Web Mining and Recommender Systems

Recommender Systems: Introduction

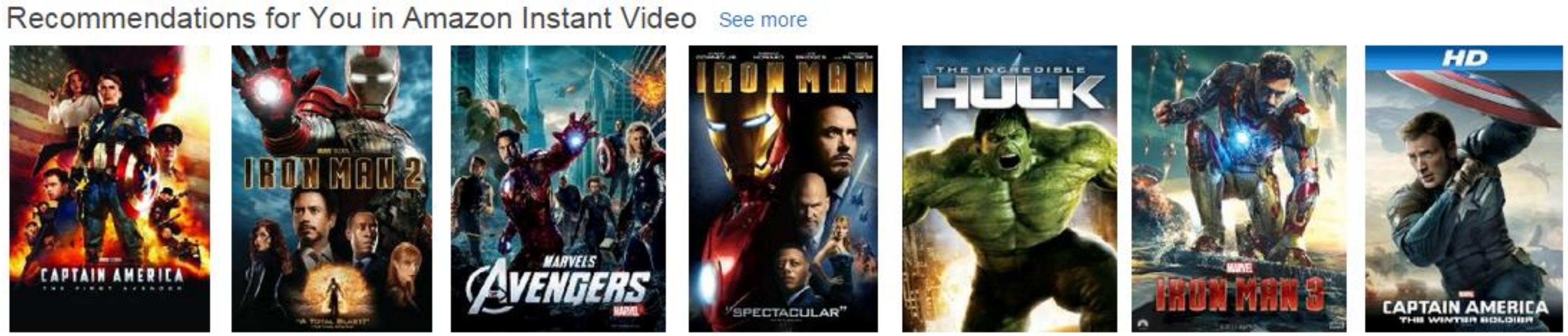
Learning Goals

- Introduced the topic of **recommender systems** and explain how they relate to supervised and unsupervised learning

Why recommendation?

The goal of recommender systems is…

- To help people discover new content

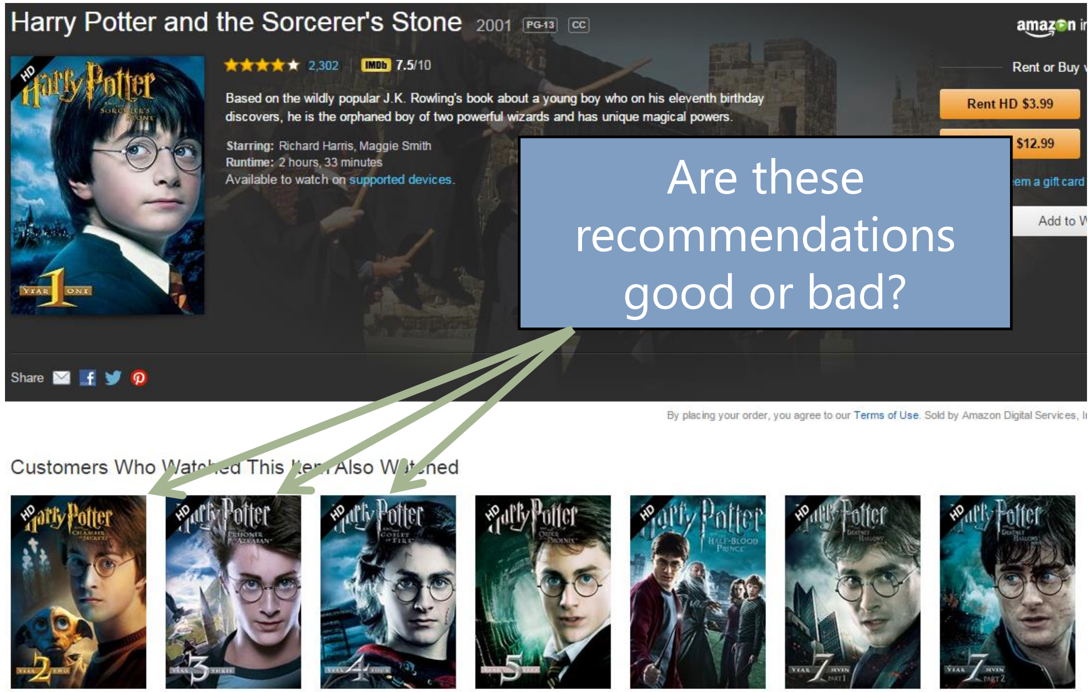


Recommendations for You in Amazon Instant Video See more

Why recommendation?

The goal of recommender systems is…

- To help us find the content we were already looking for



Harry Potter and the Sorcerer's Stone 2001 PG-13 CC

★★★★★ 2,302 IMDb 7.5/10

Based on the wildly popular J.K. Rowling's book about a young boy who on his eleventh birthday discovers, he is the orphaned boy of two powerful wizards and has unique magical powers.

Starring: Richard Harris, Maggie Smith
Runtime: 2 hours, 33 minutes
Available to watch on supported devices.

By placing your order, you agree to our Terms of Use. Sold by Amazon Digital Services, I

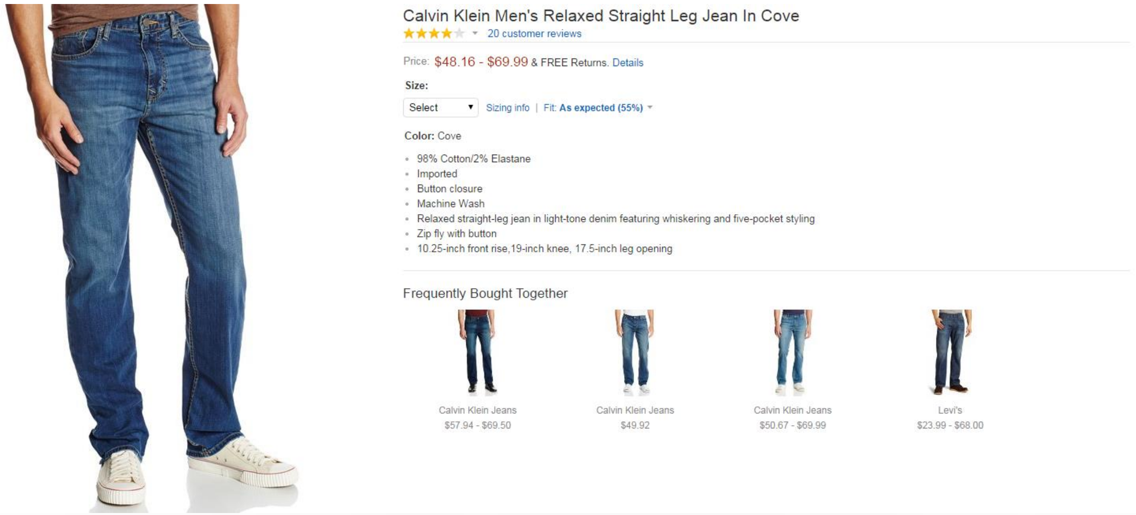
Customers Who Watched This Item Also Watched

Are these recommendations good or bad?

Why recommendation?

The goal of recommender systems is…

- To discover which things go together



Calvin Klein Men's Relaxed Straight Leg Jean In Cove

★★★★☆ 20 customer reviews

Price: $48.16 - $69.99 & FREE Returns. Details

Size:

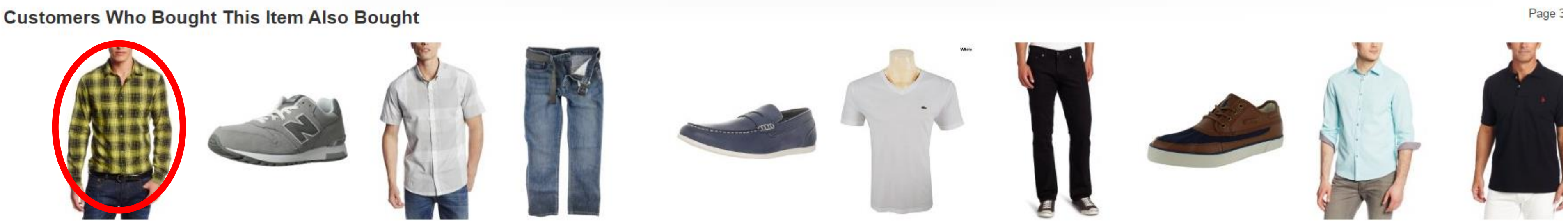
Select ▾ Sizing info | Fit: **As expected (55%)**

Color: Cove

- 98% Cotton/2% Elastane
- Imported
- Button closure
- Machine Wash
- Relaxed straight-leg jean in light-tone denim featuring whiskering and five-pocket styling
- Zip fly with button
- 10.25-inch front rise,19-inch knee, 17.5-inch leg opening

Frequently Bought Together

Calvin Klein Jeans	Calvin Klein Jeans	Calvin Klein Jeans	Levi's
$57.94 - $69.50	$49.92	$50.67 - $69.99	$23.99 - $68.00



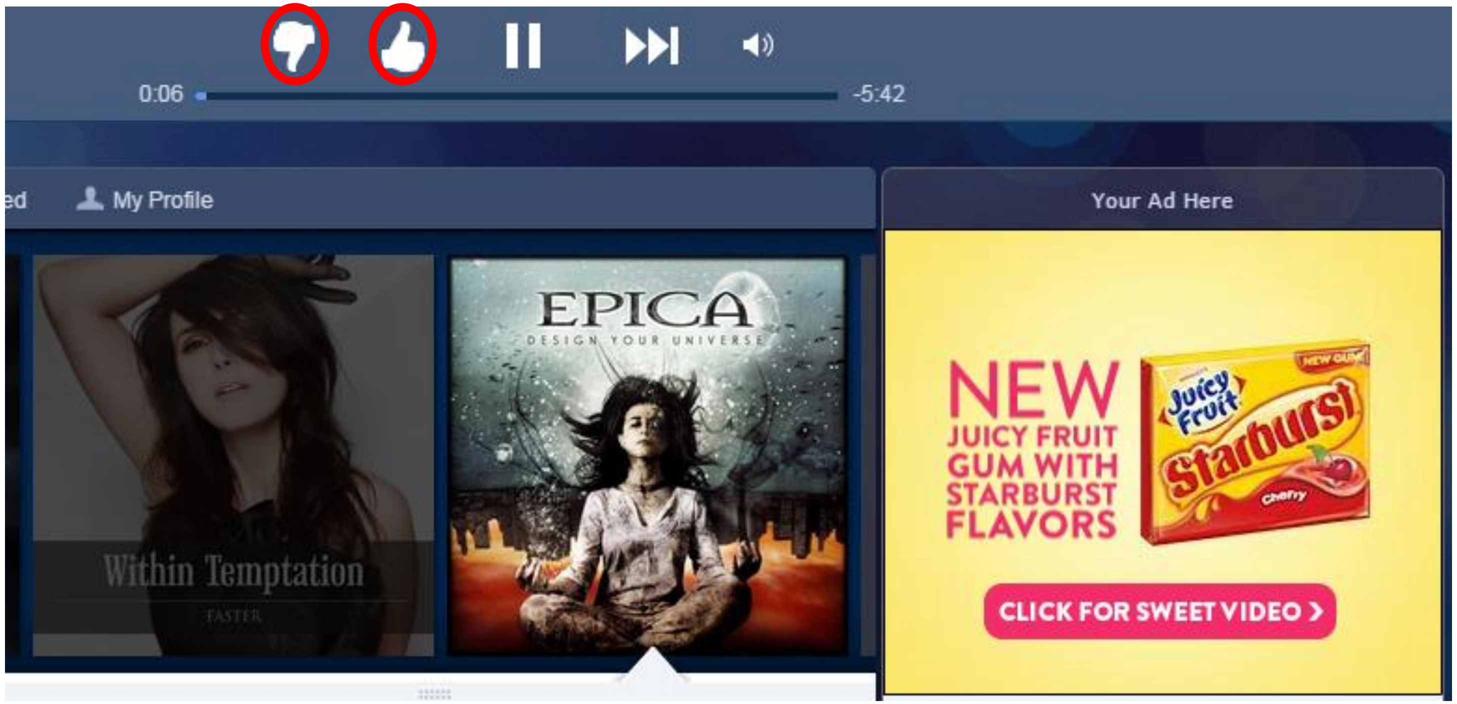
Customers Who Bought This Item Also Bought

Page 3

Why recommendation?

The goal of recommender systems is…

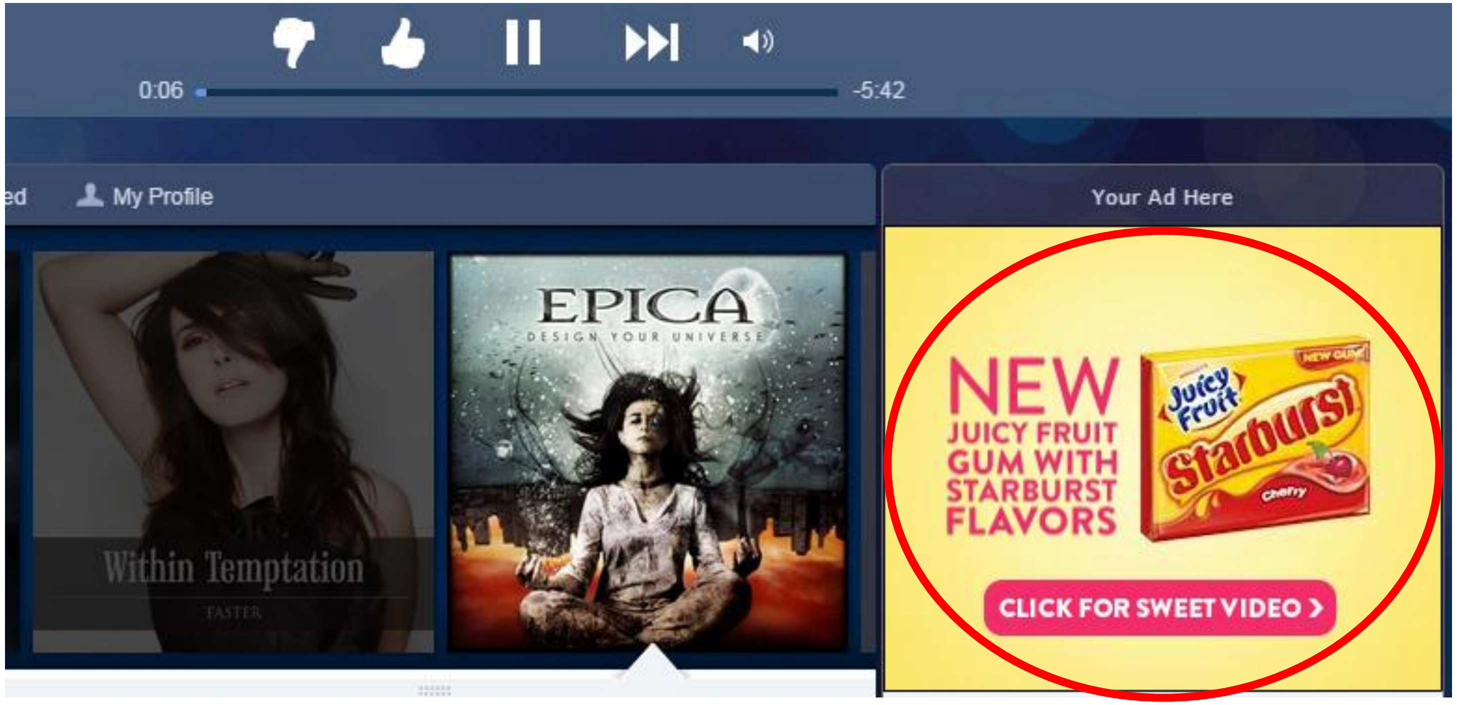
- To personalize user experiences in response to user feedback



Why recommendation?

The goal of recommender systems is…

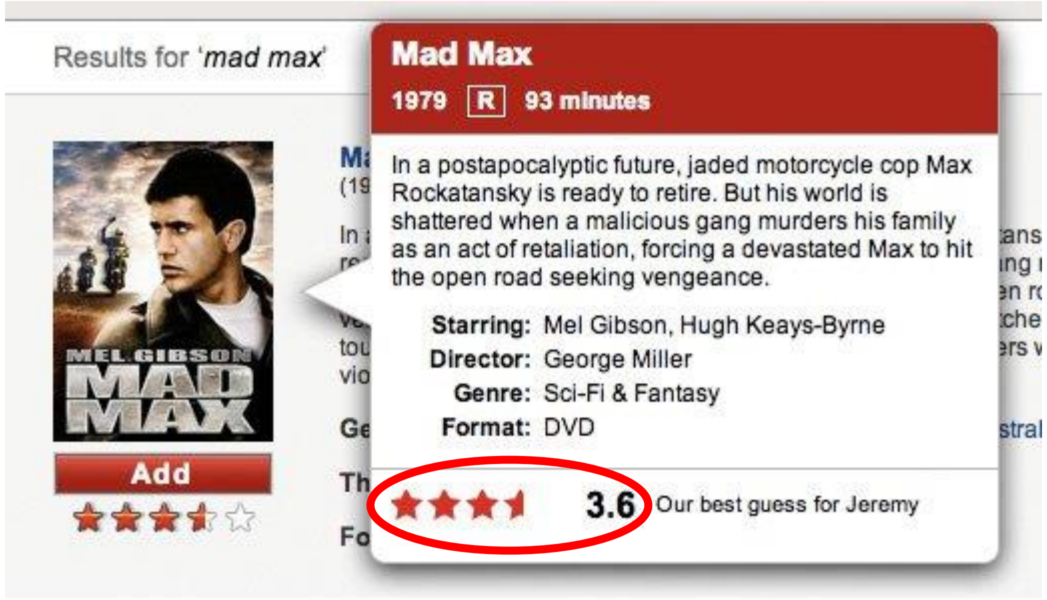
- To recommend incredible products that are relevant to our interests



Why recommendation?

The goal of recommender systems is…

- To identify things that we **like**



Results for '*mad max*'

Mad Max

1979 R 93 minutes

In a postapocalyptic future, jaded motorcycle cop Max Rockatansky is ready to retire. But his world is shattered when a malicious gang murders his family as an act of retaliation, forcing a devastated Max to hit the open road seeking vengeance.

Starring: Mel Gibson, Hugh Keays-Byrne
Director: George Miller
Genre: Sci-Fi & Fantasy
Format: DVD

3.6 Our best guess for Jeremy

Why recommendation?

The goal of recommender systems is…

- To help people discover new content
- To help us find the content we were
- To dis ogether
- To p nces in response to user feedback
- To identify things that we **like**

To **model** people’s preferences, opinions, and behavior

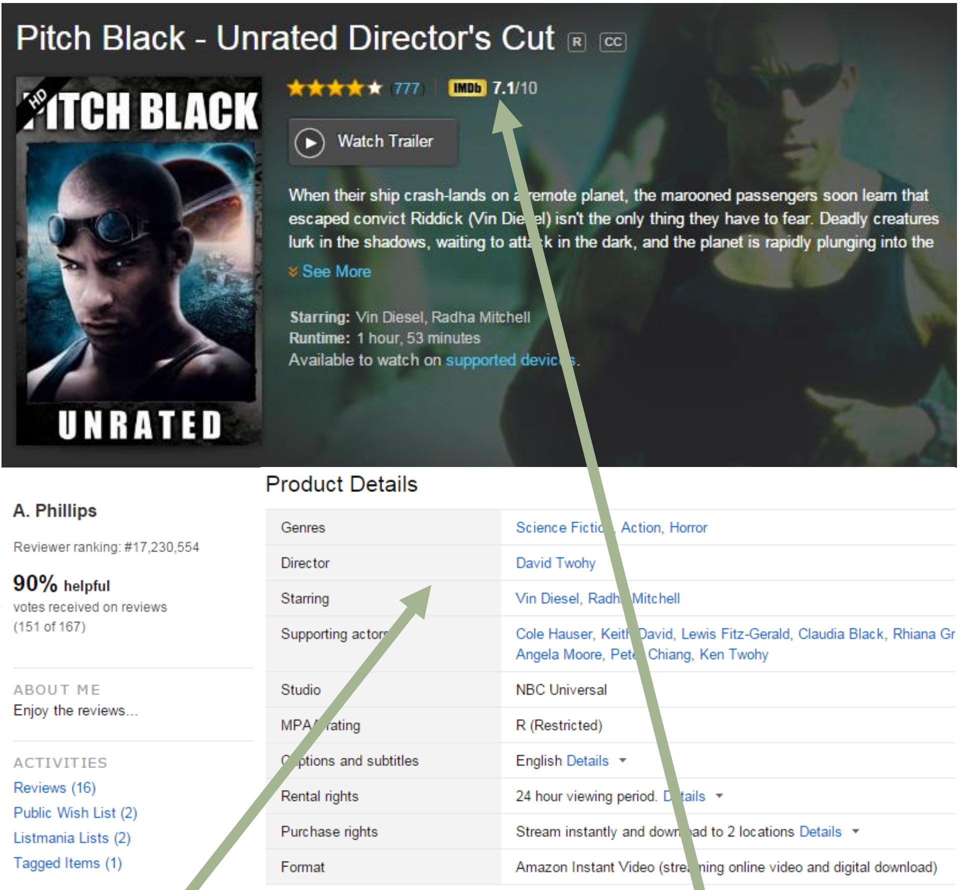
Recommending things to people

Suppose we want to build a movie recommender

e.g. which of these films will I rate highest?

