Lecture 11

Recommender Systems

By Nazerke Sultanova

Recomendations:

- making product recommendations for online shopping
- suggesting interesting web sites
- helping people find music and movies

What is Recommender Systems?

- **Recommenders are a type of filter**
- **They help users find relevant items within a huge selection**
  - How do you find an interesting movie among 95,000 choices?
  - They help you find things you didn't know to look for
- **Recommenders use preferences to predict preferences**
  - Input is feedback about likes and/or dislikes
  - Output is a list of suggested items based on feedback received
- **Two main types of recommenders**
  - Content-based
  - Collaborative filtering

Content-Based Recommenders

- **Content based recommenders consider an item's attributes**
  - These attributes describe the item
- **Examples of item attributes**
  - Movies: actor, director, screenwriter, producer, and location
  - Music: songwriter, style, musicians, vocalist, meter, and tempo
  - Books: author, publisher, subject, illustrations, and page count
- **A user's taste defines values and weights for each attribute**
  - These are supplied as input to the recommender

Collaborative Filtering

- **Collaborative filtering is an inherently social system**
  - It recommends items based on preferences of similar users
- **It's similar to how you get recommendations from friends**
  - Query those people who share your interests
  - They'll know movies you haven't seen and would probably like
    - And you'll be able to recommend some to them
- **This approach is not domain-specific**
  - System doesn't "know" anything about the items it recommends
  - The same algorithm can used to recommend any type of product
- **We'll discuss collaborative filtering in detail during this chapter**

Collaborative Filtering

- A collaborative filtering algorithm usually works by searching a large group of people and finding a smaller set with tastes similar to yours. It looks at other things they like and combines them to create a ranked list of suggestions.

Types of Collaborative Filtering

- **Collaborative filtering can be subdivided into two main types**
- **User-based: “What do users similar to you like?”**
  - For a given user, find other people who have similar tastes
  - Then, recommend items based on past behavior of those users
- **Item-based: “What is similar to other items you like?”**
  - Given items that a user likes, determine which items are similar
  - Make recommendations to the user based on those items

Collecting Preferences

- The first thing you need is a way to represent different people and their preferences.
- In Python, a very simple way to do this is to use a nested dictionary.

recommendations.py

```
# A dictionary of movie critics and their ratings of a small
# set of movies
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},
'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},
'Jack Matthews': {'Lady in the Water': 3.0, 'Snakes on a Plane': 4.0,
'The Night Listener': 3.0, 'Superman Returns': 5.0, 'You, Me and Dupree': 3.5},
'Toby': {'Snakes on a Plane':4.5,'You, Me and Dupree':1.0,'Superman Returns':4.0}}

```

- Using a dictionary is convenient for experimenting with the algorithms and for illus trative purposes. It’s easy to search and modify the dictionary.

```
In [1]: from recommendations import critics
        critics['Lisa Rose']['Lady in the Water']

Out[1]: 2.5

In [4]: critics['Toby']['Snakes on a Plane']=4.5
        critics['Toby']

Out[4]: {'Snakes on a Plane': 4.5, 'You, Me and Dupree': 1.0, 'Superman Returns': 4.0}
```

Finding Similar Users

- After collecting data about the things people like, you need a way to determine how similar people are in their tastes.
- You do this by comparing each person with every other person and calculating a similarity score.
- There are a few ways to do this, and today we describe two systems for calculating similarity scores:

  Euclidean distance and *Pearson correlation*.

Euclidean Distance Score

- One very simple way to calculate a similarity score is to use a Euclidean distance score, which takes the items that people have ranked in common and uses them as axes for a chart.
- You can then plot the people on the chart and see how close together they are

