Data Science (ITE4005)

< Long-term Project >

Recommender System using Deep Learning

2016025732 이영석

2021-06-11

1. Summary

사용자의 영화 평점을 추측하여 추천하는 추천시스템을 Matrix Factorization을 이용해 구현했다.

R (기본 트레이닝 데이터 셋) = U X V 로 분할하였고, Zero Injection은 구현하지 않았다.

다양한 방법 중 딥러닝을 통해 구현했다. Loss Function은 다음과 같다.

$$\min_{Q*,P*} \sum_{(u,i)\in K} (r_{ui} - P_u^T Q_i)^2 + \lambda(||Q_i||^2 + ||P_u||^2)$$

P와 Q가 각각 U와 V에 대응한다. 람다와 L2-Norm은 Regularization에 이용되며, 과최적화를 막는 역할에 이용된다. 다만 위 식에서 평점을 매기지 않은 rui에 대해서는 update를 하면 안되기 때문에, 구현 시 nonzero를 이용하여 값이 있는 rui에서만 Back Propagation을 적용하였다. 위 식을 미분하면 다음과 같다.

$$L = \sum (r_{ij} - U_i \cdot V_j)^2 + \lambda(\|U_i\|^2 + \|V_j\|^2)$$

$$dU_i = 2(r_{ij} - U_i \cdot V_j) V_j + 2\lambda(U_i)$$

$$dV_j = 2(r_{ij} - U_i \cdot V_j) U_i + 2\lambda(V_j)$$

위 식을 이용해 Back Propagation을 적용하면 다음과 같다.

```
error = R_train[i,j] - U[i].dot(V[j].T)
U[i] -= learning_rate * (-error * V[j] + lambda_ * U[i])
V[j] -= learning_rate * (-error * U[i] + lambda_ * V[j])
```

또한 딥러닝의 특성상 Hyper Parameter에 민감했는데, 그 세팅은 다음과 같다. 여러 실험과정을 통해 최선의 파라미터를 택했다.

```
learning_rate = 1e-2
lambda_ = 0.005
epochs = 40
F = 3 # shape parameter / U = N * F / V = M * F /
```

외에도 overfitting을 파악하기 위한 기준으로 0.1을 두었다.

코드는 다음과 같은 모듈/순서로 진행하였다. (자세한 내용은 2. Detailed Description 참고)

1) train()
    A. set_data()
    B. split_data()
    C. get_rmse()
2) test()

2. Detailed Description

A. train()

```
def train(base_name):
    global R_train, R_valid, R_hat
    df = set_data(base_name)
    R_train, R_valid = split_data(df)

    U = np.random.rand(N,F)
    V = np.random.rand(M,F)

    users, items = R_train.nonzero()
    for epoch in range(epochs):
        for i, j in zip(users,items):
            error = R_train[i,j] - U[i].dot(V[j].T)
            U[i] -= learning_rate * (-error * V[j] + lambda_ * U[i])
            V[j] -= learning_rate * (-error * U[i] + lambda_ * V[j])

        R_hat = U.dot(V.T)
        rmse_train = get_rmse(True)
        rmse_valid = get_rmse(False)
        if(epoch % 5 == 4):
            print('---------epoch : %d--------'%(epoch+1))
            print('Train rmse : ', rmse_train)
            print('Valid rmse : ', rmse_valid)

        if(rmse_train + 0.1 < rmse_valid): #prevent overfitting
            print("Overfitting occured in epoch %d"%(epoch+1))
            break

    R_hat[R_hat < 1] = 1
    R_hat[R_hat > 5] = 5
    # print(R_hat)
```

앞서 이야기한 주요한 내용이 해당 모듈에 포함된다. base_name (ex. u1.base)로 base의 데이터를 받아와 overfitting을 판별하기 위해 training set과 validation set으로 나눈다.

이후 Summary에서 포함된 내용과 같이 epoch만큼 fit을 시킨다. epoch을 도는 와중에 overfitting이 일어나면 early stopping한다. 이후 결과를 R_hat에 저장한다.

i. set_data()

```
def set_data(base_name):
    global N,M

    df = pd.read_csv(base_name, sep='\t', engine='python', encoding="cp949", header= None)
    del df[3]

    df = df.to_numpy()

    N = np.unique(df[:,0]).max()
    M = np.unique(df[:,1]).max()
    print(N, M)

    R = np.empty((N, M))
    for temp in df:
        R[temp[0]-1, temp[1]-1] = temp[2]

    return df
```

