

Concept of projection

INTERMEDIATE NETWORK ANALYSIS IN PYTHON



Eric Ma

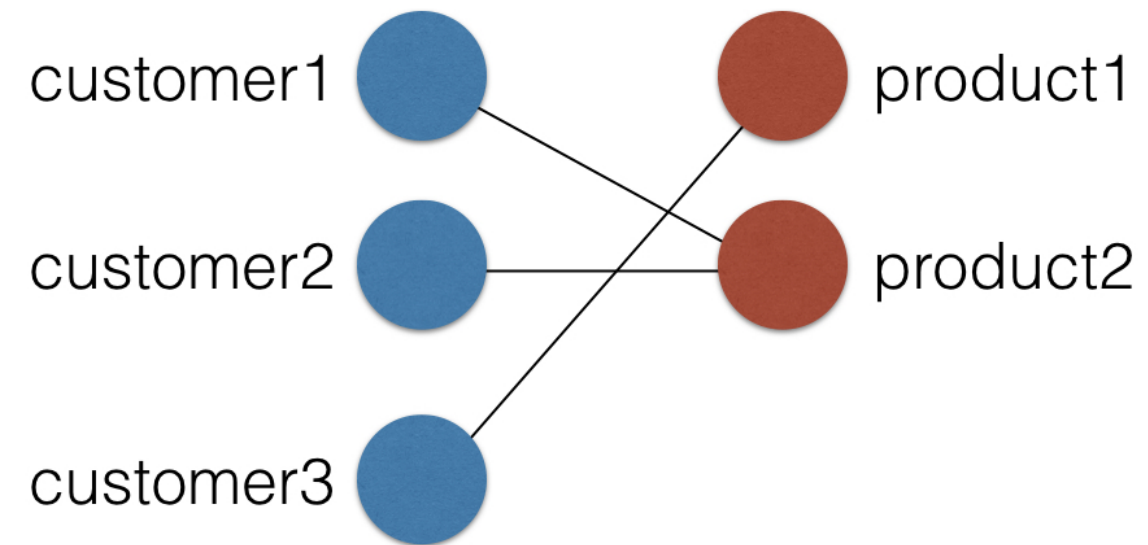
Data Carpentry instructor and author of
nxviz package

Projection

- Useful to investigate the relationships between nodes on one partition
 - Conditioned on the connections to the nodes in the other partition

