Scientific Python

Community developed, community owned

Usage and needs for sparse data in scikit learn

Sparse Data – Meeting 1
Monday, Sept. 26th
11AM - 12PM Pacific time
Julien Jerphanion (@jjerphan)

Context: sparse data support via SciPy sparse matrices

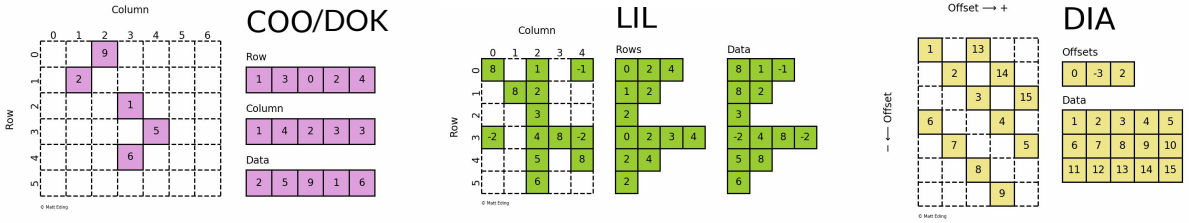
Scientific Python

Community developed, community owned

Context: sparse data support via SciPy sparse matrices

Scientific Python

Community developed, community owned



5% of use-cases, *e.g.*:

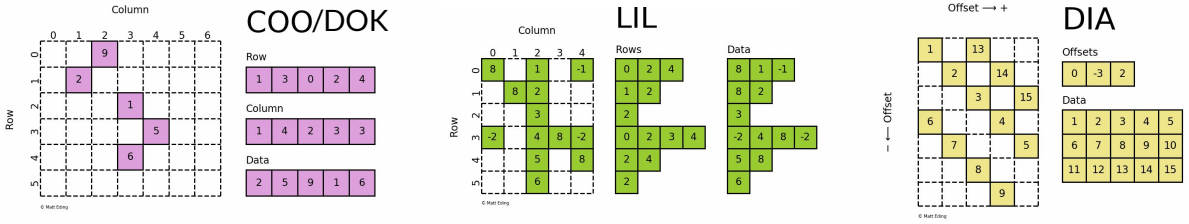
- Sparse matrices construction
- Generalized Linear Models
- `Spectral{Biclustering,Embedding}`
- `AgglomerativeClustering`

Used via SciPy's Python API.
Use-cases' needs are covered.

Animations courtesy of [Matt Eding's blog post on Sparse Matrices]

Context: sparse data support via SciPy sparse matrices

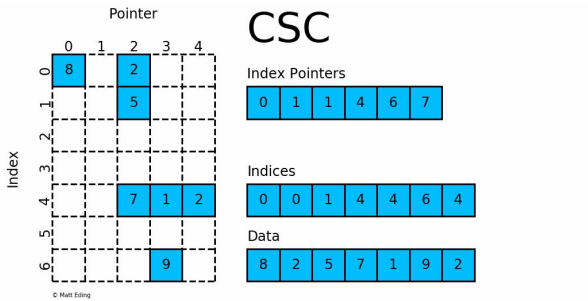
Scientific Python
Community developed, community owned

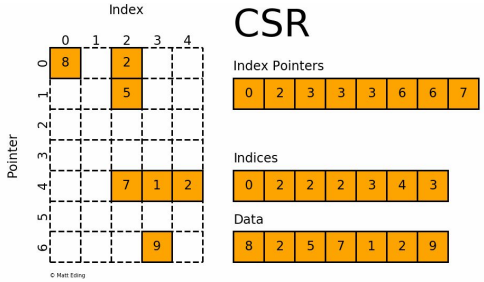


5% of use-cases, *e.g.*:

- Sparse matrices construction
- Generalized Linear Models
- `Spectral{Biclustering,Embedding}`
- `AgglomerativeClustering`

Used via SciPy's Python API.
Use-cases' needs are covered.





Used for performance:

- CSC: column-wise processing
- CSR: row-wise processing

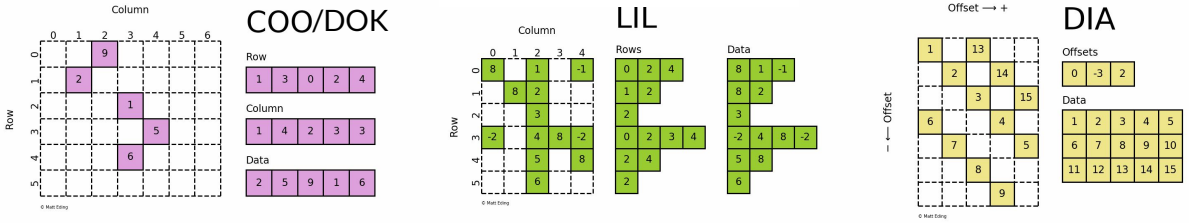
Used via:

