0 main() / do() / data 초기화 / RSME

```
if __name__ == "__main__":
    global df_target, prediction_cf, decomposistion, opt_decomposistion
    trainset, df_train = initialize_train_data()
    df_target = initialize_test_data()
    df_implicit = initialize_implicit_data()
    # task1 initialize
    similarity = cosine_similarity(df_train.fillna(0))
    # task 2 initialize
    decomposistion = initialize_task2(df_train)
    # task 3
    opt_decomposistion = initialize_task2(df_train, 500)

    user_ids = read_user_id()
    result = do(user_ids)
    write_output(result)
```

Trainset, Testset, task1의 similarity, task2(3)의 matrix factorization처럼 여러 번 사용되는 데이터들은 main에서 초기화해서 실행시간을 줄이고자 하였다.

```
def do(inputs):
    string_results = []
    for user, movie in inputs:
        key_user = int(user)
        key_movie = int(movie)
        print("(user, movie) = (%d, %d)"%(key_user, key_movie))
        string_results.append(task1(key_user, key_movie))
        string_results.append(task2(key_user, key_movie))
        string_results.append(task3(key_user, key_movie))
    return string_results
```

저번 과제와 동일한 구조로, main()으로부터 호출된 do()는 user와 movie에 대한 정보를 받아, task 1,2,3을 각각 실행한다.

```
def RMSE(predictions, targets):
    return np.sqrt(((predictions - targets)**2).mean())
```

RMSE는 numpy를 사용해서 측정한다.

1 Item-based Collaborative Filtering

```
def task1(user, movie):
    prediction_cf = task1_predict(similarity, user)
    prediction = str(prediction_cf.round(4).loc[movie]['prediction'])
    if is_in_trainset(user, movie):
        print("Task1 RMSE: ", task1_RMSE(prediction_cf, user))
    else:
        print("Task3 RMSE : ", task1_RMSE(prediction_cf, user, 'test'))
    return ','.join([str(user), str(movie), prediction])
```

```
def task1_predict(similarity, userId):
#     movieIds = sorted(trainset['movieId'].unique())
    similarity = pd.DataFrame(similarity, index=movieIds, columns=movieIds)
    rated_movies = trainset[trainset.userId == userId]['movieId'].to_frame()
    user_sim = similarity.loc[rated_movies.movieId]
    user_rating = trainset[trainset.userId == userId]['rating'].to_frame()
    sim_sum = user_sim.sum(axis=0).to_frame()
    prediction = np.matmul(user_sim.T.to_numpy(), user_rating.to_numpy())/(sim_sum.to_numpy() + 1)
    prediction.round(4)
    prediction = pd.DataFrame(prediction, index=sorted(movieIds), columns=['prediction'])
    return prediction
```

사용자가 평점을 남긴 영화들과 비슷한 영화들을 중심으로 similarity표를 통해서 예측되는 값을 유추한다.

```
# for user uid, get prediction test
def task1_RMSE(prediction_cf, uid, mode='train'):
    rmse = 0
    targets = []
    prediction = []
    if mode=='train':
        dataset = trainset[trainset.userId==uid]
    else:
        dataset = df_target[df_target.userId==uid]
    for index, rows in dataset.iterrows():
        try:
            prediction.append(prediction_cf.loc[rows['movieId']]['prediction'])
            targets.append(rows['rating'])
        except KeyError:
            pass

    rmse  = RMSE(np.asarray(prediction), np.asarray(targets))
    return rmse
```

예측한 점수가 얼마나 정확한지 평가하기 위해 RMSE 를 사용한다.

과제의 경우에는 trainset에 있는 데이터로 RSME를 검증하지만, 모델이 얼마나 정확한지 확인하기 위해서 옵션으로 ['train', 'test']를 둬서 측정할 수 있도록 했다.

2 Matrix Factorization

```
def task2(user, movie):
    prediction_mf = decomposistion[int(user)]
    prediction = str(prediction_mf.round(4).loc[movie])
    if is_in_trainset(user, movie):
        print("Task2 RMSE : ", task2_RMSE(user))
    else :
        print("Task2 RMSE : ", task2_RMSE(user, 'test'))
    return ','.join([str(user), str(movie), prediction])
```

미리 계산해 decomposition표 (matrix factorization)에서 해당 user와 movie에 대한 좌표를 찾으면 predict값을 받을 수 있다.

```
def task2_RMSE(uid, mode='train'):
    rmse  = 0
    targets = []
    prediction = []

    if mode=='train':
        dataset = trainset[trainset.userId==uid]
    else:
        dataset = df_target[df_target.userId==uid]
    for index, rows in dataset.iterrows():
        try:
            prediction.append(decomposistion.loc[int(rows['movieId']),int(rows['userId'])])
            targets.append(rows['rating'])
        except KeyError:
            pass
    rmse = RMSE(np.asarray(prediction), np.asarray(targets))
    return rmse
```

Task1과 마찬가지로 RMSE 계산에 option을 두었다.

```python
def build_svd(model, K=400):
    # task 2
    # fill 0 by the mean values of movies
    filled_model= model.apply(lambda row: row.fillna(row.mean()), axis=1)
    # df_trainshape
    u, s, vh = np.linalg.svd(filled_model, True)
    u = u[:,:K]
    Sigma = np.diag(s[:K])
    vh = vh[:K, :]

    user_factors =  np.matmul(u, np.sqrt(Sigma))
    item_factors =  np.matmul(np.sqrt(Sigma),vh)
    df_prediction = pd.DataFrame(np.matmul(user_factors, item_factors),index=model.index, columns=model.columns)
    df_prediction = df_prediction.round(4)
    return df_prediction
```

K의 기본 값을 400으로 셋팅하여 decomposed 된 matrix을 찾는 과정이다. 'numpy.linalg.svd'을 사용해서 u, sigma, v까지는 구할 수 있지만, K개에 맞춰서 자르는 과정이 별도로 필요했다.

