소비자 특성에 따른 맥주 추천 알고리즘

통계학 세미나 기말 최종 발표

최수연
응용통계학과 17

강진희
응용통계학과 16

이승윤
응용통계학과 16

김도현
경제학과 13

PROCESS
00

목차

중간 발표 이후 피드백 및 시각화

주제, 사용 데이터, 모델 및 성능 평가에 대한 변경사항 보고

맥주시장 현황 시각화 with Pandas

Part 1. 머신러닝

Data Preprocessing

Model – KNN / NMF / SVD

Model Evaluation

Prediction

Surprise 패키지 설명

Part 2. 딥러닝

Preprocessing & Library

4 types of Model

Model Evaluation

Prediction

결론

머신러닝 vs 딥러닝 성능 비교

Best Model Selection

중간고사 이후
피드백 및 시각화

PROCESS
01

중간 발표 이후 피드백

목표: 맥주 추천 알고리즘 구현

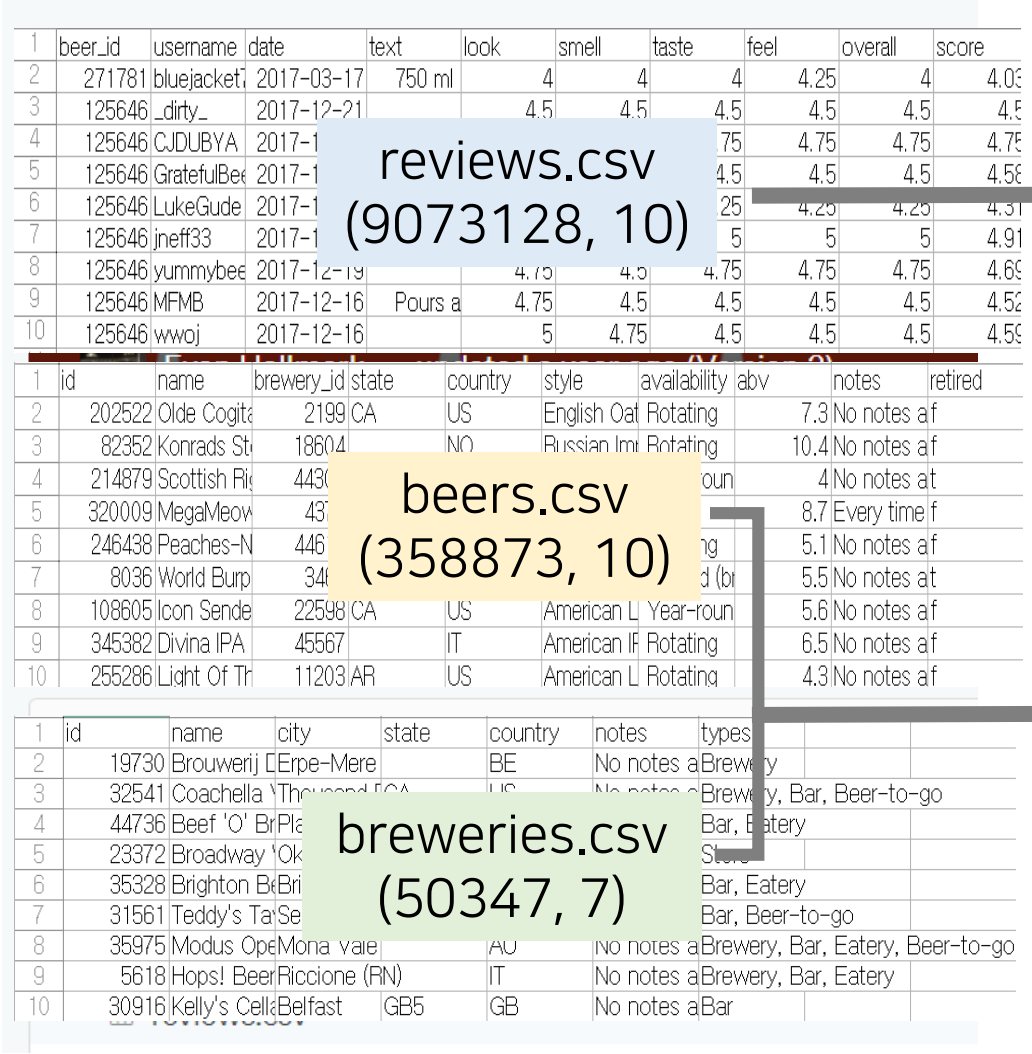
추천 시스템(recommender system)

사용자가 선호하는 상품을 예측하는 시스템

PROCESS 01

중간 발표 이후 피드백

사용 데이터와 사용 방향



reviews.csv
(9073128, 10)

beers.csv
(358873, 10)

breweries.csv
(50347, 7)

맥주 추천 알고리즘

pandas numpy matplotlib seaborn
sklearn keras surprise

맥주 시장현황 시각화

pandas numpy matplotlib

PROCESS 01

중간 발표 이후 피드백

모델 및 성능 평가에 대한 변경사항 보고

중간발표

데이터 전처리

reviews.csv의 900만개 데이터

User: 평점을 매긴 맥주가 100개 이상인 user기준

=>400만개로 축소

사용 분석 기법

UBCF, KNN, SVD, PCA 등 머신러닝 기법 위주

성능 평가 방법

K-fold / 평가지표(rmse, accuracy/precision/recall)

피드백 반영 이후

데이터 추가 전처리

Beer(Item): 평점을 매긴 user가 1000명 이상인 beer기준

User: 평점을 매긴 맥주가 250개 이상인 user기준

=> 10만개로 축소

딥러닝 추가 -> 머신러닝 / 딥러닝으로 이분화

머신러닝: SVD, KNN, NMF / 딥러닝: Neural Network

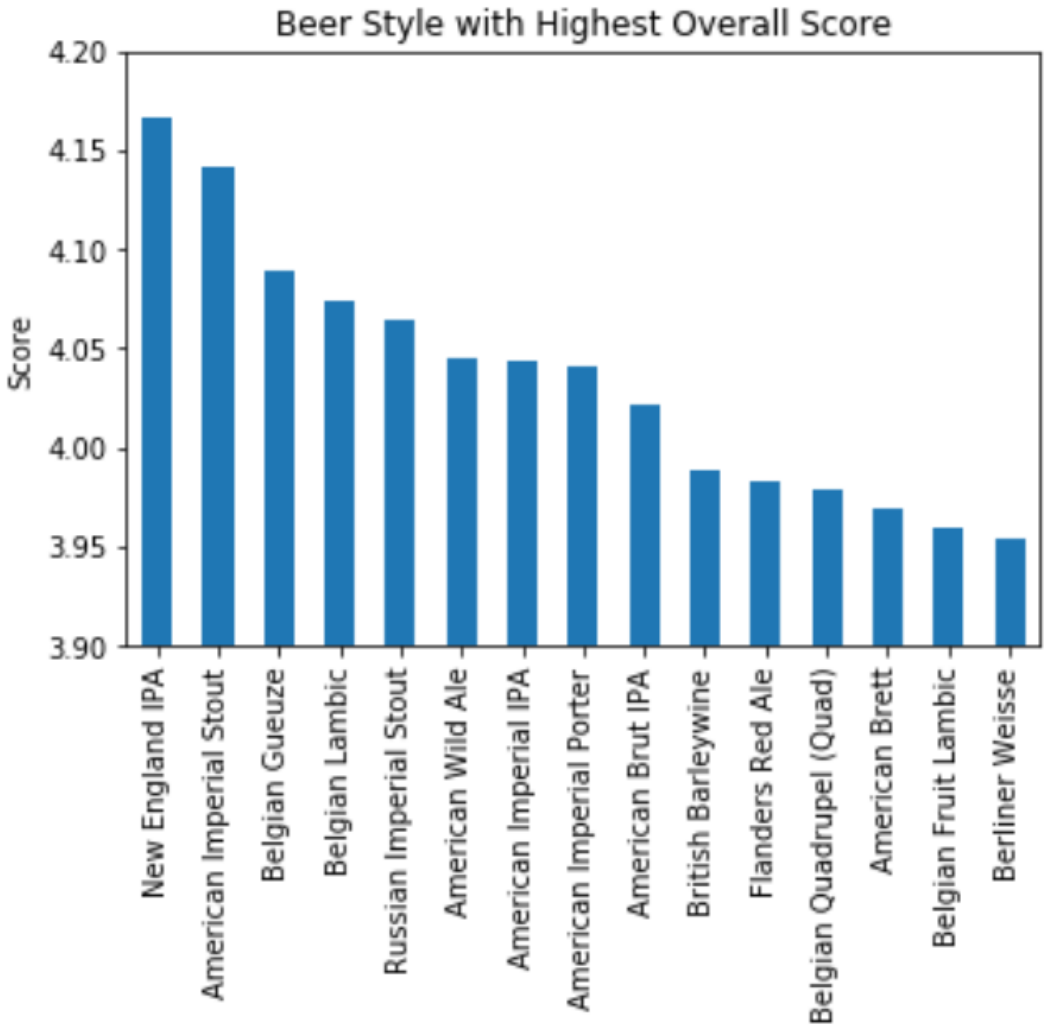
Regression에 대한 평가 지표의 변화

K-fold / GridSearchCV / 평가지표(rmse, mae)

PROCESS 01

시각화

맥주 시장 현황에 대한 시각화 with Pandas



User별 각 style에 대한 score의 평균으로 순위산정

1위 : New England IPA
2위 : Americal Imperial Stout
3위 : Belgian Gueuze
4위 : Belgian Lambic
5위 : Russian Imperial Stoutnp

머신러닝

최근접 이웃
IBCF

잠재요인
NMF / SVD

surprise 패키지
SVD

PROCESS 02

Part 1. 머신러닝

Algorithm Concept

- 콘텐츠 기반 필터링 (Contents-based filtering)
- 협업 필터링 (Collaborative filtering)
  - 최근접 이웃 방식 (Nearest Neighborhood)
    - UBCF (User-based collaborative filtering)
    - IBCF (Item-based collaborative filtering)
  - 잠재요인 방식 (Latent Factor)
    - SVD (Singular value decomposition)
    - NMF (Nonnegative matrix factorization)
- 베이스라인 모형 (Baseline)

PROCESS 02

Part 1. 머신러닝 – 데이터 전처리

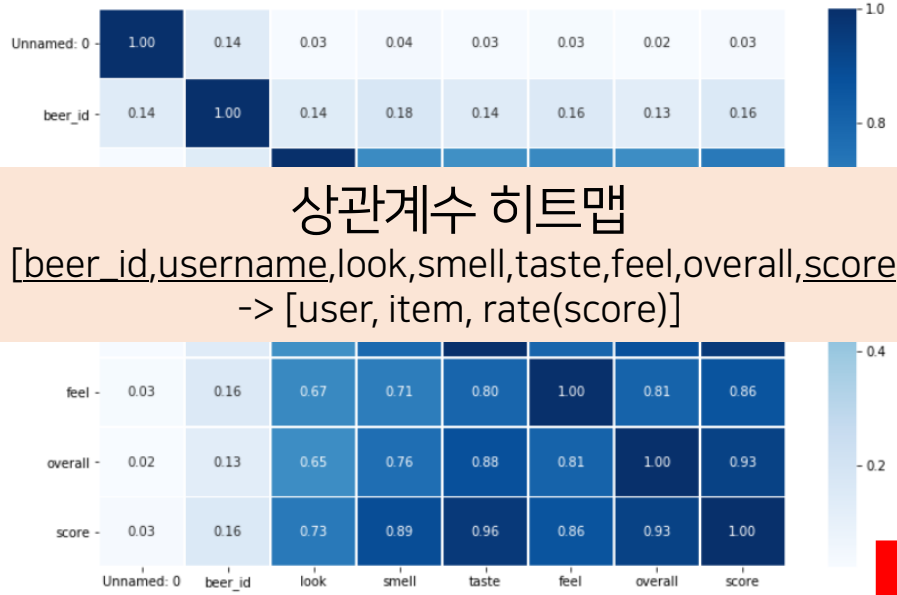
Data Preprocessing

중간발표 데이터
400만 row X 10 col

beer_id	username	look	sme	aste	feel	overall	score
271781	bluejacket74	4.00	4.0	4.00	4.25	4.00	4.03
90017	bluejacket74	4.00	4.25	4.25	4.00	4.25	4.21
47694	bluejacket74	3.50	3.50	3.50	3.50	3.50	3.50
270612	bluejacket74	4.00	4.00	4.25	4.00	4.00	4.10
96731	bluejacket74	3.00	4.00	4.25	3.75	4.00	4.02

Beer
평점을 매긴 user가 1000명 이상인 제품만 포함

User
평점을 매긴 beer가 250개 이상인 사람만 포함



상관계수 히트맵
[beer_id,username,look,smell,taste,feel,overall,score]
-> [user, item, rate(score)]

In [4]:
```
from sklearn.preprocessing import LabelEncoder
user_enc = LabelEncoder()
df['userid'] = user_enc.fit_transform(df['user'].values)
df.head()
```

