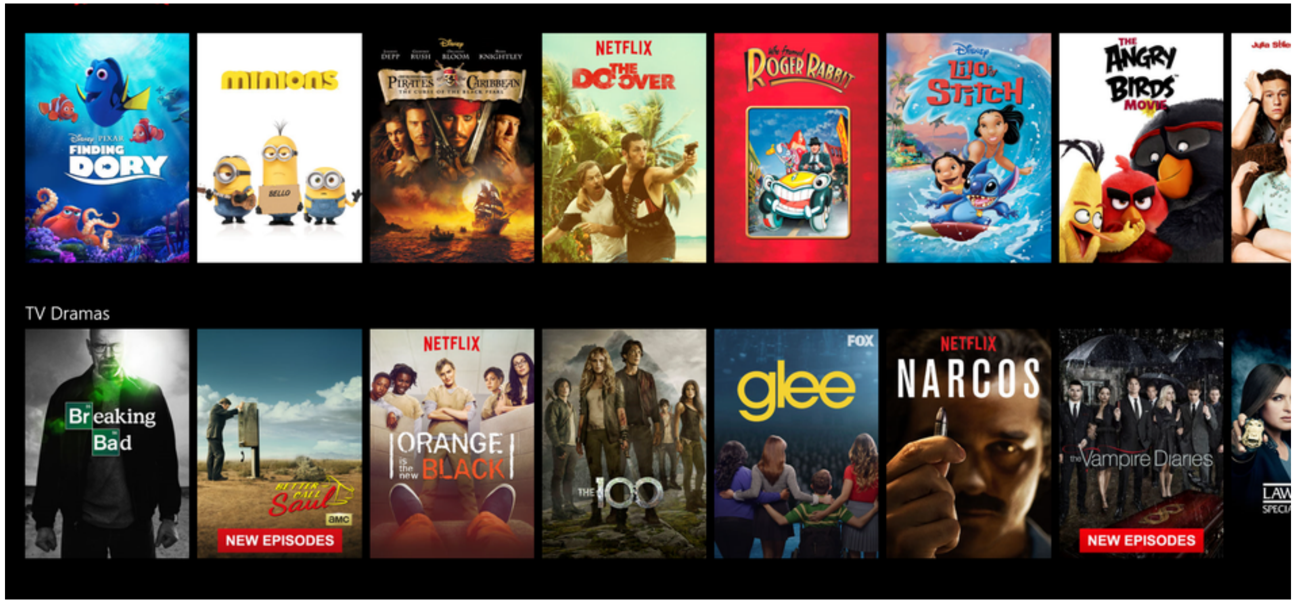
ENGR 102 PROGRAMMING PRACTICE

WEEK 6

İSTANBUL ŞEHİR UNIVERSITY

Collective Intelligence

Introduction



- By using data about which movies each customer enjoyed, Netflix is able to recommend movies to other customers that they may never have even heard of and keep them coming back for more.
- Any way to improve its recommendation system is worth a lot of money to Netflix.

Introduction

- 2006: announcement of a prize of $1 million to the first person to improve the accuracy of its recommendation system by 10 percent.
- Thousands of teams from all over the world.
- The leading team improved 7 percent within 1st year.

Introduction

- Google started in 1998, when there were Netscape, AOL, Yahoo, MSN.
- Google took a completely new approach to ranking search results by using the links on millions of web sites to decide which pages were most relevant.
- Google's search results were so much better than those of the other players that by 2004 it handled 85 percent of searches on the Web.

What do these companies have in common?

- Please elaborate…

What do these companies have in common?

- The ability to collect information
  - Sophisticated algorithms: ***combine data collected from many different sources***.
- The computational power to interpret it
  - enabling great collaboration opportunities and a better understanding of users/customers (**Data Mining**).
  - Everyone wants to understand their customers better in order to create more <u>targeted advertising</u>.

Making Recommendations

Amazon.com

- Amazon tracks the purchasing habits of all its shoppers
    - when you log onto the site, it uses this information to suggest products you might like.
- Amazon can even suggest *movies* you might like, even if you've only bought **books** from it before.

