Capítulo 1/7

Introducción al curso

**Introducción al álgebra lineal**

**¡Te damos la bienvenida al tercer módulo!**

Data Science es un área muy amplia y en constante crecimiento, que incluye muchas tecnologías diferentes y campos enteros, como la visión artificial o el procesamiento del lenguaje natural. Nos llevaría años aprender cualquier disciplina por separado en toda su profundidad. En este tercer módulo, nuestro objetivo principal es ofrecerte una visión panorámica de estos enfoques sin profundizar demasiado en los detalles. Para un científico de datos junior, es importante hacerse una idea general de la profesión y de lo que tiene a su disposición. Con el paso del tiempo, elegirás la dirección que más te atraiga o que se ajuste a tus ambiciones y habilidades.

**Álgebra lineal**

El álgebra lineal es un conocimiento esencial para los científicos de datos. Es el núcleo de los algoritmos utilizados para crear modelos de machine learning. Para crear modelos de machine learning apropiados, los científicos de datos recurren a la ayuda de las técnicas del álgebra lineal. A menudo, por ejemplo, es útil tratar los datos como si fueran una matriz o una parte de un espacio vectorial multidimensional.

En este curso, aprenderemos lo básico sobre vectores y matrices, trabajaremos con las librerías de Python NumPy, SciPy y sklearn a fin de realizar operaciones con vectores y matrices y aprenderemos a calcular distancias y métricas para mejorar nuestros modelos.

**Estructura del curso**

Los vectores serán el punto de partida de nuestro viaje por el álgebra lineal. Aprenderemos a escribir funciones vectoriales y a realizar operaciones con matrices. También veremos las herramientas de la biblioteca NumPy.

Luego pasaremos a los productos escalares. Aprenderemos a medir la longitud de los vectores y la distancia entre ellos. Hablaremos de las distancias euclidiana y de Manhattan. Además, escribiremos nuestro primer algoritmo de machine learning: el vecino más cercano.

En la última sección del curso, hablaremos de las matrices. Aprenderemos a multiplicar matrices por vectores y a transponer. Conocerás cómo funciona la regresión lineal, cómo realiza las predicciones y cómo se entrena.

Al final, como siempre, te encontrarás con el proyecto del curso. Haz clic en Siguiente lección al final de la página para ver la descripción del proyecto.

**Tus objetivos:**

* Realizar operaciones matriciales y vectoriales comunes utilizando la librería NumPy.
* Crear clases de Python para modelos de machine learning.
* Desarrollar una comprensión más profunda de un modelo de regresión lineal.

¡Sigamos mejorando tu conjunto de habilidades!

Imagen que contiene Código QR

Descripción generada automáticamente

Imagen que contiene Gráfico

Descripción generada automáticamente

**¿Cuánto tiempo llevará?**

Este curso es moderadamente desafiante. Habrá muchas fórmulas matemáticas y tareas prácticas. Puede que te lleve entre 30 y 50 horas realizarlo, dependiendo de tus conocimientos previos y de tus hábitos de estudio. Intenta no perderte y no dudes en pedir ayuda.

**Proyecto del curso**

La compañía de seguros Sure Tomorrow quiere resolver varias tareas con la ayuda de machine learning y te pide que evalúes esa posibilidad.

* Task 1: encontrar clientes que sean similares al cliente dado. Esto ayudará a los agentes de la compañía con el marketing.
* Task 2: predecir la probabilidad de que un nuevo cliente reciba una prestación del seguro. ¿Puede un modelo entrenado hacerlo mejor que un modelo aleatorio?
* Task 3: predecir el número de prestaciones de seguro que un nuevo cliente pueda recibir utilizando un modelo de regresión lineal.
* Task 4: desarrollar una forma de proteger los datos personales de los clientes sin afectar al modelo de la tarea anterior.

Es necesario desarrollar un algoritmo de transformación de datos que dificulte la recuperación de la información personal si los datos caen en manos equivocadas. Esto se denomina enmascaramiento de datos (data masking) u ofuscación de datos (data obfuscation). Pero los datos deben protegerse de tal manera que la calidad de los modelos de machine learning no se vea afectada. No es necesario elegir el mejor modelo, basta con demostrar que el algoritmo funciona correctamente.

Instrucciones del proyecto

1. Descarga los datos.
2. Comprueba que los datos no tengan errores, es decir, que no falten datos, que no haya valores extremos, etc.
3. Trabaja en cada tarea y responde a las preguntas planteadas en la plantilla del proyecto.
4. Saca conclusiones basadas en la experiencia adquirida durante la realización del proyecto.

Hay dos anexos en la plantilla del proyecto que contienen información útil.

**Descripción de datos**

El dataset está almacenado en el archivo /datasets/insurance\_us.csv. [Puedes descargar el dataset aquí.](https://practicum-content.s3.us-west-1.amazonaws.com/datasets/insurance_us.csv)

* Características: género de la persona asegurada, edad, salario y número de miembros de la familia.
* Objetivo: número de prestaciones de seguro recibidas por la persona asegurada en los últimos cinco años.

**Evaluación del proyecto**

Hemos definido los criterios de evaluación para el proyecto. Léelos con atención antes de pasar al ejercicio.

Esto es en lo que se fijarán los revisores al examinar tu proyecto:

* ¿Seguiste todos los pasos de las instrucciones?
* ¿Mantuviste la estructura del proyecto?
* ¿Mantuviste el código limpio?
* ¿Desarrollaste todos los procedimientos necesarios y respondiste a todas las preguntas?
* ¿Sacaste tus conclusiones?

Al final de la plantilla del proyecto encontrarás una lista de control más detallada.

Ya tienes tus hojas informativas y los resúmenes de los capítulos, por lo que todo está listo para continuar con el proyecto.

¡Buena suerte!

Capítulo 2/7

Vectores y operaciones vectoriales

**Introducción**

**Aprenderemos sobre los vectores, que son una buena forma de representar datos numéricos. También trabajaremos con matrices, a fin de representar los datos como una tabla con filas y columnas. Por último, las utilizaremos para practicar la manipulación y el análisis de datos.**

**Empezarás por:**

* crear una matriz numpy multidimensional a partir de una lista de Python;
* determinar el tamaño de un vector utilizando la función len();
* examinar una lista de números visualizando vectores en un plano de coordenadas;
* realizar sumas y restas de vectores del mismo tamaño;
* convertir los valores del DataFrame en vectores mediante el atributo values;
* calcular el valor medio de un conjunto de vectores;
* escribir funciones vectorizadas para realizar operaciones con vectores de forma eficiente.

**¿Cuánto tiempo tomará?**

9 lecciones de 10-15 minutos cada una.

**Descripción del ejercicio**

Observar las operaciones con vectores a través de diferentes casos prácticos.

**Creación de vectores**

**Cualquier dato numérico puede representarse como un vector.**

En Python, los conjuntos de datos a menudo se representan como listas. En matemáticas, un conjunto ordenado de datos numéricos es un vector, o un vector aritmético. Las operaciones que se puedan realizar con números, es decir, sumas, restas y multiplicaciones, también se podrán realizar con vectores. En Python, las operaciones con vectores son cientos de veces más rápidas que las operaciones con listas.

Pasemos a la librería NumPy. Su nombre se debe a las siglas de Numeric Python (Python Numérico). La librería proporciona herramientas para trabajar con vectores. Ya vimos np.arange(), que es una de sus funciones. En este curso, nos hará falta la función **np.array()**. No te dejes confundir por el nombre. Una matriz (array) es una forma útil de representar vectores en nuestro código, y a menudo usamos los términos indistintamente.

Crea una lista de dos números:

numbers1 = [2, 3] *# Lista de Python*

print(numbers1)

[2, 3]

Convierte la lista en un vector:

import numpy as np

vector1 = np.array(numbers1) *# Matriz de NumPy*

print(vector1)

[2 3]

Crea otro vector sin una variable temporal:

import numpy as np

vector2 = np.array([6, 2])

print(vector2)

[6 2]

Convierte este vector en una lista:

numbers2 = list(vector2) *# Lista a partir de vector*

print(numbers2)

[6, 2]

La columna de la estructura de DataFrame en pandas se convierte en un vector NumPy utilizando el atributo \*\*values\*\*:

import pandas as pd

data = pd.DataFrame([1, 7, 3])

print(data[0].values)

[1 7 3]

Utiliza la función len() para determinar el tamaño del vector (número de sus elementos):

print(len(vector2))

2

Veamos un ejemplo concreto. La gente va a la tienda online de ropa LuxForVIP desde dos sitios agregadores. El primero anuncia artículos del mercado de masas, mientras que el segundo sitio agrega ropa de marca de lujo. Los visitantes de LuxForVIP valoran su satisfacción con los precios y la calidad de los productos en una escala de 0 a 100. La tabla contiene las valoraciones de todos los visitantes. Una puntuación alta para el precio indica que al cliente le gustó el precio (es decir, que la relación calidad-precio del artículo es buena). Del mismo modo, una puntuación alta para la calidad indica que el cliente consideró que el artículo era de alta calidad.

import numpy as np

import pandas as pd

ratings\_values = [

[68,18], [81,19], [81,22], [15,75], [75,15], [17,72],

[24,75], [21,91], [76, 6], [12,74], [18,83], [20,62],

[21,82], [21,79], [84,15], [73,16], [88,25], [78,23],

[32, 81], [77, 35]]

ratings = pd.DataFrame(ratings\_values, columns=['Price', 'Quality'])

print(ratings)

Price Quality

0 68 18

1 81 19

2 81 22

3 15 75

4 75 15

5 17 72

6 24 75

7 21 91

8 76 6

9 12 74

10 18 83

11 20 62

12 21 82

13 21 79

14 84 15

15 73 16

16 88 25

17 78 23

18 32 81

19 77 35

1.

Crea dos vectores. El primero contiene todas las valoraciones del precio. El segundo contiene las valoraciones de la calidad.

import numpy as np

import pandas as pd

ratings\_values = [

[68,18], [81,19], [81,22], [15,75], [75,15], [17,72],

[24,75], [21,91], [76, 6], [12,74], [18,83], [20,62],

[21,82], [21,79], [84,15], [73,16], [88,25], [78,23],

[32, 81], [77, 35]]

ratings = pd.DataFrame(ratings\_values, columns=['Price', 'Quality'])

price = ratings['Price'].values # < escribe tu código aquí >

quality = ratings['Quality'].values# < escribe tu código aquí >

print('Precio: ', price)

print('Calidad: ', quality)

Resultado

Precio: [68 81 81 15 75 17 24 21 76 12 18 20 21 21 84 73 88 78 32 77]

Calidad: [18 19 22 75 15 72 75 91 6 74 83 62 82 79 15 16 25 23 81 35]

¡Es correcto!

Las columnas de números

Te dejaron en un aprieto,

Pero usar nuestros vectores hizo

Que la solución fuera pan comido.

2.

Encuentra el número total de visitantes de LuxForVIP. Encuentra el quinto vector que contiene las valoraciones de los visitantes.

import numpy as np

import pandas as pd

ratings\_values = [

[68,18], [81,19], [81,22], [15,75], [75,15], [17,72],

[24,75], [21,91], [76, 6], [12,74], [18,83], [20,62],

[21,82], [21,79], [84,15], [73,16], [88,25], [78,23],

[32, 81], [77, 35]]

ratings = pd.DataFrame(ratings\_values, columns=['Price', 'Quality'])

visitors\_count = len(ratings)# < escribe tu código aquí >

visitor4 = (ratings.loc[4].values)# < escribe tu código aquí >

print('Número de visitantes:', visitors\_count)

print('Visitante 4:', visitor4)

Resultado

Número de visitantes: 20

Visitante 4: [75 15]

¡Es correcto!

Un precio bajo no siempre es sinónimo de baja calidad. Sin embargo, parece que este es el caso.

3.

La tabla tiene el atributo values. values es una matriz bidimensional. Llama a la función list() para convertir esta matriz en una lista de vectores con las valoraciones de todos los visitantes.

import numpy as np

import pandas as pd

ratings\_values = [

[68,18], [81,19], [81,22], [15,75], [75,15], [17,72],

[24,75], [21,91], [76, 6], [12,74], [18,83], [20,62],

[21,82], [21,79], [84,15], [73,16], [88,25], [78,23],

[32, 81], [77, 35]]

ratings = pd.DataFrame(ratings\_values, columns=['Price', 'Quality'])

vector\_list = list(ratings.values)

# < escribe tu código aquí >

print(vector\_list)

Resultado

[array([68, 18]), array([81, 19]), array([81, 22]), array([15, 75]), array([75, 15]), array([17, 72]), array([24, 75])

Es correcto!

Da la bienvenida a la primera lista de respuestas.

**Presentación de vectores**

**Vamos a visualizar los vectores para examinar la lista de números en el plano de coordenadas.**

Tracemos un vector bidimensional. Consta de dos números. El primero es la coordenada en el eje horizontal *x* y el segundo es la coordenada en el eje vertical *y*. El vector se representa mediante un punto o una flecha, que une el origen y el punto con coordenadas (*x, y*).

La razón de utilizar una flecha radica en que esta indica las dos componentes de un vector: magnitud y dirección. Por ejemplo, el vector [2, 3] es un desplazamiento de dos casillas hacia la derecha a lo largo del eje x y de tres casillas hacia arriba a lo largo del eje y. Si trabajamos con varios vectores que se encuentran en la misma línea, es mejor utilizar un punto para representar un vector.

