Técnicas de Inteligencia Artificial

Óscar García

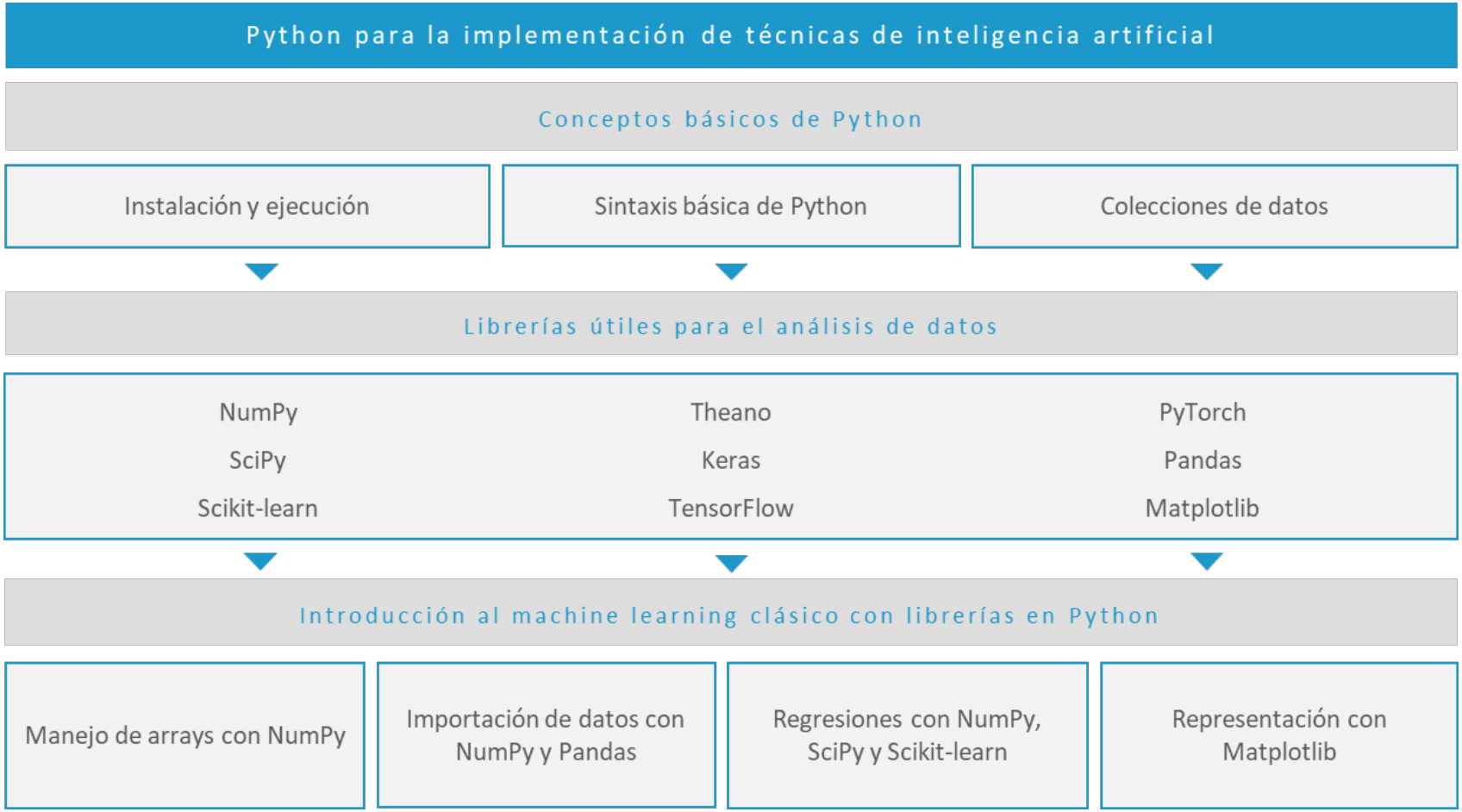
Tema 2. Python para la implementación de técnicas de inteligencia artificial

UNIVERSIDAD INTERNACIONAL DE LA RIOJA

La Universidad en Internet

Contenidos

Esquema


Python para la implementación de técnicas de inteligencia artificial
Conceptos básicos de Python
Instalación y ejecución
Sintaxis básica de Python
Colecciones de datos
Librerías útiles para el análisis de datos
NumPy
SciPy
Scikit-learn
Theano
Keras
TensorFlow
PyTorch
Pandas
Matplotlib
Introducción al machine learning clásico con librerías en Python
Manejo de arrays con NumPy
Importación de datos con NumPy y Pandas
Regresiones con NumPy, SciPy y Scikit-learn
Representación con Matplotlib

Objetivos

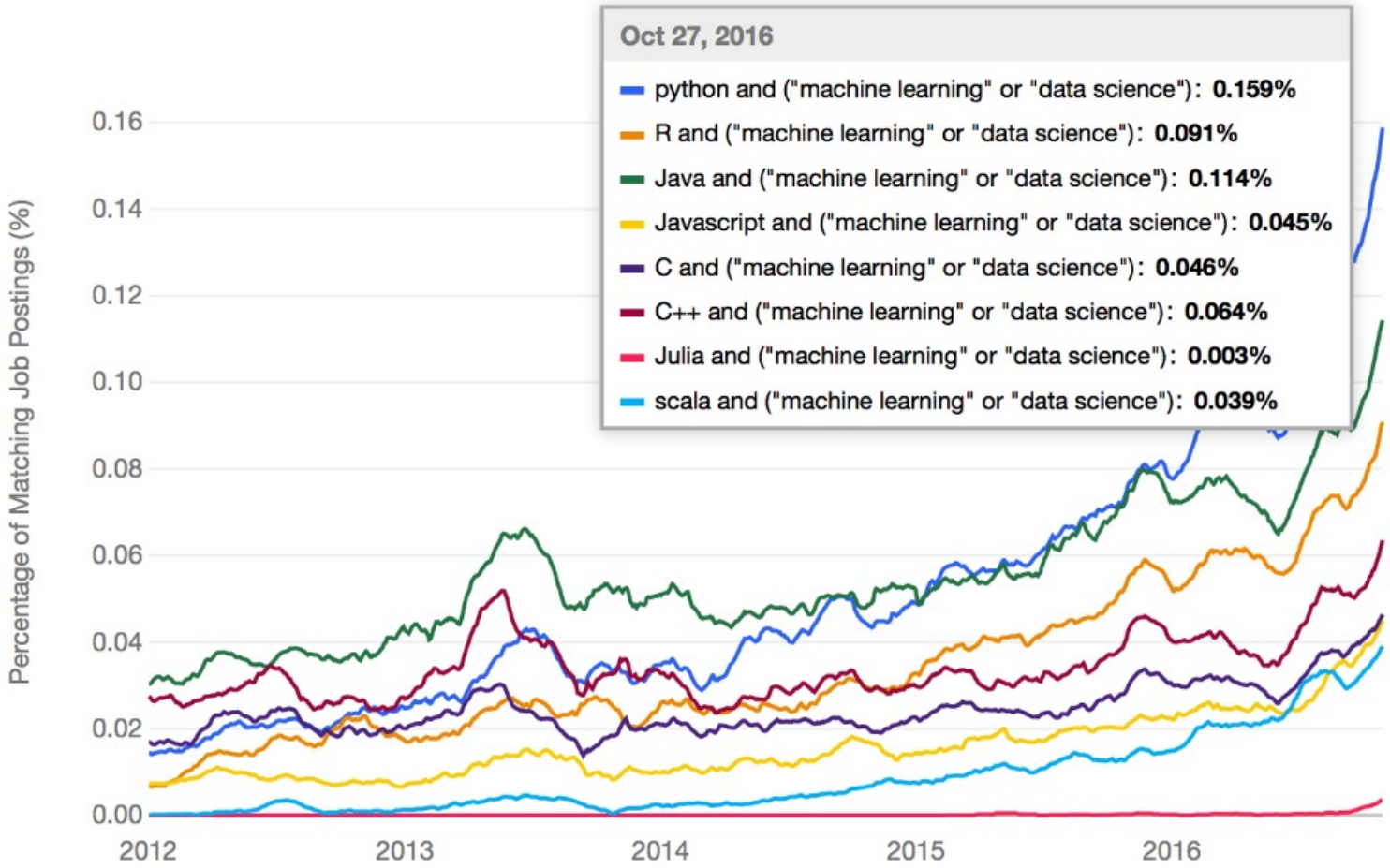
- Manejar la **sintaxis básica de Python** para emplearlo como lenguaje de análisis de datos y para aplicar técnicas ML
- Conocer y manejar las **colecciones de datos** incorporadas en Python para su uso en la aplicación de técnicas IA
- Conocer y recurrir a las **librerías más importantes sobre Python** para el manejo de datos, el ML clásico, las redes neuronales y el DL, así como la representación de datos
- **Importar datasets** desde fuentes locales o fuentes externas open data empleando librerías sobre Python apropiadas
- Aplicar **algoritmos básicos de regresión** empleando Python y las librerías apropiadas

Introducción

- Creado por Guido Van Rossum en 1991
- El lenguaje más empleado en IA y ML (Bansal, 2019).
  - **Java**: seguridad debido a bytecode / sandboxes
  - **R**: gráficos, estadísticas, análisis y visualización ML
  - **JavaScript / Node.js** (JavaScript del lado del servidor)
  - **Scala**: forma parte del core de Apache Spark
  - **C++**: soportado por diferentes API, popular como lenguaje de propósito general durante muchos años
  - **Matlab**: tratamiento matemático, versátil con arrays

Introducción

- Ofertas de empleo en ML / data science en el portal Indeed (2016)



Fuente: Puget, J.F. (2016)

Introducción

- Python es líder (57% de los científicos de datos lo prefieren) (Bansal, 2019)
- TensorFlow lanzado por Google en 2015
  - Posteriormente lanzado para otros lenguajes
- Multitud de bibliotecas
  - Pandas para la importación de datos
  - Numpy para resolver diferentes cálculos
  - Theano, Keras, scikit-learn, etc.
  - IA, ML, DL, NLP
- Sintaxis muy simple, algoritmos fáciles de implementar

Conceptos básicos

- Diferentes plataformas: Win, Mac, Linux, Raspberry Pi
- Licencia código abierto compatible GNU
- **Lenguaje interpretado**
  - Lenguajes interpretados: JavaScript, PHP, Ruby
  - Lenguajes compilados: C, C++, Pascal
- Por defecto el intérprete comprueba errores en sintaxis antes de ejecutarlo
  - El motor V8 intérprete de JavaScript no lo hace, p.ej.

