Recommendation Systems

Dr. Johan Hagelbäck

johan.hagelback@lnu.se

http://aiguy.org

Recommendation Systems

- The task:
  - Find new items (movies, books, music, …) you may like based on what you have liked before
- Usages:
  - Shopping sites: finding items you didn't know existed but that you may like
  - Streaming sites like Netflix/Spotify: find movies/songs you may like but have never heard of
  - Sites like reddit: suggest new sites that you may like

The most simple approach

- You want to see a new movie, but you don’t know which movies are good
- You ask your friends about what movies they liked that you haven’t seen yet
- You can listen more to friends that you know have good ”taste”, meaning that they usually like the same movies as you do
- Drawbacks:
  - Time consuming
  - Very limited amount of data meaning that you miss movies none of your friends have seen

Collaborative Filtering

- Collaborative Filtering is a set of techniques for making automatic recommendations for a user
- The term was first used by David Goldberg at Xerox PARC in 1992
- He developed a system for automatic recommendation of documents based on what a user previously has labeled as *interesting* or *uninteresting*

User Preferences

- The first step in developing a Collaborative Filtering system is to store the data, i.e. user preferences
- Preferences must be numeric, for example a scale between 1 and 5 for how good a movie is
- Non-numeric preferences can be translated:
  - Buy: 1, Not buy: 0
  - Buy: 2, Browsed: 1, Not buy: 0
  - Liked: 1, Disliked: -1, No vote: 0
- We will use a small data set consisting of seven users and six movies:

The Data Set

Movie	Lisa	Gene	Michael	Claudia	Mick	Jack	Toby
Lady in the Water	2.5	3.0	2.5		3.0	3.0	
Snakes on a Plane	3.5	3.5	3.0	3.5	4.0	4.0	4.5
Just My Luck	3.0	1.5		3.0	2.0		
Superman Returns	3.5	5.0	3.5	4.0	3.0	5.0	4.0
You, Me and Dupree	2.5	3.5		2.5	2.0	3.5	1.0
The Night Listener	3.0	3.0	4.0	4.5	3.0	3.0	

How to find a new movie?

- The most simple approach is to average the score on movies you haven't seen:

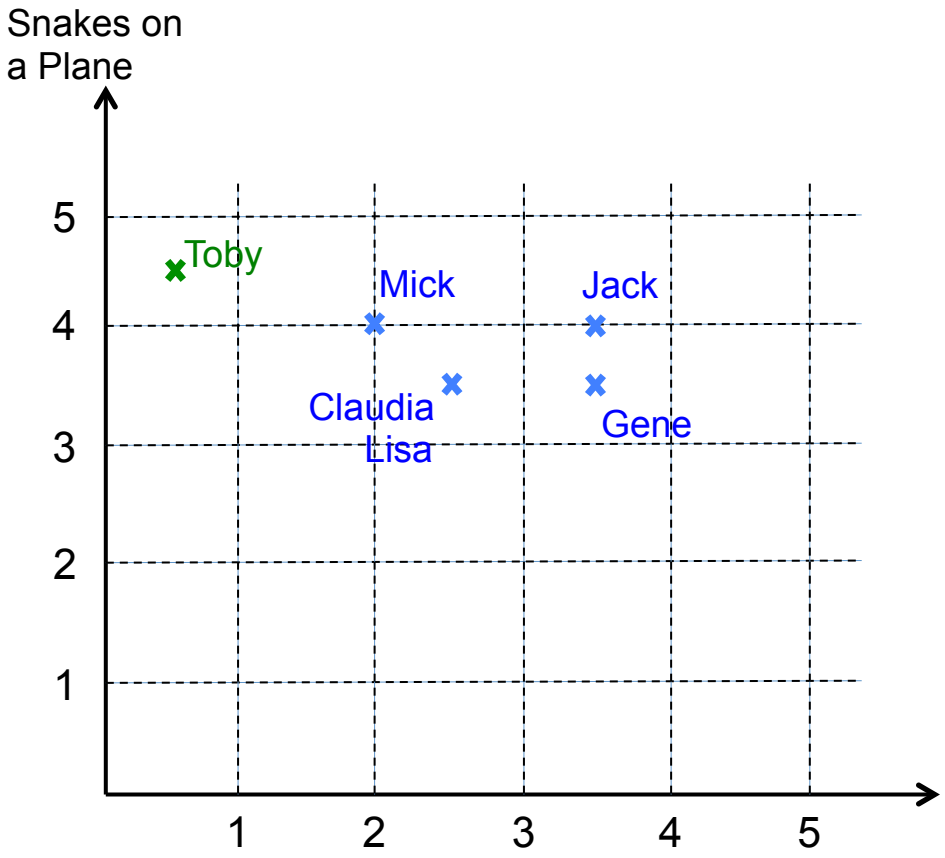
Movie	Lisa	Gene	Mike	Claudia	Mick	Jack	Toby
Lady in the Water	2.5	3.0	2.5		3.0	3.0	**2.8**
Snakes on a Plane	3.5	3.5	3.0	3.5	4.0	4.0	4.5
Just My Luck	3.0	1.5		3.0	2.0		**2.38**
Superman Returns	3.5	5.0	3.5	4.0	3.0	5.0	4.0
You, Me and Dupree	2.5	3.5		2.5	2.0	3.5	1.0
The Night Listener	3.0	3.0	4.0	4.5	3.0	3.0	**3.42**