Por ejemplo, vamos a trazar los vectores vector1 - [2, 3] y vector2 - [6, 2] utilizando puntos. Los puntos están dibujados con el método plt.plot().

import numpy as np

import matplotlib.pyplot as plt

vector1 = np.array([2, 3])

vector2 = np.array([6, 2])

plt.figure(figsize=(10, 10))

plt.axis([0, 7, 0, 7])

*# El argumento 'ro' establece el estilo del gráfico*

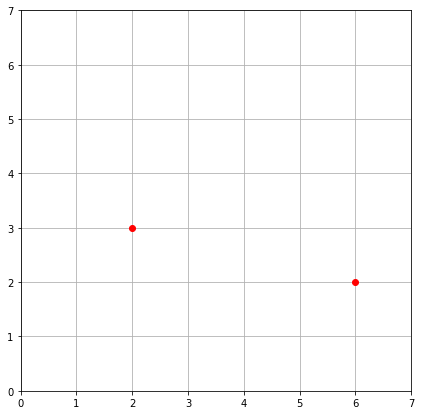
*# 'r' - rojo*

*# 'o' - círculo*

plt.plot([vector1[0], vector2[0]], [vector1[1], vector2[1]], 'ro')

plt.grid(True)

plt.show()



Vamos a utilizar flechas para dibujar los mismos vectores. En lugar de plt.plot(), llama a **plt.arrow()**.

import numpy as np

import matplotlib.pyplot as plt

vector1 = np.array([2, 3])

vector2 = np.array([6, 2])

plt.figure(figsize=(10, 10))

plt.axis([0, 7, 0, 7])

plt.arrow(

0,

0,

vector1[0],

vector1[1],

head\_width=0.3,

length\_includes\_head="True",

color='b',

)

plt.arrow(

0,

0,

vector2[0],

vector2[1],

head\_width=0.3,

length\_includes\_head="True",

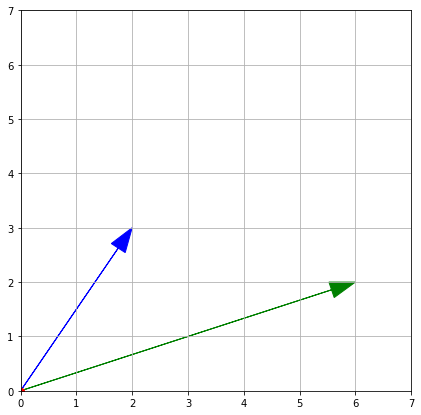
color='g',

)

plt.plot(0, 0, 'ro')

plt.grid(True)

plt.show()



El gráfico vectorial que acabamos de dibujar nos ayuda a visualizar tanto la dirección como la magnitud de una cantidad vectorial.

Ejercicio   3 / 3

1.

Traza el vector [75, 15]del ejemplo de LuxForVIP utilizando una flecha.

import numpy as np

import matplotlib.pyplot as plt

vector = np.array([75, 15])

plt.figure(figsize=(3.5,3.5))

plt.axis([0, 100, 0, 100])

plt.arrow(0, 0, vector[0], vector[1], head\_width=4, length\_includes\_head="True")

plt.xlabel('Price')

plt.ylabel('Quality')

plt.grid(True)

plt.show()

Resultado

Gráfico, Gráfico de líneas

Descripción generada automáticamente

Es correcto!

¡Vaya, qué vector!

Si añadiéramos color='r', sería aún más bonito.

2.

Traza todos los vectores bidimensionales con las valoraciones de los visitantes de LuxForVIP marcando puntos en el plano.

Pasa los vectores con las valoraciones de precio y calidad a la función plt.plot(). Asegúrate de utilizar 'ro' como segundo argumento (en el precódigo).

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

ratings\_values = [

[68,18], [81,19], [81,22], [15,75], [75,15], [17,72],

[24,75], [21,91], [76, 6], [12,74], [18,83], [20,62],

[21,82], [21,79], [84,15], [73,16], [88,25], [78,23],

[32, 81], [77, 35]]

ratings = pd.DataFrame(ratings\_values, columns=['Price', 'Quality'])

plt.figure(figsize=(3.5,3.5))

plt.axis([0, 100, 0, 100])

price = ratings['Price'].values

quality = ratings['Quality'].values

plt.plot(price, quality, 'ro')

plt.xlabel('Price')

plt.ylabel('Quality')

plt.grid(True)

plt.show()

Resultado

Gráfico, Gráfico de dispersión

Descripción generada automáticamente

¡Es correcto!

¡Qué multitud! ¡Qué multitud! Diez visitantes opinan que tanto el precio como la calidad son altos. Está claro que esta multitud vino del agregador del mercado de masas. A la segunda multitud, por el contrario, le gustó que los precios fueran razonables, pero consideró que la calidad era mala. Esta procede del agregador de marcas de lujo.

3.

Basado en el gráfico de la tarea anterior, crea dos listas separadas de vectores bidimensionales que contengan las valoraciones de los visitantes: para los visitantes que vinieron del agregador del mercado de masas y otra para los que vinieron del agregador de marcas de lujo. Nombra las variables visitors\_1 y visitors\_2.

Para ello, deberás especificar los umbrales de precio y calidad que dividen a los visitantes de ambos agregadores; y luego aplicar estos umbrales para separarlos en las variables antes mencionadas.

Muestra sus valores en la pantalla (en precódigo).

Las valoraciones de los visitantes del primer agregador se concentran en la zona superior izquierda, delimitada por las coordenadas <40 y >60. Las valoraciones del segundo sitio se encuentran en la zona inferior derecha, >60 y <40.

Puedes utilizar una declaración if...else para comprobar que cada una de las valoraciones cumpla con las condiciones de delimitación de las coordenadas y luego añadir el resultado a la lista respectiva.

import numpy as np

import pandas as pd

ratings\_values = [

[68,18], [81,19], [81,22], [15,75], [75,15], [17,72],

[24,75], [21,91], [76, 6], [12,74], [18,83], [20,62],

[21,82], [21,79], [84,15], [73,16], [88,25], [78,23],

[32, 81], [77, 35]]

ratings = pd.DataFrame(ratings\_values, columns=['Price', 'Quality'])

visitors\_1 = []

visitors\_2 = []

for visitor in list(ratings.values):

price, quality = visitor

if price < 40 and quality > 60:

visitors\_1.append(visitor)

elif price > 60 and quality < 40:

visitors\_2.append(visitor)

print('Valoraciones de los visitantes del primer agregador:', visitors\_1)

print('Valoraciones de los visitantes del primer agregador:', visitors\_2)

Resultado

Valoraciones de los visitantes del primer agregador: [array([15, 75]), array([17, 72]), array([24, 75]), array([21, 91]), array([12, 74]), array([18, 83]), array([20, 62]), array([21, 82]), array([21, 79]), array([32, 81])]

Valoraciones de los visitantes del primer agregador: [array([68, 18]), array([81, 19]), array([81, 22]), array([75, 15

¡Es correcto!

¿Quién hubiera pensado que además de crear listas en esta tarea, también dividiríamos los datos en clases y resolveríamos la primera tarea de agrupamiento?

**Suma y resta de vectores**

**Podemos sumar y restar vectores del mismo tamaño.**

Al sumar o restar vectores, se hace la operación para cada elemento de estos. Para sumar dos vectores, hay que sumar las coordenadas correspondientes de los vectores iniciales. A la hora de restar, cada coordenada del vector resultante es igual a la diferencia entre coordenadas de los vectores iniciales.

VectorCoordenadasx⃗(x1,x2,…,xn)y⃗(y1,y2,…,yn)x⃗+y⃗(x1+y1,x2+y2,…,xn+yn)x⃗−y⃗(x1−y1,x2−y2,…,xn−yn)Vector*xy*​*x*+*y*​*x*−*y*​​Coordenadas(*x*1​,*x*2​,…,*xn*​)(*y*1​,*y*2​,…,*yn*​)(*x*1​+*y*1​,*x*2​+*y*2​,…,*xn*​+*yn*​)(*x*1​−*y*1​,*x*2​−*y*2​,…,*xn*​−*yn*​)​​

Suma el vector1 — [2, 3] y el vector2 — [6, 2]:

import numpy as np

vector1 = np.array([2, 3])

vector2 = np.array([6, 2])

sum\_of\_vectors = vector1 + vector2

print(sum\_of\_vectors)

[8 5]

Resta el vector1 del vector2*:*

subtraction = vector2 - vector1

print(subtraction)

[4 -1]

Traza con flechas en el plano los vectores obtenidos:

import numpy as np

import matplotlib.pyplot as plt

vector1 = np.array([2, 3])

vector2 = np.array([6, 2])

sum\_of\_vectors = vector1 + vector2

subtraction = vector2 - vector1

plt.figure(figsize=(10.2, 10))

plt.axis([0, 8.4, -2, 6])

arrow1 = plt.arrow(

0,

0,

vector1[0],

vector1[1],

head\_width=0.3,

length\_includes\_head="True",

color='b',

)

arrow2 = plt.arrow(

0,

0,

vector2[0],

vector2[1],

head\_width=0.3,

length\_includes\_head="True",

color='g',

)

arrow3 = plt.arrow(

0,

0,

sum\_of\_vectors[0],

sum\_of\_vectors[1],

head\_width=0.3,

length\_includes\_head="True",

color='r',

)

arrow4 = plt.arrow(

0,

0,

subtraction[0],

subtraction[1],

head\_width=0.3,

length\_includes\_head="True",

color='m',

)

plt.plot(0, 0, 'ro')

plt.legend(

[arrow1, arrow2, arrow2, arrow3],

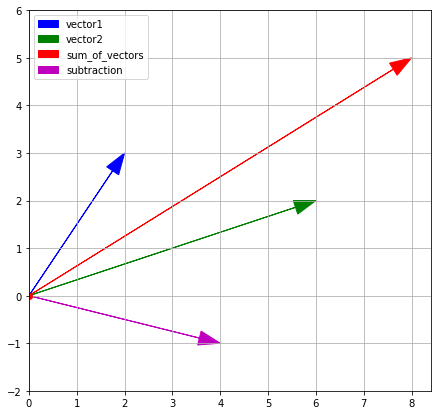
['vector1', 'vector2', 'sum\_of\_vectors', 'subtraction'],

loc='upper left',

)

plt.grid(True)

plt.show()



Si trazamos un vector que sea igual al vector1 azul en términos de longitud y dirección desde el final del vector2 verde, obtendremos el vector rojo (sum\_of\_vectors).

import numpy as np

import matplotlib.pyplot as plt

vector1 = np.array([2, 3])

vector2 = np.array([6, 2])

sum\_of\_vectors = vector1 + vector2

plt.figure(figsize=(10.2, 10))

plt.axis([0, 8.4, -2, 6])

arrow1 = plt.arrow(

0,

0,

vector1[0],

vector1[1],

head\_width=0.3,

length\_includes\_head="True",

color='b',

)

arrow2 = plt.arrow(

0,

0,

vector2[0],

vector2[1],

head\_width=0.3,

length\_includes\_head="True",

color='g',

)

arrow2new = plt.arrow(

vector1[0],

vector1[1],

vector2[0],

vector2[1],

head\_width=0.3,

length\_includes\_head="True",

color='g',

)

arrow3 = plt.arrow(

0,

0,

sum\_of\_vectors[0],

sum\_of\_vectors[1],

head\_width=0.3,

length\_includes\_head="True",

color='r',

)

plt.plot(0, 0, 'ro')

plt.legend(

[arrow1, arrow2, arrow3],

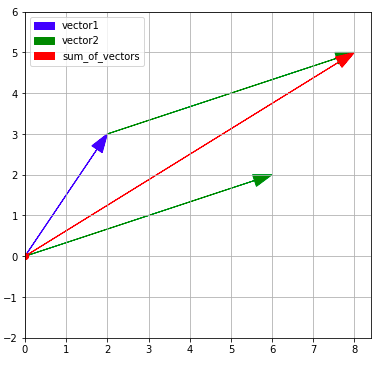
['vector1', 'vector2', 'sum\_of\_vectors'],

loc='upper left',

)

plt.grid(True)

plt.show()



El triángulo obtenido en el gráfico anterior nos da el sentido geométrico de la suma de vectores. Si cada vector es un movimiento en una dirección determinada, la suma de dos vectores es el movimiento a lo largo del primer vector seguido del movimiento a lo largo del segundo.

La diferencia entre dos vectores es un paso, por ejemplo, a lo largo del vector2, seguido de un paso en la dirección opuesta al vector1.

