Long term project Report

2016024766

김서현

1. 구현한 Recommendation System 알고리즘에 대한 설명

기본적으로 Matrix Factorization (MF) 방식을 이용해 rating을 예측하였습니다. 구현한 전체 알고리즘은 다음과 같습니다.

1) 주어진 base 파일을 읽어서 R이라는 matrix를 만든다.
2) Base 파일과 test 파일을 모두 읽어서 user_id와 item_id의 최댓값을 알아내서 저장해 둔다.
3) R에서 missing value가 아닌 값들을 모두 1로 바꾼 matrix를 만들고, 그 matrix에 대해 MF를 해서 pre-preference를 알아낸다.
4) Pre-preference를 기준으로 하위 5%에 해당하는 값들의 위치에 대해서는 R에 1을 채워 넣는다.
5) R을 가지고 MF를 해서 missing ratings에 대한 예측 값을 모두 계산하여 결과 matrix를 만든다.
6) 2)에서 저장한 최댓값을 가지고 결과 Matrix를 reindex를 한 후, 예측되지 않은 값에는 예측된 ratings의 전체 평균값을 넣는다.

MF를 할 때에는 **Gradient descent** 방법을 사용했습니다. 자세한 MF 방법은 다음과 같습니다.

1) R을 분해할 행렬인 P와 Q를 랜덤한 값을 채워 넣은 상태로 생성한다.
2) $$p'_{ik} = p_{ik} + \alpha \frac{\partial}{\partial p_{ik}} e_{ij}^2 = p_{ik} + \alpha(2e_{ij}q_{kj} - \beta p_{ik})$$
$$q'_{kj} = q_{kj} + \alpha \frac{\partial}{\partial q_{kj}} e_{ij}^2 = q_{kj} + \alpha(2e_{ij}p_{ik} - \beta q_{kj})$$

   이 식을 update rule로 사용해서 P와 Q 행렬을 업데이트한다.
   - Cost에 대한 식을 p와 q로 각각 미분해서 계산된 update rule
   - 베타는 overfitting을 막는 regularization을 위해서 계산에 추가되었음
3) P와 Q를 내적해서 prediction matrix를 생성하고 cost를 계산해서 일정 수준보다 떨어지면 종료한다.
4) 위의 과정을 정해진 Epochs 횟수만큼 반복한다.

2. 코드에 대한 설명

1) matrix_factorization, matrix_factorization_pre

```
# Matrix factorization을 하는 함수
def matrix_factorization(R, P, Q):
    Epochs = 100
    alpha = 0.005
    beta = 0.02
    Q = Q.T
    for epoch in range(Epochs):
        if epoch == 60:
            alpha = 0.0005
        for i in range(len(R)):
            for j in range(len(R[i])):
                if R[i][j] > 0:
                    # update P, Q
                    eij = R[i][j] - (np.dot(P[i, :], Q[:, j])+beta)
                    P[i, :] += alpha * (2 * eij * Q[:, j] - beta * P[i, :])
                    Q[:, j] += alpha * (2 * eij * P[i, :] - beta * Q[:, j])
        eR = np.dot(P, Q)+beta
        cost = 0
        cnt = 0
        # calculate cost
        for i in range(len(R)):
            for j in range(len(R[i])):
                if R[i][j] > 0:
                    cost += (R[i][j] - eR[i][j])**2
                    cnt += 1
        cost = (cost/cnt)**0.5
        print(epoch, cost)
        # cost가 0.1보다 작으면 학습 종료
        if cost < 0.1:
            break
    return np.dot(P, Q)
```

Matrix factorization을 수행하는 함수입니다. 정해진 Epoch만큼 P와 Q 행렬을 업데이트 하는 작업을 반복합니다. R 행렬에서 0인 값은 missing value이기 때문에 0보다 큰 값일 때에만 비교를 해서 P와 Q를 업데이트 하도록 구현했습니다. Learning rate인 alpha값은 60회 반복 후에는 더 작은 값으로 바꿔서 더 조금씩 업데이트 되도록 하였습니다. Overfitting을 막기 위해 beta를 이용해 값을 조금씩 빼면서 업데이트 하도록 구현했습니다. Cost를 계산해서 값이 0.1보다 작아지면 Epochs이 남았더라도 종료하고 최종 prediction matrix를 리턴하도록 구현했습니다. Matrix_factorization_pre 함수에서도 Epochs 같은 파라미터만 조금 다를 뿐 알고리즘은 동일하게 동작합니다.