A better approach

Finding Similar Users

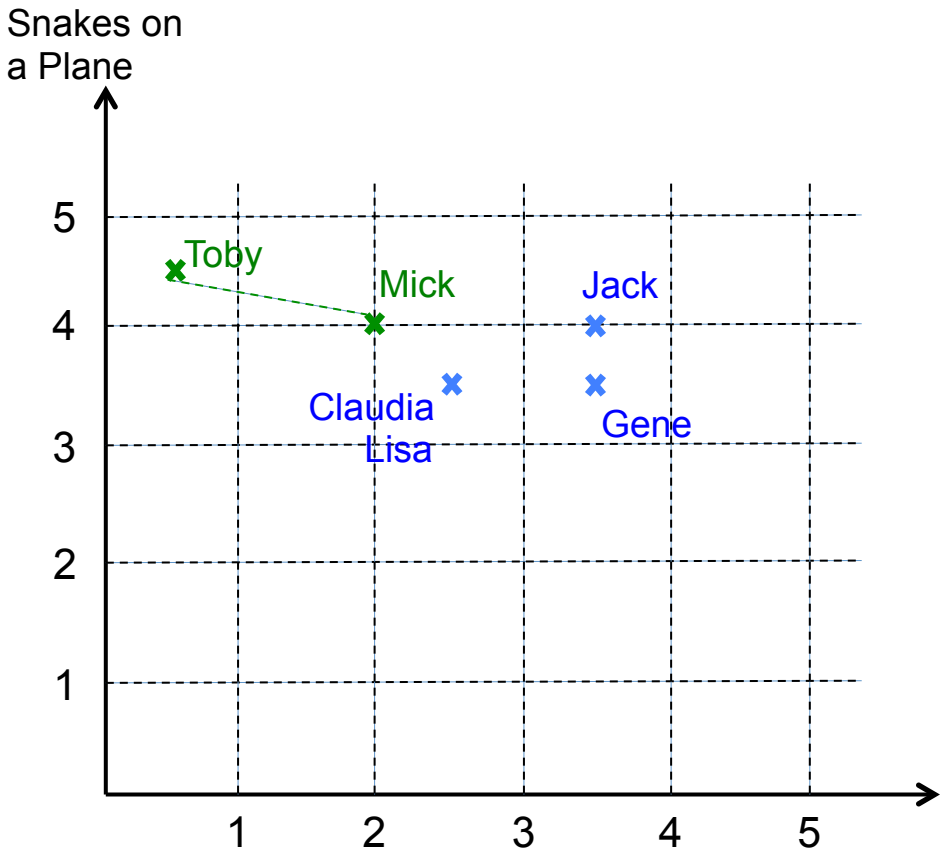
- A better approach is to find users similar to yourself
- This is done by comparing every user with every other user and calculate a *similarity score*
- There are many ways to calculate similarity
- Here we will take a look at two of them:
  - *Euclidean Distance*
  - *Pearson Correlation*