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.

Preferences

critics dict.

- # Dictionary of movie critics and their ratings 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,
                                    ...},
    ...
```

İSTANBUL ŞEHİR UNIVERSITY

Preferences

critics dict.

- # Dictionary of movie critics and their ratings 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,
                                    ...},
    ...
```

Critic

Preferences

critics dict.

- # Dictionary of movie critics and their ratings 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,
                                 ...},
    ...
```

Movie

Preferences

critics dict.

- # Dictionary of movie critics and their ratings 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,
                                    ...},
    ...
```

Rating: 1-5

recommendations.py
critics dict.

- Go to LMS and download recommendations.py (/this week/recommendations.py)
- Create a new Python file in the same directory
- Load the Python module recommendations.py

```
from recommendations import *
```

- and play around with `critics` dictionary (i.e., dataset).

Finding Similar Users

- After collecting data about the things people like, you need a way to determine how similar people are in their tastes.
- Compare each person with every other person and calculate a similarity score.

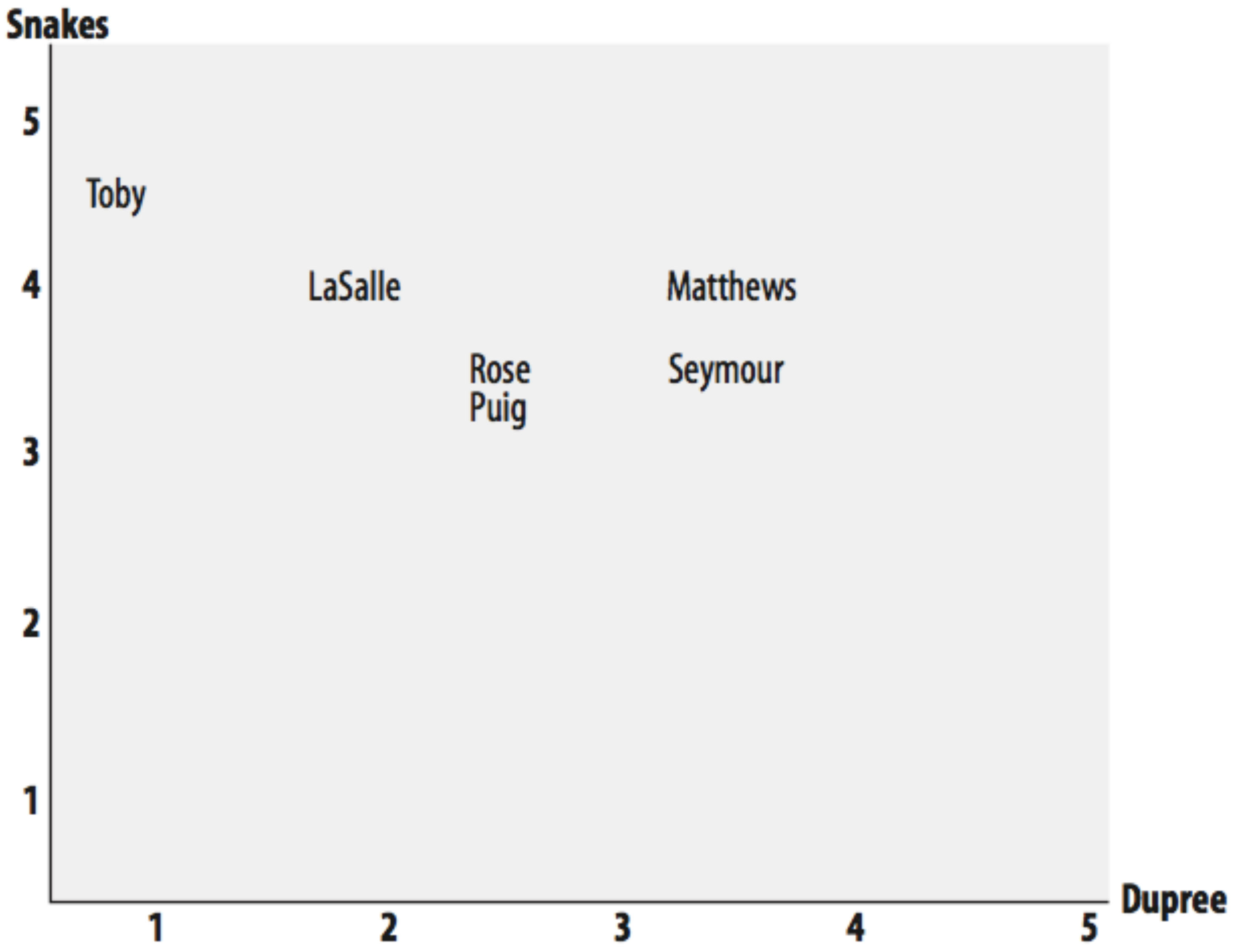
*Different ways of deciding which people are **similar***

