بسمی تعالی

موضوع پروژه: مفاهیم کتابخانه numpy در python

**آرایه در NumPy**

آرایه‌ها در NumPy تاحدودی شبیه لیست در پایتون هستند اما در اصل با آن تفاوت دارند. اجازه بدهید به طور کامل این

موضوع را تشریح کنیم. همان‌طور که از نام این کتابخانه مشخص است، NumPy از Numeric Python برگرفته شده

و امکان کار با اعداد را به طور گسترده فراهم می‌کند و آرایه‌ی NumPy ساختار داده‌ی این کتابخانه است. یعنی برای

اینکه بتوانیم با داده‌ها در این کتابخانه کار کنیم باید آن‌ها را به شکل آرایه تبدیل کرد. در ادامه‌ی مقاله‌ی آموزش کامل

کتابخانه NumPY ، با ایجاد یک آرایه‌ی numpy و عملیات ریاضی بر روی آرایه‌ها آشنا می‌شوید.

**ایجاد یک آرایه‌ی NumPy**

ساده‌ترین روش برای ساخت این آرایه استفاده از لیست پایتون است:

myPythonList = [1,9,8,3]

برای تبدیل لیست پایتون به آرایه‌ی NumPy از دستور np.array استفاده می‌کنیم. (برای چاپ خروجی در ژوپیتر نام

متغیر numpy\_array\_from\_list را تایپ می‌کنیم.)

import numpy as np

numpy\_array\_from\_list = np.array(myPythonList)

numpy\_array\_from\_list

در عمل نیازی به تعریف لیست پایتونی به طور مجزا نیست و می‌توان به شکل زیر آرایه را تعریف کرد:

a = np.array([1,9,8,3])

**توجه**: در مستندات NumPy از np.darray هم برای ایجاد آرایه استفاده شده، اما روش ذکر شده مرسوم ‌تر است.

همچنین به جای لیست می‌توان از تاپل (tuple) نیز استفاده کرد.

**عملیات ریاضی بر روی آرایه‌ها**

عملیات ریاضی همچون جمع، تفریق، ضرب و تقسیم بر روی آرایه‌ها قابل انجام است. دستور کار به صورت استفاده از

نام آرایه و عملگر موردنظر (\*,?,+,-) و عملوند است:

numpy\_array\_from\_list + 10

array([11, 19, 18, 13])

**Broadcasting یا انتشار همگانی در Numpy**

واژه‌ی Broadcasting به چگونگی رفتار Numpy با آرایه‌هایی با Shape متفاوت در خلال عملگرهای محاسباتی

اشاره دارد. به طور خلاصه، آرایه‌ی کوچکتر به اندازه‌ی آرایه‌ی بزرگتر پخش می‌شود تا به شکل و Shape یکسانی با

آن تبدیل شود. اینکه از Broadcast استفاده بکنیم یا خیر؛  به شرایط داده و الگوریتم بستگی دارد و می‌تواند با توجه به

آن‌ها به‌کارگیری این مفهوم در کارایی برنامه تأثیر مثبت یا منفی داشته باشد. Broadcasting در Numpy با توجه به

محدودیت‌های مسأله و داده، کارایی را کنترل می‌کند.

**قانون Broadcasting** : باید اندازه‌ی محور نهایی یا آخر (یا همان مقدار آخرین Shape)در هر دو آرایه‌ داده‌ یکسان

باشد؛ یا مقدار بُعد آخرِ حداقل یکی از آرایه‌ها برابر یک باشد. حالت‌های زیر را در نظر بگیرید:

A(2-D array): 4 x 3    # these numbers are the shape of array

B(1-D array):     3

Result      : 4 x 3

A(4-D array): 7 x 1 x 6 x 1

B(3-D array): 3 x 1 x 5

Result : 7 x 3 x 6 x 5

 اما این یکی قابلیت Broadcasting ندارد. چون نه مقدار بُعد آخرشان یکی است(یکی 3 و دیگری 0) و نه بُعد نهایی

هیچ یک برابر عدد 1 است:

A: 4 x 3

B: 4

ساده‌ ترین حالت Broadcast حین انجام عملیات محاسباتی میان یک آرایه و یک عدد اسکالر است.

import numpy as np

a = np.array([1.0, 2.0, 3.0])

*# Example 1*

b = 2.0

print(a \* b)

*# Example 2*

c = [2.0, 2.0, 2.0]

print(a \* c)

[ 2. 4. 6.] [ 2. 4. 6.]

