Numpy aplicado a Machine Learning

Numpy applied to Machine Learning

Autor: Julian Giraldo C
J. David Álvarez

IS&C, Universidad Tecnológica de Pereira, Pereira, Colombia
Correo-e: julian.giraldo2@utp.edu.co
j.alvarez2@utp.edu.co

***Resumen*— Este documento presenta una forma de aplicar Machine Learning mediante el lenguaje de programación Python, utilizando la librería Numpy y sus funciones con un enfoque directo al manejo de matrices y vectores bidimensionales. Primero se hace una introducción del manejo de las mismas y luego se va estudiando más a fondo incluyendo un ejercicio de la vida cotidiana en el manejo de datos grandes, también se explicará la graficación utilizando Numpy.**

***Palabras clave*— python, funciones, matrices, machine learning, programación, procesamiento de datos, archivos.**

***Abstract*— This document presents a way of applying Machine Learning through the Python programming language, using the Numpy library and its functions with a direct approach to handling two-dimensional matrices and vectors. First, an introduction to the management of them is made and then it is studied more in depth including an exercise of daily life in the management of large data, the graphing using Numpy will also be explained.**
***Key Words*— python, functions, arrays, machine learning, programming, data processing, files.**

I. INTRODUCCIÓN

Machine Learning es una disciplina científica del ámbito de la Inteligencia Artificial que crea sistemas que aprenden automáticamente. Aprender en este contexto quiere decir identificar patrones complejos en millones de datos. La máquina que realmente aprende es un algoritmo que revisa los datos y es capaz de predecir comportamientos futuros. Automáticamente, también en este contexto, implica que estos sistemas se mejoran de forma autónoma con el tiempo, sin intervención humana.

Numpy es actualmente la mejor manera de trabajar con Arrays, Vectores y Matrices, en Python. Es una de las librerías más eficientes en CPU para hacer operaciones con matrices y convertir Python en algo más parecido a Matlab y acelerar Python al máximo.

II. OBJETIVOS

- Repasar conceptos básicos de Numpy.
- Resolver un proyecto básico de Machine Learning.
- Manejo de datos.
- Graficación de datos.

III. METODOLOGIA

Primero repasaremos distintas funciones para el manejo de arreglos haciendo uso de la librería numpy. Para esto la importamos y la nombramos como np para abreviar.

```
import numpy as np
```

se crea un arreglo “a” de seis elementos “0,1,2,3,4,5”

```
a = np.array ([0,1,2,3,4,5])
```

con la instrucción ***np.array*** creamos nuestro arreglo y lo guardamos la variable “a”.

con la instrucción ***ndim*** podemos saber el número de dimensiones de nuestro arreglo.

```
print(a.ndim)
```

Con esta simple instrucción mostramos en pantalla el número de dimensiones de nuestro arreglo, siendo "a" un arreglo con los elementos "0,1,2,3,4,5", inferimos fácilmente que el número de dimensiones de "a" es 1, pero en casos en los que no conocemos el número de dimensiones de un arreglo la función ***ndim*** será de gran ayuda.

La instrucción ***shape*** nos permite conocer el numero de elementos de un array.

```
print(a.shape)
```

Esta instrucción nos mostrara en pantalla "6", recordemos que "a" es nuestro arreglo con los elementos "0,1,2,3,4,5", ***a.shape*** nos retorna la cantidad de elementos de "a".

También podemos cambiar la estructura de nuestro arreglo con la función ***reshape(#filas, #columnas).***

```
b=a.reshape(3,2)
```

lo que hacemos con esta instrucción es que a la variable "b" le estamos asignando los valore de "a" y a estos los estamos reestructurando con función ***reshape*** de modo que el array "b" tendrá 3 filas y dos columnas con los elementos de "a":

```
>>  [[0 1]
    [2 3]
    [4 5]]
```

Para modificar algún elemento en este array bidimensional podemos hacer lo siguiente:

Tenemos:

```
[[0 1]
[2 3]
[4 5]]
```

Para modificar el elemento de la segunda fila y la primera columna (2) solo debemos hacer lo siguiente.

```
b[1][0]=nuevo elemento a remplazar
```

debemos tener en cuenta que la primera fila es la fila "0" y la primera columna es la columna "0".

