Лабораторна робота №1. Основи роботи з NumPy

Мета: засвоїти основні відомості про роботу з фреймворком NumPy. Навчитись розпізнавати не типові елементи в даних.

Теоретичні відомості

Основні відомості про numpy.

Інсталяція:

- Для більшості систем з встановленим python3 достатньо виконати в терміналі команду:

```
pip install numpy
```

Вступ

Оскільки Python — інтерпретована мова, математичні алгоритми, часто працюють в ньому набагато повільніше ніж у компільованих мовах, таких як C або навіть Java. NumPy намагається вирішити цю проблему для великої кількості обчислювальних алгоритмів забезпечуючи підтримку багатовимірних масивів і безліч функцій і операторів для роботи з ними. Таким чином будь-який алгоритм, який може бути виражений в основному як послідовність операцій над масивами і матрицями, працює так само швидко, як еквівалентний код, написаний на C. NumPy можна розглядати як гарну вільну альтернативу MATLAB, оскільки мова програмування MATLAB зовні нагадує NumPy: обидві вони інтерпретовані, і обидві дозволяють користувачам писати швидкі програми поки більшість операцій проводяться над масивами або матрицями, а не над скалярами. Перевага MATLAB у великій кількості доступних додаткових тулбоксів, включаючи такі як пакет Simulink. Основні пакети, що доповнюють NumPy, це: SciPy — бібліотека, що додає більше MATLAB-подібної функціональності; Matplotlib — пакет для створення графіки в стилі MATLAB. Внутрішньо як MATLAB, так і NumPy базується на бібліотеці LAPACK, призначеної для вирішення основних задач лінійної алгебри.

Структура даних ndarray

Основною функціональністю NumPy є його "ndarray", для n-мірного масиву, структура даних. Ці масиви є покроковими зрізами пам'яті. На відміну від вбудованої структури даних списку Python, ці масиви однорідні: всі елементи одного масиву повинні бути одного типу.

Такі масиви також можуть бути зрізами буферів пам'яті, виділених розширеннями C / C ++, Cython та Fortran для інтерпретатора CPython, без необхідності копіювати дані, надаючи певний рівень сумісності з існуючими числовими бібліотеками. Ця функціональність використовується пакетом SciPy, який обгортає ряд таких бібліотек (зокрема BLAS та LAPACK).

Обмеження

Вставлення або додавання записів до масиву не так тривіально, як у списках Python. Процедура `np.pad(...)` для розширення масивів фактично створює нові масиви бажаної форми та значень відступів, копіює даний масив у новий і повертає його. Операція `np.concatenate([a1,a2])` NumPy насправді не пов'язує два масиви, а повертає новий, заповнений записами з обох даних масивів послідовно. Переформувати розмірність масиву за допомогою `np.reshape(...)` можливо лише за умови, що кількість елементів у масиві не змінюється. Це є наслідком того, що масиви NumPy повинні бути неперервними зрізами пам’яті. Пакет під назвою Blaze намагається подолати це обмеження. Алгоритми, які не виражаються як векторизована операція, зазвичай працюють повільно, оскільки їх потрібно реалізовувати у "чистому Python", тоді як векторизація може збільшити складність пам'яті деяких операцій від постійної до лінійної, оскільки повинні бути створені тимчасові масиви такі великі як входи. Кілька груп реалізували компіляцію числового коду під час виконання, щоб уникнути цих проблем; Рішення з відкритим кодом, які взаємодіють з NumPy, включають scipy.weave, numexpr та Numba. Cython та Pythran є альтернативами до них.

Багато сучасних широкомасштабних наукових обчислювальних програм мають вимоги, що перевищують можливості масивів NumPy. Наприклад, масиви NumPy зазвичай завантажуються в пам’ять комп’ютера, що може мати недостатній об’єм для аналізу великих наборів даних. Крім того, операції NumPy виконуються на одному центральному процесорі. Однак багато операцій лінійної алгебри можна пришвидшити, виконуючи їх на кластерах процесорів або на спеціалізованому обладнанні, такому як графічні процесори та TPU, на які покладаються багато програм глибокого навчання. В результаті за останні роки в науковій екосистемі пітона виникло кілька альтернативних реалізацій масивів, таких як Dask для розподілених масивів та TensorFlow або JAX для обчислень на графічних процесорах. Через свою популярність вони часто реалізують підмножину API Numpy або імітують його, так що користувачі можуть змінити реалізацію масиву з мінімальними змінами у своєму коді.