Recommending things to people

We already have a few tools in our "supervised learning" toolbox that may help us



Pitch Black - Unrated Director's Cut R CC

★★★★★ 777 IMDb 7.1/10

Watch Trailer

When their ship crash-lands on a remote planet, the marooned passengers soon learn that escaped convict Riddick (Vin Diesel) isn't the only thing they have to fear. Deadly creatures lurk in the shadows, waiting to attack in the dark, and the planet is rapidly plunging into the

See More

Starring: Vin Diesel, Radha Mitchell
Runtime: 1 hour, 53 minutes
Available to watch on supported devices.

A. Phillips

Reviewer ranking: #17,230,554

90% **helpful**
votes received on reviews
(151 of 167)

ABOUT ME
Enjoy the reviews...

ACTIVITIES
Reviews (16)
Public Wish List (2)
Listmania Lists (2)
Tagged Items (1)

Product Details

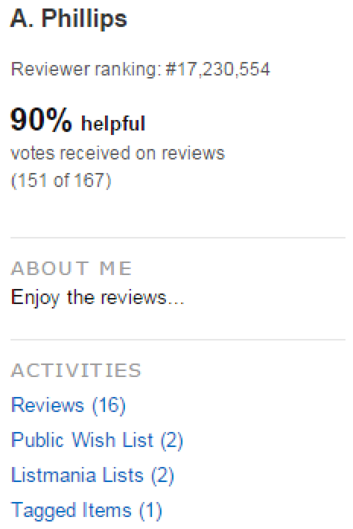
Genres	Science Fiction, Action, Horror
Director	David Twohy
Starring	Vin Diesel, Radha Mitchell
Supporting actors	Cole Hauser, Keith David, Lewis Fitz-Gerald, Claudia Black, Rhiana Gr Angela Moore, Peter Chiang, Ken Twohy
Studio	NBC Universal
MPAA rating	R (Restricted)
Captions and subtitles	English Details
Rental rights	24 hour viewing period. Details
Purchase rights	Stream instantly and download to 2 locations Details
Format	Amazon Instant Video (streaming online video and digital download)

$$\boxed{f(\text{user features}, \text{movie features}) \overset{?}{\longrightarrow} \text{star rating}}$$

Recommending things to people

$$f(\text{user features}, \text{movie features}) \overset{?}{\rightarrow} \text{star rating}$$

User features: age, gender, location, etc.



A. Phillips

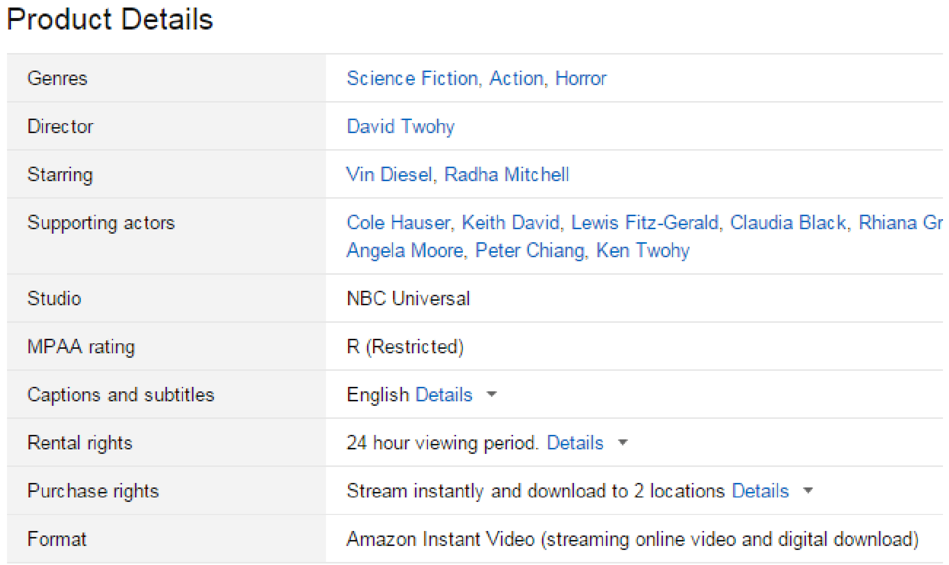
Reviewer ranking: #17,230,554

90% helpful
votes received on reviews
(151 of 167)

ABOUT ME
Enjoy the reviews...

ACTIVITIES
- Reviews (16)
- Public Wish List (2)
- Listmania Lists (2)
- Tagged Items (1)

Movie features: genre, actors, rating, length, etc.



Product Details

Genres	Science Fiction, Action, Horror
Director	David Twohy
Starring	Vin Diesel, Radha Mitchell
Supporting actors	Cole Hauser, Keith David, Lewis Fitz-Gerald, Claudia Black, Rhiana Gr Angela Moore, Peter Chiang, Ken Twohy
Studio	NBC Universal
MPAA rating	R (Restricted)
Captions and subtitles	English Details ▾
Rental rights	24 hour viewing period. Details ▾
Purchase rights	Stream instantly and download to 2 locations Details ▾
Format	Amazon Instant Video (streaming online video and digital download)

Recommending things to people

$$f(\text{user features}, \text{movie features}) \overset{?}{\rightarrow} \text{star rating}$$

With the models we've seen so far, we can build predictors that account for…

- Do women give higher ratings than men?
- Do Americans give higher ratings than Australians?
- Do people give higher ratings to action movies?
- Are ratings higher in the summer or winter?
- Do people give high ratings to movies with Vin Diesel?

So what **can't** we do yet?

Recommending things to people

$$f(\text{user features}, \text{movie features}) \overset{?}{\rightarrow} \text{star rating}$$

Consider the following linear predictor (e.g. from week 1):

$$f(\text{user features}, \text{movie features}) = \langle \phi(\text{user features}); \phi(\text{movie features}), \theta \rangle$$

$$= \langle \phi(\text{user}), \theta^{(\text{user})} \rangle + \langle \phi^{(\text{movie})}, \theta^{(\text{movie})} \rangle$$

Recommending things to people

But this is essentially just two separate predictors!

$$f(\text{user features}, \text{movie features}) =$$

$$= \underbrace{\langle \phi(\text{user features}), \theta_{\text{user}} \rangle}_{\text{user predictor}} + \underbrace{\langle \phi(\text{movie features}), \theta_{\text{movie}} \rangle}_{\text{movie predictor}}$$

That is, we're treating user and movie features as though they're **independent!**

Recommending things to people

But these predictors should (obviously?) **not** be independent

$$f(\text{user features}, \text{movie features}) = f(\text{user}) + f(\text{movie})$$

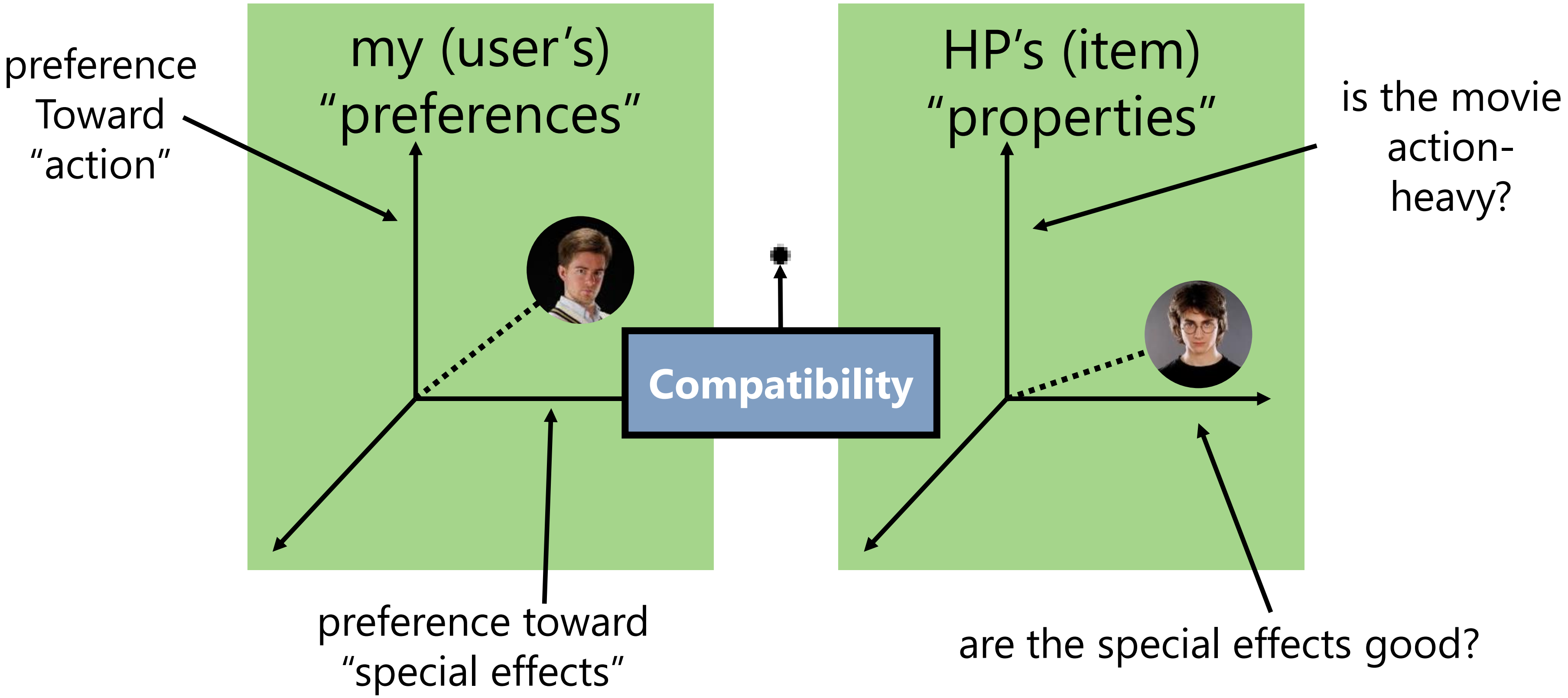
do I tend to give high ratings?

does the population tend to give high ratings to this genre of movie?

But what about a feature like “do **I** give high ratings to **this genre** of movie”?

Recommending things to people

Recommender Systems go beyond the methods we've seen so far by trying to model the **relationships** between people and the items they're evaluating



This section

Recommender Systems

1. (next) Collaborative filtering

(performs recommendation in terms of user/user and item/item similarity)

2. (later) Latent-factor models

(performs recommendation by projecting users and items into some low-dimensional space)

3. (later) The Netflix Prize

Web Mining and Recommender Systems

Similarity-based Recommender Systems

Learning Goals

- Introduced some simple recommendation strategies based on the notions of user or item similarity

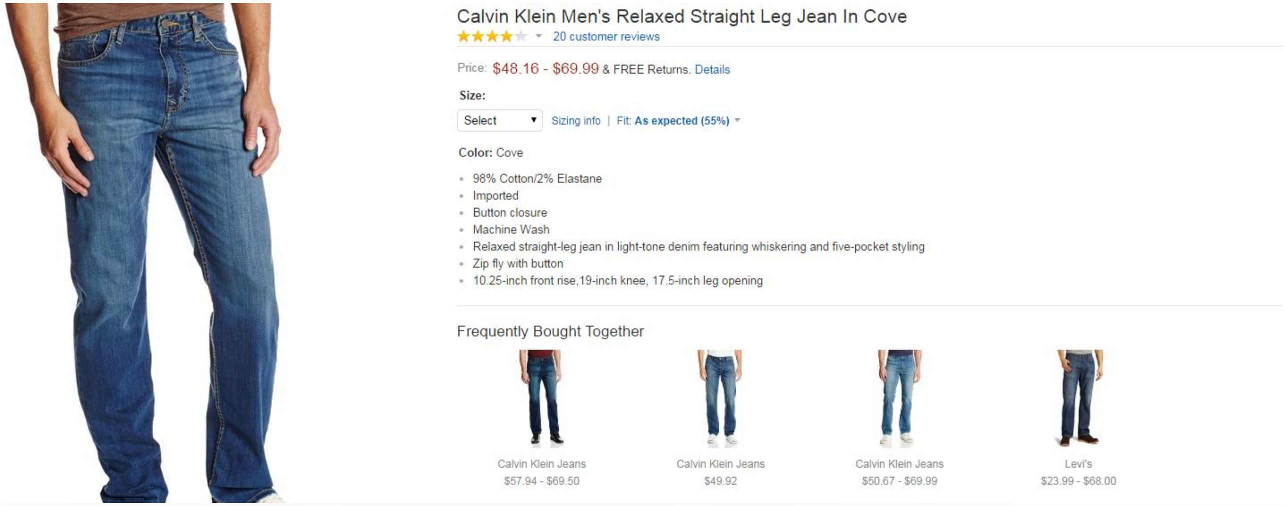
Defining similarity between users & items

Q: How can we measure the **similarity** between two **users?**
A: In terms of the **items** they purchased!

Q: How can we measure the similarity between two **items?**
A: In terms of the users who purchased them!

Defining similarity between users & items

e.g.:
Amazon



Calvin Klein Men's Relaxed Straight Leg Jean In Cove
★★★★☆ 20 customer reviews

Price: $48.16 - $69.99 & FREE Returns. Details

Size:

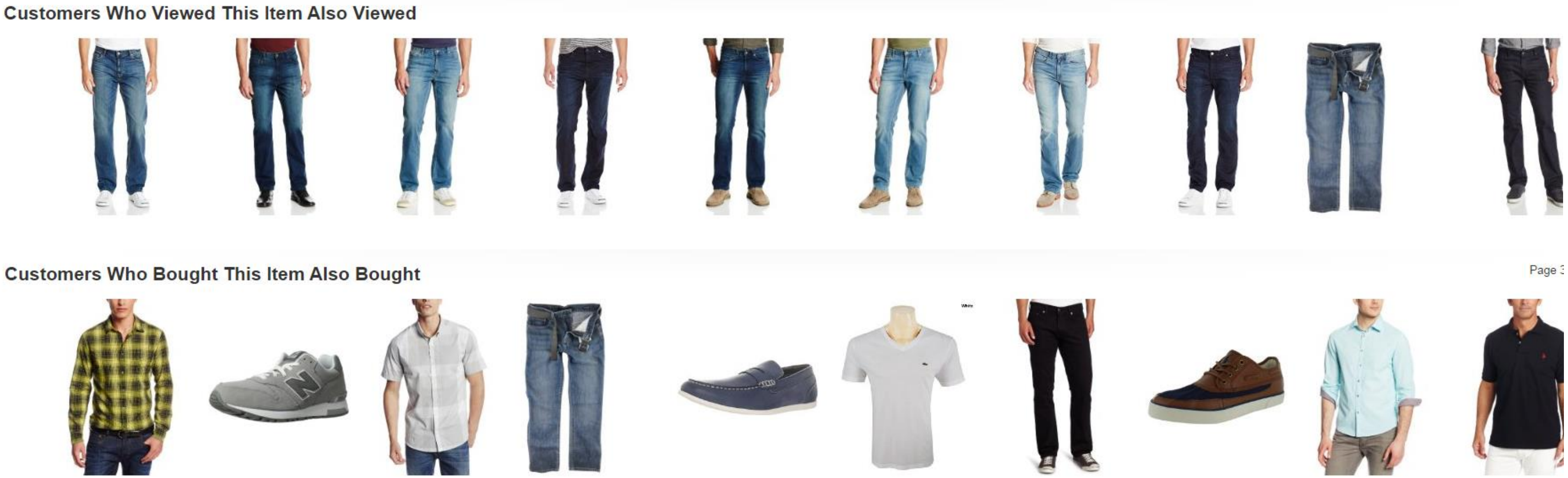
Select ▾ Sizing info | Fit: As expected (55%)

Color: Cove

- 98% Cotton/2% Elastane
- Imported
- Button closure
- Machine Wash
- Relaxed straight-leg jean in light-tone denim featuring whiskering and five-pocket styling
- Zip fly with button
- 10.25-inch front rise,19-inch knee, 17.5-inch leg opening

Frequently Bought Together

Calvin Klein Jeans	Calvin Klein Jeans	Calvin Klein Jeans	Levi's
$57.94 - $69.50	$49.92	$50.67 - $69.99	$23.99 - $68.00



Customers Who Viewed This Item Also Viewed

Customers Who Bought This Item Also Bought

Page 3

Definitions

Definitions

I_u = set of items purchased by user *u*

U_i = set of users who purchased item *i*

Definitions

Or equivalently…

$$R = \left.\overbrace{\begin{pmatrix} 1 & 0 & \cdots & 1 \\ 0 & 0 & & 1 \\ \vdots & & \ddots & \vdots \\ 1 & 0 & \cdots & 1 \end{pmatrix}}^{\text{items}}\right\}\text{users}$$

R_u = binary representation of items purchased by *u*

$R_{\cdot,i}$ = binary representation of users who purchased *i*

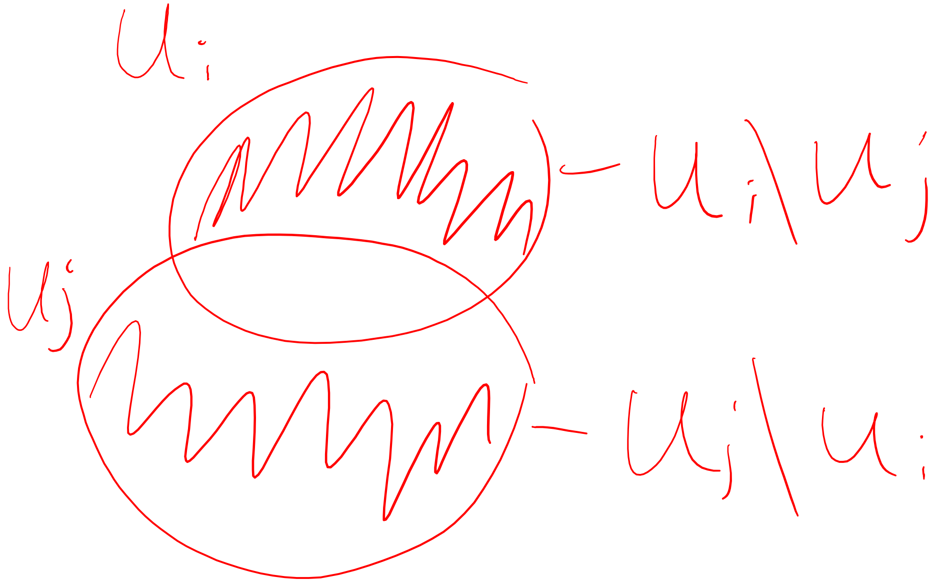
$$I_u = \left\{ i \,\middle|\, R_{ui} = 1 \right\} \qquad U_i = \left\{ u \,\middle|\, R_{ui} = 1 \right\}$$

0. Euclidean distance

Euclidean distance:

e.g. between two items i,j (similarly defined between two users)

$$|U_i \setminus U_j| + |U_j \setminus U_i| = \|R_i - R_j\|$$



0. Euclidean distance

Euclidean distance:

e.g.: U_1 = {1,4,8,9,11,23,25,34}
U_2 = {1,4,6,8,9,11,23,25,34,35,38}
U_3 = {4}
U_4 = {5}

$$|U_1 \setminus U_2| + |U_2 \setminus U_1| = 3$$

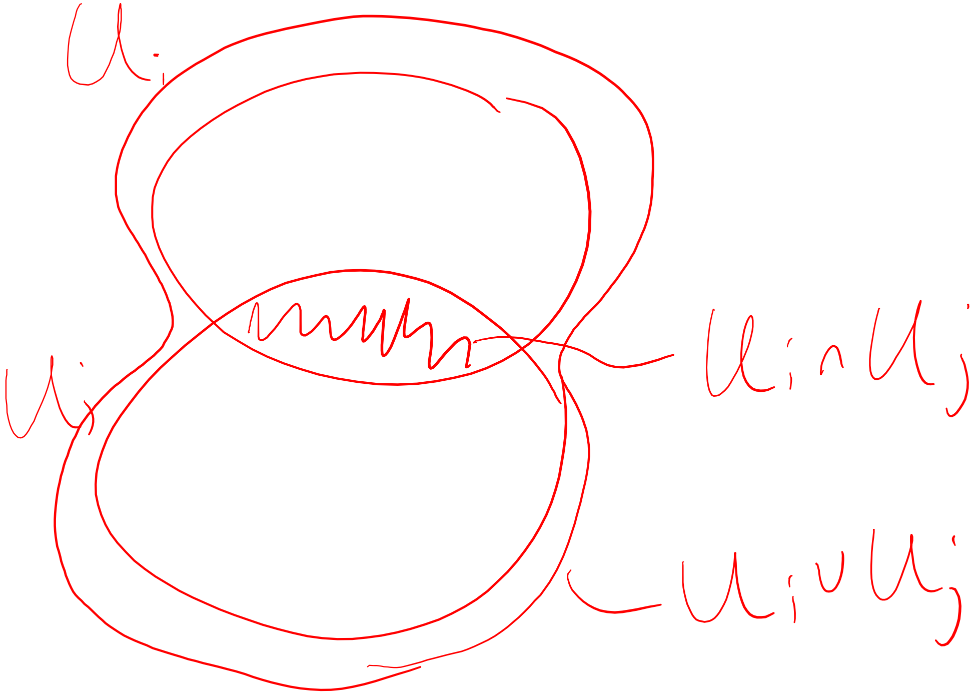
$$|U_3 \setminus U_4| + |U_3 \setminus U_4| = 2$$

Problem: favors small sets, even if they have few elements in common

1. Jaccard similarity

$$\text{Jaccard}(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

$$\text{Jaccard}(U_i, U_j) = \frac{|U_i \cap U_j|}{|U_i \cup U_j|}$$



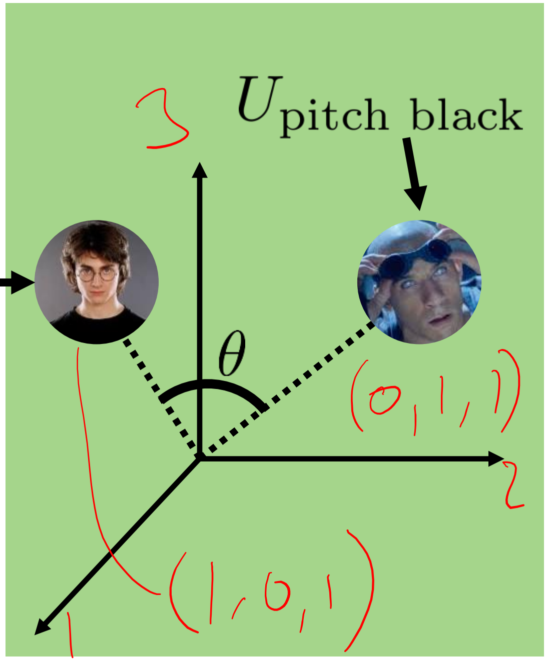
→ Maximum of 1 if the two users purchased **exactly the same** set of items
(or if two items were purchased by the same set of users)

→ Minimum of 0 if the two users purchased **completely disjoint** sets of items
(or if the two items were purchased by completely disjoint sets of users)

2. Cosine similarity

$U_{\text{harry potter}}$

(vector representation of users who purchased harry potter)



$$\cos(\theta) = 1$$

(theta = 0) → *A* and *B* point in exactly the same direction

$$\cos(\theta) = -1$$

(theta = 180) → *A* and *B* point in opposite directions (won't actually happen for 0/1 vectors)

$$\cos(\theta) = 0$$

(theta = 90) → *A* and *B* are orthogonal

$$\theta = \cos^{-1}\left(\frac{A \cdot B}{\|A\| \|B\|}\right)$$

$$\cos(\theta) = \frac{u_i \cdot u_j}{\|u_i\| \|u_j\|} \overset{\text{binary interactions}}{\rightsquigarrow} \frac{|u_i \cap u_j|}{\sqrt{|u_i| |u_j|}}$$

2. Cosine similarity

Why cosine?

