Mondragon Unibertsitatea
Goi Eskola Politeknikoa

Algebra lineal

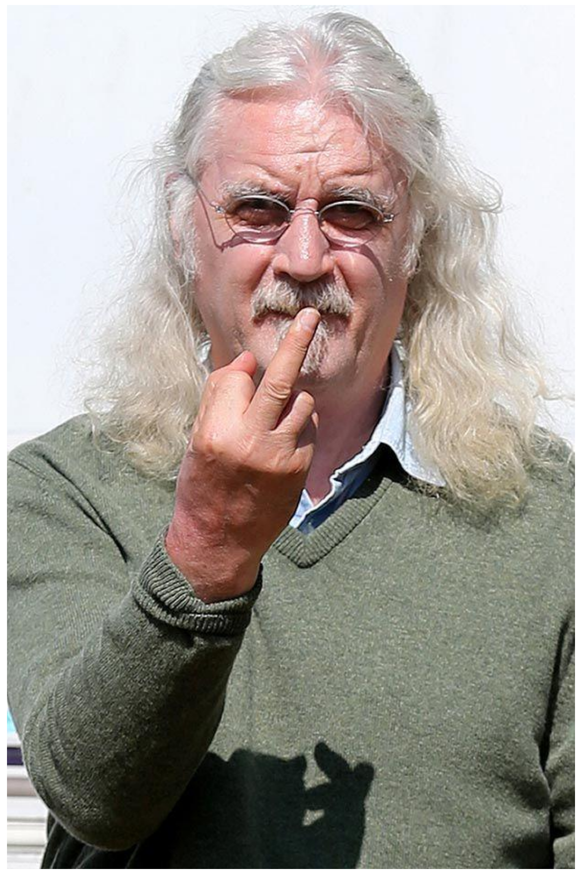
Fundamentos del Aprendizaje Automático

1

Introducción

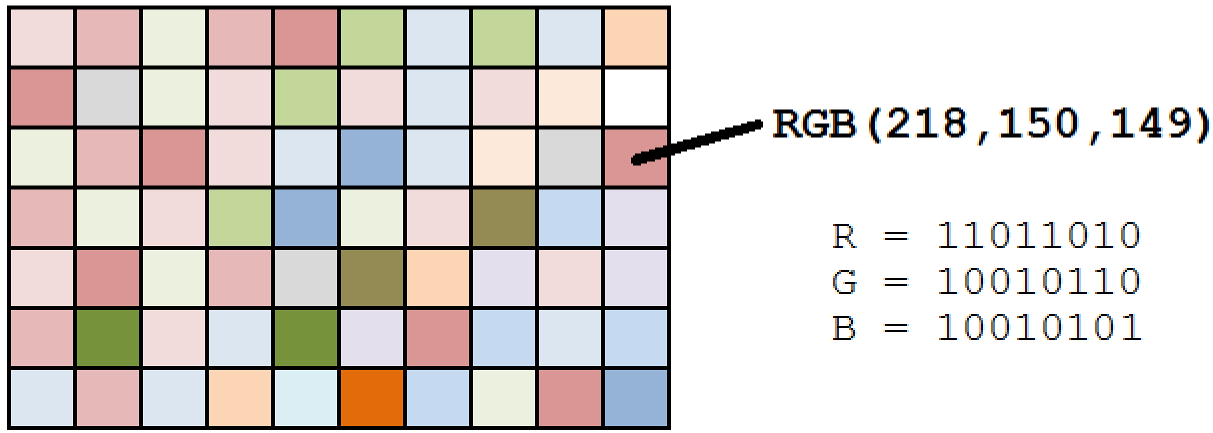
Introducción

- *"Is there anything more useless or less useful than algebra?"*
  - *Billy Connolly*

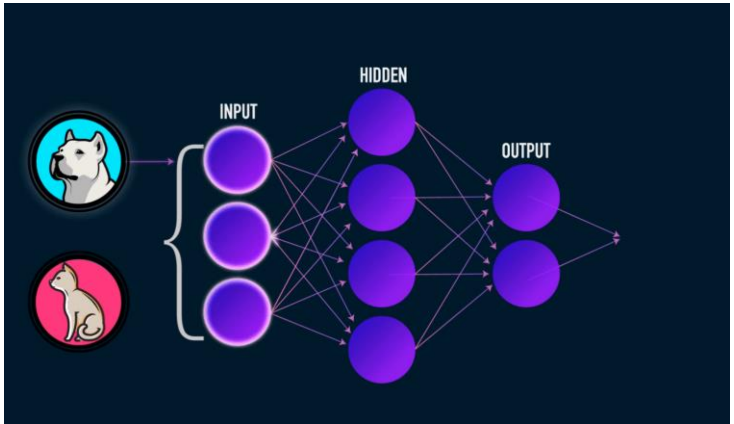


Introducción

- Ejemplos:
  - Como podemos representar una imagen y sus características de tal forma que un ordenador lo entienda?



