Lecture 7 - Introduction to Recommendation Engines

Lecture 8

Outline

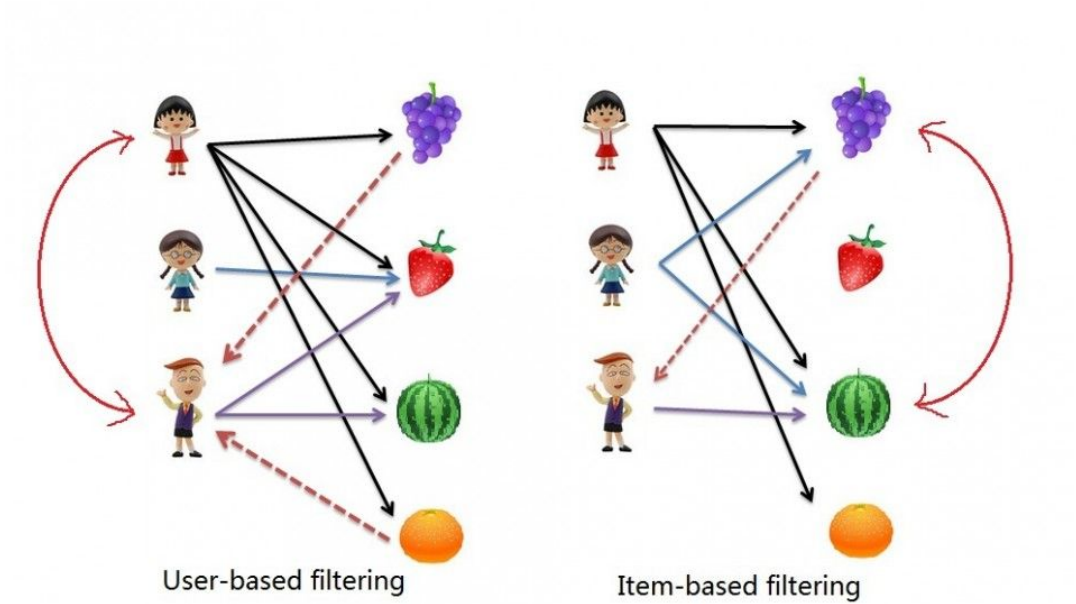
- Motivation
- Setup of the problem
- User-User vs Item-Item Recommendations - Fruit Example
- User-User vs Item-Item for Music Recommendations
- Diffusion on Bipartite Graphs
- Model Evaluation and Regularization

Examples are everywhere

The New York Times

Almost all companies use some kind of recommendation algorithm like the one we will present in class today!

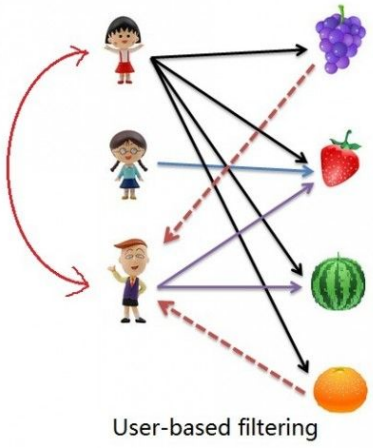
User vs Item Based Filtering

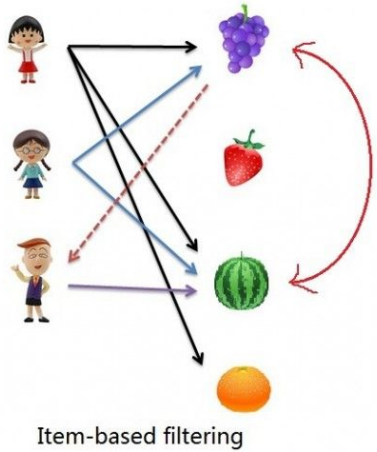


- The first step is to make pairings between users and items, ie. list all users as the rows in a matrix with columns the items.
- User-User based filtering finds similar users based on interest/purchases.
- Item-Item based filtering pairs similar items based on interest/purchases.

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

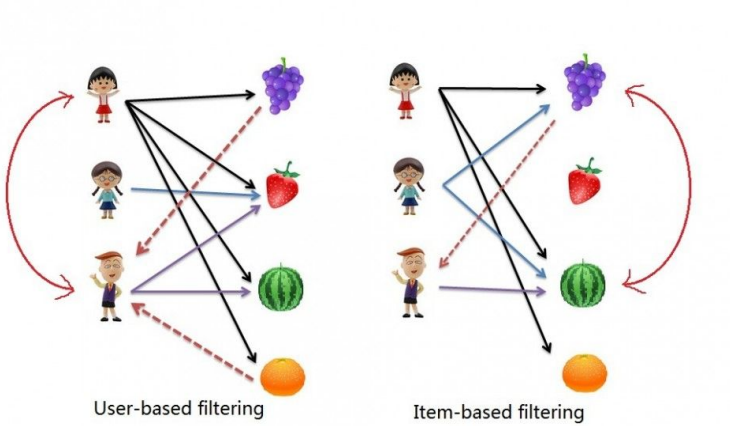

Item-based filtering

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**

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



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

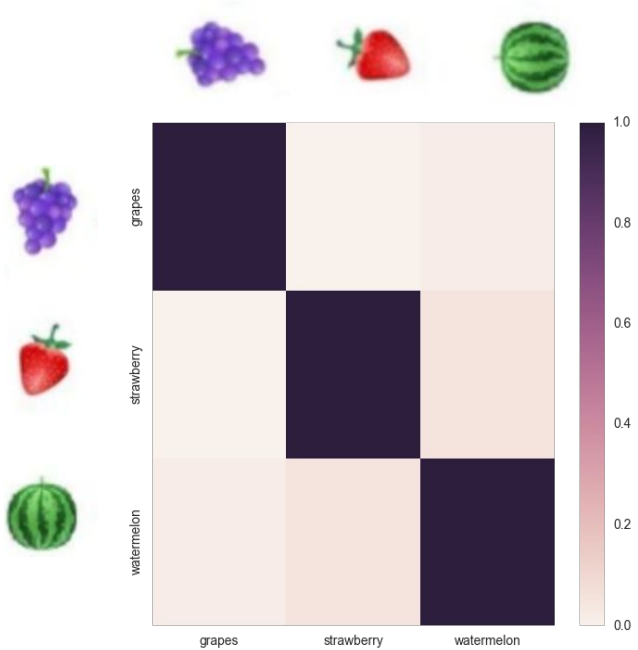
p = 4
N = 3

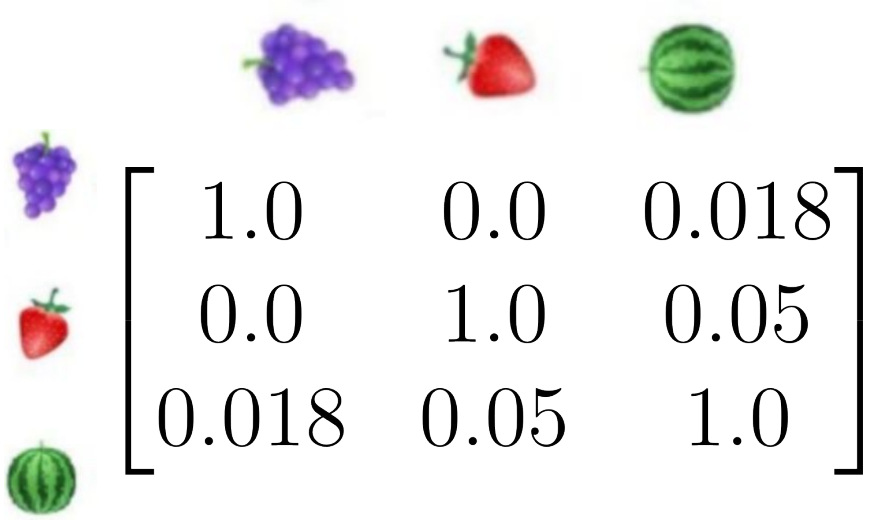
$$x_{ij} = \begin{cases} 1 & \text{if user j purchased item i} \\ 0 & \text{otherwise} \end{cases}$$

