listings['price'] = stripped_dollars.astype('float')  
7 dc_listings =  
8 dc_listings.loc[np.random.permutation(len(dc_listings))]  
9  
10 def predict_price(new_listing):  
11     temp_df = dc_listings.copy()  
12     ## Complete the function.  
13     return(new_listing)  
14  
15 acc_one = predict_price(1)  
16 acc_two = predict_price(2)  
17 acc_four = predict_price(4)  
18  
19 def predict_price(new_listing):  
20     temp_df = dc_listings.copy()  
21     temp_df['distance'] =  
22     temp_df['accommodates'].apply(lambda x: np.abs(x -  
23                                     new_listing))  
24     temp_df = temp_df.sort_values('distance')  
25     nearest_neighbors = temp_df.iloc[0:5]['price']  
26     predicted_price = nearest_neighbors.mean()  
27     return(predicted_price)  
28  
29 acc_one = predict_price(1)  
30 acc_two = predict_price(2)  
31 acc_four = predict_price(4)  
32  
33 print(acc_one)  
34 print(acc_two)  
35 print(acc_four)
```

Funciones de Predicción

- ✓ En esta lección, exploramos el problema de predecir el precio óptimo de un anuncio de alquiler de AirBnB basándonos en el precio de anuncios similares en el sitio. Hemos trabajado en todo el flujo de trabajo del aprendizaje automático, desde la selección de una característica hasta la prueba del modelo. Para explorar los fundamentos del aprendizaje automático, nos limitamos a utilizar una sola característica (el caso univariante) y un valor k fijo de 5
- ✓ En la próxima lección, aprenderemos a evaluar el rendimiento de un modelo

I.3 Evaluación del Rendimiento del Modelo



Comprobando la Calidad de las Predicciones

Ahora tenemos una función que puede predecir el precio de cualquier espacio habitable que queramos listar siempre que sepamos el número de personas que puede alojar. La función que escribimos representa un **predicted_price**, lo que significa que emite una predicción basada en la entrada del modelo.

Una forma sencilla de comprobar la calidad del modelo es:

- Dividir el conjunto de datos en 2 particiones:
 - El conjunto de entrenamiento: contiene la mayoría de las filas (75%)
 - El conjunto de prueba: contiene la minoría restante de las filas (25%)
- Utilizar las filas del conjunto de entrenamiento para predecir el valor del precio de las filas del conjunto de prueba
 - Añada una nueva columna denominada **predicted_price** al conjunto de prueba
- Compare los **predicted_price** con los valores de **price** reales del conjunto de prueba para ver la precisión de los valores predichos

Comprobando la Calidad de las Predicciones

Este proceso de validación, en el que utilizamos el conjunto de entrenamiento para hacer predicciones y el conjunto de prueba para predecir valores, se conoce como **validación de entrenamiento/prueba**. Siempre que realices aprendizaje automático, querrás realizar algún tipo de validación para asegurarte de que tu modelo de aprendizaje automático puede hacer buenas predicciones con nuevos datos. Aunque la validación de entrenamiento/prueba no es perfecta, la utilizaremos para entender el proceso de validación, para seleccionar una métrica de error, y luego nos sumergiremos en un proceso de validación más robusto más adelante en este curso.

Modifiquemos la función **predict_price** para utilizar sólo las filas del conjunto de entrenamiento, en lugar del conjunto de datos completo, para encontrar los vecinos más cercanos, promediar los valores de **price** de esas filas y devolver el valor del precio predicho. A continuación, utilizaremos esta función para predecir el precio de las filas del conjunto de prueba. Una vez que tengamos los valores de los precios predichos, podremos compararlos con los valores de los precios reales y empezar a entender la eficacia del modelo en la siguiente pantalla.

Comprobando la Calidad de las Predicciones

Para empezar, hemos asignado el primer 75% de las filas de `dc_listings` a `train_df` y el último 25% de las filas a `test_df`. Aquí hay un diagrama que explica la división:

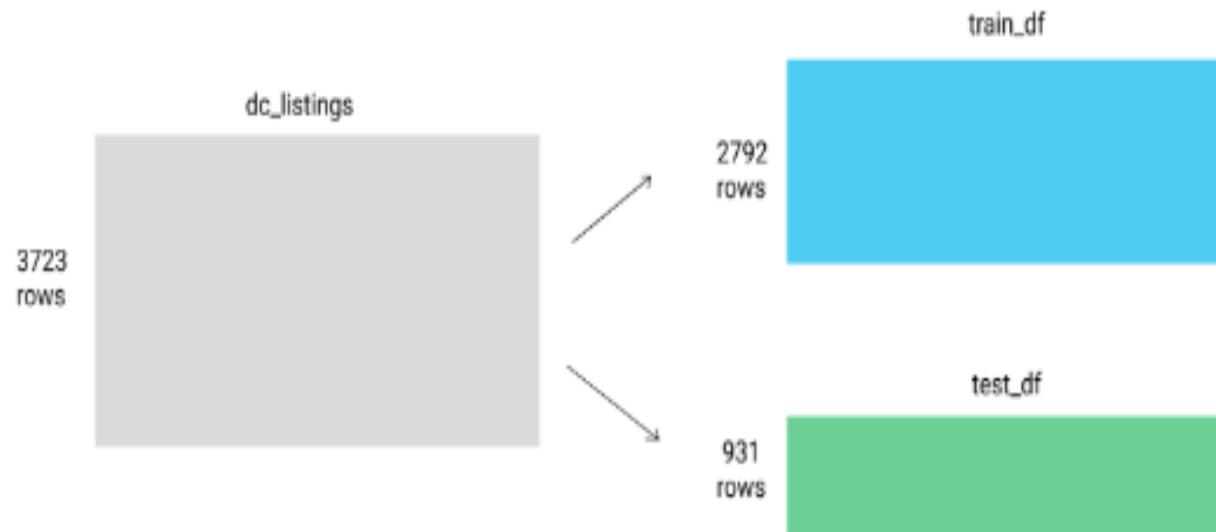


Figura 3. Diagrama de descomposición del modelo de entrenamiento y de prueba

Comprobando la Calidad de las Predicciones

Instrucciones

- Dentro de la función `predict_price`, cambie el Dataframe al que se asigna `temp_df`. Cámbielo de `dc_listings` a, `train_df` para que sólo se utilice el conjunto de entrenamiento
- Utilice el método Series apply para pasar todos los valores de la columna `accommodates` de `test_df` a través de la función `predict_price`
- Asigne el objeto Series resultante a la columna `predicted_price` de `test_df`

Comprobando la Calidad de las Predicciones

Soluciones

```
1 import pandas as pd
2 import numpy as np
3 dc_listings = pd.read_csv("dc_airbnb.csv")
4 stripped_commas = dc_listings['price'].str.replace(',', '')
5 stripped_dollars = stripped_commas.str.replace('$', '')
6 dc_listings['price'] = stripped_dollars.astype('float')
7 train_df = dc_listings.iloc[0:2792]
8 test_df = dc_listings.iloc[2792:]
9
10 def predict_price(new_listing):
11     ## DataFrame.copy() performs a deep copy
12     temp_df = dc_listings.copy()
```

```
13     temp_df['distance'] =
14         temp_df['accommodates'].apply(lambda x: np.abs(x -
15             new_listing))
16         temp_df = temp_df.sort_values('distance')
17         nearest_neighbor_prices = temp_df.iloc[0:5]['price']
18         predicted_price = nearest_neighbor_prices.mean()
19         return(predicted_price)
20
21     def predict_price(new_listing):
22         temp_df = train_df.copy()
23         temp_df['distance'] =
24             temp_df['accommodates'].apply(lambda x: np.abs(x -
25                 new_listing))
26         temp_df = temp_df.sort_values('distance')
27         nearest_neighbor_prices = temp_df.iloc[0:5]['price']
28         predicted_price = nearest_neighbor_prices.mean()
29         return(predicted_price)
30
31
32 test_df['predicted_price'] =
33 test_df['accommodates'].apply(predict_price)
```

Métricas de Error

Ahora necesitamos una métrica que cuantifique la calidad de las predicciones en el conjunto de pruebas. Esta clase de métrica se denomina métrica de error. Como su nombre indica, una métrica de error cuantifica la inexactitud de nuestras predicciones en comparación con los valores reales. En nuestro caso, la métrica de error nos indica la diferencia entre los valores de **predicted_price** y los valores reales de los espacios habitables en el conjunto de datos de prueba.

Podríamos empezar calculando la diferencia entre cada valor predicho y el real y luego promediando estas diferencias. Esto se conoce como error medio, pero no es una métrica de error eficaz para la mayoría de los casos. El error medio trata una diferencia positiva de forma diferente a una diferencia negativa, pero lo que realmente nos interesa es lo lejos que está la predicción en la dirección positiva o negativa. Si el precio real era de 200 dólares y el modelo predijo 210 o 190, se aleja 10 dólares en cualquier caso.

En su lugar, podemos utilizar el error medio absoluto, donde calculamos el valor absoluto de cada error antes de promediar todos los errores.

$$MAE = \frac{1}{n} \sum_{k=1}^n |(actual_1 - predicted_1)| + \dots + |(actual_n - predicted_n)|$$

Métricas de Error

Instrucciones

- Utiliza **numpy.absolute()** para calcular el error medio absoluto entre **predicted_price** y el **price**
- Asigna el MAE a **mae**

Soluciones

```
1 import numpy as np
2 test_df['error'] =
3     np.absolute(test_df['predicted_price'] -
4         test_df['price'])
5 mae = test_df['error'].mean()
6 print(mae)
```

Error Cuadrático Medio

Para muchas tareas de predicción, queremos penalizar los valores predichos que están más lejos del valor real mucho más que los que están más cerca del valor real.

En su lugar, podemos tomar la media de los valores de error al cuadrado, lo que se denomina error medio al cuadrado o MSE para abreviar. El MSE aclara la diferencia entre los valores predichos y los reales. Una predicción que se desvía en 100 dólares tendrá un error (de 10.000) que es 100 veces mayor que una predicción que se desvía en sólo 10 dólares (que tendrá un error de 100).

Esta es la fórmula del MSE:

$$MSE = \frac{1}{n} \sum_{k=1}^n (actual_1 - predicted_1)^2 + \dots + (actual_n - predicted_n)^2$$

Donde **n** representa el número de filas del conjunto de prueba. Calculemos el valor de MSE para las predicciones que hicimos en el conjunto de prueba.

Error Cuadrático Medio

Instrucciones

- Calcule el valor de MSE entre las columnas **predicted_price** y **price** y asigne a **mse**

Soluciones:

```
1 test_df['squared_error'] = (test_df['predicted_price'] -  
2   test_df['price'])**2  
3 mse = test_df['squared_error'].mean()  
3 print(mse)
```

Entrenamiento de Otro Modelo

El modelo que hemos entrenado alcanzó un error cuadrático medio de alrededor de **18646,5** ¿Se trata de un valor de error cuadrático medio alto o bajo? ¿Qué nos dice esto sobre la calidad de las predicciones y del modelo? Por sí mismo, el valor del error cuadrático medio de un solo modelo no es muy útil.

Las unidades del error medio al cuadrado en nuestro caso son dólares al cuadrado (no dólares), lo que hace que también sea difícil de razonar intuitivamente. Sin embargo, podemos entrenar otro modelo y luego comparar los valores del error medio al cuadrado para ver qué modelo funciona mejor en términos relativos. Recordemos que una métrica de error baja significa que la diferencia entre el precio de lista predicho y los valores de precio de lista reales es baja, mientras que una métrica de error alta significa que la diferencia es alta.

Entrenemos otro modelo, esta vez utilizando la columna de **bathrooms**, y comparemos los valores de MSE.

Entrenamiento de Otro Modelo

Instrucciones

- Modificar la función `predict_price` a la derecha para utilizar la columna baños en lugar de la columna acomodados para hacer las predicciones
- Aplique la función a `test_df` y asigne el objeto Series resultante que contiene los valores de `predicted_price` a la columna `predicted_price` de `test_df`
- Calcule el error al cuadrado entre las columnas `price` y `predicted_price` en `test_df` y asigne el objeto Serie resultante a la columna `squared_error` en `test_df`
- Calcule la media de la columna `squared_error` en `test_df` y asígnela a `mse`
- Utilice la función `print` o el inspector de variables para mostrar el valor de MSE

Entrenamiento de Otro Modelo

Soluciones

```
1 train_df = dc_listings.iloc[0:2792]
2 test_df = dc_listings.iloc[2792:]
3
4 def predict_price(new_listing):
5     temp_df = train_df.copy()
6     temp_df['distance'] =
7         temp_df['accommodates'].apply(lambda x: np.abs(x -
8             new_listing))
9     temp_df = temp_df.sort_values('distance')
10    nearest_neighbors_prices = temp_df.iloc[0:5]
11    ['price']
12    predicted_price = nearest_neighbors_prices.mean()
13    return(predicted_price)
14
15 def predict_price(new_listing):
16     temp_df = train_df.copy()
17     temp_df['distance'] =
18         temp_df['bathrooms'].apply(lambda x: np.abs(x -
19             new_listing))
20     temp_df = temp_df.sort_values('distance')
21     nearest_neighbors_prices = temp_df.iloc[0:5]
22     ['price']
23     predicted_price = nearest_neighbors_prices.mean()
24     return(predicted_price)
25
26 test_df['predicted_price'] =
27     test_df['bathrooms'].apply(lambda x: predict_price(x))
28 test_df['squared_error'] = (test_df['predicted_price'] -
29     test_df['price'])**2
30 mse = test_df['squared_error'].mean()
31 print(mse)
```

Raíz del Error Cuadrático Medio

Aunque la comparación de los valores de MSE nos ayuda a identificar qué modelo rinde más en términos relativos, no nos ayuda a entender si el rendimiento es lo suficientemente bueno en general. Esto se debe a que las unidades de la métrica MSE son al cuadrado (en este caso, dólares al cuadrado). Un valor de MSE de 16377,5 dólares al cuadrado no nos da una idea intuitiva de lo alejadas que están las predicciones del modelo del valor real del precio en dólares.

La raíz del error cuadrático medio es una métrica de error cuyas unidades son la unidad base (en nuestro caso, los dólares). RMSE para abreviar, esta métrica de error se calcula tomando la raíz cuadrada del valor MSE.

$$RMSE = \sqrt{MSE}$$

Dado que el valor del RMSE utiliza las mismas unidades que la columna del objetivo, podemos entender la distancia en dólares reales que podemos esperar que tenga el modelo.

Calculemos el valor del RMSE del modelo que hemos entrenado utilizando la columna de los **bathrooms**.

Raíz del Error Cuadrático Medio

Instrucciones

- Calcular el valor del RMSE del modelo que hemos entrenado utilizando la columna de **bathrooms** y asignarlo a **rmse**

Soluciones

```
1 def predict_price(new_listing):
2     temp_df = train_df.copy()
3     temp_df['distance'] =
4         temp_df['bathrooms'].apply(lambda x: np.abs(x -
5             new_listing))
6     temp_df = temp_df.sort_values('distance')
7     nearest_neighbors_prices = temp_df.iloc[0:5]
8     ['price']
9     predicted_price = nearest_neighbors_prices.mean()
10    return(predicted_price)
11
12 test_df['predicted_price'] =
13     test_df['bathrooms'].apply(lambda x: predict_price(x))
14 test_df['squared_error'] = (test_df['predicted_price'] -
15     test_df['price'])**2
16 mse = test_df['squared_error'].mean()
17 rmse = mse ** (1/2)
18 print(rmse)
```

Comparación del MAE y el RMSE

El modelo alcanzó un valor de RMSE de aproximadamente **135,6**, lo que implica que debemos esperar que el modelo se equivoque en **135,6** dólares de media para los valores de precio predichos. Dado que la mayoría de las viviendas se cotizan a unos pocos cientos de dólares, debemos reducir este error al máximo para mejorar la utilidad del modelo.

Hemos hablado de algunas métricas de error diferentes que podemos utilizar para entender el rendimiento de un modelo. Como mencionamos anteriormente, estas métricas de error individuales son útiles para comparar modelos. Para entender mejor un modelo específico, podemos comparar múltiples métricas de error para el mismo modelo. Esto requiere una mejor comprensión de las propiedades matemáticas de las métricas de error.

Si se observa la ecuación de MAE:

$$= \frac{1}{n} \sum_{k=1}^n |(actual_1 - predicted_1)| + \dots + |(actual_n - predicted_n)|$$

Comparación del MAE y el RMSE

Observará que las diferencias entre los valores predichos y los reales crecen linealmente. Una predicción que se desvía en 10 dólares tiene un error 10 veces mayor que una predicción que se desvía en 1 dólar. Sin embargo, si observamos la ecuación del RMSE:

$$= \sqrt{\frac{\sum_{k=1}^n (actual_k - predicted_k)^2 + \dots + (actual_n - predicted_n)^2}{n}}$$

Observará que cada error se eleva al cuadrado antes de tomar la raíz cuadrada de la suma de todos los errores. Esto significa que los errores individuales crecen cuadráticamente y tienen un efecto diferente en el valor final del RMSE.

Veamos un ejemplo utilizando datos totalmente diferentes. Hemos creado 2 objetos Serie que contienen 2 conjuntos de errores y los hemos asignado a **errors_one** and **errors_two**.

Comparación del MAE y el RMSE

Instrucciones

- Calcular el MAE para `errors_one` y asignarlo a `mae_one`
- Calcular el RMSE para `errors_one` y asignarlo a `rmse_one`
- Calcular el MAE para `errors_two` y asignarlo a `mae_two`
- Calcular el RMSE para `errors_two` y asignarlo a `rmse_two`

Soluciones

```
1 errors_one = pd.Series([5, 10, 5, 10, 5, 10, 5, 10, 5,
2                           10, 5, 10, 5, 10, 5, 10, 5, 10])
3 errors_two = pd.Series([5, 10, 5, 10, 5, 10, 5, 10, 5,
4                           10, 5, 10, 5, 10, 5, 10, 5, 1000])
5 mae_one = errors_one.sum()/len(errors_one)
6 rmse_one =
7   np.sqrt((errors_one**2).sum()/len(errors_one))
8 print(mae_one)
9 print(rmse_one)
10
11 mae_two = errors_two.sum()/len(errors_two)
12 rmse_two =
13   np.sqrt((errors_two**2).sum()/len(errors_two))
14 print(mae_two)
15 print(rmse_two)
```

Comparación del MAE y el RMSE

Mientras que la relación entre el MAE (7,5) y el RMSE (7,9056941504209481) fue de aproximadamente 1:1 para la primera lista de errores, la relación entre el MAE (62,5) y el RMSE (235,82302686548658) fue más cercana a 1:4 para la segunda lista de errores. En general, cabe esperar que el valor MAE sea mucho menor que el valor RMSE. La única diferencia entre los dos conjuntos de errores es el valor extremo de **1000** en **errors_two** en lugar de **10**. Cuando trabajamos con conjuntos de datos más grandes, no podemos inspeccionar cada valor para entender si hay uno o algunos valores atípicos o si todos los errores son sistemáticamente más altos. Observar la relación entre el MAE y el RMSE puede ayudarnos a entender si hay errores grandes pero poco frecuentes. Puedes leer más sobre la comparación de MAE y RMSE en este maravilloso post:

<https://medium.com/human-in-a-machine-world/mae-and-rmse-which-metric-is-better-e60ac3bde13d#.lyc8od1ix>

En esta misión, hemos aprendido a probar nuestros modelos de aprendizaje automático utilizando la validación cruzada básica y diferentes métricas. En las próximas 2 misiones, exploraremos cómo añadir más características al modelo de aprendizaje automático y seleccionar un valor de **k** más óptimo puede ayudar a mejorar el rendimiento del modelo.

I.4 Multivariante del Método K-Nearest Neighbors



Recapitulemos

En la última misión, exploramos cómo usar un modelo de aprendizaje automático k-nearest neighbors simple que usó solo una característica, o atributo, de la lista para predecir el precio de renta. Primero nos basamos en la columna de **accommodates**, que describe el número de personas que un espacio habitable puede acomodar cómodamente. Luego, cambiamos a la columna de **bathrooms** y observamos una mejora en la precisión. Si bien estas fueron buenas características para familiarizarse con los conceptos básicos del aprendizaje automático, está claro que usar una sola característica para comparar listados no refleja la realidad del mercado. Un apartamento que puede acomodar a 4 personas en una parte popular de Washington D.C. se alquila por mucho más alto que uno que puede acomodar a 4 personas en un área plagada de delitos.

Hay 2 formas en las que podemos retocar el modelo para tratar de mejorar la precisión (mejorar el RMSE durante la validación):

- Incrementar el número de atributos que el modelo usa para calcular similitudes cuando ranquea los vecinos más cercanos
- Incrementar **k**, el número de vecinos cercanos que el modelo usa cuando calcula la predicción

Recapitulemos

En esta misión, nos centraremos en aumentar el número de atributos que utiliza el modelo. Al seleccionar más atributos para utilizar en el modelo, tenemos que tener cuidado con las columnas que no funcionan bien con la ecuación de la distancia. Esto incluye las columnas que contienen:

- Valores no numéricos (por ejemplo, ciudad o estado)
 - La ecuación de distancia euclídea espera valores numéricos
- Valores perdidos
 - La ecuación de la distancia espera un valor para cada observación y atributo
- Valores no ordinales (por ejemplo, latitud o longitud)
 - La clasificación por distancia euclídea no tiene sentido si todos los atributos no son ordinales

En la siguiente pantalla de código, hemos leído el conjunto de datos **dc_airbnb.csv** de la última misión en pandas y hemos traído los cambios de limpieza de datos que hicimos. Vamos a ver primero los valores de la primera fila para identificar cualquier columna que contenga valores no numéricos o no ordinales. En la siguiente pantalla, eliminaremos esas columnas y luego buscaremos los valores que faltan en cada una de las columnas restantes.

Recapitulemos

Instrucciones

- Utilice el método DataFrame.info() para devolver el número de valores no nulos en cada columna

Soluciones

```
1 import pandas as pd
2 import numpy as np
3 np.random.seed(1)
4
5 dc_listings = pd.read_csv('dc_airbnb.csv')
6 dc_listings =
7 dc_listings.loc[np.random.permutation(len(dc_listings))]
8 stripped_commas = dc_listings['price'].str.replace(',',
9 ''')
10 stripped_dollars = stripped_commas.str.replace('$', '')
11 dc_listings['price'] = stripped_dollars.astype('float')
12 print(dc_listings.info())
```

Eliminación de Características

Las siguientes columnas contienen valores no numéricos:

- **room_type**: e.g. **private_room**
- **city**: e.g. **Washington**
- **state**: e.g. **DC**

Mientras que estas columnas contienen valores numéricos pero no ordinales:

- **latitude**: e.g. 38.913458
- **longitude**: e.g. -77.031
- **zipcode**: e.g. 20009

Eliminación de Características

Los valores geográficos de este tipo no son ordinales, porque un valor numérico menor no se corresponde directamente con un valor menor de forma significativa. Por ejemplo, el código postal 20009 no es más pequeño ni más grande que el código postal 75023, sino que ambos son valores únicos e identificadores. Los pares de valores de latitud y longitud describen un punto en un sistema de coordenadas geográficas y en esos casos se utilizan ecuaciones diferentes (por ejemplo, el [haverseno](#)).

Aunque podríamos convertir las columnas **host_response_rate** y **host_acceptance_rate** para que fueran numéricas (ahora mismo son tipos de datos de objeto y contienen el signo %), estas columnas describen al anfitrión y no al espacio vital en sí. Puesto que un anfitrión puede tener muchos espacios vitales y no tenemos suficiente información para agrupar de forma exclusiva los espacios vitales con los propios anfitriones, vamos a evitar el uso de cualquier columna que no describa directamente el espacio vital o el propio listado:

- **host_response_rate**
- **host_acceptance_rate**
- **host_listings_count**

Eliminación de Características

Vamos a eliminar estas 9 columnas del Dataframe.

Instrucciones

- Elimine las 9 columnas de las que hablamos anteriormente de **dc_listings**:
 - 3 que contienen valores no numéricos
 - 3 que contienen valores numéricos pero no ordinales
 - 3 que describen al anfitrión en lugar del propio espacio vital

Soluciones