2) Get_new_column_and_index

```
def get_new_column_and_index(base_df, test_df):
    mt = test_df.max().values
    mf = base_df.max().values
    user_max = max(mt[0], mf[0])
    item_max = max(mt[1], mf[1])
    column_list = range(1, item_max+1)
    index_list = range(1, user_max+1)
    return column_list, index_list
```

Base 파일과 test 파일의 내용을 dataFrame 형태로 받아서 user_id와 item_id의 최댓값을 찾습니다. Base에 없었던 user나 item이 test 파일에는 있을 수 있기 때문에, 1부터 최댓값까지의 모든 id를 최종 index와 column으로 설정했습니다.

3) Insert_pre_preference

```
# pre-preference가 낮으면 R에 1을 넣음
def insert_pre_preference(R, N, M, K):
    toOne = np.array([np.array([1 if x > 0 else 0 for x in row]) for row in R])
    P2 = np.random.rand(N, K)*0.01+0.1
    Q2 = np.random.rand(M, K)*0.01+0.1
    # MF로 pre-preference를 구함
    pre = matrix_factorization_pre(toOne, P2, Q2)
    Q1 = np.percentile(pre, 5)
    # preferencr가 하위 5%면 R에 1을 삽입함
    for i in range(len(pre)):
        for j in range(len(pre[i])):
            if pre[i][j] < Q1 and R[i][j] == 0:
                R[i][j] = 1
```

우선 R에서 missing value가 아니면 모두 1로 만든 행렬 toOne을 생성합니다. toOne에 대해서 MF를 계산해서 pre-preference를 계산합니다. Pre-preference의 하위 5% 기준을 계산하고, 그 이하인 값들은 R 행렬에서 1을 갖고 있도록 설정합니다.

4) Main

```
if __name__ == '__main__':
    if len(sys.argv) < 3:
        print(
            "USAGE: python recommender.py <base filename> <test filename>")
        sys.exit()
    basefile = f'./data/{sys.argv[1]}'
    testfile = f'./data/{sys.argv[2]}'
    base_df = pd.read_csv(basefile, sep='\t', header=None, names=[
        'user_id', 'item_id', 'rating', 'time_stamp'])
    base_df.drop('time_stamp', axis=1, inplace=True)
    test_df = pd.read_csv(testfile, sep='\t', header=None, names=[
        'user_id', 'item_id', 'rating', 'time_stamp'])
    test_df.drop('time_stamp', axis=1, inplace=True)
    rating_matrix = base_df.pivot(
        values='rating', index='user_id', columns='item_id').fillna(0)
    # 원래 갖고있던 column과 index
    column_list = rating_matrix.columns
    index_list = rating_matrix.index
    R = rating_matrix.values
    # output file에 쓸 column과 index
    res_column_list, res_index_list = get_new_column_and_index(
        base_df, test_df)
```

우선 argument로 입력 받은 값들을 읽어오고, 잘못됐다면 USAGE를 띄우고 프로그램을 종료합니다. Base 파일과 test 파일을 열어 안의 data를 읽고, base 파일의 데이터를 이용해 rating matrix인 R을 생성합니다. get_new_column_and_index() 함수를 이용해 최종 output file에 쓸 index와 column도 미리 계산합니다.

```python
# number of Users
N = len(R)
# number of Movies
M = len(R[0])
# number of Features
K = 3
# pre-preference가 낮은면 R에 1을 삽입함
insert_pre_preference(R, N, M, K)
start_time = time.time()
P = np.random.rand(N, K)*0.01+1
Q = np.random.rand(M, K)*0.01+1
predict_R = matrix_factorization(R, P, Q)
# 값이 너무 작거나 크게 예측된걸 보정함
for i in range(len(predict_R)):
    for j in range(len(predict_R[i])):
        if predict_R[i][j] < 1:
            predict_R[i][j] = 1
        if predict_R[i][j] > 5:
            predict_R[i][j] = 5
res_mean = np.mean(predict_R)
R_df = pd.DataFrame(predict_R, index=index_list, columns=column_list)
# 미리 저장해뒀던 index, column으로 reindex함
# 모르는 rating은 전체의 평균으로 설정함
result = R_df.reindex(index=res_index_list,
                      columns=res_column_list).fillna(res_mean)
predict = result.stack(dropna=False)
print(f'{time.time()-start_time}s')
output_filename = f'./test/{sys.argv[1]}_prediction.txt'
predict.to_csv(output_filename, sep='\t', index=True, header=False)
```