Si. ***Nuevo elemento a remplazar = 77***

```
>>  [[0 1]
    [77 3]
    [4 5]]
```

Podemos aplicar operaciones a todos los elementos de un arreglo con una simple instrucción.

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

Si. ***d=d*2***

```
Print(d)
```

```
>>  [2,4,6,8,10]
```

De esta sencilla manera podemos multiplicamos por 2 a cada uno de los elementos del arreglo "d".

Para controlar lo valores erróneos usamos la función NAN.
Con.

```
c = ([1,2, np.NAN,3,4)

print(c)

>>   [ 1. 2. nan 3. 4.]
```

Para saber si existen números erróneos en nuestro arreglo usamos ***isnan(arreglo)***

```
print(np.isnan(c))

>>   [False False True False False]
```

Para calcular el promedio de valores que no son nan.

```
print(np.mean(c[~np.isnan(c)]))

>>   2.5
```

Ahora, con las diferentes funciones que hemos utilizado para el manejo de arreglos, podemos aplicar diferentes funciones a un proyecto de machine learning:

Una empresa vende el servicio de proporcionar algoritmos de aprendizaje automático a través de HTTP. Con el éxito creciente de la empresa, aumenta la demanda de una mejor infraestructura para atender todas las solicitudes web entrantes. No queremos asignar demasiados recursos, ya que sería demasiado costoso. Por otro lado, perderemos dinero si no hemos reservado suficientes recursos para atender todas las solicitudes entrantes. Ahora, la pregunta es, ¿cuándo alcanzaremos el límite de nuestra infraestructura actual, que se estima en 100.000 solicitudes por hora? Nos gustaría saberlo de antemano cuando tenemos que solicitar servidores adicionales en la nube para atender todas las solicitudes con éxito sin pagar por las no utilizadas.

Entonces, nos enfrentamos con un problema básico de manejo de datos para un análisis estadístico, vamos a desarrollar un programa de machine learning, primero que nada, utilizaremos una instrucción de la librería Numpy que nos permite extraer el archivo de texto de los registros de las solicitudes por hora, esta instrucción tiene la siguiente arquitectura.

```
data = np.genfromtxt("web_traffic.tsv",
delimiter="\t")
```

Creamos la variable data para almacenar los diferentes datos que se obtendrán luego de aplicar esa instrucción, utilizamos la función genfromtxt de la librería Numpy, y esta requiere de dos valores de entrada, primero está el archivo con su formato .tsv y luego está un delimitador que vendría siendo \t para texto.

Archivos TSV

Los archivos con la extensión TSV son archivos de texto que pueden exportarse e importarse usando programas de hojas de cálculo. Se usan generalmente para información en bruto, lo que corresponde a datos que aún no han sido procesados para su uso en otras aplicaciones. Se pueden abrir y leer con cualquier software de edición de texto.

En el momento en que se obtienen los datos por medio de la función anterior, se crean 2 columnas, la primera son las horas y la segunda columna es el número de tareas ejecutadas.

Se muestran los datos en orden de 10 con la instrucción:

```
Print (data [:10], '\n')
```

Los datos se muestran de la siguiente manera gracias a esta instrucción.

```
[[1.000e+00 2.272e+03]
 [2.000e+00       nan]
 [3.000e+00 1.386e+03]
 [4.000e+00 1.365e+03]
 [5.000e+00 1.488e+03]
 [6.000e+00 1.337e+03]
 [7.000e+00 1.883e+03]
 [8.000e+00 2.283e+03]
 [9.000e+00 1.335e+03]
 [1.000e+01 1.025e+03]]
```

Se puede observar los valores en valor entero con una parte decimal y un exponencial, también se observa que hay un valor extraño “nan” no tiene un orden numérico así que esto puede generar problemas para un procesamiento de datos.

Luego de tener toda la información del contenido del archivo en la variable data, podemos averiguar el tamaño de dicho contenido y el número de columnas que tiene con la siguiente instrucción.

```
Print (data.shape)
```

Esta instrucción arroja el siguiente resultado:

```
(743, 2)
```

El número 743 indica el total de filas que hay en la variable data, y el número 2 indica el total de columnas.

