05-aplicar_numpy_ufunc_sobre_objetos_pandas

May 14, 2021

1 Aplicación de funciones universales con Pandas.

Pandas utiliza el paquete Numpy para representar internamente los datos de una serie o de un dataframe. Por lo tanto es de esperar que se pueda aplicar las funciones universales de Numpy sobre los objetos Panda Series y DataFrame para realizar un procesado eficiente de datos.

1.1 Inicialización del entorno.

Lo primero será importar el paquete Pandas con alias pd. Posteriormente visualizamos la versión usada de Pandas ya que es un dato importante para consultar la documentación. En el momento de editar este notebook la versión de pandas es: 0.25.3

Además para facilitar los ejercicios también importamos el paquete Numpy con el alias np.

[13]:
```
import pandas as pd
import numpy as np
np.set_printoptions(floatmode='fixed', precision=3)
print('Pandas versión: ', pd.__version__)
```

```
Pandas versión:  0.25.3
```

1.2 Utilizar funciones universales de Numpy con objetos Panda.

1.2.1 Funciones universales con un argumento.

Podemos utilizar la mayoría de las funciones universales del paquete Numpy con un argumento usando un objeto Pandas Series o DataFrame como argumento.

Al aplicar una función universal a un objeto Series o DataFrame se obtiene otro objeto Panda del mismo tipo y con los mismos índices.

Ejercicio 01: Aplicar la ufunc np.sqrt a una serie. Comprueba que los índices no se afectan.

La salida debería ser algo parecido a lo siguiente:

[14]:
```
np.random.seed(0)
s1 = pd.Series (np.random.randint(0, 10, 4), index=['A', 'B', 'C', 'D'])
print('Serie original:\n', s1)
s2 = np.zeros_like(s1)
#Pon tu código aquí.
#Sugerencia: usa numpy.sqrt con la serie como si fuera un ndarray.
```

```
#
print('\nsqrt(s1)=\n', s2)
```

```
Serie original:
 A    5
B    0
C    3
D    3
dtype: int64

sqrt(s1)=
 [0 0 0 0]
```

Ejercicio 02: Aplicar la ufunc np.sqrt() a un dataframe. Comprueba que los índices no se afectan.

La salida debería ser algo parecido a lo siguiente:

[15]:
```
np.random.seed(0)
df1 = pd.DataFrame(dict(zip(list('abcde'),
                            [np.random.randint(10, size=5) for i in␣
 ↪range(5)])))
print('Dataframe original:\n', df1)
df2 = np.zeros_like(df1)
#Pon tu código aquí.
#Sugerencia: usa numpy.sqrt con la serie como si fuera un ndarray.

#
print('\nsqrt(df1)=\n', df2)
```

```
Dataframe original:
    a  b  c  d  e
0  5  9  7  6  5
1  0  3  6  7  9
2  3  5  8  7  8
3  3  2  8  8  9
4  7  4  1  1  4

sqrt(df1)=
 [[0 0 0 0 0]
 [0 0 0 0 0]
 [0 0 0 0 0]
 [0 0 0 0 0]
 [0 0 0 0 0]]
```

1.2.2 Funciones universales con dos argumentos usando un valor escalar como uno de ellos.

Podemos utilizar la mayoría de las funciones universales del paquete Numpy con dos argumentos, usando un objeto Pandas Series o DataFrame como argumento y una escalar como el otro.

Al aplicar una función universal de esta forma a un objeto Series o DataFrame se obtiene otro objeto Panda del mismo tipo y con los mismos índices.

Ejercicio 03: Aplicar la ufunc np.add a una serie para sumar 1.0 a todos los valores. Comprueba que los índices no se afectan.

La salida debería ser algo parecido a lo siguiente:

[16]:
```
np.random.seed(0)
s1 = pd.Series (np.random.randint(0, 10, 4), index=['A', 'B', 'C', 'D'])
print('Serie original:\n', s1)
s2 = np.zeros_like(s1)
#Pon tu código aquí.
#Sugerencia: usa numpy.add con la serie como si fuera un ndarray
#y la escalar 1

#
print('\ns1+1 =\n', s2)
```

```
Serie original:
 A    5
B    0
C    3
D    3
dtype: int64

s1+1 =
 [0 0 0 0]
```

Ejercicio 04: Aplicar la ufunc np.divide() a un dataframe para dividir el valor entre 2 de forma entera. Comprueba que los índices no se afectan.

