Recommender System (Spring 2022)

Homework #2 (100 Pts, March 23)

Student ID 김지유

Name 2019311195

(1) [30 pts] We are given six items (**A**, **B**, **C**, **D**, **E**, and **F**) with four transactions in which each user clicks the items in a sequential manner. When a target user clicks the item **A** lastly, calculate the top-3 recommended items.

TID	Sequence
10	**A → B → C**
20	**A → B → D → C→ E**
30	**B → D → A → F**
40	**B → A → E → F**

(a) **[10 pts]** When using simple Association Rules (AR), calculate the top-3 recommended items.

Answer:

Counting scheme(A, B) = 4 **Counting scheme(A, C) = 2** **Counting scheme(A, D) = 2** **Counting scheme(A, E) = 2** **Counting scheme(A, F) = 2** **Top-3 recommended items: B, E, F**

(a) **[10 pts]** When using Markov Chains (MC), calculate the top-3 recommended items.

Answer:

Counting scheme(A, B) = 2 **Counting scheme(A, C) = 0**

Counting scheme(A, D) = 0
Counting scheme(A, E) = 1
Counting scheme(A, F) = 1
Top-3 recommended items: B, E, F

(a) **[10 pts]** When using Sequential Rules (SR), calculate the top-3 recommended items.

Answer:

Counting scheme(A, B) * Weighting scheme(A, B) = 2
Counting scheme(A, C) * Weighting scheme(A, C) = 0.83
Counting scheme(A, D) * Weighting scheme(A, D) = 0.5
Counting scheme(A, E) * Weighting scheme(A, E) = 1.25
Counting scheme(A, F) * Weighting scheme(A, F) = 1.5
Top-3 recommended items: B. F, E

(2) [50 pts] We are given template code and datasets in Python. Using a reference code, fill out your code. Run '0_check.py' and '1_main.py' to validate your implementation code.

(a) [20 pts] Write your code to implement the slope one predictor algorithm in 'models/ SlpeOnePredictor_explicit.py'. The average deviation of two items and predicted rating of the slope one predictor are defined as follows:

$$\mathrm{dev}_{i,j} = \sum_{(r_{ui}, r_{uj}) \in S_{i,j(R)}} \frac{r_{uj} - r_{ui}}{|S_{i,j(R)}|}, \quad \hat{r}_{ui} = \frac{\sum_{i \in S(u)-\{j\}} (\mathrm{dev}_{i,j} + r_{ui}) \cdot |S_{i,j(R)}|}{\sum_{i \in S(u)-\{j\}} |S_{i,j(R)}|}$$

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

Answer:

```python
import numpy as np

class SlopeOnePredictor_explicit():
    def __init__(self, train, valid):
        self.train = train
        self.valid = valid
        self.num_users = train.shape[0]
        self.num_items = train.shape[1]

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

    def fit(self):
        """
        You can pre-calculate deviation in here or calculate in predict().
        """
        # ========================= EDIT HERE ========================
        def get_dev_val(i, j):
            dev_val = 0
            users = 0
            for row in range(self.num_users):
                if (~np.isnan(self.train[row][i])) and
(~np.isnan(self.train[row][j])):
                    users += 1
                    dev_val += self.train[row][i] - self.train[row][j]

            if users == 0:
                ret = 0
            else:
                ret = dev_val / users
            return ret, users

        self.dev = np.zeros((self.num_items, self.num_items))
        self.evaled_users_mat = np.zeros((self.num_items, self.num_items))
        for i in range(self.num_items):
            for j in range(self.num_items):
                if i == j:
                    break
                else:
                    dev_temp, users = get_dev_val(i, j)
                    self.dev[i][j] = dev_temp
                    self.dev[j][i] = (-1) * dev_temp
                    self.evaled_users_mat[i][j] = users
                    self.evaled_users_mat[j][i] = users

        #return dev, evaled_users_mat
        # ============================================================
        #pass
```

```python

    def predict(self, user_id, item_ids):

        predicted_values = []
        # user i가 시청한 item들
        rated_items = np.where(~np.isnan(self.train[user_id,:]))[0]
        for one_missing_item in item_ids:
            # ========================= EDIT HERE ========================
            predicted_rate = np.sum((self.dev[one_missing_item][rated_items] +
self.train[user_id][rated_items]) *
self.evaled_users_mat[one_missing_item][rated_items]) /
np.sum(self.evaled_users_mat[one_missing_item][rated_items])
            predicted_values.append(predicted_rate)
            # ============================================================
        return predicted_values


