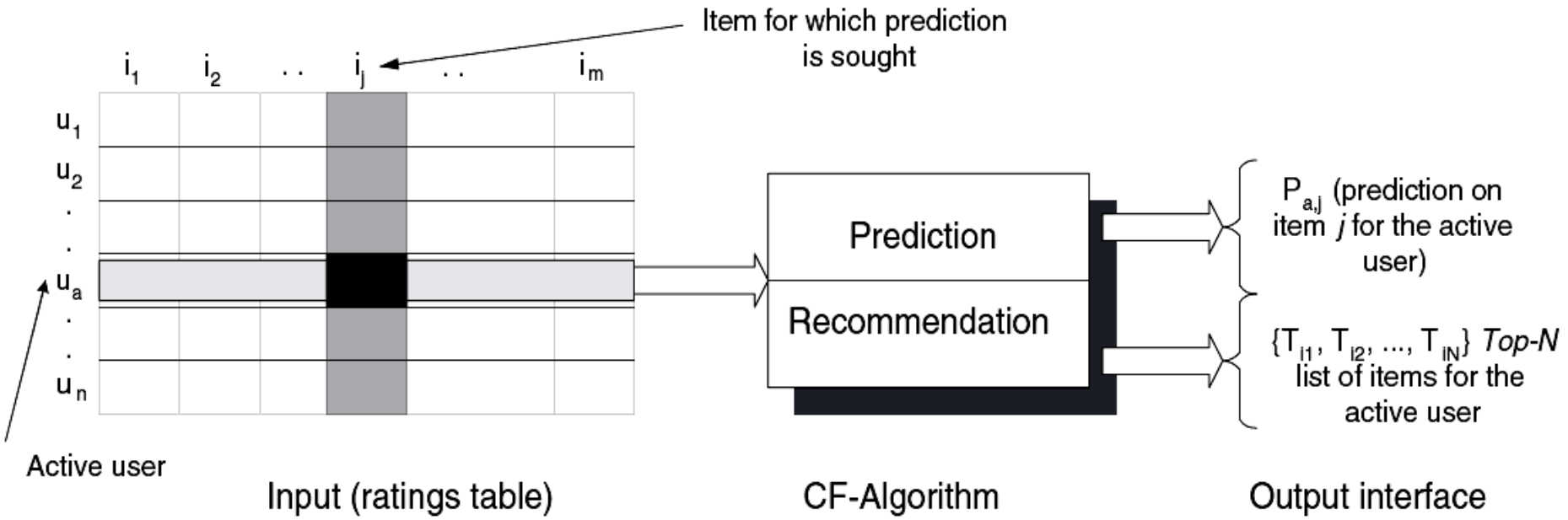
Recommender Systems

Based on: Toby Segaran: Programming Collective Intelligence

Collaborative Filtering Process



Challenge - Sparsity

- Active users may have purchased well under 1% of the items
  - E.g. 1% of 2 million books is 20,000 books.

- **Solution**: Use sparse representations of the rating matrix.

Ratings in a hashtable - Example

```
critics = {
    'Lisa Rose':  {'Lady in the Water': 2.5,
                   'Snakes on a Plane': 3.5,
                   'Just my Luck': 3.0,
                   'Superman Returns': 3.5,
                   'You, Me and Dupree': 2.5,
                   'The Night Listener': 3.0},

    'Gene Seymour': {'Lady in the Water': 3.0,
                   'Snakes on a Plane': 3.5,
                   'Just my Luck': 1.5,
                   'Superman Returns': 5.0,
                   'The Night Listener': 3.0,
                   'You, Me and Dupree': 3.5},
```

Ratings in a hashtable

```
'Michael Phillips': {'Lady in the Water': 2.5,
                     'Snakes on a Plane': 3.0,
                     'Superman Returns': 3.5,
                     'The Night Listener': 4.0},
'Claudia Puig':     {'Snakes on a Plane': 3.5,
                     'Just my Luck': 3.0,
                     'The Night Listener': 4.5,
                     'Superman Returns': 4.0,
                     'You, Me and Dupree': 2.5},
'Mick LaSalle':     {'Lady in the Water': 3.0,
                     'Snakes on a Plane': 4.0,
                     'Just my Luck': 2.0,
                     'Superman Returns': 3.0,
                     'The Night Listener': 3.0,
                     'You, Me and Dupree': 2.0},
```

