Deep Learning 을 활용한 맛집 추천 시스템

부서	Naver Data Lab – Biz data
팀원	백우현, 한대룡

목 차

Ⅰ. 서론

1.1 알고리즘 순서도

1.2 데이터 소개 및 방향제시

Ⅱ. 본론

2.1 데이터 전처리 및 EDA

2.2 User-to-Item Modeling

2.2.1 Deep Neural Network (with Entity Embedding)

2.2.2 Neural Network Matrix Factorization

2.2.3 Model Ensemble 및 평가

2.2.4 모델 Simulation 및 비교 (with 100 Sample)

2.3 Item-to-Item Modeling

2.3.1 Distance with Embedded Values

2.3.2 Deep Neural Network (with Selected Data)

2.2.3 모델 Simulation

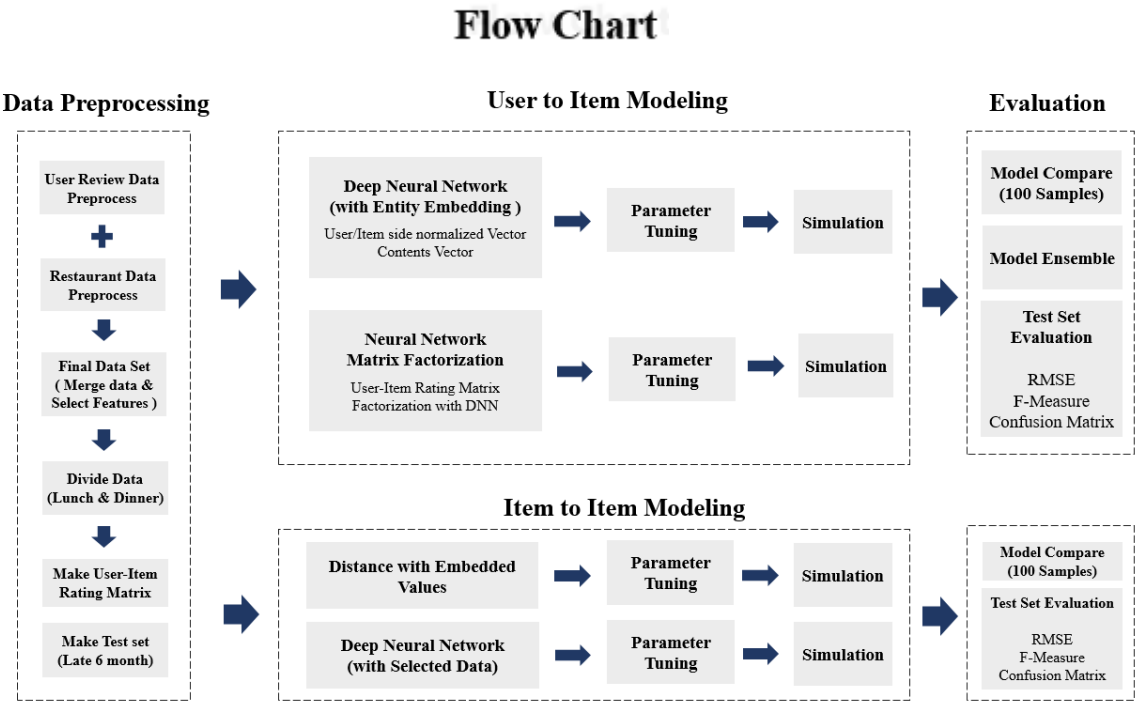
Ⅲ. 결론

3.1 의의 및 향후 연구과제

3.2 참고 문헌

Ⅰ. 서론

1.1 알고리즘 순서도



1.2 데이터 소개 및 방향제시

데이터는 '타베로그' (https://tabelog.com/) 일본 맛집 사이트를 크롤링한 데이터이다. 크게 세 가지로 구성되어 있으며 그 종류는 식당 정보 데이터, 리뷰 데이터, 식당 카테고리 데이터이다. 이 중 도쿄에 위치한 식당만 사용을 하였고, 이를 바탕으로 '도쿄 맛집 추천시스템'을 구현하였다.

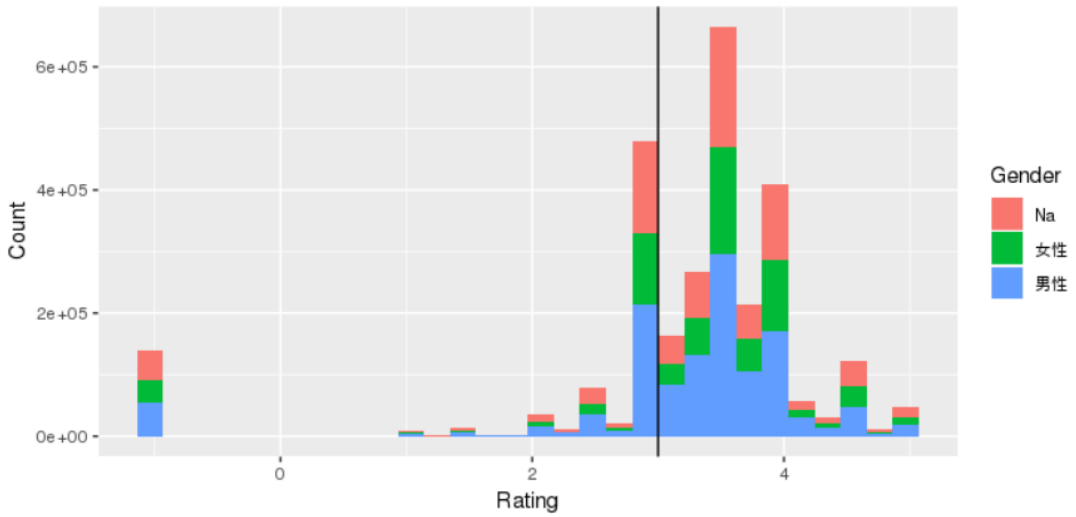
최근에 굉장히 다양한 플랫폼에서 추천시스템을 구축하여 이를 바탕으로 서비스를 제공할 만큼 추천시스템에 관심이 높아지는 추세이다. 그러나 딥러닝을 이용한 추천시스템은 아직 널리 사용되고 있지 않다. 가장 큰 이유는 아마 딥러닝 모델의 블랙박스 특성 때문일 것이다. 허나, 딥러닝 모델의 높은 정확성에 기반하여 더욱 정교한 추천시스템이 가능해진다면 이 역시 실용화될 수 있을 것일 예상한다. 따라서 여기서는 딥러닝을 이용한 추천시스템 구축에 초점을 맞췄다.

Ⅱ. 본론

2.1 데이터 전처리 및 EDA

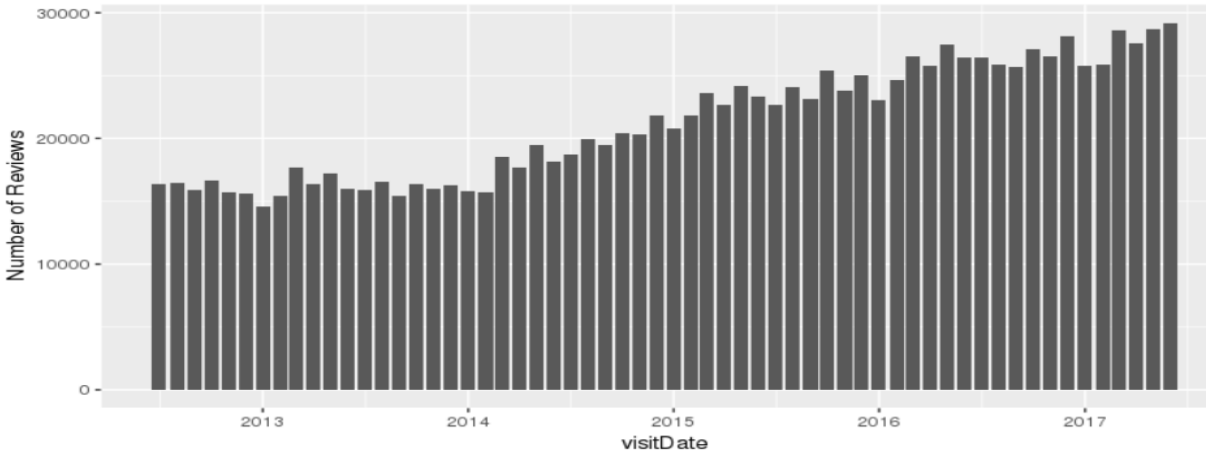
(github : Daeryong –preprocess_version1 + Woohyun – usertoitem_preprocess 부분)

1) 3점 미만 평점 삭제



위의 히스토그램을 바탕으로 대부분의 평점이 3 ~ 5 사이의 값 임을 알 수 있다. 이 중 3 미만의 값은 삭제를 하였는데 그 이유는 맛집 추천의 특성상 평점이 높은 식당에 초점을 맞출 필요가 있기 때문이다. 평점이 지나치게 낮은 식당에 대한 리뷰를 모두 포함할 경우 추천시스템에 불필요한 식당의 정보까지 데이터셋에 추가되며 이로 인해 더욱 sparse한 데이터셋이 구성된다는 문제점이 발생한다. 따라서 이를 방지하고자 3점 미만의 평점은 삭제를 하였다.

