Оптимизация вычислений в Python и большие данные

Темы на сегодня:

- Метрики расстояний и векторы
- Библиотека NumPy
- Работа с матрицами
- Линейные уравнения

МЕТРИКИ РАССТОЯНИЙ И ВЕКТОРЫ

Наиболее часто возникающий вопрос в аналитике – как сравнить некоторые объекты между собой. На этот вопрос довольно легко ответить, если это точки в пространстве. Если, например, взять 3 города, то можно построить векторы от города 1 до городов 2 и 3. А далее, зная координаты, можно посчитать длину, угол и другие параметры. Пусть эти три города: Москва, Сочи и Владивосток. Тогда:

- От Москвы до Сочи: 1631 км
- От Москвы до Владивостока: 6417 км
- Владивосток южнее чем Сочи
- Длина Сочи порядка 145 км
- 50% площади Москвы – парки и скверы
- Другие параметры

А если объект – не точка на плоскости? Разберёмся на примере фильмов «Джон Уик 3», «В бой идут одни старики» и «Сладкий ноябрь»:

Признак сравнения / Фильм	Джон Уик 3	В бой идут одни старики	Сладкий ноябрь
Боевые действия	Да	Да	Нет
Мелодрама	Нет	Нет	Да
Трагический	Нет	Да	Да
Новинка	Да	Нет	Нет
Отечественный	Нет	Да	Нет
Есть Киану Ривз	Да	Нет	Да

Для анализа таблицу транспонируют (переворачивают) и приводят к более удобному представлению:

Фильм / Признак сравнения	Боевые действия	Мелодрама	Трагический	Новинка	Отечественный	Есть Киану Ривз
Джон Уик 3	1	0	0	2019	0	1
В бой идут одни старики	1	0	1	1973	1	0
Сладкий ноябрь	0	1	1	2001	0	1

Получается, что количество совпадений при попарном сравнении фильмов следующее:

«Джон Уик 3» – «В бой идут одни старики»: 2 совпадения
«В бой идут одни старики» – «Сладкий ноябрь»: 1 совпадение
«Джон Уик 3» – «Сладкий ноябрь»: 2 совпадения

Любая таблица – по своей сути является матрицей. А из курса линейной алгебры следует, что подобный матричный вид преобразуем к векторам. Нам это необходимо для подсчёта метрик близости. Если нарисовать векторы, то ситуация будет выглядеть примерно так:

Из такого представления мы видим, что «Джон Уик 3» близок одновременно и к «Сладкому ноябрю» и к «В бой идут одни старики» при том, что последние друг от друга относительно далеко. Именно так работают алгоритмы рекомендаций: у каждого пользователя формируется определённый вектор предпочтений на основе усреднения векторов просмотренных фильмов. Алгоритм получает на вход такой вектор и сравнивает с векторами других фильмов. Те, с которыми совпадение вышло максимальным, попадают в рекомендации.

БИБЛИОТЕКА NUMPY

Как уже было сказано, для обработки таблиц удобно их представлять в виде матриц и визуализировать в виде векторов. Однако, стандартная библиотека Python не очень удобна для подобных операций. По этой причине была создана специальная библиотека **NumPy**, которая содержит в себе удобные методы работы. По умолчанию она установлена в Anaconda. В противном случае её необходимо установить отдельно. Далее – импортировать:

```
import numpy as np
```

Основным типом данных является NumPy Array – одномерный либо двухмерный массив, который задаётся следующим образом:

Пример:

```
x = np.array([1, 2, 3])
y = np.array([4, 5, 6])
print(x, y)             # array([1, 2, 3]), array([4, 5, 6])
```

Базовые операторы работают поэлементно:

Пример:

```
print(y + x)            # array([5, 7, 9])
print(y - x)            # array([3, 3, 3])
print(x * y)            # array([4, 10, 18])
print(x / y)            # array([4, 2.5, 2])
print(x ** y)           # array([4, 25, 216]). Элементы y возводятся в
                        # соответствующие степени x
print(y % x)            # array([0, 1, 0])
```

В NumPy большое количество встроенных методов, которые можно получить списком в качестве результата функции **dir(наш_Numpy_Array)**. Запоминать их все не требуется, рассмотрим лишь некоторые из них на примере.

Пример:

```
np.mean(x)              # 2.0. Среднее значение
np.arange(0, 10)        # array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9]). Аналог range
np.full(10, 1)          # array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1]). Массив размером 10,
                        # заполненный единицами
np.full([2, 5], 0)      # array([0, 0, 0, 0, 0], [0, 0, 0, 0, 0]). Двумерный массив
                        # размером 2x5, заполненная нулями
np.concatenate((x, y))  # array([1, 2, 3, 4, 5, 6]). Объединение массивов.
                        # Внимание: скобки двойные!

import random
random.shuffle(x)       # array(3, 1, 2). Перемешивание. Не является
                        # методом NumPy, но бывает полезно
```

РАБОТА С МАТРИЦАМИ

Работа с матрицами в NumPy несколько иная по сравнению с работой с обычными массивами. В частности, для задания матрицы требуются двойные квадратные скобки:

Пример

```
x = np.range(10)                          # Обычный вектор NumPy Array
y = np.array([[1, 2],[3, 4],[5, 6]])      # array([[1, 2], [3, 4], [5, 6]]). Матрица
```

Несколько методов:

- .shape – возвращает размерность матрицы в виде кортежа. Первое число – количество строк (ось ординат), второе число – количество столбцов (ось абсцисс).
- .reshape(число строк, число столбцов) – изменяет размерность матрицы. Если применено к вектору, то создаёт из его элементов соответствующую матрицу. Метод капризный: при несоответствии размерностей будет ошибка.
- .T – транспонирование матрицы.
- .ravel() – преобразование матрицы к вектору.