```
1 drop_columns = ['room_type', 'city', 'state',
  'latitude', 'longitude', 'zipcode',
  'host_response_rate', 'host_acceptance_rate',
  'host_listings_count']
2 dc_listings = dc_listings.drop(drop_columns, axis=1)
3 print(dc_listings.isnull().sum())
```

Manejo de los Valores Perdidos

De las columnas restantes, 3 tienen algunos valores perdidos (menos del 1% del número total de filas):

- **bedrooms**
- **bathrooms**
- **beds**

Como el número de filas que contienen valores perdidos para una de estas 3 columnas es bajo, podemos seleccionar y eliminar esas filas sin perder mucha información. También hay 2 columnas que tienen un gran número de valores perdidos:

- **cleaning_fee** - 37.3% de las columnas
- **security_deposit** - 61.7% de las filas

Y no podemos manejarlos fácilmente. No podemos simplemente eliminar las filas que contienen valores perdidos para estas 2 columnas porque perderíamos la mayoría de las observaciones en el conjunto de datos. En lugar de ello, vamos a eliminar estas dos columnas por completo.

Manejo de los Valores Perdidos

Instrucciones

- Elimine las columnas **cleaning_fee** y **security_deposit** de **dc_listings**
- A continuación, elimine de **dc_listings** todas las filas que contengan un valor ausente para la columnas **bedrooms**, **bathrooms** o **beds**
 - Para ello, utilice el **Dataframe method dropna()** del y establezca el parámetro del eje en **0**
 - Dado que sólo columnas **bedrooms**, **bathrooms** y **beds** contienen valores perdidos, se eliminarán las filas que contengan valores perdidos en estas columnas
- Muestre los recuentos de valores nulos para el Dataframe **dc_listings** actualizado para confirmar que no quedan valores perdidos

Manejo de los Valores Perdidos

Soluciones

```
1 dc_listings = dc_listings.drop(['cleaning_fee',  
    'security_deposit'], axis=1)  
2 dc_listings = dc_listings.dropna(axis=0)  
3 print(dc_listings.isnull().sum())
```

Normalización de Columnas

Así es como se ve el **dc_listings** Dataframe después de todos los cambios que hicimos:

accommodates	bedrooms	bathrooms	beds	price	minimum_nights	maximum_nights	number_of_reviews
2	1.0	1.0	1.0	125.0	1	4	149
2	1.0	1.5	1.0	85.0	1	30	49
1	1.0	0.5	1.0	50.0	1	1125	1
2	1.0	1.0	1.0	209.0	4	730	2
12	5.0	2.0	5.0	215.0	2	1825	34

Normalización de Columnas

Habrá observado que mientras las columnas de **accommodates**, **bedrooms**, **bathrooms**, **beds** y **minimum_nights** oscilan entre 0 y 12 (al menos en las primeras filas), los valores de las columnas de **maximun_nights** y **number_of_reviews** abarcan rangos mucho mayores. Por ejemplo, la columna de noches máximas tiene valores tan bajos como 4 y tan altos como 1825, en las primeras filas mismas. Si utilizamos estas dos columnas como parte de un modelo de k- vecinos más cercanos, estos atributos podrían acabar teniendo un efecto desmesurado en los cálculos de distancia, debido a la amplitud de los valores.

Por ejemplo, dos viviendas podrían ser idénticas en todos los atributos pero ser muy diferentes sólo en la columna de **maximun_nights**. Si uno de los listados tiene un valor de **maximun_nights** de 1825 y el otro un valor de **maximun_nights** de 4, debido a la forma en que se calcula la distancia euclídea, estos listados se considerarían muy alejados debido al gran efecto que tiene el tamaño de los valores en la distancia euclídea general. Para evitar que una sola columna tenga demasiado impacto en la distancia, podemos normalizar todas las columnas para que tengan una media de 0 y una desviación estándar de 1. Traducción realizada con la versión gratuita del traductor.

Normalización de Columnas

La normalización de los valores de cada columna a la distribución normal estándar (media de 0, desviación estándar de 1) preserva la distribución de los valores de cada columna a la vez que alinea las escalas. Para normalizar los valores de una columna a la distribución normal estándar, es necesario:

- Restar a cada valor la media de la columna
- Dividir cada valor por la desviación estándar de la columna

Esta es la fórmula matemática que describe la transformación que debe aplicarse a todos los valores de una columna: $x = \frac{x - \mu}{\sigma}$.

Donde x es un valor de una columna específica, μ es la media de todos los valores de la columna, y σ es la desviación estándar de todos los valores de la columna. Este es el aspecto del código correspondiente, utilizando pandas:

Normalización de Columnas

```
# Subtract each value in the column by the mean.  
first_transform = dc_listings['maximum_nights'] -  
dc_listings['maximum_nights'].mean()  
# Divide each value in the column by the standard  
deviation.  
normalized_col = first_transform / first_transform.std()
```

Hay que tener en cuenta que también se puede hacer lo siguiente:

```
normalized_col = first_transform /  
dc_listings['maximum_nights'].std()
```

Normalización de Columnas

Y se obtiene la misma respuesta que la anterior.

Esto se debe a que **first_transform** se limita a desplazar la media de la distribución y no tiene ningún efecto sobre la forma o la escala de la distribución. En otras palabras, la varianza de **dc_listings** es la misma que la varianza de **first_transform**.

Para aplicar esta transformación a todas las columnas de un Dataframe, puede utilizar los métodos correspondientes de Dataframe **mean()** y **std()**:

```
normalized_listings = (dc_listings - dc_listings.mean()) /  
(dc_listings.std())
```

Estos métodos fueron escritos teniendo en cuenta la transformación masiva de las columnas y cuando se llama a **mean()** o **std()**, se utilizan las medias y las desviaciones estándar de las columnas apropiadas para cada valor del Dataframe. Ahora vamos a normalizar todas las columnas de características en **dc_listings**.

Normalización de Columnas

Instrucciones

- Normalizar todas las columnas de características en `dc_listings` y asignar el nuevo Dataframe que contiene sólo las columnas de características normalizadas a `normalized_listings`
- Añadir la columna de `price` de `dc_listings` a `normalized_listings`
- Mostrar las 3 primeras filas de `normalized_listings`

Soluciones

```
1 normalized_listings = (dc_listings -  
dc_listings.mean())/(dc_listings.std())  
2 normalized_listings['price'] = dc_listings['price']  
3 print(normalized_listings.head(3))
```

Distancia Euclíadiana para el Caso Multivariante

En la última misión, entrenamos dos modelos univariantes de vecinos más cercanos. El primero utilizó el atributo de **accommodates** mientras que el segundo utilizó el atributo de **bathrooms**. Ahora vamos a entrenar un modelo que utilice ambos atributos para determinar la similitud de dos espacios habitables. Volvamos a consultar la ecuación de la distancia euclíadiana para ver cómo sería el cálculo de la distancia utilizando 2 atributos:

$$d = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2 + \cdots + (q_n - p_n)^2}$$

Como estamos utilizando 2 atributos, el cálculo de la distancia sería como:

$$d = \sqrt{(accommodates_1 - accommodates_2)^2 + (bathrooms_1 - bathrooms_2)^2}$$

Para encontrar la distancia entre 2 espacios habitables, tenemos que calcular la diferencia al cuadrado entre ambos valores de los **accommades**, la diferencia al cuadrado entre ambos valores de los **bathrooms**, sumarlos y luego sacar la raíz cuadrada de la suma resultante. Este es el aspecto de la distancia euclíadiana entre las 2 primeras filas de **normalized_listings**:

Distancia Euclíadiana para el Caso Multivariante

accommodates	bathrooms
-0.596544	-0.439151
-0.596544	0.412923

$$(q_1 - p_1) + (q_2 - p_2) + \dots + (q_n - p_n)$$

differences $(-0.596544 + 0.596544) + (-0.439151 - 0.412923)$

$$(q_1 - p_1)^2 + (q_2 - p_2)^2 + \dots + (q_n - p_n)^2$$

squared differences $(0)^2 + (-0.852074)^2$

$$\sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2 + \dots + (q_n - p_n)^2}$$

Euclidean distance
= $\sqrt{0 + 0.72603}$
= 0.852074

Distancia Euclíadiana para el Caso Multivariante

Hasta ahora, hemos calculado la distancia euclíadiana nosotros mismos escribiendo la lógica de la ecuación. En cambio, podemos utilizar la [función `distance.euclidean\(\)`](#) de `scipy.spatial`, que toma 2 vectores como parámetros y calcula la distancia euclíadiana entre ellos. La [función `euclidean\(\)`](#) espera:

- Que ambos vectores se representen con un objeto tipo lista (lista de Python, array de NumPy o serie de pandas)
- Ambos vectores deben ser unidimensionales y tener el mismo número de elementos

He aquí un ejemplo sencillo:

```
from scipy.spatial import distance
first_listing = [-0.596544, -0.439151]
second_listing = [-0.596544, 0.412923]
dist = distance.euclidean(first_listing, second_listing)
```

Utilicemos la función [`euclidean\(\)`](#) para calcular la distancia euclíadiana entre 2 filas de nuestro conjunto de datos para practicar.

Distancia Euclíadiana para el Caso Multivariante

Instrucciones

- Calcular la distancia euclíadiana utilizando sólo las características de los **accommodes** y los **bathrooms** entre la primera fila y la quinta fila en **normalized_listings** utilizando la función **distance.euclidean()**
- Asignar el valor de la distancia a **first_fifth_distance** y mostrarlo mediante la función **print**

Soluciones

```
1 from scipy.spatial import distance
2 first_listing = normalized_listings.iloc[0][['accommodes',
3 'bathrooms']]
4 fifth_listing = normalized_listings.iloc[4][['accommodes',
3 'bathrooms']]
5 first_fifth_distance = distance.euclidean(first_listing,
fifth_listing)
6 print(first_fifth_distance)
```

Introducción a Scikit-learn

Hasta ahora, hemos estado escribiendo funciones desde cero para entrenar los modelos de k- neighbors más cercanos. Si bien esto es una práctica deliberada útil para entender cómo funciona la mecánica, puedes ser más productivo e iterar más rápido utilizando una biblioteca que maneje la mayor parte de la implementación. En esta pantalla, aprenderemos sobre la [biblioteca scikit-learn \(scikit-learn library\)](#), que es la biblioteca de aprendizaje automático más popular en Python. Scikit-learn contiene funciones para todos los principales algoritmos de aprendizaje automático y un flujo de trabajo simple y unificado. Ambas propiedades permiten a los científicos de datos ser increíblemente productivos cuando entranan y prueban diferentes modelos en un nuevo conjunto de datos.

El flujo de trabajo de scikit-learn consta de 4 pasos principales:

- Instalar el modelo de aprendizaje automático específico que se desea utilizar
- Ajustar el modelo a los datos de entrenamiento
- Utilizar el modelo para hacer predicciones
- Evaluar la precisión de las predicciones

Introducción a Scikit-learn

Nos centraremos en los 3 primeros pasos en esta pantalla y en la siguiente. Cada modelo en scikit-learn se implementa como una clase independiente (separate class) y el primer paso es identificar la clase de la que queremos crear una instancia. En nuestro caso, queremos utilizar la clase KNeighborsRegressor.

Cualquier modelo que nos ayude a predecir valores numéricos, como el precio de venta en nuestro caso, se conoce como modelo de regresión. La otra clase principal de modelos de aprendizaje automático se denomina clasificación (**classification**), en la que intentamos predecir una etiqueta a partir de un conjunto fijo de etiquetas (por ejemplo, el tipo de sangre o el sexo). La palabra regressor del nombre de la clase **KNeighborsRegressor** se refiere a la clase del modelo de regresión que acabamos de discutir.

Scikit-learn utiliza un estilo orientado a objetos similar al de Matplotlib y es necesario instanciar primero un modelo vacío llamando al constructor.

```
from sklearn.neighbors import KNeighborsRegressor  
knn = KNeighborsRegressor()
```

Introducción a Scikit-learn

Si consultas la documentación([documentation](#)), te darás cuenta de que por defecto:

- **n_neighbors**: el número de vecinos, se establece en 5
- **algorithm**: para calcular los vecinos más cercanos, se establece en auto
- **p**: se establece en 2, correspondiente a la distancia euclídea

Pongamos el parámetro del **algorithm** en bruto(**brute**) y dejemos el valor de n_vecinos (**n_neighbors**) como 5, que coincide con la implementación que escribimos en la última misión. Si dejamos el parámetro del **algorithm** establecido en el valor por defecto de **auto**, scikit-learn tratará de utilizar las optimizaciones basadas en el árbol para mejorar el rendimiento (que están fuera del alcance de esta misión):

```
knn = KNeighborsRegressor(algorithm='brute')
```

Ajuste de un Modelo y Realización de Predicciones

Ahora, podemos ajustar el modelo a los datos utilizando el método de ajuste (`fit_method`). Para todos los modelos, el **método de ajuste (fit)** toma 2 parámetros necesarios:

- Objeto tipo matriz, que contiene las columnas de características que queremos utilizar del conjunto de entrenamiento
- Objeto tipo lista, que contiene los valores objetivo correctos

El objeto tipo matriz significa que el método es flexible en la entrada y se acepta un Dataframe o un array de valores NumPy 2D. Esto significa que puedes seleccionar las columnas que quieras utilizar del Dataframe y utilizarlo como primer parámetro del **método de ajuste (fit)**.

Si recuerdas que anteriormente en la misión, todos los siguientes son objetos tipo lista aceptables:

- Matriz NumPy
- Lista de Python
- Objeto Pandas Series (por ejemplo, al seleccionar una columna)

Ajuste de un Modelo y Realización de Predicciones

Puede seleccionar la columna objetivo del marco de datos y utilizarla como segundo parámetro del método de ajuste (**fit**):

```
# Split full dataset into train and test sets.  
train_df = normalized_listings.iloc[0:2792]  
test_df = normalized_listings.iloc[2792:]  
# Matrix-like object, containing just the 2 columns of interest from  
# training set.  
train_features = train_df[['accommodates', 'bathrooms']]  
# List-like object, containing just the target column, `price`.  
train_target = train_df['price']  
# Pass everything into the fit method.  
knn.fit(train_features, train_target)
```

Cuando se llama al método **fit()**, scikit-learn almacena los datos de entrenamiento que especificamos dentro de la instancia KNearestNeighbors (**knn**). Si intenta pasar datos que contengan valores perdidos o valores no numéricos en el método `fit`, scikit-learn devolverá un error. Scikit-learn contiene muchas características de este tipo que nos ayudan a prevenir errores comunes.

Ajuste de un Modelo y Realización de Predicciones

Ahora que hemos especificado los datos de entrenamiento que queremos usar para hacer predicciones, podemos usar el método predecir ([predict method](#)) para hacer predicciones en el conjunto de prueba. El método **predict** sólo tiene un parámetro requerido:

- Un objeto tipo matriz, que contiene las columnas de características del conjunto de datos sobre el que queremos hacer predicciones

El número de columnas de características que se utiliza durante el entrenamiento y la prueba tiene que coincidir o scikit-learn devolverá un error:

```
predictions = knn.predict(test_df[['accommodates', 'bathrooms']])
```

El método **predict()** devuelve un array NumPy que contiene los valores de precio predichos para el conjunto de prueba. Ahora tienes todo lo que necesitas para practicar todo el flujo de trabajo de scikit-learn.

Ajuste de un Modelo y Realización de Predicciones

Instrucciones

- Cree una instancia de la clase **KNeighborsRegressor** con los siguientes parámetros:
 - **n_neighbors: 5**
 - **algoritmo: brute**
- Utiliza el método **fit** para especificar los datos que queremos que utilice el modelo k-nearest neighbor. Utiliza los siguientes parámetros:
 - Datos de entrenamiento, columnas de características: sólo las columnas de **accommodates** y **bathrooms**, en ese orden, de **train_df**
 - Datos de entrenamiento, columna objetivo: la columna de precios de **train_df**
- Llame al método **predict** para hacer predicciones sobre:
 - Las columnas de **accommodates** y **bathrooms** de **test_df**
 - Asignar la matriz NumPy resultante de los valores de precio predichos a las predicciones (**predictions**)

Ajuste de un Modelo y Realización de Predicciones

Soluciones

```
1 from sklearn.neighbors import KNeighborsRegressor  
2  
3 train_df = normalized_listings.iloc[0:2792]  
4 test_df = normalized_listings.iloc[2792:]  
5 train_columns = ['accommodates', 'bathrooms']  
6  
7 # Instantiate ML model.  
8 knn = KNeighborsRegressor(n_neighbors=5, algorithm='brute')  
9  
10 # Fit model to data.  
11 knn.fit(train_df[train_columns], train_df['price'])  
12  
13 # Use model to make predictions.  
14 predictions = knn.predict(test_df[train_columns])
```

Cálculo del MSE con Scikit-Learn

Anteriormente en esta misión, hemos calculado los valores MSE y RMSE utilizando los operadores aritméticos de pandas para comparar cada valor predicho con el valor real de la columna del precio (`price`) de nuestro conjunto de pruebas. Como alternativa, podemos utilizar la [función `sklearn.metrics.mean_squared_error\(\)`](#). Una vez que se familiarice con los diferentes conceptos de aprendizaje automático, la unificación de su flujo de trabajo utilizando scikit-learn le ayudará a ahorrar mucho tiempo y evitar errores.

La función `mean_squared_error()` toma 2 entradas:

- Objeto tipo lista, que representa los valores verdaderos
- Objeto tipo lista, que representa los valores predichos usando el modelo

Para esta función, no mostraremos ningún código de ejemplo y dejaremos que usted entienda la función [desde la propia documentación](#) para calcular los valores MSE y RMSE para las predicciones que acabamos de hacer.

Cálculo del MSE con Scikit-Learn

Instrucciones

- Utilice la función `mean_squared_error` para calcular el valor MSE de las predicciones que hemos realizado en la pantalla anterior
- Asigne el valor MSE a `two_features_mse`
- Calcule el valor RMSE tomando la raíz cuadrada del valor MSE y asígnelo a `two_features_rmse`
- Mostrar ambas puntuaciones de error utilizando la función de impresión(`print`)

Cálculo del MSE con Scikit-Learn

Soluciones

```
1 from sklearn.metrics import mean_squared_error
2
3 train_columns = ['accommodates', 'bathrooms']
4 knn = KNeighborsRegressor(n_neighbors=5, algorithm='brute',
5 metric='euclidean')
6 knn.fit(train_df[train_columns], train_df['price'])
7 predictions = knn.predict(test_df[train_columns])
8 from sklearn.metrics import mean_squared_error
9
10 two_features_mse = mean_squared_error(test_df['price'],
11 predictions)
12 two_features_rmse = two_features_mse ** (1/2)
13 print(two_features_mse)
```

Utilización de más Funciones

Aquí hay una tabla que compara los valores de MSE y RMSE para los 2 modelos univariantes de la última misión y el modelo multivariante que acabamos de entrenar:

feature(s)	MSE	RMSE
accommodates	18646.5	136.6
bathrooms	17333.4	131.7
accommodates, bathrooms	15660.4	125.1

Como puedes ver, el modelo que entrenamos utilizando ambas características acabó funcionando mejor (menor puntuación de error) que cualquiera de los modelos univariantes de la última misión. Ahora vamos a entrenar un modelo utilizando las siguientes 4 características:

- **accommodates**
- **bedrooms**
- **bathrooms**
- **number_of_reviews**

Scikit-learn hace que sea increíblemente fácil intercambiar las columnas utilizadas durante el entrenamiento y las pruebas. Vamos a dejar esto como un reto para entrenar y probar un modelo "k-nearest neighbors" utilizando estas columnas en su lugar. Utiliza el código que escribiste en la última pantalla como guía.

Utilización de más Funciones

Instrucciones

- Cree una nueva instancia de la clase `KNeighborsRegressor` con los siguientes parámetros:
 - `n_neighbors: 5`
 - `algorithm: brute`
- Ajuste un modelo que utilice las siguientes columnas de nuestro conjunto de entrenamiento (`train_df`):
 - `Accommodates`
 - `Bedrooms`
 - `Bathrooms`
 - `number_of_reviews`
- Use el modelo para hacer predicciones en el conjunto de prueba (`test_df`) utilizando las mismas columnas. Asignar la matriz NumPy de predicciones a `four_predictions`
- Use la función `mean_squared_error()` para calcular el valor MSE de estas predicciones comparando `four_predictions` con la columna de `price` de `test_df`. Asigne el valor MSE calculado a `four_mse`
- Calcular el valor del RMSE y asignarlo a `four_rmse`
- Mostrar `four_mse` y `four_rmse` mediante la función de impresión

Utilización de más Funciones

Soluciones

```
1 features = ['accommodates', 'bedrooms', 'bathrooms',
2 'number_of_reviews']
3 from sklearn.neighbors import KNeighborsRegressor
4 knn = KNeighborsRegressor(n_neighbors=5, algorithm='brute')
5 knn.fit(train_df[features], train_df['price'])
6 four_predictions = knn.predict(test_df[features])
7 four_mse = mean_squared_error(test_df['price'], four_predictions)
8 four_rmse = four_mse ** (1/2)
9 print(four_mse)
10 print(four_rmse)
```

Utilización de Todas las Funciones

Hasta aquí todo bien. A medida que aumentamos las características que utiliza el modelo, observamos valores de MSE y RMSE más bajos:

feature(s)	MSE	RMSE
accommodates	18646.5	136.6
bathrooms	17333.4	131.7
accommodates, bathrooms	15660.4	125.1
accommodates, bathrooms, bedrooms, number_of_reviews	13320.2	115.4

Llevemos esto al extremo y utilicemos todas las características potenciales. Deberíamos esperar que las puntuaciones de error disminuyan, ya que hasta ahora añadir más características ha ayudado a hacerlo.

Utilización de Todas las Funciones

Instrucciones

- Utilice todas las columnas, excepto la de precio (`price`), para entrenar un modelo de vecinos más cercanos utilizando los mismos parámetros para la clase `KNeighborsRegressor` que los de las últimas pantallas
- Utilice el modelo para realizar predicciones en el conjunto de pruebas y asigne la matriz NumPy resultante de predicciones a `all_features_predictions`
- Calcule los valores MSE y RMSE y asígnelos a `all_features_mse` y `all_features_rmse`
- Utilice la función de impresión (`print`) para mostrar ambas puntuaciones de error

Utilización de Todas las Funciones

Soluciones

