**فصل 1**

**بردارها، ماتریس‌ها و آرایه‌ها**

**1-‌ مقدمه**

NumPy اساس ذخیره‌سازی اطلاعات یادگیری ماشین پایتون[[1]](#footnote-1) است. NumPy امکان انجام‌ عملیات‌های اثربخش بر روی ساختارهای داده را فراهم می‌آورد که اغلب اوقات در یادگیری ماشین کاربرد دارند: بردارها، ماتریس‌ها، و تنسورها. درحالی‌که در این کتاب تأکیدمان بر NumPy نیست، اما در سرتاسر فصول بعدی کتاب مکرراً ظاهر خواهد شد. این فصل پرکاربردترین عملیات‌های NumPy را شامل می‌شود که احتمالاً هنگام کار با گردش‌های کار یادگیری ماشین به آنها برخورد خواهیم کرد.

**1-1‌ ساخت بردار**

**مسأله**

باید یک بردار درست کنید.

**حل:**

با استفاده از NumPy آرایه تک-بُعدی بسازید.

 **بحث**

ساختار داده اصلی NumPy آرایه چندبُعدی است. برای ساختن بردار، صرفاً یک آرایه تک-بُعدی می‌سازیم. درست مانند بردارها، می‌توان این آرایه‌ها را نیز به‌صورت افقی (یعنی سطری) یا به‌صورت عمودی (یعنی ستونی) نشان دهیم.

**همچنین، نگاه کنید به:**

* [**‌**بردارها، بازی ریاضی](https://www.mathsisfun.com/algebra/vectors.html).
* [بردار اقلیدوسی، ویکی‌پدیا](https://en.m.wikipedia.org/wiki/Euclidean_vector)

**1-2‌ ساخت ماتریس**

**مسأله:**

باید ماتریس بسازید.

**حل:**

با استفاده از NumPy آرایه دو بُعدی بسازید:



**بحث**

برای ساخت ماتریس، می‌توانیم از آرایه دو-بُعدی NumPy استفاده کنیم. در روش حل ما، این ماتریس سه سطر و دو ستون دارد (ستون 1ها و ستون 2ها).

NumPy عملاً نوعی ساختار داده ماتریس اختصاصی دارد:





با این حال، ساختار داده این ماتریس به دو دلیل توصیه نمی‌شود. اول، این آرایه‌های ساختار داده استاندارد غیررسمیِ NumPy هستند.

دوم، اکثریت زیادی از عملیات‌های NumPy آرایه‌ها را برمی‌گردانند، نه اشیای ماتریس را.

**همچنین، نگاه کنید به:**

* ‌ [ماتریس، ویکی‌پدیا](https://en.m.wikipedia.org/wiki/Matrix_(mathematics))
* [ماتریس، دنیای ریاضی ولفرام](https://mathworld.wolfram.com/Matrix.html)

**1-3‌ ساخت ماتریس sparse[[2]](#footnote-2)**

**مسأله:**

داده‌هایی دارید که مقادیر غیرصفر آن بسیار کم است، و می‌خواهید به‌شکل کارآمدی آن را نشان دهید.

**حل:**

ماتریس sparseبسازید:



**بحث**

در بسیاری از مواقع در یادگیری ماشین با مقادیر انبوهی از داده مواجه هستیم؛ با این حال، اما بیشتر عناصر این داده‌ها صفر هستند. برای نمونه، ماتریسی را در نظر بگیرید که در آن ستون‌هایش فیلم‌های روی Netflix و سطرهایش کاربران Netflix باشند، و مقادیر عبارت باشد از تعداد دفعاتی که کاربر فیلم خاصی را تماشا می‌کند. این ماتریس ده‌ها هزار ستون و میلیون‌ها سطر خواهد داشت! با این حال، از آنجا که بیشتر کاربران بیشترِ فیلم‌ها را تماشا نمی‌کنند، در نتیجه اکثر عناصر ماتریس صفر خواهند بود.

