ARTIFICIAL INTELLIGENCE
PROFESSIONAL CERTIFICATE

CAIPC™ Version 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 un instituto examinador fundado en los Estados Unidos en el año 2015 y que está localizado en Sunrise, Florida.

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

- **CKA's (CertiProf Knowledge Ambassadors)**, son personas influyentes en sus campos de experiencia o maestría, entrenadores, formadores, consultores, blogueros, constructores de comunidades, organizadores y evangelistas, que están dispuestos a contribuir en la mejora del contenido
- **CLL's (CertiProf Lifelong Learners)**, se identifican como aprendices continuos que han demostrado su compromiso inquebrantable con el aprendizaje permanente, que es de vital importancia en el mundo digitalizado en constante cambio y expansión de hoy. Independientemente de si ganan o no el examen
- **ATP's (Accredited Trainer Partners)**, Universidades, centros de formación y facilitadores de todo el mundo que integran la red de socios
- **Autores (co-creadores)**, Expertos o practicantes de la industria que, con sus conocimientos, desarrollan contenidos para la creación de nuevas certificaciones que respondan a las necesidades de la industria
- **Staff interno**, nuestro equipo distribuido con operaciones en India, Brasil, Colombia y Estados Unidos que apoyan día a día la ejecución del propósito de CertiProf®

Our Accreditations and Affiliations

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

Insignia

Type: Certification

Artificial Intelligence Professional Certificate - CAIPC™

Issued by CertiProf

The holders of this badge has 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.

Skills

AI | Algorithms | Data Analysts | Data Scientists | Data Stewards | Machine Learning

https://www.credly.com/org/certiprof/badge/artificial-intelligence-professional-certificate-caipc

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 digitalizado 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 la certificación 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
- Quiere impulsar su crecimiento profesional

CertiProf®
certiprof.com

COMPARTE Y VERIFICA

TUS LOGROS DE APRENDIZAJE FÁCILMENTE

#CAIPC #CertiProf

AGENDA

ARTIFICIAL INTELLIGENCE
PROFESSIONAL CERTIFICATE

Fundamentos del Aprendizaje Automático

CAIPC™ Version 062021

CertiProf®

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:

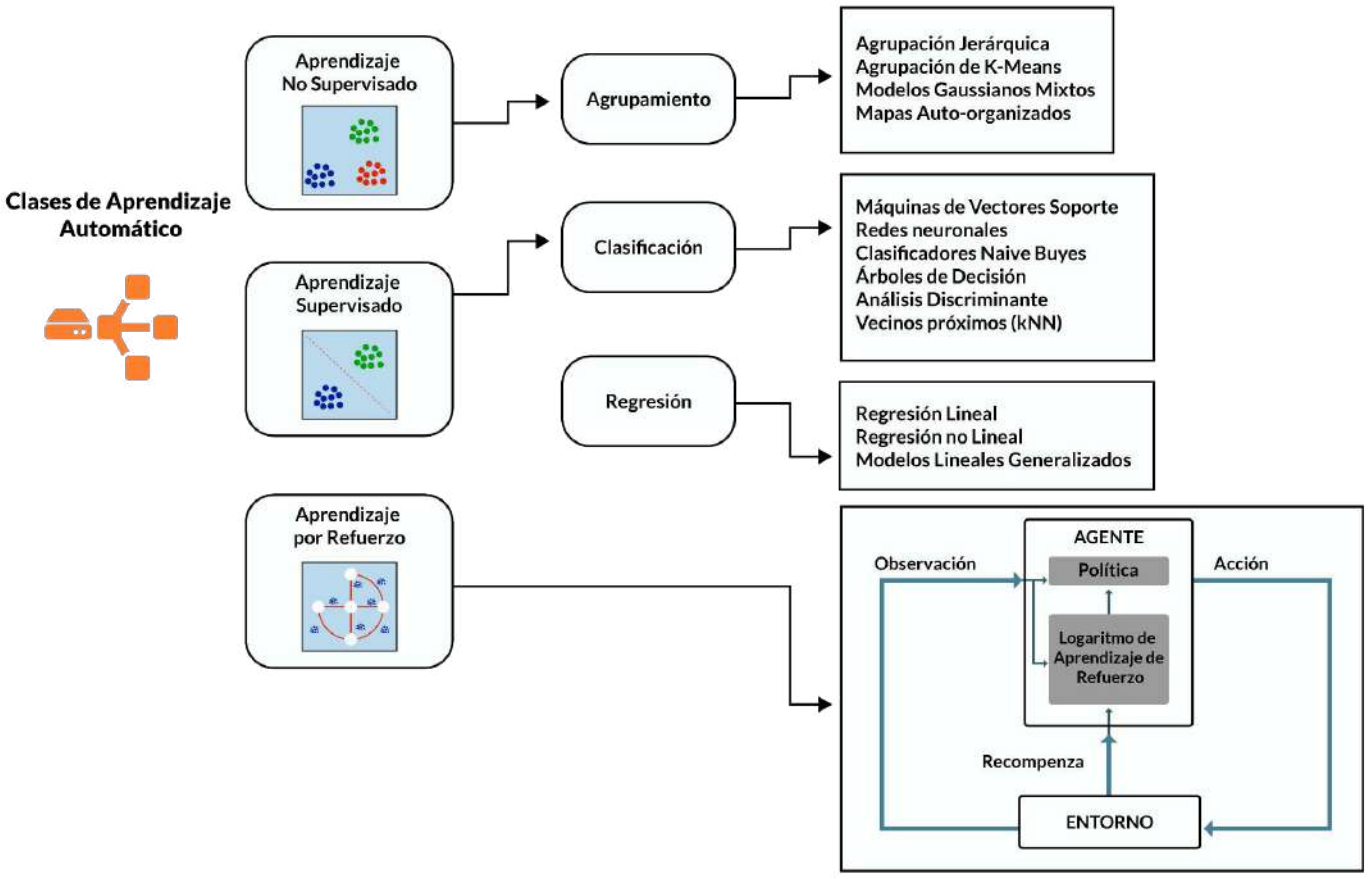


Figura 1.Tipos de Aprendizaje Automático

ARTIFICIAL INTELLIGENCE
PROFESSIONAL CERTIFICATE

I.1 Puntos Clave

CAIPC™ Version 062021

CertiProf®

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.

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.

ARTIFICIAL INTELLIGENCE
PROFESSIONAL CERTIFICATE

I.2 Introducción K-Nearest Neighbors

CAIPC™ Version 062021

CertiProf®

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 euclidiana 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, trabajaremos con su conjunto de datos del 10 de julio de 2021 sobre los anuncios de Washington, D.C. Aquí hay un enlace directo a ese conjunto de datos.

Cada fila del conjunto de datos es un anuncio específico que estaba disponible para alquilar en AirBnB en la zona de Washington, D.C. Para que el conjunto de datos sea menos engorroso, hemos eliminado muchas de las columnas del conjunto de datos original y hemos cambiado el nombre del archivo a dc_airbnb.csv. Estas son las columnas que hemos mantenido:

- **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 baños incluidos en la renta
- **bathrooms:** número de camas incluidas 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

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:

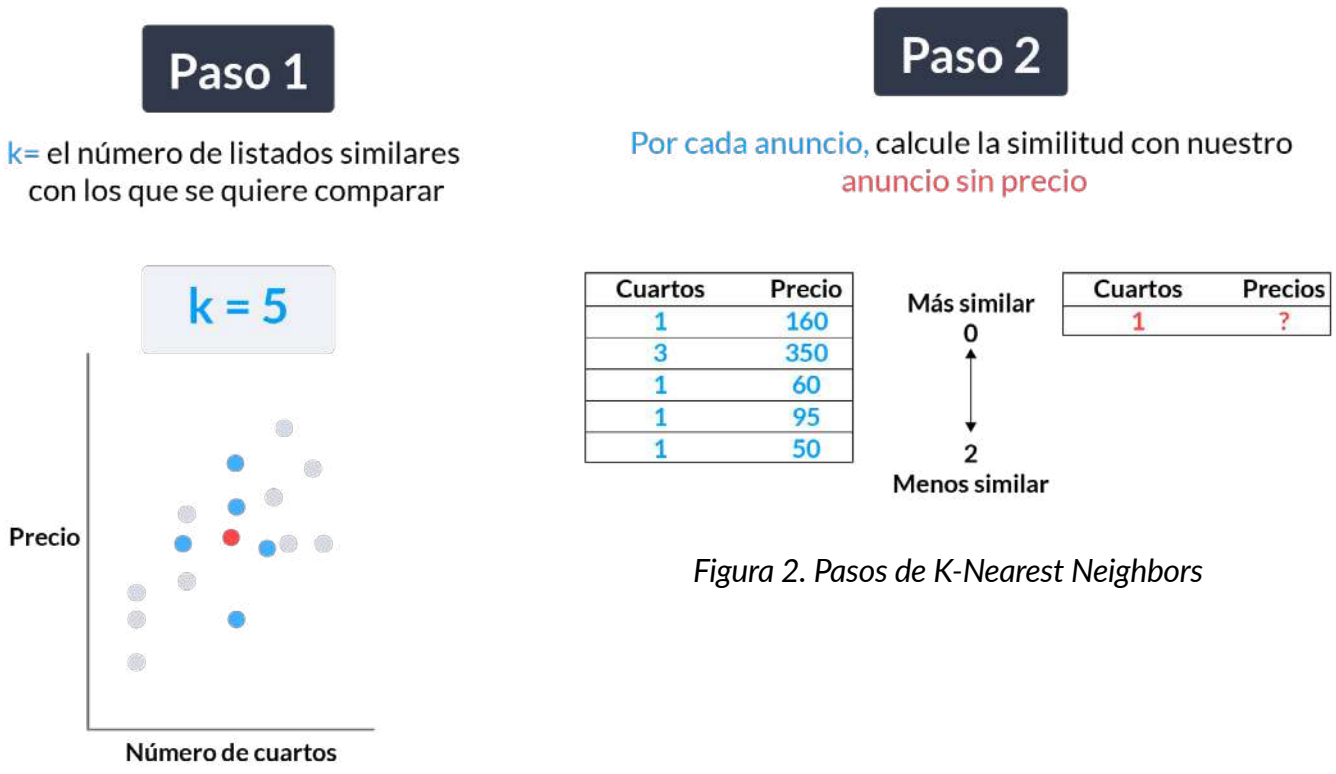
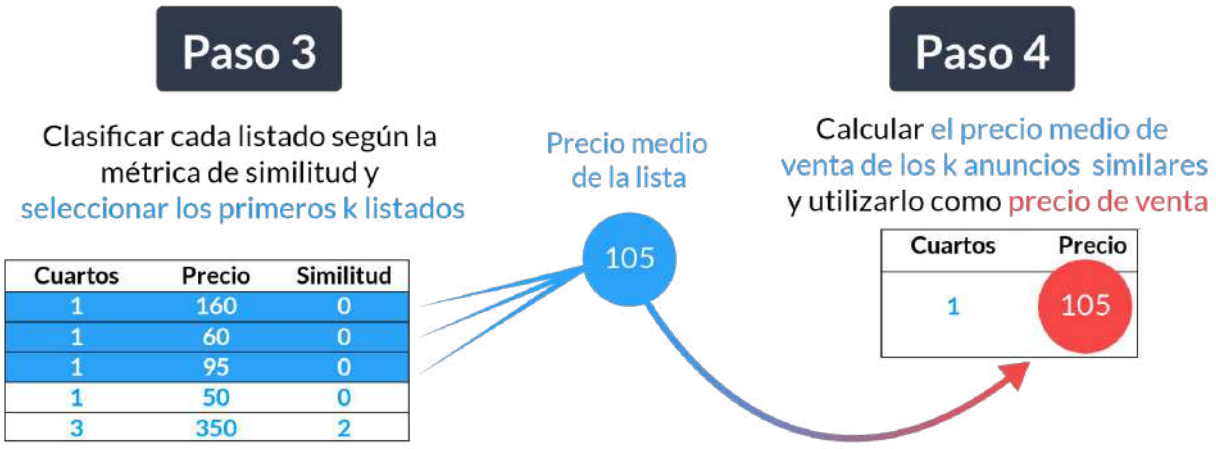