```
1 knn = KNeighborsRegressor(n_neighbors=5, algorithm='brute')
2
3 features = train_df.columns.tolist()
4 features.remove('price')
5
6 knn.fit(train_df[features], train_df['price'])
7 all_features_predictions = knn.predict(test_df[features])
8 all_features_mse = mean_squared_error(test_df['price'],
9 all_features_predictions)
9 all_features_rmse = all_features_mse ** (1/2)
10 print(all_features_mse)
11 print(all_features_rmse)
```

Utilización de Todas las Funciones

- ✓ Curiosamente, el valor del RMSE aumentó hasta 125,1 cuando utilizamos todas las características disponibles. Esto significa que la selección de las características adecuadas es importante y que el uso de más características no mejora automáticamente la precisión de la predicción. Deberíamos reformular la palanca que mencionamos antes de:
 - Aumentar el número de atributos que el modelo utiliza para calcular la similitud al clasificar a los vecinos más cercanos a:
 - Seleccione los atributos relevantes que el modelo utiliza para calcular la similitud al clasificar a los vecinos más cercanos
- ✓ El proceso de selección de las características que se utilizarán en un modelo se conoce como selección de características
- ✓ En esta misión, preparamos los datos para poder utilizar más características, entrenamos algunos modelos utilizando múltiples características y evaluamos las diferentes compensaciones de rendimiento. Hemos explorado cómo el uso de más características no siempre mejora la precisión de un modelo de. En la próxima misión, exploraremos otro botón para ajustar los modelos de k-nearest neighbors: el valor de k

I.5 Optimización de Hiperparámetros



Recapitulación

En la última misión, nos centramos en aumentar el número de atributos que utiliza el modelo. Hemos visto que, en general, añadir más atributos reduce el error del modelo. Esto se debe a que el modelo es capaz de identificar mejor los espacios vitales del conjunto de entrenamiento que son más similares a los del conjunto de prueba. Sin embargo, también observamos que el uso de todas las características disponibles no mejoraba la precisión del modelo de forma automática y que algunas de las características probablemente no eran relevantes para la clasificación de la similitud. Aprendimos que la selección de rasgos relevantes era la palanca adecuada para mejorar la precisión de un modelo, y no sólo el aumento de los rasgos utilizados en la clasificación absoluta.

En esta misión, nos centraremos en el impacto de aumentar k , el número de vecinos cercanos que el modelo utiliza para hacer predicciones. Hemos exportado los conjuntos de entrenamiento (**train_df**) y de prueba (**test_df**) de las últimas misiones a archivos CSV, **dc_airbnb_train.csv** y **dc_airbnb_test.csv** respectivamente. Vamos a leer estos dos CSV en Dataframes.

Recapitulación

Instrucciones

- Leer **dc_airbnb_train.csv** en un Dataframe y asignarlo a **train_df**
- Leer **dc_airbnb_test.csv** en un Dataframe y asignarlo a **test_df**

```
1 import pandas as pd  
2 train_df = pd.read_csv('dc_airbnb_train.csv')  
3 test_df = pd.read_csv('dc_airbnb_test.csv')
```

Optimización de Hiperparámetros

Cuando variamos las características que se utilizan en el modelo, estamos afectando a los datos que utiliza el modelo. Por otro lado, variar el valor de k afecta al comportamiento del modelo independientemente de los datos reales que se utilizan al hacer las predicciones. En otras palabras, estamos afectando al rendimiento del modelo sin intentar cambiar los datos que se utilizan.

Los valores que afectan al comportamiento y al rendimiento de un modelo y que no están relacionados con los datos utilizados se denominan **hiperparámetros**. El proceso de encontrar el valor óptimo de los hiperparámetros se conoce como optimización de hiperparámetros. Una técnica de optimización de hiperparámetros sencilla pero común se conoce como búsqueda en cuadrícula, que consiste en:

- Seleccionar un subconjunto de los posibles valores de los hiperparámetros
- Entrenar un modelo utilizando cada uno de estos valores de hiperparámetros
- Evaluar el rendimiento de cada modelo
- Seleccionar el valor de los hiperparámetros que dé lugar al valor de error más bajo

Optimización de Hiperparámetros

La búsqueda en la cuadrícula se reduce esencialmente a evaluar el rendimiento del modelo con diferentes valores de k y a seleccionar el valor de k que dé lugar al menor error. Mientras que la búsqueda en cuadrícula puede llevar mucho tiempo cuando se trabaja con grandes conjuntos de datos, los datos con los que estamos trabajando en esta misión son pequeños y este proceso es relativamente rápido.

Confirmemos que la búsqueda en cuadrícula funcionará rápidamente para el conjunto de datos con el que estamos trabajando, observando primero cómo cambia el rendimiento del modelo a medida que aumentamos el valor de k de **1** a **5**. Si recuerdas, establecimos 5 como valor de k para las dos últimas misiones. Utilicemos las características de la última misión que dieron lugar a la mejor precisión del modelo:

- **accommodates**
- **bedrooms**
- **bathrooms**
- **number_of_reviews**

Optimización de Hiperparámetros

Instrucciones

- Crear una lista que contenga los valores enteros **1, 2, 3, 4 y 5**, en ese orden, y asignarla a **hyper_params**
- Crear una lista vacía y asignarla a **mse_values**
- Utilice un bucle **for** para iterar sobre **hyper_params** y en cada iteración:
 - Instanciar un objeto KNeighborsRegressor con los siguientes parámetros:
 - **n_neighbors**: el valor actual de la variable del iterador
 - **algorithm**: brute
 - Ajuste el modelo instanciado de vecinos más cercanos a las siguientes columnas de **train_df**:
 - **accommodates**
 - **bedrooms**
 - **bathrooms**
 - **number_of_reviews**
 - Utilizar el modelo entrenado para hacer predicciones sobre las mismas columnas de **test_df** y asignarlas a las **predictions**
 - Utilice la función **mean_squared_error** para calcular el valor MSE entre las **predictions** y la columna de precios de **test_df**
 - Añadir el valor de MSE a **mse_values**
- Mostrar **mse_values** con la función **print()**

Optimización de Hiperparámetros

Soluciones

```
1 from sklearn.neighbors import KNeighborsRegressor
2 from sklearn.metrics import mean_squared_error
3 features = ['accommodates', 'bedrooms', 'bathrooms',
4 'number_of_reviews']
5 hyper_params = [1, 2, 3, 4, 5]
6 mse_values = list()
7
8 for hp in hyper_params:
9     knn = KNeighborsRegressor(n_neighbors=hp,
10     algorithm='brute')
11     knn.fit(train_df[features], train_df['price'])
12     predictions = knn.predict(test_df[features])
13     mse = mean_squared_error(test_df['price'],
14     predictions)
15     mse_values.append(mse)
16
17 print(mse_values)
```

Ampliar la Búsqueda en la Cuadrícula

Como nuestro conjunto de datos es pequeño y scikit-learn se ha desarrollado pensando en el rendimiento, el código se ejecutó rápidamente. A medida que aumentamos el valor de k de **1** a **5**, el valor de MSE cayó de aproximadamente 26.364 a aproximadamente 14.090:

k	MSE
1	26364.928327645051
2	15100.522468714449
3	14579.597901655923
4	16212.300767918088
5	14090.011649601822

Vamos a ampliar la búsqueda en la red hasta un valor de k de **20**. Aunque **20** puede parecer un punto final arbitrario para nuestra búsqueda en la cuadrícula, siempre podemos ampliar los valores que probamos si no estamos convencidos de que el valor de MSE más bajo está asociado a uno de los valores de hiperparámetro que hemos probado hasta ahora.

Ampliar la Búsqueda en la Cuadrícula

Instrucciones

- Cambiar la lista de valores de hiperparámetros, `hyper_params`, para que vaya de 1 a 20
- Crear una lista vacía y asignarla a `mse_values`
- Utilice un bucle `for` para iterar sobre `hyper_params` y en cada iteración:
 - Instancia un objeto KNeighborsRegressor con los siguientes parámetros:
 - `n_neighbors`: el valor actual de la variable del iterador
 - `algorithm`: brute
 - Ajustar el modelo k-nearest neighbors instanciado a las siguientes columnas de `train_df`:
 - `accommodates`
 - `bedrooms`
 - `bathrooms`
 - `number_of_reviews`
 - Utilice el modelo entrenado para realizar predicciones sobre las mismas columnas de `test_df` y asignarlas a las predicciones
 - Utilice la función `mean_squared_error` para calcular el valor MSE entre las predicciones y la columna de precios de `test_df`
 - Añada el valor MSE a `mse_values`
 - Mostrar `mse_values` con la función `print()`

Ampliar la Búsqueda en la Cuadrícula

Soluciones

```
1 features = ['accommodates', 'bedrooms', 'bathrooms',
2 'number_of_reviews']
3 hyper_params = [x for x in range(1, 21)]
4 mse_values = list()
5
6 for hp in hyper_params:
7     knn = KNeighborsRegressor(n_neighbors=hp,
8         algorithm='brute')
9     knn.fit(train_df[features], train_df['price'])
10    predictions = knn.predict(test_df[features])
11    mse = mean_squared_error(test_df['price'],
12        predictions)
13    mse_values.append(mse)
14
15 print(mse_values)
```

Visualización de los Valores de los Hiperparámetros

Al aumentar el valor de k de **1 a 6**, el valor de MSE disminuyó de aproximadamente 26.364 a aproximadamente 13.657. Sin embargo, al aumentar el valor de k de **7 a 20**, el valor MSE no disminuyó más, sino que se mantuvo entre 14.288 y 14.870 aproximadamente. Esto significa que el valor óptimo de k es **6**, ya que dio lugar al valor MSE más bajo.

Este patrón es algo que notará al realizar la búsqueda de cuadrícula en otros modelos también. Al principio, al aumentar k, la tasa de error disminuye hasta cierto punto, pero luego rebota y vuelve a aumentar. Confirmemos este comportamiento visualmente utilizando un gráfico de dispersión.

Visualización de los Valores de los Hiperparámetros

Instrucciones

- Utilice el método `scatter()` de `matplotlib.pyplot` para generar un gráfico de líneas con:
 - `hyper_params` en el eje X
 - `mse_values` en el eje Y
- Utilice `plt.show()` para mostrar el gráfico de líneas

Visualización de los Valores de los Hiperparámetros

Soluciones

```
1 import matplotlib.pyplot as plt
2
3 features = ['accommodates', 'bedrooms', 'bathrooms',
4 'number_of_reviews']
5 hyper_params = [x for x in range(1, 21)]
6 mse_values = list()
7
8 for hp in hyper_params:
9     knn = KNeighborsRegressor(n_neighbors=hp,
10     algorithm='brute')
11     knn.fit(train_df[features], train_df['price'])
12     predictions = knn.predict(test_df[features])
13     mse = mean_squared_error(test_df['price'],
14     predictions)
15     mse_values.append(mse)
16 plt.scatter(hyper_params, mse_values)
17 plt.show()
```

Visualización de los Valores de los Hiperparámetros

El primer modelo, que utilizó las columnas de **accommodates** y **bathrooms**, pudo alcanzar un valor de MSE de aproximadamente 14.790. El segundo modelo, que añadía la columna de **bedrooms**, consiguió un valor de MSE de aproximadamente 13.522,9, que es incluso inferior al valor de MSE más bajo que conseguimos utilizando el mejor modelo de la última misión (que utilizaba las columnas de **accommodates**, **bedrooms**, **bathrooms** y **numbers_of_reviews**). Esperemos que esto demuestre que el uso de una sola palanca para encontrar el mejor modelo no es suficiente y que es necesario utilizar ambas palancas en conjunto.

En esta misión, hemos aprendido sobre la optimización de hiperparámetros y el flujo de trabajo para encontrar el modelo óptimo para hacer predicciones. Lo siguiente en este curso es un reto, en el que practicarás los conceptos que has aprendido hasta ahora en un conjunto de datos completamente nuevo.

I.6 Validación Cruzada



En una misión anterior, aprendimos sobre la validación de entrenamiento/prueba, una técnica simple para probar la precisión de un modelo de aprendizaje automático en nuevos datos en los que el modelo no fue entrenado. En esta misión, nos centraremos en técnicas más robustas.

Para empezar, nos centraremos en la técnica de validación de retención (**holdout validation**), que implica:

- Dividir el conjunto de datos completo en 2 particiones:
 - Un conjunto de entrenamiento
 - Un conjunto de pruebas
 - Entrenar el modelo en el conjunto de entrenamiento
- Utilizar el modelo entrenado para predecir etiquetas en el conjunto de pruebas
- Calcular una métrica de error para comprender la eficacia del modelo
- Cambiar los conjuntos de entrenamiento y de prueba y repetirlo
- Promediar los errores

Concepto

En la validación por retención (**holdout validation**), solemos utilizar una división 50/50 en lugar de la división 75/25 de la validación de entrenamiento/prueba. De este modo, eliminamos el número de observaciones como fuente potencial de variación en el rendimiento de nuestro modelo.



Figura 4. Descomposición de la prueba y el entreno

Empecemos por dividir el conjunto de datos en 2 mitades casi equivalentes.

Al dividir el conjunto de datos, no olvides establecer una copia del mismo utilizando `.copy()` para asegurarte de que no obtienes resultados inesperados más adelante. Si ejecutas el código localmente en Jupyter Notebook o Jupyter Lab sin `.copy()`, notarás lo que se conoce como una advertencia SettingWithCopy. Esto no impedirá que tu código se ejecute correctamente, pero te está haciendo saber que cualquier operación que estés haciendo está tratando de establecerse en una copia de un slice de un dataframe. Para asegurarse de no ver esta advertencia, asegúrese de incluir `.copy()` siempre que realice operaciones en un marco de datos.

Concepto

Instrucciones

- Usar la función **numpy.random.permutation()** para barajar el orden de las filas en **dc_listings**
- Seleccionar las primeras 1862 filas y asignarlas a **split_one**
- Seleccionar las 1861 filas restantes y asignarlas a **split_two**

```
1 import numpy as np
2 import pandas as pd
3
4 dc_listings = pd.read_csv("dc_airbnb.csv")
5 stripped_commas = dc_listings['price'].str.replace(',','')
6 stripped_dollars = stripped_commas.str.replace('$', '')
7 dc_listings['price'] = stripped_dollars.astype('float')
8 shuffled_index =
9     np.random.permutation(dc_listings.index)
10 dc_listings = dc_listings.reindex(shuffled_index)
11
12 split_one = dc_listings.iloc[0:1862].copy()
13 split_two = dc_listings.iloc[1862: ].copy()
```

Validación de la Retención

Ahora que hemos dividido nuestro conjunto de datos en 2 marcos de datos (dataframes), vamos a:

- Entrenar un modelo de k-nearest neighbors en la primera mitad
- Probar este modelo en la segunda mitad
- Entrenar un modelo de k-nearest neighbors en la segunda mitad
- Probar este modelo en la segunda mitad

Instrucciones

- Entrenar un modelo de k-nearest neighbors model utilizando el algoritmo por defecto (**auto**) y utilizar el numero por defecto (**5**) que:
 - Utilize la columna de **accommodates** de **train_one** para entrenamiento
 - Pruebe en **test_one**
- Asigna el valor de RMSE resultante a **iteration_one_rmse**
- Entrenar un modelo de k-nearest neighbors model utilizando el algoritmo por defecto (**auto**) y utilizar el numero por defecto (**5**) que:
 - Utilize la columna de **accommodates** de **train_two** para entrenamiento
 - Pruebe en **test_two**
- Asigna el valor de RMSE resultante a **iteration_two_rmse**
- Utiliza **numpy.mean()** para calcular la media de los 2 valores de RMSE y la asigna a **avg_rmse**

Validación de la Retención

Soluciones

```
1 from sklearn.neighbors import KNeighborsRegressor
2 from sklearn.metrics import mean_squared_error
3
4 train_one = split_one
5 test_one = split_two
6 train_two = split_two
7 test_two = split_one
8 # First half
9 model = KNeighborsRegressor()
10 model.fit(train_one[["accommodates"]],
11 train_one["price"])
12 test_one["predicted_price"] =
13     model.predict(test_one[["accommodates"]])
14 iteration_one_rmse =
15     mean_squared_error(test_one["price"],
16     test_one["predicted_price"])**(1/2)
17
18 # Second half
19 model.fit(train_two[["accommodates"]],
20 train_two["price"])
21 test_two["predicted_price"] =
22     model.predict(test_two[["accommodates"]])
23 iteration_two_rmse =
24     mean_squared_error(test_two["price"],
25     test_two["predicted_price"])**(1/2)
26
27 avg_rmse = np.mean([iteration_one_rmse,
28 iteration_two_rmse])
29
30 print(iteration_one_rmse, iteration_two_rmse, avg_rmse)
```

Validación Cruzada K-Fold

Si promediamos los dos valores de RMSE del último paso, obtenemos un valor de RMSE de aproximadamente **128,96**. La validación holdout es en realidad un ejemplo específico de una clase más amplia de técnicas de validación llamada **validación cruzada k-fold (K-fold cross-validation)**. Mientras que la validación holdout es mejor que la validación de entrenamiento/prueba porque el modelo no está sesgado repetidamente hacia un subconjunto específico de los datos, ambos modelos que se entrena sólo utilizan la mitad de los datos disponibles. La **validación cruzada K-fold**, por otro lado, aprovecha una mayor proporción de los datos durante el entrenamiento mientras sigue rotando a través de diferentes subconjuntos de los datos para evitar los problemas de la validación entrenamiento/prueba.

Este es el algoritmo de la validación cruzada k-fold:

- Dividir el conjunto de datos en **k** particiones de igual longitud
 - Seleccionar **k-1** particiones como conjunto de entrenamiento y
 - Seleccionar la partición restante como conjunto de pruebas
- Entrenamiento del modelo en el conjunto de entrenamiento
- Utilizar el modelo entrenado para predecir etiquetas en el pliegue de prueba
- Calcular la métrica de error del pliegue de prueba
- Repetir todos los pasos anteriores **k-1** veces, hasta que cada partición se haya utilizado como conjunto de prueba para una iteración
- Calcular la media de los **k** valores de error

Validación Cruzada K-Fold

La validación cruzada es esencialmente una versión de la validación cruzada de k pliegues cuando k es igual a **2**. Generalmente, se utilizan **5** o **10** pliegues para la validación cruzada de k pliegues. Este es un diagrama que describe cada iteración de la validación cruzada de 5 pliegos:

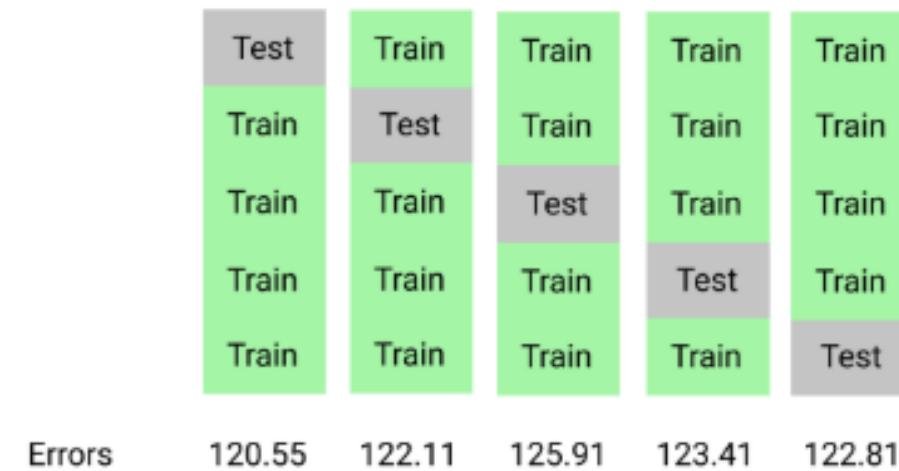


Figura 5. Diagrama que describe cada iteración de la validación cruzada de 5 veces

Validación Cruzada K-Fold

A medida que aumenta el número de pliegues, el número de observaciones en cada pliegue disminuye y la varianza de los errores por pliegue aumenta. Empecemos por dividir manualmente el conjunto de datos en 5 pliegues. En lugar de dividir en 5 marcos de datos, añadamos una columna que especifique a qué pliegue pertenece la fila. De este modo, podemos seleccionar fácilmente nuestro conjunto de entrenamiento y nuestro conjunto de prueba.

Instrucciones

- Agregue una nueva columna a `dc_listings` llamada **fold** que contenga el número de pliegue al que cada fila pertenece:
 - El pliegue **1** debe tener filas desde el índice **0** hasta el **745**, sin incluir el **745**
 - El pliegue **2** debe tener filas desde el índice **745** hasta el **1490**, sin incluir el **1490**
 - El pliegue **3** debe tener filas desde el índice **1490** hasta el **2234**, sin incluir el **2234**
 - El pliegue **4** debe tener filas desde el índice **2234** hasta el **2978**, sin incluir el **2978**
 - El pliegue **5** debe tener filas desde el índice **2978** hasta el **3723**, sin incluir el **3723**
- Muestre los recuentos de valores únicos para la columna de **fold** para confirmar que cada pliegue tiene aproximadamente el mismo número de elementos
- Muestre el número de valores perdidos en la columna **fold** para confirmar que no perdimos ninguna fila

Validación Cruzada K-Fold

Soluciones

