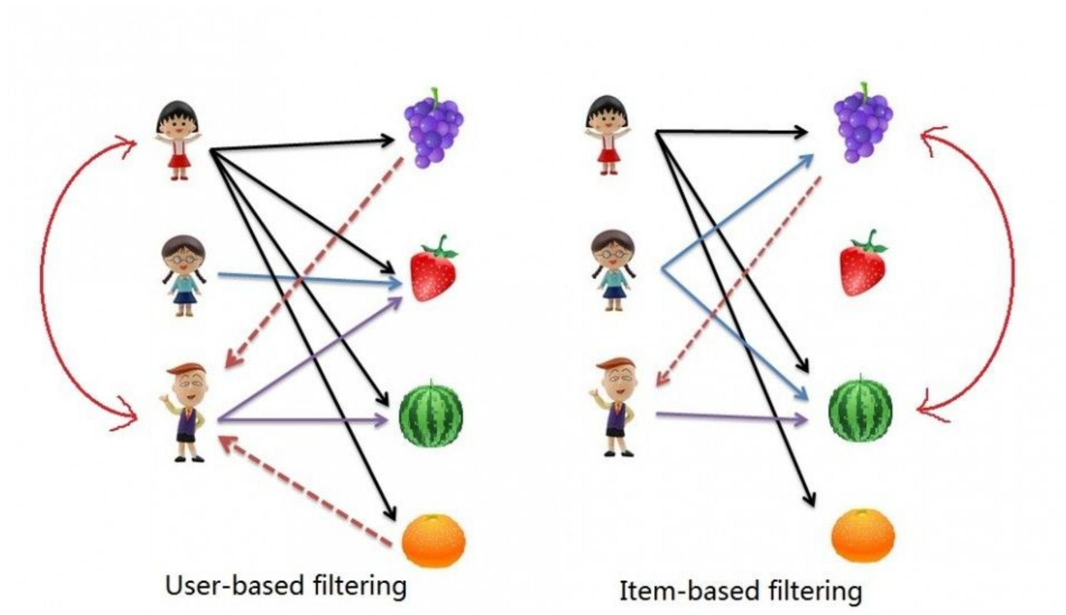
Lecture 7 - Introduction to Recommendation Engines

Lecture 8

Outline

- Setup of the problem
- User-User vs Item-Item Recommendations - Fruit Example
- User-User vs Item-Item for Music Recommendations
- Diffusion on Bipartite Graphs
- Model Evaluation and Regularization

User vs Item Based Filtering

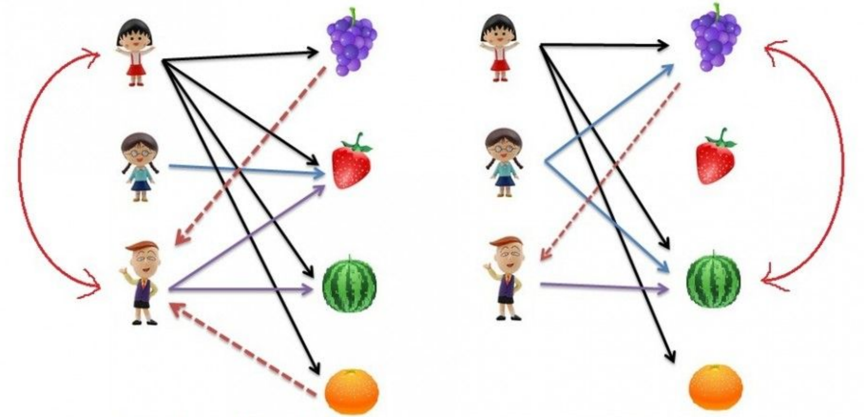


- The first step is to make pairings between users and items, ie. list all users as the rows in a matrix with columns the items.
- User-User based filtering finds similar users based on interest/purchases.
- Item-Item based filtering pairs similar items based on interest/purchases.

Making an adjacency matrix

$$X = \begin{bmatrix} (\text{user} = 1, \text{grapes} = \text{Yes}) & (\text{user} = 1, \text{apples} = \text{No}) & \cdots & \\ (\text{user} = 2, \text{grapes} = \text{Yes}) & (\text{user} = 2, \text{apples} = \text{Yes}) & \cdots & \cdots \\ \cdots & & & \end{bmatrix}$$



User-based filtering

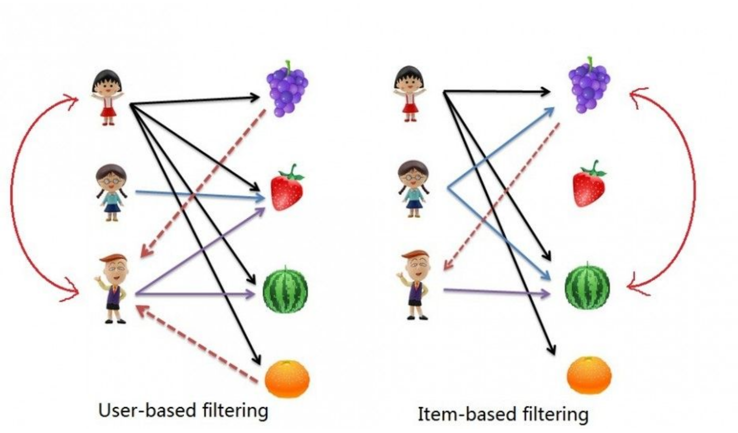
Item-based filtering

Generally X is referred to as an **adjacency matrix** and has as rows the users, and as columns the items.

