ㅐ Recommender System (Spring 2022)

Homework #1 (100 Pts, March 9)

Student ID 2019311195

Name 김지유

(1) [20 pts] We are given the following user-item rating matrix for five users and six items. Assume that a peer group of size at most 2 is used in each case, and negative correlations are filtered out.

	I1	I2	I3	I4	I5	I6
U1	5	6	7	4	3	?
U2	4	?	3	?	5	4
U3	?	3	4	1	1	?
U4	7	4	3	6	?	4
U5	1	?	3	2	2	5

(a) [10 pts] Predict the values of unspecified ratings of user 2 using the user-based collaborative filtering algorithm. Use adjusted cosine similarity with mean-centering.

Answer:

Sim(u2, u1) = -1.0

Sim(u2, u3) = -0.99

Sim(u2, u4) = 0.61

Sim(u2, u5) = -0.24

A set of top-2 neighbors for user2 = {u4, u5}

But, only u4 is left when negative correlations are filtered out.

Pred(u2, i2) = 4 + (0.61 * (-0.8)) / 0.61 = 3.2

Pred(u2, i4) = 4 + (0.61 * 1.2) / 0.61 = 5.2

(b) [10 pts] Predict the values of unspecified ratings of user 2 using the item-based collaborative filtering algorithm. Use adjusted cosine similarity with mean-centering.

Answer:

Sim(i2, i1) = -0.62

Sim(i2, i3) = 1.0

Sim(i2, i4) = -0.98

Sim(i2, i5) = -1.0

Sim(i2, i6) = 1.0

A set of top-2 neighbors for item2 = {i3, i6}

Pred(u2, i2) = ((1.0 * 3.0) + (1.0 * 4)) / (1.0 + 1.0) = 3.5

Sim(i4, i1) = 0.79

Sim(i4, i2) = -0.98

Sim(i4, i3) = -0.98

Sim(i4, i5) = 0.94

Sim(i4, i6) = -0.71

A set of top-2 neighbors for item4 = {i1, i5}

Pred(u2, i4) = ((0.79 * 4) + (0.94 * 5) / (0.79 + 0.94) = 4.5

(2) [20 pts] We are given an $m \times n$ rating matrix, where m is the number of users and n is the number of items.

(a) [5 pts] When building the similarity matrix, calculate the worst-case time complexity for item-based and user-based collaborative filtering algorithms.

Answer:

Worst-case time complexity for item-based collaborative filtering: O(mn^2)

Worst-case time complexity for user-based collaborative filtering: O(m^2n)

(b) [5 pts] When making a rating prediction for a target user, calculate the worst-case time complexity for item-based and user-based collaborative filtering algorithms.

Answer:

Worst-case time complexity for item-based collaborative filtering: O(mn) **Worst-case time complexity for user-based collaborative filtering: O(mn)**

(c) [10 pts] Assuming m=10,000 and n=1,000,000, which of the two algorithms, user-based and item-based collaborative filtering, is better? Explain why.

Answer:

User-based collaborative filtering algorithm is better than item-based collaborative filtering algorithm. **The offline process which is building the similarity matrix, is bottleneck.** **User-based collarboritve filtering algorithm takes O(m^2n) and item-based collaborative filtering algorithm takes O(mn^2).** **When m=10,000 and n=1,000,000, O(m^2n) takes less time than O(mn^2) because m is much smaller than n.**

(3) We provide template code and dataset in Python. Refer to 'models/userKNN_explicit.py,' write your code to implement the item-based collaborative filtering algorithm on 'models/itemKNN_explicit.py.' Run '1_main.py' to run the source code.

(a) [20 pts] Implement the function "fit" in 'models/itemKNN_explicit.py' using the Adjusted cosine similarity. It is defined as follows:

$$sim(i,j) = \frac{\sum_{u \in \mathcal{S}_{ij}} (r_{ui} - \bar{r}_{u*})(r_{uj} - \bar{r}_{u*})}{\sqrt{\sum_{u \in \mathcal{S}_{ij}} (r_{ui} - \bar{r}_{u*})^2} \sqrt{\sum_{u \in \mathcal{S}_{ij}} (r_{uj} - \bar{r}_{u*})^2}}$$

Note: Fill in your code here. You also have to submit your code to i-campus.

