2조 12141617 정규호 12153549 최수원 12163314 노희준

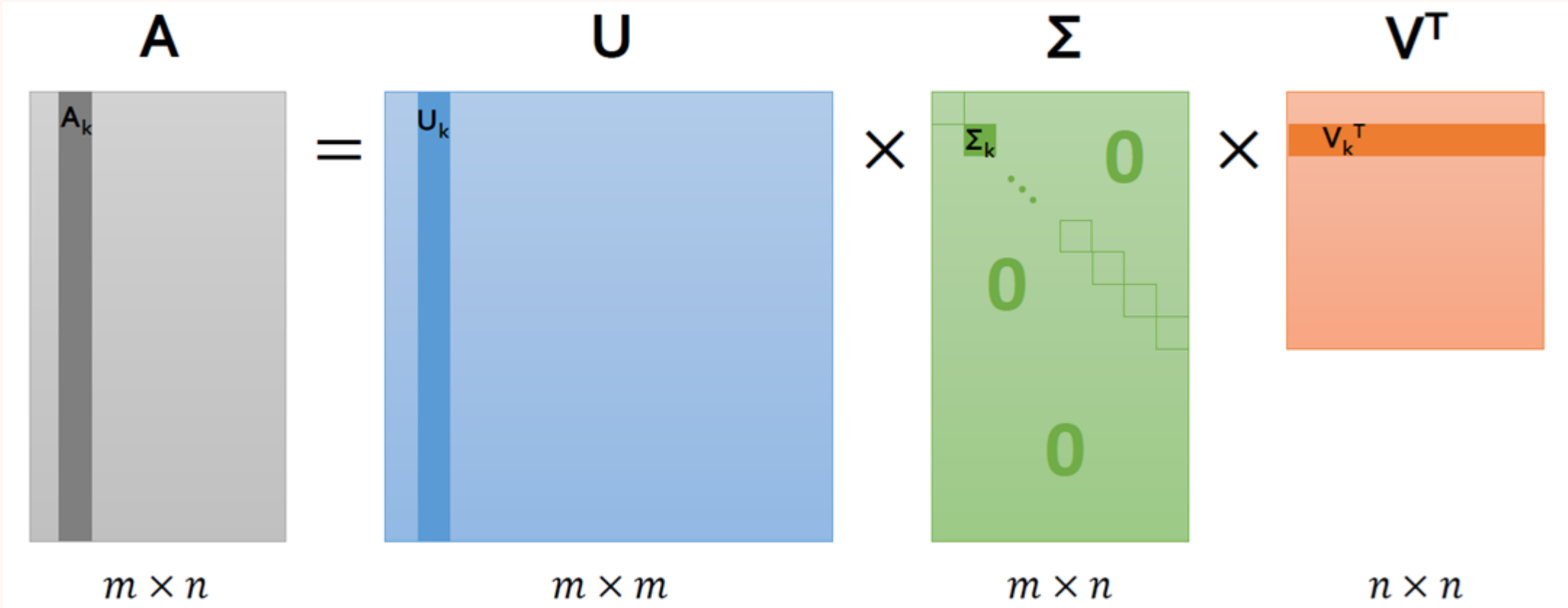
협업필터링을 통한 음식 추천 시스템

컴퓨터공학종합설계

목차

1. 협업 필터링 기반 추천 시스템 소개
2. scikit learn 라이브러리를 사용한 추천 시스템 코드 및 결과
3. scikit learn 라이브러리 성능 평가
4. surprise 라이브러리를 사용한 추천 시스템 코드 및 결과
5. svd 기반 추천 시스템 성능 평가
6. svd 기반 추천 시스템 성능평가 - 영화 데이터

협업 필터링 기반 추천 시스템 소개



SVD (Singular Value Decomposition)

SVD (특이값 분해) 는 Latent Factor Models 중에 하나로 SVD (Singular Value Decomposition) 는 Matrix Factorization 문제를 푸는 방법 중 하나이다.

