Массивы и вычисления в NumPy

Как выполнять знакомые вычисления над целой таблицей и не перепутать её оси?

Три объекта описаны двумя признаками: [1, 2], [3, 4], [5, 6]. Нам нужны средние признаков и взвешенная сумма для каждого объекта. Сначала эти действия уже были понятны вручную; теперь запишем их операциями NumPy.

1. Список и массив делают разное

Python умеет хранить последовательность чисел в списке. Однако математические операции со списком не всегда означают то, чего ждёт человек, работающий с вектором. Выражение `[1, 2] * 2` повторяет список и даёт `[1, 2, 1, 2]`. Оно не удваивает каждое число. Для поэлементных численных расчётов удобно использовать **массив NumPy**.

Команда `import numpy as np` подключает библиотеку под коротким именем. Запись `a = np.array([1, 2])` создаёт массив. Теперь `a * 2` возвращает значения 2 и 4. Операция применяется к каждому элементу, а форма сохраняется. При этом исходный `a` не меняется, если результат не присвоить обратно.

Массив обычно хранит элементы одного численного типа. Если среди целых есть дробное число, NumPy может выбрать тип с дробной частью для всего массива. Атрибут `dtype` показывает этот выбор. Для начала достаточно различать целочисленные значения и числа с плавающей точкой. Точное число бит понадобится, когда важны память или точность.

Зачем пользоваться массивом, если цикл делает то же самое? Запись становится короче и ближе к математическому действию. Одновременно появляется риск незаметно выполнить не ту операцию над осями. Поэтому привычка проверять маленький пример и форму результата остаётся необходимой.

Пример

Обычный список удобен для разнородных записей, например `["поездка", 12]`. Числовой массив удобен для однородных измерений, когда над всеми элементами выполняются одни и те же операции.

> Массив содержит [2, 3, 6]. Каждый элемент умножили на два. Чему равна сумма результата?

2. Форма и выбор координат

Создадим `a = np.array([[1, 2], [3, 4], [5, 6]])`. Внешний список содержит три строки, каждая из двух чисел. Поэтому `a.shape` равен `(3, 2)`, а `a.ndim` равен двум: у массива две оси. Общее число элементов можно получить через `a.size`, здесь оно равно шести.

Запись `a[1, 0]` выбирает строку с индексом один и столбец с индексом ноль, то есть число три. В Python обе оси начинаются с нулевого индекса. Запись `a[1]` выбирает целую вторую строку и возвращает одномерный массив `[3, 4]`.

Двоеточие означает все позиции по оси. Поэтому `a[:, 0]` возвращает первый столбец `[1, 3, 5]`, а `a[0, :]` первую строку `[1, 2]`. Срез `a[:2]` оставляет первые две строки. Правая граница среза не включается, как у обычного списка.

Одномерный массив формы `(2,)` и матрица формы `(1, 2)` содержат одинаковое количество чисел, но различаются числом осей. Во втором случае явно есть одна строка и два столбца. Не добавляйте или удаляйте ось наугад ради исчезновения ошибки: сначала определите, означает она объекты, признаки или что-то ещё.

Заметка

`shape` возвращает кортеж размеров. Запятая в `(2,)` означает кортеж из одного элемента; это не второй размер с пропущенным числом.

> Массив NumPy равен `[[10, 20], [30, 40], [50, 60]]`. Чему равно значение по индексу `[2, 0]`?

3. Ось, по которой сворачивается сумма

Команда `a.sum()` складывает все шесть элементов и даёт 21. Но часто нужен итог по каждому признаку или по каждому объекту. Тогда следует указать ось, вдоль которой объединяются значения.

В выражении `a.mean(axis=0)` сворачивается ось строк. Для первого столбца берутся 1, 3 и 5, для второго 2, 4 и 6. Результат `[3., 4.]` содержит два средних, по одному на столбец. Точка напоминает, что результат имеет тип с дробной частью, даже если среднее получилось целым.

В выражении `a.mean(axis=1)` сворачивается ось столбцов. Каждая строка даёт отдельное среднее: 1,5; 3,5; 5,5. Результат содержит три числа, по одному на объект. Проверка длины результата помогает обнаружить перепутанную ось.