2) 최근 5년 데이터만 사용



초기 데이터는 2000년도의 리뷰 데이터가 존재를 할 정도로 굉장히 많은 리뷰가 존재하였다. 하지만, 과거의 식당 데이터는 그 식당이 현재 존재하지 않을 수도 있으며 현재의 식당 트렌드를 반영하긴 어렵다고 판단을 하였다. 따라서 2012년 7월 1일부터 2017년 6월 30일까지 최근 5개년의 데이터만 추출을 하여 이용하였다.

3) 리뷰 수 상위 1% 삭제

리뷰 데이터의 공통된 문제점은 특정 인물이 지나치게 많은 리뷰를 남기는 경우가 빈번하는 것이다. 오른쪽 결과를 보면 알 수 있듯이 리뷰를 가장 많이 한 사람은 무려 5년 동안 5900개의 리뷰를 남겼음을 알 수 있다. 이 데이터를 모두 사용할 경우 특정 인물에게 지나치게 편향된 결과를 초래하는 큰 문제점이 발생한다. 따라서, 지나치게 많은 리뷰를 한 리뷰를 삭제를 하였는데 그 기준은 상위 1%로 선정을 하였다.

	reviewerId	count
1	/rvwr/001561557/	5897
2	/rvwr/000618208/	4885
3	/rvwr/000216899/	3539
4	/rvwr/00010250/	3129
5	/rvwr/000789416/	3102
6	/rvwr/00014456/	3088
7	/rvwr/mapper/	3080
8	/rvwr/00012472/	3064
9	/rvwr/00012490/	3059
10	/rvwr/gems/	2984

4) Lunch & Dinner 구분

리뷰 데이터에는 ratingType 변수가 존재하는데 이는 그 리뷰가 점심 메뉴에 대한 것인지 저녁 메뉴에 대한 것인지 정보가 포함되어 있다. 한 식당 내에서 점심 메뉴에 대한 평점과 저녁 메뉴에 대한 평점이 상이하다면 이를 구분해서 진행할 필요가 있고, 그렇지 않다면 모두 통합하여 사용해도 무방하다. 따라서 이 차이가 유의미한지를 확인하기 위해 paired t-test(대응표본 T 검정)를 진행하였다. 한 식당 내에서 점심에 대한 리뷰와 저녁에 대한 리뷰가 존재하는 것이므로 일반적인 t-test가 아니라 paired t-test를 이용하였으며 결과값은 아래와 같다.

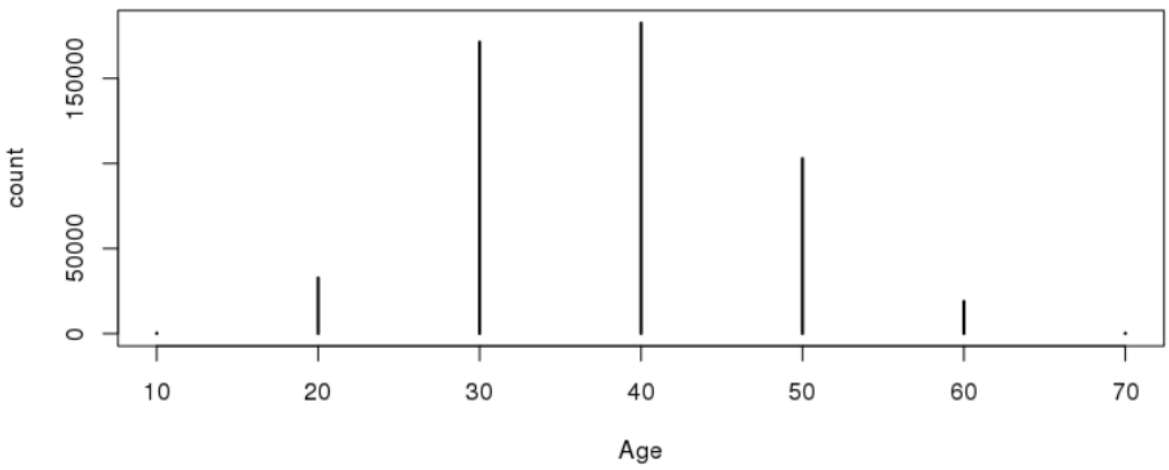
```
        Welch Two Sample t-test

data:  review_dinner$mean_score and review_lunch$mean_score
t = 20.81, df = 183260, p-value < 2.2e-16
alternative hypothesis: true difference in means is not equal to 0
```

귀무가설은 식당 내에서 점심 평점과 저녁 평점이 차이가 없다는 것을 의미하며 대립가설은 차이가 있음을 의미한다. 위의 결과에서 볼 수 있듯이 유의수준 하에서 귀무가설을 기각하므로 식당 내의 점심 평점과 저녁 평점의 모평균이 다르다는 결론을 도출했다. 실제로 일본의 경우 한 식당 내에서도 점심 메뉴와 저녁 메뉴가 상이한 경우가 많이 존재하기 때문에 이런 결과가 나온

것이라 예상된다. 따라서 점심 데이터와 저녁 데이터를 한 번에 사용하는 것이 아니라 각각 구분을 하여 모델링을 진행하였다.

5) 나이 10대 & 70대 삭제



총 127만 개의 데이터 중 10대와 70대의 샘플 수는 각각 300개 정도밖에 되지 않았다. 유저의 나이 역시 모델링 과정에서 인풋 데이터로 사용이 되기에 다음과 같은 10대와 70대처럼 불균등한 변수는 결과값의 왜곡을 초래하는 문제를 야기시킬 수 있다. 또한, 추천시스템의 특성상 대다수의 사람들에게 적용될 수 있어야 하므로 한 쪽으로 치우친 데이터를 이용하는 것이 오히려 모델의 성능을 악화시킬 것이라 판단하여 이에 해당하는 데이터를 삭제하였다.

6) 개별 모델링

앞서 말했듯이 lunch와 dinner를 각각 구분하였고, 추가적으로 특별구 별로 구분하여 모델링을 진행하였다. 초기에는 도쿄의 모든 식당을 대상으로 하나의 통합된 모델을 제공하려 했으나 데이터의 수가 지나치게 많아 모델을 학습시키는데 어려움이 따랐고, 자신의 지역과 거리가 지나치게 먼 식당을 추천하는 방식이 현실적이지 않다고 판단했기 때문이다. 즉, 다른 구의 식당을 추천하기 보단 구를 구분하여 근처의 식당을 추천하는 것이 합리적이라 판단하였다. 다음에 나오는 자료는 모두 미나토 구(港区)의 데이터를 활용한 자료이다.

7) User / Item Rating Matrix 생성

이후 진행할 모든 모델링 과정에서는 반복적으로 User / Item Rating Matrix 가 사용된다. 이 Matrix는 Row에 Unique한 User가, Column에 Unique한 Item(식당)이 각각 위치하며 따라서 User와 Item의 Unique한 level 수가 Matrix의 차원이 된다. 이 때 한 User가 같은 식당을 여러 번 방문하여 평점이 복수로 남겨진 경우가 일부 발견되었는데 이는 모두 평균 값으로 통일 시켜서 한 User가 반드시 한 Item에 평점 하나를 남기는 식으로 바꾸어서 Matrix의 값으로 사용하였다. 예를 들어 A라는 사람이 b식당에 세 번 방문하여 3.5, 3.2, 3.2 점을 남겼다면 User / Item Matrix의 [A, b] 값에는 세 값의 평균인 3.3점이 기입되는 식이다. 모델링 중인 미나토 구의 User / Item Matrix의 차원은 Lunch : 15585 X 6094, Dinner : 16602 X 7198 이다.

8) Train / Test set 설정

현재의 데이터는 최근 5년간의 리뷰/평점 데이터와 식당 정보 데이터로 이루어져 있다. 앞으로 모델이 서비스로 제시될 경우 미래에 User가 Item에 평가하게 될 예측 평점을 구하게 되므로, 과거의 식당 평점을 예측하는 것보단 최신의 식당에 대한 평점을 예측하는 것이 합리적이고, 실용적이라고 판단했다. 따라서 모델 평가를 위한 데이터 Split 역시 시계열적으로 시행하였다. 최종 모델 평가를 위한 Test Set으로는 데이터의 가장 최근 6개월의 리뷰 데이터를 사용했으며, 기간은 2017년 1월 1일부터 6월 30일까지다. 그 이전까지의 데이터에서는 80%가량을 Train Set으로 하여 모델 학습에 사용되었으며 20%를 Validation Set으로 Split하면서 과적합 방지 및 파라미터 튜닝을 진행하였다.