Cosine Distance Between **Items**

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

$$X^T X$$



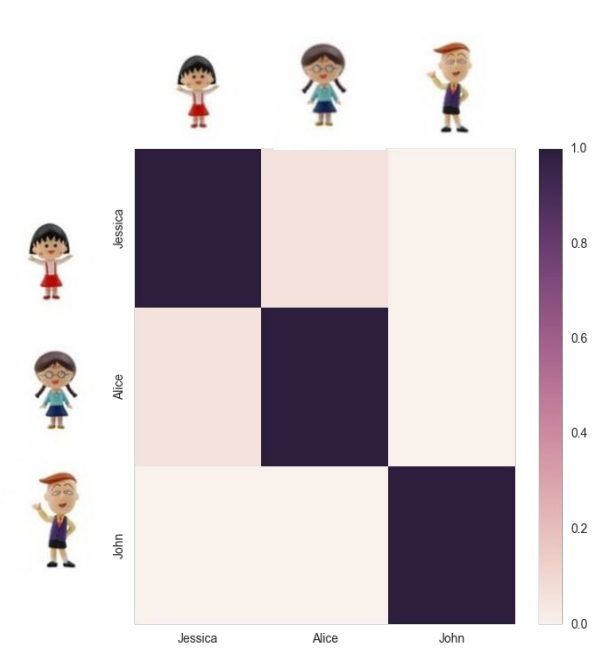


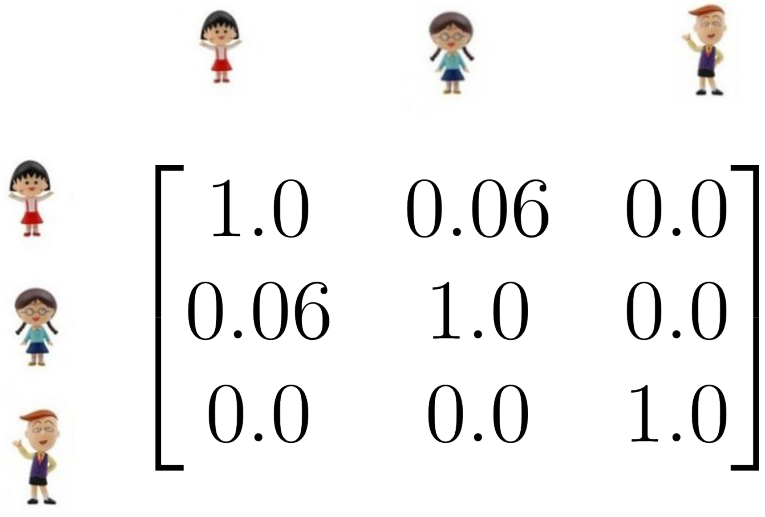
$$\begin{bmatrix} 1.0 & 0.0 & 0.018 \\ 0.0 & 1.0 & 0.05 \\ 0.018 & 0.05 & 1.0 \end{bmatrix}$$

Cosine Distance Between **Users**

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

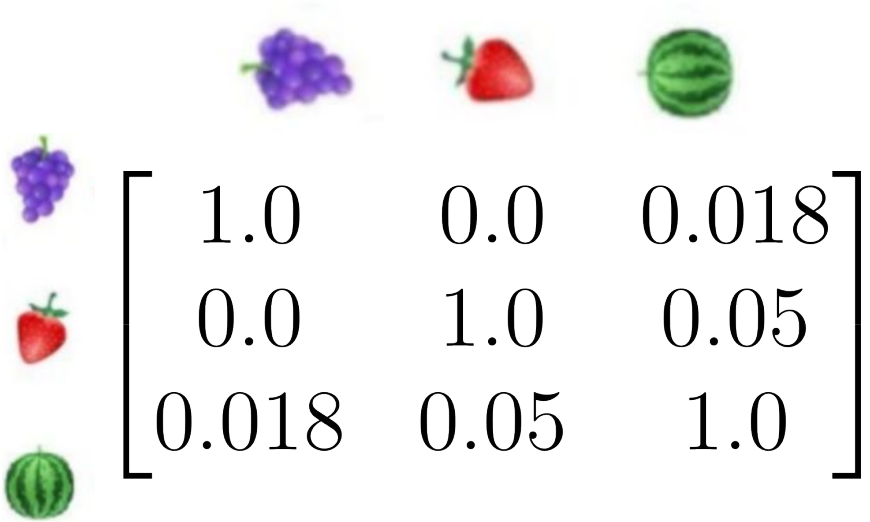
$$XX^T$$



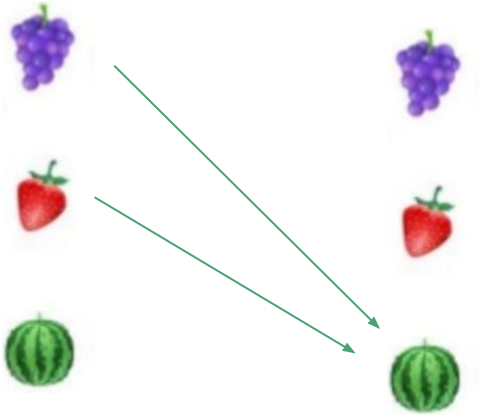


$$\begin{bmatrix} 1.0 & 0.06 & 0.0 \\ 0.06 & 1.0 & 0.0 \\ 0.0 & 0.0 & 1.0 \end{bmatrix}$$

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

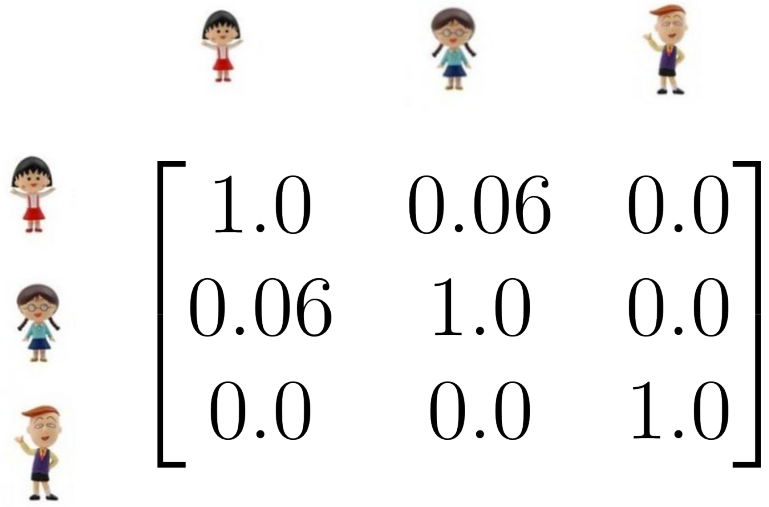


$$\begin{bmatrix} 1.0 & 0.0 & 0.018 \\ 0.0 & 1.0 & 0.05 \\ 0.018 & 0.05 & 1.0 \end{bmatrix}$$

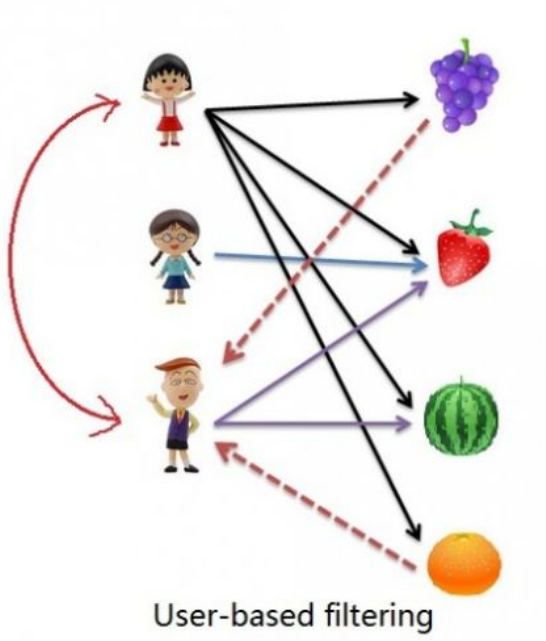


- Sort items from highest to lowest scores.
- For each item the user has liked or purchased, suggest the top K items that the user hasn't already liked/purchased.

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.

