[IT Essential] Matrix Factorization #2

- 진행: 최인국 컨설턴트
- 날짜: 2020.09.15

- 목차

1. 개념 설명

1. Matrix Factorization

1. 정의

- 행렬을 곱으로 계산 할 수 있는 다른 행렬들로 분해하여 예측하는 알고리즘

2. Data 로 보는 활용 예시

- 유저 별 여행지 선호도
- 여행지 -- 선호도 경향이 눈에 띄는 유저
    - 괌, 하와이 : 휴양지로 유명한 여행지 -- 마이콜
    - 파리, 로마, 베니스 : 유럽 문화, 미술, 건축물을 볼 수 있는 여행지 -- 사오정
    - 길동의 경우
        - 파리, 베니스에 대해서는 높은 점수를 주었음
        - 괌, 하와이에 대해서는 낮은 점수를 주었음
        - 로마에 대해서도 높은 점수를 줄 것이라 예측할 수 있음
    - 둘리의 경우
        - 파리, 로마, 베니스에는 낮은 점수를 주었음
        - 괌에는 높은 점수를 주었음

- 괌과 비슷한 여행지인 하와이에도 높은 점수를 줄 것이라 예측할 수 있음

	괌	파리	로마	하와이	베니스
마이콜	2	8	9	1	8
사오정	8	2	1	8	1
길동	1	5	?	1	7
팔계	7	2	1	8	1
오공	1	8	9	2	9
둘리	9	1	2	?	2
삼장법사	6	1	2	7	2

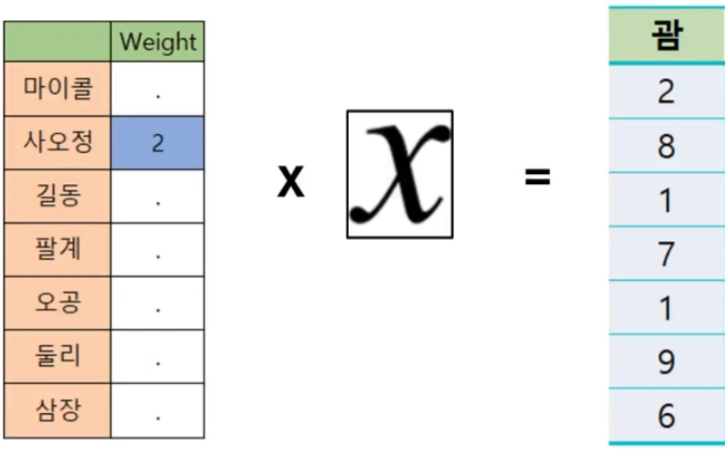
- 데이터의 크기가 커지면 사람이 한 눈에 데이터를 파악하고, 빈 곳('?'부분)의 값을 예측하기 어렵다.
- 이 때 사용하는 알고리즘이 `Matrix Factorization`

3. Matrix Factorization

1. 각 테이블 간의 관계

1. 유저 별 가중치 테이블 - 임의의 데이터 x - 특정 여행지 평가 테이블

- 유저 별 가중치에 대한 테이블과 임의의 데이터 x 를 곱하여, 괌에 대한 평가를 계산할 수 있다.
- 반대로, 괌에 대한 rating table 이 있으면 임의의 데이터 x 를 통해 유저의 가중치 테이블을 알 수 있다.
- 마찬가지로, 하와이에 대한 rating table 를 분해하면 임의의 데이터 x 와 하와이에 대한 선호도 테이블을 얻을 수 있다.



2. 특정 유저의 가중치 테이블 - 임의의 데이터 x - 여행지 별 평가 테이블

- 둘리라는 유저의 관광지 별 선호 데이터가 있다면 이 테이블은 다시 임의의 값 x 와 각 관광지 별 가중치 테이블로 분해 될 수 있다.

	괌	파리	로마	하와이	베니스
Weight	.	1	.	.	.

X x

=

둘리	8	2	1	8	1

2. Matrix Factorization

1. Matirix Factorization 의 목표

- 유저 별 여행지에 대한 평가 테이블(맨 좌측)은 두 개의 테이블로 분해가 된다.
    1. 유저 별 가중치 테이블
    2. 여행지 별 가중치 테이블
    - 역으로, 위의 두 테이블을 구하면 평가 테이블의 빈 곳의 값을 예측할 수 있다.
    - 즉, Matrix Factorization 을 활용하여
        1. 유저의 테이블과 여행지별 테이블을 구하고,
        2. 테이블을 바탕으로 특정 유저가 평가하지 않은 특정 관광지를 어떻게 평가할지에 대해 구한다!