Let’s assume we have **n users** and **p items.** Generally **n is much larger than p.**

The covariance matrix revisited - **Item/Item**

$$X^T X = \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \\ \dots \\ \mathbf{x_p} \end{bmatrix} \begin{bmatrix} \mathbf{x}_1 & \mathbf{x}_2 & \cdots & \mathbf{x}_p \end{bmatrix}$$



$$\mathbf{x_i} = \text{ purchase history for item i}$$

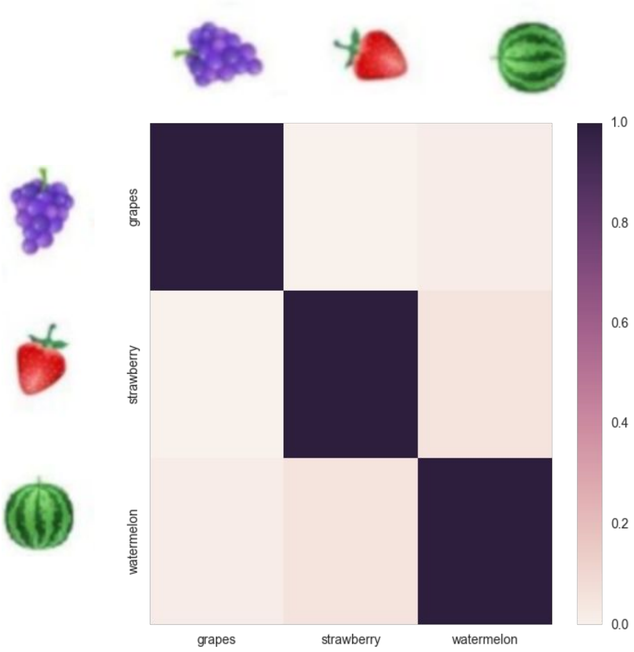
p = 4
N = 3

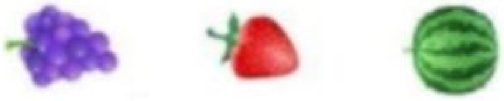
$$x_{ij} = \begin{cases} 1 & \text{if user j purchased item i} \\ 0 & \text{otherwise} \end{cases}$$

Cosine Distance Between **Items**

$$[\mathrm{Corr}]_{ij} = \frac{\mathbf{x_i} \cdot \mathbf{x}_j}{\|\mathbf{x_i}\| \|\mathbf{x_j}\|} = \cos(\delta_{ij})$$

$$X^T X$$



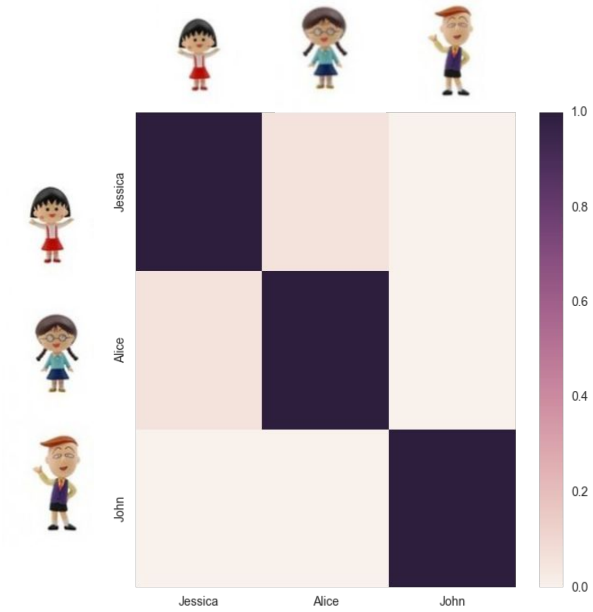


$$\begin{bmatrix} 1.0 & 0.0 & 0.018 \\ 0.0 & 1.0 & 0.05 \\ 0.018 & 0.05 & 1.0 \end{bmatrix}$$

Cosine Distance Between **Users**

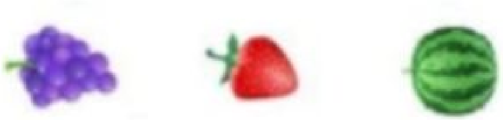
$$[\mathrm{Corr}]_{ij} = \frac{\mathbf{x_i} \cdot \mathbf{x}_j}{\|\mathbf{x_i}\| \|\mathbf{x_j}\|} = \cos(\delta_{ij})$$

$$XX^T$$

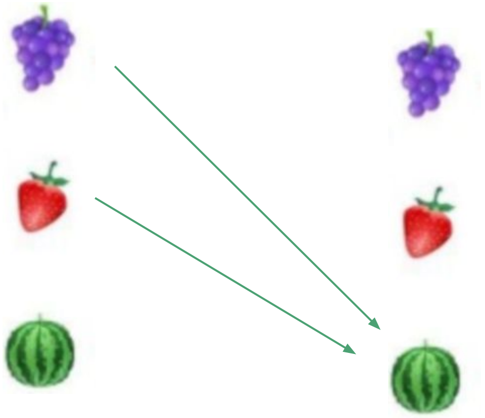


$$\begin{bmatrix} 1.0 & 0.06 & 0.0 \\ 0.06 & 1.0 & 0.0 \\ 0.0 & 0.0 & 1.0 \end{bmatrix}$$

How are recommendations made? **Item/Item**



$$\begin{bmatrix} 1.0 & 0.0 & 0.018 \\ 0.0 & 1.0 & 0.05 \\ 0.018 & 0.05 & 1.0 \end{bmatrix}$$

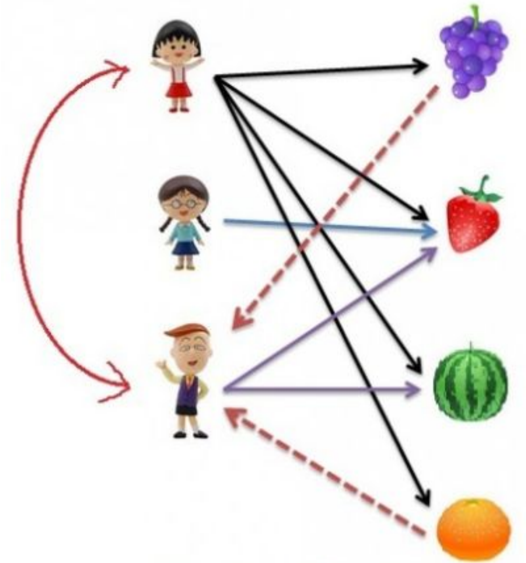


- Sort items from highest to lowest scores.
- For each item the user has liked or purchased, suggest the top K items that the user hasn't already liked/purchased.

