Prácticas de Aprendizaje Automático

Parte 2: Scikit-learn, Numpy y Matplotlib

Índice

Arrays en Numpy. Funciones básicas.

Numpy es el paquete principal en python para manejar arrays de N dimensiones de forma eficiente (código en C/C++/Fortran).

Incluye funciones para realizar operaciones matriciales, algebra lineal, transformaciones de Fourier y generación de números aleatorios.

Importamos el paquete con **import numpy** (se recomienda usar mejor **import numpy as np** para poder llamarla con np).

Arrays en Numpy. Funciones básicas.

Crear array no inicializado, ceros o unos:

- No inicializado = **np.empty(<shape>, <type>)** / zeros = **np.zeros(<shape>, <type>)** / ones = **np.ones(<shape>, <type>)**.
- <shape> -> Tupla con el tamaño por dimensión. Ejemplo: Matriz 5x5 -> (5, 5).
- <type> -> Tipo de numpy (np.int32, np.float32, ...).
- También podemos crear un array nuevo con la misma forma y tipo que otro usando **np.empty_like** / **np.zeros_like** / **np.ones_like**

Crear array aleatorio:

- Uniforme: **np.random.uniform(low=<min_val>, high=<max_val>, size=<shape>)**. Este array es de números reales entre <min_val> y <max_val>.
- Uniforme (enteros): **np.random.random_integers(low=<min_val>, high=<max_val>, size=<shape>)**. Este array es de números enteros entre <min_val> y <max_val>.

Arrays en Numpy. Funciones básicas.

Obtener tamaño del array: **array.shape**

Obtener tipo array: **array.dtype**

Cambiar tipo array: **array.astype(<new_numpy_type>)**

Cambiar forma del array (tienen que mantenerse el mismo número de elementos): **array.reshape(<new_shape>)**. Se puede poner una dimensión como -1, de forma que su nuevo tamaño se calculará en función del tamaño del resto de dimensiones y del número de elementos.

Ejemplo:

```
d1 = np.zeros((9, ), np.int32)
d1 = d1.reshape((-1, 3))

print(d1)
```

Arrays en Numpy. Funciones básicas.

Estadísticas:

- Mínimo: **array.min(axis=<dim>)**
- Máximo: **array.max(axis=<dim>)**
- Indice del mínimo: **array.argmin(axis=<dim>)**
- Indice del máximo: **array.argmax(axis=<dim>)**
- Media: **array.mean(axis=<dim>)**
- Media ponderada: **np.average(array, axis=<dim>, weights=<pesos>)**
- Desviación estándar: **array.std(axis=<dim>)**
- Varianza: **array.var(axis=<dim>)**

Arrays en Numpy. Funciones básicas.

Estadísticas:

- Mediana: **np.median(array, axis=<dim>)**
- Percentiles: **np.percentile(array, q <lista percentiles>, axis=<dim>)**
- Suma: **array.sum(axis=<dim>)**
- Multiplicación: **array.prod(axis=<dim>)**
- Algún elemento es verdad: **array.any(axis=<dim>)**
- Todos son verdad: **array.all(axis=<dim>)**

<dim> es la dimensión a lo largo de la cual se calcula. Si no ponemos nada, por defecto se calcula como si el array fuese de una sola dimensión.

Todas estas funciones, salvo **any** y **all** tiene otra versión **np.nan**<función> (**np.nansum**, **np.nanmedian**) que realiza dicha función tratando los valores NaN como zeros.

Arrays en Numpy. Funciones básicas.

Permutar dimensiones: **array.swapaxes(<dim1>, <dim2>)**

Copiar array: array2 = **array.copy()** (Sin esto, array2 tendría una referencia).

Ordenar array de menor a mayor: **array.sort(axis=<dim>)**

Índices que ordenan array de menor a mayor: **np.argsort(array, axis=<dim>)**.

Arrays en Numpy. Funciones básicas.