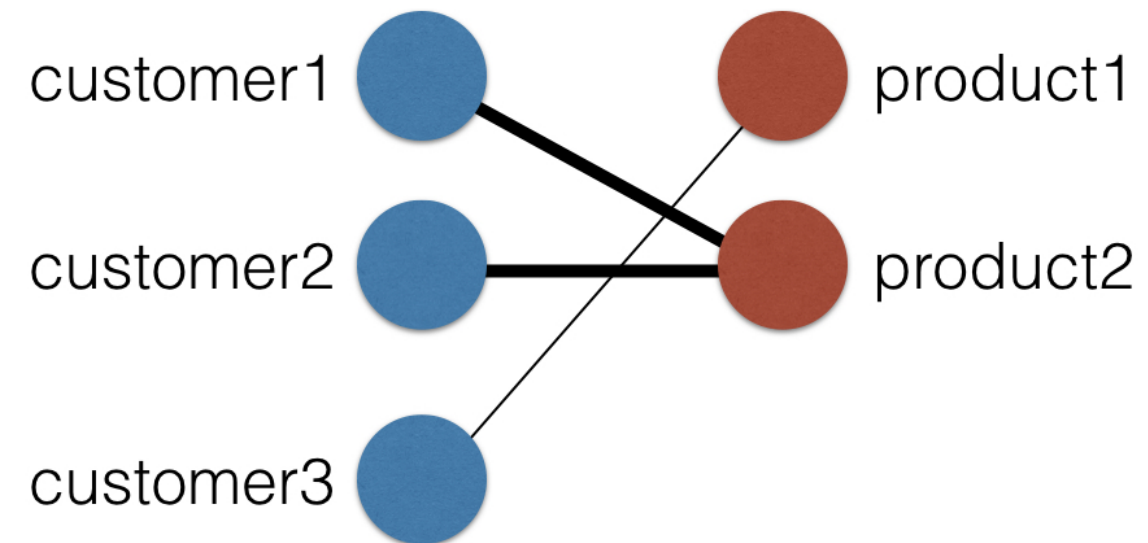
Projection

- Unipartite representation of bipartite connectivity



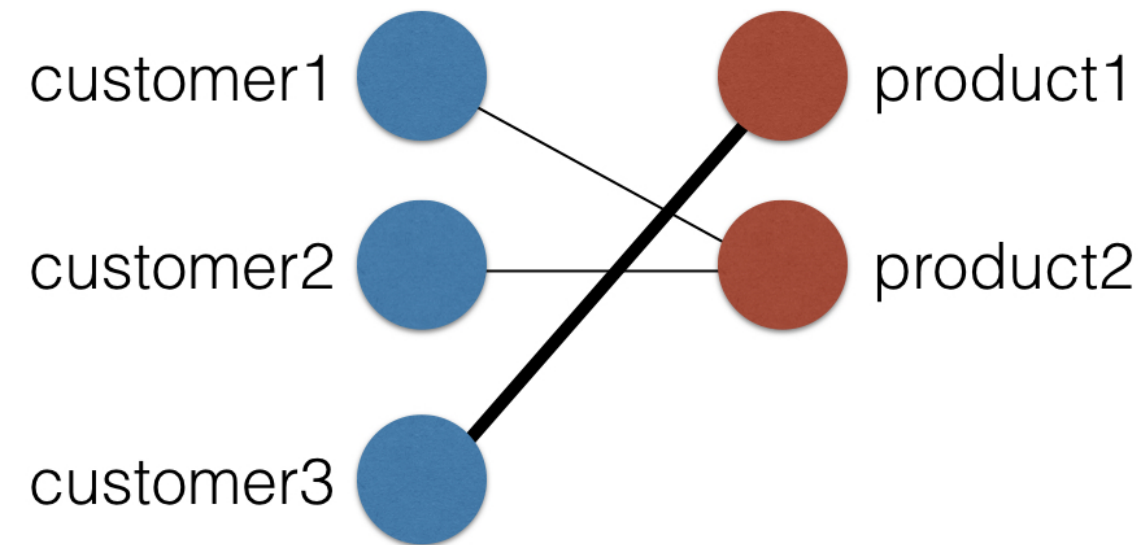
Projection

- Unipartite representation of bipartite connectivity



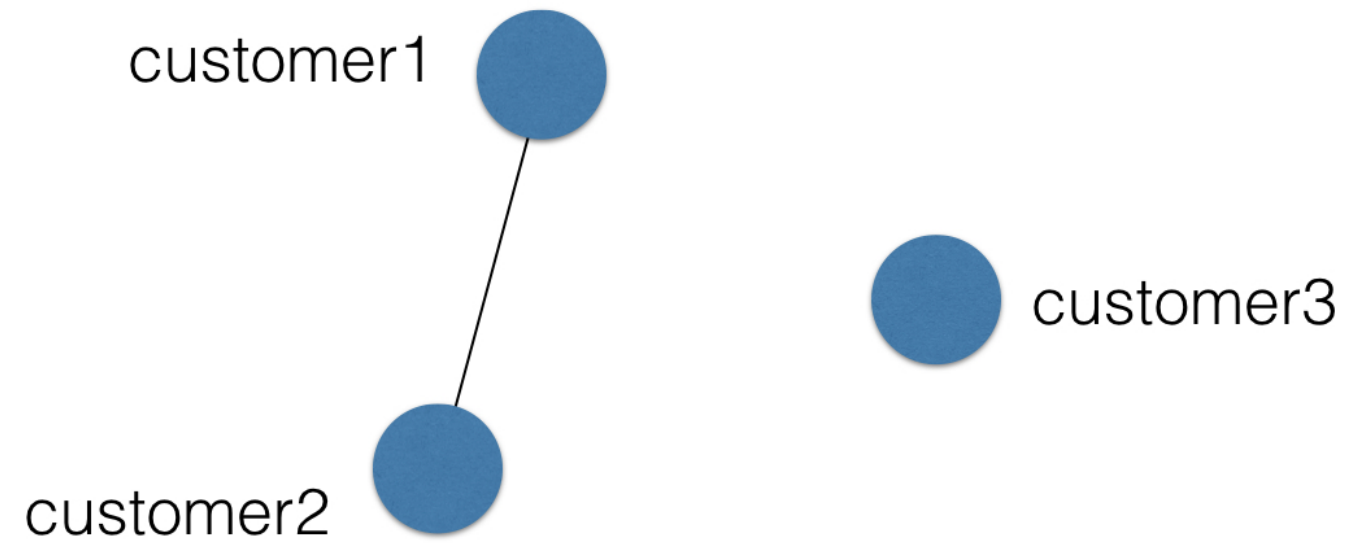
Projection

- Unipartite representation of bipartite connectivity



Projection

- Unipartite representation of bipartite connectivity



Graphs on Disk

- Flat edge lists
- CSV files: nodelist + metadata, edgelist + metadata

Reading network data

```
import networkx as nx
G = nx.read_edgelist('american-revolution.txt')
G.edges(data=True)[0:5]
```

```
[('Parkman.Elias', 'LondonEnemies', {'weight': 1}),
 ('Parkman.Elias', 'NorthCaucus', {'weight': 1}),
 ('Inglis.Alexander', 'StAndrewsLodge', {'weight': 1}),
 ('NorthCaucus', 'Chadwell.Mr', {'weight': 1}),
 ('NorthCaucus', 'Pearce.IsaacJun', {'weight': 1})]
```

- Text File

```
Barrett.Samuel LondonEnemies {'weight': 1}
Barrett.Samuel StAndrewsLodge {'weight': 1}
Marshall.Thomas LondonEnemies {'weight': 1}
Eaton.Joseph TeaParty {'weight': 1}
Bass.Henry LondonEnemies {'weight': 1}
```


Bipartite projection

```
G.nodes()
```

```
['product2', 'customer3', 'customer1', 'product3',  