Matrices



NLP

	I	like	hate	databases
D1	1	1	0	1
D2	1	0	1	1

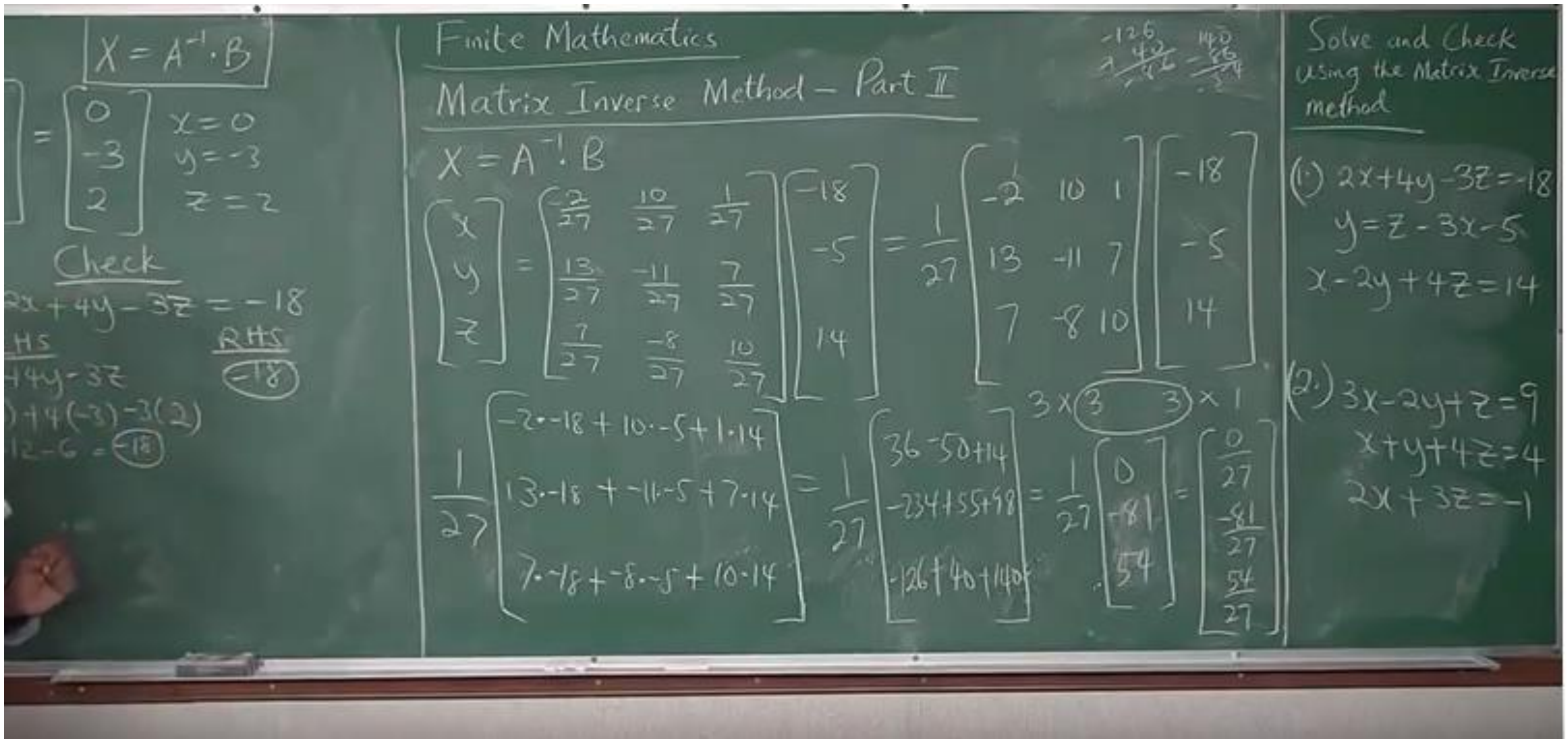
Bases de la algebra lineal

- Matrices
  - Vectores
  - Normas
  - Tipos de matrices
  - Tensores
- Factorización
- Prácticas
  - Implementación KNN
  - (Implementación PCA)

2

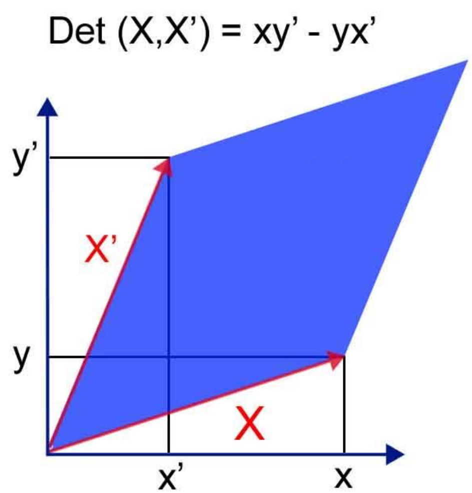
Matrices

Matrices


Finite Mathematics
Matrix Inverse Method – Part II
X = A⁻¹·B
x = 0
y = -3
z = 2
Check
2x + 4y − 3z = −18
RHS
Solve and Check using the Matrix Inverse method
(1.) 2x+4y−3z=−18
y=z−3x−5
x−2y+4z=14
(2.) 3x−2y+z=9
x+y+4z=4
2x+3z=−1