Euclidean Distance



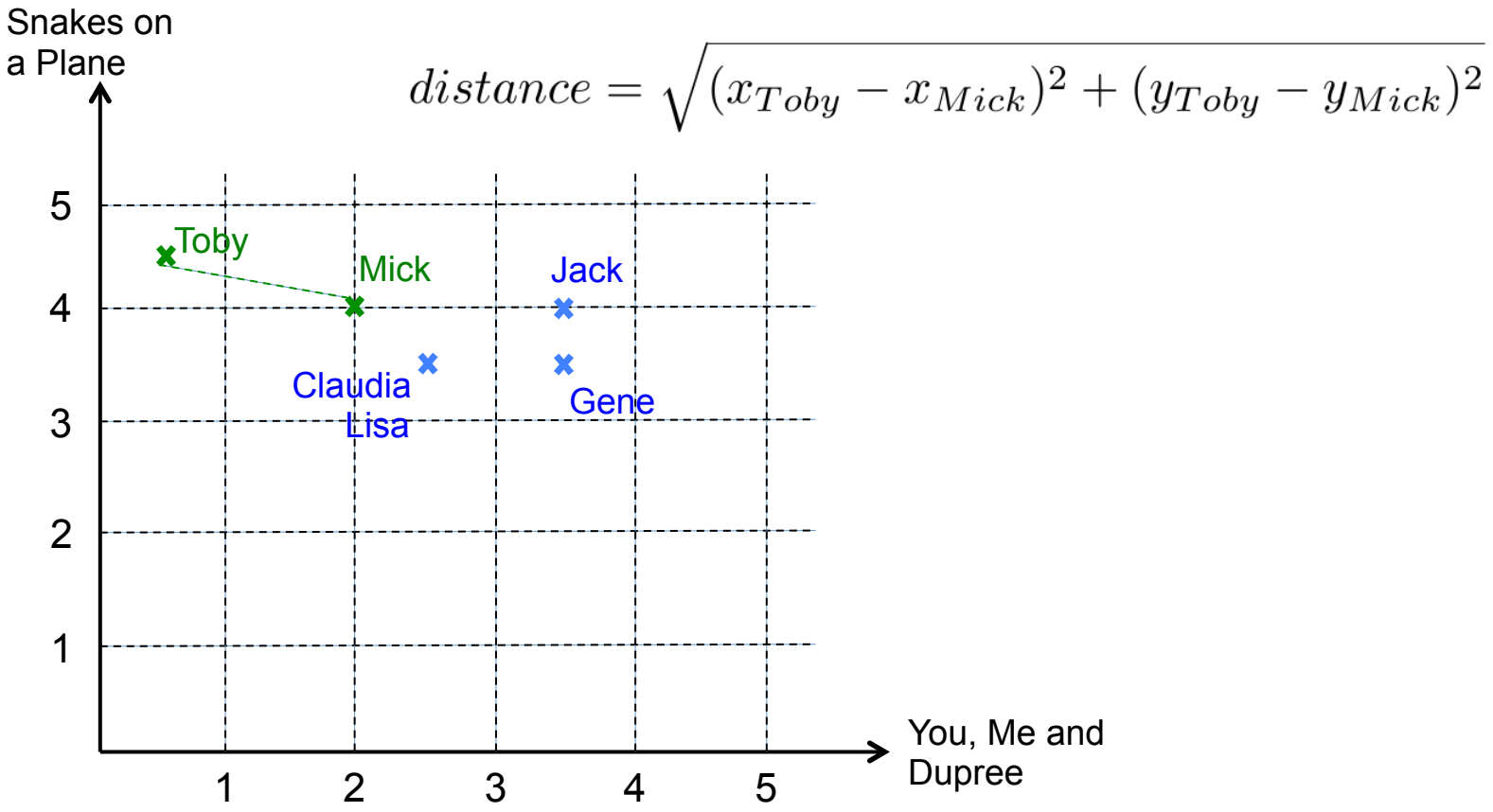
Question: which user is the best match for Toby?

Euclidean Distance



For user Toby,
Mick is the closest
match!

Euclidean Distance



Euclidean Distance

- The distance is however smaller for people who are more similar, but we want the opposite!
- Therefore we have to invert it, and add a 1 to avoid division by zero:

$$similarity = \frac{1}{1 + \sqrt{(x_{Toby} - x_{Mick})^2 + (y_{Toby} - y_{Mick})^2}}$$

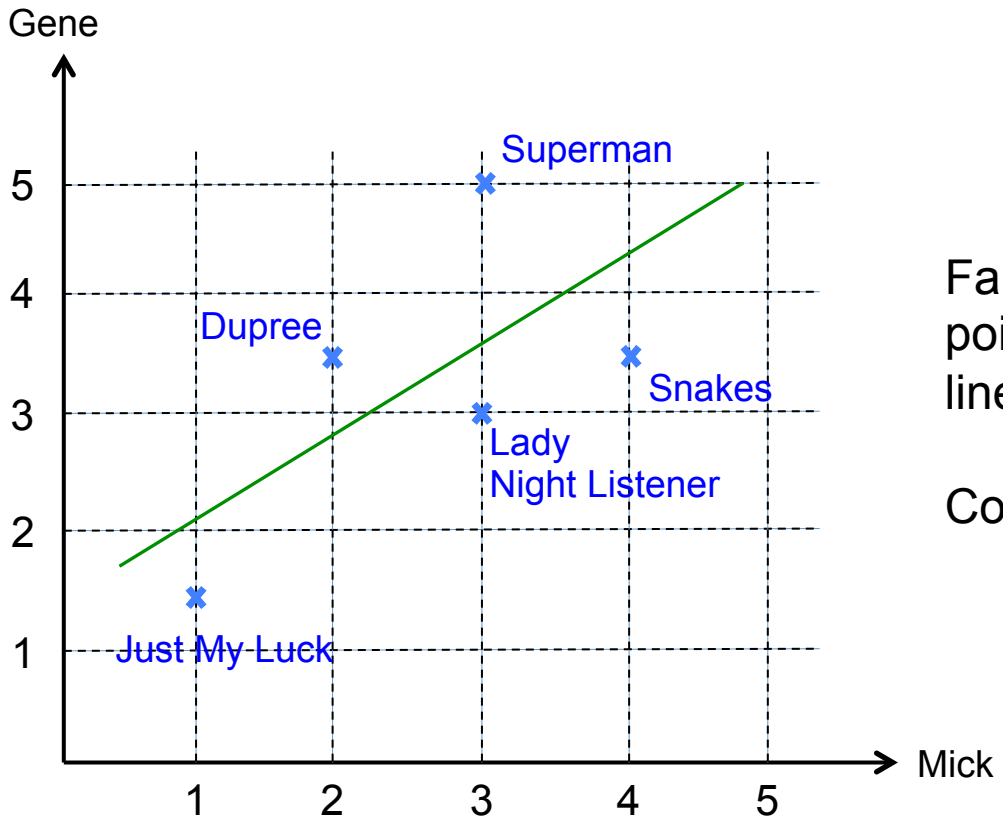
Euclidean Distance

```
float euclidean(User A, User B)
  //Init variables
  sim=0
  //Counter for number of matching products
  n = 0
  //Iterate over all rating combinations
  for (Rating rA : A.rating)
    for (Rating rB : B.rating)
      if (rA == rB)
        sim += (rA.score – rB.score)**2 //a*a
        n += 1
  //No ratings in common – return 0
  if (n == 0)
    return 0
  //Calculate inverted score
  inv = 1 / (1 + sim)
  return inv
```


