Recommender system 구현

컴퓨터소프트웨어학부

2016025996

조성진

Long Term Project – Recommender system 구현을 위하여 pytorch를 사용한 딥러닝 학습방법을 사용했다.

사용한 라이브러리는 다음과 같다.

```
import numpy as np
import pandas as pd
import torch
import torch.nn as nn
import torch.nn.functional as F
import sys
```

터미널에서 실행시킬 경우 터미널 인자를 다음과 같은 형식으로 받는다

```
input_file = sys.argv[1]
test_file = sys.argv[2]
start = input_file.split(".")[0]
predict_file = start + ".base_prediction.txt"
```

알고리즘에 앞서 data preprocessing을 진행하는데 우선 유저와 아이템의 번호가 1번부터 주어져있고, 중간중간에 비어있는 유저의 번호와 아이템의 번호가 존재하여 그것을 각각 0부터 매칭시켜 라벨을 다시 매겨줬다.

```
user_idx = np.unique(np.array(df)[:,0])
item_idx = np.unique(np.array(df)[:,1])
userId2idx = {uId:idx for (idx, uId) in enumerate(user_idx)}
itemId2idx = {iId:idx for (idx, iId) in enumerate(item_idx)}
# 있는 인덱스끼리 연결
df[0] = df[0].apply(lambda x: userId2idx.get(x))
df[1] = df[1].apply(lambda x: itemId2idx.get(x))
```

그리고 정확도 향상을 위하여 One class collaborative filtering을 사용하였는데, 이것이 후에 사용할 rating 1~5점 사이에 점수를 주는 형식과 조금 맞지 않는다고 판단하여 클래스를 각각 1개씩 만들었다.

OCCF에 관한 클래스는 다음과 같다.

```
class OCCF(nn.Module):
    def __init__(self, num_user, num_item, embedding_size):
        super(OCCF, self).__init__()
        self.embedding_user = nn.Embedding(num_user, embedding_size)
        self.embedding_item = nn.Embedding(num_item, embedding_size)
        self.embedding_user_bias = nn.Embedding(num_user, 1)
        self.embedding_item_bias = nn.Embedding(num_item, 1)
        torch.nn.init.xavier_uniform_(self.embedding_user.weight)
        torch.nn.init.xavier_uniform_(self.embedding_item.weight)
        torch.nn.init.xavier_uniform_(self.embedding_user_bias.weight)
        torch.nn.init.xavier_uniform_(self.embedding_item_bias.weight)
....
    def forward(self, user, item):
        embedded_user = self.embedding_user(user)
        embedded_item = self.embedding_item(item)
        bias_user = self.embedding_user_bias(user).squeeze()
        bias_item = self.embedding_item_bias(item).squeeze()
        return (embedded_user * embedded_item).sum(1) + bias_user + bias_item
```

유저와 아이템의 번호를 임베딩시켜 weight 와 bias의 sum으로 점수를 계산한다.

```
def train_OCCF(model, epoch=15, lr=0.01, wd=0):
    optimizer= torch.optim.AdamW(model.parameters(), lr, weight_decay=wd)
    for i in range(epoch):
        model.train()
        users = torch.LongTensor(train_set[0].values).to(device)
        items = torch.LongTensor(train_set[1].values).to(device)
        ratings = torch.FloatTensor([1.0 for _ in range(len(train_set[2].values))]).to(device)
        predict = model(users, items)
        l2_lambda = 0.001
        l2_reg = torch.tensor(0.).to(device)
        for param in model.parameters():
            l2_reg += torch.norm(param)
        loss = F.mse_loss(predict, ratings)
        loss += l2_lambda * l2_reg
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        # print("epoch %d loss %.3f" % (i+1, loss))
```

학습 함수는 다음과 같으며 L2 Regularization을 사용했다.