=

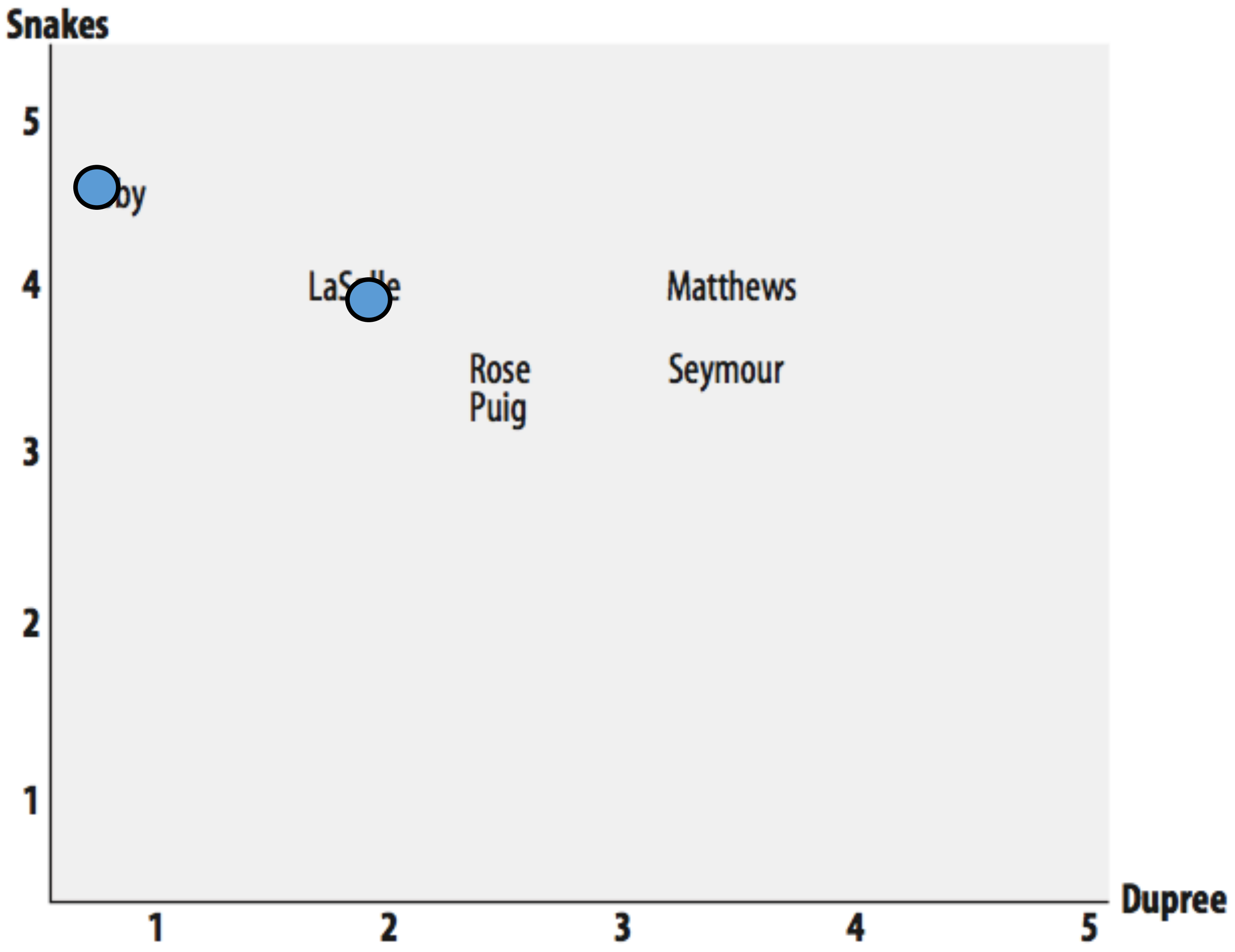
Different Algorithms

- Similarity scores:
  - Euclidean distance,
  - Pearson correlation,
  - etc.