Pearson Correlation Score

- PCS is a more sophisticated way to calculate similarity
- The correlation coefficient is a measure of how well two sets of data fit on a straight line
- If all data points fit on the straight line, we have a perfect match resulting on correlation score 1
- PCS tends to give better score for data that isn't well normalized, for example if a harsh user routinely give lower scores than the other users
- It is also more robust to *grade inflation*, where one user consistently gives higher (or lower) scores than the other users

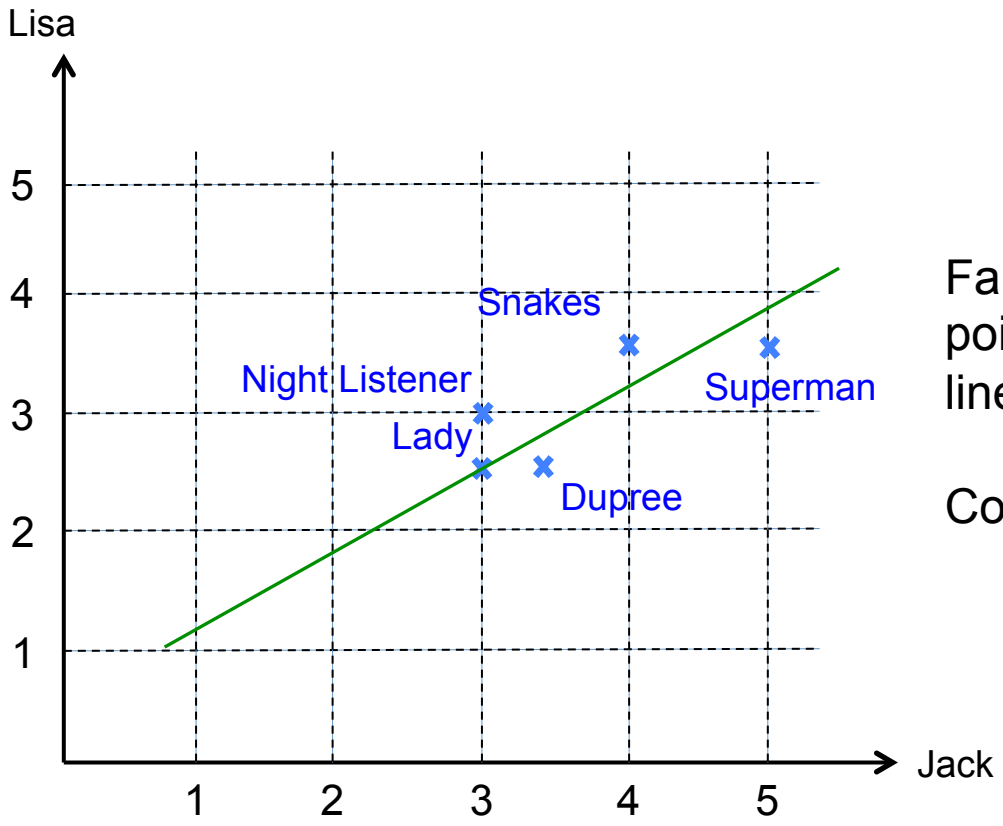
Pearson Correlation Score



Fairly bad fit: the data points are far from the line.

Correlation: 0.4

Pearson Correlation Score



Fairly good fit: the data points are close to the line.