می‌توان این‌گونه تصور کرد که عدد اسکالر به تعداد اندازه‌ ی آرایه‌ ای که قرار است با آن در محاسبات شرکت کند، کپی

می‌شود. البته این یک توصیف مفهومی از کار است و Numpy با توجه به هوشمندی که در طراحی‌اش به کار رفته،

برای حفظ کارایی عملکرد و حافظه، چنین کپی‌ای از آن عدد اسکالر ایجاد نمی‌کند.

**شکل آرایه (Shape of Array)**

در این بخش از مقاله‌ی آموزش کامل کتابخانه NumPY ، شکل آرایه را توضیح خواهیم داد، که می‌تواند یک بعدی،

دوبعدی، یا سه بعدی باشد. شکل آرایه را می‌توان با دستور shape. بدست آورد و نیز با دستور dtype. می‌توان نوع

داده‌های آرایه و مقدار فضای اختصاص یافته به آن را فهمید.

import numpy as np

a = np.array([1,2,3])

print(a.shape)

print(a.dtype)

#### Different type

b = np.array([1.1,2.0,3.2])

(3, ) int64

float64

**آرایه‌ی دو بعدی**

بعد دوم را می‌توان با افزودن یک کاما "," تعریف کرد. البته توجه کنید که داده‌های پس از کاما هم داخل براکت باز و

بسته [] ( ویا پرانتز () اگر که تاپل باشد ) باید قرار گیرند.

c = np.array([(1,2,3),

(4,5,6)])

print(c.shape)

(2,3)

**آرایه‌ی سه بعدی**

آرایه‌ها با ابعاد بالاتر به شکل زیر تعریف می‌شوند: (دوباره یک کاما و یک براکت باز و بسته ی دیگر افزوده می‌شود.)

d = np.array([

[[1, 2,3],

[4, 5, 6]],

[[7, 8,9],

[10, 11, 12]]

])

print(d.shape)

(3,2,2)

**آرایه با مقادیر صفر و یا با مقادیر یک در NumPy**

می‌توان با دستورات np.zeros و np.ones به ترتیب ماتریس‌هایی ایجاد کرد که تمامی عناصر آن صفر و یک باشند.

چنین ماتریس‌هایی می‌توانند به طور مثال برای مقداردهی اولیه‌ی ماتریس وزن‌ها در شبکه‌ی عصبی مورد استفاده قرار

گیرند. قاعده‌ی کلی به شکل زیر است:

numpy.zeros(shape, dtype=float, order='C')

numpy.ones(shape, dtype=float, order='C')

shape شکل آرایه را تعیین می‌کند و dtype یا همان datatype نوع داده را مشخص می‌کند و پارامتری اختیاری است

که به طور پیش‌فرض مقدار آن float64 است. پارامتر order هم اختیاری است و مقدار پیش‌فرض C را می‌گیرد و

یک استایل پایه برای سطرهای ماتریس است. در زیر چند مثال را می‌بینیم.

*#Example numpy zero*

import numpy as np

np.zeros((2,2))

*#Example numpy zero with datatype*

np.zeros((2,2), dtype=np.int16)

*#Example numpy one 2D Array with datatype*

np.ones((1,2,3), dtype=np.int16)

خروجی‌ها به ترتیب زیر هستند:

array([[0., 0.], [0., 0.]]) array([[0, 0], [0, 0]], dtype=int16) array([[[1, 1, 1], [1, 1, 1]]], dtype=int16)

**numpy.reshape  و  numpy.flatten در پایتون**

گاهی شما نیاز به تغییر (reshape) داده‌ها از فرم عرضی به طولی دارید. برای این کار از تابع reshape به شکل زیر

استفاده می‌کنیم(َa ورودی است)

import numpy as np

e = np.array([(1,2,3), (4,5,6)])

print(e)

e.reshape(3,2)

array([1, 2, 3, 4, 5, 6])