2.2 User-to-Item Modeling

2.2.1 Deep Neural Network (with Entity Embedding)

(github : Woohyun – usertoitem – usertoitem_DNN_code 부분)

1) 기본 원리

이 모델의 목적은 i번째 User가 j번째 식당을 이용할 때의 예측 평점을 구하는 것이다. 즉, 하나의 OBS는 i번째 User와 j번째 식당에 대한 정보가 된다. 이에 사용되는 Input Features는 User-side Normalized Vector, Item-side Normalized Vector, 그리고 각 User와 Item에 대한 Contents 정보다. 전처리 과정을 거치고 User-Item Rating Matrix를 구한 바 있다. row는 User, column Item(식당)인 다음과 같은 Data를 생각하면 될 것이다. 먼저 수행해야할 부분은 위 정보들을 조합하여 DNN의 Input으로 사용될 Vector형태를 만드는 것이다. 우선 학습속도와 지역최소점 회피를 위해 Rating Vector를 각 Side로 정규화해주고 각 User와 Item에 해당하는 Contents를 불러온다.

	식당1	식당2	…	식당j	식당 j+1
User1	3.5	0	0	0	4.1
User2	0	0	0	0	3.3
User3	0	5	4.4	0	0
…	3.9	4.5	0	3.8	0
User i	4.5	0	0	4.2	0
User i+1	3.4	5.0	0	0	0
User i+2	0	0	3.6	3.0	0

User-side Normalized Rating Vector

1.16	-0.73	-0.73	1.03	-0.73

User i's Contents

나이대	성별	방문 월	흡연가능	주차가능	카드가능
		카테고리1	카테고리2	카테고리3	가격대

Item_i's Contents

Item-side Normalized Rating Vector

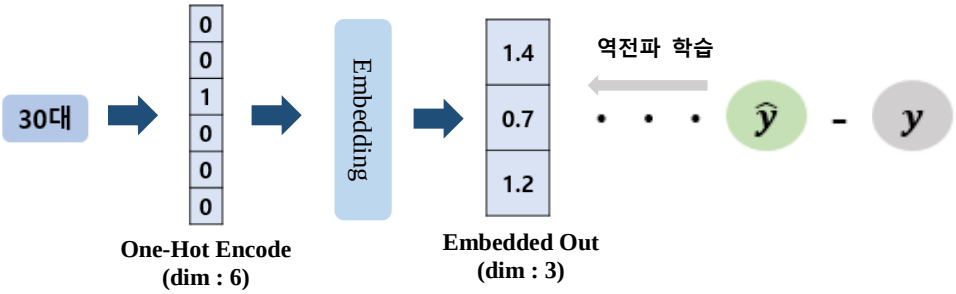
-0.79	-0.79	-0.79	1.12	1.32	-0.79	0.72

2) Embedding

평점 4.2를 y_{ij}라고 한다면 모델에 사용되는 Features X는 위 그림처럼 크게 User-Side Rating Vector, Item-side Rating Vector, Contents Vector의 세 그룹으로 나뉘게 된다. 그러나 이 세 그룹을 바로 Concate하여 모델의 Input으로 사용하기는 어렵다. 우선 Rating Vector의 차원이 매우 크기 때문에 상대적으로 차원이 작은 Contents Features의 영향력이 작아질 수 있다. User와 Item의 속

성과 직접적인 관련이 있는 Contents Features이니만큼 어느정도의 비중은 고정시켜줄 필요가 있을 것이다. 또한 실제 Rating Matrix는 위의 그림보다 훨씬 Sparse하다. 따라서 Rating Vector 값의 대부분이 동일하며 이를 그대로 Input으로 사용하기 보다 더 적은 차원으로 함축시켜서 사용하는 것이 타당할 것이다. 뿐만 아니라 Contents Features도 모두 Categorical 변수로, 특히 카테고리2, 3, 가격대 등은 그 Category의 level을 몇 십개까지 보유하고 있다. 이를 곧바로 One-hot Encoding하여 사용하게 되면 차원이 과도하게 커져 효율적인 학습이 저하될 수 있다. 이러한 이유로 인해 최초 Input Vectors는 concat되기 전 Embedding의 과정을 거친다.

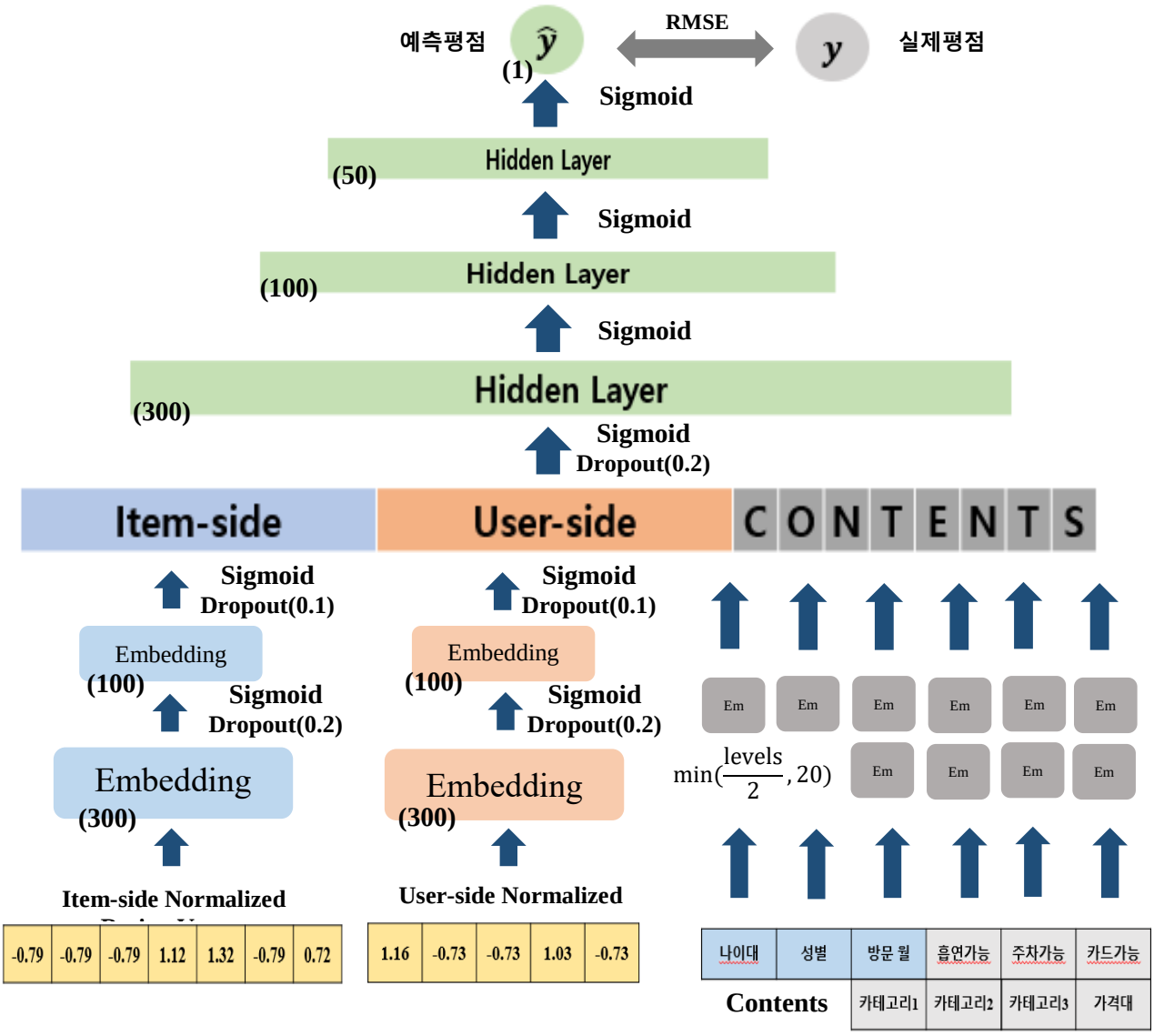
Keras에서 제공하는 Embedding 함수를 통해 현재 Input의 차원과 선호하는 Output차원을 조정해나갈 수 있다. 이 때 Categorical 변수들은 우선 One-hot Encoding 되어 입력되며 Hidden Layer를 통해 적절한 차원으로 축소된 형태로 나오게 된다. 이 과정에서 사용되는 Weight는 후에 최종 신경망을 통한 $\hat{y}$과 y간의 error를 통해 역전파로 학습된다. (이 경우. 예측평점과 평점사이의 RMSE) Embedding Output 차원의 값을 정하는 하나의 규칙은 없지만 일반적으로 따르는 $\min(\frac{\text{levels}}{2}, 20)$ 을 기준으로 +- 2차원 정도로 바꿔가면서 튜닝했다. 예를 들어 13개의 level을 가지고 있던 '가격대' 변수는 13/2 ≈ 6의 차원으로 Embedding 되었다. 가장 큰 차원이었던 '카테고리3'의 경우 93 차원을 20차원으로 축소시켰다. 시각화를 위해 '나이대'변수의 '30대'라는 level이 Embedding 되는 과정을 도식화하면 다음과 같다.