Propiedades

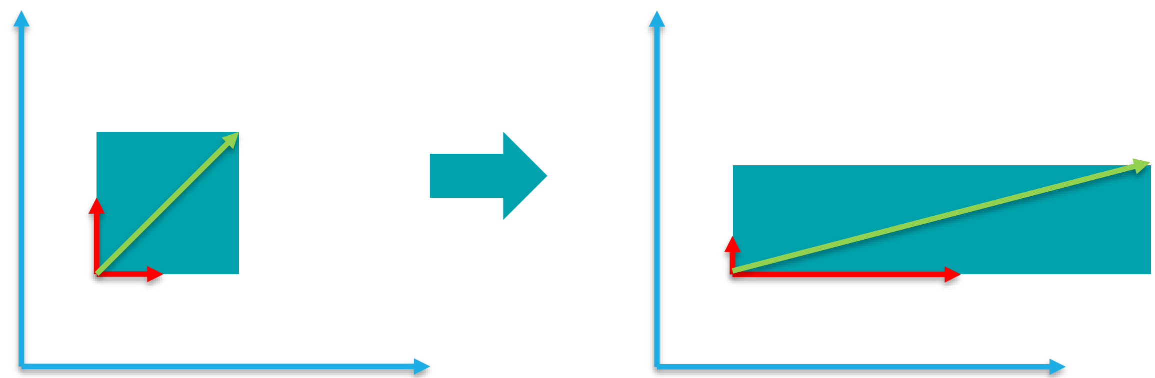
- **Rango**: número máximo de columnas linealmente independientes
- **Determinante**: Una forma multilineal alternada de un cuerpo
  - Objetivo: estudio de número de soluciones de los sistemas de ecuaciones lineales
  - Noción general: áreas y volúmenes



Propiedades

- Vectores y valores propios (Eigenvector y eigenvalue)
  - Vector propio: se dice del vector V que en una aplicación lineal T multiplica al valor propio no nulo **k** en la siguiente forma: *T(V) = kV*
  - Resultado: un múltiplo **escalar** de sí mismos, no cambian su dirección
    - Ese escalar se denomina como valor propio

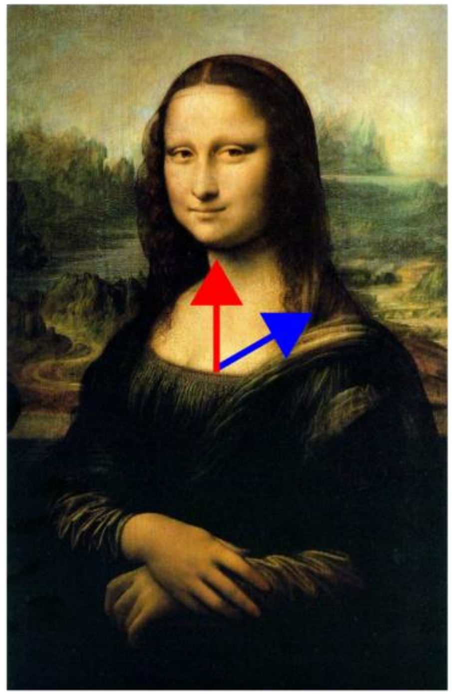
Ejemplo: transformación linear

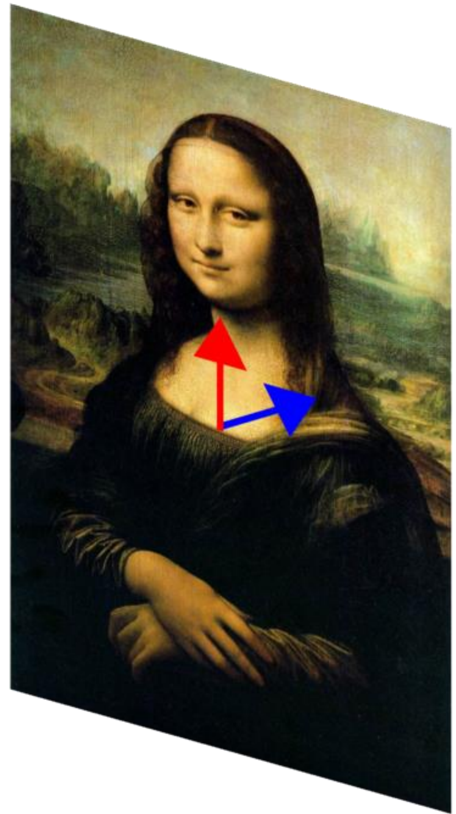


$$\begin{bmatrix} 1 & 2 \\ 0 & 3 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \end{bmatrix} = \begin{bmatrix} 5 \\ 6 \end{bmatrix}$$

  - *Los vectores cuya dirección se mantiene después de una transformación linear se llaman los vectores propios de una matriz => vectores rojos*

