소프트웨어 응용 중간보고서

컴퓨터과학부 2015920062 홍성현

컴퓨터과학부 2016920027 오준영

목차

1. 프로젝트 진행 과정

우리 조는 python을 사용하여 recommendation system을 만들기로 하였다. 처음에는 item-item collaborative filtering을 이용하여 recommendation system을 구현하려 했지만 몇 가지 문제가 생겨 이 방법은 사용하지 못했다. 첫 번째 문제는 item 간의 similarity를 구하는 과정에서 분모가 0이 되는 경우가 발생하는 것이었다. 우리는 pearson correlation coefficient를 사용하여 similarity를 구했는데, 특정 item의 평점 평균을 구한 뒤 평점에서 그 값을 빼서 cosine similarity를 구하는 식의 특성상 특정 item에 대한 사용자들의 평점이 모두 같을 때는 벡터 성분들이 모두 0이 되고, 따라서 분모가 0이 되어버리는 문제가 발생했다.

$$\rho = \frac{\sum_i (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_i (x_i - \bar{x})^2}\sqrt{\sum_i (y_i - \bar{y})^2}}$$

식 1. *Pearson correlation coefficient*

```
5555991584 [5.0, 5.0, 5.0, 5.0, 4.0, 5.0, 3.0, 5.0, 5.0, 5.0, 5.0, 4.0, 5.0, 5.0, 4.0, 5.0, 5.0, 5.0, 4.0, 5.0, 5.0, 5.0, 5.0]
B0000000ZW [4.0, 5.0, 4.0, 5.0, 5.0, 3.0, 5.0, 1.0, 3.0, 5.0, 3.0, 5.0, 5.0, 4.0, 5.0]
B00000016T [5.0, 5.0, 5.0, 5.0, 5.0, 5.0, 5.0, 5.0, 5.0, 5.0, 4.0, 5.0, 2.0, 5.0, 5.0, 5.0, 5.0, 5.0, 5.0, 5.0, 4.0, 5.0, 5.0, 5
B00000016W [5.0, 1.0, 5.0, 5.0, 5.0, 5.0, 5.0, 5.0, 5.0, 5.0, 1.0, 1.0, 5.0, 5.0, 5.0, 4.0, 5.0, 5.0, 5.0, 5.0, 3.0, 5.0, 5.0, 5.0, 5
B00000017R [5.0, 4.0, 5.0, 5.0, 5.0, 4.0, 3.0, 5.0, 3.0, 5.0, 5.0, 5.0, 5.0, 5.0, 5.0, 4.0, 5.0, 5.0, 5.0, 5.0]
B0000001P4 [5.0, 5.0, 5.0, 5.0, 5.0]
B0000002HZ [5.0, 5.0, 5.0, 4.0, 4.0, 5.0]
B0000002J9 [5.0, 5.0, 5.0, 5.0, 4.0, 5.0]
B0000002JR [5.0, 4.0, 5.0, 5.0, 5.0, 4.0, 5.0, 5.0, 5.0, 5.0]
```

그림 1. 각 item들의 알려진 평점 정보들

그림 1은 item들의 현재 알려진 평점 정보 중 일부를 나열한 것이다. 그림에서 item 'B0000001P4'의 평점이 모두 같은 것을 확인할 수 있으며, 이 경우 평점들의 평균이 평점과 같게 된다. 이렇게 되면 식 1의 $\sqrt{\sum_i (x_i - \bar{x})^2}$ 부분이 0이 되고 따라서 분모도 0이 되어버린다. 우리는 우선 이러한 경우는 배제하고 similarity 값을 구하기 위해서 이런 경우는 similarity 값을 0으로 하여 계산해 본 결과 item간의 similarity 값은 구할 수 있었지만 또다른 문제에 직면하게 되었다. 그 문제는 특정 사용자가 아직 평가하지 않은 item의 평점을 예측하려 할 때 그 item과 유사하면서 사용자가 평점을 부여한 항목이 없는 경우가 생길 수 있는 것이었다. 예를 들어, item ID 'B00GRYKUAI'의 다른 item들과의 similarity는 다음과 같다.

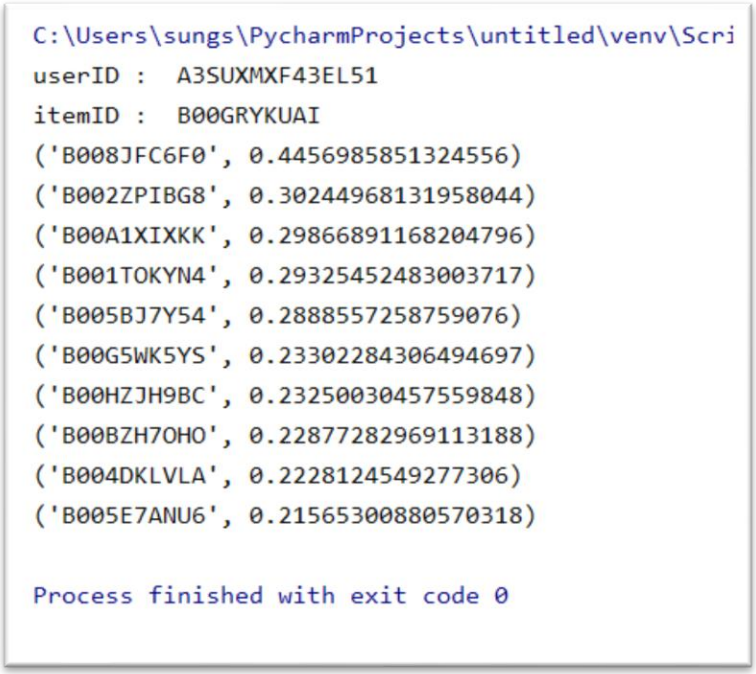

그림 2. Item 'B00GRYKUAI'과 다른 item들 간의 유사도. 상위 10개만 표시하였다.