- Unlike Jaccard, works for arbitrary vectors
- E.g. what if we have **opinions** in addition to purchases?

$$R = \begin{pmatrix} 1 & 0 & \cdots & 1 \\ 0 & 0 & & 1 \\ \vdots & & \ddots & \vdots \\ 1 & 0 & \cdots & 1 \end{pmatrix} \rightarrow \begin{pmatrix} -1 & 0 & \cdots & 1 \\ 0 & 0 & & -1 \\ \vdots & & \ddots & \vdots \\ 1 & 0 & \cdots & -1 \end{pmatrix}$$

bought and **liked**

didn’t buy

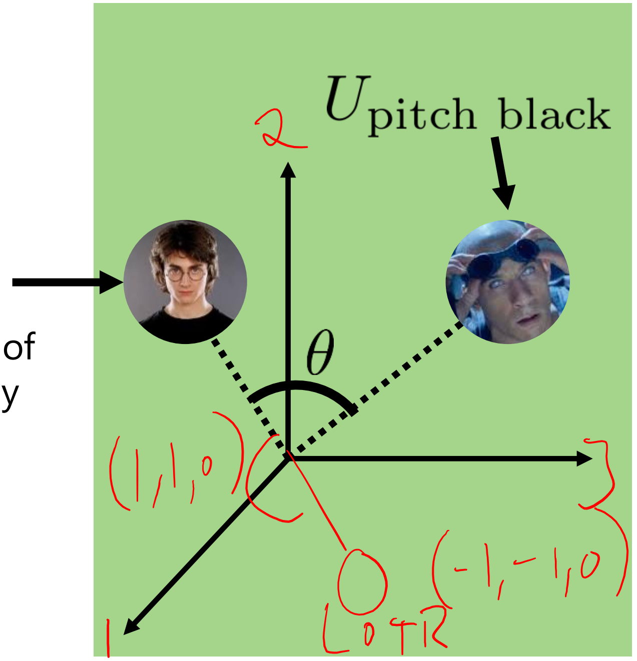
bought and **hated**

2. Cosine similarity

E.g. our previous example, now with "thumbs-up/thumbs-down" ratings

$U_{\text{harry potter}}$

(vector representation of users' **ratings of** Harry Potter)



$$\cos(\theta) = 1$$

(theta = 0) → Rated by the same users, and they all agree

$$\cos(\theta) = -1$$

(theta = 180) → Rated by the same users, but they **completely disagree** about it

$$\cos(\theta) = 0$$

(theta = 90) → Rated by different sets of users

4. Pearson correlation

What if we have numerical ratings (rather than just thumbs-up/down)?

$$R = \begin{pmatrix} -1 & 0 & \cdots & 1 \\ 0 & 0 & & -1 \\ \vdots & & \ddots & \vdots \\ 1 & 0 & \cdots & -1 \end{pmatrix} \rightarrow \begin{pmatrix} 4 & 0 & \cdots & 2 \\ 0 & 0 & & 3 \\ \vdots & & \ddots & \vdots \\ 5 & 0 & \cdots & 1 \end{pmatrix}$$

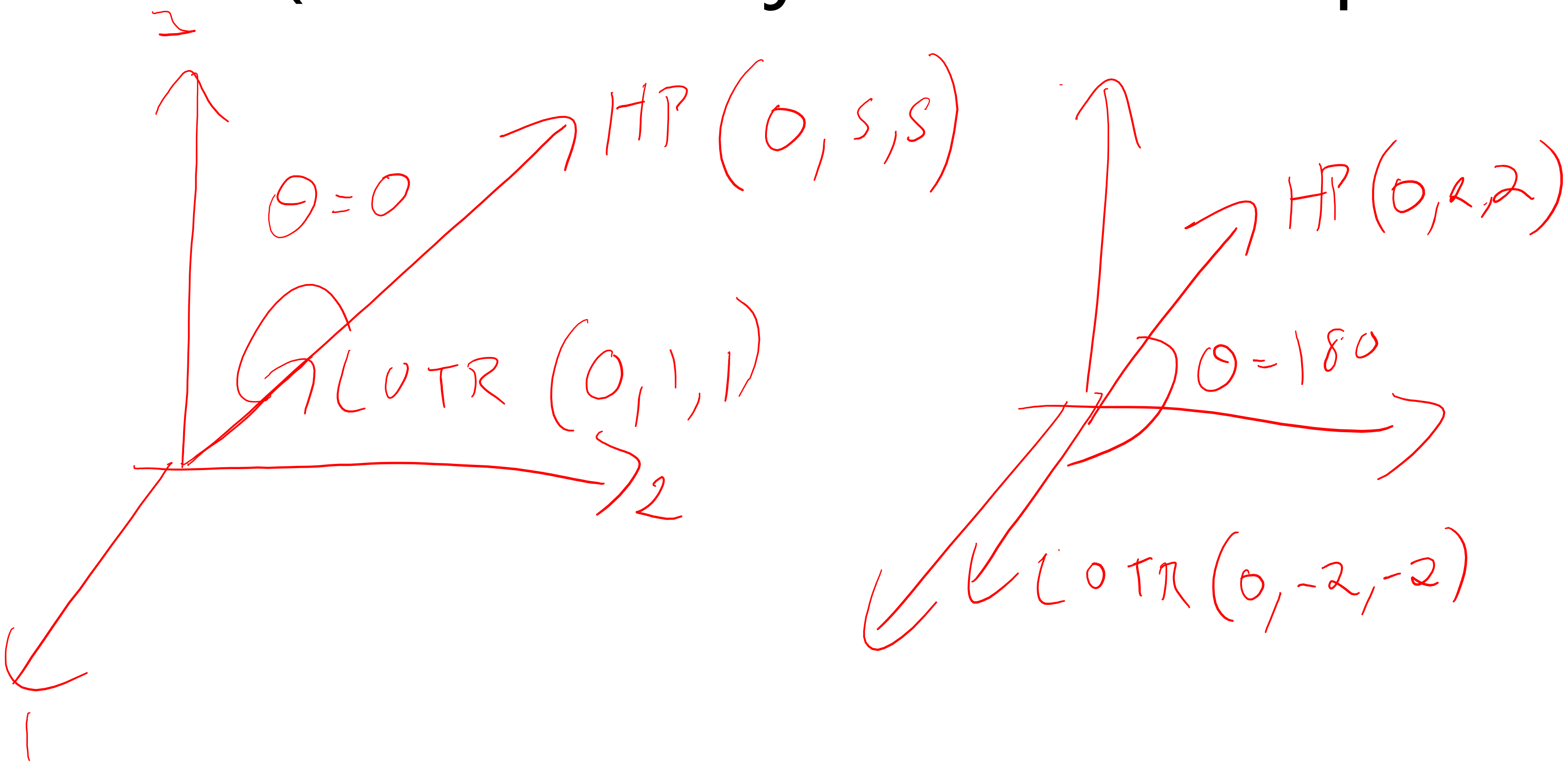
bought and **liked**

didn’t buy

bought and **hated**

4. Pearson correlation

What if we have numerical ratings (rather than just thumbs-up/down)?



4. Pearson correlation

What if we have numerical ratings (rather than just thumbs-up/down)?

- We wouldn’t want 1-star ratings to be parallel to 5-star ratings
- So we can subtract the average – values are then **negative** for below-average ratings and **positive** for above-average ratings

items rated by both users   average rating by user *v*

$$\mathrm{Sim}(u,v) = \frac{\sum_{i \in I_u \cap I_v} (R_{u,i} - \bar{R}_u)(R_{v,i} - \bar{R}_v)}{\sqrt{\sum_{i \in I_u \cap I_v} (R_{u,i} - \bar{R}_u)^2 \sum_{i \in I_u \cap I_v} (R_{v,i} - \bar{R}_v)^2}}$$

4. Pearson correlation

Compare to the cosine similarity:

Pearson similarity (between users):

items rated by both users   average rating by user *v*

$$\text{Sim}(u,v) = \frac{\sum_{i \in I_u \cap I_v} (R_{u,i} - \bar{R}_u)(R_{v,i} - \bar{R}_v)}{\sqrt{\sum_{i \in I_u \cap I_v} (R_{u,i} - \bar{R}_u)^2 \sum_{i \in I_u \cap I_v} (R_{v,i} - \bar{R}_v)^2}}$$

Cosine similarity (between users):

$$\text{Sim}(u,v) = \frac{\sum_{i \in I_u \cap I_v} R_{u,i} R_{v,i}}{\sqrt{\sum_{i \in I_u \cap I_v} R_{u,i}^2 \sum_{i \in I_u \cap I_v} R_{v,i}^2}}$$

Note: slightly different from previous definition. Here similarity is determined only based on items *both* users have consumed

4. Pearson correlation

$$\mathrm{Sim}(u,v) = \frac{\sum_{i \in I_u \cap I_v} R_{u,i} R_{v,i}}{\sqrt{\sum_{i \in I_u \cap I_v} R_{u,i}^2 \sum_{i \in I_u \cap I_v} R_{v,i}^2}}$$

$$\mathrm{Cosine}(A,B) = \frac{A \cdot B}{\|A\| \|B\|}$$

Consider **all items** in the denominator, or just shared items?

Just shared: two users should be considered maximally similar if they've rated shared items the same way. If only one user has rated an item, we have no evidence that the other user is different.
All: Two users who've rated items the same way *and only rated the same items* should be more similar than two users who've rated some different items.

Ultimately, these are *heuristics,* and either definition could be used depending on the situation

Collaborative filtering in practice

How does amazon generate their recommendations?

Given a product:

Let U_i be the set of users who viewed it

Rank products according to: $\frac{|U_i \cap U_j|}{|U_i \cup U_j|}$ (or cosine/pearson)

Linden, Smith, & York (2003)

Collaborative filtering in practice

Can also use similarity functions to estimate ratings:

$$r(u,i) = \frac{1}{Z} \sum_{j \in I_u \setminus \{i\}} \mathrm{Sim}(i,j)\, r(u,j)$$

$$\hookrightarrow \sum_{j \in I_u \setminus \{i\}} \mathrm{Sim}(i,j)$$

Collaborative filtering in practice

Note: (surprisingly) that we built something pretty useful out of **nothing but rating data** – we didn’t look at any features of the products whatsoever

Collaborative filtering in practice

But: we still have a few problems left to address…

1. This is actually kind of slow given a huge enough dataset – if one user purchases one item, this will change the rankings of **every other item that was purchased by at least one user in common**
2. Of no use for **new users** and **new items** ("cold-start" problems
3. Won't necessarily encourage diverse results

Learning Outcomes

- Introduced several similarity measures for different types of data (interactions, likes, ratings)
- Showed how recommender systems can operate purely based on interactions, without observed features

Web Mining and Recommender Systems

Similarity based recommender – implementation

Learning Goals

- Walk through a quick implementation of a similarity-based recommender

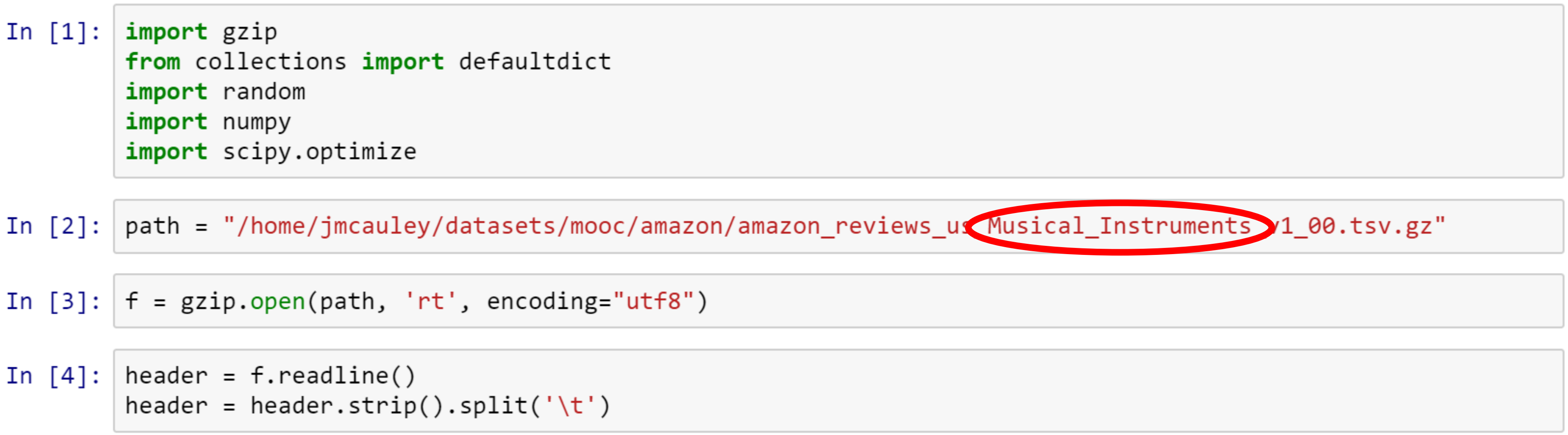
Code

Code on course webpage

Uses Amazon "Musical Instrument" data from https://s3.amazonaws.com/amazon-reviews-pds/tsv/index.txt

Code: Reading the data

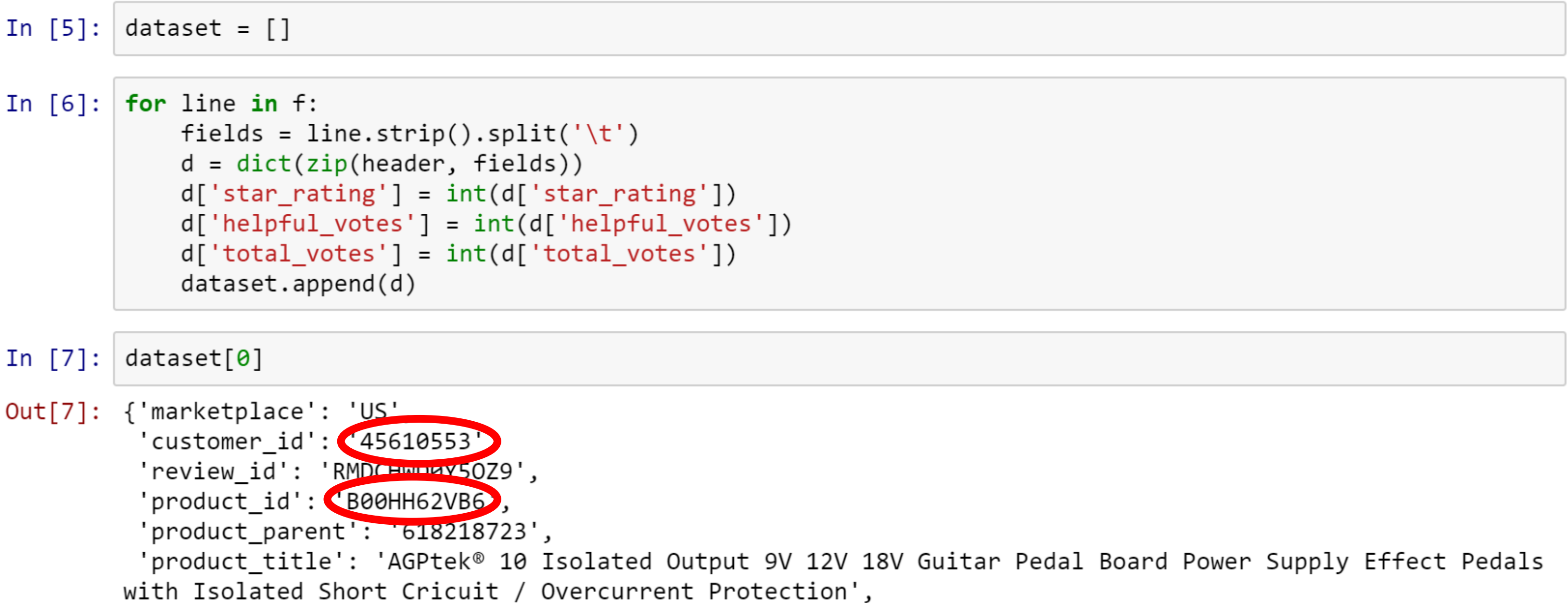
Read the data:



```
In [1]: import gzip
        from collections import defaultdict
        import random
        import numpy
        import scipy.optimize

In [2]: path = "/home/jmcauley/datasets/mooc/amazon/amazon_reviews_us_Musical_Instruments_v1_00.tsv.gz"

In [3]: f = gzip.open(path, 'rt', encoding="utf8")

In [4]: header = f.readline()
        header = header.strip().split('\t')
```

Code: Reading the data

Our goal is to make recommendations of products based on users' purchase histories. The only information needed to do so is **user and item IDs**



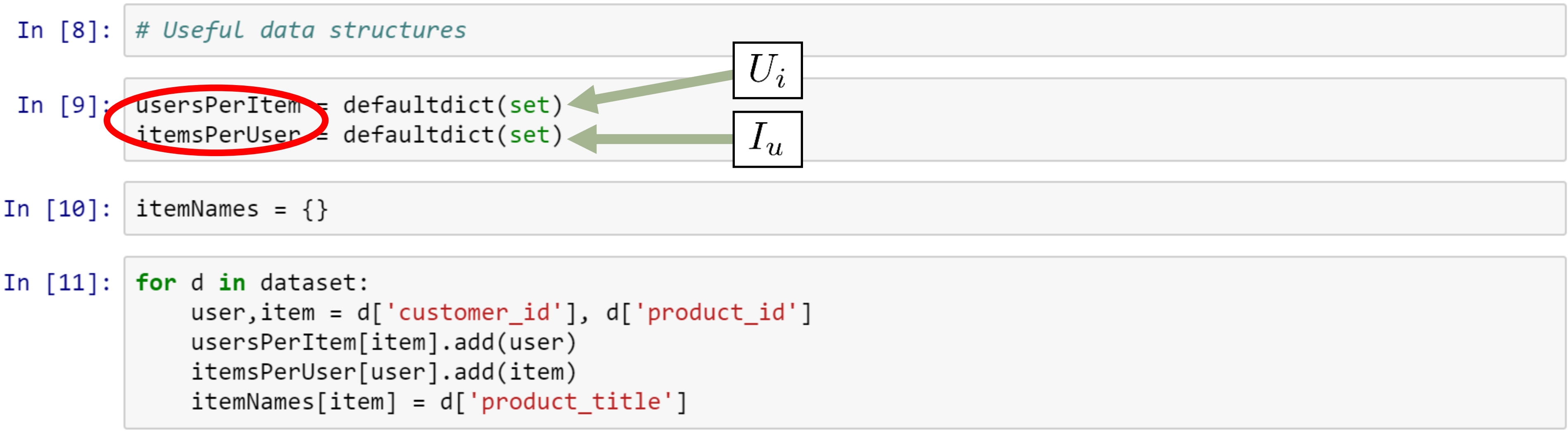
```
In [5]: dataset = []
```

```
In [6]: for line in f:
            fields = line.strip().split('\t')
            d = dict(zip(header, fields))
            d['star_rating'] = int(d['star_rating'])
            d['helpful_votes'] = int(d['helpful_votes'])
            d['total_votes'] = int(d['total_votes'])
            dataset.append(d)
```

```
In [7]: dataset[0]

Out[7]: {'marketplace': 'US',
         'customer_id': '45610553',
         'review_id': 'RMDCHWD0Y5OZ9',
         'product_id': 'B00HH62VB6',
         'product_parent': '618218723',
         'product_title': 'AGPtek® 10 Isolated Output 9V 12V 18V Guitar Pedal Board Power Supply Effect Pedals
        with Isolated Short Cricuit / Overcurrent Protection',
```