<scikit learn 라이브러리>

- scikit learn 라이브러리에서 제공해주는 truncateSVD는 시그마 행렬의 대각원소(특이값) 가운데 상위 n개만 골라내준다. 즉 예상 선호도가 높은 순서대로 음식을 출력할 수 있다.

<surprise 라이브러리>

-surprise 라이브러리의 SVD와 accuracy를 활용하여 모델을 만들고 모델에 대한 정확도를 평가 할 수 있다.

scikit learn 라이브러리를 사용한 추천 시스템 코드 및 결과

```python
sigma = np.diag(sigma)
# 대칭 행렬로 변환

svd_user_predicted_ratings = np.dot(np.dot(U, sigma), Vt) + user_ratings_mean.reshape(-1,1)

svd_preds = pd.DataFrame(matrix_user_mean, columns = user_food_ratings.columns)



#함수 제작
def recommend_foods(svd_preds, user_id, ori_foods, ori_ratings, num_recommendations = 5):
    user_row_number = user_id-1
    sorted_user_predictions = svd_preds.iloc[user_row_number].sort_values(ascending=False)
    user_data = ori_ratings[ori_ratings.userId == user_id]
    user_history = user_data.merge(ori_foods, on = 'foodId').sort_values(['rating'], ascending = False)
    recommendations = ori_foods[~ori_foods['foodId'].isin(user_history['foodId'])]
    recommendations = recommendations.merge(pd.DataFrame(sorted_user_predictions).reset_index(), on = 'foodId')
    recommendations = recommendations.rename(columns = {user_row_number: 'Predictions'}).sort_values('Predictions', ascending = False).iloc[:num_recommendations, :]

    return user_history, recommendations

already_rated, predictions = recommend_foods(svd_preds, 77, foods, ratings, 10)
```

주요 코드

	foodId	userId	rating	name
3	6	7	5	한식 [삼계탕]
26	42	7	5	일식 [생선회]
6	9	7	5	한식 [삼겹살]
33	54	7	5	서양식 / 동남아시아 음식 [햄버거]
10	16	7	5	한식 [매운탕]
19	29	7	5	한식 [간장게장]
1	3	7	5	한식 [김치찌개]
12	18	7	4	한식 [순대국밥]
28	45	7	4	일식 [라면]
2	4	7	4	한식 [된장찌개]

	foodId	name	Predictions
8	1	한식 [비빔밥]	-1.833333
12	35	중식 [양꼬치]	-1.833333
21	57	서양식 / 동남아시아 음식 [타코]	-1.833333
28	56	서양식 / 동남아시아 음식 [부리또]	-1.833333
19	53	서양식 / 동남아시아 음식 [피자]	-1.833333
18	51	서양식 / 동남아시아 음식 [치킨]	-1.833333
17	49	일식 [스키야키]	-1.833333
16	46	일식 [메밀소바]	-1.833333
15	43	일식 [초밥]	-1.833333
14	39	중식 [훠궈]	-1.833333

<7번 사용자>

무작위로 23개의 음식을 제거하였고 선호도를 예측한 후에 선호도가 높은 순서대로 10개의 음식을 출력하였습니다. 그리고 실제 선호도 평균은 3.2가 나왔습니다.

