Divisi

Learning from Semantic Networks and Sparse SVD

Rob Speer, Kenneth Arnold, and Catherine Havasi

MIT Media Lab / Mind Machine Project

June 30, 2010

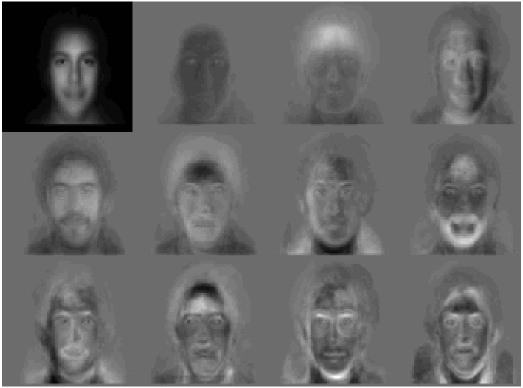
What is Divisi?

```
$ pip install divisi2 csc-pysparse
$ python
>>> from csc import divisi2
```

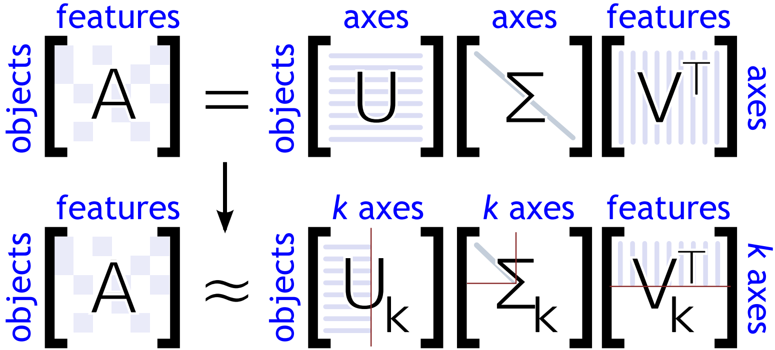
- A sparse SVD library for Python
- Includes tools for working with the results
- Keeps track of labels for what your data means
- Developed for use with AI, semantic networks
  - Used in Open Mind Common Sense project

What is SVD?

- Also known as principal component analysis
- Describes things as a sum of components, which arise from their similarity to other things



What is SVD?



features
objects
A
=
axes
objects
U
axes
Σ
features
V
T
axes
features
objects
A
≈
k axes
objects
U
k
k axes
Σ
k
features
V
T
k
k axes

Applications

- Recommender systems
- Latent semantic analysis
- Signal processing
- Image processing
- Generalizing knowledge

Dependencies

- Depends on:
  - NumPy
  - PySparse
  - NetworkX (optional)
- Uses a Cython wrapper around SVDLIBC (included)

Architecture

- Basic objects are vectors and matrices (with optional labels)
- Stored data can be sparse or dense

Modules

`csc.divisi2`	imports many useful starting points
`csc.divisi2.sparse`	SparseVector and SparseMatrix
`csc.divisi2.dense`	DenseVector and DenseMatrix
`csc.divisi2.reconstructed`	lazy matrix products
`csc.divisi2.ordered_set`	a list/set hybrid for labels
`csc.divisi2.labels`	Functions and mixins for working with labeled data
`csc.divisi2.network`	Functions for taking input from graphs, semantic networks
`csc.divisi2.dataset`	Functions for working with other pre-defined kinds of input
`csc.divisi2.fileIO`	load and save pickles, graphs, etc.
`csc.divisi2.operators`	Ufunc-like functions that preserve labels
`csc.divisi2.blending`	work with multiple datasets at once

Movie recommendations

```
>>> from csc import divisi2
>>> from csc.divisi2.dataset import movielens_ratings
>>> movie_data = divisi2.make_sparse(
      movielens_ratings('data/movielens/u')).squish(5)

>>> print movie_data
SparseMatrix (1341 by 943)
         305         6           234         63          ...
L.A. Con 4.000000    4.000000       ---      3.000000
Dr. Stra 5.000000    5.000000    4.000000       ---
Hunt For    ---         ---      3.000000       ---
Jungle B    ---      1.000000    2.000000       ---
Grease ( 3.000000       ---      3.000000       ---
```