Categorical 변수 Embedding에서는 기대되는 또다른 효과도 있는데, 바로 level사이의 유사성이 고려된다는 점이다. 우리 데이터의 경우 '방문 월'에서 5월로가 6월이 5월과 11월보다 유사한 값으로 Embedding되면서 고려될 수 있고, '가격대'가 비슷한 level끼리 유사한 값을 띌 수 있다. Embedding은 이후 NNMF 모델에서도 사용되기 때문에 이후에 한 번더 살펴보도록 하겠다.

3) DNN

Embedding을 거친 각 Features는 하나의 vector로 concat되어 DNN의 Input으로 활용된다. 3개의 층을 거쳐 최종 아웃풋 노드는 하나의 값을 가지며, 이것이 i번째 User가 j번째 식당을 이용할 때의 예측 평점($\hat{y}$)이 된다. 이는 위에 설명했듯, 실제 평점(y)와의 Root Mean Square를 감소하는 방향으로 역전파 학습된다. 이를 간단한 도식으로 설명하면 다음과 같다.



4) Parameter Tuning

위 그림에서 각 Layer의 아래 괄호 안에 쓰여진 숫자는 해당 Layer를 거쳐 나오는 값의 차원 수이며, Dropout 옆에 쓰여진 숫자는 Dropout Ratio다. 해당 수치를 포함하여 Batch_size, Epochs, Activation Function 역시 두 차례의 Grid Search를 통해 튜닝 되었다. (자세한 결과는 CSV파일을 참고바랍니다.) 위의 하이퍼파라미터는 모두 Lunch 모델 튜닝의 결과이고 Dinner 모델을 튜닝한 결과 대부분의 파라미터는 유사하나 Dropout Ratio가 모두 0일 때가 가장 최적인 결과를 보였다.

Lunch Model Parameter Top3

batch_size	epochs	firstdrop	seconddrop	active	Firstnode (Emb1)	Secondnode (Emb2)	thirdnode	rmse
800	15	0.2	0.1	sigmoid	300	100	300	0.3254
800	10	0.2	0	sigmoid	500	200	300	0.3256
800	15	0.2	0	sigmoid	300	100	300	0.3263

Dinner Model Parameter Top3

batch_size	epochs	firstdrop	seconddrop	active	Firstnode (Emb1)	Secondnode (Emb2)	thirdnode	rmse
800	15	0	0	sigmoid	300	200	100	0.3353
800	15	0	0	sigmoid	500	200	200	0.3362
800	15	0	0.1	sigmoid	300	100	100	0.3380

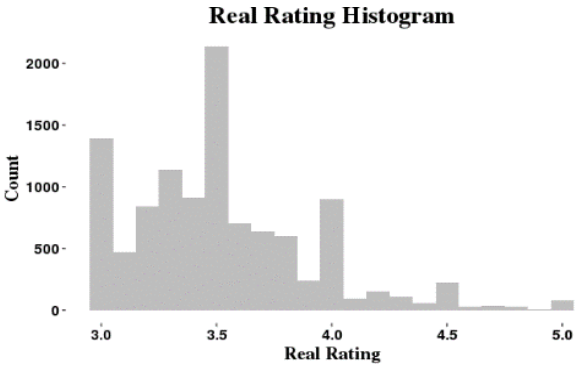
튜닝에 사용된 파라미터와 Grid range는 각각 다음과 같다

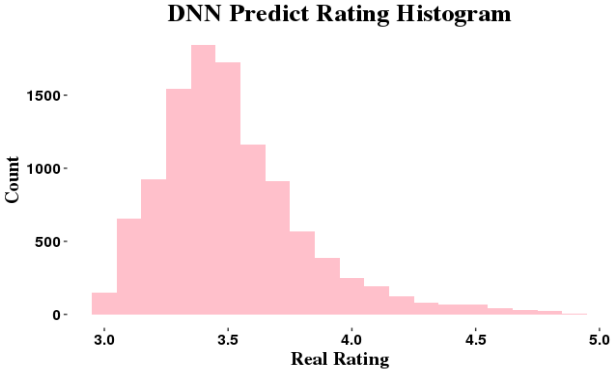
```
# 1차 Grid Search
grid = {'batch_size': [300,500,800,1000],
         'epochs': [5,10,20],
         'active': ['sigmoid','tanh','relu'] }

# 1차 결과를 바탕으로 일부 파라미터 고정 후 2차 Grid Search
grid2 = {'batch_size': [800],
         'epochs': [10,15],
         'firstdrop': [0,0.2],
         'seconddrop': [0,0.1],
         'active': ['sigmoid'],
        'firstnode': [300,500,800],
         'secondnode': [100,200],
         'thirdnode': [100,200,300]
        }
```

또한 최초에는 Rating Vector를 Embedding할 때 Categorical 변수와 같이 하나의 단일 층만 사용했는데, 기본적으로 처음 Input에서의 차원이 크다 보니 한 번에 차원 축소가 잘 이루어지지 않은 듯 했다. 이후 하나의 은닉층을 추가로 설정하여 두 개의 층으로 차원 축소를 해보니 학습 속도는 소폭 상승한 것에 비해 Validation Error가 상당히 감소하였고, 층을 하나 더 추가하자 학습 속도는 대폭 상승하고 Error는 거의 변함이 없었기 때문에 최종적으로는 위 그림과 같이 두 층을 사용하게 되었다.

최종 모델이 완성되고 Hold out 해두었던 Test set(최근 6개월)에 대한 Predict값의 분포는 아래와 같다. 실제 평점과 비교해보면 3.0부근에 대한 예측이 어려움을 알 수 있다. RMSE값은 0.344이며 구체적인 모델 간 수치 비교는 Evaluation 파트에서 살펴보도록 하겠다.



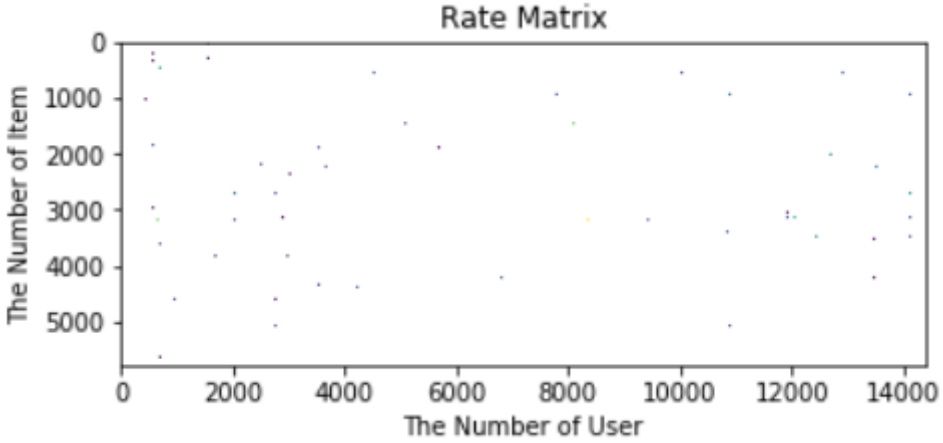


2.2.2 Neural Network Matrix Factorization (NNMF)

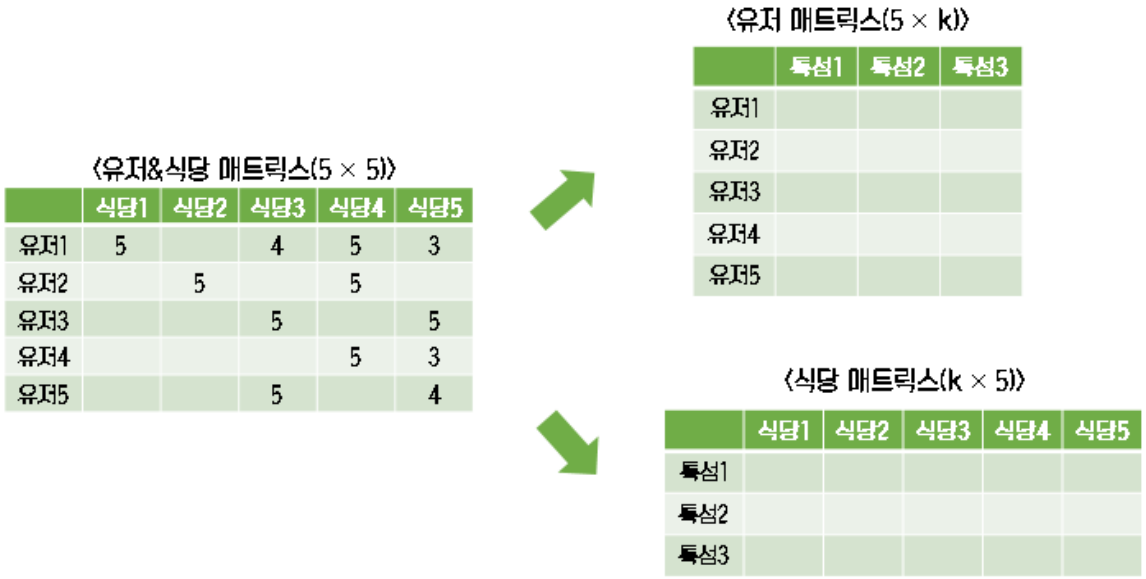
(github : Daeryong – modeling – neural network matrix factorization부분)

1) matrix factorization이란?

non-negative matrix factorization(비음수 행렬)과 neural network matrix factorization 모두 줄여서 nnmf라고 칭하는데 여기서 nnmf는 neural network matrix factorization을 의미한다. Matrix factorization은 추천시스템에서 이미 널리 사용되는 알고리즘이다. PCA(principal component analysis), SVD(singular value decomposition)의 방식과 마찬가지로 차원 축소를 통해 불필요한 계산을 축소하고, 노이즈를 제거함으로써 효율적인 추천이 가능하다. 특히나 데이터가 sparse한 경우 기존의 협업필터링을 적용하면 여러 문제가 발생하는데 이런 문제를 효과적으로 해결이 가능하다.



