Assignment #4

Recommendation System

한양대학교 컴퓨터소프트웨어학부 2015004911 임성준
데이터사이언스(ITE4005, 12877) - 김상욱 교수님

---

1. 개요

Recommendation System이란 trining data를 사용하여 test data를 예측하여, 사용자에 맞게 추천하는 시스템이다.
이번 과제에서는 "Matrix Factorization" 알고리즘을 사용하여 추천 시스템을 구현하였다. Matrix Factorization 알고리즘은 collaborative filtering 알고리즘의 일종이다.

2. 목적

이 프로젝트의 목적은 주어진 training data를 이용하여 영화의 평점을 예측하는 모델을 만들고, test data를 이 모델에 적용하는 것이다.

3. 개발환경

```
OS : macOS Big Sur 11.3
Language : Python 3.9.1
Source code editor : Visual Studio Code
```

4. 프로그램 구조 및 알고리즘

1. training file, test file을 입력한다.
2. 입력한 file로부터 data를 읽어온다.
3. training data를 Matrix Factorization에 대입해 model을 구축한다.
4. 구축된 model에 test data를 대입해 확인한다.
5. 결과를 output file에 쓴다.

5. 코드 설명

```python
class MatrixFactorization():
    def __init__(self, rating, k, learning_rate, reg_param, epochs, test_data):
        """
        rating: Rating matrix
        k: Latent parameter
        learning_rate: Learning rate
        reg_param: Regularization Strength
        epochs: Training epochs
        """
```

```
        self.rating = rating.values
        self.k = k
        self.learning_rate = learning_rate
        self.reg_param = reg_param
        self.epochs = epochs

        self.test_data = test_data

        self.n_users, self.n_items = rating.shape

        # init latent features
        self.m_user = np.random.normal(size=(self.n_users, k))
        self.m_item = np.random.normal(size=(self.n_items, k))

        # init biases
        self.b_user = np.zeros(self.n_users)
        self.b_item = np.zeros(self.n_items)
        self.bias = np.mean(self.rating[np.where(self.rating != 0)])

        self.user_id_idx = {}
        self.item_id_idx = {}

        for idx, user_id in enumerate(rating.index):
            self.user_id_idx[user_id] = idx
        for idx, item_id in enumerate(rating.columns):
            self.item_id_idx[item_id] = idx
```

- `class MatrixFactorization` : 주어진 rating으로 Matrix Factorization 연산을 수행한다.
- `__init__` : 변수들을 초기화한다.

```
    def train(self):
        self.training_process = []

        for epoch in range(self.epochs):
            for i in range(self.n_users):
                for j in range(self.n_items):
                    if self.rating[i][j] > 0:
                        self.gradient_descent(i, j)

            cost = self.get_cost()
            self.training_process.append((epoch, cost))

            print("Epoch: %d/%d, Cost = %.4f" %(epoch+1, self.epochs, cost))
```

- `train` : 정의된 epoch만큼 training을 진행한다.

```
    def gradient_descent(self, i, j):
        # get error
        prediction = self.get_prediction(i,j)
        error = self.rating[i][j] - prediction

        # update biases
        self.b_user[i] += self.learning_rate * (error - self.reg_param *
self.b_user[i])
        self.b_item[j] += self.learning_rate * (error - self.reg_param *
self.b_item[j])

        # update latent feature
        d_user = (error * self.m_item[j, :]) - (self.reg_param * self.m_user[i,
:])
        d_item = (error * self.m_user[i, :]) - (self.reg_param * self.m_item[j,
:])
        self.m_user[i, :] += self.learning_rate * d_user
        self.m_item[j, :] += self.learning_rate * d_item
```

- `gradient_descent` : train()에서, epoch마다 bias와 latent 를 재설정한다.

```
    def get_prediction(self, i, j):
        return self.bias + self.b_user[i] + self.b_item[j] +
np.dot(self.m_user[i, :], self.m_item[j, :].T)

    def get_cost(self):
        cost = 0
        xi, yi = self.rating.nonzero()
        predicted = self.bias + self.b_user[:, np.newaxis] +
self.b_item[np.newaxis:, ] + np.dot(self.m_user, self.m_item.T)

        for x, y in zip(xi, yi):
            cost += pow(self.rating[x,y] - predicted[x,y], 2)

        return np.sqrt(cost) / len(xi)
```