그리고 최종 score을 predict할때 사용하는 class로 다음 코드와 같은 구현을 하였는데,

```
class MF_bias(nn.Module):
    def __init__(self, num_user, num_item, mf_embedding_size=8, fn_embedding_size=32):
        super(MF_bias, self).__init__()
        self.last_layer_size = 16
        self.mf_embedding_user = nn.Embedding(num_user, mf_embedding_size)
        self.mf_embedding_item = nn.Embedding(num_item, mf_embedding_size)
        self.fn_embedding_user = nn.Embedding(num_user, fn_embedding_size)
        self.fn_embedding_item = nn.Embedding(num_item, fn_embedding_size)
        torch.nn.init.xavier_uniform_(self.mf_embedding_user.weight)
        torch.nn.init.xavier_uniform_(self.mf_embedding_item.weight)
        torch.nn.init.xavier_uniform_(self.fn_embedding_user.weight)
        torch.nn.init.xavier_uniform_(self.fn_embedding_item.weight)
        self.linear_stack = nn.Sequential(
            nn.Linear(2*fn_embedding_size, 64),
            nn.Dropout(0.2),
            nn.ReLU(),
            nn.Linear(2*fn_embedding_size, 32),
            nn.Dropout(0.2),
            nn.ReLU(),
            nn.Linear(32, 16),
            nn.Dropout(0.2),
            nn.ReLU(),
            # nn.Linear(16, 8),
            # nn.Dropout(0.3),
            # nn.ReLU(),
        )
        self.before_out = nn.Linear(self.last_layer_size + mf_embedding_size, 1)
....
    def forward(self, user, item):
        mf_embedded_user = self.mf_embedding_user(user)
        mf_embedded_item = self.mf_embedding_item(item)
        fn_embedded_user = self.fn_embedding_user(user)
        fn_embedded_item = self.fn_embedding_item(item)
        mf_mul = torch.mul(mf_embedded_user, mf_embedded_item)
        fn_concat = torch.cat([fn_embedded_user, fn_embedded_item], dim=-1)
        fn_result = self.linear_stack(fn_concat)
        merged = torch.cat([mf_mul, fn_result], dim=-1)
        ratings = self.before_out(merged)
        return ratings.sum(1)
```

해당 구현은 먼저 user와 item의 matrix multiplication을 통해 연관성을 찾고 user와 item의 fully connected layer을 통한 행렬과 concatenate시킨다. 다음과 같이 한 이유는 matrix factorization만을 통해서 정보를 표현할 수 있는 한계가 있기 때문에(선형 결합) 비선형의 성질을 추가하기 위해 Fully connected layer와 활성화 함수를 통해 비선형의 성질을 추가해서 조금더 표현의 범위를 늘렸다.
그리고 학습하는 함수는 다음과 같다.

```
def train(model, epoch=15, lr=0.01, wd=0):
    optimizer= torch.optim.AdamW(model.parameters(), lr, weight_decay=wd)
    for i in range(epoch):
        model.train()
        users = torch.LongTensor(train_set[0].values).to(device)
        items = torch.LongTensor(train_set[1].values).to(device)
        ratings = torch.FloatTensor(train_set[2].values).to(device)
        predict = model(users, items)
        l2_lambda = 0.001
        l2_reg = torch.tensor(0.).to(device)
        for param in model.parameters():
            l2_reg += torch.norm(param)
        loss = F.mse_loss(predict, ratings)
        loss += l2_lambda * l2_reg
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        if i % 200 == 0:
            print("epoch %d loss %.3f" % (i+1, loss.item()))
```

학습하는 두 함수는 크게 다른점이 없으나 OCCF를 위한 학습은 타겟 점수가 1점이고 score prediction을 위한 학습은 진짜 점수가 들어간다.

```
def predict_score(model):
    model.eval()
    users = torch.LongTensor(test_set[0].values).to(device)
    items = torch.LongTensor(test_set[1].values).to(device)
    score = model(users, items).tolist()
    score = list(map(lambda x: 5 if x >= 5 else 1 if x <=1 else round(x), score))
    return score
```

그렇게 최종적으로 점수를 얻을 수 있을텐데, 점수가 범위 바깥으로 넘어간 값도 있고 정수값도 아닐 수 있으므로 계산값이 5를 넘으면 5점으로, 1점이하면 1점으로, 아니면 반올림해서 점수를 매겼다.