Correlation: 0.75

Pearson Correlation Score

- To calculate the Pearson Correlation do the following steps:
  - Calculate the number of movies both users have rated: n
  - Calculate the sum of all scores for user 1: sum1
  - Calculate the sum of all scores for user 2: sum2
  - Calculate the sum of the squares of scores for user 1: sum1sq
  - Calculate the sum of the squares of scores for user 2: sum2sq
  - Calculate the product of scores for user 1 and user 2: pSum

Pearson Correlation Score

- Calculate the Pearson Correlation score r:

$$num = pSum - (\frac{sum1 \cdot sum2}{n})$$

$$den = \sqrt{(sum1sq - \frac{sum1^2}{n}) \cdot (sum2sq - \frac{sum2^2}{n})}$$

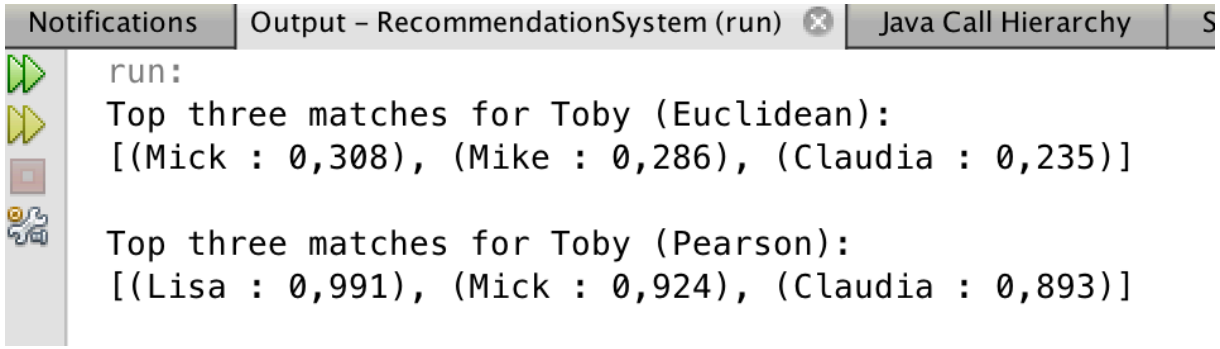
$$r = \frac{num}{den}$$

Pearson Correlation Score

```
float pearson(User A, User B)
  //Init variables
  sum1=0, sum2=0, sum1sq=0, sum2sq=0, pSum=0
  //Counter for number of matching products
  n = 0
  //Iterate over all rating combinations
  for (Rating rA : A.rating)
    for (Rating rB : B.rating)
      if (rA == rB)
        sum1 += rA.score
        sum2 += rB.score
        sum1sq += rA.score**2 //rA*rA
        sum2sq += rB.score**2 //rB*rB
        pSum += rA.score * rB.score
        n += 1
  //No ratings in common – return 0
  if (n == 0)
    return 0
  //Calculate Pearson
  num = pSum – (sum1 * sum2 / n)
  den = sqrt((sum1sq – sum1**2 / n) * (sum2sq – sum2**2 / n))
  return num/den
```


Find top three matches for a user

- This can be done as follows:
  - Calculate the similarity score between the user and all other users
  - Store the scores in a list
  - Sort the list in descending order (highest scores first)
  - Return the first 3 entries in the list
- The result:



Which similarity metric to use?

- There are many other metrics than the two mentioned:
  - *Manhattan Distance*
  - *Jaccard Coefficient*
  - …
- There is no universal answer to which one is the best to use
- It depends on the application
- Try at least *Euclidean* and *Pearson* to see which one works best in your case!

Finding recommended movies

Recommending Items

- Finding similar users is just the first step
- What we really want to know is a movie recommendation
- To do this, we need to calculate a weighted score for each user and movie
- Task: to find a movie recommendation for user Toby
- We calculate and put the weighted scores in a table (note that Mike is removed from the dataset to make the table easier to visualize):

Weighted Scores

- Calculate the weighted score as the similarity multiplied by the rating (assuming Pearson as similarity measure):

User	Similarity	Night	WS Night	Lady	WS Lady	Luck	WS Luck
Lisa	0.99	3.0	2.97	2.5	2.48	3.0	2.97
Gene	0.38	3.0	1.14	3.0	1.14	1.5	0.57
Claudia	0.89	4.5	4.02			3.0	2.68
Mick	0.92	3.0	2.77	3.0	2.77	2.0	1.85
Jack	0.66	3.0	1.99	3.0	1.99		

Weighted Scores

- Calculate the totals:

User	Similarity	Night	WS Night	Lady	WS Lady	Luck	WS Luck
Lisa	0.99	3.0	2.97	2.5	2.48	3.0	2.97
Gene	0.38	3.0	1.14	3.0	1.14	1.5	0.57
Claudia	0.89	4.5	4.02			3.0	2.68
Mick	0.92	3.0	2.77	3.0	2.77	2.0	1.85
Jack	0.66	3.0	1.99	3.0	1.99		
Total			12.89		8.38		8.07

Weighted Scores

- Calculate the sum of similarity for all users with matching movies:

User	Similarity	Night	WS Night	Lady	WS Lady	Luck	WS Luck
Lisa	0.99	3.0	2.97	2.5	2.48	3.0	2.97
Gene	0.38	3.0	1.14	3.0	1.14	1.5	0.57
Claudia	0.89	4.5	4.02			3.0	2.68
Mick	0.92	3.0	2.77	3.0	2.77	2.0	1.85
Jack	0.66	3.0	1.99	3.0	1.99		
Total			12.89		8.38		8.07
Sum (similarity)			3.84		2.95		3.18

Weighted Scores

- Calculate the total divided by sum of similarity:

User	Similarity	Night	WS Night	Lady	WS Lady	Luck	WS Luck
Lisa	0.99	3.0	2.97	2.5	2.48	3.0	2.97
Gene	0.38	3.0	1.14	3.0	1.14	1.5	0.57
Claudia	0.89	4.5	4.02			3.0	2.68
Mick	0.92	3.0	2.77	3.0	2.77	2.0	1.85
Jack	0.66	3.0	1.99	3.0	1.99		
Total			12.89		8.38		8.07
Sum_sim			3.84		2.95		3.18
Total / Sum_sim			3.35		2.83		2.53

Which movie to see?

- From the table we can see that Toby shall watch The Night Listener:

User	Similarity	Night	WS Night	Lady	WS Lady	Luck	WS Luck
Lisa	0.99	3.0	2.97	2.5	2.48	3.0	2.97
Gene	0.38	3.0	1.14	3.0	1.14	1.5	0.57
Claudia	0.89	4.5	4.02			3.0	2.68
Mick	0.92	3.0	2.77	3.0	2.77	2.0	1.85
Jack	0.66	3.0	1.99	3.0	1.99		
Total			12.89		8.38		8.07
Sum_sim			3.84		2.95		3.18
Total / Sum_sim			3.35		2.83		2.53

And with Euclidean Distance

- Same table using Euclidean Distance as similarity score:

User	Similarity	Night	WS Night	Lady	WS Lady	Luck	WS Luck
Lisa	0.222	3.0	0.66	2.5	0.555	3.0	0.66
Gene	0.108	3.0	0.324	3.0	0.324	1.5	0.162
Claudia	0.235	4.5	1.0575			3.0	0.705
Mick	0.308	3.0	0.924	3.0	0.924	2.0	0.616
Jack	0.118	3.0	0.354	3.0	0.354		
Total			3.3195		2.157		2.143
Sum_sim			0.991		0.756		0.873
Total / Sum_sim			3.35		2.85		2.45

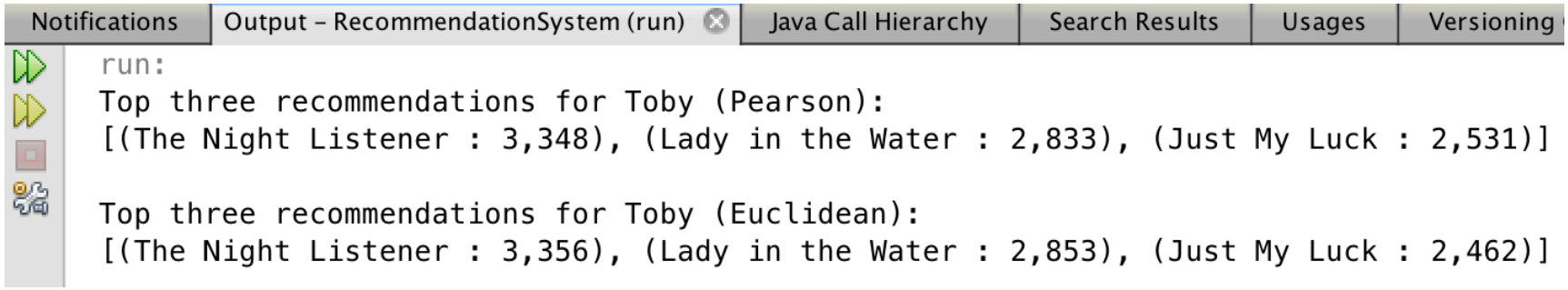
Which movie to see?

- The Night Listener is still the recommended movie for Toby:

User	Similarity	Night	WS Night	Lady	WS Lady	Luck	WS Luck
Lisa	0.222	3.0	0.66	2.5	0.555	3.0	0.66
Gene	0.108	3.0	0.324	3.0	0.324	1.5	0.162
Claudia	0.235	4.5	1.0575			3.0	0.705
Mick	0.308	3.0	0.924	3.0	0.924	2.0	0.616
Jack	0.118	3.0	0.354	3.0	0.354		
Total			3.3195		2.157		2.143
Sum_sim			0.991		0.756		0.873
Total / Sum_sim			3.35		2.85		2.45

Testing it

- The following output is generated when running the code:



- Note that values can differ slightly from the tables due to decimal rounding
- Again, note that Mike is removed from the data set