Traza el vector de substraction*:*

import numpy as np

import matplotlib.pyplot as plt

vector1 = np.array([2, 3])

vector2 = np.array([6, 2])

sum\_of\_vectors = vector1 + vector2

subtraction = vector2 - vector1

plt.figure(figsize=(10.2, 10))

plt.axis([0, 8.4, -2, 6])

arrow1 = plt.arrow(

0,

0,

vector1[0],

vector1[1],

head\_width=0.3,

length\_includes\_head="True",

color='b',

)

arrow2 = plt.arrow(

0,

0,

vector2[0],

vector2[1],

head\_width=0.3,

length\_includes\_head="True",

color='g',

)

arrow4 = plt.arrow(

0,

0,

subtraction[0],

subtraction[1],

head\_width=0.3,

length\_includes\_head="True",

color='m',

)

arrow1new = plt.arrow(

vector2[0],

vector2[1],

-vector1[0],

-vector1[1],

head\_width=0.3,

length\_includes\_head="True",

color='b',

)

plt.plot(0, 0, 'ro')

plt.legend(

[arrow1, arrow2, arrow4],

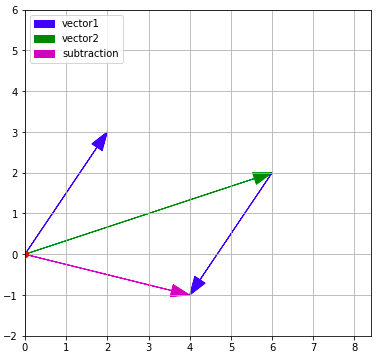
['vector1', 'vector2', 'subtraction'],

loc='upper left',

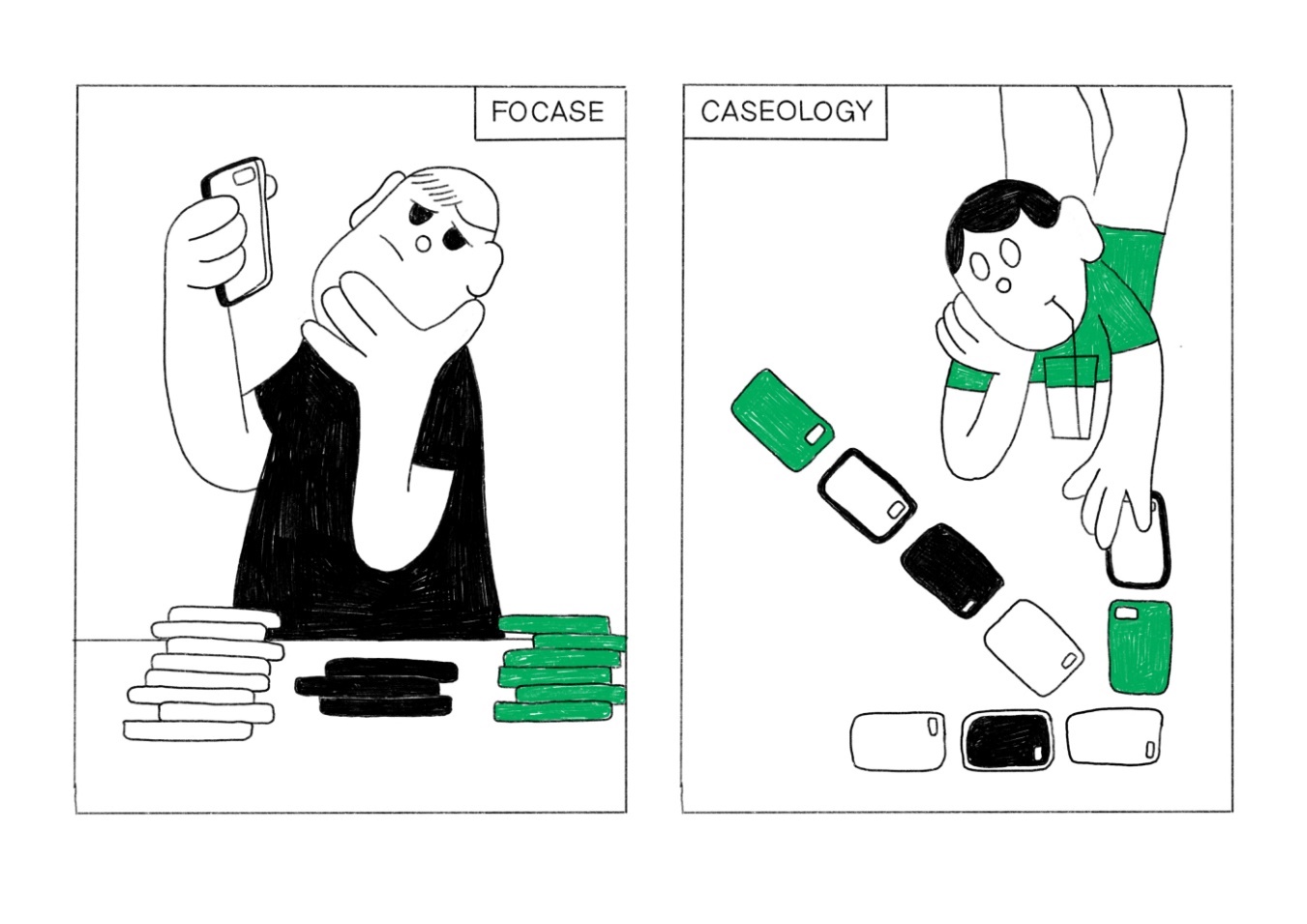
)

plt.grid(True)

plt.show()



Aquí tenemos un ejemplo: dos tiendas online con un surtido idéntico de productos, FoCase y Caseology, están planeando una fusión. Las tablas contienen fragmentos de sus listas de existencias. Las columnas representan los nombres de los productos y las cantidades.



Consulta las existencias de FoCase:

import numpy as np

import pandas as pd

quantity\_1 = [25, 63, 80, 91, 81, 55, 14, 76, 33, 71]

models = [

'Capa de silicone para o iPhone 8',

'Capa de couro para o iPhone 8',

'Capa de silicone para o iPhone XS',

'Capa de couro para o iPhone XS',

'Capa de silicone para o iPhone XS Max',

'Capa de couro para o iPhone XS Max',

'Capa de silicone para o iPhone 11',

'Capa de couro para o iPhone 11',

'Capa de silicone para o iPhone 11 Pro',

'Capa de couro para o iPhone 11 Pro',

]

stocks\_1 = pd.DataFrame({'Quantity': quantity\_1}, index=models)

print(stocks\_1)

Quantity

Capa de silicone para o iPhone 8 25

Capa de couro para o iPhone 8 63

Capa de silicone para o iPhone XS 80

Leather case for iPhone XS 91

Capa de silicone para o iPhone XS Max 81

Capa de couro para o iPhone XS Max 55

Capa de silicone para o iPhone 11 14

Capa de couro para o iPhone 11 76

Capa de silicone para o iPhone 11 Pro 33

Capa de couro para o iPhone 11 Pro 71

Consulta las existencias de Caseology:

quantity\_2 = [82, 24, 92, 48, 32, 45, 4, 34, 12, 1]

stocks\_2 = pd.DataFrame({'Quantity': quantity\_2}, index=models)

print(stocks\_2)

Quantity

Capa de silicone para o iPhone 8 82

Capa de couro para o iPhone 8 24

Capa de silicone para o iPhone XS 92

Capa de couro para o iPhone XS 48

Capa de silicone para o iPhone XS Max 32

Capa de couro para o iPhone XS Max 45

Capa de silicone para o iPhone 11 4

Capa de couro para o iPhone 11 34

Capa de silicone para o iPhone 11 Pro 12

Capa de couro para o iPhone 11 Pro 1

1.

Toma la columna de cantidades de cada tabla y conviértela en un vector. Muestra los resultados (en precódigo).

import numpy as np

import pandas as pd

quantity\_1 = [25, 63, 80, 91, 81, 55, 14, 76, 33, 71]

models = [

'Capa de silicone para o iPhone 8',

'Capa de couro para o iPhone 8',

'Capa de silicone para o iPhone XS',

'Capa de couro para o iPhone XS',

'Capa de silicone para o iPhone XS Max',

'Capa de couro para o iPhone XS Max',

'Capa de silicone para o iPhone 11',

'Capa de couro para o iPhone 11',

'Capa de silicone para o iPhone 11 Pro',

'Capa de couro para o iPhone 11 Pro',

]

stocks\_1 = pd.DataFrame({'Quantity': quantity\_1}, index=models)

quantity\_2 = [82, 24, 92, 48, 32, 45, 4, 34, 12, 1]

stocks\_2 = pd.DataFrame({'Quantity': quantity\_2}, index=models)

vector\_of\_quantity\_1 = stocks\_1['Quantity'].values

vector\_of\_quantity\_2 = stocks\_2['Quantity'].values# < escribe el código aquí >

print(

'Existencias de la primera tienda:',

vector\_of\_quantity\_1,

'\nExistencias de la segunda tienda:',

vector\_of\_quantity\_2,

)

Resultado

Existencias de la primera tienda: [25 63 80 91 81 55 14 76 33 71]

Existencias de la segunda tienda: [82 24 92 48 32 45 4 34 12 1]

¡Es correcto!

Preparados, listos, ¡fusión! (en FoCaseology)

2.

Encuentra el vector de existencias de FoCaseology después de la fusión.

Muestra los resultados (en precódigo).

import numpy as np

import pandas as pd

quantity\_1 = [25, 63, 80, 91, 81, 55, 14, 76, 33, 71]

models = [

'Capa de silicone para o iPhone 8',

'Capa de couro para o iPhone 8',

'Capa de silicone para o iPhone XS',

'Capa de couro para o iPhone XS',

'Capa de silicone para o iPhone XS Max',

'Capa de couro para o iPhone XS Max',

'Capa de silicone para o iPhone 11',

'Capa de couro para o iPhone 11',

'Capa de silicone para o iPhone 11 Pro',

'Capa de couro para o iPhone 11 Pro',

]

stocks\_1 = pd.DataFrame({'Quantity' : quantity\_1}, index=models)

quantity\_2 = [82, 24, 92, 48, 32, 45, 4, 34, 12, 1]

stocks\_2 = pd.DataFrame({'Quantity' : quantity\_2}, index=models)

vector\_of\_quantity\_1 = stocks\_1['Quantity'].values

vector\_of\_quantity\_2 = stocks\_2['Quantity'].values

vector\_of\_quantity\_united = np.add(vector\_of\_quantity\_1, vector\_of\_quantity\_2)# < escribe el código aquí >

print(vector\_of\_quantity\_united)

Resultado

[107 87 172 139 113 100 18 110 45 72]

¡Es correcto!

Por si te lo estabas preguntando, en este caso, no hemos perdido ni un solo caso. Por si acaso.

3.

Crea un DataFrame para el inventario de la tienda después de la fusión. Muestra los resultados (en precódigo).

import numpy as np

import pandas as pd

quantity\_1 = [25, 63, 80, 91, 81, 55, 14, 76, 33, 71]

models = [

'Capa de silicone para o iPhone 8',

'Capa de couro para o iPhone 8',

'Capa de silicone para o iPhone XS',

'Capa de couro para o iPhone XS',

'Capa de silicone para o iPhone XS Max',

'Capa de couro para o iPhone XS Max',

'Capa de silicone para o iPhone 11',

'Capa de couro para o iPhone 11',

'Capa de silicone para o iPhone 11 Pro',

'Capa de couro para o iPhone 11 Pro',

]

stocks\_1 = pd.DataFrame({'Quantity' : quantity\_1}, index=models)

quantity\_2 = [82, 24, 92, 48, 32, 45, 4, 34, 12, 1]

stocks\_2 = pd.DataFrame({'Quantity' : quantity\_2}, index=models)

vector\_of\_quantity\_1 = stocks\_1['Quantity'].values

vector\_of\_quantity\_2 = stocks\_2['Quantity'].values

vector\_of\_quantity\_united = np.add(vector\_of\_quantity\_1, vector\_of\_quantity\_2)# < escribe el código aquí >

print(vector\_of\_quantity\_united)

**Multiplicación de un vector por un escalar**

**Además de sumar y restar los vectores, también los podemos multiplicar por escalares.**

Cada coordenada del vector se multiplica por el mismo número:

VectorCoordenadasx⃗(x1,x2,…,xn)kx⃗(kx1,kx2,…,kxn)Vector*xkx*​Coordenadas(*x*1​,*x*2​,…,*xn*​)(*kx*1​,*kx*2​,…,*kxn*​)​​

En caso de que el número sea negativo, todas las coordenadas también cambiarán de signo.

Primero, vamos a multiplicar el vector1 - [2, 3] por un número positivo:

import numpy as np

vector1 = np.array([2, 3])

vector3 = 2 \* vector1

print(vector3)

[4 6]

Ahora lo multiplicaremos por un número negativo:

vector4 = -1 \* vector1

print(vector4)

[-2 -3]

Cuando se trata de la multiplicación por un número positivo, los vectores mantienen su dirección en el plano, aunque las flechas cambian de longitud. Cuando se trata de la multiplicación por un número negativo, los vectores cambian al sentido opuesto, además de ser escalados.

import numpy as np

import matplotlib.pyplot as plt

vector1 = np.array([2, 3])

vector3 = 2 \* vector1

vector4 = -1 \* vector1

plt.figure(figsize=(10.2, 10))

plt.axis([-4, 6, -4, 6])

arrow1 = plt.arrow(

0,

0,

vector1[0],

vector1[1],

head\_width=0.3,

length\_includes\_head="True",

color='b',

)

arrow3 = plt.arrow(

0,

0,

vector3[0],

vector3[1],

head\_width=0.3,

length\_includes\_head="True",

color='g',

)

arrow4 = plt.arrow(

0,

0,

vector4[0],

vector4[1],

head\_width=0.3,

length\_includes\_head="True",

color='m',

)

plt.plot(0, 0, 'ro')

plt.legend(

[arrow1, arrow3, arrow4],

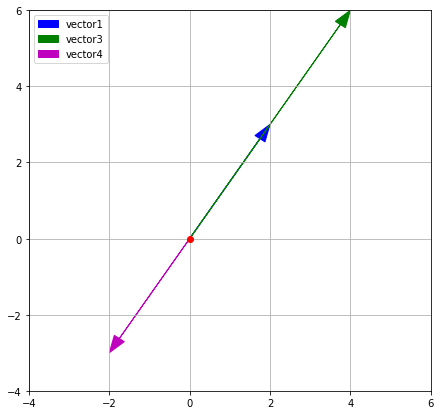
['vector1', 'vector3', 'vector4'],

loc='upper left',

)

plt.grid(True)

plt.show()



Aquí tenemos un ejemplo en el que añadimos los precios a los datos de la tienda online después de la fusión.

import numpy as np

import pandas as pd

quantity\_1 = [25, 63, 80, 91, 81, 55, 14, 76, 33, 71]

models = [

'Silicone case for iPhone 8',

'Leather case for iPhone 8',

'Silicone case for iPhone XS',

'Leather case for iPhone XS',

'Silicone case for iPhone XS Max',

'Leather case for iPhone XS Max',

'Silicone case for iPhone 11',

'Leather case for iPhone 11',

'Silicone case for iPhone 11 Pro',

'Leather case for iPhone 11 Pro',

]

stocks\_1 = pd.DataFrame({'Quantity': quantity\_1}, index=models)

quantity\_2 = [82, 24, 92, 48, 32, 45, 4, 34, 12, 1]

stocks\_2 = pd.DataFrame({'Quantity': quantity\_2}, index=models)

vector\_of\_quantity\_1 = stocks\_1['Quantity'].values

vector\_of\_quantity\_2 = stocks\_2['Quantity'].values

vector\_of\_quantity\_united = vector\_of\_quantity\_1 + vector\_of\_quantity\_2

stocks\_united = pd.DataFrame(

{'Quantity': vector\_of\_quantity\_united}, index=models

)

stocks\_united['Price'] = [30, 21, 32, 22, 18, 17, 38, 12, 23, 29]

print(stocks\_united)

Quantity Price

Silicone case for iPhone 8 107 30

Leather case for iPhone 8 87 21

Silicone case for iPhone XS 172 32

Leather case for iPhone XS 139 22

Silicone case for iPhone XS Max 113 18

Leather case for iPhone XS Max 100 17

Silicone case for iPhone 11 18 38

Leather case for iPhone 11 110 12

Silicone case for iPhone 11 Pro 45 23

Leather case for iPhone 11 Pro 72 29

Ejercicio   2 / 3

1.

