Recommender System

실행환경

OS: Mac OS, Windows 10

Language: Python 3.9

실행방법

```
// In the Recommender system folder.

python recommender.py [trainset_file_path] [testset_file_path]

Ex) python recommender.py ./test/u1.base ./test/u1.test
```

알고리즘

Data Preprocessing

```
train_file_path = './test/u1.base'
test_file_path = './test/u1.test'
result_file_path = train_file_path + "_prediction.txt"
columns = ["user_id", "item_id", "rating", "time_stamp"]

train_df = pd.read_csv(train_file_path, names=columns, delimiter='\t')
test_df = pd.read_csv(test_file_path, names=columns, delimiter='\t')

# We are not going to use the last column "time_stamp".
train_df.drop(columns=train_df.columns[-1], inplace=True)
test_df.drop(columns=test_df.columns[-1], inplace=True)

train_user_item_matrix = train_df.pivot(index='user_id', columns='item_id', values='rating')
test_user_item_matrix = test_df.pivot(index='user_id', columns='item_id', values='rating')
```

파일에서 데이터를 불러와서 train rating matrix, test rating matrix를 만든다

Matrix Factorization

변수 선언 Phase

```
# test 빼기
def matrix_factorization(train_matrix, test_matrix, lmbd=0.02, k=1, std_dev=0.5, epochs=100, learning_rate=0.01):
    num_of_user = len(train_matrix.index)
    num_of_item = len(train_matrix.columns)

    u = std_dev * np.random.randn(num_of_user, k) # user embedding
    v = std_dev * np.random.randn(num_of_item, k) # item embedding

    bias_user = np.zeros((num_of_user, 1))
    bias_item = np.zeros((num_of_item, 1))

    matrix_mean = np.mean(train_matrix.values[~np.isnan(train_matrix.values)])

    train_not_nan_indices = train_matrix.stack(dropna=True).index.tolist()
    test_not_nan_indices = test_matrix.stack(dropna=True).index.tolist()

    train_set = [(i, j, train_matrix.loc[i, j]) for i, j in train_not_nan_indices]
    test_set = [(i, j, test_matrix.loc[i, j]) for i, j in test_not_nan_indices]
```

```
    train_user_item_matrix_index = train_matrix.index.tolist()
    train_user_item_matrix_columns = train_matrix.columns.tolist()

    train_avg_costs = []
    test_avg_costs = []
```

rating 값이 존재하는 user수에 맞게, U vector (user vector), V vector (item vector)을 만든다.

이때 주어진 k값에 의하여 latent matrix의 dimension이 결정된다.

bias user, bias item을 zero로 initialize한다. zero로 하는 이유는 따로 실험해본 결과 np.random.rand, np.random.randn을 활용하여 initialize했을 때보다 훨씬 성능이 잘 나왔기 때문이다.

존재하는 rating값들로 평균을 계산한 후, train matrix, test matrix에서 rating 값이 존재하는 row, column, rating value값을 저장한다.

학습 Phase

```
# continued from def matrix_factorization(... arg): ...

    for epoch in range(epochs):
        np.random.shuffle(train_set)

        train_avg_cost = 0
        test_avg_cost = 0

        for i, j, ground_truth in train_set:
            absolute_i = train_user_item_matrix_index.index(i)
            absolute_j = train_user_item_matrix_columns.index(j)

            logit = matrix_mean + bias_user[absolute_i] + bias_item[absolute_j] + u[absolute_i, :].dot(v[absolute_j, :].T)
            e = ground_truth - logit

            u[absolute_i, :] += learning_rate * (e * v[absolute_j, :] - lmbd * u[absolute_i, :])
            v[absolute_j, :] += learning_rate * (e * u[absolute_i, :] - lmbd * v[absolute_j, :])

            bias_user[absolute_i] += learning_rate * (e - lmbd * bias_user[absolute_i])
            bias_item[absolute_j] += learning_rate * (e - lmbd * bias_item[absolute_j])

            rms =  sqrt(mean_squared_error([ground_truth], logit))
            train_avg_cost += rms / len(train_set)

        for i, j, ground_truth in test_set:
            if i not in train_user_item_matrix_index or j not in train_user_item_matrix_columns:
                continue

            absolute_i = train_user_item_matrix_index.index(i)
            absolute_j = train_user_item_matrix_columns.index(j)

            logit = matrix_mean + bias_user[absolute_i] + bias_item[absolute_j] + u[absolute_i, :].dot(v[absolute_j, :].T)

            test_rms = sqrt(mean_squared_error([ground_truth], logit))
            test_avg_cost += test_rms / len(test_set)

        train_avg_costs.append(train_avg_cost)
        test_avg_costs.append(test_avg_cost)

        print('Epoch: {} / {}\ntrain cost: {}\ntest cost: {}'.format(epoch + 1, epochs, train_avg_cost, test_avg_cost))

        if epoch > 0:
            if test_avg_costs[-2] < test_avg_cost:
                return matrix_mean, u, v, bias_user, bias_item

    return matrix_mean, u, v, bias_user, bias_item
```

train set의 순서가 결과에 영향을 끼치는 것을 방지하기 위해 train set을 shuffle해주었다.

