Python Data Products
Course 4: Implementing and Deploying data-driven predictive systems

Lecture: Using our similarity-based recommender for rating prediction
Learning objectives

In this lecture we will...
• Show how similarity-based recommenders can be used as a heuristic for rating prediction
• Provide code examples to do so
Collaborative filtering for rating prediction

In the previous lecture we provided code to make recommendations based on the **Jaccard similarity**

How can the same ideas be used for rating prediction?
A simple heuristic for rating prediction works as follows:

• The user (u)’s rating for an item i is a weighted combination of all of their previous ratings for items j
• The weight for each rating is given by the Jaccard similarity between i and j
Collaborative filtering for rating prediction

This can be written as:

\[ r(u, i) = \frac{1}{Z} \sum_{j \in I_u \setminus \{i\}} r_{u,j} \cdot \text{sim}(i, j) \]

Normalization constant

All items the user has rated other than \(i\)

\[ Z = \sum_{j \in I_u \setminus \{i\}} \text{sim}(i, j) \]
Code: Collaborative filtering for rating prediction

Now we can adapt our previous recommendation code to predict ratings

```
In [22]: # More utility data structures

In [23]: reviewsPerUser = defaultdict(list)
reviewsPerItem = defaultdict(list)

In [24]: for d in dataset:
    user, item = d['customer_id'], d['product_id']
    reviewsPerUser[user].append(d)
    reviewsPerItem[item].append(d)

In [25]: ratingMean = sum([d['star_rating'] for d in dataset]) / len(dataset)

In [26]: ratingMean
Out[26]: 4.251102772543146
```

We’ll use the mean rating as a baseline for comparison.

List of reviews per user and per item
Our rating prediction code works as follows:

```python
In [27]: def predictRating(user, item):
    ratings = []
    similarities = []
    for d in reviewsPerUser[user]:
        i2 = d['product_id']
        if i2 == item: continue
        ratings.append(d['star_rating'])
        similarities.append(Jaccard(usersPerItem[item], usersPerItem[i2]))
    if (sum(similarities) > 0):
        weightedRatings = [(x*y) for x,y in zip(ratings, similarities)]
        return sum(weightedRatings) / sum(similarities)
    else:
        # User hasn't rated any similar items
        return ratingMean
```

\[ r(u, i) = \frac{1}{Z} \sum_{j \in I_u \setminus \{i\}} r_{u,j} \cdot \text{sim}(i, j) \]
As an example, select a rating for prediction:

In [28]: dataset[1]

Out[28]: {'marketplace': 'US',
    'customer_id': '14640079',
    'review_id': 'RZSL0BALIYUNU',
    'product_id': 'B003LRN53I',
    'product_parent': '986692292',
    'product_title': 'Sennheiser HD203 Closed-Back DJ Headphones',
    'product_category': 'Musical Instruments',
    'star_rating': 5,
    'helpful_votes': 0,
    'total_votes': 0,
    'vine': 'N',
    'verified_purchase': 'Y',
    'review_headline': 'Five Stars',
    'review_body': 'Nice headphones at a reasonable price.',
    'review_date': '2015-08-31'}

In [29]: u, i = dataset[1]["customer_id"], dataset[1]["product_id"]

In [30]: predictRating(u, i)

Out[30]: 5.0
Similarly, we can evaluate accuracy across the entire corpus:

```python
In [31]: def MSE(predictions, labels):
    differences = [(x-y)**2 for x,y in zip(predictions,labels)]
    return sum(differences) / len(differences)

In [32]: alwaysPredictMean = [ratingMean for d in dataset]

In [33]: cfPredictions = [predictRating(d['customer_id'], d['product_id']) for d in dataset]

In [34]: labels = [d['star_rating'] for d in dataset]

In [35]: MSE(alwaysPredictMean, labels)
Out[35]: 1.4796142779564334

In [36]: MSE(cfPredictions, labels)
Out[36]: 1.6146130004291603
```
Collaborative filtering for rating prediction

Note that this is just a **heuristic** for rating prediction

- In fact in this case it did *worse* (in terms of the MSE) than always predicting the mean
  - We could adapt this to use:
    1. A different similarity function (e.g. cosine)
    2. Similarity based on users rather than items
    3. A different weighting scheme
Summary of concepts

• Showed how similarity-based recommenders can be used to predict ratings
• Provided code for an implementation of this idea

On your own...

• Adapt the code to implement one of the modifications on the previous slide (e.g. cosine, or similarity between users rather than items), and measure its effect on performance