LabelEncoder
Object 형식의 username을 int형식으로 변환

1	66674	bluejacket74	4.04	190
2	48824	bluejacket74	3.71	190
3	55939	bluejacket74	4.10	190
4	68916	bluejacket74	4.34	190

	user	item	rate
0	196	242	3.0
1	186	302	3.0
2	22	377	1.0
3	244	51	2.0
4	166	346	1.0
5	298	474	4.0
6	115	265	2.0
7	253	465	5.0
8	305	451	3.0
9	6	86	3.0

10만 row X 3 col

머신러닝

최근접 이웃
IBCF

잠재요인
NMF / SVD

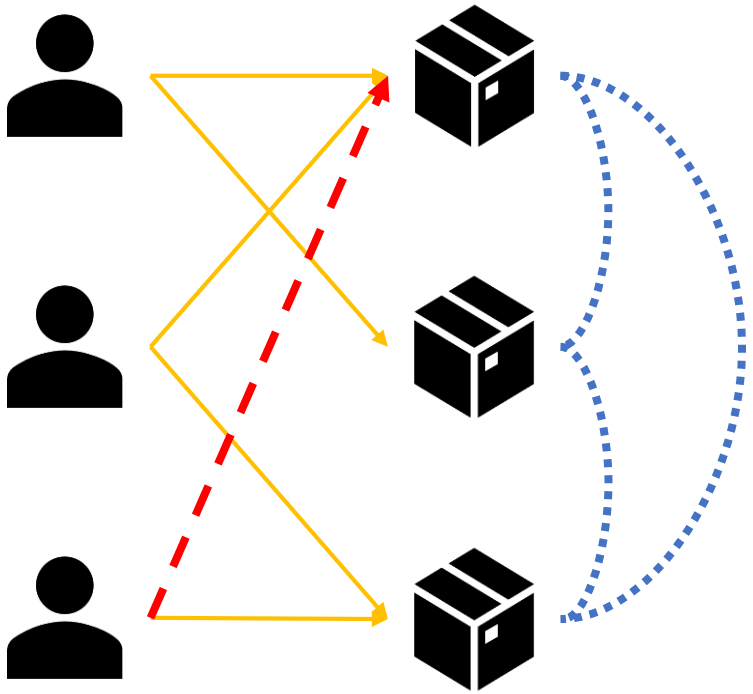
surprise 패키지
SVD

PROCESS
02

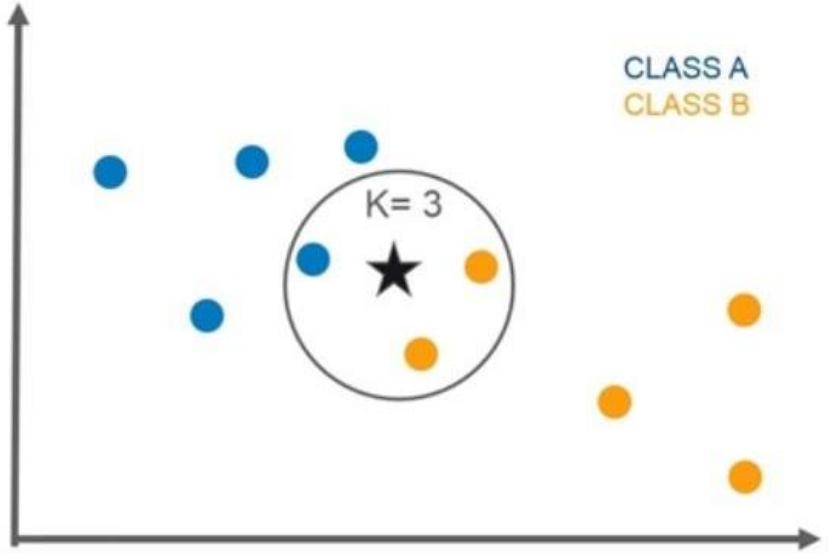
Part 1. 머신러닝 – IBCF

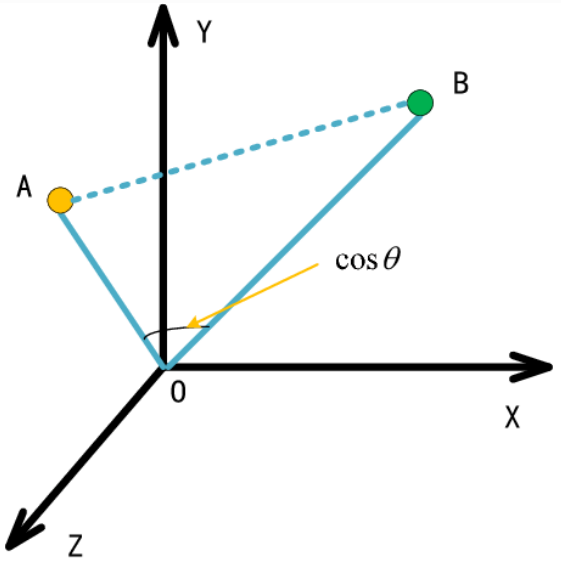
Model – KNN

IBCF(Item-based collaborative filtering)



KNN





코사인 유사도

PROCESS 02

Part 1. 머신러닝 – IBCF

Model – KNN

preprocessing

username과 beerID를 기준으로 rating정보만 담은 피벗 테이블 작성

```
ratings_matrix = data.pivot_table('rating', index='user', columns='beer') #ItemID를 기준으로 정렬
ratings_matrix.shape
```

```
(333, 481)
```

```
ratings_matrix.head()
```

실제 평점 데이터를 피벗 테이블로 변환

user	b											72	182256	
57md	3.69	NaN	NaN	NaN	3.63	NaN	NaN	NaN	4.00	2.18	...	3.98	NaN	3.94	3.94
ALC82	NaN	NaN	NaN	NaN	4.50	4.73	NaN	NaN	3.90	2.25	...	NaN	NaN	4.38	NaN
AgentMunky	4.07	3.95	3.16	3.88	4.30	NaN	4.15	NaN	3.46	1.72	...	NaN	4.24	NaN	NaN
Alieniloquium	NaN	4.07	NaN	3.88	4.12	3.95	3.41	NaN	NaN	NaN	...	3.44	NaN	4.12	NaN
Arbitrator	NaN	2.51	3.27	3.56	4.28	4.32	4.01	4.0	4.17	NaN	...	NaN	NaN	NaN	NaN

userID기준으로 작성된 rating matrix를 beerID기준으로 전치한다

```
ratings_matrix_T = ratings_matrix.transpose() #User의 평점이 아니라 맥주간의 유사도를 측정
ratings_matrix_T.head(3)
```

beerID기준으로 전치

user	57md	ALC82	A				3	BB1313	BEER88	BEE
beer										
6	3.69	0.0	4.07	0.00	0.00	0.0	0.00	3.6	0.0	
30	0.00	0.0	3.95	4.07	2.51	0.0	3.77	0.0	4.0	
31	0.00	0.0	3.16	0.00	3.27	0.0	0.00	0.0	0.0	

수리연산을 위해 NaN를 0으로 변환

```
# beerID는 나중에
# rating_beer = pd.merge(data, beers, on='beer_ID')

# columns='beer_ID' 로
#ratings_matrix = rati                                    er', columns='beer_ID')

# NaN 값을 모두 0 으로 변환
ratings_matrix = ratings_matrix.fillna(0)
ratings_matrix.head(3)
```

결측값을 0으로 대체

Item(beerID)간의 유사도 측정

- index가 user로 설정되어있던 ratings matrix를 itemID가 index가 되도록 변환

```
from sklearn.metrics.pairwise import cosine_similarity

# 코사인 유사도 matrix 작성
item_sim = cosine_similarity(ratings_matrix_T, ratings_matrix_T) #Item유사도 - corr.matrix처럼 n*n행렬

# cosine_similarity() 로 반환된 넘파이 행렬을 맥주ID로 매핑하여 DataFrame으로 변환
item_sim_df = pd.DataFrame(data=item_sim, index=ratings_matrix.columns,columns=ratings_matrix.columns).round(4)
item_sim_df.head()
```

아이템 간의 코사인 유사도 측정

beer	6	30										181572	182256	187317
beer														
6	1.0000	0.7117	0.6496	0.7137	0.7147	0.6112	0.6738	0.6923	0.7124	0.6083	...	0.4869	0.4740	0.4043	0.4582	0.4639
30	0.7117	1.0000	0.7582	0.9201	0.9364	0.8062	0.8288	0.7936	0.8541	0.7063	...	0.6231	0.5792	0.4821	0.6126	0.5958
31	0.6496	0.7582	1.0000	0.7624	0.7592	0.6532	0.6952	0.7148	0.7234	0.6334	...	0.4531	0.4725	0.3797	0.4495	0.4679
33	0.7137	0.9201	0.7624	1.0000	0.9179	0.7924	0.8332	0.8032	0.8376	0.7051	...	0.5939	0.5657	0.4816	0.5787	0.5799
34	0.7147	0.9364	0.7592	0.9179	1.0000	0.8495	0.8489	0.8123	0.8813	0.7525	...	0.6397	0.5919	0.5468	0.6199	0.6196

PROCESS 02

Part 1. 머신러닝 – IBCF

Model – KNN

유사도가 가장 높은 상위 n가지 추출하는 모듈 만들기

> ※주의: 자

corr_top(item,n)
아이템 ID(beer_id)를 받아 특정 beer에 대한 유사도가 가장 높은 상위 n개를 출력하는 함수

```
def corr_top(i
    ans = pd.DataFrame(item_sim_df[item].sort_values(ascending = False)[1:n])
    return ans
```

```
def predict_rating(ratings_arr, item_sim_arr ):
    ratings_pred = ratings_arr.dot(item_sim_arr)/ np.array([np.abs(item_sim_arr).sum(axis=1)])
    return ratings_pred
```

```
ratings_pred = predict_rating(ratings_matrix.values , item_sim_df.values)
ratings_pred_matrix = pd.DataFrame(data=ratings_pred, index= ratings_matrix.index,columns = ratings_matrix.columns)
ratings_pred_matrix.head()
```

predict_rating(rating_arr, item_sim_arr)
실제 평점데이터와 아이템 유사도를 계산한 데이터를 array형식으로 받아 예측평점을 데이터프레임 형식으로 반환하는 함수

beer													79482
user													
57md													48413
ALC82													29951
AgentMunky	2.563730	2.579625	2.580783	2.575660	2.572463	2.551211	2.568175	2.565421	2.568891	2.549082	...	2.515182	2.522537
Alieniloquium	2.025298	2.119282	2.035929	2.109260	2.137561	2.167387	2.133007	2.067375	2.096473	2.077914	...	2.202948	2.135681
Arbitrator	2.021488	2.059952	2.064185	2.061413	2.054135	2.061562	2.058637	2.055588	2.048894	2.025354	...	1.923869	1.913971

PROCESS 02

Part 1. 머신러닝 – IBCF

Model Evaluation

```
from sklearn.metrics import mean_squared_error

# 사용자가 평점을 부여한 액주에 대해서만 예측 성능 평가 RMSE 를 구함.
```

```
# 결과 확인하기
ratings_pred = predict_rating_topsim(ratings_matrix.values , item_sim_df.values, 20)
print('아이템 기반 인접 TOP-20 이웃 RMSE: ', get_rmse(ratings_pred, ratings_matrix.values )) # top 20개 예측한 matrix와


# 계산된 예측 평점 데이터는 DataFrame으로 재생성
ratings_pred_matrix = pd.DataFrame(data=ratings_pred, index= ratings_matrix.index,columns = ratings_matrix.columns)
```

```
C:\Users\genie\Anaconda3\lib\site-packages\ipykernel_launcher.py:10: FutureWarning: Using a non-tuple sequence for mul
q)]` instead of `arr[seq]`. In the future this will be interpreted as an array index, `arr[np.array(seq)]`, which will
  # Remove the CWD from sys.path while we load stuff.
C:\Users\genie\Anaconda3\lib\site-packages\ipykernel_launcher.py:11: FutureWarning: Using a non-tuple sequence for mul
q)]` instead of `arr[seq]`. In the future this will be interpreted as an array index, `arr[np.array(seq)]`, which will
  # This is added back by InteractiveShellApp.init_path()
아이템 기반 인접 TOP-20 이웃 RMSE:  0.6286827438667582
```

```
            pred[row, col] = item_sim_arr[col, :][top_n_items].dot(ratings_arr[row, :][top_n_items].T)
            pred[row, col] /= np.sum(np.abs(item_sim_arr[col, :][top_n_items])) ## 0~5 사이로 범위 맞추

    return pred
```