Conceptos básicos

- **Lenguaje multiparadigma**, soporta al mismo tiempo:
  - Programación orientada a objetos
    - *C++, Java, C#, R, etc.*
  - Programación imperativa
    - *C, C++, Java, C#, R, etc.*
    - *Programación declarativa: HTML, SQL, XML*
    - En concreto: procedural (subrutinas)
  - Programación funcional
    - Basada en cálculo lambda: subrutinas inspiradas en funciones matemáticas que no modifican el estado del programa

Conceptos básicos

- **Fuertemente tipado**
  - No se permiten violaciones de los tipos de datos en las variables
  - El programador ha de cambiar entre un tipo y otro de datos mediante conversión explícita (casting)
  - *Fuertemente tipados: C/C++, Java, C#, Go, TypeScript*
- **Tipado dinámico**
  - Permite a una variable tomar un tipo u otro, mediante la conversión apropiada
  - *Tipado dinámico: JavaScript*
  - *Tipado estático: C, C++*

Instalación

- Comprobar si está ya instalado (ver 3.8 desde 2019)

```
PS C:\Users\xxx> python --version
Python 3.8.0
```

- Instalación (incluye gestor de paquetes PIP)
  - https://www.python.org/
- Instalación en Linux:

```
sudo apt-get update
sudo apt-get install python
sudo apt-get install pip
```

Instalación

- Otra opción tanto en Windows como en Linux:
  - **Conda o Miniconda**
  - https://docs.conda.io/
  - Incluyen entornos, intérpretes, gestión de paquetes
- Interesante también contar con un IDE
  - **Atom**: https://atom.io/
  - **Visual Studio Code**: https://code.visualstudio.com/
  - Incluyen entornos, intérpretes, gestión de extensiones: *linters*, resaltado de sintaxis, etc.

Intérprete de instrucciones

- Los archivos tienen extensión **.py** (o variantes, **.ipynb**)
- También podemos lanzar el intérprete en línea:

```
PS C:\Users\xxx > python
Python 3.8.0 (tags/v3.8.0:fa919fd, Oct 14 2019,
19:37:50) [MSC v.1916 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for
more information.
>>> print("¡Bienvenidos al tema 2 de la asignatura!")
¡Bienvenidos al tema 2 de la asignatura!
>>>
```

- Si trabajamos con **notebooks** en línea como **Jupyter** o los provistos por **Kaggle**, también podemos trabajar como si fuera el intérprete de instrucciones

Indentación y comentarios

A diferencia de otros lenguajes, en los cuales no es importante y sólo se hace por legibilidad del código, en Python es importante el sangrado (*indentación*) a la hora de definir los distintos bloques de ejecución.

```
"""
Ejemplo de programa básico en Python
con el fin de ver indentación y formas de comentarios
"""

if 5 > 2:
    print("5 es mayor que 2")
else:
    # nunca deberíamos estar en este punto del programa
    pass # esta sentencia no hace nada
```

Variables, tipos y casting

En Python no es necesario declarar una variable (a diferencia de lenguajes de tipado estático como C/C++), ésta se crea en el momento de utilizarla por primera vez, momento en el cual se define su tipo, que además puede cambiar con el tiempo con una nueva asignación de la variable a un tipo nuevo

```
x = 5
# ahora x es un número
print(x)                  # 5
print(type(x))            # <class 'int'>
print(type(x).__name__) # int

x = "¡hola!"
# ahora x es una cadena
print(x)                  # "¡hola'""
print(type(x))            # <class 'str'>
print(type(x).__name__) # str
```

Variables, tipos y casting

```
x, y, z = 1, 2, 3
a, b, c = "naranja", "fresa", "platano"
d = e = f = 5
suma = x + y
```

Podemos asignar el mismo o distinto valor a varias variables en una única sentencia, pero esto no las asocia entre sí de ninguna forma.

```
#print("La suma de " + x + " + " + y + " es " + suma)
print(suma) # 3
print(x + y) # 5
print(a + " " + b) # "naranja fresa"
```

Podemos usar la función `print()` para imprimir por consola literales y variables, pero no podemos concatenar cadenas y números como se hace en otros lenguajes (como JavaScript)

Variables globales y locales

Por defecto una variable definida fuera de una función es global, y cuando es definida dentro de una función es local, a no ser que utilicemos la palabra clave **global** para modificar este comportamiento.

```
x = 5 # variable global
def miFuncion1(a):
    b = 1  # b es local
    return a + b
def miFuncion2(a):
    x = 2   # esta x es local y diferente a x global
    return a + x
def miFuncion3(a):
    global y # y es global
    y = 1
    return a + y
def miFuncion4(a):
    global x # para modificar el valor de la x global
    x = 0
    return a + x
print(miFuncion1(1))    # 4
print(miFuncion2(2))    # 4
print(miFuncion3(3))    # 1
print(y)                # 1
print(miFuncion4(4))    # 4
print(x)                # 0
```

Una variable global puede ser empleada desde cualquier punto del programa, pero una variable local sólo puede ser empleada dentro de la función que la define.

Tipos de datos

Tipos	Python
Texto	str
Numéricos	int, float, complex
Secuencia	list, tuple, range
Mapas	dict
Conjuntos	set, frozenset
Booleanos	bool
Binarios (no nos interesan, pero existen)	bytes, bytearray, memoryview

Tipos de colecciones

Tipo en Python	Descripción	Ejemplos
list	Secuencia ordenada de objetos	[1, 2, 2, “plátano”, 1j, True] [0.5, 3.2, 3.2]
tuple	Secuencia ordenada de objetos no modificables (inmutables)	(1, 2, 2, “plátano”, 1j, True) (0.5, 3.2, 3.2)
set	Colección de objetos únicos sin orden	{1, 2, 3, “plátano”, 1j, True} {“plátano”, “fresa”}
dict	Pares clave-valor no ordenados	{“nombre” : “Pedro”, “edad” : 39}

Tipado dinámico

```
x = "Hola"                  # str
x = 5                       # int
x = 2.5                     # float
x = 2 + 1j                  # complex
x = ["perro", "gato"]       # list
x = ("perro", "gato")       # tuple
x = range(5)                # range(0,5)
x = {"nombre" : "Pedro", "edad" : 39} # dict
x = {"perro", "gato"}       # set
x = True                    # bool
```

El tipo del dato se asigna automáticamente en el momento de asignar un valor a una variable

Especificar tipo con constructor

```
x = str("Hola")
x = int(5)
x = float(2.5)
x = complex(2 + 1j)
x = list(("perro", "gato"))
x = tuple(("perro", "gato"))
x = range(5)
x = dict(nombre="Pedro", edad=39)
x = set(("perro", "gato"))
x = bool(3)    # True
```

En el caso de los diccionarios, se emplea la notación de invocación a la función especificando el valor de cada parámetro específico

Casting

```
a = 2        # int
b = 2.5      # float
c = 12e3     # float
d = -12.5E5     # float
e = 2j       # complex
f = complex(c)
g = int(2.5)     # 2
h = float(1)     # 1.0
# i = float(2j) # no se puede castear desde complex
a = int("3")     # 3
b = str(2)       # "2"
c = str(3.0)     # "3.0"
x = 5


print(isinstance(x, int))
#> True
```

Cuando trabajemos con números podemos hacerlo con enteros, flotantes o complejos, y la conversión (*casting*) entre unos y otros puede hacerse de forma explícita mediante el uso de métodos constructor

Cadenas

```
x = "Bienvenidos a la asignatura IA"
# arrays comienzan en 0
print(x[1])              #> "i"
# rango [2, 6), siempre el derecho exclusive
print(x[2:6])            #> "enve"
# rango [-6, -4), siempre el derecho exclusive
# contando desde el final del array
# siendo el último elemento el -1
print(x[-6:-4])          #> "ur"
```

Las cadenas funcionan como el resto de los arrays en cuanto a que son iterables y cada elemento es accesible mediante el uso de corchetes

Cada elemento es un carácter Unicode, pero no existe un tipo *char* como en otros lenguajes. Un carácter es una cadena de tamaño 1

Cadenas

```
x = "  Bienvenidos a la asignatura IA  "
print(x.strip())
# "Bienvenidos a la asignatura IA"
print(x.lower())
# "  bienvenidos a la asignatura ia  "
print(x.upper())
# "  BIENVENIDOS A LA ASIGNATURA IA  "
print(x.replace("B", "X"))
# "  Xienvenidos a la asignatura IA"
print(x.split(" ")) # elegimos separador como argumento
# ['', '', 'Bienvenidos', 'a', 'la', 'asignatura', 'IA', '', '']
print("IA" in x)
# True
print("IA" not in x)
# False
```

Algunos métodos para manipular cadenas que en algunos casos son similares en otras colecciones.