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")
```

XX^T

X^TX

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

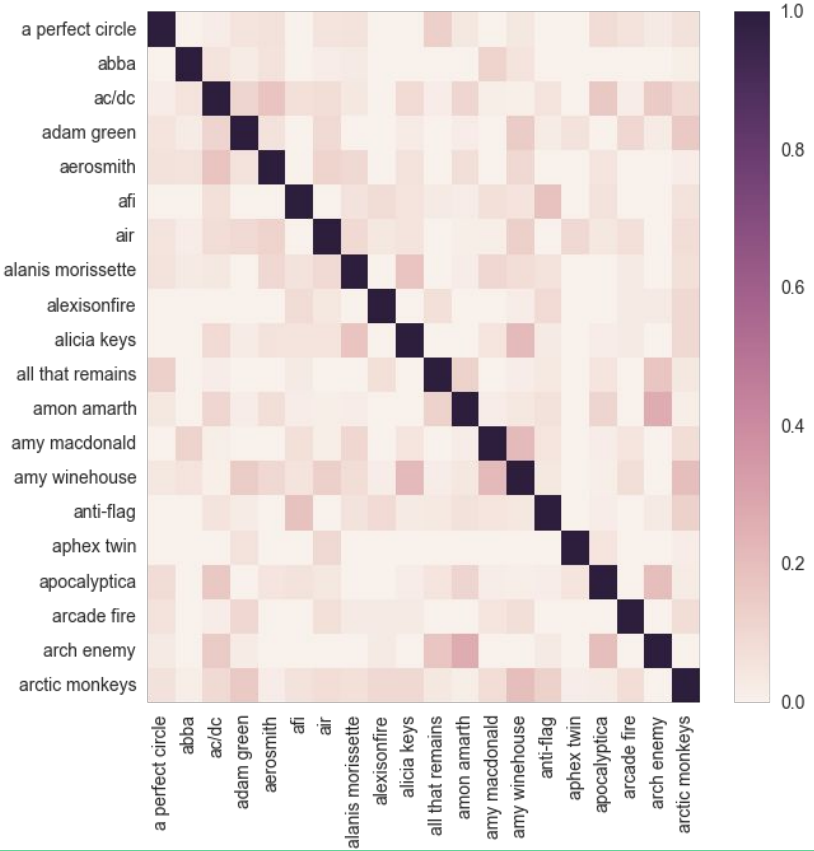
Item/item filtering

```
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
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

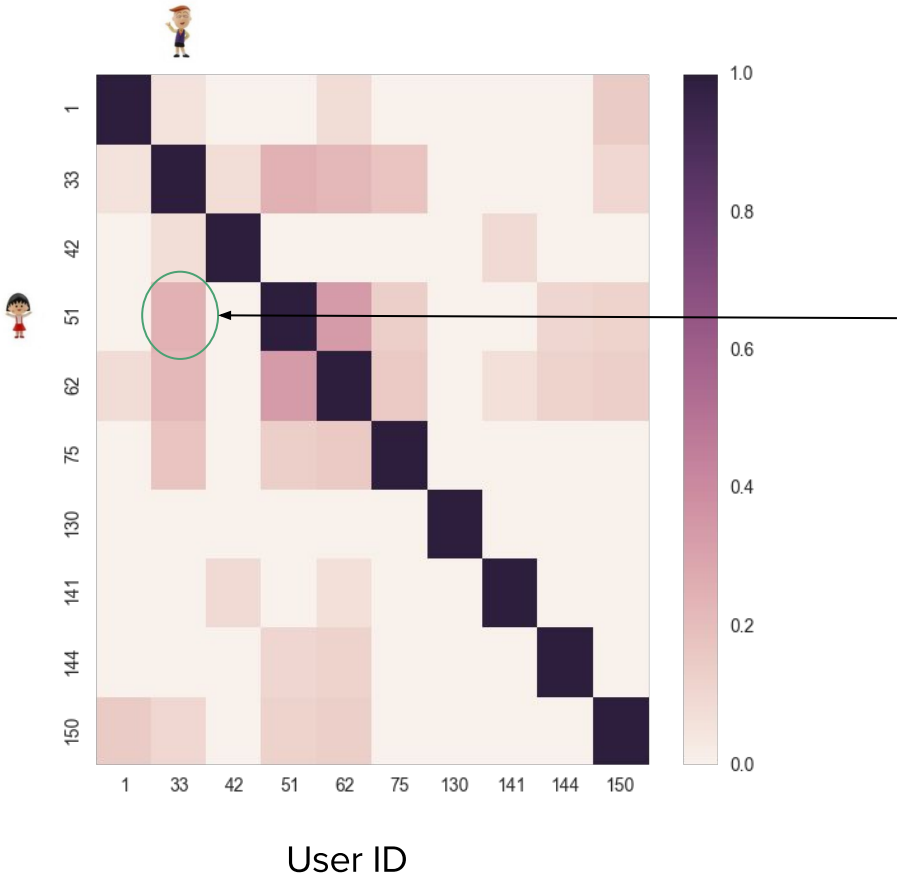
```
In [53]: data_neighbours.head(6).ix[:6,2:4]
```

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

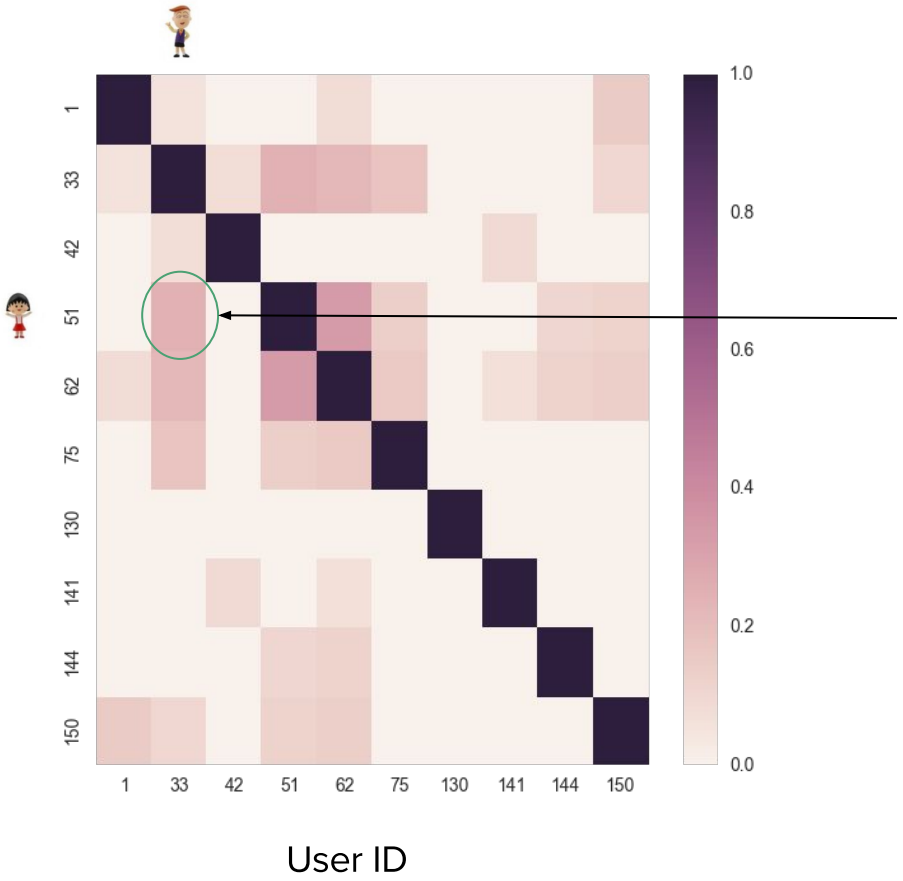
"**These are some suggestions based on your interest in AC/DC:** *Red Hot Chili Peppers, Metallica, Iron Maden*"

User/User based filtering



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

User/User based filtering



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

Answer: We find the items that user 51 has that user 33 doesn't, and vice versa. Imagine them sharing suggestions. This is also known as nearest neighbors.

Graph Diffusion and Random Walks

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

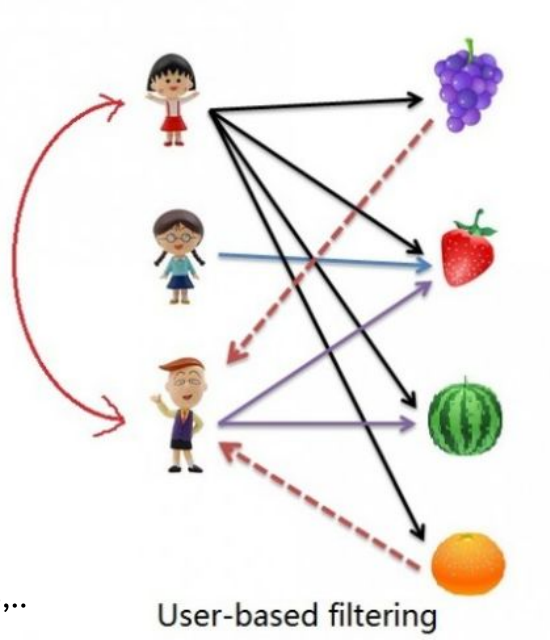
$A = X$ Is the adjacency matrix

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

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

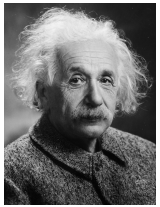
These are **transition probabilities**.