```

(b) [10 pts] Refer to 'models/MF_explicit.py,' write your code to implement the matrix factorization algorithm with modeling user & item bias on 'models/BiasedMF_explicit.py.' Initialize all the variables following a normal distribution N(0, 0.01). The predicted rating of the biased MF is defined as follows:

$\hat{r}_{ui} = o_u + p_i + u_u v_i^T$ where o_u and p_i denote bias for user u and item i, respectively.

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

Answer:

```python
import numpy as np
import torch


class BiasedMF_explicit_model(torch.nn.Module):
    def __init__(self, num_users, num_items, n_features):
        super().__init__()
        # ========================= EDIT HERE ========================
        self.user_factors = torch.nn.Embedding(num_users, n_features+2,
sparse=False)
        self.item_factors = torch.nn.Embedding(num_items, n_features+2,
sparse=False)

        torch.nn.init.normal_(self.user_factors.weight, std=0.01)
        torch.nn.init.normal_(self.item_factors.weight, std=0.01)

        torch.nn.init.ones_(self.user_factors.weight[:,-1])
        torch.nn.init.ones_(self.item_factors.weight[:,-2])
        # ========================= EDIT HERE ========================
```

```python

    def forward(self):
        reconstruction = None
        # ========================= EDIT HERE ========================
        reconstruction = torch.matmul(self.user_factors.weight,
self.item_factors.weight.T)

        # ========================= EDIT HERE ========================
        return reconstruction

```

```python
class BiasedMF_explicit():
    def __init__(self, train, valid, n_features=20, learning_rate = 1e-2,
reg_lambda =0.1, num_epochs = 100):
        self.train = train
        self.valid = valid
        self.num_users = train.shape[0]
        self.num_items = train.shape[1]
        self.num_epcohs = num_epochs
        self.n_features = n_features

        self.y = np.zeros_like(self.train)
        for i, row in enumerate(self.train):
            self.y[i, np.where(row > 0.5)[0]] = 1.0

        self.model = BiasedMF_explicit_model(self.num_users, self.num_items,
self.n_features)#.cuda()
        self.optimizer = torch.optim.Adam(self.model.parameters(),
lr=learning_rate, weight_decay=reg_lambda)

```

```python
    def mse_loss(self, y, target, predict):
        return (y * (target - predict) ** 2).sum()

```

```python
    def fit(self):
        ratings = torch.FloatTensor(self.train)#.cuda()
        weights = torch.FloatTensor(self.y)#.cuda()

        # U와 V를 업데이트 함.
        for epoch in range(self.num_epcohs):
            self.optimizer.zero_grad()

            # 예측
            prediction = self.model.forward()
            loss = self.mse_loss(weights, ratings, prediction)

            # Backpropagate
            loss.backward()
```

```python

        # Update the parameters
        self.optimizer.step()


    with torch.no_grad():
        self.reconstructed = self.model.forward().cpu().numpy()


def predict(self, user_id, item_ids):
    return self.reconstructed[user_id, item_ids]