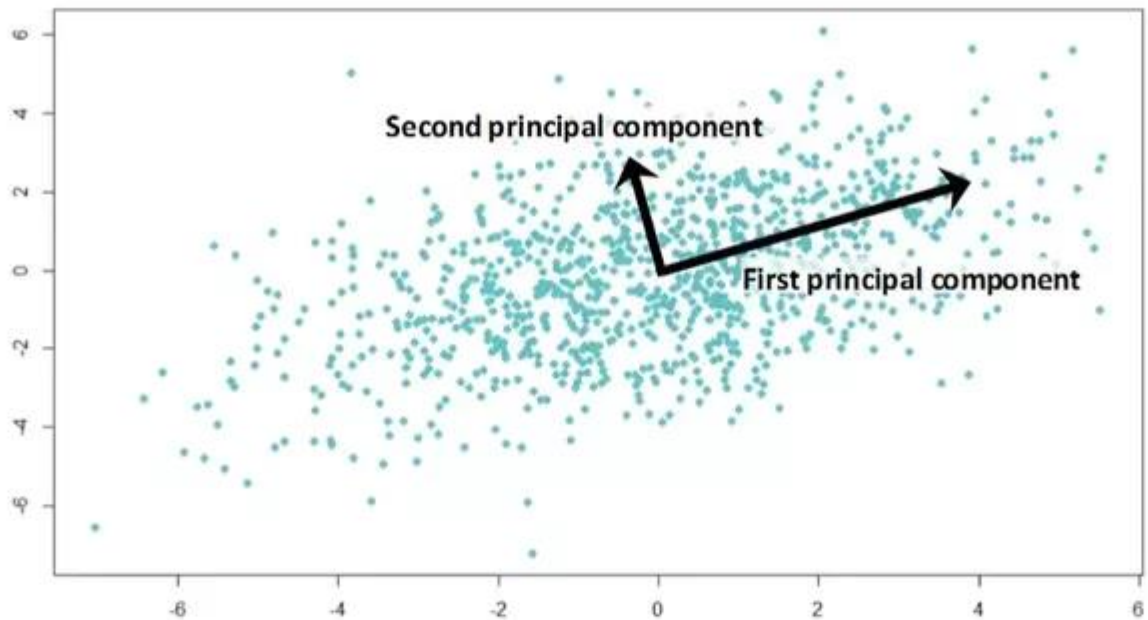
Propiedades





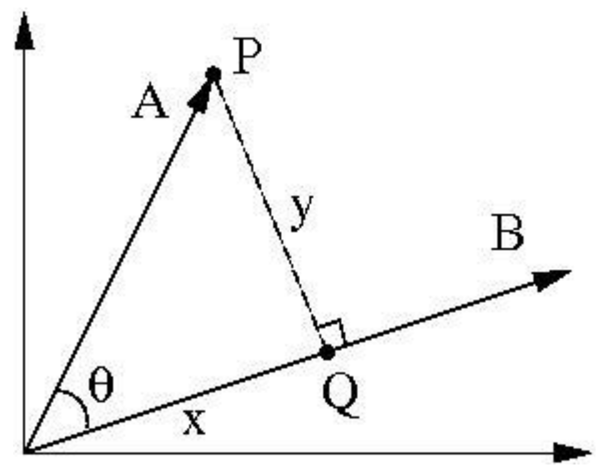
Propiedades

- Uso de Eigenvectors en ciencia de datos:
  - Reducción de dimensionalidad y elementos redundantes en los datos
  - PCA: Principal component análisis
    - Los datos se proyectan en direcciones donde la varianza es máxima



Producto escalar (dot product)

- Podemos calcular la suma de los elementos multiplicados de dos vectores de la misma longitud y devuelve un único número (escalar)
- El producto escalar es básico para calcular proyecciones de vectores, decomposiciones y determinar la ortogonalidad.
  - Proyecciones de vectores
  - **Ortogonalidad**: Dos vectores son ortogonales o **perpendiculares** cuando forman ángulo recto entre sí. Si el producto escalar de dos vectores es cero, ambos vectores son ortogonales
  - Decomposiciones



$$c = a.b$$
$$c = (a1\ x\ b1 + a2\ x\ b2 + a3\ x\ b3)$$

Producto escalar (dot product)

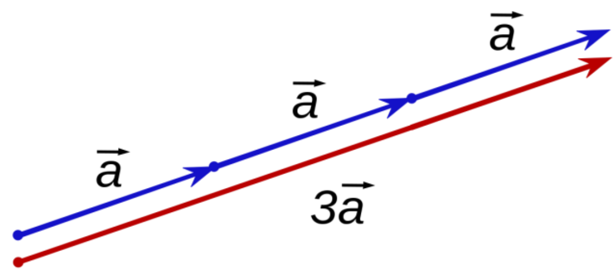
Mondragon Unibertsitatea
Goi Eskola Politeknikoa

In [1]:
```
from numpy import array

a = array([1, 2, 3])
print(a)

b = array([1, 2, 3])
print(b)
# multiplicar vectores
c = a.dot(b)
print(c)
```

```
[1 2 3]
[1 2 3]
14
```

Multiplicación vector - escalar

- Un vector puede ser multiplicado por un escalar, teniendo como efecto el escalado de la magnitud del vector
  - $c = s\ x\ v = sv$
  - $c = (s\ x\ v_1, s\ x\ v_2,\ s\ x\ v_3\ )$
  - $$\begin{aligned} c[0] &= v[0]\ x\ s \\ c[1] &= v[1]\ x\ s \\ c[2] &= v[2]\ x\ s \end{aligned}$$




```
from numpy import array
# define vector
a = array([1, 2, 3])

print(a)
# define scalar
s = 0.5
print(s)
# multiplication
c = s * a
print(c)
```

```
[1 2 3]
0.5
[0.5 1.  1.5]
```

