HW9: What to Watch

Part 1: Scraping ✅

Part 2: Recommending

Reminders:

- Part 1 due Apr 23 at 11:59pm
- Part 2 due Apr 30 at 11:59pm, **no late days accepted**

Movie Recommender

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

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

```
{514: {2028: 5.0, 1210: 2.0},
 279: {1210: 4.0, 1307: 2.5, 56367: 0.5}}
```

(L11) 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?

Movie Recommender

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

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

```python
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
```

*Keep this in mind when calculating the **average** rating of movies per genre...*

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

Dot Product is the sum of elementwise products of vectors A and B.

If

```
A = {"Comedy" : 4.0, "Action" : 3.0}
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}`

Dot Product

Dot Product is the sum of elementwise products of vectors A and B.

If

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

then:

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

(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*?

```python
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*?

```python
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}`

(L13) Reflect: what is the meaning of this result?

Default Dictionaries

Default Dictionaries allow for us to avoid a `KeyError` if we ask for a key that's not already present.

```python
from collections import defaultdict
dd = defaultdict(f)
...
print(dd[key])
```

- If `key in dd`, prints the value associated with `key` in `dd` like a normal dictionary.
- If `key not in dd`, prints the result of `f`

Default Dictionaries

In **(L11)**, write what gets printed for the following snippet. (Assume `defaultdict` has been imported.)

```python
targets = defaultdict(lambda: 11)
targets["first"] = 12
print(targets["first"])
targets["second"] -= 2
print(targets["second"])
print(targets["third"])
```

Default Dictionaries as Counters

`int()` is a function that turns its input into an integer, or returns `0` if it is given no arguments.

```python
print(int()) # prints 0
```

A default dictionary initialized like `defaultdict(int)` uses a default value of `0`! **(C12)** Rewrite `count_movies_by_genre` to use a `defaultdict`.

```python
def count_movies_by_genre(self, user_id: int) -> dict[str, int]:
    user_ratings = self.all_user_ratings[user_id]
    counter = {}
    for movie_id in user_ratings:
        genres = self.movie_info[movie_id][1]
        for genre in genres:
            if genre not in counter:
                counter[genre] = 1
            else:
                counter[genre] = counter[genre] + 1
    return counter
```