이제 사용자 'A3SUXMXF43EL51'이 이 item에 대해 매긴 평점을 예측하고 싶다고 가정해보자. 위의 유사도 리스트 중 해당 사용자가 평점을 매긴 항목들만 추려서 다시 표시해보면 아래 그림과 같은 결과가 나온다.

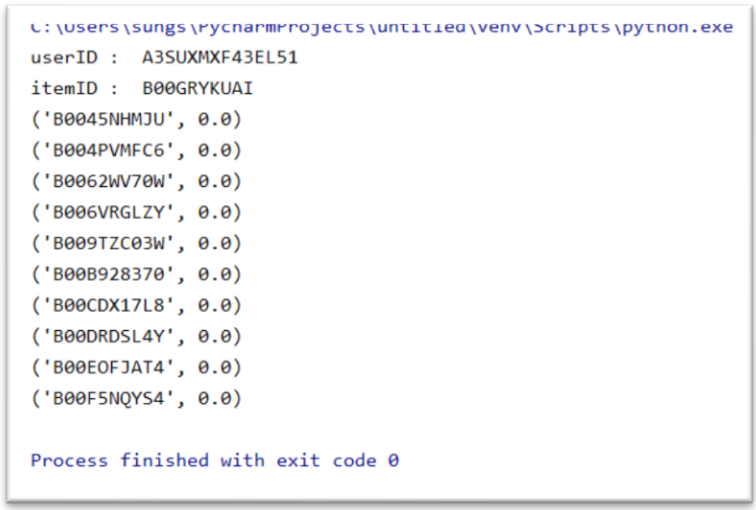

그림 3. 사용자 'A3SUXMXF43EL51'이 평점을 매긴 item들만을 이용하여 유사도 순위를 표시

우리는 이런 문제가 발생하는 이유를 생각해보았고, 평점 데이터들이 부족하여 평점들이 저장되어 있는 matrix가 sparse matrix 상태이기 때문에 이러한 문제가 발생한다는 것을 알 수 있었다. 이 시점에서 우리는 item-item collaborative filtering이 아닌 다른 방법을 찾아보다가 Stochastic Gradient Descent가 큰 크기의 sparse matrix에서 좋은 성능을 보이는 것을 알게 되었고, 이 방법을 사용하여 프로그램을 구현하기로 결정하였다.

2. 구현 기능

프로그램을 실행시키면 먼저 콘솔 상으로 메뉴화면이 출력된다. 메뉴화면에서는 숫자 1부터 3 중 하나의 숫자를 입력 받게 되고, 각 숫자에 따라 user mode, admin mode, 프로그램 종료 중 하나의 상태로 이동하게 된다. Admin mode와 user mode에서의 구현 기능들은 아래와 같다.

NUM	ADMIN MODE	USER MODE
1	평점 행렬 구하기	추천 항목 출력
2	평점 행렬 불러오기	
3	평점 행렬 보여주기	
4	에러 그래프 출력	

Admin mode의 각 항목부터 차례대로 살펴보자.

2-1. 평점 행렬 구하기

이 기능은 먼저 기존 행렬의 SVD를 구한 이후에 실행될 수 있다. 프로그램을 처음 실행했을 때 클래스 Recommend_engine_SGD의 인스턴스가 생성되고, 이 클래스의 생성자에서 load_SVD() 메소드를 호출하게 된다. 만일 이미 SVD 파일이 존재할 경우 이 파일을 불러와서 사용하며, 존재하지 않는 경우 내부에서 cal_SVD() 메소드를 호출하여 새로운 SVD파일을 만들어 사용한다. 두 메소드의 구조는 다음과 같다.

```python
# cal_SVD() : Perform SVD with original rating matrix and save the results.
def cal_SVD(self):
    factor_k =100
    U, s, V = svds(self.matrix, factor_k)
    S = np.zeros((len(s), len(s)))
    for i in range(len(s)):
        S[i][i] = s[i]
    PT = np.dot(S, V)
    np.save('SVD_Q.npy', U)
    np.save('SVD_PT.npy', PT)
```

그림 4. cal_SVD()

```python
# load_SVD() : load SVD
def load_SVD(self):
    try:
        self.mat_q = np.load('SVD_Q.npy')
        self.mat_pt = np.load('SVD_PT.npy')
    except IOError:
        print('Make SVD...')
        self.cal_SVD()
        self.load_SVD()
```

그림 5. load_SVD()

SVD는 python의 scipy 라이브러리의 함수 svds()를 통해 생성되며 100개의 factor를 가지게 된다. 이 SVD는 SGD의 처음 Q행렬과 P Transpose[1] 행렬로 사용된다. 다음은 기존 평점 행렬에서 test set과 train set을 분리하는 과정을 수행한다. 전체 평점의 개수 64,706개 중 랜덤으로 5,000개의 데이터를 뽑아 test set으로 사용하였으며, 나머지 평점들은 train set으로 구성하였다. 이 기능은 make_train_test() 메소드를 통해 구현되었다.

[1] 이 다음부터는 PT로 표기한다.

```python
#make_train_test() : make train set and test set from the original matrix.
def make_train_test(self):
    train_set=self.matrix.copy()
    test_set=np.zeros(self.matrix.shape)

    nonzero_row, nonzero_col=self.get_nonzero(train_set)
    idx_list=np.random.choice(range(len(nonzero_row)), 5000, replace=False) # num of test set : 5000

    for idx in idx_list:
        test_set[nonzero_row[idx], nonzero_col[idx]]=train_set[nonzero_row[idx], nonzero_col[idx]]
        train_set[nonzero_row[idx], nonzero_col[idx]]=0

    return train_set, test_set
```

그림 6. Make_train_test()

랜덤으로 데이터를 뽑는 작업은 numpy.random의 choice() 함수를 이용하였다. 첫 번째 인자로 평점 행렬에서 0이 아닌 원소들의 개수를 주었고, 두 번째 인자를 통해 5,000개의 test sample을 뽑아냈다. 중복 원소를 뽑지 않기 위해 replace=false 옵션을 추가하였다. 이렇게 뽑아낸 sample들은 별도의 test 행렬을 만들어 동일한 자리에 넣어주고, training 행렬에서 sample을 뽑아낸 자리에는 0을 넣어주었다. 이 작업을 통해 training set에서는 test set에 있는 평점들이 존재하지 않는다고 여기고 학습을 진행할 것이고, 학습이 끝난 뒤에 test set에 들어있는 평점들과 원래 위치에 있는 점수들을 비교하여 학습이 얼마나 잘 되었는지를 파악할 수 있을 것이다.