PROCESS 02

Part 1. 머신러닝 – IBCF

Prediction

```
def
    # 결과 확인하기
    recomm_beer = recomm_beer_by_userid(ratings_pred_matrix, 190, n=10)

    # 평점 데이터를 DataFrame으로 생성.
    recomm_beer = pd.DataFrame(data=recomm_beer.values,index=recomm_beer.index,columns=['pred_score'])
    recomm_beer
    get_unlist_beer(ratings_matrix, userId)
    all_beer_list = ratings_matrix.columns.tolist()

    # list com
    unlist_bee

    return unlist_beer
```

get_unlist_beer(ratings_matrix, userId)
특정 유저에 대하여 평가 정보가 없는 beer의 리스트를 출력하는 함수

190번 user가 평점을 매기지 않은 맥주 중
평점이 높을 것으로 예상되는 상위 10개 목록 추천

```
def recomm_beer_by_userid(pred_df, userId, n):
    # 예측 평점 DataFrame에서 사용자id index와 unlist_beer로
    pred_df.columns = pred_df.columns.astype(str)
    unlist_beer = get_unlist_beer(ratings_matrix, userId) #

    # l
    use
    unl

    # 최종
    recomm_beer = pred_df.loc[userId,unlist_beer].sort_valu

    return recomm_beer
```

추출하여 가장
평점을 매기지
은 평
n].round(2)

recom_beer_by_userid(pred_df, userId
예측 평점 데이터를 기반으로 특정 user가 평점을 매기지
점으로 예상되는 제품을 n개 추천하기

beer	pred_score
7971	4.20
57908	4.11
656	4.11
808	4.09
27804	4.08
73764	4.00
21950	3.97
24905	3.96
35738	3.95
16909	3.95

머신러닝

최근접 이웃
IBCF

잠재요인
NMF / SVD

surprise 패키지
SVD

PROCESS
02

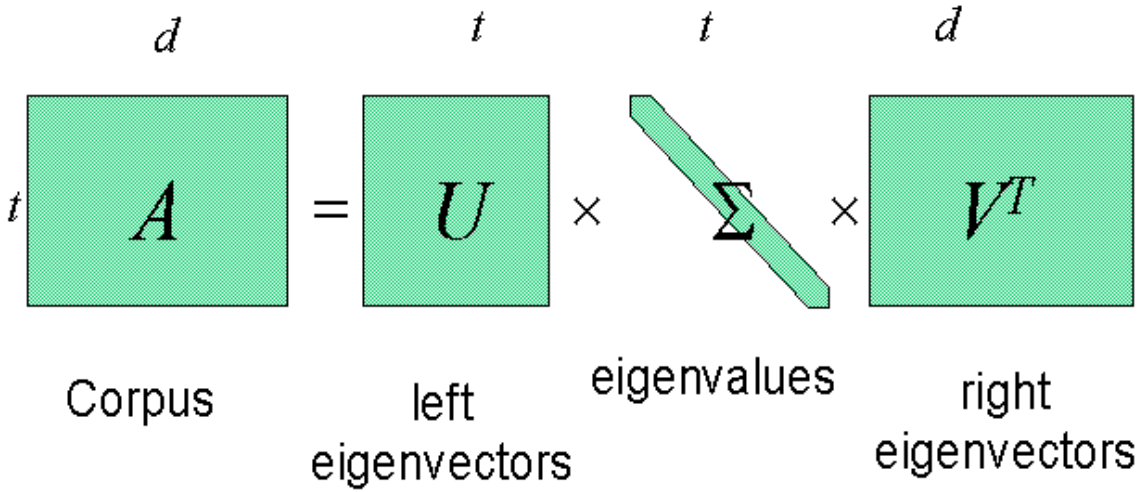
Part 1. 머신러닝 – 잠재요인 협업 필터링 SVD

SVD (Singular Value Decomposition)

SVD (Singular Value Decomposition) 는 Matrix Factorization 문제를 푸는 방법 중 하나이다.

$m \times n$ 크기의 행렬 R은 다음과 같이 세 행렬의 곱으로 나타낼 수 있다. 이를 특이치 분해(Singular Value Decomposition) 라고 한다.

$$R = U\Sigma V^T$$



이 식에서

- U 는 $m \times m$ 크기의 행렬로 역행렬이 대칭 행렬
- Σ 는 $m \times n$ 크기의 행렬로 비대각 성분이 0
- V 는 $n \times n$ 크기의 행렬로 역행렬이 대칭 행렬

Σ의 대각 성분은 특이치라고 하며 전체 특이치 중에서 가장 값이 큰 k개의 특이치만을 사용하(SVD), 다음과 같은 행렬을 만들수 있다.

- $\hat{U}$ 는 U에서 가장 값이 큰 k개의 특이치에 대응하는 k개의 성분만을 남긴 $m \times k$ 크기의 행
- $\hat{\Sigma}$ 는 가장 값이 큰 k개의 특이치에 대응하는 k개의 성분만을 남긴 $k \times k$ 크기의 대각 행렬
- $\hat{V}$ 는 V에서 가장 값이 큰 k개의 특이치에 대응하는 k개의 성분만을 남긴 $k \times n$ 크기의 행렬

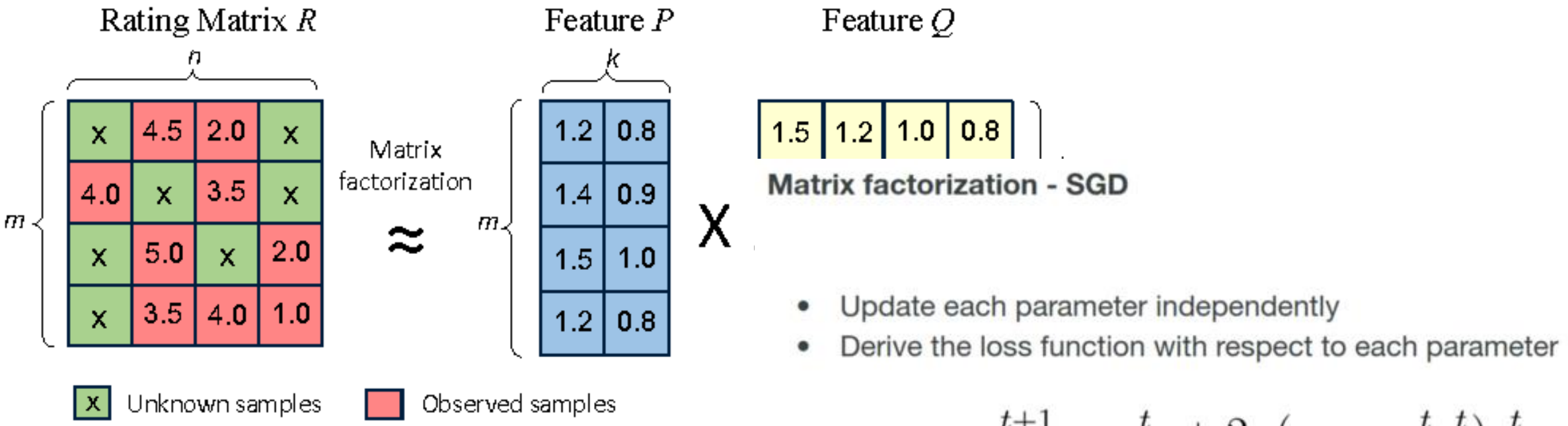
이 행렬을 다시 조합하면 원래의 행렬과 같은 크기를 가지고 유사한 원소를 가지는 행렬을 만들 수 있다.

$$\hat{U}\hat{\Sigma}\hat{V}^T = \hat{R} \approx R$$

PROCESS
02

Part 1. 머신러닝 – 잠재요인 협업 필터링 NMF

NMF(Nonnegative Matrix Factorization)



Matrix factorization - SGD

- Update each parameter independently
- Derive the loss function with respect to each parameter

Solutions:

$$p_{ki}^{t+1} = p_{ki}^{t} + 2\gamma(r_{ij} - p_i^t q_j^t)q_{kj}^t$$
$$q_{kj}^{t+1} = q_{kj}^{t} + 2\gamma(r_{ij} - p_i^t q_j^t)p_{ki}^t$$

PROCESS 02

Part 1. 머신러닝 – 잠재요인 협업 필터링 NMF

Model – NMF

4.2 Not Null인 위치추출해 MSE계산

```python
from sklearn.metrics import mean_squared_error

def get_rmse(R,P,Q,non_zeros):
    error=0

    #두개의 분해된 행렬 P와 Q.T의 내적으로 예측행
    full_

    #실제
    x_nor
    y_nor
    R_nor
    full_pred_matrix_non_zeros=full_pred_matrix[x
    mse=mean_squared_error(R_non_zeros,full_pred_
    rmse=np.sqrt(mse)

    return rmse
```

get_rmse(R,P,Q,non_z
null이 아닌 부분에 대하여
실제행렬 R과 예측 행렬간의 차

4.3 SGD를 이용한 Matrix Factorization

```python
def matrix_factorization(R,K,steps,learning_rate=0.01,r_lambda=0.01):
    num_users,num_items=R.shape

    #P와 Q매트릭스의 크기를 지정하고 정규분포를 가진 랜덤한 값으로 입력
    np.random.seed(1)
    P=np.random.normal(scale=1./K,size=(num_users,K))
    Q=np.random.normal(scale=1./K,size=(num_items,K))

    prev_rmse=10000
    break_count=0

    #R>0 인 행위치, 열위치, 값을 non_zeros 리스트에 저장
    non_zeros=[(i,j,R[i,j]) for i in range(num_users) for j in range(num_items) if R[i,j]>0]

    #
    f

            #Regularization을 반영한 SGD업데이트 공식 적용
            P[i,:]=P[i,:]+learning_rate*(eij*Q[j,:]-r_lambda*P[i,:])
            Q[j,:]=Q[j,:]+learning_rate*(eij*P[i,:]-r_lambda*Q[j,:])

        rmse=get_rmse(R,P,Q,non_zeros)
        if (step%10)==0:
            print("###iteration step: ",step,"rmse: ",rmse)

    return P,Q
```

Matrix_factorization(R,K,steps)
R을 P와 Q의 내적으로 분해하는 함수
확률적 경사하강법으로 실제행렬 R과 예측 행렬 간의 차이가 작아지도록 행렬을 업데이트

PROCESS 02

Part 1. 머신러닝 – 잠재요인 협업 필터링 NMF

Model – NMF

4.6 구현

```
ratings_matrix=ratings.pivot_table('rating',index='userid',columns='beerid')
rating_beers=pd.merge(ratings,beers,on='beerid')
ratings_matrix=rating_beers.pivot_table('rating',index='userid',columns='beerid')
ratings_matrix.head()
```

beerid	6	30	31	33	34	39	59	61	63	65	...	178857	179482	181572	182256	187317	189272	197183	202078	211516	22128

실제 평점 행렬 R

```
P,Q=matrix_factorization(ratings_matrix.values,K=50,steps=200,learning_rate=0.01,r_lambda=0.01)
pred_matrix=np.dot(P,Q.T)
```

```
###iteration step: 0 rmse: 2.378424781741225
###iteration step: 10 rmse: 0.3439810849529834
###iteration step: 20 rmse: 0.3245207219041075
###iteration step: 30 rmse: 0.300126573072935
###iteration step: 40 rmse: 0.2752329880293731
###iteration step: 50 rmse: 0.2542928478167513
###iteration step: 60 rmse: 0.23954231630562253
###iteration step: 70 rmse: 0.2301163081897057
###iteration step: 80 rmse: 0.22438221141758433
###iteration step: 90 rmse: 0.22091342270269693
###iteration step: 100 rmse: 0.2187158851810717
###iteration step: 110 rmse: 0.21721892456364225
###iteration step: 120 rmse: 0.2161278783180866
###iteration step: 130 rmse: 0.21529043465640743
###iteration step: 140 rmse: 0.2146230913009091
###iteration step: 150 rmse: 0.21407653396548232
###iteration step: 160 rmse: 0.21361955018989456
###iteration step: 170 rmse: 0.21323122397242067
###iteration step: 180 rmse: 0.21289689862080896
###iteration step: 190 rmse: 0.21260594127964733
```

SGD를 통해 rmse가 가장 낮은 행렬 P,Q.T찾기
->예측행렬

PROCESS 02

Part 1. 머신러닝 – 잠재요인 협업 필터링 NMF

[redacted] Model Evaluation

```
from surprise.model_selection import cross_validate
from surprise import NMF

reader=Reader(rating_scale=(1.0,5.0))
data=Dataset.load_from_df(ratings[['userid','beerid','rating']],reader)

algo=NMF(random_state=0)
cross_validate(algo,data,measures=['RMSE','MAE'],cv=5,verbose=True)
```