Проговорите действие без номера: «усреднить наблюдения отдельно для каждого признака». Затем сопоставьте фразу с устройством массива: наблюдения лежат строками, значит сворачиваем нулевую ось. Такой порядок полезнее, чем заучить, что `axis=0` всегда означает среднее по признакам. Если объекты уложены иначе, смысл изменится.

Обозначения

Выражение	Для массива 3 × 2
mean(axis=0)	два средних, по столбцам
mean(axis=1)	три средних, по строкам
mean()	одно среднее всех элементов

> У массива NumPy `[[2, 4], [6, 8], [10, 12]]` вычислили среднее по оси 0. Чему равно первое число результата?

Профи. Почему ось исчезает из формы и как это выдаёт перепутанную ось

Параметр `axis` называет ось, вдоль которой значения объединяются, и именно эта ось пропадает из формы результата. У массива формы $(3, 2)$ вызов `mean(axis=0)` убирает первую ось,

остаётся $(2,)$; `mean(axis=1)` убирает вторую, остаётся $(3,)$. Правило работает в любой размерности: у формы $(2, 3, 4)$ после `sum(axis=1)` остаётся $(2, 4)$, всего восемь чисел.

То же правило объясняет частую ошибку центрирования. Вычесть из каждого признака его среднее — это `a - a.mean(axis=0)`: формы $(3, 2)$ и $(2,)$ согласуются по последней оси, и вектор средних распространяется на все строки. А вот `a - a.mean(axis=1)` для той же матрицы падает: NumPy выравнивает оси справа и сравнивает 2 с 3. Исправление: сохранить свёрнутую ось единичной. `a.mean(axis=1, keepdims=True)` имеет форму $(3, 1)$, и из каждой строки вычитается её собственное среднее.

Проверка в несколько шагов для $a = [[1, 2], [3, 4], [5, 6]]$: средние по столбцам $[3, 4]$; центрированная матрица $[[-2, -2], [0, 0], [2, 2]]$; суммы по строкам $[-4, 0, 4]$. Сумма каждого столбца после центрирования равна нулю; это и есть контроль, который выдаёт перепутанную ось: при вычитании построчных средних нули появятся в суммах строк, а не столбцов.

> Массив NumPy `a = [[1, 2], [3, 4], [5, 11]]`. Вычислили `c = a - a.mean(axis=0)`. Чему равна сумма последней строки `c`, то есть `c[2].sum()`? Ответ с точностью до сотых.

> Массив NumPy `a` имеет форму `(4, 3)`: четыре объекта, три признака. Какое выражение вычтет из каждой строки её собственное среднее и не вызовет ошибку?
>
> - `a - a.mean(axis=1)`
> - `a - a.mean(axis=0)`
> - `a - a.mean()`
> - `a - a.mean(axis=1, keepdims=True)`

4. Маска и два вида умножения

Условие `a[:, 0] > 2` сравнивает каждый элемент первого столбца с двойкой. Для нашего массива получается маска `[False, True, True]`. При выборе `a[a[:, 0] > 2]` останутся две строки: `[3, 4]` и `[5, 6]`. Маска имеет одно логическое значение на строку, поэтому её длина должна совпадать с числом строк.

Знак `*` перемножает элементы, а `@` выполняет матричное произведение. Возьмём веса `w = np.array([2, 1])`. В выражении `a * w` первая колонка умножится на два, вторая на один; результат останется размера 3 на 2. NumPy применяет один вектор весов ко всем строкам. Такое согласованное распространение размеров называют broadcasting.

Выражение `a @ w` после умножения складывает вклады внутри каждой строки. Оно даёт `[4, 10, 16]`, одно число на объект. В опыте ниже можно изменять второй вес. Первый остаётся равным двум. Предскажите результат при нулевом втором весе: останутся удвоенные значения первого столбца, то есть 2, 6 и 10.