Test set 구성 작업이 끝나면 본격적으로 SGD를 이용하여 training set을 훈련시킨다. SGD training 함수는 다음과 같이 구성되어 있다.

```python
#train() : Train the training set with SGD.
def train(self, iter):
    self.train_set_error=[] #clear train set error.
    self.test_set_error=[] #clear test set error.
    if self.trained==1: #if rating matrix is already trained, check
        print('rating matrix is already trained. do you want to train again?(y, n)')
        k=input()
        if k=='n':
            return

    nonzero_row, nonzero_col=self.get_nonzero(self.train_set) #Returns the row number and column number of non-zero elements.
    start=time.time()
    prev_rmse=0.
    for i in range(iter):
        print("iter ", i)
        for idx in range(len(nonzero_row)):
            row=nonzero_row[idx] # Row number of nonzero element in train set.
            col=nonzero_col[idx] # Column number of nonzero element in train set.
            baseline=self.get_baseline(row, col) #set baseline estimate
            err=2*(self.train_set[row, col] - (baseline + np.dot(self.mat_q[row, :], self.mat_pt[:, col])))
```

```
            if err > 100000 :
                print("err exceeded over 100000. change learning rate and try again.")
                return

            self.mat_q[row, :] += self.lr * (err * self.mat_pt[:, col] - self.mat_q[row, :]) #update q
            self.mat_pt[:, col] += self.lr * (err * self.mat_q[row, :] - self.mat_pt[:, col]) #update pt

            self.item_bias[row] += self.lr * (err - self.item_bias[row]) #update item bias
            self.user_bias[col] += self.lr * (err - self.user_bias[col]) #update user bias

        train_rmse=self.get_rmse(self.train_set)
        test_rmse=self.get_rmse(self.test_set)

        print("train set error : ", train_rmse)
        print("test set error : ", test_rmse)
        self.train_set_error.append(train_rmse)
        self.test_set_error.append(test_rmse)
```

```
        if abs(prev_rmse - test_rmse) <= 1e-3: # End training when the RMSE difference in test set is less than 0.001.
            print("train finished!")
            print('-'*5, 'reult matrix','-'*5)
            self.rating_matrix=self.get_final_matrix()
            print(self.rating_matrix)
            np.save('rating_matrix.npy', self.rating_matrix)
            np.save('train_set_error.npy', self.train_set_error)
            np.save('test_set_error.npy', self.test_set_error)
            print("time : ", time.time() - start)
            self.trained=1
            return
        prev_rmse=test_rmse
```

그림 7. Training 함수

train() 함수는 학습 횟수를 지정하는 매개변수 iter를 받는다. 먼저 이미 학습이 완료되었던 적이 있는지 확인하고, 만일 그렇다면 사용자에게 다시 한 번 학습을 진행할 것인지를 확인하고, 그 선택에 따라 학습을 다시 진행할지 메뉴로 돌아갈지를 결정한다. 학습을 처음 진행하거나 다시 한 번 진행하는 것을 선택했을 경우 행렬 Q와 PT를 학습시키는 과정을 다음과 같이 진행한다.

1. Training set의 평점 데이터 중 하나를 선택
2. Baseline estimate 계산
3. 목적 함수의 미분값을 구한다.
4. 선택한 평점 데이터의 행 번호와 같은 행렬 Q의 행을 training
5. 선택한 평점 데이터의 열 번호와 같은 행렬 PT의 열을 training
6. User bias와 item bias 업데이트
7. Train set과 test set의 RMSE 구하기
8. Test set의 RMSE값 변화가 0.001 이하가 될 때까지 1번부터 7번 과정을 반복한다.

Baseline estimate 계산은 get_baseline(row, col) 함수를 통해서 이루어진다. 이 함수는 기존 평점 행렬의 전체 평점 평균에 row번째 item의 bias와 col번째 사용자의 bias를 전부 더해서 리턴해준다. Item의 bias와 사용자의 bias는 학습 과정에서 조정되며, bias 초기값은 각 item 평점과 사용자 평점의 평균에서 전체 평균을 뺀 값으로 정해진다. Baseline 계산이 끝나면 행렬 Q의 row 번째 행과 행렬 PT의 col 번째 열의 학습을 다음과 같이 수행한다.

```
self.mat_q[row, :] += self.lr * (err * self.mat_pt[:, col] - self.mat_q[row, :]) #update q
self.mat_pt[:, col] += self.lr * (err * self.mat_q[row, :] - self.mat_pt[:, col]) #update pt
```

식 3. 행렬 Q와 PT의 업데이트

self.lr은 학습률을 저장하고 있으며, 현재 이 프로그램에서는 0.0001이다. err값은 다음과 같은 식을 통해 구해진다.

```
err=2*(self.train_set[row, col] - (baseline + np.dot(self.mat_q[row, :], self.mat_pt[:, col])))
```

식 4. err값 구하기

만일 err 값이 100000을 넘어설 경우 변화량이 수렴하지 않고 발산하고 있는 상태이다. 이 경우에는 학습률을 낮추어야 한다.