How are recommendations made? **Item/Item**

$$\begin{bmatrix} 1.0 & 0.06 & 0.0 \\ 0.06 & 1.0 & 0.0 \\ 0.0 & 0.0 & 1.0 \end{bmatrix}$$

- Find top K most similar users.
- Find items that those users like that the user has not yet.
- Recommend those items.



User-based filtering

Problems with user-user

Earlier collaborative filtering systems based on rating similarity between users (known as user-user collaborative filtering) had several problems:

- Systems performed poorly when they had many items but comparatively few ratings.
- Computing similarities between all pairs of users is expensive (computationally).
- User profiles changed quickly and the entire system model had to be recomputed.

Music Suggestions via Collaborative Filtering

Suggesting bands based on your history

```
In [2]: df=pd.read_csv('http://www.salemmarafi.com/wp-content/uploads/2014/04/lastfm-matrix-germany.csv')

In [4]: X = df.drop(['user'],1)

In [123]: df.head()
```

Out[123]:

	user	a perfect circle	abba	ac/dc	adam green	aerosmith	afi	air	alanis morissette	alexisonfire	alicia keys	all that remains	amon amarth	amy macdonald	amy winehouse	anti-flag	aphex twin	apoca
0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	33	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
2	42	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	51	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	62	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

```
In [14]: X.shape
```

Computing the item/item and user/user scores

```
In [15]: user_user = 1-pairwise_distances(X, metric="cosine")
         item_item = 1-pairwise_distances(X.T, metric="cosine")
```

$$X X^T$$

$$X^T X$$

Compute the **item/item** and **user/user** matrices.

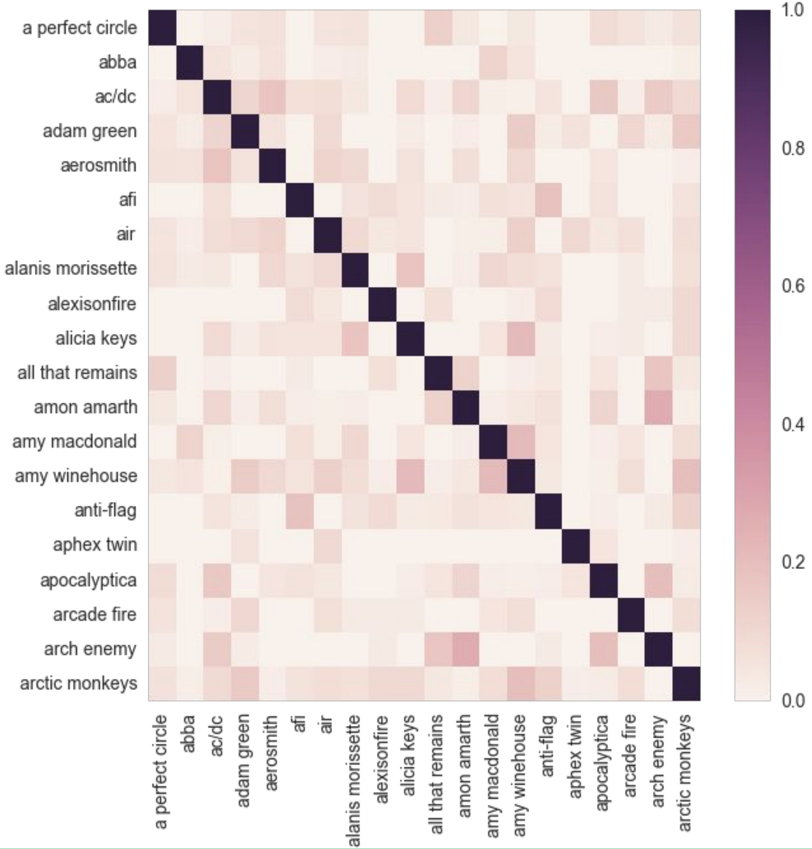
Item/item filtering

In [24]:
```
df_items.head()
```

Out[24]:

	a perfect circle	abba	ac/dc	adam green	aerosmith	afi	air	alanis morissette	alexisonfire	alicia keys	all that remains	am am
a perfect circle	1.000000	0.000000	0.017917	0.051554	0.062776	0.000000	0.051755	0.060718	0	0.000000	0.13012	0.0
abba	0.000000	1.000000	0.052279	0.025071	0.061056	0.000000	0.016779	0.029527	0	0.000000	0.00000	0.0
ac/dc	0.017917	0.052279	1.000000	0.113154	0.177153	0.067894	0.075730	0.038076	0	0.088333	0.02040	0.1
adam green	0.051554	0.025071	0.113154	1.000000	0.056637	0.000000	0.093386	0.000000	0	0.025416	0.00000	0.0
aerosmith	0.062776	0.061056	0.177153	0.056637	1.000000	0.000000	0.113715	0.100056	0	0.061898	0.00000	0.0

Item/Item Matrix Visualized



- AC/DC and Aerosmith have a high correlation.
- Aphex Twin and Air have a high correlation.