Operaciones elemento a elemento:

- Array con array: Suma (**a1+a2**), producto(**a1*a2**), resta(**a1-a2**), división(**a1/a2**), división entera(**a1//a2**), potencia (**a1**a2**), mayor / mayor igual (**a1>a2** / **a1>=a2**), menor / menor igual (**a1<a2** / **a1<=a2**), igual (**a1==a2**) y no igual (**a1!=a2**).
- Escalar (c) con array (a): Suma (**c+a**), producto(**c*a**), resta(**a-c** ó **c-a**), división(**a/c** ó **c/a**), división entera(**a//c** ó **c//a**), potencia (**a**c** ó **c**a**), mayor / mayor igual (**a>c** / **a>=c**), menor / menor igual (**a<c** / **a<=c**), igual (**a==c**) y no igual (**a!=c**).
- Resto: **np.mod(a1, a2) / np.mod(a, c) / np.mod(c, a)**.
- Valor absoluto: **np.abs(a)**.
- Raíz cuadrada: **np.sqrt(a)**.
- Exponencial (e**a): **np.exp(a)**.
- Logaritmo natural / Logaritmo 2 / Logaritmo 10: **np.log(a)** / **np.log2(a)** / **np.log10(a)**.

Arrays en Numpy. Funciones básicas.

Operaciones elemento a elemento:

- Funciones trigonométricas: **np.cos(a)**, **np.sin(a)**, **np.tan(a)**, **np.arccos(a)**, **np.arcsin(a)**, **np.arctan(a)**.
- Signo: **np.sign(a)**.
- Mínimo elemento a elemento: np.minimum(a1, a2).
- Máximo elemento a elemento: np.maximum(a1, a2).
- ceil, floor, redondear al entero más cercano: **np.ceil(a)**, **np.floor(a) np.rint(a)**.
- Obtener los valores únicos: np.unique(array).
- ¿Están los elementos de un array en otro? (in de python): np.in1d(a1, a2).
- Unión, intersección, diferencia (de conjuntos) y diferencia simétrica: **np.union1d(a1, a2)**, **np.intersect1d(a1, a2)**, **np.setdiff1d(a1, a2)** y **setxor1d(a1, a2)**.

Arrays en Numpy. Funciones básicas.

Operaciones con matrices:

- Producto vectorial/matricial: **array1.dot(array2)**.
- Transpuesta: **array1.transpose()**.
- Diagonal (como array de 1d): **np.diagonal(array1)**.
- Traza: **np.trace(array1)**.
- Determinante: **np.linalg.det(array1)**.
- Inversa: **np.linalg.inv(array1)**.
- Descomposición en valores singulares (SVD): **np.linalg.svd(array1)**.

Arrays en Numpy. Funciones básicas.

Vectorización:

- If else vectorizado: **np.where(<condición array>, <valor cond true>, <valor cond false>)**.
- Repetir array: **np.repeat(array, <nº repeticiones>, axis=<dim>)**.
- Podemos usar la función **np.apply_over_axes(f, array, axes=(<dim1>, <dim2>, ...))** para aplicar la función f sobre las dimensiones de array indicadas. ¡Ojo! El orden en el que se indican las dimensiones es el que se seguirá a la hora de realizar el cálculo.
- Del mismo modo, **np.apply_along_axis(f, axis=<dim>, array)** aplica la función solo sobre la dimensión indicada.

Hay muchas más funciones en el paquete, se recomienda echarle un vistazo a la documentación disponible en: http://www.numpy.org/

Indexado Numpy.

Permite hacer lo mismo que las listas de python y mucho más. Ejemplo:

```
import numpy as np

m = np.arange(0, 6).reshape(2, 3) #Crea un array con valores 0,1,..6

print('Mostrar la primera fila')
print(m[0, :])
print('Mostrar las columnas pares')
print(m[:, ::2])
print('Mostrar la esquina inferior derecha')
print(m[-1, -1])
```

Además, podemos indexar usando arrays de enteros o un array con booleanos (True, False):

```
m[m<3] = 0
print('Todos los elemntos menores a 3 son 0 ahora')
print(m)
```

Datos. Terminología.

