Implementation of Recommender

2018008395 박정호

1. Algorithm

이번 과제는 유저가 매긴 rating 정보를 이용해서 다음 상품을 추천하는 Recommendation System을 구현하는 것이었다. 필자는 강의에서 소개된 알고리즘 중에서 Matrix Multiplication을 사용했고, 이 구현을 위해 머신 러닝에서 흔히 사용하는 Gradient Descent를 사용했다.

Gradient Descent에 사용하는 Loss Function은 과제 채점에 사용하는 함수인 RMSE를 사용했고, 행렬 전체에 대한 gradient가 아닌, 유저가 매긴 rating 각각에 대한 gradient를 매 epoch마다 전부 사용해서 학습을 진행했다. 구현 자체는 단순한 형태였기에, 대부분의 시간은 regularization parameter와 learning rate의 최적값을 찾는데에 사용했다.

여러 번의 실험 끝에 regularization parameter는 0.05, learning rate는 0.005가 적어도 주어진 데이터에 대해서는 최적의 hyperparameter인 것을 알 수 있었다. 물론 이는 채점 기준 중 하나인 reasonable time에 대한 고려로 인해 Matrix Multiplication의 학습에 사용할 factor와 epoch를 비교적 작게 유지한 것이 영향을 주었을 것이라 생각한다. 좀 더 학습에 투자할 수 있는 시간이 길었다면 다른 조합의 hyperparameter가 최적이었을 수도 있다고 생각한다.

따로 validation set을 마련한다던지, overfitting을 방지하기 위해 early stopping을 구현하지는 않았다. 애초에 iteration 횟수가 50으로 상당히 작다고 생각했기 때문에, early stopping이 그리 큰 의미를 갖지 못할 것이라 생각했기 때문이다.

기본적으로 test set에는 training set에 없는 user나 item이 들어오지 않는다는 언급이 있었으나, 실제 테스트 셋에 이것이 존재하는 것을 확인했다. 따라서, 해당 유저가 training set에 없다면 training set에 주어진 rating의 평균값을, 해당 유저는 training set에 있었으나 item은 없었다면 해당 유저의 predicted rating의 평균값을 예상 결과로 했다.

2. Implementation

1. 입력 처리

```
# read input file(training file, test file)
# it is just wrapper or pandas read_csv
# it adds some column names
def read_data(filename):
    df = pd.read_csv(filename,
                     sep='\t',
                     header=None,
                     names=['user_id', 'item_id', 'rating', 'timestamp'])

    return df
```

단순히 pandas의 read_csv함수의 wrapper함수이다. 다만, main 함수에서의 흐름을 깔끔하게 하기 위해 이런 식으로 분리해두었다. 후의 구현의 가독성을 위해 각 열의 명칭을 따로 추가했다.

2. Recommender Class

1. 기본 구성

```
# Recommender System Class (Matrix Multiplication)
class Recommender:
    # Constructor
    def __init__(self, rating_matrix, factor, learning_rate, regularization):
        self.rating_matrix = rating_matrix.values
        self.idx = np.array(self.rating_matrix.nonzero()).T
        self.n = self.idx.size

        self.num_user = rating_matrix.shape[0]
        self.num_item = rating_matrix.shape[1]
        self.factor = factor
        self.regularization = regularization

        self.learning_rate = learning_rate

        self.user_idx_map = {}
        self.item_idx_map = {}

        idx = 0
        for user_id in rating_matrix.index:
            self.user_idx_map[user_id] = idx
            idx+=1
```

```
        idx = 0
        for item_id in rating_matrix.columns:
            self.item_idx_map[item_id] = idx
            idx+=1

    # Training Function
    # using Gradient Descent
    def train(self, num_iterations = 50):
        …

    # prediction function for single user-item pair
    def predict(self, user_id, item_id):
        …

    # test function
    def test(self, test_dataset):
        ...
```

좀 더 가독성 있는 구현을 위해 class를 구현했다. 생성자 이외의 함수는 후술하겠으며, 여기서는 기본적인 구조만 다룬다. idx는 sparse한 rating matrix에서 rating이 매겨진 칸이 어디인지를 저장한 배열이며, user_idx_map과 item_idx_map은 [0, num_users), [0, num_items)의 범위 내로 user id와 item id가 들어온다는 보장이 없었으므로, rating matrix의 idx와 실제 id를 매핑하기 위해 만들었다.