Toma la columna de precios y conviértela en un vector numérico. Muestra los resultados (en precódigo).

import numpy as np

import pandas as pd

quantity\_1 = [25, 63, 80, 91, 81, 55, 14, 76, 33, 71]

models = [

'Silicone case for iPhone 8',

'Leather case for iPhone 8',

'Silicone case for iPhone XS',

'Leather case for iPhone XS',

'Silicone case for iPhone XS Max',

'Leather case for iPhone XS Max',

'Silicone case for iPhone 11',

'Leather case for iPhone 11',

'Silicone case for iPhone 11 Pro',

'Leather case for iPhone 11 Pro',

]

stocks\_1 = pd.DataFrame({'Quantity' : quantity\_1}, index=models)

quantity\_2 = [82, 24, 92, 48, 32, 45, 4, 34, 12, 1]

stocks\_2 = pd.DataFrame({'Quantity' : quantity\_2}, index=models)

vector\_of\_quantity\_1 = stocks\_1['Quantity'].values

vector\_of\_quantity\_2 = stocks\_2['Quantity'].values

vector\_of\_quantity\_united = vector\_of\_quantity\_1 + vector\_of\_quantity\_2

stocks\_united = pd.DataFrame(

{'Quantity': vector\_of\_quantity\_united}, index=models

)

stocks\_united['Price'] = [30, 21, 32, 22, 18, 17, 38, 12, 23, 29]

price\_united = stocks\_united['Price'].values# < escribe tu código aquí >

print(price\_united)

¡Es correcto!

Ahora puedes hacer cualquier cosa con los precios, sea sumar, restar o multiplicar. Ya no son solo precios, ¡también son magnitudes vectoriales!

2.

FoCaseology ha anunciado un descuento del 10 % en toda su gama. Encuentra el nuevo vector de precios, teniendo en cuenta el descuento. Muestra los resultados (en precódigo).

import numpy as np

import pandas as pd

quantity\_1 = [25, 63, 80, 91, 81, 55, 14, 76, 33, 71]

models = [

'Silicone case for iPhone 8',

'Leather case for iPhone 8',

'Silicone case for iPhone XS',

'Leather case for iPhone XS',

'Silicone case for iPhone XS Max',

'Leather case for iPhone XS Max',

'Silicone case for iPhone 11',

'Leather case for iPhone 11',

'Silicone case for iPhone 11 Pro',

'Leather case for iPhone 11 Pro',

]

stocks\_1 = pd.DataFrame({'Quantity': quantity\_1}, index=models)

quantity\_2 = [82, 24, 92, 48, 32, 45, 4, 34, 12, 1]

stocks\_2 = pd.DataFrame({'Quantity': quantity\_2}, index=models)

vector\_of\_quantity\_1 = stocks\_1['Quantity'].values

vector\_of\_quantity\_2 = stocks\_2['Quantity'].values

vector\_of\_quantity\_united = vector\_of\_quantity\_1 + vector\_of\_quantity\_2

stocks\_united = pd.DataFrame(

{'Quantity': vector\_of\_quantity\_united}, index=models

)

stocks\_united['Price'] = [30, 21, 32, 22, 18, 17, 38, 12, 23, 29]

price\_united = stocks\_united['Price'].values

price\_discount\_10 = price\_united \* 0.9# < escribe tu código aquí >

stocks\_united['10% discount price'] = price\_discount\_10.astype(int)

print(stocks\_united)

Resultado

Quantity Price 10% discount price

Silicone case for iPhone 8 107 30 27

Leather case for iPhone 8 87 21 18

Silicone case for iPhone XS 172 32 28

Leather case for iPhone XS 139 22 19

Silicone case for iPhone XS Max 113 18 16

Leather case for iPhone XS Max 100 17 15

Silicone case for iPhone 11 18 38 34

Leather case for iPhone 11 110 12 10

Silicone case for iPhone 11 Pro 45 23 20

Leather case for iPhone 11 Pro 72 29 26

¡Es correcto!

¡Fundas de teléfono con descuento! ¡Qué momento para estar con vida!

3.

Cuando las rebajas terminaron, FoCaseology subió los precios un 10 %. Crea la lista de los precios aumentados. Muestra los resultados (en precódigo).

import numpy as np

import pandas as pd

quantity\_1 = [25, 63, 80, 91, 81, 55, 14, 76, 33, 71]

models = [

'Silicone case for iPhone 8',

'Leather case for iPhone 8',

'Silicone case for iPhone XS',

'Leather case for iPhone XS',

'Silicone case for iPhone XS Max',

'Leather case for iPhone XS Max',

'Silicone case for iPhone 11',

'Leather case for iPhone 11',

'Silicone case for iPhone 11 Pro',

'Leather case for iPhone 11 Pro',

]

stocks\_1 = pd.DataFrame({'Quantity': quantity\_1}, index=models)

quantity\_2 = [82, 24, 92, 48, 32, 45, 4, 34, 12, 1]

stocks\_2 = pd.DataFrame({'Quantity': quantity\_2}, index=models)

vector\_of\_quantity\_1 = stocks\_1['Quantity'].values

vector\_of\_quantity\_2 = stocks\_2['Quantity'].values

vector\_of\_quantity\_united = vector\_of\_quantity\_1 + vector\_of\_quantity\_2

stocks\_united = pd.DataFrame(

{'Quantity': vector\_of\_quantity\_united}, index=models

)

stocks\_united['Price'] = [30, 21, 32, 22, 18, 17, 38, 12, 23, 29]

price\_united = stocks\_united['Price'].values

price\_discount\_10 = price\_united \* 0.9

stocks\_united['10% discount prise'] = price\_discount\_10.astype(int)

price\_no\_discount = price\_discount\_10 \* 1.1# < escribe tu código aquí >

stocks\_united['10% raise price'] = price\_no\_discount.astype(int)

print(stocks\_united)

Resultado

Quantity ... 10% raise price

Silicone case for iPhone 8 107 ... 29

Leather case for iPhone 8 87 ... 20

Silicone case for iPhone XS 172 ... 31

Leather case for iPhone XS 139 ... 21

Silicone case for iPhone XS Max 113 ... 17

Leather case for iPhone XS Max 100 ... 16

Silicone case for iPhone 11 18 ... 37

Leather case for iPhone 11 110 ... 11

Silicone case for iPhone 11 Pro 45 ... 22

Leather case for iPhone 11 Pro 72 ... 28

[10 rows x 4 columns]

¡Es correcto!

¡Se acabaron las fundas de teléfono con descuento! La decepción es inconmensurable. El día está arruinado. El lado bueno de las cosas es que un aumento del 10 %, después de una reducción del 10 %, todavía nos da un precio ligeramente inferior al que teníamos al principio.

**Valor medio de los vectores**

**Vamos a ver cómo se calcula el valor medio de un conjunto de vectores. Este valor representará la media de un conjunto de puntos de datos almacenados en un vector.**

Si, por ejemplo, los vectores individuales de un conjunto describen a los clientes en función de sus características, entonces el valor medio de los vectores suele describir a un cliente típico o *estadísticamente promedio*.

Para el conjunto de vectores 𝑎1,𝑎2…𝑎𝑛*a*1​,*a*2​…*an*​ (donde ∗n∗∗*n*∗ es el número total de vectores), el valor medio de los vectores es la suma de todos los vectores multiplicada por 1𝑛*n*1​. El resultado es un nuevo vector aˉ*a*ˉ:

aˉ=1n(a1+a2+⋯+an)*a*ˉ=*n*1​(*a*1​+*a*2​+⋯+*an*​)

Si el conjunto está formado por un solo vector (𝑛=1*n*=1), será igual a la media: 𝑎=𝑎1*a*=*a*1​. El valor medio de dos vectores es 𝑎=𝑎1+𝑎22*a*=2*a*1​+*a*2​​. El valor medio de un par de vectores bidimensionales es la mitad del segmento que une 𝑎1*a*1​ y 𝑎2*a*2​.

Encuentra la media del vector1 — [2, 3] y del vector2 — [6, 2]:

import numpy as np

vector1 = np.array([2, 3])

vector2 = np.array([6, 2])

vector\_mean = 0.5 \* (vector1 + vector2)

print(vector\_mean)

[4. 2.5]

Al valor medio lo llamamos vector\_mean. La primera coordenada del nuevo vector es el valor medio de las primeras coordenadas del vector1 y del vector2, y la segunda coordenada es el valor medio de las segundas coordenadas del vector1 y del vector2.

Dibujemos estos vectores en el plano. Traza el vector vector1 + vector2 y multiplícalo por 0,5.

import numpy as np

import matplotlib.pyplot as plt

vector1 = np.array([2, 3])

vector2 = np.array([6, 2])

vector\_mean = 0.5 \* (vector1 + vector2)

plt.figure(figsize=(10, 10))

plt.axis([0, 8.4, -1, 6])

arrow1 = plt.arrow(

0,

0,

vector1[0],

vector1[1],

head\_width=0.3,

length\_includes\_head="True",

color='b',

)

arrow2 = plt.arrow(

0,

0,

vector2[0],

vector2[1],

head\_width=0.3,

length\_includes\_head="True",

color='g',

)

arrow\_sum = plt.arrow(

0,

0,

vector1[0] + vector2[0],

vector1[1] + vector2[1],

head\_width=0.3,

length\_includes\_head="True",

color='r',

)

arrow\_mean = plt.arrow(

0,

0,

vector\_mean[0],

vector\_mean[1],

head\_width=0.3,

length\_includes\_head="True",

color='m',

)

plt.plot(vector1[0], vector1[1], 'ro')

plt.plot(vector2[0], vector2[1], 'ro')

plt.plot(vector\_mean[0], vector\_mean[1], 'ro')

plt.legend(

[arrow1, arrow2, arrow\_sum, arrow\_mean],

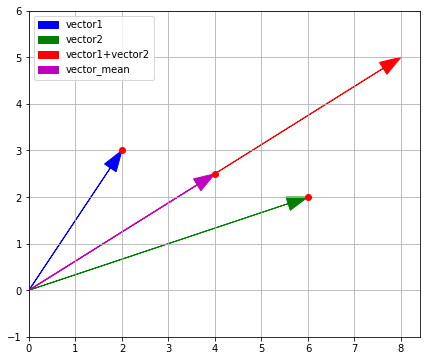
['vector1', 'vector2', 'vector1+vector2', 'vector\_mean'],

loc='upper left',

)

plt.grid(True)

plt.show()



Intentemos calcular la valoración media de los visitantes de la famosa tienda LuxForVIP.

Para calcular la valoración media del precio, utiliza la función sum(). Esta función encontrará la suma de todos los elementos del vector.

import numpy as np

import pandas as pd

ratings\_values = [

[68, 18],

[81, 19],

[81, 22],

[15, 75],

[75, 15],

[17, 72],

[24, 75],

[21, 91],

[76, 6],

[12, 74],

[18, 83],

[20, 62],

[21, 82],

[21, 79],

[84, 15],

[73, 16],

[88, 25],

[78, 23],

[32, 81],

[77, 35],

]

ratings = pd.DataFrame(ratings\_values, columns=['Price', 'Quality'])

price = ratings['Price'].values *# Matriz NumPy con todas las valoraciones de precios*

sum\_prices = sum(price) *# suma de todas las valoraciones de precios*

average\_price\_rat = sum(price) / len(price) *# valor medio de las valoraciones de precios*

print(average\_price\_rat)

49.1

1.

Encuentra la valoración media de la calidad. Muestra los resultados (en precódigo).

import numpy as np

import pandas as pd

ratings\_values = [

[68,18], [81,19], [81,22], [15,75], [75,15], [17,72],

[24,75], [21,91], [76, 6], [12,74], [18,83], [20,62],

[21,82], [21,79], [84,15], [73,16], [88,25], [78,23],

[32, 81], [77, 35]]

ratings = pd.DataFrame(ratings\_values, columns=['Price', 'Quality'])

quality = ratings['Quality'].values # Matriz NumPy con todas las valoraciones de precios

sum\_quality = sum(quality) # suma de todas las valoraciones de precios

average\_quality\_rat = sum(quality) / len(quality) # valor medio de las valoraciones de precios

print(average\_quality\_rat)

Resultado

48.4

¡Es correcto!

El visitante promedio no está satisfecho con la calidad.

2.

Combina las valoraciones medias de calidad y precio de todos los visitantes en un vector. Muestra los resultados (en precódigo).

import numpy as np

import pandas as pd

ratings\_values = [

[68,18], [81,19], [81,22], [15,75], [75,15], [17,72],

[24,75], [21,91], [76, 6], [12,74], [18,83], [20,62],

[21,82], [21,79], [84,15], [73,16], [88,25], [78,23],

[32, 81], [77, 35]]

ratings = pd.DataFrame(ratings\_values, columns=['Price', 'Quality'])

price = ratings['Price'].values

sum\_prices = sum(price)

average\_price\_rat = sum(price) / len(price)

quality = ratings['Quality'].values

average\_quality\_rat = sum(quality) / len(quality)

average\_rat = np.array([average\_price\_rat, average\_quality\_rat])

# < escribe tu código aquí >

print(average\_rat)

Resultado

[49.1 48.4]

¡Es correcto!

Al visitante promedio tampoco le gusta el precio.

3.

Indica el valor obtenido en el plano de coordenadas. ¿Puede considerarse el vector resultante como la valoración media del visitante? Muestra los resultados (en precódigo).