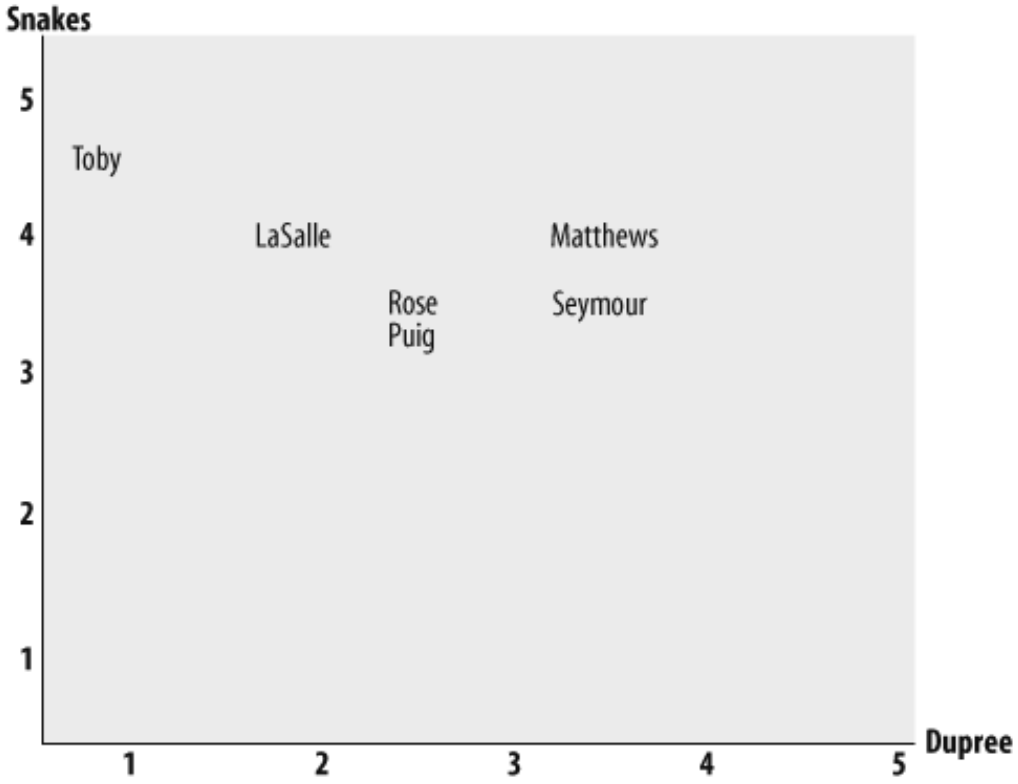
Ratings in a hashtable

```
'Jack Matthews':    {'Lady in the Water': 3.0,
                     'Snakes on a Plane': 4.0,
                     'Superman Returns': 5.0,
                     'The Night Listener': 3.0,
                     'You, Me and Dupree': 3.5},

'Toby':             {'Snakes on a Plane': 4.5,
                     'Superman Returns': 4.0,
                     'You, Me and Dupree': 1.0}
}
```

Finding Similar Users

- Simple way to calculate a similarity score is to use Euclidean distance, which considers the items that people have ranked in common
  - **each user is represented as a vector**.



People in preference space (assuming two movies)

Euclidean Distance

- Suppose we have two vectors of ratings:

$$\mathbf{x} = [x_1, \ldots, x_m]$$
$$\mathbf{y} = [y_1, \ldots, y_m]$$

- Then we can measure their similarity by the Euclidean distance:

$$sim_{\mathbf{x},\mathbf{y}} = \sqrt{\sum_{i=1}^{m} (x_i - y_i)^2}$$

Only the *i*'s for which both **x** and **y** have known ratings participate

Problem with Euclidean Distance

E.g.,

- suppose a critic rated five movies by 1, 2, 3, 5, 8.
- and another critic rated those movies by .01, .02, .03, .05, .08.

- Obviously they are quite similar in their relative tastes, yet their Euclidean distance is big.

Fix

E.g.,

- suppose a user rated five movies by 1, 2, 3, 5, 8.
- and another user rated those movies by .01, .02, .03, .05, .08.

- We can consider vectors
  x=[1, 2, 3, 5, 8] and **y**=[.01, .02, .03, .05, .08] and see that the **angle** (theta) between them is 0 degrees,
  i.e. they are very similar **direction-wise**.