**اتصال آرایه‌ها به هم در NumPy (Stacking)**

چندین آرایه می‌توانند در امتداد محورهای مختلف به همدیگر متصل شوند. که در این بخش از مقاله‌ی آموزش کامل

کتابخانه NumPY ، انواع اتصال آرایه‌ها به هم را توضیح می‌دهیم.

np.vstack: آرایه‌ها را به طور عمودی به هم وصل می‌کند.

np.hstack: آرایه‌ها را به طور افقی متصل می‌کند.

np.column\_stack: برای اتصال آرایه‌ی یک بُعدی به عنوان ستون به سطر آرایه‌ی دوبعدی.

np.concatenate: برای اتصال آرایه‌ها به همدیگر در خلال یک محور معین (یعنی محور یا همان axis به عنوان پارامتر ورودی تابع تعریف می‌شود).

مثال زیر نحوه‌ی کارکرد هر یک را نشان می‌دهد.

import numpy as np

a = np.array([[1, 2],

[3, 4]])

b = np.array([[5, 6],

[7, 8]])

*# vertical stacking*

print("Vertical stacking:\n", np.vstack((a, b)))

*# horizontal stacking*

print("\nHorizontal stacking:\n", np.hstack((a, b)))

c = [5, 6]

*# stacking columns*

print("\nColumn stacking:\n", np.column\_stack((a, c)))

*# concatenation method*

print("\nConcatenating to 2nd axis:\n", np.concatenate((a, b), 1))

Vertical stacking: [[1 2] [3 4] [5 6] [7 8]] Horizontal stacking: [[1 2 5 6] [3 4 7 8]] Column

stacking: [[1 2 5] [3 4 6]] Concatenating to 2nd axis: [[1 2 5 6] [3 4 7 8]]

**تقسیم آرایه در NumPy (Splitting)**

در این بخش از مقاله‌ی آموزش کامل کتابخانه NumPY ، توابعی برای تقسیم آرایه‌ها را معرفی می‌کنیم.

توابعی برای تقسیم آرایه‌ها تعریف شده که به شرح زیر است:

:np.hsplitتقسیم آرایه به شکل افقی.

:np.vsplitتقسیم آرایه به شکل عمودی.

:np.array\_split آرایه را در خلال امتداد تعیین شده در ورودی (با پارامتر axis)تقسیم می‌کند.

import numpy as np

a = np.array([[1, 3, 5, 7, 9, 11],

[2, 4, 6, 8, 10, 12]])

*# horizontal splitting*

print("Splitting along horizontal axis into 2 parts:\n", np.hsplit(a, 2))

*# vertical splitting*

print("\nSplitting along vertical axis into 2 parts:\n", np.vsplit(a, 2))

Splitting along horizontal axis into 2 parts: [array([[1, 3, 5], [2, 4, 6]]), array([[ 7, 9, 11], [ 8, 10, 12]])] Splitting along vertical axis into 2 parts: [array([[ 1, 3, 5, 7, 9, 11]]), array([[ 2, 4, 6, 8, 10, 12]])]

**پیمایش آرایه در NumPy**

پیمایش آرایه یا Iterate کردن آن یعنی عناصر آن را یکی یکی مشاهده کنیم. چون با آرایه‌های چند بعدی در

numpy سروکار داریم، پیمایش را می‌توان با حلقه‌ی for در پایتون انجام داد. آرایه‌ی یک بعدی (1D) را به شکل زیر

پیمایش می‌کنیم:

import numpy as np

arr = np.array([1, 2, 3])

for x in arr:

print(x) 1 2 3 <<<<

در آرایه‌ی دو بعدی (2D) پیمایش به صورت سطر به سطر انجام می‌شود:

import numpy as np

arr = np.array([[1, 2, 3], [4, 5, 6]])

for x in arr:

print(x)

[1 2 3] [4 5 6]

اگر بخواهیم مقادیر به صورت اسکالر برگردانده شود، به شکل زیر عمل می‌کنیم.

import numpy as np

arr = np.array([[1, 2, 3], [4, 5, 6]])

for x in arr:

for y in x:

print(y)