ماتریس‌های Sparse فقط عناصر غیرصفر را ذخیره می‌کنند و فرض‌شان بر این است که تمام مقادیر دیگر صفر خواهند بود، در نتیجه صرفه‌جویی قابل‌توجهی در محاسبات انجام می‌شود. در روش حل‌مان، با دو مقدار غیرصفر یک آرایه NumPy ساختیم، سپس آن را به ماتریس Sparse تبدیل کردیم. اگر به این ماتریس Sparse نگاه کنیم، می‌بینیم که فقط مقادیر غیرصفر ذخیره شده‌اند:





انواع مختلفی از ماتریس Sparse وجود دارد. با این حال، در ماتریس‌های *سطر Sparse فشرده[[3]](#footnote-3)* (CSR)، (1,1) و (2,0) به‌ترتیب، نمایه‌های (نمایه‌شده با صفر) مربوط به مقادیر غیرصفر 1 و 3 را نشان می‌دهند. برای نمونه، عنصر 1 در سطر دوم و ستون دوم است. اگر ماتریس بسیار بزرگتری را با تعداد بسیار زیادی از عناصر صفر بسازیم و این ماتریس بزرگتر را با ماتریس Sparse اولیه‌مان مقایسه کنیم، مزیت ماتریس‌های Sparse را مشاهده خواهیم کرد:











همان‌طور که می‌بینیم، با وجود این‌که تعداد بسیار زیادی از عناصر صفر را در ماتریس بزرگتر اضافه کردیم، اما نمایش Sparse آن دقیقاً همان ماتریس Sparse اولیه است. یعنی، افزودن عناصر صفر اندازه ماتریس Sparse را تغییر نداد.

همان‌طور که گفته شد، انواع مختلف و بسیار زیادی ماتریس Sparse وجود دارد، مانند ستون Sparse فشرده،[[4]](#footnote-4) لیست لیست‌ها[[5]](#footnote-5)، فرهنگ‌ واژگانی از کلیدها[[6]](#footnote-6). درحالی‌که توضیح انواع مختلف این ماتریس‌ها و نتایج‌شان خارج از حوصله این کتاب است، اما شایان توجه است که در عین حال که «بهترین» نوع ماتریس Sparse وجود ندارد، ولی تفاوت‌های معناداری بین آنها وجود دارد و باید از دلیل ترجیح یکی بر دیگری آگاه باشیم.

**همچنین، نگاه کنید به:**

* **‌**[ماتریس‌های Sparse، مستندسازی SciPy](https://docs.scipy.org/doc/scipy/reference/sparse.html)
* [**‌**101 روشِ ذخیره‌سازی ماتریس Sparse](https://medium.com/@jmaxg3/101-ways-to-store-a-sparse-matrix-c7f2bf15a229)

**1-4 پیش تخصیص آرایه‌های NumPy**

**مسئله**

شما باید آرایه‌هایی با اندازه معین را با مقداری از قبل تخصیص دهید.

**راه‌حل**

NumPy دارای توابعی برای تولید بردارها و ماتریس‌هایی با هر اندازه با استفاده از 0، 1 یا مقادیر دلخواه شما است:



**بحث**

تولید آرایه‌هایی که از قبل با داده‌ها پر شده‌اند، برای چندین هدف مفید است، مانند عملکرد بهتر کد یا استفاده از داده‌های مصنوعی برای آزمایش الگوریتم‌ها. در بسیاری از زبان‌های برنامه نویسی، پیش تخصیص‌یک آرایه از مقادیر پیش فرض (مانند 0s)‌یک روش معمول در نظر گرفته می‌شود.

**1-5‌ انتخاب عناصر**

**مسأله:**

باید یک یا چند عنصر در یک بردار یا ماتریس انتخاب کنید.