R을 기반으로 user의 수와 item의 수를 계산하고, feature의 수인 K값에 맞게 P, Q matrix를 생성합니다. P와 Q의 초기값은 랜덤하게 들어갑니다. (다만 여러 번 실험을 하면서 K는 작을수록, P, Q의 초기값은 1에 가까울수록 정확도가 높게 prediction 되었습니다.)

Insert_pre_preference() 함수를 이용해 R에 1을 5%정도 채워 넣습니다. 그 R을 matrix_factorization() 함수의 인자로 넣어 prediction matrix를 생성합니다. Prediction matrix의 값이 1과 5 사이로 들어오도록 조정한 뒤, 최종 index, column을 행렬에 붙이고 dataFrame 형태로 바꿔 최종 output file에 write합니다.

3. 컴파일 방법

Python 3.9.1 버전으로 작성한 프로그램입니다.

테스트는 windows10 내의 command prompt를 사용하여 진행했습니다.

1) Pandas library와 numpy library를 사용하기 때문에 각 라이브러리를 설치해야 합니다.

- pip install pandas
- pip install numpy

2) recommender.py 소스파일, data 폴더, test 폴더를 같은 디렉토리 안에 넣습니다.

3) data 폴더 안에는 base 파일들과 test 파일들을 넣습니다.

4) test 폴더 안에는 test 파일들과 prediction한 결과 파일들, 정답 파일, test 프로그램 (PA4.exe)을 모두 넣습니다.

5) "python recommender.py <base filename> <test filename>"의 형식으로 실행시킵니다.

Ex) `python recommender.py u1.base u1.test`

4. 실행 결과

1) U1

```
C:\Users\yds04\Documents\4학년 1학기\데이터 사이언스\과제\assignment4\test>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: 0.9359129
```

실행시간: 326s

2) U2

```
C:\Users\yds04\Documents\4학년 1학기\데이터 사이언스\과제\assignment4\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: 0.9317356
```

실행시간: 324s

3) U3

```
C:\Users\yds04\Documents\4학년 1학기\데이터 사이언스\과제\assignment4\test>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: 0.9247038
```

실행시간: 330s

4) U4

```
C:\Users\yds04\Documents\4학년 1학기\데이터 사이언스\과제\assignment4\test>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: 0.9281796
```

실행시간: 324s

5) U5

```
C:\Users\yds04\Documents\4학년 1학기\데이터 사이언스\과제\assignment4\test>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: 0.9258194
```

실행시간: 322s

6) 결과 분석

- 처음 실험할 때는 Pre-use preference를 미리 구하지 않고 그냥 MF를 한 것이 더 좋은 결과를 내는 경우가 많았지만, 계속 파라미터를 조절하면서 실험하여 최종적으로는 pre-use preference를 미리 계산하는게 더 결과가 잘 나오게 되었습니다.
- P와 Q의 초기값을 처음에는 0과 1 사이의 랜덤한 값으로 잡았지만 실험을 하다보니 1에 가까운 값으로 초기값을 잡았을 때 RMSE가 작게 나오는 것을 알게 되었습니다.
- Feature의 개수인 K가 크면 결과가 더 잘 나올 줄 알았으나 K=3일 때 결과가 가장 잘 나왔습니다.
- Epochs도 무조건 크다고 결과가 더 좋게 나오는 것은 아니었습니다.

5. 특이사항

1) 결과 파일은 바로 test 폴더 내에 생성되도록 구현했습니다.
2) Recommender.py를 실행시키면 실행되는 동안 epoch과 현재 cost를 출력합니다.
3) Recommender.py가 동작을 종료할 때는 총 걸린 시간을 출력합니다.
4) 주어진 base 파일, test 파일, 결과로 나온 prediction 파일들도 git에 함께 제출하였습니다.