2. Training

```
    # Training Function
    # using Gradient Descent
    def train(self, num_iterations = 50):
        self.U = np.random.normal(scale=0.1, size=(self.num_user, self.factor)
)
        self.V = np.random.normal(scale=0.1, size=(self.num_item, self.factor)
)

        self.U_bias = np.zeros(self.num_user)
        self.V_bias = np.zeros(self.num_item)

        self.bias = self.rating_matrix[self.rating_matrix != 0].mean()

        for _ in range(num_iterations):
            np.random.shuffle(self.idx)
            for user_id, item_id in self.idx:
                predicted_rating = self.predict(user_id, item_id)
                rating = self.rating_matrix[user_id][item_id]
```

```
                error = rating - predicted_rating

                self.U_bias[user_id] += self.learning_rate * (error - self.reg
ularization * self.U_bias[user_id])
                self.V_bias[item_id] += self.learning_rate * (error - self.reg
ularization * self.V_bias[item_id])

                dU = error * self.V[item_id] - self.regularization * self.U[us
er_id]
                dV = error * self.U[user_id] - self.regularization * self.V[it
em_id]

                self.U[user_id] += self.learning_rate * dU
                self.V[item_id] += self.learning_rate * dV
```

실제 데이터로 학습하는 함수이다. 앞에서 언급했듯이 gradient descent를 사용했으며, 전체 matrix에 대한 gradient를 구하기보다는, 각 rating에 대한 gradient를 구하는 식으로 진행했다. 일종의 batch를 쓴 효과를 내려 한 것이다. 혹시라도 학습에 사용되는 rating의 고정된 순서에 영향을 받을까 싶어 매 epoch마다 idx를 무작위 셔플했다. 다만 실험 결과 이것이 학습에 미치는 영향은 거의 없었다.

```
    # prediction function for single user-item pair
    def predict(self, user_id, item_id):
        return self.bias + self.U_bias[user_id] + self.V_bias[item_id] + np.do
t(self.U[user_id], self.V[item_id])
```

train 함수에 사용된 개별 user-item pair에 대한 rating prediction 함수이다. 현재까지 구해진 bias, U, V를 모두 사용해서 하나의 rating prediction을 내놓는다.

3. Test

```
    # test function
    def test(self, test_dataset):
        result = []
        predicted_matrix = self.bias + np.array([self.U_bias]).T + np.array([s
elf.V_bias]) + np.dot(self.U, self.V.T)

        for _, row in test_dataset.iterrows():
            user_id = row['user_id']
            item_id = row['item_id']

            predicted_rating = None

            if user_id in self.user_idx_map:
                user_idx = self.user_idx_map[user_id]
                if item_id in self.item_idx_map:
```

```
                    item_idx = self.item_idx_map[item_id]

                    predicted_rating = min(5, max(1, predicted_matrix[user_idx
][item_idx]))
                else:
                    predicted_rating = min(5, max(1, predicted_matrix[user_idx
].mean()))
            else:
                predicted_rating = self.bias

            result.append(predicted_rating)

        return result
```

주어진 test set에 대해서 prediction을 수행하는 함수이다. 여기서는 test set에서의 예외상황을 처리하기 위해 training set에 없었던 유저나 아이템에 대한 처리를 추가했고, 또한 학습 결과가 [1, 5]의 범위에 들어오도록 조정했다.