Norma del vector

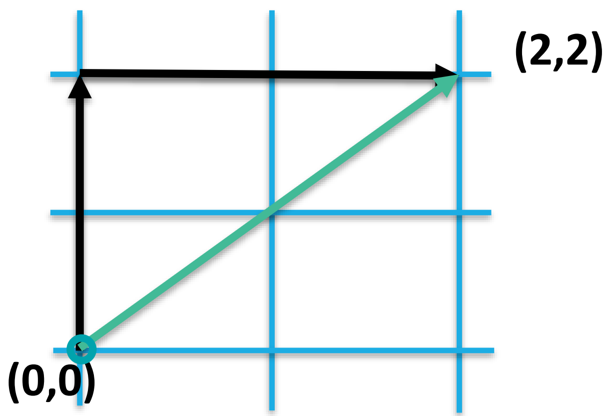
- La **longitud del vector** se denomina comúnmente como la norma del vector o magnitud del vector.
  - La norma siempre será un número positivo, excepto para un vector con todos sus valores a 0.
  - Se calcula utilizando una medida que mide la distancia del vector desde su espacio vectorial de origen.
    - Ejemplo para v (1,2,3) => su espacio vectorial de origen sería (0,0,0)

Cálculo de la norma

- Existen varias formas para calcular la norma de un vector, siendo tres las más comunes:
  - Norma L^1
  - Norma L^2
  - Norma L^3

Norma L^1

- También denominada como norma **Manhattan o taxicab**
  - El nombre se refiere a la distancia que tiene que recorrer un taxi en un grid para llegar desde su origen hasta el destino
  - $L^1(v) = |v|_1 = \sum_{i=1}^{n} |vi|$




```
import numpy as np
from numpy.linalg import norm
# define vector
a = np.array([2, 2])
print(a)
# calculate norm
l1 = norm(a, 1)
print(l1)
```

```
[2 2]
4.0
```

Norma L^2

- También conocida como la norma Euclidiana
  - Calcula la distancia desde el origen del espacio del vector
  - $L^1(v) = |v|_1 = \sqrt[2]{{a_1}^2 + {a_2}^2 + {a_3}^2}$
  - d(q,p) = $\sqrt[2]{\sum_{i=1}^{n}(q_i - pi)^2}$

```
import numpy as np
from numpy.linalg import norm
# define vector
a = np.array([2, 2])
print(a)
# calculate norm
l1 = norm(a)
print(l1)
```

```
[2 2]
2.8284271247461903
```

Norma L^3

- Norma de vector máximo (vector max norm)
  - La norma máxima se calcula devolviendo el valor máximo del vector
    - Devuelve la mayor magnitud entre los elementos del vector
  - $L_{inf}(v) = ||v||_{inf}$ = max a_1, a_2, a_3

```
from math import inf
from numpy import array
from numpy.linalg import norm

a = array([1, 2, 3])
print(a)

maxnorm = norm(a, inf)
print(maxnorm)
```

```
[1 2 3]
3.0
```

Tipos de matrices

- Matriz cuadrada
  - Mismo número de columnas y filas
  - Conceptos de eigenvectors o determinantes solo en matrices cuadradas
  - Uso:
    - Transformaciones lineales simples
      - Rotación de imágenes

$$M = \begin{pmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \\ 1 & 2 & 3 \end{pmatrix}$$

Tipos de matrices

- Matriz simétrica
  - “It is no exaggeration to say that symmetric matrices are the most important matrices the world will ever see”
    - Introduction to Linear Algebra

$$M = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 2 & 1 & 2 & 3 & 4 \\ 3 & 2 & 1 & 2 & 3 \\ 4 & 3 & 2 & 1 & 2 \\ 5 & 4 & 3 & 2 & 1 \end{pmatrix}$$

  - Una matriz siempre es cuadrada e igual a su propia transpuesta

Tipos de matrices

- Matriz triangular
  - Matriz cuadrada donde todos los valores de la parte superior derecha y/o parte inferior izquierda son 0.
  - Ejemplo: matriz triangular inferior

$$M = \begin{pmatrix} 1 & 2 & -3 \\ 0 & 2 & 3 \\ 0 & 0 & 3 \end{pmatrix}$$

  - Las ecuaciones con matrices son más fáciles de resolver si las matrices son triangulares

Tipos de matrices

Mondragon Unibertsitatea
Goi Eskola Politeknikoa

- Matriz diagonal
  - Matrices donde los valores fuera de la diagonal de la matriz tienen el valor 0
  - La matriz no tiene que ser cuadrada

$$D = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 4 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

  - Objetivo principal: caracterizar la matriz
    - Se pueden calcular instantaneamente la determinante, rango, eigenvalues, si es invertible…