- **Variable**: Característica de interés. Se puede representar como *un dataframe, una matriz, un vector*.
- **Muestra Observada**: Conjunto de valores de la variable obtenidos de manera homogénea. Sería *una fila del dataframe, fila de la matriz, componente del vector*.
- **Tamaño muestral**: Número de datos observados. Serían *la longitud del dataframe, de la matriz, del vector*.
- Tipos de atributos:
  - Cualitativo : Intrínsecamente no tiene carácter numérico (categórica).
  - Cuantitativo : Intrínsecamente numérico:
    - Discreto (cantidad finita o numerable de valores).
    - Contínuo (valores reales).

Datos. Generación de datos.

Numpy cuenta con varias funciones para la generación de datos pseudo-aleatorios (Random sampling) dentro de numpy.random.

Podemos fijar la semilla del generador de números pseudo-aleatorios para tener resultados reproducibles con **np.random.seed(seed)** (seed es un entero).

Además, incluye funciones para alterar aleatoriamente el orden de un array:

- **np.random.shuffle(x)**: Modifica x cambiando el orden de los elementos aleatoriamente. (Función in-place, no devuelve nada).
- **np.random.permutation(x)**: Devuelve el array x con sus elementos desordenados (de forma aleatoria).

Datos. Generación de datos continuos.

Distribución	Comando (np.random.)
Normal	normal(loc, scale, size)
Exponencial	exponential(scale, size)
Gamma	gamma(shape, scale, size)
Poisson	poisson(lam, size)
Weibull	weibull(a, size)

Datos. Generación de datos continuos.

Distribución	Comando (np.random.)
Beta	beta(a, b, size)
t de Student (estandarizada)	standard_t(df, size)
F	f(dfnum, dfden, size)
Chi cuadrado	chisquare(df, size)
Binomial	binomial(n, p, size=None)

Datos. Generación de datos discretos.

A parte de **random_integers**, podemos generar datos discretos muestreando aleatoriamente de un conjunto. Para estos, podemos usar la función:

np.random.choice(a, size, replace=True, p=None):

- a: Lista o array con los posible valores.
- size: Forma del array a generar (tupla o lista con el tamaño de cada dimensión).
- replace: Indica si los elementos se sacarán de la muestra, de forma que no se repitan.

- p: Array con la probabilidad de cada elemento. Es opcional.

Datos. Lectura de datos.

Hay muchas formas de leer datos de disco. Por ejemplo, podemos usar la funcionalidad existente en python para leer una matriz de un fichero de texto simple con cada línea siendo una fila y estando los elementos separados por espacio:

```
matriz = []

f = open('mat.txt', 'r')

for l in f:
    fila_matriz = []
    l = l.rstrip()

    for e in l.split(' '):
        if e != '':
            fila_matriz.append(float(e))

    if len(fila_matriz) > 0:
        matriz.append(fila_matriz)

f.close()

matriz = np.array(matriz, np.float64)
```

```
1 2 3
4.5 5.5 6.7
1 2 4.5
```

Datos. Lectura de datos.

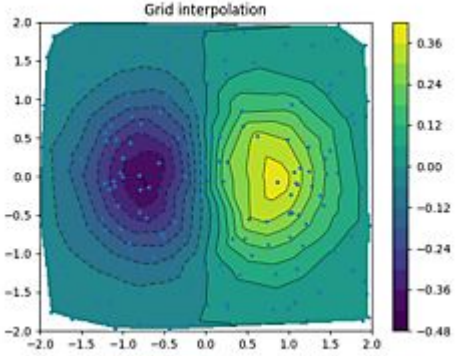
Podemos hacer lo mismo usando la función **np.loadtxt(<path fichero>, delimiter=<delimitador>, dtype=<tipo>, skiprows=<saltar filas al inicio>)**. Con esta función se puede leer también un csv en una matriz de numpy y con **np.savetxt(<path fichero>, <array>, delimiter=<delimitador>, header=<cabecera>)** guardarlo en este formato.

