

# Prácticas de Aprendizaje Automático

## Clase 2: Repaso

Pablo Mesejo y Francisco Baldán

Universidad de Granada

Departamento de Ciencias de la Computación e Inteligencia Artificial



UNIVERSIDAD  
DE GRANADA



# Índice

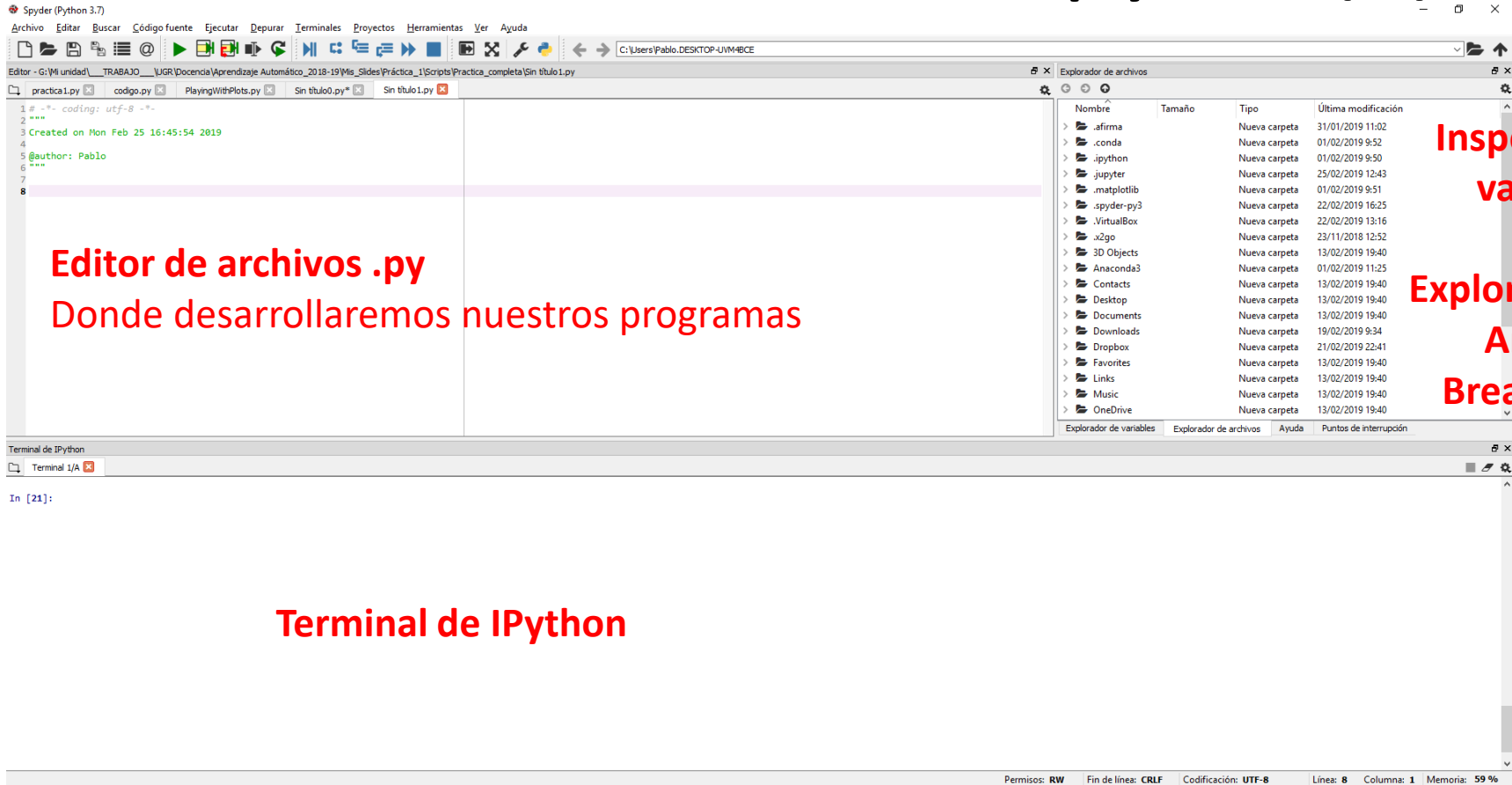
1. Revisando lo visto en la clase anterior
  - a) Anaconda/Conda/Spyder
  - b) Ejemplos de repaso de introducción a Python
2. Repaso NumPy