Recommending based on items

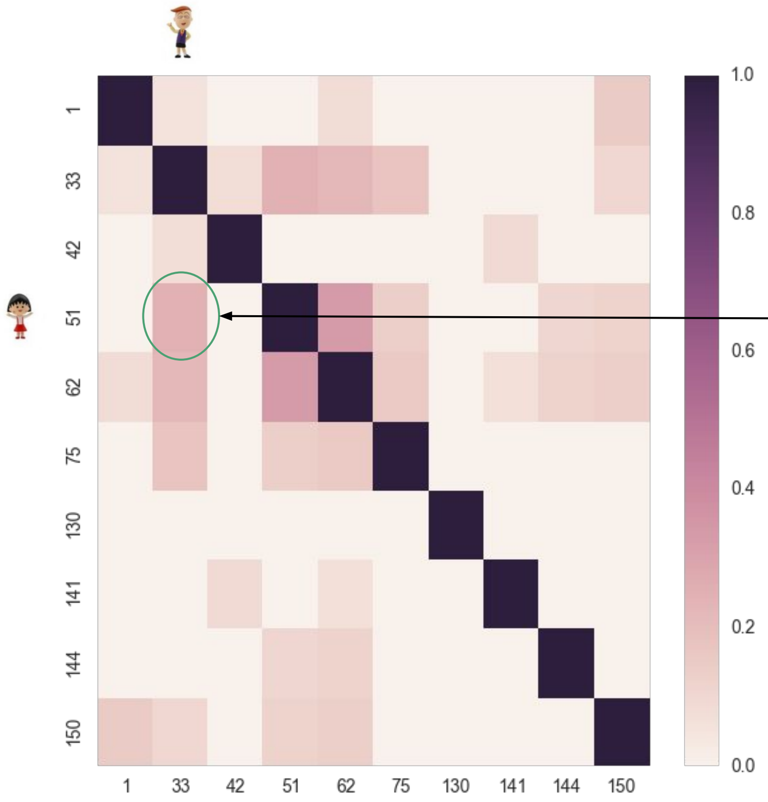
```
In [53]: data_neighbours.head(6).ix[:6,2:4]
```

Out[53]:

	2	3	4
a perfect circle	tool	dredg	deftones
abba	madonna	robbie williams	elvis presley
ac/dc	red hot chili peppers	metallica	iron maiden
adam green	the libertines	the strokes	babyshambles
aerosmith	u2	led zeppelin	metallica
afi	funeral for a friend	rise against	fall out boy

"**These are some suggestions based on your interest in AC/DC:** *Red Hot Chili Peppers, Metallica, Iron Maden*"

User/User based filtering



User ID

Users 51 and 33 are very similar. How do we use this information?

Graph Diffusion and Random Walks

How are recommendations made? **Item/Item**

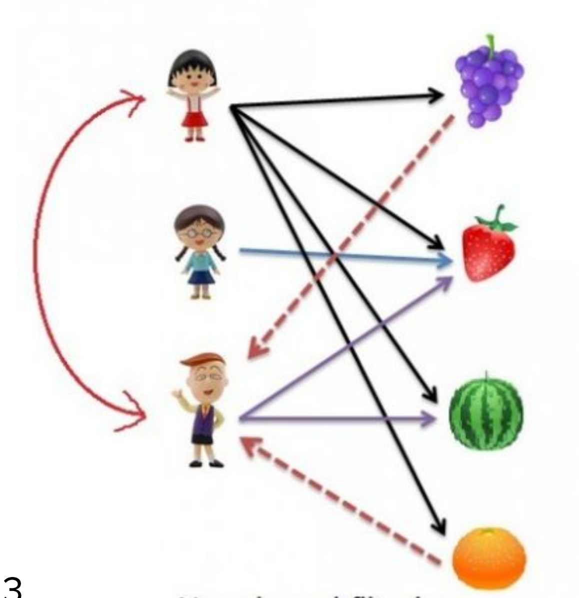
$A = X$ Is the adjacency matrix

$$p(n|j) = \frac{A_{nj}}{\sum_n A_{nj}}$$ User n, item j

$$p(j|n) = \frac{A_{nj}}{\sum_j A_{nj}}$$ Item j, user n

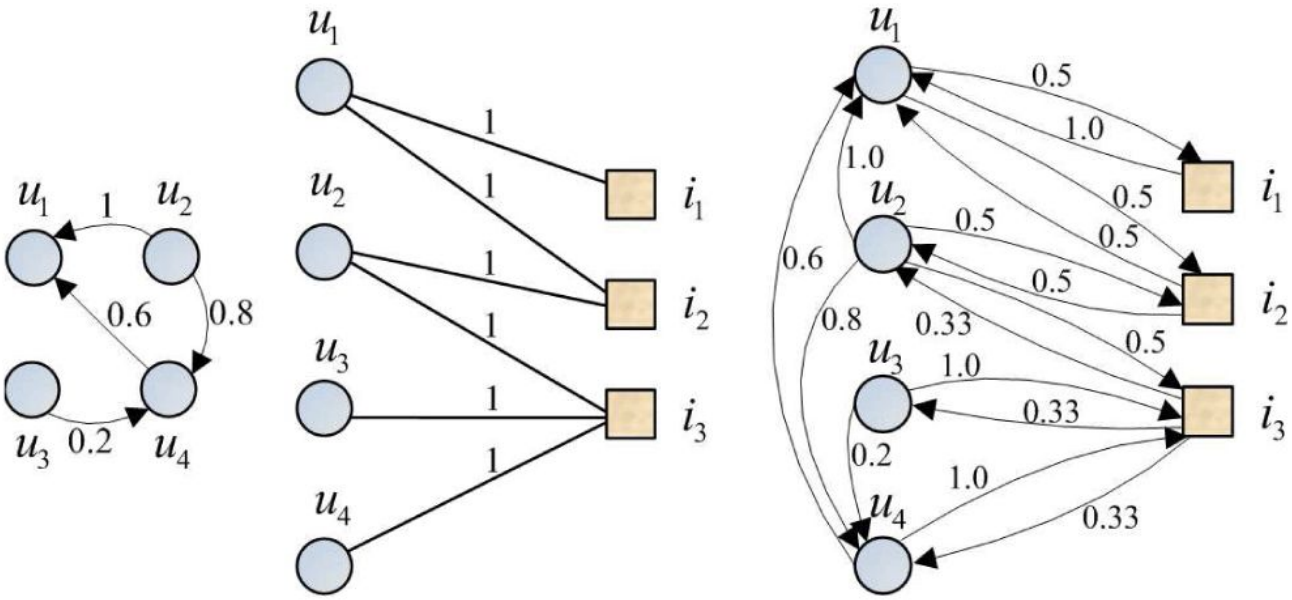
These are **transition probabilities**.

