long-term recommendation system project

Summary of the algorithm

environment

OS : Ubuntu 18.04

CPU : Intel(R) Xeon(R) CPU @ 2.20GHz

GPU : NVIDIA TESLA P-100 / NVIDIA TESLA T-4

structure

```
├─recommendation_system/
|  ├─recommender.py # train and evaluation
|  ├─train.py # train
|  ├─model.py # model code
|  ├─loss.py # loss code
|  ├─utils.py # utils code
|  ├─config.json # training configuration
|  ├─u1.pt # u1 model weight and matrix
|  ├─u2.pt # u2 model weight and matrix
|  ├─u3.pt # u3 model weight and matrix
|  ├─u4.pt # u4 model weight and matrix
|  ├─u5.pt # u5 model weight and matrix
|  ├─data-1/
|  |  ├ ...
|  ├─test/
|  |  ├ ...
```

introduce

기존 CF는 data imputation 기법으로의 CF와 recommendation 기법으로의 CF가 존재.

따라서 deep learning을 통한 matrix factorization을 data imputation과 recommendation 기법으로서 반복적으로 사용해 imputed data의 quality과 recommendation의 quality를 지속적으로 향상시키는 방식으로 training을 진행

algorithm

1. deep learning model로 matrix factorization을 진행
    - 초기 matrix에서 unrated data를 1로 data imputation을 적용
2. latent vector로 실제 matrix를 복원하는 방식으로 모델 학습을 진행
3. STEPS 주기마다 model inference해 unrated data를 imputation한다.
4. 3번에서 update된 imputed matrix를 이용해 다시 1번부터 반복

configuration

`config.json`에서 training option을 변경할 수 있다.

```
{
    "STEPS" : 1000, // data imputation 주기
    "EPOCHS" : 500, // 학습 epoch 수
    "EARLY_STOP":3000, // early stopping 횟수
    "log" : 50, // console logging 주기
    "hidden_dims" : [256,128,64,128,256] // embedder block 수와 dimension
}
```

Detail methods

model

```
class RecommendNet(nn.Module):
  def __init__(self,user_num,item_num,hidden_dims=[256,128,64,128,256]):
    super(RecommendNet,self).__init__()

    ...
    self.user_net= UserEmbedder(item_num=item_num,hidden_dims=hidden_dims)
    self.item_net= ItemEmbedder(user_num=user_num,hidden_dims=hidden_dims)
    ...

  def forward(self,x):
    # x = (user_num,item_num)
    user_vec = self.user_net(x) # (user_num,hidden_dim)
    item_vec = self.item_net(x.T) # (item_num,hidden_dim)

    result = user_vec @ item_vec.T

    return result # (user_num,item_num)
```

userEmbedder와 ItemEmbedder는 모두 fully connected layer와 skip connection으로 이루어진 network이다.

```
class ResBlock(nn.Module):
  def __init__(self,in_dim,out_dim):
    super(ResBlock,self).__init__()
    self.block1= nn.Sequential(
        nn.BatchNorm1d(in_dim),
        nn.GELU(),
        nn.Dropout(p=0.5),
        nn.Linear(in_dim,in_dim),
        )
    self.block2=nn.Sequential(
        nn.BatchNorm1d(in_dim),
        nn.GELU(),
        nn.Dropout(p=0.5),
        nn.Linear(in_dim,out_dim),
        )

  def forward(self,x):
    return self.block2(x+self.block1(x))
```

loss

실제 matrix와 모델의 예측값의 차이가 줄도록 loss를 설정

```
class RecommendLoss(nn.Module):
    def __init__(self):
        super(RecommendLoss,self).__init__()

    def forward(self,target,pred):
        return torch.sqrt(((target-pred)**2).mean())
```

train

1. STEP만큼 모델 학습을 진행한 뒤, 해당 모델로 data imputation을 진행해 matrix를 update
2. 최종 epoch이 지난 뒤, model weight와 imputed matrix를 저장
    - `u#.pt`로 해당 데이터를 저장

```
model = model.to(device)

for epoch in range(EPOCHS):

    ...

    for step in range(STEPS):
        data = matrix.clone().to(device)

        optimizer.zero_grad()

        pred= model.forward(data)
        loss = criterion(data,pred)
        loss.backward()
        optimizer.step()
        scheduler.step(loss)

        ...

    print(f"FINISH {epoch+1} epoch.")

    # STEP이 지난 후 matrix를 모델을 이용해 data imputation 진행
    print("="*100)
    print("EVALUATION..")
    with torch.no_grad():
        ...

        pred = model.forward(matrix.to(device))

        ...

        print(f"UPDATE matrix..")
        matrix = pred

        for idx,d in enumerate(train_data[:,[0,1]]):
            matrix[d[0],d[1]]=train_data[idx,2]

        matrix = matrix.clip(min=1,max=5)
    # 모델 weight와 imputated matrix를 저장
    model_weights = os.path.basename(train_path).split('.')[0] + ".pt"
    with open(model_weights,'wb') as f:
        torch.save({'model':model.state_dict(),'matrix':matrix},f)
    print("save model weights in {}.".format(model_weights))
```