**حل:**

با استفاده از آرایه‌های NumPy این کار آسان است:



3

# Select second row, second column

Matrix [1, 1]

5

**بحث**

مانند خیلی از موارد در پایتون، آرایه‌های NumPy نیز با صفر نمایه می‌شوند، به این معنا که نمایه عنصر اول 0 است، نه 1. با توجه به این امر، NumPy انواع بسیار گوناگونی از روش‌های انتخاب عناصر یا گروه‌هایی از عناصر آرایه‌ها را در اختیارمان می‌گذارد (یعنی، نمایه‌سازی[[7]](#footnote-7) و برش[[8]](#footnote-8)):

# Select all elements of a vector

Vector [:]

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

# Select everything up to and including the third element

Vector [:3]

Array ([1, 2, 3])

# select everything after the third element

Vector [3:]

Array ([4, 5, 6] )

# select the last element

Vector [-1]

6

# Select the first two rows and all columns of a matrix

Matrix [:2, :]

Array ([1, 2, 3],

[4, 5, 6] )

# Select all rows and the second column

Matrix [:, 1:2]

Array ([[2], [5], [8]] )

**1-6‌ توصیف ماتریس**

**مسأله:**

می‌خواهید شکل، اندازه، و ابعاد ماتریس را توصیف کنید.

**حل:**

از قابلیت‌های Shape، size و ndim استفاده کنید:



(3, 4)

# View number of elements (rows\*columns)

Matrix.size

12

# View number of dimensions

matrix.ndim

2

**بحث**

این کار ممکن است ساده به نظر برسد (و ساده نیز است)؛ با این حال، بررسی شکل و اندازه آرایه برای محاسبات بیشتر و همچنین، بررسی شهودیِ پس از انجام عملیات، همواره ارزشمند خواهد بود.

**1-7‌ اجرای عملیات‌ها برای عناصر**

**مسأله:**

می‌خواهید یک تابع را روی عناصر مختلف در یک آرایه اعمال و اجرا کنید.

**حل:**

از قابلیت vectorize در NumPy استفاده کنید:





**بحث**

کلاس Vectorize در NumPy تابع را به تابعی تبدیل می‌کند که می‌تواند برای همه عناصر یک آرایه یا برشی از آرایه اجرا شود. شایان توجه است که Vectorize اساساً یک حلقه for روی عناصر است و سرعت را افزایش نمی‌دهد. علاوه بر این، آرایه‌های NumPy به ما امکان می‌دهند تا عملیات‌هایی را بین آرایه‌ها انجام دهیم، هرچند ابعادشان یکسان نباشد (فرآیندی که *پخش[[9]](#footnote-9)* نامیده می‌شود). برای نمونه، می‌توانیم با استفاده از فرآیند پخش نسخه بسیار ساده‌تری از راه‌حل‌مان را بسازیم:

# Add 100 to all elements

Matrix + 100

Array ([[101, 102, 103],

[104], [105], [106],

[107, 108, 108]])

**1-8‌ یافتن مقادیر بیشینه و کمینه**

**مسأله:**

باید مقادیر بیشینه و کمینه را در یک آرایه پیدا کنید.

**حل:**

از قابلیت max و min در NumPy استفاده کنید:



9

# Return minimum element

np.min (matrix)

1

**بحث**

اغلب اوقات، می‌خواهیم مقدار بیشینه و کمینه را در یک آرایه یا زیرمجموعه‌ای از یک آرایه بدانیم. می‌توان با توابع max و min این کار را انجام داد. با استفاده از پارامتر axis، می‌توانیم این عملیات را در طول محور معینی اجرا کنیم:

# Find maximum element in each column

np.max (matrix, axis = 0)

Array ([7, 8. 9])

# Find maximum element in each row

Np.max (matrix, axis = 1)

Array ([3, 6 , 9])

**1-9‌ محاسبه میانگین، واریانس، و انحراف معیار**

**مسأله:**

می‌خواهید نوعی آمار توصیفی را درباره یک آرایه محاسبه کنید.

**حل:**

از قابلیت‌های mean، var و std در NumPy استفاده کنید:

# Load library

Import numpy as np

Create matrix

Matrix = np.array([[1, 2, 3],

[4, 5, 6],

[7, 8, 9]])

# Return mean

np.mean(matrix)

5.0

****

6.666666666666667

# Return standard deviation

Np.std(matrix)

2.581988874716112

**بحث**

درست مثل max و min، می‌توانیم آمار توصیفی کل ماتریس را به‌راحتی بدست آوریم یا محاسباتی را در راستای محور واحدی انجام دهیم:

# Find the mean value in each column

Np.mean (matrix, axis = 0)

Array ([4., 5., 6.])