- The probability that item j will be chosen by user n = 1,2,3,..
- The probability that user n will select items j = 1,2,3...
- This is an example of a **discrete random walk on a bipartite graph**. You can jump from **users to items and back** along the graph.



User-based filtering

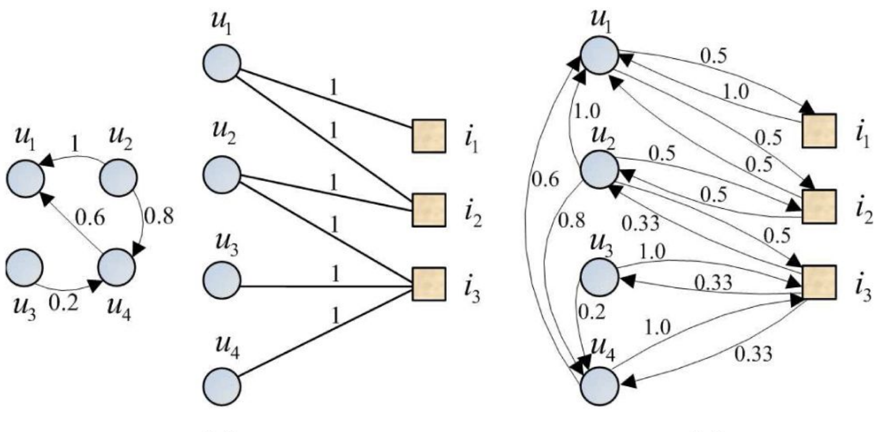
Constructing the graph



(a)

(b)

Constructing the graph



(a) (b)

$$M := p(n|j) = \begin{bmatrix} 1 & 0.5 & 0 \\ 0 & 0.5 & 0.33 \\ 0 & 0 & 0.33 \\ 0 & 0 & 0.33 \end{bmatrix}$$

$$G := p(j|n) = \begin{bmatrix} 0.5 & 0.5 & 0 \\ 0 & 0.5 & 0.5 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{bmatrix}$$

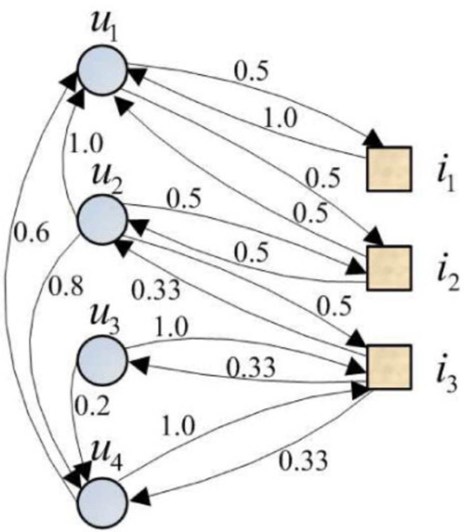
$$A = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{bmatrix}$$

Think of these as probabilities of jumping along edges of the graph drawn above.

Propensity for user n item j

$$M := p(n|j) = \begin{bmatrix} 1 & 0.5 & 0 \\ 0 & 0.5 & 0.33 \\ 0 & 0 & 0.33 \\ 0 & 0 & 0.33 \end{bmatrix}$$

$$G := p(j|n) = \begin{bmatrix} 0.5 & 0.5 & 0 \\ 0 & 0.5 & 0.5 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{bmatrix}$$



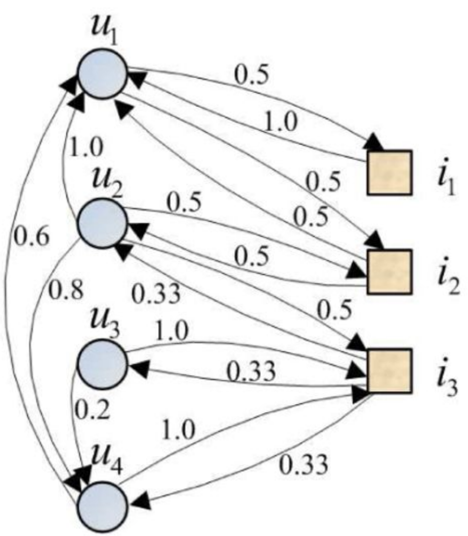
$$\pi(n, j) = \sum_{j', n'} p(j|n')p(n'|j')q^0_{n,j'} \quad q_{n=1} = [1, 1, 0]^T$$

$$A = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{bmatrix}$$

Propensity for user n item j

$$M := p(n|j) = \begin{bmatrix} 1 & 0.5 & 0 \\ 0 & 0.5 & 0.33 \\ 0 & 0 & 0.33 \\ 0 & 0 & 0.33 \end{bmatrix}$$

$$G := p(j|n) = \begin{bmatrix} 0.5 & 0.5 & 0 \\ 0 & 0.5 & 0.5 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{bmatrix}$$