1 2 3 4 5 6

در پیمایش آرایه‌ی سه بعدی (3D) آرایه‌های دوبعدی آن را جدا می‌کنیم:

import numpy as np

arr = np.array([[[1, 2, 3], [4, 5, 6]], [[7, 8, 9], [10, 11, 12]]])

for x in arr:

print(x)

[[1 2 3] [4 5 6]] [[ 7 8 9] [10 11 12]]

**جبرخطی در NumPy**

ماژول جبرخطی در NumPy متدهای بسیاری را برای اعمال روی انواع آرایه‌ها در NumPy فراهم می‌کند. شما با این

ماژول قادر هستید موارد زیر را محاسبه کنید:

* مرتبه، اثر ماتریس (trace of matrix) و دترمینان یک آرایه
* مقادیر ویژه‌ی یک ماتریس (eigen values)
* ضرب‌های ماتریسی و برداری (ضرب نقطه‌ای، داخلی، خارجی و ...)
* حل معادلات خطی یا تانسور

مثال زیر چگونگی اعمال عملیات جبری را نشان می‌دهد.

import numpy as np

A = np.array([[6, 1, 1],

[4, -2, 5],

[2, 8, 7]])

print("Rank of A:", np.linalg.matrix\_rank(A))

print("\nTrace of A:", np.trace(A))

print("\nDeterminant of A:", np.linalg.det(A))

print("\nInverse of A:\n", np.linalg.inv(A))

print("\nMatrix A raised to power 3:\n", np.linalg.matrix\_power(A, 3))

Rank of A: 3 Trace of A: 11 Determinant of A: -306.0 Inverse of A: [[ 0.17647059 -0.00326797 -0.02287582] [ 0.05882353 -0.13071895 0.08496732] [-0.11764706 0.1503268 0.05228758]] Matrix A raised to power 3: [[336 162 228] [406 162 469] [698 702 905]]

تصور کنید می‌خواهیم این معادله‌ی خطی را محاسبه کنیم.

x + 2\*y = 8

3\*x + 4\*y = 18

این معادله با به‌کارگیری متد linalg.solve قابل حل است.

import numpy as np

*# coefficients*

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

*# constants*

b = np.array([8, 18])

print("Solution of linear equations:", np.linalg.solve(a, b))

Solution of linear equations: [ 2.  3.]

**تولید اعداد تصادفی در NumPy**

برای تولید اعداد تصادفی در NumPy به طور کلی از تابع random استفاده می‌کنیم و می‌توانیم برای اینکه این اعداد

تصادفی در یک توزیع خاص آماری با ویژگی‌های مورد نظر ما باشند، از توابع دیگری نیز کمک بگیریم.

:numpy.random.rand(d0,d1,…,dn)آرایه‌ای در ابعاد مشخص ایجاد می‌کند و درایه‌های آن را با اعداد تصادفی پر می‌کند. d0,d1,…,dn ابعاد آرایه را مشخص می‌کنند و اعدادی صحیح و اختیاری‌اند. اگر آرگومانی داده نشود یک مقدار اعشاری تولید می‌کند.

:numpy.random.normal(loc, scale, size)آرایه‌ای در شکل و ابعاد مشخص تولید می‌کند و درایه‌های آن را با مقادیر تصادفی که بخشی از یک توزیع نرمال (گوسی) محسوب می‌شوند، پر می‌کند. به این توزیع به خاطر شکلی که دارد، منحنی زنگ هم می‌گویند. و اما پارامترهای این تابع:

* :locمیانگین توزیع.
* :scaleانحراف معیار توزیع.
* :sizeتعداد نمونه داده‌هاست.

مثال‌های زیر را درنظر بگیرید:

*# numpy.random.rand() method*

import numpy as np

*# 1D Array*

array = np.random.rand(5)

print("1D Array filled with random values : \n", array)

1D Array filled with random values : [ 0.84503968  0.61570994  0.7619945   0.34994803  0.40113761]

**کار با تاریخ و زمان در NumPy**

در این بخش از مقاله‌ی آموزش کامل کتابخانه NumPY کار با تاریخ و زمان در NumPY  را آموزش می‌دهیم، ]

NumPy دارای انواعی از  ساختارهای داده (Data Structure) است که به طور طبیعی از عملکرد

datetime پشتیبانی می‌کنند. این نوع داده 'datetime64' نامیده می‌شود که از کتابخانه‌ای در زبان پایتون با همین نام