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

	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Mean	Std
RMSE (testset)	0.3562	0.3581	0.3539	0.3548	0.3555	0.3557	0.0014
MAE (testset)	0.2667	0.2662	0.2654	0.2650	0.2662	0.2659	0.0006
Fit time	13.51	13.55	12.87	12.86	12.43	12.94	0.51
Test time	0.27	0.22	0.22	0.39	0.27	0.27	0.06

```
{'test_rmse': array([0.35616075, 0.35806161, 0.35393645, 0.35479529, 0.35553672]
 'test_mae': array([0.2666897 , 0.26621431, 0.26535372, 0.26499436, 0.26617974]),
 'fit_time': (13.507419347763062,
  13.54579758644104,
  12.367648839950562,
  12.856643676757812,
  12.427874088287354),
 'test_time': (0.2671973705291748,
  0.22034955024719238,
  0.22249555587768555,
  0.3949429988861084,
  0.2682802677154541)}
```

```
#최적화 파라미터 지정
param_grid={'n_epochs':[20,40,60],'n_factors':[20,50,100]}

#CV를 3개 폴드 세트로 지정, 성능평가는 rmse,mae로 수행
gs=GridSearchCV(NMF,param_grid,measures=['rmse','mae'],cv=3)
gs.fit(data)

#최고 RMSE Evaluation점수와 그때의 하이퍼 파라미터
print(gs.best_score['rmse'])
print(gs.best_params['rmse'])
```

```
0.351807412355293 06
{'n_epochs': 60, 'n_factors': 20}
```

Part 1. 머신러닝 – 잠재요인 협업 필터링 NMF

Prediction

4.4 추천 함수 생성

```
def get_unuse_beers(ratings_matrix,userid):
    #userid로 입력받은 사용자의 모든 맥주정보를 추출해 Series로 반환
    #반환된 user_rating은 맥주명(name)을 인덱스로 가지는 Series
    user_rating=ratings_matrix.loc[userid,:]

    #user_rating이 0보다 크면 기존에 먹어본 맥주임. 대상 인덱스를 추출해 list로 만듦
    already_use=user_rating[user_rating>0].index.tolist()

    #모든 맥주를 list로 만듦
    beers_list=ratings_matrix.columns.tolist()

    #list comprehensio
    unuse_list=[beer f
    print('평점 매긴 맥
          '전체 맥주 수

    return unuse_list
```

get_unused_beers(ratings_matrix,userid)
행렬분해를 통해 구해진 예측행렬을 바탕으로 특정 user가 먹어보지 않은 맥주에 대한 예측평점을 상위 10개 계산하여 산출하는 함수

```
def recomm_beer_by_userid(pred_df,userid,unuse_list,top_n=10):
    #예측 평점 DataFrame에서 사용자 id 인덱스와 unseen_list로 들어온 맥주명 칼럼을 추출해
    #가장 예측 평점 높은순으로 정렬
    recomm_beers=pred_df.loc[userid,unuse_list].sort_values(ascending=False)[:top_n]

    return recomm_beers
```

PROCESS 02

Part 1. 머신러닝 – 잠재요인 협업 필터링 NMF

Prediction

```
unuse_list=get_unuse_beers(ratings_matrix,'bluejacket74')

recomm_beers=recomm_beer_by_userid(ratings_pred_matrix,'bluejacket74',unuse_list,top_n=10)

recomm_beers=pd.DataFrame(data=recomm_beers.values,index=recomm_beers.index,columns=['pred_score'])
pd.merge(recomm_beers,beers,on='beerid',how='left')[['name','beerid','pred_score']]
```

평점 매긴 맥주 수: 317 추천 대상 맥주 수: 164 전체 맥주 수: 481

	name	beerid	pred_score
0	Wisconsin Belgian Red	1577	4.796646
1	Parabola	41815	4.745755
2	Hunahpu's Imperial Stout	47022	4.720099
3	Cantillon Fou' Foune	5281	4.704758
4	Bourbon County Brand Barleywine Ale	100421	4.702571
5	King Sue	113674	4.687802
6	Pliny The Elder	7971	4.624046
7	Haze	125646	4.618017
8	Bourbon County Brand Coffee Stout	57747	4.616881
9	Vanilla Rye Bourbon County Brand Stout	131782	4.613687

'bluejacket74(=190번째 user)' User가
평점을 매기지 않은 맥주 중 평점이 높을 것으로 예상되는
상위 10개 목록

머신러닝

최근접 이웃
IBCF

잠재요인
NMF / SVD

surprise 패키지
SVD

Part 1. 머신러닝 – surprise 패키지

Accuracy 성능평가

추천성능 평가기준

`accuracy` 서브패키지에서는 다음과 같은 추천성능 평가기준을 제공한다. 이 식에서 $\hat{R}$은 테스트 데이터셋을 뜻한다.

- RMSE (Root Mean Squared Error)

$$\text{RMSE} = \sqrt{\frac{1}{|\hat{R}|} \sum_{\hat{r}(u,i) \in \hat{R}} (r(u,i) - \hat{r}(u,i))^2}$$

- MAE (Mean Absolute Error)

$$\text{MAE} = \frac{1}{|\hat{R}|} \sum_{\hat{r}(u,i) \in \hat{R}} |r(u,i) - \hat{r}(u,i)|$$

- FCP (Fraction of Concordant Pairs)

$$\text{FCP} = \frac{\text{number of concordant pairs}}{\text{number of discordant pairs}}$$

SURPRISE

A Python library for recommender systems

(Or rather: a Python library for rating prediction algorithms)

PROCESS 02

Part 1. 머신러닝 – surprise 패키지

Surprise 모델 제작 방식

모델 제작 순서

- 데이터셋의 split, folds 메서드를 사용해 k-fold 트레이닝 데이터셋과 테스트 데이터셋 만듦
- 모형 알고리즘 객체 생성
- training 데이터셋으로 모수를 추정한 후, test 메서드로 테스트 데이터셋에 대한 예측 실시
- accuracy 계산
- ★(evalueate 로 한번에도 가능)

```
!pip install surprise
import surprise
```

1. 데이터셋 준비
2. 칼럼 순서는 항상 [user_id,beer_id,rating]순으로 지정
3. Train / Test 분할

```
from surprise import Reader, Dataset
from surprise.model_selection import train_test_split

reader = Reader(line_format='user item rating', sep=',', rating_scale=(0.5,5))
#데이터의 칼럼 순서는 uid, iid, rating 순서를 반드시 지켜야 함
data = Dataset.load_from_df(df[['userid','beer','rating']],reader)

trainset, testset = train_test_split(data, test_size=.25, random_state=0) #train, test set 분할
```

PROCESS 02

Part 1. 머신러닝 – surprise 패키지

베이스라인 모형

```
bsl_options = {
    'method': 'als',
    'n_epochs': 5,
    'reg_u': 12,
    'reg_i': 5
}
algo = surprise.BaselineOnly(bsl_options)

np.random.seed(0)
acc = np.zeros(3)
cv = KFold(3)
for i, (trainset, testset) in enumerate(cv.split(data)):
    algo.fit(trainset)
    predictions = algo.test(testset)
    acc[i] = surprise.accuracy.rmse(predictions, verbose=True)
acc.mean()
```

```
Estimating biases using als...
RMSE: 0.3396
Estimating biases using als...
RMSE: 0.3333
Estimating biases using als...
RMSE: 0.3370

0.3366439478906325
```

베이스라인 모형

베이스라인 모형(baseline model)은 사용자 아이디 u, 상품 아이디 i, 두 개의 카테고리 값 입력에서 평점 $r(u,i)$의 예측치 $\hat{r}(u,i)$ 을 예측하는 가장 단순한 회귀분석모형으로 다음과 같이 사용자와 상품 특성에 의한 평균 평점의 합으로 나타난다.

$$\hat{r}(u,i) = \mu + b(u) + b(i)$$

```
#위의 코드 한번에
from surprise.model_selection import cross_validate
cross_validate(algo, data)
```

```
Estimating biases using als...
Estimating biases using als...
Estimating biases using als...
Estimating biases using als...
Estimating biases using als...

{'test_rmse': array([0.33391078, 0.33564717, 0.33871842, 0.33571915, 0.33760788]),
 'test_mae': array([0.24006561, 0.23816496, 0.24181065, 0.24070886, 0.2412288 ]),
```

PROCESS 02

Part 1. 머신러닝 – surprise 패키지

KNN – 최근접 이웃방식

surprise 패키지에서는 다음과 같은 유사도 기준을 제공한다.

- 평균제곱차이 유사도 (Mean Squared Difference Similarity)
- 코사인 유사도 (Cosine Similarity)
- 피어슨 유사도 (Pearson Similarity)
- 피어슨-베이스라인 유사도 (Pearson-Baseline Similarity)

KNN 가중치 예측 방법

- KNNBasic : 평점들을 단순히 가중 평균함, N^k는 k개의 가장 유사도가 큰 벡터의 집합
- KNNWithMeans : 평점들을 평균값 기준으로 가중 평균함
- KNNBaseline : 평점들을 베이스라인 모형의 값 기준으로 가중 평균함

```
from surprise import KNNWithMeans
from surprise import KNNBaseline
```

```
#우리는 코사인 유사도 사용!
sim_options = {'name': 'cosine'}
algo = surprise.KNNBasic(sim_options=sim_options)
cross_validate(algo, data)
```

```
{'test_rmse': array([0.36395562, 0.36613159, 0.3626647 , 0.36377216, 0.36405512]),
 'test_mae': array([0.25703913, 0.25690976, 0.25611493, 0.25782994, 0.25679896]),
```

KNNBasic의 Rmse는 평균 0.364

```
sim_options = {'name': 'cosine'}
algo = surprise.KNNBaseline(sim_options=sim_options)
cross_validate(algo, data)
```

```
{'test_rmse': array([0.33012503, 0.33180334, 0.33894008, 0.33964316, 0.3331826 ]),
 'test_mae': array([0.23667383, 0.23764794, 0.2416309 , 0.2418265 , 0.2374919 ]),
```

KNNBaseline의 Rmse는 평균 0.339

```
sim_options = {'name': 'cosine'}
algo = surprise.KNNWithMeans(sim_options=sim_options)
cross_validate(algo, data)
```

```
{'test_rmse': array([0.34245794, 0.33727805, 0.33479002, 0.34338102, 0.33852872]),
 'test_mae': array([0.24605065, 0.2446332 , 0.24367377, 0.24841396, 0.24476033]),
```

KNNWithMeans의 Rmse는 평균 0.335

PROCESS 02

Part 1. 머신러닝 – surprise 패키지

SVD, NMF – 잠재요인 방식

```
algo = surprise.SVD(n_factors=50)
cross_validate(algo, data)
```

```
{'test_rmse': array([0.34233825, 0.33834952, 0.33394751, 0.33504995, 0.3382407 ]),
 'test_mae': array([0.24322278, 0.24272906, 0.23993683, 0.24038491, 0.24241232]),
 'fit_time': (9.126651763916016,
  5.745615720748901,
  5.960139751434326,
  5.223
  5.9331
 'test_ti
 0.3400
 0.2054
 0.2712
 0.2094376087188720 7)}
```

```
algo = surprise.SVD(n_factors=50)
cross_validate(algo, data)['test_rmse'].mean()
```

```
0.3374783596637191
```

SVD의 Rmse는 평균 0.3374

```
from surprise import NMF
algo = surprise.NMF(n_factors=50)
cross_validate(algo, data)
```

```
{'test_rmse': array([0.73958185, 0.7442849 , 0.74048016, 0.74314277, 0.7321391 ]),
 'test_mae': array([0.65931511, 0.66244435, 0.65857706, 0.66147633, 0.65264787]),
 'fit_time': (16.08202576637268,
  15.507524728775024,
  16.544325351715088,
  19.686875343322754,
  1
 'te
 0
 0
 0
 0
```

```
algo = surprise.NMF(n_factors=50)
cross_validate(algo, data)['test_rmse'].mean()
```

```
0.7389788234720699
```

NMF의 Rmse는 평균 0.7389

PROCESS 02

Part 1. 머신러닝 – surprise 패키지

SVD – 잠재요인 방식

```
#10 fold CV
cross_validate(algo, data, measures = ['RMSE','MAE'], cv=10, verbose=True)
```

Evaluating RMSE, MAE of algorithm SVD on 10 split(s).