	Weight
마이콜	.
사오정	.
길동	.
팔계	.
오공	.
둘리	.
삼장	.

X

	괌	파리	로마	하와이	베니스
Weight	.	.	.	.	.

≒

	괌	파리	로마	하와이	베니스
마이콜	2	8	9	1	8
사오정	8	2	1	8	1
길동	1	5	?	1	7
팔계	7	2	1	8	1
오공	1	8	9	2	9
둘리	9	1	2	?	2
삼장법사	6	1	2	7	2

2. 예측 결과의 정확도를 높이기 위한 방법

- 그런데, 유저 별 테이블이 1차원, 여행지 별 테이블이 1차원이라면 이 테이블들을 통해 만들어지는 평가 테이블 또한 차원이 작기 때문에 결과 값 r 이 단조로울 수 있다.
    - Weight 를 k 개 차원으로 늘리게 되면 표현 할 수 있는 값이 많아지고, 그 결과 평가 테이블 값도 더 정밀하게 예측할 수 있다.
    - User latent factors (P) 유저 별 Weight 를 K 개 차원으로 늘린 테이블
    - Item latent factors (Q) 여행지 별 Weight 를 K 개 차원으로 늘린 테이블
    - P 와 Q 를 곱해서 Rating 테이블 (R) 을 구하고 '?' 값을 구한다.

	K1	K2
마이콜	.	.
사오정	2	1
길동	.	.
팔계	.	.
오공	.	.
둘리	.	.
삼장	.	.

User latent factors = P

X

	괌	파리	로마	하와이	베니스
K1	.	.	.	2	.
K2	.	.	.	4	.

Item latent factors = Q

≒

	괌	파리	로마	하와이	베니스
마이콜	2	8	9	1	8
사오정	8	2	1	8	1
길동	1	5	?	1	7
팔계	7	2	1	8	1
오공	1	8	9	2	9
둘리	9	1	2	?	2
삼장법사	6	1	2	7	2

R

P x Q = R

3. Bias

- 특정 여행지별 값(맨 우측)이 있을 때, 유저 별 Weight 테이블 과 X 값만 있다면 예측이 부족할 수 있다.
    - Linear regression 에서 bias(y 축)가 없을 경우 (0, 0) 을 지나는 1차원 그래프만 만들 수 있었던 것을 떠올려 보자.
    - 마찬가지로 Matirix Factorization 에서도 유저 별 데이터를 통해 특정 여행지에 대한 평가를 예측하기 위해서 유저에 대한 bias 테이블이 필요하다.

	Weight
마이콜	.
사오정	2
길동	.
팔계	.
오공	.
둘리	.
삼장	.

X x +

	Bias
마이콜	.
사오정	1
길동	.
팔계	.
오공	.
둘리	.
삼장	.

=

괌
2
8
1
7
1
9
6

    - 여행지 별 데이터를 통해 예측 할 때도 여행지에 대한 bias 테이블이 필요하다.

	괌	파리	로마	하와이	베니스
Weight	.	1	.	.	.

x x

\+

	괌	파리	로마	하와이	베니스
Bias	.	.	.	.	.

=

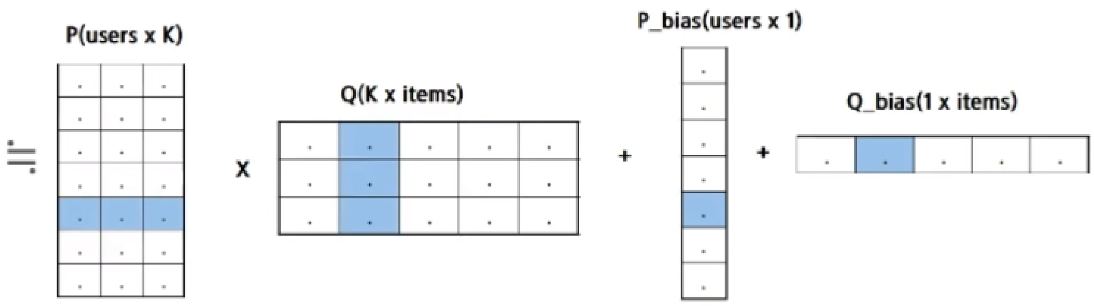
둘리	8	2	1	8	1

4. 최적의 R 을 예측하기 위해 필요한 4개의 Matrix

1. User latent factors (P)
2. Item latent factors (Q)
3. P 에 대한 Bias 테이블
4. Q 에 대한 Bias 테이블

R(users x items)