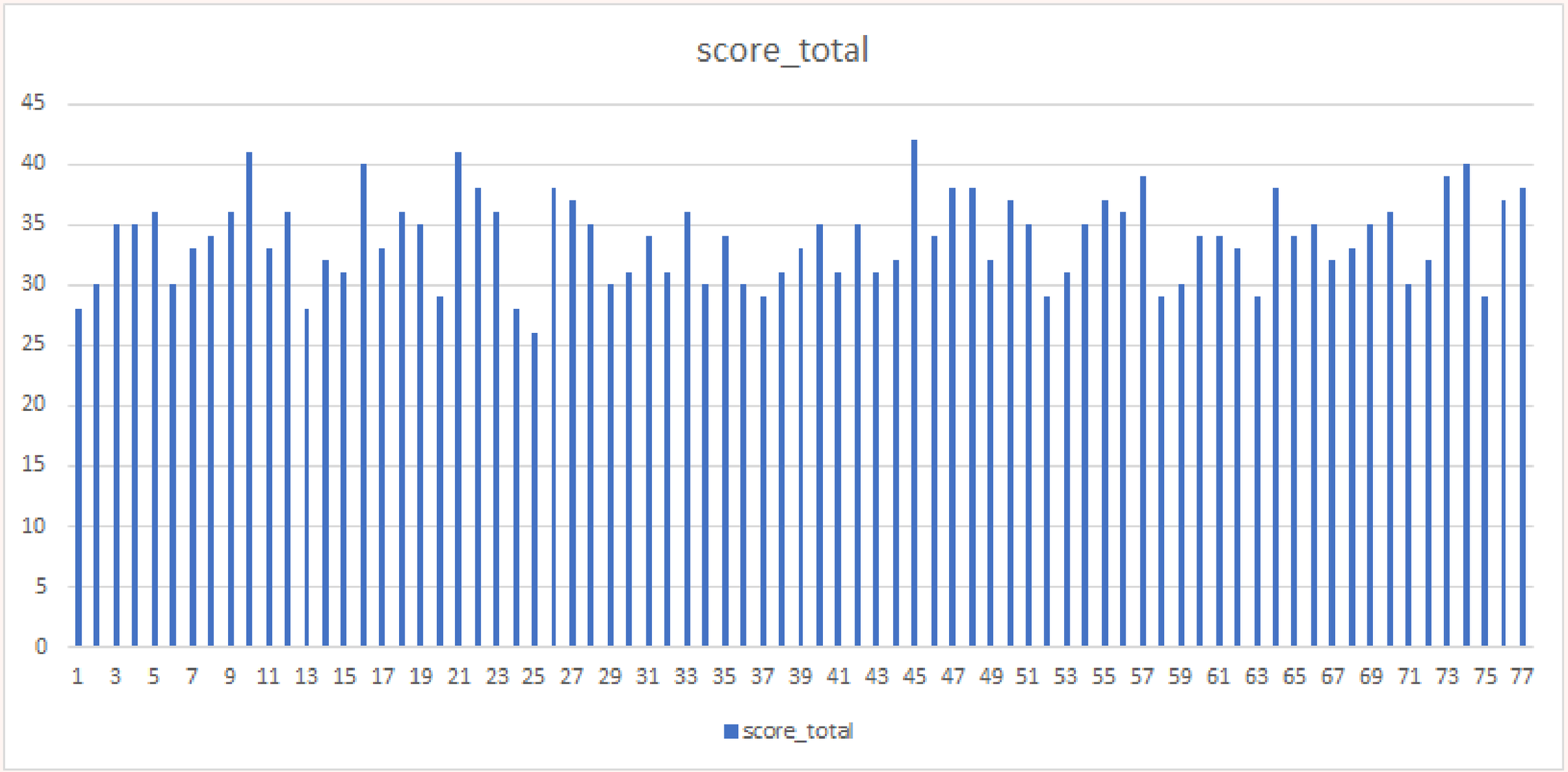
	foodId	userId	rating	name
0	31	10	5	중식 [짜장면]
1	32	10	5	중식 [짬뽕]
28	59	10	5	서양식 / 동남아시아 음식 [쌀국수]
27	58	10	5	서양식 / 동남아시아 음식 [리조또]
26	57	10	5	서양식 / 동남아시아 음식 [타코]
25	56	10	5	서양식 / 동남아시아 음식 [부리또]
24	55	10	5	서양식 / 동남아시아 음식 [스테이크]
23	54	10	5	서양식 / 동남아시아 음식 [햄버거]
22	53	10	5	서양식 / 동남아시아 음식 [피자]
21	52	10	5	서양식 / 동남아시아 음식 [스파게티]

	foodId	name	Predictions
0	1	한식 [비빔밥]	-2.416667
1	2	한식 [볶음밥]	-2.416667
28	29	한식 [간장게장]	-2.416667
27	28	한식 [샤브샤브]	-2.416667
26	27	한식 [물회]	-2.416667
25	26	한식 [소머리 국밥]	-2.416667
24	25	한식 [나주곰탕]	-2.416667
23	24	한식 [찜닭]	-2.416667
22	23	한식 [육개장]	-2.416667
21	22	한식 [낙지볶음]	-2.416667

