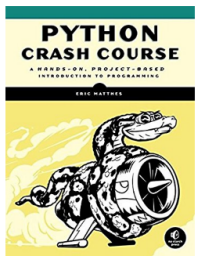
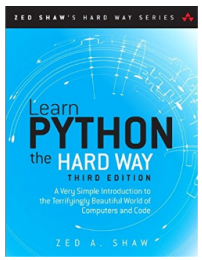
ENGR 102 PROGRAMMING PRACTICE

WEEK 7

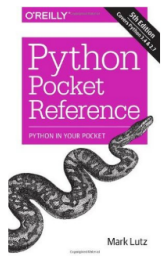
Matching Products



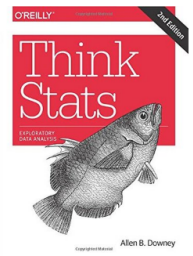
Python Crash Course: A Hands-On, Project-Based Introduction to…
› Eric Matthes
143
#1 Best Seller in Children's Programming Books



Learn Python the Hard Way: A Very Simple Introduction to the…
Zed A. Shaw
129
Paperback
$33.05 Prime



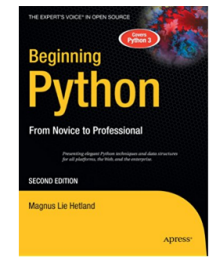
Python Pocket Reference: Python In Your Pocket (Pocket Reference…
› Mark Lutz
141
Paperback
$10.25 Prime



Think Stats: Exploratory Data Analysis
Allen B. Downey
10
Paperback
$29.27 Prime

Think Bayes: Bayesian Statistics in Python
Allen B. Downey
21
Paperback
$28.49 Prime

Think Complexity: Complexity Science and Computational Modeling
Allen B. Downey
20
Paperback
$31.67 Prime



Beginning Python: From Novice to Professional, 2nd Edition (The Experts…
› Magnus Lie Hetland
43
Paperback
$25.43 Prime

- Now you know how to find similar people and recommend products.
- What if you want to see which products are similar to each other?
- How would you do it?

Matching Products

- Determine similarity by
  - looking at **who liked** a particular **item** and
    looking at **what items a particular person liked**
  - seeing the other **things they liked**.
    seeing the other **people** who **liked** the same **things**

Matching Products

`transformPrefs(...)`

- Just need to swap the people (i.e., critics) and the items (i.e., movies).

```
    {'Lisa Rose': {'Lady in the Water': 2.5, 'Snakes on a Plane': 3.5},
    'Gene Seymour': {'Lady in the Water': 3.0, 'Snakes on a Plane': 3.5}}
```

to:

```
    {'Lady in the Water':{'Lisa Rose':2.5,'Gene Seymour':3.0},
    'Snakes on a Plane':{'Lisa Rose':3.5,'Gene Seymour':3.5}} etc..
```

Matching Products

`transformPrefs(...)`

- The following function performs the necessary transformation:

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

Matching Products

transformPrefs(...)

```
from recommendations import *

movies = transformPrefs(critics)

print topMatches(movies, 'Superman Returns')
```

[(0.68, 'Snakes on a Plane'),
(0.66, 'You, Me, and Dupree'),
(0.49, 'Lady in the Water')]

Matching Products

Another Motivation

- An online retailer might collect purchase histories for the purpose of recommending products to individuals.
- Switching the products and people would allow to search for people who might buy certain products.
  - *planning a marketing effort for a big clearance day.*
- New links on a link-recommendation site are seen by the people who are most likely to enjoy them.

Recommending Watchers for Movies

```
from recommendations import *

movies = transformPrefs(critics)

print getRecommendations(movies, 'Just My Luck')
```

Recommendation Scalability Considerations

- Our recommendation engine requires the use of all the rankings from every user in order to create a dataset.
- OK for a few thousand people or items
  - how about for a very large site like Amazon, which has millions of customers and products?

Item-Based Filtering

An alternative Approach

- So far, we have seen **user-based** collaborative filtering.
- An alternative is known as **item-based** collaborative filtering.
- In cases with very large datasets, it allows many of the calculations to be performed in advance so that a user needing recommendations can get them more quickly.

