Ioannis Sakkaris, HW4 report, CS591

TASK 1

For this task we were asked to predict the rating that a user would give to a specific item. In our train.json document we could find user-item tuples with the rating of the user. So, we had to use this information to make our predictions as accurate as possible. Thus, for each user in train.json I computed the amount of times he purchased any item, putting it in a dictionary (userdensity). For each item I computed the amount of times it was purchased, putting also in a dictionary (itemdensity). Then for each of the users and items I computed the average score they either gave or got depending if it was a user or an item, respectively. These values were also put in dictionaries.

Now, once I got in the file with the missing ratings, I had a user and an item. If that user was in userdensity and item was in itemdensity, I got the number of purchases for the user (unum) and the item (inum). I also got the average ratings for the user and the item. To find the predicted rating I used the equation
(unum/(unum+inum))*user_average+(inum/(unum+inum))*item_average
which gives a weighted average of the sum of the averages of the item and the user. Moreover, I realized if that gave me a value larger than five or less than one, it would be better if I change them to 5 and 1 respectively. For the tuples that were not in userdensity and itemdensity I checked them independently and returned just this item or users average. Finally due to the inconsistence of such datasets, considering as a factor the global average made our results better.

TASK 2

For this task we were asked to predict if a user bought an item. My best solution included \ allowing for more items to be included in the mostPopular items. I got a solution which was better that the baseline by almost 10% from 0.74 to 0.80. I tried using item categories to improve from the baseline, but at best I got a score a bit below it. What I did was, to see how other users that bought the item in question were similar in items bought from the same categories with the user in question. This approach yilded a score of around 0.7. I also checked for similarity between items that the user had bought and the item in question. This actually gave very bad results.

TASK 3

In this task I thought of something very simple, which actually makes a lot of sense. Each, row in train.json had a nhelpful and an outOf column. For each different outOf value I computed the average nhelpful/outOf rate and put them in a dictionary. Now when I was asked to compute the amount of nhelpful for each (user,item,outOf) tuples I just used outOf went back to the dictionary I created, got the value for the average nhelpful/outOf rate and then multiplied it by outOf. That gave me a very good score, close to 61818.