Análisis Machine Learning en Python

Machine Learning Analysis in Python

Autor: Edisson Andres Montes, Alejandro Ríos
IS&C, Universidad Tecnológica de Pereira, Pereira, Colombia
Correo-e: edisson.montes@utp.edu.co, alejandro.rios@utp.edu.co

***Resumen*— Este documento presenta un resumen de las diferentes funcionalidades que nos provee el lenguaje de programación Python para el tratamiento de datos y base de la implementación de machine learning, aquí se detalla cada línea de la forma más clara y sencilla posible que permita entender el funcionamiento de diferentes librerías de Python como numpy y matplotlib, así también se presenta un pequeño ejemplo documentado de una tratamiento de datos para machine learning con el objetivo de visualizar la aplicación de las diferentes librerías de Python.**

***Palabras clave*— numpy, machine learning, matplotlib, programación, Python, tratamiento de datos.**

***Abstract*— This document presents a summary of the different functionalities provided by the Python programming language for data processing and the basis for the implementation of machine learning, here each line is detailed in the clearest and simplest way possible that allows us to understand the operation of different Python libraries such as numpy and matplotlib, as well as a small documented example of a data treatment for machine learning with the aim of visualizing the application of the different Python libraries.**

***Key Word*— data processing, numpy, machine learning, matplotlib, programming, Python.**

I. INTRODUCCIÓN

Machine Learning: Machine Learning o Aprendizaje automático hace referencia a la capacidad de una máquina o software para aprender mediante la adaptación de ciertos algoritmos de su programación respecto a cierta entrada de datos en su sistema. Es decir, la máquina no aprende por sí misma, sino un algoritmo de su programación, que se modifica con la constante entrada de datos en la interfaz, y que puede, de ese modo, predecir escenarios futuros o tomar acciones de manera automática según ciertas condiciones. Como estas acciones se realizan de manera autónoma por el sistema, se dice que el aprendizaje es automático, sin intervención humana.

Python y el Machine Learning: Python es un lenguaje de programación interpretado que busca desarrollar una sintaxis que priorice la legibilidad del código. Este lenguaje de programación es conocido como multiparadigma ya que soporta diferentes orientaciones. En Python podrás orientar el código a objetos, a programación imperativa y funcional. El machine learning en Python es la forma más popular de desarrollar estos modelos. Desde su lanzamiento en 1991 el lenguaje de programación Python ha tenido un crecimiento asombroso, convirtiéndose en uno de los preferidos por los programadores y con el auge de la inteligencia artificial tomo el liderato al ser el lenguaje de programación más utilizado para el desarrollo de este tipo de proyectos.

II. DESCRIPCION DE CODIGO

Primero se debe importar la librería numpy la cual es quien nos facilita la gestión de las listas en Python. Esta librería es muy útil a la hora de trabajar con machine learning debido a todas las funciones que trae consigo; para una fácil implementación usaremos un alias(np) que nos permita hacer el llamado a sus funciones de manera mas cómoda.

```
import numpy as np
```

Creamos un array de seis (6) elementos (0, 1, 2, 3, 4, 5) e imprimimos este array para visualizar los datos; a este vector le asignamos la variable a. Utilizamos la función **dim** para conocer la dimensión del array y finalmente se utiliza **shape** que nos facilita la cantidad de elementos que hay dentro del vector.

```
a = np.array([0,1,2,3,4,5])
print(a, '\n')
print(a.ndim, '\n')
print(a.shape)
```

Luego utilizamos la función **reshape** para cambiar la estructura del array, la cual antes era de una (1) dimensión y ahora tendrá dos (2); eso sí, esta nueva reestructuración la guardamos en una nueva variable. Como la nueva estructura tiene dos (2) dimisiones, cada dimensión tendrá tres (3) elementos. Imprimimos el nuevo array para ver su contenido y verificar la reestructuración.

```
b =a.reshape((3,2))
print(b, '\n')
print(b.ndim, '\n')
print(b.shape)
```

Modificaremos el primer elemento de la segunda fila del array b y le asignaremos el valor setenta y siente (77) e imprimimos el array para mirar el cambio de esta manera.

```
b[1][0] = 77
print(b)
```

NOTA: Debido a que el array 'b' se construyó con base en el array a, el cambio afecta también al array 'a', por lo que será importante verificar su contenido.

Para evitar que se modifique el array original se utiliza la función **reshape(()).copy()** y lo guardamos en una nueva que llamaremos 'c'. Ahora si se modifica el primer valor de la primera columna de c y le asignamos a este campo el número menos noventa y nueve (-99) el array original no se verá afectado; finalmente imprimimos ambos arrays para verificar que el cambio solo se efectuó en el array deseado.

```
c = a.reshape((3,2)).copy()
print(c, '\n')
c[0][0] = -99
print(a, '\n')
print(c)
```

En algunos casos necesitamos realizar una operación a todos los elementos de nuestro array, para esto vasta con realizar la operación necesaria (suma, restas, etc) entre la variable y el valor que deseemos

```
d = np.array([1,2,3,4,5])
print(d*2, '\n')
```

Para esta operación obtendremos el resultado: [2 4 6 8 10].
En el caso de querer elevar los valores a algún índice vasta con usar la notación de doble multiplicación (**)

```
print(d**2)
```

Ahora para realizar operaciones de comparación, definimos de nuevo un array 'a' y le asignamos los valores 1, 2, 3, 4, 5. Ahora se imprime el array, pero este devolverá 'true' o 'false' dependiendo de la condición que implementemos, para este caso serán todos los valores mayores a 4. Esto se hace a partir de operadores lógicos (>, <, =, etc).

```
a = np.array([1,2,3,4,5])
print(a>4, '\n')
```