- 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.



Some history of Random Walks and Graph Diffusion

- Original idea by Karl Pearson public in *Nature* in 1905.



- Albert Einstein also published his seminal paper on Brownian motion – the complicated path of a large dust particle in air – which he modeled as a random walk, driven by collisions with gas molecules.
- In the limit of small distances, Random Walks converge to the laplace operator (diffusion of gases)!

Random Walks and Diffusion

$$X_n = \frac{1}{\sqrt{n}} \sum_k Z_k$$

$$Z_k = \begin{cases} 1 & \text{with probability } 0.5 \\ -1 & \text{with probability } 0.5 \end{cases}$$

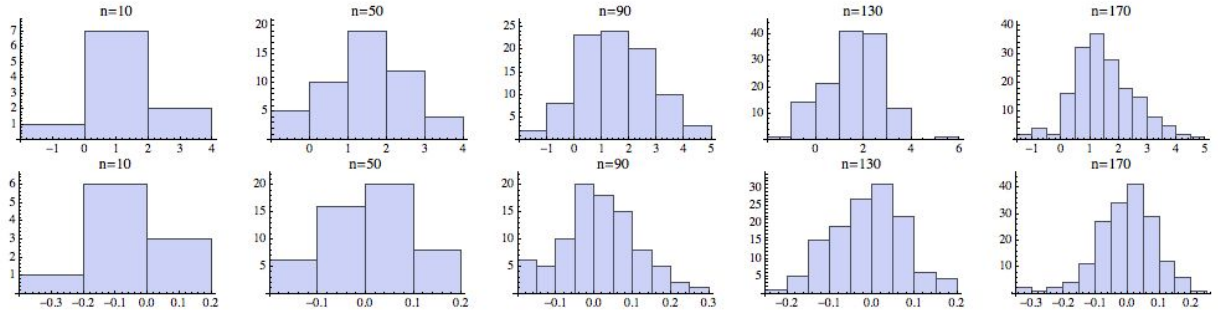
$$\mathbb{E}(Z_k) = 0$$

$$\mathrm{Var}(Z_k) = 1$$

Central Limit Theorem: If the Z_k are all independent, then X_n converges to a normal distribution with mean zero and unit variance.

Random Walks and Diffusion

$$X_n = \frac{1}{\sqrt{n}} \sum_k Z_k \qquad Z_k = \begin{cases} 1 & \text{with probability } 0.5 \\ -1 & \text{with probability } 0.5 \end{cases}$$



Central Limit Theorem: If the Z_k are all independent, then X_n converges to a normal distribution with mean zero and unit variance.

Random Walks and Diffusion

$$X_n = \frac{1}{\sqrt{n}} \sum_k Z_k$$

$$Z_k = \begin{cases} 1 & \text{with probability } 0.5 \\ -1 & \text{with probability } 0.5 \end{cases}$$

$$\delta t = \frac{T}{n}$$

$$W_k = \sum_{j \leq k} \sqrt{\delta t} Z_j$$

$$t_k = k\delta t$$

$$\mathrm{Var}(W_k) = k\mathrm{Var}(\sqrt{\delta t} Z) = k\delta t = t_k$$

This is why we chose the scaling the way we did!

Random Walks and Diffusion

$$W_k = \sum_{j \leq k} \sqrt{\delta t} Z_j \qquad \mathrm{Var}(W_k) = k\mathrm{Var}(\sqrt{\delta t} Z) = k\delta t = t_k$$

$$W_k \to \mathcal{N}(0, t) \text{ as } k \to +\infty \qquad p(x, t) = \frac{1}{\sqrt{2\pi t}} e^{-x^2/2t}$$

Random Walks and Diffusion

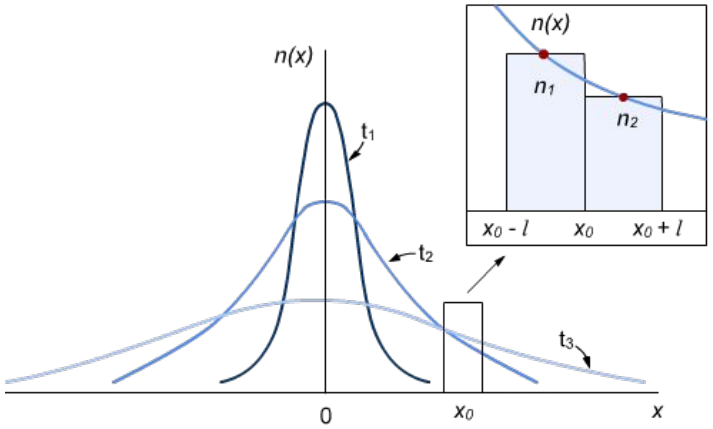
$$W_k = \sum_{j \leq k} \sqrt{\delta t} Z_j \qquad \mathrm{Var}(W_k) = k\mathrm{Var}(\sqrt{\delta t} Z) = k\delta t = t_k$$

$$W_k \to \mathcal{N}(0, t) \text{ as } k \to +\infty \qquad p(x,t) = \frac{1}{\sqrt{2\pi t}} e^{-x^2/2t}$$

Diffusion Equation!

$$\frac{\partial p(x,t)}{\partial t} = \frac{1}{2} \frac{\partial^2 p(x,t)}{\partial^2 x}$$

Diffusion Equation



$$\frac{\partial p(x,t)}{\partial t} = \frac{1}{2}\frac{\partial^2 p(x,t)}{\partial^2 x}$$

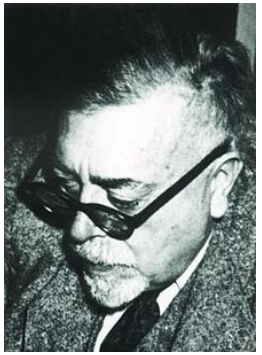
An initially concentrated gas will diffuse and spread out quickly (recall the variance grows linear in time).

$$p(x,t) = \frac{1}{\sqrt{2\pi t}} e^{-x^2/2t}$$

Famous Weiners

Norbert Wiener

- Invented the notion of '*Wiener process'*, also known as *Brownian motion,* from the physical observations of Robert Brown.
- Professor at MIT and a child prodigy, obtaining his Bachelor's degree at 14 years old.
- Invented the field of *cybernetics*.

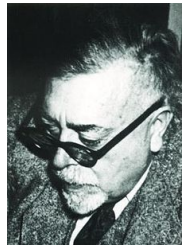


Anthony Weiner

- Enjoys showing his *'Weiner'*.
- Was a public high school math teacher.
- Seems to enjoy the company of 14 year olds.
- Pioneer in the field of *'sexting'*.

Wiener Process

1. $W_0 = 0$ Initial condition is 0
2. $W_{s+t} - W_s \sim \mathcal{N}(0, t)$ Normally distributed and independent in time.
3. $\lim_{s \to t} \mathbb{P}(|W_t - W_s| \geq \epsilon) \to 0$ Continuous in time (the left is convergence in measure).

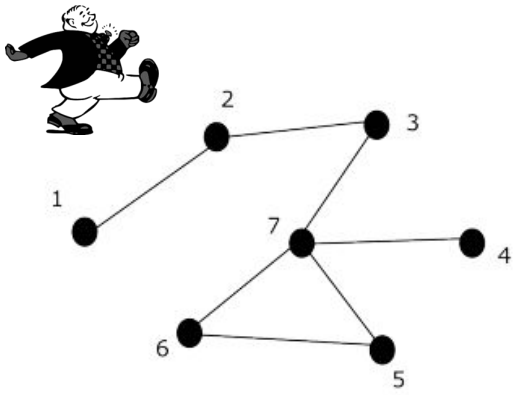


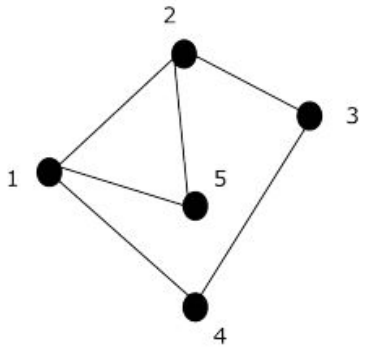
Can be constructed as the scaling limit of a discrete random walk (what we just saw).

Our first example of a *Stochastic Process* - a random variable which is a function of time. This is how time series data is modeled (more on this later).

Random Walks on Graphs

Imagine that the paths are restricted however, and we have various ‘weights’ associated with the likelihood of going in that direction.