	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Fold 6	Fold 7	Fold 8	Fold 9	Fold 10	Mean	Std
RMSE (testset)	0.3306	0.3399	0.3409	0.3381	0.3354	0.3304	0.3389	0.3353	0.3404	0.3398	0.3370	0.0037
MAE (testset)	0.2401	0.2436	0.2434	0.2408	0.2390	0.2384	0.2427	0.2395	0.2430	0.2417	0.2412	0.0018
Fit time	5.74	5.66	6.18	7.08	8.89	7.36	6.98	7.32	7.60	7.04	6.99	0.91
Test time	0.12	0.13	0.13	0.46	0.18	0.11	0.23	0.21	0.12	0.13	0.18	0.10

```
#GridSearch
param_grid = {'n_epochs':[20,40,60], 'n_factors':[20,50,100]} #최적화할 파라미터 딕셔너리 형태로 지정
gs = GridSearchCV(SVD, param_grid, measures = ['rmse'], cv=10) #CV를 10개 폴트 세트로 지정, 성능 평가는 rmse, mae로 수행.
gs.fit(data)
```

```
#최고 RMSE Evaluation 점수와 그때의 하이퍼 파라미터는?
print(gs.best_score['rmse'])
print(gs.best_params['rmse'])
```

```
0.3301601942969134
{'n_epochs': 60, 'n_factors': 50}
```

PROCESS 02

Part 1. 머신러닝 – surprise 패키지

SVD – 잠재요인 방식

제대로 모델이 만들어졌을까? 한번 추천해보자.

```
#유저가 평가하지 않은 beer목록을 반환해 주는 함수 : get_unbeer_surprise

def get_unbeer_surprise(df, userid):
    beers = df[df['userid']==userid]['beer'].unique().tolist() #해당 유저가 평가한 beer

    total_beers = df['be

    unbeers = [beer for

    print('이미 평가한 beer 수: ',

    return unbeers
```

Get
유저

```
get_unbeer_surprise(df, 190)
```

```
이미 평가한 beer 수:  317 안 평
```

```
def recomm_beer_by_surprise(algo, userid, unbeers, top_n=10):

    predictions = [algo.predict(userid, beer) for beer in unbeers] #대입된 userid가 평가 안한 beer에 algo.predict 반복 적용

    def sortkey_est(pred):
        return pred.est #앞의 prediciton에서 예상 평점인 pred.est만 반환

    predictions.sort(key=sortkey_est, reverse=True) #predicions를 pred.est값에 따라 sorting함. reverse=True이므로 내림차순 정렬(평점 큰 순서로)
    top_predictions=predicti

    #top_n으로 추출된 영화
    top_beer_ids = [ int(pred
    top_beer_rating = [ pred.est for pred in top_predictions ]
    top_beer_titles = beers[beers.id.isin(top_beer_ids)]['name'] #데이터셋 필요
    top_beer_country = beers[beers.id.isin(top_beer_ids)]['country']
    top_beer_preds = [ (id, title, rating, country) for id, title, rating, country in zip(top_beer_ids, top_beer_titles, top_beer_rating, top_beer_country) ]
    #앞에 정의한 것들을 id, 제목, 평점 순으로 반환

    return top_beer_preds
```

Recomm_beer_by_surprise(algo,userid,unbeers)
unbeers 리스트를 가지고 직접 beer를 추천해주는 함수

PROCESS 02

Part 1. 머신러닝 – surprise 패키지

SVD – 잠재요인 방식

```
print('<<TOP-10 추천 맥주 리스트>>')
for top_beer in top_beer_preds:
    print("추천맥주: ", top_beer[1], " / beer_id:", top_beer[0], " / 예상 평점: ", np.round(top_beer[2],3), " / 이 맥주의 판매국: ", top_beer[3])
```

```
<<TOP-10 추천 맥주 리스트>>
추천맥주:  Julius  / beer_id: 115317  / 예상 평점:  4.781  / 이 맥주의 판매국:  US
추천맥주:  Cantillon Fou' Foune  / beer_id: 131782  / 예상 평점:  4.678  / 이 맥주의 판매국:  BE
추천맥주:  Dinner  / beer_id: 16814  / 예상 평점:  4.675  / 이 맥주의 판매국:  US
추천맥주:  Pseudo Sue  / beer_id: 57747  / 예상 평점:  4.664  / 이 맥주의 판매국:  US
추천맥주:  Bourbon County Brand Coffee Stout  / beer_id: 7971  / 예상 평점:  4.663  / 이 맥주의 판매국:  US
추천맥주:  Pliny The Elder  / beer_id: 150209  / 예상 평점:  4.634  / 이 맥주의 판매국:  US
추천맥주:  Heady Topper  / beer_id: 86237  / 예상 평점:  4.626  / 이 맥주의 판매국:  US
추천맥주:  Supplication  / beer_id: 72170  / 예상 평점:  4.62  / 이 맥주의 판매국:  US
추천맥주:  Double Dry Hopped Melcher Street  / beer_id: 22227  / 예상 평점:  4.617  / 이 맥주의 판매국:  US
추천맥주:  Vanilla Rye Bourbon County Brand Stout  / beer_id: 5281  / 예상 평점:  4.616  / 이 맥주의 판매국:  US
```

딥러닝

PROCESS 03

Part 2. 딥러닝

Data Preprocessing

원본 데이터
900만 row X 10 col

[userid, beerid, score]
세가지 변수만 사용

LabelEncoder
Object 형식의 username을 int형식으로 변환

```
#keras 라이브러리 로드
from keras.models import Model
from keras.layers import Input, Reshape, Dot
from keras.layers import Embedding, Dense, Dropout
from keras.optimizers import Adam
from keras.regularizers import l2
from keras.layers import Add, Activation, Lambda, Concatenate
```

Keras 라이브러리 사용

Part 2. 딥러닝

기본 함수
(Recommender V1)

Optimizer = Adam
Batch size = 1000
Epoch = 10
Learning rate = 0.001

```
class EmbeddingLayer:
    def __init__(self, n_items, n_factors):
        self.n_items = n_items
        self.n_factors = n_factors

    def __call__(self, x):
        x = Embedding(self.n_items, self.n_fact
        x = Reshape((self.n_factors,))(x)
        return x

def RecommenderV1(n_users, n_beers, n_fa

    #input: user, beer
    user = Input(shape=(1,))
    u = EmbeddingLayer(n_users, n_factors)(
    ub = EmbeddingLayer(n_users, 1)(user) #

    beer = Input(shape=(1,))
    b = EmbeddingLayer(n_beers, n_factors)(
    bb = EmbeddingLayer(n_beers, 1)(beer) #

    #input 연산
    x = Dot(axes=1)([u, b]) #예측행렬 R_hat은
    x = Add()([x, ub, bb])

    #최종 모델 만들기(input은 user, beer 2개를
    model = Model(inputs=[user, beer], outpu
    opt = Adam(lr=0.001)
    model.compile(loss=['mse'], metrics = ['m

    return model
```

```
Model: "model_1"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_1 (InputLayer)            (None, 1)            0
__________________________________________________________________________________________________
input_2 (InputLayer)            (None, 1)            0
__________________________________________________________________________________________________
embedding_1 (Embedding)         (None, 1, 50)        8246750     input_1[0][0]
__________________________________________________________________________________________________
embedding_3 (Embedding)         (None, 1, 50)        15477100    input_2[0][0]
__________________________________________________________________________________________________
reshape_1 (Reshape)             (None, 50)           0           embedding_1[0][0]
__________________________________________________________________________________________________
reshape_3 (Reshape)             (None, 50)           0           embedding_3[0][0]
__________________________________________________________________________________________________
embedding_2 (Embedding)         (None, 1, 1)         164935      input_1[0][0]
__________________________________________________________________________________________________
embedding_4 (Embedding)         (None, 1, 1)         309542      input_2[0][0]
__________________________________________________________________________________________________
dot_1 (Dot)                     (None, 1)            0           reshape_1[0][0]
                                                                 reshape_3[0][0]
__________________________________________________________________________________________________
reshape_2 (Reshape)             (None, 1)            0           embedding_2[0][0]
__________________________________________________________________________________________________
reshape_4 (Reshape)             (None, 1)            0           embedding_4[0][0]
__________________________________________________________________________________________________
add_1 (Add)                     (None, 1)            0           dot_1[0][0]
                                                                 reshape_2[0][0]
                                                                 reshape_4[0][0]
==================================================================================================
Total params: 24,198,327
Trainable params: 24,198,327
Non-trainable params: 0
__________________________________________________________________________________________________
```

PROCESS 03

Part 2. 딥러닝

기본 함수(Recommender V1)

```
## fit model
import numpy as np
seed = 7
np.random.seed(seed)
history_mf = re1model.fit(x=X_train_array, y=y_train, batch_size=1000, epochs=10,
                verbose=1, validation_data=(X_test_array, y_test))
```

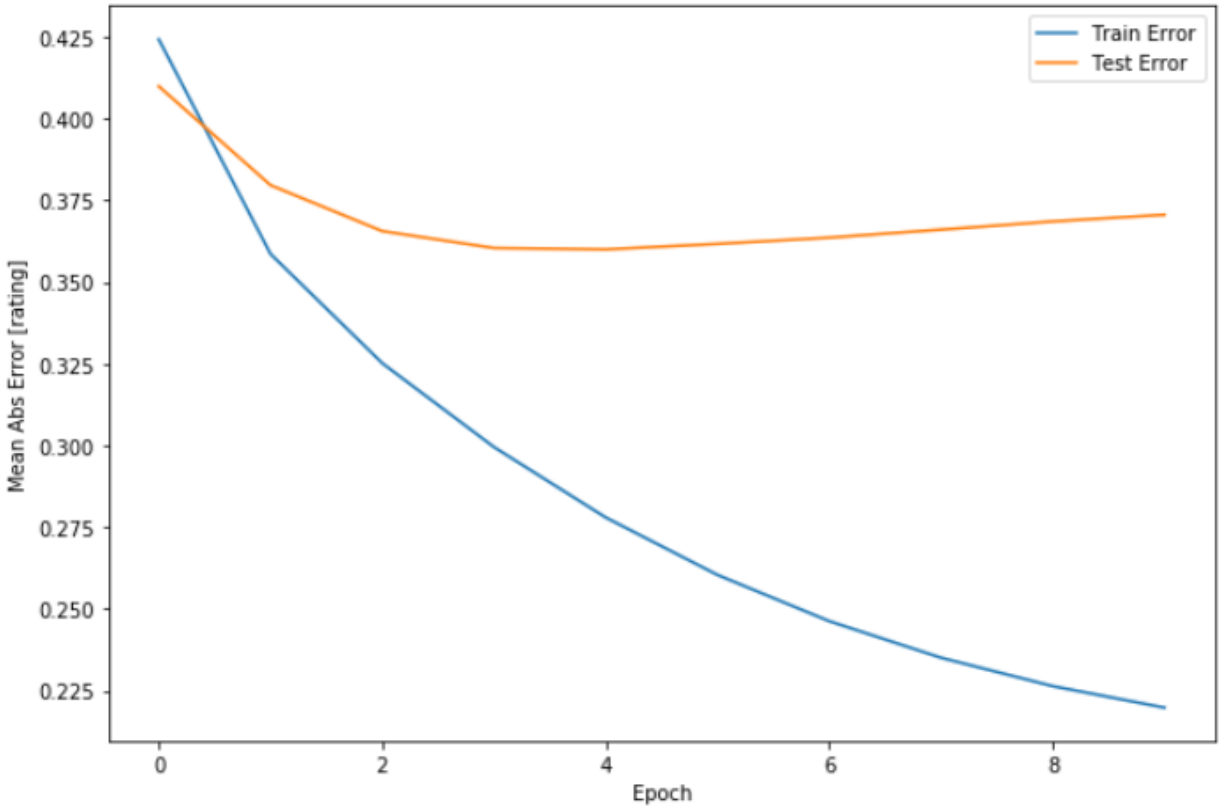
모델 학습시키기
: 모든 batch_size 1000,
Epochs 10으로 통일
: 재현가능한 결과를 얻기 위해
random seed는 7로 통일