	괌	파리	로마	하와이	베니스
마이콜	2	8	9	1	8
사오정	8	2	1	8	1
길동	1	5	?	1	7
팔계	7	2	1	8	1
오공	1	8	9	2	9
둘리	9	1	2	?	2
삼장법사	6	1	2	7	2



2. 수식 유도

1. Cost 방정식

1. 임의의 4개의 테이블(Matrix) 값을 통해 예측값 r 구하기
   - p, q, b 값들은 위 4번의 그림에서 파란색으로 칠해진 칸들과 각각 매칭된다.

$$r_{ui}\ \ (predict) = p_u q_i + b_u + b_i$$

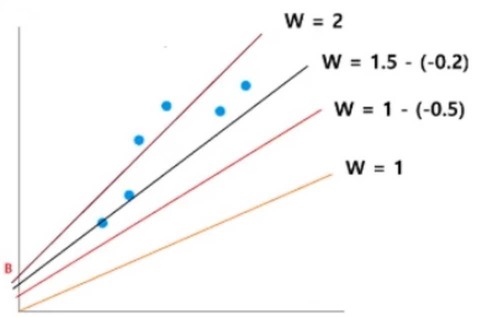
2. 실제 값과 예측한 값 의 차이를 Matrix Factorization 에서도 Error 라고 한다.
3. cost 는 k 개 차원에서 각각의 square error 값을 모두 더한 값

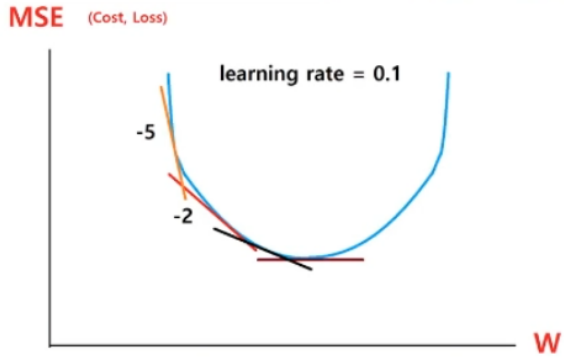
$$cost(p_u, q_i, b_u, b_i) = \sum_{u,i\epsilon \underbrace{k}_{\text{k개 차원}}} \underbrace{\boxed{(\underbrace{\underbrace{p_u q_i + b_u + b_i}_{predict} - r_{ui}}_{error})^2}}_{Square\ error}$$

++ Linear Regression ++

> 자세한 내용은 다시보기 & 강의 정리 자료를 참고해주세요!

- 최적의 Weight 를 구하기 위해 Cost 방정식을 편미분하여 변화율을 찾고, 경사하강법(Gradient descent)을 사용한다.
- Matrix Factorization 도 변수만 좀 달라질 뿐, 내용은 유사하다.





$$W_{(new)} = W_{(cur)} - \lambda \frac{1}{m} \sum_{i=1}^{m} (Wx_{(i)} + b - y_{(i)}) x_{(i)}$$

$$b_{(new)} = b_{(cur)} - \lambda \frac{1}{m} \sum_{i=1}^{m} Wx_{(i)} + b - y_{(i)}$$

$$y_{(predict)} = Wx + b$$

MSE

$$cost(W, b) = \frac{1}{m} \sum_{i=1}^{m} (Wx_{(i)} + b - y_{(i)})^2$$

2. 편미분하여 최적의 값 찾기

- 최적의 Weight 와 bias 찾기

$$cost(p_u, q_i, b_u, b_i) = \sum_{u,i \epsilon k} (p_u q_i + b_u + b_i - r_{ui})^2$$