주어진 데이터(ex. u1.base) 를 세팅하는 함수. R을 numpy (N X M 행렬)로 세팅하고 df를 반환한다.

ii. split_data()

```
def split_data(df):
    df_train, df_valid = train_test_split(df, test_size=0.1)

    R_train = np.empty((N, M))
    for temp in df_train:
            R_train[temp[0]-1, temp[1]-1] = temp[2] #index 0부터 시작

    R_valid = np.empty((N, M))
    for temp in df_valid:
            R_valid[temp[0]-1, temp[1]-1] = temp[2] #index 0부터 시작

    return R_train, R_valid
```

Training set과 validation set으로 나눈다. 각각 numpy 행렬((N X F), (M X F))로써 반환한다.

iii. get_rmse

```
def get_rmse(flag):
    if flag : #True : train set, False : validation set
        xi, yi = R_train.nonzero()
    else :
        xi, yi = R_valid.nonzero()

    cost = 0
    for x, y in zip(xi, yi):
        if flag:
            cost += pow(R_train[x,y] - R_hat[x,y], 2)
        else:
            cost += pow(R_valid[x,y] - R_hat[x,y], 2)
    return np.sqrt(cost/len(xi))
```

1 epoch마다 rmse를 체크하기 위해 존재하는 함수. flag로 validation과 train을 구분한다. nonzero에 해당하는 값만 rmse를 계산하여 반환한다.

B. test()

```
def test(test_name):
    df_test = pd.read_csv(test_name, sep='\t', engine='python', encoding="cp949", header= None)
    del df_test[3]
    df_test = df_test.to_numpy()

    output_name = test_name.split(sep='.')[0] + '.base_prediction.txt'
    with open(output_name, 'w') as f:
        mse = 0
        for i in range(len(df_test)):
            user = df_test[i][0] - 1
            item = df_test[i][1] - 1

            # Exception Handling
            if user >= R_hat.shape[0]:
                predict = R_hat.mean(axis=item)
            elif item >= R_hat.shape[1]:
                predict = R_hat.mean()
            else:
                predict = R_hat[user, item]

            err = df_test[i][2] - predict
            mse += err * err
            line = str(user+1) + '\t' + str(item+1) + '\t' + str(predict) + '\n'
            f.write(line)
    mse /= df_test.shape[0]+1
    rmse = math.sqrt(mse)
    print('rmse for test data : ',rmse)
```

test set을 가져오고 한 줄씩 읽어가며 rmse를 계산. 또한 output file에 계산한 R_hat에서 예측한

값도 적어준다. Exception Handling 부분에서 R_hat 행렬에 없는 새로운 유저나 아이템이 나타날 경우를 대비한다. (순서대로 해당 아이템의 평균 평점, 전체의 평균 평점으로 대체)

3. Instruction for compiling

A. python 설치

B. Base 파일, Test 파일, RecSys_2016025732.py 같은 디렉토리내 위치

C. Execute the program with four arguments : training data name, test data name

```
\LongtermProject\data-2> python RecSys_2016025732.py u2.base u2.test
```

4. Testing

A. cmd 입력/출력 (3. Instruction for compiling 참고)

```
PS E:\네이버_동기화폴더\Naver MYBOX\4학년 1학기\데이터 사이언스\실습\LongtermProject\data-2> python .\RecSys_2016025732.py u2.base u2.test
943 1682
---------epoch : 5--------
Train rmse :  0.9411466282429625
Valid rmse :  0.9770129286882586
---------epoch : 10--------
Train rmse :  0.9165528044411543
Valid rmse :  0.9658668084665659
---------epoch : 15--------
Train rmse :  0.8962966013755552
Valid rmse :  0.9606594200623427
---------epoch : 20--------
Train rmse :  0.8814402537633486
Valid rmse :  0.9582845323489102
---------epoch : 25--------
Train rmse :  0.8716932625599136
Valid rmse :  0.9582408710485999
---------epoch : 30--------
Train rmse :  0.8651939088809938
Valid rmse :  0.9590898924168684
---------epoch : 35--------
Train rmse :  0.8606704830865547
Valid rmse :  0.9602348769450882
Overfitting occured in epoch 36
rmse for test data :  0.9575700593711725
```

➔ 5 epoch당 Train set과 Valid set의 rmse를 출력한다.

➔ Overfitting이 일어나면 Early Stopping 진행

➔ Test set의 rmse를 출력한다

B. 채점 프로그램 실행 결과

```
C:₩Users₩이영석₩Desktop₩test>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: 0.9575964
```

결과는 실행시마다 차이가 있지만 대부분 0.95대에 분포했다.