CIS 1100

Recommending

Python
Fall 2024
University of Pennsylvania

HW9: What to Watch

Part 1: Scraping ✅

Part 2: Recommending

Reminders:

- Due Dec 9 at 11:59pm
- No late days accepted

Recommending

After scraping, we have movie info and user ratings.

We want a way of making a recommendation for a user.

1. Turn each user's ratings into their **genre preferences** by taking the average score that they assign to movies of each genre.
2. Determine a way of comparing one user's preferences to another. (We'll use **cosine similarity.**)
3. Compare the preferences of the user seeking a recommendation to all other users' preferences in order to find the most similar other user.
4. Suggest movies that the most similar other user likes a lot.

User-Based Recommendations

Easier to make recommendations based on *what other people like* rather than some *essential properties about what you like.*

Modeling User Reviews as Preferences

Genres
Action, Adventure, Comedy, Fantasy

Genre can be a useful proxy for more detailed properties of a movie.

We'll model a user's overall preferences by calculating the average scores they assign to movies tagged with a particular genre.

Activity: Ratings to Preferences

```
movie_info = {
    1: ("Harry's Adventure", ("Comedy", "Adventure")),
    2: ("Travis' Tragedy", ("Drama", "IMAX", "Comedy")),
}

# Maps movie IDs to Sadia's ratings of those movies.
sadias_ratings = {1: 3, 2: 4}
```

No need to write code to do these, just mental decoding and arithmetic.

(S7) What rating does Sadia give to *Travis' Tragedy*?

(S8) What is the average rating that Sadia awards to **thriller** movies?

(S9) What is the average rating that Sadia awards to **comedy** movies?

(S10) What is the average rating that Sadia awards to **drama** movies?

Movie Recommender

A `MovieRecommender` stores an attribute called `self.movie_info`. It will look something like this.

```
{1210: ('Star Wars: Episode VI - Return of the Jedi',
        ('Action', 'Adventure', 'Sci-Fi')),
 2028: ('Saving Private Ryan', ('Action', 'Drama', 'War')),
 1307: ('When Harry Met Sally...', ('Comedy', 'Romance')),
 5418: ('Bourne Identity, The', ('Action', 'Mystery', 'Thriller')),
 56367: ('Juno', ('Comedy', 'Drama', 'Romance')),
 3751: ('Chicken Run', ('Animation', 'Children', 'Comedy'))}
```

(L11) If `self.movie_info` stores a dictionary with this shape, write an expression that can look up the *title* of a movie with ID `3943`.

(C12) Finish this function, which prints out each genre associated with the input `movie_id`

```
def print_all_genres(self, movie_id: int):
```

Movie Recommender

A `MovieRecommender` stores an attribute called `self.all_user_ratings`.

It will look something like this. (Actually much longer.)

```
{514: {2716: 5.0, 780: 2.0},
 279: {780: 4.0, 300: 2.5, 1010: 0.5}}
```

(L13) What do the "outer" keys (514, 279) represent? What do the "inner" keys (2716 or 300) represent? What do the `float` values (5.0, 4.0) represent?

HW9: What to Watch

Part 1: Scraping ✅

Part 2: Recommending

Reminders:

- Due Dec 9 at 11:59pm
- No late days accepted
- Autograder coming really soon, I promise
  - we haven't forgotten
  - you can check your correctness on the first few parts using examples in the write-up
  - the autograder tests take *forever* to write because they involve a lot of actual calculations and I really really don't want to have the tests tell you the wrong things so I'm doing a lot of math

Movie Recommender

(C12) Finish this method belonging to the `MovieRecommender` class. Remember the attibutes!

```
self.all_user_ratings: dict[int, dict[int, float]]
self.movie_info: dict[int, tuple[str, tuple]]
```

```
def count_movies_by_genre(self, user_id: int) -> dict[str, int]:
    """Return a dictionary mapping genres to the number of movies that
    the input user has rated from that genre."""

    counter = {}

    ...


    return counter
```

Cosine Similarity

Representing something complex as a bunch of numbers? ✅

Figuring out which bunches of numbers are more or less similar? 😕

Cosine similarity calculates this for us!

- `1` ➡️ identical in direction
- `0` ➡️ perpendicular in direction
- `-1` ➡️ opposite in direction
  - (not actually possible in our case since all numbers are positive)

Cosine Similarity & Vectors

As in the reading, *vectors* are traditionally represented as lists/arrays. But we're using dicts...

- Genres don't have unique numeric identifiers, so we would need a way of encoding genres into the list positions.
  - i.e. in `[4.0, 5.0, 0.0, 3.0]`, which genre gets the `5.0` reading??
- **Sparsity**: There are 19 genres in the dataset, but most people don't rate all of them.
  - `{"Comedy" : 4.0, "Action" : 3.0}` might become the following instead if we needed a list of 19 elements:

```
[4.0, 3.0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

Calculations

Cosine Similarity is calculated like so:

$$\frac{\mathbf{A} \cdot \mathbf{B}}{||\mathbf{A}|| \times ||\mathbf{B}||}$$

"the ratio of the dot product to the product of the magnitudes"

That's hard, but:

- the top term (dot product) is the sum of elementwise products of vectors A and B
- the magnitude of a vector is the square root of the sum of the squares of the elements.

Dot Product

If `A = {"Comedy" : 4.0, "Action" : 3.0}` and `B = {"Action" : 5.0, "Drama" : 2.5}`, then:

$$A \cdot B = 4 \times 0 + 3 \times 5 + 0 \times 2.5 = 15$$

(S7) Calculate the dot product between two vectors `A = {"Comedy" : 4.0, "Action" : 4.0}` and `B = {"Action" : 5.0, "Comedy" : 5.0}`

(C14) Here's a function to calculate the dot product between two *lists* (assuming same length). How would we convert this to work when our vectors are *dicts*?

```
def dot(a: list[float], b: list[float]) -> float:
    total = 0
    for i in range(len(a)):
        total += a[i] * b[i]
    return total
```

Magnitude

If `A = {"Comedy" : 4.0, "Action" : 3.0}` then the magnitude of `A` is:

$$||A|| = \sqrt{4^2 + 3^2} = \sqrt{25} = 5$$

(S8) Calculate the magnitude of `A = {"Comedy" : 4.0, "Action" : 4.0}`

(S9) Calculate the magnitude of `B = {"Action" : 5.0, "Comedy" : 5.0}`

(C16) Here's a function to calculate the magnitude of a vector as a list of floats. How would we convert this to work when our vectors are *dicts*?

```
import math
def mag(a : list[float]) -> float:
    squared = map(lambda x : x * x, a)
    squared_sum = sum(squared)
    return math.sqrt(squared_sum)
```

Cosine Similarity Wrapped

(S10) Combine S7, S8, S9 to calculate the cosine similarity between:

- `A = {"Comedy" : 4.0, "Action" : 4.0}` and
- `B = {"Action" : 5.0, "Comedy" : 5.0}`

(L11) Reflect: what is the meaning of this result?