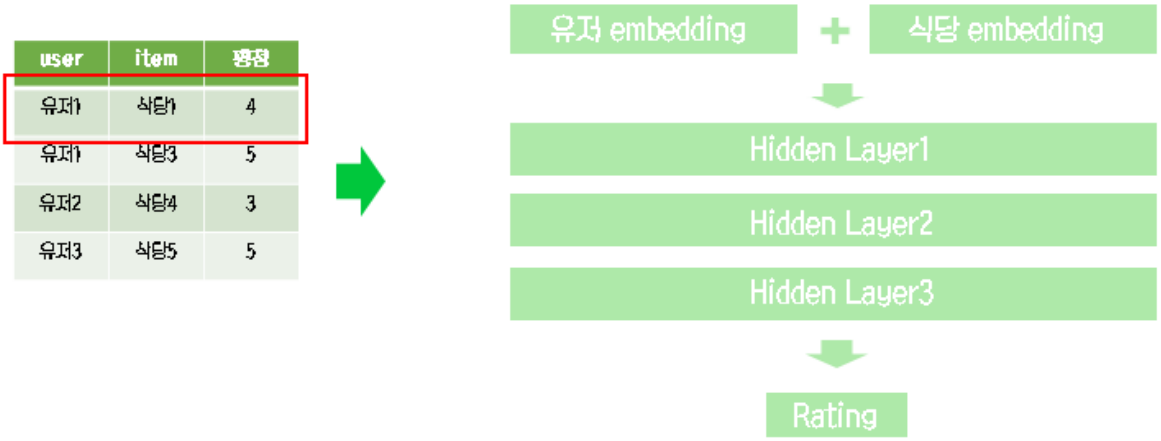
위의 그림은 유저와 식당 매트릭스를 시각화한 것이다. 그림 안의 파란색 점이 의미하는 것은 그 위치에 해당하는 유저가 그 위치에 해당하는 식당에 평점, 즉 리뷰를 남겼음을 의미한다. 한눈에 볼 수 있듯이 유저와 식당의 수가 많다 보니 굉장히 sparse함을 알 수 있다. 15760 * 6108 매트릭스에 값이 채워져 있는 부분은 73121개로 이를 퍼센트로 나타내면 0.01%도 되지 않는 정도의 수치이다. 이렇게 데이터가 sparse한 경우 기존의 협업필터링을 이용할 경우 상당히 많은 문제를 야기시키는 반면 Matrix factorization 방식은 sparse한 매트릭스를 낮은 차원의 dense한 매트릭스로 분해를 하고 이를 통해 더욱 정확한 추천이 가능하게 한다. 이 알고리즘에 대해 간단히 살펴보기 위해 5명의 유저와 5개의 식당이 존재한다고 가정해보자. 아래의 표와 같이 해당 유저가 식당에 리뷰를 남겼다면 숫자가 존재하고 아니라면 숫자가 존재하지 않을 것이다.



위의 그림에서 볼 수 있듯이 유저 & 식당 매트릭스를 각각 유저 매트릭스와 식당 매트릭스로 분해했음을 알 수 있다. 따라서 유저매트릭스의 행은 유저의 수와 같고 열은 유저들이 갖는 특성을 의미한다. 마찬가지로 식당매트릭스의 열은 식당의 수와 같고 행은 식당들이 갖는 특성을 의미한다. 따라서 유저 매트릭스의 열과 식당 매트릭스의 행의 수는 무조건 같아야만 한다. 행렬을 분리하는 과정에 대해 간단히 소개하자면 먼저 유저 행렬과 식당 행렬에 임의 값을 채우게 된다. 그리고 두 매트릭스를 곱하여 원래의 매트릭스와 비교를 함으로써 원래의 평점과 예측된 평점의 rmse를 구할 수 있으며, 이 과정을 원래의 매트릭스와 유사해질 때까지 반복하는 것이다. 분해된 매트릭스를 바탕으로 유저 매트릭스의 i번째 행과 식당 매트릭스의 j번째 열을 곱함으로써 i번째 유저의 j번째 식당에 대한 평점을 예측할 수 있다.

2) neural network matrix factorization

nnmf 알고리즘 역시 mf와 원리는 같다. 다만, 한가지 큰 차이가 존재하는데 nnmf의 경우 행렬의 곱을 이용하지 않는다는 점이다. 행렬의 곱 대신 Neural network의 특성을 이용하여 i번째 유저와 j번째 식당의 정보를 하나의 벡터로 펼친 뒤 target에 평점을 두고 이를 학습시키는 원리이다. 학습 과정에 대해 자세히 살펴보도록 하겠다.



먼저 유저의 수와 식당의 수만큼 각각 임베딩을 시켜 매트릭스를 만든 후 그 안에는 임의의 값을 생성한다. 첫 번째 데이터에서 유저1이 식당1에 평점을 남겼으므로 유저 매트릭스의 1행과 식당 매트릭스의 1열의 정보를 가져와 이를 합쳐서 input으로 두고 4인 평점과 비교를 하여 학습시킨다. 즉, 각각의 사람과 식당을 하나의 카테고리로 설정하는 것과 같으며 이의 과정은 임베딩을 시키는 작업을 통해 진행된다. 또한, 이 경우 매트릭스의 곱을 이용하는 것이 아니므로 user의 factor수와 item의 factor수를 일치시킬 필요가 없다. 그렇다면 이 factor의 수는 어떻게 구하는가? 여러 방법이 존재하나 여기서 이용한 방식은 factor의 수 역시 우리가 설정하는 hyper parameter라고 간주를 한 후, 이 역시 grid search를 통해 최적화시키는 방법이다.

추가적으로 행렬의 곱이 아니라 하나의 벡터로 입력하기 때문에 일단 mf모델에선 사용하지 않는 유저의 정보(성별, 나이), 식당의 정보(카테고리, 카드 & 흡연 & 주차 가능 여부) 역시 추가적으로 이용이 가능하다. 유저1이 식당1에 4의 평점을 남겼기 때문에 그 벡터를 학습시킬 때 추가적으로 유저1의 정보와 식당1의 정보를 추가하는 과정을 진행하였다. 카드, 주차, 흡연 가능여부와 식당의 카테고리1,2,3, 음식 가격과 같은 식당 정보와 성별과 나이 같은 유저 정보를 추가하였으며 앞선 DNN 모델의 임베딩과 동일하게 추가하였다. 임베딩 과정을 진행하는 이유는 한 변수 내에서 비슷한 특성을 가진 요소들을 유사하게 재배치함으로써 더 좋은 성능을 낼 수 있기 때문이다. 이를 도식화하면 다음과 같다.

임베딩된 결과를 살펴보면 먼저 식당 대분류에 대해 살펴보겠다. 식당 대분류는 식당, 여관&오베르, 주점, 카페&디저트 이렇게 4가지로 구성되어 있다. 임베딩된 결과는 아래와 같은데 이를 이용하 여 코사인 유사도를 구해보면 다음과 같은 결과가 나온다.