<10번 사용자>

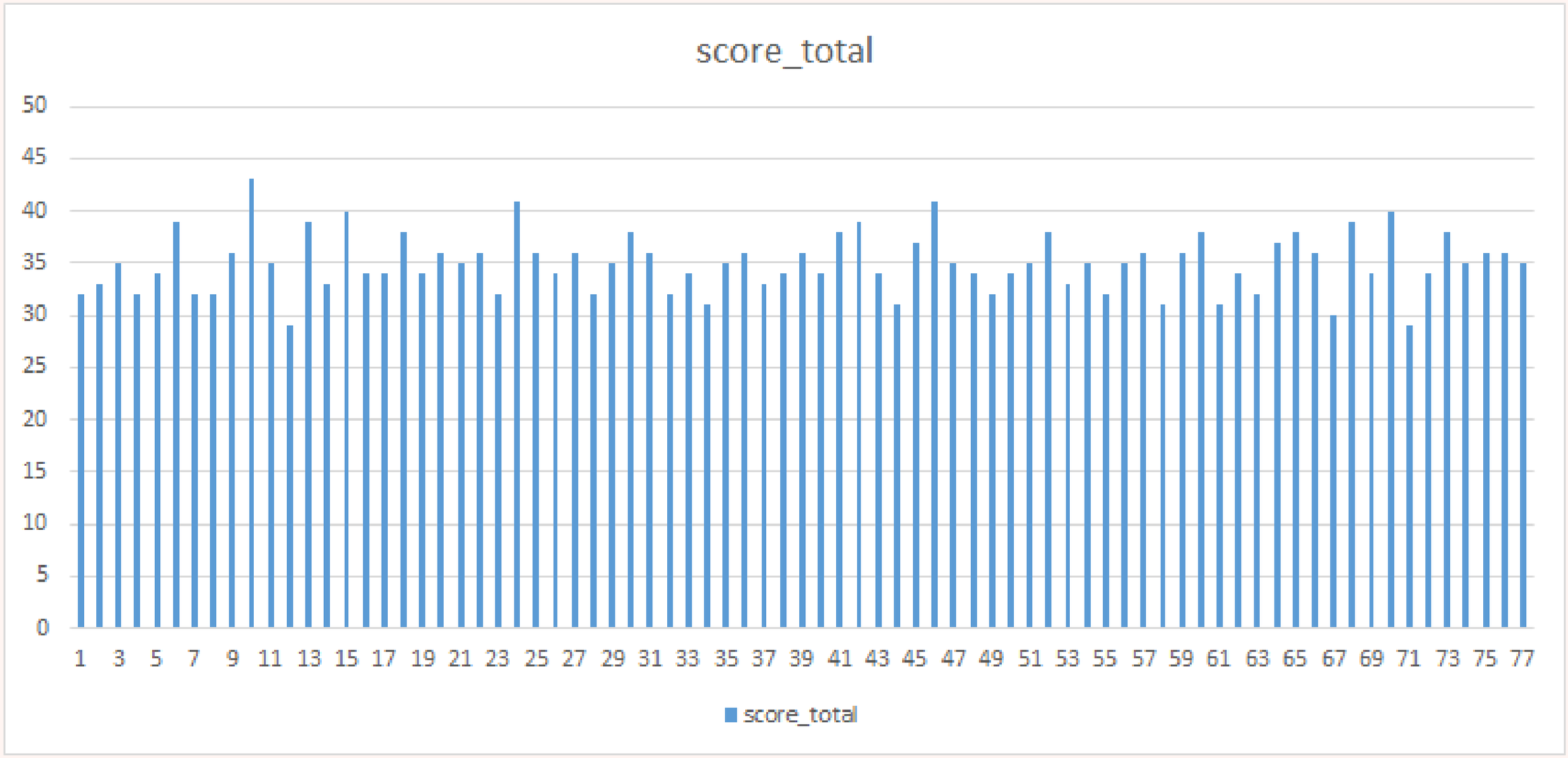
모든 한식(30개)을 제거하였고 선호도를 예측한 후에 선호도가 높은 순서대로 10개의 음식을 출력하였습니다. 그리고 실제 선호도 평균은 4.2가 나왔습니다.