위의 과정이 끝나면 아래와 같은 식을 통해 item bias 및 user bias를 업데이트한다.

```
self.item_bias[row] += self.lr * (err - self.item_bias[row]) #update item bias
self.user_bias[col] += self.lr * (err - self.user_bias[col]) #update user bias
```

식 5. item bias 및 user bias 수정

이와 같은 식은 목적함수를 item bias와 user bias로 미분하여서 얻을 수 있었다. 사용한 목적함수는 다음과 같다,

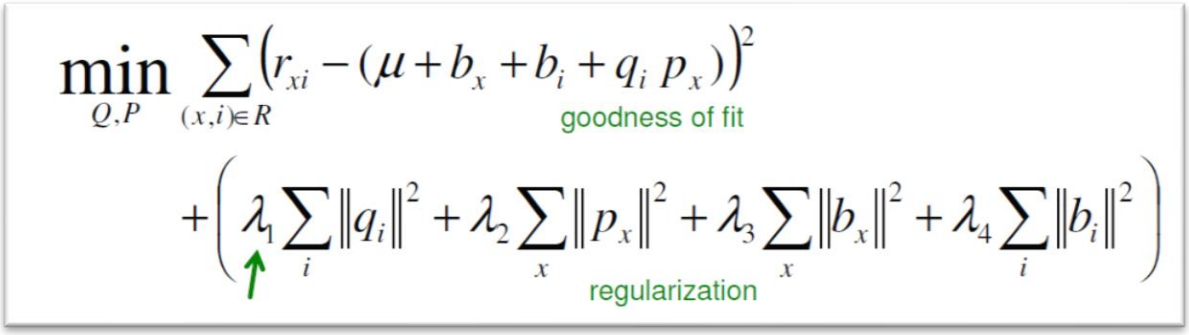

그림 8. 목적 함수의 구조

이 식을 b_x에 대해 미분하면 $-2 * \{r_{xi} - (\mu + b_x + b_i + q_i * p_x)\} + 2 * \lambda_3 * b_x$ 가 된다. 여기서 λ_3에 0.5를 대입하여 $2 * \lambda_3$을 1로 바꿔주면 $-2 * \{r_{xi} - (\mu + b_x + b_i + q_i * p_x)\} + b_x$ 가 되며, 여기에 학습률 c와 -1을 곱해주면 $c * [2 * \{r_{xi} - (\mu + b_x + b_i + q_i * p_x)\} - b_x]$ 가 된다. $2 * \{r_{xi} - (\mu + b_x + b_i + q_i * p_x)\}$를 err로 치환해주면 Δb_x를 구할 수 있다. 동일한 방식으로 Δb_i도 구할 수 있으며, 이렇게 식 5를 유도하게 되었다.

bias 조정까지 모두 끝나면 get_rmse(matrix) 메소드를 통하여 training set과 test set의 RMSE 값을 구한다. 각각의 RMSE값은 매 iteration마다 콘솔로 출력되며, 이중 test set의 RMSE 값의 iteration별 차이가 0.001 이하일 경우 학습이 끝나게 된다. 차이가 0.001 이상일 경우에는 행렬 Q, PT 및 bias의 조정 과정을 다시 수행한다. 학습이 끝나면 최종 결과 행렬을 파일로 저장하고, 최종 결과 행렬과 RMSE 값, 수행시간을 다음과 같이 출력한다.

```
iter  35
rain set error :  0.6206375052706407
est set error :  0.8578231962818855
iter  36
rain set error :  0.6193046834343056
est set error :  0.8567568159602417
iter  37
rain set error :  0.6180253341501561
est set error :  0.8557446305053129
iter  38
rain set error :  0.6167966414214439
est set error :  0.8547828833890702
rain finished!
----- reult matrix -----
[4.96804261 5.24372294 4.4627555  ... 5.48757248 5.06805619 5.44987154]
 [3.35933777 4.77109289 3.82024724 ... 4.86361878 4.47173117 4.86824804]
 [4.11412672 5.22523978 4.40323618 ... 5.44405622 5.05529805 5.44349781]
 ...
 [4.17201362 5.3841075  4.38340057 ... 5.44085454 5.0485502  5.43803616]
 [3.68134771 4.92291475 3.93843417 ... 5.02360238 4.64053889 5.03051994]
 [3.76189402 4.32485337 3.25783083 ... 4.45129749 4.02672676 4.36839331]]
ime :  187.7708146572113
```

그림 9. 최종 결과값

10번의 학습을 수행했을 때 test set의 평균 RMSE 값과 수행 시간은 다음과 같다.

회차	RMSE	수행 시간
1	0.859	79.57초
2	0.855	72.04초
3	0.867	79.55초
4	0.874	90.92초
5	0.859	90.27초
6	0.866	77.75초
7	0.856	89.04초
8	0.877	73.98초
9	0.877	86.30초
10	0.867	89.03초
평균	0.866	75.31초

2-2. 평점 행렬 불러오기

이 항목은 기존에 저장해 놓았던 학습 결과 행렬을 불러온다. 만약 저장된 파일이 없을 경우, 콘솔로 오류 메시지를 띄운다.

2-3. 평점 행렬 출력하기

이 항목은 현재 시스템에 저장되어 있는 학습 결과 행렬을 콘솔에 출력해준다. 만일 현재 학습되어 있는 행렬이 저장되어 있지 않다면 현재 가지고 있는 평점 정보로만 구성한 행렬을 보여준다.

2-4. 에러 그래프 출력

Iteration에 따른 training set과 test set의 RMSE 변화를 그래프로 보여준다. 그래프를 그리는 것은 python의 matplotlib를 이용하였으며, 이 라이브러리와 관련된 문법들은 http://codetorial.net/ 사이트를 참조하였다. 이 항목은 평점 행렬 학습을 마쳤거나, 저장되어 있는 평점 행렬을 불러왔을 때만 사용 가능하다. 출력된 그래프는 다음과 같다.