**1-10‌ تغییرشکل آرایه‌ها**

**مسأله:**

می‌خواهید شکل آرایه (تعداد سطرها و ستون‌ها) را بدون تغییر مقادیر عنصر تغییر دهید.

**حل:**

از قابلیت reshape استفاده کنید:



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

[7, 8, 9, 10, 11, 12]])

**بحث**

با کمک قابلیت reshape می‌توانیم ساختار آرایه را چنان تغییر دهیم که همان داده‌ها حفظ شوند اما به‌صورت تعداد متفاوتی از سطرها و ستون‌ها مرتب شوند. تنها شرط لازم برای انجام این کار آن است که تعداد عناصر در شکل ماتریس اول و ماتریس جدید (یعنی‌ اندازه‌ها) یکسان باشد. می‌توانیم با استفاده از قابلیت size اندازه ماتریس را مشاهده کنیم:

Matrix.size

12

یکی از آرگومان‌های مفید در قابلیت reshape عدد (1-) است، که عملاً به معنای «مقدار لازم» است، بنابراین، reshape (1, -1) به معنای یک سطر و تعداد ستون‌های لازم است:

Matrix.reshape (1, -1)

Array([[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]])

در نهایت، اگر یک عدد صحیح ارائه دهیم، reshape یک آرایه یک-بعدی از آن طول را باز خواهد گرداند:

Matrix.reshape (12)

Array([1. 2, 3, 4, 5, 6 , 7, 8, 9, 10, 11, 12]])

**1-11 ‌ ترانهاده (Transpose) بردار یا ماتریس**

**مسأله:**

باید ترانهاده یک بردار یا ماتریس را محاسبه کنید.

**حل:**

از تابع T استفاده کنید:



Array([[1, 4, 7],

[2, 5, 8],

[3, 6, 9])

**بحث**

ترانهادن عملیاتی رایج در جبر خطی است که در آن نمایه‌های ستون و سطر هر عنصر تعویض می‌شوند. یکی از نکات ظریفی که معمولاً در خارج از کلاس جبر خطی از آن غفلت می‌شود، آن است که از نظر فنی بردار قابل ترانهادن نیست، زیرا فقط مجموعه‌ای از مقادیر است:

# Transpose vector

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

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

با این حال، معمولاً تبدیل بردار سطری به بردار ستونی یا بالعکس را ترانهاده می‌نامند (به دومین جفت از براکت‌ها توجه کنید):

# Transpose row victor

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

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

**1-12‌ هموارسازی[[10]](#footnote-10) ماتریس**

**مسأله:**

باید یک ماتریس را به آرایه تک-بُعدی تبدیل کنید.

**حل:**

از تابع flatten استفاده کنید:

# Load library

Import nimpy as np

# Create matrix

Matrix = np.array([[1, 2, 3],

[4, 5, 6],

[7, 8, 9]])

# Flatten matrix

Matrix.flatten ()

Array(1, 2, 3, 4, 5, 6, 7, 8, 9])

**بحث**

تابع flatten روشی ساده برای تبدیل ماتریس به آرایه تک-بُعدی است. در حالت دیگر، می‌توانیم از قابلیت reshape برای ساخت بردار سطری استفاده کنیم:

Matrix.reshape (1, -1)

Array([[1, 2, 3, 4, 5, 6, 7, 8, 9]])

**1-13 ‌ یافتن مرتبه ماتریس**

**مسأله:**

می‌خواهید مرتبه یک ماتریس را بدانید.

**حل:**

از روش جبر خطی NumPy matrix\_rank استفاده کنید:

# Load library

Import numpy as np

# Create matrix

Matrix = np.array ([[1, 1, 1],

[1, 1, 1],

[1, 1, 15]])

# Return matrix rank

Np.linalg.matrix\_rank(matrix)

2

**بحث**

مرتبه ماتریس ابعاد فضای برداری است که با ستون‌ها یا سطرهایش مشخص می‌شود. یافتن مرتبه ماتریس در NumPy به‌ لطف استفاده از قابلیت matrix\_rank آسان است.