# Repaso Clase de Introducción a Python

# Anaconda/Conda/Spyder (1)

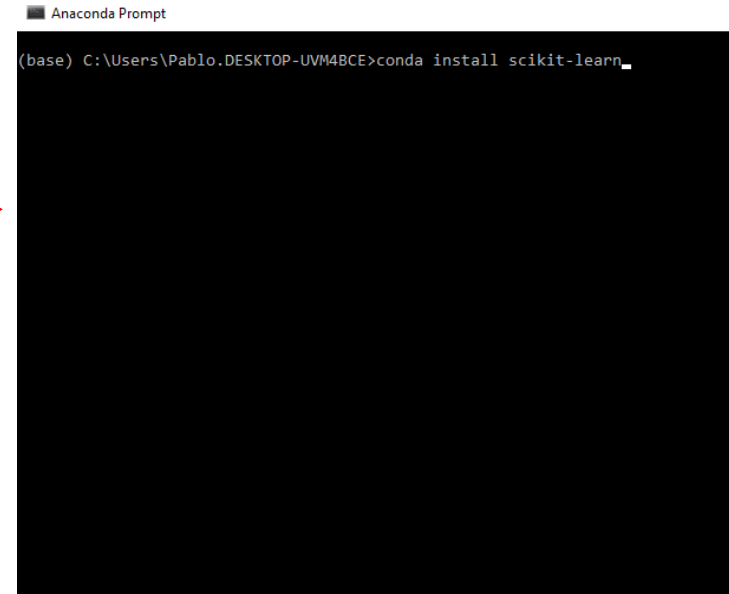
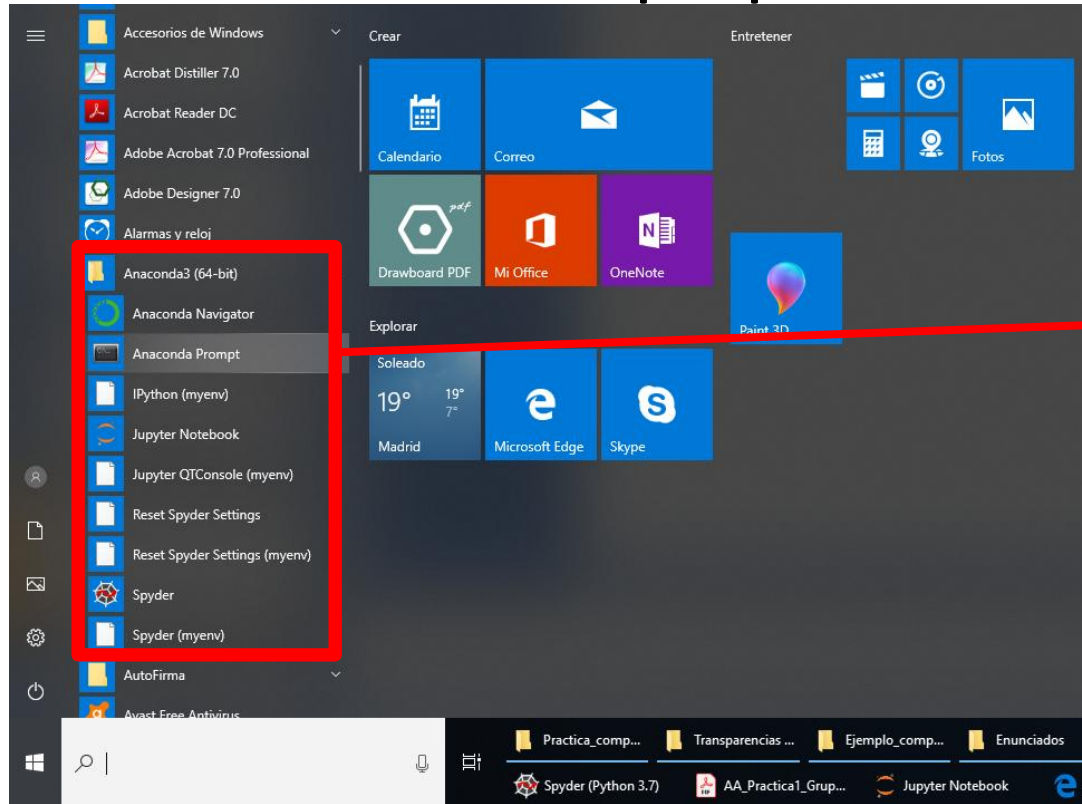
- ¿Todos lo tenéis instalado?
- ¿Algún problema con el entorno?