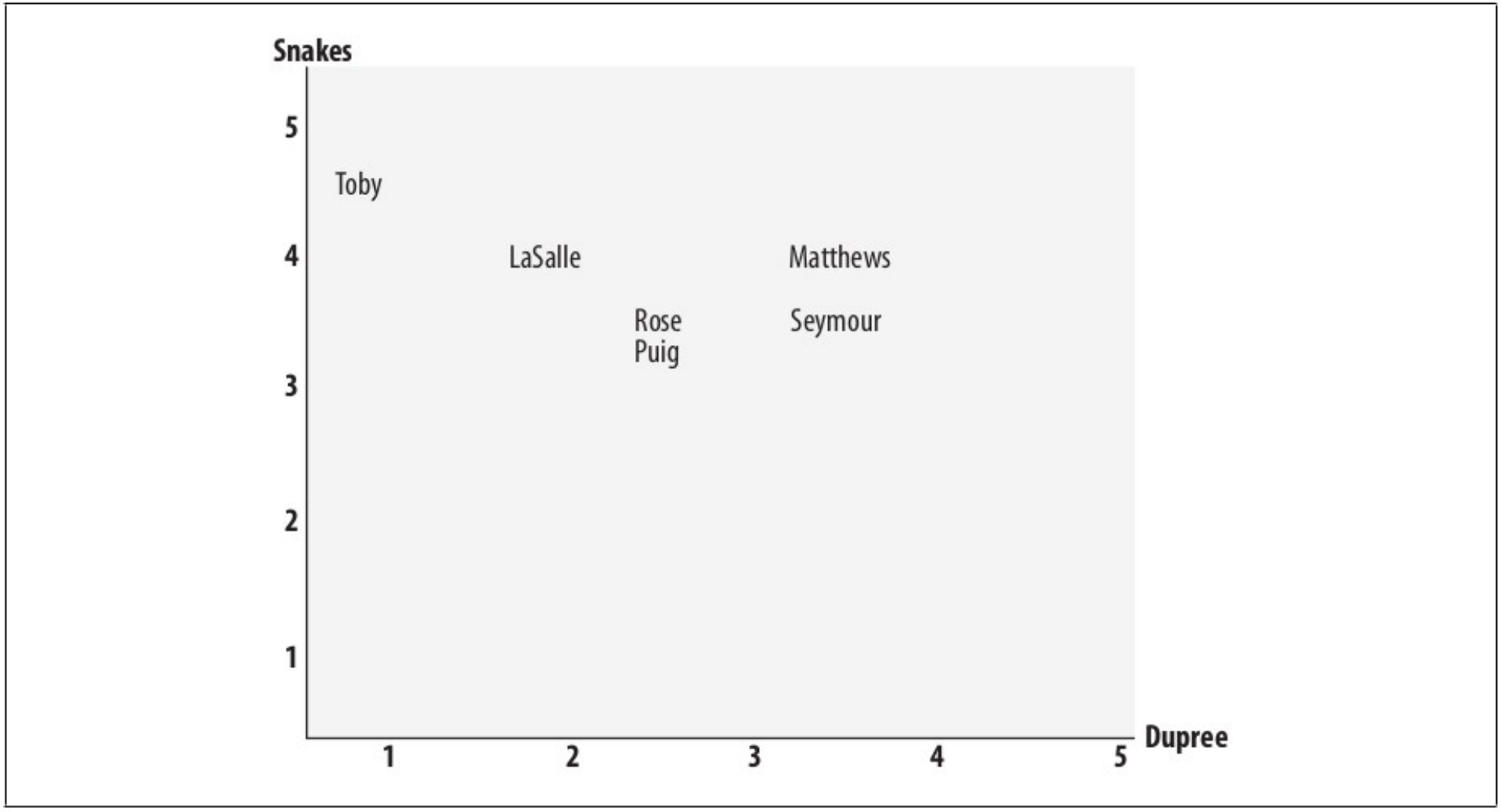

Figure 2-1. People in preference space

Example

- To calculate the distance between Toby and LaSalle in the chart, take the difference in each axis, square them and add them together, then take the square root of the sum.

```
In [14]: from math import sqrt
         sqrt(pow(5-4,2)+pow(4-1,2))

Out[14]: 3.1622776601683795
```

- This formula calculates the distance, which will be smaller for people who are more similar. However, you need a function that gives higher values for people who are similar.
- This can be done by adding 1 to the function (so you don’t get a division-by-zero error) and inverting it:

```
In [15]: 1/(1+sqrt(pow(5-4,2)+pow(4-1,2)))
Out[15]: 0.240253073352042
```

- This new function always returns a value between 0 and 1, where a value of 1 means that two people have identical preferences.
- You can put everything together to create a function for calculating similarity.