 'customer2', 'product1']
```

```
G.edges()
```

```
[('product2', 'customer1'),  
 ('product2', 'customer2'),  
 ('customer3', 'product1')]
```

Bipartite projection

```
cust_nodes = [n for n in G.nodes() if G.node[n]
               ['bipartite'] == 'customers']

cust_nodes
```

```
['customer3', 'customer1', 'customer2']
```

Bipartite projection

```
G_cust = nx.bipartite.projected_graph(G, cust_nodes)  
G_cust.nodes()
```

```
['customer1', 'customer3', 'customer2']
```

```
G_cust.edges()
```

```
[('customer1', 'customer2')]
```

Degree centrality

- Recall degree centrality definition
$$\frac{\text{number of neighbors}}{\text{number of possible neighbors}}$$
- Denominator: number of nodes on opposite partition

Bipartite degree centrality

```
nx.bipartite.degree_centrality(G, cust_nodes)
```

```
{'customer1': 0.3333333333333333,  
 'customer2': 0.3333333333333333,  
 'customer3': 0.3333333333333333,  
 'product1': 0.3333333333333333,  
 'product2': 0.6666666666666666,  
 'product3': 0.0}
```

Bipartite degree centrality

```
nx.degree_centrality(G)
```

```
{'customer1': 0.2,  
 'customer2': 0.2,  
 'customer3': 0.2,  
 'product1': 0.2,  
 'product2': 0.4,  
 'product3': 0.0}
```

Let's practice!