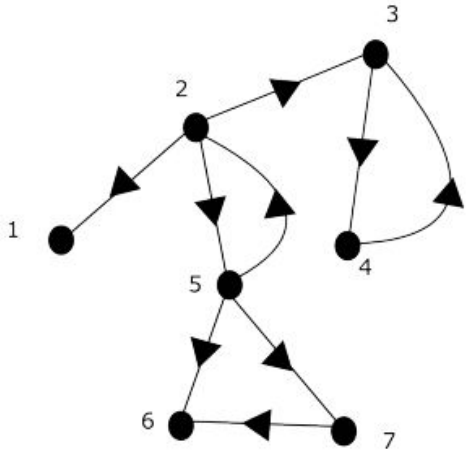
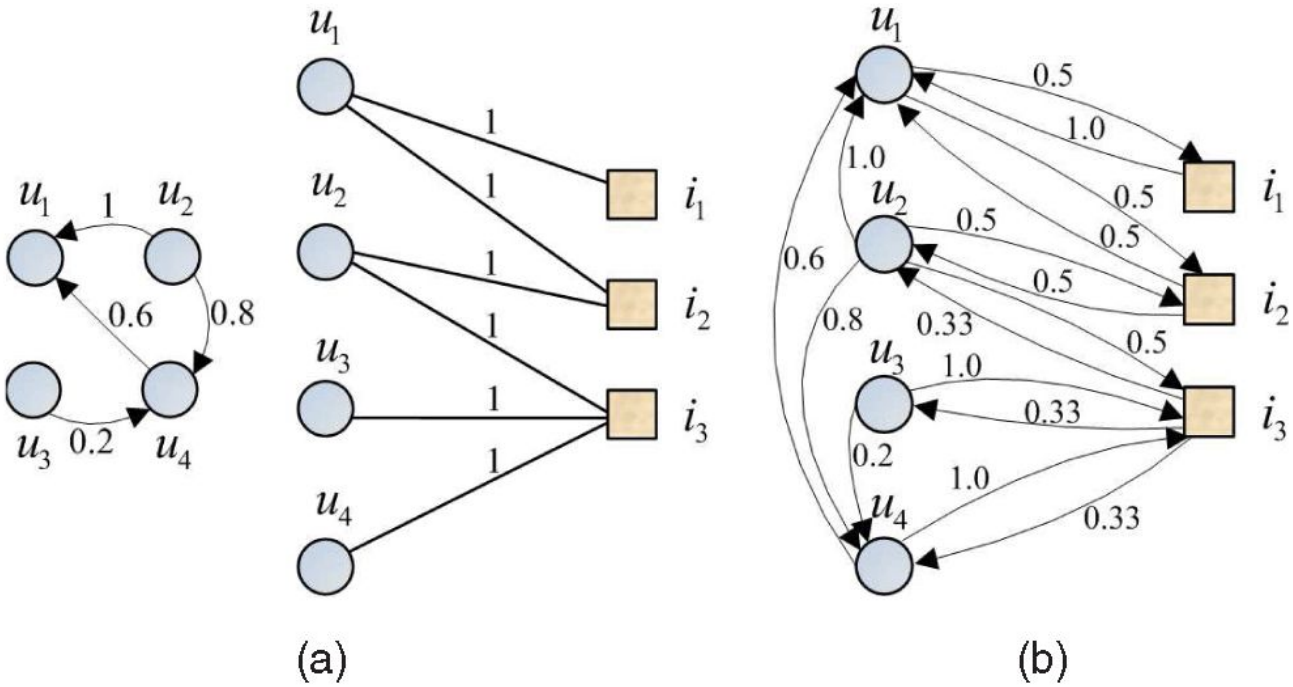
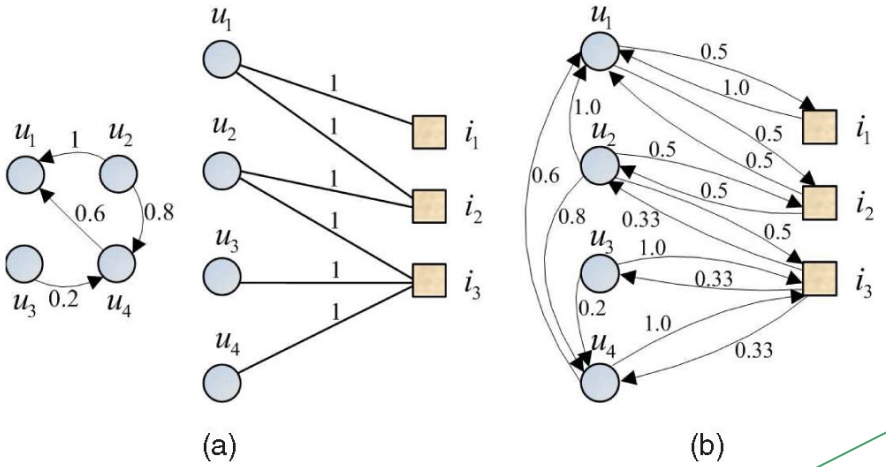


FIGURE 1

Constructing the graph



Constructing the graph



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

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

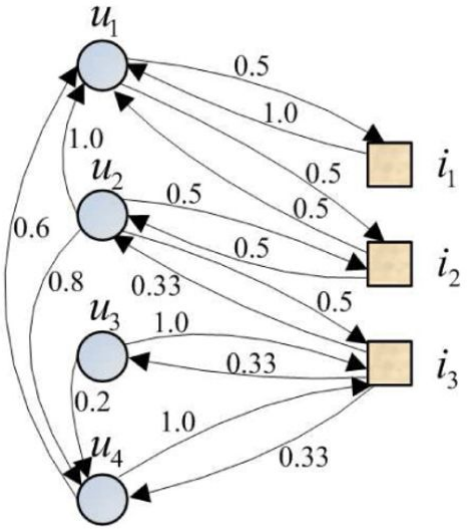
$$G := p(j|n) = \begin{bmatrix} 0.5 & 0.5 & 0 \\ 0 & 0.5 & 0.5 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{bmatrix}$$

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

Propensity for user n item j

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

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



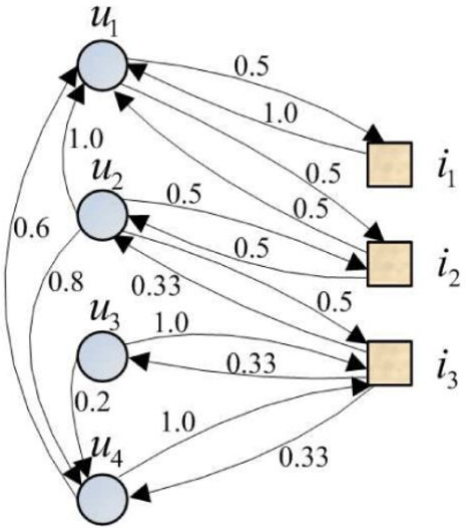
$$\pi(n,j) = \sum_{j',n'} p(j|n')p(n'|j')q^0_{n,j'} \quad q_{n=1} = [1,1,0]^T \rightarrow A = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{bmatrix}$$

Well defined probability by the law of total probability (read about this)

Propensity for user n item j

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

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



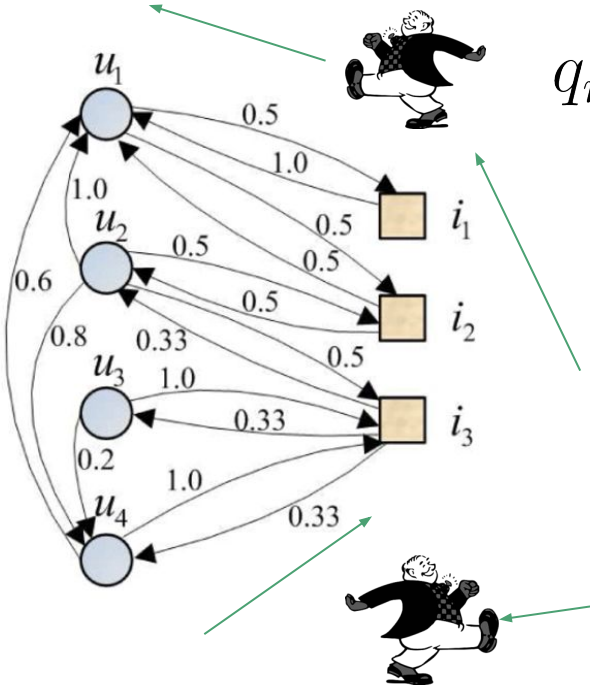
$$\pi_n = G^T M q_n$$

$G^T M$ 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



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

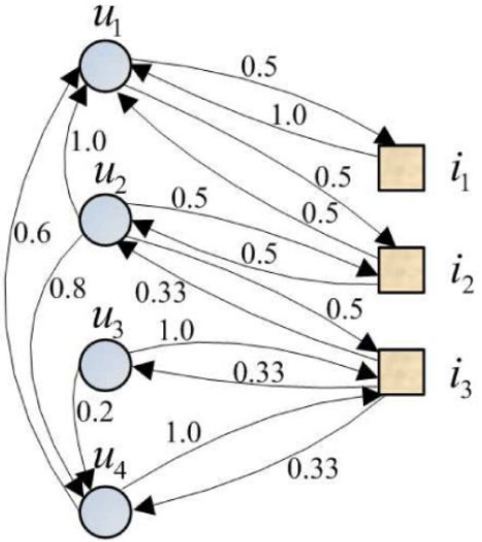
- Start off with a user vector which lists the items they like.
- Take random steps with various probabilities to go from **items -> users.**
- Take another random step back to items **users -> items.**

Regularization for Diffusion?