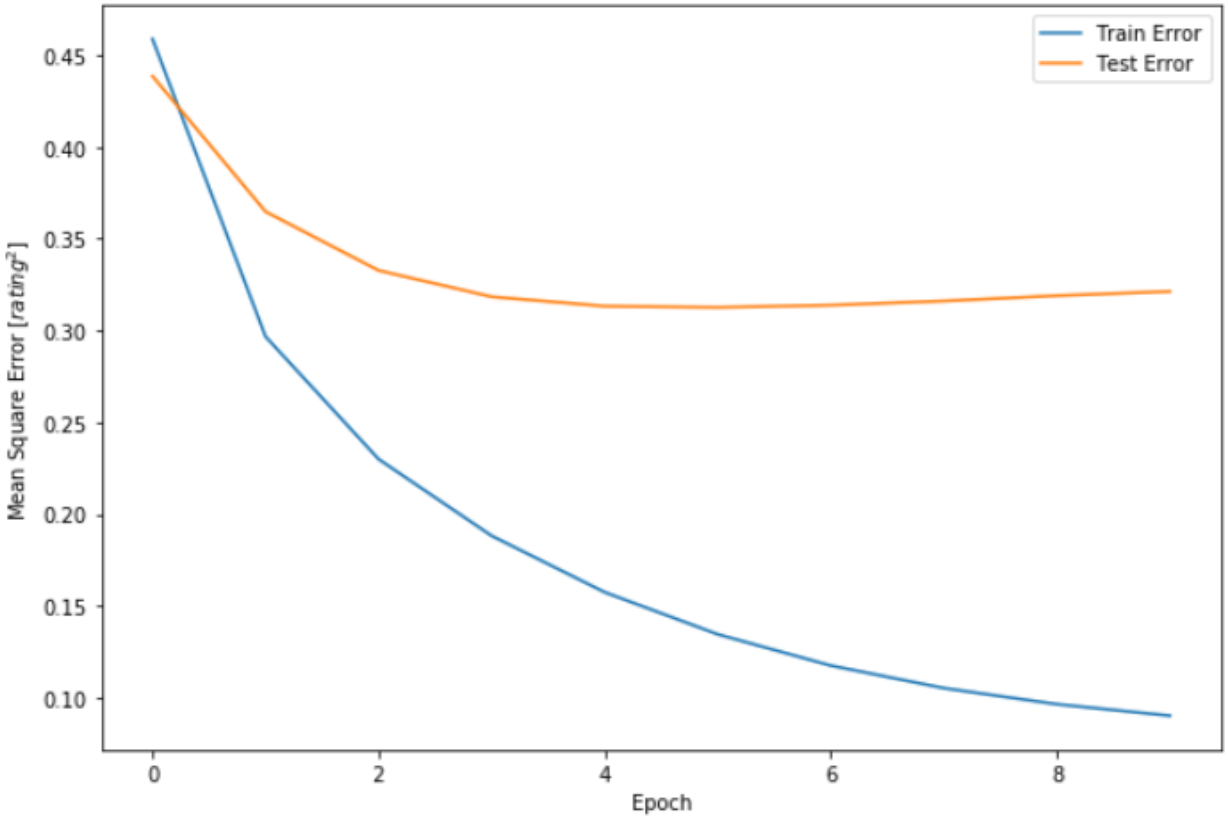
```
Train on 7258502 samples, validate on 1814626 samples
Epoch 1/10
7258502/7258502 [==============================] - 1784s 246us/step - loss: 0.4587 - mae: 0.4242 - mse: 0.4587 - val_loss: 0.4385 - val_mae: 0.40
98 - val_mse: 0.4385
Epoch 2/10
7258502/7258502 [==============================] - 1720s 237us/step - loss: 0.2967 - mae: 0.3585 - mse: 0.2967 - val_loss: 0.3648 - val_mae: 0.37
96 - val_mse: 0.3648
Epoch 3/10
7258502/7258502 [==============================] - 1721s 237us/step - loss: 0.2300 - mae: 0.3251 - mse: 0.2300 - val_loss: 0.3327 - val_mae: 0.36
55 - val_mse: 0.3327
Epoch 4/10
7258502/7258502 [==============================] - 1718s 237us/step - loss: 0.1883 - mae: 0.2995 - mse: 0.1883 - val_loss: 0.3184 - val_mae: 0.36
04 - val_mse: 0.3184
Epoch 5/10
7258502/7258502 [==============================] - 1723s 237us/step - loss: 0.1576 - mae: 0.2779 - mse: 0.1576 - val_loss: 0.3133 - val_mae: 0.35
99 - val_mse: 0.3133
Epoch 6/10
7258502/7258502 [==============================] - 1725s 238us/step - loss: 0.1347 - mae: 0.2604 - mse: 0.1347 - val_loss: 0.3127 - val_mae: 0.36
16 - val_mse: 0.3127
```

PROCESS
03

Part 2. 딥러닝

기본 함수(Recommender V1)





Test error

Train error

지는 않았다.

Part 2. 딥러닝

기본 함수(Recommender V1)

```
class EmbeddingLayer:
    def __init__(self, n_items, n_factors):
        self.n_items = n_items
        self.n_factors = n_factors

    def __call__
        x = Embe
        x = Resha
        return x

def Recomme

    #input: use
    user = Inpu
    u = Embedd
    ub = Embed

    beer = Input(shape=(1,))
    b = EmbeddingLayer(n_beers, n_factors)(beer) #b은 beer * n_factor
    bb = EmbeddingLayer(n_beers, 1)(beer) #beer_bias

    #input 연산
    x = Dot(axes=1)([u, b]) #예측행렬 R_hat은 u*b를 한 것.
    x = Add()([x, ub, bb])

    #최종 모델 만들기(input은 user, beer 2개를 받고, output은 x를 반환.)
    model = Model(inputs=[user, beer], outputs=x)
    opt = Adam(lr=0.001)
    model.compile(loss=['mse'], metrics = ['mae','mse'] ,optimizer=opt)

    return model
```

1) Layer의 개수 추가

```
#input 연산- activation 2가지를 추가. relu와 sigmoid
x = Concatenate()([u, b])

x = Dense(10, kernel_initializer='he_normal')(x)
x = Activation('relu')(x)

x = Dense(1, kernel_initializer='he_normal')(x)
x = Activation('sigmoid')(x)
x = Lambda(lambda x: x * (max_rating - min_rating) + min_rating)(x)
```

2) Activation 다양화

```
#input 연산
x = Concatenate()([u, b])

x = Dense(10, kernel_initializer='he_norma
x = Activation('tanh')(x)

x = Dense(5, kernel_initializer='he_norma
x = Activation('sigmoid')(x)

x = Dense(1, kernel_initializer='he_norma
x = Activation('relu')(x)
```

3) Optimizer 다양화

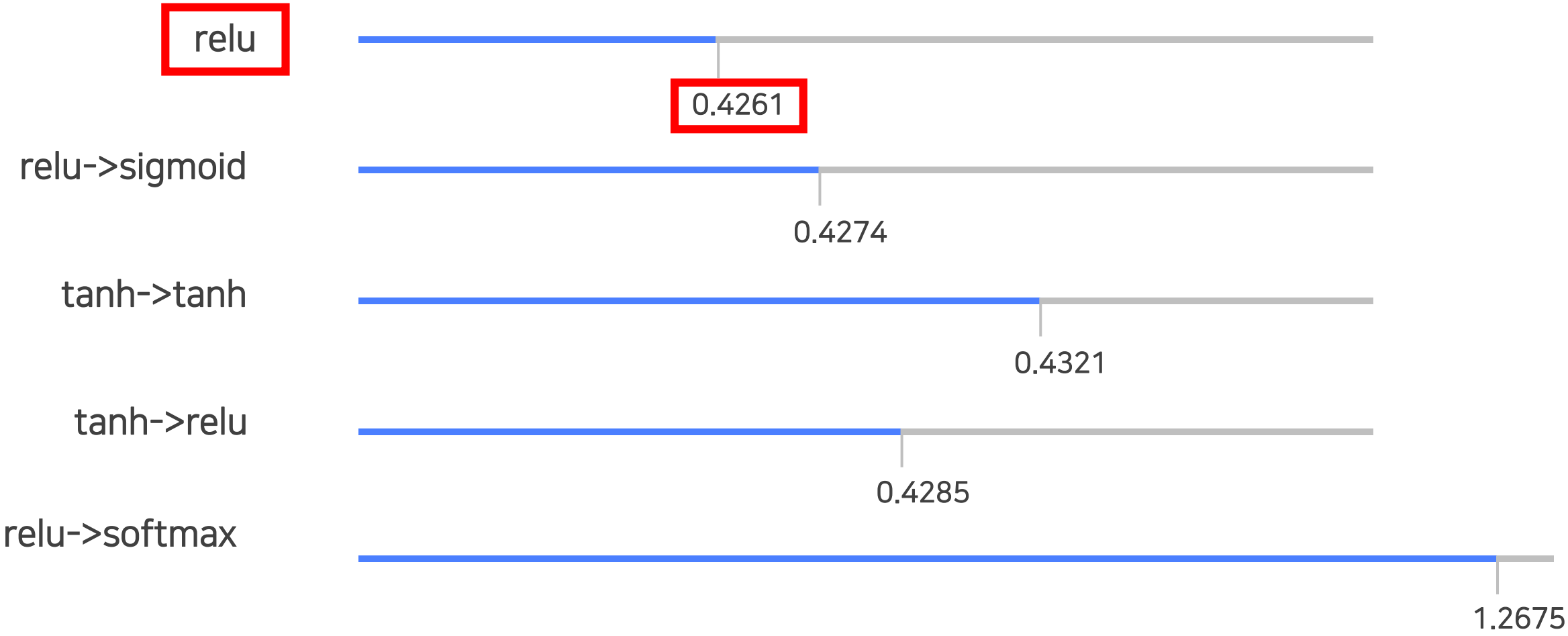
```
model.compile(loss=['mse'], metrics = ['mae','mse'],optimizer='RMSprop')
return model
```

PROCESS
03

Part 2. 딥러닝

기본 함수(Recommender V1)에서 레이어의 activation 다양화

기준: test RMSE

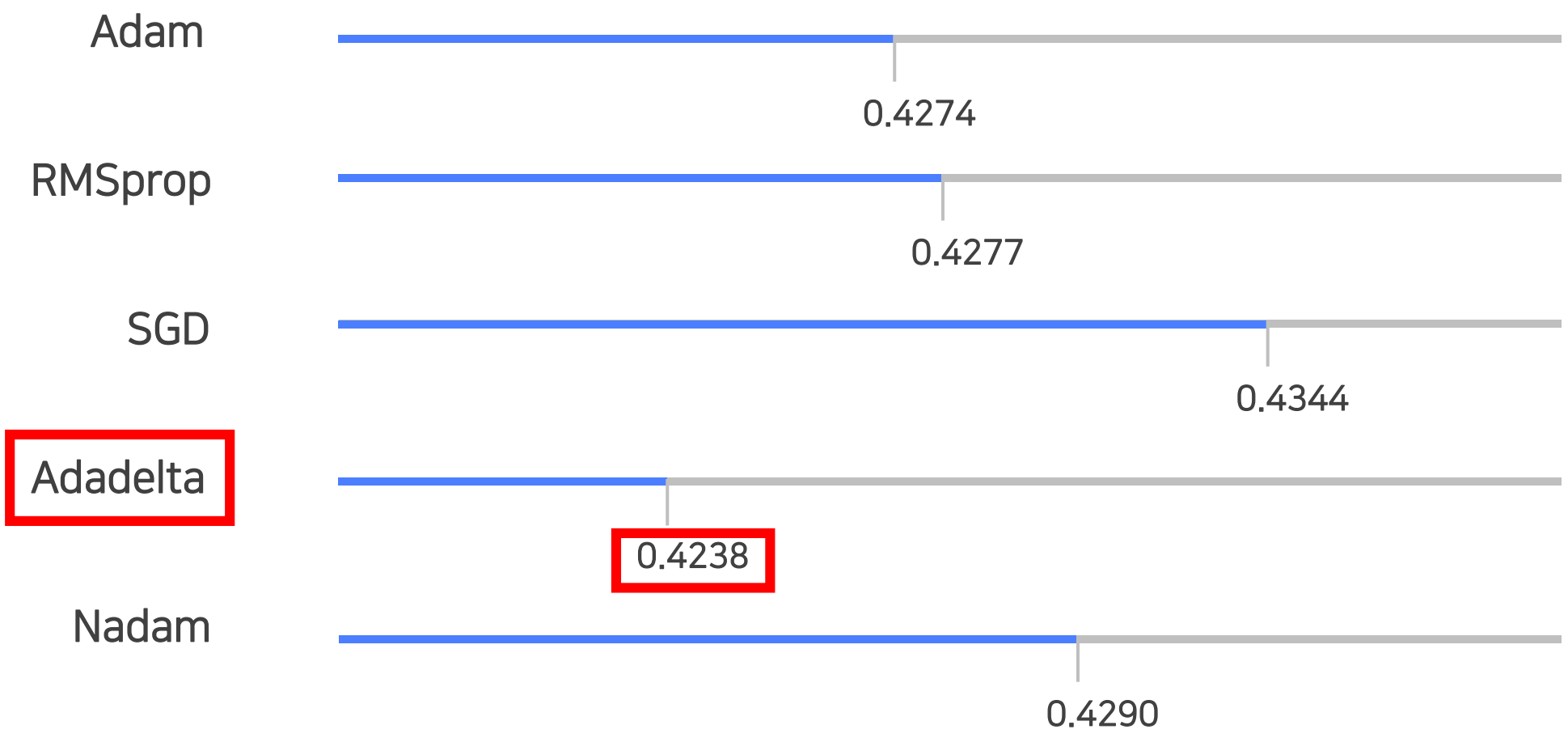


PROCESS
03

Part 2. 딥러닝

기본 함수(Recommender V1)에서 optimizer만 바꿔보기
Activation 함수: relu->sigmoid

기준: test RMSE



Part 2. 딥러닝

신경망에서 과적합 방지하기 위해 사용하는 방법

정리하면 신경망에서 과대적합을 방지하기 위해 가장 널리 사용하는 방법은 다음과 같습니다:

- 더 많은 훈련 데이터를 모읍니다.
- 네트워크의 용량을 줄입니다.(output의 reshape)
- 가중치 규제를 추가합니다.(L1 or L2)
- 드롭아웃을 추가합니다.(dropout)

모델을 너무 오래 훈련하면 과대적합되기 시작하고 테스트 세트에서 일반화되지 못하는 패턴을 훈련 세트에서 학습합니다. 과대적합과 과소적합 사이에서 균형을 잡아야 합니다.

1) 과대적합을 막는 가장 좋은 방법은 더 많은 훈련 데이터를 사용하는 것입니다. 많은 데이터에서 훈련한 모델은 자연적으로 일반화 성능이 더 좋습니다.
2) 데이터를 더 준비할 수 없을 때 그다음으로 가장 좋은 방법은 규제(regularization)와 같은 기법을 사용하는 것입니다. 모델이 저장할 수 있는 정보의 양과 종류에 제약을 부과하는 방법입니다. 네트워크가 소수의 패턴만 기억할 수 있다면 최적화 과정 동안 일반화 가능성이 높은 가장 중요한 패턴에 초점을 맞출 것입니다.

