Recommender System (Spring 2022)

Homework #2 (100 Pts, March 23)

Student ID 2021710602

Name 김정의

(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: A와 같은 transaction에 있던경우

B : 4 D : 2 F: 2 1등 : B
C : 2 E : 2 공동2등 : CDEF

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

Answer: A 바로 다음에 나타난 item,

10 : A→B 30. A→F
20 : A→B 40 : A→E
→ B : 2 E : 1 1등 : B
C : 0 F : 1 공동 2등 E, F
D : 0

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

Answer:

$B = 1+1$ $D = \frac{1}{2}$ 1등 : B
$C = \frac{1}{2} + \frac{1}{3}$ $E = \frac{1}{4} + 1$ 2등 : F
$F = 1 + \frac{1}{2}$ 3등 : 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:

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

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

Answer:

```
    def fit(self):
        """
        You can pre-calculate deviation in here or calculate in predict().
        """

        pass

    def predict(self, user_id, item_ids): # user_id =  2 , item_ids =  [1,3]

        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 ========================
            # [user_id, one_missing_item] 구하기
            bunmo = 0
            bunza = 0
            for r in rated_items:

                temp = self.train[:,one_missing_item] - self.train[:,r]
                item_counting = sum( ~np.isnan(temp) )
                temp_mean = np.nanmean(temp)

                if np.isnan(temp_mean):
                    continue
                else:

                    user_plus_mean =self.train[user_id,r]+ temp_mean
                    bunza  += user_plus_mean * item_counting
                    bunmo  += item_counting

            predicted_values.append( bunza/bunmo)


            # ============================================================
        return predicted_values
```

fit pass하고
predict에서 구하는 방식

(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
class BiasedMF_explicit_model(torch.nn.Module):
    def __init__(self, num_users, num_items, n_features):
        super().__init__()
        # ======================== EDIT HERE ========================

        # padding_index를 세로로 거는법을 몰라서 유저팩터부분을 거꾸로만들고 계산할때 뒤집겠습니다
        self.user_factors = torch.nn.Embedding( n_features+2, num_users,padding_idx= n_features+1) # sparse=False가 디폴트
        self.item_factors = torch.nn.Embedding(n_features+2 , num_items,padding_idx = n_features)

        torch.nn.init.normal_(self.user_factors.weight, std=0.01)
        torch.nn.init.normal_(self.item_factors.weight, std=0.01)
        self.user_factors.weight.data[n_features+1,:]=1.0
        self.item_factors.weight.data[n_features,:]=1.0



        # ======================== EDIT HERE ========================

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

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

(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
class SVDpp_explicit_model(torch.nn.Module):
    def __init__(self, num_users, num_items, n_features):
        super().__init__()
        # ======================== EDIT HERE ========================
        self.item_to_nf = torch.nn.Embedding(num_items,n_features)
        self.item_to_nf_bias = torch.zeros(num_items,2,requires_grad=False)

        self.nf_to_item = torch.nn.Embedding(n_features+2,num_items,padding_idx=n_features)

        torch.nn.init.normal_(self.item_to_nf.weight, std=0.01)
        torch.nn.init.normal_(self.nf_to_item.weight, std=0.01)
        self.nf_to_item.weight.data[n_features,:]=1.0



        # ======================== EDIT HERE ========================

    def forward(self, implicit_train_matrix):
        reconstruction = None
        # ======================== EDIT HERE ========================
        # item_to_nf 랑 bias랑 붙이기
        item_to_nf_concat = torch.cat((self.item_to_nf.weight,self.item_to_nf_bias),1)

        reconstruction = torch.matmul(implicit_train_matrix,item_to_nf_concat )
        reconstruction = torch.matmul(reconstruction ,self.nf_to_item.weight )

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

```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 ========================
        small_num = float(1e-10)
        implicit_ratings = implicit_ratings * small_num

        # ======================== 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]
```

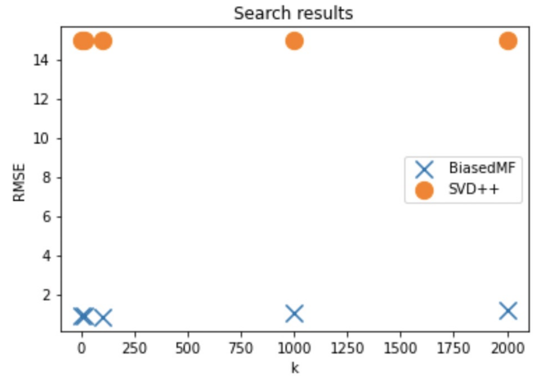
(3) [20 pts] Given the data ('naver_movie_dataset_100k.csv' and 'movielens_1m.csv'), draw the plots of RMSE by adjusting rank for biased MF and SVD++ respectively. With adjusting dimension sizes(=rank), explain the results and how much rank affects RMSE. Run '2_search.py' to run the code.

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

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

Answer:

```
# of users: 6040,  # of items: 3706,  # of ratings: 1000209
BiasedMF RSME (rank=1):  0.9132231920294599
BiasedMF RSME (rank=10):  0.8842474094803707
BiasedMF RSME (rank=100):  0.8361730496916066
BiasedMF RSME (rank=1000):  1.0162354962377491
BiasedMF RSME (rank=2000):  1.1736006712530431
SVD++ RSME (rank=1):  14.953892811307245
SVD++ RSME (rank=10):  14.953892811307245
SVD++ RSME (rank=100):  14.953892811307245
SVD++ RSME (rank=1000):  14.953892811307245
SVD++ RSME (rank=2000):  14.953892811307245
```



Biased 모델의 경우 rank=100 인때 rmse가 가장 낮았다. 100 인때가 잠재적 특징을 가장 잘 잡아내는 크기인 것으로 판단된다.

SVD++ 경우 뭔가 모델링이 잘못된 것 같습니다.