Приклади

Array creation

```
>>> import numpy as np
>>> x = np.array([1, 2, 3])
>>> x
array([1, 2, 3])
>>> y = np.arange(10)  # like Python's list(range(10)), but returns an
array
```

```
>>> y
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

Basic operations

```
>>> a = np.array([1, 2, 3, 6])
>>> b = np.linspace(0, 2, 4)  # create an array with four equally spaced
points starting with 0 and ending with 2.
>>> c = a - b
>>> c
array([ 1.        ,  1.33333333,  1.66666667,  4.        ])
>>> a**2
array([ 1,  4,  9, 36])
```

Universal functions

```
>>> a = np.linspace(-np.pi, np.pi, 100)
>>> b = np.sin(a)
>>> c = np.cos(a)
```

Linear algebra

```
>>> from numpy.random import rand
>>> from numpy.linalg import solve, inv
>>> a = np.array([[1, 2, 3], [3, 4, 6.7], [5, 9.0, 5]])
>>> a.transpose()
array([[ 1. ,  3. ,  5. ],
       [ 2. ,  4. ,  9. ],
       [ 3. ,  6.7,  5. ]])
>>> inv(a)
array([[-2.27683616,  0.96045198,  0.07909605],
       [ 1.04519774, -0.56497175,  0.1299435 ],
       [ 0.39548023,  0.05649718, -0.11299435]])
>>> b =  np.array([3, 2, 1])
>>> solve(a, b)  # solve the equation ax = b
array([-4.83050847,  2.13559322,  1.18644068])
>>> c = rand(3, 3) * 20  # create a 3x3 random matrix of values within
[0,1] scaled by 20
>>> c
array([[  3.98732789,   2.47702609,   4.71167924],
       [  9.24410671,   5.5240412 ,  10.6468792 ],
       [ 10.38136661,   8.44968437,  15.17639591]])
>>> np.dot(a, c)  # matrix multiplication
array([[  53.61964114,   38.8741616 ,   71.53462537],
       [ 118.4935668 ,   86.14012835,  158.40440712],
```

```
       [ 155.04043289,  104.3499231 ,  195.26228855]])
>>> a @ c # Starting with Python 3.5 and NumPy 1.10
array([[  53.61964114,   38.8741616 ,   71.53462537],
       [ 118.4935668 ,   86.14012835,  158.40440712],
       [ 155.04043289,  104.3499231 ,  195.26228855]])
```

Детальнішу документацію можна знайти за посиланням
https://numpy.org/doc/stable/user/quickstart.html

Визначення нетипових елементів. Z-score
Визначення нетипових елементів(outliers detection) - процедура визначення елементів в датасеті, які сильно відрізняються від решти елементів цього датасету. Це частий крок який використовують при обробці та аналізі даних.
Один за найпростіших алгоритмів визначення - Z-score. Z-score показує наскільки сильно відрізняється елемент даних від типового(середнього). Усі елементи які мають Z-score > 3 вважаються нетиповими.

$$Z = \left| \frac{x - \mu}{\sigma} \right|$$

x - значення ознак з набору даних
μ - середнє значення для даної ознаки
σ - стандартне відхилення

Завдання: Порахувати outliers для датасету і колонок згідно варіантів, а саме:

1. Ознайомитись з функціоналом numpy. Для цього виконати вправи з Google Notebook до рівня Intermediate включно. (додаткові 2 бали)
2. Порахувати z-score незалежно для кожної з цифрових колонок згідно варіанту. Для обрахунку використати лише numpy. NaN значення ігнорувати.
3. Агрегувати пораховані z-score (наприклад усередненням або іншим чином). Візуалізувати датасет і знайдені нетипові дані(ті, для яких агрегований z-score > 3) на декількох 2d scatter plot чи 3d scatterplot. На осях повинні знаходитись ознаки(колонки) з датасету. Нетипові дані і решту датасету візуалізувати різними кольорами. У випадку якщо нетипових елементів менше 5, обрати 5-10(на вибір студента) елементів з найвищим агрегованим Z-score як нетипові.

Варіант #	Датасет	Назви колонок з якими потрібно зробити аналіз
1	titanic	'parch','age', 'fare'
2	iris	'sepal_length', 'sepal_width', ‘petal_length’
3	planets	‘mass’, 'distance', 'orbital_period', 'year'
4	car_crashes	‘total’, ‘speeding’, ‘alcohol’
5	iris	'petal_length', 'petal_width', 'sepal_length', '
6	car_crashes	‘not_distracted’, ‘no_previous’, ‘ins_premium’
7	tips	'total_bill', 'tip', ‘size’,
8	diamonds	'carat', ‘depth’, ‘price’
9	penguins	‘bill_length_mm’, ‘bill_depth_mm’, ‘flipper_length_mm’, ‘body_mass_g’

Приклад завантаження даних:

```
#%%
import seaborn as sns
import numpy as np
import pandas as pd
#%%
tit = sns.load_dataset("titanic")
iris = sns.load_dataset("iris")
diamonds = sns.load_dataset('diamonds')
tips = sns.load_dataset('tips')
planets = sns.load_dataset('planets')


#%%
class Zscore:

   def __init__(self, np_columns: np.ndarray):


   def get_score(self, x):
       pass
```

```
    def get_average_score(self, x):
        pass

#%%
score_calc = Zscore(tit["age"])
#%%
scores = score_calc.get_score(tit["age"])
#%%
```

Питання для самоконтролю:

1. Чому обрахунки з використанням numpy швидші ніж без?
2. Чим відрізняється numpy array від Python list?
3. Як створити numpy array?
4. Що таке Z-score?
5. Які методи агрегації Z-score ви знаєте?
6. Чому дані які ви вказали як нетипові є такими?
7. Чому є багато альтернатив numpy? З чим пов’язана їхня необхідність?
8. Що таке центральна гранична теорема?
9. Чому елементи з Z-score > 3 вважаються нетиповими?
10. Чому ми можемо використовувати Z-score який базується на параметрах нормального розподілу?