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

ratings\_values = [

[68,18], [81,19], [81,22], [15,75], [75,15], [17,72],

[24,75], [21,91], [76, 6], [12,74], [18,83], [20,62],

[21,82], [21,79], [84,15], [73,16], [88,25], [78,23],

[32, 81], [77, 35]]

ratings = pd.DataFrame(ratings\_values, columns=['Price', 'Quality'])

price = ratings['Price'].values

sum\_prices = sum(price)

average\_price\_rat = sum(price) / len(price)

quality = ratings['Quality'].values

average\_quality\_rat = sum(quality) / len(quality)

average\_rat = np.array([average\_price\_rat, average\_quality\_rat])

plt.figure(figsize=(7, 7))

plt.axis([0, 100, 0, 100])

plt.plot(average\_rat[0], average\_rat[1], 'mo', markersize=15, label='Average Rating')

plt.plot(price, quality, 'ro')

plt.xlabel('Price')

plt.ylabel('Quality')

plt.grid(True)

plt.title('Distribution of ratings and mean value for the whole sample')

plt.show()

Resultado

Gráfico, Gráfico de dispersión

Descripción generada automáticamente

¡Es correcto!

Hmm... Resulta que no hay visitantes cuyas valoraciones se acerquen al valor medio. ¿Cómo es posible? Recuerda que los visitantes provienen de dos agregadores diferentes y valoran la tienda de forma distinta.

4.

Calcula por separado las valoraciones medias de los visitantes que proceden del agregador de mercado de masas y de los que proceden del agregador de marcas de lujo. Muestra los resultados (en precódigo).

import numpy as np

import pandas as pd

ratings\_values = [

[68,18], [81,19], [81,22], [15,75], [75,15], [17,72],

[24,75], [21,91], [76, 6], [12,74], [18,83], [20,62],

[21,82], [21,79], [84,15], [73,16], [88,25], [78,23],

[32, 81], [77, 35]]

ratings = pd.DataFrame(ratings\_values, columns=['Price', 'Quality'])

clients\_1 = []

clients\_2 = []

for client in list(ratings.values):

if client[0] < 40 and client[1] > 60:

clients\_1.append(client)

else:

clients\_2.append(client)

average\_client\_1 = sum(clients\_1) / len(clients\_1)

print('Valoración media del primer agregador: ', average\_client\_1)

average\_client\_2 = sum(clients\_2) / len(clients\_2)# < escribe tu código aquí >

print('Valoración media del segundo agregador: ', average\_client\_2)

Resultado

Valoración media del primer agregador: [20.1 77.4]

Valoración media del segundo agregador: [78.1 19.4]

¡Es correcto!

Las valoraciones de los dos grupos difieren significativamente. Aparentemente, los visitantes de "lujo" no están contentos con la calidad, mientras que los compradores del mercado de masas sí están satisfechos con esa misma.

5.

Indica los valores obtenidos en el diagrama con las valoraciones individuales de los visitantes. ¿Puede considerarse cada media obtenida como la valoración media de los visitantes del grupo correspondiente? Muestra los resultados (en precódigo).

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

ratings\_values = [

[68,18], [81,19], [81,22], [15,75], [75,15], [17,72],

[24,75], [21,91], [76, 6], [12,74], [18,83], [20,62],

[21,82], [21,79], [84,15], [73,16], [88,25], [78,23],

[32, 81], [77, 35]]

ratings = pd.DataFrame(ratings\_values, columns=['Price', 'Quality'])

price = ratings['Price'].values

quality = ratings['Quality'].values

clients\_1 = []

clients\_2 = []

for client in list(ratings.values):

if client[0] < 40 and client[1] > 60:

clients\_1.append(client)

else:

clients\_2.append(client)

average\_client\_1 = sum(clients\_1)/len(clients\_1)

average\_client\_2 = sum(clients\_2)/len(clients\_2)

plt.figure(figsize=(7, 7))

plt.axis([0, 100, 0, 100])

# dibuja la media del grupo 1

# 'b' — azul

plt.plot(average\_client\_1[0], average\_client\_1[1], 'bo', markersize=15, label='Group 1 Mean')

# dibuja la media del grupo 2

# 'g' - verde

plt.plot(average\_client\_2[0], average\_client\_2[1], 'go', markersize=15, label='Group 1 Mean')

plt.plot(price, quality, 'ro')

plt.xlabel('Price')

plt.ylabel('Quality')

plt.grid(True)

plt.title('Distribución de valoraciones y valor medio para cada grupo')

plt.show()

¡Es correcto!

Como puedes ver, nuestros visitantes típicos son bastante leales a sus respectivos grupos.

Resultado

Gráfico, Gráfico de dispersión

Descripción generada automáticamente

**Funciones vectorizadas**

**Las funciones pueden escribirse utilizando vectores. A este tipo se le denomina**funciones vectorizadas**. Nos ayudan a operar sobre un vector completo de una manera eficiente.**

Las herramientas de NumPy nos permiten realizar varias operaciones con vectores. Si utilizamos el método np.array() para crear un vector, podemos multiplicar y dividir dos matrices del mismo tamaño (esta multiplicación y división que discutiremos a continuación, son elemento a elemento. Existen otros tipos de multiplicación y división de matrices, pero las discutiremos más adelante.). Como resultado, obtendremos un nuevo vector del mismo tamaño:

import numpy as np

array1 = np.array([2, -4, 6, -8])

array2 = np.array([1, 2, 3, 4])

array\_mult = array1 \* array2

array\_div = array1 / array2

print('Producto de dos matrices: ', array\_mult)

print('Cociente de dos matrices: ', array\_div)

Producto de dos matrices: [ 2 -8 18 -32]

Cociente de dos matrices: [ 2. -2. 2. -2.]

Si las operaciones aritméticas se realizan sobre una matriz y un solo número, la acción se aplica a cada elemento de la matriz. Y de nuevo, se forma una matriz del mismo tamaño.

Vamos a ver cómo se verá si realizamos sumas, restas y divisiones sobre una matriz con un escalar:

import numpy as np

array2 = np.array([1, 2, 3, 4])

array2\_plus\_10 = array2 + 10

array2\_minus\_10 = array2 - 10

array2\_div\_10 = array2 / 10

print('Suma: ', array2\_plus\_10)

print('Resta:', array2\_minus\_10)

print('Cociente:', array2\_div\_10)

Suma: [11 12 13 14]

Resta: [-9 -8 -7 -6]

Cociente: [0.1 0.2 0.3 0.4]

El mismo principio de "elemento por elemento" se aplica a las matrices cuando tratamos con operaciones matemáticas estándar como las de exponentes o logaritmos.

Vamos a elevar una matriz a la segunda potencia:

import numpy as np

numbers\_from\_0 = np.array([0, 1, 2, 3, 4])

squares = numbers\_from\_0 \*\* 2

print(squares)

[ 0 1 4 9 16]

Todo esto también lo podemos hacer con listas a través de bucles, pero las operaciones con vectores en NumPy son mucho más rápidas. Uno de los modos más útiles de aprovechar esta capacidad de las matrices NumPy es escribir funciones. Podemos pasar matrices como entradas a las funciones al igual que lo hacemos con otros tipos de datos.

Aquí tenemos un ejemplo: hay una matriz de valores cuyos datos deben ser escalados a efectos de nuestro análisis. Cada elemento de la matriz debe ser escalado a un número que oscile entre 0 y 1, donde 0 es el valor mínimo de la matriz y 1 el valor máximo. Aquí está la fórmula que podemos utilizar para este tipo de función:

f(x)=x−MINMAX−MIN*f*(*x*)=MAX−MIN*x*−MIN​

Para aplicar esta función a todos los elementos de la matriz, llamamos a los métodos max() y min() a fin de encontrar sus valores máximos y mínimos. Una vez definida nuestra función, podemos pasarle una matriz de números. Como resultado, obtenemos una matriz de la misma longitud, pero con elementos convertidos:

import numpy as np

def min\_max\_scale(values):

return (values - min(values)) / (max(values) - min(values))

our\_values = np.array([-20, 0, 0.5, 80, -1])

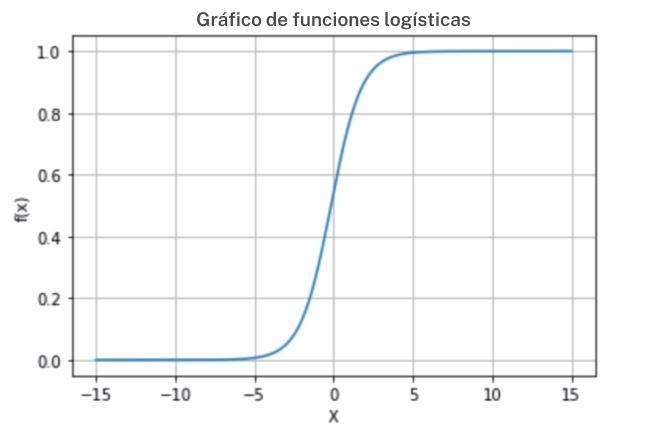
print(min\_max\_scale(our\_values))

[0. 0.2 0.205 1. 0.19 ]

A veces los valores son arbitrariamente grandes. En estos casos, una función de escalado basada en los valores mínimo y máximo no será efectiva. Podemos resolver este problema utilizando la función logística o la transformación logística:

f(x)=11+exp(−x)*f*(*x*)=1+exp(−*x*)1​

donde exp⁡()exp() es la función exponente (del lat. *exponere*, "exponer"). Eleva e*e*, el número de Euler, a la potencia del argumento. Este número recibió el nombre del gran matemático suizo Leonhard Euler y es aproximadamente igual a 2.718281828.



Escribe la función logistic\_transform() para realizar la transformación logística. Aplícala a todos los elementos de la matriz. Muestra los resultados (en precódigo).

import numpy as np

def logistic\_transform(values):

return 1 / (1 + np.exp(-values))

*# < escribe tu código aquí >*

our\_values = np.array([-20, 0, 0.5, 80, -1])

print(logistic\_transform(our\_values))

Resultado

[2.06115362e-09 5.00000000e-01 6.22459331e-01 1.00000000e+00

2.68941421e-01]

Es correcto!

El número más pequeño se convertirá en cero mientras que el más grande se convertirá en uno. Es magia.

**Vectorización de métricas**

**Vamos a escribir funciones para la evaluación de métricas mediante vectores.**

¿Cómo podemos aplicar la vectorización a las métricas de evaluación? Veamos un ejemplo. Almacena un conjunto de valores reales en la variable target y valores pronosticados en la variable predictions. Ambos conjuntos son de tipo np.array.

import numpy as np

target = np.array([0.9, 1.2, 1.4, 1.5, 1.9, 2.0])

predictions = np.array([1.0, 1.2, 1.4, 1.6, 1.8, 2.0])

Utiliza las funciones estándar de NumPy para calcular las métricas de evaluación:

* sum() — (para encontrar la suma de los elementos de una matriz)
* mean() — (para calcular el valor medio)

Llámalas de la siguiente manera: <array name>.sum() y <array name>.mean().

Por ejemplo, esta es la fórmula para calcular el error cuadrático medio (*ECM*):

ECM=1n∑i=1n(targeti−predictionsi)2ECM=*n*1​∑*i*=1*n*​(target*i*​−predictions*i*​)2

donde 𝑛*n* es la longitud de cada matriz y ΣΣ es la suma de todas las observaciones de la muestra (𝑖*i* varía de 1 a 𝑛*n*). Los elementos ordinales de los vectores target y predictions se denotan por targetitarget*i*​ and predictionsi∗predictions*i*​∗.

Escribe la fórmula utilizando sum():

def mse1(target, predictions):

n = target.size

return ((target - predictions) \*\* 2).sum() / n

Vamos a escribir la fórmula de ECM utilizando mean():

def mse2(target, predictions):

return ((target - predictions) \*\* 2).mean()

Para asegurarnos de que los resultados coincidan, apliquemos las dos funciones ECM a las matrices target y predictions.

print(mse1(target, predictions), mse2(target, predictions))

0.0049999999999999975 0.0049999999999999975

¡Los resultados coinciden!

Ejercicio   2 / 2

1.

Escribe la función para calcular el EAM utilizando mean(). Encuentra el EAM y muestra los resultados (en precódigo).

EAM=1n∑i=1n∣targeti−prediction

2.

Calcula el RECM mediante la fórmula:

RECM=ECM=1n∑i=1n(targeti−predictionsi)2

Muestra los resultados (en precódigo).

import numpy as np

def rmse(target, predictions):

return np.mean((target - predictions) \*\* 2) \*\* 0.5# < escribe tu código aquí >

target = np.array([0.9, 1.2, 1.4, 1.5, 1.9, 2.0])

predictions = np.array([1.0, 1.2, 1.4, 1.6, 1.8, 2.0])

print(rmse(target, predictions))

Resultado

0.07071067811865474

¡Es correcto!

¡Ahora nuestras métricas están oficialmente vectorizadas!

Capítulo 2/7 · Última lección

Vectores y operaciones vectoriales

**Conclusión**

En este capítulo aprendiste a:

* crear una matriz numpy multidimensional a partir de una lista de Python;
* determinar el tamaño de un vector utilizando len();
* examinar una lista de números visualizando vectores en un plano de coordenadas;
* realizar operaciones en vectores del mismo tamaño;
* convertir los valores del DataFrame en vectores mediante el atributo value;
* calcular el valor medio de un conjunto de vectores;
* escribir funciones vectorizadas para realizar operaciones con vectores de forma eficiente.

**Llévate esto contigo**

Descárgate el resumen del capítulo y la hoja informativa para poder consultarlos cuando los necesites.

* [Resumen del capítulo: Vectores y operaciones vectoriales](https://practicum-content.s3.us-west-1.amazonaws.com/new-markets/DS_sprint_11/ES/Resumen_del_captulo_Vectores_y_operaciones_vectoriales.pdf?etag=e1003efff4abca2eda9468e205213df4)
* [Hoja informativa: Vectores y operaciones vectoriales](https://practicum-content.s3.us-west-1.amazonaws.com/new-markets/DS_sprint_11/moved_DS_11_sprint_Hoja_informativa_Vectores_y_operaciones_vectoriales_esp.pdf)

En el próximo capítulo, aprenderemos a multiplicar vectores por escalares, encontrar la longitud de los vectores y la distancia entre los mismos.