برگرفته شده‌است. در مثال زیر چند نمونه از کاربرد زمان و تاریخ را می‌بینید.

import numpy as np

*# creating a date*

today = np.datetime64('2017-02-12')

print("Date is:", today)

print("Year is:", np.datetime64(today, 'Y'))

*# creating array of dates in a month*

dates = np.arange('2017-02', '2017-03', dtype='datetime64[D]')

print("\nDates of February, 2017:\n", dates)

print("Today is February:", today in dates)

*# arithmetic operation on dates*

dur = np.datetime64('2017-05-22') - np.datetime64('2016-05-22')

print("\nNo. of days:", dur)

print("No. of weeks:", np.timedelta64(dur, 'W'))

*# sorting dates*

a = np.array(['2017-02-12', '2016-10-13', '2019-05-22'], dtype='datetime64')

print("\nDates in sorted order:", np.sort(a))

Date is: 2017-02-12 Year is: 2017 Dates of February, 2017: ['2017-02-01' '2017-02-02' '2017-02-03' '2017-02-04' '2017-02-05' '2017-02-06' '2017-02-07' '2017-02-08' '2017-02-09' '2017-02-10' '2017-02-11' '2017-02-12' '2017-02-13' '2017-02-14' '2017-02-15' '2017-02-16' '2017-02-17' '2017-02-18' '2017-02-19' '2017-02-20' '2017-02-21' '2017-02-22' '2017-02-23' '2017-02-24' '2017-02-25' '2017-02-26' '2017-02-27' '2017-02-28'] Today is February: True No. of days: 365 days No. of weeks: 52 weeks Dates in sorted order: ['2016-10-13' '2017-02-12' '2019-05-22']

**numpy.asarray در پایتون**

تابع as array برای تبدیل داده‌ی ورودی به آرایه استفاده می‌شود. ورودی می‌تواند به شکل لیست، تاپل و یا

ndarray و... باشد. فرم کلی به شکل زیر است:

numpy.asarray(data, dtype=None, order=None)

* :data داده‌ای که می‌خواهید به آرایه تبدیل کنید.
* :dtype یک پارامتر اختیاری است و اگر مقداردهی نشود، مقدار آن از نوع داده‌های ورودی گرفته می‌شود.
* :order یک پارامتر اختیاری است و مقدار پیش‌فرض ان C است و استایل پایه برای سطرهاست و مقدار دیگری که می‌تواند بگیرد F است.

ماتریس دو بُعدی زیر را که شامل چهار سطر و ستون پرشده با یک است در نظر بگیرید:

import numpy as np

A = np.matrix(np.ones((4,4)))

نمی‌توان مقدار درایه‌های این ماتریس را با کپی و انتساب به شکل زیر تغییر داد:

np.array(A)[2]=2

print(A)

[[1. 1. 1. 1.] [1. 1. 1. 1.]  [1. 1. 1. 1.]  [1. 1. 1. 1.]]

**numpy.arrange در پایتون**

این تابع در NumPy آرایه‌ای را برمی‌گرداند که نتیجه‌ی آن بازه‌ای از اعداد است که فاصله‌ی بین آن‌ها برابر است.

قاعده‌ی کلی دستور به شکل زیر است:

numpy.arrange(start, stop,step)

مثال:

import numpy np

np.arrange(1, 14, 4)

array([ 1, 5, 9, 13]

**numpy.linspace  و numpy.logspace در پایتون**

این تابع نیز دنباله‌ای از اعداد را تولید می‌کند و قاعده‌ی تعریف آن به صورت زیر است:

numpy.linspace(start, stop, num, endpoint)

* :start مقدار شروع دنباله.
* :stop مقدار پایان دنباله (خود مقدار پایانی در نظر گرفته می‌شود.)
* :num تعداد اعداد مورد نظر که مقدار پیش‌فرض 50 را دارد.
* :endpoint اگر True باشد، (که به طور پیش فرض هست) مقدار stop را به عنوان مقدار نهایی درنظرمی‌گیرد و اگر False بود آن را در نظر نمی‌گیرد.