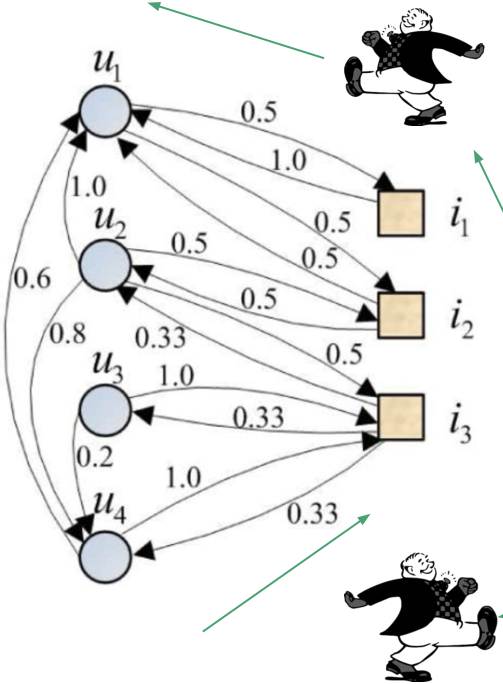
$$\boxed{\pi_n = G^T M q_n}$$

$G^T M$ Diffusion Operator

$$\pi(n, j) = \sum_{j', n'} p(j|n')p(n'|j')q^0_{n,j'} \quad q_{n=1} = [1, 1, 0]^T$$

$$A = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{bmatrix}$$

How the random walk works



$$q_{n=1} = [1, 1, 0]^T$$

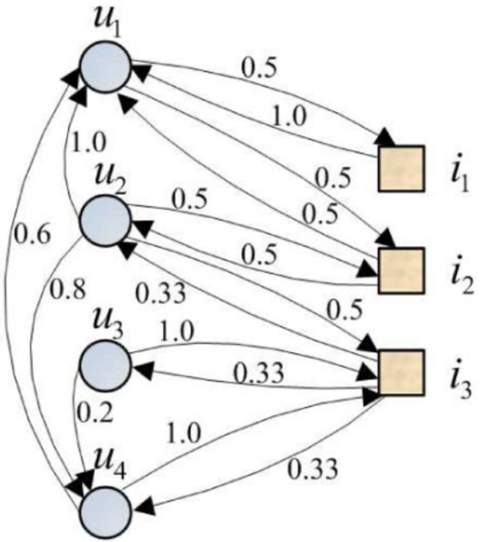
- Start off with a user vector which lists the items they like.
- Take random steps with various probabilities to go from **items -> users.**
- Take another random step back to items **users -> items.**

Regularization for Diffusion?

$$\pi_n = G^T M q_n$$

$$\pi_n^{m,\alpha} = (1-\alpha)(G^T M)^m q_n + \alpha q_{\text{pop}}$$

- The m exponent is a **regularization term**.
- The operator $G^T M$ is also a **transition matrix** from **items to items.**
- The process above for m >= 1 is known as a **Markov Chain**.

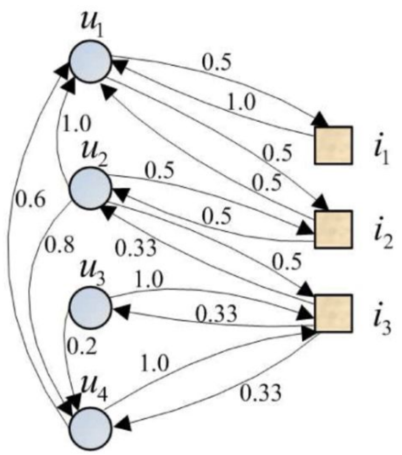


Markov Chains

$$P(X_{n+1} = x_{n+1} | X_n = x_n, \cdots X_1 = x_1) = P(X_{n+1} = x_{n+1} | X_n = x_n)$$

- Markov chains are defined as ‘memoryless’ processes - ie. the probability of the (n+1) st state depends only on the state before it (n).

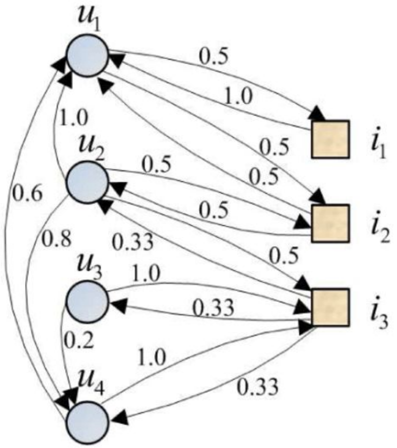
- The operator we defined above satisfies this property. It expresses the probability of going from item i to item j.



Limiting Distribution

$$G^T M y \rightarrow y_\infty \text{ as } m \rightarrow +\infty$$

What is y_∞ ?