# Anaconda/Conda/Spyder (2)



# Anaconda/Conda/Spyder (3)

- Actualizando paquetes con conda en Windows



# Anaconda/Conda/Spyder (y 4)

- Saber si tenéis que instalar un paquete

Anaconda Prompt

```
(base) C:\Users\Pablo.DESKTOP-UVM4BCE>conda list
# packages in environment at C:\Users\Pablo.DESKTOP-UVM4BCE\Anaconda3:
#
# Name                          Version                      Build      Channel
matplotlib                      3.0.2                       py37hc8f65d3_0
numpy                           1.15.4                      py37h19fb1c0_0
numpy-base                     1.15.4                      py37hc3f5095_0
scikit-learn                   0.20.2                     py37h343c172_0
```

# Repaso Python (1)

```
def funcion1(arr,value=3):  
    return([x*2 for x in arr if x < value//2])
```

```
funcion1([3,6,8,10,1,2,1],5)
```



# Repaso Python (1)

```
def funcion1(arr,value=3):  
    return([x*2 for x in arr if x < value//2])
```

```
funcion1([3,6,8,10,1,2,1],5)
```

[2,2]

# Repaso Python (2)

¿Podríaais decir la función que se corresponde con este código Python?

```
E(u,v) = (u**3*np.exp(v-2)-4*v**3*np.exp(-u))**2
```

# Repaso Python (2)

¿Podrías decir la función que se corresponde con este código Python?

```
E(u,v) = (u**3*np.exp(v-2)-4*v**3*np.exp(-u))**2
```

$$E(u, v) = (u^3 e^{(v-2)} - 4v^3 e^{-u})^2$$

# Repaso Python (3)

¿Qué imprime este código?

```
def fun1(b):  
    b.append(1)  
    b = 'New Value'  
    print('Dentro de fun1: ', b)  
  
a = [0]  
fun1(a)  
print('Despues de fun1: ', a)
```

# Repaso Python (3)

¿Qué imprime este código?

```
def fun1(b):  
    b.append(1)  
    b = 'New Value'  
    print('Dentro de fun1: ', b)
```

```
a = [0]  
fun1(a)  
print('Despues de fun1: ', a)
```

Cuando se llama a `fun1`, `b` y `a` apuntan al mismo valor ([0])

Cuando hacemos `b.append(1)` → [0] se convierte en [0, 1]

Cuando hacemos `b = 'New Value'` → ahora `b` apunta a una nueva lista en memoria que contiene 'New Value'. Pero `a` apunta todavía a la lista [0, 1]

```
Dentro de fun1:  New Value  
Despues de fun1:  [0, 1]
```

# Repaso Python (3)

¿Y si queremos que el valor modificado se vea fuera (es decir, que cambie el valor de a)?

```
def fun1(b):  
    b.append(1)  
    b = 'New Value'  
    print('Dentro de fun1: ', b)  
  
a = [0]  
fun1(a)  
print('Despues de fun1: ', a)
```

# Repaso Python (3)

¿Y si queremos que el valor modificado se vea fuera (es decir, que cambie el valor de a)?

```
def fun1(b):
```

```
    b.append(1)
```

```
    b = 'New Value'
```

```
    print('Dentro de fun1: ', b)
```

```
    return b
```

Introducimos un `return`

Reasignamos la variable a la salida de la función

```
a = [0]
```

```
a = fun1(a)
```

```
print('Despues de fun1: ', a)
```

```
Dentro de fun1:  New Value
Despues de fun1:  New Value
```

# Repaso Python (4)

¿Qué contiene tupla tras la última asignación?

```
In [40]: tupla = (5, 't1', True, 0.5)
```

```
In [41]: tupla[2]
```

```
Out[41]: True
```

```
In [42]: tupla[2] = False
```