Figura 2. Pasos de K-Nearest Neighbors



Hay dos cosas que debemos desgranar con más detalle:

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

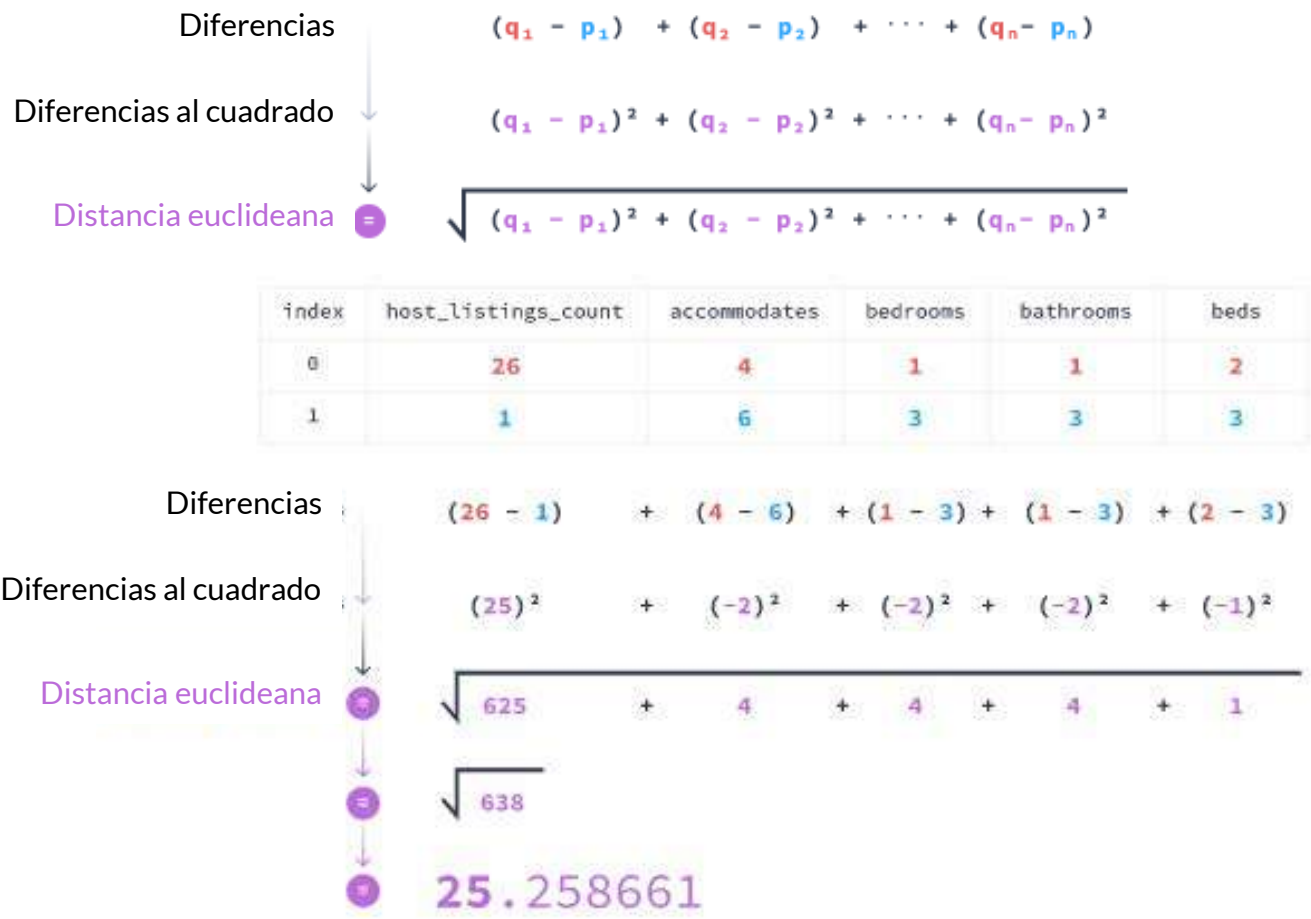
En esta lección, definiremos la métrica de similitud que vamos a utilizar. A continuación, implementaremos el algoritmo de **k** vecinos más cercanos y lo utilizaremos para sugerir un precio para un nuevo anuncio sin precio. En esta lección utilizaremos un valor k de **5**.

