Язык Python. Часть 2

Лекция 12

Broadcasting

- Подравниваем количество измерений
- Если в одном аргументе измерений меньше, дополняем слева единицами
- shape-ы несовместимы, если хотя бы в одной позиции значения не равны
- И ни одно из них не единица

Пример

```
1 import numpy as np
2
3 data_1 = np.array([11, 22, 33, 44, 55], dtype=np.int16)
4 data_2 = np.array([1, 2, 3, 4], dtype=np.int16).reshape((4,
5
6 print(data_1 + data_2)
```

Расширение размерности через индексацию

- Есть специальный элемент, который можно использовать в индесации
- np.newaxis
- Его наличие добавляет измерение
- Может быть удобнее, чем reshape

Пример

```
1 import numpy as np
2
3 print(np.array([1, 10]).shape)
4 print(np.array([1, 10])[np.newaxis].shape)
5 print(np.array([1, 10])[np.newaxis, np.newaxis].shape)
6 print(np.array([1, 10])[np.newaxis, 0].shape)
7 print(np.array([1, 10])[0, np.newaxis].shape)
```

Пример

```
1 import numpy as np
2
3 print(np.arange(100).shape)
4 print(np.arange(100)[10:20].shape)
5 print(np.arange(100)[10:20, np.newaxis].shape)
6 print(np.arange(100)[np.newaxis, 10:20].shape)
```

Пример

```
1 import numpy as np
2
3 x = np.arange(5)
4 result = x[:, np.newaxis] + x[np.newaxis, :]
5 print(result)
```

Пример

```
1 import numpy as np
2
3 print(np.arange(36).reshape((4, 9))[np.newaxis, 1:3].shape)
4 print(np.arange(36).reshape((4, 9))[1:2, np.newaxis, 1:3].sh
5 print(np.arange(36).reshape((4, 9))[1:3, 2:5, np.newaxis].sh
6 print(np.arange(36).reshape((4, 9))[np.newaxis, 2].shape)
7 print(np.arange(36).reshape((4, 9))[1, np.newaxis, 1:3].shap
8 print(np.arange(36).reshape((4, 9))[1:2, np.newaxis, 2].shap
```

Массивы как индекс

- Одномерный ndarray может использоваться в качестве индекса по одному из измерений
- Элементы - индексы
- Могут быть отрицательными
- Могут повторяться

Пример

```
1 import numpy as np
2
3 data = np.array([122, 10, 13, 40, 43, 22])
4 print(data[np.array([2, -1, 2, 4])])
```

Логическое индексирование

- Начнем с одномерного случая
- Возьмем одномерный ndarray любого типа
- И другой, тоже одномерный, того же размера
- Только с логическими значениями

Логическое индексирование

- Второй можно использовать как индекс первого
- Получим одномерный ndarray
- Возможно, меньшего размера
- Войдут все элементы первого, для которых по тому же индексу во втором - True

Пример

```
1 import numpy as np
2
3 data = np.array([122, 10, 13, 40, 43, 22])
4 print(data[np.array([True, False, True, True, False, True])]
```

Типичное применение

- Выполняем логическую операцию
- Между вектором и константой
- Ее результат используем как индекс в том же векторе
- Работает как фильтр

Пример

```
1 import numpy as np
2
3 data = np.array([122, 10, 13, 40, 43, 22])
4 print(data[data > 30])
```

Многомерный вариант

- Принципиальная основа та же
- Но есть сложность
- Мы не можем сохранить многомерность
- Даже двумерность
- По разным измерениям может быть разное число элементов
- Поэтому получаем список того, что попало в результат

Многомерный вариант

- Но это может не устраивать
- Например, есть таблица замеров температуры
- Строки - время замеров
- Столбцы - место

Многомерный вариант

- Сравнение data > 0 даст перечень результатов замеров
- А хочется знать, когда и где они были
- Есть функция np.where
- Принимает ndarray с типом boolean
- Возвращает кортеж одномерных ndarray

Поэлементное применение функций

- Можно применять функции к каждому элементу ndarray
- Большой перечень встроенных
- https://numpy.org/doc/stable/reference/routines.ma
- Можно сделать свою

Пример

```
1 import numpy as np
2
3 data = np.arange(120).reshape((12, 10))
4 values = np.sin(data / 10.0)
5 print(values)
```

Пример

```
1 import numpy as np
2
3 data = np.arange(120).reshape((12, 10))
4 values = np.sin(data / 10.0)
5 p_pos = np.where(values > 0)
6 print(p_pos)
```

copyto

- Копируем данные в ndarray
- Из другого ndarray
- Или из python-коллекции
- Учитывается broadcast

Пример

```
1 import numpy as np
2
3 data = np.arange(120).reshape((12, 10))
4 print(data)
5 print(data[3:7, 2])
6 np.copyto(data[2:5, 3:7], data[3:7, 2])
7 print(data)
```

copyto

- В простом виде того же можно добиться присваиванием
- И broadcast тоже будет
- Но у copyto есть параметр where
- Можно установить маску тех позиций, где будет присваивание