La salida debería ser algo parecido a lo siguiente:

[17]:
```
np.random.seed(0)
df1 = pd.DataFrame(dict(zip(list('abcde'),
                            [np.random.randint(10, size=5) for i in␣
 ↪range(5)])))
print('Dataframe original:\n', df1)
df2 = np.zeros_like(df1)
#Pon tu código aquí.
#Sugerencia: usa numpy.floor_divide con la serie como si fuera un ndarray.

#
print('\n df1 // 2=\n', df2)
```

```
Dataframe original:
    a  b  c  d  e
0  5  9  7  6  5
1  0  3  6  7  9
2  3  5  8  7  8
3  3  2  8  8  9
4  7  4  1  1  4

 df1 // 2=
 [[0 0 0 0 0]
 [0 0 0 0 0]
 [0 0 0 0 0]
 [0 0 0 0 0]
 [0 0 0 0 0]]
```

1.2.3 Funciones universales con dos argumentos usando dos objetos Pandas.

En las funciones universales que representan operaciones binarias usando dos objetos Series o DataFrame, un paso previo a realizar la operación es alinear los ejes usando las etiquetas de los mismos (ver esta referencia).

Ejercicio 05: Aplicar la ufunc np.add para obtener una serie como suma de otras dos series s1, s2. Comprueba que los índices se alinean antes de realizar la operación.

La salida debería ser algo parecido a lo siguiente:

[18]:
```
np.random.seed(0)
s1 = pd.Series (np.random.randint(0, 10, 4),
                index=['A', 'B', 'C', 'D'])
s2 = pd.Series (np.random.randint(0, 10, 4),
                index=['C', 'A', 'D', 'B'])
print('s1:\n', s1)
print('s2:\n', s2)
s3 = np.zeros_like(s1)
#Pon tu código aquí.
#Sugerencia: usa numpy.add con la serie como si fuera un ndarray
#y la escalar 1

#
print('\ns1+s2 =\n', s3)
```

```
s1:
 A    5
B    0
C    3
D    3
dtype: int64
s2:
 C    7
```

```
A    9
D    3
B    5
dtype: int64

s1+s2 =
 [0 0 0 0]
```

Si se aplica la operación a dos objetos DataFrame, `df1`, `df2`, para que los ejes se alineen correctamente, es necesario aplicar la función de la forma `df1.`*`operaciónn`*`(df2)`, donde *operación* debe ser una operación de las siguientes:

- df1.add(df2), o df1+df2.
- df1.subtract(df2), o df1-df2
- df2.multiply(df2), o df1*df2
- df1.truediv(df2), df1.divide(df2) o df1 / df2
- df1.floordiv(df2) o df1 // df2
- df1.mod(df2) o df1 % df2
- df1.pow(df2) o df1 ** df2

Además usar la operación, por ejemplo `subtract()` en lugar del operador `-`, permite especificar parámetros adiconales que se verán más adelante.

Ejercicio 06: Aplicar la ufunc DataFrame.add para obtener un dataframe como la suma de otros dos dataframes df1, df2, cuyos ejes no están alineados. Comprueba que los índices se alinean antes de realizar la operación.

La salida debería ser algo parecido a lo siguiente:

[19]:
```
np.random.seed(0)
df1 = pd.DataFrame(np.random.randint(10,size=(3,4)),
                   index=list('ABC'), columns=list('abcd'))
df2 = pd.DataFrame(np.random.randint(10,size=(3,4)),
                   index=list('BAC'), columns=list('dabc'))
print('df1:\n', df1)
print('df2:\n', df2)
df3 = np.zeros_like(df1)
#Pon tu código aquí.
#Sugerencia: Contempla usar el operador + o el formato df.add()

#
print('\ndf1+df2=\n', df3)
```

```
df1:
    a  b  c  d
A  5  0  3  3
B  7  9  3  5
C  2  4  7  6
df2:
    d  a  b  c
```

```
B  8  8  1  6
A  7  7  8  1
C  5  9  8  9

df1+df2=
 [[0 0 0 0]
 [0 0 0 0]
 [0 0 0 0]]
```

Ejercicio 07: Aplicar la ufunc numpy.add para obtener un dataframe como la suma de otros dos. Comprobar que si se aplica directamente una operación univesal Numpy a dos objetos DataFrame los ejes no se alinean.

La salida debería ser algo parecido a lo siguiente:

[20]:
```
np.random.seed(0)
df1 = pd.DataFrame(np.random.randint(10,size=(3,4)),
                   index=list('ABC'), columns=list('abcd'))
df2 = pd.DataFrame(np.random.randint(10,size=(3,4)),
                   index=list('BAC'), columns=list('dabc'))
print('df1:\n', df1)
print('df2:\n', df2)
df3 = np.zeros_like(df1)
#Pon tu código aquí.
#Sugerencia: usa la ufunc np.add.

#
print('\ndf1+df2=\n', df3)
```

```
df1:
    a  b  c  d
A  5  0  3  3
B  7  9  3  5
C  2  4  7  6
df2:
    d  a  b  c
B  8  8  1  6
A  7  7  8  1
C  5  9  8  9

df1+df2=
 [[0 0 0 0]
 [0 0 0 0]
 [0 0 0 0]]
```

1.2.4 Gestión de valores perdidos.

Cuando se alinean los ejes, puede ocurrir que para uno de los argumentos falte un dato. En general este dato será completado con el valor 'NaN' antes de operar.