INTERMEDIATE NETWORK ANALYSIS IN PYTHON

Bipartite graphs as matrices

INTERMEDIATE NETWORK ANALYSIS IN PYTHON



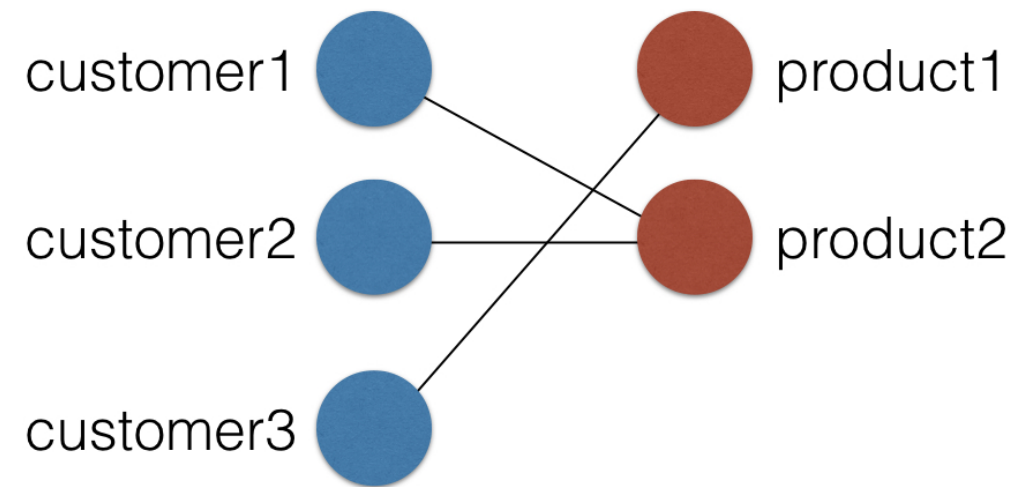
Eric Ma

Data Carpentry instructor and author of
nxviz package

Matrix representation

- Rows: nodes on one partition
- Columns: nodes on other partition
- Cells: 1 if edge present, else 0

Matrix representation



	1	2
1		
2		
3		

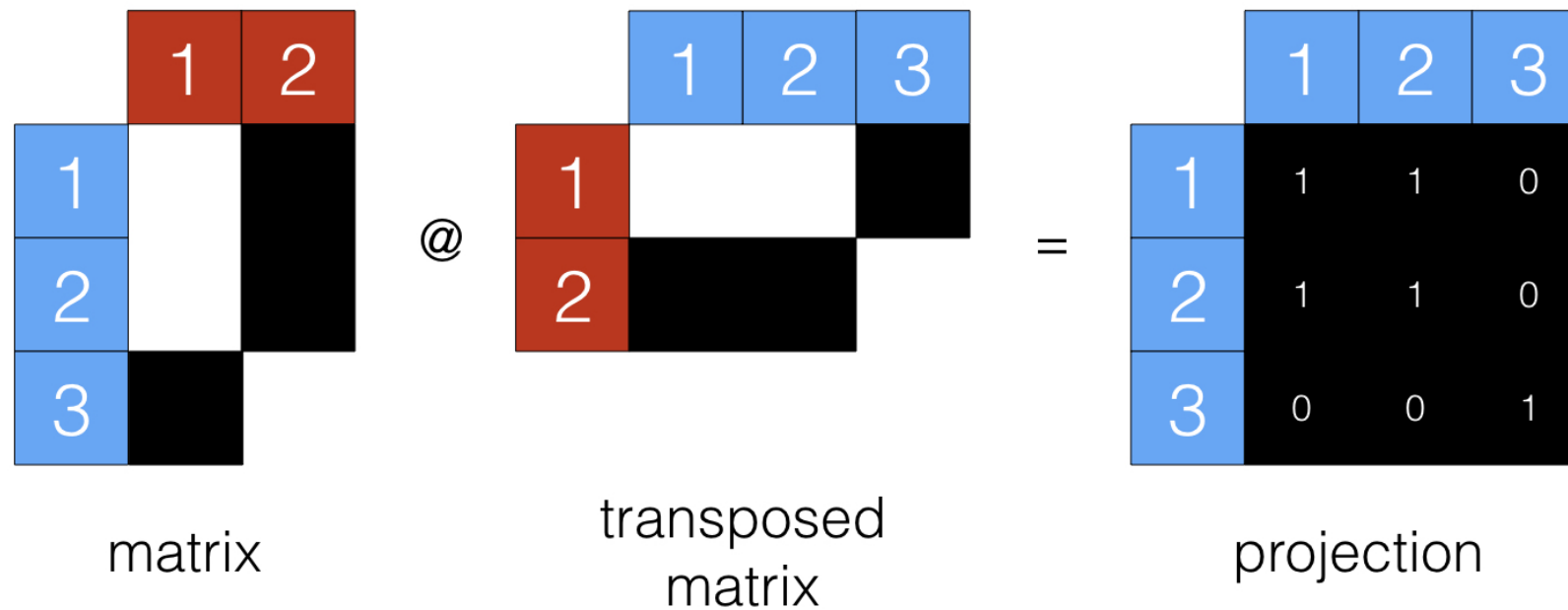
Example code

```
cust_nodes = [n for n in G.nodes() if G.node[n]
               ['bipartite'] == 'customers']
prod_nodes = [n for n in G.nodes() if G.node[n]
               ['bipartite'] == 'products']
mat = nx.bipartite.biadjacency_matrix(G,
                                     row_order=cust_nodes, column_order=prod_nodes)
mat
```

```
<3x2 sparse matrix of type '<class 'numpy.int64'>'
with 3 stored elements in Compressed Sparse Row format>
```