```
1 dc_listings.loc[dc_listings.index[0:745], "fold"] = 1
2 dc_listings.loc[dc_listings.index[745:1490], "fold"] = 2
3 dc_listings.loc[dc_listings.index[1490:2234], "fold"] =
4 3
5 dc_listings.loc[dc_listings.index[2234:2978], "fold"] =
6 4
7 dc_listings.loc[dc_listings.index[2978:3723], "fold"] =
8 5
9
10 print(dc_listings['fold'].value_counts())
11 print("\n Num of missing values: ",
12     dc_listings['fold'].isnull().sum())
```

Validación Cruzada K-Fold

Hasta ahora, hemos trabajado bajo el supuesto de que un RMSE más bajo siempre significa que un modelo es más preciso. Por desgracia, esto no es del todo cierto. Un modelo tiene dos fuentes de error, el sesgo y la varianza.

El sesgo describe el error que resulta de las malas suposiciones sobre el algoritmo de aprendizaje. Por ejemplo, suponer que sólo una característica, como el peso de un coche, está relacionada con la eficiencia del combustible de un coche le llevará a ajustar un modelo de regresión simple y univariante que dará lugar a un alto sesgo. La tasa de error será alta, ya que la eficiencia de combustible de un coche se ve afectada por muchos otros factores además de su peso.

La varianza describe el error que se produce debido a la variabilidad de los valores predichos de un modelo. Si nos dieran un conjunto de datos con 1.000 características de cada coche y utilizáramos cada una de ellas para entrenar un modelo de regresión multivariante increíblemente complicado, tendríamos un sesgo bajo pero una varianza alta. En un mundo ideal, queremos un sesgo bajo y una varianza baja, pero en la realidad, siempre hay una compensación.

La desviación estándar de los valores de RMSE puede ser un indicador de la varianza de un modelo, mientras que el RMSE medio es un indicador del sesgo de un modelo. El sesgo y la varianza son las dos fuentes de error observables en un modelo que podemos controlar indirectamente.

I.7 Proyecto Guiado: Predicción de los Precios de Automóviles



Proyecto Guiado: Predicción de los Precios de Automóviles

En este curso, hemos explorado los fundamentos del aprendizaje automático utilizando el algoritmo de k-vecinos más cercanos. En este proyecto guiado, practicarás el flujo de trabajo de aprendizaje automático que has aprendido hasta ahora para predecir el precio de mercado de un coche utilizando sus atributos. El conjunto de datos con el que trabajaremos contiene información sobre varios coches. Para cada coche tenemos información sobre los aspectos técnicos del vehículo, como la cilindrada del motor, el peso del coche, los kilómetros por galón, la velocidad de aceleración del coche, etc. Puedes leer más sobre el conjunto de datos aquí y puedes descargarlo directamente desde aquí. Aquí tienes un avance del conjunto de datos:

<https://archive.ics.uci.edu/ml/datasets/automobile>

symboling	normalized_losses	make	fuel_type	aspiration	num_doors
3	?	alfa-romero	gas	std	two
3	?	alfa-romero	gas	std	two
1	?	alfa-romero	gas	std	two
2	164	audi	gas	std	four
2	164	audi	gas	std	four

Proyecto Guiado: Predicción de los Precios de Automóviles

Instrucciones

- Lee **imports-85.data** en un dataframe llamado **cars**. Si lees el archivo usando **pandas.read_csv()** sin especificar ningún valor de parámetro adicional, notarás que los nombres de las columnas no coinciden con los de la documentación del conjunto de datos ([dataset's documentation](#)) ¿Por qué cree que esto es así y cómo puede solucionarlo?
- Determine qué columnas son numéricas y pueden utilizarse como características y qué columna es la columna de destino
- Muestre las primeras filas del marco de datos y asegúrese de que se parece a la vista previa del conjunto de datos

Soluciones

Puede encontrar las soluciones para este proyecto guiado [aquí](#).

II Cálculo para el Aprendizaje Automático

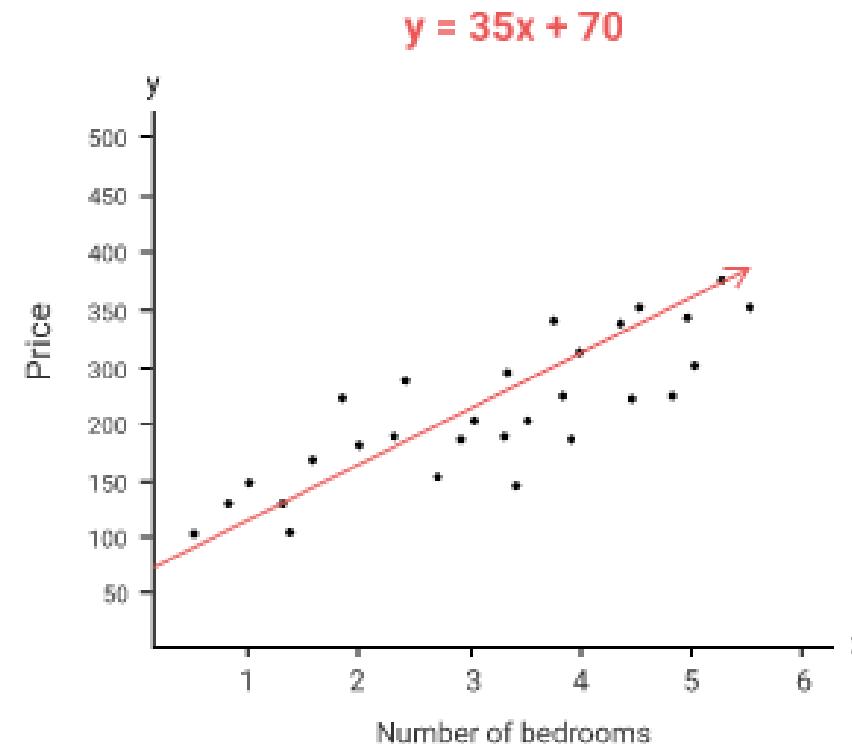
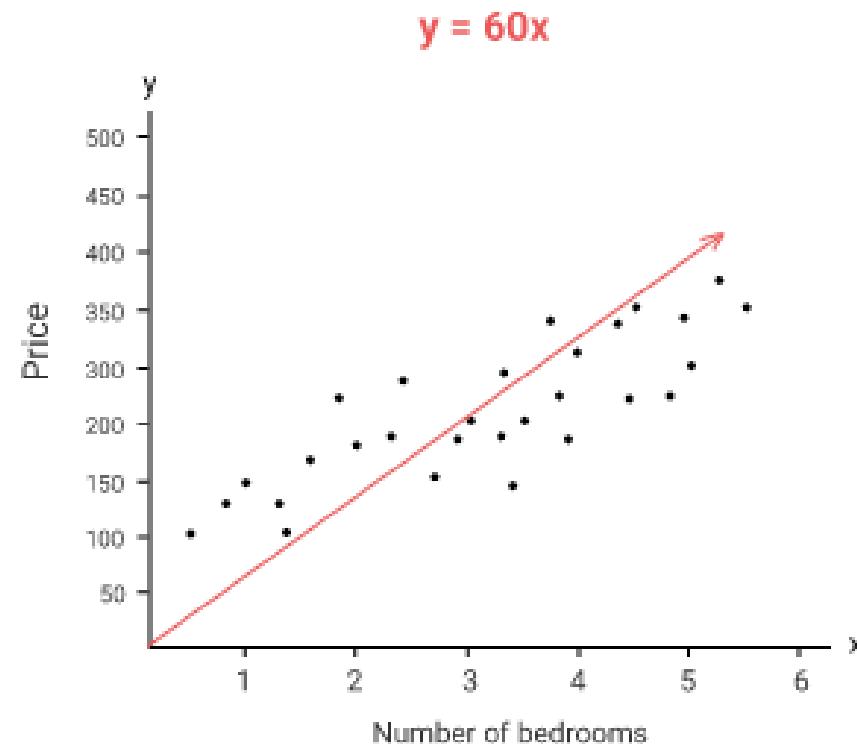


Cálculo para el Aprendizaje Automático

En el curso anterior, exploramos el flujo de trabajo del aprendizaje automático utilizando el algoritmo de k-próximos más cercanos. Elegimos el algoritmo de k-próximos más cercanos porque construir la intuición de cómo funciona el algoritmo no requiere ninguna matemática. Aunque el algoritmo es fácil de entender, no podemos utilizarlo para conjuntos de datos más grandes porque el propio modelo se representa utilizando todo el conjunto de entrenamiento. Cada vez que queremos hacer una predicción sobre una nueva observación, tenemos que calcular la distancia entre cada observación de nuestro conjunto de entrenamiento y nuestra nueva observación, y luego ordenar por distancia ascendente. Se trata de una técnica muy intensiva desde el punto de vista informático.

En la mayoría de las técnicas de aprendizaje automático que veremos a continuación, el modelo se representa como una función matemática. Esta función matemática se aproxima a la función subyacente que describe cómo se relacionan las características con el atributo objetivo. Una vez que derivamos esta función matemática utilizando el conjunto de datos de entrenamiento, hacer predicciones en el conjunto de datos de prueba (o en un futuro conjunto de datos) es computacionalmente barato. El siguiente diagrama muestra dos funciones de regresión lineal diferentes que se aproximan al conjunto de datos (nótese que los valores de este conjunto de [datos son aleatorios](https://app.dataquest.io/course/calculus-for-machine-learning)).

Cálculo para el Aprendizaje Automático



Cálculo para el Aprendizaje Automático

Antes de que podamos sumergirnos en el uso de modelos de regresión lineal para el aprendizaje automático, tendremos que entender algunas ideas clave del cálculo. El cálculo proporciona un marco para entender cómo se comportan las funciones matemáticas. El cálculo nos ayuda a:

- Entender la inclinación en varios puntos
- Encontrar los puntos extremos de una función
- Determinar la función óptima que mejor representa un conjunto de datos

Empecemos por plantear un problema motivador, al que nos referiremos a lo largo de este curso. Supongamos que se nos da la siguiente ecuación, que describe la trayectoria de un balón después de ser pateado por un jugador de fútbol: $y= -(x^2)+3x-1$

X es el tiempo en segundos mientras que Y es la posición vertical de la pelota. Naturalmente, nos gustaría saber la posición más alta que alcanzó la pelota y a qué hora ocurrió. Aunque podemos graficar la ecuación y estimar el resultado visualmente, si queremos el tiempo y la posición vertical precisos tendremos que usar el cálculo. En este curso, exploraremos los diferentes conceptos de cálculo necesarios para poder encontrar este punto preciso.

Empecemos por visualizar esta función.

Cálculo para el Aprendizaje Automático

Instrucciones

- Usar `numpy.linspace()` para generar un array NumPy que contenga **100** valores de **0** a **3** y asignarlo a **x**
- Transformar **x** aplicando la función: $y=-(x^2)+3x-1$ Asignar el array resultante de valores transformados a **y**
- Utiliza `pyplot.plot()` para generar un gráfico de líneas con **x** en el eje-x e **y** en el eje-y
- Haz una lluvia de ideas sobre cómo calcular la altura máxima y encontrar el momento exacto en que se produjo

Soluciones

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 x = np.linspace(0, 3, 100)
4 y = -1*(x**2) + x*3 - 1
5 plt.plot(x,y)
```

Comprender las Funciones Lineales y No Lineales

Antes de entrar en el análisis de la curva de la altura de una pelota, tendremos que entender primero algunas ideas clave. Exploraremos estos conceptos utilizando primero las líneas rectas simples y, a continuación, aplicaremos estos conceptos a las curvas. Una línea recta simple se define más claramente como una **función lineal**. Todas las funciones lineales se pueden escribir de la siguiente forma:

$$y=mx+b$$

En el caso de una función lineal concreta, m y b son valores constantes, mientras que x e y son variables. $y=3x+1$ e $y=5$ son ejemplos de funciones lineales.

Centrémonos por ahora en la función $y=3x+1$. Esta función multiplica por 3 cualquier valor de x que le pasemos y luego le suma 1.

Comprender las Funciones Lineales y No Lineales

Empecemos por adquirir una comprensión geométrica de las funciones lineales. A continuación, encontrarás una imagen que te ayudará a entender cómo se desplaza o cambia la recta al alterar los valores de m y/o b .

- ¿Cómo cambia la línea cuando mantienes m fijo pero varías b ?
- ¿Cómo cambia la línea cuando mantienes b fijo pero varías m ?
- ¿Qué valor controla la inclinación de la recta?
- ¿Qué ocurre con la recta cuando m se fija en 0?

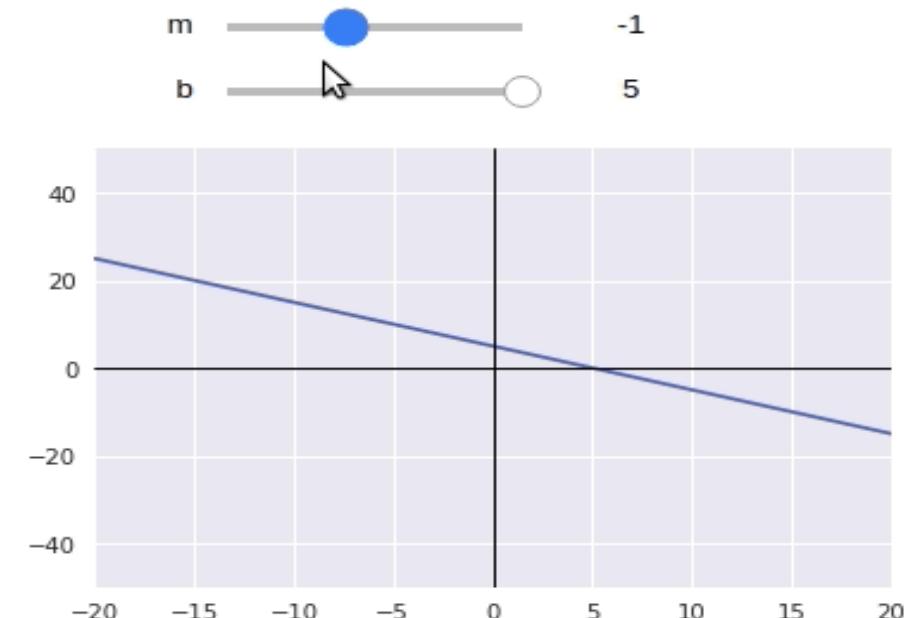


Figura 6. Ejemplo de función lineal

Comprender las Funciones Lineales y No Lineales

Hasta ahora hemos trabajado con funciones lineales, en las que podemos determinar la pendiente de la función a partir de la propia ecuación. Sin embargo, si volvemos a nuestra ecuación de la trayectoria de la pelota, te darás cuenta de que no coincide con la forma $y=mx+b$:

$$y=-(x^2)+3x-1$$

Esto se debe a que esta función es una función **no lineal**. Las funciones no lineales no representan líneas rectas, sino curvas como la que hemos trazado en el primer paso de esta misión. Las salidas y de una función no lineal no son proporcionales a los valores de entrada x. Un incremento en x no resulta en un incremento constante en y.

Comprender las Funciones Lineales y No Lineales

Siempre que x se eleve a una potencia no igual a 1, tenemos una función no lineal. He aquí algunos ejemplos más de funciones no lineales:

$$y = x^3$$

$$y = x^3 + 3x^2 + 2x - 1$$

$$y = \frac{1}{-x^2}$$

$$y = \sqrt{x}$$

En la siguiente imagen, observa cómo cambia la pendiente con diferentes valores de x_1 y x_2 .

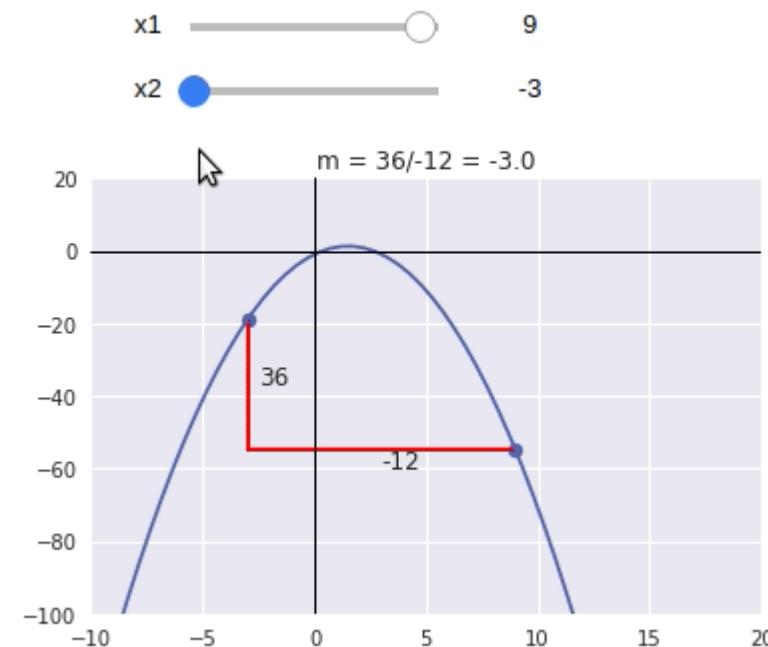


Figura 7. Ejemplo de función no lineal

Comprensión de los Límites

Al final de la última misión, fijamos un primer punto en nuestra curva, dibujamos una línea secante entre ese primer punto y un segundo punto, y observamos lo que ocurría cuando acercábamos el segundo punto al primero a lo largo de la curva. Cuanto mayor era el intervalo entre los dos puntos en el eje x, más divergía la inclinación de la línea secante de la inclinación de la curva. Cuanto más cercano es el intervalo, más se ajusta la línea secante a la pendiente del primer punto de la curva.

En esta misión, formalizaremos más la idea de la pendiente y aprenderemos a calcular la pendiente de las ecuaciones no lineales en cualquier punto. **A medida que avanzas en el resto de este curso, te recomendamos encarecidamente que sigas las matemáticas que presentamos utilizando lápiz y papel.** Empezaremos introduciendo una notación matemática que formaliza la observación que hicimos al final de la última misión. Si intentamos enunciar la observación introduciendo valores en la ecuación de la pendiente, nos encontraremos con el problema de la división por cero:

$$m = \frac{f(x_2) - f(x_1)}{x_2 - x_1} = 0/0$$

Comprensión de los Límites

Aunque la pendiente es indefinida cuando x_1 y x_2 son equivalentes, queremos ser capaces de afirmar y razonar sobre el valor al que se aproxima la pendiente cuando x_2 se acerca a x_1 . Para ello, tenemos que replantear el problema como un **límite**. Un límite describe el valor al que se aproxima una función cuando la variable de entrada a la función se aproxima a un valor específico. En nuestro caso, la variable de entrada es x_2 y nuestra función es $m=f(x_2)-f(x_1)$. La siguiente notación matemática formaliza la afirmación "A medida que x_2 se aproxima a 3, la pendiente entre x_1 y x_2 se aproxima a -3" utilizando un límite:

$$\lim_{x_2 \rightarrow 3} \frac{f(x_2) - f(x_1)}{x_2 - x_1} = -3$$

$\lim_{x_2 \rightarrow 3}$ es otra forma de decir "A medida que x_2 se acerca a 3". Como hemos fijado x_1 en 3, podemos sustituir x_1 por 3 en la función:

$$\lim_{x_2 \rightarrow 3} \frac{f(x_2) - f(3)}{x_2 - 3} = \lim_{x_2 \rightarrow 3} \frac{f(x_2) + 1}{x_2 - 3} = -3$$

Encontrar Puntos Extremos

En la última misión, aprendimos a utilizar los límites para calcular el punto al que se aproxima una función cuando el valor de entrada se acerca a un valor específico. Hemos aplicado esta técnica para calcular la pendiente de la recta tangente en un punto concreto de nuestra función no lineal:

$$y = -(x^2) + 3x - 1$$

Si recuerdas la primera misión de este curso, nos interesa determinar el punto más alto de esta curva.

Encontrar Puntos Extremos

Si alguna vez has hecho senderismo en una montaña, estarás familiarizado con la forma en que el sendero se inclina hacia arriba hasta llegar a la cima. Sin embargo, una vez que se llega a la cima, todos los caminos de vuelta descienden. Entender cómo varía la pendiente a lo largo de una curva proporciona una lente útil para determinar el punto máximo de una curva.

Empezaremos construyendo una intuición visual sobre cómo se relacionan la pendiente de una función y su punto máximo. En la imagen siguiente, hemos generado dos gráficos. A medida que xx varía, el gráfico de la izquierda visualiza **cómo cambia la línea tangente de la curva**, mientras que el gráfico de la derecha visualiza **cómo cambia la pendiente de esta línea tangente**.

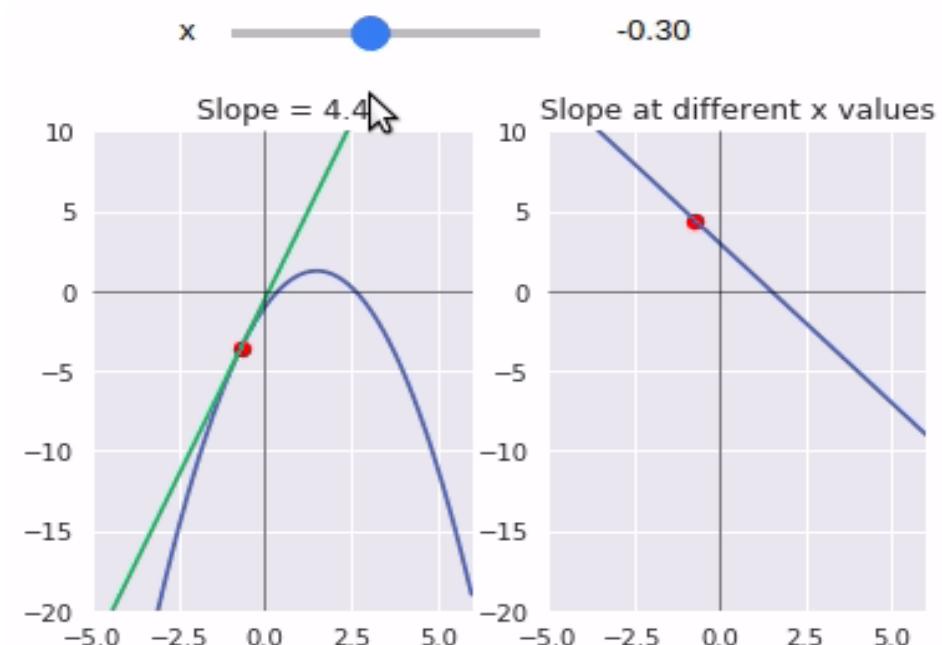


Figura 8. La línea tangente de la curva cambia

III Álgebra Lineal para el Aprendizaje Automático



Álgebra Lineal para el Aprendizaje Automático

El álgebra lineal es un pilar del aprendizaje automático. No se puede desarrollar una profunda comprensión y aplicación del aprendizaje automático sin ella. Mediante explicaciones claras, bibliotecas estándar de Python y lecciones tutoriales paso a paso, descubrirá qué es el álgebra lineal, la importancia del álgebra lineal para el aprendizaje automático, las operaciones vectoriales y matriciales, y mucho más.

<https://mml-book.github.io/book/mml-book.pdf>

Sistemas Lineales

En el último curso, exploramos el marco del cálculo y lo utilizamos para:

- Comprender la pendiente de las funciones lineales
- Comprender la derivada (la pendiente como función) de las funciones no lineales
- Encontrar valores extremos en funciones no lineales

Aunque aprendimos los fundamentos de la pendiente a través de las funciones lineales, en el último curso nos centramos principalmente en las funciones no lineales.

En este curso, nos centraremos en la comprensión de las funciones lineales. Específicamente, exploraremos el marco del álgebra lineal, que proporciona una manera de representar y entender las soluciones a los sistemas de ecuaciones lineales. Un sistema de ecuaciones lineales consiste en múltiples funciones relacionadas con un conjunto común de variables. La palabra ecuación lineal se utiliza a menudo indistintamente con función lineal. Muchos procesos del mundo real pueden modelarse utilizando múltiples ecuaciones lineales relacionadas. Empezaremos explorando un ejemplo concreto de sistema lineal, otra palabra para sistema de ecuaciones lineales, antes de profundizar en el álgebra lineal.

Problema del Salario Óptimo

Supongamos que tenemos que elegir entre dos ofertas de trabajo diferentes. La primera oferta de trabajo tiene un salario semanal base de 1000 dólares y paga 30 dólares por hora. Podemos representar esta oferta como $y=1000+30x$, donde y representa los dólares ganados esa semana y xx representa las horas trabajadas esa semana. La segunda oferta de trabajo tiene un salario semanal base de 100 dólares y paga 50 dólares por hora. Podemos representar esta oferta como $y=100+50x$, donde y también representa los dólares ganados esa semana y xx también representa las horas trabajadas esa semana.