Distancia Euclidiana

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 Euclidiana.** Esta es la fórmula general de la distancia euclidiana:

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

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 euclidiana entre las dos primeras observaciones del conjunto de datos utilizando únicamente las columnas **host_listing_count, accommodates, bedrooms, bathrooms** y **bed**.



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

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.**

Instrucciones

1. Calcule la distancia euclidiana 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

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

Calcular la Distancia para Todas las Observaciones

La distancia euclidiana 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 euclidiana, 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 p1=q1, entonces d=|q1-p1|, 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 euclidiana 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

accommodates
8

dc_listings

index	accommodates	distance	calculated distance
0	4	|4 - 8|	4
1	6	|6 - 8|	2
0	4	|1 - 8|	7
1	6	|2 - 8|	6

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

```
new_listing = 3
dc_listings['distance'] =
dc_listings['accommodates'].apply(
    lambda x: np.abs(x - new_listing)
)
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
...
```

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**

Solutions

```
import numpy as np
np.random.seed(1)
dc_listings =
dc_listings.loc[np.random.permutation(len(dc_listings))]
dc_listings = dc_listings.sort_values('distance')
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.

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

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

Funciones de Predicción

¡Felicidades! ¡Acabas de hacer tu primera predicción! Basándonos en el precio medio de otros listados que alojan a tres personas, deberíamos cobrar **156,6** 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.

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**

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**

Soluciones

```
# Brought along the changes we made to the 'dc_listings' Dataframe.
dc_listings = pd.read_csv('dc_airbnb.csv')
stripped_commas = dc_listings['price'].str.replace(',', '')
stripped_dollars = stripped_commas.str.replace('$', '')
dc_listings['price'] = stripped_dollars.astype('float')
dc_listings = dc_listings.loc[np.random.permutation(len(dc_listings))]

def predict_price(new_listing):
    temp_df = dc_listings.copy()
    ## Complete the function.
    return(new_listing)

acc_one = predict_price(1)
acc_two = predict_price(2)
```

```
acc_four = predict_price(4)
def predict_price(new_listing):
    temp_df = dc_listings.copy()
    temp_df['distance'] = temp_df['accommodates'].apply(lambda x: np.abs(x - new_listing))
    temp_df = temp_df.sort_values('distance')
    nearest_neighbors = temp_df.iloc[0:5]['price']
    predicted_price = nearest_neighbors.mean()
    return(predicted_price)