Проверьте свой прогноз

Признаки × веса = взвешенные суммы

$$1 \times 2 + 2 \times 1 = 4$$
$$3 \times 2 + 4 \times 1 = 10$$
$$5 \times 2 + 6 \times 1 = 16$$

Три объекта, два признака. Меняется второй вес; результат содержит одно число на объект.

Три строки признаков умножаются на общий вектор весов. Справа появляется одно число на объект.

Интерактивный эксперимент — в электронной версии урока.

Broadcasting удобен, но не любое сочетание размеров допустимо. В нашем случае длина весов равна числу столбцов. Вектор из трёх весов уже не описывает эти два признака. Получив ошибку размеров, напечатайте `a.shape` и `w.shape`, подпишите смысл осей и только затем исправляйте вход.

Ссылки
NumPy: broadcasting описывает точные правила согласования размеров. На первых примерах ограничьтесь числом и вектором длины, равной числу столбцов.

Профи. **Что делает @: умножить, затем свернуть ось признаков**

Матричное произведение `a @ w` для матрицы $(3, 2)$ и вектора $(2,)$ — это два уже знакомых действия подряд: поэлементное `a * w` и сумма по оси признаков `sum(axis=1)`. Для строки $[3, 4]$ и весов $[2, 1]$: произведения $[6, 4]$, сумма 10. Так получаются все три числа $[4, 10, 16]$: одно на объект, ось признаков свёрнута, ось объектов осталась.

Формы подсказывают, что сворачивается: у `@` должны совпасть внутренние размеры, последняя ось левого операнда и единственная ось вектора. Запись $(3, 2) \cdot (2,) \to (3,)$ показывает, что «двойка» исчезла. Поэтому `w @ a` с теми же данными падает: внутренние размеры 2 и 3 не совпадают. А вот если бы объектов тоже было два, `w @ a` сработало бы и молча вернуло взвешенную сумму строк вместо столбцов; форма результата эту ошибку уже не выдаст. Ручной пример на одной строке — единственная защита в таком случае.

Собрать @ из умножения и суммы

Цель: Получите [4, 10, 16] без оператора @ и подтвердите совпадение через allclose.

```
import numpy as np

a = np.array([[1, 2], [3, 4], [5, 6]])
w = np.array([2, 1])
direct = a @ w
manual = (a * w).sum(axis=0)  # TODO: сверните ось признаков, а не
объектов
same = manual.shape == direct.shape and np.allclose(direct, manual)
print(f"a @ w = {direct}; вручную = {manual}; совпало: {same}")
```

Эпик. **Когда проверка формы молчит: квадратная матрица**

Правило «длина весов равна числу столбцов» спасает, пока число объектов и число признаков различаются: неверная ось сразу даёт ошибку размеров. Контрпример: два объекта и два признака, $a = [[1, 2], [3, 4]]$. Вы хотите оставить только первый объект и пишете `a * np.array([1, 0])`, думая о весах объектов.

NumPy выравнивает оси справа, поэтому вектор применяется к столбцам: результат $[[1, 0], [3, 0]]$, обнулён второй признак, а не второй объект. Ошибки нет, форма результата $(2, 2)$ правильная, сумма элементов 4 вместо ожидаемых 3. Та же история с `a @ w` и `w @ a`: при квадратной матрице оба выражения допустимы и дают разные векторы.

Отсюда правило посильнее проверки формы: контрольный пример должен иметь разное число объектов и признаков, даже если боевые данные квадратные. Тогда перепутанная ось снова станет ошибкой, а не тихо неверным числом. Второй приём: задавать веса объектов явно формой $(n, 1)$. Массив `np.array([[1], [0]])` умножается построчно, и намерение записано в самой форме.

> Массив NumPy `a = [[1, 2], [9, 4]]`, веса `w = np.array([1, 0])`. Чему равна сумма всех элементов `a * w`, то есть `(a * w).sum()`?

5. Проверить расчёт двумя способами

В лаборатории нужно получить средние по столбцам. Стартовый код усредняет по другой оси. Найдите параметр, который отвечает за выбор, и измените его. Перед запуском ожидайте два числа, а не три. Затем сравните результат с ручными суммами: `(1 + 3 + 5) / 3` и `(2 + 4 + 6) / 3`.