Code: Useful data structures

Build data structures representing the set of items for each user and users for each item:

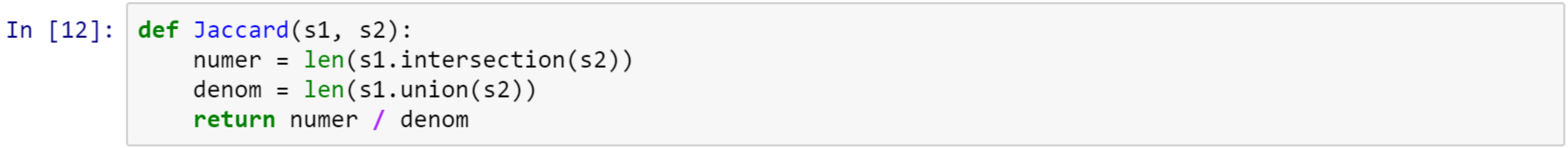


```
In [8]:  # Useful data structures

In [9]:  usersPerItem = defaultdict(set)
         itemsPerUser = defaultdict(set)

In [10]: itemNames = {}

In [11]: for d in dataset:
             user,item = d['customer_id'], d['product_id']
             usersPerItem[item].add(user)
             itemsPerUser[user].add(item)
             itemNames[item] = d['product_title']
```

Code: Jaccard similarity

The Jaccard similarity implementation follows the definition directly:

$$\text{Jaccard}(A, B) = \frac{|A \cap B|}{|A \cup B|}$$



In [12]:

```python
def Jaccard(s1, s2):
    numer = len(s1.intersection(s2))
    denom = len(s1.union(s2))
    return numer / denom
```

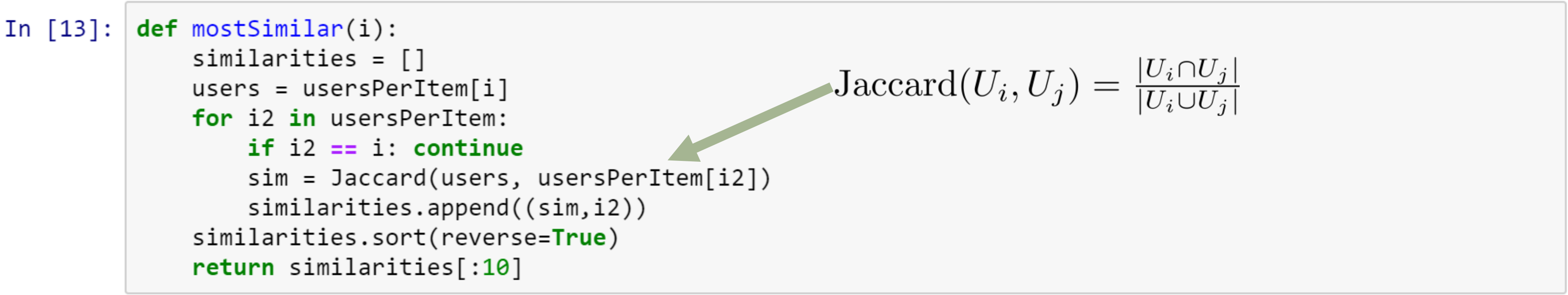
Recommendation

We want a recommendation function that return **items similar to a candidate item *i*.** Our strategy will be as follows:

- Find the set of users who purchased *i*
- Iterate over all other items other than *i*
- For all other items, compute their similarity with *i* *(and store it)*
- Sort all other items by (Jaccard) similarity
- Return the most similar

Code: Recommendation

Now we can implement the recommendation function itself:

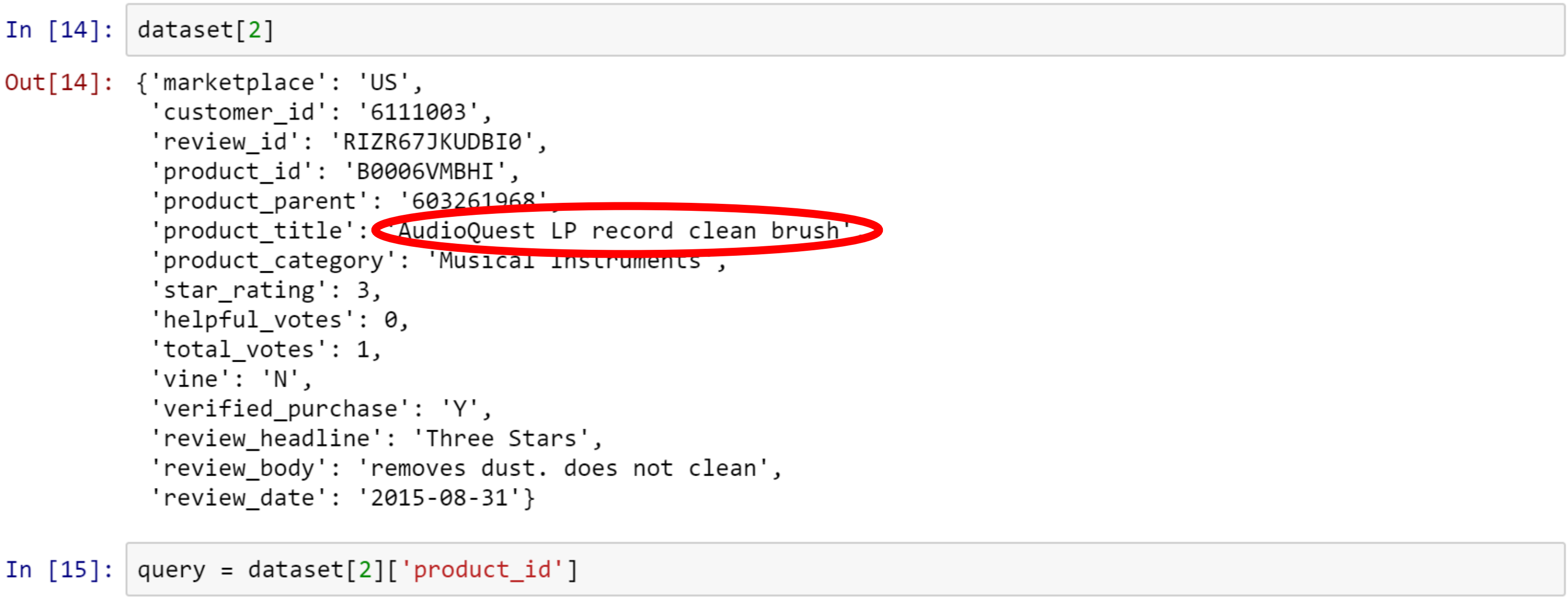


```
In [13]: def mostSimilar(i):
             similarities = []
             users = usersPerItem[i]
             for i2 in usersPerItem:
                 if i2 == i: continue
                 sim = Jaccard(users, usersPerItem[i2])
                 similarities.append((sim,i2))
             similarities.sort(reverse=True)
             return similarities[:10]
```

$$\text{Jaccard}(U_i, U_j) = \frac{|U_i \cap U_j|}{|U_i \cup U_j|}$$

Code: Recommendation

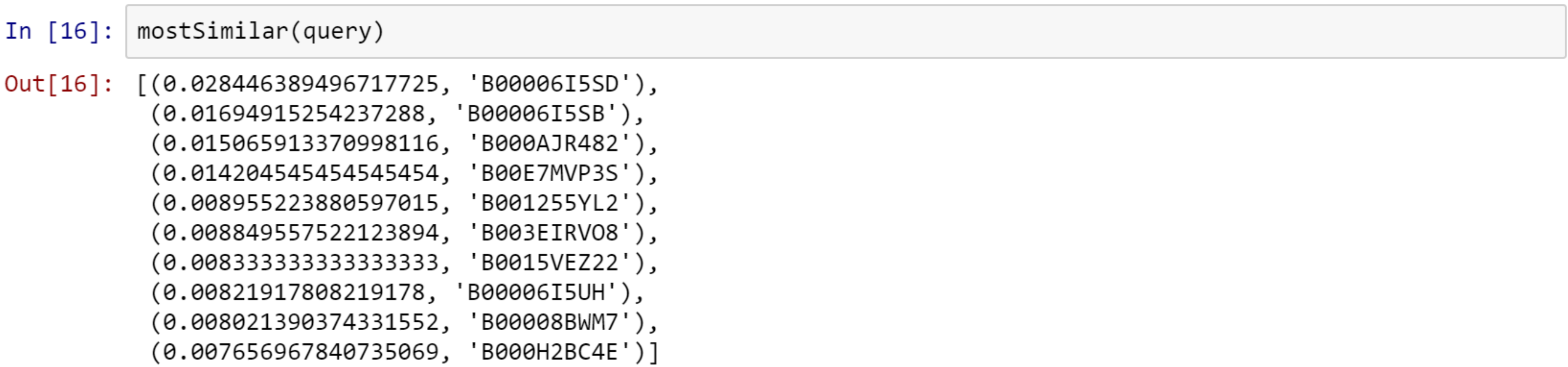
Next, let's use the code to make a recommendation.
The query is just a product ID:



```
In [14]: dataset[2]

Out[14]: {'marketplace': 'US',
          'customer_id': '6111003',
          'review_id': 'RIZR67JKUDBI0',
          'product_id': 'B0006VMBHI',
          'product_parent': '603261968',
          'product_title': 'AudioQuest LP record clean brush',
          'product_category': 'Musical Instruments',
          'star_rating': 3,
          'helpful_votes': 0,
          'total_votes': 1,
          'vine': 'N',
          'verified_purchase': 'Y',
          'review_headline': 'Three Stars',
          'review_body': 'removes dust. does not clean',
          'review_date': '2015-08-31'}

In [15]: query = dataset[2]['product_id']
```

Code: Recommendation

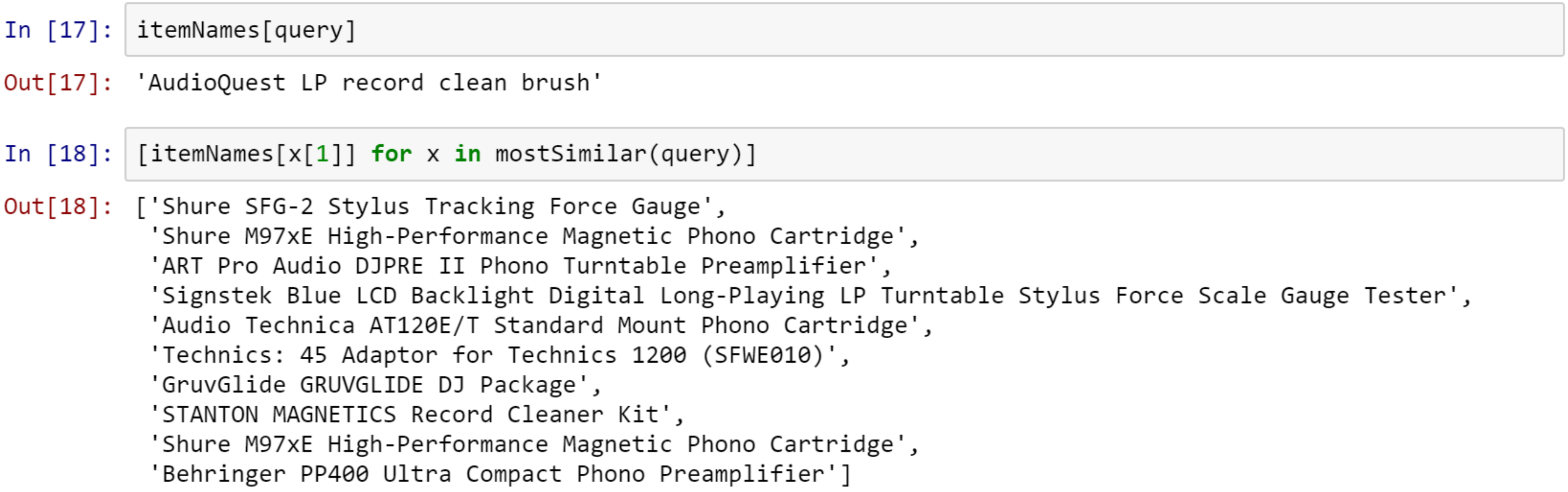
Next, let's use the code to make a recommendation.
The query is just a product ID:



```
In [16]:  mostSimilar(query)

Out[16]:  [(0.028446389496717725, 'B00006I5SD'),
           (0.01694915254237288, 'B00006I5SB'),
           (0.015065913370998116, 'B000AJR482'),
           (0.014204545454545454, 'B00E7MVP3S'),
           (0.008955223880597015, 'B001255YL2'),
           (0.008849557522123894, 'B003EIRVO8'),
           (0.008333333333333333, 'B0015VEZ22'),
           (0.00821917808219178, 'B00006I5UH'),
           (0.008021390374331552, 'B00008BWM7'),
           (0.007656967840735069, 'B000H2BC4E')]
```

Code: Recommendation

Items that were recommended:



```
In [17]: itemNames[query]

Out[17]: 'AudioQuest LP record clean brush'

In [18]: [itemNames[x[1]] for x in mostSimilar(query)]

Out[18]: ['Shure SFG-2 Stylus Tracking Force Gauge',
          'Shure M97xE High-Performance Magnetic Phono Cartridge',
          'ART Pro Audio DJPRE II Phono Turntable Preamplifier',
          'Signstek Blue LCD Backlight Digital Long-Playing LP Turntable Stylus Force Scale Gauge Tester',
          'Audio Technica AT120E/T Standard Mount Phono Cartridge',
          'Technics: 45 Adaptor for Technics 1200 (SFWE010)',
          'GruvGlide GRUVGLIDE DJ Package',
          'STANTON MAGNETICS Record Cleaner Kit',
          'Shure M97xE High-Performance Magnetic Phono Cartridge',
          'Behringer PP400 Ultra Compact Phono Preamplifier']
```

Recommending more efficiently

Our implementation was not very efficient. The slowest component is the iteration over all other items:

- Find the set of users who purchased *i*
- **Iterate over all other items other than *i***
- For all other items, compute their similarity with *i* *(and store it)*
- Sort all other items by (Jaccard) similarity
- Return the most similar

This can be done more efficiently as most items will have no overlap

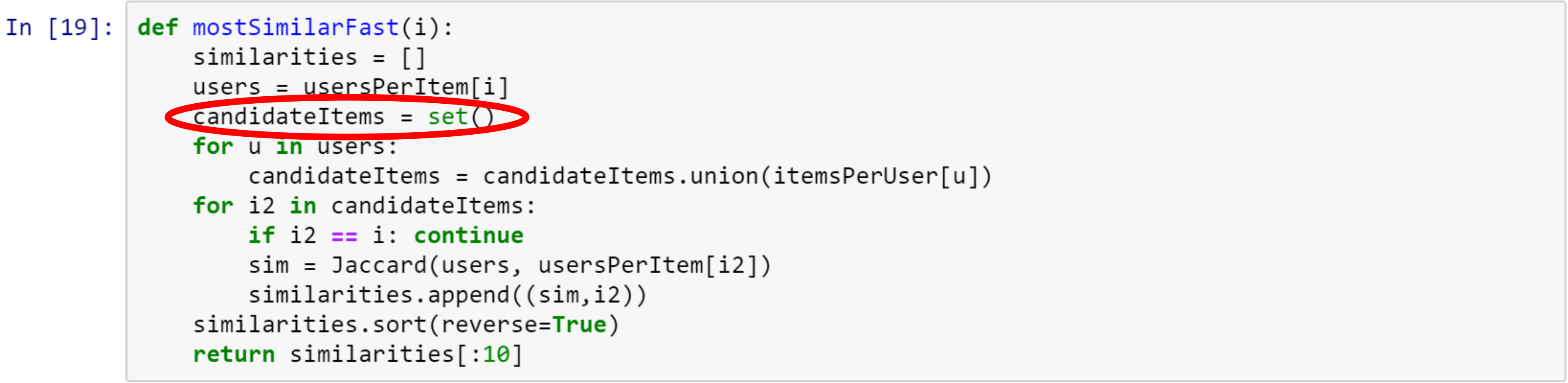
Recommending more efficiently

In fact it is sufficient to iterate over **those items purchased by one of the users who purchased *i***

- Find the set of users who purchased *i*
- **Iterate over all users who purchased *i***
- Build a candidate set from all items those users consumed
- For items in this set, compute their similarity with *i* *(and store it)*
- Sort all other items by (Jaccard) similarity
- Return the most similar

Code: Faster implementation

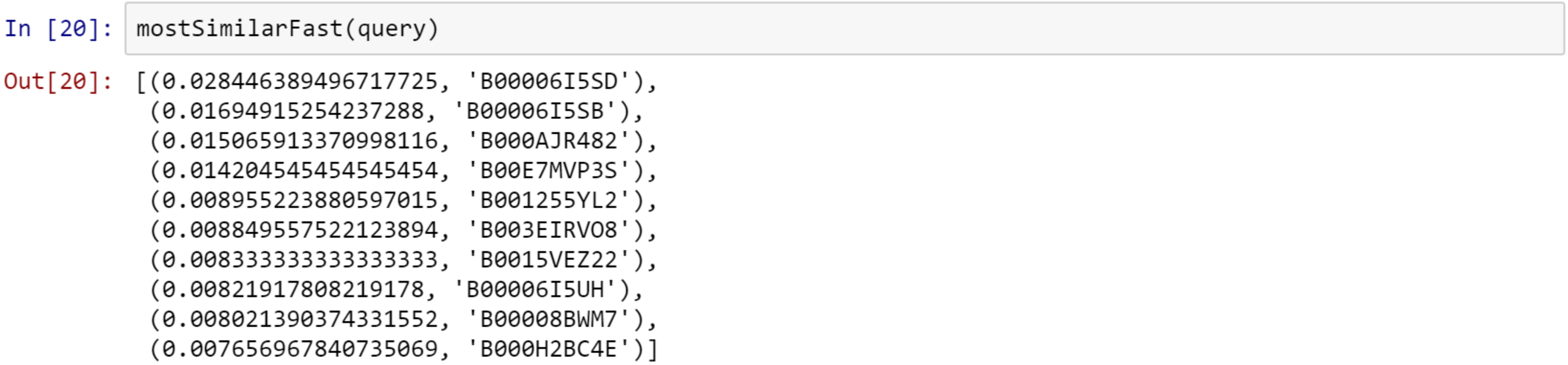
Our more efficient implementation works as follows:



```
In [19]: def mostSimilarFast(i):
             similarities = []
             users = usersPerItem[i]
             candidateItems = set()
             for u in users:
                 candidateItems = candidateItems.union(itemsPerUser[u])
             for i2 in candidateItems:
                 if i2 == i: continue
                 sim = Jaccard(users, usersPerItem[i2])
                 similarities.append((sim,i2))
             similarities.sort(reverse=True)
             return similarities[:10]
```

Code: Faster recommendation

Which ought to recommend the same set of items, but **much** more quickly:



```
In [20]: mostSimilarFast(query)

Out[20]: [(0.028446389496717725, 'B00006I5SD'),
          (0.01694915254237288, 'B00006I5SB'),
          (0.015065913370998116, 'B000AJR482'),
          (0.014204545454545454, 'B00E7MVP3S'),
          (0.008955223880597015, 'B001255YL2'),
          (0.008849557522123894, 'B003EIRVO8'),
          (0.008333333333333333, 'B0015VEZ22'),
          (0.00821917808219178, 'B00006I5UH'),
          (0.008021390374331552, 'B00008BWM7'),
          (0.007656967840735069, 'B000H2BC4E')]
```

Learning Outcomes

- Walked through an implementation of a similarity-based recommender, and discussed some of the computational challenges involved

Web Mining and Recommender Systems

Similarity-based rating prediction

Learning Goals

- Show how a similarity-based recommender can be used for rating prediction

Collaborative filtering for rating prediction

In the previous section we provided code to make recommendations based on the **Jaccard similarity**

How can the same ideas be used for rating prediction?

