科学技術英語 英文読解課題

Ec5 24 番 平田 蓮

付録 A 発展的な NumPy

この付録では、配列計算のための NumPy ライブラリをより深く掘り下げていきます。これには ndarray 型の内部の詳細や、より高度な配列操作とそのアルゴリズムが含まれます。

この付録は雑多な内容を含んでおり続けて読む必要はありません。

A.1 ndarray オブジェクトの内部構造

NumPy の ndarray は、同じ型のデータのブロックを (連続か区切られているかにかかわらず) 多次元配列オブジェクトとして提供します。データ型 (dtype) は、データが浮動小数点、整数、真偽値、あるいはこれまで見てきた他の型のどれであるかを決定します。

ndarray を柔軟にしている理由の一つに、すべての配列オブジェクトがデータをブロック区切りでみていることがあります。例えば配列に対する参照 `arr[::2, ::-1]` がデータを一切コピーしていないことを不思議に思うかもしれません。その理由は、ndarray は単なるデータとその型の集まりではなく、配列がメモリ内をさまざまなステップサイズで移動できるようにする「ストライド」情報を持っているからです。より正確に言うと、ndarray の内部は以下のような構成になっています。

- RAM やメモリマップ内にあるデータの集まりへのポインタ
- 配列内の一定個数のデータの型
- 配列の形を表すタプル
- 各次元に沿って要素を一つ進めるために遷移するバイト数 (整数値) のタプル

図 1 は、ndarray の内部構造を示す模式図です。

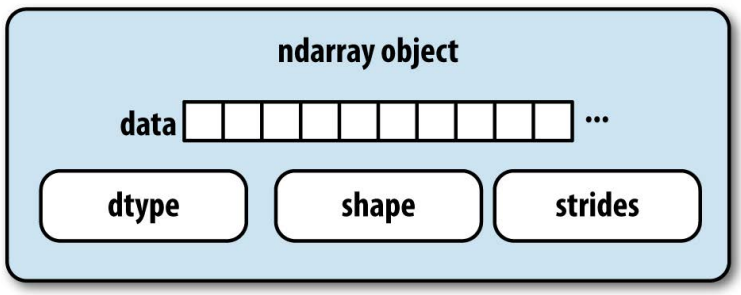

図 1　NumPy の ndarray オブジェクト

例えば、10 × 5 の配列は (10, 5) の形を持っています。

```
np.ones((10, 5)).shape # (10, 5)
```

float64(8 バイト) 型の典型的な 3×4×5 配列は、(160, 40, 8) のストライドを持っています。(一般的に特定の軸のストライドが大きいほど、その軸に沿って計算を行うコストが高くなるので、ストライドについて知っておくと便利です。)

```
np.ones((3, 4, 5), dtype=np.float64).strides # (160, 40, 8)
```

典型的な NumPy ユーザーが配列のストライドに興味を持つことは稀ですが、ストライドは 0 配列をコピー、構築するための重要な要素です。ストライドには負の値を設定することもでき、配列をメモリ上で逆向きに移動させることができます (例えば、`obj[::-1]` や `obj[:, ::-1]` のようなスライスがこれに該当します)。

NumPy の dtype の階層構造

配列の中に整数、浮動小数点数、文字列、Python オブジェクトが含まれているかどうかのチェックが必要なコードがあるかもしれません。浮動小数点数には複数の型 (float16〜float128) があるため、その配列の dtype が型一覧の中にあるかどうかをチェックするのは非常に冗長です。幸い、dtype には `np.integer` や `np.floating` などのスーパークラスがあり、`np.issubdtype()` と組み合わせて使用することができます。

```
ints = np.ones(10, dtype=np.uint16)
floats = np.ones(10, dtype=np.float32)

np.issubdtype(ints.dtype, np.integer) # True
np.issubdtype(floats.dtype, np.floating) # True
```

特定の dtype の親クラスをすべて見るには、その型の `mro` メソッドを呼び出します。

```
np.float64.mro() # [numpy.float64, numpy.floating, numpy.inexact, numpy.number, numpy.
    generic, float, object]
```

図 2 は、dtype の階層と親、サブクラスの関係を示したものです。

A.2 発展的な配列操作

配列を扱う方法には、インデックス、スライス、ブーリアンのサブセットといった洒落た方法以外のものもたくさんあります。データ分析アプリケーションの大部分は pandas の高レベル関数で処理されますが、既存のライブラリにはないデータアルゴリズムを書かなければならない場合もあるでしょう。