```

(c) [20 pts] Refer to 'models/MF_explicit.py,' write your code to implement the SVD++ algorithm with on 'models/SVDpp_explicit.py'. Run '0_check.py' to validate your implementation. Initialize all the variables following normal distribution $N(0, 0.01)$. The predicted rating of the SVD++ is defined as follows:

$$\hat{r}_{ui} = \sum_{s=1}^{k+2} (u_{us} + [FY]_{us})\, v_{is} = \sum_{s=1}^{k+2} \left(u_{us} + \sum_{h \in \mathcal{I}_u} \frac{y_{hs}}{\sqrt{\mathcal{I}_u + \varepsilon}} \right) v_{is} = o_u + p_i + \sum_{s=1}^{k} (u_{us} + [FY]_{us})\, v_{is}$$

$$\text{where } \varepsilon = 1e - 10$$

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

Answer:

```python
import numpy as np
import torch


class SVDpp_explicit_model(torch.nn.Module):
    def __init__(self, num_users, num_items, n_features):
        super().__init__()
        # ========================= EDIT HERE ========================
        self.user_factors = torch.nn.Embedding(num_users, n_features+2,
sparse=False)
        self.item_factors = torch.nn.Embedding(num_items, n_features+2,
sparse=False)
        self.latent_item_matrix = torch.nn.Embedding(num_items, n_features+2,
sparse=False)

        torch.nn.init.normal_(self.user_factors.weight, std=0.01)
        torch.nn.init.normal_(self.item_factors.weight, std=0.01)
        torch.nn.init.normal_(self.latent_item_matrix.weight, std=0.01)
```

```python

        torch.nn.init.ones_(self.user_factors.weight[:,-1])
        torch.nn.init.ones_(self.item_factors.weight[:,-2])
        torch.nn.init.zeros_(self.latent_item_matrix.weight[:,-1])
        torch.nn.init.zeros_(self.latent_item_matrix.weight[:,-2])
        # ========================= EDIT HERE ========================

    def forward(self, implicit_train_matrix):
        reconstruction = None
        # ========================= EDIT HERE ========================
        uv = torch.matmul(self.user_factors.weight, self.item_factors.weight.T)
        fy = torch.matmul(implicit_train_matrix, self.latent_item_matrix.weight)
        fyv = torch.matmul(fy, self.item_factors.weight.T)
        reconstruction = uv + fyv
        # ========================= EDIT HERE ========================
        return reconstruction

```

```python
class SVDpp_explicit():
    def __init__(self, train, valid, n_features=20, learning_rate = 1e-2,
reg_lambda =0.1, num_epochs = 100):
        self.train = train
        self.valid = valid
        self.num_users = train.shape[0]
        self.num_items = train.shape[1]
        self.num_epcohs = num_epochs
        self.n_features = n_features

        self.y = np.zeros_like(self.train)
        for i, row in enumerate(self.train):
            self.y[i, np.where(row > 0.5)[0]] = 1.0

        self.model = SVDpp_explicit_model(self.num_users, self.num_items,
self.n_features)#.cuda()
        self.optimizer = torch.optim.Adam(self.model.parameters(),
lr=learning_rate, weight_decay=reg_lambda)

```

```python
    def mse_loss(self, y, target, predict):
        return (y * (target - predict) ** 2).sum()

```

```python
    def fit(self):
        ratings = torch.FloatTensor(self.train)#.cuda()
        weights = torch.FloatTensor(self.y)#.cuda()

        implicit_ratings = torch.FloatTensor(self.train).bool().float()

        # TODO: normalize implicit ratings with the eplison
```