- So, we can employ the **cosine** of theta:
  - the greater the cosine, the closer to 0 degrees theta is,
  - i.e. the more similar the two rating vectors are,
  - i.e. the more similar the users are.

$$sim_{\mathbf{x},\mathbf{y}} = \cos\theta = \frac{\mathbf{x}\cdot\mathbf{y}}{\|\mathbf{x}\|\|\mathbf{y}\|} = \frac{1.03}{\sqrt{103}\sqrt{0.0103}} = 1$$

The greatest value **cos** can have, which happens for theta=0.

However…

E.g.,

- suppose a user rated five movies by 1, 2, 3, 5, 8.
- and another user rated those movies by .01, .02, .03, .05, .08.

- Now suppose the second user, seeing he has been too harsh, increases by 0.1 all his ratings. So, the vectors are now

  x=[1, 2, 3, 5, 8] and **y**=[.11, .12, .13, .15, .18]

- They are still very similar, but cosine similarity will get confused:

$$sim_{\mathbf{x},\mathbf{y}} = \cos\theta = \frac{\mathbf{x}\cdot\mathbf{y}}{\|\mathbf{x}\|\|\mathbf{y}\|} = \frac{2.93}{\sqrt{103}\sqrt{0.0983}} = 0.92$$

Their similarity is **reduced**, while it should have been (intuitively) **invariant**.

Better Fix: Pearson Correlation

- Recall the vectors are:

  x=[1, 2, 3, 5, 8] and **y**=[.11, .12, .13, .15, .18]

- First **centralize** by subtracting their mean, then compute cosine similarity.
- m_x = (1 + 2 + 3 + 5 + 8)/5 = 3.8
- m_y = (.11 + .12 + .13 + .15 + .18)/5 = .138

x'=[1-3.8, 2-3.8, 3-3.8, 5-3.8, 8-3.8] =

[−2.8, −1.8, −0.8, 1.2, 4.2]

y'=[.11-.138, .12-.138, .13-.138, .15-.138, .18-.138] =

[−0.028, −0.018, −0.008, 0.012, 0.042]

$$sim_{x,y} = \cos\theta = \frac{\mathbf{x}'\cdot\mathbf{y}'}{\|\mathbf{x}'\|\|\mathbf{y}'\|} = \frac{0.308}{\sqrt{30.8}\sqrt{0.00308}} = 1$$

as we intuitively expect.

This is called **Pearson Correlation Coefficient**.

Pearson Correlation Formula

$$\mathbf{x} = [x_1, \ldots, x_m]$$
$$\mathbf{y} = [y_1, \ldots, y_m]$$

- The formula more detailed is:

$$sim_{\mathbf{x},\mathbf{y}} = \frac{\sum_{i=1}^{m} (x_i - \bar{x}) \cdot (y_i - \bar{y})}{\sqrt{\sum_{i=1}^{m} (x_i - \bar{x})^2} \cdot \sqrt{\sum_{i=1}^{m} (y_i - \bar{y})^2}}$$

Only the *i*'s for which both **x** and **y** have known ratings participate

Pearson Correlation Numbers

- The correlation coefficient is always between -1 and +1.
- The closer the correlation is to +/-1, the closer to a perfect linear relationship. **E.g.**

-1.0 to -0.7 strong negative association.

-0.7 to -0.3 weak negative association.

-0.3 to +0.3 little or no association.

+0.3 to +0.7 weak positive association.

+0.7 to +1.0 strong positive association.

How to use similarity?

- To recommend item *i* to user *u*, predict how *u* would have rated *i* by computing a **weighted average** of the ratings that other users have given to *i*.

- The weights are the similarities.
  - The more similar a user *v* is to *u*, the bigger the weight for *v*'s rating of *i*.

$$\hat{r}_{u,i} = \frac{\sum_{v \in U_i} r_{v,i} \cdot sim_{v,u}}{\sum_{v \in U_i} sim_{v,u}}$$

U_i is the set of users who have rated *i*.