Collaborative filtering for rating prediction

A simple heuristic for rating prediction works as follows:

- The user (*u*)'s rating for an item *i* is a weighted combination of all of their previous ratings for items *j*
- The weight for each rating is given by the Jaccard similarity between *i* and *j*

Collaborative filtering for rating prediction

This can be written as:

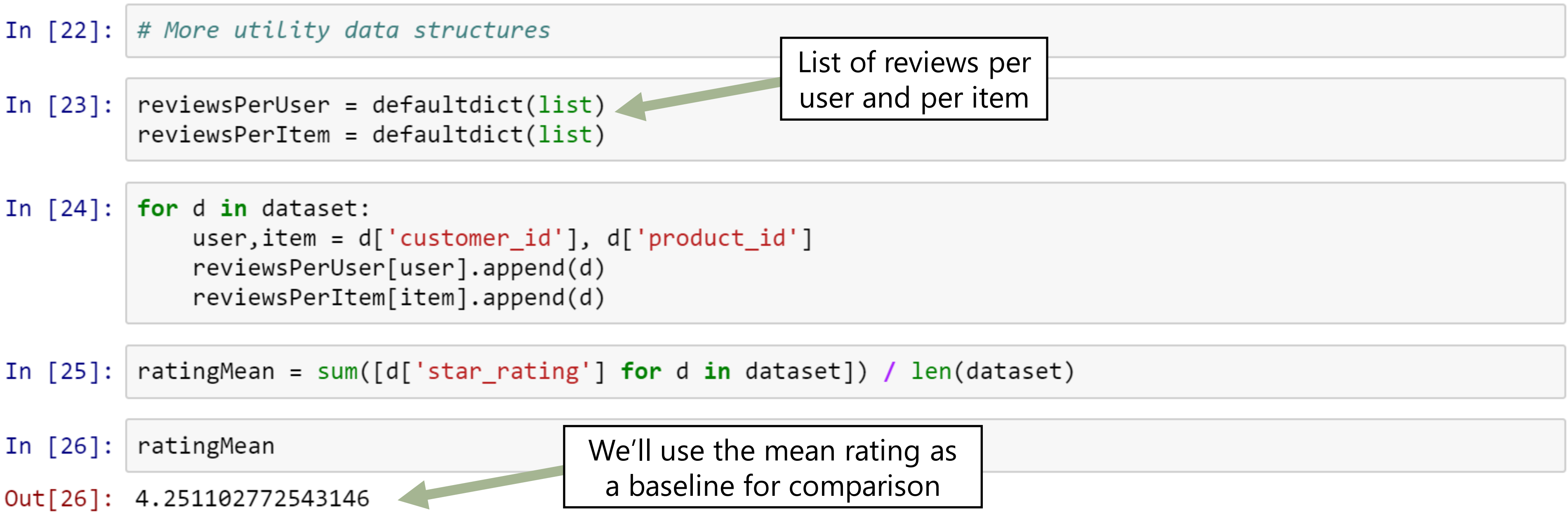
$$r(u,i) = \frac{1}{Z} \sum_{\underbrace{j \in I_u \setminus \{i\}}_{\text{All items the user has rated other than } i}} r_{u,j} \cdot \mathrm{sim}(i,j)$$

Normalization constant

$$Z = \sum_{j \in I_u \setminus \{i\}} \mathrm{sim}(i,j)$$

Code: CF for rating prediction

Now we can adapt our previous recommendation code to predict ratings



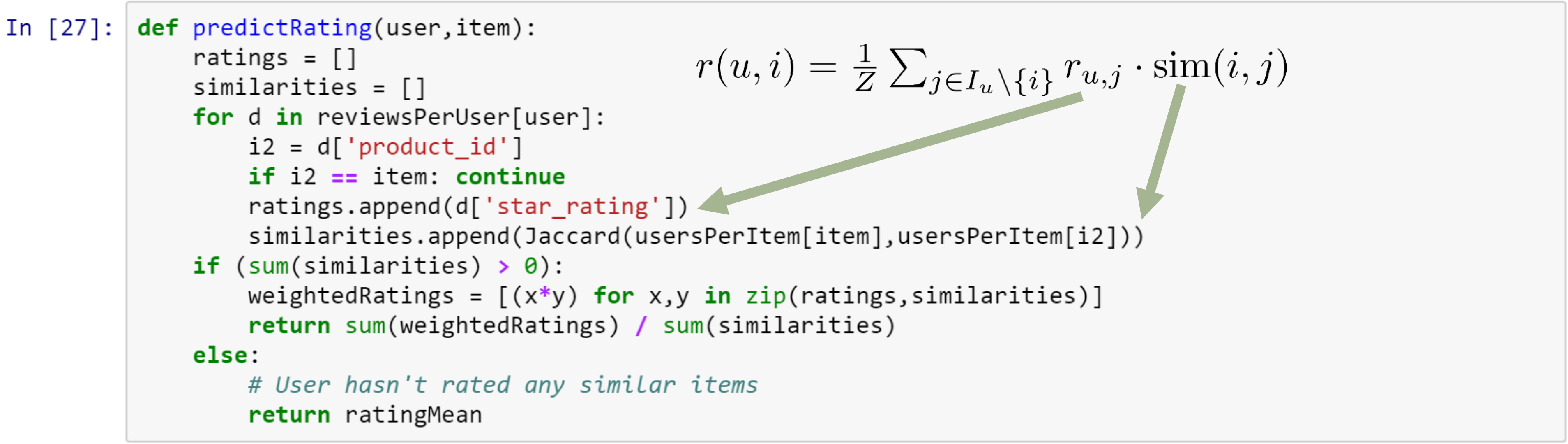
```
In [22]: # More utility data structures

In [23]: reviewsPerUser = defaultdict(list)
         reviewsPerItem = defaultdict(list)

In [24]: for d in dataset:
             user,item = d['customer_id'], d['product_id']
             reviewsPerUser[user].append(d)
             reviewsPerItem[item].append(d)

In [25]: ratingMean = sum([d['star_rating'] for d in dataset]) / len(dataset)

In [26]: ratingMean

Out[26]: 4.251102772543146
```

List of reviews per user and per item

We'll use the mean rating as a baseline for comparison

Code: CF for rating prediction

Our rating prediction code works as follows:



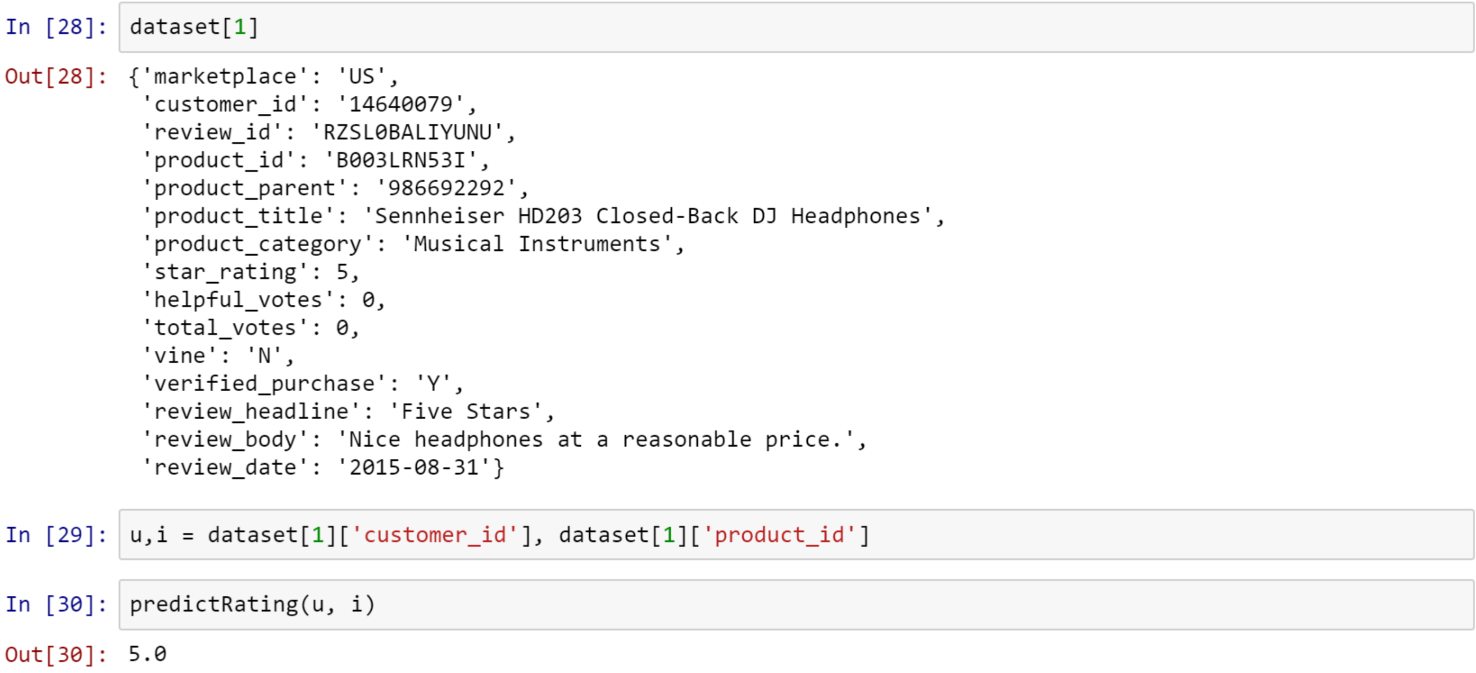
In [27]:

```python
def predictRating(user,item):
    ratings = []
    similarities = []
    for d in reviewsPerUser[user]:
        i2 = d['product_id']
        if i2 == item: continue
        ratings.append(d['star_rating'])
        similarities.append(Jaccard(usersPerItem[item],usersPerItem[i2]))
    if (sum(similarities) > 0):
        weightedRatings = [(x*y) for x,y in zip(ratings,similarities)]
        return sum(weightedRatings) / sum(similarities)
    else:
        # User hasn't rated any similar items
        return ratingMean
```

$$r(u,i) = \frac{1}{Z} \sum_{j \in I_u \setminus \{i\}} r_{u,j} \cdot \mathrm{sim}(i,j)$$

Code: CF for rating prediction

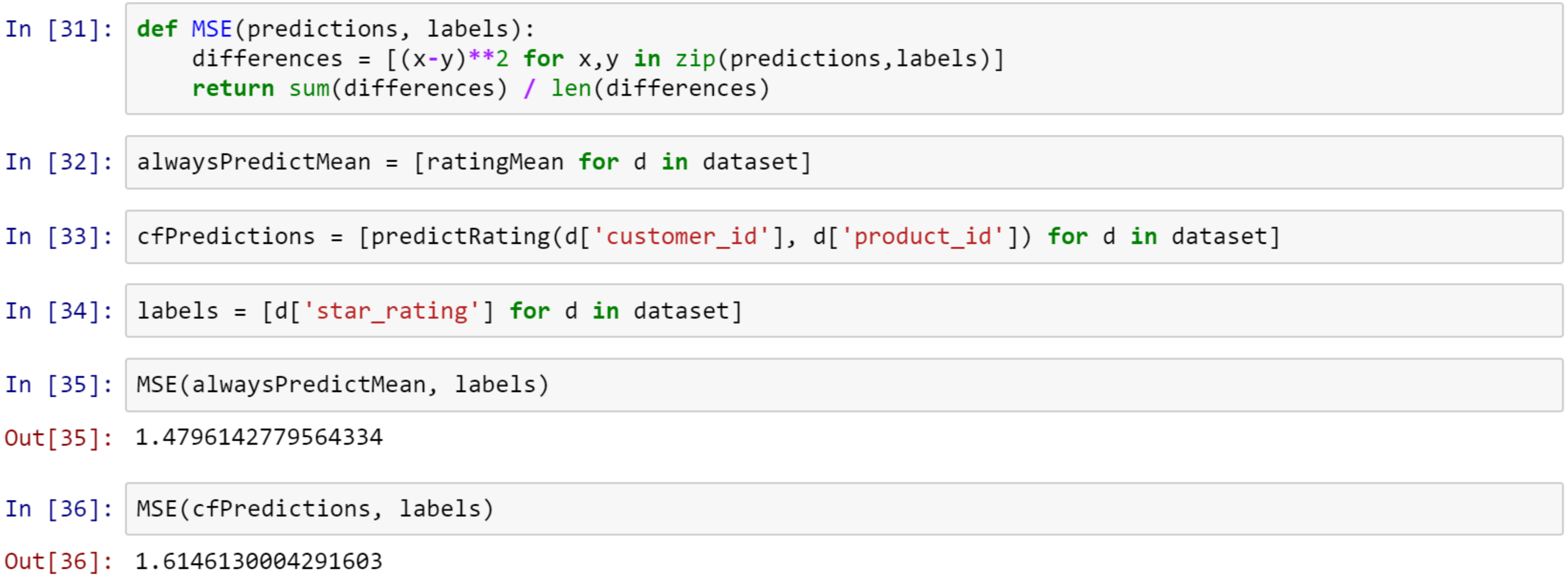
As an example, select a rating for prediction:



```
In [28]: dataset[1]

Out[28]: {'marketplace': 'US',
          'customer_id': '14640079',
          'review_id': 'RZSL0BALIYUNU',
          'product_id': 'B003LRN53I',
          'product_parent': '986692292',
          'product_title': 'Sennheiser HD203 Closed-Back DJ Headphones',
          'product_category': 'Musical Instruments',
          'star_rating': 5,
          'helpful_votes': 0,
          'total_votes': 0,
          'vine': 'N',
          'verified_purchase': 'Y',
          'review_headline': 'Five Stars',
          'review_body': 'Nice headphones at a reasonable price.',
          'review_date': '2015-08-31'}

In [29]: u,i = dataset[1]['customer_id'], dataset[1]['product_id']

In [30]: predictRating(u, i)

Out[30]: 5.0
```

Code: CF for rating prediction

Similarly, we can evaluate accuracy across the entire corpus:



```
In [31]: def MSE(predictions, labels):
             differences = [(x-y)**2 for x,y in zip(predictions,labels)]
             return sum(differences) / len(differences)

In [32]: alwaysPredictMean = [ratingMean for d in dataset]

In [33]: cfPredictions = [predictRating(d['customer_id'], d['product_id']) for d in dataset]

In [34]: labels = [d['star_rating'] for d in dataset]

In [35]: MSE(alwaysPredictMean, labels)

Out[35]: 1.4796142779564334

In [36]: MSE(cfPredictions, labels)

Out[36]: 1.6146130004291603
```

Collaborative filtering for rating prediction

Note that this is just a **heuristic** for rating prediction

- In fact in this case it did *worse* (in terms of the MSE) than always predicting the mean
- We could adapt this to use:

1. A different similarity function (e.g. cosine)
2. Similarity based on users rather than items
3. A different weighting scheme

Learning Outcomes

- Examined the use of a similarity-based recommender for rating prediction

Web Mining and Recommender Systems

Latent-factor models

Learning Goals

- Show how recommendation can be cast as a supervised learning problem
- (Start to) introduce **latent factor models**

Summary so far

Recap

1. Measuring similarity between users/items for **binary** prediction
Jaccard similarity
2. Measuring similarity between users/items for **real-valued** prediction
cosine/Pearson similarity

Now: Dimensionality reduction for **real-valued** prediction *latent-factor models*

Latent factor models

So far we've looked at approaches that try to define some definition of user/user and item/item **similarity**

Recommendation then consists of

- Finding an item *i* that a user likes (gives a high rating)
- Recommending items that are similar to it (i.e., items *j* with a similar rating profile to *i*)

Latent factor models

What we’ve seen so far are **unsupervised** approaches and whether the work depends highly on whether we chose a “good” notion of similarity

So, can we perform recommendations via **supervised** learning?

Latent factor models

e.g. if we can model

$$f(\text{user features}, \text{movie features}) \rightarrow \text{star rating}$$

Then recommendation will consist of identifying

$$recommendation(u) = \arg\max_{i \in \text{unseen items}} f(u, i)$$

The Netflix prize

In 2006, Netflix created a dataset of **100,000,000** movie ratings
Data looked like:

$$(\text{userID}, \text{itemID}, \text{time}, \text{rating})$$

The goal was to reduce the (R)MSE at predicting ratings:

$$\text{RMSE}(f) = \sqrt{\tfrac{1}{N} \textstyle\sum_{u,i,t \in \text{test set}} (\underset{\substack{\uparrow \\ \text{model's prediction}}}{f(u,i,t)} - \underset{\substack{\uparrow \\ \text{ground-truth}}}{r_{u,i,t}})^2}$$

Whoever first manages to reduce the RMSE by **10%** versus
Netflix's solution wins **$1,000,000**

The Netflix prize

This led to **a lot** of research on rating prediction by minimizing the Mean-Squared Error

NETFLIX

(it also led to a lawsuit against Netflix, once somebody managed to de-anonymize their data)

We’ll look at a few of the main approaches

Rating prediction

Let's start with the simplest possible model:

$$f(\underset{\text{user}}{\underset{\uparrow}{u}}, \underset{\text{item}}{\underset{\uparrow}{i}}) = \alpha$$

$$\alpha = \overline{R}$$

Rating prediction

What about the **2nd** simplest model?

$$f(u, i) = \alpha + \beta_u + \beta_i$$

user item

how much does this user tend to rate things above the mean?

does this item tend to receive higher ratings than others

e.g.

$\alpha = 4.2$

$\beta_{\text{pitch black}} = -0.1$

$\beta_{\text{julian}} = -0.2$

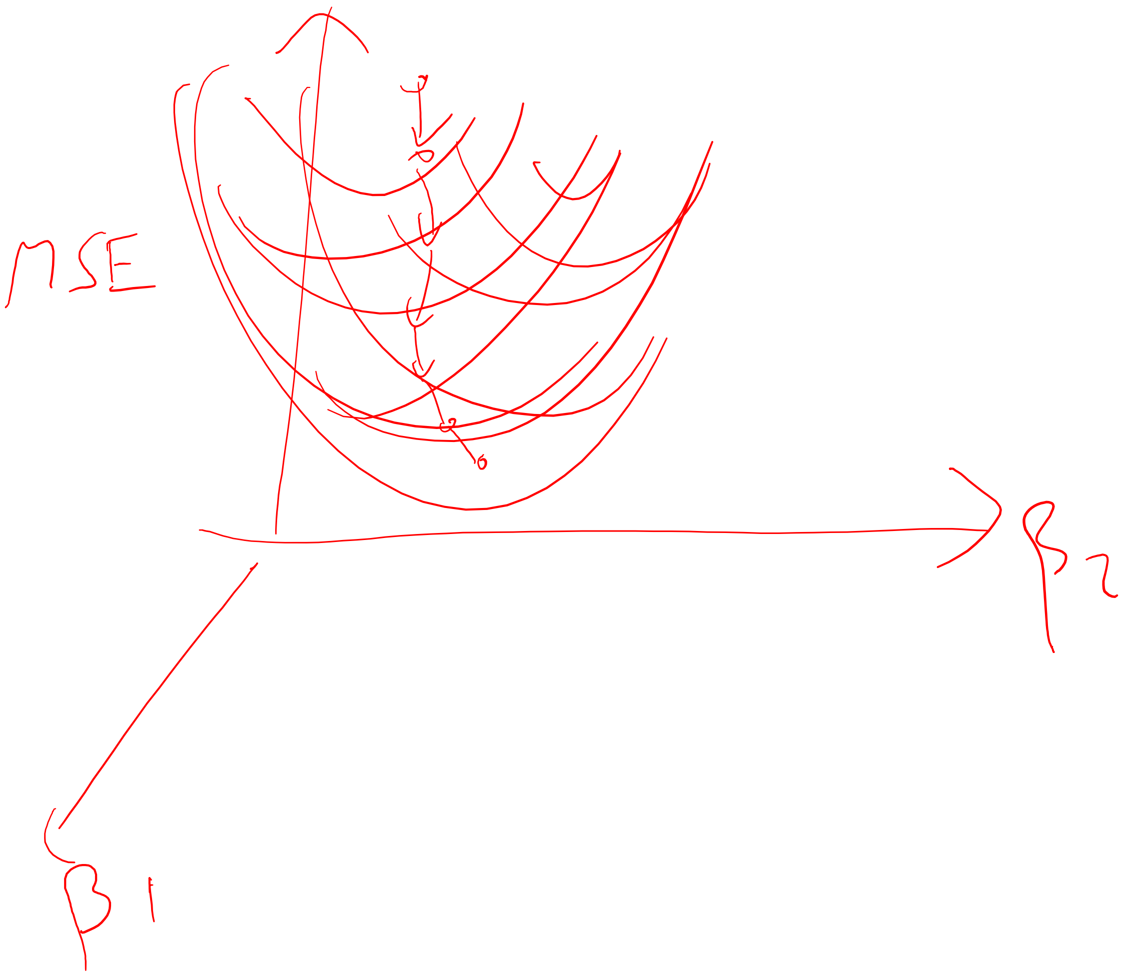
Rating prediction

The optimization problem becomes:

$$\arg\min_{\alpha,\beta} \underbrace{\sum_{u,i}(\alpha + \beta_u + \beta_i - R_{u,i})^2}_{\text{error}} + \underbrace{\lambda \left[\sum_u \beta_u^2 + \sum_i \beta_i^2\right]}_{\text{regularizer}}$$

Jointly convex in \beta_i, \beta_u. Can be solved by iteratively removing the mean and solving for beta

Jointly convex?