# Repaso Python (4)

¿Qué contiene tupla tras la última asignación?

```
In [40]: tupla = (5, 't1', True, 0.5)
```

```
In [41]: tupla[2]
```

```
Out[41]: True
```

```
In [42]: tupla[2] = False
```

```
Traceback (most recent call last):
```

```
File "<ipython-input-42-1f9f497d5a2b>", line 1, in <module>  
    tupla[2] = False
```

```
TypeError: 'tuple' object does not support item assignment
```

# Repaso Python (y 5)

- Importando módulos. Algo útil para la práctica 0...

```
In [1]: from sklearn import datasets
```

```
In [2]: iris = datasets.load_iris()  
...: X = iris.data  
...: y = iris.target
```

```
In [1]: import sklearn as skl
```

```
In [2]: iris = skl.datasets.load_iris()  
Traceback (most recent call last):
```

```
File "<ipython-input-2-2392f01db2c1>", line 1, in <module>  
    iris = skl.datasets.load_iris()
```

```
AttributeError: module 'sklearn' has no attribute 'datasets'
```

## ¿Por qué?

**datasets** es un sub-paquete de **sklearn**

→ Cuando importas un paquete solamente las variables/funciones/clases en el **\_\_init\_\_.py** de ese paquete son directamente visibles, no los sub-paquetes o módulos.

# Repaso NumPy

# Repaso Numpy (1)

¿Qué contiene X?

```
import numpy as np
```

```
X1 = np.zeros((10,1))
```

```
X2 = np.ones((10,1))
```

```
X3 = np.ones((10,1))*2
```

```
X4 = np.ones((10,1))*3
```

```
X = np.concatenate((X1,X2,X3,X4))
```

```
X = np.reshape(X,(10,4))
```

# Repaso Numpy (1)

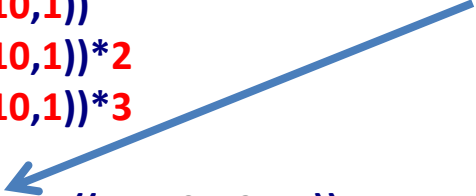
¿Qué contiene X?

```
import numpy as np
```


```
X1 = np.zeros((10,1))  
X2 = np.ones((10,1))  
X3 = np.ones((10,1))*2  
X4 = np.ones((10,1))*3
```

```
X = np.concatenate((X1,X2,X3,X4))  
X = np.reshape(X,(10,4))
```

Concatena por defecto con respecto a axis=0. Aquí obtenemos un vector columna de 40 elementos



El reshape se hace, por defecto, con respecto al axis=1 (**order= 'C'**)! Es decir, por filas!  
Si os interesa hacerlo en el axis=0 (**order= 'F'**)



```
Out[2]:  
array([[0., 0., 0., 0.],  
       [0., 0., 0., 0.],  
       [0., 0., 1., 1.],  
       [1., 1., 1., 1.],  
       [1., 1., 1., 1.],  
       [2., 2., 2., 2.],  
       [2., 2., 2., 2.],  
       [2., 2., 3., 3.],  
       [3., 3., 3., 3.],  
       [3., 3., 3., 3.]])
```

# Repaso Numpy (2)

¿Qué contiene y?

```
import numpy as np
```

```
X1 = np.zeros((10,1))
```

```
X2 = np.ones((10,1))
```

```
X3 = np.ones((10,1))*2
```

```
X4 = np.ones((10,1))*3
```

```
X = np.concatenate((X1,X2,X3,X4))
```

```
X = np.reshape(X,(10,4))
```

```
y = np.sum(X,axis=1)
```

Out[2]:

```
array([[0., 0., 0., 0.],  
       [0., 0., 0., 0.],  
       [0., 0., 1., 1.],  
       [1., 1., 1., 1.],  
       [1., 1., 1., 1.],  
       [2., 2., 2., 2.],  
       [2., 2., 2., 2.],  
       [2., 2., 3., 3.],  
       [3., 3., 3., 3.],  
       [3., 3., 3., 3.]])
```