Movie recommendations

```
>>> from csc import divisi2
>>> from csc.divisi2.dataset import movielens_ratings
>>> movie_data = divisi2.make_sparse(
      movielens_ratings('data/movielens/u')).squish(5)

>>> print movie_data
SparseMatrix (1341 by 943)
         305         6           234         63          ...
L.A. Con 4.000000    4.000000       ---      3.000000
Dr. Stra 5.000000    5.000000    4.000000       ---
Hunt For    ---         ---      3.000000       ---
Jungle B    ---      1.000000    2.000000       ---
Grease ( 3.000000       ---      3.000000       ---
```

Accessing data

```
>>> movie_data.row_labels
<OrderedSet of 1341 items like L.A. Confidential (1997)>
>>> movie_data.col_labels
<OrderedSet of 943 items like 305>
>>> movie_data[0,0]
4.0
>>> movie_data.entry_named('L.A. Confidential (1997)', 305)
4.0
```

Mean centering

Subtract out a constant "bias" from each row and column:

```
>>> movie_data2, row_shift, col_shift, total_shift =\
...   movie_data.mean_center()

>>> print movie_data2
SparseMatrix (1341 by 943)
          305        6          234        63       ...
L.A. Con  0.153996   0.053571      ---     -0.917526
Dr. Stra  1.190244   1.064838   0.542243      ---
Hunt For     ---        ---    -0.366959      ---
Jungle B     ---    -2.616438  -1.190037      ---
Grease ( -0.383420      ---    -0.181818      ---
...
```

Mean centering

Subtract out a constant "bias" from each row and column:

```
>>> movie_data2, row_shift, col_shift, total_shift =\
...   movie_data.mean_center()

>>> print movie_data2
SparseMatrix (1341 by 943)
          305       6         234       63       ...
L.A. Con  0.153996  0.053571     ---   -0.917526
Dr. Stra  1.190244  1.064838  0.542243     ---
Hunt For     ---       ---   -0.366959     ---
Jungle B     ---   -2.616438 -1.190037     ---
Grease ( -0.383420     ---   -0.181818     ---
...
```

Computing SVD results

```
>>> U, S, V = movie_data2.svd(k=100)
```

A ReconstructedMatrix multiplies the SVD factors back together lazily.

```
>>> recommendations = divisi2.reconstruct(
...     U, S, V,
...     shifts=(row_shift, col_shift, total_shift))

>>> print recommendations
<ReconstructedMatrix: 1341 by 943>

>>> print recommendations[0,0]
4.18075428957
```

Computing SVD results

```
>>> U, S, V = movie_data2.svd(k=100)
```

A ReconstructedMatrix multiplies the SVD factors back together lazily.

```
>>> recommendations = divisi2.reconstruct(
...     U, S, V,
...     shifts=(row_shift, col_shift, total_shift))

>>> print recommendations
<ReconstructedMatrix: 1341 by 943>

>>> print recommendations[0,0]
4.18075428957
```

Computing SVD results

```
>>> U, S, V = movie_data2.svd(k=100)
```

A ReconstructedMatrix multiplies the SVD factors back together lazily.

```
>>> recommendations = divisi2.reconstruct(
...     U, S, V,
...     shifts=(row_shift, col_shift, total_shift))

>>> print recommendations
<ReconstructedMatrix: 1341 by 943>

>>> print recommendations[0,0]
4.18075428957
```

Computing SVD results

```
>>> U, S, V = movie_data2.svd(k=100)
```

A ReconstructedMatrix multiplies the SVD factors back together lazily.

```
>>> recommendations = divisi2.reconstruct(
...     U, S, V,
...     shifts=(row_shift, col_shift, total_shift))

>>> print recommendations
<ReconstructedMatrix: 1341 by 943>

>>> print recommendations[0,0]
4.18075428957
```

Getting recommendations

```
>>> recs_for_5 = recommendations.col_named(5)
>>> recs_for_5.top_items(5)
[('Star Wars (1977)', 4.8162083389753922),
 ('Return of the Jedi (1983)', 4.5493663133402142),
 ('Wrong Trousers, The (1993)', 4.5292462987734297),
 ('Close Shave, A (1995)', 4.4162031221502778),
 ('Empire Strikes Back, The (1980)', 4.3923239529719762)]
```