Find matching movies

Finding matching products

- It is also possible to find matching products
- To do this we need to transpose the data set so movies replaces users:

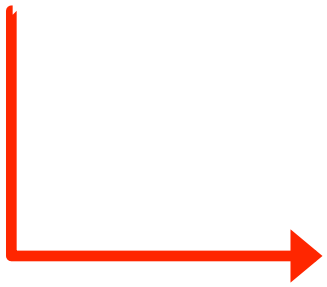
From:

Lisa: [(Lady in the Water : 2.5), (Snakes on a Plane : 3.5)]
Gene: [(Lady in the Water : 3.0), (Snakes on a Plane : 3.5)]

To:

Lady in the Water: [(Lisa : 2.5), (Gene : 3.0)]
Snakes on a Plane: [(Lisa : 3.5), (Gene : 3.5)]

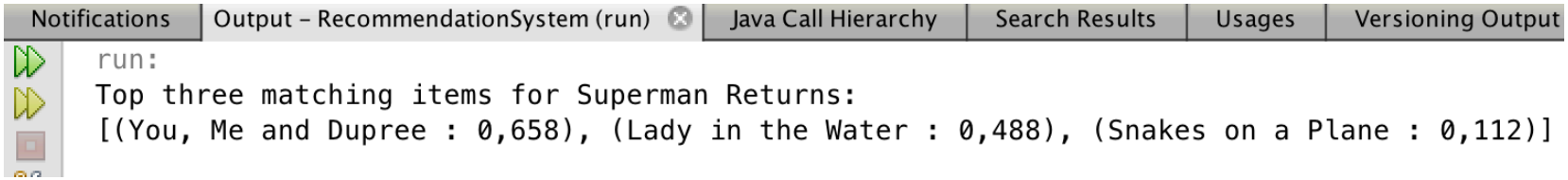
Movie	Lisa	Gene	Michael	Claudia	Mick	Jack	Toby
Lady in the Water	2.5	3.0	2.5		3.0	3.0	
Snakes on a Plane	3.5	3.5	3.0	3.5	4.0	4.0	4.5
Just My Luck	3.0	1.5		3.0	2.0		
Superman Returns	3.5	5.0	3.5	4.0	3.0	5.0	4.0
You, Me and Dupree	2.5	3.5		2.5	2.0	3.5	1.0
The Night Listener	3.0	3.0	4.0	4.5	3.0	3.0	



User	Lady	Snakes	Luck	Superman	Dupree	Night
Lisa	2.5	3.5	3.0	3.5	2.5	3.0
Gene	3.0	3.5	1.5	5.0	3.5	3.0
Michael	2.5	3.0		3.5		4.0
Claudia		3.5	3.0	4.0	2.5	4.5
Mick	3.0	4.0	2.0	3.0	2.0	3.0
Jack	3.0	4.0		5.0	3.5	3.0
Toby		4.5		4.0	1.0	

Finding matching products

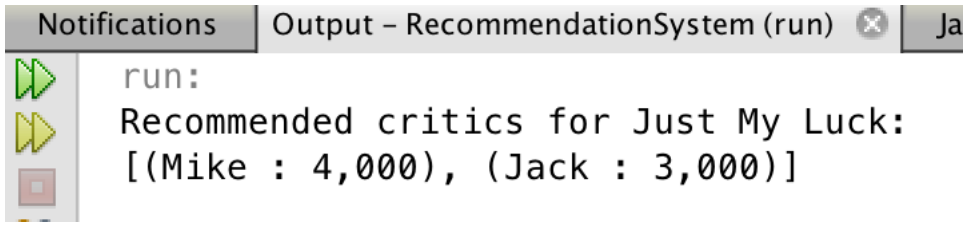
- We can then use the same method as we previously used to find similar users to find matching products:



- Transposing the data set is however a rather slow operation if the data set is large

Finding good critics

- Similarly, we can use the recommendations method previously used to recommend users
- Instead of finding matching users to recommend a movie, we find matching movies to recommend a user
- It can be used to find users that are similar, for example finding users that are likely to give high scores to a movie:



User-based collaborative filtering

- We have to calculate the similarity between a user with every other user and then compare every product each user has rated.
- This approach is called *user-based collaborative filtering*.
- This works for small data sets, but will be very ineffective for large data sets.
- Also, if there are many users there is most likely very little overlap between most users.