Tipos de matrices

- Matriz de identidad
  - Matriz cuadrada que no cambia el vector cuando se multiplica con esta matriz
  - Se representa usando la notación ***I***

$$I = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Tipos de matrices

- Matriz ortogonal
  - Dos vectores son ortogonales si su producto escalar es cero.
    - Una línea es ortogonal a la otra cuando son perpendiculares
  - Si la longitud de esos vectores es 1, los vectores se consideran ortonormales, ya que son ortogonales y están normalizadas
  - **Matriz ortogonal**: matriz cuadrada cuyas columnas son vectores unitarios ortogonales
  - $A.A^T = I \quad \Rightarrow \quad A^{-1} = A^T$

Tensores

- Los tensores son objetos matemáticos que generalizan escalares, vectores y matrices a dimensiones más altas
  - Array multidimensional

https://hadrienj.github.io/posts/Deep-Learning-Book-Series-2.1-Scalars-Vectors-Matrices-and-Tensors/

Tensores

- Tensores en Python

```python
from numpy import array
# primer tensor
A = array([
[[1,2,3], [4,5,6], [7,8,9]],
[[11,12,13], [14,15,16], [17,18,19]],
[[21,22,23], [24,25,26], [27,28,29]]])
# segundo tensor
B = array([
[[1,2,3], [4,5,6], [7,8,9]],
[[11,12,13], [14,15,16], [17,18,19]],
[[21,22,23], [24,25,26], [27,28,29]]])
```

- A.ndim
- >>> 3

Tensores: aritmética

- C = A + B   // Suma
- C = A – B   // Resta
- C = A * B   // Producto Hadamard
- C = A / B   // División
- C = tensordot (A, B, axes=0) // tensor product

Tensores: aritmética

- Suma - resta
  - Se llevan a cabo entre tensores de la misma dimensión
  - Elemento a elemento

```
>>> # suma
... C = A + B
... print(C)
```

```
[[[ 2  4  6]
  [ 8 10 12]
  [14 16 18]]

 [[22 24 26]
  [28 30 32]
  [34 36 38]]

 [[42 44 46]
  [48 50 52]
  [54 56 58]]]
```

Tensores: aritmética

- Tensor Hadamard product
  - Multiplicación elemento a elemento de un tensor con otro tensor con las mismas dimensiones
    - Resultado: nuevo tensor con las mismas dimensiones donde cada escalar es la multiplicación elemento a elemento del tensor padre

$$A = \begin{pmatrix} a_{1,1,1} & a_{1,2,1} & a_{1,3,1} \\ a_{2,1,1} & a_{2,2,1} & a_{2,3,1} \end{pmatrix}, \begin{pmatrix} a_{1,1,2} & a_{1,2,2} & a_{1,3,2} \\ a_{2,1,2} & a_{2,2,2} & a_{2,3,2} \end{pmatrix}$$

$$B = \begin{pmatrix} b_{1,1,1} & b_{1,2,1} & b_{1,3,1} \\ b_{2,1,1} & b_{2,2,1} & b_{2,3,1} \end{pmatrix}, \begin{pmatrix} b_{1,1,2} & b_{1,2,2} & b_{1,3,2} \\ b_{2,1,2} & b_{2,2,2} & b_{2,3,2} \end{pmatrix}$$

$$C = A \circ B$$

$$C = \begin{pmatrix} a_{1,1,1} \times b_{1,1,1} & a_{1,2,1} \times b_{1,2,1} & a_{1,3,1} \times b_{1,3,1} \\ a_{2,1,1} \times b_{2,1,1} & a_{2,2,1} \times b_{2,2,1} & a_{2,3,1} \times b_{2,3,1} \end{pmatrix}, \begin{pmatrix} a_{1,1,2} \times b_{1,1,2} & a_{1,2,2} \times b_{1,2,2} & a_{1,3,2} \times b_{1,3,2} \\ a_{2,1,2} \times b_{2,1,2} & a_{2,2,2} \times b_{2,2,2} & a_{2,3,2} \times b_{2,3,2} \end{pmatrix}$$

```
# multiplicación entre tensores
C = A * B
```

Tensores: aritmética

- Producto entre tensores

Entre vectores

$$a = \begin{pmatrix} a_1 \\ a_2 \end{pmatrix}$$

$$b = \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}$$

$$C = a \otimes b$$

$$C = \begin{pmatrix} a_1 \times \begin{pmatrix} b_1 \\ b_2 \end{pmatrix} \\ a_2 \times \begin{pmatrix} b_1 \\ b_2 \end{pmatrix} \end{pmatrix}$$

$$C = \begin{pmatrix} a_1 \times b_1 & a_1 \times b_2 \\ a_2 \times b_1 & a_2 \times b_2 \end{pmatrix}$$