- SciPy's Python API
- Dedicated low-level Cython routines in scikit-learn working on the arrays directly, *e.g.*: [`sklearn/utils/sparsefuncs_fast`](#) [`DistanceMetrics.(r)dist_csr`](#) [`{Dense,Sparse}²DatasetsPairs`](#)

Animations courtesy of Matt Eding's blog post on Sparse Matrices

Context: sparse data support via SciPy sparse matrices

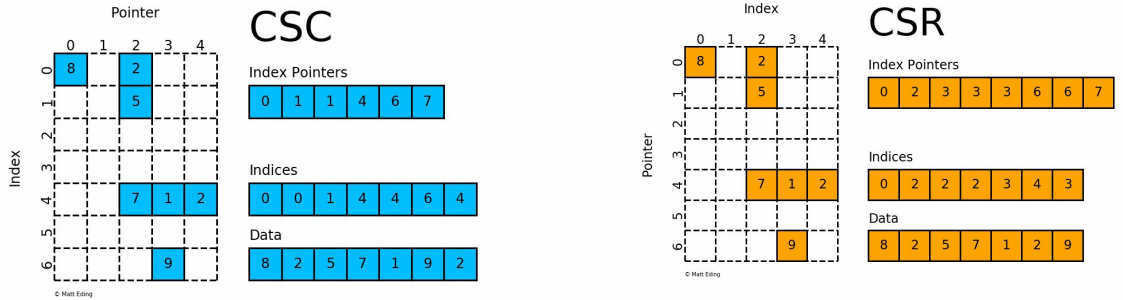
Scientific Python
Community developed, community owned



5% of use-cases, *e.g.*:

- Sparse matrices construction
- Generalized Linear Models
- `Spectral{Biclustering,Embedding}`
- `AgglomerativeClustering`

Used via SciPy's Python API.
Use-cases' needs are covered.



Used for performance:

- CSC: column-wise processing
- CSR: row-wise processing

Used via:

- SciPy's Python API
- Dedicated low-level Cython routines in scikit-learn working on the arrays directly, *e.g.*: [sklearn/utils/sparsefuncs_fast](#) [DistanceMetrics.(r)dist_csr](#) [{Dense,Sparse}²DatasetsPairs](#)

95% of usages: a lot of transformers and machine learning algorithms

Use-cases' needs can better be addressed.

Animations courtesy of Matt Eding's blog post on Sparse Matrices

Sparse-Dense matrices operations: usage and needs

Scientific Python

Community developed, community owned

Sandwich product use case in Generalized Linear Models' solvers

$$\mathbf{X}^\top \mathrm{diag}(\mathbf{d})\, \mathbf{X}$$

where $\mathbf{X}$: sparse and dense by blocks

Sandwich product use case in Generalized Linear Models' solvers

$$\mathbf{X}^\top \mathrm{diag}(\mathbf{d})\, \mathbf{X}$$

where $\mathbf{X}$: sparse and dense by blocks

Generalized Matrix-Matrix Multiplication on dense-sparse matrices' pairs

$$\mathbf{C} \leftarrow \alpha\, \mathrm{op}(\mathbf{A})\, \mathrm{op}(\mathbf{B}) + \beta\, \mathbf{C}$$

where $\mathrm{op} \in \left\{\mathrm{Id}, \cdot^\top\right\}$ $\mathbf{C}$: dense $\mathbf{A}$ $\mathbf{B}$: dense, CSR

Sandwich product use case in Generalized Linear Models' solvers

$$\mathbf{X}^\top \mathrm{diag}(\mathbf{d})\, \mathbf{X}$$

where $\mathbf{X}$: sparse and dense by blocks

Generalized Matrix-Matrix Multiplication on dense-sparse matrices' pairs

$$\mathbf{C} \leftarrow \alpha\, \mathrm{op}(\mathbf{A})\, \mathrm{op}(\mathbf{B}) + \beta\, \mathbf{C}$$

where $\mathrm{op} \in \left\{\mathrm{Id}, \cdot^\top\right\}$ $\mathbf{C}$: dense $\mathbf{A}$ $\mathbf{B}$: dense, CSR

Needs: efficient low-level implementations for such operations

Sparse-Dense matrices operations: possible solutions

- 💡 Recode the needed operations in Cython
  - ✅ Pros:
    - full control on those implementations (tailorable to our use-cases)
  - ❌ Cons:
    - maintenance complexity:
      - fused-type restrictions on Cython extension types
      - `indptr`, `indices` dtypes runtime dependence (`scipy#16774`)