Matrix projection

- Projection computable using matrix multiplication



Matrix projection

- Projection computable using matrix multiplication

The diagram illustrates the matrix multiplication of two 3x3 matrices. The first matrix (left) has a blue column with values 1, 2, 3 and a red row with values 1, 2. The second matrix (middle) has a red column with values 1, 2 and a blue row with values 1, 2, 3. The result matrix (right) is a 3x3 matrix with a blue column (1, 2, 3) and a black row (1, 1, 0). The multiplication is represented by the @ symbol and the equals sign.

	1	2	
1			
2			
3			

 @

	1	2	3
1			
2			

 =

	1	2	3
1	1	1	0
2	1	1	0
3	0	0	1

Matrix projection

- Projection computable using matrix multiplication

The diagram illustrates the projection of a 3x3 matrix onto a 3x3 matrix using matrix multiplication. The first matrix (left) is a 3x3 matrix with columns 1 and 2 highlighted in red, and columns 1 and 2 highlighted in yellow. The second matrix (middle) is a 3x3 matrix with columns 1 and 2 highlighted in red, and columns 1 and 2 highlighted in yellow. The result matrix (right) is a 3x3 matrix with columns 1 and 2 highlighted in blue, and columns 1 and 2 highlighted in yellow.

	1	2	3
1	1	1	0
2	1	1	0
3	0	0	1

Matrix projection

- Projection computable using matrix multiplication

The diagram illustrates the matrix multiplication of two 3x3 matrices to produce a 3x3 result matrix. The first matrix (left) has a blue column with values 1, 2, 3 and a red row with values 1, 2. The second matrix (middle) has a red column with values 1, 2 and a blue row with values 1, 2, 3. The result matrix (right) is a 3x3 matrix with a blue column (1, 2, 3) and a yellow row (0, 0, 1). The result matrix is computed as follows:

	1	2	3
1	1	1	0
2	1	1	0
3	0	0	1

Matrix projection

- Projection computable using matrix multiplication

The diagram illustrates the projection of a graph onto a lower-dimensional space using matrix multiplication. It shows three matrices: Matrix A (3x3), Matrix B (3x3), and Matrix C (3x3).