Getting non-obvious recommendations

Use fancy indexing to select only movies the user hasn't rated.

```
>>> unrated = movie_data2.col_named(5).zero_entries()

>>> recs_for_5[unrated].top_items(5)
[('Wallace & Gromit: [...] (1996)', 4.1967566435489 8),
 ('Terminator, The (1984)', 4.1025473251923152),
 ('Casablanca (1942)', 4.0439402179346571),
 ('Pather Panchali (1955)', 4.0041287679 77936),
 ('Dr. Strangelove [...] (1963)', 3.9979437577787826)]
```

Getting non-obvious recommendations

Use fancy indexing to select only movies the user hasn't rated.

```
>>> unrated = movie_data2.col_named(5).zero_entries()

>>> recs_for_5[unrated].top_items(5)
[('Wallace & Gromit: [...] (1996)', 4.19675664354898),
 ('Terminator, The (1984)', 4.1025473251923152),
 ('Casablanca (1942)', 4.0439402179346571),
 ('Pather Panchali (1955)', 4.00412876797936),
 ('Dr. Strangelove [...] (1963)', 3.9979437577787826)]
```

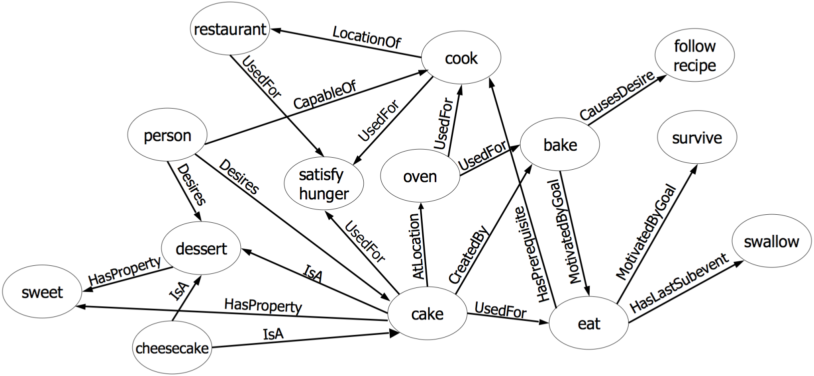
Semantic networks

- Divisi is particularly designed to take input from semantic networks
- Supports NetworkX graph format
- Divisi can find similar nodes, suggest missing links, etc.

ConceptNet

- ConceptNet is a crowdsourced semantic network of general, common sense knowledge
  - "Coffee can be located in a mug."
  - "Programmers want coffee."
  - "Coffee is used for drinking."
- We like ConceptNet, so we include a graph of it with Divisi

Sample of ConceptNet



Building a matrix from a network

```
>>> graph = divisi2.load('data:graphs/conceptnet_en.graph')

>>> from csc.divisi2.network import sparse_matrix
>>> A = sparse_matrix(graph, 'nodes', 'features', cutoff=3)

>>> print A
SparseMatrix (12564 by 19719)
         IsA/spor   IsA/game   UsedFor/   UsedFor/  ...
baseball 3.609584   2.043731   0.792481   0.500000
sport       ---     1.292481      ---     1.000000
yo-yo       ---        ---        ---        ---
toy         ---     0.500000      ---     1.160964
dog         ---        ---        ---     0.792481
...
```

Building a matrix from a network

```
>>> graph = divisi2.load('data:graphs/conceptnet_en.graph')

>>> from csc.divisi2.network import sparse_matrix
>>> A = sparse_matrix(graph, 'nodes', 'features', cutoff=3)

>>> print A
SparseMatrix (12564 by 19719)
         IsA/spor   IsA/game   UsedFor/   UsedFor/  ...
baseball 3.609584   2.043731   0.792481   0.500000
sport       ---     1.292481      ---     1.000000
yo-yo       ---        ---        ---        ---
toy         ---     0.500000      ---     1.160964
dog         ---        ---        ---     0.792481
...
```

Building a matrix from a network

```
>>> graph = divisi2.load('data:graphs/conceptnet_en.graph')

>>> from csc.divisi2.network import sparse_matrix
>>> A = sparse_matrix(graph, 'nodes', 'features', cutoff=3)

>>> print A
SparseMatrix (12564 by 19719)
         IsA/spor   IsA/game   UsedFor/   UsedFor/  ...
baseball 3.609584   2.043731   0.792481   0.500000
sport       ---     1.292481      ---     1.000000
yo-yo       ---        ---        ---        ---
toy         ---     0.500000      ---     1.160964
dog         ---        ---        ---     0.792481
...
```

