Course Title (과목명)	인공지능통신
HW Number (HW 번호)	Project 3
Submit Date (제출일)	2021-06-02
Grade (학년)	4학년
ID (학번)	20151483, 20161482
Name (이름)	이창헌, 박준용

Project 3

1. 프로젝트 목표

- 넷플릭스 영화 추천 시스템 구현
- 12주차 강의 자료의 내용을 이용하여 유저의 특정 영화에 대한 점수 예측

3. 프로젝트 수행 방법

(1) 점수 예측 알고리즘 구현 및 설명

제출한 파일중에서 **Project3_toy_example.ipynb**에서 12주차 강의 자료에서 다룬 10 users, 5 movies에 대해서 30 train data samples로 10 test data samples을 예측하는 예시에 대해서 구현한 알고리즘을 우선적으로 실험해보았다.

제출한 파일중에서 **Project3_100user.ipynb**는 실제 프로젝트 3에서 요구하는 100 users, 1997 movies에 대해서 5216 train data samples로 1304 test data samples를 예측하는 경우에도 같은 알고리즘을 그대로 적용했으나 이 경우에는 baseline predictor와 neighborhood predictor 그리고 그 predictor로 만든 prediction matrix의 크기가 너무 커서 보고서에서 표시하기 어렵기 때문에 알고리즘의 설명은 **Project3_toy_example**으로 한다.

구현한 알고리즘의 방법은 아래와 같다.

1. Test data를 csv파일에서 읽어와서 row가 user, column이 movie이고 rating 값을 담은 matrix_R (rating matrix)을 작성한다.

```
1 matrix_R

[[5, 4, 4, 0, 0],
 [0, 3, 5, 0, 4],
 [5, 2, 0, 0, 3],
 [0, 0, 3, 1, 2],
 [4, 0, 0, 4, 5],
 [0, 3, 0, 3, 5],
 [3, 0, 3, 2, 0],
 [5, 0, 4, 0, 5],
 [0, 2, 5, 4, 0],
 [0, 0, 5, 3, 4]]
```

$$\mathbf{R} = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \\ 10 \end{array} \begin{array}{c} \begin{array}{ccccc} A & B & C & D & E \end{array} \\ \begin{pmatrix} 5 & 4 & 4 & - & \mathbf{5} \\ - & 3 & 5 & \mathbf{3} & 4 \\ 5 & 2 & - & \mathbf{2} & 3 \\ - & \mathbf{2} & 3 & 1 & \mathbf{2} \\ 4 & - & \mathbf{5} & 4 & 5 \\ \mathbf{5} & 3 & - & 3 & 5 \\ 3 & \mathbf{2} & 3 & 2 & - \\ 5 & \mathbf{3} & 4 & - & 5 \\ \mathbf{4} & 2 & 5 & 4 & - \\ \mathbf{5} & - & 5 & 3 & 4 \end{pmatrix} \end{array}$$

test data

training data

2. $b = (A^T A)^{-1} A^T c$의 수식을 사용해서 user bias, movie bias를 계산하기 위해서 training data로부터 matrix_A를 작성하고 그 pseudo inverse를 구한다. Training data의 label과 average(이 경우에는 3.67)의 차이로부터 matrix_C를 작성한다. 결과적으로 계산된 user bias, movie bias를 matrix_B로 저장한다.

```
1 matrix_A

[[1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0],
 [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0],
 [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0],
 [0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0],
 [0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0],
 [0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1],
 [0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0],
 [0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0],
 [0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1],
 [0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0],
 [0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0],
 [0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1],
 [0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0],
 [0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0],
 [0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1],
 [0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0],
 [0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0],
 [0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0],
 [0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1],
 [0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0],
 [0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1],
 [0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0],
 [0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1],
 [0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0],
 [0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1]]
```

```
[8] 1 matrix_C

[[1.3333333333333335],
 [0.3333333333333335],
 [0.3333333333333335],
 [-0.6666666666666665],
 [1.3333333333333335],
 [0.3333333333333335],
 [1.3333333333333335],
 [-1.6666666666666665],
 [-0.6666666666666665],
 [-0.6666666666666665],
 [-2.6666666666666665],
 [-1.6666666666666665],
 [0.3333333333333335],
 [0.3333333333333335],
 [1.3333333333333335],
 [-0.6666666666666665],
 [-0.6666666666666665],
 [1.3333333333333335],
 [-0.6666666666666665],
 [-0.6666666666666665],
 [-1.6666666666666665],
 [1.3333333333333335],
 [0.3333333333333335],
 [1.3333333333333335],
 [-1.6666666666666665],
 [1.3333333333333335],
 [0.3333333333333335],
 [1.3333333333333335],
 [-0.6666666666666665],
 [0.3333333333333335]]
```

```
[9] 1 matrix_B

array([[ 0.62592593],
       [ 0.42348485],
       [-0.28148148],
       [-1.77474747],
       [ 0.5202862 ],
       [ 0.49250842],
       [-1.23897306],
       [ 0.45126263],
       [ 0.39991582],
       [ 0.22525253],
       [ 0.71792929],
       [-1.19873737],
       [ 0.6030303 ],
       [-0.6040404 ],
       [ 0.32525253]])
```