# Repaso Numpy (2)

¿Qué contiene y?

```
import numpy as np
```

```
X1 = np.zeros((10,1))
```

```
X2 = np.ones((10,1))
```

```
X3 = np.ones((10,1))*2
```

```
X4 = np.ones((10,1))*3
```

```
X = np.concatenate((X1,X2,X3,X4))
```

```
X = X.reshape(X,(10,4))
```

```
y = np.sum(X,axis=1)
```

Out[2]:

```
array([[0., 0., 0., 0.],  
       [0., 0., 0., 0.],  
       [0., 0., 1., 1.],  
       [1., 1., 1., 1.],  
       [1., 1., 1., 1.],  
       [2., 2., 2., 2.],  
       [2., 2., 2., 2.],  
       [2., 2., 3., 3.],  
       [3., 3., 3., 3.],  
       [3., 3., 3., 3.]])
```

```
array([ 0.,  0.,  2.,  4.,  4.,  8.,  8., 10., 12., 12.])
```



Suma por filas!

# Repaso Numpy (3)

¿Qué contiene X\_class?

```
import numpy as np
```

```
X1 = np.zeros((10,1))
```

```
X2 = np.ones((10,1))
```

```
X3 = np.ones((10,1))*2
```

```
X4 = np.ones((10,1))*3
```

```
X = np.concatenate((X1,X2,X3,X4))
```

```
X = np.reshape(X,(10,4))
```

```
y = np.sum(X,axis=1)
```

```
classes = np.unique(y)
```

```
X_class = [X[y==c_i] for c_i in classes]
```

Out[2]:

```
array([[0., 0., 0., 0.],  
       [0., 0., 0., 0.],  
       [0., 0., 1., 1.],  
       [1., 1., 1., 1.],  
       [1., 1., 1., 1.],  
       [2., 2., 2., 2.],  
       [2., 2., 2., 2.],  
       [2., 2., 3., 3.],  
       [3., 3., 3., 3.],  
       [3., 3., 3., 3.]])
```

```
array([ 0.,  0.,  2.,  4.,  4.,  8.,  8., 10., 12., 12.]])
```



# Repaso Numpy (3)

¿Qué contiene X\_class?

```
import numpy as np
```

```
X1 = np.zeros((10,1))
```

```
X2 = np.ones((10,1))
```

```
X3 = np.ones((10,1))*2
```

```
X4 = np.ones((10,1))*3
```

```
X = np.concatenate((X1,X2,X3,X4))
```

```
X = np.reshape(X,(10,4))
```

```
y = np.sum(X,axis=1)
```

```
classes = np.unique(y)
```

```
X_class = [X[y==c_i] for c_i in classes]
```

Out[2]:

```
array([[0., 0., 0., 0.],  
       [0., 0., 0., 0.],  
       [0., 0., 1., 1.],  
       [1., 1., 1., 1.],  
       [1., 1., 1., 1.],  
       [2., 2., 2., 2.],  
       [2., 2., 2., 2.],  
       [2., 2., 3., 3.],  
       [3., 3., 3., 3.],  
       [3., 3., 3., 3.]])
```

```
array([ 0.,  0.,  2.,  4.,  4.,  8.,  8., 10., 12., 12.])
```

```
for c_i in classes:  
    print(c_i)  
0.0  
2.0  
4.0  
8.0  
10.0  
12.0
```

Devuélveme las  
entradas/filas de **X** cuya  
suma (**y**) sea igual a las  
clases encontradas (**classes**)