evaluation

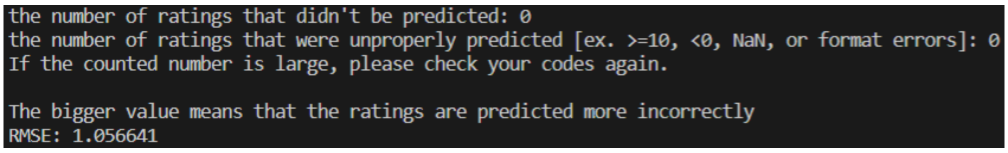
`recommender.py`에서 evaluation을 진행

1. `u#.pt`가 없는 경우 model train을 수행
2. imputed matrix로 model prediction을 수행

```
def evaluation(train_path,test_path,config_path):
    model_weights = os.path.basename(train_path).split('.')[0] + ".pt"

    # 1. u#.pt가 없는 경우 model train을 수행
    if not os.path.exists(model_weights):
        ...
        train(train_path,test_path,config_path)
        ...

    ...

    checkpoint = torch.load(model_weights,map_location=device)
    matrix = checkpoint['matrix']

    model = RecommendNet(user_num=matrix.shape[0],item_num=matrix.shape[1],hidden_dims=config['hidden_dims'])
    model.load_state_dict(checkpoint['model'])

    ...

    # 2. imputed matrix로 model prediction을 수행
    with torch.no_grad():
        model.eval()
        pred = model.forward(matrix)

    pred_list=[]
    for t in test_data:
      pred_list.append(f"{t[0]}\t{t[1]}\t{pred[t[0],t[1]]}\n")

    save_path = os.path.basename(train_path).split('.')[0]+".base_prediction.txt"
    with open(save_path,"w") as f:
        f.writelines(pred_list)
    print("save output file in {}.".format(save_path))
```

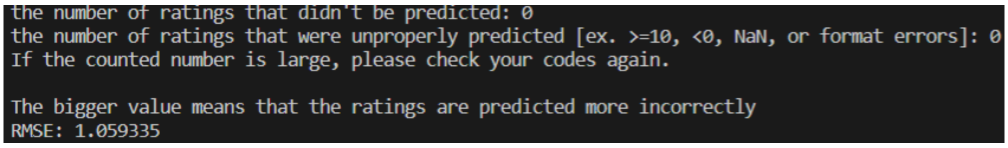
Results

u1 result



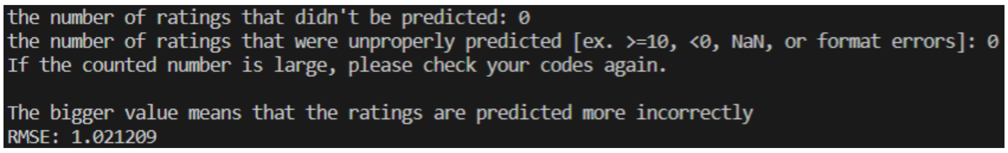
```
the number of ratings that didn't be predicted: 0
the number of ratings that were unproperly predicted [ex. >=10, <0, NaN, or format errors]: 0
If the counted number is large, please check your codes again.

The bigger value means that the ratings are predicted more incorrectly
RMSE: 1.056641
```

u2 result



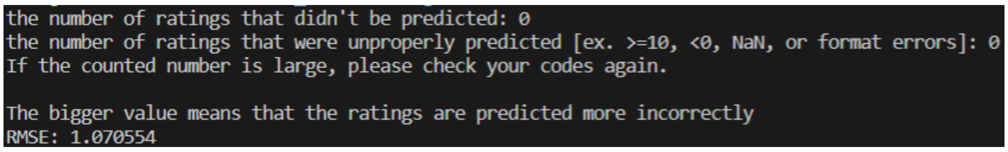
```
the number of ratings that didn't be predicted: 0
the number of ratings that were unproperly predicted [ex. >=10, <0, NaN, or format errors]: 0
If the counted number is large, please check your codes again.

The bigger value means that the ratings are predicted more incorrectly
RMSE: 1.059335
```

u3 result



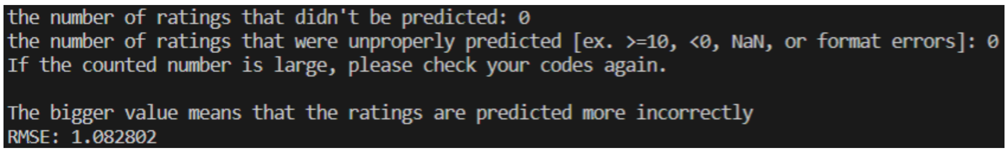
```
the number of ratings that didn't be predicted: 0
the number of ratings that were unproperly predicted [ex. >=10, <0, NaN, or format errors]: 0
If the counted number is large, please check your codes again.

The bigger value means that the ratings are predicted more incorrectly
RMSE: 1.021209
```

u4 result



```
the number of ratings that didn't be predicted: 0
the number of ratings that were unproperly predicted [ex. >=10, <0, NaN, or format errors]: 0
If the counted number is large, please check your codes again.

The bigger value means that the ratings are predicted more incorrectly
RMSE: 1.070554
```

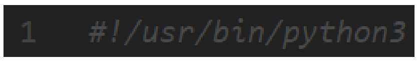
u5 result



```
the number of ratings that didn't be predicted: 0
the number of ratings that were unproperly predicted [ex. >=10, <0, NaN, or format errors]: 0
If the counted number is large, please check your codes again.

The bigger value means that the ratings are predicted more incorrectly
RMSE: 1.082802
```

Execute guide

1. `recommender.py`의 첫 줄에 python 실행 위치를 적어줍니다.



```
#!/usr/bin/python3
```

in recommender.py

2. 현재 유저의 `recommender.py` 실행권한을 허용합니다.

```
chmod 755 recommender.py
```