Cadenas

Es interesante aprender a formatear la salida de las cadenas para su uso en programas interactivos que creemos y ver los resultados de nuestro análisis de datos.

```
materia = "IA"
alumnos = 150
mensaje = "En la asignatura " + materia
mensaje += " hay " + str(alumnos) + " alumnos"

print(mensaje)
#> En la asignatura IA hay 150 alumnos

print("En la asignatura {} hay {} alumnos".format(materia, alumnos))
#> En la asignatura IA hay 150 alumnos

print("Hay {1} alumnos en la asignatura {0}".format(materia, alumnos))
#> Hay 150 alumnos en la asignatura IA
```

Expresiones booleanas

```
x = 5
y = 4
z = None    # de tipo especial NoneType
print(x > y)
#> True
# las siguientes expresiones son True
print(bool("Hola"))
print(bool(3))
print(bool(["perro", "gato"]))
# las siguientes expresiones son False
print(bool(z))
print(bool([]))
print(bool(0.0))
print(bool({}))
```

Los `bool` pueden tomar valores `True` o `False` y nos sirven para evaluar expresiones lógicas en el programa con el fin de controlar el flujo del programa, entre otras posibilidades

Cualquier elemento diferente de cero y colecciones no vacías castearán a `True`, mientras que números igual a cero o colecciones vacías castearán a `False`

Operadores aritméticos

Operador	Nombre	Ejemplos
+	Adición	x + y
-	Sustracción	x - y
*	Multiplicación	x * y
/	División	x / y
%	Módulo (resto)	x % y
**	Exponenciación	x ** y
//	División entera	x // y

Operadores de asignación

Operador	Ejemplo	Equivalencia
=	x = y	x = y
+=	x += y	x = x + y
-=	x –= y	x = x – y
*=	x *= y	x = x * y
/=	x /= y	x = x / y
%=	x %= y	x = x % y
**=	x **= y	x = x ** y
//=	x //= y	x = x // y

Operadores de comparación

Operador	Nombre	Ejemplos
==	Igual	x == y
!=	No igual	x != y
>	Mayor que	x > y
<	Menor que	x < y
>=	Mayor o igual que	x >= y
<=	Menor o igual que	x <= y

Operadores lógicos

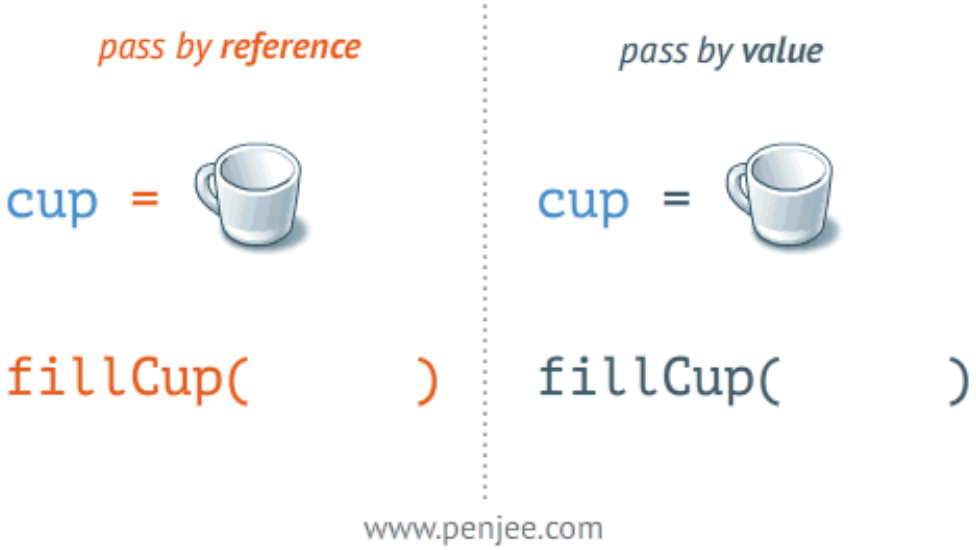
Operador	Nombre	Ejemplos
and	Producto lógico	x > 2 and x < 10
or	Suma lógica	x < 5 or x > 15
not	Negación lógica	not (x > 2 and x < 10)

Operadores de identidad

Operador	Descripción	Ejemplos
is	Devuelve True si ambas variables son el mismo objeto	x is y
is not	Devuelve True si las variables no son el mismo objeto	x is not y

Operadores de identidad

```
x = ["perro", "gato"]
y = ["perro", "gato"]
z = x

print(x == y)   # True
print(y is x)   # False
print(z == x)   # True
print(z is x)   # True

z[0] = "ratón"
print(x)    # ['ratón', 'gato']
```



Fuente de la animación: https://blog.penjee.com/wp-content/uploads/2015/02/pass-by-reference-vs-pass-by-value-animation.gif

Operadores de membresía

Operador	Descripción	Ejemplos
in	Devuelve True si una secuencia con el valor especificado x se encuentra en el objeto y	x in y
not in	Devuelve True si ninguna secuencia con el valor especificado x se encuentra en el objeto y	x not in y

Colecciones de datos:
Listas, tuplas, conjuntos y diccionarios

Tipo en Python	Descripción	Ejemplos
list	Secuencia ordenada de objetos	[1, 2, 2, “plátano”, 1j, True] [0.5, 3.2, 3.2]
tuple	Secuencia ordenada de objetos no modificables (inmutables)	(1, 2, 2, “plátano”, 1j, True) (0.5, 3.2, 3.2)
set	Colección de objetos únicos sin orden	{1, 2, 3, “plátano”, 1j, True} {“plátano”, “fresa”}
dict	Pares clave-valor no ordenados	{“nombre” : “Pedro”, “edad” : 39}

Listas

- Las listas son **colecciones de objetos ordenadas y modificables**
- **Se escriben utilizando corchetes con los elementos separados por comas**
- **Podemos acceder a los diferentes elementos empleando el operador corchete** y acceder a un rango de elementos al igual que hacíamos con las cadenas
- Asimismo, podemos utilizar la función `len()` y el método `append()` para añadir nuevos elementos

Listas

```
x = ["perro", "gato", "ratón"]
print(len(x))
#> 3
print(x)
#> ['perro', 'gato', 'ratón']
x[1] = "gatete"
print(x)
#> ['perro', 'gatete', 'ratón']
print(x[1:])
#> ['gatete', 'ratón']
print(x[-2:-1])
#> ['gatete']
```

Listas

```
x = ["perro", "gato", "ratón"]

for e in x:
    print(e)

if "perro" in x:
    print("Hay un perro en la lista")
```

Listas

```
x = ["perro", "gato", "ratón"]
print(x)
#> ['perro', 'gato', 'ratón']
x.append("loro")
print(x)
#> ['perro', 'gato', 'ratón', 'loro']
x.remove("gato") # elimina el primero que encuentre
print(x)
#> ['perro', 'ratón', 'loro']
x.pop() # elimina el último elemento (o el índice que espefiquemos)
print(x)
#> ['perro', 'ratón']
del x[0]
print(x)
#> ['ratón']
del x   # elimina la lista entera
#print(x)   # x ya no está definido
```

Listas

- Es importante recordar que **el operador asignación no duplica objetos**, de modo que tenemos que clonar su contenido (duplicar los datos en otro espacio de la memoria) si no queremos alterar los datos originales al realizar operaciones
- **Esto sirve para cualquier colección de datos**

Listas

```
x = ["perro", "gato", "ratón"]
y = x.copy()    # método copia
z = list(x)     # otra opción: con el constructor

w = x + y       # concatenamos dos listas
print(w)
#> ['perro', 'gato', 'ratón', 'perro', 'gato', 'ratón']
```

Tuplas

- Las tuplas son **colecciones ordenadas pero inmutables, es decir, no podemos alterar su valor**
- Las tuplas funcionan de forma muy similar a las listas a la hora de acceder a sus elementos, iterarlas, comprobar si un elemento se encuentra en ellas, etc., pero **no podemos añadir o eliminar elementos o modificar el valor de uno de sus elementos**
- Para hacer esto, tendríamos que copiar su valor en una lista, realizar las modificaciones y asignar su contenido completo a la tupla original

Conjuntos

- Los conjuntos (`set`) son **colecciones de datos no ordenadas y que no permiten elementos duplicados**
- Es decir, no podemos predecir en qué orden se mostrarán (`print`) o iterarán sus elementos (`for … in`)
- **De hecho, cada vez que realicemos una ejecución de este tipo el orden puede ser distinto**
- **Tampoco podemos utilizar el operador corchete** para acceder a un elemento determinado, puesto que no tienen orden alguno
- **No pueden contener dos elementos con el mismo valor**

Conjuntos

```
x = {"perro", "gato"}
print(x)
#> {'gato', 'perro'} # u otro orden diferente
print("loro" in x)
#> False
x.add("loro")
print(x)
#> {'perro', 'loro', 'gato'} # u otro orden
x.remove("perro")
print(x)
#> {'loro', 'gato'} # u otro orden
x.discard("perro")  # no produce error si no existe
print(x)
#> {'loro', 'gato'} # u otro orden
x.update(("mosca", "pato"))
print(x)
#> {'loro', 'pato', 'mosca', 'gato'} # u otro orden
```

Diccionarios

- Los diccionarios son el equivalente a mapas o hashes en otros lenguajes de programación
- **Son un conjunto de elementos clave-valor sin orden y en los cuales se puede modificar el contenido de los elementos**
- Son también similares a los objetos nativos de JavaScript que pueden ser convertidos a y desde JSON
- **Sin embargo, son mucho más flexibles y una clave no tiene por qué ser sólo de tipo cadena, sino que puede ser cualquier objeto, incluso un entero, un flotante o un complejo, por ejemplo**

Diccionarios

```
x = {
    "nombre": "Pedro",
    "apellido": "García",
    "edad": 39
}
print(x)
#> {'nombre': 'Pedro', 'apellido': 'García', 'edad': 39}
y = x["nombre"]
print(y)
#> Pedro
z = x.get("nombre")
print(z)
#> Pedro
x["nombre"] = "Perico"
print(x)
#> {'nombre': 'Perico', 'apellido': 'García', 'edad': 39}
```

Diccionarios: acceso a elementos

```
x = {
    "nombre": "Pedro",
    "apellido": "García",
    "edad": 39
}

for i in x:
    print(x[i]) # valor

for i in x.values():
    print(i)    # valor

for i, j in x.items():
    print(i, j) # clave valor
```

Diccionarios: varias formas de iterar

```
x = {
    "nombre": "Pedro",
    "apellido": "García",
    "edad": 39
}

print("apellido" in x)
#> True
print("Pedro" in x)
#> False
print(len(x))
#> 3
```

Diccionarios: añadir y eliminar elementos

```
x = {
    "nombre": "Pedro",
    "apellido": "García",
    "edad": 39
}
x["país"] = "España"
print(x)
#> {'nombre': 'Pedro', 'apellido': 'García', 'edad': 39, 'país': 'España'}
x.pop("edad")
print(x)
#> {'nombre': 'Pedro', 'apellido': 'García', 'país': 'España'}
del x["apellido"]
print(x)
#> {'nombre': 'Pedro', 'país': 'España'}
del x
#print(x) # x ya no existe
```

Diccionarios: anidamiento

```
x = {
    "nombre": "Pedro",
    "apellido": "García",
    "edad": 39,
    "mascotas" : ["perro", "gato"],
    "cónyuge": {
        "nombre": "Rosa",
        "apellido": "Pérez",
        "edad": 37
         }
    }
```

Instalación de paquetes en Python (PIP)

- Lista de paquetes disponibles: https://pypi.org/
- *También es posible utilizar un gestor como Conda*
- **Instalación:**

```
PS C:\Users\xxx> pip install numpy
```