Сравнение вычислений с дробными числами иногда требует допуска. Функция `np.allclose` проверяет, достаточно ли близки соответствующие значения. Причина в конечной точности представления дробей: математически одинаковые выражения могут дать чуть различающиеся последние цифры. Это не разрешение игнорировать большие ошибки и неверные формы.

Усреднить каждый признак

Цель: Получите два средних по столбцам массива.

```
import numpy as np

a = np.array([[1, 2], [3, 4], [5, 6]])
means = a.mean(axis=1)  # TODO: выберите ось наблюдений
print(f"Форма: {a.shape}; средние: {means}")
```

Есть ещё одна ловушка: обычный срез массива часто разделяет память с исходным массивом. Если взять `b = a[:, 0]` и изменить `b[0]`, изменится и соответствующий элемент `a`. Для независимой копии используйте `.copy()`. Это полезно проверять на маленьком массиве прежде, чем редактировать важные данные.

Напечатайте исходный массив перед преобразованием и после него. Если вы ожидали только новый результат, а вход изменился, проверьте присваивания и срезы. Полный повтор лаборатории заново создаёт массив, поэтому позволяет отличить действие кода от следов предыдущего опыта.

История

NumPy объединяет многомерные массивы и численные операции в общей библиотеке научного Python. О проекте. Здесь мы используем эту структуру для повторения знакомых сумм и произведений.

Подумай

Попробуйте заменить каждое число массива единицей. Средние по любой оси должны состоять из единиц. Этот контроль проверит форму, но не сможет отличить некоторые арифметические ошибки; нужен также пример с разными значениями.

Эпик. Оценка памяти: когда copy() стоит дорого

Совет «делайте `.copy()`, чтобы не испортить исходный массив» верен на шести числах и может сломать программу на настоящем наборе. Оцените порядок величины до запуска. Массив `float64` хранит каждое число в 8 байтах, поэтому объём равен произведению размеров формы на 8. Таблица из 2 миллионов наблюдений и 50 признаков занимает $2 \cdot 10^6 \cdot 50 \cdot 8 = 8 \cdot 10^8$ байт, около 800 МБ. Копия — ещё столько же; исходный массив и четыре его копии — уже 4 ГБ.

Срез-представление, которого мы остерегались ради сохранности данных, здесь становится преимуществом: `a[:, :10]` не копирует данные и занимает память только под описание окна. Перевод в `float32` уменьшает объём вдвое ценой точности около семи значащих цифр: для длин сообщений или числа поездок этого достаточно, для накопления суммы миллиардов слагаемых уже нет.

Перенос на другую ситуацию: изображение 4000×3000 пикселей с тремя каналами в типе `uint8` занимает 36 МБ, а после неосторожного `astype(float64)` — 288 МБ. Прежде чем вызывать `copy()` или менять тип, умножьте размеры формы на байты на элемент и сравните с доступной памятью.

> Массив NumPy типа `float64` содержит 2 миллиона строк и 37 столбцов. Сколько мегабайт он занимает, если один мегабайт считать равным миллиону байт?

Где это встретится дальше

Перед операцией проверьте форму массива и смысл осей. После операции проверьте форму результата и один пример вручную.

Выбрать строку, столбец и подмножество по маске.

Посчитать средние по нужной оси.

Отличить поэлементное умножение от матричного.

Домашняя работа

1. Для массива NumPy `[[2, 4], [6, 8]]` найдите средние по обеим осям.
2. Умножьте матрицу NumPy `[[2, 4], [6, 8]]` через `@` на вектор `[1, -1]`.
3. ★ В двумерном массиве NumPy переставили строки, сохранив все значения. Объясните, почему проверка только общего среднего не обнаруживает эту перестановку.

Источники

- NumPy: основы (https://numpy.org/doc/stable/user/absolute_beginners.html).
- NumPy: broadcasting (https://numpy.org/doc/stable/user/basics.broadcasting.html).
- NumPy: копии и представления (https://numpy.org/doc/stable/user/basics.copies.html).