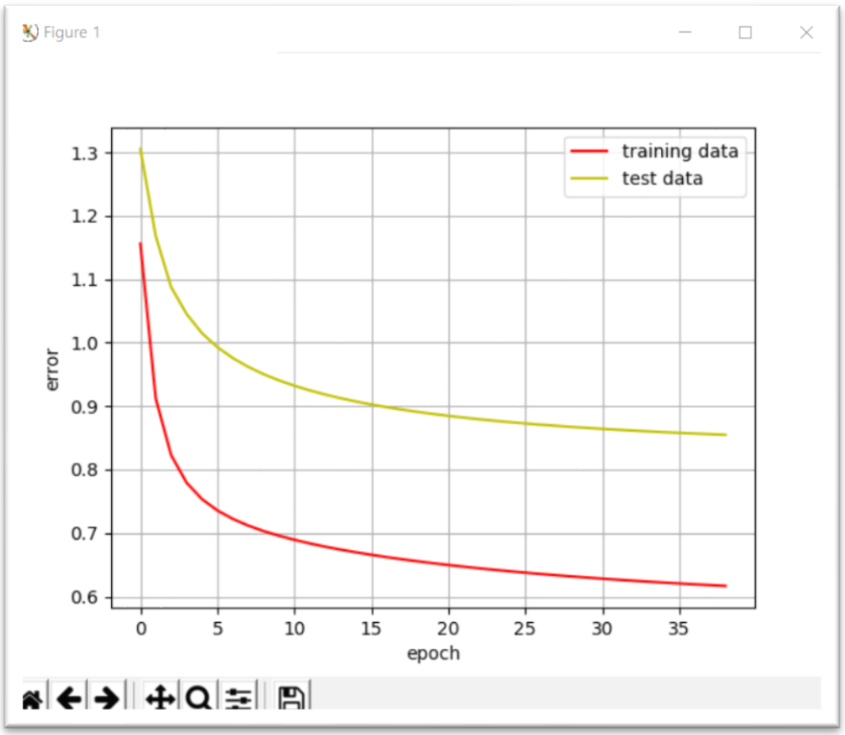

그림 10. Iteration에 따른 error값

다음은 user mode의 기능에 대해 살펴보자. User mode는 최종 결과 행렬을 불러오거나, 직접 계산하여 구했을 때 접근할 수 있다.

2-5. 추천 항목 출력

이 기능은 사용자의 ID와 추천받을 항목의 개수를 입력받고 그에 맞게 사용자가 평점을 매기지 않은 항목들을 예측 평점이 높은 순서대로 출력해준다. 수행 과정은 다음과 같다.

```
----- music recommendation system ver 0.1 -----
1. user mode
2. admin mode
3. quit
select : 2

 ----- admin mode -----
1. calculate rating matrix
2. load rating matrix
3. show rating matrix
4. show error graph
5. quit
select : 2
Load completed.

 ----- admin mode -----
1. calculate rating matrix
2. load rating matrix
3. show rating matrix
4. show error graph
5. quit
select : 5

1. user mode
2. admin mode
3. quit
select : 1
```

그림 11. 추천 내역 출력 1

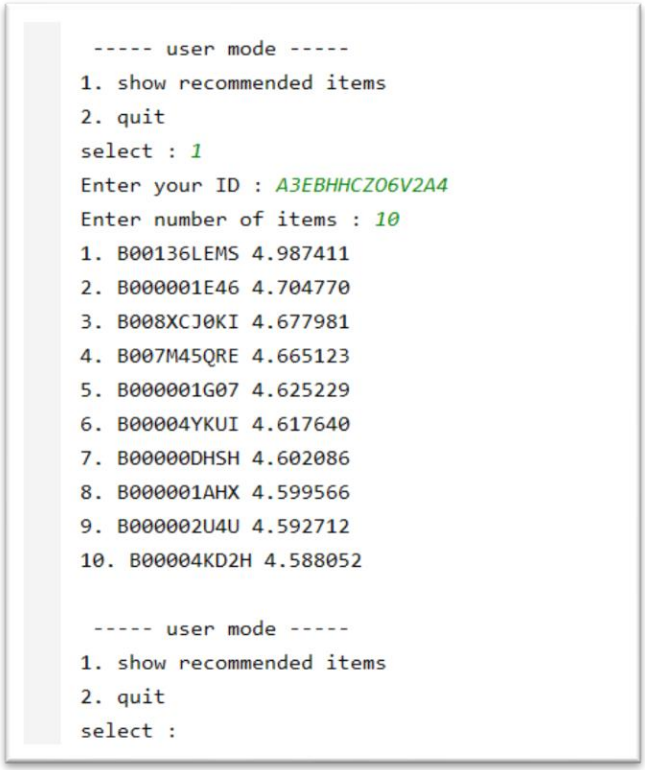

그림 12. 추천 내역 출력 2

User mode에서의 기능은 앞으로 더 구현할 예정이며, 추천목록에서 특정 항목 삭제하기, 새로운 사용자의 평점 내역 추가하기 등의 기능을 구현할 것이다.

3. 내부 구조

우리는 내부 연산들을 수행하는 파일인 'project_bias.py'와, 사용자와 프로그램 관리자를 위한 메뉴를 띄워주는 파일인 'recommendation_menu.py'를 만들었다.

project_bias.py에는 Make_Graph 클래스, Make_Rating_Matrix 클래스, Recommend_Engine_SGD 클래스가 존재한다. Make_Graph 클래스는 에러 그래프를 생성하고 이를 보여주는 기능을 담당한다. Make_Rating_matrix 클래스는 평점 데이터가 담겨있는. json 파일에서 데이터를 뽑아내 item-user rating matrix를 만드는 기능을 담당한다.