Ejercicio 08: Aplicar la operación '+' para obtener una serie como suma de otras dos series s1, s2. Comprueba que los índices se alinean antes de realizar la operación y cómo se rellena con el NaN los valores perdidos.

La salida debería ser algo parecido a lo siguiente:

[21]:
```
np.random.seed(0)
s1 = pd.Series (np.random.randint(0, 10, 3),
                index=['A', 'B', 'C'])
s2 = pd.Series (np.random.randint(0, 10, 3),
                index=['B', 'C', 'D'])
print('s1:\n', s1)
print('s2:\n', s2)
s3 = np.zeros_like(s1)
#Pon tu código aquí.
#Sugerencia: usa el operador '+'.

#
print('\ns1+s2 =\n', s3)
```

```
s1:
 A    5
B    0
C    3
dtype: int64
s2:
 B    3
C    7
D    9
dtype: int64

s1+s2 =
 [0 0 0]
```

Si el valor 'NaN' no es el conveniente para rellenar, podemos usar el método `Series.add()` indicando con el argumento 'fill_value' el valor deseado para rellenar los valores perdidos.

Ejercicio 09: Aplicar la operación '+' para obtener una serie como suma de otras dos series s1, s2. Se desea que se utilice el valor '0' para rellenar valores peridos al alinear los ejes.

La salida debería ser algo parecido a lo siguiente:

[22]:
```
np.random.seed(0)
s1 = pd.Series (np.random.randint(0, 10, 3),
                index=['A', 'B', 'C'])
s2 = pd.Series (np.random.randint(0, 10, 3),
                index=['B', 'C', 'D'])
print('s1:\n', s1)
print('s2:\n', s2)
s3 = np.zeros_like(s1)
```

```
#Pon tu código aquí.
#Sugerencia: usa el método .add() indicando el valor 0 para
#rellenar valores perdidos.

#
print('\ns1+s2 =\n', s3)
```

```
s1:
 A    5
B    0
C    3
dtype: int64
s2:
 B    3
C    7
D    9
dtype: int64

s1+s2 =
 [0 0 0]
```

1.2.5 Combinando Series y DataFrames.

De la misma forma que un array 1d puede combinarse con un array 2d en Numpy aplicando reglas de difusión (*broadcasting*), en Pandas también los objetos Series y DataFrame se pueden combinar en una operación. De nuevo previamente a realizar la operación se realizará un alineado de ejes.

Ejemplo 10: Dado un dataframe `df1` se requiere obtener una versión restando la segunda fila a todo el dataframe.

La salida debería ser algo parecido a lo siguiente:

[23]:
```
df1 = pd.DataFrame(np.random.randint(0, 10, (3,4)),
                   columns=[list('ABCD')])
print("DataFrame original:\n", df1)
df2 = np.zeros_like(df1.shape)
#Pon tu código aquí. las operaciones
#Sugerencia: utiliza iloc para obtener una serie correspondiente
#a la segunda fila y réstala al dataframe.

#
print("DataFrame modificado:\n", df2)
```

```
DataFrame original:
    A  B  C  D
0  3  5  2  4
1  7  6  8  8
2  1  6  7  7
DataFrame modificado:
```

```
 [0 0]
```

Ejemplo 11: Dado un dataframe `df1` se requiere obtener una versión restando la segunda columna a todo el dataframe.

La salida debería ser algo parecido a lo siguiente:

[24]:
```
df1 = pd.DataFrame(np.random.randint(0, 10, (3,4)),
                   columns=[list('ABCD')])
print("DataFrame original:\n", df1)
df2 = np.zeros_like(df1.shape)
#Pon tu código aquí.
#Sugerencia: utiliza iloc para obtener una serie correspondiente
#a la segunda columna y réstala al dataframe. En este caso hay que
#usar el método subtract para poder indicar el eje sobre el que
#aplicar la operación.

#
print("DataFrame modificado:\n", df2)
```

```
DataFrame original:
    A  B  C  D
0  8  1  5  9
1  8  9  4  3
2  0  3  5  0
DataFrame modificado:
 [0 0]
```

[ ]:
```

```