scikit learn 라이브러리 성능 평가



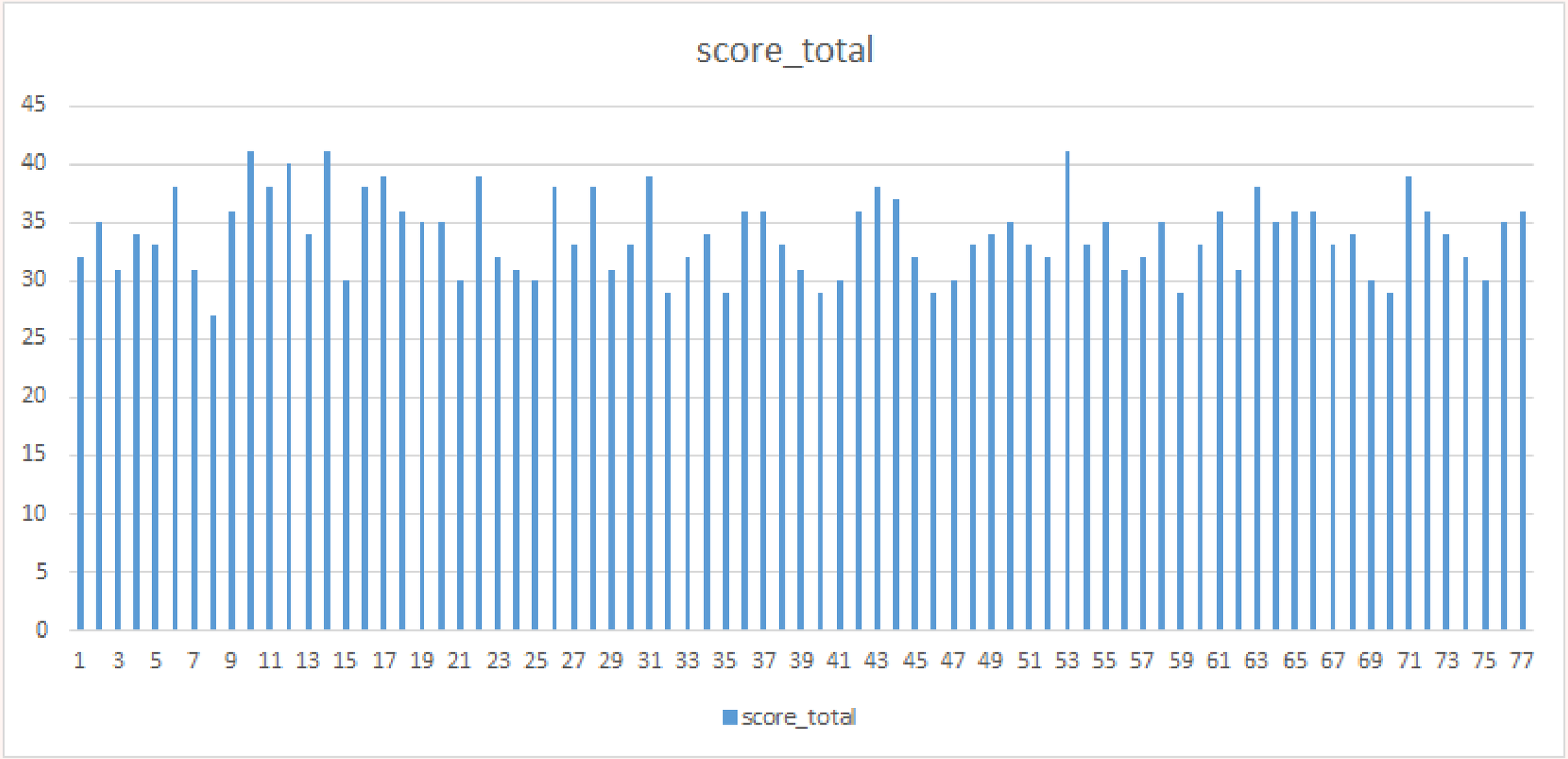
60개의 음식 목록중 20개를 제거

평균 : 3.38



60개의 음식 목록중 30개를 제거

평균 3.50



60개의 음식 목록중 40개를 제거

평균 3.40

surprise 라이브러리를 사용한 추천 시스템 코드 및 결과

```python
tab = pd.crosstab(rating['userID'],rating['food'])
# print(tab) # userID를 행, food를 열로 표시함

# 음식 rating
# 두 개의 집단변수를 가지고 나머지 rating을 그룹화 한다.
rating_g = rating.groupby(['userID','food'])
rating_g.sum()
# print(rating_g.sum())
tab = rating_g.sum().unstack() # 행렬구조로 변환 (userID를 행, food를 열로 표시함)
# print(tab)

# 2. rating 데이터셋 생성
reader = Reader(rating_scale=(1,5)) # 평점 범위
data = Dataset.load_from_df(df=rating,reader=reader)
# rating이라는 데이터 프레임은 reader (1~5)의 평점 범위를 가진다.
# print(data)

# 3. train/test set
train = data.build_full_trainset() # 훈련 셋
test = train.build_testset() # 검정 셋

# 4. model 생성
model = SVD(n_factors=100, n_epochs=300,random_state=123) # 몇 개의 latent factor 로 요인 벡터를 만들지를 정함
                                                          # 20 3.79 3.52 3.52
model.fit(train)

# 5. id가 6인 사용자 치킨 예상 rating
user_id = '37' # 추천 대상자
item_ids = ['볶음밥','양꼬치','메밀소바','스테이크','쌀국수']
actual_rating = 0

for item_id in item_ids:
    print(model.predict(user_id,item_id,actual_rating))
```

주요코드