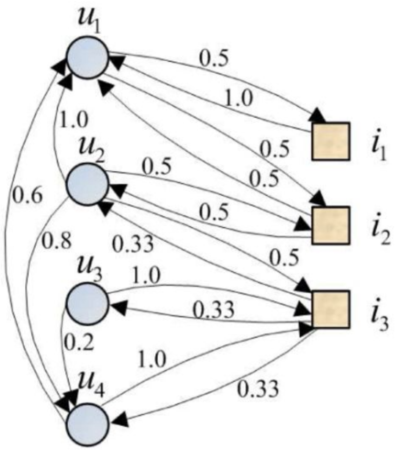


Limiting Distribution

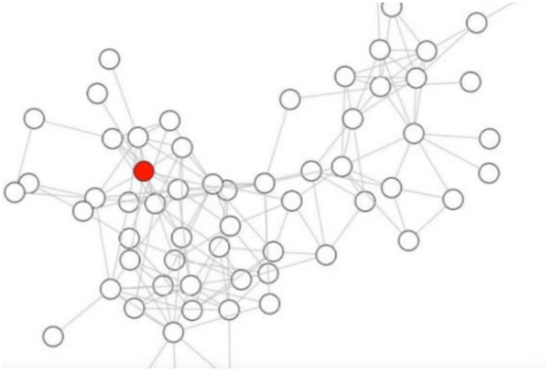
$$G^T My \to y_\infty \text{ as } m \to +\infty$$

Answer: $Ay_\infty = y_\infty$

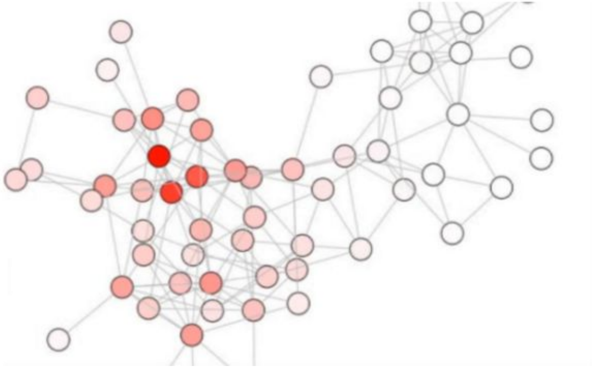
y_∞ Is the <u>unique eigenvector with eigenvalue 1.</u>
It's know as the <u>uniform distribution.</u>



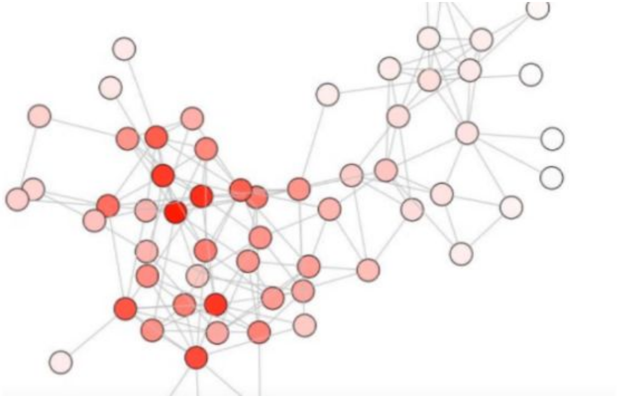
Visualizing Graph Diffusion



N = 1

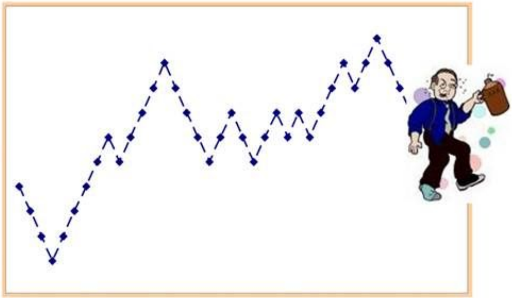


N = 2



N = 3

$$G^T M y \to y_\infty \text{ as } m \to +\infty$$



What is the uniform distribution?

$$y_\infty = [1, 1, 1, \cdots, 1]^T$$

$$A = \begin{bmatrix} p_{11} & p_{12} & 1 - p_{11} - p_{12} \\ p_{21} & p_{22} & 1 - p_{22} - p_{22} \\ p_{31} & p_{32} & 1 - p_{31} - p_{32} \end{bmatrix}$$

$$Ay_\infty = y_\infty$$

- Now all of the edges in the graphs have equal weights! This is the uniform distribution.
- Therefore m is a <u>form of regularization.</u> It helps <u>prevent over fitting.</u>

A simple example

$$G^T M = \begin{bmatrix} 0.9 & 0.5 \\ 0.1 & 0.5 \end{bmatrix}$$

$$\pi(n, j) = \sum_{j', n'} p(j|n')p(n'|j')q^0_{n,j'}$$

$$\mathbf{q}_0 = [1 \;\; 0]^T$$

Let's imagine we have 2 items now, apples and grapes. The matrix to the left is the transition matrix as we constructed it previously.

Summary

- This is a probabilistic generalization of the collaborative filtering algorithms used above.
- Rather than just ranking by scores, we compute transition probabilities by composing two random walks.
- This operator is known as a diffusion operator (related to the heat equation).

Model Evaluation

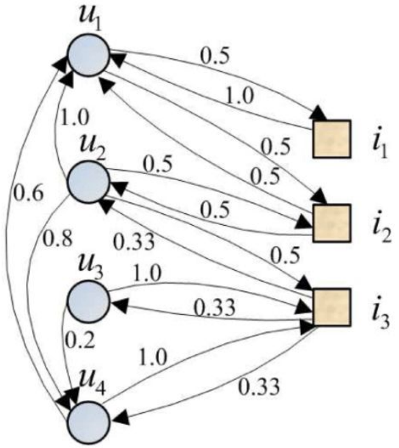
How do you rate a recommendation engine?

- Note that now we have a list of predictions, ranked in order of decreasing probability.
- How many should we choose to ‘guess’? (**Answer:** Depends!)
- In general you are trying to predict based on many possible suggestions.
- As before, we need to break the problem into **training/testing sets**. This is a bit trickier than before though!
- The best way to evaluate a recommendation engine is through A/B tests.

