Compsci 101
Stable Sorting, Recommender

Susan Rodger
November 17, 2022

U is for …

• URL
  • https://duke.edu

• Usenet
  • Original source of FAQ, Flame, Spam, more

• UI and UX
  • User is front and center

John Riedl

• Co-Inventor of Recommender systems
• PhD at Purdue University
• Professor at Univ. of Minnesota
• ACM Software System Award – GroupLens System
• Died of cancer in 2013

• Quote from his son about John:
  “He once looked into how likely people are to follow your book recommendations based on how many books you recommend. We went to his talk at the AH Conference in which he described the answer. It turns out that if you recommend too many books to people, they get overwhelmed and are less likely to follow your suggestions. As he told us in his talk, the optimal number of books to recommend turns out to be about two. Then he proceeded to recommend eight books during the talk.”

Announcements

• APT-6 due Tuesday, Nov 29
• Assignment 5 Clever GuessWord due tonight
• Lab 9 Friday
  • There is a prelab!
• No reading or QZ until last week of classes
• Exam 3 is December 1
Exam 3 – in person – Thurs, Dec 1

- Exam is in class on paper – 10:15am
  - Need pen or pencil
- See materials under 12/1 date
  - Exam 3 Reference sheet - part of exam
- Covers
  - topics /reading through today
  - APTs through APT6
  - Labs through Lab 9
  - Assignments through Assignment 5, parts of Assign 6 helpful

Exam 3 topics include ...

- List, tuples, list comprehensions
- Loops – for loop, while loop, indexing with a loop
- Reading from a file
  - Converting data into a list of things
- Parallel lists
- Sets – solving problems
- Dictionaries – solving problems
- Sorting – lists, tuples
- No turtles, no images - but note we are practicing other concepts with images

Exam 3

- Exam 3 is your own work!
- No looking at other people’s exam
- You cannot use any notes, books, computing devices, calculators, or any extra paper
- Bring only a pen or pencil
- The exam has extra white space and has the Exam 3 reference sheet as part of the exam.
- Do not discuss any problems on the exam with others until it is handed back

Exam 3 – How to Study

- Practice writing code on paper!
- Rewrite an APT
- Try to write code from lecture from scratch
- Try to write code from lab from scratch
- Practice from old exams
- Put up old Sakai quizzes, but better to practice writing code
- Look at Exam 3 reference sheet when writing code!
Recommendation Systems: Yelp

- Are all users created equal?
- Weighting reviews
- What is a recommendation?

https://www.youtube.com/watch?v=PniMEmM89iY
Recommender Systems: Amazon

• How does Amazon create recommendations?

Recommendation Systems: Netflix

• Netflix offered a prize in 2009
  • Beat their system? Win a million $$
  • http://nyti.ms/sPvR

Compsci 101 Recommender

• Doesn’t work at the scale of these systems, uses publicly accessible data, but …
  • Movie data, food data, book data

• Make recommendations
  • Based on ratings, how many stars there are
  • Based on weighting ratings by users like you!

• Collaborative Filtering: math, stats, compsci

Machine learning!
Where to eat? Simple Example

<table>
<thead>
<tr>
<th>Tandoor</th>
<th>IlForno</th>
<th>McDon</th>
<th>Loop</th>
<th>Panda</th>
<th>Twin</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>3</td>
<td>5</td>
<td>0</td>
<td>-3</td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>-3</td>
</tr>
<tr>
<td>-3</td>
<td>3</td>
<td>3</td>
<td>5</td>
<td>1</td>
<td>-1</td>
</tr>
</tbody>
</table>

- Rate restaurants on a scale of (low) -5 to 5 (high)
  - Each row is one user’s ratings
  - But a zero/0 means no rating, not ambivalent
- What restaurant should I choose to go to?
  - What do the ratings say? Let’s take the average!

Calculating Averages

- What is average rating of eateries?
  - Tandoor: \((1 + -3)/2 = -1.00\)
  - Don’t count rating if not rated

- Il Forno: \((3 + 1 + 3)/3 = 2.33\)
- Where should we eat? What’s the best average
- Highest: \(8/2 = 4\) for McDonalds and The Loop
Python Specification

- **Items:** list of strings (header in table shown)
  
  ```
  items = ['DivinityCafe', 'FarmSteak', 'IlForno', 'LoopPizzaGrill', 'McDonalds', 'PandaExpress', 'Tandoor', 'TheCommons', 'TheSkillet']
  ```

- **Values in dictionary are ratings:** int list
  
  ```
  len(ratings[i]) == len(items)
  ```

Recommender averages

- **def averages(items, ratings):**

- **Input:** items -- list of restaurants/strings
- **Input:** dictionary of rater-name to ratings
  
  - ratings: list of ints, [1, 0, -1, ... 1] -- parallel list to list of restaurants
  
  - kth rating maps to kth restaurant

- **Output:** recommendations
  
  - List of tuples (name, avg rating) or (str, float)
  
  - Sort by rating from high to low

WOTO-1 Averages

Drawbacks of Averaging, Instead ...