Normalization (Scaled PCA)

Divisi provides `.normalize_rows()`, `.normalize_cols()`, and `.normalize_all()` methods for performing an SVD with rescaled rows and/or columns.

```
>>> U, S, V = A.normalize_all().svd(k=100)
```

Finding similar nodes

`reconstruct_similarity(`U`, `Σ`)` is a matrix that compares the rows of $U\Sigma$ using cosine similarity.

```
>>> sim = divisi2.reconstruct_similarity(U, S)

>>> sim.row_named('table').top_items()
[(u'table', 1.0), (u'dine room', 0.811), (u'gate leg table',
  0.809), (u'dine table', 0.758), (u'dine room table', 0.751),
 (u'kitchen drawer', 0.747), (u'cutlery drawer', 0.703),
 (u'sideboard', 0.698), (u'silverware drawer', 0.694),
 (u'restaurant table', 0.692)]
```

Finding similar nodes

`reconstruct_similarity(`U`, `Σ`)` is a matrix that compares the rows of $U\Sigma$ using cosine similarity.

```
>>> sim = divisi2.reconstruct_similarity(U, S)

>>> sim.row_named('table').top_items()
[(u'table', 1.0), (u'dine room', 0.811), (u'gate leg table',
  0.809), (u'dine table', 0.758), (u'dine room table', 0.751),
 (u'kitchen drawer', 0.747), (u'cutlery drawer', 0.703),
 (u'sideboard', 0.698), (u'silverware drawer', 0.694),
 (u'restaurant table', 0.692)]
```

Finding similar nodes

`reconstruct_similarity(`U`, `Σ`)` is a matrix that compares the rows of $U\Sigma$ using cosine similarity.

```
>>> sim = divisi2.reconstruct_similarity(U, S)

>>> sim.row_named('table').top_items()
[(u'table', 1.0), (u'dine room', 0.811), (u'gate leg table',
  0.809), (u'dine table', 0.758), (u'dine room table', 0.751),
 (u'kitchen drawer', 0.747), (u'cutlery drawer', 0.703),
 (u'sideboard', 0.698), (u'silverware drawer', 0.694),
 (u'restaurant table', 0.692)]
```

Making predictions

```
>>> predict = divisi2.reconstruct(U, S, V)

>>> [divisi2.labels.format_label(x) for x, value
...  in predict.row_named('learn').top_items(5)]
[u'read\\Causes', u'book\\UsedFor', u'read\\UsedFor',
 u'read magazine\\Causes', u'study\\Causes']
```

Making predictions

```
>>> predict = divisi2.reconstruct(U, S, V)

>>> [divisi2.labels.format_label(x) for x, value
...  in predict.row_named('learn').top_items(5)]
[u'read\\Causes', u'book\\UsedFor', u'read\\UsedFor',
 u'read magazine\\Causes', u'study\\Causes']
```

Suggesting new assertions

Open Mind Common Sense

Knowledge about **learn**

Similar concepts: learn study learn new education entertainment knowledge

Open Mind wants to know...

Are these statements true?

- The effect of **learning** is **be educated.**
  Yes No Sort of
- **learning** requires **go to a library**
  Yes No Sort of
- **learning about a subject** is for **learning**
  Yes No Sort of
- One of the things you do when you **learning about science** is **learn**
  Yes No Sort of
- One of the things you do when you **learn about a subject** is **learn**
  Yes No Sort of

What else does Divisi do?

- Comparing SVD predictions against test data
- Fast spreading activation
- Landmark multi-dimensional scaling (experimental)
- CCIPCA (streaming version of SVD, experimental)
- Plans for the future:
  - Non-negative Matrix Factorization
  - Integration with SciPy 0.8?

What else does Divisi do?

- Comparing SVD predictions against test data
- Fast spreading activation
- Landmark multi-dimensional scaling (experimental)
- CCIPCA (streaming version of SVD, experimental)
- Plans for the future:
    - Non-negative Matrix Factorization
    - Integration with SciPy 0.8?

Getting Divisi

- Installing: `pip install divisi2 csc-pysparse`
- Git repository:
  http://github.com/commonsense/divisi2
- Documentation:
  http://csc.media.mit.edu/docs/divisi2

We'd love your help and feedback — feel free to talk to us about Python machine learning, or find us on GitHub and help us add features!