〈식당 카테고리: 식당, 주점, 카페&디저트, 여관&오베르〉

Rest_Type	factor1	factor2
식당	-0.014667	-0.086734
여관,오베르	0.010406	0.001016
주점	-0.007033	-0.053833
카페,디저트	0.046784	-0.11058

Rest_Type	Rest_Type	유사도
식당	여관, 오베르	-0.26
식당	주점	0.99
식당	카페, 디저트	0.84
여관, 오베르	주점	-0.22
여관, 오베르	카페, 디저트	0.29
주점	카페, 디저트	0.86

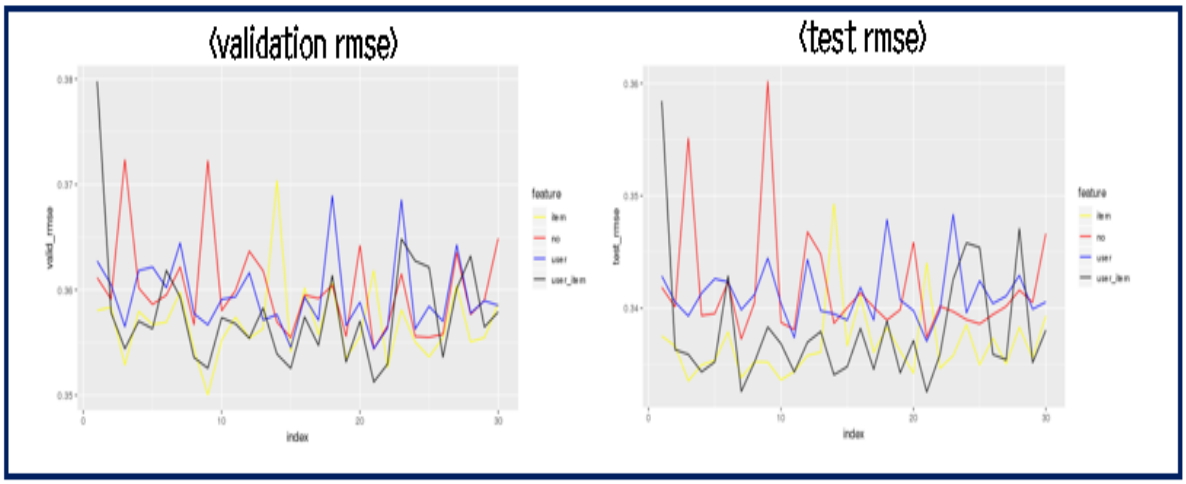
실제로 대부분의 식당들이 식당과 주점의 카테고리를 가지고 있는 경우가 많았는데 위의 결과 역시 식당과 주점이 상당히 유사함을 보여준다. 이와 같은 embedding 과정은 성별과 나이에 존재하는 무수히 많은 NA를 대체하는데도 효과적이었다. 아래의 결과처럼 성별을 2차원으로 임베딩시켰는데 NA의 경우 여성과 굉장히 유사하게 나옴을 알 수 있다.

〈성별 : 남자, 여자, NA〉

	factor1	factor2
na	-0.033573	0.044235
female	-0.016501	0.006057
male	0.009851	0.044668

Gender	Gender	유사도
NA	Female	0.84
NA	Male	0.64
Female	Male	0.13

모든 변수들을 무조건적으로 활용한 것이 아니라 총 30번의 validation set을 구성한 뒤 동일한 validation set에 대하여 모든 변수를 넣은 경우, 유저의 정보만 넣은 경우, 식당의 정보만 경우, 아무런 정보를 넣지 않은 경우를 비교해보았다.



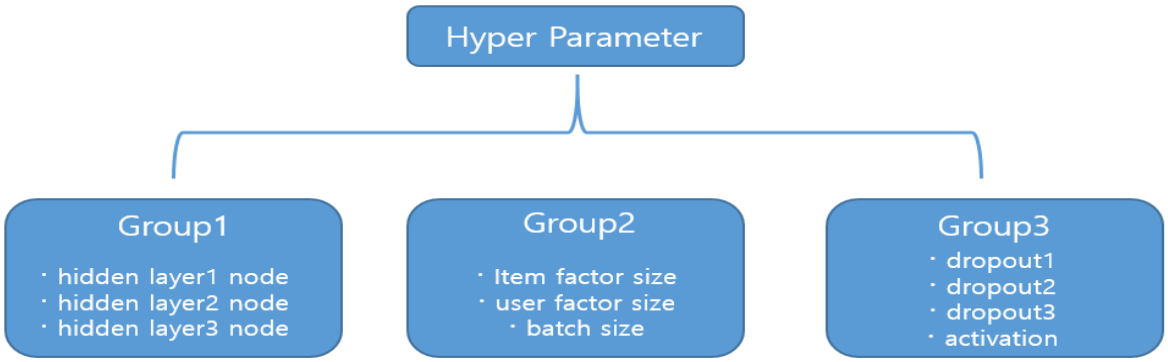
feature	Valid rmse	Test rmse
Item & User	0.356	0.335
User	0.359	0.341
Item	0.356	0.336
Not	0.36	0.341

위의 결과값은 30개의 서로 다른 validation셋에 대해 item정보만 넣은 경우, user정보만 넣은 경우, 모든 정보를 넣은 경우, 아무런 정보를 넣지 않은 경우의 rmse를 계산하여 평균낸 것인데 결과에서 볼 수 있듯이 성능에 굉장히 큰 영향을 끼치진 않음을 확인할 수 있었다. 이는 matrix factorization의 과정에서 item과 user의 정보가 latent factor(잠재변수)로서 이미 어느정도 반영이 된 상태이므로 추가적인 정보를 넣어도 큰 영향을 미치진 않는다는 것을 확인하였다. 허나 아무런 정보를 이용하지 않는 경우보단 변수를 추가했을 때의 성능이 조금이나마 좋으므로 뒤의 과정에서 item과 user의 정보를 모두 이용하여 진행하였다.

추가적으로, 추천시스템에선 앞서 말한 non negative matrix factorization (비음수행렬분해)를 일반적으로 이용한다. 이는 매트릭스를 분해하는 과정에서 행렬의 값이 음수가 되지 않도록 제약조건을 설정하는 것인데 이유는 행렬 분해의 경우 정해진 답이 있는 게 아니라 최적화를 시키는 것이므로 모델이 더욱 빠르게 학습하는 효과를 가진다. 허나, 이 경우 non negative 제약조건을 이용할 경우 성능이 오히려 악화되었으며 추가적으로 모델을 학습하는 시간이 그리 오래 걸리지 않아 non negative constraint는 별도로 이용하지 않았다.

3) parameter tuning

Grid search를 이용하여 일반적인 hyper parameter를 tuning 하였는데 앞서 말했듯이 식당과 유저의 임베딩 차원 역시 tuning을 할 필요가 있었다. 모든 parameter 조합을 한 번에 조정하기엔 어려움이 있어 3개의 그룹으로 묶어 파라미터를 튜닝하였으며 그 그룹은 아래와 같다.



위와 같이 그룹을 나누어 대략적인 parameter tuning을 한 후 세부적인 튜닝은 모든 조합을 이용하여 다시 튜닝을 하였으며 결과는 아래와 같다.

<Lunch parameter tuning>

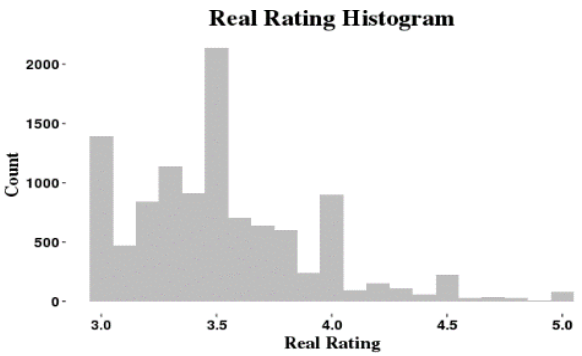
first_node	second_nc	third_node	batch_size	user_facto	rest_factor	dropout1	dropout2	dropout3	activation	activation	epoch	learning_r	valid_rmse	test_rmse
250	310	150	700	300	200	0	0	0	sigmoid		10	0.001	0.352703	0.336539
250	300	150	700	200	300	0.1	0	0	sigmoid		10	0.001	0.352879	0.335917

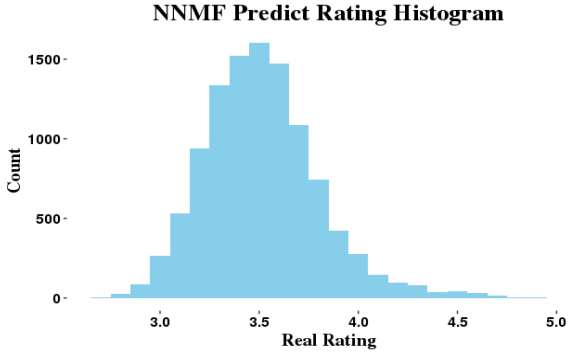
<Dinner parameter tuning>

first_node	second_nc	third_node	batch_size	user_facto	rest_factor	dropout1	dropout2	dropout3	activation	activation	epoch	learning_r	valid_rmse	test_rmse
200	300	50	100	300	200	0	0	0.05	sigmoid		10	0.001	0.389834	0.376397
200	200	50	500	300	200	0.1	0.1	0.05	sigmoid		10	0.001	0.392742	0.379862

Grid search를 통해 hidden layer의 activation function은 sigmoid를 사용했고, 최종 output의 activation function은 아무 것도 활용하지 않은 경우 가장 성능이 우수했다.

Hold out 해두었던 Test set(최근 6개월)에 대한 Predict값의 분포는 아래와 같다. RMSE값은 0.3435이며 구체적인 모델 간 수치 비교는 Evaluation 파트에서 살펴보도록 하겠다.