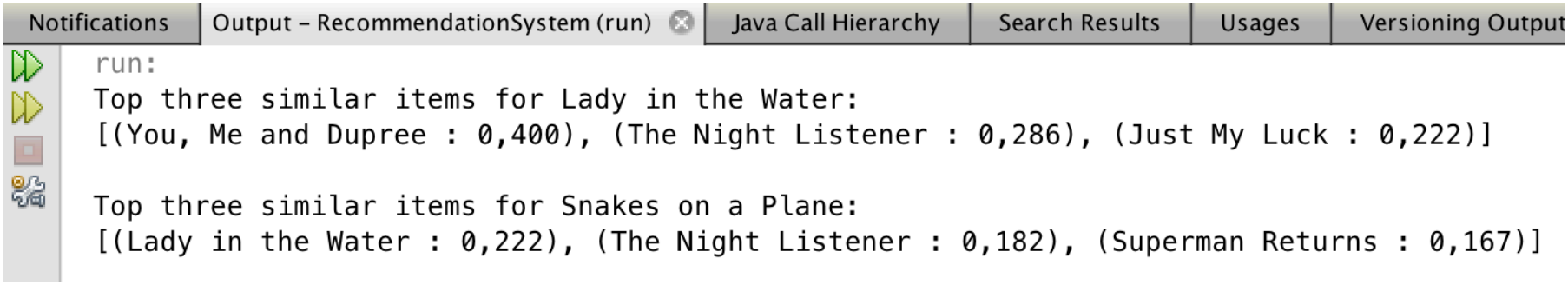
Item-based collaborative filtering

Item-based collaborative filtering

- Another approach is *item-based collaborative filtering*
- It is similar to when we transpose the data set and calculate matching products
- We can however make an assumption that the comparison between products will not change as much as comparisons between users
- We can therefore pre-calculate and store the top N matching products for each product in a new data set
- Finding matching products is then a simple look-up in the pre-generated data set.

Item-based collaborative filtering

- If we do this on our data set the following output is produced (assuming Euclidean distance score):



Recommending movies again

- We can also use the data set pre-generated using the *item-based collaborative filtering* approach to find recommended movies.
- Now, users are not involved at all
- Instead we use the pre-calculated similarity score between movies and generate a similar table:

Recommending movies using IB CF

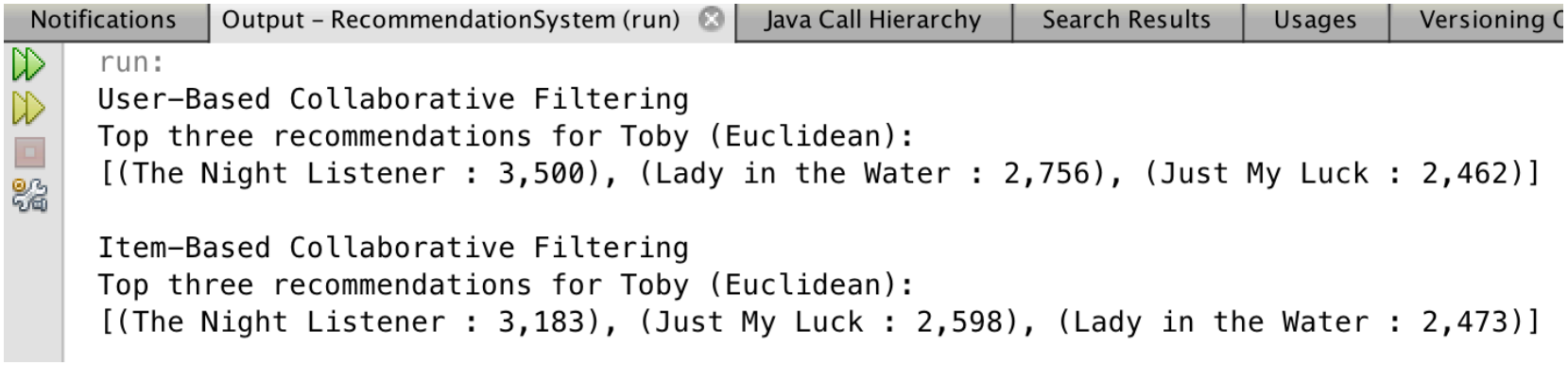
- Calculate recommendations for user Toby:

Movie	Rating	Night	WR Night	Lady	WR Lady	Luck	WR Luck
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 (Total WR / Total)			3.183		2.598		2.473

- The Night Listener is still the recommended movie for Toby.

Comparison

- The results from User-Based and Item-Based Collaborative Filtering differs slightly:



- This is due to how the algorithms work

User-Based or Item-Based?

- Getting a list of recommendations is faster for item-based for large data sets
- The drawback is that the similar items table must be generated and updated regularly, which is a very slow operation
- Item-Based is usually more accurate on sparse data sets, i.e. data sets with little overlap between users
- Our data set (and the data set you shall use in the assignment) is however dense; every user has rated nearly every movie
- User-Based is also simpler to implement

Real-world data set

- The *GroupLens* project at University of Minnesota has collected and generated several data sets for public use
- The data set that is most interesting for us is *MovieLens*:
  - https://grouplens.org/datasets/movielens/
- The data set is generated from the movie recommendation service movielens.org
- The data set comes in different sizes, but the 100 000 ratings data set is more than enough to experiment with
- You can try it with the application you develop for the assignment!