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

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

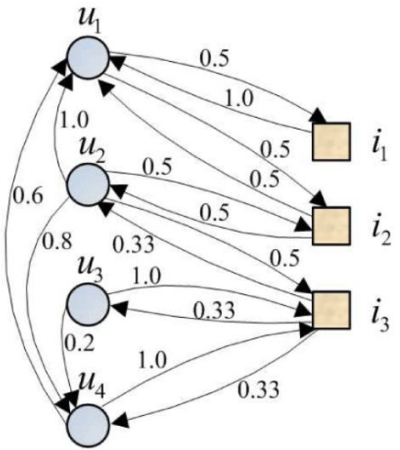
- The m exponent is a **regularization term**.
- The operator $G^T M$ is also a **transition matrix** from **items to items.**
- The process above for m >= 1 is known as a **Markov Chain**. The exponent is like taking time steps in the diffusion process.



Markov Chains

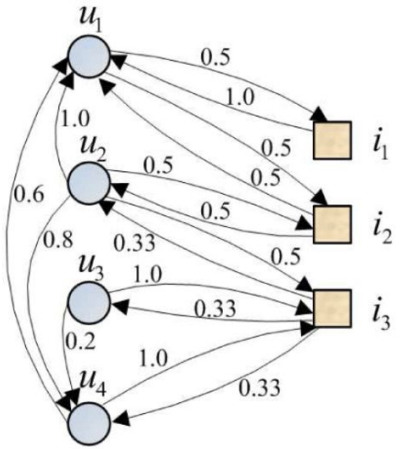
$$P(X_{n+1} = x_{n+1} | X_n = x_n, \cdots X_1 = x_1) = P(X_{n+1} = x_{n+1} | X_n = x_n)$$

- 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

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



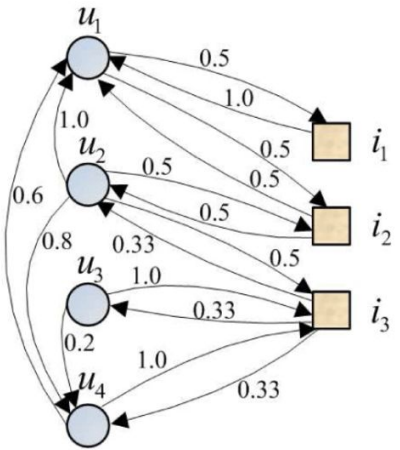
What is y_∞ ?

Limiting Distribution

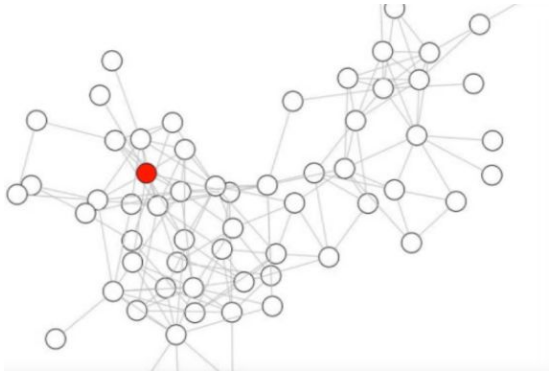
$$G^T M y \to y_\infty \text{ as } m \to +\infty$$

Answer: $Ay_\infty = y_\infty$

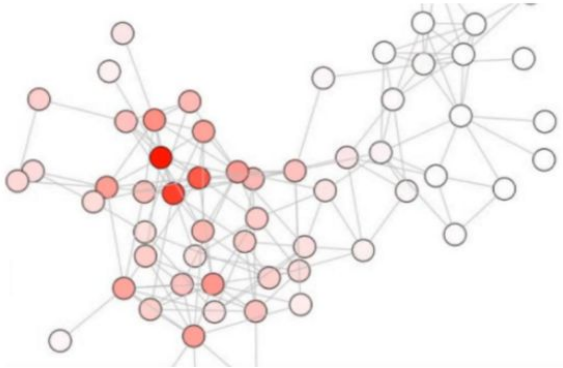
y_∞ Is the unique eigenvector with eigenvalue 1.
It's know as the uniform distribution.



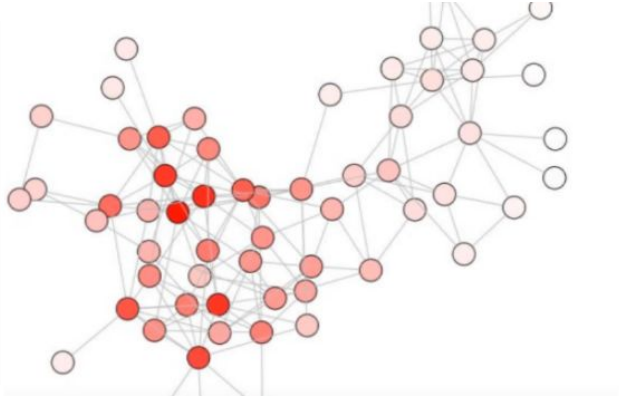
Visualizing Graph Diffusion



N = 1



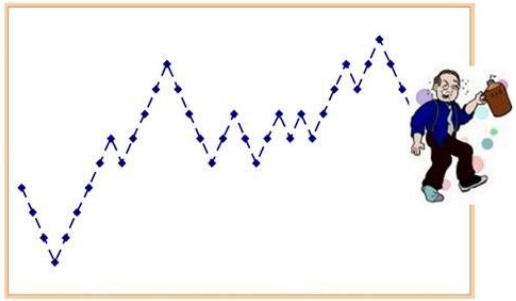
N = 2



N = 3

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

Think of a particle concentrated on a single node, and the diffusion operator letting it spread across the network and find new recommendations on the allowed paths.



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 form of regularization. It helps prevent over fitting.

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 and Regularization

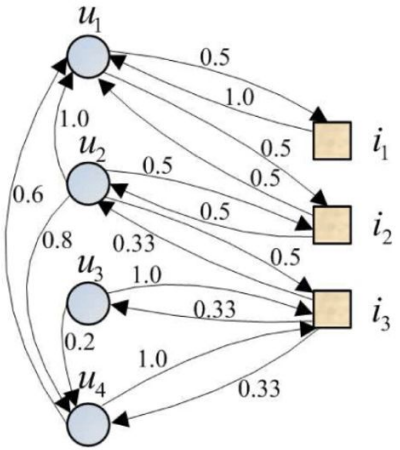
How do you rate a recommendation engine?

- 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.