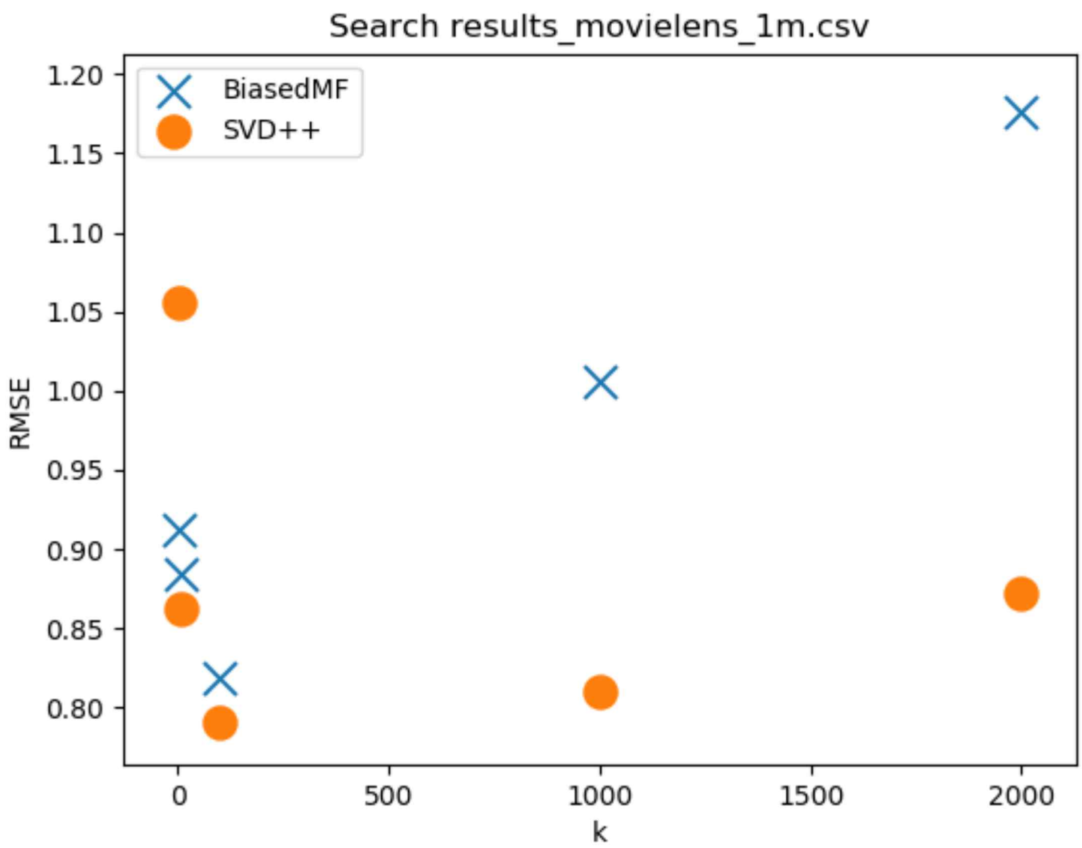
```
        # ======================== EDIT HERE ======================
        epsilon = 1e-10
        ju = torch.sum(implicit_ratings, dim=1)
        ju = torch.sqrt(ju).view((-1, 1))
        implicit_ratings = implicit_ratings / (ju + epsilon)
        # ======================== EDIT HERE ======================

        # U와 V를 업데이트 함.
        for epoch in range(self.num_epcohs):
            self.optimizer.zero_grad()

            # 예측
            prediction = self.model.forward(implicit_ratings)
            loss = self.mse_loss(weights, ratings, prediction)

            # Backpropagate
            loss.backward()

            # Update the parameters
            self.optimizer.step()

        with torch.no_grad():
            self.reconstructed =
self.model.forward(implicit_ratings).cpu().numpy()
        self.implicit_ratings = implicit_ratings.cpu().numpy()

    def predict(self, user_id, item_ids):
        return self.reconstructed[user_id, item_ids]
```

(3) [20 pts] Given the data ('naver_movie_dataset_100k.csv' and 'movielens_1m.csv'), draw the plots of RMSE by adjusting rank for biased MF and SVD++ respectively. With adjusting dimension sizes(=rank), explain the results and how much rank affects RMSE. Run '2_search.py' to run the code.

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

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

Answer:



[movielens_1m.csv plot]