acc_one = predict_price(1)
acc_two = predict_price(2)
acc_four = predict_price(4)
print(acc_one)
print(acc_two)
print(acc_four)
```

- **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**

ARTIFICIAL INTELLIGENCE
PROFESSIONAL CERTIFICATE

I.3 Evaluación del Rendimiento del Modelo

CAIPC™ Version 062021

CertiProf®

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

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.

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:

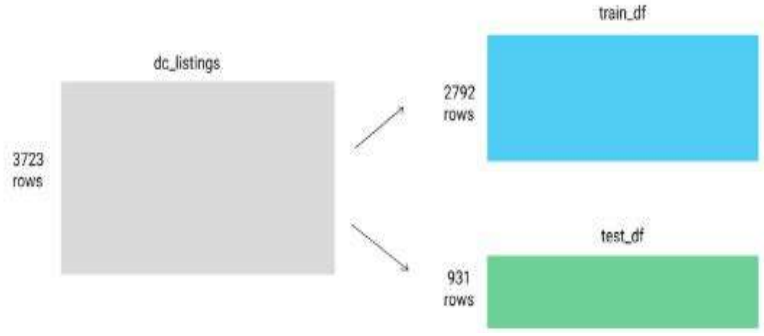


Figura 3. Diagrama de descomposición del modelo de entrenamiento y de prueba

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**

Soluciones

```
import pandas as pd
import numpy as np
dc_listings = pd.read_csv("dc_airbnb.csv")
stripped_commas = dc_listings['price'].str.replace(',', '')
stripped_dollars = stripped_commas.str.replace('$', '')
dc_listings['price'] = stripped_dollars.astype('float')
train_df = dc_listings.iloc[0:2792]
test_df = dc_listings.iloc[2792:]

def predict_price(new_listing):
    ## DataFrame.copy() performs a deep copy
    temp_df = dc_listings.copy()
    temp_df['distance'] = temp_df['accommodates'].apply(lambda x: np.abs(x - new_listing))
    temp_df = temp_df.sort_values('distance')
    nearest_neighbor_prices = temp_df.iloc[0:5]['price']
    predicted_price = nearest_neighbor_prices.mean()
    return(predicted_price)
def predict_price(new_listing):
    temp_df = train_df.copy()
    temp_df['distance'] = temp_df['accommodates'].apply(lambda x: np.abs(x - new_listing))
    temp_df = temp_df.sort_values('distance')
    nearest_neighbor_prices = temp_df.iloc[0:5]['price']
    predicted_price = nearest_neighbor_prices.mean()
    return(predicted_price)

test_df['predicted_price'] = 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.

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

Instrucciones

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

Soluciones

```
import numpy as np
test_df['error'] =
np.absolute(test_df['predicted_price'] -
test_df['price'])
mae = test_df['error'].mean()
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:

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

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.

Instrucciones

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

Soluciones:

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

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

Soluciones

```
train_df = dc_listings.iloc[0:2792]
test_df = dc_listings.iloc[2792:]

def predict_price(new_listing):
    temp_df = train_df.copy()
    temp_df['distance'] =
temp_df['accommodates'].apply(lambda x: np.abs(x -
new_listing))
    temp_df = temp_df.sort_values('distance')
    nearest_neighbors_prices = temp_df.iloc[0:5]
['price']
    predicted_price = nearest_neighbors_prices.mean()
    return(predicted_price)
def predict_price(new_listing):
    temp_df = train_df.copy()
    temp_df['distance'] =
temp_df['bathrooms'].apply(lambda x: np.abs(x -
new_listing))
```

```
    temp_df['distance'] =
temp_df['bathrooms'].apply(lambda x: np.abs(x -
new_listing))
    temp_df = temp_df.sort_values('distance')
    nearest_neighbors_prices = temp_df.iloc[0:5]
['price']
    predicted_price = nearest_neighbors_prices.mean()
    return(predicted_price)

test_df['predicted_price'] =
test_df['bathrooms'].apply(lambda x: predict_price(x))
test_df['squared_error'] = (test_df['predicted_price'] -
 test_df['price'])**(2)
mse = test_df['squared_error'].mean()
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.**