**همچنین، نگاه کنید به:**

* [**‌**مرتبه ماتریس، CliffsNotes](https://www.cliffsnotes.com/study-guides/algebra/linear-algebra/real-euclidean-vector-spaces/the-rank-of-a-matrix)

**1-14‌ محاسبه دترمینان ‌**

**مسأله:**

باید دترمینان ماتریس را بدست آورید.

**حل:**

با استفاده از تابع جبر خطی NumPy det می‌توان دترمینان را بدست آورد:



0.0

**بحث**

گاهی اوقات محاسبه دترمینان ماتریس مفید واقع می‌شود.

NumPy با قابلیت det محاسبه دترمینان را ساده می‌کند.

**همچنین، نگاه کنید به:**

* ‌دترمینان/عصاره جبر خطی، فصل 5، 3Blue1Brown
* دترمینان، دنیای ریاضی ولفرام

**1-15 بدست آوردن قطر ماتریس**

**مسأله:**

باید عناصر قطر ماتریس را بدست آورید.

**حل:**

از تابع diagnol استفاده کنید:





Array ([1, 4, 9])

**بحث**

با استفاده از قابلیت diagonal بدست آوردن عناصر قطری ماتریس آسان می‌شود. همچنین، می‌توان با استفاده از پارامتر offset عناصر یک قطر را از قطر اصلی بدست آورد:

# Return diagonal one above the main diagonal

Matrix.diagonal (offset = 1)

Array ([2, 6])

# Return diagonal one above the main diagonal

Matrix.diagonal (offset = -1)

Array ([2, 8])

**1-16‌ محاسبه اثر[[11]](#footnote-11) ماتریس**

**مسأله:**

باید اثر ماتریس را محاسبه کنید.

**حل:**

از تابع trace استفاده کنید:



14

**بحث**

اثر ماتریس مجموع عناصر قطری است و اغلب اوقات در روش‌های یادگیری ماشین استفاده می‌شود. با توجه به آرایه چندبعدی NumPy، می‌توانیم با استفاده از قابلیت trace اثر ماتریس را محاسبه کنیم. همچنین، می‌توانیم قطر ماتریس را برگردانیم و مجموع آن را محاسبه کنیم:

# Return diagonal and sum elements

Sum (matrix.diagonal () )

14

**همچنین، نگاه کنید به:**

اثر ماتریس مربعی

**1-17‌ یافتن مقادیر ویژه و بردارهای ویژه**

**مسأله:**

باید مقادیر ویژه و بردارهای ویژۀ یک ماتریس مربعی را پیدا کنید.

**حل:**

از قابلیت linalg.eig در NumPy استفاده کنید:



# View engenvalues

Eigenvalues

Array ([13.55075847, 0.74003145, -3.29078992])

# View eigenvectors

Eigenvectors

Array([[-0.17622017, -0.96677403, -0.53373322],

[-0.435951, 0.2053623, -0.64324848],

[-0.88254925, 0.15223105, 0.54896288]])

**بحث**

بردارهای ویژه کاربرد وسیعی در کتابخانه‌های یادگیری ماشین دارند. به‌طور شهودی، با توجه به تبدیل خطی که با ماتریس A نمایش داده شده است، بردارهای ویژه بردارهایی هستند که هنگام استفاده از آن تبدیل، فقط مقیاس‌شان تغییر می‌کند (نه جهت‌شان). به‌صورت رسمی‌تر:

**Av=**

که در آن A ماتریس مربعی است، λ حاوی مقادیر ویژه و v حاوی بردارهای ویژه است. در مجموعه‌ابزار جبر خطی، می‌توانیم با کمک قابلیت eig مقادیر ویژه، و بردارهای ویژه هر نوع ماتریس مربعی را محاسبه کنیم.