Sparse-Dense matrices operations: possible solutions

Scientific Python
Community developed, community owned

💡 Recode the needed operations in Cython
- ✅ Pros:
  - full control on those implementations (tailorable to our use-cases)
- ❌ Cons:
  - maintenance complexity:
    - fused-type restrictions on Cython extension types
    - `indptr`, `indices` dtypes runtime dependence (`scipy#16774`)

💡 Vendor/use some of SciPy's private C++ routines for CSR/CSC:
- ✅ Pros:
  - efficient and stable implementations
  - `data`, `indptr`, `indices` dtypes independent
- ❌ Cons:
  - no-public C/C++/Cython API (but implementations are vendorable)
  - sandwich product not implemented

Sparse-Dense matrices operations: possible solutions

💡 Recode the needed operations in Cython
- ✅ Pros:
  - full control on those implementations (tailorable to our use-cases)
- ❌ Cons:
  - maintenance complexity:
    - fused-type restrictions on Cython extension types
    - `indptr`, `indices` dtypes runtime dependence (`scipy#16774`)

💡 Vendor/use some of SciPy's private C++ routines for CSR/CSC:
- ✅ Pros:
  - efficient and stable implementations
  - `data`, `indptr`, `indices` dtypes independent
- ❌ Cons:
  - no-public C/C++/Cython API (but implementations are vendorable)
  - sandwich product not implemented

💡 Dependent (optionally) on another library like tabmat:
- ✅ Pros:
  - efficient and stable implementations
  - support block-wise sparse and dense structure arrays
- ❌ Cons:
  - would add another dependency
  - no-public C/C++/Cython API
  - potential costly data structures' adaptations

🎯 Needs/wishes:

- API UX uniformity across NumPy and SciPy usages
  - output containers' type returning match input containers' type
  - *e.g.*: `np.{h,v}stack` supporting SciPy matrices

- Ideally multi-thread and efficient implementations of the previous operations:
  - usable via a Python API with the `@` operator
  - usable via a Cython or C API
  - based on SciPy's sparsetools (i.e. sparse matrices C++ routine)?

Proposals of discussions for the Summit

Scientific Python
Community developed, community owned

🎯 Needs/wishes:

- API UX uniformity across NumPy and SciPy usages
  - output containers' type returning match input containers' type
  - *e.g.*: `np.{h,v}stack` supporting SciPy matrices

- Ideally multi-thread and efficient implementations of the previous operations:
  - usable via a Python API with the `@` operator
  - usable via a Cython or C API
  - based on SciPy's sparsetools (i.e. sparse matrices C++ routine)?

❓ Questions:

- Regarding `indices` and `indptr` types
  - have unsigned integers historically been used for implementations?
  - does `scipy#16774` makes sense? If so, is it solvable?

- Regarding the Array API:
  - would Sparse Arrays support this standard?

Proposals of discussions for the Summit

Scientific Python
Community developed, community owned

🎯 Needs/wishes:

- API UX uniformity across NumPy and SciPy usages
  - output containers' type returning match input containers' type
  - *e.g.*: `np.{h,v}stack` supporting SciPy matrices

- Ideally multi-thread and efficient implementations of the previous operations:
  - usable via a Python API with the `@` operator
  - usable via a Cython or C API
  - based on SciPy's sparsetools (i.e. sparse matrices C++ routine)?

❓ Questions:

- Regarding `indices` and `indptr` types
  - have unsigned integers historically been used for implementations?
  - does `scipy#16774` makes sense? If so, is it solvable?

- Regarding the Array API:
  - would Sparse Arrays support this standard?

💡 Remarks:

- `pydata/sparse` from scikit-learn's perspective
  - interesting extensions of sparse matrices to n-dimensional sparse arrays
  - useful for the community
  - can it be useful for our needs?

Proposals of discussions for the Summit

Scientific Python
Community developed, community owned

🎯 Needs/wishes:

- API UX uniformity across NumPy and SciPy usages
  - output containers' type returning match input containers' type
  - *e.g.*: `np.{h,v}stack` supporting SciPy matrices

- Ideally multi-thread and efficient implementations of the previous operations:
  - usable via a Python API with the `@` operator
  - usable via a Cython or C API
  - based on SciPy's sparsetools (i.e. sparse matrices C++ routine)?

❓ Questions:

- Regarding `indices` and `indptr` types
  - have unsigned integers historically been used for implementations?
  - does `scipy#16774` makes sense? If so, is it solvable?

- Regarding the Array API:
  - would Sparse Arrays support this standard?

💡 Remarks:

- `pydata/sparse` from scikit-learn's perspective
  - interesting extensions of sparse matrices to n-dimensional sparse arrays
  - useful for the community
  - can it be useful for our needs?

Thank you for your attention!