- Si no tenemos permisos de administrador:

```
PS C:\Users\xxx> pip install numpy --user
```

- **Desinstalar** un paquete:

```
PS C:\Users\xxx> pip uninstall numpy
```

- **Ver la lista** disponible de paquetes mediante:

```
PS C:\Users\xxx> pip list
```

Utilizar un paquete en nuestro programa

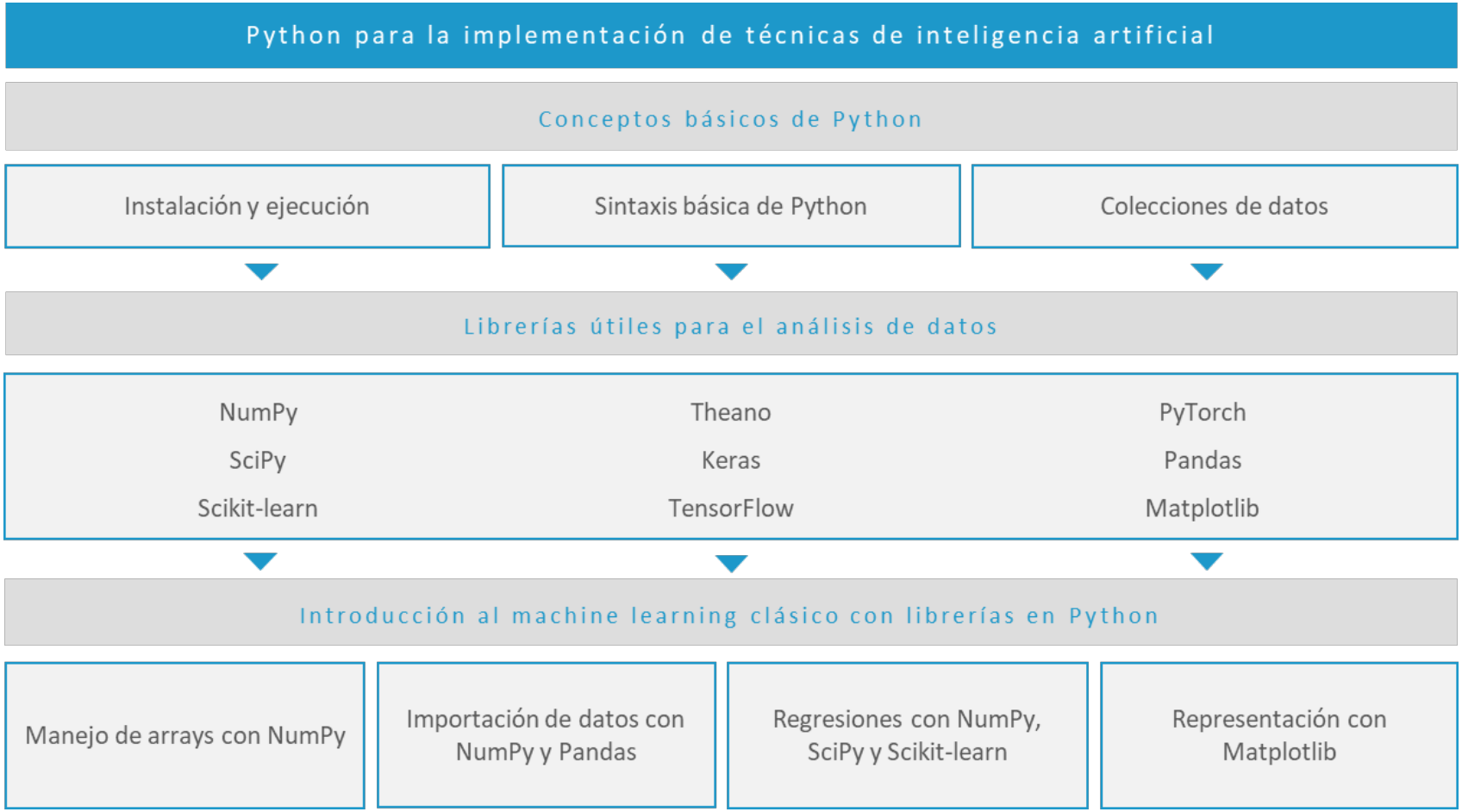
```
import numpy


# otra alternativa si queremos usar alias...
import numpy as np
```

Importación de datos

- Importación de archivos en formato .txt
- Importación de archivos planos CSV mediante la librería estándar de Python
- Importación de archivos planos CSV mediante NumPy
- Importación de archivos planos CSV mediante Pandas
- Importación desde una URL
- Importación desde otras librerías (toy datasets)

Librerías más utilizadas en Machine Learning

Resumen


Python para la implementación de técnicas de inteligencia artificial
Conceptos básicos de Python
Instalación y ejecución
Sintaxis básica de Python
Colecciones de datos
Librerías útiles para el análisis de datos
NumPy
SciPy
Scikit-learn
Theano
Keras
TensorFlow
PyTorch
Pandas
Matplotlib
Introducción al machine learning clásico con librerías en Python
Manejo de arrays con NumPy
Importación de datos con NumPy y Pandas
Regresiones con NumPy, SciPy y Scikit-learn
Representación con Matplotlib

Gracias por vuestra atención
¿Dudas?

Imagen por Peggy und Marco Lachmann-Anke
Licencia: Creative Commons Zero

Anexos

NumPy

- https://numpy.org/
- Procesamiento de matrices
- Matrices muldimensionales
- Funciones matemáticas de alto nivel
- Cálculos científicos fundamentales de Machine Learning
- Álgebra lineal
- Transformadas de Fourier
- Números aleatorios
- Otras librerías emplean NumPy por debajo

NumPy


```
# Ejemplo de programa usando NumPy
# para realizar operaciones matemáticas básicas
import numpy as np

# Creación de dos tensores de rango 2 (matrices de 2x2)
x = np.array([[1, 2], [3, 4]])
y = np.array([[5, 6], [7, 8]])

# Creación de dos tensores de rango 1 (vectores de dimensión 2
)
v = np.array([9, 10])
w = np.array([11, 12])
```

NumPy


```
# Producto escalar de vectores
print(np.dot(v, w), "\n")

# Producto de matriz por un vector
print(np.dot(x, v), "\n")

# Producto de matrices
print(np.dot(x, y))
```

SciPy

- https://www.scipy.org/
- Popular entre ingenieros ML
- Optimización
- Álgebra lineal
- Integración
- Estadística
- Manipulación de imágenes
- La librería SciPy forma parte de la pila de SciPy (ScpiPy stack)

SciPy

- En el siguiente ejemplo podemos ver cómo **manipular una imagen** utilizando diferentes librerías con SciPy

- Podemos **elegir una imagen cualquiera en JPEG** en nuestro ordenador y ver el resultado de las operaciones descritas

- **Nota:** para que el siguiente ejemplo funcione es necesario instalar `numpy`, como se ha indicado antes, así como las librerías:
  - `scipy, imageio, Pillow, scikit-image`

SciPy


```
# Manipulación de imágenes mediante SciPy
import imageio
import skimage
from skimage import transform

# Lectura de una imagen JPEG a un array numpy
img = imageio.imread("C:/Users/xxx/gato.jpg")
print(img.dtype, img.shape)

# Tintamos la imagen
img_tint = img * [1, 0.45, 0.3]
```

SciPy


```
# Guardamos la imagen tintada
imageio.imsave("C:/Users/xxx/gato_tintado.jpg", img_tint)

# Redimensionamos la imagen tintada
img_tint_resize = skimage.transform.resize(img_tint, (300, 300
))

# Guardamos la imagen resultante
imageio.imsave("C:/Users/xxx/gato_tintado_redimensionado.jpg",
 img_tint_resize)
```

SciPy


```
# Guardamos la imagen tintada
imageio.imsave("C:/Users/xxx/gato_tintado.jpg", img_tint)

# Redimensionamos la imagen tintada
img_tint_resize = skimage.transform.resize(img_tint, (300, 300
))

# Guardamos la imagen resultante
imageio.imsave("C:/Users/xxx/gato_tintado_redimensionado.jpg",
 img_tint_resize)
```

SciPy

Fuente: https://www.geeksforgeeks.org/best-python-libraries-for-machine-learning/

Scikit-learn

- https://scikit-learn.org/
- Una de las más populares de Machine Learning
- Construida sobre NumPy y SciPy
- Aprendizaje supervisado
- Aprendizaje no supervisado
- Minería de datos
- Análisis de datos

Scikit-learn

- En el siguiente ejemplo se importa uno de los dataset más conocidos en el aprendizaje de machine learning
- **El dataset "iris",** con dimensiones de pétalos y sépalos de flores) y se utiliza un clasificador de árbol de decisión
  - *Algoritmo CART, que veremos en el Tema 3*
- Este es un buen ejemplo, además, de importación tanto de datos como de modelos
- **Nota:** es necesario instalar el paquete skilearn (o scikit-learn) para su funcionamiento

Scikit-learn


```
# Python script using Scikit-learn
# for Decision Tree Clasifier

# Ejemplo con Scikit-learn
# utilizando un árbol de decisión
from sklearn import datasets
from sklearn import metrics
from sklearn.tree import DecisionTreeClassifier

# cargamos el dataset "iris"
dataset = datasets.load_iris()
```

Scikit-learn


```
# ajustamos los datos mediante un modelo CART
model = DecisionTreeClassifier()
model.fit(dataset.data, dataset.target)
print(model)

# realizamos predicciones
expected = dataset.target
predicted = model.predict(dataset.data)

# resumen del ajuste (fit) del modelo
print(metrics.classification_report(expected, predicted))
print(metrics.confusion_matrix(expected, predicted))
```

Scikit-learn

```
DecisionTreeClassifier(
ccp_alpha=0.0,
class_weight=None,
criterion='gini',
max_depth=None,
max_features=None,
max_leaf_nodes=None, min_impurity_decrease=0.0,
min_impurity_split=None, min_samples_leaf=1,
min_samples_split=2,
min_weight_fraction_leaf=0.0,
presort='deprecated',
random_state=None,
splitter='best')
```

Scikit-learn


```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        50
           1       1.00      1.00      1.00        50
           2       1.00      1.00      1.00        50

    accuracy                           1.00       150
   macro avg       1.00      1.00      1.00       150
weighted avg       1.00      1.00      1.00       150

[[50  0  0]
 [ 0 50  0]
 [ 0  0 50]]
```