Answer: predict 함수를 작성할 때, 평균값을 빼지 않은 self.train이 필요하여 __init__ 부분에 self.train = self.train – self.user_mean[:, None]을 self.normalized_train = self.train – self.user_meain[:, None]으로 수정하였습니다.

```python
def __init__(self, train, valid, top_k):
        self.train = train
        self.valid = valid
        self.num_users = train.shape[0]
        self.num_items = train.shape[1]
        self.top_k = top_k

        for i, row in enumerate(self.train):
            self.train[i, np.where(row < 0.5)[0]] = np.nan

        self.user_mean = np.nanmean(self.train, axis=1)
        self.user_mean[np.isnan(self.user_mean)] = 0.0
        #self.train = self.train - self.user_mean[:,None]
        self.normalized_train = self.train - self.user_mean[:,None]

    def fit(self):
        item_item_sim_matrix = np.zeros((self.num_items, self.num_items))
        """
        Calculate item similarities using pearson correlation coefficient (PCC)
        """
        for item_i in range(0, self.num_items):
            for item_j in range(item_i+1, self.num_items):
                # ========================= EDIT HERE ========================
                #pass
                #a = self.train[:, item_i]
                #b = self.train[:, item_j]
                a = self.normalized_train[:, item_i]
                b = self.normalized_train[:, item_j]

                co_rated = ~np.logical_or(np.isnan(a), np.isnan(b))
                a = np.compress(co_rated, a)
                b = np.compress(co_rated, b)

                if len(a) == 0:
                    continue

                dot_a_b = np.dot(a, b)
                if dot_a_b == 0:
                    continue

                item_item_sim_matrix[item_i, item_j] = dot_a_b / (np.linalg.norm(a)
* np.linalg.norm(b))
                # ============================================================
```

(b) [20 pts] Implement the function "predict" in 'models/itemKNN_explicit.py. (1) Find the items user rated, (2) sort by similarity, (3) get Top-K Neighbors using self.top_k, and (4) predict using neighbors. Note that user_mean is subtracted at __init__(), so you should consider it at predict().

Note: Fill in your code here. You also have to submit your code to i-campus.

Answer:

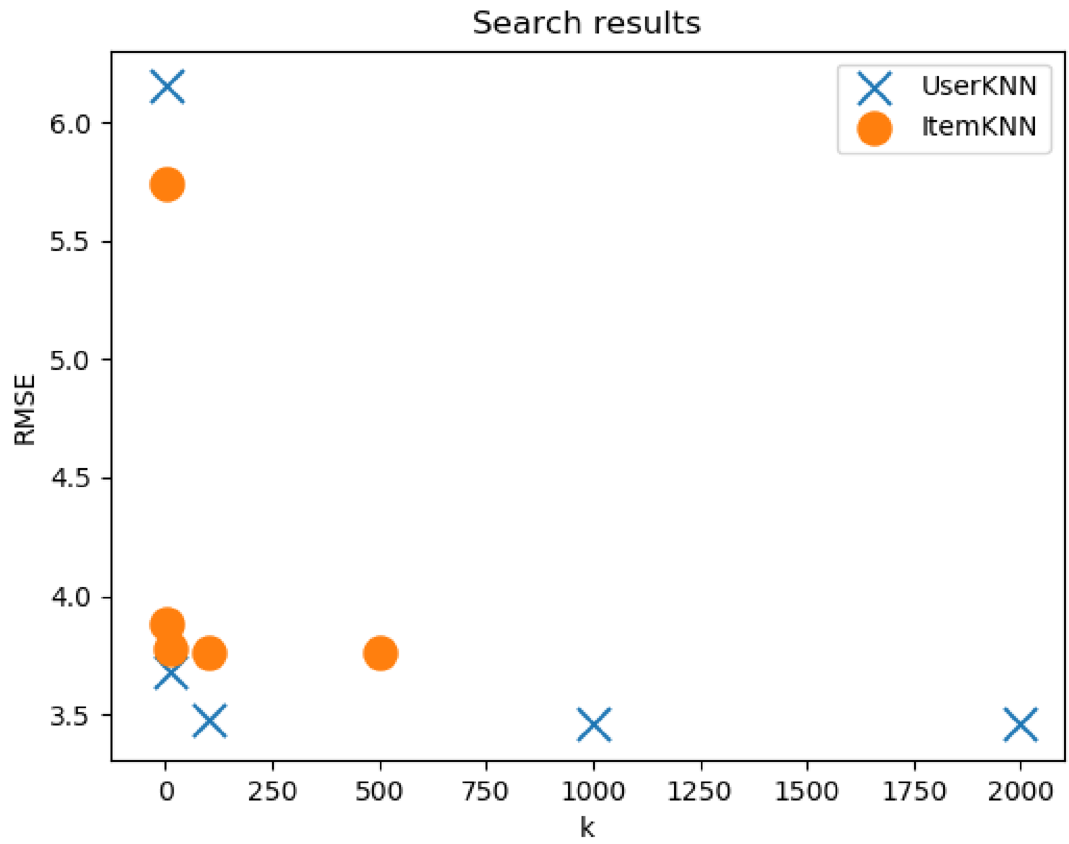
```
# ======================== EDIT HERE ========================
            #pass
            # 유사도 정렬
            sorted_items = np.argsort(unsorted_sim)
            sorted_items = sorted_items[::-1]

            # Top K 이웃 구하기
            if(self.top_k > len(sorted_items)):
                top_k = len(sorted_items)
            else:
                top_k = self.top_k
            sorted_items = sorted_items[0:top_k]
            top_k_items = rated_items[sorted_items]

            # 예측 값 구하기
            if(top_k == 0):
                predicted_values.append(0.0)
            else:
                items_rate = self.train[one_missing_user, top_k_items]
                items_sim = self.item_item_sim_matrix[item_id, top_k_items]

                items_sim[items_sim < 0.0] = 0.0

                if np.sum(items_sim) == 0.0:
                    predicted_rate = self.user_mean[one_missing_user]
                else:
                    predicted_rate = np.sum(items_rate*items_sim) /
np.sum(items_sim)

                predicted_values.append(predicted_rate)
            # ============================================================
```