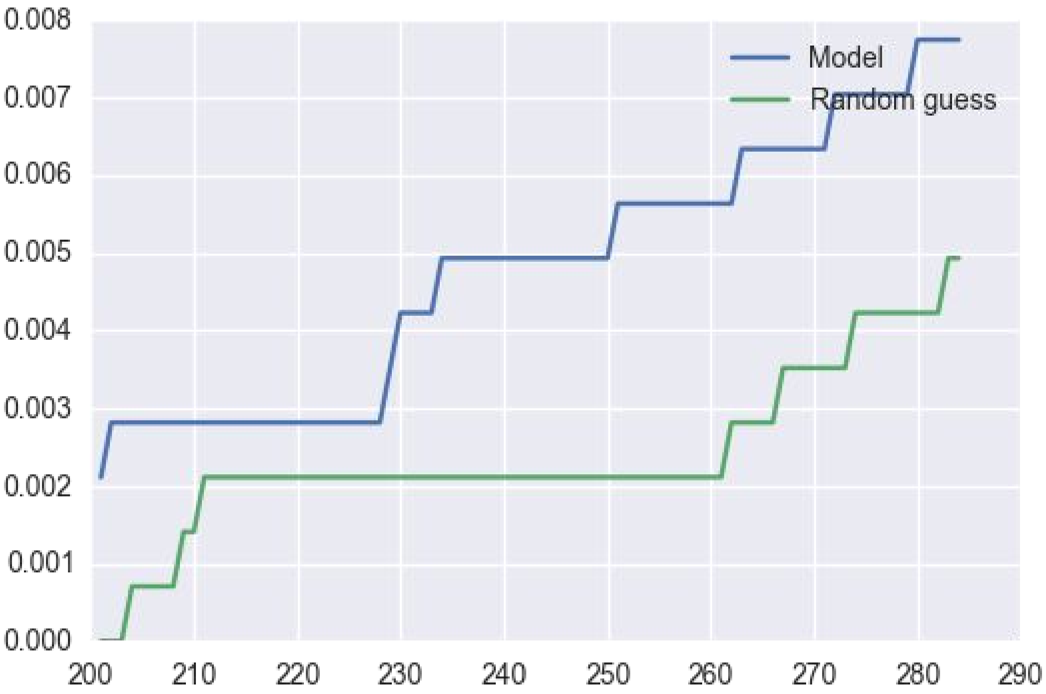
Leaving out validation data

- Let’s build a graph of items to items using either graph diffusion or the item/item similarity matrix, **based on 80% of the users.**

- For a given vector q, **take 80% of the vector and see if you can predict the next 20%.**
- Note that we have two splits of training/testing now.
    - Only use **80% of the users to build the graph.**
    - Only use **80% of the user vector in the validation set** to feed into the graph, and see if you can **predict the remaining 20% of their vector.**



Measuring Performance - Cumulative Recall

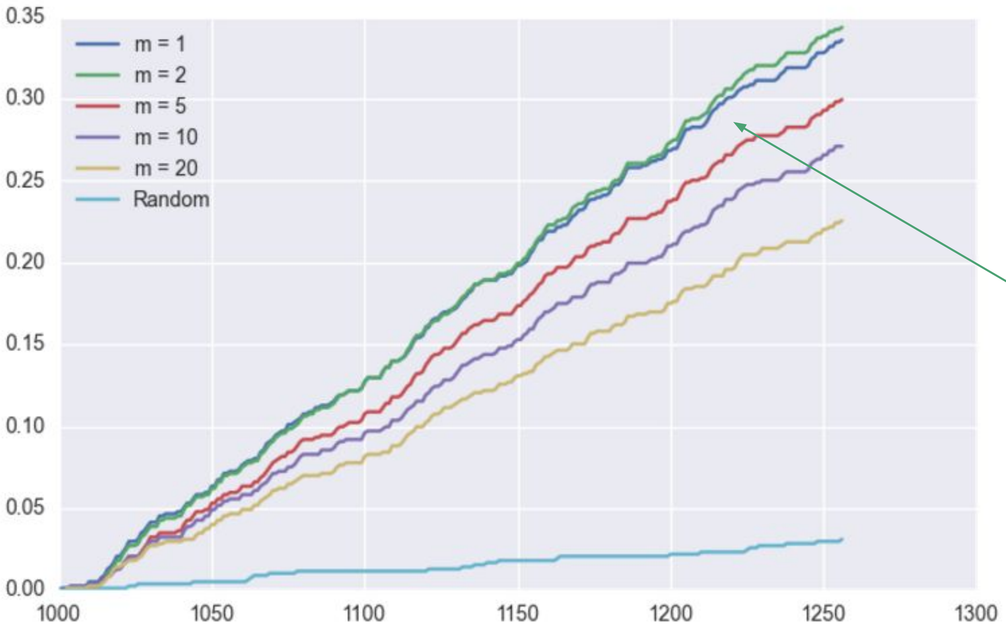


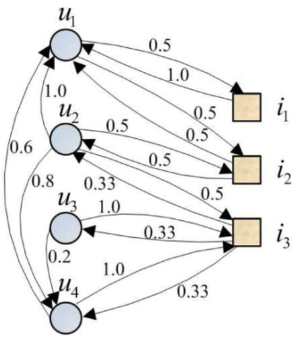
- Here we plot the performance of our model by comparing it to a random guess (much like ROC).
- We've made '5 predictions' and compared this to 5 random guesses.
- This is known as a gains chart

- *Item/Item collaborative model*

Graph Diffusion and Regularization

$$\pi_n^{m,\alpha} = (1-\alpha)(G^T M)^m q_n + \alpha q_{\text{pop}}$$





The choice of m =2 maximizes the total recall on testing data for our dataset. Thus we choose this exponent.