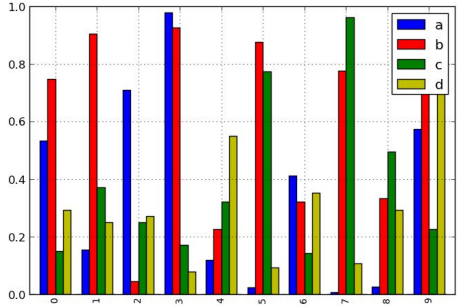
Si bien, si vamos a guardar o leer un array de numpy es preferible usar el formato binario npy. Para esto, para guardar usamos **np.save(<path fichero>, <array>)** y **np.load(<path fichero>)** para leerlo.

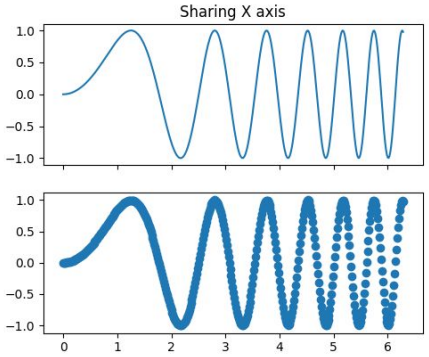
Así mismo, con datos en csv también se puede usar la librería pandas, pero esta no va a ser usada en este curso.

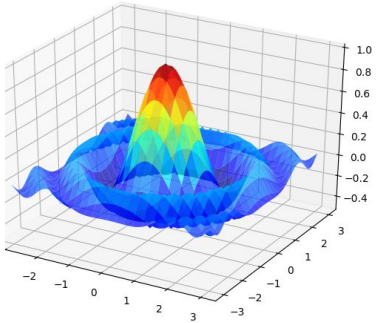
Representación con Matplotlib.

Nos permite dibujar distintos tipos de gráficos para pintar los datos de forma sencilla y rápida usando listas o vectores de numpy (¡incluso puede con gráficos 3d!)







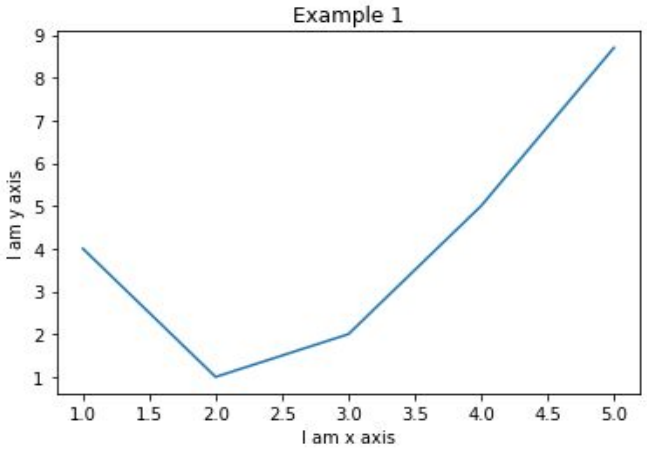


Representación con Matplotlib.

Las funciones básicas necesarias para pintar con matplotlib son:

```
import matplotlib.pyplot as plt

y = [4, 1, 2, 5, 8.7]
x = range(1, len(y)+1)
plt.plot(x, y)              #Pinta una línea
plt.xlabel('I am x axis') #Nombre del eje x
plt.ylabel('I am y axis') #Nombre del eje y
plt.title('Example 1')    #Título
plt.show()                  #Mostrar gráfico
```



Representación con Matplotlib. Tipos.

Lineas: **plt.plot(<x>, <y>, <color y forma>)**.

Barras: **plt.bar(<x>,<y>, color=<color>, orientation={‘vertical’, ‘horizontal’})**.

Puntos: **plt.scatter(<x>, <y>, c=<color index>, cmap=<map del color>, s=<tamaño>)** cmap puede ser alguno de los declarados en **plt.cm**.