Matrix A (Left):

	1	2
1		
2		
3		

Matrix B (Middle):

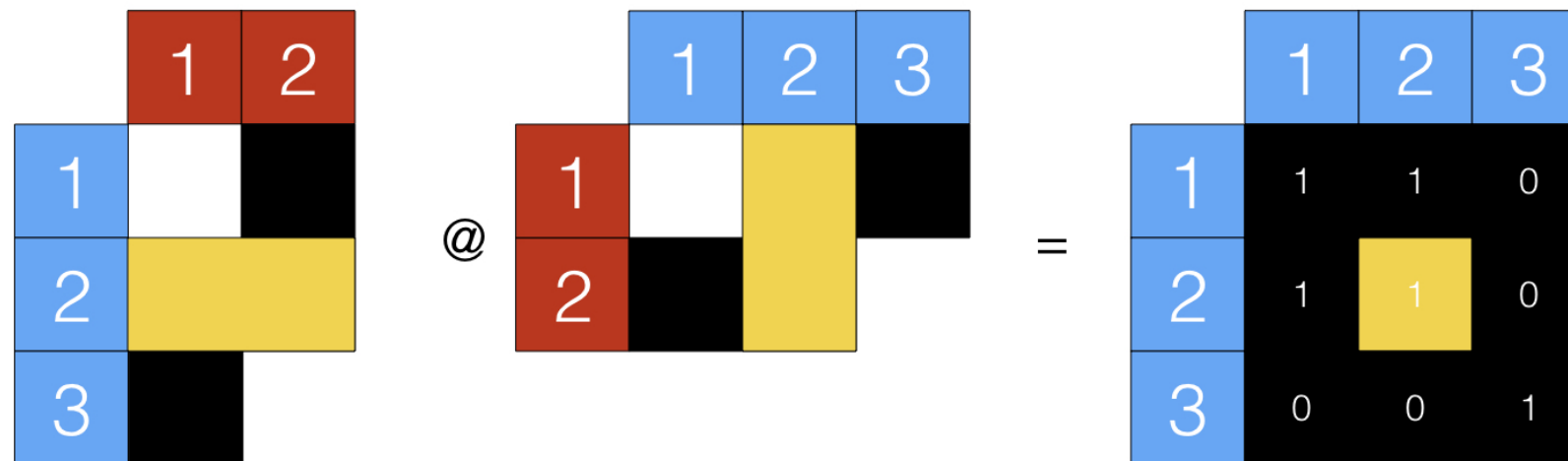
	1	2	3
1			
2			

Matrix C (Right):

	1	2	3
1	1	1	0
2	1	1	0
3	0	0	1

Matrix projection

- Projection computable using matrix multiplication



Matrix projection

- Projection computable using matrix multiplication

The diagram illustrates the matrix multiplication of two 3x3 matrices to produce a 3x3 result matrix. The first matrix has a blue column on the left with values 1, 2, 3 and a red row on top with values 1, 2. The second matrix has a red column on the left with values 1, 2 and a blue row on top with values 1, 2, 3. The result matrix is a 3x3 matrix with a blue column on the left (1, 2, 3) and a black body. The top row of the black body contains 1, 1, 0. The middle row contains 1, 1, 0 (with the last 0 in a yellow cell). The bottom row contains 0, 0, 1.

	1	2
1		
2		
3		

 @

	1	2	3
1			
2			

 =

	1	2	3
1	1	1	0
2	1	1	0
3	0	0	1

Matrix projection

- Projection computable using matrix multiplication

The diagram illustrates the matrix multiplication $A @ B = C$ for a projection operation. Matrix A is a 3x2 matrix with columns 1 and 2. Matrix B is a 2x3 matrix with columns 1, 2, and 3. Matrix C is the resulting 3x3 matrix.

	1	2
1	1	0
2	0	1
3	0	0

@

	1	2	3
1	1	0	0
2	0	1	0
3	0	0	1

=

	1	2	3
1	1	1	0
2	1	1	0
3	0	0	1

Matrix projection

- Projection computable using matrix multiplication

The diagram illustrates the matrix multiplication $A @ B = C$ for a projection operation. Matrix A is a 3x2 matrix, Matrix B is a 2x3 matrix, and Matrix C is the resulting 3x3 matrix.