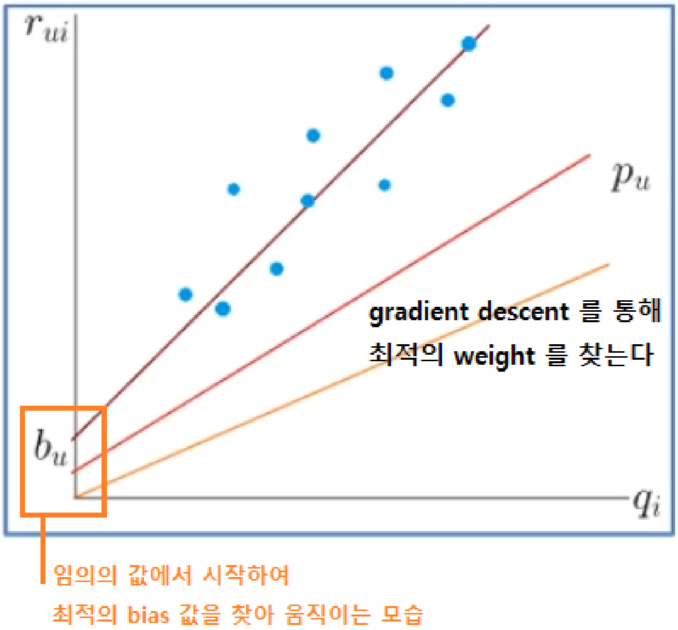
learning rate

최적의 p를 찾기 위해 편미분

$$p_{u(new)} = p_{u(cur)} - \boxed{\lambda} \sum_{u,i \epsilon k} q_i (p_u q_i + b_u + bi - r_{ui})$$

현재의 p

$$b_{u(new)} = b_{u(cur)} - \lambda \sum_{u,i \epsilon k} p_u q_i + b_i + b_u - r_{ui}$$



- 편미분 하는 방법

$$cost(p_u, q_i, b_u, b_i) = \sum_{u,i\epsilon k}(p_uq_i + b_u + b_i - r_{ui})^2$$

$$\frac{\partial}{\partial p_u}\sum_{u,i\epsilon k}(p_uq_i + b_u + b_i - r_{ui})^2$$

p에 대해서 편미분

$$= \frac{\partial}{\partial p_u}\sum_{u,i\epsilon k}(p_uq_i)^2 + r_{ui}^2 - 2r_{ui}p_uq_i - 2r_{ui}b_u - 2r_{ui}b_i + 2b_ub_i + b_i^2 + b_u^2 + 2b_up_uq_i + 2b_ip_uq_i$$

$$= \sum_{u,i\epsilon k} 2q_i(p_uq_i + b_u + bi - r_{ui}) \approx \boxed{\sum_{u,i\epsilon k} q_i(p_uq_i + b_u + bi - r_{ui})}$$

$$\frac{\partial}{\partial q_i}cost(p_u, q_i, b_u, b_i) \approx \boxed{\sum_{u,i\epsilon k} p_u(p_uq_i + b_u + bi - r_{ui})}$$

$$\frac{\partial}{\partial b_i}\sum_{u,i\epsilon k}(p_uq_i + b_u + b_i - r_{ui})^2$$

bias 에 대해서 편미분

$$= \frac{\partial}{\partial b_i}\sum_{u,i\epsilon k}(p_uq_i)^2 + r_{ui}^2 - 2r_{ui}p_uq_i - 2r_{ui}b_u - 2r_{ui}b_i + 2b_ub_i + b_i^2 + b_u^2 + 2b_up_uq_i + 2b_ip_uq_i$$

$$= \sum_{u,i\epsilon k} 2(p_uq_i + b_i + b_u - r_{ui}) \approx \boxed{\sum_{u,i\epsilon k} p_uq_i + b_i + b_u - r_{ui}}$$

$$\frac{\partial}{\partial b_u}cost(p_u, q_i, b_u, b_i) \approx \boxed{\sum_{u,i\epsilon k} p_uq_i + b_i + b_u - r_{ui}}$$

- 편미분한 식들과 Linear Regression 과 비교

- 거의 유사하다

$$p_{u(new)} = p_{u(cur)} - \lambda \sum_{u,i\epsilon k} q_i(p_u q_i + b_u + bi - r_{ui})$$

$$b_{u(new)} = b_{u(cur)} - \lambda \sum_{u,i\epsilon k} p_u q_i + b_i + b_u - r_{ui}$$

$$\boxed{W_{(new)} = W_{(cur)} - \lambda \frac{1}{m} \sum_{i=1}^{m} (W x_{(i)} + b - y_{(i)}) x_{(i)}}$$

$$q_{i(new)} = q_{i(cur)} - \lambda \sum_{u,i\epsilon k} p_u(p_u q_i + b_u + bi - r_{ui})$$

$$b_{i(new)} = b_{i(cur)} - \lambda \sum_{u,i\epsilon k} p_u q_i + b_i + b_u - r_{ui}$$

$$\boxed{b_{(new)} = b_{(cur)} - \lambda \frac{1}{m} \sum_{i=1}^{m} W x_{(i)} + b - y_{(i)}}$$