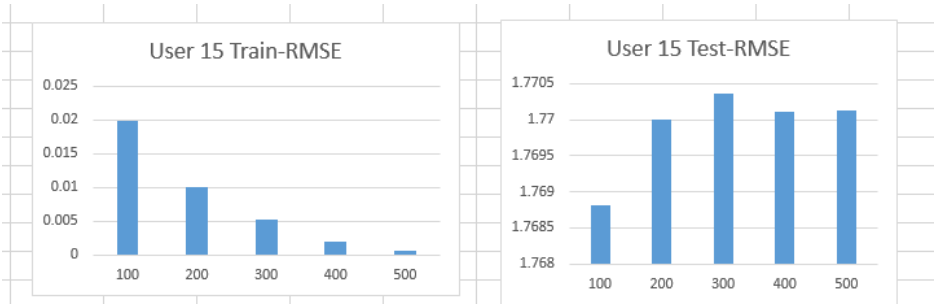
SVD 모델을 생성하는 단계로 factor의 사이즈에 맞게 User factor와 item factor를 자르는 작업이 익숙하지 않아 애를 먹었다. 이 과정에서 dot 연산과 @연산이 동일한 연산임을 배웠다.

3 모델 최적화

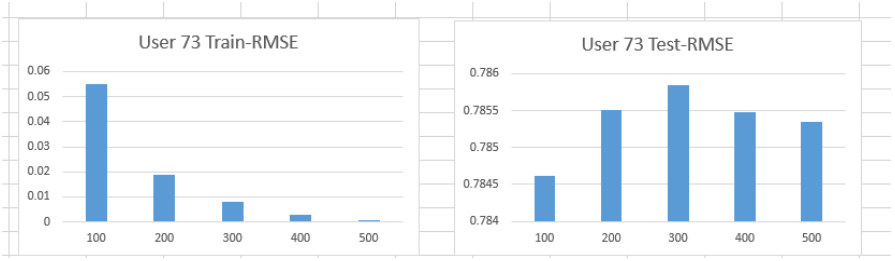
[실험]

2번 SVD 모델을 최적화하기 위해서 RMSE가 가장 작은 K를 찾았다. 기본적인 구조는 2번과 동일하며, 정확도를 위해서 test dataset에서 RMSE를 metric으로 사용해서 정확도를 측정했다. overfitting이 발생할 수 있기 때문에 방지하기 위해 최대 2 case에서 값을 찾았다.

User 15에 대해 train, test data로 측정한 결과는 아래와 같다.

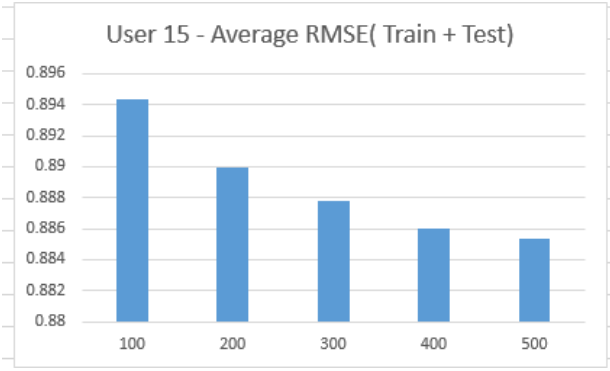


User 73에 대해 train, test data로 측정한 결과는 아래와 같다.



두 경우에 대해 overfitting이 되는 것을 방지하기 위해

Train과 test data의 RMSE 평균이 가장 작은 500을 선택했다. K = 500



실험을 위한 코드는 아래와 같다.

```
import csv
k_rmse = []
users = ['15','73']
for user in users:
    for K in range(100, 600, 100):
        opt_decomposistion = initialize_task2(df_train, K)
        prediction_mf2 = opt_decomposistion[int(user)]
        k_rsme.append(','.join([str(K), str(task3_RSME(int(user),'test'))]))

with open('rmse_opt_big.csv', 'w', newline='') as myfile:
    wr = csv.writer(myfile, quoting=csv.QUOTE_ALL)
    wr.writerow(k_rmse)
```

실제 모델 구현을 할 때는 K로 decomposed 된 matrix을 생성했다. (이후 구조는 2번과 동일 하기 때문에 생략)

```
# task 3
opt_decomposistion = build_svd(df_train, 500)
```

4 input의 RMSE 값

Train data set을 사용해서 rmse를 구하면 다음과 같다.

```
(user, movie) = (1, 31)
Task1 RMSE:  0.8798726475548319
Task2 RMSE :  0.13125659412006696
Task3 RMSE :  0.013771619367380136
(user, movie) = (2, 10)
Task1 RMSE:  0.8660385948896735
Task2 RMSE :  0.07132754044768257
Task3 RMSE :  0.016565766063275165
(user, movie) = (3, 1235)
Task1 RMSE:  0.7257598040213652
Task2 RMSE :  0.21896086676954538
Task3 RMSE :  0.02610635862743624
(user, movie) = (4, 10)
Task1 RMSE:  0.8994866686779577
Task2 RMSE :  0.009859304354412693
Task3 RMSE :  0.0020241192710730807
```