配列の変形

多くの場合、データを一切コピーせずに配列を他の形へ変形できます。そのためには、配列の `reshape` メソッドに新しい形を表すタプルを渡します。例えば、ある値を持つ一次元の配列を行列に並べ替えたいとします。(結果は図 3 の通りです。)

```
arr = np.arange(8) # [0, 1, 2, 3, 4, 5, 6, 7]
arr.reshape((4, 2)) # [[0, 1], [2, 3], [4, 5], [6, 7]]
```

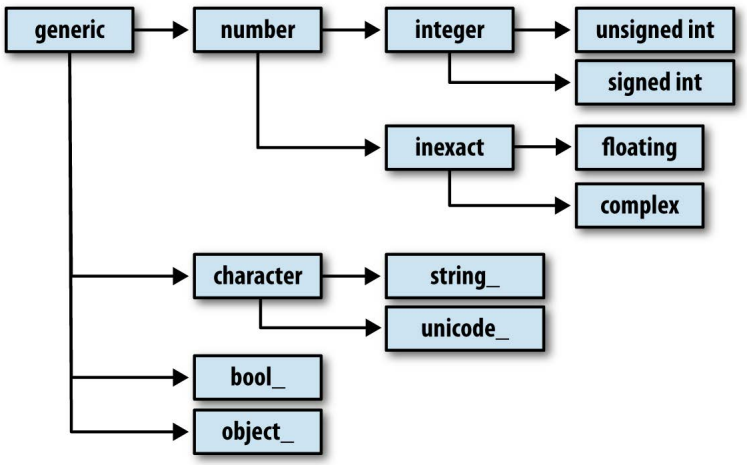

図 2　NumPy の dtype の階層構造

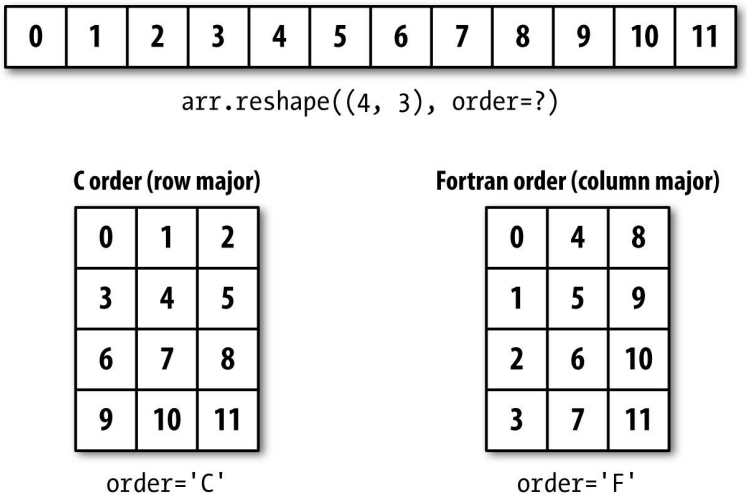

図 3　C(行優先) 順序と Fortran(列優先) 順序での変形

多次元配列は次のように変形することもできます。

```
arr.reshape((4, 2)).reshape((2, 4)) # [[0, 1, 2, 3], [4, 5, 6, 7]]
```

形を表す次元数のうち、一つは-1 にすることが可能で、その場合、その次数はデータから推測されます。

```
arr = np.arange(15)
arr.reshape((5, -1)) # [[ 0, 1, 2], [ 3, 4, 5], [ 6, 7, 8], [ 9, 10, 11], [12, 13,
    14]]
```

配列の shape 属性はタプルであるので、reshape() に渡すことも可能です。

```
other_arr = np.ones((3, 5))
other_arr.shape # (3, 5)
arr.reshape(other_arr.shape) # [[ 0, 1, 2, 3, 4], [ 5, 6, 7, 8, 9], [10, 11, 12,
    13, 14]]
```

一次元から高次元への変形の逆の操作は一般的にフラットニングやラベリングと呼ばれます。

```
arr = np.arange(15).reshape((5, 3)) # [[ 0, 1, 2], [ 3, 4, 5], [ 6, 7, 8], [ 9,
    10, 11], [12, 13, 14]]
arr.ravel() # [ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14]
```

ravel() は変形後の値が隣接している場合、元の値をコピーしません。flatten メソッドは常にデータのコピーを返すという点を除いて、ravel と同じ動作をします。

```
arr.flatten() # [ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14]
```

データは違う順序で変形したり、ラベリングしたりすることができます。これは、NumPy の新規ユーザーにとっては少しニュアンスの異なるトピックであるため、次のサブトピックになっています。