به عنوان مثال می‌توان ده مقدار با فاصله‌ی یکسان در بازه‌ی 1 تا 5 به شکل زیر تولید کرد:

import numpy as np

np.linspace(1.0, 5.0, num=10)

 array([1. , 1.44444444, 1.88888889, 2.33333333, 2.77777778, 3.22222222, 3.66666667, 4.11111111, 4.55555556, 5. ]

اگر نمی‌خواهید آخرین عدد در بازه‌ جزو اعداد تولیدی باشد، همان‌طور که گفته‌ شد مقدار endpoint را برابر False قرار دهید:

np.linspace(1.0, 5.0, num=5, endpoint=False)

**اندیس‌دهی (Indexing) و برش (Slicing) آرایه‌های NumPy در پایتون**

برش ماتریس و یا آرایه توسط NumPy کار ساده‌ای است. به مثال‌های زیر توجه کنید:

*## Slice*

import numpy as np

e = np.array([(1,2,3), (4,5,6)])

print(e)

[[1 2 3], [4 5 6]]

print('First row:', e[0])

print('Second row:', e[1])

First row: [1 2 3] Second row: [4 5 6]

* مقدار قبل از کاما در براکت، به سطر اشاره می‌کند.
* مقدار سمت راست کاما به ستون اشاره می‌کند.
* اگر بخواهید تنها ستون را انتخاب کنید، قبل از شماره‌ی ستون باید دو نقطه (:) قرار دهید [ :column\_index] .
* دو نقطه  [ :column\_index] به این معنی است که شما تمامی سطرهای آن ستون را انتخاب کرده‌اید.

print('Second column:', e[:,1])

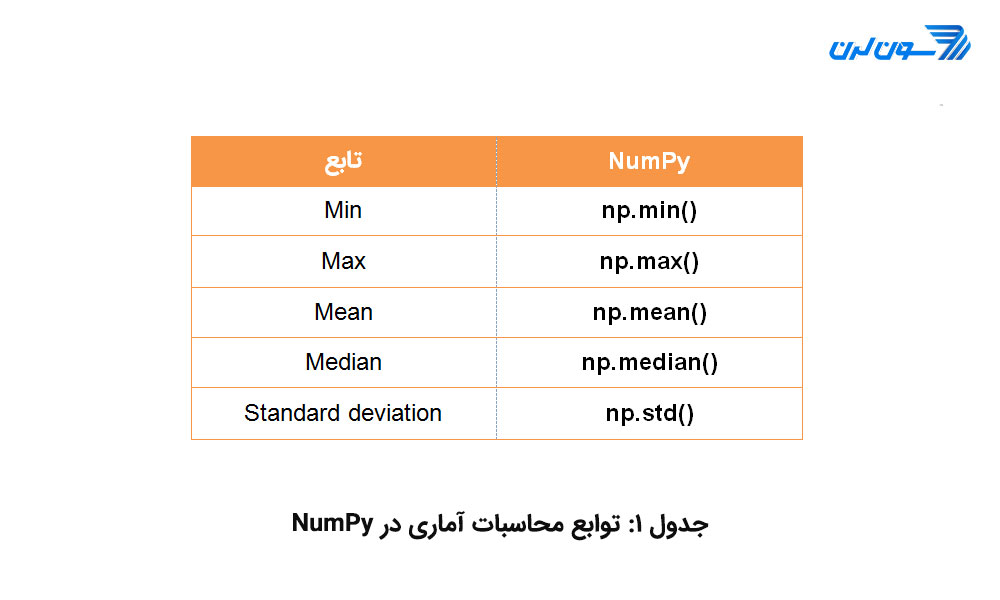
Second column: [2 5]

**توابع آماری NumPy**

این کتابخانه تعدادی تابع آماری برای محاسبه‌ی مواردی همچون مینیمم، ماکزیمم، میانگین، میانه، دهک و صدک،

انحراف معیار، واریانس و ... برای اعمال روی آرایه‌ها و ماتریس‌های ورودی دارد. در جدول زیر برخی از این توابع

آمده‌اند:



آرایه‌ی زیر را در نظر بگیرید:

import numpy as np

normal\_array = np.random.normal(5, 0.5, 10)

print(normal\_array)