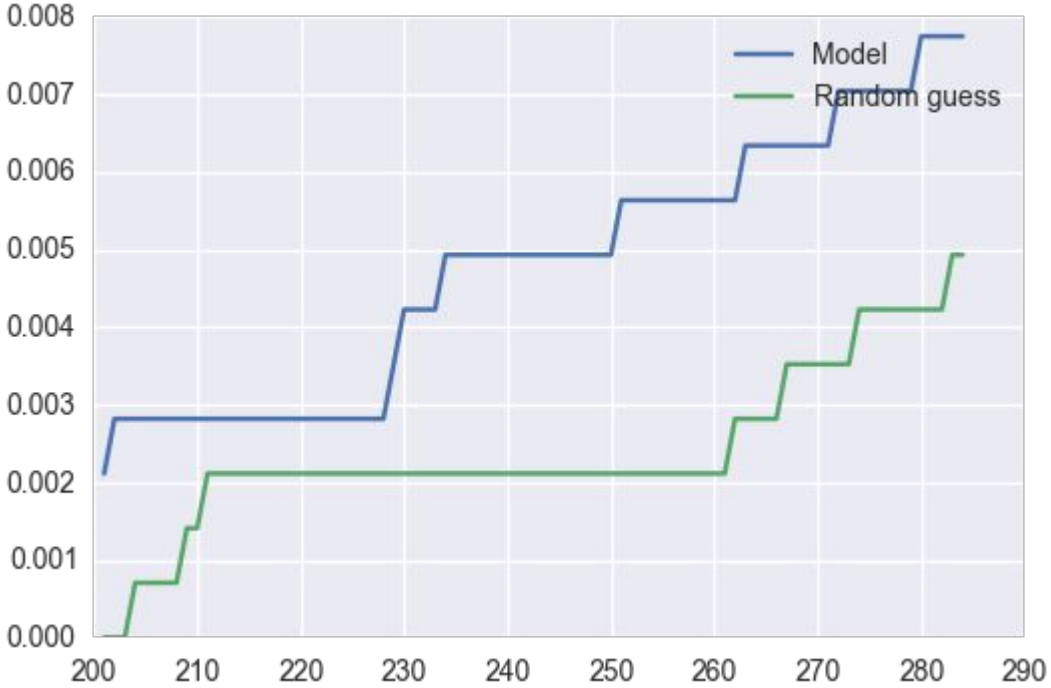
Leaving out validation data

- 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

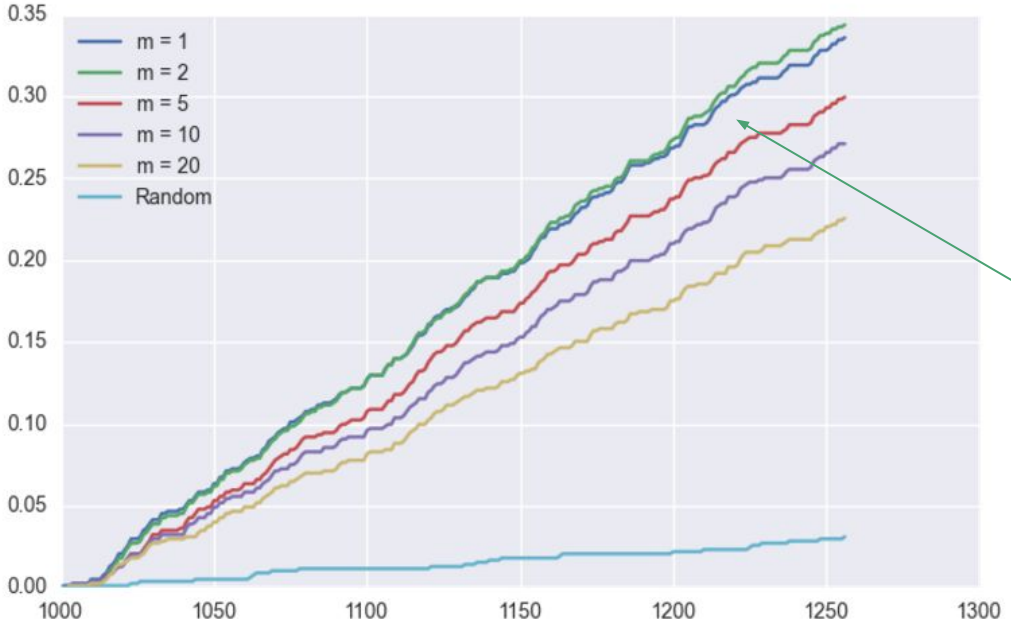


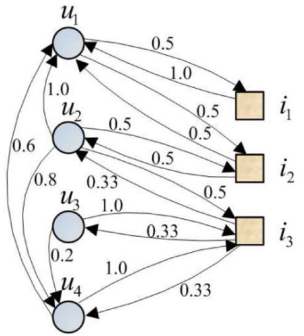
- Here we plot the performance of our model by comparing it to a random guess (much like ROC).
- We've made '5 predictions' and compared this to 5 random guesses.
- This is known as a gains chart

- *Item/Item collaborative model*

Graph Diffusion and Regularization

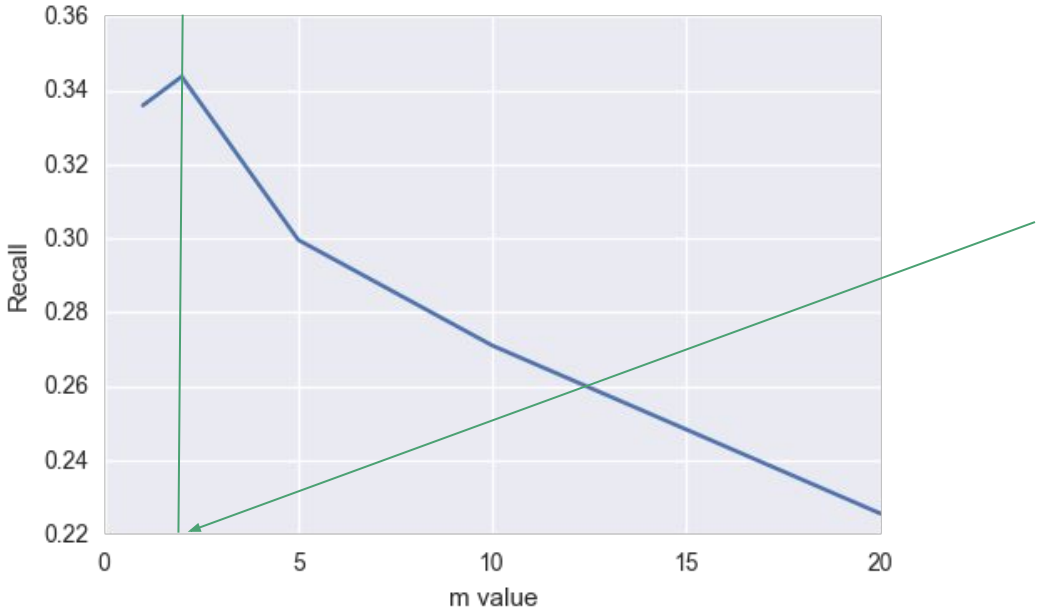
$$\pi_n^{m,\alpha} = (1-\alpha)(G^T M)^m q_n + \alpha q_{\text{pop}}$$





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

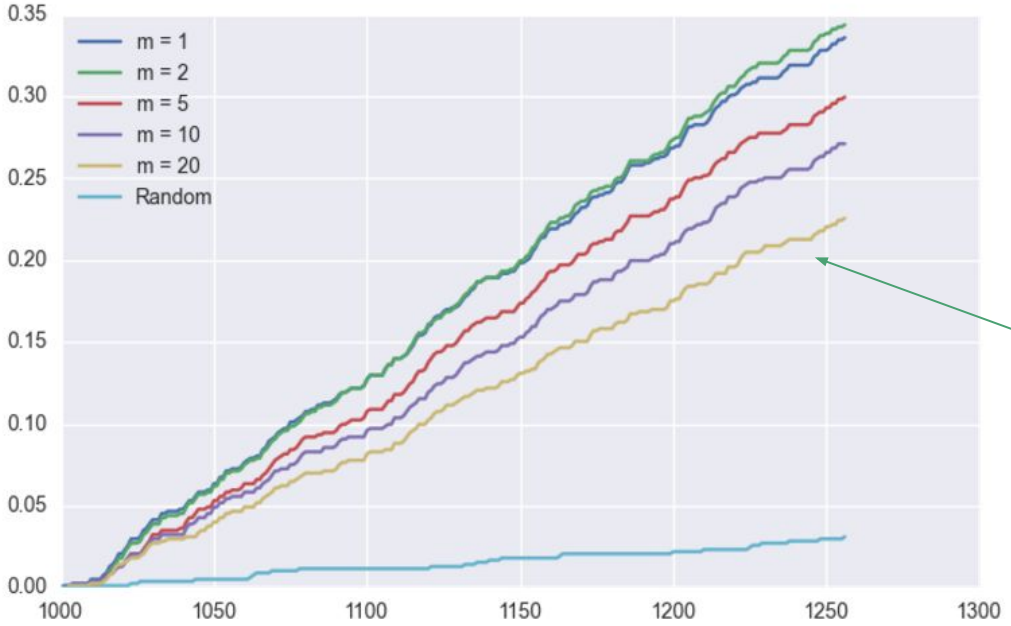
Finding the best m - parameter selection

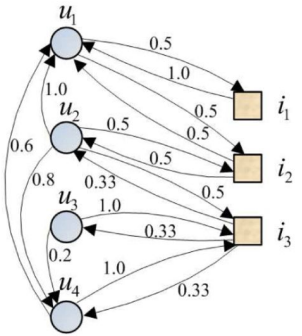


The best value of m is 2, since it maximizes recall on the holdout set.

