Homework-4 Report

Name: Yixuan Xiao (same as Kaggle username)

BUID: U66975269

BU username: yxxiao

Task 1. Rating prediction

For this task, I implemented item-based collaborative filtering. The reason for chosen item-based CF instead of user-based CF is that in the train.json, the number of users is 500k+, which is much larger than the number of items 170k+. Thus to save computational time and space, item-based is preferred.

1.1 Baseline prediction

After loading in the training data, the global mean value was first computed. The global mean was considered as the average rating from all users to all items, namely,

$$\mu = \frac{sum\ of\ ratings\ from\ all\ reviews}{total\ number\ of\ reviews}$$

Next the bias of rating of each user (b_u) and item (b_i) was computed. This was done by iteratively update b_u and b_i until the values of them converge. At first iteration, b_u was initialized to all zeros. The criterion for convergence was chosen to be the accumulative difference of each value of b_i was less than 1e-10. The regularization term λ, was chosen to be 1.

$$b_i = \frac{\sum_{u \in R(i)}(r_{ui} - \mu - b_u)}{\lambda + |R(i)|} \qquad b_u = \frac{\sum_{i \in R(u)}(r_{ui} - \mu - b_i)}{\lambda + |R(u)|}$$

With these, we can have a baseline prediction of each user to each item as

$$b_{ui} = \mu + b_u + b_i$$

1.2 Item-item similarity

The item-item similarity was computed as the empirical Pearson correlation coefficient and then scaled. The formula for the empirical Pearson correlation coefficient is

$$\hat{\rho_{ij}} = \frac{\sum_{u \in U(i,j)}(r_{ui} - b_{ui})(r_{uj} - b_{uj})}{\sqrt{\sum_{u \in U(i,j)}(r_{ui} - b_{ui})^2 \sum_{u \in U(i,j)}(r_{uj} - b_{uj})^2}}$$

Since the correlation coefficients range from -1 to 1, I rescaled them to 0 to 1, where 0 meaning not at all similar and 1 being strongly similar. And then the similarity is rescaled for small support

$$s_{ij} = \frac{|U(i,j)| - 1}{|U(i,j) - 1 + \lambda|}\hat{\rho_{ij}}$$

1.3 Rating prediction

To predict the rating from a pair of user and item, the baseline prediction is first computed. If the user or the item is new, meaning it is not in the training data, the bias term is set to 0. If both the user and item are not new in the training data, then the weighted sum of ratings from other

items that this user has rated is calculated, where the weights are the similarity between the current item and the items that this current user has rated. Thus, the predicted rating is

$$\hat{r}_{ui} = b_{ui} + \frac{\sum_{j \in s_k(i,u)} s_{ij}(r_{uj} - b_{uj})}{\sum_{j \in s_k(i,u)} s_{ij}}$$

In the result of my rating prediction, I observed several values that are little more than 5.0, so I reset these values to 5.

With this implementation, the rmse of my rating prediction is 1.40041.

Task 2. Purchase Prediction

In this task, the data is binary. For each user and item pair in the training data, 1 meaning the user has bought this item and 0 meaning the opposite. To save computation space and time, instead of saving the full matrix of user and item pairs, the training data is stored as a user dictionary and an item dictionary, where in the user dictionary, each user id is key, and the list of items that this user has bought is value; whereas in the item dictionary, each item id is key, and the list of users that have bought it is its value.

To Implement collaborative filtering, first the average of the number of items that a user buys and the average of the number of users that an item gets bought are computed, and they are 1.962023 and 5.841633, respectively. The next step is to compute the item-item similarity matrix, and since the data is binary, jaccard similarity is computed.

$$s_{ij} = \frac{m_{ij}}{\alpha + m_i + m_j - m_{ij}}$$

The regularization term α is chosen to be 0.1. And the similarity is also shrieked for small support if a pair of users has bought less than 10 same items. And last the similarity is rescale between 0 and 1, 0 meaning not at all similar, 1 meaning very similar.

Besides the collaborative filtering algorithm for prediction, I also considered clustering items so that when encounter a new item in the test set, or an item with not so much similar items, rather than predict it with a global mean, I have more information about the cluster that this item belongs to, and I maybe able to predict more accurately. With the information from the meta.json, each item has categories and price. With these two features, I applied kmeans algorithm on the items. I observed that all items in the test set are in the meta.json, so every item now belongs to a cluster. The implementation of kmeans is very similar to the what I did for clustering in task 3.