Matrix A (3x2):

	1	2
1		
2		
3		

Matrix B (2x3):

	1	2	3
1			
2			

Matrix C (3x3):

	1	2	3
1	1	1	0
2	1	1	0
3	0	0	1

Matrix projection

- Projection computable using matrix multiplication

The diagram illustrates the matrix multiplication of two 3x3 matrices to produce a 3x3 result matrix. The first matrix has a blue column on the left with values 1, 2, 3 and a red row on top with values 1, 2. The second matrix has a red column on the left with values 1, 2 and a blue row on top with values 1, 2, 3. The result matrix is a 3x3 matrix with a blue column on the left (1, 2, 3) and a yellow cell at the bottom right (3,3). The rest of the matrix is black, with white numbers 1 in the cells (1,1), (1,2), (2,1), and (2,2).

	1	2
1		
2		
3		

 @

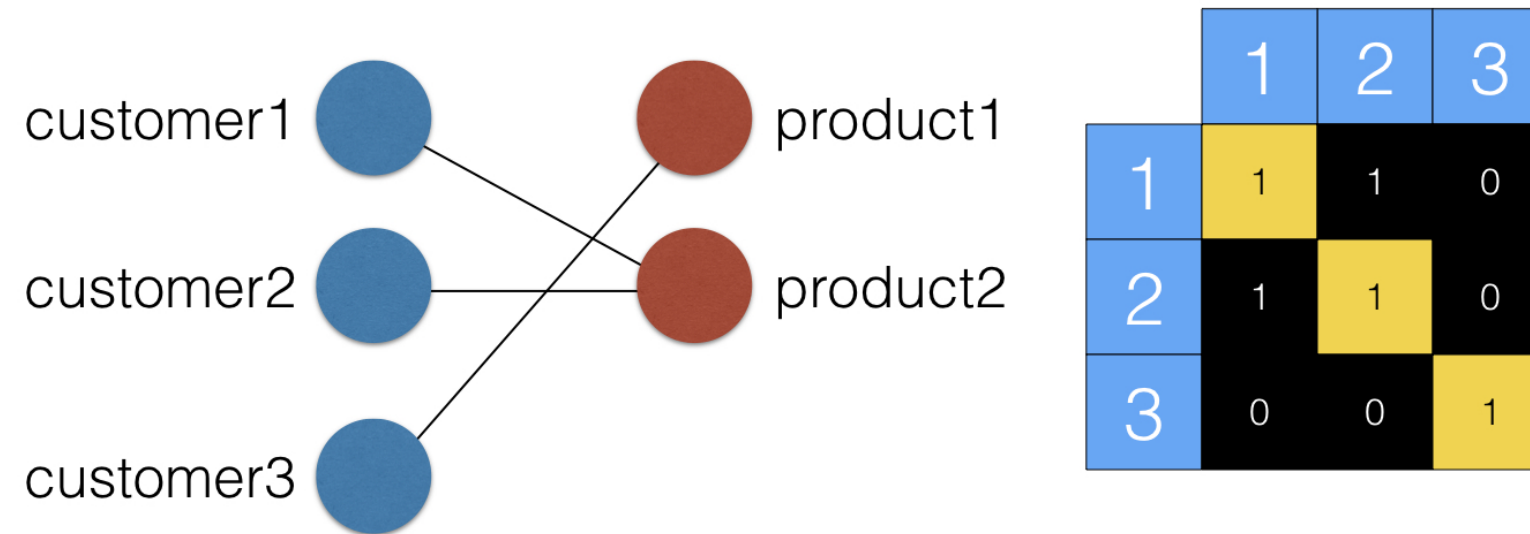
	1	2	3
1			
2			

 =

	1	2	3
1	1	1	0
2	1	1	0
3	0	0	1

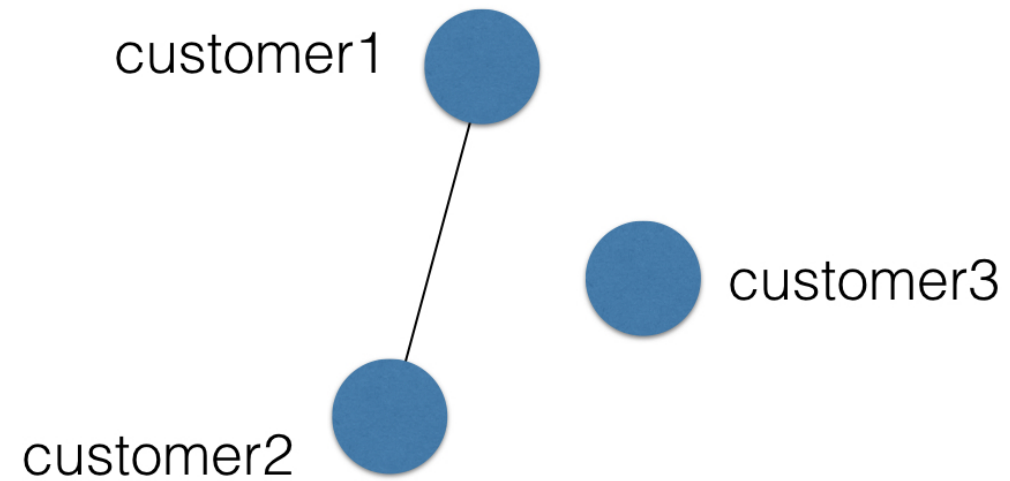
Matrix projection

- Projection computable using matrix multiplication



Matrix projection

- Projection computable using matrix multiplication



	1	2	3
1	1	1	0
2	1	1	0
3	0	0	1

Matrix multiplication in Python

```
mat @ mat.T
```

```
<5x5 sparse matrix of type '<class 'numpy.int64'>'  
with 23 stored elements in Compressed Sparse Row format>
```

```
mat.T @ mat
```

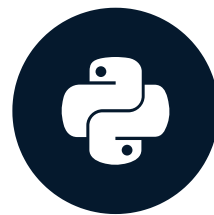
```
<10x10 sparse matrix of type '<class 'numpy.int64'>'  
with 50 stored elements in Compressed Sparse Column format>
```


Let's practice!

INTERMEDIATE NETWORK ANALYSIS IN PYTHON

Representing network data with pandas

INTERMEDIATE NETWORK ANALYSIS IN PYTHON



Eric Ma