Rating prediction

Differentiate:

$$\arg\min_{\alpha,\beta} \sum_{u,i}(\alpha + \beta_u + \beta_i - R_{u,i})^2 + \lambda \left[\sum_u \beta_u^2 + \sum_i \beta_i^2\right]$$

$$\frac{\partial obj}{\partial \beta_u} \quad \sum_{i \in I_u} 2\left(\alpha + \beta_u + \beta_i - R_{u,i}\right) + 2\lambda\beta_u$$

Rating prediction

Differentiate:

$$\frac{\partial \text{obj}}{\partial \beta_u} = \sum_{i \in I_u} 2(\alpha + \beta_u + \beta_i - R_{u,i}) + 2\lambda\beta_u$$

Two ways to solve:

1. "Regular" gradient descent
2. Solve $\frac{\partial \text{obj}}{\partial \beta_u} = 0$ (sim. for beta_i, alpha)

Rating prediction

Differentiate:

$$\frac{\partial \text{obj}}{\partial \beta_u} = \sum_{i \in I_u} 2(\alpha + \beta_u + \beta_i - R_{u,i}) + 2\lambda\beta_u$$

Solve $\frac{\partial \text{obj}}{\partial \beta_u} = 0$:

$$-\lambda\beta_u = \sum_{i \in I_u} (\alpha + \beta_u + \beta_i - R_{u,i})$$

$$-(\lambda + |I_u|)\beta_u = \sum_{i \in I_u} (\alpha + \beta_i - R_{u,i})$$

$$\beta_u = \frac{\sum_{i \in I_u} (\alpha + \beta_i - R_{u,i})}{-(\lambda + |I_u|)}$$

Rating prediction

Iterative procedure – repeat the following updates until convergence:

$$\alpha^{(t)} = \frac{\sum_{u,i\in\text{train}}(R_{u,i}-(\beta_u+\beta_i))}{N_{\text{train}}}$$

$$\beta_u^{(t+1)} = \frac{\sum_{i\in I_u} R_{u,i}-(\alpha+\beta_i)}{\lambda+|I_u|}$$

$$\beta_i^{(t+2)} = \frac{\sum_{u\in U_i} R_{u,i}-(\alpha+\beta_u)}{\lambda+|U_i|}$$

(exercise: write down derivatives and convince yourself of these update equations!)

Rating prediction

Looks good (and actually works surprisingly well), but doesn’t solve the basic issue that we started with

$$f(\text{user features}, \text{movie features}) =$$

$$= \underbrace{\langle \phi(\text{user features}), \theta_{\text{user}} \rangle}_{\text{user predictor}} + \underbrace{\langle \phi(\text{movie features}), \theta_{\text{movie}} \rangle}_{\text{movie predictor}}$$

That is, we’re **still** fitting a function that treats users and items independently

Learning Outcomes

- Introduced (some of) the **latent factor model**
- Thought about how describe rating prediction as a regression/supervised learning task
- Discussed the history of this type of recommendation system

Web Mining and Recommender Systems

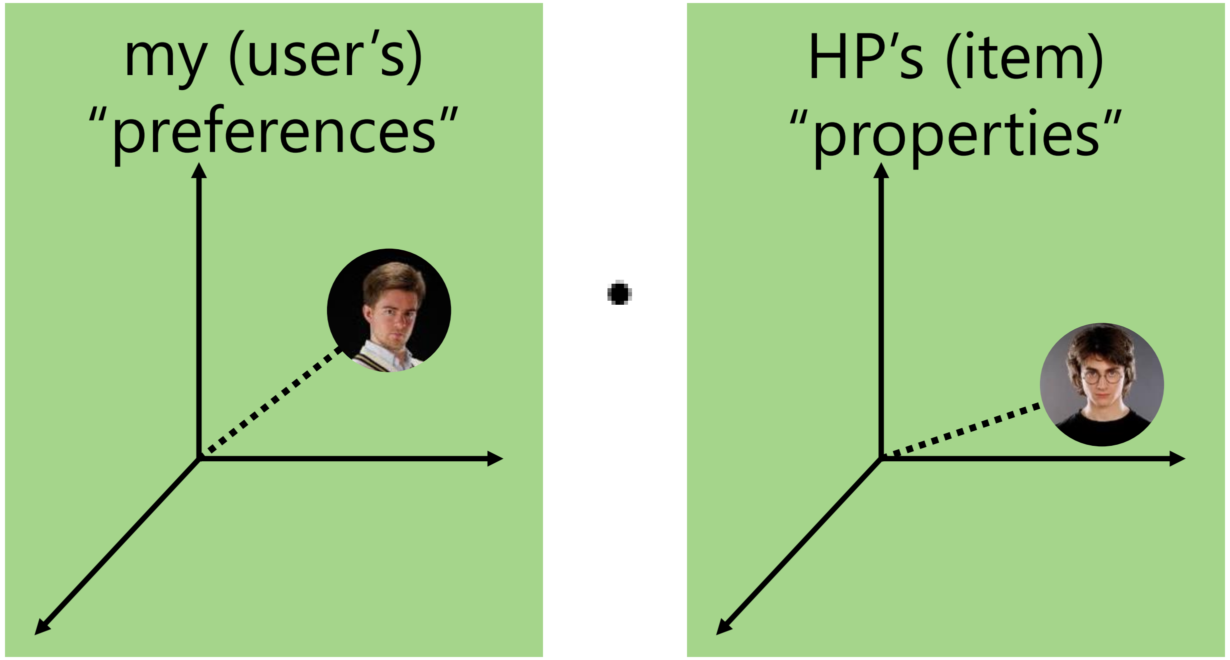
Latent-factor models (part 2)

Learning Goals

- Complete our presentation of the latent factor model

Recommending things to people

How about an approach based on **dimensionality reduction?**



my (user’s) “preferences”

•

HP’s (item) “properties”

i.e., let’s come up with low-dimensional representations of the users and the items so as to best explain the data

Dimensionality reduction

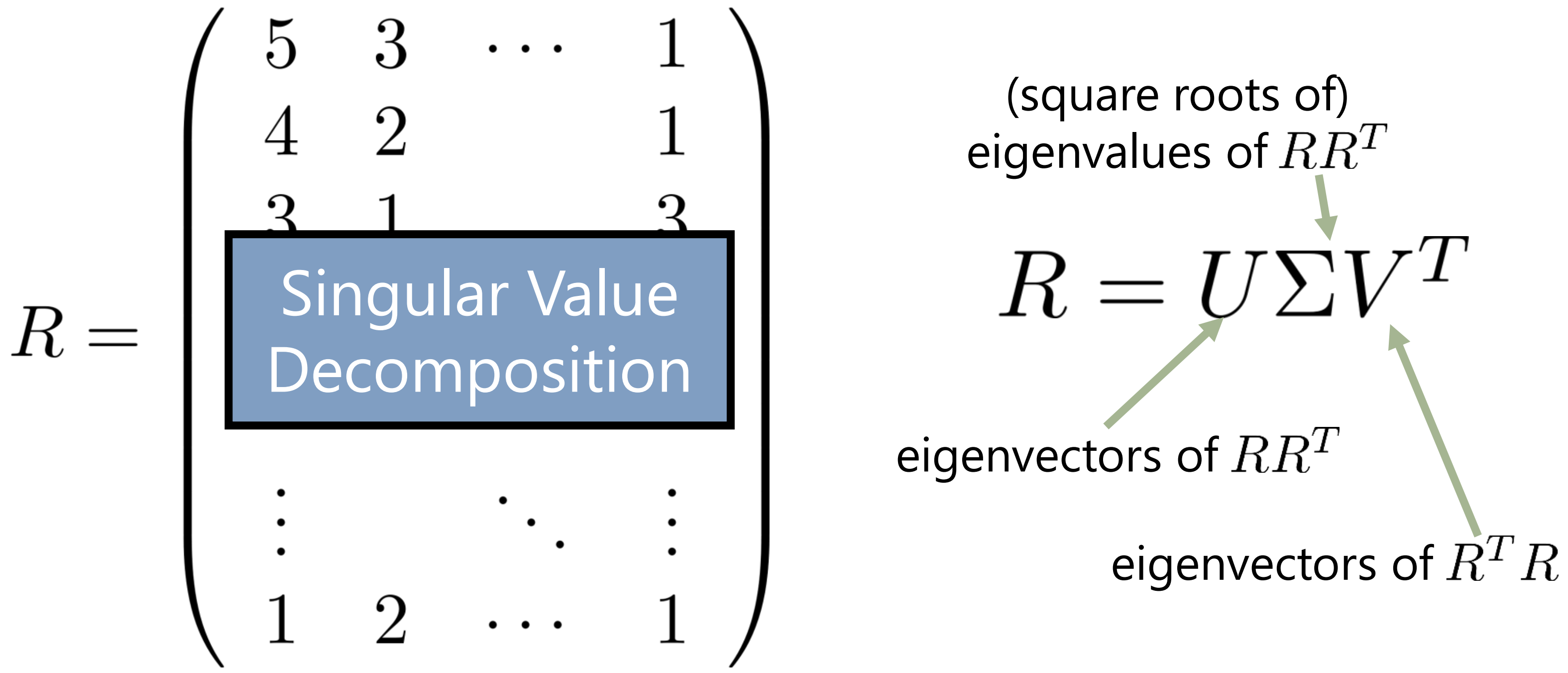
We already have some tools that ought to help us, e.g. from dimensionality reduction:

$$R = \begin{pmatrix} 5 & 3 & \cdots & 1 \\ 4 & 2 & & 1 \\ 3 & 1 & & 3 \\ 2 & 2 & & 4 \\ 1 & 5 & & 2 \\ \vdots & & \ddots & \vdots \\ 1 & 2 & \cdots & 1 \end{pmatrix}$$

What is the best low-rank approximation of *R* in terms of the mean-squared error?

Dimensionality reduction

We already have some tools that ought to help us, e.g. from dimensionality reduction:



$$R = \begin{pmatrix} 5 & 3 & \cdots & 1 \\ 4 & 2 & & 1 \\ 3 & 1 & & 3 \\ \vdots & & \ddots & \vdots \\ 1 & 2 & \cdots & 1 \end{pmatrix}$$

Singular Value Decomposition

(square roots of) eigenvalues of RR^T

$$R = U\Sigma V^T$$

eigenvectors of RR^T

eigenvectors of $R^T R$

The “best” rank-K approximation (in terms of the MSE) consists of taking the eigenvectors with the highest eigenvalues

Dimensionality reduction

But! Our matrix of ratings is only partially observed; and it's **really big!**

$$R = \begin{pmatrix} 5 & 3 & \cdots & \cdot \\ 4 & 2 & & 1 \\ 3 & \cdot & & 3 \\ \cdot & 2 & & 4 \\ 1 & 5 & & \cdot \\ \vdots & & \ddots & \vdots \\ 1 & 2 & \cdots & \cdot \end{pmatrix}$$

Missing ratings

SVD is **not defined** for partially observed matrices, and it is **not practical** for matrices with 1Mx1M+ dimensions

Latent-factor models

Instead, let's solve approximately using gradient descent

$$R = \left.\underbrace{\begin{pmatrix} 5 & 3 & \cdots & \cdot \\ 4 & 2 & & 1 \\ 3 & \cdot & & 3 \\ \cdot & 2 & & 4 \\ 1 & 5 & & \cdot \\ \vdots & & \ddots & \vdots \\ 1 & 2 & \cdots & \cdot \end{pmatrix}}_{\text{items}}\right\}\text{users}$$

K-dimensional representation of each item

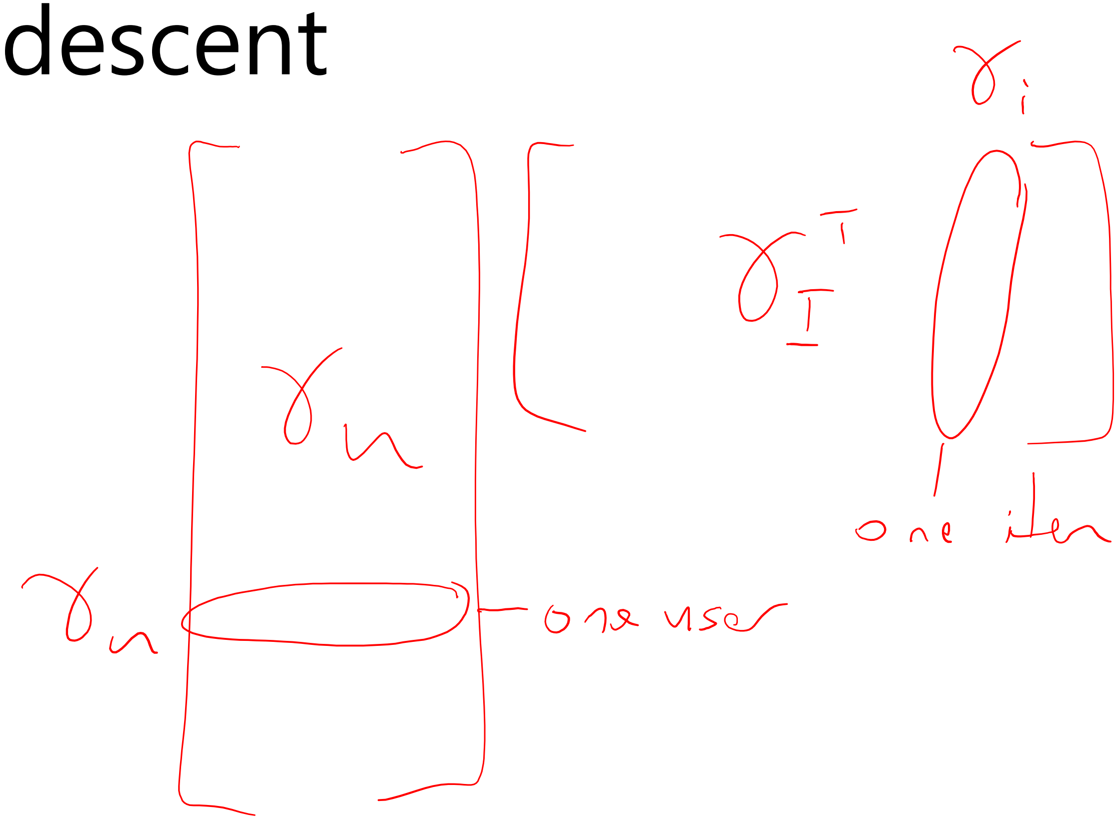
$$R \simeq UV^T$$

K-dimensional representation of each user

Latent-factor models

Instead, let's solve approximately using gradient descent

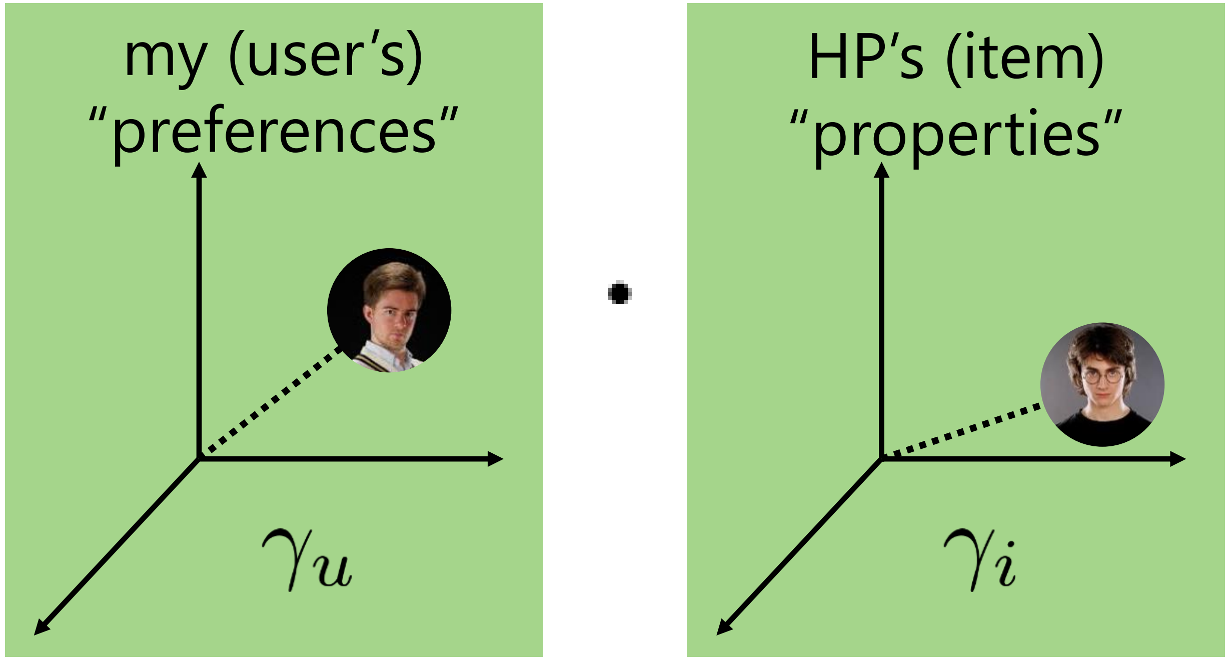
$$R = \begin{pmatrix} 5 & 3 & \cdots & \cdot \\ 4 & 2 & & 1 \\ 3 & \cdot & & 3 \\ \cdot & 2 & & 4 \\ 1 & 5 & & \cdot \\ \vdots & & \ddots & \vdots \\ 1 & 2 & \cdots & \cdot \end{pmatrix}$$



Latent-factor models

Let’s write this as:

$$f(u,i) = \alpha + \beta_u + \beta_i + \gamma_u \cdot \gamma_i$$



my (user’s) “preferences”

HP’s (item) “properties”

Latent-factor models

Let's write this as:

$$f(u,i) = \alpha + \beta_u + \beta_i + \gamma_u \cdot \gamma_i$$

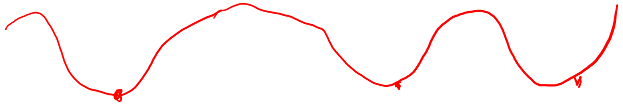
Our optimization problem is then

$$\arg\min_{\alpha,\beta,\gamma} \underbrace{\textstyle\sum_{u,i}(\alpha+\beta_u+\beta_i+\gamma_u\cdot\gamma_i-R_{u,i})^2}_{\text{error}}+\lambda \underbrace{\left[\textstyle\sum_u \beta_u^2 + \sum_i \beta_i^2 + \sum_i \|\gamma_i\|_2^2 + \sum_u \|\gamma_u\|_2^2\right]}_{\text{regularizer}}$$

Latent-factor models

Problem: this is certainly not convex

$$\underset{U}{\begin{bmatrix} \\ \gamma_u \\ \\ \end{bmatrix}} \begin{bmatrix} \gamma_i^T \\ \end{bmatrix} \approx R$$



Latent-factor models

Oh well. We'll just solve it approximately
Again, two ways to solve:

1. "Regular" gradient descent
2. Solve $\frac{\partial \mathrm{obj}}{\partial \gamma_u} = 0$ (sim. For beta_i, alpha, etc.)

(**Solution 1** is much easier to implement, though **Solution 2** might converge more quickly/easily)

Latent-factor models (Solution 1)

$$\arg\min_{\alpha,\beta,\gamma} \sum_{u,i} (\alpha+\beta_u+\beta_i+\underbrace{\gamma_u\cdot\gamma_i}_{\sum_k \gamma_{uk}\cdot\gamma_{ik}} - R_{u,i})^2 + \lambda \left[\sum_u \beta_u^2 + \sum_i \beta_i^2 + \sum_i \|\gamma_i\|_2^2 + \sum_u \|\gamma_u\|_2^2\right]$$

$$\frac{\partial obj}{\partial \gamma_{uk}} = \sum_{i \in I_u} 2\gamma_{ik}\left(\alpha + \beta_u + \beta_i + \gamma_u \cdot \gamma_i - R_{ui}\right)$$

$$+ \lambda 2 \gamma_{uk}$$

Latent-factor models (Solution 2)

Observation: if we know either the user or the item parameters, the problem becomes "easy"

$$f(u,i) = \alpha + \beta_u + \beta_i + \gamma_u \cdot \gamma_i$$

e.g. fix gamma_i – pretend we're fitting parameters for features

Latent-factor models

(Harder solution): iteratively solve the following subproblems

objective:

$$\underbrace{\arg\min_{\alpha,\beta,\gamma} \sum_{u,i}(\alpha+\beta_u+\beta_i+\gamma_u\cdot\gamma_i-R_{u,i})^2+\lambda\left[\sum_u \beta_u^2+\sum_i \beta_i^2+\sum_i \|\gamma_i\|_2^2+\sum_u \|\gamma_u\|_2^2\right]}$$

$$= \arg\min_{\alpha,\beta,\gamma} \mathit{objective}(\alpha,\beta,\gamma)$$

1) fix γ_i. Solve $\arg\min_{\alpha,\beta,\gamma_u} \mathit{objective}(\alpha,\beta,\gamma)$

2) fix γ_u. Solve $\arg\min_{\alpha,\beta,\gamma_i} \mathit{objective}(\alpha,\beta,\gamma)$

3,4,5…) repeat until convergence

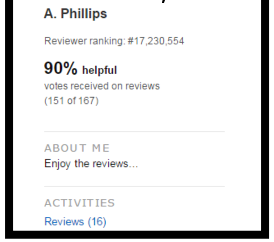
Each of these subproblems is “easy” – just regularized least-squares, like we’ve been doing since we studied regression. This procedure is called **alternating least squares.**

Latent-factor models

Observation: we went from a method which uses **only** features:

$$f(\text{user features}, \text{movie features}) \rightarrow \text{star rating}$$

User features: age, gender, location, etc.



A. Phillips

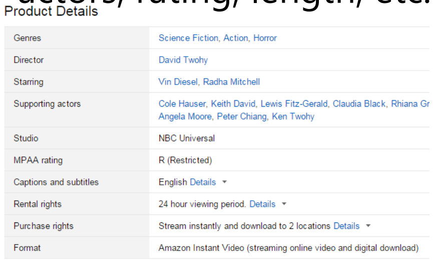
Reviewer ranking: #17,230,554

90% **helpful**
votes received on reviews
(151 of 167)

ABOUT ME
Enjoy the reviews...