Queremos saber qué oferta de trabajo es mejor. Si sabemos exactamente la cantidad de dinero que queremos ganar cada semana (y), podemos sustituir ese valor en ambas ecuaciones y resolver para x para identificar qué trabajo nos exigirá menos horas. Si sabemos exactamente el número de horas que queremos trabajar cada semana (x), podemos sustituir ese valor en ambas ecuaciones y resolver para y para identificar qué trabajo nos hará ganar más dinero por la misma cantidad de horas trabajadas. En cambio, si queremos entender:

- ¿A partir de qué número de horas trabajadas podemos esperar ganar la misma cantidad de dinero en cualquiera de los dos trabajos?
- ¿Cuántas horas tenemos que trabajar para ganar más dinero en el primer trabajo que en el segundo?

Para responder a la primera pregunta, tenemos que encontrar el valor x en el que los dos valores y son equivalentes. Una vez que sepamos dónde se cruzan, podremos averiguar fácilmente la respuesta a la segunda pregunta.

Instrucciones

- Usar `numpy.linspace()` para generar **1000** valores espaciados uniformemente entre **0** y **50** y asignarlos a **x**
- Transforme **x** usando la ecuación $y=30x+1000$ y asigne el resultado a **y1**
- Transforme **x** utilizando la ecuación $y=50x+100$ y asigne el resultado a **y2**
- Genera 2 gráficos de líneas en la misma gráfica secundaria:
 - Uno con **x** en el eje x y **y1** en el eje y. Establezca el color de la línea en "**orange**"
 - Uno con **x** en el eje x e **y2** en el eje y. Establezca el color de la línea en "**blue**"
- Omitir la selección de un rango de valores para los ejes x e y, y en su lugar dejar que matplotlib seleccione automáticamente en base a los datos

Soluciones

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 x = np.linspace(0, 50, 1000)
4 y1 = 30*x + 1000
5 y2 = 50*x + 100
6
7 plt.plot(x, y1, c='orange')
8 plt.plot(x, y2, c='blue')
```

Vectores

En la última misión, aprendimos a utilizar una matriz aumentada y las operaciones de fila que preservan las relaciones en un sistema para resolver un sistema de funciones lineales. En esencia, una matriz es una forma de representar una tabla de números. Todas las matrices con las que trabajamos en la última misión contenían 2 filas y 3 columnas. Aquí está la primera que configura nuestro sistema lineal:

$$\left[\begin{array}{cc|c} 30 & -1 & -1000 \\ 50 & -1 & -100 \end{array} \right]$$

Esto se conoce como una matriz de 2×3 (se pronuncia "matriz de dos por tres"). La convención en álgebra lineal es especificar primero el número de filas (2) y luego el número de columnas (3). Cada una de las filas y columnas de esta matriz se representa como una lista de números.

Una lista de números se conoce como **vector**. Una fila de una matriz se conoce como **vector fila**, mientras que una columna se conoce como **vector columna**. Estos son los vectores fila de la matriz:

$$[30 \quad -1 \quad -1000]$$

$$[50 \quad -1 \quad -100]$$

Vectores

Estos son los vectores columna de la matriz:

$$\begin{bmatrix} 30 \\ 50 \end{bmatrix}$$

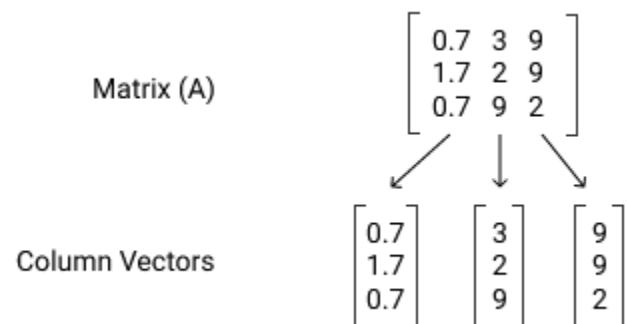
$$\begin{bmatrix} -1 \\ -1 \end{bmatrix}$$

$$\begin{bmatrix} -1000 \\ -100 \end{bmatrix}$$

En esta misión, aprenderemos más sobre los vectores columna y sus operaciones asociadas para ayudarnos a entender ciertas propiedades de los sistemas lineales. Terminaremos esta misión justificando el enfoque que utilizamos en la última misión para resolver el sistema lineal conectando algunas ideas clave de las matrices y los vectores. Empezaremos construyendo alguna intuición geométrica de los vectores. Generalmente, la palabra vector se refiere al vector columna (lista ordenada de elementos en una sola columna) y nos referiremos al vector columna de esa manera durante el resto de este curso.

Álgebra Matricial

Al igual que los vectores, las matrices tienen su propio conjunto de operaciones algebraicas. En esta misión, aprenderemos las principales operaciones matriciales y llegaremos a utilizar algunas de ellas para resolver la ecuación matricial. Empecemos por la suma y la resta de matrices. Si recuerdas la misión anterior, una matriz está formada por uno o más vectores columna.



Por ello, las operaciones de los vectores también se trasladan a las matrices. Podemos realizar sumas y restas vectoriales entre vectores con el mismo número de filas. Podemos realizar sumas y restas de matrices entre matrices que contengan el mismo número de filas y columnas.

Valid Sums	Invalid Sum
$\begin{bmatrix} 0.7 & 3 \\ 1.7 & 2 \end{bmatrix} + \begin{bmatrix} 1 & 3 \\ 2 & 1 \end{bmatrix}$ $\begin{bmatrix} 0.7 & 3 \\ 9 & 1.7 \\ 2 & 9 \end{bmatrix} + \begin{bmatrix} 0.7 & 3 \\ 9 & 1.7 \\ 2 & 9 \end{bmatrix}$	$\begin{bmatrix} 0.7 & 3 & 9 \\ 1.7 & 2 & 9 \end{bmatrix} + \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix}$

Valid Sums:

$\begin{bmatrix} 0.7 & 3 \\ 1.7 & 2 \end{bmatrix} + \begin{bmatrix} 1 & 3 \\ 2 & 1 \end{bmatrix}$ $\begin{bmatrix} 0.7 & 3 \\ 9 & 1.7 \\ 2 & 9 \end{bmatrix} + \begin{bmatrix} 0.7 & 3 \\ 9 & 1.7 \\ 2 & 9 \end{bmatrix}$

r c **r c**
2 x 2 2 x 2

r c **r c**
3 x 2 3 x 2

Invalid Sum:

$\begin{bmatrix} 0.7 & 3 & 9 \\ 1.7 & 2 & 9 \end{bmatrix} + \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix}$

r c **r c**
2 x 3 3 x 1

Álgebra Matricial

Al igual que con los vectores, la suma y la resta de matrices funciona distribuyendo las operaciones entre los elementos específicos y combinándolos.

$$A + B = \begin{bmatrix} A \\ 0.7 & 3 & 9 \\ 1.7 & 2 & 9 \\ 0.7 & 9 & 2 \end{bmatrix} + \begin{bmatrix} B \\ 0 & 3 & 3 \\ 1 & 2 & 1 \\ 0 & 1 & 2 \end{bmatrix} = \begin{bmatrix} 0.7 & 6 & 12 \\ 2.7 & 4 & 10 \\ 0.7 & 10 & 4 \end{bmatrix}$$

Por último, también podemos multiplicar una matriz por un valor escalar, al igual que podemos hacerlo con un vector.

$$5A = 5 \begin{bmatrix} A \\ 0.7 & 3 & 9 \\ 1.7 & 2 & 9 \\ 0.7 & 9 & 2 \end{bmatrix} = \begin{bmatrix} 3.5 & 15 & 45 \\ 8.5 & 10 & 45 \\ 3.5 & 45 & 10 \end{bmatrix}$$

Conjuntos de Soluciones

En este curso, hemos explorado dos formas diferentes de encontrar la solución de $A\vec{x}=\vec{b}$ cuando \vec{b} no es un vector que contiene todos los ceros ($\vec{b}\neq 0$). La primera forma que exploramos fue la eliminación gaussiana, que consiste en utilizar las operaciones de fila para transformar la representación aumentada de un sistema lineal a la forma escalonada y, finalmente, a la forma escalonada reducida.

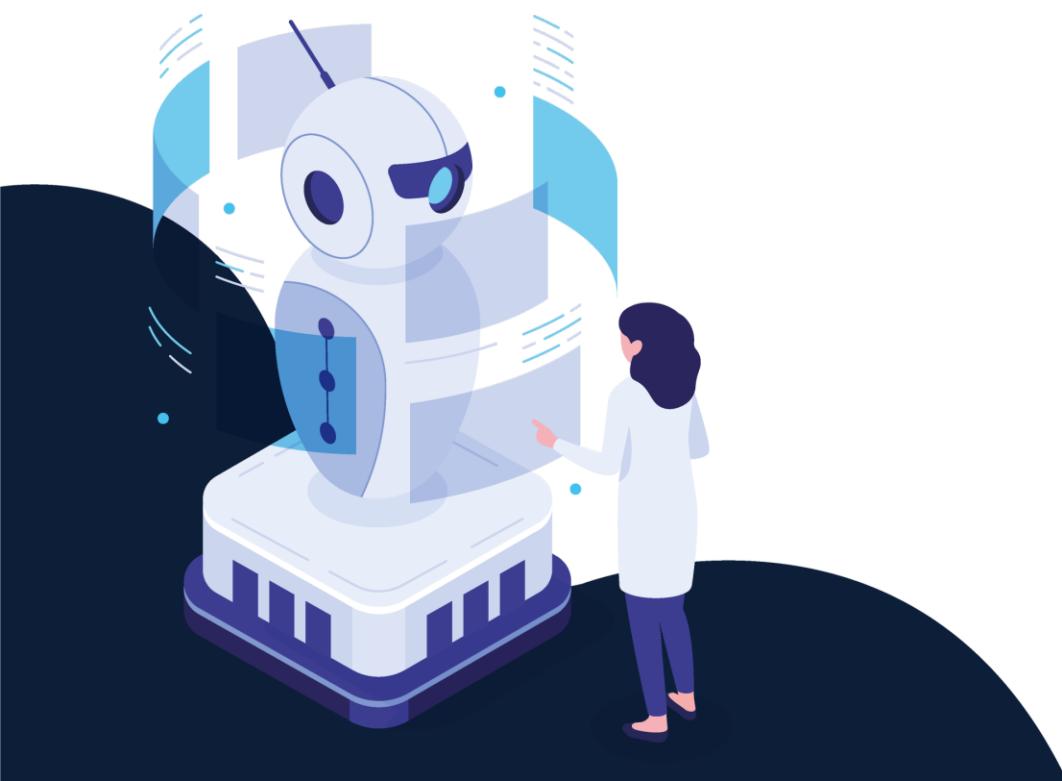
La segunda forma que exploramos fue calcular la matriz inversa de A y multiplicar por la izquierda ambos lados de la ecuación para encontrar \vec{x} .

Aunque podemos usar estas técnicas para resolver la mayoría de los sistemas lineales que encontraremos, necesitamos aprender qué hacer cuando:

- **El conjunto de soluciones de un sistema lineal no existe**
- **El conjunto de soluciones de un sistema lineal no es un único vector**
- **\vec{b} es igual a $\vec{0}$**

En esta misión, terminaremos este curso explorando estas tres situaciones.

IV Regresión Lineal para el Aprendizaje Automático



Regresión Lineal para el Aprendizaje Automático

La regresión lineal es uno de los algoritmos de aprendizaje automático más fáciles y populares. Es un método estadístico que se utiliza para el análisis predictivo. En esta sección aprenderás sobre los algoritmos paramétricos de aprendizaje automático y los fundamentos del modelo de regresión lineal.

Modelo de Regresión Lineal

En el primer curso de este paso, Fundamentos del Aprendizaje Automático, recorrimos el flujo de trabajo completo del aprendizaje automático utilizando el algoritmo de K-nearest neighbors . El algoritmo de K-nearest neighbors funciona encontrando ejemplos etiquetados similares del conjunto de entrenamiento para cada instancia del conjunto de prueba y los utiliza para predecir la etiqueta. El algoritmo de K-nearest neighbors se conoce como un algoritmo de aprendizaje basado en instancias porque se basa completamente en instancias anteriores para hacer predicciones. El algoritmo de K-nearest neighbors no trata de entender o capturar la relación entre las columnas de características y la columna objetivo.

Dado que todo el conjunto de datos de entrenamiento se utiliza para encontrar los vecinos más cercanos de una nueva instancia para hacer predicciones de etiquetas, este algoritmo no se adapta bien a conjuntos de datos medianos y grandes. Si tenemos un millón de instancias en nuestro conjunto de datos de entrenamiento y queremos hacer predicciones para cien mil nuevas instancias, tendríamos que ordenar el millón de instancias del conjunto de entrenamiento por distancia euclídea para cada instancia. El siguiente diagrama proporciona una visión general de la complejidad de K-nearest neighbors.

Modelo de Regresión Lineal

Proceso de Entrenamiento

Datos de Entrenamiento

Características	Etiquetas
x1 x2 x3 xn	y

Características
Seleccionadas

Proceso de Prueba

Hecho, no hay trabajo computacional en el proceso de entrenamiento

Datos de la Prueba

Características	Etiquetas
x1 x2 x3 xn	y

Utilizar las instancias del conjunto de entrenamiento para hacer predicciones

La predicción es computacionalmente intensiva $O(n^2)$ por predicción

Características	Etiquetas
x1 x2 x3 xn	y

Ordenado por distancia de la prueba de distancia

- Para cada instancia de prueba:
1. Ordenar n distancias de entrenamiento a partir de la Distancia Euclídea

Ordenar: $O(n^2)$

Figura 9. Proceso de entrenamiento

Figura 10. Proceso de prueba - Paso 1

Modelo de Regresión Lineal

- Para cada instancia de prueba
2. Seleccionar las 2 instancias de entrenamiento más importantes del conjunto de entrenamiento ordenado y promediar sus etiquetas

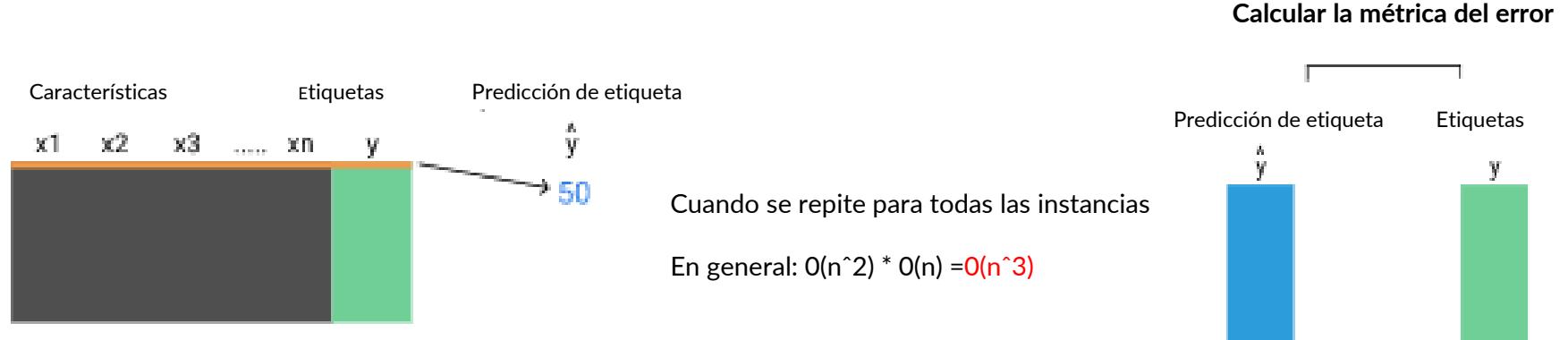


Figura 11. Proceso de prueba - Paso 2

Figura 12. Proceso de prueba - Paso 3

Modelo de Regresión Lineal

En su lugar, debemos aprender sobre los enfoques de aprendizaje automático paramétrico, como la regresión lineal y la regresión logística. A diferencia del algoritmo k-próximo, el resultado del proceso de entrenamiento de estos algoritmos de aprendizaje automático es una función matemática que se aproxima lo mejor posible a los patrones del conjunto de entrenamiento. En el aprendizaje automático, esta función suele denominarse modelo.

En este curso, exploraremos el modelo de aprendizaje automático más utilizado: el modelo de regresión lineal. Los enfoques paramétricos de aprendizaje automático funcionan haciendo suposiciones sobre la relación entre las características y la columna objetivo. En la regresión lineal, la relación aproximada entre las columnas de características y la columna objetivo se expresa como una ecuación de regresión lineal:

$$y = a_1x_1 + a_2x_2 + a_3x_3 + \dots + a_nx_n$$

El siguiente diagrama ofrece una visión general del proceso de aprendizaje automático para la regresión lineal. Por ahora, el objetivo no es entender todo el proceso, sino más bien comparar y contrastar con el enfoque no paramétrico **k-nearest neighbors**.

Modelo de Regresión Lineal

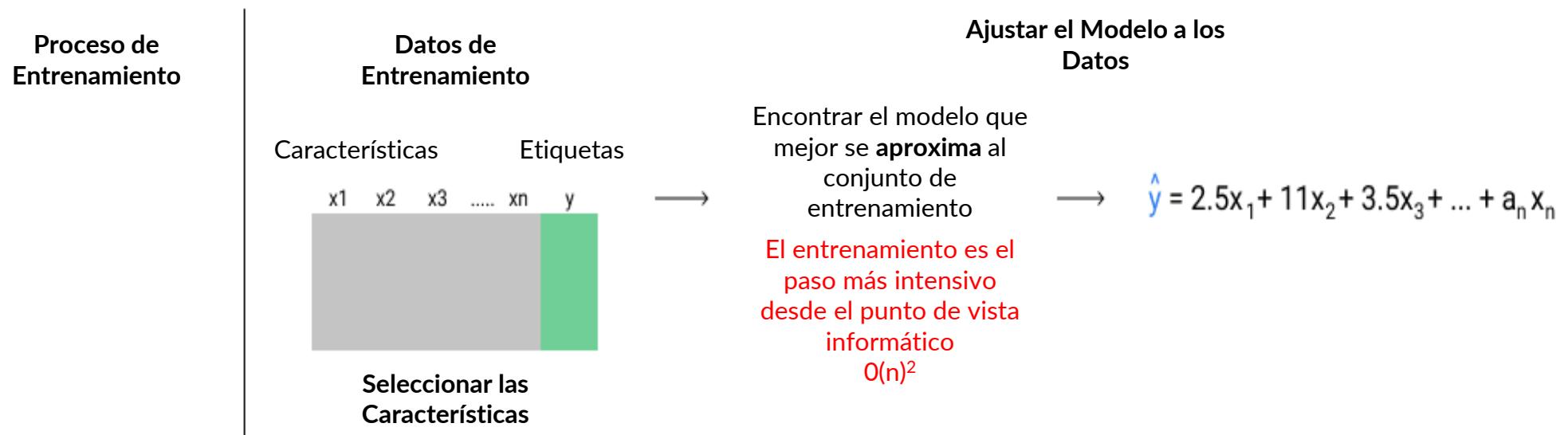


Figura 13. Proceso de aprendizaje automático para la regresión lineal-Proceso de entrenamiento

Modelo de Regresión Lineal

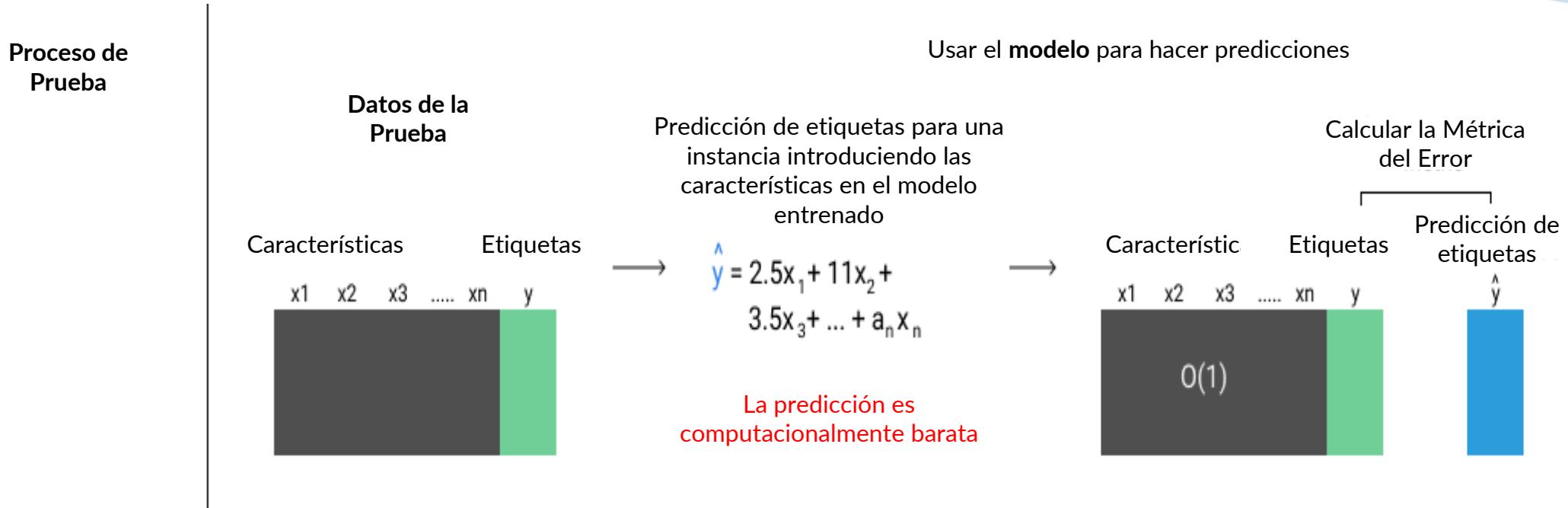


Figura 14. Proceso de aprendizaje automático para la regresión lineal-Proceso de prueba

Modelo de Regresión Lineal

En esta misión, vamos a proporcionar una visión general de cómo utilizamos un modelo de regresión lineal para hacer predicciones. Utilizaremos scikit-learn para el proceso de entrenamiento del modelo, de modo que podamos centrarnos en ganar intuición para el enfoque de aprendizaje basado en modelos para el aprendizaje automático.

En misiones posteriores de este curso, nos adentraremos en las matemáticas que hay detrás de cómo se ajusta un modelo al conjunto de datos, cómo seleccionar y transformar las características, y mucho más.

Selección de Características

En el flujo de trabajo del aprendizaje automático, una vez que hemos seleccionado el modelo que queremos utilizar, la selección de las características adecuadas para ese modelo es el siguiente paso importante. En esta misión, exploraremos cómo utilizar la correlación entre las características y la columna objetivo, la correlación entre las características y la varianza de las características para seleccionarlas. Seguiremos trabajando con el mismo conjunto de datos(dataset) de viviendas de la última misión.

Nos centraremos específicamente en la selección de las columnas de características que no tienen valores perdidos o que no necesitan ser transformadas para ser útiles (por ejemplo, columnas como Año de Construcción (Year Built) y Año de Remodelación (Year Remod/Add)). Exploraremos cómo tratar ambos aspectos en una misión posterior de este curso.

Para empezar, veamos qué columnas entran en alguna de estas dos categorías.

Selección de Funciones

Instrucciones

- Lea **AmesHousing.txt** en un dataframe llamado **data**. Asegurese de separar el delimitador **\t**
- Cree un dataframe llamado **train**, el cual contenga las primeras 1460 filas de **data**
- Cree un dataframe llamado **test**, el cual contenga el resto de las filas de **data**
- Seleccione las columnas **integer** y **float** de **train** y asignelas a las variables **numerical_train**
- Elimine las siguientes columnas de **numerical_train**:
 - **PID** (Place ID no es útil para modelar)
 - **Year Built**
 - **Year Remod/Add**
 - **Garage Yr Blt**
 - **Mo Sold**
 - **Yr Sold**
- Calcule el número de valores perdidos de cada columna en **numerical_train**. Cree un objeto Series donde el index esté hecho de columnas de nombres y los valores asociados sean los números de los valores faltantes
- Asigne este objeto Series a **null_series**. Seleccione el subconjunto **null_series** para dejar solo las columnas con sin valores faltantes, y asigne la objeto Series resultante a **full_cols_series**
- Muestre **full_cols_series** usando la función **print()**

Selección de Funciones

Soluciones