Entre tensores

$$A = \begin{pmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \end{pmatrix}$$

$$B = \begin{pmatrix} b_{1,1} & b_{1,2} \\ b_{2,1} & b_{2,2} \end{pmatrix}$$

$$C = A \otimes B$$

$$C = \begin{pmatrix} a_{1,1} \times \begin{pmatrix} b_{1,1}, b_{1,2} \\ b_{2,1}, b_{2,2} \end{pmatrix} & a_{1,2} \times \begin{pmatrix} b_{1,1}, b_{1,2} \\ b_{2,1}, b_{2,2} \end{pmatrix} \\ a_{2,1} \times \begin{pmatrix} b_{1,1}, b_{1,2} \\ b_{2,1}, b_{2,2} \end{pmatrix} & a_{2,2} \times \begin{pmatrix} b_{1,1}, b_{1,2} \\ b_{2,1}, b_{2,2} \end{pmatrix} \end{pmatrix}$$

$$C = \begin{pmatrix} a_{1,1} \times b_{1,1} & a_{1,1} \times b_{1,2} & a_{1,2} \times b_{1,1} & a_{1,2} \times b_{1,2} \\ a_{1,1} \times b_{2,1} & a_{1,1} \times b_{2,2} & a_{1,2} \times b_{2,1} & a_{1,2} \times b_{2,2} \\ a_{2,1} \times b_{1,1} & a_{2,1} \times b_{1,2} & a_{2,2} \times b_{1,1} & a_{2,2} \times b_{1,2} \\ a_{2,1} \times b_{2,1} & a_{2,1} \times b_{2,2} & a_{2,2} \times b_{2,1} & a_{2,2} \times b_{2,2} \end{pmatrix}$$

Tensores: aritmética

```
from numpy import array
from numpy import tensordot

A = array([1,2])

B = array([3,4])

C = tensordot(A, B, axes=0)
print(C)
```

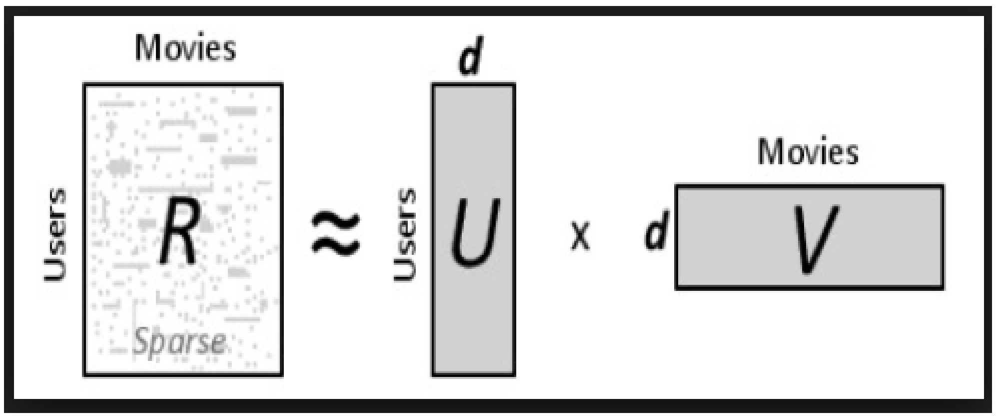
```
[[3 4]
 [6 8]]
```

3

Factorización

Descomposición de matrices

- Estas técnicas empezaron a destacar en un concurso de Netflix (2006)
  - Premio: 1 millón de dólares
  - Objetivo: mejorar el rendimiento de su recomendador en un 10%
  - Datos:
    - + 100 000 000 ratings
    - 480 000 usuarios
    - 17 770 películas



Descomposición de matrices

- Métodos que descomponen la matriz:
  - Partes más simples
  - **Objetivo**: agilizar el cálculo de operaciones matriciales más complejas

- Analogía: factorización de números

$$10 \rightarrow 2 \times 5$$

Descomposición de matrices

1. Descomposición LU
2. Descomposición QR
3. Descomposición Cholesky

Descomposición de matrices

- LU
  - Matrices cuadradas

$$A = L \, . \, U$$

- A = matriz cuadrada que queremos descomponer
- L = (Lower) Matriz triangular inferior
- U = (Upper) Matriz triangular superior

Descomposición de matrices

- QR
  - No limitado a matrices cuadradas

$$A = Q\ .R$$

  - A = matriz que queremos descomponer
  - Q = matriz m x m
  - R = Matriz triangular superior

Descomposición de matrices

- Cholesky
  - Matrices simétricas cuadradas
  - Todos los valores tienen que ser positivos

$$A = L \, . \, L^T$$

o

$$A = U \, . \, U^T$$

- A = matriz que queremos descomponer
- L = (Lower) Matriz triangular inferior
- U = (Upper) Matriz triangular superior

Eigendecomposition

- Eigendecomposition
  - Descompone una matriz en vectores y valores propios
  - Una de las técnicas de descomposición más usadas

- Objetivo:
  - Determinar el rol de los valores y vectores propios
  - Cómo calcular la descomposición en Python (Numpy)
  - Cómo confirmar que un vector es un vector propio
  - Reconstrucción de la matriz

Eigendecomposition

- Un vector es un vector propio si satisface la siguiente ecuación (eigenvalue equation):

$$A \cdot v = \lambda \cdot v$$

- A = matriz cuadrada
- v = vector propio de la matriz
- λ = valor propio (escalar)

Eigendecomposition

$$A \cdot v = \lambda \cdot v$$

$$A = Q \cdot \Lambda \cdot Q^T$$

- Q = matriz compuesta por vectores propios
- Λ= Matriz diagonal compuesta de valores propios

- Objetivo:
  - Simplificar los cálculos de otras operaciones con matrices
  - Calcular componentes principals de una matriz (PCA)