[5.56171852 4.84233558 4.65392767 4.946659 4.85165567 5.61211317 4.46704244 5.22675736 4.49888936 4.68731125]

توابع آماری را روی داده‌های تولید شده به شکل زیر اعمال می‌کنیم:

*### Min*

print(np.min(normal\_array))

*### Max*

print(np.max(normal\_array))

*### Mean*

print(np.mean(normal\_array))

*### Median*

print(np.median(normal\_array))

*### Std*

print(np.std(normal\_array))

4.467042435266913 5.612113171990201 4.934841002270593 4.846995625786663

0.3875019367395316

**ضرب در پایتون به کمک NumPy**

numpy.dot ضرب نقطه‌ای بین ماتریس‌ها یا همان ضرب ماتریسی را انجام می‌دهد. قاعده‌ی کلی به شکل زیر است:

numpy.dot(x, y, out=None)

x و y آرایه‌ی ورودی‌اند که باید 1D یا 2D (یک بُعدی یا دو بُعدی) باشند. خروجی اگر ورودی آرایه یک بعدی باشد،

مقدار اسکالر و در غیر این‌صورت یک آرایه است. به مثال زیر توجه کنید:

*## Linear algebra*

*### Dot product: product of two arrays*

f = np.array([1,2])

g = np.array([4,5])

*### 1\*4+2\*5*

np.dot(f, g)

14

اگر ورودی‌ها به صورت اسکالر باشند و یا ضرب عضو به عضو اعضای آرایه‌ها مدنظر باشد از numpy.multiply

می‌توان استفاده کرد. اما اگر ضرب ماتریسی برای آرایه‌های دوبعدی می‌خواهیم انجام دهیم numpy.matmulترجیح

داده می‌شود.

**تفاوت میان Copy و View در NumPy**

تفاوت میان یک کپی و یک view از یک آرایه در این است که کپی یک آرایه‌ی جدید است اما؛ view فقط یک دید از

آرایه ایجاد می‌کند. کپی مسئول داده‌های خود است و هر گونه تغییر داده در آن، مستقل از آرایه‌ی اولیه است و بالعکس،

هر تغییری در آرایه‌ی اولیه در کپی آن بی‌تأثیر است. درمورد view، این قضیه صادق نیست و تغییرات داده‌ها در

view و آرایه‌ی اولیه بر یکدیگر اثر می‌گذارد.

**Copy**

یک کپی از آرایه ایجاد می‌کنیم و تغییرات آرایه و کپی آن را بر همدیگر مشاهده می‌کنیم:

import numpy as np

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

x = arr.copy()

arr[0] = 42

print(arr)

print(x)

[42 2 3 4 5] [1 2 3 4 5]

**View**

یک view از آرایه ایجاد می‌کنیم و تغییرات آرایه و view آن را بر همدیگر مشاهده می‌کنیم:

import numpy as np

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

x = arr.view()

x[0] = 31

print(arr)

print(x)

[31 2 3 4 5] [31 2 3 4 5]

همان‌طور که گفته شد کپی بر داده‌های خود مالکیت دارد اما view نه. برای بررسی مالکیت آرایه‌ای درNumPy ،

می‌توان مقدار ویژگی base آن را بررسی کرد. اگر None بود، یعنی مالکیت دارد و اگر به آبجکت (Object) اصلی و

اولیه اشاره کرد یعنی مالکیت ندارد.

import numpy as np

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

x = arr.copy()

y = arr.view()

print(x.base)

print(y.base)

None [1 2 3 4 5]

نتیجه گیری

در علم داده با توابع کتابخانه‌ی NumPy که از پکیج‌های اساسی در ساخت سایر پکیج‌های داده‌کاوی با پایتون است، آشنا

شدیم. NumPy به عنوان کتابخانه‌ای با استفاده ی گسترده در محاسبات نقشی مهم در سایر کتابخانه‌های محاسباتی پایتون

همچون sikit-learn ،tensorflow ،matplot ،opencv و... دارد. داشتن درک درستی از این کتابخانه در فهم سایر

کتابخانه‌های پیشرفته‌تر بسیار مفید خواهد بود.

پروژه کتابخانه numpy در پایتون

مهدی باغگلی

منبع:

https://7learn.com/blog/complete-numpy-library-tutorial