# Repaso Numpy (3)

¿Qué contiene X\_class?

```
import numpy as np
```

```
X1 = np.zeros((10,1))
```

```
X2 = np.ones((10,1))
```

```
X3 = np.ones((10,1))*2
```

```
X4 = np.ones((10,1))*3
```

```
X = np.concatenate((X1,X2,X3,X4))
```

```
X = np.reshape(X,(10,4))
```

```
y = np.sum(X,axis=1)
```

```
classes = np.unique(y)
```

```
X_class = [X[y==c_i] for c_i in classes]
```

```
Out[2]:  
array([[0., 0., 0., 0.],  
       [0., 0., 0., 0.],  
       [0., 0., 1., 1.],  
       [1., 1., 1., 1.],  
       [1., 1., 1., 1.],  
       [2., 2., 2., 2.],  
       [2., 2., 2., 2.],  
       [2., 2., 3., 3.],  
       [3., 3., 3., 3.],  
       [3., 3., 3., 3.]])  
array([ 0.,  0.,  2.,  4.,  4.,  8.,  8., 10., 12., 12.])
```

```
Out[6]:  
[array([[0., 0., 0., 0.],  
       [0., 0., 0., 0.]]) array([[0., 0., 1., 1.]]) array([[1., 1., 1., 1.],  
       [1., 1., 1., 1.]]) array([[2., 2., 2., 2.],  
       [2., 2., 2., 2.]]) array([[2., 2., 3., 3.]]) array([[3., 3., 3., 3.],  
       [3., 3., 3., 3.]])]
```

```
In [7]: X_class[0]
```

```
Out[7]:  
array([[0., 0., 0., 0.],  
       [0., 0., 0., 0.]])
```

y == 0.0

```
In [8]: X_class[1]
```

```
Out[8]: array([[0., 0., 1., 1.]])
```

y == 2.0

# Repaso Numpy (4)

¿Qué hacen estas 4 líneas de código?

```
Z = np.arange(10)
v = np.random.uniform(0,10)
index = (np.abs(Z-v)).argmin()
print(Z[index])
```

# Repaso Numpy (4)

¿Qué hacen estas 4 líneas de código?

```
Z = np.arange(10)
v = np.random.uniform(0,10)
index = (np.abs(Z-v)).argmin()
print(Z[index])
```

```
In [23]: Z
Out[23]: array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

In [24]: v
Out[24]: 4.1970769994524515

In [25]: index = (np.abs(Z-v)).argmin()
...: print(Z[index])
4

In [26]: index
Out[26]: 4
```

**Dado un array Z, y un valor v, ¿cuál es el valor de Z más próximo a v?**

# Repaso Numpy (5)

¿Cómo es la matriz que imprimen de salida estas tres líneas de código?

```
a = np.arange(10).reshape(2,-1)
```

```
b = np.repeat(1, 10).reshape(2,-1)
```

```
np.concatenate([a, b], axis=0)
```

# Repaso Numpy (5)

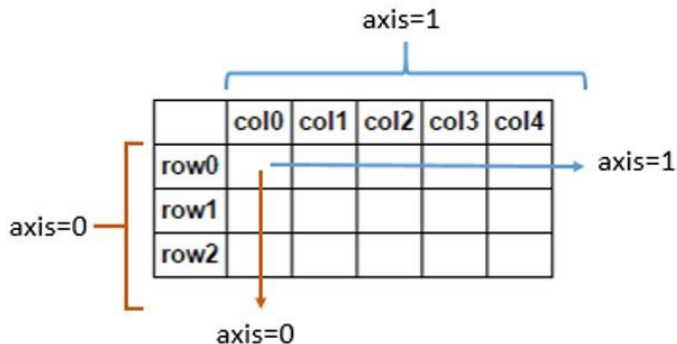
¿Cómo es la matriz que imprimen de salida estas tres líneas de código?

```
a = np.arange(10).reshape(2,-1)
```

```
b = np.repeat(1, 10).reshape(2,-1)
```

```
np.concatenate([a, b], axis=0)
```

Lo concatenamos por filas!



```
In [36]: a
```

```
Out[36]:
```

```
array([[0, 1, 2, 3, 4],  
       [5, 6, 7, 8, 9]])
```

```
In [37]: b
```

```
Out[37]:
```

```
array([[1, 1, 1, 1, 1],  
       [1, 1, 1, 1, 1]])
```

```
In [38]: np.concatenate([a, b], axis=0)
```

```
Out[38]:
```

```
array([[0, 1, 2, 3, 4],  
       [5, 6, 7, 8, 9],  
       [1, 1, 1, 1, 1],  
       [1, 1, 1, 1, 1]])
```

# Prácticas de Aprendizaje Automático

## Clase 2: Repaso

Pablo Mesejo y Francisco Baldán

Universidad de Granada

Departamento de Ciencias de la Computación e Inteligencia Artificial



UNIVERSIDAD  
DE GRANADA