OCCF를 사용할 때 theta value를 0.3으로 매겼으며, base file을 분석해본 결과 평균점수가 대체적으로 높게 매겨짐을 확인했기 때문에, OCCF를 통해 얻어진 값을 base score을 0점, 1점, 2점, 3점중 최종결과에 가장 높게 나오는 점수를 사용했고, 3점이 가장 좋은 정확도를 보였다.

```
predict_OCCF = predict_R(model_OCCF)
less_than = predict_OCCF < 0.3
train_set_adder = []
val_set_adder = []
base_score = 3
for user_index, line in enumerate(less_than):
    item_index = np.where(line.cpu() == True)[0]
    df_exist = np.array(df[(df[0] == user_index)][1])
    item_index = np.setdiff1d(item_index, df_exist)
    # if np.random.rand(1) < 0.8:
    #     for item_i in item_index:
    #         train_set_adder.append([user_index, item_i, base_score, 0])
    # else:
    #     for item_i in item_index:
    #         val_set_adder.append([user_index, item_i, base_score, 0])
    list_df = []
    for item_i in item_index:
        list_df.append([user_index, item_i, base_score, 0])
    if list_df:
        list_df = pd.DataFrame(list_df)
        df = pd.concat([df,list_df], axis=0)

train_set = df
```

그리고 테스트 파일을 불러오고 주어진 형식대로 저장하고 끝낸다.

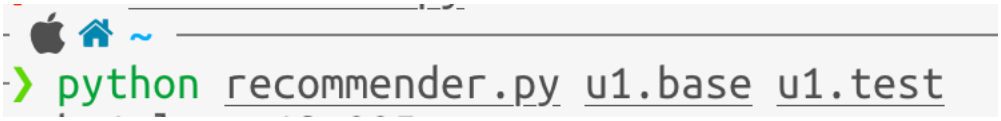
```
test_dataset = []
with open(test_file, "r") as f:
    texts = f.readlines()
    for text in texts:
        text = text.strip('\n').split('\t')
        test_dataset.append([int(text[0]), int(text[1]), int(text[2]), int(text[3])])

test_set = pd.DataFrame(test_dataset, dtype=np.int32)

test_set[0] = test_set[0].apply(lambda x: userId2idx.get(x, 0))
test_set[1] = test_set[1].apply(lambda x: itemId2idx.get(x, 0))

predict = predict_score(model)

with open(predict_file, 'w') as f:
    for idx, data in enumerate(test_dataset):
        f.write('{}\t{}\t{}\t{}\n'.format(data[0], data[1], predict[idx], data[3]))
```

각 테스트 파일을 실행시킨 결과 RMSE는 다음과 같다.

```
root@fa0a3e3773b9:/test/Recommender# mono PA4.exe u1
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: 0.9860781
root@fa0a3e3773b9:/test/Recommender# mono PA4.exe u2
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: 0.9928243
root@fa0a3e3773b9:/test/Recommender# mono PA4.exe u3
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: 0.9838953
root@fa0a3e3773b9:/test/Recommender# mono PA4.exe u4
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: 0.9859767
root@fa0a3e3773b9:/test/Recommender# mono PA4.exe u5
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: 0.990202
```

해당 프로그램을 google colab을 통해 작성했고, 테스트 장소도 colab이지만 파이썬 파일로 변환하여 저장하였다. GPU(cuda)환경을 사용하고, 없으면 실행시간이 매우 길어지나(u1 기준으로 11분 소모) GPU사용시 계산이 매우 빨라진다.

실행방법은 다음과 같다.



파이썬 버전은 3.7.10이다.