Instrucciones

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

Soluciones

```
def predict_price(new_listing):
    temp_df = train_df.copy()
    temp_df['distance'] =
temp_df['bathrooms'].apply(lambda x: np.abs(x -
new_listing))
    temp_df = temp_df.sort_values('distance')
    nearest_neighbors_prices = temp_df.iloc[0:5]
['price']
    predicted_price = nearest_neighbors_prices.mean()
    return(predicted_price)

test_df['predicted_price'] =
test_df['bathrooms'].apply(lambda x: predict_price(x))
test_df['squared_error'] = (test_df['predicted_price'] -
 test_df['price'])**(2)
mse = test_df['squared_error'].mean()
rmse = mse ** (1/2)
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:

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

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:

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

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.**

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

```
errors_one = pd.Series([5, 10, 5, 10, 5, 10, 5, 10, 5,
10, 5, 10, 5, 10, 5, 10, 5, 10])
errors_two = pd.Series([5, 10, 5, 10, 5, 10, 5, 10, 5,
10, 5, 10, 5, 10, 5, 10, 5, 1000])
mae_one = errors_one.sum()/len(errors_one)
rmse_one =
np.sqrt((errors_one**2).sum()/len(errors_one))
print(mae_one)
print(rmse_one)

mae_two = errors_two.sum()/len(errors_two)
rmse_two =
np.sqrt((errors_two**2).sum()/len(errors_two))
print(mae_two)
print(rmse_two)
```

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.

ARTIFICIAL INTELLIGENCE
PROFESSIONAL CERTIFICATE

I.4 Multivariante del Método K-Nearest Neighbors

CAIPC™ Version 062021

CertiProf®

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 habitale puede acomodar comodamente. Luego, cambiamos a la columna de **bathrooms** y observamos una majora 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

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 euclidiana 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 euclidiana 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.

Instrucciónes

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

Soluciones

```
import pandas as pd
import numpy as np
np.random.seed(1)

dc_listings = pd.read_csv('dc_airbnb.csv')
dc_listings =
dc_listings.loc[np.random.permutation(len(dc_listings))]
stripped_commas = dc_listings['price'].str.replace(',',
'')
stripped_dollars = stripped_commas.str.replace('$', '')
dc_listings['price'] = stripped_dollars.astype('float')
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

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**

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

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

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

Soluciones

```
dc_listings = dc_listings.drop(['cleaning_fee',
'security_deposit'], axis=1)
dc_listings = dc_listings.dropna(axis=0)
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

Habrá observado que mientras las columnas de **accommodates, bedrooms, bathroms, 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 euclidiana, estos listados se considerarían muy alejados debido al gran efecto que tiene el tamaño de los valores en la distancia euclidiana 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.

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 □ es la desviación estándar de todos los valores de la columna. Este es el aspecto del código correspondiente, utilizando pandas:

```
# 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()
```

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.**

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 Euclidiana 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 euclidiana 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 **acommodates**, 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 euclidiana entre las 2 primeras filas de **normalized_listings:**

	accommodates	bathrooms
	-0.596544	-0.439151
	-0.596544	0.412923

$(q_1 - p_1) + (q_2 - p_2) + \ldots + (q_n - p_n)$	differences	(-0.596544 + 0.596544)	+	(-0.439151 - 0.412923)
$(q_1 - p_1)^2 + (q_2 - p_2)^2 + \ldots + (q_n - p_n)^2$	squared differences	$(0)^2$	+	$(-0.852074)^2$
$\sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2 + \ldots + (q_n - p_n)^2}$	Euclidean distance	$= \sqrt{0 + 0.72603}$		
		$= 0.852074$		

Hasta ahora, hemos calculado la distancia euclidiana 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 euclidiana entre ellos. La función euclidiana() 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 euclidiana entre 2 filas de nuestro conjunto de datos para practicar.

Instrucciones

- Calcular la distancia euclidiana utilizando sólo las características de los **acommodates** 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][['accommodates',
  'bathrooms']]
3 fifth_listing = normalized_listings.iloc[4][['accommodates',
  'bathrooms']]
4 first_fifth_distance = distance.euclidean(first_listing,
  fifth_listing)
5 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 entrenan 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

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()
```

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 euclidiana

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 quieres 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)

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.

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.