```
1 import pandas as pd
2 data = pd.read_csv('AmesHousing.txt', delimiter="\t")
3 train = data[0:1460]
4 test = data[1460:]
5 # Default code
6 import pandas as pd
7 data = pd.read_csv('AmesHousing.txt', delimiter="\t")
8 train = data[0:1460]
9 test = data[1460:]
10
11 # Solution code
12 numerical_train = train.select_dtypes(include=['int',
13 'float'])
13 numerical_train = numerical_train.drop(['PID', 'Year
Built', 'Year Remod/Add', 'Garage Yr Blt', 'Mo Sold',
'Yr Sold'], axis=1)
14 null_series = numerical_train.isnull().sum()
15 full_cols_series = null_series=null_series == 0]
16 print(full_cols_series)
```

Descenso de Gradientes

En las misiones anteriores, aprendimos cómo el modelo de regresión lineal estima la relación entre las columnas de características y la columna objetivo y cómo podemos utilizarlo para hacer predicciones. En esta misión y en la siguiente, discutiremos las dos formas más comunes de encontrar los valores óptimos de los parámetros para un modelo de regresión lineal. Cada combinación de valores de parámetros únicos forma un modelo de regresión lineal único, y el proceso de encontrar estos valores óptimos se conoce como ajuste del modelo.

En ambos enfoques del ajuste del modelo, trataremos de minimizar la siguiente función:

$$MSE = \frac{1}{n} \sum_{i=1}^n (\hat{y}_i - y_i)^2$$

Esta función es el error cuadrático medio entre las etiquetas predichas realizadas con un modelo determinado y las etiquetas verdaderas. El problema de elegir un conjunto de valores que minimicen o maximicen otra función se conoce como problema de optimización (optimization problem). Para intuir el proceso de optimización, empecemos con un modelo de regresión lineal de un solo parámetro:

$$\hat{y} = a_1 x_1$$

Descenso de Gradientes

Nótese que esto es diferente de un modelo de regresión lineal simple, que en realidad tiene dos parámetros : a_0 y a_1 .

$$\hat{y} = a_1 x_1 + a_0$$

Utilicemos la columna Gr Liv Area para el parámetro único:

$$SalePrice = a_1 * GrLivArea$$

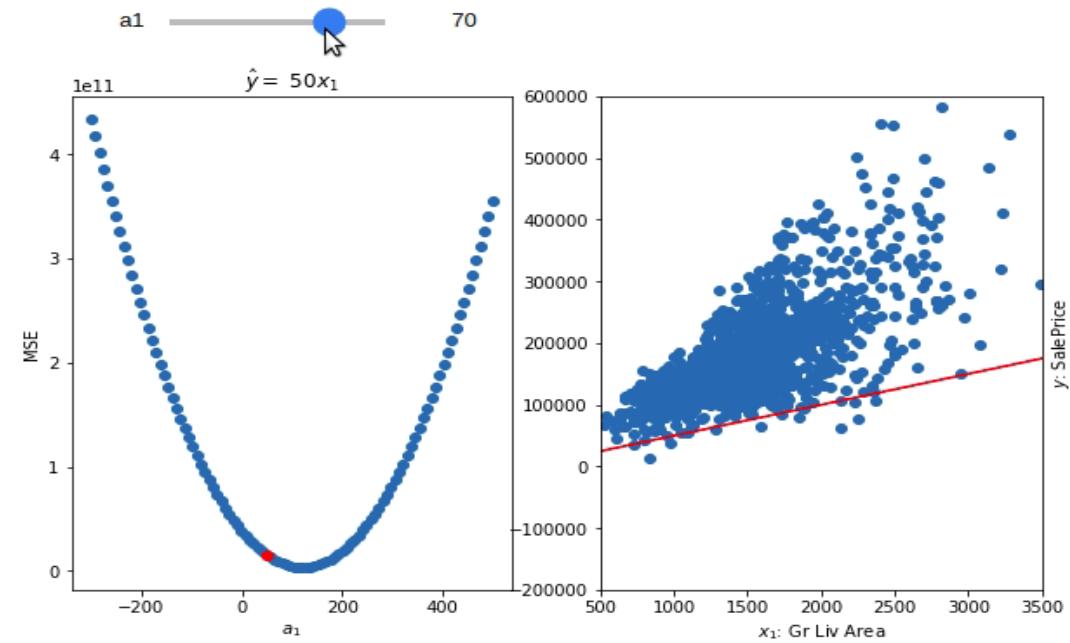


Figura 15. Modelo de regresión lineal simple para el precio de venta

Mínimos Cuadrados Ordinarios

En la última misión, exploramos una técnica iterativa para el ajuste de modelos denominada descenso de gradiente. El algoritmo de descenso de gradiente requiere múltiples iteraciones para converger en los valores óptimos de los parámetros y el número de iteraciones depende en gran medida de los valores iniciales de los parámetros y de la tasa de aprendizaje que seleccionemos.

En esta misión, exploraremos una técnica llamada estimación por **mínimos cuadrados ordinarios** o estimación OLS para abreviar. A diferencia del descenso de gradiente, la estimación OLS proporciona una fórmula clara para calcular directamente los valores óptimos de los parámetros que minimizan la función de coste. Para entender la estimación OLS, necesitamos primero enmarcar nuestro problema de regresión lineal en forma de matriz. La mayoría de las veces hemos trabajado con la siguiente forma del modelo de regresión lineal:

$$\hat{y} = a_0 + a_1x_1 + a_2x_2 + \dots + a_nx_n$$

<https://app.dataquest.io/login?target-url=%2Fm%2F238%2Forinary-least-squares>

Mínimos Cuadrados Ordinarios

Mientras que esta forma representa la relación entre las características ($x_1 \times 1$ a $x_n \times n$) y la columna objetivo (y) bien cuando hay sólo unos pocos valores de los parámetros, no escala bien cuando tenemos cientos de parámetros. Si recuerdas el curso [de Álgebra Lineal para el Aprendizaje Automático](#), exploramos cómo la notación matricial nos permite representar y razonar mejor sobre un sistema lineal con muchas variables. Con esto en mente, aquí está la forma matricial de nuestro modelo de regresión lineal:

$$Xa = \hat{y}$$

Donde X es una matriz que representa las columnas del conjunto de entrenamiento que utiliza nuestro modelo, a es un vector que representa los valores de los parámetros, y \hat{y} es el vector de predicciones. Aquí hay un diagrama con algunos valores de muestra para cada uno:

	<u>Training Data</u>	<u>Parameters</u>	<u>Labels</u>
m rows	n columns	n columns	
	$x_1 \quad x_2 \dots x_n$ $1 \quad 99 \quad 50 \dots 50$ 1 95 31 ... 31 1 30 20 ... 20 1 70 15 ... 15	$a_0 \quad a_1 \quad a_2 \dots a_n$	$=$ $\begin{bmatrix} 100 \\ 250 \\ 108 \\ \dots \\ 19 \end{bmatrix}$
	X	a	\hat{y}

Mínimos Cuadrados Ordinarios

Ahora que hemos comprendido la representación matricial del modelo de regresión lineal, echemos un vistazo a la fórmula de estimación OLS que da como resultado el vector óptimo a:

$$a = (X^T X)^{-1} X^T y$$

Empecemos por calcular la estimación OLS para encontrar los mejores parámetros de un modelo con las siguientes características:

```
features = ['Wood Deck SF', 'Fireplaces', 'Full Bath',
'1st Flr SF', 'Garage Area',
'Gr Liv Area', 'Overall Qual']
```

En las siguientes pantallas, nos sumergiremos en la derivación matemática de la técnica de estimación OLS. Es importante tener en cuenta que lo más probable es que nunca se implemente esta técnica en un papel de ciencia de datos y en su lugar se utilizará una implementación existente y eficiente (scikit-learn utiliza OLS bajo el capó cuando se llama `fit()` en una instancia [LinearRegression](#)).

Mínimos Cuadrados Ordinarios

Instrucciones

- Cree un dataframe, **X**, donde:
 - Su número de filas es el mismo que el del **train** (definido en el código de visualización)
 - La primera columna se llama **bias** y está poblada de **1s** en todo momento
 - Las siguientes columnas son las de las **features** del **train**, en el mismo orden
- Seleccionar la columna **SalePrice** (Precio de venta) del conjunto de entrenamiento y asignarla a **y**
- Utilice la fórmula de estimación OLS para obtener los valores óptimos de los parámetros. Guarde la estimación en la variable **ols_estimation**

Mínimos Cuadrados Ordinarios

Soluciones

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 import pandas as pd
4
5 data = pd.read_csv('AmesHousing.txt', delimiter="\t")
6 train = data[0:1460]
7 test = data[1460:]
8
9 features = ['Wood Deck SF', 'Fireplaces', 'Full Bath',
10             '1st Flr SF', 'Garage Area',
11             'Gr Liv Area', 'Overall Qual']
12 X = train[features]
13 X['bias'] = 1
14 X = X[['bias']+features]
15 first_term = np.linalg.inv(
16     np.dot(
17         np.transpose(X),
18         X
19     )
20 )
21 second_term = np.dot(
22     np.transpose(X),
23     y
24 )
25 ols_estimation = np.dot(first_term, second_term)
26 print(ols_estimation)
```

Características de Procesamiento y Transformación

Para entender cómo funciona la regresión lineal, nos hemos atascado utilizando o descartando algunas características del conjunto de datos de entrenamiento que no contenían valores perdidos y ya estaban en una representación numérica conveniente. En esta misión, exploraremos cómo transformar algunas de las características restantes para poder utilizarlas en nuestro modelo. En términos generales, el proceso de procesamiento y creación de nuevas características se conoce como ingeniería de características (feature engineering). La ingeniería de características es un poco un arte y tener conocimientos en el dominio específico (en este caso inmobiliario) puede ayudarle a crear mejores características. En esta misión, nos centraremos en algunas estrategias independientes del dominio que funcionan para todos los problemas.

En la primera mitad de esta misión, nos centraremos sólo en las columnas que no contienen valores perdidos, pero que aún no tienen el formato adecuado para ser utilizadas en un modelo de regresión lineal. En la segunda mitad de esta misión, exploraremos algunas formas de tratar los valores perdidos.

Características de Procesamiento y Transformación

Entre las columnas que no contienen valores perdidos, algunos de los problemas más comunes son:

- La columna no es numérica (por ejemplo, un código de zona representado mediante texto)
- La columna es numérica pero no ordinal (por ejemplo, los valores del código postal)
- La columna es numérica pero no es representativa del tipo de relación con la columna objetivo (por ejemplo, los valores del año)

Comencemos por filtrar el conjunto de entrenamiento solo a las columnas que no contienen valores perdidos.

Instrucciones

- Seleccione sólo las columnas del marco de datos de **train** (entrenamiento) que no contengan valores perdidos
- Asigne el marco de datos resultante, que contiene sólo estas columnas, a **df_no_mv**
- Utilice la pantalla de variables para familiarizarse con estas columnas

Características de Procesamiento y Transformación

Soluciones

```
1 import pandas as pd  
2  
3 data = pd.read_csv('AmesHousing.txt', delimiter='\t')  
4 train = data[0:1460]  
5 test = data[1460:]  
6  
7 train_null_counts = train.isnull().sum()  
8 print(train_null_counts)  
9 df_no_mv =  
    train[train_null_counts[train_null_counts==0].index]
```

Proyecto Guiado: Pronóstico de los Precios de Venta de la Vivienda

En este curso, comenzamos construyendo la intuición para el aprendizaje basado en modelos, exploramos cómo funcionaba el modelo de regresión lineal, comprendimos cómo funcionaban los dos enfoques diferentes para el ajuste de modelos y algunas técnicas para limpiar, transformar y seleccionar características. En este proyecto guiado, podrás practicar lo que has aprendido en este curso explorando formas de mejorar los modelos que hemos construido. Trabajarás con datos de vivienda de la ciudad de Ames, Iowa, Estados Unidos, desde 2006 hasta 2010. Puedes leer más sobre por qué se recogieron los datos [aquí](#). También puedes leer sobre las diferentes columnas de los datos [aquí](#).

[https://app.dataquest.io/login?target-
url=%2Fm%2F40%2Fguided-project%253A-predicting-house-
sale-prices](https://app.dataquest.io/login?target-url=%2Fm%2F40%2Fguided-project%253A-predicting-house-sale-prices)

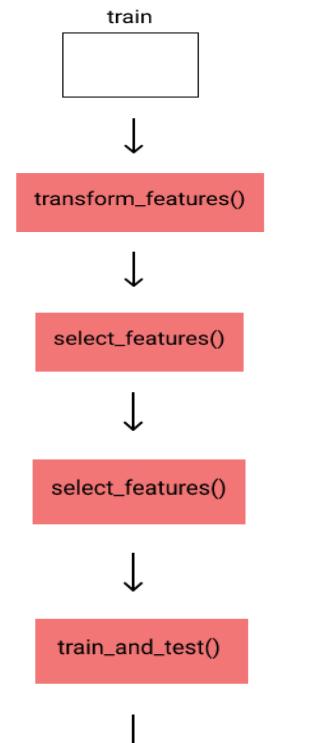


Figura 16. Cadena de funciones para la regresión lineal

Proyecto Guiado: Pronóstico de los Precios de Venta de la Vivienda

Instrucciones

- Importar pandas, matplotlib y numpy en el entorno. Importa también las clases que necesites de scikit-learn
- Lee **AmesHousing.tsv** en un marco de datos de pandas
- Para las siguientes funciones, recomendamos crearlas en las primeras celdas del cuaderno. De esta manera, puedes añadir celdas al final del cuaderno para hacer experimentos y actualizar las funciones en estas celdas.
 - Cree una función llamada **transform_features()** que, por ahora, sólo devuelve el marco de datos de **train**
 - Crea una función llamada **select_features()** que, por ahora, sólo devuelve las columnas **Gr Liv Area** y **SalePrice** del marco de datos de **train**
 - Cree una función llamada **train_and_test()** que, por ahora:
 - Selecciona las primeras **1460** filas de los **datos** y las asigna al **train**
 - Seleccione las filas restantes de los **datos** y las asigne a **test**
 - Entrene un modelo utilizando todas las columnas numéricas excepto la columna **SalePrice** - Precio de venta - (la columna objetivo) del marco de datos devuelto por **select_features()**
 - Prueba el modelo en el conjunto de pruebas y devuelve el valor de RMSE

Proyecto Guiado: Pronóstico de los Precios de Venta de la Vivienda

Soluciones

Puede encontrar el cuaderno de soluciones para este proyecto guiado [aquí](#).

V Aprendizaje Automático en Python



Regresión Logística

La **regresión lineal** es una técnica de aprendizaje automático supervisado que funciona bien cuando la columna objetivo que intentamos predecir, la variable dependiente, está ordenada y es continua. En cambio, si la columna objetivo contiene valores discretos, la regresión lineal no es apropiada.

En esta misión, exploraremos cómo construir un modelo de predicción para este tipo de problemas, que se conocen como **problemas de clasificación**. En la clasificación, nuestra columna objetivo tiene un conjunto limitado de valores posibles que representan diferentes categorías para una fila. Utilizamos números enteros para representar las diferentes categorías para seguir utilizando funciones matemáticas para describir cómo las variables independientes se corresponden con la variable dependiente.

He aquí algunos ejemplos de problemas de clasificación:

Problema	Ejemplos de Características	Tipo	Categorías	Categorías Numéricas
¿Debemos aceptar a este estudiante sobre la base de su solicitud de posgrado?	GPA de la Universidad, Puntuación SAT, Calidad de las Recomendaciones	Binaria	No Aceptar, Aceptar	0,1
¿Cuál es el tipo de sangre más probable de los hijos de 2 padres?	Tipo de sangre del padre 1, Tipo de sangre del padre 2	Multi-clase	A, B, AB, O	1, 2, 3, 4

Regresión Logística

Por ahora nos centraremos en la clasificación binaria, donde las únicas dos opciones de valores son:

- **0 para la condición de Falso**
- **1 para la condición Verdadera**

Antes de aprender más sobre la clasificación, vamos a entender los datos.

Introducción a la Evaluación de Clasificadores Binarios

En la lección anterior, aprendimos sobre la clasificación, la regresión logística y cómo utilizar scikit-learn para ajustar un modelo de regresión logística a un conjunto de datos sobre admisiones a escuelas de posgrado. Seguiremos trabajando con el conjunto de datos, que contiene datos sobre 644 solicitudes con las siguientes columnas:

- **GRE - Puntuación del solicitante en el Graduate Record Exam, una prueba generalizada para futuros estudiantes de posgrado**
 - La puntuación oscila entre 200 y 800
- **GPA - Promedio de notas de la universidad**
 - Continuo entre 0,0 y 4,0
- **Admit - Valor binario**
 - Valor binario, 0 o 1, donde 1 significa que el solicitante fue admitido en el programa y 0 significa que fue rechazado

Aquí tiene una vista previa del conjunto de datos:

admit	gpa	gre
0	3.177277	594.102992
0	3.412655	631.528607
0	2.728097	553.714399
0	3.093559	551.089985
0	3.141923	537.184894

Introducción a la Evaluación de Clasificadores Binarios

Utilicemos el modelo de regresión logística de la última misión para predecir las etiquetas de clase para cada observación en el conjunto de datos y añadamos estas etiquetas al marco de datos en una columna separada.

Instrucciones

- Utilice el método LogisticRegression **predict** para devolver la etiqueta para cada observación en el conjunto de datos, las **admissions**. Asigne la lista devuelta a labels
- Añada una nueva columna al marco de datos de **admissions** denominada etiqueta predicha que contenga los valores de las etiquetas
- Utilice el método Series **value_counts** y la función **print** para mostrar la distribución de los valores en la columna **predicted_label**
- Utilice el método dataframe **head** y la función de impresión para mostrar las cinco primeras filas de **admissions**

Introducción a la Evaluación de Clasificadores Binarios

Soluciones

```
1 import pandas as pd
2 import matplotlib.pyplot as plt
3 from sklearn.linear_model import LogisticRegression
4
5 admissions = pd.read_csv("admissions.csv")
6 model = LogisticRegression()
7 model.fit(admissions[["gpa"]], admissions["admit"])
8 admissions = pd.read_csv("admissions.csv")
9 model = LogisticRegression()
10 model.fit(admissions[["gpa"]], admissions["admit"])
11
12 labels = model.predict(admissions[["gpa"]])
13 admissions["predicted_label"] = labels
14 print(admissions["predicted_label"].value_counts())
15 print(admissions.head())
```

Clasificación Multiclas

El conjunto de datos con el que vamos a trabajar contiene información sobre varios coches. Para cada coche tenemos información sobre los aspectos técnicos del vehículo, como la cilindrada del motor, el coche, las millas por galón y la aceleración del coche. Esta información permite predecir el origen del vehículo, ya sea Norteamérica, Europa o Asia. A diferencia de nuestros anteriores conjuntos de datos de clasificación, tenemos tres categorías entre las que elegir, lo que hace que nuestra tarea sea más difícil.

He aquí un avance de los datos:

18.0	8	307.0	130.0	3504.	12.0	70	1
"chevrolet chevelle malibu"							
15.0	8	350.0	165.0	3693.	11.5	70	1
"buick skylark 320"							
18.0	8	318.0	150.0	3436.	11.0	70	1
"plymouth satellite"							

Clasificación Multiclasificación

El conjunto de datos está alojado en la Universidad de California Irvine en su [repositorio de aprendizaje automático \(machine learning repository\)](#). El repositorio de aprendizaje automático de la UCI contiene muchos conjuntos de datos pequeños que son útiles cuando se ensucian las manos con el aprendizaje automático.

Verás que la **carpeta de datos (Data Folder)** contiene diferentes archivos. Trabajaremos con [auto-mpg.data](#), que omite las 8 filas que contienen valores perdidos para la eficiencia del combustible (columna mpg). Hemos convertido estos datos en un archivo CSV llamado auto.csv para usted.

Estas son las columnas del conjunto de datos:

- **mpg** -- Millas por galón, Continuo
- **cylinders** -- Número de cilindros en el motor, Entero, Ordinal y Categórico
- **displacement** -- Tamaño del motor, Continuo
- **horsepower** -- Caballos de fuerza producidos, Continuo
- **weight** -- Peso del coche, Continuo
- **acceleration** -- Aceleración, Continuo
- **year** -- Año de construcción del coche, Entero y Categórico
- **origin** -- Entero y categórico. 1: Norteamérica, 2: Europa, 3: Asia

Clasificación Multiclas

Instrucciones

- Importa la librería Pandas y lee **auto.csv** en un Dataframe llamado **cars**
- Utilice el método **Series.unique()** para asignar los elementos únicos de la columna **origin** a **regiones unique**. A continuación, utilice la función de **print** para mostrar **unique_regions**

Soluciones

```
1 import pandas as pd  
2 cars = pd.read_csv("auto.csv")  
3 print(cars.head())  
4 unique_regions = cars["origin"].unique()  
5 print(unique_regions)
```

Sobreajustes

Al explorar la regresión, hemos mencionado brevemente el sobreajuste y los problemas que puede causar. En esta lección, exploraremos cómo identificar el sobreajuste y qué se puede hacer para evitarlo. Para explorar el sobreajuste, utilizaremos un conjunto de datos sobre coches que contiene 7 características numéricas que podrían tener un efecto sobre la eficiencia del combustible de un coche:

- **cylinders** -- el número de cilindros en el motor
- **displacement** -- la cilindrada del motor
- **horsepower** -- los caballos de fuerza del motor
- **weight** -- el peso del coche
- **acceleration** -- la aceleración del coche
- **model year** --el año en que salió al mercado el modelo del coche (por ejemplo, 70 corresponde a 1970)
- **origin** --dónde se fabricó el coche (0 si es Norteamérica, 1 si es Europa, 2 si es Asia)

- La columna **mpg** es nuestra columna objetivo y queremos predecir usando las otras características
- El conjunto de datos está alojado en la Universidad de California Irvine en su [repositorio de aprendizaje automático \(machine learning repository\)](#). Trabajaremos con [auto-mpg.data](#), que omite las 8 filas que contienen valores perdidos para la eficiencia del combustible (columna **mpg**)
- El código de inicio importa Pandas, lee los datos en un marco de datos y limpia algunos valores desordenados. Explora el conjunto de datos para familiarizarte con él
- Al leer el código de inicio, es posible que descubras una sintaxis diferente. Si ejecutas el código localmente en Jupyter Notebook o Jupyter Lab, notarás una advertencia [SettingWithCopy](#). Esto no impedirá que tu código se ejecute correctamente, pero te avisa de que la operación que estás realizando está intentando establecerse en una copia de un slice de un dataframe. Para resolver esto, se considera una buena práctica incluir **.copy()** siempre que se realicen operaciones en un marco de datos

Fundamentos de Agrupación (Clustering)

Hasta ahora, hemos hablado de la regresión y la clasificación. Ambos son tipos de aprendizaje automático supervisado (supervised machine learning). En el aprendizaje supervisado, se puede entrenar un algoritmo para predecir una variable desconocida a partir de variables conocidas. Otro tipo importante de aprendizaje automático es el llamado aprendizaje no supervisado (unsupervised learning). En el aprendizaje no supervisado, no intentamos predecir nada. En su lugar, tratamos de encontrar patrones en los datos.