Diagrama de cajas: **plt.boxplot(<x>, notch=<pintar los intervalos de confianza>, sym=<Str** con el símbolo para los outliers. Si no se pone nada no los pintará**>, vert=<pintar en vertical u horizontal>)**.

Para más información acerca del color, forma y otros parámetros, consultar documentación.

Representación con Matplotlib.

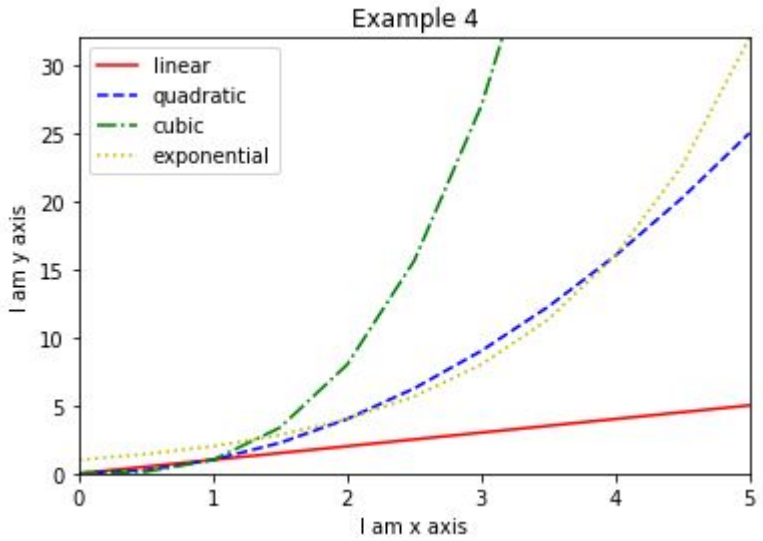
Podemos alterar algunas cosas más del gráfico mostrado:

- Cambiar rango de los ejes: **plt.axis([0, 6, 0, 10])** (primero x, luego y)
- Añade la legenda (requiere añadir **label=<name>** a la llamada de cada comando de dibujado (como plot)): **plt.legend(loc=<location>)** <location> puede ser: ‘best’, ‘upper right’, ‘upper left’, ‘lower left’, ‘lower right’, ‘right’, ‘center left’, ‘center right’, ‘lower center’, ‘upper center’, ‘center’.

Representación con Matplotlib.

Podemos pintar varias líneas (y otros elementos) en el mismo gráfico:

```
#Example 4
#Plotting multiple lines with legend
max_val = 5.
t = np.arange(0., max_val+0.5, 0.5)
plt.plot(t, t, 'r-', label='linear')
plt.plot(t, t**2, 'b--', label='quadratic')
plt.plot(t, t**3, 'g-.', label='cubic')
plt.plot(t, 2**t, 'y:', label='exponential')
plt.xlabel('I am x axis')
plt.ylabel('I am y axis')
plt.title('Example 4')
plt.legend()
plt.axis([0, max_val, 0, 2**max_val])
plt.show()
```



Representación con Matplotlib.

Podemos pintar varios gráficos en una misma figura:

```
plt.title('Example 5')
max_val = 5.
t = np.arange(0., max_val+0.5, 0.5)
ax = plt.subplot("211")      #Crear una figura de 2 puestas una sobre otra
ax.set_title("Linear")
ax.plot(t, t, 'r-')
ax.set_ylabel('I am y axis')
ax.axis([0, max_val, 0, max_val])

ax = plt.subplot("212")      #Crear 2º figura
ax.set_title("Quadratic")
ax.plot(t, t**2, 'b--')
ax.axis([0, max_val, 0, max_val**2])

plt.tight_layout()           #Para que deje margenes entre las 2 figuras
plt.show()
```

Representación con Matplotlib.

ax = **plt.subplot("ijk")**:

- i: Número de filas de figuras.
- j: Número de columnas de figuras.
- k: Identificador de la figura a pintar (para determinar la posición).