$$\mathbf{b}_u^* = [0.62, 0.42, -0.28, -1.78, 0.52, 0.49, -1.24, 0.45, 0.40, 0.23]^T$$

User

$$\mathbf{b}_i^* = [0.72, -1.20, 0.60, -0.60, 0.33]^T$$

Movie

3. 이제 bias값들을 알고 있기 때문에 baseline predictor를 사용해서 rating matrix를 예측할 수 있다. 예측된 값들을 matrix_R_hat으로 저장한다.

```
[11]  1 matrix_R_hat

[[5.01, 3.09, 4.9, 3.69, 4.62],
 [4.81, 2.89, 4.69, 3.49, 4.42],
 [4.1, 2.19, 3.99, 2.78, 3.71],
 [2.61, 0.69, 2.49, 1.29, 2.22],
 [4.9, 2.99, 4.79, 3.58, 4.51],
 [4.88, 2.96, 4.76, 3.56, 4.48],
 [3.15, 1.23, 3.03, 1.82, 2.75],
 [4.84, 2.92, 4.72, 3.51, 4.44],
 [4.78, 2.87, 4.67, 3.46, 4.39],
 [4.61, 2.69, 4.49, 3.29, 4.22]]
```

$$\hat{\mathbf{R}} = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \\ 10 \end{array} \begin{array}{c} \begin{array}{ccccc} A & B & C & D & E \end{array} \\ \begin{pmatrix} 5.00 & 3.09 & 4.90 & - & \mathbf{4.62} \\ - & 2.89 & 4.69 & \mathbf{3.49} & 4.42 \\ 4.10 & 2.19 & - & \mathbf{2.78} & 3.71 \\ - & \mathbf{1.00} & 2.49 & 1.29 & 2.22 \\ 4.90 & - & \mathbf{4.79} & 3.58 & 4.51 \\ \mathbf{4.88} & 2.96 & - & 3.56 & 4.48 \\ 3.15 & \mathbf{1.23} & 3.03 & 1.82 & - \\ 4.84 & \mathbf{2.92} & 4.72 & - & 4.44 \\ \mathbf{4.84} & 2.92 & 4.72 & 3.51 & - \\ \mathbf{4.61} & - & 4.49 & 3.29 & 4.22 \end{pmatrix} \end{array}$$

4. Baseline predictor로 예측한 rating과 실제 rating의 차이를 matrix_P로 저장한다.

```
[13]  1 matrix_P

[[-0.01, 0.91, -0.9, 0, 0],
 [0, 0.11, 0.31, 0, -0.42],
 [0.9, -0.19, 0, 0, -0.71],
 [0, 0, 0.51, -0.29, -0.22],
 [-0.9, 0, 0, 0.42, 0.49],
 [0, 0.04, 0, -0.56, 0.52],
 [-0.15, 0, -0.03, 0.18, 0],
 [0.16, 0, -0.72, 0, 0.56],
 [0, -0.87, 0.33, 0.54, 0],
 [0, 0, 0.51, -0.29, -0.22]]
```