Graph Diffusion and Regularization

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

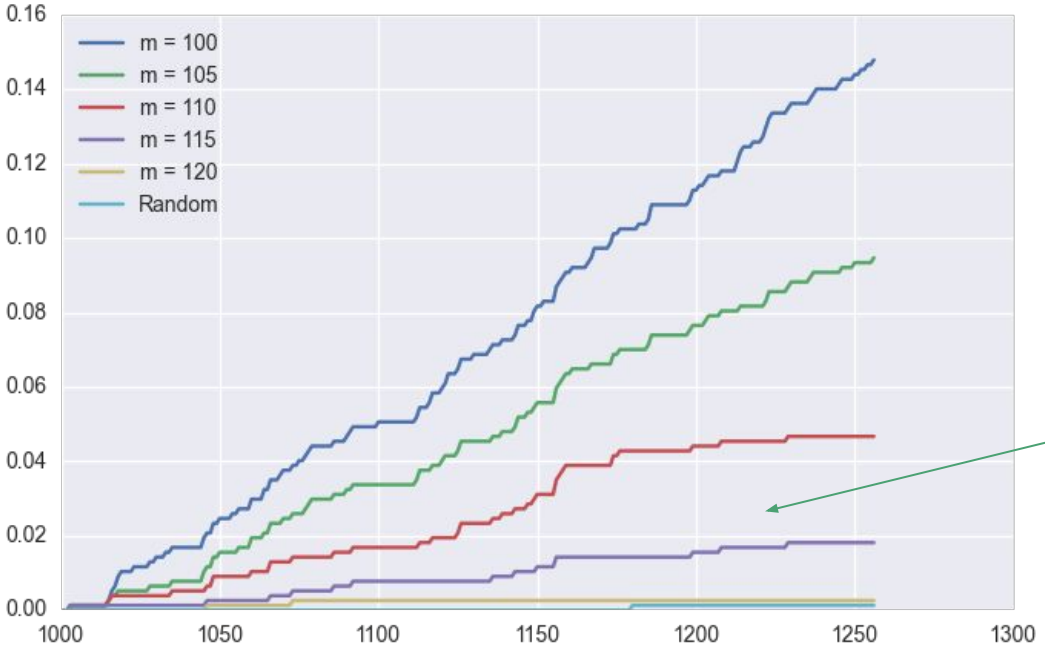


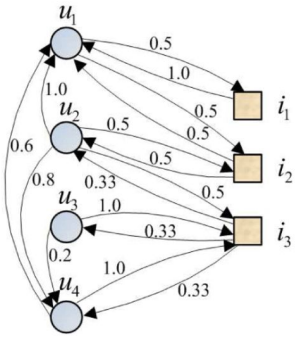


Recall that as we increase m we converge to the uniform distribution on the graph, so that this will eventually be no better than random.

Graph Diffusion and Regularization

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





Recall that as we increase m we converge to the uniform distribution on the graph, so that this will eventually be no better than random.

Python Implementation of Nearest Neighbors

1 contributor

36 lines (28 sloc) | 1.51 KB | Raw | Blame | History

```
# This program generates for each userid, 5 nearest neighbors, by using collaborative filterting. In particular, the sparse mat
# of userid/event is generated, then one takes the dot product of this matrix (normalized) with itself.
# This program should load csv files generated by RASparse_rowcol_generator.py (there are 2 files below since I did this in two

from scipy import sparse
import pandas as pd
from scipy.sparse import coo_matrix
import numpy as np
from sklearn.preprocessing import normalize

# Load in all row/column/id entries which will form our sparse matrix.
df_rowcols1 = pd.read_csv('../RA_row_col_id_urlSept25_2.csv', delim_whitespace=True)
df_rowcols2 = pd.read_csv('../RA_row_col_id_urlSept25_2Part2.csv', delim_whitespace=True)
rowcols = [df_rowcols1,df_rowcols2]
df_rowcols = pd.concat(rowcols, ignore_index=True).drop_duplicates()


# Generate sparse userid/event matrix.
rows = np.array(df_rowcols['row'])
columns = np.array(df_rowcols['column'])
data = [1.0]*len(columns)
X = coo_matrix((data, (rows,columns)), shape=(75988+1,25022+1))

# Normalize all of the columns
X_n = normalize(X, norm='l2', axis=1)

# Take dot product with transpose to generate matrix of user/user similarity.
Y = X_n.dot(X_n.T)

# Output the nearest neighbors (5) for each userid, by taking the top 5 entries from each row in Y.
print 'n1 n2 n3 n4 n5 row'
for i in range(0, 75988+1):
        row_nn = np.squeeze(np.asarray(Y.getrow(i).todense()))
        nnarr = np.argsort(row_nn)[-5:]
        print nnarr[0], nnarr[1], nnarr[2], nnarr[3], nnarr[4], i
```

Python Implementation of Graph Diffusion

In [261]:
```
df=X[0:]
M = df.div(df.sum(axis=1), axis=0)
G = df/df.sum().astype(np.float64)
```

In [262]:
```
G.T.shape
```

Out[262]: (285, 200)

In [244]:
```
M.shape
```

Out[244]: (1257, 285)

In [256]:
```
G.T.shape
```

Out[256]: (285, 1257)

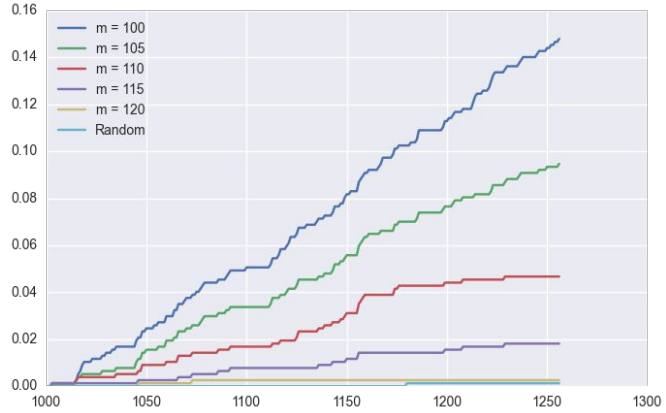
In [257]:
```
M.shape
```

Out[257]: (1257, 285)

In [259]:
```
A = pd.DataFrame(G.values*M.values, columns=df.columns, index=df.index)
```

Python Implementation of Graph Diffusion

```
best_m=1
recall_last = 0
plt.figure(figsize=(8,5))
recall_m= []
mvals=[1,2,3,4,5,6,7]
for m in mvals:
    B = (A)**m
    recall=[]
    guesses = 3
    recall_total = 0
    recall_guess = 0

    recall_guesses = []

    for n in range(1000,1257):
        user = np.append(B[n:n+1].values[0][0:200],np.zeros(85))
        user_prefs = df_items.dot(user/np.sum(user))

        user_prefs.sort()
        already_liked=df_items[df_items.columns[np.where(B[n:n+1].values[0]>0)][0:5]].columns.values

        test_data=df_items[df_items.columns[np.where(B[n:n+1].values[0]>0)][5:]].columns.values

        user_suggestions =[s for s in user_prefs.index[::-1][0:guesses] if s not in already_liked]

        correct = [c for c in user_suggestions if c in test_data]
        guess_list = [g for g in random.sample(user_prefs.index, min(guesses,len(user_prefs))) if g in test_data]

        recall_total = recall_total + float(len(correct))
        recall_guess = recall_guess + float(len(guess_list))

        recall.append(recall_total)
        recall_guesses.append(recall_guess)

    plt.plot(range(1000,1257),recall/(np.array(guesses)*257),label='m = ' + str(m))
    if recall_total > recall_last:
        recall_last = recall_total
        best_m = m
    recall_m.append(recall_total/(guesses*257))

plt.plot(range(1000,1257),recall_guesses/(np.array(guesses)*257),label='Random')
plt.legend(loc=2)
plt.show()
#print 'The best choice of m is ' + str(best_m)
plt.ylabel('Recall')
plt.xlabel('m value')
plt.plot(mvals,recall_m)
```



Additional Points

- Graph Diffusion generally results in better performance and allows for regularization which reduces overfitting.

- These models are good when you have a decent amount of user data, but not good for the ‘cold start’ problem often.

- This is batch offline learning - we will cover multiarmed bandits soon, which is a Bayesian approach to recommendations. It updates prior distributions in real time.