**همچنین، نگاه کنید به:**

* [**‌**توضیح بصری مقادیر ویژه و بردارهای ویژه، Setosa.io](https://setosa.io/ev/eigenvectors-and-eigenvalues/)
* بردارهای ویژه و مقادیر ویژه/عصاره جبر خطی، فصل 10، 3Blue1Brown.

**1-18 ‌ محاسبه حاصلضرب‌های نقطه‌ای**

**مسأله:**

باید حاصلضرب نقطه‌ای دو بردار را محاسبه کنید.

**حل:**

از قابلیت dot در NumPy استفاده کنید:



32

**بحث**

حاصلضرب نقطه‌ای دو بردار، a و b، به‌صورت زیر تعریف می‌شود:

که در آن، ai عنصر iام در بردار a است. می‌توانیم از کلاس dot در NumPy برای محاسبه حاصلضرب نقطه‌ای استفاده کنیم. همچنین در پایتون 3.5+، می‌توان از عملگر @ استفاده کرد که جدید است:

# Calculate dot product

Vector\_a @ vector\_b

32

**همچنین، نگاه کنید به:**

* ‌حاصلضرب نقطه‌ای بردار و طول بردار، خان آکادمی
* [حاصلضرب نقطه‌ای، یادداشت‌های ریاضی آنلاین پل](https://tutorial.math.lamar.edu/Classes/CalcII/DotProduct.aspx)

**1-19‌ جمع و تفریق ماتریس‌ها**

**مسأله:**

می‌خواهیم دو ماتریس را با هم جمع یا از هم کم کنیم.

**حل:**

از قابلیت add و substract در NumPy استفاده کنید:

# Load library

Import numpy as np

# Create matrix

Matrix\_a = np.array([[1, 1, 1],

[1, 1, 1],

[1, 1, 2]])

# Create matrix

Matrix\_b = np.array ([[1, 3, 1],

[1, 3, 1],

[1, 3, 8]])

# Add two matrices

Np.add(matrix\_a, matrix\_b)

Array ([[2, 4, 2],

[2, 4, 2],

[2, 4, 10]])

# Subtract two matrices

Np.subtract(matrix\_a, matrix\_b)



**بحث**

به روش دیگر، می‌توانیم از عملگرهای + و – استفاده کنیم:

# Add two matrices

Matrix\_a + matrix\_b

Array ([[2, 4, 2],

[2, 4, 2],

[2, 4, 10]])

**1-20‌ ضرب ماتریس‌ها**

**مسأله:**

می‌خواهیم دو ماتریس را در هم ضرب کنیم:

**حل:**

از قابلیت dot استفاده کنید:



Array ([[2, 5],

[3, 7]])

**بحث**

در روش دیگر، در پایتون 3.5+، می‌توانیم از عملگر @ استفاده کنیم:

# Multiply two matrices

Matrix\_a @ matrix\_b

Array ([[2, 5],

[3, 7]])

اگر بخواهیم ضرب نظیر به نظیر درایه‌ها را انجام دهیم، می‌‌توانیم از عملگر \* استفاده کنیم:

# Multiply two matrices element-wise

Matrix\_a \* matrix\_b

Array ([[1, 3],

[1, 4]])

**همچنین، نگاه کنید به:**

* [‌آرایه در برابر عملیات‌های ماتریس، سایت شرکت (MathWorks)](https://www.mathworks.com/help/matlab/matlab_prog/array-vs-matrix-operations.html?requestedDomain=true)

**1-21‌ وارون‌سازی ماتریس**

**مسأله:**

می‌خواهیم وارون یک ماتریس مربعی را محاسبه کنیم:

**حل:**

از تابع inv در NumPy استفاده می‌کنیم:



Array ([[-1.66666667, 1.33333333],

[0.66666667, -0.33333333]])

**بحث**

وارون ماتریس مربعی **A** ماتریس دیگری به نام است، طوری‌که:

**A = I**

که در آن، **I** ماترس واحد یا همانی است. در NumPy، می‌توانیم از قابلیت lingal.inv برای محاسبه استفاده کنیم، به‌شرطی که وجود داشته باشد. برای مشاهده عملی این کار، می‌توانیم ماتریس را در وارون آن ضرب کنیم و ماترس واحد را بدست آوریم:

# Multiply matrix and its inverse

Matrix @ np.linalg.inv (matrix)

Array ([[1., 0.],

[0., 1.]])