ACTIVITIES
Reviews (16)

Movie features: genre, actors, rating, length, etc.



Product Details

Genres	Science Fiction, Action, Horror
Director	David Twohy
Starring	Vin Diesel, Radha Mitchell
Supporting actors	Cole Hauser, Keith David, Lewis Fitz-Gerald, Claudia Black, Rhiana Gr Angela Moore, Peter Chiang, Ken Twohy
Studio	NBC Universal
MPAA rating	R (Restricted)
Captions and subtitles	English Details ▾
Rental rights	24 hour viewing period. Details ▾
Purchase rights	Stream instantly and download to 2 locations Details ▾
Format	Amazon Instant Video (streaming online video and digital download)

to one which **completely ignores** them:

$$\arg\min_{\alpha,\beta,\gamma} \sum_{u,i} (\alpha+\beta_u+\beta_i+\gamma_u\cdot\gamma_i-R_{u,i})^2+\lambda\left[\sum_u \beta_u^2+\sum_i \beta_i^2+\sum_i \|\gamma_i\|_2^2+\sum_u \|\gamma_u\|_2^2\right]$$

Latent-factor models

Should we use features or not?
1) Argument **against** features:

In principle, the addition of features adds **no expressive power** to the model. We **could** have a feature like "is this an action movie?", but if this feature were useful, the model would "discover" a latent dimension corresponding to action movies, and we wouldn't need the feature anyway

In the limit, this argument is valid: as we add more ratings per user, and more ratings per item, the latent-factor model should automatically discover any useful dimensions of variation, so the influence of observed features will disappear

Latent-factor models

Should we use features or not?
2) Argument **for** features:

But! Sometimes we **don't** have many ratings per user/item

Latent-factor models are next-to-useless if **either** the user or the item was never observed before

$$f(u,i) = \boxed{\alpha + \beta_u + \beta_i} + \gamma_u \cdot \gamma_i$$

reverts to zero if we've never seen the user before (because of the regularizer)

Latent-factor models

Should we use features or not?
2) Argument **for** features:

This is known as the **cold-start** problem in recommender systems. Features are not useful if we have many observations about users/items, but are useful for **new** users and items.

We also need some way to handle users who are **active**, but don't necessarily rate anything, e.g. through **implicit feedback**

Overview & recap

Recently we've followed the programme below:

1. Measuring similarity between users/items for **binary** prediction (e.g. Jaccard similarity)
2. Measuring similarity between users/items for **real-valued** prediction (e.g. cosine/Pearson similarity)
3. Dimensionality reduction for **real-valued** prediction (latent-factor models)
4. **Finally** – dimensionality reduction for **binary** prediction

Learning Outcomes

- Completed our presentation of the latent factor model
- Revisited the relationship between recommendation and other types of learning

Web Mining and Recommender Systems

One-class recommendation

Learning Goals

- (Briefly) discuss how latent factor models might be adapted for interaction data (advanced)
- Summarize our discussion of recommender systems so far

One-class recommendation

How can we use **dimensionality reduction** to predict **binary** outcomes?

- Previously we saw **regression** and **logistic regression.** These two approaches use the same type of linear function to predict real-valued and binary outputs
- We can apply an analogous approach to binary recommendation tasks

This is referred to as **“one-class”** recommendation

One-class recommendation

Suppose we have binary (0/1) observations (e.g. purchases) or pos./neg. feedback (thumbs-up/down)

$$R = \begin{pmatrix} 1 & 0 & \cdots & 1 \\ 0 & 0 & & 1 \\ \vdots & & \ddots & \vdots \\ 1 & 0 & \cdots & 1 \end{pmatrix} \text{ or } \begin{pmatrix} -1 & ? & \cdots & 1 \\ ? & ? & & -1 \\ \vdots & & \ddots & \vdots \\ 1 & ? & \cdots & -1 \end{pmatrix}$$

purchased didn't purchase liked didn't evaluate didn't like

One-class recommendation

So far, we've been fitting functions of the form

$$R \simeq UV^T$$

- Let's change this so that we maximize the **difference** in predictions between positive and negative items
- E.g. for a user who likes an item *i* and dislikes an item *j* we want to maximize:

$$\max \ln \sigma(\gamma_u \cdot \gamma_i - \gamma_u \cdot \gamma_j)$$

One-class recommendation

We can think of this as maximizing the probability of correctly predicting pairwise preferences, i.e.,

$$p(i \text{ is preferred over } j) = \sigma(\gamma_u \cdot \gamma_i - \gamma_u \cdot \gamma_j)$$

- As with logistic regression, we can now maximize the likelihood associated with such a model by gradient ascent
- In practice it isn’t feasible to consider all pairs of positive/negative items, so we proceed by stochastic gradient ascent – i.e., randomly sample a (positive, negative) pair and update the model according to the gradient w.r.t. that pair

One-class recommendation

$$\max \ln \sigma(\underbrace{\gamma_u \cdot \gamma_i - \gamma_u \cdot \gamma_j}_{x_{uij}})$$

$$\ln\left(\frac{1}{1+e^{-x_{uij}}}\right)$$

$$\text{obj} \quad \sum_{uij \in T} \underbrace{\ln\left(\frac{1}{1+e^{-x_{uij}}}\right)}_{-\ln\left(1+e^{-x_{uij}}\right)}$$

$$\frac{\partial \text{obj}}{\partial \gamma_{uk}} = \sum_{ij \in I_u} \frac{(\gamma_{ik}-\gamma_j)e^{-x_{uij}}}{1+e^{-x_{uij}}}$$

Summary

Recap

1. Measuring similarity between users/items for **binary** prediction
Jaccard similarity
2. Measuring similarity between users/items for **real-valued** prediction
cosine/Pearson similarity
3. Dimensionality reduction for **real-valued** prediction
latent-factor models
4. Dimensionality reduction for **binary** prediction
one-class recommender systems

References

Further reading:

One-class recommendation:

http://goo.gl/08Rh59

Amazon's solution to collaborative filtering at scale:

http://www.cs.umd.edu/~samir/498/Amazon-Recommendations.pdf

An (expensive) textbook about recommender systems:

http://www.springer.com/computer/ai/book/978-0-387-85819-7

Cold-start recommendation (e.g.):

http://wanlab.poly.edu/recsys12/recsys/p115.pdf

Web Mining and Recommender Systems

Extensions of latent-factor models, (and more on the Netflix prize)

Learning Goals

- Discuss several extensions of the latent factor model
- Further discuss the history of the Netflix Prize

Extensions of latent-factor models

So far we have a model that looks like:

$$f(u,i) = \alpha + \beta_u + \beta_i + \gamma_u \cdot \gamma_i$$

How might we extend this to:

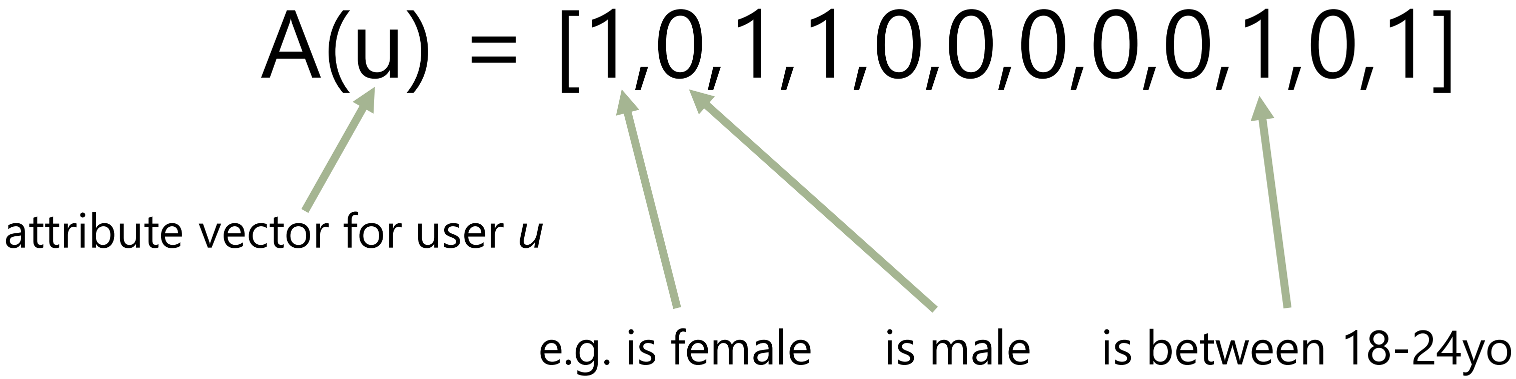
- Incorporate features about users and items
- Handle implicit feedback
- Change over time

See **Yehuda Koren** (+Bell & Volinsky)'s magazine article:
"Matrix Factorization Techniques for Recommender Systems"
IEEE Computer, 2009

Extensions of latent-factor models

1) Features about users and/or items

(simplest case) Suppose we have **binary attributes** to describe users or items



$$A(u) = [1,0,1,1,0,0,0,0,0,1,0,1]$$

attribute vector for user *u*

e.g. is female

is male

is between 18-24yo

Extensions of latent-factor models

1) Features about users and/or items

(simplest case) Suppose we have **binary attributes** to describe users or items

- Associate a **parameter vector** with each attribute
- Each vector encodes how much a particular feature "offsets" the given latent dimensions

$$A(u) = [1,0,1,1,0,0,0,0,0,1,0,1]$$

attribute vector for user *u*

e.g. y_0 = [-0.2,0.3,0.1,-0.4,0.8]
~ "how does being male impact gamma_u"

Extensions of latent-factor models

1) Features about users and/or items

(simplest case) Suppose we have **binary attributes** to describe users or items

- Associate a **parameter vector** with each attribute
- Each vector encodes how much a particular feature "offsets" the given latent dimensions
- Model looks like:

$$f(u,i) = \alpha + \beta_u + \beta_i + (\gamma_u + \textstyle\sum_{a \in A(u)} \rho_a) \cdot \gamma_i$$

- Fit as usual:

$$\arg\min_{\alpha,\beta,\gamma,\rho} \textstyle\sum_{u,i \in \text{train}} \underbrace{(f(u,i) - r_{u,i})^2}_{\text{error}} + \underbrace{\lambda \Omega(\beta, \gamma)}_{\text{regularizer}}$$

Extensions of latent-factor models

2) Implicit feedback

Perhaps many users will never actually rate things, but may still interact with the system, e.g. through the movies they view, or the products they purchase (but never rate)

- Adopt a similar approach – introduce a binary vector describing a user's actions

$$N(u) = [1,0,0,0,1,0,....,0,1]$$

implicit feedback vector for user *u*

e.g. y_0 = [-0.1,0.2,0.3,-0.1,0.5]
Clicked on "Love Actually" but didn't watch

Extensions of latent-factor models

2) Implicit feedback

Perhaps many users will never actually rate things, but may still interact with the system, e.g. through the movies they view, or the products they purchase (but never rate)

- Adopt a similar approach – introduce a binary vector describing a user's actions
- Model looks like:

$$f(u,i) = \alpha + \beta_u + \beta_i + (\gamma_u + \frac{1}{\underset{\uparrow}{\|N(u)\|}} \textstyle\sum_{a \in N(u)} \rho_a) \cdot \gamma_i$$

normalize by the number of actions the user performed

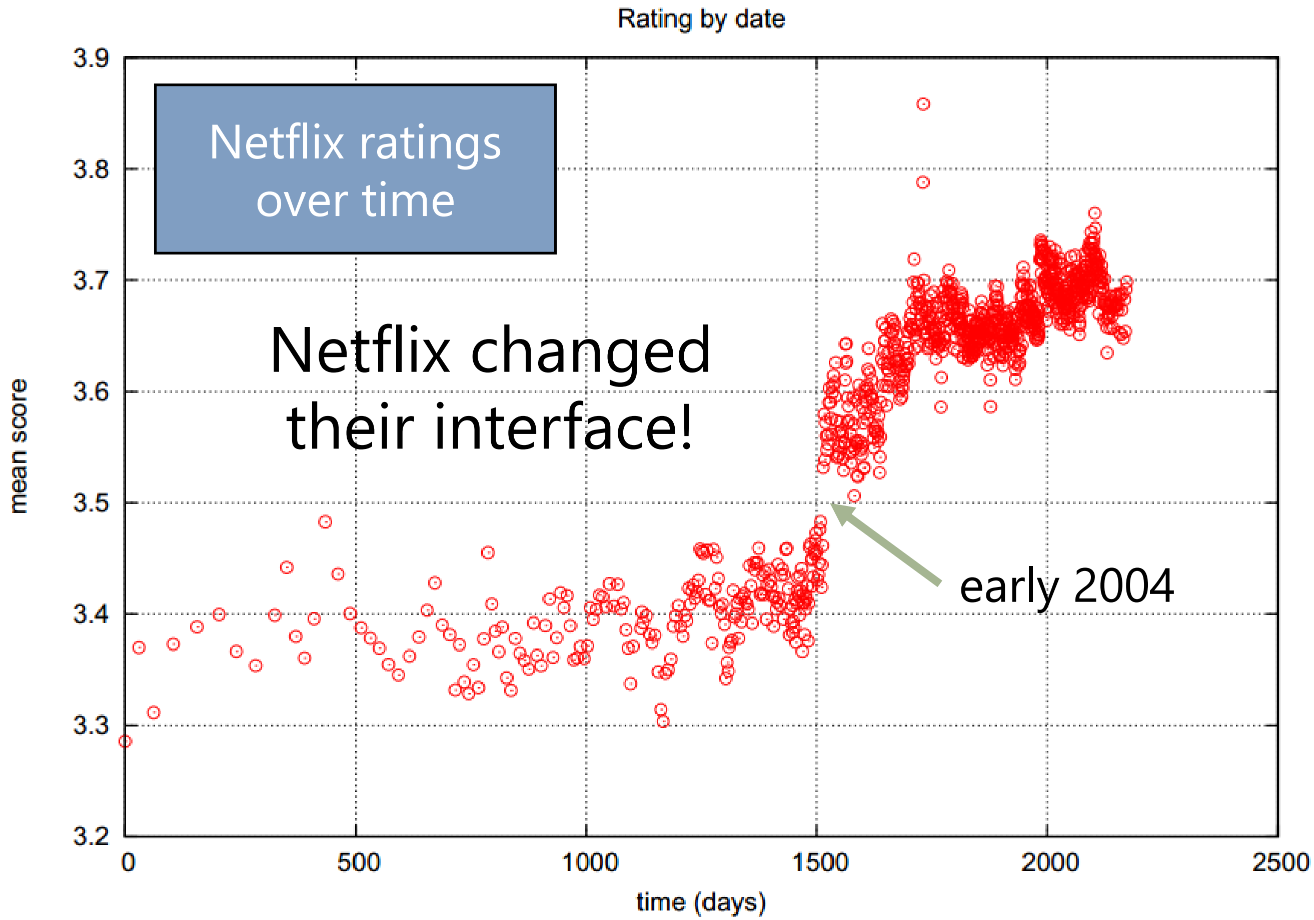
Extensions of latent-factor models

3) Change over time

There are a number of reasons why rating data might be subject to temporal effects…

Extensions of latent-factor models

3) Change over time



Netflix ratings over time

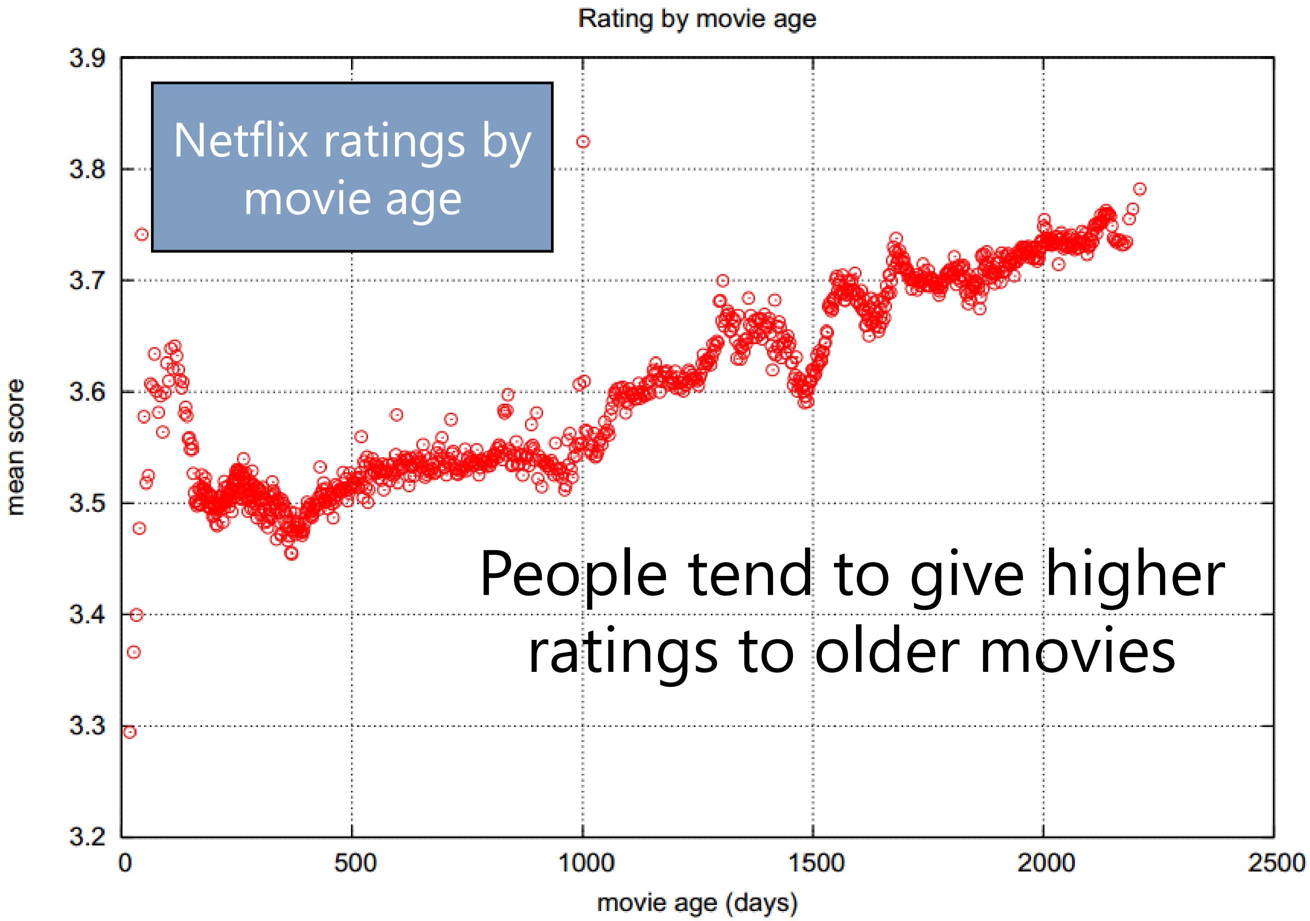
Netflix changed their interface!

early 2004

Figure from Koren: “Collaborative Filtering with Temporal Dynamics” (KDD 2009)

Extensions of latent-factor models

3) Change over time



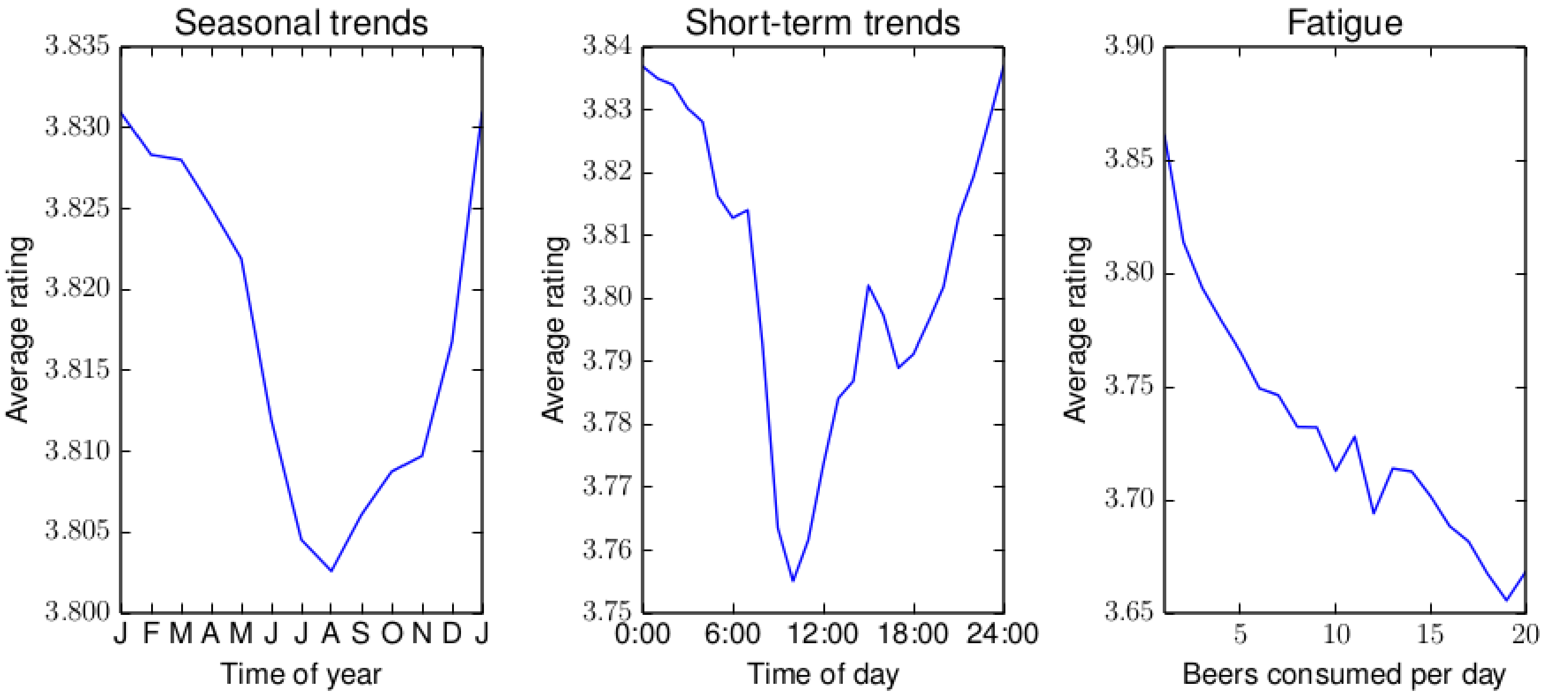
Netflix ratings by movie age

People tend to give higher ratings to older movies

Figure from Koren: “Collaborative Filtering with Temporal Dynamics” (KDD 2009)