(c) [20 pts] Given the data ('naver_movie_dataset_small.csv' and 'movielens_100k.csv'), draw the plots of RMSE by adjusting k (i.e., the number of nearest neighbors) for UserKNN and ItemKNN, respectively. (i) For varying k, explain the results and how much k affects RMSE. (ii) evaluate whether UserKNN is better/worse than ItemKNN. Run '2_search.py' to run the source code.

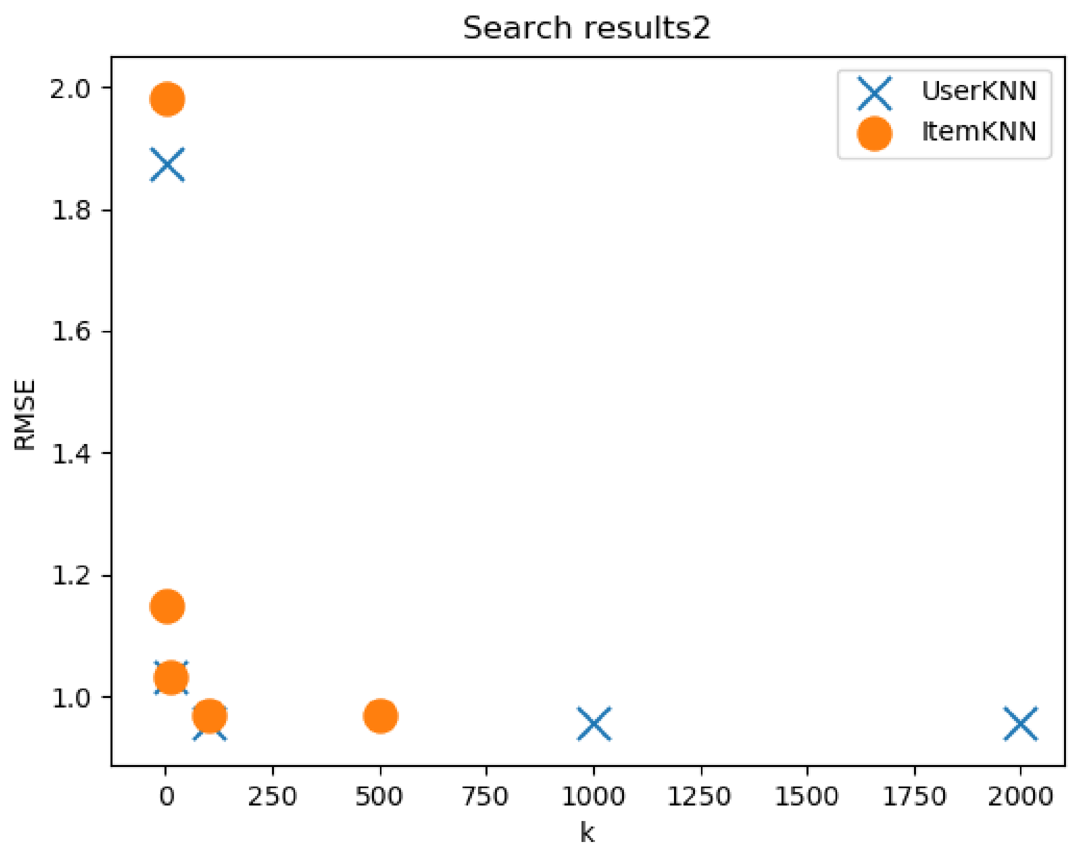
Note: Please show the results for two datasets in the code.

Note: Show your plots and explanations in short (3-5) lines.

Answer:



[Plot for naver_movie_dataset_small.csv]



[Plot for movielens_100k.csv]

(i) For varying k, explain the results and how much k affects RMSE.

As k increase, RMSE decreases and model gets better. As k becomes larger, the accuracy of the model increases because it is recommended by referring to more neighbors' information.

(ii) evaluate whether UserKNN is better/worse than ItemKNN. Run '2_search.py' to run the source code.

In terms of accuracy, UserKNN is slightly better than ItemKNN. But, in terms of time complexity, ItemKNN is much faster than UserKNN.