Part 2. 딥러닝

과적합 방지를 위한 규제

```
def RecommenderV2(n_users, n_movies, n_factors, min_rating, max_rating):

    #input: user, beer
    user = Input(shape=(1,))
    u = EmbeddingLayer(n_users, n_factors)(user)

    beer = Input(shape=(1,))
    b = EmbeddingLayer(n_beers, n_factors)(beer)

    #output: x, 연산과정
    x = Concatenate()([u, b])

    x = Dense(10, kernel_initializer='he_normal')(x)
    x = Activation('relu')(x)

    x = Dense(1, kernel_initializer='he_normal')(x)
    x = Activation('sigmoid')(x)
    x = Lambda(lambda x: x * (max_rating - min_rating) + min_rating)(x)

    #완성되는 model
    model = Model(inputs=[user, beer], outputs=x)
    opt = Adam(lr=0.001)
    model.compile(loss=['mse'], metrics = ['mae','mse'] ,optimizer=opt)

    return model
```

Usage of regularizers

Regularizers allow to apply penalties on layer parameters or layer activity during optimization. These penalties are incorporated in the loss function that the network optimizes.

The penalties are applied on a per-layer basis. The exact API will depend on the layer, but the layers `Dense`, `Conv1D`, `Conv2D` and `Conv3D` have a unified API.

These layers expose 3 keyword arguments:

- `kernel_regularizer`: instance of `keras.regularizers.Regularizer`
- `bias_regularizer`: instance of `keras.regularizers.Regularizer`
- `activity_regularizer`: instance of `keras.regularizers.Regularizer`

```
def RecommenderV3(n_users, n_movies, n_factors, min_rating, max_rating):

    #input: user, beer
    user = Input(shape=(1,))
    u = EmbeddingLayer(n_users, n_factors)(user)

    beer = Input(shape=(1,))
    b = EmbeddingLayer(n_beers, n_factors)(beer)

    #output: x, 연산과정
    x = Concatenate()([u, b])
    x = Dropout(0.05)(x)

    x = Dense(10, kernel_initializer='he_normal')(x)
    x = Activation('relu')(x)
    x = Dropout(0.5)(x)

    x = Dense(1, kernel_initializer='he_normal')(x)
    x = Activation('sigmoid')(x)
    x = Lambda(lambda x: x * (max_rating - min_rating) + min_rating)(x)

    #완성되는 model
    model = Model(inputs=[user, beer], outputs=x)
    opt = Adam(lr=0.001)
    model.compile(loss=['mse'], metrics = ['mae','mse'] ,optimizer=opt)

    return model
```

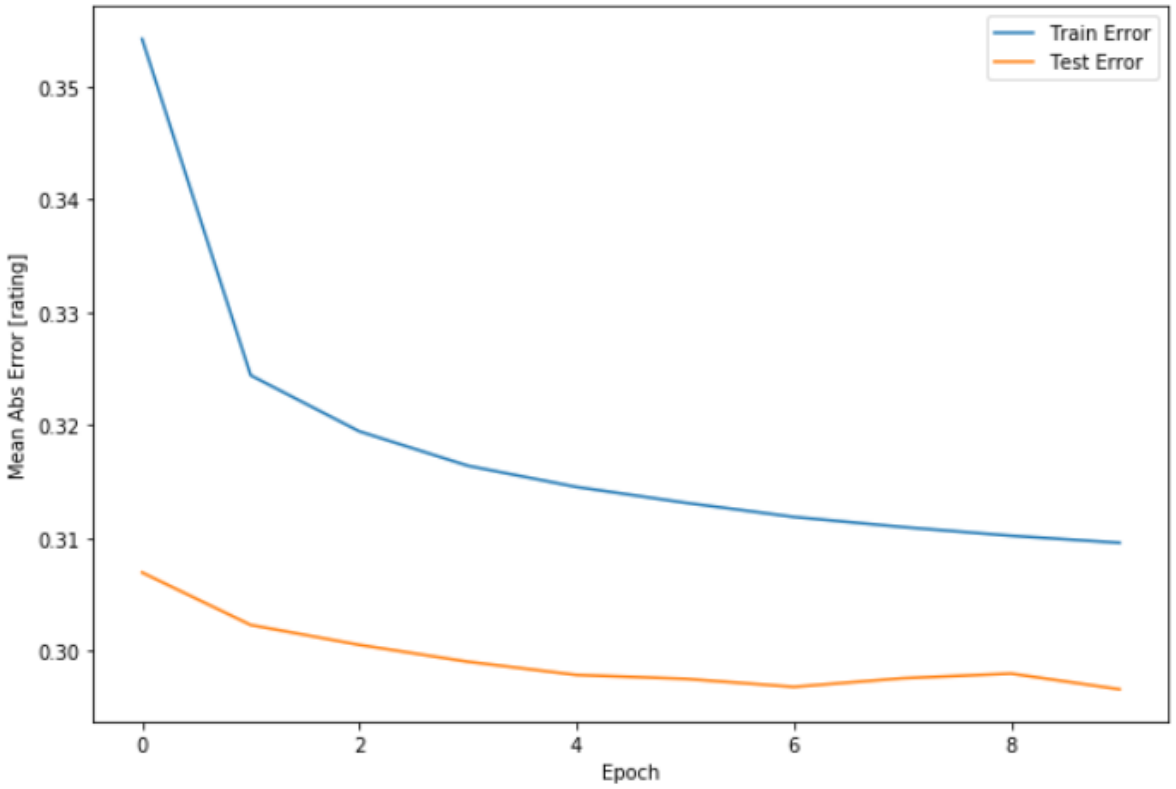
커널에 L2 규제

Dropout

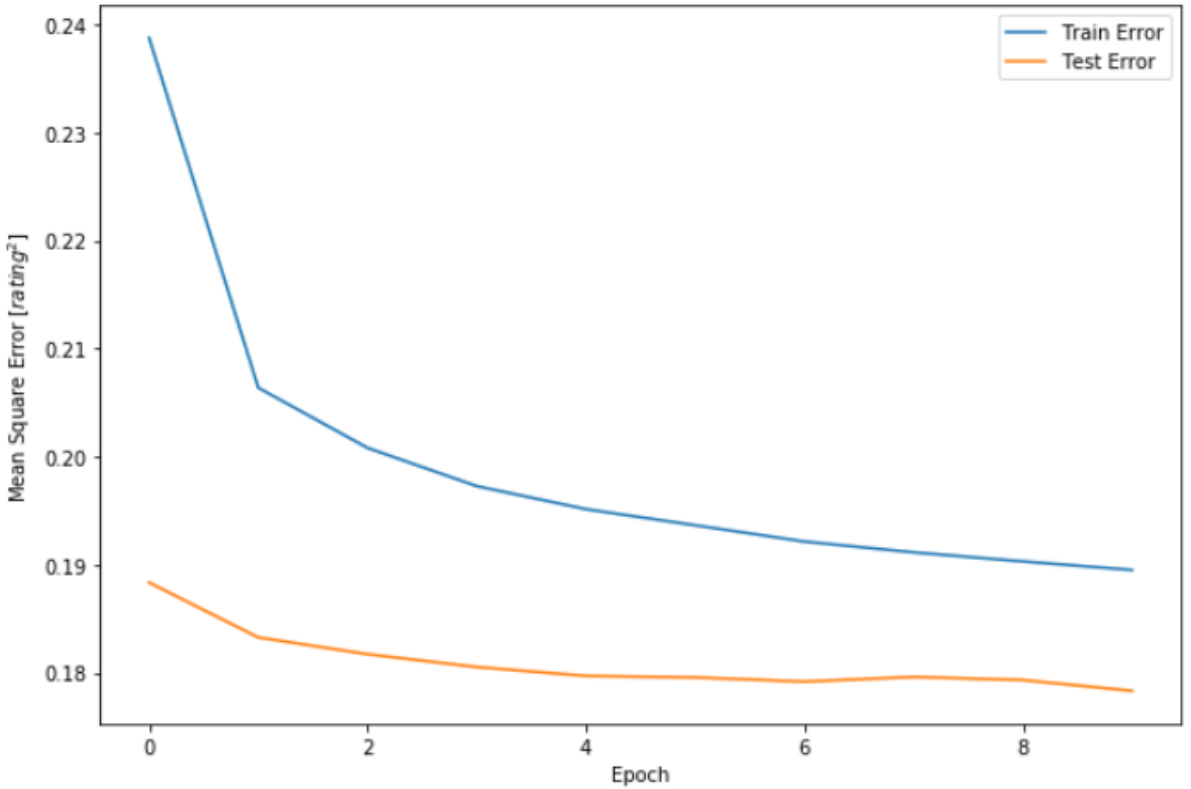
PROCESS
03

Part 2. 딥러닝

Dropout



MAE



MSE

Part 2. 딥러닝

그런데, 어떻게 test error가 train error보다 적게 나올 수가 있을까?
이것에 대해 구글링한 결과 찾아온 답변

Validation Error less than training error?

Asked 3 years, 11 months ago Active 1 month ago Viewed 97k times

59

I found two questions here and here about this issue but there is no obvious answer or explanation yet.I enforce the same problem where the validation error is less than training error in my Convolution Neural Network. What does that mean?

One possibility: If you are using dropout regularization layer in your network, it is reasonable that the validation error is smaller than training error. Because usually dropout is activated when training but deactivated when evaluating on the validation set. You get a more smooth (usually means better) function in the latter case.

share cite improve this answer

answered Apr 6 '16 at 14:02

D-K
1,229 2 5 2

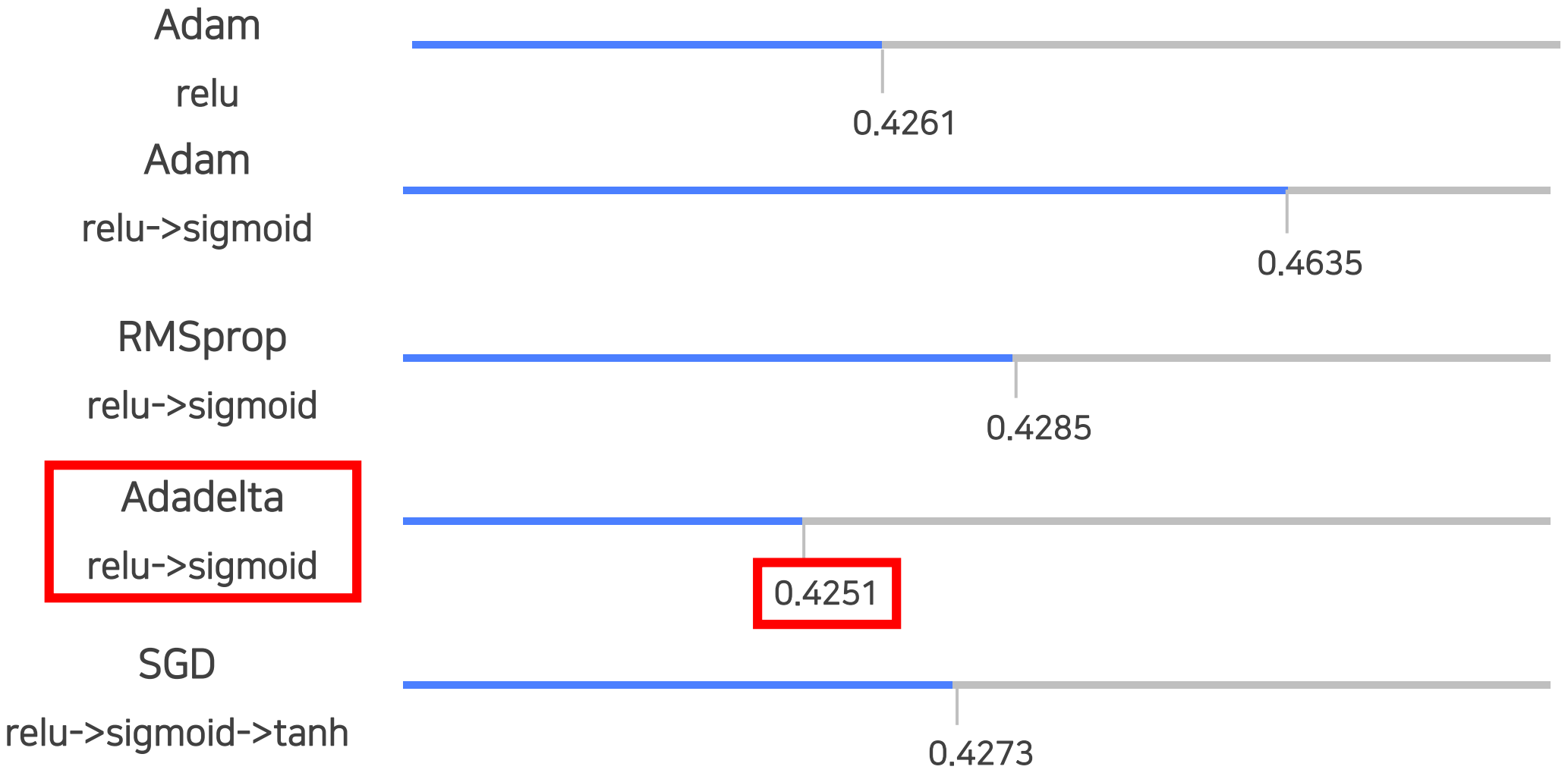
추천을 많이 받은 답변 중 하나로, “당신이 dropout 규제를 사용하고 있다면 그럴 수 있습니다. 왜냐하면 보통 dropout은 training에선 활성화되나 test set을 평가할 때는 그러지 않을 수 있기 때문입니다. 그러니 이 경우에는 좀 더 smooth한 function을 사용하세요.”