$$\tilde{\mathbf{R}} = \mathbf{R} - \hat{\mathbf{R}} = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \\ 10 \end{array} \begin{array}{c} \begin{array}{ccccc} A & B & C & D & E \end{array} \\ \begin{pmatrix} 0 & 0.91 & -0.90 & - & ? \\ - & 0.11 & 0.31 & ? & -0.42 \\ 0.90 & -0.19 & - & ? & -0.71 \\ - & ? & 0.51 & -0.29 & -0.22 \\ -0.90 & - & ? & 0.42 & 0.49 \\ ? & 0.040 & - & -0.56 & 0.52 \\ -0.15 & ? & -0.031 & 0.18 & - \\ 0.16 & ? & -0.72 & - & 0.56 \\ ? & -0.87 & 0.33 & 0.54 & - \\ ? & - & 0.51 & -0.29 & -0.22 \end{pmatrix} \end{array}$$

5. Cosine coefficient를 사용해서 movies간의 similarity를 계산해서 matrix_S로 저장한다.

```
[15]  1 matrix_S

[[0, -0.22, -0.4, -0.97, -0.75],
 [-0.22, 0, -0.84, -0.73, 0.5],
 [-0.4, -0.84, 0, -0.22, -0.93],
 [-0.97, -0.73, -0.22, 0, 0.07],
 [-0.75, 0.5, -0.93, 0.07, 0]]
```

$$\mathbf{D} = \begin{array}{c} \\ A \\ B \\ C \\ D \\ E \end{array} \begin{array}{c} \begin{array}{ccccc} A & B & C & D & E \end{array} \\ \begin{pmatrix} - & -0.21 & -0.41 & -0.97 & -0.75 \\ -0.21 & - & -0.84 & -0.73 & 0.51 \\ -0.41 & -0.84 & - & -0.22 & -0.93 \\ -0.97 & -0.73 & -0.22 & - & 0.068 \\ -0.75 & 0.51 & -0.93 & 0.068 & - \end{pmatrix} \end{array}$$

6. 5의 과정에서 구한 similarity matrix와 4의 과정에서 구한 baseline predictor로 예측한 rating과 실제 rating의 차이 matrix_P를 이용하는 neighborhood predictor로 rating을 새롭게 예측한다. Neighborhood의 기준은 영화 사이의 similarity 값이 0.6이상으로 유사도가 높은 영화들만 neighboring movie로 판단하도록 하였다.

$$New\ prediction = Baseline\ predition + \sum \frac{similarity \times (neighboring\ movie's\ baseline\ prediction - training\ label\ rating}{|similarity|}$$

Neighborhood prediction은 위의 수식에 따라서 하나의 영화 A에 대해서 그 영화와 유사도가 높은 영화들과 baseline predictor로 예측한 유사도가 높은 영화들의 예측을 실제 training data label의 차이를 사용해서 기존의 baseline predictor로 예측한 영화 A의 rating을 보정한다. Neighborhood predictor로 보정해서 새롭게 예측한 rating은 matrix_R_neigh으로 저장한다.

```
[17]  1 matrix_R_neigh

[[5.01, 3.99, 3.99, 3.3, 5.12],
 [5.23, 2.58, 4.86, 3.38, 4.11],
 [4.81, 2.19, 4.45, 2.35, 2.81],
 [2.87, 0.55, 2.71, 1.29, 1.71],
 [4.45, 2.57, 4.3, 4.48, 5.41],
 [4.97, 3.52, 4.47, 3.52, 4.48],
 [2.97, 1.16, 3.03, 1.97, 2.83],
 [4.28, 3.64, 4.16, 3.35, 4.77],
 [4.24, 2.44, 5.54, 4.33, 4.06],
 [4.87, 2.55, 4.71, 3.29, 3.71]]
```

$$\hat{\mathbf{R}}^N = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \\ 10 \end{array} \begin{array}{c} \begin{array}{ccccc} A & B & C & D & E \end{array} \\ \begin{pmatrix} 5.00 & 3.99 & 3.99 & - & \mathbf{5.00} \\ - & 2.58 & 4.86 & \mathbf{3.38} & 4.11 \\ 4.81 & 2.19 & - & \mathbf{2.35} & 2.81 \\ - & \mathbf{1.00} & 2.71 & 1.29 & 1.71 \\ 4.46 & - & \mathbf{4.30} & 4.49 & 5.42 \\ \mathbf{4.97} & 3.52 & - & 3.52 & 4.48 \\ 2.97 & \mathbf{1.16} & 3.03 & 1.97 & - \\ 4.28 & \mathbf{3.64} & 4.16 & - & 4.77 \\ \mathbf{4.25} & 2.44 & 5.54 & 4.33 & - \\ \mathbf{4.87} & - & 4.71 & 3.29 & 3.71 \end{pmatrix} \end{array}$$

이렇게 구현된 알고리즘을 사용해서 실험해본 결과 baseline predictor를 사용해서 rating을 예측한 결과 30개의 train data sample에 대해서 56.67%의 train accuracy를 10개의 test data에 대해서는 60%의 test accuracy를 보여주었다.

