Compsci 101
Recommender

Susan Rodger
April 13, 2023

V is for ...

- Viral Video
  - Husky Dog sings with iPAD – 18 million views
  - [https://www.youtube.com/watch?v=Mk4bmK-acEM](https://www.youtube.com/watch?v=Mk4bmK-acEM)

- Virtual Memory
  - It is and is not there!

- Virtual Reality
  - Augmenting IRL

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 Tonight, April 13
- Assign 6 – Recommender, due Thursday, 4/20
- APT-7 out today, due Tuesday, 4/25
- Assign 7 – out today, due Wed. 4/26
  - But can be turned in by Sunday 4/30 with no penalty
- Lab 10 this week is different!
  - No pre-lab this week
Lab 10 is required, cannot drop it

• Do Lab 10 on your own all in one session
• It is about 60 minutes. It is not timed
• You will receive an email from me today with instructions for completing Lab 10
• Do lab 10 anytime between 4/13 and Monday 4/17
• Lab 10 is a lesson on a new topic and includes practice with the topic, an assessment, and a survey
• Your grade is based on a good attempt and completing each part

Extra Credit for Exam 3

• Go to Sakai quiz: Exam 3 Bonus (opens later today)
• Answer the survey question from Prof. Kristin Stephens-Martinez's (only one question!)

• If 65% of class answer the question
  • 1 extra point on Exam 3 for everyone
• If 75% of class answer the question
  • An additional extra point on Exam 3 for everyone

Assignment 7: Create, Due April 26
Can turn in no penalty by April 30,
No turnin after April 30!
This assignment is required!

Pick one:
  Video: Green dance, advertisement for 101, song, other
  Poem or Multiple Haikus
  Story
  Comic
  One-pager
  Feedback

Let's see some examples

Video Advertisement for CompSci 101
Recommendation Systems: Yelp

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

https://www.youtube.com/watch?v=PniMEnM89iY
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

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!

<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>

Calculating Averages

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

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

\[
\begin{array}{cccccccc}
\text{Tandoor} & \text{IlForno} & \text{McDon} & \text{Loop} & \text{Panda} & \text{Twin} \\
0 & 3 & 5 & 0 & -3 & 5 \\
1 & 1 & 0 & 3 & 0 & -3 \\
-3 & 3 & 3 & 5 & 1 & -1 \\
\end{array}
\]
Items: list of strings (header in table shown)

- Items: list of strings (header in table shown)
  - \( i = \text{"DivinityCafe", "FarmSteak", "IlForno", "LoopPizzaGrill", "McDonalds", "PandaExpress", "Tandoor", "TheCommons", "TheSkillet"} \)

- Values in dictionary are ratings: int list
  - \( \text{len(ratings[i]) == len(items)} \)

```
ratings = \n{"Sarah Lee": [3, 3, 3, 5, 0, -3],
 "Melanie": [5, 0, 3, 0, 1, 3, 3, 1],
 "Jim": [0, 1, 0, -3, 1, 3, 0, 1],
 "Sly one": [5, 0, 1, 3, 0, 1, 3, 3],
 "Sung-Hoon": [0, -1, -1, 5, 1, 3, -3, 1, -3],
 "Nama Grace": [5, 0, 3, -5, -1, 0, 1, 3, 0],
 "Harry": [5, 3, 0, -1, -3, -5, 0, 5, 1],
 "Whi": [1, 3, 0, 1, 3, 3, 0, 3, 0]}
```

Recommender averages

- def averages(items, ratings):
  - Input: items -- list of restaurants/strings
  - Input: dictionary of rater-name to ratings
  - ratings: list of ints, \([1, 0, -1, \ldots, 1]\) -- 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 → 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:
  - similarities("rodger", ratings)

Same as before, dictionary of rater to ratings

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:
  - similarities("rodger", ratings)

- Return [("rater1", #), ("rater2", #), ...]
  - 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?

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?

\[2 \times 0 + 4 \times 3 + 0 \times 5 + 1 \times 0 + 3 \times (-3) + 2 \times 5 = 13\]

Sam’s ratings $[0, 3, 5, 0, -3, 5] \times 13$

Sam’s weighted:

$[0, 39, 65, 0, -39, 65]$

What is Chris’s similarity and weights?