With the similarity measure from collaborative filtering and the clustering labels, the purchase prediction is computed based on three factors. (1) For every pair of user and item in the test set, first compute the weighted sum of purchase based on the item similarity, and then rescale the similarity. If this user has purchase history larger than the average number of items that a user buys by 20, 10 and 5 times, than he is more likely to buy new things, therefore the similarity measure is multiplied by 2, 1.5 and 1.2. Similarly, if the item has a past history of being bought more than 20, 10 and 5 times of the average times of an item being bought, than this item is popular and it is more likely to be bought again, so the similarity will get multiplied by 2, 1.5 and 1.2. And if the similarity measure get larger than 0.3, I consider that this user is likely to buy this item. (2) Find out whether this item is in the clusters that this user tends to buy. To do this, find out the clusters of items that this users has bought before, and check if the current item is in them. If the item is in more than 0.2 of the clusters that this user's past bought items belongs to, than I made the decision that this user is likely to buy this item. (3) Find out whether this item is

popular. Using the baseline prediction provided in the baselines.py, there are items that belong to a popular set. Thus to determine whether a new item thats not in the training set is popular or not, I first find out the cluster that this item belongs to, and count the number of popular items in this cluster. If the number of popular items over the total number of items in this cluster is greater than 0.5, thus I consider that this cluster is popular and this current item is popular as well, and thus this user is likely to buy a popular item. To make the final prediction, if a pair of user and item passes one of the above three criterion, than the purchase prediction is made to be 1, otherwise it is 0.

Task 3. Helpfulness Prediction

For the helpfulness prediction, I implemented item-based collaborative filtering and kmeans clustering. Different than task 1 where a rating is stored for each user and item pair, the fraction of helpfulness votes over all votes and the number of all votes are stored.

3.1 Collaborative filtering

Similar as the implementation of collaborative filtering in task 1, the global average of the fraction of helpfulness votes that a review can get is computed first, and this value is 0.7443. Then a global average of total number of votes that a review can get is also computed, and this value is 7.033746. The next step is to compute the the bias of fraction of helpfulness votes of each user and item. Similarly, an iterative update is computed until convergence. The item-item similarity is computed as Pearson correlation coefficients and rescaled between 0 and 1. The final step is to compute the helpful votes prediction.

3.2 Kmeans clustering

The motivation for implementing Kmeans clustering on users and items is that there are many unseen users and items in the test set, therefore if barely using collaborative filtering, reviews with new users and items will be predicted with global mean and that will suffer from low prediction accuracy. However, if we can group the users and items with clustering, and find which cluster that a new user or a new item belongs to, then we can substitute the global mean prediction for these reviews with a local mean prediction, which should increase the prediction accuracy.

For this specific task, I implemented kmeans clustering on both users and items based on the rating, review text and the total number of votes they get. To process the review text, I first use CountVectorizer to convert strings to vectors, and use lsa (TruncatedSVD) to reduce the matrix dimension. Each feature is normalized before feed into the kmeans clustering algorithm. Since the size of the training data is large, I use the MiniBatchKmeans from sklearn to fasten the computation speed.

3.3 Helpfulness prediction

The prediction is made up of two factors, one is the prediction from the collaborative filtering implementation, another is the cluster means from the user and item clusters. The number of helpful votes is computed as the fraction of helpful votes times the total number of votes. The output of both CF and kMeans are the fraction of helpfulness votes.

The CF prediction is computed as the baseline prediction b_{ui} plus the weighted sum of fractions of helpful votes from the current items' similar items. Considering one situation where a review might get 1.0 (100%) helpfulness votes and it only has 1 vote, whereas a review got 0.8 helpfulness votes but has hundreds of total votes. Of course the second review is more trustworthy compared to the first one, but it has a lower value of fraction of helpfulness votes.

Thus in this prediction, the factor of the total number of votes that a review gets should also be considered. In my implementation, if a review has less than 5 votes, than the similarity is shrieked by the votes it gets over 5.

For the kmeans clustering prediction, for each pair of users and items, I first find the clusters that this user and item belongs to, respectively, and then find the mean value of the fraction of helpfulness votes by taking the average of it within that user and item cluster.

My decision for making the final prediction is as follows. If the user and item are both seen in the training set, then the CF prediction, user kmeans clustering and item kmeans clustering takes 0.5, 0.25 and 0.25 weights, and the final prediction is computed as the weighted sum of the 3 predictions. If the user or the item is new to the training set, then the CF prediction is just the baseline prediction, so the CF, user kmeans and item kmeans take 0.2, 0.4 and 0.4 weights, and the final prediction is the weighted sum of the 3 predictions. This implementation results in 69589.17833 absolute error.

3.4 Parameters for prediction improvement

There are several parameters that can be cross-validated to improve the prediction result. (1) The number of components of lsa when processing review texts. In my implementation, I chose the number of components to be 100, but there definitely is room for improvement for the review preprocessing. Other parameter in the CountVectorizer step, such as stop words, min and max word frequencies, can also be fine tuned. (2) The weights of each features when passing data to kmeans. There are three features, namely review text, rating and total number of votes that are being clustered for users and items. Currently they are scaled evenly, but there might be room for improvement if one of the features takes more or less weight. (3) Few votes similarity shrinkage. Current implementation shrink the similarity if the review gets less than 5 votes, but this number may be altered for better prediction to reflect the popularity and trustworthiness of a review. (4) The algorithm and parameters of clustering that applied to users and items. There should be better ways to cluster users and items so that each user and item will be predicted by a more accuratete local means. Knobs that can be changed are the numbers of clusters for users and items, the batch size, or other clustering algorithms available. (5) How much weight each prediction should get for computing the final weighted sum of predictions. The weights I listed in 3.3 is set of best weights I have tried on Kaggle, but there might be a better way to distribute the weights of 3 predictions for a better result.