Recommender Report

2016024893 오성준

1. Summary of your algorithm

□ 저는 이번 과제를 gradient descent를 이용한 matrix factorization으로 해결하고자 했습니다. 제가 사용한Gradient descent를 이용한 matrix factorization의 개념은 다음과 같습니다.

$$\mathbf{R} \approx \mathbf{P} \times \mathbf{Q}^T = \hat{\mathbf{R}}$$

□ 행렬을 두개로 나눈다는 아이디어는 여타 다른 matrix factorization과 다르지 않지만, svd 같은 방법들이 효과를 발휘하지 못하는 것을 확인했기 때문에 (rmse 값이 1.5가 넘게 나왔습니다) 미분을 이용한 gradient descent 방법을 생각해냈습니다. 아무래도 95% 정도의 data가 비어있을 정도로 sparse하기 때문에 단순히 svd를 사용하면 문제가 생기는 것 같습니다.

□ 위의 그림에서 P의 shape는 (user수, k) Q의 shape는 (k, item수)로 잡아서 적당한 k를 잡을 수 있도록 했습니다. 그래서 결과적으로 원래 있던 R을 통해서 R hat을 만드는 것을 목적으로 수행했습니다. 코드에선 각각을 user matrix, item matrix로 구현했습니다.

$$\hat{r}_{ij} = p_i^T q_j = \sum_{k=1}^{k} p_{ik} q_{kj}$$

□ R hat의 값은 P와 Q의 내적으로 구할 수 있습니다.

$$e_{ij}^2 = (r_{ij} - \hat{r}_{ij})^2 = (r_{ij} - \sum_{k=1}^{K} p_{ik} q_{kj})^2$$

□ 이제 error를 구해서 learning을 구현할 것입니다. 에러는 위의 식과 같은데, R과 R hat의 차의 제곱을 구한 것과 같다고 할 수 있습니다. 이 식을 미분합니다.

$$\frac{\partial}{\partial p_{ik}} e_{ij}^2 = -2(r_{ij} - \hat{r}_{ij})(q_{kj}) = -2e_{ij} q_{kj}$$
$$\frac{\partial}{\partial q_{ik}} e_{ij}^2 = -2(r_{ij} - \hat{r}_{ij})(p_{ik}) = -2e_{ij} p_{ik}$$

□ 위의 식은 error의 제곱을 미분한 식으로 여가 learning rate인 alpha를 곱해서 P와 Q의 각 parameter들의 learning을 진행합니다.

$$p'_{ik} = p_{ik} + \alpha \frac{\partial}{\partial p_{ik}} e_{ij}^2 = p_{ik} + 2\alpha e_{ij} q_{kj}$$
$$q'_{kj} = q_{kj} + \alpha \frac{\partial}{\partial q_{kj}} e_{ij}^2 = q_{kj} + 2\alpha e_{ij} p_{ik}$$

- 위에서 말했다시피 이런 식으로 learning을 진행합니다.

$$e_{ij}^2 = (r_{ij} - \sum_{k=1}^{K} p_{ik} q_{kj})^2 + \frac{\beta}{2} \sum_{k=1}^{K} (||P||^2 + ||Q||^2)$$

- 여기에 저는 regularization이라는 방식을 사용했습니다. 즉, over fittng을 막기 위해서, error에 각각 일정 값을 더해줍니다. 여기서 오른쪽 위에 나와 있는 P는 P[i][k]를 의미하고 q는 Q[i][k]라고 생각하시면 됩니다. 여기에다가 beta/2를 곱해줘서 또 지나친 영향은 배제시키려 했습니다.

- 위의 과정을 정해진 epoch, k, alpha, beta로 진행하고 결과적으로 output file을 만들어 냈습니다.

2. Detailed description of your codes (for each function)

- Class recommender를 사용해서 구현했고 recommender에 함수들을 선언하고 사용했습니다. 이제 각 함수에 대한 내용을 설명하겠습니다.

1) Init function

```
def __init__(self, base_f, test_f):

        self.non_empty_count = 0

        # matrix 만들기
        self.test_file_name = test_f
        self.base_file_name = base_f
        self.user_item_matrix = self.base_file_read_create_matrix()

        # matrix factorization 구현
        user_matrix, item_matrix = self.matrix_factorization()
        self.user_item_matrix_hat = np.dot(user_matrix,item_matrix.T)

        # output file 만들기
        self.test_file_read_create_output()
```

- 이번 구현의 전체적인 flow를 맡았습니다. 주석에 적혀 있는 데로, 진행합니다.