• Suppose Fran is $[2, 4, 0, 1, 3, 2]$
  • Chris’s similarity is:
    \[2 \times 1 + 4 \times 1 + 0 \times 0 + 1 \times 3 + 3 \times 0 + 2 \times (-3) = 3\]
  • Chris’ weighted ratings:
    \[3 \times [1, 1, 0, 3, 0, -3]\]
    \n  \n\[3, 3, 0, 9, 0, -9\]

Sam is more similar to Fran than Chris.
Sam’s weightings will count more than Chris’.
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)
```python
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', #)]
```

```
<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>3</td>
<td>5</td>
<td>0</td>
<td>-3</td>
<td>5</td>
</tr>
<tr>
<td>Chris</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>-3</td>
</tr>
<tr>
<td>Nat</td>
<td>-3</td>
<td>3</td>
<td>3</td>
<td>5</td>
<td>1</td>
<td>-1</td>
</tr>
<tr>
<td>Fran</td>
<td>2</td>
<td>4</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>2</td>
</tr>
</tbody>
</table>
```

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

weights = similarities("Fran", ratings)
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[?] = ?
```

Calculating Weighted Average

<table>
<thead>
<tr>
<th>Tandoor</th>
<th>II Forno</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>
</tr>
<tr>
<td>Chris</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>Nat</td>
<td>-3</td>
<td>3</td>
<td>3</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>Fran</td>
<td>2</td>
<td>4</td>
<td>0</td>
<td>1</td>
<td>3</td>
</tr>
</tbody>
</table>

```
Tandoor  II Forno  McDon  Loop  Panda  Twin
Sam      0        39     65     0      -39   65
Chris    1        1      0      3      0     0
Nat      -3       3      3      5      1     -3
Fran     2        4      0      1      3     2
```

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

- 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!!!! ?

Calculating Weighted Average

```
recommendations("Fran", items, ratings, 2)
```

WOTO-2 Sims to Recs


- From Similarities to Recommendations

```
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**

RecommenderEngine before RecommenderMaker and use TestRecommender

Implement functions in this order

Can be implemented before Recommender stuff or after

Function Call Ordering

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

Start with inner most call and work outwards

Test on your computer and on Gradescope as you go!
When and What’s in CompSci 101

• Problem to solve
  • Use 7 steps
  • Step 5: How do you translate algorithm to code?
    • What do you use to solve it?
    • When do you use it?

What are the “what’s”? 

• Data Structures: list, set, dictionary, tuple
  • Combined: list of lists, dictionary of key to list
• Loops: from for to while, index loop
• Other:
  • List comprehensions
  • Parallel lists
  • Lambda
  • If...if...if
  • If...elif...else

Quick When’s and What’s for 101

• Whichever makes more sense to you:
  • Parallel lists vs dictionaries
  • If...if...if vs if...elif...else
  • List comprehension vs for loop
  • Tuples vs Lists
    • If you want to prevent mutation -> tuples
  • Need single line function
    • Lambda vs create normal helper function

APT – Sorted Freqs

APT SortedFreqs

**Problem Statement**
The frequency with which a data occurs is sometimes an important statistic. In this problem you'll determine how frequently strings occur and return a list representing the frequencies of each different/unique string. The list returned contains as many frequencies as there are unique strings. The returned frequencies represent an alphabetic/lexicographic ordering of the unique words, so the first frequency is how many times the alphabetically first word occurs and the last frequency is the number of times the alphabetically last word occurs.

Consider these strings (quotes for clarity, they're not part of the strings):

```
[“apple”, “pear”, “cherry”, “apple”, “cherry”, “pear”, “apple”, “banana”]
```

The list returned is `{3,1,2,2}` since the alphabetically first word is “apple” which occurs 3 times; the second word alphabetically is “banana” which occurs once, and the other words each occur twice.
What's the best way to ...

• **SortedFreqs**
  - [https://www2.cs.duke.edu/csed/pythonapt/sortedfrequ.html](https://www2.cs.duke.edu/csed/pythonapt/sortedfrequ.html)

• **Count how many times each string occurs**
  - Create `d = {}`, iterate over list updating values

• **OR**
  - Use `data.count(w)` for each `w`

---

APT: SortByFreqs

---

Wait, wait, but what's ...

• **SortByFreqs**
  - [https://www2.cs.duke.edu/csed/pythonapt/sortbyfrequ.html](https://www2.cs.duke.edu/csed/pythonapt/sortbyfrequ.html)

• **Sort by # occurrences high to low**
  - Tuples with count/lambda and reverse=True?
  - Break ties in alphabetical order: two passes
Wait, wait, but what's ...

- **SortByFreqs**
  - [https://www2.cs.duke.edu/csed/pythonapt/sortbyfreqs.html](https://www2.cs.duke.edu/csed/pythonapt/sortbyfreqs.html)

  - **Sort by # occurrences high to low**
    - Tuples with count/lambda and reverse=True?
    - Break ties in alphabetical order: two passes

  ```python
  def sort(data):
      tups = [(data.count(t), t) for t in sorted(set(data))]
      result = [t[1] for t in sorted(tups, key=lambda x : x[0], reverse=True)]
      return result
  ```

- **WOTO-3 (DIDN'T DO)**

  ```python
  data = ["apple", "pear", "cherry", "apple", "pear", "apple", "banana"]
  tups = [(3, "apple"),(1, "banana"),(1,"cherry"),(2,"pear")]
  sorted(...) line 8 = [(3,"apple"),(2,"pear"),(1, "banana"),(1,"cherry")]
  result = ["apple", "pear", "banana", "cherry"]
  PRINT A LOT!
  ```