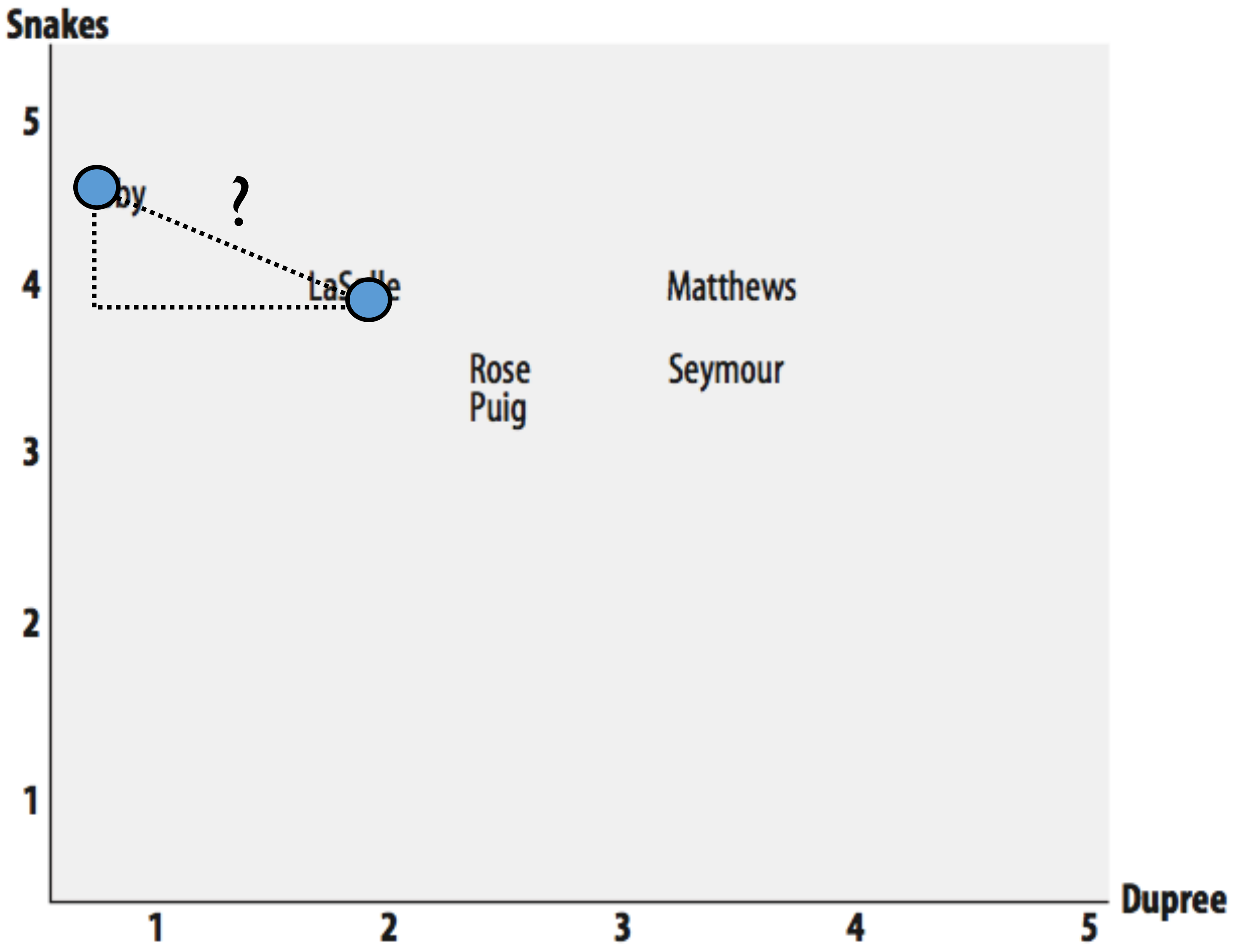
Euclidean Distance



Euclidean Distance



Euclidean Distance



Python Tip:
List Comprehension

```
[expression for variable in list]
or
[expression for variable in list if condition]

list1 = [1, 2, 3, 4, 5, 6]
list2 = [v*10 for v in list1]
print list2
???
list3 = [v*10 for v in list1 if v > 3]
print list3
???
```

Euclidean Example

- person1 = {a:3, b:3, c:5}
- person2 = {a:6, b:7, d:8}

- first find the set of common items
  - {a, b}
- calculate the Euclidean distance between person ratings
  - $d = [(6-3)^2 + (7-3)^2]\hat{}(0.5) = 5$
- normalize so that distance is between 0 and 1
  - $\frac{1}{1+d} = \frac{1}{6}$

Example

- Write a function that inputs critics dictionary and two critic names and returns the Euclidean distance between them.

Euclidean Distance

sim_distance(...)

```
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 si])

    return 1/(1+sqrt(sum_of_squares))
```