PROCESS 03

Part 2. 딥러닝

L2 규제를 적용한 버전
- activation, layer, optimizer 바꿔보기

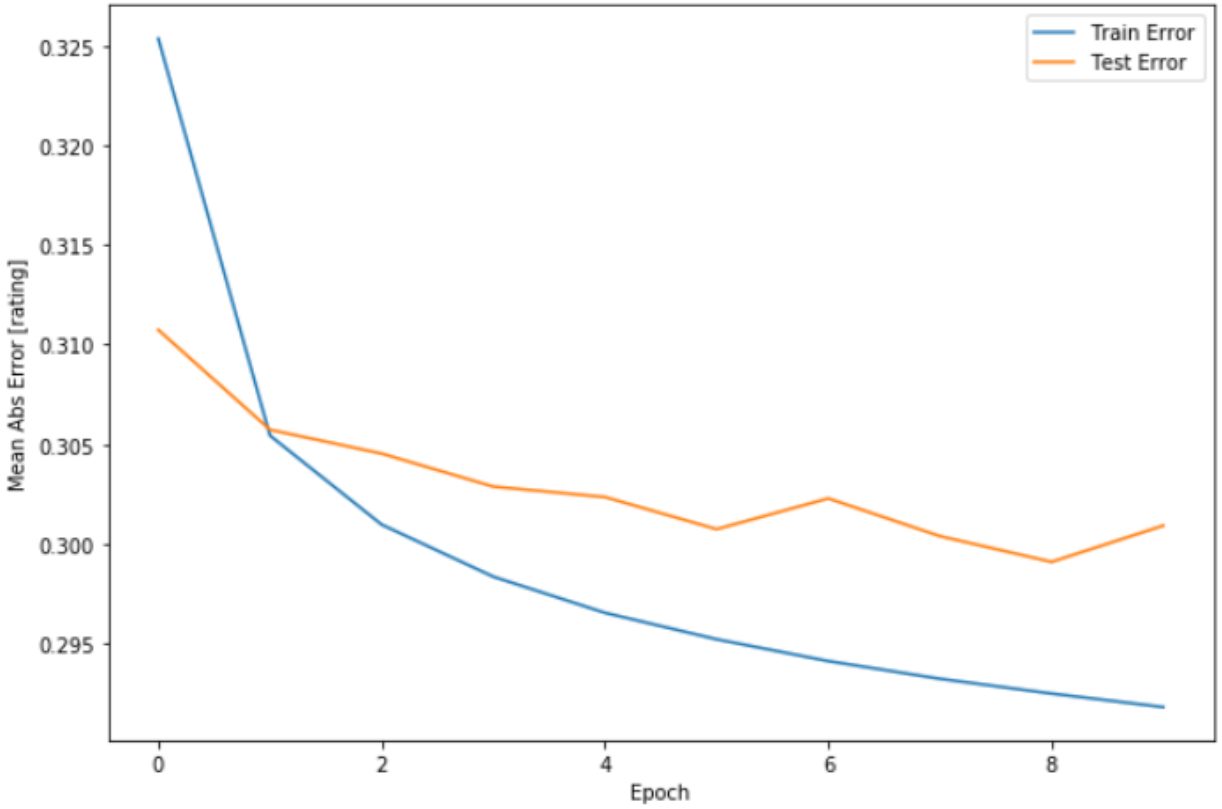
기준: test RMSE



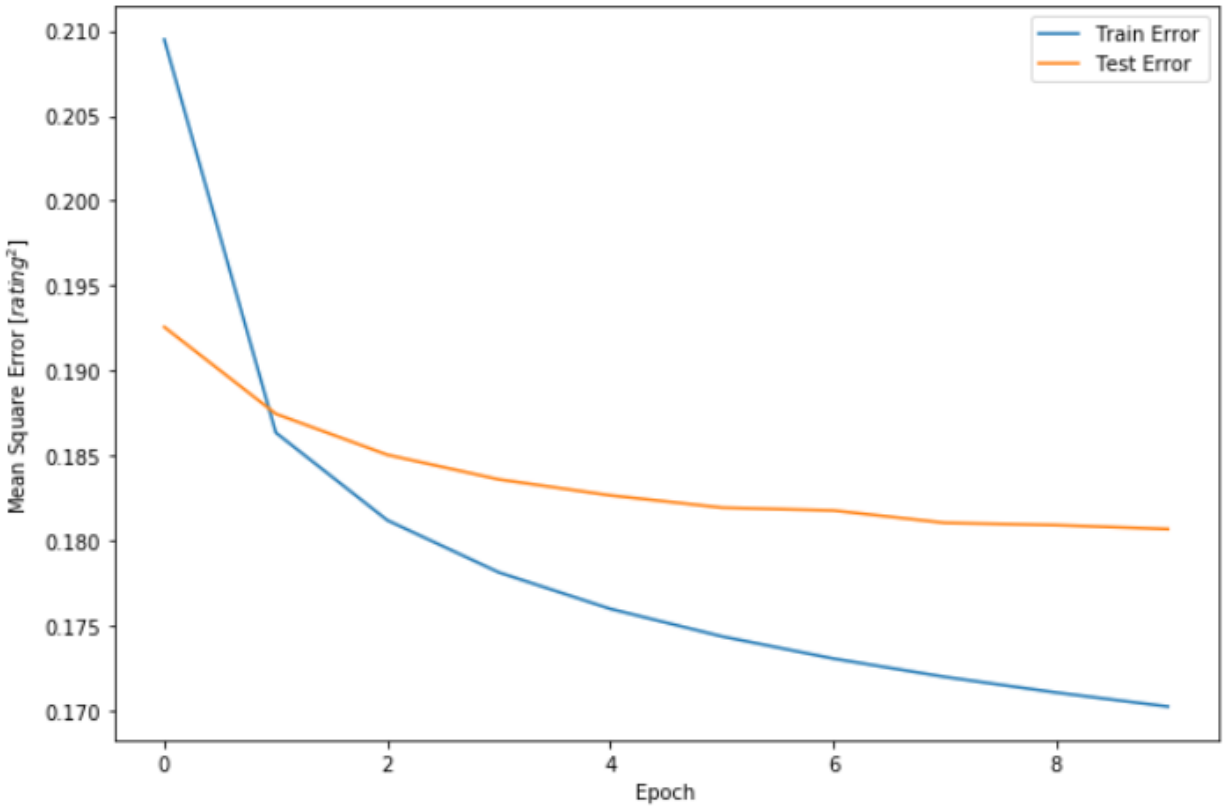
PROCESS
03

Part 2. 딥러닝

Adadelta, relu+sigmoid, L2



MAE



MSE

PROCESS 03

Part 2. 딥러닝

Adadelta, relu+sigmoid, L2 모델

```
def get_unbeer_keras(df, userid):
    beers = df[df['user']==userid]['beer'].unique().tolist() #해당 유저가 평가한 beer
    total_beers = df['beer'].unique().tolist() #모든 beer
    unbeers = [beer for beer in total_beers if beer not in beers] #해당 유저가 평가안한 beer
    return unbeers

def recomm_beer_by_keras(userid, model, top_n):
    unbeers = get_unbeer_keras(df,userid)
    predict =[v[0][0] for v in  [model.predict([[userid],[beer]]) for beer in unbeers]]
    ndf= pd.DataFrame({'beer':unbeers,'rating':predict})
    final = ndf.sort_values(by='rating', ascending=False)[:top_n]
    return final
```

```
unbeers = get_unbeer_keras(df2, 190)
recomm_beer_by_keras(190, re18model, 10)
```

Wall time: 4min 51s

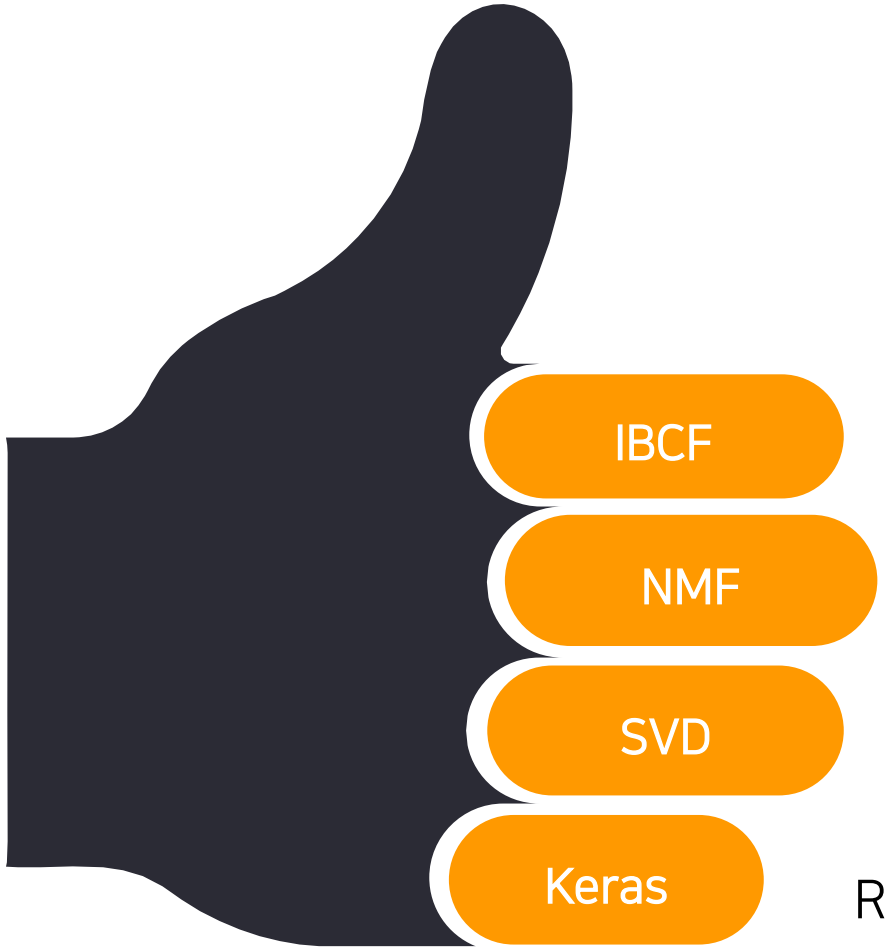
	beer	rating
186538	68022	4.821929
191828	52574	4.785089
210211	122288	4.777422
85480	86880	4.764430
108760	34828	4.761798
12954	76214	4.753008
169444	98340	4.751093
187446	58238	4.744469
83413	164241	4.743034
109217	80528	4.737667

결론

PROCESS 04

결론

Best Model



RMSE: 0.6287

RMSE = 0.2609

RMSE = 0.3301

RMSE = 0.4238

잠재요인 협업필터링 NMF

Factors = 50
Epoch = 200
Learning rate = 0.01

PROCESS
05

추천 알고리즘의 의의

1. 개인별 맞춤 추천을 통한 소비자의 탐색효과 증대
2. 아이템에 대한 사용자의 취향 예측을 통한
   새로운 마케팅의 효율성 극대화
3. 맥주뿐만 아니라, 모든 컨텐츠에 대해
   연관 상품이나 높은 선호도의 상품 추천을 적용할 수 있음

PROCESS
05

Reference

파이썬 머신러닝 완벽 가이드

1. https://datascienceschool.net/view-notebook/fcd3550f11ac4537acec8d18136f2066/
2. https://github.com/NicolasHug/Surprise/issues/140 surprise에서 평점이 다 똑같이 나오는 오류 참고
3. https://www.kaggle.com/ruancmoral/beer-analysis beer & breweries.csv 데이터 가공
4. https://www.johnwittenauer.net/deep-learning-with-keras-recommender-systems/ 케라스 코드 참고
   https://www.onceupondata.com/2019/02/10/nn-collaborative-filtering/
   https://www.tensorflow.org/tutorials/keras/overfit_and_underfit?hl=ko 과적합 방지
5. 이미지
   1. https://images.app.goo.gl/TeK7wVDbei9PYHdy8 KNN
   2. https://images.app.goo.gl/PjWhBoic63EHSdV89 코사인
   3. https://images.app.goo.gl/wjVCjmhamJv1MCQn8 NMF with SGD
   4. https://images.app.goo.gl/f9hSfWgPZHHhWiTv9 SVD

발표를 마치겠습니다
감사합니다 ☺