```
user: 6          item: 비빔밥        r_ui = 0.00   est = 3.04   {'was_impossible': False}
user: 6          item: 순두부찌개     r_ui = 0.00   est = 3.52   {'was_impossible': False}
user: 6          item: 마라탕        r_ui = 0.00   est = 2.93   {'was_impossible': False}
user: 6          item: 규카츠        r_ui = 0.00   est = 3.51   {'was_impossible': False}
user: 6          item: 스파게티      r_ui = 0.00   est = 3.60   {'was_impossible': False}
```

6번 사용자의 다음의 음식들에 대한 선호도 예측

	실제 선호도	예측 선호도
비빔밥	2	3.04
순두부찌개	4	3.52
마라탕	2	2.93
규카츠	3	3.51
스파게티	3	3.60

```
user: 77        item: 비빔밥        r_ui = 0.00   est = 3.01   {'was_impossible': False}
user: 77        item: 순두부찌개     r_ui = 0.00   est = 3.53   {'was_impossible': False}
user: 77        item: 마라탕        r_ui = 0.00   est = 2.92   {'was_impossible': False}
user: 77        item: 규카츠        r_ui = 0.00   est = 3.50   {'was_impossible': False}
user: 77        item: 스파게티      r_ui = 0.00   est = 3.59   {'was_impossible': False}
```

77번 사용자의 다음의 음식들에 대한 선호도 예측

	실제 선호도	예측 선호도
비빔밥	5	3.01
순두부찌개	5	3.53
마라탕	2	2.92
규카츠	5	3.50
스파게티	5	3.59

svd 기반 추천 시스템 성능 평가

$$\text{평균제곱근오차}(RMSE) = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\text{실제값} - \text{예측값})^2}$$