# of users: 6040, # of items: 3706, # of ratings: 1000209

BiasedMF RSME (rank=1): 0.9118173452993139

BiasedMF RSME (rank=10): 0.883645348753175

BiasedMF RSME (rank=100): 0.8188960433142412

BiasedMF RSME (rank=1000): 1.0053960343744885

BiasedMF RSME (rank=2000): 1.1759728906737148

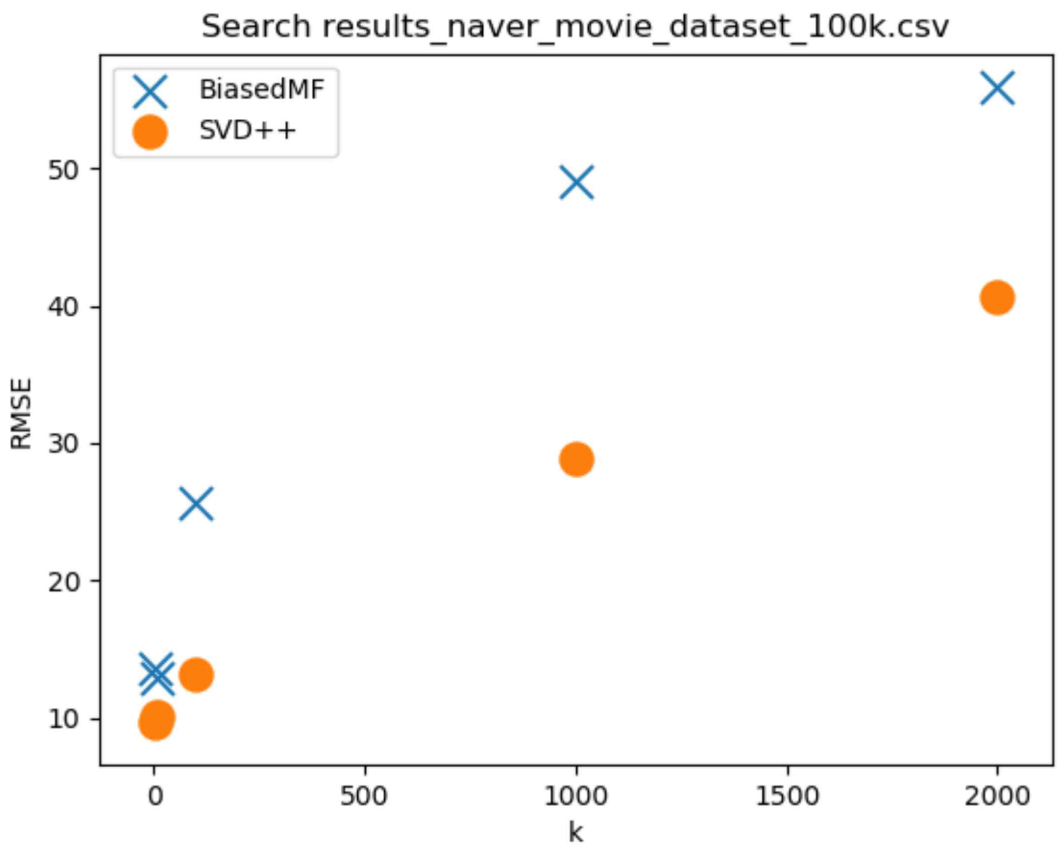
SVD++ RSME (rank=1): 1.0550646534679657

SVD++ RSME (rank=10): 0.8621007092760302

SVD++ RSME (rank=100): 0.7911692853369093

SVD++ RSME (rank=1000): 0.8106433849873129

SVD++ RSME (rank=2000): 0.8723396829284731



[naver_movie_dataset_100k.csv plot]

# of users: 4046, # of items: 16126, # of ratings: 104159

BiasedMF RSME (rank=1): 13.524585373119024

BiasedMF RSME (rank=10): 12.96083773210796

BiasedMF RSME (rank=100): 25.607214876625655

BiasedMF RSME (rank=1000): 49.10693724165516

BiasedMF RSME (rank=2000): 56.00732615500694

SVD++ RSME (rank=1): 9.696116149907047

SVD++ RSME (rank=10): 10.041603372406914

SVD++ RSME (rank=100): 13.106302549234462

SVD++ RSME (rank=1000): 28.869124306204547

SVD++ RSME (rank=2000): 40.693140023660646

From rank=1 to rank=100, RMSE seems to decrease.

But, from rank=1000, RMSE seems to increase again.

SVD++ shows lower RMSE than BiasedMF.