Recommend_Engine_SGD 클래스는 SGD를 담당하며, 여기에서 Make_Rating_Matrix 클래스와 Make_Graph 클래스의 인스턴스를 생성하여 그 기능을 사용한다.

Recommendation_menu.py에는 Rec_Menu 클래스가 존재한다. 이 클래스는 콘솔창에 메뉴를 띄워주는 역할을 한다

4. 문제점

현재 우리가 구현한 추천 시스템에는 두 가지의 문제가 존재한다. 첫째, 사용자에게 추천한 아이템을 추천한 이유를 설명할 수가 없다. 이 문제를 해결하기 위해 우리가 구현한 시스템에 contented based recommendation이나, word2Vec와 같은 방법을 결합해서 해결 방법을 찾아볼 것이다. 두 번째 문제는 실제로는 낮은 평점을 받은 항목이 높은 평점으로 예측되는 경우가 있다. 이러한 문제를 확인하기 위해서 우리가 예측한 행렬의 평점과 기존에 알려져 있던 평점을 비교해 보았다.

```
pridict :  4.89846036936569 target :  5.0
pridict :  4.608813404980366 target :  5.0
pridict :  4.811080017257286 target :  5.0
pridict :  4.8661184568803035 target :  5.0
pridict :  4.922806278830821 target :  5.0
pridict :  2.3039123900530085 target :  2.0
pridict :  2.1058253476999416 target :  2.0
pridict :  4.899758189252792 target :  5.0
pridict :  2.1991477626820477 target :  2.0
pridict :  4.803391394073858 target :  5.0
pridict :  4.730499852767019 target :  5.0
pridict :  2.0900037186820377 target :  2.0
pridict :  3.0339487951891027 target :  3.0
pridict :  3.2513728045975747 target :  3.0
pridict :  3.0821462708109055 target :  3.0
pridict :  2.9802785799039833 target :  3.0
pridict :  2.3498131757566476 target :  2.0
pridict :  3.0806305871804107 target :  3.0
pridict :  4.685223427378782 target :  5.0
pridict :  2.367593764590721 target :  2.0
pridict :  2.904855210049239 target :  3.0
pridict :  2.2340468725699565 target :  2.0
pridict :  3.3684556511690236 target :  3.0
pridict :  2.2737281556692945 target :  2.0
pridict :  3.0796323266521286 target :  3.0
pridict :  3.2608627181205914 target :  3.0
pridict :  3.973573510185752 target :  1.0
```

그림 13. 예측 평점과 실제 평점과의 차이

대부분의 항목들이 예측 평점과 실제 평점의 차이가 얼마 나지 않지만, 마지막 항목을 보면 실제로는 평점 1점짜리 항목이 3.9점으로 예측되어 있다. RMSE는 전체 행목들에 대해 예측 평점과 실제 평점의 차이를 구하기 때문에 이렇게 한 항목의 차이가 큰 경우를 잘 잡아내지 못하는 것 같다. 이러한 문제를 해결하기 위하여 다른 추천 시스템과의 결합이나, 다른 해결책을 찾아봐야 할 것 같다.

5. 정리

우리는 이렇게 SGD를 기반으로 한 recommendation system을 구현하였다. 사용자에게 추천 이유를 설명해주지 못하는 문제나 낮은 평점을 높은 평점으로 인식하는 문제들이 아직 남아있고, 사용자를 위한 기능이 아직 부족하다는 단점이 있지만 앞으로 남은 시간동안 더 노력해서 우리의 시스템의 성능이 더 좋아지도록 보완할 것이다.

6. 전체 코드

6-1. recommendation_menu.py

```python
import project_bias as pb

class Rec_Menu:
    def __init__(self):
        self.rec = pb.Recommend_Engine_SGD()
        self.loaded=0
        self.calculated=0
        print('-' * 5, 'music recommendation system ver 0.1', '-' * 5)
        while True:
            print('1. user mode')
            print('2. admin mode')
            print('3. quit')
            select = input('select : ')

            if select == '1':
                self.user_mode()
            elif select == '2':
                self.admin_mode()
            elif select == '3':
                print()
                return
            else:
                print("Please enter a vaild number. (1-3)\n")
```

```
24
25      def user_mode(self):
26          if self.loaded==0 and self.calculated==0:
27              print("please load or calculate rating matrix from admin mode.\n")
28              return
29
30          while True:
31              print("\n", '-' * 5, 'user mode', '-' * 5)
32              print('1. show recommended items')
33              print('2. quit')
34              select = input('select : ')
35
36              if select == '1':
37                  self.rec.recommend_item()
38              elif select == '2':
39                  print()
40                  return
41              else:
42                  print("Please enter a vaild number. (1-2)\n")
43
44      def admin_mode(self):
45          while True:
46              print("\n", '-' * 5, 'admin mode', '-' * 5)
47              print('1. calculate rating matrix')
48              print('2. load rating matrix')
49              print('3. show rating matrix')
50              print('4. show error graph')
51              print('5. quit')
52              select = input('select : ')
```

```
53
54              if select == '1':
55                  self.rec.train(500)
56                  self.calculated=1
57
58              elif select == '2':
59                  check=self.rec.load_rating_matrix()
60                  if check:
61                      print("Load completed.")
62                      self.loaded=1
63                  else:
64                      print("Load failed. File not exist")
65
66              elif select == '3':
67                  self.rec.show_rating_matrix()
68
69              elif select == '4':
70                  self.rec.show_graph()
71
72              elif select=='5':
73                  print()
74                  return
75              else:
76                  print("Please enter a vaild number. (1-5)\n")
77
78  m=Rec_Menu()
```