- Are all user’s ratings the same to me?
  - Weight/value ratings of people most similar to me

- Collaborative Filtering
  - How do we determine who is similar to/"near" me?

- Mathematically: treat ratings as vectors in an N-dimensional space, \( N = \# \) of items that are rated
  - a.k.a. weight has higher value \( \rightarrow \) closer to me

Determining "closeness"

- Calculate a number measuring closeness to me
  - The higher the number, the closer to me
  - I’m also a rater, "me" is parameter to function

- Function:
  - \( \text{similarities("rodger", ratings)} \)

- Determine "closeness"
  - Calculate a number measuring closeness to me
  - The higher the number, the closer to me
  - I’m also a rater, "me" is parameter to function
  - Function:
    - \( \text{similarities("rodger", ratings)} \)

- Return \([("rater1", #), ("rater2", #), \ldots]\)
  - List of tuples based on closeness to me
  - sorted high-to-low by similarity
  - “rodger” should not be in that list!
What's close? Dot Product

  - For \([3,4,2]\) and \([2,1,7]\)
    - \(3 \times 2 + 4 \times 1 + 2 \times 7 = 6 + 4 + 14 = 24\)

- How close am I to each rater?
- What happens if the ratings are
  - Same sign? Me: 3, -2 Other: 2, -5
  - Different signs? Me: -4 Other: 5
  - One is zero? Me: 0 Other: 4
- What does it mean when # is...
  - Big? Small? Negative?

We agree!

We disagree!

These cancel out!

What does it mean when # is...

Writing similarities

- Given a name and a dictionary, return list of tuples

```python
def similarities(name, ratings):
    return [('name0', #), ...('nameN', #)]
```

- What is the # here?
  - Dot product of two lists
  - One list is fixed (name)
  - Other list varies (loop)

- Think: How many tuples are returned?
Collaborative Filtering

• Once we know raters "near" me? Weight them!
  • How many raters to consider? 1? 10?
  • Suppose Fran is [2, 4, 0, 1, 3, 2]
  • What is Sam’s similarity to Fran?

\[
\begin{align*}
\text{Sam} &:& [0, 3, 5, 0, -3, 5] \\
\text{Chris} &:& [1, 1, 0, 3, 0, -3] \\
\text{Nat} &:& [-3, 3, 3, 5, 1, -1]
\end{align*}
\]

What is Chris’s similarity and weights?

• Suppose Fran is [2, 4, 0, 1, 3, 2]
• Chris’s similarity is:

\[
\begin{align*}
\text{Sam} &:& [3, 3, 0, 9, 0, -9] \\
\text{Chris} &:& [3, 3, 0, 9, 0, -9] \\
\text{Nat} &:& [-3, 3, 3, 5, 1, -1]
\end{align*}
\]
Steps for Recommendations

- Start with you, a rater/user and all the ratings
  - Get similarity "weights" for users: dot product
- Calculate new weighted ratings for all users
  - \([\text{weight} \times r \text{ for } r \text{ in ratings}]\)
- Based on these new ratings, find average
  - Don’t use zero-ratings
- Check recommendations by … (not required)
  - Things I like are recommended? If so, look at things I haven’t tried!

Recommendations

- Get new weighted averages for each eatery
  - Then find the best eatery I’ve never been to

```python
def recommendations(name, items, ratings, numUsers):
    return [('eatery0', #), ...('eateryN', #)]
```

Fran gets a recommendation (considering numUsers raters)
```
rc = recommendations("Fran", items, ratings, 3)
#use this to provide evals to Fran
```

Similarities Summarized

- How do we get weighted ratings?

```python
def similarities(name, ratings):
    return [('name', #), ...('name', #)]
```

```python
def recommendations(name, items, ratings, numUsers):
    return [('eatery0', #), ...('eateryN', #)]
```

```
weights = similarities("Fran", ratings)
```

Number of “nearby” raters
Making Recommendations

• How do we get weighted ratings? Call average?

weights = similarities("Fran", ratings)
weights = #slice based on numUsers
weightedRatings = {}. # new dictionary
for person, weight in weights:
    weightedRatings[?] = ?