Capítulo 3/7

Distancia entre vectores

**Introducción**

**Vamos a aprender a calcular la distancia entre vectores. Esto nos ayudará a hacer un modelo de machine learning completo.**

Entender el concepto de distancia entre vectores nos ayuda a aprender cómo funcionan los algoritmos de machine learning basados en la distancia. El algoritmo K-Nearest Neighborhood (KNN), el algoritmo K-Means Clustering y el Support Vector Machine (SVM) son algunos de los algoritmos de machine learning que se utilizan para crear modelos. Estos tres algoritmos utilizan la distancia euclidiana, que es un método importante para medir las distancias entre vectores y que veremos en este capítulo.

Nuestro conocimiento de cómo funcionan estos algoritmos nos ayudará a saber cómo ajustar adecuadamente los parámetros de nuestro modelo para conseguir, por ejemplo, una mayor puntuación de exactitud.

**Resumen del capítulo**

Ahora vamos a empezar nuestro viaje de aprendizaje descubriendo cómo calcular las diferentes distancias entre vectores, aprendiendo sobre el producto escalar y conociendo cómo funciona el Algoritmo KNN, que es uno de los algoritmos basados en la distancia. Luego, antes de terminar este capítulo, llegaremos a ver cómo podemos crear un modelo personalizado mediante el uso de las clases de Python y las distancias que habremos aprendido.

**Empezarás por:**

* calcular un producto escalar utilizando numpy.dot() y el operador de multiplicación de matrices;
* calcular la distancia euclidiana mediante numpy.dot() y euclidean();
* calcular la distancia de Manhattan con la función cityblock();
* diferenciar entre la distancia euclidiana y la de Manhattan;
* determinar la distancia entre vectores multidimensionales utilizando las distancias euclidiana y de Manhattan;
* utilizar la distancia euclidiana para implementar el algoritmo de vecinos más cercanos (KNN);
* desarrollar una clase de Python para crear un modelo de machine learning.

**¿Cuánto tiempo tomará?**

Ocho lecciones de 10-15 minutos cada una.

**El producto escalar**

**Vamos a descubrir qué es el producto escalar y cómo calcularlo.**

Imagínate que eres un contable que trabaja para una startup y quieres determinar la cantidad total de dinero que se debe pagar a todos los consultores que han trabajado para la empresa durante el mes anterior.

Los datos que necesitas están contenidos en dos vectores: no\_of\_hours y hourly\_rate. La matriz no\_of\_hours contiene el número de horas que han trabajado todos los consultores, mientras que la matriz hourly\_rate contiene la tarifa por hora de cada consultor respectivamente.

import numpy as np

no\_of\_hours = np.array([160, 150, 80, 95, 60, 40])

hourly\_rate = np.array([30, 35, 40, 40, 45, 50])

Para calcular el importe total, hay que realizar el siguiente cálculo:

print(sum(no\_of\_hours \* hourly\_rate))

21750

El importe total que la startup debe pagar a los consultores es de $21 750.

Multiplicamos todos los componentes y luego sumamos los valores obtenidos. También podemos describirlo diciendo que hemos calculado el producto punto o producto escalar. Como resultado de esta operación, hemos obtenido un nuevo número. Ese número es un escalar. A diferencia de los vectores, que tienen más de una característica (por ejemplo, magnitud y dirección), los escalares son unidimensionales.

Esta es la fórmula del producto escalar de dos vectores 𝑎=[𝑥1,𝑥2…𝑥𝑛]*a*=[*x*1​,*x*2​…*xn*​] y 𝑏=[𝑦1,𝑦2…𝑦𝑛]*b*=[*y*1​,*y*2​…*yn*​]:

Ten en cuenta que los vectores deben tener el mismo tamaño.

a⋅b=x1∗y1+x2∗y2+⋯+xn∗yn*a*⋅*b*=*x*1​∗*y*1​+*x*2​∗*y*2​+⋯+*xn*​∗*yn*​

El producto escalar de los vectores a*a* y b*b* se marca normalmente con paréntesis ⟨𝑎,𝑏⟩⟨*a*,*b*⟩ o un punto 𝑎⋅𝑏*a*⋅*b*.

Encuentra el mismo producto escalar utilizando la función numpy.dot() y asegúrate de que los resultados coincidan:

import numpy as np

no\_of\_hours = np.array([160, 150, 80, 95, 60, 40])

hourly\_rate = np.array([30, 35, 40, 40, 45, 50])

print(np.dot(no\_of\_hours , hourly\_rate ))

21750

El operador de multiplicación de matrices nos permite calcular el producto de manera aún más sencilla. El operador se marca con @.

import numpy as np

no\_of\_hours = np.array([160, 150, 80, 95, 60, 40])

hourly\_rate = np.array([30, 35, 40, 40, 45, 50])

print(no\_of\_hours @ hourly\_rate )

21750

Los resultados del cálculo son los mismos en todos los métodos.

El producto escalar es una de las múltiples formas de combinar vectores al que recurrimos en varias ocasiones importantes. Lo más importante para nuestros propósitos es el cálculo de distancias entre vectores. La aplicación del producto escalar es esencial para los algoritmos de machine learning basados en la distancia.

Veamos otro ejemplo. Vector 𝑞*q* contiene las cantidades de cada modelo de auricular en el almacén. Vector 𝑐*c* contiene los precios de estos productos. El valor total de todos los artículos es la suma de los productos de la cantidad de cada modelo y de su precio, o para decirlo en términos más sencillos, el producto escalar (𝑞,𝑐)(*q*,*c*).

La tabla contiene las cantidades y los precios de los modelos de auriculares de dos tiendas que pertenecen a la cadena minorista Ear You Are.

import numpy as np

import pandas as pd

store1\_price = [

209.9,

119.9,

53.9,

31.9,

19.9,

109.9,

59.99,

22.9,

81.11,

32.9,

]

store1\_quantity = [19, 11, 8, 15, 23, 7, 14, 9, 10, 4]

store2\_price = [209.9, 124.9, 42.9, 27.9, 23.9, 109.9, 49.9, 24.9, 89.9, 32.9]

store2\_quantity = [10, 16, 20, 9, 18, 12, 10, 11, 18, 22]

models = [

'Apple AirPods Pro',

'Apple AirPods MV7N2RU/A',

'JBL Tune 120TWS',

'JBL TUNE 500BT',

'JBL JR300BT',

'Huawei Freebuds 3',

'Philips TWS SHB2505',

'Sony WH-CH500',

'Sony WF-SP700N',

'Sony WI-XB400',

]

items1 = pd.DataFrame(

{'Price': store1\_price, 'Quantity': store1\_quantity}, index=models

)

items2 = pd.DataFrame(

{'Price': store2\_price, 'Quantity': store2\_quantity}, index=models

)

print('Store 1\n', items1, '\n')

print('Store 2\n', items2)

Store 1

Price Quantity

Apple AirPods Pro 209.9 19

Apple AirPods MV7N2RU/A 119.9 11

JBL Tune 120TWS 53.9 8

JBL TUNE 500BT 31.9 15

JBL JR300BT 19.9 23

Huawei Freebuds 3 109.9 7

Philips TWS SHB2505 59.99 14

Sony WH-CH500 22.9 9

Sony WF-SP700N 81.11 10

Sony WI-XB400 3290 4

Store 2

Price Quantity

Apple AirPods Pro 209.9 10

Apple AirPods MV7N2RU/A 124.9 16

JBL Tune 120TWS 42.9 20

JBL TUNE 500BT 27.9 9

JBL JR300BT 23.9 18

Huawei Freebuds 3 109.9 12

Philips TWS SHB2505 49.9 10

Sony WH-CH500 24.9 11

Sony WF-SP700N 89.9 18

Sony WI-XB400 32.9 22

Ejercicio

Encuentra el valor total de los artículos para cada tienda Ear You Are. Guarda los valores en las variables items1\_value y items2\_value.

Encuentra el valor total de los artículos en ambas tiendas y guárdalo en la variable valor\_total. Muestra los resultados (en precódigo).

import numpy as np

import pandas as pd

store1\_price = [209.9, 119.9, 53.9, 31.9, 19.9, 109.9, 59.99, 22.9, 81.11, 32.9]

store1\_quantity = [19, 11, 8, 15, 23, 7, 14, 9, 10, 4]

store2\_price = [209.9, 124.9, 42.9, 27.9, 23.9, 109.9, 49.9, 24.9, 89.9, 32.9]

store2\_quantity = [10, 16, 20, 9, 18, 12, 10, 11, 18, 22]

models = ['Apple AirPods Pro',

'Apple AirPods MV7N2RU/A',

'JBL Tune 120TWS',

'JBL TUNE 500BT',

'JBL JR300BT',

'Huawei Freebuds 3',

'Philips TWS SHB2505',

'Sony WH-CH500',

'Sony WF-SP700N',

'Sony WI-XB400',

]

items1 = pd.DataFrame({'Price':store1\_price,

'Quantity':store1\_quantity}, index=models)

items2 = pd.DataFrame({'Price':store2\_price,

'Quantity':store2\_quantity}, index=models)

items1\_price = items1['Price'].values

items1\_quantity = items1['Quantity'].values

items2\_price = items2['Price'].values # < completa aquí >

items2\_quantity = items2['Quantity'].values # < completa aquí >

# valor de los artículos de la tienda 1

items1\_value = items1\_price @ items1\_quantity # < completa aquí >

# Valor de los artículos de la tienda 2

items2\_value = items2\_price @ items2\_quantity # < completa aquí >

# Valor total de los artículos de ambas tiendas

total\_value = items1\_value + items2\_value # < completa aquí >

print('Valor total de los artículos de las dos tiendas:', total\_value, 'USD')

Resultado

Valor total de los artículos de las dos tiendas: 19502.760000000002 USD

¡Es correcto!

Vaya, los auriculares no son baratos hoy en día.

**Distancia euclidiana**

**En esta lección aprenderemos a utilizar el producto escalar para encontrar la distancia entre los puntos del plano.**

Empecemos por observar el vector a=(x,y)*a*=(*x*,*y*). ¿Cómo podemos saber su longitud? Lo primero que podemos hacer es imaginar que nuestro vector es la hipotenusa de un triángulo rectángulo. Esto es útil porque, si recordamos un poco las propiedades de los triángulos rectángulos, podemos utilizar el teorema de Pitágoras para encontrar la longitud de nuestro vector.

Puede que te estés diciendo: "Muy bien, estupendo, pero ¿qué tiene eso que ver con el producto escalar?". Bueno, fíjate en lo que ocurre cuando calculas el producto escalar de un vector y de sí mismo:

a⋅a=(x∗x)+(y∗y)=x2+y2*a*⋅*a*​=(*x*∗*x*)+(*y*∗*y*)=*x*2+*y*2​

Si sacamos la raíz cuadrada de este, tendremos el mismo resultado que obtenemos gracias al teorema de Pitágoras:

=x2+y2=*x*2+*y*2​

Está bastante bien, ¿verdad? Lo mejor de esto es que nos ofrece una forma práctica de calcular las longitudes de los vectores utilizando np.dot():

import numpy as np

a = np.array([5, 6])

*# mediante el teorema de Pitágoras*

print(((a[0] \*\* 2) + (a[1] \*\* 2)) \*\* 0.5)

print()

*# mediante el producto punto ; ¿no es esto más bonito?*

print(np.dot(a, a) \*\* 0.5)

7.810249675906654

7.810249675906654

Ahora podemos utilizar esta información para encontrar la diferencia entre dos vectores: el vector a*a* y el vector b*b*.

Como puedes ver, podemos pensar en la diferencia entre los dos vectores como en un nuevo vector (b−a)(*b*−*a*). Entonces, con el truco de antes, podemos calcular la longitud de este vector:

d2(a,b)=(b−a)⋅(b−a)=(x2−x1)2+(y2−y1)2*d*2​(*a*,*b*)​=(*b*−*a*)⋅(*b*−*a*)​=(*x*2​−*x*1​)2+(*y*2​−*y*1​)2​​

Esta manera de medir la distancia se llama distancia euclidiana, que se utiliza en muchos algoritmos de machine learning como métrica de distancia por defecto. La distancia euclidiana es siempre la distancia más corta entre dos puntos en un plano.

Esta distancia se escribe como *d2(a,b)d2​(a,b),* donde *dd* tiene el subíndice 2 para indicar que las coordenadas del vector están elevadas a la segunda potencia.

Ahora vamos a utilizar NumPy para encontrar la distancia euclidiana entre *a=(5,6)a=(5,6)* y *b=(1,3)b=(1,3)*:

import numpy as np

a = np.array([5, 6])

b = np.array([1, 3])

d = np.dot(b - a, b - a) \*\* 0.5

print('La distancia entre a y b es de', d)

La distancia entre a y b es de 5.0

SciPy tiene una librería dedicada al cálculo de distancias, que se llama **distance**. Podemos importarla desde scipy.spatial y llamar a distance.euclidean() para calcular la distancia euclidiana:

import numpy as np

from scipy.spatial import distance

a = np.array([5, 6])

b = np.array([1, 3])

d = distance.euclidean(a, b)

print('La distancia entre a y b es de', d)

La distancia entre a y b es de 5.0

Los resultados de los cálculos realizados mediante la fórmula y mediante la función son los mismos. La función puede utilizarse para calcular distancias tanto entre puntos como entre vectores que unen el origen y esos puntos. Calcularemos la distancia entre vectores en las tareas que vienen a continuación.

Aquí tienes un ejemplo. El mapa muestra las localidades que están ubicadas en el área de entrega de la empresa de entrega con drones Flying Stuff. Consideremos que Willowford es el origen. Las coordenadas del resto de las localidades están trazadas a lo largo de los ejes X e Y en kilómetros.