Theano

- http://deeplearning.net/software/theano/
- El ML son básicamente matemáticas y estadística
- Theano permite definir, evaluar y optimizar expresiones matemáticas que implican conjuntos multidimensionales
- Optimiza el uso de CPU y GPU
- Uso en pruebas unitarias y autoverificación
- Muy potente
- Usada en proyectos científicos a gran escala
- Uso sencillo y accesible para proyectos pequeños

Theano

```
import theano
import theano.tensor as T
x = T.dmatrix('x')
s = 1 / (1 + T.exp(-x))
logistic = theano.function([x], s)
y = logistic([[0, 1], [-1, -2]])
print(y)
```

```
[[0.5        0.73105858]
 [0.26894142 0.11920292]]
```

Keras

- https://keras.io/
- Keras es una de las librerías de Machine Learning más populares para Python
- Redes neuronales de alto nivel capaz de funcionar sobre TensorFlow, CNTK, o Theano
- Puede funcionar sin problemas en la CPU y la GPU
- Keras permite que sea realmente sencillo construir y diseñar redes neuronales para los principiantes en el ML
- Una de las mejores características de Keras es que permite la creación de prototipos de forma fácil y rápida

TensorFlow

- https://www.tensorflow.org/
- Librería de código abierto muy popular para el cálculo numérico de alto rendimiento
- Desarrollada por el equipo de Google Brain en Google
- Definición y ejecución de cálculos que implican tensores
- *Un **tensor** es cierta clase de entidad algebraica que generaliza los conceptos de escalar, vector y matriz de una forma que sea independiente de cualquier sistema de coordenadas elegido*
- Entrenamiento y ejecución de redes neuronales profundas
- Ampliamente utilizado en el campo de la investigación y aplicación del machine learning

TensorFlow

- 2015: Lanzado por Google como código abierto, basado en Python
- 2016: Google lanza las TPU
  - ASIC específico para operar con tensores
  - Aritmética de 8 bit
  - Disponibles en el Cloud
- 2018: Tensorflow.js para JavaScript / Node.js
- 2018: Google Edge TPU
- 2019: TensorFlow 2.0: C++, Haswell, Java, Go y Rust
- También bibliotecas de terceros para C#, R y Scala

TensorFlow

- **Ejemplo de reconocimiento de dígitos escritos a mano**
- MNIST es una gran base de datos de dígitos escritos a mano que se usa comúnmente para entrenar sistemas de procesamiento de imágenes
  - De la web de Lecun (Yann Lecun, h=122, es, junto a Geoffrey Hinton y Yoshua Bengio, los tres *Padrinos de la Inteligencia Artificial* o *Padrinos del Machine Learning*).
- **Nota:** es necesario instalar el paquete `tensorflow` o `tensorflow-gpu` para el funcionamiento del siguiente ejemplo (instalará más de 400MB en el ordenador, incluyendo dependencias, como parte de Keras)
- Además, bajo Windows es necesario contar con la última versión de Visual C++ Redistributable para su funcionamiento:
  - https://support.microsoft.com/en-us/help/2977003/the-latest-supported-visual-c-downloads

TensorFlow


```
# Operaciones con tensores (arrays)
# Descarga e instala el paquete TensorFlow 2.0 version. Importa Tensor
Flow en tu programa:
from __future__ import absolute_import, division, print_function, unic
ode_literals
# Instala TensorFlow
import tensorflow as tf
# Carga y prepara el conjunto de datos MNIST. Convierte los ejemplos d
e numeros enteros a numeros de punto flotante:
mnist = tf.keras.datasets.mnist
(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0
```

TensorFlow


```
# Construye un modelo tf.keras.Sequential apilando capas. Escoge un op
timizador y una función de perdida para el entrenamiento de tu modelo:
model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(input_shape=(28, 28)),
  tf.keras.layers.Dense(128, activation='relu'),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(10, activation='softmax')
])
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

TensorFlow


```
# Entrena y evalua el modelo:
model.fit(x_train, y_train, epochs=5)
model.evaluate(x_test,  y_test, verbose=2)
```

```
Downloading data from
https://storage.googleapis.com/tensorflow/tf-keras-
datasets/mnist.npz
11493376/11490434 [==============================] - 1s 0us/step
```

TensorFlow

```
Train on 60000 samples
Epoch 1/5
60000/60000 [==============================] - 4s 65us/sample - loss: 0.2982 -
accuracy: 0.9128
Epoch 2/5
60000/60000 [==============================] - 3s 56us/sample - loss: 0.1424 -
accuracy: 0.9571
Epoch 3/5
60000/60000 [==============================] - 3s 57us/sample - loss: 0.1073 -
accuracy: 0.9677
Epoch 4/5
60000/60000 [==============================] - 3s 57us/sample - loss: 0.0891 -
accuracy: 0.9714
Epoch 5/5
60000/60000 [==============================] - 4s 59us/sample - loss: 0.0745 -
accuracy: 0.9768
10000/10000 - 1s - loss: 0.0760 - accuracy: 0.9781
```

[0.0760388328890549, 0.9781]

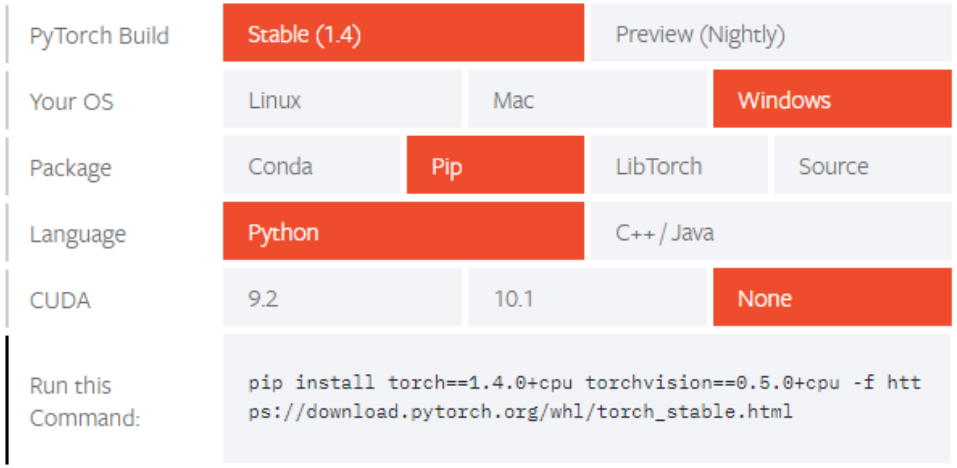
PyTorch

- https://pytorch.org/
- Una de las librerías más populares de ML de código abierto para Python
- Basada en Torch, que es una biblioteca de ML de código abierto que se implementa en C con un *wrapper* en Lua
- Tiene una **extensa selección de herramientas** y librerías que incluye **visión artificial (*Computer Vision*)** o **Procesamiento de Lenguaje Natural (NLP)**, entre otros
- Permite a los desarrolladores realizar cálculos tensoriales con aceleración en la GPU y también es de ayuda en la creación de gráficos computacionales

PyTorch

- Nota: es necesario instalar el paquete torch para el funcionamiento del siguiente ejemplo. Sin embargo, conviene hacerlo de la siguiente forma explicada en la web incluso aunque se emplee PIP o Conda. De esta forma se utilizará PIP o Conda, pero descargando el paquete desde la web:
  - https://pytorch.org/get-started/locally/

- Ejemplo bajo Windows y sin CUDA
- ***Compute Unified Device Architecture*** – Arquitectura Unificada de Dispositivos de Cómputo de NVIDIA



PyTorch

PyTorch

```
# Ejemplo para definir tensores que ajusten a una red neuronal
 de dos capas
# a datos aleatorios y calcule la pérdida
import torch
dtype = torch.float
device = torch.device("cpu")        # Para su uso en la CPU
# device = torch.device("cuda:0")   # Para su uso en la GPU
# N = batch size;
# D_in = dimensión capa de entrada
# H = dimensión capa oculta
# D_out = dimensión capa de salida
N, D_in, H, D_out = 64, 1000, 100, 10
```

PyTorch


```
# Creamos datos aleatorios de entrada y de salida
x = torch.randn(N, D_in, device = device, dtype = dtype)
y = torch.randn(N, D_out, device = device, dtype = dtype)

# Inicializamos a pesos aleatorios
w1 = torch.randn(D_in, H, device = device, dtype = dtype)
w2 = torch.randn(H, D_out, device = device, dtype = dtype)
```

PyTorch


```
learning_rate = 1e-6
for t in range(500):
    # Paso hacia adelante: calculamos la "y" predicha
    h = x.mm(w1)
    h_relu = h.clamp(min = 0)
    y_pred = h_relu.mm(w2)
    # Calculamos la pérdida
    loss = (y_pred - y).pow(2).sum().item()
    print(t, loss)
    # Retroprograpación para calcular gradientes de w1 y w2 respecto a las pérdidas
    grad_y_pred = 2.0 * (y_pred - y)
    grad_w2 = h_relu.t().mm(grad_y_pred)
    grad_h_relu = grad_y_pred.mm(w2.t())
    grad_h = grad_h_relu.clone()
    grad_h[h < 0] = 0
    grad_w1 = x.t().mm(grad_h)
    # Actualizamos los pesos usando gradient desdenciente
    w1 -= learning_rate * grad_w1
    w2 -= learning_rate * grad_w2
```

PyTorch

```
0 31596260.0
1 29391412.0
2 31699210.0
...
498 4.088189234607853e-05
499 4.044295928906649e-05
```

Pandas

- https://pandas.pydata.org/
- Análisis de datos
- No está directamente relacionada con el ML, sino con la preparación de dataset antes del entrenamiento
- Extracción y preparación de datos
- Proporciona estructuras de datos de alto nivel y una amplia variedad de herramientas para el análisis de datos
- Asimismo, proporciona muchos métodos incorporados para tantear, combinar y filtrar datos

Pandas


