Алгоритмы хранения разреженных матриц

Александр Устюжанин, 317 г.

ВМК МГУ имени М. В. Ломоносова

16 ноября 2018 г.

Overview

1. Что такое разреженная матрица?
2. Алгоритмы ее хранения
3. Заключение

Что такое разреженная матрица?

Разрежённая матрица — это матрица с преимущественно нулевыми элементами.

$$\begin{pmatrix} 1.0 & 0 & 5.0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 3.0 & 0 & 0 & 0 & 0 & 11.0 & 0 \\ 0 & 0 & 0 & 0 & 9.0 & 0 & 0 & 0 \\ 0 & 0 & 6.0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 7.0 & 0 & 0 & 0 & 0 \\ 2.0 & 0 & 0 & 0 & 0 & 10.0 & 0 & 0 \\ 0 & 0 & 0 & 8.0 & 0 & 0 & 0 & 0 \\ 0 & 4.0 & 0 & 0 & 0 & 0 & 0 & 12.0 \end{pmatrix}$$

Алгоритмы хранения разреженных матриц

- Coordinate Format (COO)
- List of Lists Format (LIL)
- Compressed Sparse Row Format (CSR)
- Dictionary of Keys Format (DOK)
- Block Compressed Row Format (BSR)

Coordinate Format (COO)

Структура:

- Хранится 3 массива: row, col, data
- row, col - индексы ненулевых элементов матрицы
- data - соответственно их значения, т.е data[i] значение элемента с координатами (row[i], col[i]).

```
from scipy.sparse import coo_matrix
row  = np.array([0, 3, 1, 0])
col  = np.array([0, 3, 1, 2])
data = np.array([4, 5, 7, 9])
coo_matrix((data, (row, col)), shape=(4, 4)).toarray()
>>> array([[4, 0, 9, 0],
           [0, 7, 0, 0],
           [0, 0, 0, 0],
           [0, 0, 0, 5]])
```

Coordinate Format (COO)

Плюсы:

- Просто осуществляются поэлементные операции.
- Позволяет дублировать входные данные.
- Быстрое преобразование в форматы CSR / CSC и обратно.

Минусы:

- Напрямую не поддерживает остальные арифметические операции и слайсинг.

Применение:

- Надо быстро построить разреженную матрицу
- Чтобы затем конвертировать в CSR или CSC format для быстрых арифметических и матричных векторных операций.

List of Lists Format (LIL)

Структура:

- Связный список списков, каждый из которых соответствует строке матрицы.
- Каждый из них представляет собой отсортированный список индексов столбцов ненулевых элементов.
- Соответствующие значения ненулевых элементов хранятся в data.

```
from scipy.sparse import lil_matrix
mtx = sparse.lil_matrix((4, 5))
data = np.round(rand(2, 3))
data
>>> array([[1., 1., 1.],
           [1., 0., 1.]])
mtx[:2, [1, 2, 3]] = data
mtx
>>> <4x5 sparse matrix of type '<... 'numpy.float64'>'
with 5 stored elements in Linked List Format>

print(mtx)
>>> (0, 1)  1.0
>>> (0, 2)  1.0
>>> (0, 3)  1.0
>>> (1, 1)  1.0
>>> (1, 3)  1.0
```

List of Lists Format (LIL)

Плюсы:

- Поддержка слайсинга
- Изменения структуры матрицы эффективны

Минусы:

- Медленные арифметические LIL – LIL операции (лучше использовать CSR или CSC)
- Медленный слайсинг столбцов (лучше использовать CSC)
- Медленные матричные векторный операции (лучше использовать CSR или CSC)

Применение:

- LIL удобен в быстром построении небольших разреженных матриц

Для арифметических и матричных векторных операций рекомендуется переводить в CSR или CSC формат.
При больших размерах матрицы рекомендуется использовать COO формат.

Compressed Sparse Row Format (CSR)

Структура:

- Хранится 3 массива: indices, indptr, data
- indices - индексы столбцов ненулевых элементов
- data - соответствующие их значения
- indptr - индексы начала новой строки относительно массива indices

Также есть аналогичный для столбцов формат CSC.

```
from scipy.sparse import csr_matrix
indptr = np.array([0, 2, 3, 6])
indices = np.array([0, 2, 2, 0, 1, 2])
data = np.array([1, 2, 3, 4, 5, 6])
csr_matrix((data, indices, indptr), shape=(3, 3)).toarray()
>>> array([[1, 0, 2],
           [0, 0, 3],
           [4, 5, 6]])
```

Compressed Sparse Row Format (CSR)

Плюсы:

- Эффективны арифметические операции CSR + CSR, CSR * CSR и т.д
- Эффективный слайсинг по строкам
- Быстрое векторное произведение матриц

Минусы:

- Медленный слайсинг столбцов (лучше использовать CSC)
- Неэффективно менять структуру матрицы (лучше LIL или DOK)

Применение:

- Арифметические и матричные векторные операции

Dictionary of Keys Format (DOK)

Структура:

- Заводится словарь с ключами (row, column) и соответствующими им ненулевыми значениями values

```
from scipy.sparse import dok_matrix
S = dok_matrix((5,5), dtype=float32)
for i in range(5):
        for j in range(5):
                S[i,j] = i+j # Update element
```

Dictionary of Keys Format (DOK)

Плюсы:

- Удобно при последовательном построении матрицы
- Доступ к элементу за O(1)
- Поддержка слайсинга
- Изменения структуры матрицы эффективны
- Быстрое преобразование в COO формат

Минусы:

- Медленные арифметические вычисления
- Не позволяет дублировать входные данные

Применение:

- Когда заранее неизвестны некоторые элементы матрицы или они изменяются

Block Compressed Row Format (BSR)

Структура:

- Хранится 3 массива: indices, indptr, data
- Матрица размера (M, N) разбивается на блоки размера (R, C) - размер должен нацело делиться
- indices - индексы столбцов с ненулевыми элементами каждого блока
- data - список матриц соответствующих значений блока размера (R, C)
- indptr - индексы начала нового блока относительно массива indices

```
from scipy.sparse import bsr_matrix
indptr = np.array([0,2,3,6])
indices = np.array([0,2,2,0,1,2])
data = np.array([1,2,3,4,5,6]).repeat(4).reshape(6,2,2)
bsr_matrix((data,indices,indptr), shape=(6,6)).todense()
>>> matrix([[1, 1, 0, 0, 2, 2],
            [1, 1, 0, 0, 2, 2],
            [0, 0, 0, 0, 3, 3],
            [0, 0, 0, 0, 3, 3],
            [4, 4, 5, 5, 6, 6],
            [4, 4, 5, 5, 6, 6]])
```

Block Compressed Row Format (BSR)

Плюсы:

- Быстрые векторные и арифметические операции
- Достигается большая эффективность вычислений по сравнению с CSR

Минусы:

- Неэффективен для разреженных блоков

Применение:

- Такие же как и у CSR
- Матрица состоит из "плотных" блоков

Заключение

- Хотим быстро построить универсальную неудобную sparse матрицу - COO format
- Хотим быстро и просто построить легко расширяемую медленно работающую sparse матрицу - LIL или DOK format
- Хотим производить вычисления над sparse матрицами - CSR/CSC или BSR format