- 첫번째로 입력을 받아서 user와 item을 row와 column으로 하고 rating을 저장한 user_item_matrix를 만듭니다.

- 두번째로 matrix factorization을 구현합니다. User matrix와 item matrix를 받아서 내적을 통해서

hat matrix를 만듭니다.

□ Hat matrix에 있는 값을 기준으로 prediction file을 만듭니다.

2) Base_File_Read_Create_Matrix function

```
def base_file_read_create_matrix(self):
        max_user_id = 0
        max_item_id = 0

        # 1차 file read 시작, 최대 id값들 받아올 예정
        f = open(self.base_file_name, 'r')

        while True:
            line = f.readline()
            if not line: break
            tmp = line[:-1].split('\t')
            #user\t item\t rating\t timestamp\n
            max_user_id = max(max_user_id, int(tmp[0]) )
            max_item_id = max(max_item_id, int(tmp[1]) )

        f.close()

        f = open(self.test_file_name, 'r')
        while True:
            line = f.readline()
            if not line: break
            tmp = line[:-1].split('\t')
            #user\t item\t rating\t timestamp\n
            max_user_id = max(max_user_id, int(tmp[0]) )
            max_item_id = max(max_item_id, int(tmp[1]) )

        f.close()

        # 1차 file read 끝, empty matrix 만들고 다시 한 번 읽을 것임
        print(max_user_id, max_item_id)
        # np empty array 만들기

        user_item_matrix = np.empty((max_user_id, max_item_id), float)

        # 2차 file read 시작, matrix 완성하고 return 할 예정
        f = open(self.base_file_name, 'r')

        while True:
```

```
            line = f.readline()
            if not line: break
            tmp = line[:-1].split('\t')
            # user\t item\t rating\t timestamp\n
            tmp = [int(x) for x in tmp]

            # 메모리 낭비 줄이기 위해서 user와 item id 모두 1을 뺀 값 사용함
            # 마지막에 처리해 줄 것
            self.non_empty_count += 1
            user_item_matrix[tmp[0] - 1][tmp[1] - 1] = tmp[2]

        f.close()

        return user_item_matrix
```

- 입력을 받은 파일을 두 번 읽습니다. 그러면서 user_item_matrix를 만들고 return 합니다.

- 첫번째 읽을 때는, base와 test file을 읽으면서, 최대 user id와 item id를 받아옵니다. 그걸 이용해서 (max user_id, max item_id)의 형태의 빈 Numpy array를 생성합니다.

- 한번 더 base file을 보면서 rating을 저장합니다. 자연스럽게 비어있는 값은 0이 됩니다. 이제 array를 return 합니다.

3) Matrix_Factorization function

```
def matrix_factorization(self):
        K = 5
        epochs = 20
        alpha = 0.001
        beta = 0.02

        R = self.user_item_matrix
        N = len(self.user_item_matrix)
        M = len(self.user_item_matrix[0])

        default = 0.75
        print(default)

        user_matrix = np.zeros((N, K)) + default
        item_matrix = np.zeros((M, K)) + default
        item_matrix = item_matrix.T

        for epoch in range(epochs + 1):
            E = R - np.dot(user_matrix, item_matrix)
```

```
            for i in range(N):
                for j in range(M):
                    if R[i][j] > 0:
                        # calculate error
                        eij = R[i][j] - np.dot(user_matrix[i,:],item_matrix[:,j])

                        for k in range(K):
                            user_matrix[i][k] = user_matrix[i][k] + alpha * (2 * eij * item_matrix[k][j])
                            item_matrix[k][j] = item_matrix[k][j] + alpha * (2 * eij * user_matrix[i][k])

            eR = R - np.dot(user_matrix,item_matrix)

            e = 0

            for i in range(N):
                for j in range(M):
                    if R[i][j] > 0:
                        e = e + pow(eR[i][j], 2)

                        for k in range(K):
                            e = e + (beta/2) * (pow(user_matrix[i][k],2) + pow(item_matrix[k][j],2))

            e = e / self.non_empty_count

            if epoch % 5 == 0: print(epoch, e)

        return user_matrix, item_matrix.T
```

- 1번에서 설명한 알고리즘을 구현한 코드입니다. User matrix와 item matrix를 return합니다. 알고리즘 그대로 구현한 코드이기 때문에, 자세한 설명은 생략하겠습니다. 다만 코드에서 default는 user matrix와 item matrix의 초기값을 설정한 것입니다. 뒤에서 또 설명하겠습니다.