6-2. project_bias.py

```
1   import sys
2   import json
3   import math
4   import numpy as np
5   import matplotlib.pyplot as plt
6   from scipy.sparse.linalg import svds
7   import time
8
9   class Make_Rating_Matrix:
10      def __init__(self):
11          try:
12              self.data = open("Digital_Music_5.json", 'r')
13          except IOError:
14              print("file open failed!")
15              sys.exit()
```

```
16
17      def get_rating_matrix(self):
18          items={}
19          users={}
20          i=0
21          j=0
22          for line in self.data:
23              k = json.loads(line)
24              if k['asin'] not in items:
25                  items[k['asin']] = i
26                  i += 1
27              if k['reviewerID'] not in users:
28                  users[k['reviewerID']] = j
29                  j += 1
30
31          items_size = len(items)
32          users_size = len(users)
33          rating = np.zeros((items_size, users_size))
```

```
34
35          self.data.seek(0)
36          for line in self.data:
37              k = json.loads(line)
38              rating[items[k['asin']], users[k['reviewerID']]] = k['overall']
39          return rating, items, users
40
41  class Make_Graph:
42      def __init__(self, train_set_error, test_set_error):
43          self.train_set_error=train_set_error
44          self.test_set_error=test_set_error
45
46          x = np.arange(len(self.train_set_error))
47          line1=plt.plot(self.train_set_error, 'r')
48          line2=plt.plot(self.test_set_error, 'y')
49          plt.grid(axis='both')
50          plt.xlabel('epoch')
51          plt.ylabel('error')
52          plt.xticks(np.arange(0, len(x), step=5))
53          print(len(x)//5)
54          plt.legend(labels=('training data', 'test data'))
```

```
55
56      def show_graph(self):
57          plt.show()
58
59  class Recommend_Engine_SGD:
60      def __init__(self, lambda_q=1, lambda_pt=1, lr=0.0001):
61          '''
62          :param lambda_q : lambda 1, default value is 1
63          :param lambda_pt : lambda 2, default value is 1
64          :param lr : sgd's learning rate
65
66          matrix_row_len : row length of rating matrix, num of item
67          matrix_col_len : column length of rathing matrix, num of user
68          test_set : test set of rating matrix, choose 5,000 nonzero elements
69          train_set : training set, replace the selected elements in the test_set with zeros
70          total_mean : mean of original rating matrix's non-zero elements
71          item_bias : item bias
72          user_bias : user bias
73          '''
74          m=Make_Rating_Matrix()
75
76           self.matrix, self.items, self.users=m.get_rating_matrix()
77           self.matrix_row_len=len(self.matrix)
78           self.matrix_col_len=len(self.matrix[0])
79           self.rating_matrix=np.zeros((1, 1))
80           self.train_set, self.test_set = self.make_train_test()
81           #self.mat_q, self.mat_pt=cal_SVD(self.train_set)
82           self.lr=lr
83           self.total_mean=self.get_mean()
84           self.item_bias, self.user_bias=self.get_bias()
85           self.train_set_error=[]
86           self.test_set_error=[]
87           self.trained=0 # if rating matrix is trained, variable trained turns to 1.
88           self.loaded=0 # if admin load rating matrix, variable loaded turns to 1.
89
90           self.load_SVD()
91
92      # cal_SVD() : Perform SVD with original rating matrix and save the results.
93       def cal_SVD(self):
94          factor_k =100
95          U, s, V = svds(self.matrix, factor_k)
96          S = np.zeros((len(s), len(s)))
97          for i in range(len(s)):
98              S[i][i] = s[i]
99          PT = np.dot(S, V)
100         np.save('SVD_Q.npy', U)
101         np.save('SVD_PT.npy', PT)
```

```
102
103         # load_SVD() : load SVD
104         def load_SVD(self):
105             try:
106                 self.mat_q = np.load('SVD_Q.npy')
107                 self.mat_pt = np.load('SVD_PT.npy')
108             except IOError:
109                 print('Make SVD...')
110                 self.cal_SVD()
111                 self.load_SVD()
112
113         # get_mean() : Returns the average of nonzero elements in the original rating matrix
114         def get_mean(self):
115             nonzero_row, nonzero_col=self.get_nonzero(self.matrix)
116             res=0
117             for i in range(len(nonzero_row)):
118                 row=nonzero_row[i]
119                 col=nonzero_col[i]
120                 res+=self.matrix[row, col]
121             return res/len(nonzero_row)
122
123         # get_bias() : Returns item bias, user bias
124         def get_bias(self):
125             i_bias=np.zeros(self.matrix_row_len)
126             u_bias=np.zeros(self.matrix_col_len)
```

```
127
128             for idx in range(self.matrix_row_len):
129                 i_bias[idx]=np.mean(self.matrix[idx, self.matrix[idx, :].nonzero()])-self.total_mean
130
131             for idx in range(self.matrix_col_len):
132                 u_bias[idx] = np.mean(self.matrix[self.matrix[:, idx].nonzero(), idx])-self.total_mean
133             return i_bias, u_bias
134
135         '''
136             get_nonzero() : Returns two vector that stores indices of nonzero elements of given matrix.
137                             nonzero_row : store row index
138                             nonzero_col : store column index
139                             matrix[nonzero_row[i], nonzero_col[i]] = nonzero element of matrix
140         '''
141         def get_nonzero(self, matrix):
142             nonzero_row, nonzero_col = matrix.nonzero()
143             return nonzero_row, nonzero_col
144
145         #make_train_test() : make train set and test set from the original matrix.
146         def make_train_test(self):
147             train_set=self.matrix.copy()
148             test_set=np.zeros(self.matrix.shape)
149
150             nonzero_row, nonzero_col=self.get_nonzero(train_set)
151             idx_list=np.random.choice(range(len(nonzero_row)), 5000, replace=False) # num of test set : 5000
152
153             for idx in idx_list:
154                 test_set[nonzero_row[idx], nonzero_col[idx]]=train_set[nonzero_row[idx], nonzero_col[idx]]