```
# Ejemplo para ordenar un conjunto de datos en una tabla
import pandas as pd
data = {"country": ["Brazil", "Russia", "India", "China"
, "South Africa"],
       "capital": ["Brasilia", "Moscow", "New Delhi", "B
eijing", "Pretoria"],
       "area": [8.516, 17.10, 3.286, 9.597, 1.221],
       "population": [200.4, 143.5, 1252, 1357, 52.98] }
data_table = pd.DataFrame(data)
print(data_table)
```

Pandas

	country	capital	area	population
0	Brazil	Brasilia	8.516	200.40
1	Russia	Moscow	17.100	143.50
2	India	New Delhi	3.286	1252.00
3	China	Beijing	9.597	1357.00
4	South Africa	Pretoria	1.221	52.98

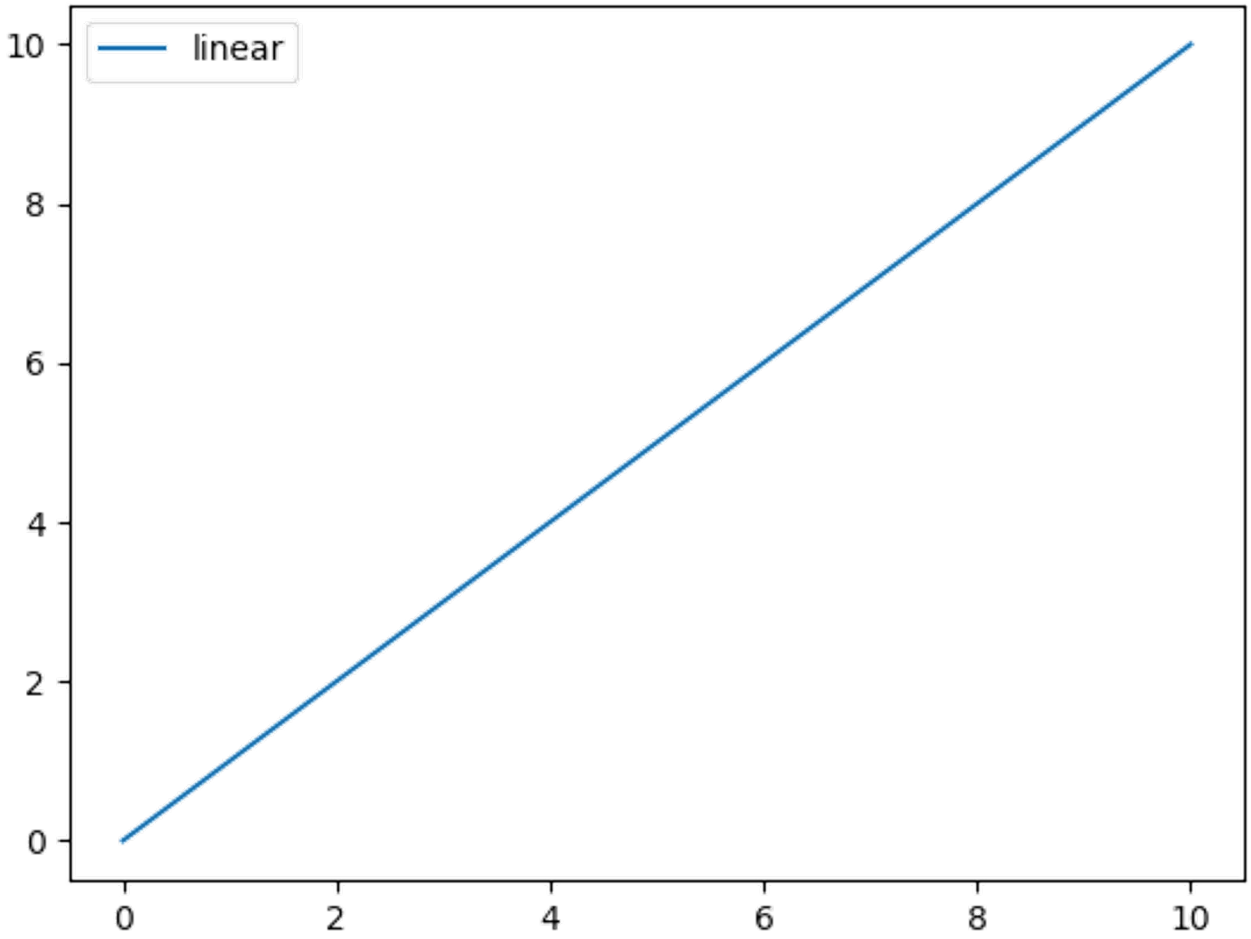
matplotlib

- https://matplotlib.org/
- Visualización de datos
- Particularmente útil para visualizar los patrones de los datos
- Ploteo en 2D usada para crear gráficos y diagramas en 2D
- Un módulo llamado pyplot facilita a los programadores el trazado
- Proporciona características para controlar los estilos de línea, las propiedades de las fuentes, los ejes de formato, etc.
- Proporciona varios tipos de gráficos y diagramas para la visualización de datos, es decir, histogramas, tablas de error, chats de barras, etc.

matplotlib


```
# Ejemplo sencillo para trazar una función lineal
import matplotlib.pyplot as plt
import numpy as np
# Preparamos los datos
x = np.linspace(0, 10, 100)
# Dibujamos los datos
plt.plot(x, x, label ='linear')
# Incluimos una leyenda
plt.legend()
# Mostramos el gráfico
plt.show()
```

matplotlib



NumpPy para el manejo de datos

- NumPy nos permite trabajar con arrays de forma muy eficiente, así como álgebra lineal, transformadas de Fourier y matrices
- Aunque Python dispone de listas que permiten trabajar con arrays, éstas son muy lentas
- NumPy proporciona arrays 50 veces más rápidos, en parte porque los elementos están todos almacenados en posiciones contiguas de memoria y en parte porque parte de NumPy está escrita en C/C++.
- Una vez hemos instalado NumPy siguiendo las explicaciones de las secciones anteriores, podemos usar los arrays de NumPy (objetos de clase `darray`)

Arrays


```
import numpy as np

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

print(arr)
#> [1 2 3 4 5]
print(type(arr))
#> <class 'numpy.ndarray'>
```

Dimensión del array


```
import numpy as np

a = np.array(42)
b = np.array([1, 2, 3, 4, 5])
c = np.array([[1, 2, 3], [4, 5, 6]])
d = np.array([[[1, 2, 3], [4, 5, 6]], [[1, 2, 3], [4, 5, 6]]])

print(a.ndim) # 1
print(b.ndim) # 2
print(c.ndim) # 2
print(d.ndim) # 4

print(a)          # 42
print(b[3])       # 4
print(c[0, 1])    # 2
print(d[0, 1, 1]) # 5
```

Array slicing

- Al igual que con los tipos built-in datos (cadenas, listas, etc.), en los arrays NumPy se puede hacer un slicing de los arrays utilizando notación de corchetes y dos puntos, pero con mucha mayor flexibilidad:
  - `array[start:end]`
  - `array[start:end:step]`
  - Si no pasamos un comienzo se considera 0
  - Si no pasamos un final se considera la longitud del array en esa dimensión
  - Si no pasamos un step se considera 1
  - Siempre se incluye el elemento en la posición *start*, pero se excluye el elemento en la posición *end*.

Array slicing


```
import numpy as np
arr = np.array([1, 2, 3, 4, 5, 6, 7])
print(arr[4:])
#> [5 6 7]
print(arr[:4])
#> [1 2 3 4]
print(arr[-4:-1])
#> [4 5 6]
print(arr[1:5:2])
#> [2 4]

arr = np.array([[1, 2, 3, 4, 5], [6, 7, 8, 9, 10]])
print(arr[0:2, 1:4])
#> [[2 3 4]
#   [7 8 9]]
```

Tipos de datos


```
import numpy as np
```

NumPy nos permite especificar con mucho mayor detalle los tipos de datos de los elementos en nuestros arrays

```
arr = np.array([1, 2, 3, 4], dtype='i4')


print(arr)
#> [1 2 3 4]
print(arr.dtype)
#> int32
```

Si un valor no puede ser convertido, se lanzará una excepción de tipo `ValueError`

Tipos de datos


```
import numpy as np

arr = np.array([1.1, 2.1, 3.1])

newarr = arr.astype(int)
# newarr = arr.astype("i") # alternativa

print(newarr)
#> [1 2 3]
print(newarr.dtype)
#> int32
```

Si queremos convertir el tipo de datos de un array existente, usaremos el método `astype()`

COPY y VIEW

- La principal diferencia entre una copia (obtenida con el método copy()) y una vista (obtenida con el método view()) de un array NumPy es que la copia es un nuevo array, mientras que la vista es una vista, valga la redundancia, del array original
- Es similar al concepto de asignación por referencia o de clonado de los datos en Python
- Si modificamos los datos en una copia, esto no afectará al original, y viceversa
- Por el contrario, si modificamos los datos en una vista, esto afectará al original, y viceversa

COPY y VIEW


```
import numpy as np
# COPY
arr = np.array([1, 2, 3, 4, 5])
x = arr.copy()
arr[0] = 39
print(arr)
#> [39  2  3  4  5]
print(x)
#> [1 2 3 4 5]
print(arr.base)
#> None
print(x.base)
#> None
```

Podemos comprobar si un array es propietario de sus propios datos mediante el atributo base

Éste devuelve el array original si el array es una vista

En caso de que el array sea una copia, este atributo devuelve None

COPY y VIEW


```
import numpy as np
# VIEW
arr = np.array([1, 2, 3, 4, 5])
x = arr.view()
arr[0] = 39
print(arr)
#> [39  2  3  4  5]
print(x)
#> [39  2  3  4  5]
print(arr.base)
#> None
print(x.base)
#> [39  2  3  4  5]
```

Podemos comprobar si un array es propietario de sus propios datos mediante el atributo base

Éste devuelve el array original si el array es una vista

En caso de que el array sea una copia, este atributo devuelve None

Forma (shape de un array)

- Con el atributo `shape` podemos obtener una tupla indicando en cada índica el número de elementos en cada dimensión
- Si hemos definido una dimensión mínima con `ndmin` en la creación del array, ésta será una forma de visualizarlo
- El método `reshape()` nos permite redimensionar un array (cambios entre 1-D y 2-D o 3D)
- Dicho método devuelve una vista, no una copia
- Con `reshape(-1)` podemos aplanar un array multidimensional en un array unidimensional

Forma (shape de un array)