Para planificar la ruta del dron, almacenaremos los datos de las entregas en tres variables:

* x\_axis — coordenadas de cada población a lo largo del eje X
* y\_axis — coordenadas a lo largo del eje Y
* shipments — número medio de entregas semanales en cada ciudad

Combinaremos estos datos en un DataFrame y los mostraremos en la pantalla:

import numpy as np

import pandas as pd

x\_axis = np.array(

[

0.0,

0.18078584,

9.32526599,

17.09628721,

4.69820241,

11.57529305,

11.31769349,

14.63378951,

]

)

y\_axis = np.array(

[

0.0,

7.03050245,

9.06193657,

0.1718145,

5.1383203,

0.11069032,

3.27703365,

5.36870287,

]

)

deliveries = np.array([5, 7, 4, 3, 5, 2, 1, 1])

town = [

'Willowford',

'Otter Creek',

'Springfield',

'Arlingport',

'Spadewood',

'Goldvale',

'Bison Flats',

'Bison Hills',

]

data = pd.DataFrame(

{

'x\_coordinates\_km': x\_axis,

'y\_coordinates\_km': y\_axis,

'Deliveries': deliveries,

},

index=town,

)

print(data)

x\_coordinates\_km y\_coordinates\_km Deliveries

Willowford 0.000000 0.000000 5

Otter Creek 0.180786 7.030502 7

Springfield 9.325266 9.061937 4

Arlingport 17.096287 0.171815 3

Spadewood 4.698202 5.138320 5

Goldvale 11.575293 0.110690 2

Bison Flats 11.317693 3.277034 1

Bison Hills 14.633790 5.368703 1

Calcularemos las distancias entre los puntos utilizando las longitudes de los vectores. Con este fin, vamos a extraer las coordenadas de los puntos en la variable llamada vectors.

vectors = data[['x\_coordinates\_km', 'y\_coordinates\_km']].values

print(vectors)

[[ 0. 0. ]

[ 0.18078584 7.03050245]

[ 9.32526599 9.06193657]

[17.09628721 0.1718145 ]

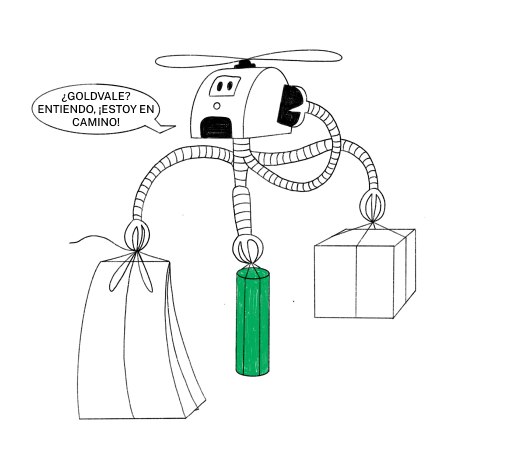
[ 4.69820241 5.1383203 ]

[11.57529305 0.11069032]

[11.31769349 3.27703365]

[14.63378951 5.36870287]]

Vamos a optimizar el proceso de reparto.



1.

Crea una tabla con las distancias entre las localidades y guárdala en la variable distances. Presenta los datos en forma de lista de listas. Cada fila debe representar la distancia entre cada población y todas las demás.

Añade los nombres de todas las localidades a la tabla y muestra los resultados (en precódigo).

import numpy as np

import pandas as pd

from scipy.spatial import distance

x\_axis = np.array([0., 0.18078584, 9.32526599, 17.09628721,

4.69820241, 11.57529305, 11.31769349, 14.63378951])

y\_axis = np.array([0.0, 7.03050245, 9.06193657, 0.1718145,

5.1383203, 0.11069032, 3.27703365, 5.36870287])

deliveries = np.array([5, 7, 4, 3, 5, 2, 1, 1])

town = [

'Willowford',

'Otter Creek',

'Springfield',

'Arlingport',

'Spadewood',

'Goldvale',

'Bison Flats',

'Bison Hills',

]

data = pd.DataFrame(

{

'x\_coordinates\_km': x\_axis,

'y\_coordinates\_km': y\_axis,

'Deliveries': deliveries,

},

index=town,

)

vectors = data[['x\_coordinates\_km', 'y\_coordinates\_km']].values

distances = []

#Completa el código:

for town\_from in range(len(town)):

row = []

for town\_to in range(len(town)):

value = distance.euclidean(vectors[town\_from], vectors[town\_to])# < escribe tu código aquí >

row.append(value)

distances.append(row)# < escribe tu código aquí >

distances\_df = pd.DataFrame(distances, index=town, columns=town)

print(distances\_df)

Resultado

Willowford Otter Creek ... Bison Flats Bison Hills

Willowford 0.000000 7.032826 ... 11.782578 15.587520

Otter Creek 7.032826 0.000000 ... 11.752414 14.548226

Springfield 13.003049 9.367403 ... 6.118404 6.466869

Arlingport 17.097151 18.253103 ... 6.560071 5.750786

Spadewood 6.962431 4.897694 ... 6.876194 9.938258

Goldvale 11.575822 13.331114 ... 3.176805 6.082853

Bison Flats 11.782578 11.752414 ... 0.000000 3.920660

Bison Hills 15.587520 14.548226 ... 3.920660 0.000000

[8 rows x 8 columns]

¡Es correcto!

¡Gran trabajo! Parece que las distancias son admisibles. Ahora vamos a utilizar esta información para encontrar la mejor ubicación para el nuevo almacén de Flying Stuff.

2.

El número de entregas semanales en cada ciudad ya se conoce. Gracias a toda esta información, puedes seleccionar la mejor población para ubicar el almacén de Flying Stuff.

Calcula para cada localidad cuántos kilómetros recorrerá el dron en una semana a condición de que el almacén se encuentre en esa ciudad. Busca las distancias entre poblaciones, duplícalas (ida y vuelta) y multiplícalas por el número de entregas semanales. Guarda el resultado en la lista deliveries\_in\_week.

Encuentra la población con el menor recorrido total.

Muestra los resultados (en precódigo).

import numpy as np

import pandas as pd

from scipy.spatial import distance

x\_axis = np.array([0., 0.18078584, 9.32526599, 17.09628721,

4.69820241, 11.57529305, 11.31769349, 14.63378951])

y\_axis = np.array([0.0, 7.03050245, 9.06193657, 0.1718145,

5.1383203, 0.11069032, 3.27703365, 5.36870287])

deliveries = np.array([5, 7, 4, 3, 5, 2, 1, 1])

town = [

'Willowford',

'Otter Creek',

'Springfield',

'Arlingport',

'Spadewood',

'Goldvale',

'Bison Flats',

'Bison Hills',

]

data = pd.DataFrame(

{

'x\_coordinates\_km': x\_axis,

'y\_coordinates\_km': y\_axis,

'Deliveries': deliveries,

},

index=town,

)

vectors = data[['x\_coordinates\_km', 'y\_coordinates\_km']].values

distances = []

for town\_from in range(len(town)):

row = []

for town\_to in range(len(town)):

value = distance.euclidean(vectors[town\_from], vectors[town\_to])

row.append(value)

distances.append(row)

deliveries\_in\_week = []

for i in range(len(town)):

total\_distance = 2 \* np.dot(np.array(distances[i]), deliveries)

deliveries\_in\_week.append(total\_distance)# < escribe tu código aquí >

deliveries\_in\_week\_df = pd.DataFrame(

{'Distance': deliveries\_in\_week}, index=town

)

print(deliveries\_in\_week\_df)

print()

print('Localidad del almacén:', deliveries\_in\_week\_df['Distance'].idxmin()) # < escribe tu código aquí >)

Resultado

Distance

Willowford 475.734656

Otter Creek 409.688787

Springfield 454.776713

Arlingport 701.242510

Spadewood 334.564924

Goldvale 513.067627

Bison Flats 459.977717

Bison Hills 577.345345

Localidad del almacén: Spadewood

¡Es correcto!

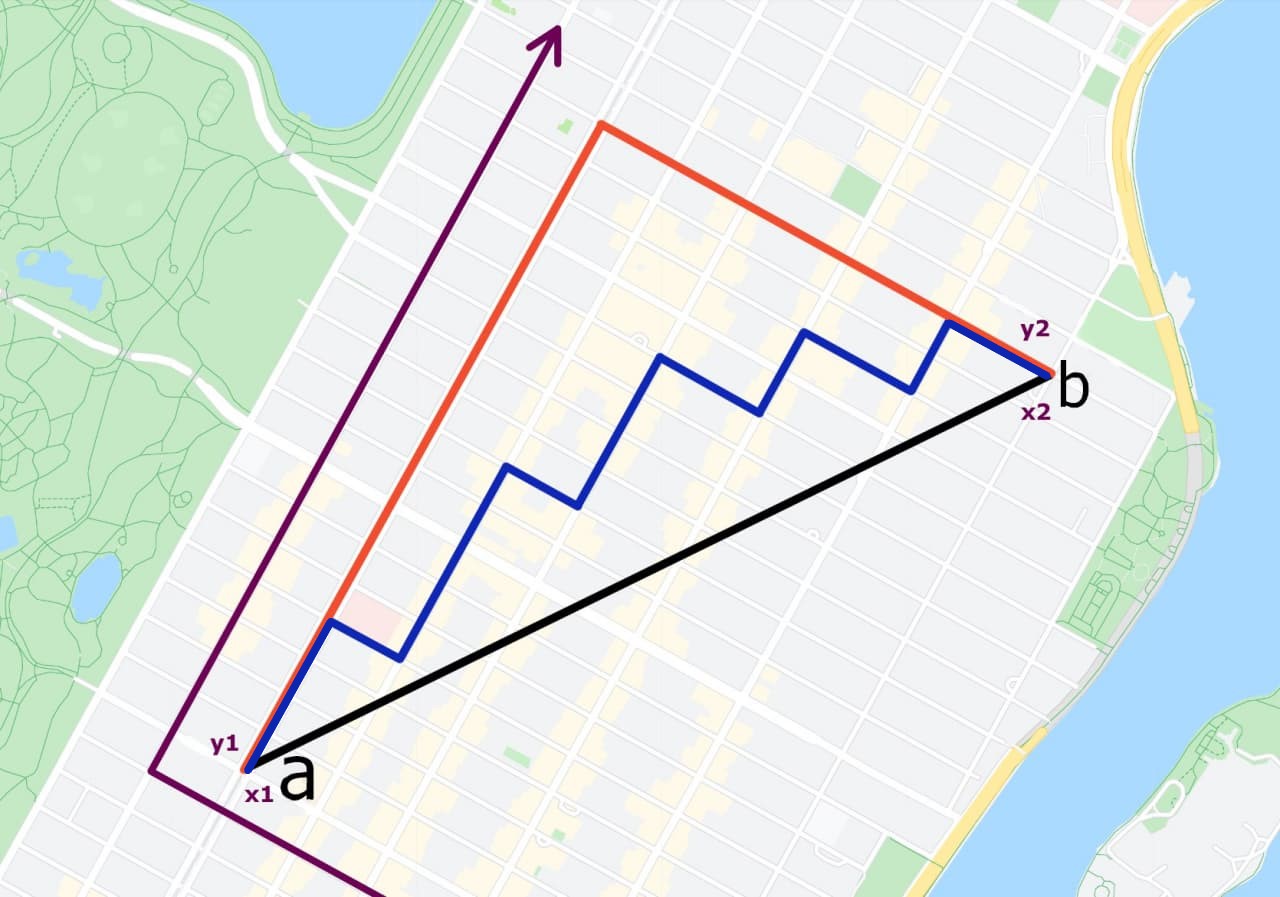
Según tu trabajo, los propietarios de Flying Stuff deberían construir su nuevo almacén en Spadewood. ¡Se te da muy bien esto!

**Distancia Manhattan**

**Veamos ahora la otra métrica de distancia comúnmente utilizada, que es la distancia Manhattan.**

Estaría bien si pudiéramos recorrer todas las distancias en línea recta, como el dron de Flying Stuff de la lección anterior. Pero la verdad es que a menudo hay otras cosas que se interponen en nuestro camino y nos obligan a tomar una ruta que no es la óptima.

Imagínate que estás en Nueva York y necesitas ir del punto a*a* al punto b*b*. Lo ideal sería tomar el camino negro que representa la distancia euclidiana. Sin embargo, a menos que nos salgan alas, nos veremos en la obligación de tomar un camino que recorra las distintas manzanas en lugar de movernos en línea recta.



Estos otros trayectos se denominan distancia Manhattan o distancia entre manzanas. Por definición, la distancia Manhattan es la suma de las diferencias absolutas de las coordenadas de todos los puntos. Podemos calcular la distancia Manhattan entre los puntos a=(𝑥1,𝑦1)*a*=(*x*1​,*y*1​) y b=(𝑥2,𝑦2)*b*=(*x*2​,*y*2​) utilizando la siguiente fórmula:

d1(a,b)=∣x1−x2∣+∣y1−y2∣*d*1​(*a*,*b*)=∣*x*1​−*x*2​∣+∣*y*1​−*y*2​∣

Esta distancia la escribimos como d1(a,b)*d*1​(*a*,*b*) donde d*d* tiene el subíndice 1 para indicar que las coordenadas del vector se elevan a la primera potencia (a diferencia del proceso de la distancia euclidiana, por ejemplo).

Para ilustrar mejor esta idea, veamos de nuevo nuestro mapa de Manhattan. Podemos encontrar la distancia Manhattan para los puntos a=(𝑥1,𝑦1)*a*=(*x*1​,*y*1​) y b=(𝑥2,𝑦2)*b*=(*x*2​,*y*2​) y representar esta distancia utilizando una línea azul o una roja. ¿Notas algo interesante en las longitudes de las líneas roja y azul? ¡Son iguales! Gracias a la distancia Manhattan, podemos encontrar más de un camino que se ajuste a nuestros cálculos.

1.