Ahora bien, para facilitar el manejo de dichos datos lo mejor que se puede hacer es dividir el vector en 2. Uno que represente toda la Columna 1 y otro toda la columna 2.

```
x = data [:,0]

y = data [:,1]
```

Luego de aplicar esta instrucción, los nuevos vectores quedarían de esta forma.

```
[ 1.  2.  3.  4.  5.  6.  7.  8.  9. 10. 11. 12.
13. 14.15. 16. 17. 18. 19. 20. 21. 22. 23.
24. 25. 26. 27. 28.29. 30. 31. 32. 33. 34.
35. 36. 37. 38. 39. 40. 41. 42.43. 44. 45.
46. 47. 48. 49. 50. 51. 52. 53. 54. 55.
56.57. 58. 59. 60. 61. 62. 63. 64. 65. 66.
67. 68. 69. 70.71. 72. 73. 74. 75. 76. 77.
78. 79. 80. 81. 82. 83. 84.85. 86. 87. 88.
89. 90. 91. 92. 93. 94. ... 743 ]
```

Para ver todos los valores, puede visualizar el documento de jupyter.

Este arreglo representa la columna 1 mencionada anteriormente.

```
[2272. nan 1386. 1365. 1488. 1337. 1883.
2283. 1335. 1025. 1139. 1477.1203. 1311.
1299. 1494. 1159. 1365. 1272. 1246. 1071.
1876. nan 1410. 925. 1533. 2104. 2113.
1993. 1045. 2090. 2227. 1413. 1718. 1721.
1291.1838. 2540. 1608. 2455. 1929. 1767.
1203. 1761. 1723. 2160. 808. nan
1324. 1809. 1933. 1351. 2013. 1207. 2170.
1700. 1899. 1757. 1475. 1921. ... 4881]
```

Para ver todos los valores, puede visualizar el documento de jupyter.

Este arreglo representa el número de tareas ejecutadas.
Ahora se desea saber la dimensión de los dos vectores para verificar que sean simétricos y no tengan un valor de más.

```
Print (x.ndim, '\n')
Print (y.ndim, '\n')
```

Luego de ejecutar esta instrucción, el valor que nos muestra es 1 en ambos. Así que son de dimensión 1.

```
Print (x.shape, '\n')
Print (y.shape)
```

Con x.shape podemos saber los elementos contenidos en los vectores sin necesidad de mostrarlos todos y contarlos uno por uno. Así mismo, en x y en y los elementos son 743, entonces, tienen el mismo número en ambas columnas.

Mientras se hizo el análisis de cada vector con los "Print" aparecen unos valores desconocidos "nan", entonces hay que saber la forma para detectar estos valores automáticamente e identificarlos para poder evitar erorres en el momento de los procesos estadísticos.

```
Print (np.sum (np.isnan (y)))
```

Primero que nada, la instrucción np.isnan lo que hace es saber cuál valor del vector (y) es un nan y así tenerlo en cuenta, Luego de tener en cuenta los valores nan, la instrucción np.sum entra a hacer la suma de todos los valores nan encontrados. En este caso el valor que devuelve es el ***8.***

En el vector (x) también se puede aplicar dicha instrucción pero se sabe que el valor mostrado será 0 ya que no hay valores nan en dicho vector.

Para eliminar los valores erróneos "nan" procedemos a hacer un control de los dos vectores para que no queden asimétricos en algún momento del procedimiento. Primero, mostramos el número de elementos en el vector (x) y (y).

```
Print (x.shape, '\n')
Print (y.shape, '\n')
```

El valor que muestra ese en ambos (743)

Sabemos que entre esos 743 elementos el vector hay 8 que pueden generar problemas así que es necesario expulsarlos de allí. Con la siguiente instrucción podemos hacer lo planteado anteriormente de forma automática.

```
x = x [~np.isnan(y)]
y = y [~np.isnan(y)]
```

Se utiliza la anterior instrucción "np.isnan" para identificar los valores nan pero ahora se le agrega una negación "**~**" para así invertir la lógica. Ahora no es que me seleccione los que son de tipo "nan" sino que me seleccione todos los números menos los "nan", y luego de hacer esa operación lógica se agregan todos los números en los vectores (x) y (y). En la parte de (x) se utiliza la operación lógica de (y) ya que en (x) no existen valores nan y es fundamental dejar los dos vectores simétricos para los procesos estadísticos.