Algorithm	test_rmse	fit_time	test_time
SVD	1.179707	0.209771	0.010982
SVDpp	1.180873	2.883951	0.092432
KNNWithZScore	1.184204	0.009625	0.174212
KNNWithMeans	1.187253	0.007645	0.164893
KNNBaseline	1.189876	0.011613	0.276940
BaselineOnly	1.201390	0.004632	0.007649
SlopeOne	1.202429	0.005982	0.067820
NMF	1.222547	0.580781	0.030251
CoClustering	1.223205	0.088450	0.010950
KNNBasic	1.249348	0.005320	0.140957
NormalPredictor	1.885489	0.012300	0.028257

-라이브러리 안의 알고리즘 성능들을 평가한 결과 , SVD 의 성능이 가장 좋게 나왔습니다.

rmse=1.17907

-오차 값이 적을 수록 좋은 성능을 가짐

```
algo = SVD()
algo.fit(train_set)

train_svd = algo.test(train_set.build_testset())
rmse = accuracy.rmse(train_svd)
mae = accuracy.mae(train_svd)

test_svd = algo.test(test_set)
rmse = accuracy.rmse(test_svd)
mae = accuracy.mae(test_svd)
```

```
RMSE: 2.9222
MAE:  2.1760
RMSE: 2.9720
MAE:  2.3967
```

29개의 음식에 대한 152명의 데이터를 기준으로한 성능평가

```
algo = SVD()
algo.fit(train_set)

train_svd = algo.test(train_set.build_testset())
rmse = accuracy.rmse(train_svd)
mae = accuracy.mae(train_svd)

test_svd = algo.test(test_set)
rmse = accuracy.rmse(test_svd)
mae = accuracy.mae(test_svd)
```

```
RMSE: 0.8196
MAE:  0.6719
RMSE: 1.1153
MAE:  0.9221
```

60개의 음식에 대한 77명의 데이터를 기준으로한 성능평가

svd 기반 추천 시스템 - 영화데이터

In [4]:
```
ratings = pd.read_csv('C:/Users/희준/Desktop/ratings.csv', usecols = ['userId', 'movieId', 'rating'])
reader = Reader(rating_scale=(1, 5))

ratings = Dataset.load_from_df(ratings, reader)

raw_ratings = ratings.raw_ratings

random.Random(10).shuffle(raw_ratings)

threshold = int(.9 * len(raw_ratings))
A_raw_ratings = raw_ratings[:threshold]
B_raw_ratings = raw_ratings[threshold:]

ratings.raw_ratings = A_raw_ratings

train_set = ratings.build_full_trainset()
test_set = ratings.construct_testset(B_raw_ratings)
```

In [5]:
```
algo = SVD()
cross_val = cross_validate(algo, ratings, measures=['RMSE', 'MAE'], cv=5, verbose=True)
```

```
Evaluating RMSE, MAE of algorithm SVD on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    0.9005  0.8987  0.8981  0.9020  0.9011  0.9001  0.0015
MAE (testset)     0.6930  0.6909  0.6915  0.6955  0.6950  0.6932  0.0018
Fit time          6.19    5.78    5.88    5.88    5.64    5.87    0.18
Test time         0.29    0.15    0.15    0.20    0.14    0.19    0.06
```

In [6]:
```
algo = SVD()
algo.fit(train_set)

train_svd = algo.test(train_set.build_testset())
rmse = accuracy.rmse(train_svd)
mae = accuracy.mae(train_svd)

test_svd = algo.test(test_set)
rmse = accuracy.rmse(test_svd)
mae = accuracy.mae(test_svd)
```

```
RMSE: 0.6392
MAE:  0.4959
RMSE: 0.8982
MAE:  0.6895
```

영화 데이터로 측정해본

추천 시스템의 성능

RMSE : 0.6392

MAE : 0.4959

감사합니다