WOTO-2 Sims to Recs

• From Similarities to Recommendations

Calculating Weighted Average

<table>
<thead>
<tr>
<th></th>
<th>Tandoor</th>
<th>IlForno</th>
<th>McDon</th>
<th>Loop</th>
<th>Panda</th>
<th>Twin</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sam</td>
<td>0</td>
<td>39</td>
<td>65</td>
<td>0</td>
<td>-39</td>
<td>65</td>
</tr>
<tr>
<td>Chris</td>
<td>1</td>
<td>3</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>-3</td>
</tr>
<tr>
<td>Nat</td>
<td>-3</td>
<td>36</td>
<td>36</td>
<td>60</td>
<td>12</td>
<td>-12</td>
</tr>
<tr>
<td>Total</td>
<td>-36</td>
<td>75</td>
<td>101</td>
<td>60</td>
<td>-27</td>
<td>53</td>
</tr>
<tr>
<td>Avg</td>
<td>-36</td>
<td>37.5</td>
<td>50.5</td>
<td>60</td>
<td>-13.5</td>
<td>26.5</td>
</tr>
</tbody>
</table>

recommendations("Fran",items,ratings,2)
• Make recommendation for Fran? Best? Worst?
• Fran should eat at Loop! Even though only using Nat’s rating
• But? Fran has been to Loop! Gave it a 1, ... McDonalds!!!! ??

Assignment Modules

RecommenderEngine
1.averages(...)  
2.similarities(...)  
3.recommendations(...)  

MovieReader
1.getdata(...)  

BookReader
1.getdata(...)  

RecommenderMaker
1.makerecs(...)  

TestRecommender
Assignment Modules

RecommenderEngine
1. averages(…)
2. similarities(…)
3. recommendations(…)

MovieReader
1. getdata(…)

BookReader
1. getdata(…)

RecommenderMaker
1. makerecs(…)

TestRecommender

Function Call Ordering

- Some_Reader_Module.getdata(…)
- RecommenderMaker.makerecs(…)
  - RecommenderEngine.recommendations(…)
  - RecommenderEngine.similarities(…)
  - RecommenderEngine.averages(…)

APT: MedalTable


Start with inner most call and work outwards
Test on your computer and on Gradescope as you go!
Tracking the Data

- What do we need to obtain for each country?
  - What's the data, how do we store it?
  - What's the data, how do we calculate it?

- Method and code to transform input
  - What will we store, how do we initialize/update
  - Verifying we've done this properly

Use a dictionary?

- 3 dictionaries
  - Country to Gold count
  - Country to Silver count
  - Country to Bronze count

Use a dictionary?

- 3 dictionaries
  - Country to Gold count
  - Country to Silver count
  - Country to Bronze count

- 1 dictionary
  - Map country to [gld cnt, sil cnt, bro cnt]
Example

• How would we create a dictionary for:

[“KOR TPE UKR”, “KOR KOR TPE”, “KOR JPN JPN”]

Example: dictionary d

• Process first string: "KOR TPE UKR"
  • d[“KOR”] = [1, 0, 0]  # gold
  • d[“TPE”] = [0, 1, 0]  # silver
  • d[“UKR”] = [0, 0, 1]  # bronze

• Process second string: "KOR KOR TPE"
  • d[“KOR”] = [2, 0, 0]  # gold
  • d[“KOR”] = [2, 1, 0]  # silver
  • d[“TPE”] = [0, 1, 1]  # bronze

Example: dictionary d (2)

• What we have so far:

• Process third string: "KOR JPN JPN"
  • d[“KOR”] = [3, 1, 0]  # gold
  • d[“JPN”] = [0, 1, 0]  # silver
  • d[“JPN”] = [0, 0, 1]  # bronze
Example: dictionary d (2)

- What we have so far:
  - d[“KOR”] = [2, 1, 0]
  - d[“TPE”] = [0, 1, 1]
  - d[“UKR”] = [0, 0, 1]

- Process third string: "KOR JPN JPN"
  - d[“KOR”] = [3, 1, 0]  # gold
  - d[“JPN”] = [0, 1, 0]  # silver
  - d[“JPN”] = [0, 1, 1]  # bronze

Sorting the Data

- Write a helper function to build the dictionary
  - d = build_dict(results)
  - Where results is string of countries for each event
- Use dictionary to get list of tuples
  - [(‘JPN’, [0, 1, 1]), (‘KOR’, [3, 1, 0]), (‘TPE’, [0, 1, 1]), (‘UKR’, [0, 0, 1])]

- Then do passes to sort the data
  - Will discuss sorting the data in lab

Some of the code

```python
def build_dict(data):
    d = {}
    for item in data:
        [gld, sil, bro] = item.split()
```
Some of the code

```python
def build_dict(data):
    d = {}
    for item in data:
        [gld, sil, bro] = item.split()
        if gld not in d:
            d[gld] = [0, 0, 0]
        d[gld][0] += 1
```

If country not in, set to [0, 0, 0] since has no medals counted yet

Update appropriate index, [0] is the gold count