Euclidean Distance
`sim_distance(...)`

- load the Python module recommendations.py

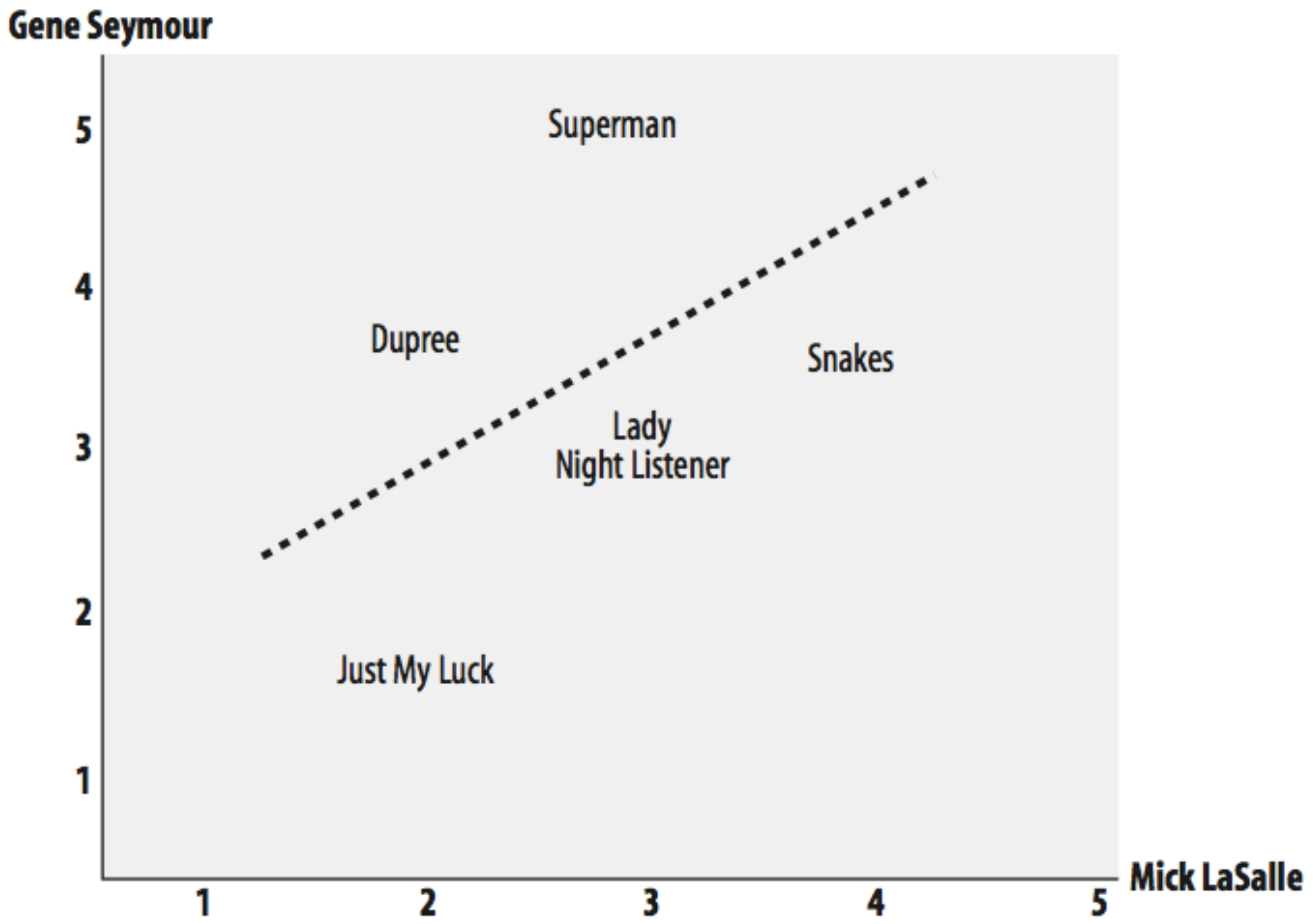
```
from recommendations import *
```

- Play around with sim_distance function, which uses Euclidean distance as a similarity measure.