In [35]:
```
def sim_distance(prefs,person1,person2):
# Get the list of shared_items
    si={}
    for item in prefs[person1]:
        if item in prefs[person2]:
            si[item]=1
# if they have no ratings in common, return 0
    if len(si)==0: return 0
# Add up the squares of all the differences
    sum_of_squares=sum([pow(prefs[person1][item]-prefs[person2][item],2)
                        for item in prefs[person1] if item in prefs[person2]])
    return 1/(1+sum_of_squares)
sim_distance(r.critics, 'Lisa Rose','Gene Seymour')
```

Out[35]:
```
0.14814814814814814
```

Pearson Correlation Score

- A slightly more sophisticated way to determine the similarity between people's interests is to use a Pearson correlation coefficient. The correlation coefficient is a measure of how well two sets of data fit on a straight line.
- The formula for this is more complicated than the Euclidean distance score, but it tends to give better results in situations where the data isn't well normalized—for example, if critics' movie rankings are routinely more harsh than average.

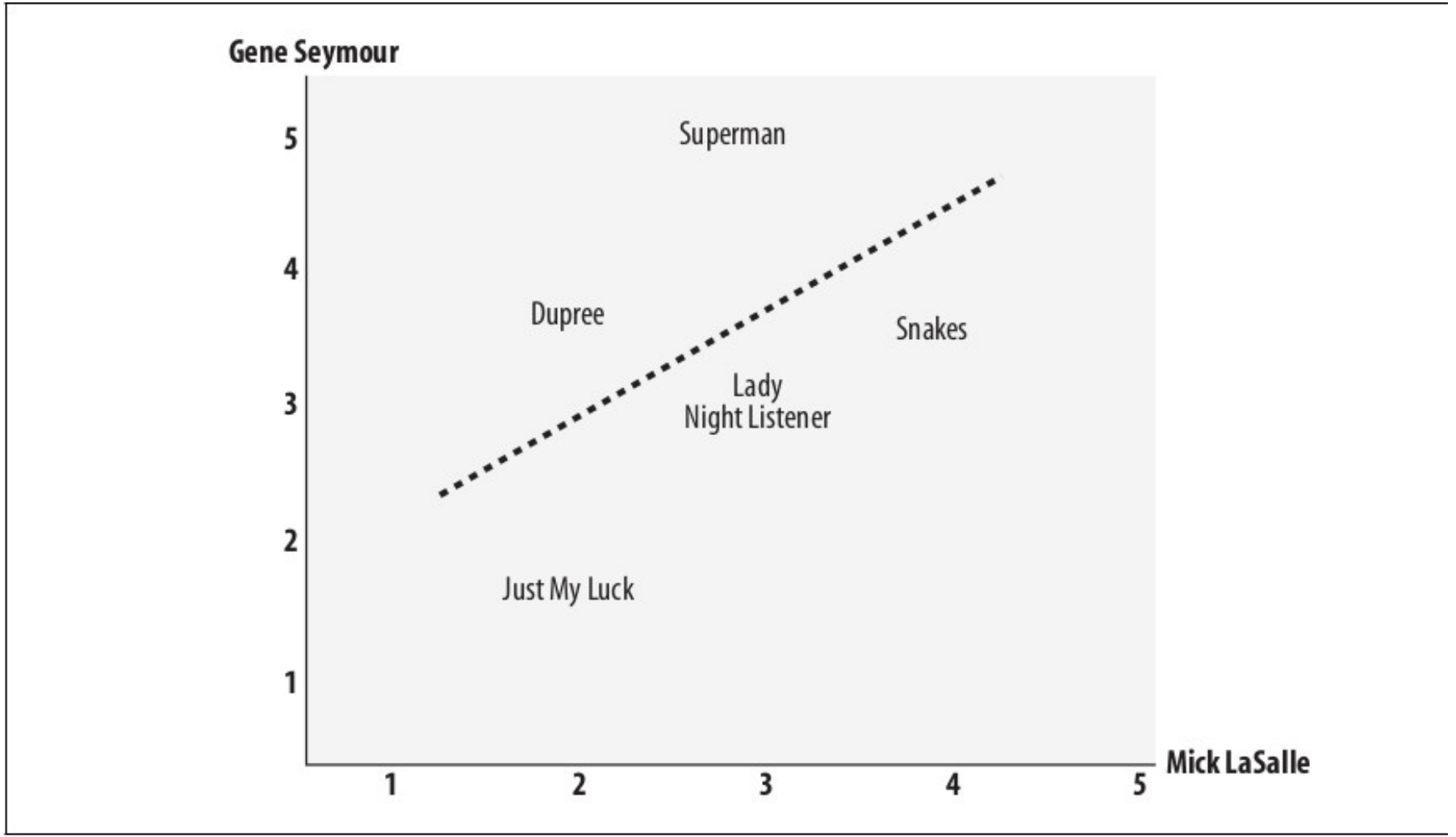

Figure 2-2. Comparing two movie critics on a scatter plot

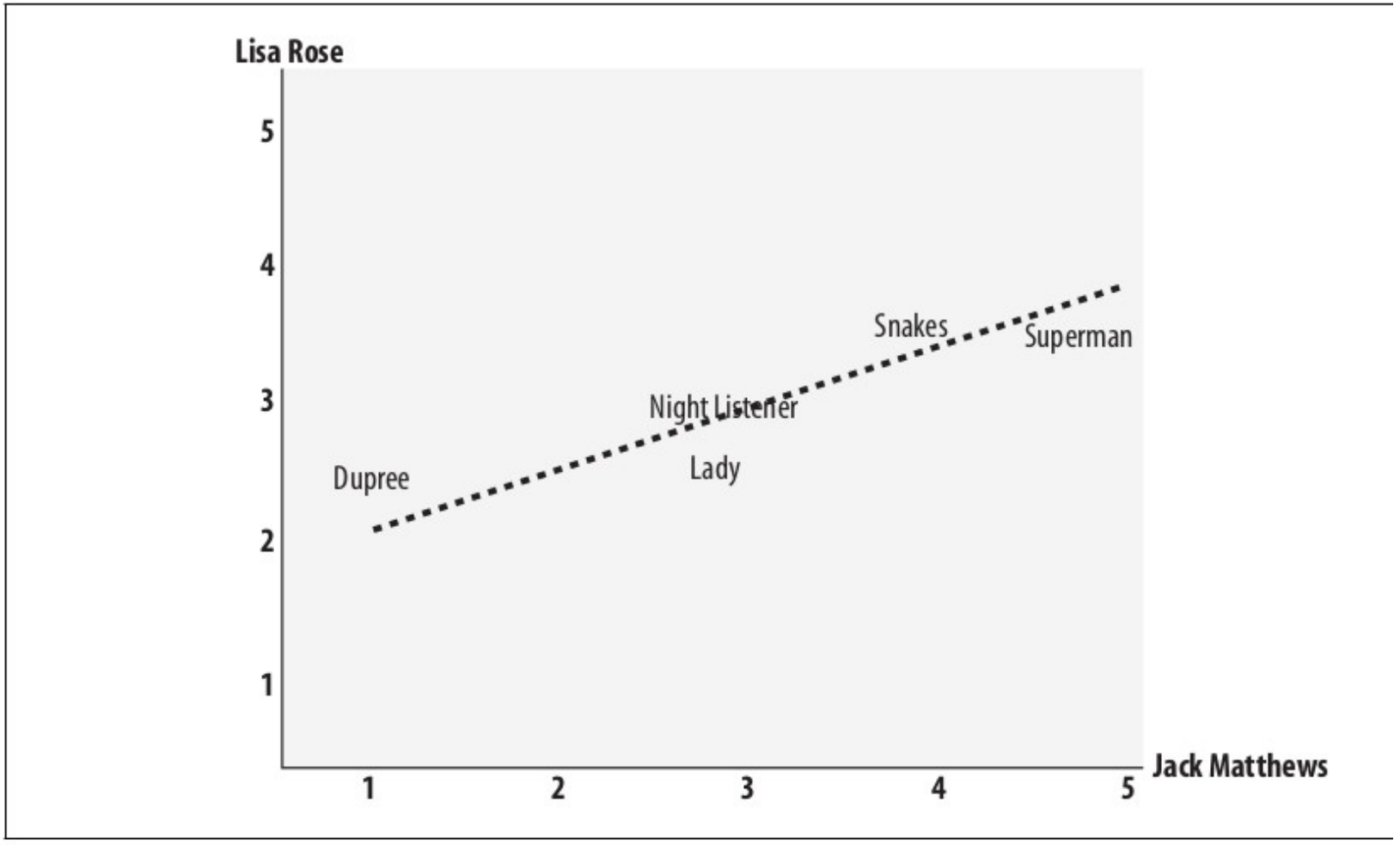

Figure 2-3. Two critics with a high correlation score

Ranking Critics

In [43]:
```
# Returns the best matches for person from the prefs dictionary.
# Number of results and similarity function are optional params.
def topMatches(prefs,person,n=5,similarity=sim_pearson):
    scores=[(similarity(prefs,person,other),other)
    for other in prefs if other!=person]
# Sort the list so the highest scores appear at the top
    scores.sort( )
    scores.reverse( )
    return scores[0:n]
topMatches(r.critics,'Toby',n=3)
```