**همچنین، نگاه کنید به:**

* [‌وارون یک ماتریس](http://www.mathwords.com/i/inverse_of_a_matrix.htm)

**1-22‌ تولید مقادیر تصادفی**

**مسأله:**

می‌خواهید مقادیر شبه تصادفی تولید کنید.

**حل:**

از قابلیت random در NumPy استفاده کنید:

# Load library

Iport numpy as np

# Set seed

Np.random.seed (0)

# Generate three random floates between 0.0 and 1.0

Np.random.random (3)

Array ([0.5488135, 0.71518937, 0.60276338])

**بحث**

NumPy تعداد بسیار گوناگونی از ابزارها را برای تولید اعداد تصادفی در اختیارمان می‌گذارد، که بسیار بیشتر از آن است که بتوان در اینجا ذکر کرد. در روش حل ما، مقادیر شناور تولید می‌کنیم، با این حال، معمولاً اعداد صحیح نیز تولید می‌شود:

# Generate three random integers between 1 and 10

Np.random.random (0, 11, 3)

Array ([3, 7, 9])

در روش دیگر، می‌توانیم اعداد را از طریق کشیدن آنها از توزیع تولید کنیم:



Array (-1.42232584, 1.52006949, -0.29139398])

# Draw three numbers from a logistic distribution with mean

0.0 and scale of 1.0

Np.random.logistic (0.0, 1.0, 3)

Array ([-0.98118713, -0.08939902, 1.46416405])

# Draw three numbers greater than or equal to 1.0 and less

than 2.0

np.random.uniform (1.0, 2.0, 3)

array (1,47997717, 1.3927848, 1.83607876])

در نهایت، گاهی نیاز است یک عدد تصادفی چند بار برگردانده شود (یعنی همان عدد قبلی مجددا برگرانده شود) تا نتایج قابل پیش‌بینی و تکرارپذیر باشد[[12]](#footnote-12). می‌توانیم این کار را با تعیین «بذرِ»[[13]](#footnote-13) مولد اعداد شبه تصادفی انجام دهیم. فرآیندهای تصادفی با بذر یکسان همواره خروجی یکسانی تولید خواهد کرد. از بذرها در سرتاسر این کتاب استفاده خواهیم کرد، طوری‌که کدی که در کتاب‌ می‌بینید و کدی که روی رایانه‌تان اجرا می‌کنید، نتایج یکسانی را تولید می‌کند.

1. . پیتون تلفظ فرانسوی و پایتون [پایتِن] تلفظ انگلیسی این واژه است که در اصل نام یک نوع مار بزرگ است. هر دو تلفظ در زبان فارسی نیز رایج هستند. [↑](#footnote-ref-1)
2. . ماتریس خلوت یا اسپارس ماتریسی است که بیشتر درایه‌های آن صفر باشد. [↑](#footnote-ref-2)
3. compressed sparse row (CSR) [↑](#footnote-ref-3)
4. compressed sparse column [↑](#footnote-ref-4)
5. Lists of lists [↑](#footnote-ref-5)
6. dictionary of keys [↑](#footnote-ref-6)
7. Indexing [↑](#footnote-ref-7)
8. Slicing [↑](#footnote-ref-8)
9. Broadcasting [↑](#footnote-ref-9)
10. Flattening [↑](#footnote-ref-10)
11. trace [↑](#footnote-ref-11)
12. توضیح مترجم: در آموزش روش‌های یادگیری ماشین زیاد پیش می‌آید که از اعداد تصادفی استفاده شود. از آنجایی که اغلب نیاز است همان اعداد تصادفی قبلی دوباره قابل دسترس باشند تا مدل با همان اعداد قبلی دوباره آموزش داده شود و نتایج قبلی تولید شود، استفاده از بذر بسیار مناسب خواهد بود. [↑](#footnote-ref-12)
13. seed [↑](#footnote-ref-13)