Extensions of latent-factor models

3) Change over time

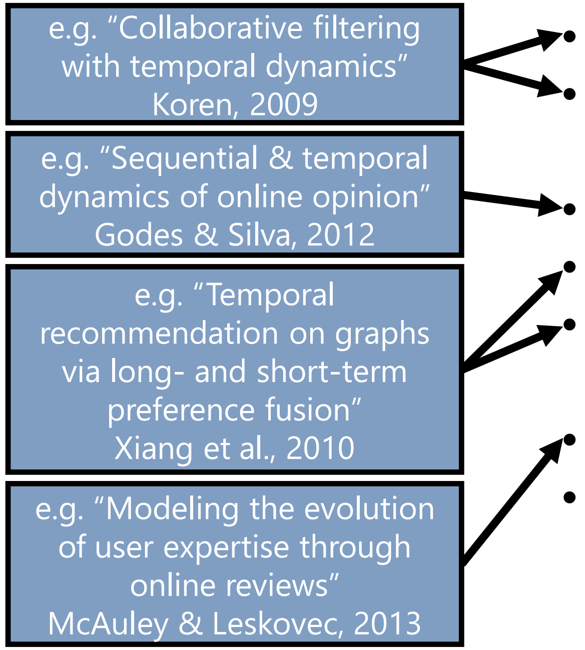


A few temporal effects from beer reviews

Extensions of latent-factor models

3) Change over time

There are a number of reasons why rating data might be subject to temporal effects…



e.g. "Collaborative filtering with temporal dynamics" Koren, 2009

e.g. "Sequential & temporal dynamics of online opinion" Godes & Silva, 2012

e.g. "Temporal recommendation on graphs via long- and short-term preference fusion" Xiang et al., 2010

e.g. "Modeling the evolution of user expertise through online reviews" McAuley & Leskovec, 2013

- Changes in the interface
- People give higher ratings to older movies (or, people who watch older movies are a biased sample)
- The community's preferences gradually change over time
- My girlfriend starts using my Netflix account one day
- I binge watch all 144 episodes of buffy one week and then revert to my normal behavior
- I become a "connoisseur" of a certain type of movie
- Anchoring, public perception, seasonal effects, etc.

Extensions of latent-factor models

3) Change over time

Each definition of temporal evolution demands a slightly different model assumption (we'll see some in more detail later tonight!) but the basic idea is the following:

1) Start with our original model:

$$f(u,i) = \alpha + \beta_u + \beta_i + \gamma_u \cdot \gamma_i$$

2) And define some of the parameters as a function of time:

$$f(u,i,t) = \alpha + \beta_u(t) + \beta_i(t) + \gamma_u(t) \cdot \gamma_i$$

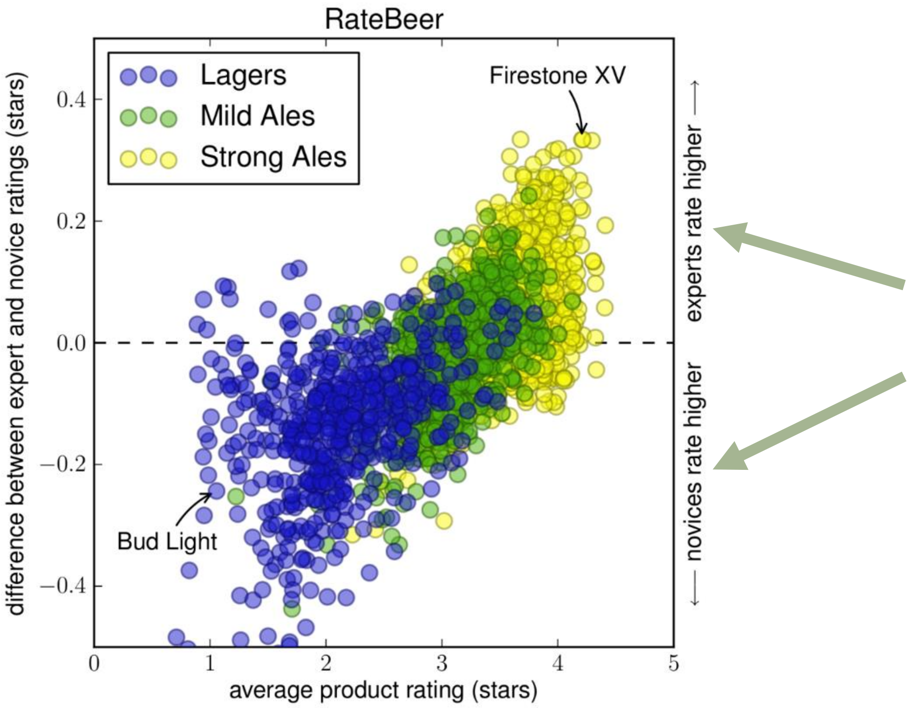
3) Add a regularizer to constrain the time-varying terms:

$$\arg\min_{\alpha,\beta,\gamma} \sum_{u,i,t \in \text{train}} (f(u,i,t) - r_{u,i,t})^2 + \lambda_1 \Omega(\beta,\gamma) + \underbrace{\lambda_2 \|\gamma(t) - \gamma(t+\delta)\|}_{\text{parameters should change smoothly}}$$

Extensions of latent-factor models

3) Change over time

Case study: how do people acquire tastes for beers (and potentially for other things) over time?



Differences between "beginner" and "expert" preferences for different beer styles

Extensions of latent-factor models

4) Missing-not-at-random

- Our decision about whether to purchase a movie (or item etc.) is a function of how we **expect** to rate it
- Even for items we've purchased, our decision to **enter a rating** or write a review **is a function of our rating**
  - e.g. some rating distribution from a few datasets:

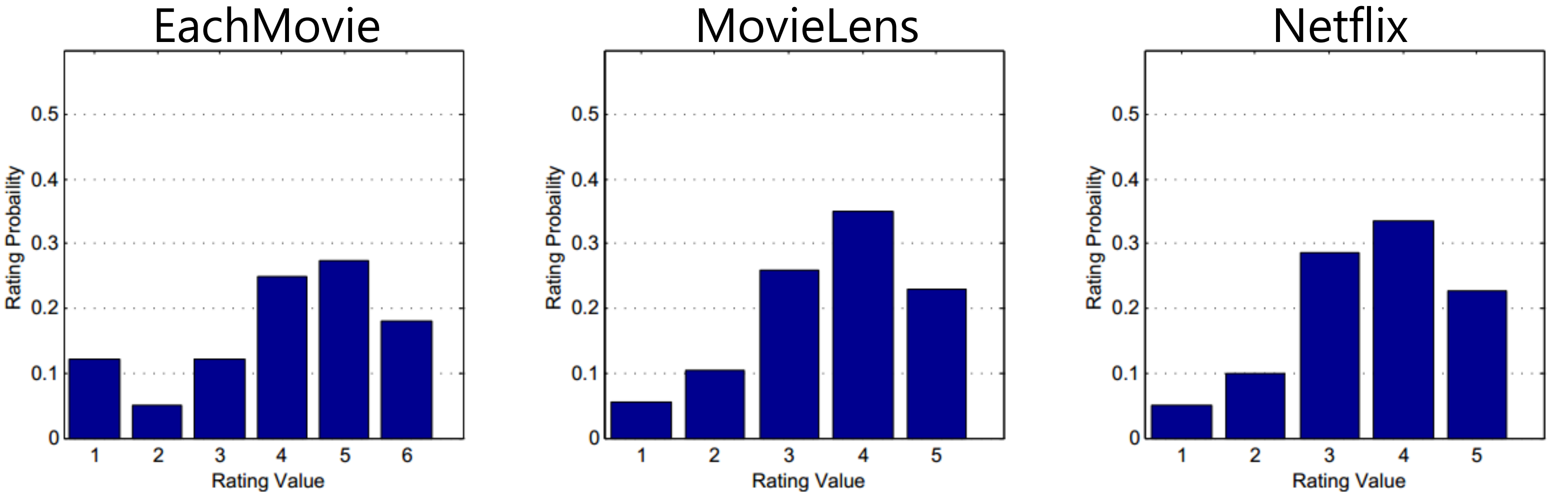
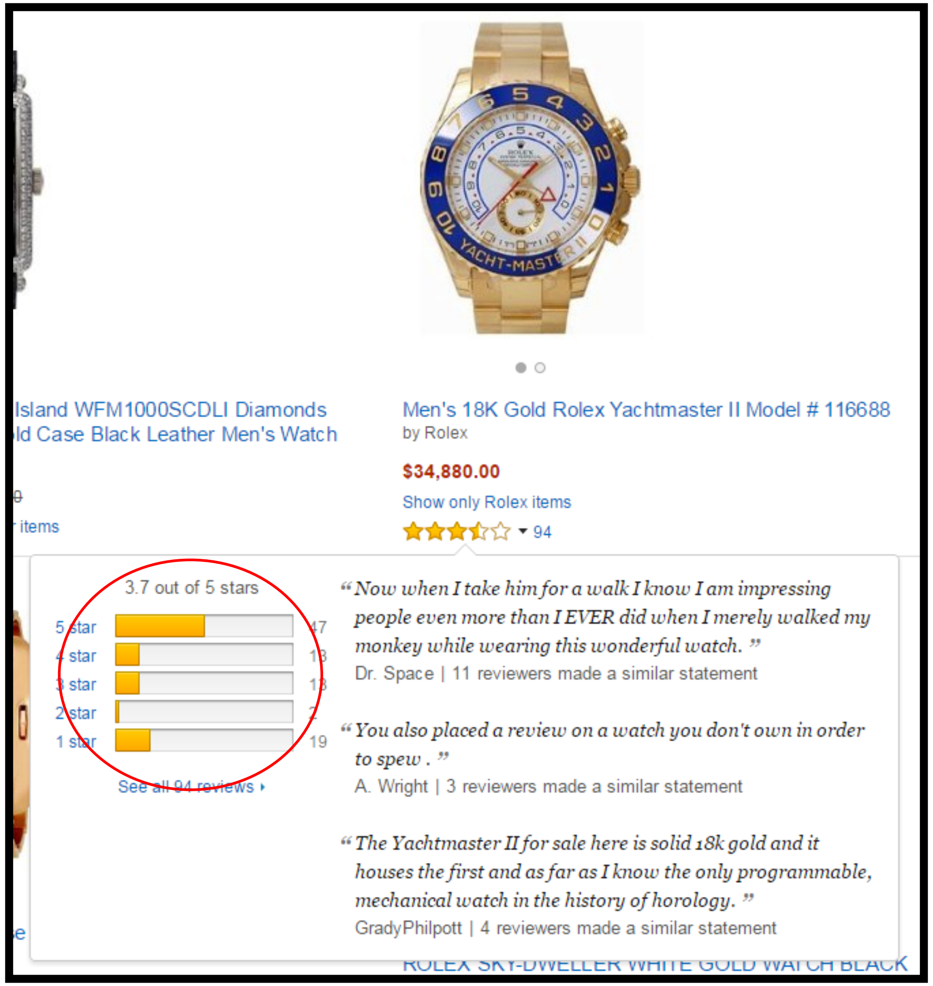


Figure from Marlin et al. "Collaborative Filtering and the Missing at Random Assumption" (UAI 2007)

Extensions of latent-factor models

4) Missing-not-at-random

e.g. Men's watches:



Island WFM1000SCDLI Diamonds
ld Case Black Leather Men's Watch

items

Men's 18K Gold Rolex Yachtmaster II Model # 116688
by Rolex

$34,880.00

Show only Rolex items

★★★½☆ ▾ 94

3.7 out of 5 stars

5 star	47
4 star	13
3 star	13
2 star	2
1 star	19

See all 94 reviews ›

“Now when I take him for a walk I know I am impressing people even more than I EVER did when I merely walked my monkey while wearing this wonderful watch.”
Dr. Space | 11 reviewers made a similar statement

“You also placed a review on a watch you don't own in order to spew .”
A. Wright | 3 reviewers made a similar statement

“The Yachtmaster II for sale here is solid 18k gold and it houses the first and as far as I know the only programmable, mechanical watch in the history of horology.”
GradyPhilpott | 4 reviewers made a similar statement

ROLEX SKY-DWELLER WHITE GOLD WATCH BLACK

Extensions of latent-factor models

4) Missing-not-at-random

- Our decision about whether to purchase a movie (or item etc.) is a function of how we **expect** to rate it
- Even for items we've purchased, our decision to **enter a rating** or write a review **is a function of our rating**
- So we can predict ratings more accurately by building models that account for these differences

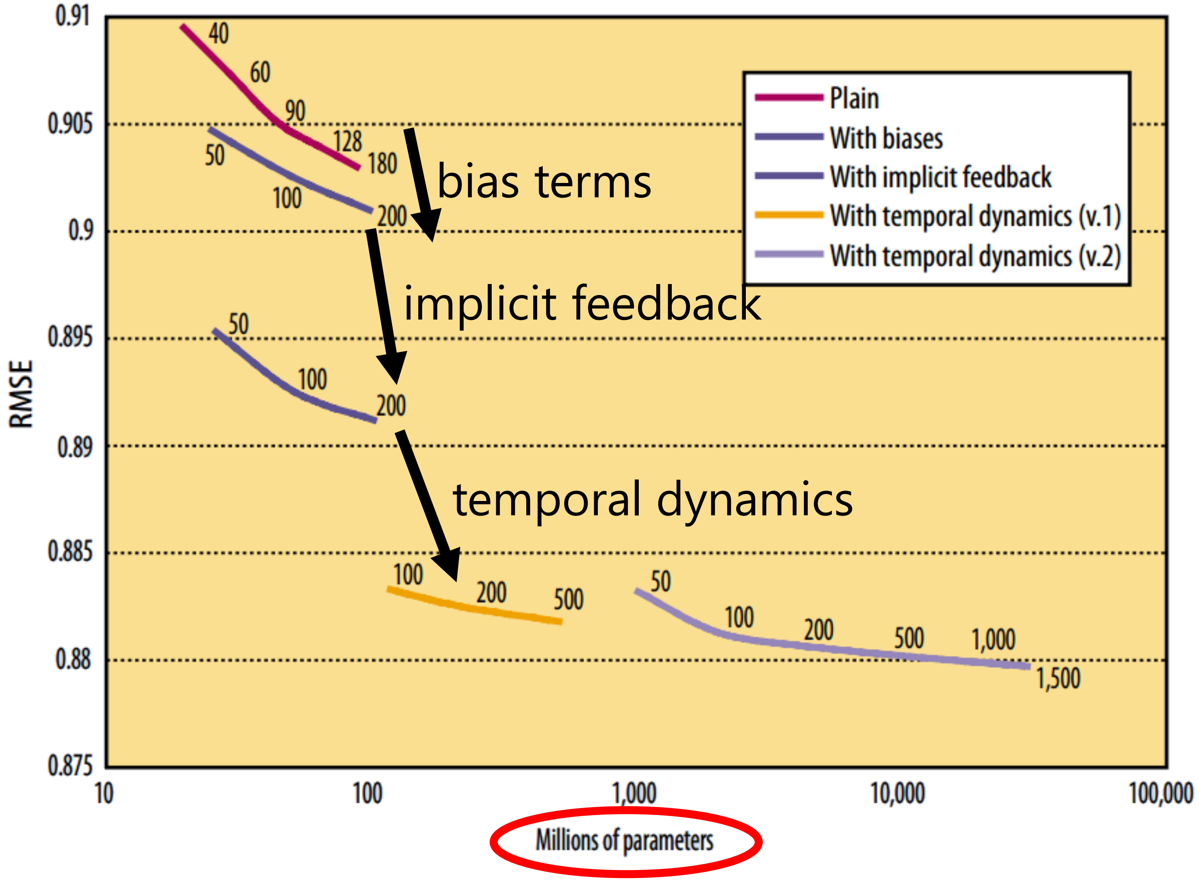
1. Not-purchased items have a different prior on ratings than purchased ones
2. Purchased-but-not-rated items have a different prior on ratings than rated ones

Figure from Marlin et al. "Collaborative Filtering and the Missing at Random Assumption" (UAI 2007)

Moral(s) of the story

How much do these extension help?

Moral: increasing complexity helps a bit, but changing the model can help **a lot**



bias terms

implicit feedback

temporal dynamics

Figure from Koren: "Collaborative Filtering with Temporal Dynamics" (KDD 2009)

Moral(s) of the story

So what actually happened with Netflix?

- The AT&T team "BellKor", consisting of Yehuda Koren, Robert Bell, and Chris Volinsky were early leaders. Their main insight was how to effectively incorporate temporal dynamics into recommendation on Netflix.
- Before long, it was clear that no one team would build the winning solution, and Frankenstein efforts started to merge. Two frontrunners emerged, "BellKor's Pragmatic Chaos", and "The Ensemble".
- The BellKor team was the first to achieve a 10% improvement in RMSE, putting the competition in "last call" mode. The winner would be decided after 30 days.
- After 30 days, performance was evaluated on the hidden part of the test set.
- Both of the frontrunning teams had **the same** RMSE (up to some precision) but BellKor's team submitted their solution 20 minutes earlier and won $1,000,000

For a less rough summary, see the Wikipedia page about the Netflix prize, and the nytimes article about the competition: http://goo.gl/WNpy7o

Moral(s) of the story

Afterword

- Netflix had a class-action lawsuit filed against them after somebody de-anonymized the competition data
- $1,000,000 seems to be **incredibly cheap** for a company the size of Netflix in terms of the amount of research that was devoted to the task, and the potential benefit to Netflix of having their recommendation algorithm improved by 10%
- Other similar competitions have emerged, such as the Heritage Health Prize ($3,000,000 to predict the length of future hospital visits)

- But… the winning solution never made it into production at Netflix – it's a monolithic algorithm that is very expensive to update as new data comes in*

*source: a friend of mine told me and I have no actual evidence of this claim

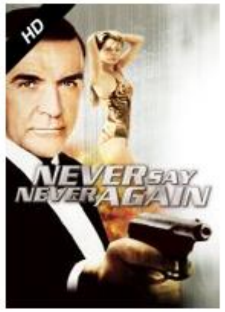
Moral(s) of the story

Finally…

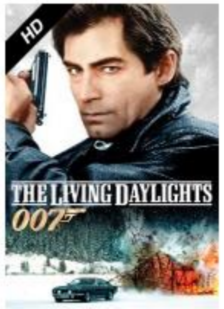
Q: Is the RMSE really the right approach? Will improving rating prediction by 10% actually improve the user experience by a significant amount?
A: Not clear. Even a solution that only changes the RMSE slightly could drastically change which items are top-ranked and ultimately suggested to the user.
Q: But… are the following recommendations actually any good?
A1: Yes, these are my favorite movies!
or **A2:** No! There's no **diversity**, so how will I discover **new** content?

5.0 stars

5.0 stars



5.0 stars



5.0 stars

4.9 stars

4.9 stars

4.8 stars

4.8 stars

predicted rating

Summary

Various extensions of latent factor models:

- Incorporating features

e.g. for cold-start recommendation

- Implicit feedback

e.g. when ratings aren't available, but other actions are

- Incorporating temporal information into latent factor models

seasonal effects, short-term "bursts", long-term trends, etc.

- Missing-not-at-random

incorporating priors about items that were not bought or rated

- The Netflix prize

Learning Outcomes

- Discussed several extensions of latent factor models
- Described what types of solutions worked on the Netflix Prize
- Thought about potential limitations of the solutions we've seen so far

References

Further reading:

Yehuda Koren's, Robert Bell, and Chris Volinsky's IEEE computer article:
http://www2.research.att.com/~volinsky/papers/ieeecomputer.pdf
Paper about the "Missing-at-Random" assumption, and how to address it:
http://www.cs.toronto.edu/~marlin/research/papers/cfmar-uai2007.pdf
Collaborative filtering with temporal dynamics:
http://research.yahoo.com/files/kdd-fp074-koren.pdf
Recommender systems and sales diversity:
http://papers.ssrn.com/sol3/papers.cfm?abstract_id=955984