3. 출력 처리

```
# write test result to output file
def output_process(filename, test_dataset, test_result):
    file = open(filename, mode='w')

    for idx, row in test_dataset.iterrows():
        file.write(str(row['user_id']))
        file.write('\t')
        file.write(str(row['item_id']))
        file.write('\t')
        file.write(str(test_result[idx]))
        file.write('\n')
```

앞의 test 함수에서 return한 test_result와 test set을 사용해서 output file을 만드는 함수이다. 이전과는 달리 매 데이터를 읽을 때마다 테스트하지 않고, 모든 테스트가 끝난 뒤에 출력하는 방식을 선택했다.

4. main 함수

```
def main(argv):
    if len(argv)<3:
        print('PLEASE, give 2 arguments (train filename, test filename)')
        return

    train_dataset = read_data(argv[1])
```

```
    test_dataset = read_data(argv[2])

    rating_matrix = pd.pivot_table(train_dataset,
                                   index='user_id',
                                   columns='item_id',
                                   values='rating').fillna(0)

    recommender = Recommender(rating_matrix, 10, 0.005, 0.05)
    start_time = time.time()
    recommender.train()
    end_time = time.time()
    print('Training Time : ', end_time-start_time)

    start_time = time.time()
    test_result = recommender.test(test_dataset)
    output_process(argv[1]+'_prediction.txt', test_dataset, test_result)
    end_time = time.time()
    print('Test and Output Time : ', end_time-start_time)
```

위에서 구현한 함수를 모두 사용하는 함수이다. 또한 여기서 training dataset을 pivot table로 만들어서 rating matrix를 구성했다. 학습시간과 테스트 시간을 분리해서 측정했다.

3. Usage

우선 전체 코드는 python으로 작성되었으며, 버전은 3.8.2이다.

numpy 와 pandas를 활용했으며, 버전은 각각 1.18.2, 1.2.3이다.

또한 python은 인터프리터 언어이기에 별도의 컴파일 명령어나 Makefile을 쓰지 않았으며, 아래의 밑줄 그어진 부분의 명령어로 실행할 수 있다. 필자는 .py 파일에 대한 환경변수 설정이 되어있지 않았기 때문에 파일명만으로 실행하는 것이 불가능했음을 밝힌다.

```
PS C:\Users\pch68\2021_ite4005_2018008395\long-term project> python .\recommender.py u1.base u1.test
Training Time :  160.78348755836487
Test and Output Time :  3.507686138153076
```

4. Result

```
PS C:\Users\pch68\2021_ite4005_2018008395\long-term project> python .\recommender.py u3.base u3.test
Training Time :  109.33841896057129
Test and Output Time :  3.6039881706237793
PS C:\Users\pch68\2021_ite4005_2018008395\long-term project> python .\recommender.py u4.base u4.test
Training Time :  99.16807579994202
Test and Output Time :  3.5317986011505127
PS C:\Users\pch68\2021_ite4005_2018008395\long-term project> python .\recommender.py u5.base u5.test
Training Time :  102.3893518447876
Test and Output Time :  3.519120216369629
PS C:\Users\pch68\2021_ite4005_2018008395\long-term project> .\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: 0.9372932
PS C:\Users\pch68\2021_ite4005_2018008395\long-term project> .\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: 0.9261257
PS C:\Users\pch68\2021_ite4005_2018008395\long-term project> .\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: 0.9213368
PS C:\Users\pch68\2021_ite4005_2018008395\long-term project> .\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: 0.9186312
PS C:\Users\pch68\2021_ite4005_2018008395\long-term project> .\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: 0.9218336
```

대략적으로 학습에 드는 시간은 2분 남짓, 스크린샷에는 보이지 않았지만, 오래 걸리는 경우에는 3분 가까이까지도 갔다. 그에 비해 test 시간은 3,4초로 아주 짧았다. RMSE는 대체로 0.92 정도인 것을 볼 수 있다.