2.2.3 Model Ensemble 및 비교/평가

(github : Woohyun – usertoitem - Final test set result 부분)

지금까지 딥러닝을 활용한 두 가지 User-to-Item 모델링을 진행하였다. 두 모델 모두 어느 정도의 하이퍼파라미터 튜닝과정을 거쳤고 최종 모델이 확정되면서 Test Set에 대한 예측력 비교가 가능해졌다. 모델에 대한 평가는 크게 두 분야로 나뉘어 진행할 것인데, 우선 Test Set에 대한 기본적인 평가 Metric (RMSE, F-Measure, Accuracy)를 비교해본다. 이후 시뮬레이션 파트에서는 Test Set에서 100명의 Sample User를 층화추출하여 각 모델이 추천하는 식당들이 얼마나 유사한지, 혹은 얼마나 다른지 살펴볼 것이다.

평가 Metric에 대한 간단한 설명이 필요한데, 지금까지의 모델 학습에 실제 평점과 예측 평점의 RMSE(Root mean square)를 사용했다면 이와 더불어 실제로 이 User가 이 식당을 만족했는지, 혹은 만족하지 않았는지를 파악하는 Metric도 필요했다. 이에 평점 데이터의 50% Quantile이 정확하게 3.5임을 이용하여 평점이 3.5보다 크거나 높으면 만족, 작으면 불만족으로 가정하고 1, 0의 Binary Case로 y를 바꾸었다(Balanced data). 실제 만족여부와 예측된 만족여부를 통해 F-Measure, Accuracy를 구하고 비교할 수 있었다.

또한 각 모델에서 예측된 평점을 앙상블하여 오차가 더 낮아지는 지 여부도 확인할 수 있었는데, 구체적인 결과는 다음과 같다. (Test Set은 최근 6개월간의 식당 리뷰 10756 개로 동일)

Test Set Predict Results

	RMSE	F-Measure	Accuracy
DNN_entity_Embedding	0.3444970	0.6751805	0.6736705
NNMF	0.3356643	0.6668619	0.6826887
Emsemble	0.3294698	0.6766803	0.6838044

Confusion Matrix

Prediction \ Reference	0	1
0	3648	2409
1	1101	3598

DNN with Entity Embedding

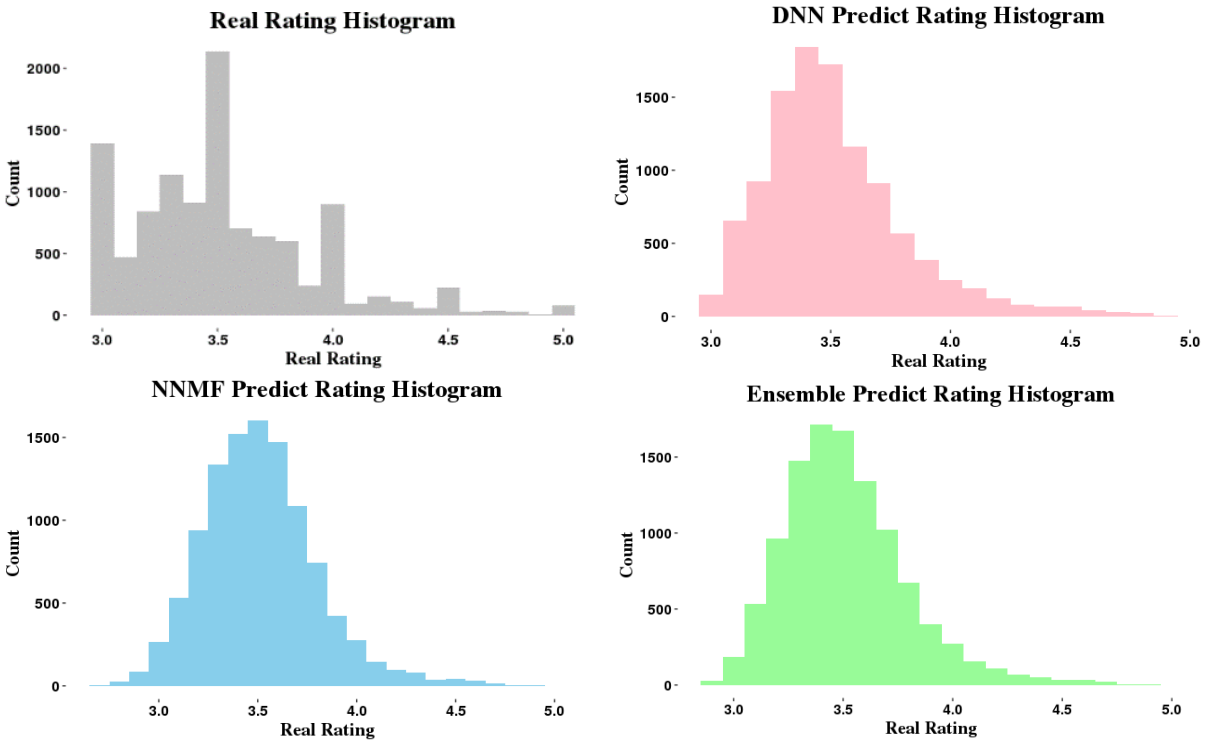
Prediction \ Reference	0	1
0	3416	2080
1	1333	3927

NNMF

Prediction \ Reference	0	1
0	3559	2211
1	1190	3796

Ensemble

서로 다른 알고리즘을 적용시켰음에도 불구하고 두 모델의 예측력은 거의 비슷하게 나타났다. 그러나 구체적으로 보면 DNN with Entity Embedding 모델은 상대적으로 0인 것, 즉 3.5 미만인 평점을 잘 맞춘 반면(3648), NNMF는 3.5 이상인 평점을 잘 맞추었다(3927). 앙상블 결과도 두 모델의 예측 평점이 평균내어지면서 3.5 미만을 맞춘 경우와 이상을 맞춘 경우가 대략 두 모델의 중간 정도로 나타내어지고 있음을 알 수 있었다. (각각 3557과 3796) 이는 예측 평점의 분포를 보면 한 눈에 파악할 수 있는다.



먼저 세 모델의 예측 평점에 대한 분포를 보면, DNN은 좌측으로 치우치게 예측했고 NNMF는 좀더 가운데로 위치하도록 예측하여 그 앙상블 결과는 약간 좌측으로 치우친 형태로 Mix 된 형태를 띄고 있다. 실제 평점의 분포가 다소 좌측으로 치우쳐 있음을 감안하면 두 모델의 앙상블 결과가 상당히 긍정적임을 알 수 있다. 위 표에서 확인할 수 있듯, RMSE값 역시 앙상블 모델에서 가장 낮았다.

그러나 세 결과 모두 실제 평점 분포에서 나타나는 큰 특징 중 하나를 잡아내지 못했다. 실제 평점에서 사람들은 3.0, 3.5, 4.0, 4.5와 같은 0.5 간격을 두고 나뉘어지는 '5배수 구간'에서 많은 평점을 매겼음을 한 눈에 파악할 수 있을 것이다. 이러한 현상은 사람 심리가 작용한 결과로 의심하고 모델링 전 EDA에서도 우려한 바 있었다. 특히 3.0지점에서 나타나는 평점과 예측값 사이의 큰 차이가 모델 Error에 상당부분 기여하고 있었다.

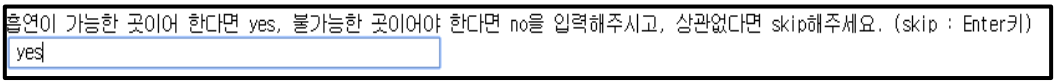
2.2.4 모델 Simulation 및 비교 (with 100 Sample)