이후 train set에서 row, column, rating value를 가져와

$$\min_{p^*,q^*,b^*} \sum_{(u,i)\in\kappa} (r_{ui} - \mu - b_u - b_i - p_u{}^T q_i)^2 + \lambda$$
$$(\| p_u \|^2 + \| q_i \|^2 + b_u{}^2 + b_i{}^2)$$

이 식을 minimize 시키기 위하여 u vector, v vector, user bias, item bias로 편미분 한 후, gradient descent방법을 활용해 delta값을 기존값에서 빼 준다. (기존 식에 마이너스가 붙었기 때문에 코드에서는 “+”로 표현되었다.)

람다는 overfitting을 막기위한 정규화 parameter이며 0.02가 실험적으로 가장 좋은 값임이 확인되었다.

이때, k = 1일때가 가장 성능이 좋았는데, 직관적으로 이해가 가지 않았다.

matrix factorization library를 직접 불러와 다양한 k값에 대해서 실험을 진행했을 때에도 k = 1일때가 가장 좋은 성능을 보였다.

또한 주어진 데이터셋에 대하여 구현된 알고리즘과 불러온 matrix factorization library의 RMSE를 비교했을 때, 전자에서는 0.99 ~ 1.03 사이의 값을, 후자에서는 0.94 ~ 0.96의 값을 보여주었다.

1. 0.05정도의 차이는 구현의 디테일 차이에 의해 나는 오차라고 판단
2. library에서도 k = 1일때 가장 좋은 성능

이기에 따라서 알고리즘의 문제가 아니라고 판단하고 계속 구현을 진행하였다.

Ex) 직접구현_실험_log.txt

```
K: 1  lambda: 0.0 train_loss: 0.7392179173467098  test_loss: 0.8265095474404268 epoch: 3
K: 1  lambda: 0.0021660617565076304 train_loss: 0.7393317328949738  test_loss: 0.8282792686203195 epoch: 3
K: 1  lambda: 0.004321373782642483  train_loss: 0.738786879441312 test_loss: 0.8267131778263104 epoch: 3
K: 1  lambda: 0.006466042249231586  train_loss: 0.7391645895272773  test_loss: 0.8274744799228975 epoch: 3

... (more than 200 lines)

K: 39 lambda: 0.0021660617565076304 train_loss: 0.6004232758950695  test_loss: 1.0117855500906454 epoch: 7
K: 39 lambda: 0.004321373782642483  train_loss: 0.5991166743946205  test_loss: 1.0146027395696322 epoch: 7
K: 39 lambda: 0.006466042249231586  train_loss: 0.5991662374554085  test_loss: 1.0091856308630311 epoch: 7
```

Test set은 validation set으로 활용되었다. 존재하지 않는 item들에 대해서는 validation과정을 스킵하였다.

validation loss가 하락하다가 증가하는 시점에서 early stopping을 하였다.

Prediction

```
predicted_user_item_matrix = u.dot(v.T) + mean + bias_user + bias_item.T

predicted_user_item_matrix[predicted_user_item_matrix < 0] = 0
predicted_user_item_matrix[predicted_user_item_matrix > 5] = 5

result_df = pd.DataFrame(predicted_user_item_matrix).apply(lambda x: np.round(x, 0))

result_df = result_df.set_index(train_matrix.index)
result_df.columns = train_matrix.columns
```

matrix factorization으로 계산된 u, v, bias 값들을 활용하여 prediction matrix를 생성한다.

이때 0보다 작은 값에는 0을 5보다 큰 값에는 5를 할당해주고 각 rating value는 반올림 했다.

```
for i in range(test_df.shape[0]):
    user_id, item_id = test_df.values[i][0], test_df.values[i][1]
    if user_id not in result_df.index:
        result_df.loc[user_id] = round(mean)
    if item_id not in result_df.columns:
        result_df[item_id] = round(mean)
```

존재하지 않는 값에 대해선 global mean value를 넣어주었다.