Eigendecomposition

- *Los vectores propios son aquellos vectores que no cambian su orientación cuando se multiplican por la matriz original, solo se escala por un valor que determina el valor propio*
  - *Ejemplos:*
    - Compresión de imágenes
    - PCA
    - Factorización para predicciones colaborativas (Netflix)

Eigendecomposition

Almacenamiento: 15%

Eigendecomposition

- Ejemplo **PCA**:

Eigenvalues = [1.5725 0.0378]

Unit Eigenvectors = [[0.7553   -0.6553],
[0.6554   0.7553]]

Significado del valor 1.57: la componente principal

(0.7554 0.6554) explica una varianza 57 % mayor que las variables originales

Matriz 2D Original

Proyección 1D

$$\begin{bmatrix} -0.12 & -0.55 \\ -0.65 & -0.46 \\ 1.67 & 1.61 \\ -0.38 & -0.52 \\ -0.80 & -0.25 \\ -0.71 & -0.45 \\ -0.53 & -0.66 \\ 1.82 & 1.41 \\ 0.07 & 0.23 \\ -0.37 & -0.36 \end{bmatrix} \begin{bmatrix} 0.7553 \\ 0.6553 \end{bmatrix} = \begin{bmatrix} -0.45 \\ -0.79 \\ 2.31 \\ -0.63 \\ -0.76 \\ -0.82 \\ -0.83 \\ 2.29 \\ 0.20 \\ -0.51 \end{bmatrix}$$

Vector propio

Eigendecomposition

- Cálculo en Python

```
from numpy import array
from numpy.linalg import eig

A = array([
        [1, 2, 3],
        [4, 5, 6],
        [7, 8, 9]
])

#factorizar
values,vectors = eig(A)
```

```
[[-0.23197069 -0.78583024  0.40824829]
 [-0.52532209 -0.08675134 -0.81649658]
 [-0.8186735   0.61232756  0.40824829]]
```

```
print(values)    => [ 1.61168440e+01 -1.11684397e+00 -9.75918483e-16]
print(vectors)   =>  [[-0.23197069 -0.78583024  0.40824829]
                      [-0.52532209 -0.08675134 -0.81649658]
                      [-0.8186735   0.61232756  0.40824829]]
```

Eigendecomposition

- Confirmación: es un vector propio? => $A \cdot v = \lambda \cdot v$

```
#confirmación
from numpy import array
from numpy.linalg import eig
# define matrix
A = array([
[1, 2, 3],
[4, 5, 6],
[7, 8, 9]])

# factorizar
values, vectors = eig(A)

# confirmar un vector propio (segundo en el vector)
v = vectors[:, 1]

B = A.dot(vectors[:, 1])
print(B) => [ 0.87764976  0.09688771 -0.68387434]

C = vectors[:, 1] * values[1]
print(C)  => [ 0.87764976  0.09688771 -0.68387434]
```

Eigendecomposition

- Reconstrucción de la matriz

$$A = Q \cdot \Lambda \cdot Q^T$$

```
values, vectors = eig(A)
Q = vectors
R = inv(Q)
# Creamos una matriz diagonal con los
valores propios
L = diag(values)

A = Q.dot(L).dot(R)

print(A) =>  [[1. 2. 3.]
              [4. 5. 6.]
              [7. 8. 9.]]
```

SVD

- Singular Value Decomposition
  - Todas las matrices tienen su SVD
    - Método más usado

"The Singular Value Decomposition is a highlight of linear algebra"

Page 371, Introduction to Linear Algebra, 2016.

SVD

1. ¿Qué es?
2. ¿Cómo calcular?
3. Calculo de la pseudoinversa y ejecución de la reducción de dimensionalidad

SVD

$$A = U \cdot \Sigma \cdot V^T$$

- A = matriz n x m que queremos descomponer
- U = matriz m x m
- Σ = matrix diagonal n x m
- V = matrix n x n

- Los valores diagonales de la matriz Σ son los valores singulalres de la matriz A

SVD

- Diferencias con las descomposición de vectores y valores propios:
  1. Se puede aplicar en matrices rectangulares
     - Vectores y valores propios solo en cuadradas
  2. Eigen values => singular values, Eigenvectors => singular vectors
  3. Si una matriz A (m x n) donde m > n, A siempre se puede describir mediante SVD
     - No todas las matrices se pueden descomponer en vectores y valores propios

SVD

- Aplicaciones
  - Cálculo de la inversa
  - Método de reducción de datos
  - Compresión de imagenes
  - Reducción de ruido
  - Regresión lineal de mínimos cuadrados
  - …

"Applying the SVD to a matrix is like looking inside it with X-ray vision"

Descomposición de matrices

Mondragon Unibertsitatea
Goi Eskola Politeknikoa

1. Buscad un paper (Google Scholar) de machine learning donde se utilice la operación (PPT o Jupyter notebook)
2. Escribid un resumen de vuestro método de descomposición de matrices (Jupyter notebook)
   - Resumen / objetivo para el que se usa la operación en ese caso concreto
   - Crear un ejemplo utilizando la operación con datos en un array

Mondragon Unibertsitatea

Goi Eskola Politeknikoa

Aitor Agirre

aaguirre@mondragon.edu