```
import numpy as np
arr = np.array([[1, 2, 3, 4], [5, 6, 7, 8]])
print(arr.shape)
#> (2, 4)
arr = np.array([[1, 2, 3, 4], [5, 6, 7, 8]], ndmin=3)
print(arr.shape)
#> (1, 2, 4)
newarr = arr.reshape(-1)
print(newarr)
#> [1 2 3 4 5 6 7 8]
arr = np.array([1, 2, 3, 4, 5, 6, 7, 8])
newarr = arr.reshape(2, 4)
print(newarr)
#> [[1 2 3 4]
#   [5 6 7 8]]
```

Iterando por un array


```
import numpy as np
arr = np.array([1, 2, 3])
for x in arr:
  print(x)
# 1
# 2
# 3
arr = np.array([[1, 2, 3], [4, 5, 6]])
for x in arr: # iteramos por cada fila
  for y in x: # iteramos por cada elemento en cada fila
    print(y)
# 1
# 2
# 3
# 4
# 5
# 6
```

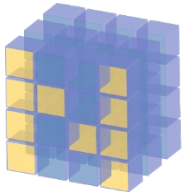
Concatenar arrays


```
import numpy as np

arr1 = np.array([1, 2, 3])
arr2 = np.array([4, 5, 6])

arr = np.concatenate((arr1, arr2))
print(arr)
#> [1 2 3 4 5 6]

arr = np.concatenate((arr1, arr2), axis=1)
print(arr)
#> [[1 2 5 6]
#>  [3 4 7 8]]
```

Dividir arrays


```
import numpy as np
arr = np.array([[1, 2], [3, 4], [5, 6], [7, 8], [9, 10], [11,
12]])
newarr = np.array_split(arr, 3)
print(newarr)

#> [array([[1, 2],
#         [3, 4]]), array([[5, 6],
#         [7, 8]]), array([[ 9, 10],
#         [11, 12]])]
```

Búsquedas en arrays




```
import numpy as np
arr = np.array([1, 2, 3, 4, 5, 4, 4])
# buscamos los índices donde el elemento sea igual a 4
x = np.where(arr == 4)
print(x)
#> (array([3, 5, 6], dtype=int64),)

# buscamos los índices donde el elemento sea par
arr = np.array([1, 2, 3, 4, 5, 6, 7, 8])
x = np.where(arr%2 == 0)
print(x)
#> (array([1, 3, 5, 7], dtype=int64),)
```

Ordenando arrays


```
import numpy as np

arr = np.array(['perro', 'gato', 'loro'])
print(np.sort(arr))
#> ['gato' 'loro' 'perro']

arr = np.array([[3, 2, 4], [5, 0, 1]])
print(np.sort(arr))
#> [[2 3 4]
#>  [0 1 5]]
```

Filtrando arrays


```
import numpy as np

arr = np.array([39, 40, 41, 42])

x = [True, False, True, False]

newarr = arr[x]
print(newarr)
#> [39 41]
```

Importación de datos

- Importación de archivos en formato .txt
- Importación de archivos planos CSV mediante la librería estándar de Python
- Importación de archivos planos CSV mediante NumPy
- Importación de archivos planos CSV mediante Pandas
- Importación desde una URL
- Importación desde otras librerías (toy datasets)

Importación de ficheros, comprobación previa

```
import os
os.chdir("C:\\Users\\xxx")
print(os.getcwd())
```

Importación de datos en formato .txt

```
import os
nombre = "datos.txt"
archivo = open(nombre, mode='r')
texto = archivo.read()
archivo.close()
# Podemos mostrar en pantalla el archivo que se acaba de
 cargar
print(texto)
```

Importación de CSV mediante librería estándar

```
import os
os.chdir("C:\\Users\\xxx")
os.getcwd()

import csv
f = open("diabetes.csv")
reader = csv.reader(f)
for row in reader:
    print (row)
```

Importación de CSV mediante librería estándar

```
['Pregnancies', 'Glucose', 'BloodPressure', 'SkinTh
ickness', 'Insulin', 'BMI', 'DiabetesPedigreeFuncti
on', 'Age', 'Outcome']
['6', '148', '72', '35', '0', '33.6', '0.627', '50'
, '1']
['1', '85', '66', '29', '0', '26.6', '0.351', '31',
'0']
...
```

Importación de CSV mediante NumPy

```
import numpy
filename = 'diabetes.csv'
raw_data = open(filename)
data = numpy.loadtxt(raw_data, delimiter=",", skiprows=1
)
print(data.shape)
print(data)
```

Podemos usar también la librería NumPy, con la función `numpy.loadtxt()`, que asume que el archivo no tiene cabeceras, a no ser que lo especifiquemos con el modificador `skiprows`

Importación de CSV mediante NumPy

```
(768, 9)
[[   6.     148.      72.    ...,    0.627   50.        1.   ]
 [   1.      85.      66.    ...,    0.351   31.        0.   ]
 [   8.     183.      64.    ...,    0.672   32.        1.   ]
 ...,
 [   5.     121.      72.    ...,    0.245   30.        0.   ]
 [   1.     126.      60.    ...,    0.349   47.        1.   ]
 [   1.      93.      70.    ...,    0.315   23.        0.
]]
```

Importación de CSV mediante Pandas

- Usaremos la función `readcsv()` para la carga, muy versátil:
  - **Detectar automáticamente los headers** sin que se lo tengamos que especificar
  - **Saltar líneas** con el modificador `skiprow`
  - **Detectar automáticamente el tipo de datos (entero, flotante, cadenas, etc.)**
  - **Identificar campos erróneos o vacíos**
  - **Convertir los datos CSV en un `dataframe` de Pandas**, que son estructuras de datos especialmente diseñadas para aplicar técnicas de ciencia de datos, siendo posible su uso como tablas o series temporales
  - Emplear la opción `chunksize` para cargar los datos en fragmentos de tamaño configurable en lugar de cargar todos los datos en un único espacio de memoria
    - De este modo, se mejora la eficiencia

Importación de CSV mediante Pandas

```
import pandas
filename = 'diabetes.csv'
data = pandas.read_csv(filename, header=0)

print(data.shape)
print (data.head(10))
```

Importación de CSV mediante Pandas

```
(768, 9)
   Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
0            6      148             72             35        0  33.6
1            1       85             66             29        0  26.6
2            8      183             64              0        0  23.3
3            1       89             66             23       94  28.1
4            0      137             40             35      168  43.1
5            5      116             74              0        0  25.6
6            3       78             50             32       88  31.0
7           10      115              0              0        0  35.3
8            2      197             70             45      543  30.5
9            8      125             96              0        0   0.0
```

Importación de CSV mediante Pandas

	DiabetesPedigreeFunction	Age	Outcome
0	0.627	50	1
1	0.351	31	0
2	0.672	32	1
3	0.167	21	0
4	2.288	33	1
5	0.201	30	0
6	0.248	26	1
7	0.134	29	0
8	0.158	53	1
9	0.232	54	1

Importación desde una URL (Pandas)

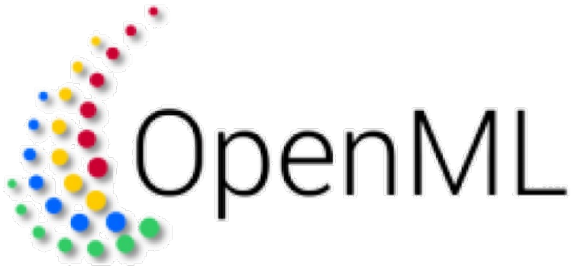
```
import pandas
url = "https://www.openml.org/data/get_csv/37/dataset_37_diabetes.
arff"
data = pandas.read_csv(url)
print(data)

# En Pandas, el modificador tail nos ofrece una vista de los datos
 mucho más atractiva
# Ver final
data.tail()
```

Importación desde una URL (Pandas)

```
 preg  plas  pres  ...   pedi  age           class
0       6   148    72  ...  0.627   50  tested_positive
1       1    85    66  ...  0.351   31  tested_negative
2       8   183    64  ...  0.672   32  tested_positive
3       1    89    66  ...  0.167   21  tested_negative
4       0   137    40  ...  2.288   33  tested_positive
..    ...   ...   ...  ...    ...  ...              ...
763    10   101    76  ...  0.171   63  tested_negative
764     2   122    70  ...  0.340   27  tested_negative
765     5   121    72  ...  0.245   30  tested_negative
766     1   126    60  ...  0.349   47  tested_positive
767     1    93    70  ...  0.315   23  tested_negative

[768 rows x 9 columns]
```

Importación desde una URL (Pandas)

- https://www.openml.org/



- Podemos ver los detalles del dataset sobre diabetes del ejemplo empleado (o bien buscar otros diferentes utilizando el motor de búsqueda) en:
  - https://www.openml.org/d/37

- https://www.kaggle.com/datasets

kaggle

Importación desde otras librerías (toy datasets)

- Como ya hemos visto con anterioridad, existen datasets que vienen incluidos “de serie” en las librerías de machine learning
- Lo hemos visto en los ejemplos de `scikit-learn`, pero también los hay incluidos en paquetes como `statsmodels` o `seaborn`
- Este tipo de conjuntos de datos para hacer pruebas se conocen como “toy datasets”

Machine learning básico con Python

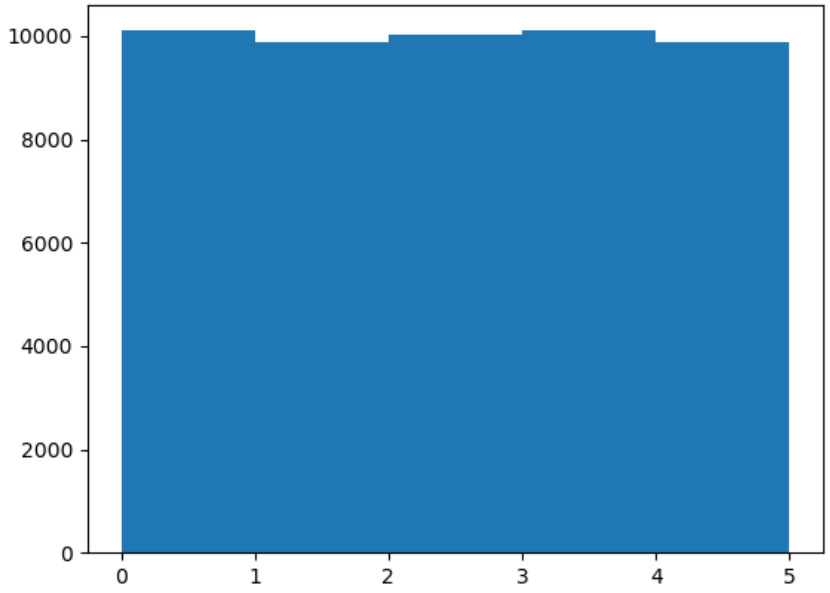
- Como hemos visto con anterioridad, podemos dividir los tipos de datos de nuestro dataset en:
- Numéricos
  - Discretos (el número de hijos por mujer)
  - Continuos (la longitud del pétalo de una flor)
- Categóricos (perro, gato, loro)
- Ordinales (suspenso, aprobado, notable, sobresaliente)

Medidas estadísticas