Escribe la función manhattan\_distance() para encontrar la distancia Manhattan. Esta función debería tomar dos vectores como argumentos y devolver la distancia. Resuelve la tarea con la ayuda de una función vectorizada. No utilices ningún bucle.

import numpy as np

def manhattan\_distance(first, second):

# Calcula la suma de las diferencias absolutas entre las coordenadas

return np.sum(np.abs(first - second))

first = np.array([3, 11])

second = np.array([1, 6])

print(manhattan\_distance(first, second))

Resultado

7

¡Es correcto!

¡No te perderás en NY! Eso es seguro.

2.

En una zona dada de Manhattan, vamos a encontrar el taxi más cercano de los tres que están desocupados.

Variables declaradas:

* avenues\_df— lista de avenidas con coordenadas
* streets\_df — lista de calles con coordenadas
* address — ubicación del cliente (avenida y calle)
* taxis — ubicación de los taxis

Fíjate en que, al igual que en el caso de la distancia euclidiana, podemos calcular las distancias desde la dirección hasta los coches utilizando la función correspondiente de la librería SciPy. Si no puedes adivinar cómo se llama, echa un vistazo a la [documentación del módulo de distancias de SciPy](https://docs.scipy.org/doc/scipy/reference/spatial.distance.html) (materiales en inglés) o consulta la pista.

Calcula las distancias y guarda el resultado en la variable taxis\_distance. Determina el número de serie del taxi más cercano. Muestra la ubicación del taxi (avenida y calle) en la pantalla (en precódigo).

import numpy as np

import pandas as pd

from scipy.spatial import distance

avenues\_df = pd.DataFrame(

[0, 153, 307, 524], index=['Park', 'Lexington', '3rd', '2nd']

)

streets\_df = pd.DataFrame(

[0, 81, 159, 240, 324], index=['76', '75', '74', '73', '72']

)

address = ['Lexington', '74']

taxis = [

['Park', '72'],

['2nd', '75'],

['3rd', '76'],

]

address\_vector = np.array(

[avenues\_df.loc[address[0]], streets\_df.loc[address[1]]]

)

taxi\_distances = []

for taxi in taxis:

taxi\_vector = np.array([avenues\_df.loc[taxi[0]], streets\_df.loc[taxi[1]]])

taxi\_distances.append(distance.cityblock([x[0] for x in address\_vector], [x[0] for x in taxi\_vector]))

index = np.array(taxi\_distances).argmin()# < escribe tu código aquí >

print(taxis[index])

Resultado

['3rd', '76']

¡Es correcto!

Ahora también puedes moverte fácilmente por Nueva York en taxi. Si consigues coger uno.

**Métricas en el espacio multidimensional**

**Hasta ahora, solo hemos examinado la distancia entre los puntos de un plano, pero nuestras métricas de distancia también funcionan en el espacio multidimensional.**

A la hora de trabajar con los datos para tareas de machine learning, cada observación puede considerarse como un vector. Por supuesto, como ya sabes, las observaciones con las que trabajamos no suelen ser bidimensionales, más bien contienen información en muchas dimensiones.

vector1 es un vector en cuatro dimensiones:

vector1 = np.array([1, 2, 3, 4])

El hecho de tener los datos representados en un vector multidimensional nos permite encontrar información significativa utilizando la distancia euclidiana como métrica de distancia. Esta métrica de distancia puede ayudarnos a identificar el grado de relación entre diferentes observaciones.

Para calcular la distancia euclidiana entre vectores multidimensionales 𝑎=(𝑥1,𝑥2,…𝑥𝑛)*a*=(*x*1​,*x*2​,…*xn*​) y 𝑏=(𝑦1,𝑦2,…𝑦𝑛)*b*=(*y*1​,*y*2​,…*yn*​), podemos aplicar la misma fórmula de antes, pero con un término extra por cada punto de datos adicional:

d2(a,b)=(x1−y1)2+(x2−y2)2+⋯+(xn−yn)2=∑i=1n(xi−yi)2*d*2​(*a*,*b*)​=(*x*1​−*y*1​)2+(*x*2​−*y*2​)2+⋯+(*xn*​−*yn*​)2​=*i*=1∑*n*​(*xi*​−*yi*​)2​​

La distancia Manhattan también puede calcularse de la siguiente manera:

d1(a,b)=∣x1−y1∣+∣x2−y2∣+⋯+∣xn−yn∣=∑i=1n∣xi−yi∣*d*1​(*a*,*b*)​=∣*x*1​−*y*1​∣+∣*x*2​−*y*2​∣+⋯+∣*xn*​−*yn*​∣=*i*=1∑*n*​∣*xi*​−*yi*​∣​

Incluso cuando el número de coordenadas es superior a dos, podemos utilizar las conocidas funciones distance.euclidean() y distance.cityblock() para calcular distancias en el espacio multidimensional.

Vamos a calcular las distancias euclidiana y Manhattan entre los vectores 𝑎=(4,2,3,0,5)*a*=(4,2,3,0,5) y 𝑏=(1,0,3,2,6)*b*=(1,0,3,2,6).

Podemos ver que tanto nuestro cálculo manual como la función de SciPy no tienen problemas para lidiar con la distancia multidimensional:

import numpy as np

from scipy.spatial import distance

a = np.array([4, 2, 3, 0, 5])

b = np.array([1, 0, 3, 2, 6])

*# cálculo manual*

d = np.dot(b - a, b - a)\*\*0.5

print(d)

print()

*# cálculo con la función euclidiana*

e = distance.euclidean(a, b)

print(e)

4.242640687119285

4.242640687119285

Lo mismo ocurre con la distancia Manhattan:

import numpy as np

from scipy.spatial import distance

a = np.array([4, 2, 3, 0, 5])

b = np.array([1, 0, 3, 2, 6])

d = np.abs(b - a).sum()

print(d)

print()

e = distance.cityblock(a, b)

print(e)

8

8

Veamos un ejemplo de la vida real. En el sitio web de la agencia inmobiliaria Cribswithclass.com, cada anuncio tiene una serie de parámetros: número de dormitorios, superficie total, tamaño de la cocina, etc. Si al cliente le gusta algún apartamento en particular, el sistema de recomendación le ofrecerá opciones similares.

Los datos de los apartamentos se encuentran en la tabla **pandas**:

import pandas as pd

columns = ['bedrooms', 'total area', 'kitchen', 'living area', 'floor', 'total floors']

realty = [

[1, 38.5, 6.9, 18.9, 3, 5],

[1, 38.0, 8.5, 19.2, 9, 17],

[1, 34.7, 10.3, 19.8, 1, 9],

[1, 45.9, 11.1, 17.5, 11, 23],

[1, 42.4, 10.0, 19.9, 6, 14],

[1, 46.0, 10.2, 20.5, 3, 12],

[2, 77.7, 13.2, 39.3, 3, 17],

[2, 69.8, 11.1, 31.4, 12, 23],

[2, 78.2, 19.4, 33.2, 4, 9],

[2, 55.5, 7.8, 29.6, 1, 25],

[2, 74.3, 16.0, 34.2, 14, 17],

[2, 78.3, 12.3, 42.6, 23, 23],

[2, 74.0, 18.1, 49.0, 8, 9],

[2, 91.4, 20.1, 60.4, 2, 10],

[3, 85.0, 17.8, 56.1, 14, 14],

[3, 79.8, 9.8, 44.8, 9, 10],

[3, 72.0, 10.2, 37.3, 7, 9],

[3, 95.3, 11.0, 51.5, 15, 23],

[3, 69.3, 8.5, 39.3, 4, 9],

[3, 89.8, 11.2, 58.2, 24, 25],

]

df\_realty = pd.DataFrame(realty, columns=columns)

print(df\_realty)

bedrooms total area kitchen living area floor total floors

0 1 38.5 6.9 18.9 3 5

1 1 38.0 8.5 19.2 9 17

2 1 34.7 10.3 19.8 1 9

3 1 45.9 11.1 17.5 11 23

4 1 42.4 10.0 19.9 6 14

5 1 46.0 10.2 20.5 3 12

6 2 77.7 13.2 39.3 3 17

7 2 69.8 11.1 31.4 12 23

8 2 78.2 19.4 33.2 4 9

9 2 55.5 7.8 29.6 1 25

10 2 74.3 16.0 34.2 14 17

11 2 78.3 12.3 42.6 23 23

12 2 74.0 18.1 49.0 8 9

13 2 91.4 20.1 60.4 2 10

14 3 85.0 17.8 56.1 14 14

15 3 79.8 9.8 44.8 9 10

16 3 72.0 10.2 37.3 7 9

17 3 95.3 11.0 51.5 15 23

18 3 69.3 8.5 39.3 4 9

19 3 89.8 11.2 58.2 24 25

Podemos considerar el conjunto de parámetros de cada anuncio como un vector. Necesitamos calcular las distancias euclidiana y Manhattan.

Ejercicio   2 / 2

1.

Guarda los vectores de los apartamentos con índices 3 y 11 en las variables vector\_first y vector\_second. Encuentra las distancias euclidiana y Manhattan entre los mismos. Muestra los resultados (en precódigo).

import pandas as pd

from scipy.spatial import distance

columns = [

'dormitorios',

'superficie total',

'cocina',

'superficie habitable',

'planta',

'número de plantas',

]

realty = [

[1, 38.5, 6.9, 18.9, 3, 5],

[1, 38.0, 8.5, 19.2, 9, 17],

[1, 34.7, 10.3, 19.8, 1, 9],

[1, 45.9, 11.1, 17.5, 11, 23],

[1, 42.4, 10.0, 19.9, 6, 14],

[1, 46.0, 10.2, 20.5, 3, 12],

[2, 77.7, 13.2, 39.3, 3, 17],

[2, 69.8, 11.1, 31.4, 12, 23],

[2, 78.2, 19.4, 33.2, 4, 9],

[2, 55.5, 7.8, 29.6, 1, 25],

[2, 74.3, 16.0, 34.2, 14, 17],

[2, 78.3, 12.3, 42.6, 23, 23],

[2, 74.0, 18.1, 49.0, 8, 9],

[2, 91.4, 20.1, 60.4, 2, 10],

[3, 85.0, 17.8, 56.1, 14, 14],

[3, 79.8, 9.8, 44.8, 9, 10],

[3, 72.0, 10.2, 37.3, 7, 9],

[3, 95.3, 11.0, 51.5, 15, 23],

[3, 69.3, 8.5, 39.3, 4, 9],

[3, 89.8, 11.2, 58.2, 24, 25],

]

df\_realty = pd.DataFrame(realty, columns=columns)

vector\_first = df\_realty.loc[3].values # < escribe tu código aquí >

vector\_second = df\_realty.loc[11].values# < escribe tu código aquí >

print('Distancia euclidiana:',distance.euclidean(vector\_first, vector\_second)) # < escribe tu código aquí >

print('Distancia Manhattan:', distance.cityblock(vector\_first, vector\_second))# < escribe tu código aquí >

Resultado

Distancia euclidiana: 42.73417835877976

Distancia Manhattan: 71.7

¡Es correcto!

En función del tipo de cálculo de la distancia por el que optemos, podemos determinar distintas medidas de similitud para dos apartamentos. La decisión de utilizar una u otra suele depender de cual de ellas mejora en mayor medida las métricas del usuario.

2.

Supongamos que a un cliente le ha gustado el apartamento con índice 12. Encuentra el apartamento más similar basándote en la distancia euclidiana.

Crea una lista con las distancias desde cada vector hasta el vector 12. Calcula el índice de la lista más similar y guárdalo en la variable best\_index.

import numpy as np

import pandas as pd

from scipy.spatial import distance

columns = [

'bedrooms',

'total area',

'kitchen',

'living area',

'floor',

'total floors',

]

realty = [

[1, 38.5, 6.9, 18.9, 3, 5],

[1, 38.0, 8.5, 19.2, 9, 17],

[1, 34.7, 10.3, 19.8, 1, 9],

[1, 45.9, 11.1, 17.5, 11, 23],

[1, 42.4, 10.0, 19.9, 6, 14],

[1, 46.0, 10.2, 20.5, 3, 12],

[2, 77.7, 13.2, 39.3, 3, 17],

[2, 69.8, 11.1, 31.4, 12, 23],

[2, 78.2, 19.4, 33.2, 4, 9],

[2, 55.5, 7.8, 29.6, 1, 25],

[2, 74.3, 16.0, 34.2, 14, 17],

[2, 78.3, 12.3, 42.6, 23, 23],

[2, 74.0, 18.1, 49.0, 8, 9],

[2, 91.4, 20.1, 60.4, 2, 10],

[3, 85.0, 17.8, 56.1, 14, 14],

[3, 79.8, 9.8, 44.8, 9, 10],

[3, 72.0, 10.2, 37.3, 7, 9],

[3, 95.3, 11.0, 51.5, 15, 23],

[3, 69.3, 8.5, 39.3, 4, 9],

[3, 89.8, 11.2, 58.2, 24, 25],

]

df\_realty = pd.DataFrame(realty, columns=columns)

# índice del apartamento preferido

preference\_index = 12

preference\_vector = df\_realty.loc[preference\_index].values

distances = []

for i in range(df\_realty.shape[0]):

other\_vector = df\_realty.loc[i].values

distances.append(distance.euclidean(preference\_vector, other\_vector)) # < escribe tu código aquí >

# < escribe tu código aquí >

# argsort() devuelve los índices en orden

# aumentar los valores de una matriz

best\_index = np.array(distances).argsort()[1]# < escribe tu código aquí >

print('Índice del apartamento más similar:', best\_index)

Resultado

Índice del apartamento más similar: 15

¡Es correcto!

El sistema ha recomendado el apartamento con índice 15. ¡Tiene un dormitorio extra! Sin embargo, la cocina es casi dos veces más pequeña.

**Conclusión**

¡Felicidades! Has completado el curso Álgebra lineal.

Has aprendido a:

* realizar operaciones con matrices y vectores;
* crear clases de aprendizaje automático;
* explicar cómo funciona la regresión lineal.

Ahora sigue el curso de métodos numéricos. Aprenderás sobre el descenso de gradiente y la potenciación de gradiente.