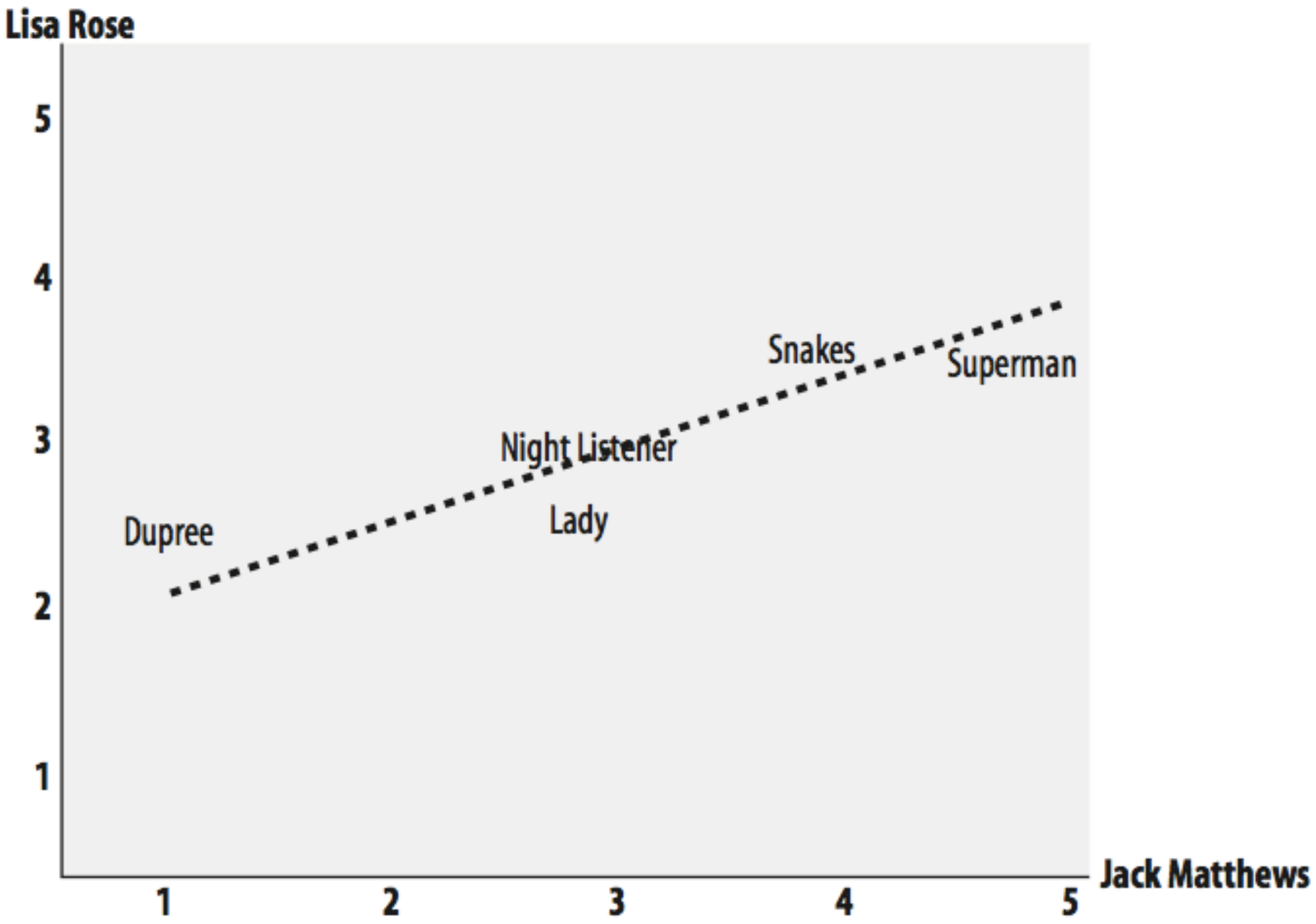
Pearson Correlation Score

- 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
- It tends to give better results in situations where the data isn't well normalized
    - for example, if some critics' movie rankings are routinely more harsh than average.

OK correlation!



Good correlation!



Pearson Correlation

`sim_pearson(...)`

- load the Python module recommendations.py

```
from recommendations import *
```

- Play around with sim_pearson function, which uses Correlation coefficient as a similarity measure.

Ranking the critics

“Which movie critics have tastes similar to mine so that I know whose advice I should take when deciding on a movie!”

- Create a function that ***scores everyone against a given person*** and finds the closest matches.

Example

- Write a function that
  - inputs
    - critics dictionary
    - a critic name *person*
    - integer *k*
    - similarity function reference
  - returns the top *k* critics closest to *person*.

Ranking the critics

topMatches(...)

```
# Returns the best matches for person from the prefs dict.
def topMatches(prefs, person, k=5, similarity=sim_pearson):

    scores = [(similarity(prefs,person,other),other) \
                for other in prefs if other != person]

    scores.sort()

    scores.reverse()

    return scores[0:k]
```

Ranking the critics
topMatches(...)

- load the Python module recommendations.py

```
from recommendations import *
```

- Play around with topMatches function.

Recommending Items

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

Recommending Items

getRecommendations(...)

```python
def getRecommendations(prefs, person, similarity = sim_pearson):
    totals = {}
    simSums = {}
    for other in prefs:
        if other == person: continue
        sim = similarity(prefs, person, other)
        if sim <= 0: continue
        for item in prefs[other]:
            if item not in prefs[person]:
                totals.setdefault(item, 0)
                totals[item] += prefs[other][item]*sim
                simSums.setdefault(item, 0)
                simSums[item] += sim
    rankings = [(total/simSums[item],item)
                              for item, total in totals.items()]
    rankings.sort(reverse = True)
    return rankings
```

Recommending Items

getRecommendations(...)

- Play around with getRecommendations function.

```
from recommendations import *
print getRecommendations(critics, ‘Ali’)
```