Out[43]:
```
[(0.991240707161929, 'Lisa Rose'),
 (0.9244734516419049, 'Mick LaSalle'),
 (0.893405147441564 7, 'Claudia Puig')]
```

Recommending Items

- Finding a good critic to read is great, but what I really want is a movie recommendation right now. I could just look at the person who has tastes most similar to mine and look for a movie he likes that I haven’t seen yet, but that would be too permissive.
- WHY?

Table 2-2. Creating recommendations for Toby

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

```python
# Gets recommendations for a person by using a weighted average
# of every other user's rankings
def getRecommendations(prefs,person,similarity=sim_pearson):
    totals={}
    simSums={}
    for other in prefs:
# don't compare me to myself
        if other==person: continue
        sim=similarity(prefs,person,other)
# ignore scores of zero or lower
        if sim<=0: continue
        for item in prefs[other]:
# only score movies I haven't seen yet
            if item not in prefs[person] or prefs[person][item]==0:
# Similarity * Score
                totals.setdefault(item,0)
                totals[item]+=prefs[other][item]*sim
# Sum of similarities
                simSums.setdefault(item,0)
                simSums[item]+=sim
# Create the normalized list
    rankings=[(total/simSums[item],item) for item,total in totals.items( )]
# Return the sorted list
    rankings.sort( )
    rankings.reverse( )
    return rankings
print(getRecommendations(r.critics,'Toby'))
print(getRecommendations(r.critics,'Toby', similarity=sim_distance))
```

```
[(3.3477895267131017, 'The Night Listener'), (2.8325499182641614, 'Lady in the Water'), (2.530980703765565, 'Just My Luck')]
[(3.5002478401415877, 'The Night Listener'), (2.7561242939959363, 'Lady in the Water'), (2.461988486074374, 'Just My Luck')]
```

Matching Products

- Now you know how to find similar people and recommend products for a given person, but what if you want to see which products are similar to each other?
- You may have encountered this on shopping web sites, particularly when the site hasn’t collected a lot of information about you.

Transform dictionary

```
{'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..
```

Function for transforming:

```
def transformPrefs(prefs):
  result={}
  for person in prefs:
    for item in prefs[person]:
      result.setdefault(item,{})

      # Flip item and person
      result[item][person]=prefs[person][item]
  return result
```

Item-Based Filtering

- In cases with very large datasets, item-based collaborative filtering can give better results, and it allows many of the calculations to be performed in advance so that a user needing recommendations can get them more quickly.

Calculate similar items:

```
def calculateSimilarItems(prefs,n=10):
  # Create a dictionary of items showing which other items they
  # are most similar to.
  result={}

  # Invert the preference matrix to be item-centric
  itemPrefs=transformPrefs(prefs)
  c=0
  for item in itemPrefs:
    # Status updates for large datasets
    c+=1
    if c%100==0: print "%d / %d" % (c,len(itemPrefs))
    # Find the most similar items to this one
    scores=topMatches(itemPrefs,item,n=n,similarity=sim_distance)
    result[item]=scores
  return result
```

Getting Recommendations

Table 2-3. Item-based recommendations for Toby

Movie	Rating	Night	R.xNight	Lady	R.xLady	Luck	R.xLuck
Snakes	4.5	0.182	0.818	0.222	0.999	0.105	0.474
Superman	4.0	0.103	0.412	0.091	0.363	0.065	0.258
Dupree	1.0	0.148	0.148	0.4	0.4	0.182	0.182
Total		0.433	1.378	0.713	1.764	0.352	0.914
Normalized			3.183		2.598		2.473

User-Based or Item-Based Filtering?

- Item-based filtering is significantly faster than user-based when getting a list of recommendations for a large dataset, but it does have the additional overhead of maintaining the item similarity table.
- Also, there is a difference in accuracy that dependson how “sparse” the dataset is.