Ahora mostramos el número de elementos de los dos vectores como se hizo anteriormente y el resultado es, (735) para cada vector. Se sabe que habían 8 valores "nan" y los elementos totales eran (743) y si restamos (743-8) efectivamente el resultado es (735).

Graficación de datos

Para graficar los datos que tenemos en los dos vectores, tenemos que importar la librería ***matplotlib*** con la siguiente instrucción.

```
import matplotlib.pyplot as plt
```

Luego de tener la librería ***matplotlib*** se procede a dibujar los puntos en la gráfica, indicando el tamaño y la cantidad.

```
plt.scatter(x, y, s=10)
```

En esta instrucción ***plt.scatter*** se introduce el vector (x) y el vector (y) y el tamaño 10 para los círculos que se encontrarán en el gráfico.

```
plt.title ("Tráfico Web del último mes")
plt.xlabel ("Tiempo")
plt.ylabel ("Solicitudes/Hora")
```

Con estas tres funciones se establece el título del gráfico, el nombre del eje x y el nombre del eje y.

```
plt.xticks([w*7*24 for w in range(10)],
['semana %i' % w for w in range(10)])
```

Con esta instrucción se establecen los puntos de medición en el eje x con un for para hacer varios ciclos hasta llegar a un límite que en este caso el limite seria el total de horas, ya que queremos dividir todas esas horas en semanas, entonces por eso se encuentra esta operación ***w*7*24,*** 7 representa los días de la semana y 24 las horas en un día. El ***range(10)*** Es para establecer cuantas semanas se pondrán en dicho gráfico.

Luego de establecer la información en todos los ejes, se procede a hacer un auto escalamiento para que el gráfico tenga una apariencia adecuada y se interprete de manera clara.

```
plt.autoscale (tight=True)
```

El último paso para que el gráfico quede terminado, es hacer la cuadricula para que así los datos se vean en orden y no hayan confusiones. La instrucción para hacer la cuadricula es.

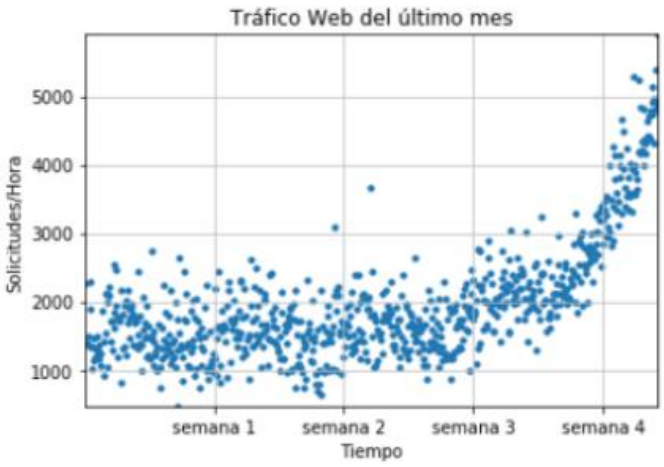
```
plt.grid (True, linestyle='-',
color='0.75')
```

La, Linestyle establece que tipo de línea estará en la cuadricula, el color se establece en código decimal, en este caso el color “0.75” sería un color gris claro.

Por ultimo mostramos la gráfica en pantalla con la instrucción.

```
plt.show()
```

La gráfica quedaría de esta forma.



Los datos quedaron en forma de dispersión, con sus respectivas semanas y solicitudes por hora. Todo esto fue realizado con la librería Numpy para analizar los datos y realizar correcciones si es necesario. Se pueden hacer demasiadas cosas con Numpy y Matplot. Procesamiento de vectores, matrices. Y desde aquí son las bases de Machine Learning para predecir sucesos mediante la estadisitica.

REFERENCIAS

Referencias en la Web:

[1] https://cleverdata.io/que-es-machine-learning-big-data/
[2] https://medium.com/@jcrispis56/machine-learning-parte-2-numpy-fe068abe5703
[3] https://techlandia.com/archivo-tsv-info_292135/