- `get_prediction` : bias값을 반영한 값을 출력한다.
- `get_cost` : loss function이다. 실제 값과 예측 값 사이의 제곱으로 계산된다.

```
    def test(self):
        test_user_id = self.test_data[:, 0]
        test_item_id = self.test_data[:, 1]
        R = self.bias + np.expand_dims(self.b_user, -1) +
np.expand_dims(self.b_item, 0) + np.dot(self.m_user, self.m_item.T)
```

```
        ret = []
        for user_id, item_id in zip(test_user_id, test_item_id):
            if item_id in self.item_id_idx:
                user_idx = self.user_id_idx[user_id]
                item_idx = self.item_id_idx[item_id]

                r = max(0, R[user_idx][item_idx])
                r = min(5, r)
                ret.append(r)
            else:
                ret.append(self.bias)

        return np.array(ret)
```

- `test` : test data를 위에서 구현한 model에 적용한다.

```
def read_data(train_file, test_file):
    # Read file
    header = ['user_id', 'item_id', 'rating', 'time_stamp']
    train_data = pd.read_csv('data-2/'+train_file, sep='\t', names=header)
    test_data = pd.read_csv('data-2/'+test_file, sep='\t', names=header)

    # Remove time stamp
    train_data.drop('time_stamp', axis=1, inplace=True)
    test_data.drop('time_stamp', axis=1, inplace=True)

    # Make pivot table
    rating = pd.pivot_table(train_data, 'rating', index='user_id',
columns='item_id').fillna(0)

    train_data = np.array(train_data)
    test_data = np.array(test_data)

    return train_data, test_data, rating

def write_data(test_data, rating_result, train_file):
    user_id = test_data[:, 0]
    item_id = test_data[:, 1]

    with open('test/'+train_file+'_prediction.txt', 'w') as file:
        for u, i, r in zip(user_id, item_id, rating_result):
            file.write(str(u) + '\t' + str(i) + '\t' + str(r) + '\n')
```

```
if __name__ == "__main__":
    train_file = sys.argv[1]
    test_file = sys.argv[2]

    train_data, test_data, rating = read_data(train_file, test_file)

    LATENT = 3
    LEARNING_RATE = 0.001
    REGULAR_STRENGTH = 0.02
    NUM_EPOCHS = 100

    model = MatrixFactorization(rating=rating, k=LATENT,
learning_rate=LEARNING_RATE,
                    reg_param=REGULAR_STRENGTH, epochs=NUM_EPOCHS,
test_data=test_data)
    model.train()
    rating_result = model.test()

    write_data(test_data, rating_result, train_file)
```

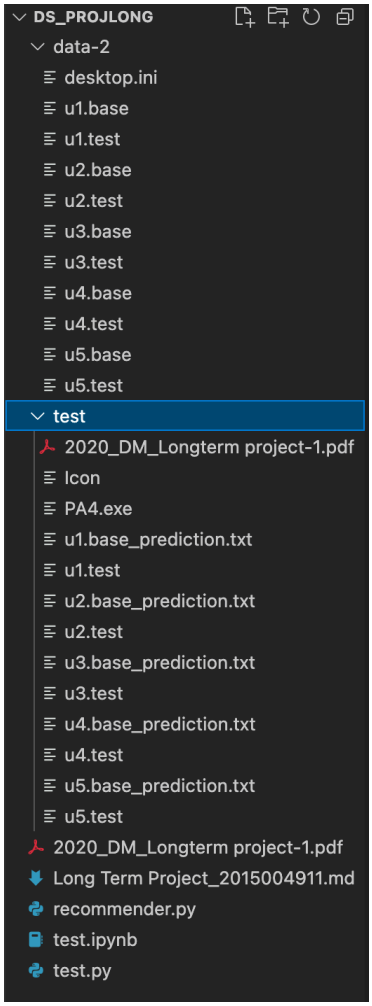
- `read_data` : train file과 test file을 읽고, data를 저장한다.
- `write_data` : 저장된 rating의 에측값을 output file에 쓴다.
- `__main__` : main.

6. 실행 방법 및 실행 결과

```
(base) limsungjun@Sungjuns-MacBook DS_projLONG % python3 recommender.py u1.base u1.test
Epoch: 1/100, Cost = 0.0060
Epoch: 2/100, Cost = 0.0053
Epoch: 3/100, Cost = 0.0048
$Epoch: 4/100, Cost = 0.0045
Epoch: 5/100, Cost = 0.0043
Epoch: 6/100, Cost = 0.0042
Epoch: 7/100, Cost = 0.0040
```

- `python3 recommender.py training_file test_file`

python으로 구현했기에 terminal 상에서는 python을 실행하는 명령어 **`python3`** , **`recommender.py`** , **`training file`** , **`test file`** 순서로 입력해서 실행한다. `read_data(), write_data()` 코드에 *input file*과 *output file*의 경로를 설정했다. 그렇기 때문에, input file (training file, test file)은 **`data-2`** 폴더에 있어야 하고, output file은 **`test`** 폴더에 생성된다.



구현한 코드의 file directory 구조이다.