Neighborhood predictor를 사용해서 rating을 예측한 결과는 train data sample에 대해서 83.33%의 train accuracy를 test data에 대해서는 60%의 test accuracy를 보여주었다.

30개의 train data와 10개의 test data로 이루어진 작은 example에 대해서는 데이터의 개수가 너무 적어서 test accuracy는 같았지만 train accuracy의 경우 neighborhood predictor를 사용한 예측이 baseline predictor를 사용한 예측에 비해서 약 47%의 performance 향상을 보여주어서 neighborhood predictor를 사용한 예측이 훨씬 정확하다는 것을 확인할 수 있었다.

(2) Project 3 dataset에 대해서 성능 확인 및 결과 설명

- **학습 결과 확인**

제출한 파일중에서 **Project3_100user.ipynb**는 실제 프로젝트 3에서 요구하는 100 users, 1997 movies에 대해서 5216 train data samples을 사용해서 baseline predictor를 train하고 similarity matrix를 계산해서 neighborhood prediction을 구했다.

이제 이 baseline predictor와 neighborhood predictor를 사용해서 1304 test data에 대해서 rating을 예측해보고 실제 label과 비교해서 test accuracy를 구할 수 있었다.

	Train accuracy	**Test accuracy**
Baseline predictor	43.35%	36.20%
Neighborhood predictor	69.13%	54.37%

- **학습 결과 비교 및 분석**

구현한 알고리즘을 사용해서 실험해본 결과 baseline predictor를 사용해서 rating을 예측한 결과 5216개의 train data sample에 대해서 43.35%의 train accuracy를 1304개의 test data에 대해서는 36.20%의 test accuracy를 보여주었다.

Neighborhood predictor를 사용해서 rating을 예측한 결과는 train data sample에 대해서 69.13%의 train accuracy를 test data에 대해서는 54.37%의 test accuracy를 보여주었다.

Baseline predictor에서 user bias와 movie bias는 train data에 대해서 optimize한 결과이기 때문에 train accuracy가 test accuracy에 비해서 더 높은 결과를 내는 것은 당연하다.

또한 movies 사이에 연관성을 나타내는 similarity가 0.6 이상일 경우 neighboring movie로 판단해서 rating을 예측하는데 참고하는 neighborhood model은 단순하게 bias값만 사용하는 baseline model에 비해서 예측이 더 정확하게 보정되는 과정이 한 단계 추가되기 때문에 약 50.19%정도로 성능이 크게 향상되었다.

	Tolerance = 1-Similarity	Test accuracy
Neighborhood predictor	0.4	54.37%
	0.7	55.14%
	1	55.21%

하나의 Movie에 대해서 다른 movie가 neighboring movie로 정의되는데 필요한 최소한의 similarity의 수치를 1에서 뺀 값을 tolerance라고 두고 tolerance를 변경하면서 test accuracy에 어떤 영향이 있는지 살펴보았다.

Tolerance가 0.4으로 similarity가 0.6 이상인 영화들로 neighborhood로 정의하는 모델에 비해서 tolerance가 0.7으로 similarity가 0.3 이상인 영화들 즉 더 많은 neighboring 영화들을 사용해서 정확도를 보정하는 경우에 더 정확한 예측이 이루어져서 test accuracy가 향상되었다.

최종적으로 tolerance를 1으로 설정한 neighborhood model이 가장 높은 performance를 가진다는 것을 확인할 수 있었다. 이 경우에는 서로 다른 영화 사이에 조금이라도 연관성이 있으면 모두 정확도를 보정하는데 사용하기 때문에 다른 모델에 비해서 computational cost가 증가하는 대신 최고의 performance를 보여준다.