Item-Based Filtering

An alternative Approach

- Approach:
  - precompute the most similar items for each item.
- To make recommendations to a user:
  - look at his top-rated items and
  - create a weighted list of the items most similar to those.

Item-Based Filtering

An alternative Approach

- The first step still requires you to examine all the data. How is this approach more efficient?
  - comparisons between items will not change as often as comparisons between users.
- So what?
  - you do not have to continuously calculate each item's most similar items.
  - do it once, and reuse multiple times.
  - Do it at low-traffic times or on a computer separate from your main application.

calculateSimilarItems

- Write a function calculateSimilarItems that
  - takes as an input
    - dictionary *itemprefs* of critics along with items they rated
    - integer *n*,
  - returns a dictionary of items
    - each mapped to an array containing top *n* most similar items

Example

```
print calculateSimilarItems(critics, 2)
```

```
{   'Just My Luck': [   (0.35, 'Lady in the Water'),
                        (0.32, 'You, Me, and Dupree')],
    'Lady in the Water': [   (0.45, 'You, Me, and Dupree'),
                             (0.39, 'The Night Listener')],
    ...
}
```

Step 1: Calculate Similar Items

calculateSimilarItems(...)

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

    # Invert the preference matrix to be item-centric
    itemPrefs = transformPrefs(prefs)

    for item in itemPrefs:
        # Find the most similar items to this one
        scores = topMatches(itemPrefs, item, n, sim)
        result[item] = scores

    return result
```

Step 1: Calculate Similar Items `calculateSimilarItems(...)`

```
from recommendations import *
itemsim = calculateSimilarItems(critics)
```

- This function only has to be run frequently enough to keep the item similarities up-to-date.
- Run it more often early on when the user base and number of ratings is small
  - *as the user base grows, the similarity scores between items will usually become more stable.*

Step 2: Get Recommendations

- Approach (Algorithm)
  - Get all the movies/items that the user has rated.
  - Find the similar movies/items.
  - Weigh them according to how similar they are.
- Unlike our previous approach, the critics are not involved at all
  - Instead, there is a grid of

    movies I have watched and rated vs.
    movies I have not watched.

Step 2: Get Recommendations

Table 2-3. Item-based recommendations for Toby

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

Step 2: Get Recommendations

Table 2-3. Item-based recommendations for Toby

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

Step 2: Get Recommendations

Table 2-3. Item-based recommendations for Toby

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

Step 2: Get Recommendations

Table 2-3. Item-based recommendations for Toby

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

Step 2: Get Recommendations

Table 2-3. Item-based recommendations for Toby

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

Step 2: Get Recommendations

getRecommendedItems(...)

```
def getRecommendedItems(prefs, itemSim, user):
  userRatings = prefs[user]
  scores = {}
  totalSim = {}

  # Loop over items rated by this user
  for item, rating in userRatings.items( ):
    # Loop over items similar to this one
    for similarity, item2 in itemSim[item]:
      # Ignore if this user has already rated this item
      if item2 in userRatings: continue

      # Weighted sum of rating times similarity
      scores.setdefault(item2, 0)
      scores[item2] += similarity * rating

      # Sum of all the similarities
      totalSim.setdefault(item2, 0)
      totalSim[item2] += similarity

  # Divide each total score by total weighing to get an average
  rankings = [(score/totalSim[item], item) for item, score in scores.items()]
  rankings.sort(reverse=True)

  return rankings
```

Step 2: Get Recommendations

`getRecommendedItems(...)`

```
from recommendations import *

itemsim = calculateSimilarItems(critics)

print getRecommendedItems(critics, itemsim, 'Toby')
```

User-Based or Item-Based Filtering?

- Data Set Size
  - Large datasets: Item-based filtering is significantly faster than user-based
- What about the item similarity table?
  - Sparsity: In the movie data, since every critic has rated nearly every movie, the dataset is dense (not sparse).
  - Item-based filtering usually outperforms user-based filtering in ***sparse*** datasets (in terms of accuracy)
  - They perform about equally in **dense** datasets.