Пример

```
y.shape                   # (3, 2)
x.reshape(5, 2)           # array([[0, 1], [2, 3], [4, 5], [6, 7], [8, 9]])
x.reshape(3, 3)           # ValueError
y.T                       # array([[1, 3, 5], [ 2, 4, 6]])
y.ravel()                 # array([1, 2, 3, 4, 5, 6])
y.reshape(6)              # array([1, 2, 3, 4, 5, 6]). То же самое, что и ravel()
y.reshape(1, 6)           # array([[1, 2, 3, 4, 5, 6]]). Уже другой результат
```

Для создания матриц также есть ряд методов:

- .zeros(размер) – создание нулевого вектора нужного размера
- .eye(размер) – единичная матрица нужного размера
- .diag(чем_заполнить, смещение) – задание диагональной матрицы в общем виде
- .random.random(размер) – создание случайного вектора нужного размера. Каждое значение – в интервале от 0 до 1.
- .linspace(первое_значение, последнее_значение, всего_значений) – задание линейного вектора

Пример

```
np.zeros(5)                       # array([0, 0, 0, 0, 0])
np.eye(3)                         # array([[1, 0, 0], [0, 1, 0], [0, 0, 1]])
np.diag(np.arange(1,3), k=-1)     # array([[0, 0, 0, 0], [1, 0, 0, 0], [0, 2, 0, 0], [0, 0, 3, 0]])
np.random.random(3)               # array([0,12345678, 0,90123456, 0,78901234])
np.linspace(0, 2, 9)              # array([0, 0.25, 0.5, 0.75, 1, 1.25, 1.5, 1.75, 2])
```

Для более сложных распределений значений следует воспользоваться библиотекой **SciPy**, которая предоставляет большое число способов применения математического аппарата для вычислений. В Anaconda она, аналогично NumPy, предустановлена. Более подробно о ней можно прочесть по ссылке: https://docs.scipy.org/doc/.

СКАЛЯРНОЕ ПРОИЗВЕДЕНИЕ ВЕКТОРОВ И УГОЛ МЕЖДУ НИМИ

После всего рассмотренного возникает закономерный вопрос: «А как же это всё поможет для вычисления метрик?». Вспомним, что по сути, нас интересует угол между векторами. Для этого воспользуемся такой вещью как **скалярное произведение векторов**.

По определению: если a, b – векторы, то скалярное произведение представляет собой произведение модулей этих векторов (их длин) на косинус угла между ними. Причём если $a = (a_1, a_2, a_3)$, $b = (b_1, b_2, b_3)$, то оно будет равно $a_1b_1 + a_2b_2 + a_3b_3$. Вычислить можно как вручную, так и с помощью метода **.dot(вектор_1, вектор_2)**.

Пример:

```
a = np.array([4, 3])
b = np.array([2, 1])
np.dot(a, b)                    # 11
```

Можно выразить косинус искомого угла как результат деления скалярного произведения на произведение модулей векторов. Сам же угол – будет результатом взятия арккосинуса. Для векторов выше будет следующая ситуация:

Пример:

```
def cosine(a, b):
      aLength = np.linalg.norm(a)         # Вычисление длины вектора
      bLength = np.linalg.norm(b)
      return np.dot(a, b) / (aLength * bLength)

np.arccos(cosine(a, b))                   # 0,179… (в радианах)
np.arccos(cosine(a, b))*360 / 2 / np.pi   # 10,3… (в градусах)
```

Используя методы визуализации, такие как библиотеки **matplotlib**, **plotly** и другие, можно получить результат в графическом виде.

ПЕРЕМНОЖЕНИЕ МАТРИЦ И ЛИНЕЙНЫЕ УРАВНЕНИЯ

Так или иначе все операции, которые проводятся над матрицами сводятся решению так называемых **систем линейных уравнений**. Их краеугольным камнем является перемножение матриц.

Пусть есть матрицы a и b размером lxm и mxn соответственно (обязательное условие). l – количество строк, n – количество столбцов.

- $a = [[a_{11}, a_{12}, \dots a_{1m}], [a_{21}, a_{22}, \dots a_{2m}], \dots, [a_{l1}, a_{l2}, \dots a_{lm}]]$
- $b = [[b_{11}, b_{12}, \dots b_{1m}], [b_{21}, b_{22}, \dots b_{2m}], \dots, [b_{l1}, b_{l2}, \dots b_{lm}]]$

Тогда произведением матриц будет матрица lxn, где каждый элемент будет:

$$c_{ij} = \sum_{k=1}^{m} a_{ik} b_{kj}$$

Скалярное произведение векторов – частный случай перемножения матриц. По этой причине метод **.dot** и выполняет перемножение матриц.

Пример:

```
a = np.array([[1, 2], [3, 4]])
b = np.array([[5, 6], [7, 8]])
np.dot(a, b)                          # array([[19, 22], [43, 50]])
```

Есть и другой способ: с помощью типа matrix, который отличается от NumPy Array. Преобразование из NumPy Array в matrix производится с помощью метода **.mat**:

Пример:

```
aM = np.matrix([[1, 2], [3, 4]])
bM = np.mat(b)
aM * bM                               # matrix([[19, 22], [43, 50]])
```

Собственно, простейшая система линейных уравнений (СЛУ) имеет вид:

$$\begin{cases} a_{11}x + a_{12}y = b_1 \\ a_{21}x + a_{22}y = b_2 \end{cases}$$

Пример:

$$\begin{cases} x + 3y = 9 \\ 2x - 4y = 8 \end{cases}$$

```
a = np.array([[1, 3][2, -4]])
b = np.array([9, 8])
result = np.linalg.solve(a, b)          # array([6, 1]). Решение СЛУ
np.allclose(np.dot(a, result), b)       # True
```