Esto nos permitirá hacer cambios en el array a partir de una condición que queremos plasmar, esto lo logramos usando:

```
a[a>4] = 1
print(a, '\n')
```

Ahora de igual manera podremos realizar cambios a partir del nuevo resultado, por ejemplo, cambiar los valores uno (1) por 777.

```
a[a==1] = 777
print(a, '\n')
```

Así nuestro resultado final será [777 2 3 4 777]
La función de numpy **isnan** no permite conocer si dentro de un array se encuentran datos de este tipo (nan o valores erróneos), esta función nos devuelve un array de valores booleanos que nos permite identificar que posiciones dentro del array son nan

```
c = np.array([1, 2, np.NAN, 3, 4])
print(c, 'n')
print(np.isnan(c), '\n')
```

Ahora bien, también podremos hacer uso del símbolo de negación '~', para filtrar los datos de un array, solo conservar los datos que no son erróneos o tipo nan de esta manera:

```
print(c[~np.isnan(c)], '\n')
```

Como complemento la función **mean** nos permite calcular el promedio de los valore de un array, asi complementando las funcionabilidades de nan, podemos calcular el promedio del array luego de filtrar los datos nan que no permitirían calcular el promedio si no son descartados.

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

III. IMPLEMENTACION PARA MACHINE LEARNING

III.I. Planteamiento del problema:

"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."

Para el desarrollo de este sistema, debemos poseer en primera instancia el conjunto de datos que deseamos analizar, para esto tomaremos un archivo (.tsv) que contiene el numero de solicitudes web entrantes de cada hora.

Para su debida manipulación extraeremos los datos de este documento en un array 'data' que poseerá en una columna las horas y en otra el numero de solicitudes, con la ayuda de la función **genfromtxt** de numpy y la función **shape** para confirmar el numero de datos.

```
data = np.genfromtxt("web_traffic.tsv",
delimiter="\t")
print(data[:10], '\n')
print(data.shape)
```

Luego dividimos el array anterior en dos vectores independientes que llamaremos **x, y**; y los imprimiremos para ver el resultado, esto apoyados de la sintaxis [:, "numero"] que nos permitirá elegir el rango de valores, en el caso del vector **x** quien serán todos los datos de la columna 0 y en **y** todos los datos de la columna 1.

```
x = data[:,0]
y = data[:,1]
print(x, '\n')
print(y, '\n')
```

De manera comprobativa mostraremos la dimensión y el número de datos en los vectores, con las funciones **ndim** y **shape** anteriormente explicadas.

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

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

Luego podremos hacer uso de la función isnan() para conocer los valores nan que tiene el vector **y**.

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

Posteriormente luego de conocer si poseemos números nan en nuestro vector, deberemos eliminar estos datos con el fin de realizar el correcto análisis; y comprobaremos su eliminación comparando el tamaño de los vectores antes y después de la eliminación

```
print(x.shape, '\n')
print(x.shape, '\n')

x = x[~np.isnan(y)]
y = y[~np.isnan(y)]
print(x.shape, '\n')
print(x.shape, '\n')
```

Finalmente, realizaremos el proceso de graficación para esto realizaremos lo siguiente: Importaremos una nueva librería llamada **matplotlib** la cual nos permitirá graficar, asignaremos un alias para su uso, Esta se importa mediante:

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

Ahora graficaremos **X** y **Y** mediante unos círculos de tamaña diez (10) con la función:

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

Luego asignaremos a la grafica un titulo y nos labels para el eje **X** (espacio en el tiempo) y **Y** (número de solicitudes en una hora):

```
plt.scatter(x, y, s=10)
plt.title("Tráfico Web del último mes")
plt.xlabel("Tiempo")
plt.ylabel("Solicitudes/Hora")
```

Seguidamente se hace una división de la grafica dependiendo la cantidad de datos y agregamos los putos según corresponda, esto con el fin de agrupar los datos en semanas, eso se hace a partir de la función **ticks()**, a partir de la operación (w*7*24) agruparemos los datos que se encuentran en horas en un día(24h) y luego en una semana(7dias).

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

Ya para finalizar se dibuja una cuadrícula punteada ligeramente opaca con la función **grid()** y mostramos en pantalla el resultado final mediante la función **show().**

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

Como resultado obtendremos una grafica con los datos debidamente agrupados en semanas y que nos permite ver el flujo y crecimiento de los datos a través del tiempo

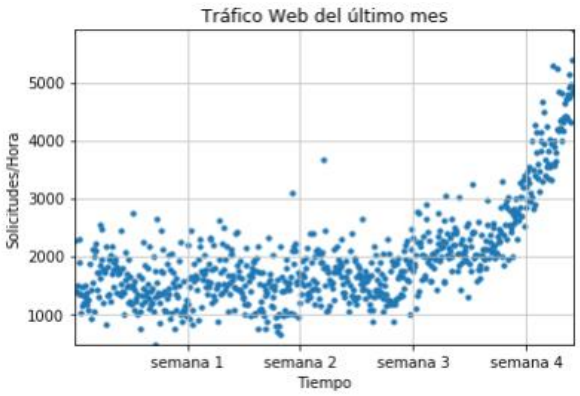

Figura 1. Graficación de datos

REFERENCIAS

Referencias en la Web:

[1]
https://medium.com/datos-y-ciencia/curso-de-machine-learning-preparar-un-entorno-python-para-machine-learning-fda495449a87
[2]
https://www.grapheverywhere.com/machine-learning-en-python/