- 사실 이 알고리즘에서 가장 어려웠던 건 k, epochs, alpha, beta, default와 같은 값을 구해야 하는 부분이었습니다. 저는 여러가지 시도를 하면서 가장 낮은 값을 만드는 parameter 값을 구했고 그 결과가 k = 5, epochs = 20, alpha = 0.001, beta = 0.02, default = 0.75 입니다.

- 하나 특이하게 생각했던 건 error가 줄어들면 rmse 값이 줄어 들거라 생각했으나 되려, overfitting이 일어나서 되려 값이 증가하는 상황이 일어났습니다.

- Epoch이 지나치게 증가하거나, alpha가 지나치게 증가하거나, beta가 지나치게 감소하면 이런 상황이 일어나는 것 같습니다.

- 되려 반대 상황에선 learning이 잘 진행되지 않아서 값이 잘 나오지 않았습니다.

- 그러므로 지금 제가 구한 parameter값들은 제가 귀납적으로 해본 결과이고, 저는 5개의 base file들을 이 이상 해결하지 못했습니다. 저는 지금 상태가 최적의 상태라고 생각하고 제출합니다.

4) Test_File_Read_Create_Outout function

```
def test_file_read_create_output(self):

        f = open(self.test_file_name, 'r')
        f_prediction = open(self.base_file_name + '_prediction.txt', 'w')

        while True:
            line = f.readline()
            if not line: break
            tmp = line[:-1].split('\t')
            tmp_int = [int(x) for x in tmp]
            #user\t item\t rating\t timestamp\n

            predict_rating = self.user_item_matrix_hat[tmp_int[0] - 1][tmp_int[1] - 1]
            f_prediction.write(tmp[0] + '\t' + tmp[1] + '\t' + str(predict_rating) + '\n')

        f.close()
        f_prediction.close()
```

- Prediction 값을 정해진 format과 이름으로 저장합니다.

3. Instructions for compiling your source codes at TA's computer

1) requirements.txt

- 프로젝트에 requirements.txt를 함께 보냅니다. 이전 버전들과 크게 다르지 않고, 몇가지 library를 설치하긴 했지만, 실제로 사용하진 않았기 때문에 별 문제가 없을 거라고 생각합니다. 폴더에서 보실 수 있습니다.

2) 실행 명령어

- 가상 환경에서 terminal에 python recommender.py (input file name) (test file name)을 입력해서 CLI로 사용할 수 있습니다.

```
(ds) seongjun-o@seongjun-oui-MacBookPro Project4_Recommender % python recommender.py u1.base u1.test
943 1682
0.75
0 1.0455969599410078
5 0.9295204733067496
10 0.9172561875471072
15 0.9130857788464432
20 0.9110492177916005
```

- 이 때, 첫 줄부터 순서대로,

- Max user id, max item id
- User matrix와 item matrix의 초기값
- Epoch 5번에 한번씩 error의 평균값을 보여줍니다.
- 마지막 줄은 걸린 시간을 초단위로 보여줍니다. 1분이 넘는 시간이 걸리기 때문에 참고 하셧으면 좋겠습니다.

3) 채점

- 노트북에서 exe 파일이 돌아가지 않아서 자체적으로 rmse.py라는 프로그램을 구현해서 채점했습니다. 그 보다 먼저로 친구한테 부탁해서 채점을 해봤고, 잘 작동하는 것을 확인했습니다. 아래는 제가 자체적으로 구현한 코드 입니다.

```
(ds) seongjun-o@seongjun-oui-MacBookPro Project4_Recommender % python rmse.py u1
0.9577897810411705
(ds) seongjun-o@seongjun-oui-MacBookPro Project4_Recommender % python rmse.py u2
0.9477779117799496
(ds) seongjun-o@seongjun-oui-MacBookPro Project4_Recommender % python rmse.py u3
0.9424114892666683
(ds) seongjun-o@seongjun-oui-MacBookPro Project4_Recommender % python rmse.py u4
0.9416085849951651
(ds) seongjun-o@seongjun-oui-MacBookPro Project4_Recommender % python rmse.py u5
0.9404456361459504
```

4. Any other specification of your implementation and testing

- 추가적인 파일을 2개 만들었습니다. 하나는 input data의 상태를 파악하기 위해서 만든 input.py와 전체 data의 평균과 값의 분포를 보여줍니다.

- 다른 하나는 rmse.py로 exe파일이 실행되지 않아서 제가 만든 채점 코드입니다.

- 기본적으로 Mac Os를 사용했기 때문에 문제가 생기면 알려주시길 바랍니다.