Instrucciones

- Cree una instancia de la clase KNeighborsRegressor con los siguientes parámetros:
  - **n_neighbors: 5**
  - **algorithm: 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**)

Soluciones

```
from sklearn.neighbors import KNeighborsRegressor

train_df = normalized_listings.iloc[0:2792]
test_df = normalized_listings.iloc[2792:]
train_columns = ['accommodates', 'bathrooms']

# Instantiate ML model.
knn = KNeighborsRegressor(n_neighbors=5, algorithm='brute')

# Fit model to data.
knn.fit(train_df[train_columns], train_df['price'])

# Use model to make predictions.
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.

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)**

Soluciones

```
from sklearn.metrics import mean_squared_error

train_columns = ['accommodates', 'bathrooms']
knn = KNeighborsRegressor(n_neighbors=5, algorithm='brute',
metric='euclidean')
knn.fit(train_df[train_columns], train_df['price'])
predictions = knn.predict(test_df[train_columns])
from sklearn.metrics import mean_squared_error

two_features_mse = mean_squared_error(test_df['price'],
predictions)
two_features_rmse = two_features_mse ** (1/2)
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.

Instrucciones

- Cree una nueva instancia de la clase KNeighborsRegressor class 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

Soluciones

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

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

Soluciones

```
knn = KNeighborsRegressor(n_neighbors=5, algorithm='brute')

features = train_df.columns.tolist()
features.remove('price')

knn.fit(train_df[features], train_df['price'])
all_features_predictions = knn.predict(test_df[features])
all_features_mse = mean_squared_error(test_df['price'],
all_features_predictions)
all_features_rmse = all_features_mse ** (1/2)
print(all_features_mse)
print(all_features_rmse)
```

- **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**

ARTIFICIAL INTELLIGENCE
PROFESSIONAL CERTIFICATE

I.5 Optimización de Hiperparámetros

CAIPC™ Version 062021

CertiProf®

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.

In this mission, we'll focus on the impact of increasing **k**, the number of nearby neighbors the model uses to make predictions. We exported both the training (**train_df**) and test sets (**test_df**) from the last missions to CSV files, **dc_airbnb_train.csv** and **dc_airbnb_test.csv** respectively. Let's read both these CSV's into Dataframes.

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**

```
import pandas as pd
train_df = pd.read_csv('dc_airbnb_train.csv')
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

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**

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()**

Soluciones

```
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import mean_squared_error
features = ['accommodates', 'bedrooms', 'bathrooms',
'number_of_reviews']
hyper_params = [1, 2, 3, 4, 5]
mse_values = list()

for hp in hyper_params:
    knn = KNeighborsRegressor(n_neighbors=hp,
algorithm='brute')
    knn.fit(train_df[features], train_df['price'])
    predictions = knn.predict(test_df[features])
    mse = mean_squared_error(test_df['price'],
predictions)
    mse_values.append(mse)

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.

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 objecto 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()**

Soluciones

```
features = ['accommodates', 'bedrooms', 'bathrooms',
'number_of_reviews']
hyper_params = [x for x in range(1, 21)]
mse_values = list()

for hp in hyper_params:
    knn = KNeighborsRegressor(n_neighbors=hp,
algorithm='brute')
    knn.fit(train_df[features], train_df['price'])
    predictions = knn.predict(test_df[features])
    mse = mean_squared_error(test_df['price'],
predictions)
    mse_values.append(mse)
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.

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

Soluciones

```
import matplotlib.pyplot as plt

features = ['accommodates', 'bedrooms', 'bathrooms',
'number_of_reviews']
hyper_params = [x for x in range(1, 21)]
mse_values = list()

for hp in hyper_params:
    knn = KNeighborsRegressor(n_neighbors=hp,
algorithm='brute')
    knn.fit(train_df[features], train_df['price'])
    predictions = knn.predict(test_df[features])
    mse = mean_squared_error(test_df['price'],
predictions)
    mse_values.append(mse)
plt.scatter(hyper_params, mse_values)
plt.show()
```

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.