Daniel Kershaw – Senior Data Scientist – Mendeley/Elsevier

Collaborative Filtering Python Tutorial

Data Science Loughborough University 2017

Recommender systems are used within the majority of products that users interact with every day, be this whilst browsing Amazon or watching movies on Netflix.

Collaborative filtering is a dominant method within the field of recommender systems, its core aim is to server a user recommendations based off users whom are similar to them.

Typically, the workflow of a collaborative filtering system is:

- A user expresses his or her preferences by rating items (e.g. books, movies or CDs) of the system. These ratings can be viewed as an approximate representation of the user's interest in the corresponding domain.
- The system matches this user's ratings against other users' and finds the people with most "similar" tastes.
- With similar users, the system recommends items that the similar users have rated highly but not yet being rated by this user (presumably the absence of rating is often considered as the unfamiliarity of an item)

Cosine similarity if often used as a similarity measure. Thus for two users **X**, **Y** their similarity is defined as:

$$sim(x, y) = \frac{\vec{x} \cdot \vec{y}}{\|\vec{x}\| \times \|\vec{y}\|}$$

Task - 1

Implement the missing methods in the Python collaborative filtering code.

1. Copy/Download code from https://github.com/danjamker/LoughboroughUDS-RecSys
2. Implement `person_corrolations(person_one, person_two)` method. The method takes two list, each representing the interest of the users.
    a. Generate list of items they have in common
    b. Calculate the number of items they have in common
    c. For each user calculate the square root of the number of items in their list
    d. Multiple the square roots of each user together
    e. Return the number of items in common divided by the values from above
3. Change the data to add or remove items to see recommendations change

Task – 2

One of the issues with collaborative filtering is that it treats all of a users activity the same, even if it was generated years ago. To model the temporality of tastes we are going to introduce a time decay function.

1. Modify the data structure to take into account a relative date (e.g. how many days ago) they have read/accessed an item.

2. Implement a decay function which takes two parameters, a rating (for this work 1) and a decay constant (how extreme the decay should be), and the time since the item was seen.
3. Use an exponential function to decay the value and return the new decayed value.
4. Modify the code on line 49 to use the new decay function.