Utilizaremos un algoritmo llamado k-means clustering para dividir nuestros datos en clusters. k-means clustering utiliza la distancia euclídea para formar clusters de senadores similares. En una lección posterior nos adentraremos en la teoría del clustering de k-means y construiremos el algoritmo desde cero. Por ahora, es importante entender la agrupación a un nivel alto, así que aprovecharemos la biblioteca scikit-learn para entrenar un modelo k-means.

El algoritmo k-means agrupa a los senadores que votan de forma similar en los proyectos de ley en clusters. A cada grupo se le asigna un centro y se calcula la distancia euclídea de cada senador al centro. Los senadores se asignan a los clusters en función de la proximidad. A partir de nuestros conocimientos previos, pensamos que los senadores se agrupan según las líneas de partido.

Fundamentos de Agrupación (Clustering)

El algoritmo k-means requiere que especifiquemos el número de agrupaciones (clusters) inicialmente, porque sospechamos que las agrupaciones se producirán según las líneas de los partidos, y la gran mayoría de los senadores son republicanos o demócratas. Elegiremos 2 para nuestro número de agrupaciones.

Utilizaremos la clase **KMeans** de scikit-learn para realizar la agrupación, porque no estamos prediciendo nada. No hay riesgo de sobreajuste, así que entrenaremos nuestro modelo en todo el conjunto de datos. Después del entrenamiento, podremos determinar las etiquetas de los clusters que indican el cluster de cada senador.

Podemos inicializar el modelo así:

```
kmeans_model = KMeans(n_clusters=2, random_state=1)
```

Fundamentos de Agrupación (Clustering)

El código anterior inicializa el modelo k-means con **2** clusters y un estado aleatorio de **1** para permitir que se reproduzcan los mismos resultados cada vez que se ejecute el algoritmo.

Entonces podremos utilizar el método **fit transform()** para ajustar el modelo a los **votos** y obtener la distancia de cada senador a cada cluster. El resultado será similar al del ejemplo siguiente:

```
array([[ 3.12141628,  1.3134775 ],
       [ 2.6146248 ,  2.05339992],
       [ 0.33960656,  3.41651746],
       [ 3.42004795,  0.24198446],
       [ 1.43833966,  2.96866004],
       [ 0.33960656,  3.41651746],
       [ 3.42004795,  0.24198446],
       [ 0.33960656,  3.41651746],
       [ 3.42004795,  0.24198446],
       [ 0.31287498,  3.30758755],
       ...]
```

Se trata de un array NumPy con dos columnas. La primera columna es la distancia euclídea de cada senador al primer cluster y la segunda columna es la distancia euclídea al segundo cluster. Los valores de las columnas indican lo "lejos" que está el senador de cada clúster. Cuanto más lejos esté el grupo, menos se alinearán el historial de votos del senador con el del grupo.

Agrupación (Clustering) de K-means

En la cobertura mediática de la NBA, los periodistas deportivos suelen centrarse en unos pocos jugadores y crear historias sobre la singularidad de las estadísticas de estos jugadores. Como científicos de datos, es probable que seamos escépticos en cuanto a la singularidad de cada jugador. En esta lección, utilizaremos la ciencia de los datos para explorar esta idea observando un conjunto de datos de información de jugadores de la temporada 2013-2014. Estas son las columnas con las que trabajaremos:

- **player** - el nombre del jugador
- **pos** - la posición del jugador
- **g** - el número de partidos jugados
- **pts** - el total de puntos anotados por el jugador
- **fg.** - el porcentaje de tiros de campo
- **ft.** - el porcentaje de tiros libres

Consulta el glosario de [Basketball Reference](#) para obtener una explicación de cada columna.

Proyecto Guiado: Predicción de la Bolsa

Predicción de la Bolsa de valores. Si no has pasado por un proyecto guiado en esta interfaz, aquí tienes una rápida introducción. Puedes utilizar estas herramientas en línea:

- [Google colaboratory](#)
- [Cocalc Cálculo Colaborativo y Ciencia de Datos](#)

Para obtener una visión más completa, te recomendamos que realices el proyecto guiado [Working With Data Downloads](#). En este proyecto, trabajarás con datos del índice [S&P500](#). El S&P500 es un índice bursátil. Antes de entrar en lo que es un índice, tendremos que empezar con los fundamentos del mercado de valores.

Proyecto Guiado: Predicción de la Bolsa

Algunas empresas cotizan en bolsa, lo que significa que cualquiera puede comprar y vender sus acciones en el mercado abierto. Una acción da derecho al propietario a cierto control sobre la dirección de la empresa y a un porcentaje (o parte) de los beneficios de la misma. Cuando se compran o venden acciones, es lo que comúnmente se conoce como negociar una acción.

El precio de una acción se basa en la oferta y la demanda de una acción determinada. Por ejemplo, las acciones de Apple tienen un precio de 120 dólares por acción en diciembre de 2015 - <http://www.nasdaq.com/symbol/aapl>. Una acción que tiene menos demanda, como Ford Motor Company, tiene un precio más bajo -- <http://finance.yahoo.com/q?s=F>. En el precio de las acciones también influyen otros factores, como el número de acciones que ha emitido una empresa.

Las acciones se negocian a diario y el precio puede subir o bajar desde el principio de un día de negociación hasta el final en función de la demanda. Las acciones que tienen más demanda, como Apple, se negocian con más frecuencia que las acciones de empresas más pequeñas.

Proyecto Guiado: Predicción de la Bolsa

Los índices reúnen los precios de varios valores y permiten ver la evolución del mercado en su conjunto. Por ejemplo, el Promedio Industrial Dow Jones (the Dow Jones Industrial Average) agrupa los precios de las acciones de 30 grandes empresas estadounidenses. El índice S&P500 agrupa las cotizaciones de 500 grandes empresas. Cuando un fondo índice sube o baja, se puede decir que el mercado primario o el sector que representa está haciendo lo mismo. Por ejemplo, si el precio del Promedio Industrial Dow Jones baja un día, se puede decir que las acciones estadounidenses en general bajaron (es decir, la mayoría de las acciones estadounidenses bajaron de precio).

Utilizará los datos históricos del precio del índice S&P500 para hacer predicciones sobre los precios futuros. Predecir si un índice sube o baja ayuda a pronosticar cómo se comporta el mercado de valores en su conjunto. Dado que las acciones tienden a correlacionarse con el rendimiento de la economía en su conjunto, también puede ayudar a realizar previsiones económicas.

Hay miles de operadores que ganan dinero comprando y vendiendo fondos cotizados (Exchange Traded Funds o ETF). Los **ETF** le permiten comprar y vender índices como si fueran acciones. Esto significa que usted podría "comprar" el **ETF** del índice S&P500 cuando el precio es bajo y vender cuando es alto para obtener un beneficio. La creación de un modelo predictivo podría permitir a los operadores ganar dinero en el mercado de valores.

Proyecto Guiado: Predicción de la Bolsa

Las columnas del conjunto de datos son:

Date -- La fecha del registro

Open -- El precio de apertura del día (cuando comienza la negociación)

High -- El precio de negociación más alto del día

Low -- El precio de negociación más bajo del día

Close -- El precio de cierre del día (cuando finaliza la negociación)

Volumen -- El número de acciones negociadas

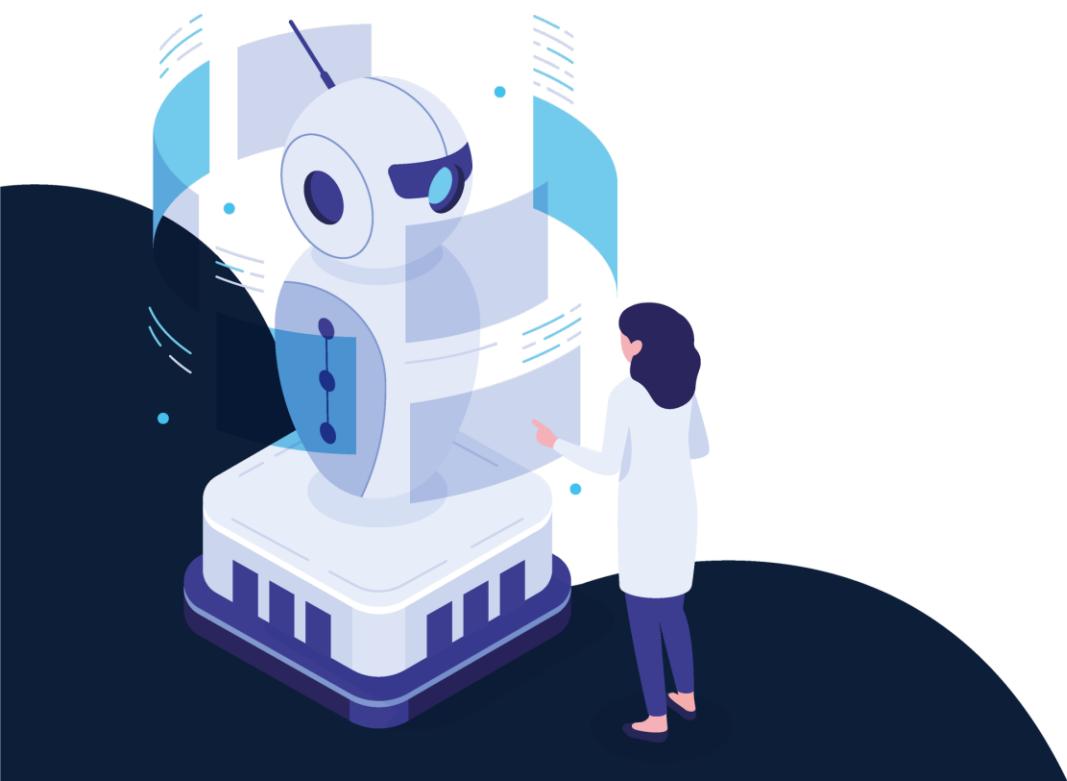
Adj Close -- El precio de cierre diario, ajustado retroactivamente para incluir cualquier acción corporativa

Utilizará este conjunto de datos para desarrollar un modelo predictivo. Entrenará el modelo con datos de 1950-2020 e intentará hacer predicciones de 2013-2015.

Nota: No deberías hacer operaciones con ningún modelo desarrollado en esta guía. Operar con acciones tiene riesgos y nada en esta guía constituye un consejo para operar con acciones.

En esta lección, trabajará con un archivo csv que contiene los precios de los índices. Cada fila del archivo contiene un registro diario del precio del índice S&P500 desde 1950 hasta 2015. El conjunto de datos se almacena en sphist.csv.

VI Árbol de Decisiones



Árbol de Decisiones

- El árbol de decisión es una técnica de aprendizaje supervisado que puede utilizarse tanto para problemas de clasificación como de regresión, pero se prefiere para resolver problemas de clasificación. Se trata de un clasificador estructurado en forma de árbol, en el que los nodos internos representan las características de un conjunto de datos, las ramas representan las reglas de decisión y cada nodo hoja representa el resultado
- En un árbol de decisión, hay dos nodos, que son el nodo de decisión y el nodo hoja. Los nodos de decisión se utilizan para tomar cualquier decisión y tienen múltiples ramas, mientras que los nodos hoja son el resultado de esas decisiones y no contienen más ramas
- Las decisiones o la prueba se realizan sobre la base de las características del conjunto de datos dado
- Se trata de una representación gráfica para obtener todas las posibles soluciones a un problema/decisión en función de unas condiciones dadas
- Se denomina árbol de decisión porque, de forma similar a un árbol, comienza con el nodo raíz, que se expande en otras ramas y construye una estructura similar a un árbol
- Para construir un árbol, se utiliza el algoritmo CART, que significa algoritmo de árbol de clasificación y regresión
- Un árbol de decisión simplemente formula una pregunta y, en función de la respuesta (Sí/No), divide el árbol en subárboles
- El siguiente diagrama explica la estructura general de un árbol de decisión:

Árbol de Decisiones

Nota: Un árbol de decisión puede contener datos categóricos (SÍ/NO) así como datos numéricos.

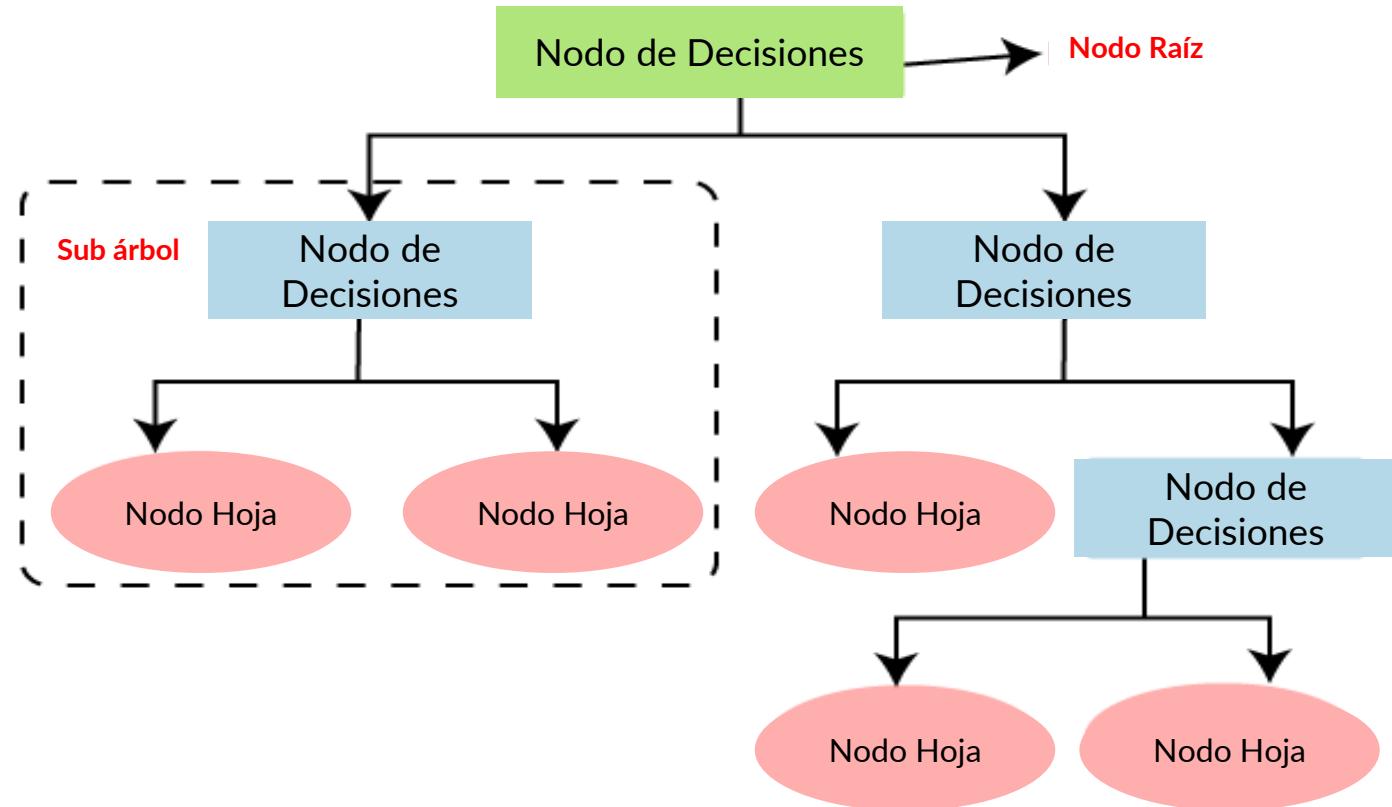


Figure 17. Decision tree diagram

¿Por qué utilizar Árboles de Decisiones?

Hay varios algoritmos en el aprendizaje automático, por lo que la elección del mejor algoritmo para el conjunto de datos y el problema dados es el punto principal que hay que recordar al crear un modelo de aprendizaje automático. A continuación se exponen las dos razones para utilizar el árbol de decisión:

- Los árboles de decisión suelen imitar la capacidad de pensamiento humano al tomar una decisión, por lo que son fáciles de entender
- La lógica detrás del árbol de decisión puede entenderse fácilmente porque muestra una estructura similar a la de un árbol

Terminología de los Árboles de Decisiones

- **Nodo raíz:** El nodo raíz es el punto de partida del árbol de decisión. Representa todo el conjunto de datos, que a su vez se divide en dos o más conjuntos homogéneos
- **Nodo hoja:** Los nodos hoja son el nodo de salida final, y el árbol no se puede segregar más después de obtener un nodo hoja
- **División:** La división es el proceso de dividir el nodo de decisión/nodo raíz en subnodos según las condiciones dadas
- **Rama/Subárbol:** Un árbol formado por la división del árbol
- **Poda:** La poda es el proceso de eliminar las ramas no deseadas del árbol
- **Nodo padre/hijo:** El nodo raíz del árbol se llama nodo padre, y los demás nodos se llaman nodos hijos

Cómo Funciona el Algoritmo del Árbol de Decisiones

En un árbol de decisión, para predecir la clase del conjunto de datos dado, el algoritmo parte del nodo raíz del árbol. Este algoritmo compara los valores del atributo raíz con el atributo del registro (conjunto de datos real) y, basándose en la comparación, sigue la rama y salta al siguiente nodo.

En el siguiente nodo, el algoritmo vuelve a comparar el valor del atributo con los demás subnodos y sigue avanzando. Continúa el proceso hasta llegar al nodo hoja del árbol. El proceso completo puede entenderse mejor utilizando el siguiente algoritmo:

- **Paso 1:** Comienza el árbol con el nodo raíz, llamado S, que contiene el conjunto de datos completo.
- **Paso 2:** Encontrar el mejor atributo en el conjunto de datos utilizando la medida de selección de atributos (ASM)
- **Paso 3:** Dividir S en subconjuntos que contengan los posibles valores de los mejores atributos
- **Paso 4:** Generar el nodo del árbol de decisión que contiene el mejor atributo
- **Paso 5:** Hacer recursivamente nuevos árboles de decisión utilizando los subconjuntos del conjunto de datos creados en el paso 3. Continúe este proceso hasta que se llegue a una etapa en la que no se puedan clasificar más los nodos y se llame al nodo final como nodo hoja

Cómo Funciona el Algoritmo del Árbol de Decisiones

Ejemplo: Supongamos que hay un candidato que tiene una oferta de trabajo y quiere decidir si debe aceptar la oferta o no. Entonces, para resolver este problema, el árbol de decisión comienza con el nodo raíz (atributo Salario por ASM). El nodo raíz se divide en el siguiente nodo de decisión (distancia a la oficina) y un nodo hoja basado en las etiquetas correspondientes. El siguiente nodo de decisión se divide a su vez en un nodo de decisión (instalación de taxi) y un nodo de hoja. Por último, el nodo de decisión se divide en dos nodos de hoja (ofertas aceptadas y ofertas rechazadas). Considere el siguiente diagrama:

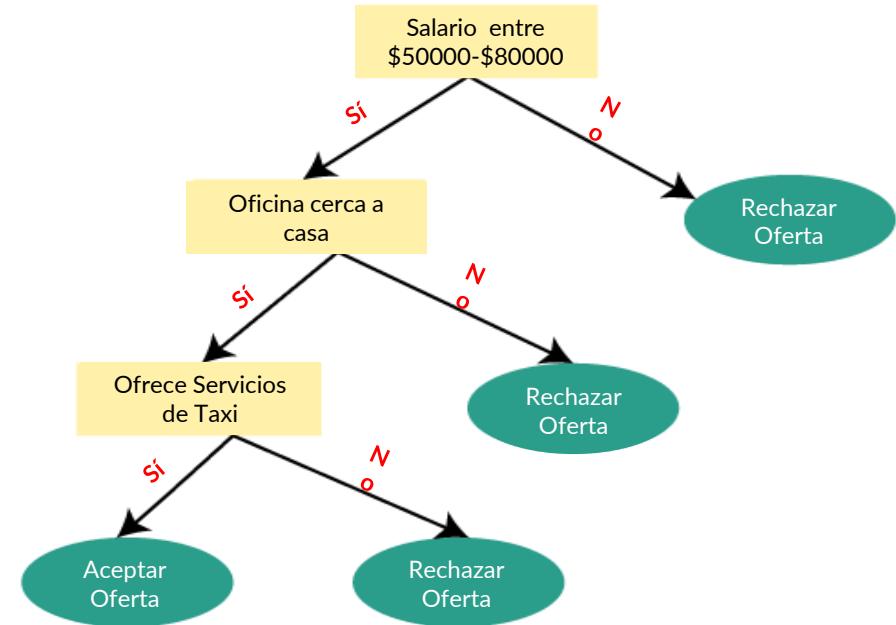


Figura 18. Medidas de Selección de Atributos

Cómo Funciona el Algoritmo del Árbol de Decisiones

Cuando se implementa un árbol de decisión, el principal problema es cómo seleccionar el mejor atributo para el nodo raíz y los subnodos. Por lo tanto, para resolver estos problemas existe una técnica que se denomina medida de selección de atributos o ASM. Con esta medida, podemos seleccionar fácilmente el mejor atributo para los nodos del árbol. Hay dos técnicas populares para ASM, que son:

- Ganancia de información
- Índice de Gini

Cómo Funciona el Algoritmo del Árbol de Decisiones

1. Ganancia de información:

- La ganancia de información es la medida de los cambios en la entropía tras la segmentación de un conjunto de datos basado en un atributo
- Calcula cuánta información nos proporciona una característica sobre una clase
- Según el valor de la ganancia de información, dividimos el nodo y construimos el árbol de decisión
- Un algoritmo de árbol de decisión siempre intenta maximizar el valor de la ganancia de información, y el nodo/atributo que tiene la mayor ganancia de información se divide primero. Se puede calcular mediante la siguiente fórmula: **Ganancia de información= Entropía(S) - [(Media ponderada) * Entropía(cada característica)] or Information Gain= Entropy(S) - [(Weighted Avg) * Entropy(each feature)]**

Entropía: La entropía es una métrica que mide la impureza de un atributo determinado. Especifica la aleatoriedad de los datos. La entropía puede calcularse como:

$$\text{Information Gain} = \text{Entropy}(S) - [(\text{Weighted Avg}) * \text{Entropy}(\text{each feature})]$$

Donde,

- **S= Número total de muestras**
- **P(yes)= probabilidades de si**
- **P(no)= probabilidades de no**

Cómo Funciona el Algoritmo del Árbol de Decisiones

2. Índice de Gini:

- El índice de Gini es una medida de impureza o pureza que se utiliza al crear un árbol de decisión en el algoritmo CART (árbol de clasificación y regresión)
- Un atributo con un índice de Gini bajo debería ser preferido en comparación con un índice de Gini alto
- Sólo crea divisiones binarias, y el algoritmo CART utiliza el índice de Gini para crear divisiones binarias
- El índice de Gini se puede calcular mediante la siguiente fórmula:

$$\text{Gini Index} = 1 - \sum_j p_j^2$$

Poda: Obtención de un Árbol de Decisión Óptimo

La poda (pruning) es un proceso que consiste en eliminar los nodos innecesarios de un árbol para obtener el árbol de decisión óptimo.