Data Carpentry instructor and author of
nxviz package

CSV files for network data storage

- CSV File

```
person,party,weight  
Barrett.Samuel,LondonEnemies,1  
Barrett.Samuel,StAndrewsLodge,1  
Marshall.Thomas,LondonEnemies,1  
Eaton.Joseph,TeaParty,1  
Bass.Henry,LondonEnemies,1
```

CSV files for network data storage

- Advantages:
 - Human-readable
 - Do further analysis with pandas
- Disadvantages:
 - Repetitive; disk space
- Two DataFrames: node and edge lists

Node list and edge list

- Node list
 - Each row is one node
 - The columns represent metadata attached to that node
- Edge list
 - Each row is one edge
 - The columns represent the metadata attached to that edge

Pandas and graphs

```
G.nodes(data=True)
```

```
[(0, {'bipartite': 0}),  
(1, {'bipartite': 0}),  
(2, {'bipartite': 0}),  
...]
```

```
odelist = []  
for n, d in G.nodes(data=True):  
    node_data = dict()  
    node_data['node'] = n  
    node_data.update(d)  
    oodelist.append(node_data)
```

Pandas and graphs

odelist

```
[{'bipartite': 0, 'node': 0},  
{ 'bipartite': 0, 'node': 1},  
{ 'bipartite': 0, 'node': 2},  
{ 'bipartite': 0, 'node': 3},  
{ 'bipartite': 0, 'node': 4},...]
```

Pandas and graphs

```
import pandas as pd  
pd.DataFrame(nodelist)
```

```
   bipartite  node  
0          0     0  
1          0     1  
2          0     2  
3          0     3  
4          0     4  
5          1     5  
6          1     6  
7          1     7
```

```
pd.DataFrame(nodelist).to_csv('my_file.csv')
```


Let's practice!

INTERMEDIATE NETWORK ANALYSIS IN PYTHON