3. Python Code

1. predicted_R 구하기

> epoch, iteration 개념은 [여기] 참고

```
iteration = 5000        # iteration: 학습, 즉 epoch 을 몇 번 할 것인가
k = 3                   # k: weight 의 차원
R = np.array([          # R: user 당 여행지에 대해 rating 한 값
    [2, 8, 9, 1, 8],
    [8, 2, 1, 8, 1],
    [1, 5, -1, 1, 7],   # -1 부분이 앞으로 예측해야 하는 값
    [7, 2, 1, 8, 1],
    [1, 8, 9, 2, 9],
    [9, 1, 2, -1, 2],
    [6, 1, 2, 7, 2]])

predicted_R = matrix_factorization(R, k, iteration)

print(predicted_R)
```

- 출력 결과
    - 주어진 R 과 비교하면 거의 동일하며 -1 부분만 새로 값이 들어간 것을 확인할 수 있다.

```
[[2.0  8.0  8.9  0.9  8.0]
 [7.9  1.9  1.0  8.0  0.9]
 [0.9  4.9  7.4  1.0  7.0]
 [7.0  2.0  0.9  7.9  1.0]
 [0.9  8.0  9.0  2.0  8.9]
 [8.9  1.0  2.0  9.2  1.9]
 [6.0  0.9  1.9  6.9  2.0]]
```

2. matrix_factorization 함수

```
import numpy as np

def matrix_factorization(R, k, iteration):

    user_count, item_count = R.shape

    weight_P = np.random.normal(size=(user_count, k))  # 임의의 P, Q
    weight_Q = np.random.normal(size=(item_count, k))  # k 개 차원

    bias_P = np.zeros(user_count)
    bias_Q = np.zeros(item_count)

    for iter in range(iteration):       # for 문을 통한 트레이닝 과정
        for u in range(user_count):     # 특정 열의 값과
            for i in range(item_count): # 특정 행의 값을 가져온다
                r = R[u, i]
                if r >= 0:  # -1 은 예측을 해야하는 값. 트레이닝 시 제외시킨다.
                    error = prediction(weight_P[u, :],
                                       weight_Q[i, :],
                                       bias_P[u],
                                       bias_Q[i]) - r  # error 값 구하기

                                                       # 편미분 값 구하기
                    weight_delta_Q, bias_delta_Q = gradient(error,
                                                            weight_P[u, :])
                    weight_delta_P, bias_delta_P = gradient(error,
                                                            weight_Q[i, :])

                    weight_P[u, :] -= weight_delta_P
                    bias_P[u] -= bias_delta_P

                    weight_Q[i, :] -= weight_delta_Q
                    bias_Q[i] -= bias_delta_Q

    return weight_P.dot(weight_Q.T)
            + bias_P[:, np.newaxis] + bias_Q[np.newaxis:, ]
```

3. Prediction 함수

$$r_{ui} \quad (predict) = p_u q_i + b_u + b_i$$

```
def prediction(P, Q, b_P, b_Q):

    return P.dot(Q.T) + b_P + b_Q
```

4. Gradient 함수

$$q_{i(new)} = q_{i(cur)} - \lambda \sum_{u,i\epsilon k} p_u(p_u q_i + b_u + bi - r_{ui})$$

$$b_{i(new)} = b_{i(cur)} - \lambda \sum_{u,i\epsilon k} p_u q_i + b_i + b_u - r_{ui}$$

```
def gradient(error, weight):

    learning_rate = 0.01

    weight_delta = learning_rate * np.dot(weight.T, error)

    bias_delta = learning_rate * np.sum(error)

    return weight_delta, bias_delta
```

4. 추가 학습

> 컨설턴트님께서 강의에서 말씀해주신 추가 학습하면 좋은 키워드 (+ 참고 링크)

1. SVD, SGD

- SVD와 Latent Factor 모형
- Matrix Factorization
- SGD (Stochastic Gradient Descent)
- Using Funk SVD with SGD?

2. Over fitting

- Overfitting (과적합)
- 모델 선택, 언더피팅(underfitting), 오버피팅(overfitting)
- 오버피팅 및 언더피팅 이해 및 극복하기