Пример

```
 1 import numpy as np
 2
 3 data = np.sin(np.arange(48).reshape((8, 6)) * 3)
 4 orig = data.copy()
 5 data_2 = np.cos(np.arange(8).reshape((4, 2)))
 6 target = data[3:7, 2:4]
 7 print(target)
 8 print()
 9 print(data)
10 print()
11 np.copyto(target, data_2, where=target > 0)
12 print(data == orig)
```

reshape/ravel/flatten

- reshape позволяет указать -1 в одной позиции
- Тогда numpy сам его посчитает, если возможно
- Можно указать только -1
- Это будет интерпретация как одномерного вектора

reshape/ravel/flatten

- ravel - синоним reshape(-1)
- flatten - почти как ravel
- Только создает копию
- ravel/reshape - переинтерпретируют те же данные

Транспонирование

- reshape и товарищи занимаются переинтерпретацией данных
- Но бывают другие преобразования
- Например, на бумаге можно переписать матрицу по-другому
- Столбцы сделать строками, а строки - столбцами

Транспонирование

- Есть функция np.transpose
- И атрибут T
- Обе не копируют данные
- Только обрабатывают в другом порядке

Пример

```
1 import numpy as np
2
3 data = np.arange(36).reshape((4, -1))
4 print(data)
5 print(np.transpose(data))
6 print(data.T)
7 data.T[0, 1] = 55
8 print(data)
```

Обобщим

- Вспомним двумерную картинку
- Каждый пиксель - три элемента
- Три измерения
- Высота - первое, ширина - второе
- Цветовые каналы - третье

Обобщим

- Может быть удобно поменять измерения
- Чтобы сначала шла ширина
- Потом высота
- А потом - три компонента
- Есть функция swapaxes

Пример

```
1 import numpy as np
2
3 data = np.arange(24).reshape((2, 3, -1))
4 print(data)
5 print(np.swapaxes(data, 0, 1))
6 print(data.T)
7 data.T[0, 1] = 55
8 print(data)
```

Обобщим

- transpose - частный случай swapaxes
- В двумерном и трехмерном случае
- В двумерном - это swapaxes с параметрами 0 и 1
- В трехмерном - 0 и 2
- Но дальше это не работает

Многомерные преобразования

- Многомерное транспонирование - это переворачивание всего shape-а
- Это сложно представить наглядно
- Но легко описать формально
- Возьмем 4-мерный ndarray
- И заполним уникальными значениями

Многомерные преобразования

- И возьмем 4 индекса
- Не какие-то конкретно, а любые
- И всегда будет верно равенство
- data[a, b, c, d] == data.T[d, c, b, a]

Многомерные преобразования

- Аналогично все работает для любой размерности
- Включая 2 и 3
- А для swapaxes - будет немного подругому
- Перестановка двух индексов будет давать равенство

Пример

```
1 import numpy as np
2
3 data = np.arange(120).reshape((2, 3, 4, -1))
4 print(data.shape)
5 print(data[0, 1, 2, 3])
6 print(data.T.shape)
7 print(data.T[3, 2, 1, 0])
```

Пример

```
1 import numpy as np
2
3 data = np.arange(120).reshape((2, 3, 4, -1))
4 print(data.shape)
5 print(data[0, 1, 2, 3])
6 print(np.swapaxes(data, 0, 2).shape)
7 print(np.swapaxes(data, 0, 2)[2, 1, 0, 3])
```

Другие варианты

- Снова картинка с тремя каналами
- Хотим, чтобы сначала шли каналы
- А потом высота и ширина
- То есть сначала двумерные данные по красному каналу
- А потом по зеленому и синему

Другие варианты

- И это не swapaxes
- У нас 2 превращается в 0
- 0 - в 1, 1 - в 2
- Можно сделать 2 раза swap
- 2 с 0, потом 0 с 1

Другие варианты

- Но есть метод moveaxes
- Можно указать список размерностей
- И другой список - в какие размерности они превращаются
- Должны быть одного размера
- И уникальные (и корректные) значения в каждом

Пример

```
 1 import numpy as np
 2
 3 data = np.arange(72).reshape((4, 6, 3))
 4 print(data.shape)
 5 print(data[2, 3, :])
 6 data_2 = np.moveaxes(data, [0, 1, 2], [1, 2, 0])
 7 data_3 = np.moveaxes(data, [0, 1], [1, 2])
 8
 9 print(data_2.shape)
10 print(data_3.shape)
11
12 print(data_2[:, 2, 3])
13 print(data_3[:, 2, 3])
```

Сжатие размерностей

- Были примеры увеличения размерностей
- Это может быть нужно
- Например, для broadcasting-а
- Если больше не нужно, можно вызвать squeeze
- И убрать все размерности размером 1

Пример

```
1 import numpy as np
2
3 data = np.arange(72).reshape((4, 1, 6, 3))
4 print(data.shape)
5 print(np.squeeze(data).shape)
6
7 data = np.arange(1)
8 print(data.shape)
9 print(np.squeeze(data).shape)
```