155                 train_set[nonzero_row[idx], nonzero_col[idx]]=0
```

```
156
157         return train_set, test_set
158
159     #train() : Train the training set with SGD.
160     def train(self, iter):
161         self.train_set_error=[] #clear train set error.
162         self.test_set_error=[] #clear test set error.
163         if self.trained==1: #if rating matrix is already trained, check
164             print('rating matrix is already trained. do you want to train again?(y, n)')
165             k=input()
166             if k=='n':
167                 return
168
169         nonzero_row, nonzero_col=self.get_nonzero(self.train_set) #Returns the row number and column number of non-zero elements.
170         start=time.time()
171         prev_rmse=0.
172         for i in range(iter):
173             print("iter ", i)
174             for idx in range(len(nonzero_row)):
175                 row=nonzero_row[idx] # Row number of nonzero element in train set.
176                 col=nonzero_col[idx] # Column number of nonzero element in train set.
177                 baseline=self.get_baseline(row, col) #set baseline estimate
178                 err=2*(self.train_set[row, col] - (baseline + np.dot(self.mat_q[row, :], self.mat_pt[:, col])))
179
180                 if err > 100000 :
181                     print("err exceeded over 100000. change learning rate and try again.")
182                     return
183
184                 self.mat_q[row, :] += self.lr * (err * self.mat_pt[:, col] - self.mat_q[row, :]) #update q
185                 self.mat_pt[:, col] += self.lr * (err * self.mat_q[row, :] - self.mat_pt[:, col]) #update pt
186
187                 self.item_bias[row] += self.lr * (err - self.item_bias[row]) #update item bias
188                 self.user_bias[col] += self.lr * (err - self.user_bias[col]) #update user bias
189
190             train_rmse=self.get_rmse(self.train_set)
191             test_rmse=self.get_rmse(self.test_set)
192
193             print("train set error : ", train_rmse)
194             print("test set error : ", test_rmse)
195             self.train_set_error.append(train_rmse)
196             self.test_set_error.append(test_rmse)
197
198             if abs(prev_rmse - test_rmse) <= 1e-3: # End training when the RMSE difference in test set is less than 0.001.
199                 print("train finished!")
200                 print('-'*5, 'reult matrix','-'*5)
201                 self.rating_matrix=self.get_final_matrix()
202                 print(self.rating_matrix)
203                 np.save('rating_matrix.npy', self.rating_matrix)
204                 np.save('train_set_error.npy', self.train_set_error)
205                 np.save('test_set_error.npy', self.test_set_error)
206                 print("time : ", time.time() - start)
207                 self.trained=1
208                 return
209             prev_rmse=test_rmse
210
211     # get_baseline() : Return baseline estimate for given row number and column number.
212     def get_baseline(self, row, col):
213         return self.total_mean + self.item_bias[row] + self.user_bias[col]
214
215     # get_final_matrix() : Return learning matrix.
216     def get_final_matrix(self):
```

```
        tmp=np.full(self.matrix.shape, self.total_mean)
        for i in range(self.matrix_col_len):
            tmp[:, i]+=self.item_bias

        for i in range(self.matrix_row_len):
            tmp[i, :]+=self.user_bias

        tmp+=np.dot(self.mat_q, self.mat_pt)
        return tmp

    #load_rating_matrix() : load rating matrix, train set error, test set error and store them in each variables.
    def load_rating_matrix(self):
        self.rating_matrix=np.load('rating_matrix.npy')
        self.train_set_error=list(np.load('train_set_error.npy'))
        self.test_set_error=list(np.load('test_set_error.npy'))

        if len(self.rating_matrix) > 1:
            self.loaded=1
            return True
        else: return False
```

```
    #get_rmse() : return RMSE for given matrix.
    def get_rmse(self, matrix):
        sum=0
        nonzero_row, nonzero_col = self.get_nonzero(matrix)
        pred_matrix=self.get_final_matrix()
        for idx in range(len(nonzero_row)):
            row=nonzero_row[idx]
            col=nonzero_col[idx]
            target=matrix[row, col]
            sum+=(target-pred_matrix[row, col])**2
        sum=np.sqrt(sum/len(nonzero_row))
        return sum

    #show_rating_matrix() : print rating matrix.
    def show_rating_matrix(self):
        if self.loaded==0 and self.trained==0:
            print(self.matrix)
            return

        print(self.rating_matrix)
        print("training set error rate : ", self.train_set_error[-1])
        print("test set error rate : ", self.test_set_error[-1])
```

```
260         print("test set error rate : ", self.test_set_error[-1])
261
262     #show_graph() : show error graph.
263     def show_graph(self):
264         if self.trained==0 and self.loaded==0:
265             print("Please train or load the rating matrix.")
266             return
267         graph=Make_Graph(self.train_set_error ,self.test_set_error)
268         graph.show_graph()
269
270     #recommend_item() : recommend k items for user.
271     def recommend_item(self):
272         userID=input("Enter your ID : ")
273         num=int(input("Enter number of items : "))
274         user_idx=self.users[userID]
275         nonzero_row, nonzero_col=self.get_nonzero(self.matrix)
276         testk=[]
277         for i in range(len(nonzero_col)):
278             if nonzero_col[i]==user_idx:
279                 row = nonzero_row[i]
280                 testk.append(row)
281
282         test_list=self.rating_matrix[:, user_idx]
283         rec_list=[]
284         itemIDs = list(self.items.keys())
285         for i in range(len(test_list)):
286             rec_list.append([itemIDs[i], test_list[i]])
287         for i in testk:
288             rec_list[i][1]=0
289         rec_list.sort(key=lambda x : x[1], reverse=True)
290         for i in range(num):
291             print('%d. %s %lf'%(i+1, rec_list[i][0], rec_list[i][1]))
```

7. 사진 출처

식 1. (Pearson correlation coefficient) : https://thebook.io/006723/ch07/06/02/

그림 8. Recommendation 강의자료 두 번째 PDF 44페이지

이외의 식이나 그림들은 직접 제작했습니다.