This is using "user-user" similarities

Recommendation Example

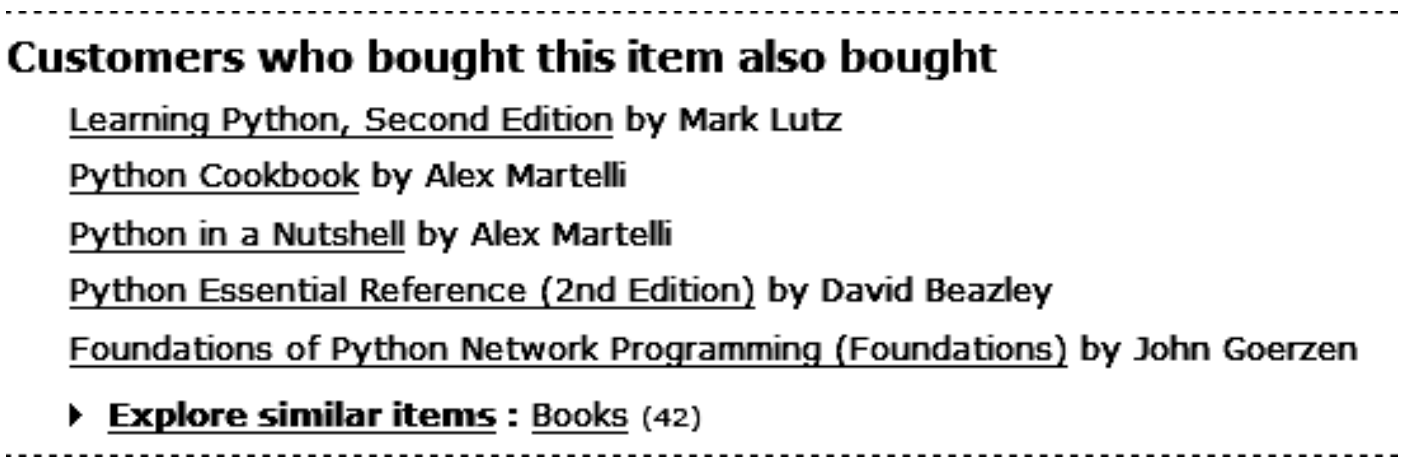
Critic	Similarity	Night	S.xNight	Lady	S.xLady	Luck	S.xLuck
Rose	0.99	3.0	2.97	2.5	2.48	3.0	2.97
Seymour	0.38	3.0	1.14	3.0	1.14	1.5	0.57
Puig	0.89	4.5	4.02			3.0	2.68
LaSalle	0.92	3.0	2.77	3.0	2.77	2.0	1.85
Matthews	0.66	3.0	1.99	3.0	1.99		
Total			12.89		8.38		8.07
Sim.Sum			3.84		2.95		3.18
Total/Sim. Sum			3.35		2.83		2.53

We are trying to recommend for user Toby.

The highest predicted rating for Toby is for movie “The night listener”

Matching Products

- Recall Amazon…



Transform the data

{'Lisa Rose': {'Lady in the Water': 2.5, 'Snakes on a Plane': 3.5},
'Gene Seymour': {'Lady in the Water': 3.0, 'Snakes on a Plane': 3.5}}

to:

{'Lady in the Water':{'Lisa Rose':2.5,'Gene Seymour':3.0},
'Snakes on a Plane':{'Lisa Rose':3.5,'Gene Seymour':3.5}}
etc..

Then, compute similarities between items, rather than users.

Getting Similar Items

```
>> movies=recommendations.transformPrefs(recommendations.critics)
>> recommendations.topMatches(movies,'Superman Returns')
    [(0.657, 'You, Me and Dupree'),
     (0.487, 'Lady in the Water'),
     (0.111, 'Snakes on a Plane'),
     (-0.179, 'The Night Listener'),
     (-0.422, 'Just My Luck')]
```

Whom to invite to a premiere?

>>**recommendations.getRecommendations(**
movies,'Just My Luck')

[(4.0, 'Michael Phillips'),
(3.0, 'Jack Matthews')]

- For another example, reversing the products with the people, as done here, would allow an online retailer to search for people who might buy certain products.

Recommendations using item-item similarities

$$\hat{r}_{u,i} = \frac{\sum_{j \in I_u} r_{u,j} \cdot sim_{i,j}}{\sum_{j \in I_u} sim_{i,j}}$$

I_u is the set of items rated by user u.

This is using "item-item" similarities

Pros and Cons

- **User-user**:
  - Diversified recommendations.
  - Eccentric users will not get good recommendations.

- **Item-item**:
  - Similarities more reliable between items (unless item is niche).
  - Eccentric users will get better recommendations.

Performance: Building a Cache

Create a dictionary of items showing which other items they are most similar to.

Evaluating Rec. Systems

- For each existing rating, hide it and try to predict it.
- Compute the average absolute error.

$$RMSE = \sqrt{\frac{\sum_{u \in U, i \in I} \left(r_{u,i} - \hat{r}_{u,i} \right)^2}{N}}$$