```
from scipy import stats

speed = [99,86,87,88,111,86,103,87,94,78,77,85,86]

print(numpy.mean(speed))
print(numpy.median(speed))
print(stats.mode(speed))
print(numpy.std(speed))
print(numpy.var(speed))
print(numpy.percentile(speed, 75))
```

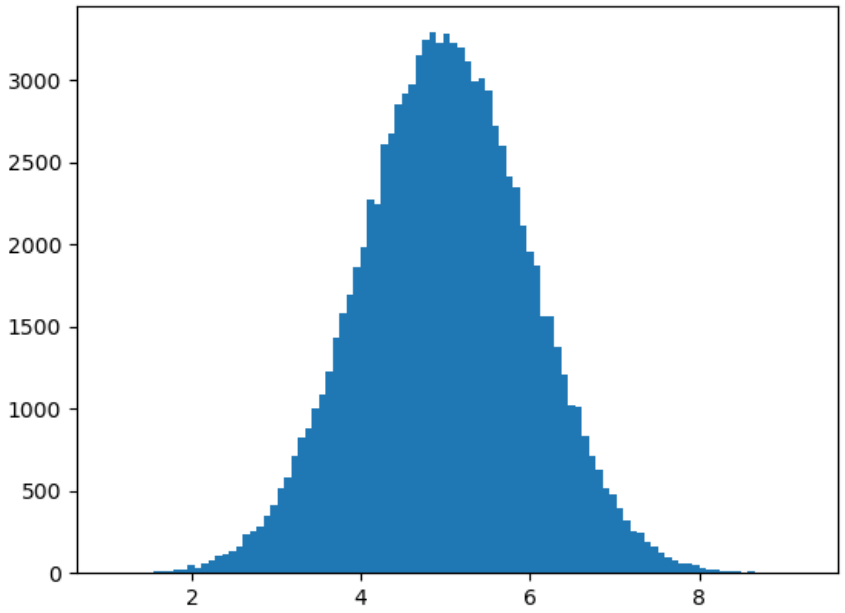
Distribución de los datos

```
import numpy
import matplotlib.pyplot as plt

x = numpy.random.uniform(0.0, 5.0, 50000)

plt.hist(x, 5)
plt.show()
```



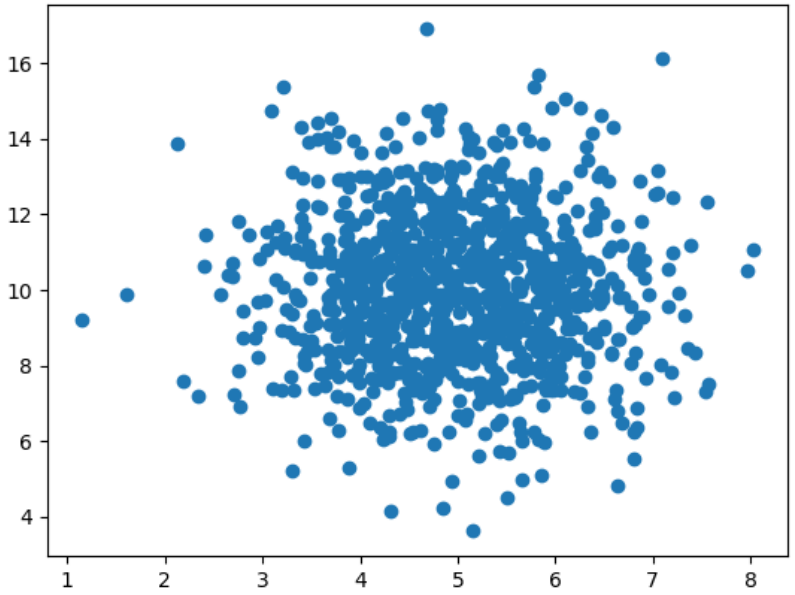
Distribución de los datos

```
import numpy
import matplotlib.pyplot as plt

x = numpy.random.normal(5.0, 1.0, 100000)

plt.hist(x, 100)
plt.show()
```



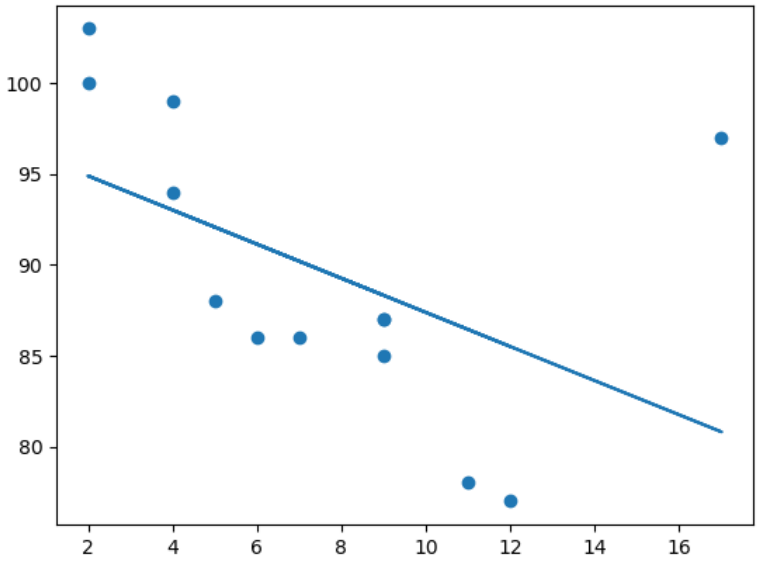
Distribución de los datos

```
import numpy
import matplotlib.pyplot as plt

x = numpy.random.normal(5.0, 1.0, 1000)
y = numpy.random.normal(10.0, 2.0, 1000)

plt.scatter(x, y)
plt.show()
```



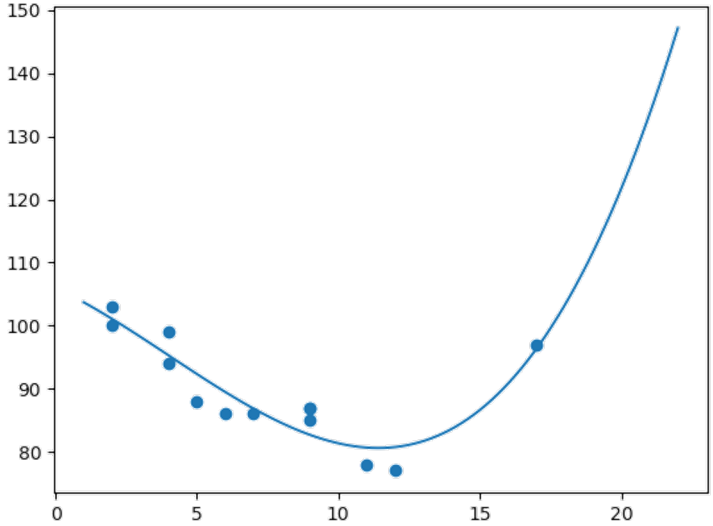
Regresión lineal

```
import matplotlib.pyplot as plt
from scipy import stats

x = [4,7,9,5,2,17,2,9,4,11,12,9,6]
y = [99,86,87,88,100,97,103,87,94,78,77,85,86]

slope, intercept, r, p, std_err = stats.linregress(x, y)

def myfunc(x):
  return slope * x + intercept

mymodel = list(map(myfunc, x))

plt.scatter(x, y)
plt.plot(x, mymodel)
plt.show()
```



Regresión polinómica

```
import numpy
import matplotlib.pyplot as plt

x = [4,7,9,5,2,17,2,9,4,11,12,9,6]
y = [99,86,87,88,100,97,103,87,94,78,77,85,86]

mymodel = numpy.poly1d(numpy.polyfit(x, y, 3))

myline = numpy.linspace(1, 22, 100)

plt.scatter(x, y)
plt.plot(myline, mymodel(myline))
plt.show()
```



Regresión múltiple

```
import pandas
from sklearn import linear_model

df = pandas.read_csv("https://www.openml.org/data/get_csv/52659/no
2.arff")

X = df[['cars_per_hour', 'temperature_at_2m']]
y = df['no2_concentration']

regr = linear_model.LinearRegression()
regr.fit(X, y)
print(regr.coef_)
#> [ 0.39296818 -0.03256069]
```

Regresión múltiple

```
# tratamos de predecir:
# concentración de NO2 en un punto en el que
# pasan 7 coches a la hora
# la temperatura a 2m de altura es de 5 grados
predictedNO2 = regr.predict([[7, 5]])
print(predictedNO2)
#> [3.57363206]
```

Referencias bibliográficas

- Bansal, H. (2019, octubre 18). Best Languages For Machine Learning in 2020! Medium. https://becominghuman.ai/best-languages-for-machine-learning-in-2020-6034732dd242
- Best Python libraries for Machine Learning. (2019, enero 18). GeeksforGeeks. https://www.geeksforgeeks.org/best-python-libraries-for-machine-learning/
- Puget, J.F. (2016) The Most Popular Language For Machine Learning and Data Science Is… KDnuggets. Recuperado 21 de marzo de 2020, de https://www.kdnuggets.com/the-most-popular-language-for-machine-learning-and-data-science-is.html/

Referencias bibliográficas

- Recuero de los Santos, P. (2019, octubre 1). *Python para todos: 5 formas de cargar datos para tus proyectos de Machine Learning - Think Big Empresas*. Think Big. https://empresas.blogthinkbig.com/python-5-formas-de-cargar-datos-csv-proyectos-machine-learning/
- Rogers, S., & Girolami, M. (2016). A first course in machine learning. Chapman and Hall/CRC.
- Russell, S., & Norvig, P. (2016). Artificial Intelligence: A Modern Approach Global Edition (Third Edition). Pearson.
- Unipython (2019) Importar datos con Python. (2019, mayo 13). https://unipython.com/importar-datos-con-python/

www.unir.net