Un árbol demasiado grande aumenta el riesgo de sobreajuste, y un árbol pequeño puede no capturar todas las características importantes del conjunto de datos. Por lo tanto, una técnica que disminuye el tamaño del árbol de aprendizaje sin reducir la precisión se conoce como poda. Existen principalmente dos tipos de tecnología de poda de árboles:

- **Poda de complejidad de costes**
- **Poda de error reducido**

Ventajas del Árbol de Decisiones

- Es fácil de entender, ya que sigue el mismo proceso que sigue un ser humano al tomar cualquier decisión en la vida real
- Puede ser muy útil para resolver problemas relacionados con la toma de decisiones
- Ayuda a pensar en todos los posibles resultados de un problema
- Requiere menos limpieza de datos en comparación con otros algoritmos

Desventajas del Árbol de Decisiones

- El árbol de decisión contiene muchas capas, lo que lo hace complejo
- Puede tener un problema de sobreajuste, que puede resolverse con el **algoritmo Random Forest**
- Para más etiquetas de clase, la complejidad computacional del árbol de decisión puede aumentar

Implementación en Python del Árbol de Decisiones

Ahora implementaremos el árbol de decisión utilizando Python. Para ello, utilizaremos el conjunto de datos "user_data.csv", que hemos utilizado en modelos de clasificación anteriores. Utilizando el mismo conjunto de datos, podemos comparar el clasificador de árbol de decisión con otros modelos de clasificación como KNN SVM, LogisticRegression, etc.

Los pasos también seguirán siendo los mismos, que se indican a continuación:

- Paso de pre-procesamiento de datos
- Ajuste de un algoritmo de árbol de decisión al conjunto de entrenamiento
- Predicción del resultado de la prueba
- Precisión de la prueba del resultado (creación de la matriz de confusión)
- Visualización del resultado del conjunto de pruebas

Implementación en Python del Árbol de Decisiones

1. Paso de Pre-procesamiento de Datos:

A continuación se muestra el código para el paso de pre-procesamiento.

```
import numpy as nm  
import matplotlib.pyplot as mtp  
import pandas as pd  
  
#importing datasets  
data_set= pd.read_csv('user_data.csv')  
  
#Extracting Independent and dependent Variable  
x= data_set.iloc[:, [2,3]].values  
y= data_set.iloc[:, 4].values  
  
# Splitting the dataset into training and test set.  
from sklearn.model_selection import train_test_split  
x_train, x_test, y_train, y_test= train_test_split(x, y, test_size= 0.25, random_state=0)  
  
#feature Scaling  
from sklearn.preprocessing import StandardScaler  
st_x= StandardScaler()  
x_train= st_x.fit_transform(x_train)  
x_test= st_x.transform(x_test)
```

Implementación en Python del Árbol de Decisiones

En el código anterior, hemos pre-procesado los datos. Donde hemos cargado el conjunto de datos, que se da como:

Index	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	Male	19	19000	0
1	15810944	Male	35	20000	0
2	15668575	Female	26	43000	0
3	15603246	Female	27	57000	0
4	15804002	Male	19	76000	0
5	15728773	Male	27	58000	0
6	15598044	Female	27	84000	0
7	15694829	Female	32	150000	1
8	15600575	Male	25	33000	0
9	15727311	Female	35	65000	0
10	15570769	Female	26	80000	0
11	15606274	Female	26	52000	0
12	15746139	Male	20	86000	0
13	15704987	Male	32	18000	0
14	15628972	Male	18	82000	0
15	15697686	Male	29	80000	0

Implementación en Python del Árbol de Decisiones

2. Ajuste de un algoritmo de árbol de decisión al conjunto de entrenamiento

Ahora vamos a ajustar el modelo al conjunto de entrenamiento. Para ello, importaremos la clase **DecisionTreeClassifier** de la biblioteca **sklearn.tree**. A continuación se muestra el código para ello:

```
#Fitting Decision Tree classifier to the training set
From sklearn.tree import DecisionTreeClassifier
classifier= DecisionTreeClassifier(criterion='entropy', random_state=0)
classifier.fit(x_train, y_train)
```

En el código anterior, hemos creado un objeto clasificador, en el que hemos pasado dos parámetros principales:

- "criterion" = "entropy": El criterio se utiliza para medir la calidad de la división, que se calcula por la ganancia de información dada por la entropía
- **random_state=0**: Para generar los estados aleatorios

A continuación se muestra la salida de esto:

```
Out[8]:
DecisionTreeClassifier(class_weight=None, criterion='entropy', max_depth=None,
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=False,
random_state=0, splitter='best')
```

Implementación en Python del Árbol de Decisiones

3. Predicción del resultado de la prueba

Ahora vamos a predecir el resultado del conjunto de pruebas. Crearemos un nuevo vector de predicción y_pred. A continuación se muestra el código para ello:

```
#Predicting the test set result  
y_pred= classifier.predict(x_test)
```

Salida:

En la siguiente imagen de salida, se da la salida predicha y la salida real de la prueba. Podemos ver claramente que hay algunos valores en el vector de predicción, que son diferentes de los valores del vector real. Se trata de errores de predicción.

The image displays two side-by-side tables, both titled "y_pred - NumPy array" and "y_test - NumPy array". Both tables have rows numbered from 12 to 24. Each row contains two entries: the index number on the left and either a red or blue square on the right. Red squares represent value 0, and blue squares represent value 1. In the "y_pred" table, there are several blue squares (values 1) at indices 13, 15, 16, 18, 21, and 23. In the "y_test" table, there are several blue squares (values 1) at indices 13, 15, 16, 18, 21, and 23. This visual comparison highlights discrepancies between the predicted and actual values, indicating prediction errors.

	y_pred - NumPy array	y_test - NumPy array
12	0	0
13	1	0
14	0	0
15	1	0
16	1	0
17	0	0
18	1	1
19	0	0
20	0	0
21	1	1
22	0	0
23	1	1
24	0	0

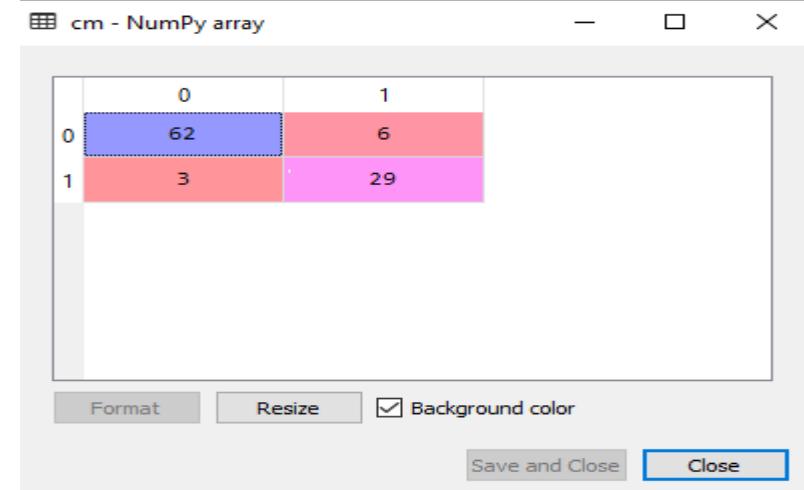
Implementación en Python del Árbol de Decisiones

4. Comprobar la precisión del resultado (Creación de la matriz de confusión)

En el resultado anterior, hemos visto que había algunas predicciones incorrectas, por lo que si queremos saber el número de predicciones correctas e incorrectas, tenemos que utilizar la matriz de confusión. A continuación se muestra el código para ello:

```
#Creating the Confusion matrix  
from sklearn.metrics import confusion_matrix  
cm= confusion_matrix(y_test, y_pred)
```

Output:



En la imagen de salida anterior, podemos ver la matriz de confusión, que tiene $6+3= 9$ predicciones incorrectas y $62+29=91$ predicciones correctas. Por lo tanto, podemos decir que, en comparación con otros modelos de clasificación, el clasificador de árbol de decisión hizo una buena predicción.

Implementación en Python del Árbol de Decisiones

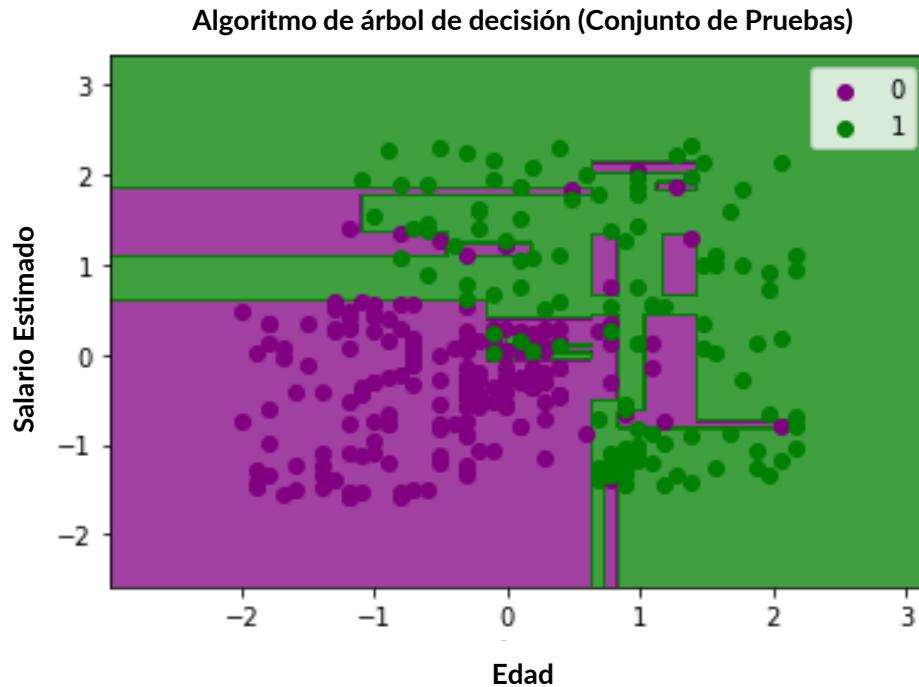
5. Visualización del resultado del conjunto de entrenamiento:

Aquí visualizaremos el resultado del conjunto de entrenamiento. Para visualizar el resultado del conjunto de entrenamiento, trazaremos un gráfico para el clasificador del árbol de decisión. El clasificador predecirá Si o No para los usuarios que han comprado o no han comprado el coche SUV como hicimos en la Regresión Logística. A continuación se muestra el código para ello:

```
#Visualizing the trianing set result
from matplotlib.colors import ListedColormap
x_set, y_set = x_train, y_train
x1, x2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() - 1, stop = x_set[:, 0].max() + 1,
nm.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1, step = 0.01))
mtp.contourf(x1, x2, classifier.predict(nm.array([x1.ravel(), x2.ravel()]).T).reshape(x1.shape),
alpha = 0.75, cmap = ListedColormap(('purple','green' )))
mtp.xlim(x1.min(), x1.max())
mtp.ylim(x2.min(), x2.max())
for i, j in enumerate(nm.unique(y_set)):
    mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
    c = ListedColormap(('purple', 'green'))(i), label = j)
mtp.title('Decision Tree Algorithm (Training set)')
mtp.xlabel('Age')
mtp.ylabel('Estimated Salary')
mtp.legend()
mtp.show()
```

Implementación en Python del Árbol de Decisiones

Output:



El resultado anterior es completamente diferente del resto de modelos de clasificación. Tiene líneas verticales y horizontales que están dividiendo el conjunto de datos según la edad y la variable de salario estimado.

Como podemos ver, el árbol está tratando de capturar cada conjunto de datos, que es el caso de sobreajuste.

Implementación en Python del Árbol de Decisiones

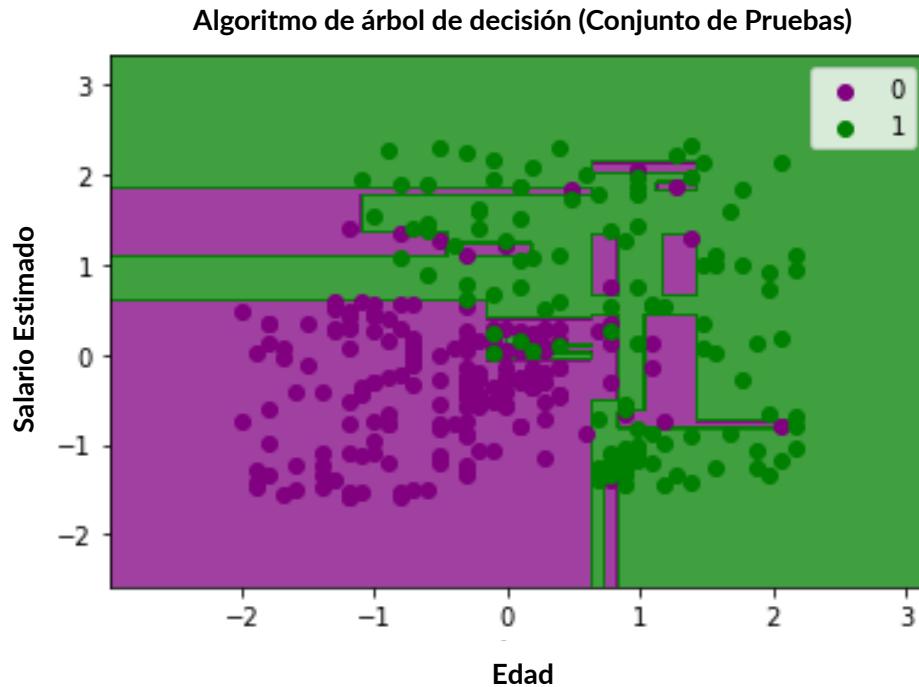
6. Visualización del resultado del conjunto de pruebas:

La visualización del resultado del conjunto de prueba será similar a la visualización del conjunto de entrenamiento, excepto que el conjunto de entrenamiento será reemplazado por el conjunto de prueba.

```
#Visualizing the test set result
from matplotlib.colors import ListedColormap
x_set, y_set = x_test, y_test
x1, x2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() - 1, stop = x_set[:, 0].max() + 1,
nm.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1, step = 0.01))
mtp.contourf(x1, x2, classifier.predict(nm.array([x1.ravel(), x2.ravel()]).T).reshape(x1.shape),
alpha = 0.75, cmap = ListedColormap(('purple','green' )))
mtp.xlim(x1.min(), x1.max())
mtp.ylim(x2.min(), x2.max())
for i, j in enumerate(nm.unique(y_set)):
    mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
    c = ListedColormap(('purple', 'green'))(i), label = j)
mtp.title('Decision Tree Algorithm(Test set)')
mtp.xlabel('Age')
mtp.ylabel('Estimated Salary')
mtp.legend()
mtn.show()
```

Implementación en Python del Árbol de Decisiones

Output:



Como podemos ver en la imagen anterior, hay algunos puntos de datos verdes dentro de la región púrpura y viceversa. Por lo tanto, estas son las predicciones incorrectas que hemos discutido en la matriz de confusión.

Proyecto Guiado: Predicción del Alquiler de Bicicletas

Muchas ciudades de Estados Unidos cuentan con estaciones de uso compartido de bicicletas en las que se pueden alquilar por horas o por días. Washington D.C. es una de estas ciudades. El Distrito recoge datos detallados sobre el número de bicicletas que la gente alquila por horas y días.

Hadi Fanaee-T, de la Universidad de Oporto, recopiló estos datos en un archivo CSV, con el que trabajarás en este proyecto. El archivo contiene **17380** filas, en las que cada fila representa el número de bicicletas alquiladas durante una sola hora de un solo día. Puedes descargar los datos del sitio web de la Universidad de California, Irvine. Si necesitas ayuda en algún momento, puedes consultar el cuaderno de soluciones en nuestro repositorio de GitHub.

Este es el aspecto de las cinco primeras filas:

instant	dteday	season	yr	mnth	hr	holiday	weekday	workingday	weathersit	temp	atemp	hum	windspeed	casual	registered	cnt
1	2011-01-01	1	0	1	0	0	6	0	1	0.24	0.2879	0.81		0	3	13 16
2	2011-01-01	1	0	1	1	0	6	0	1	0.22	0.2727	0.8		0	8	32 40
3	2011-01-01	1	0	1	2	0	6	0	1	0.22	0.2727	0.8		0	5	27 32
4	2011-01-01	1	0	1	3	0	6	0	1	0.24	0.2879	0.75		0	3	10 13
5	2011-01-01	1	0	1	4	0	6	0	1	0.24	0.2879	0.75		0	0	1 1

Proyecto Guiado: Predicción del Alquiler de Bicicletas

Estas son las descripciones de las columnas correspondientes:

- **instant** - Un número de identificación secuencial único para cada fila
- **dteday** - La fecha de los alquileres
- **season** - La temporada en la que se produjeron los alquileres
- **yr** - El año en que se produjeron los alquileres
- **mnth** - El mes en que se produjeron los alquileres
- **hr** - La hora en que se produjo el alquiler
- **holiday** - Si el día era o no festivo
- **weekday** - El día de la semana (como número, de 0 a 7)
- **workingday** - Si el día era o no laborable
- **weathersit** - El tiempo (como variable categórica)
- **temp** - La temperatura, en una escala de 0 a 1
- **atemp** - La temperatura ajustada
- **hum** - La humedad, en una escala de 0 a 1
- **windspeed** - La velocidad del viento, en una escala de 0 a 1
- **casual** - El número de ciclistas ocasionales (personas que no se han inscrito previamente en el programa de bicicletas compartidas)
- **registered** - El número de ciclistas registrados (personas que ya se han inscrito)
- **cnt** - El número total de bicicletas alquiladas (casuales + registradas)

Proyecto Guiado: Predicción del Alquiler de Bicicletas

En este proyecto, intentarás predecir el número total de bicicletas alquiladas en una hora determinada. Predecirá la columna `cnt` utilizando todas las demás columnas, excepto las casuales y las registradas. Para lograrlo, crearás algunos modelos de aprendizaje automático diferentes y evaluarás su rendimiento.

Instrucciones

- Utilice la biblioteca [pandas](#) para leer `bike_rental_hour.csv` en el marco de datos `bike_rentals`
- Imprime las primeras filas de `bike_rentals` y echa un vistazo a los datos
- Haga un histograma de la columna `cnt` de `bike_rentals`, y eche un vistazo a la distribución de los alquileres totales
- Utilice el método [corr](#) en el marco de datos `bike_rentals` para explorar cómo se correlaciona cada columna con `cnt`

Referencias y Bibliografía

Trabajos

- [1] M. Nasiri, B. Minaei, and Z. Sharifi, "Adjusting data sparsity problem using linear algebra and machine learning algorithm," *Appl. Soft Comput.*, vol. 61, pp. 1153–1159, Dec. 2017, doi: 10.1016/j.asoc.2017.05.042.
- [2] G. Marzano and A. Novembre, "Machines that Dream: A New Challenge in Behavioral-Basic Robotics," *Procedia Comput. Sci.*, vol. 104, pp. 146–151, Jan. 2017, doi: 10.1016/j.procs.2017.01.089.
- [3] Y. Ao, H. Li, L. Zhu, S. Ali, and Z. Yang, "The linear random forest algorithm and its advantages in machine learning assisted logging regression modeling," *J. Pet. Sci. Eng.*, vol. 174, pp. 776–789, Mar. 2019, doi: 10.1016/j.petrol.2018.11.067.
- [4] D. A. Otchere, T. O. Arbi Ganat, R. Gholami, and S. Ridha, "Application of supervised machine learning paradigms in the prediction of petroleum reservoir properties: Comparative analysis of ANN and SVM models," *J. Pet. Sci. Eng.*, vol. 200, p. 108182, May 2021, doi: 10.1016/j.petrol.2020.108182.
- [5] Y. Iwamoto et al., "Development and Validation of Machine Learning-Based Prediction for Dependence in the Activities of Daily Living after Stroke Inpatient Rehabilitation: A Decision-Tree Analysis," *J. Stroke Cerebrovasc. Dis.*, vol. 29, no. 12, p. 105332, Dec. 2020, doi: 10.1016/j.jstrokecerebrovasdis.2020.105332.

Referencias y Bibliografía

- [6] K. Maheswari, A. Priya, A. Balamurugan, and S. Ramkumar, "Analyzing student performance factors using KNN algorithm," Mater. Today Proc., Feb. 2021, doi: 10.1016/j.matpr.2020.12.1024.
- [7] L. Liang et al., "Status evaluation method for arrays in large-scale photovoltaic power stations based on extreme learning machine and k-means," Energy Rep., vol. 7, pp. 2484–2492, Nov. 2021, doi: 10.1016/j.egyr.2021.04.039.
- [8] W. A. van Eeden et al., "Predicting the 9-year course of mood and anxiety disorders with automated machine learning: A comparison between auto-sklearn, naïve Bayes classifier, and traditional logistic regression," Psychiatry Res., vol. 299, p. 113823, May 2021, doi: 10.1016/j.psychres.2021.113823.
- [9] C. M. Yeşilkanat, "Spatio-temporal estimation of the daily cases of COVID-19 in worldwide using random forest machine learning algorithm," Chaos Solitons Fractals, vol. 140, p. 110210, Nov. 2020, doi: 10.1016/j.chaos.2020.110210.

Referencias y Bibliografía

Libros

- [10] A. Geron, Hands-On Machine Learning with Scikit-Learn, Keras, and Tensorflow: Concepts, Tools, and Techniques to Build Intelligent Systems, \$ {number}nd édition. Sebastopol, CA: O'Reilly Media, Inc, USA, 2019.
- [11] Fondements de l'apprentissage automatique. Cambridge, MA: The MIT Press, 2012.

Plataformas de Autoaprendizaje

- <https://www.dataquest.io>
- <https://www.analyticsvidhya.com>
- <https://cloudacademy.com>
- <https://data-flair.training>
- <https://learn.datacamp.com>

COMPARTE Y VERIFICA

TUS LOGROS DE APRENDIZAJE FÁCILMENTE

#CAIPC #CertiProf



CertiProf®
certiprof.com



¡Síguenos, ponte en contacto!



certiprof.com

*CERTIPROF® is a registered trademark of CertiProf,
LLC in the United States and/or other countries.*

CertiProf®