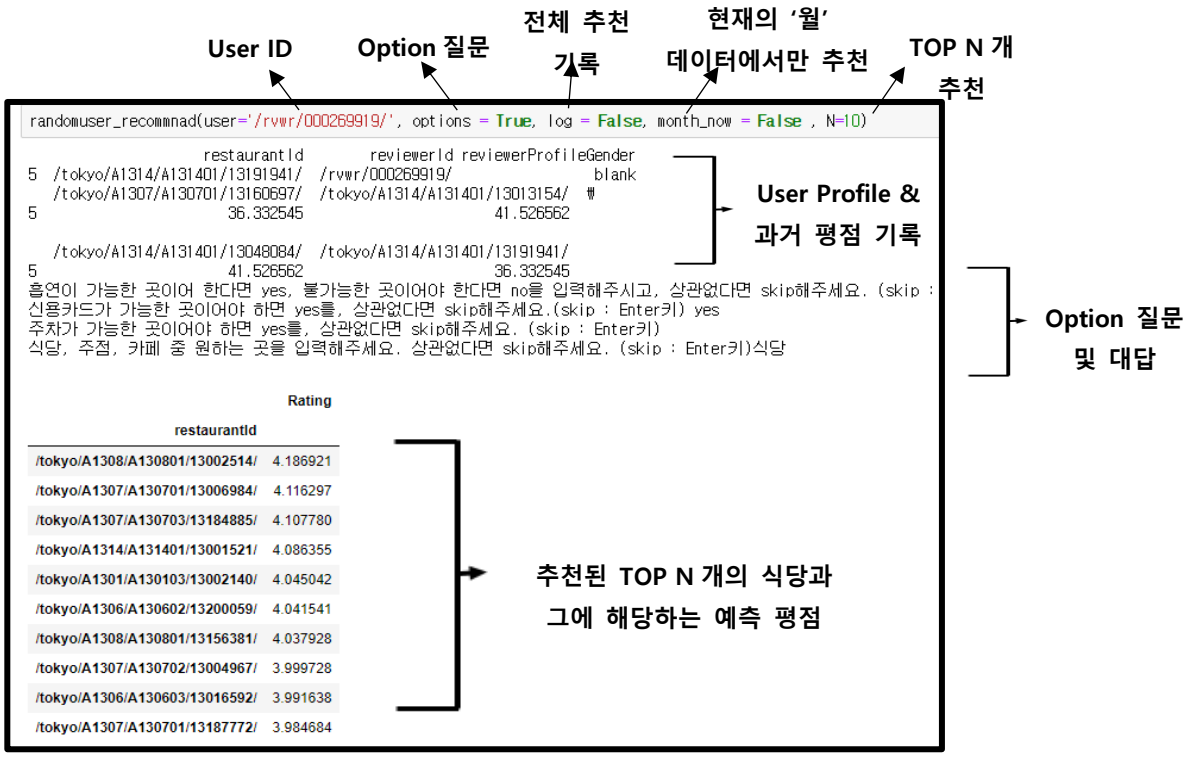
(github : Woohyun – usertoitem – usertoitem_DNN_code 부분)

분석 주제의 특성상 모델의 Error를 낮추는 것도 중요하지만 실제로 User에게 적절한 식당을 추천해주는 것인지에 대한 논의도 필요할 것이다. 물론 모든 유저를 다 Input해보면서 식당을 추천 받은 후 적절한지 파악하는 것은 시간상 불가능하기도 하거니와, 사실 해당 User가 방문하지 않은 식당에 대해서 얼마나 만족할지를 논하는 것이기 때문에 정해진 정답이 없고 평가도 주관에 의지하게 되기 마련이다. (정확한 답은 실제로 User가 식당을 방문한 후의 만족도일 것이다.)

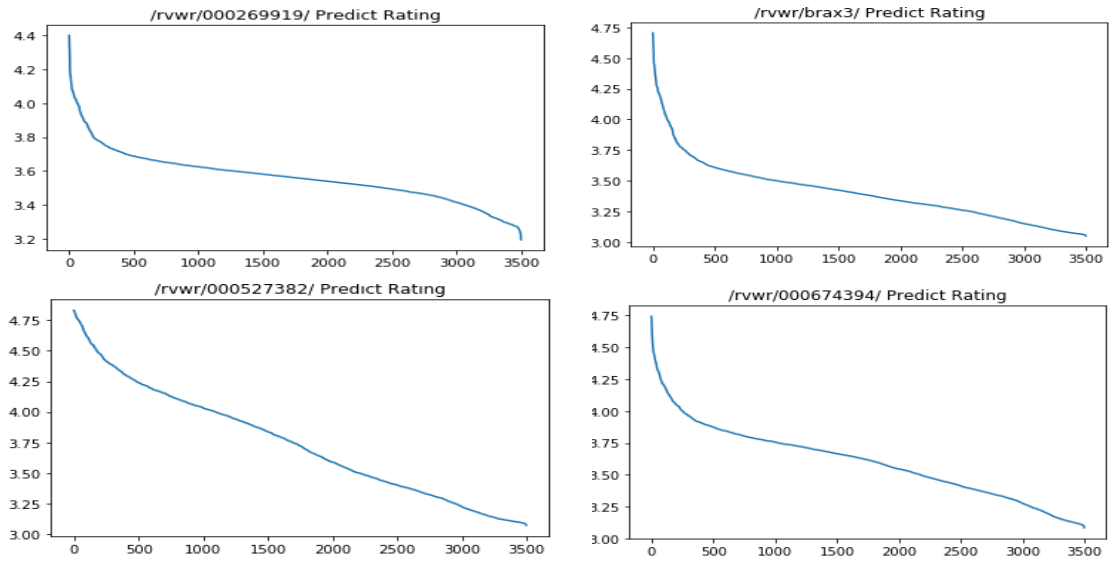
그러나 시뮬레이션을 통해 어느정도 모델의 유의미성을 평가해볼 수 있다. 또한 해당 모델의 유용성을 사람들에게 설득하려면 시뮬레이션이 가장 와 닿는 방법일 것이다. 두 모델 모두 같은 시뮬레이션 방식을 썼는데 비교적 간단하다. 한 명의 User가 자신의 ID를 입력하면 해당 ID에 해당하는 User 변수들은 그대로 두고 Item(식당) 변수만 바꿔가면 다른 모든 Item에 대한 예측 평점을 구하는 것이다. 이 중 가장 예측 평점이 높은 N개의 Item을 추천해주는 식이다. DNN 모델을 예시로 들면 User-side Rating Vector와 User관련 Contents(성별, 나이 등)은 그대로 두고 나머지 Item 변수들만 바꿔가면서 유저가 다른 모든 식당을 방문했을 때의 예측값을 구하게 된다.

시뮬레이션 모델에는 Option을 설정할 경우 User가 일부 변수들에 대하여 Filtering 할 수 있는 기회가 제공되도록 코딩되어있기도하다. 이는 User가 직접 Filter할 수 있는 변수를 활용하여 추천의 정확도를 높이기 위함이기도 하다. 예를 들어 Option의 처음 문구는 다음과 같다.





시뮬레이션 코드에 User id를 입력하여 위 과정을 거치고 도출된 식당들에 대한 예측 평점이 어떠한 양상을 띄는지 간단한 시각화로 확인할 수 있다. 몇 User에 대한 시뮬레이션 결과는 다음과 같다. (Plot Title : 'User id' Predict Rating)



한 User에 대한 다른 식당들에 대한 예측 평점 값은 평점 Ranking이 낮은 구간에서는 비교적 낮은 기울기로 상승하다가, Ranking이 상위권이 될수록 가파른 기울기를 보인다. 일반적으로 추천되는 Top N개의 식당들의 예측 평점 값들은 차이가 극명하게 나뉨을 확인했다. 모델이 잘못되었을 경우 대부분의 예측 평점이 동일하게 나타나는 현상일 발생할 수도 있는데 다행히 그러한 문제는 발견되지 않았다.

다음으로 두 모델이 추천해주는 식당이 어느정도 유사한지, 혹은 어느정도 다른지 확인하기 위해 Test Set에서 100명의 User를 추출하였다. 이 때 User가 남긴 평점의 개수가 결과에 영향을 주는지도 파악 해보기 위해 평점 수 별로 다음과 같이 층화추출 하였다.

Group1 : 평점 1개 남긴 User : 20명

Group2 : 평점 2개~10개 남긴 User : 50명

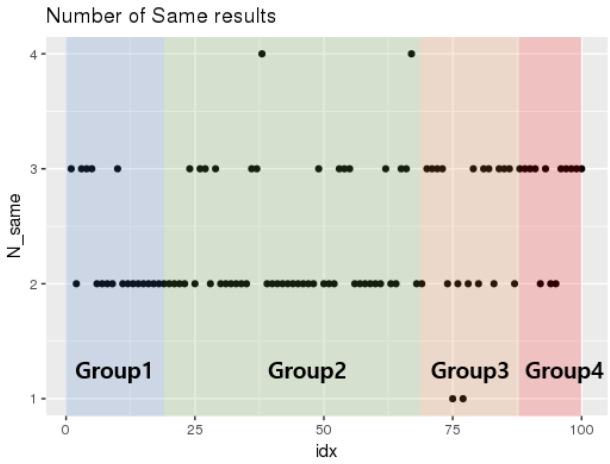
Group3 : 평점 11개~20개 남긴 User : 20명

Group4 : 평점 21개 이상 남긴 User : 10명

각 모델에서 추출된 100명의 User에 대해 각각 Top 10개의 식당을 추천받았다. 10개의 식당 중 두 모델 모두에서 추천받은 식당의 수는 다음과 같다.

Same	1	2	3	4	5개 이상
Count	2	58	38	2	0

흥미로웠던 점은 모든 User에서 항상 추천받는 식당이 겹쳤다는 것이다. 뿐만 아니라 대부분 2개~3개가 겹치고 동시에 5개 이상 겹치는 경우는 아예 없었으므로 두 모델이 아예 다른 결과를 내놓는 것은 아니면서도 적절히 다양성을 지니고 있음을 확인했다. 겹치는 것이 아예 없으면 모델의 신뢰성을 의심받을 수 있고, 너무 많으면 다른 모델을 사용한 의미가 퇴색되기 때문이다.



User의 리뷰수에 따른 결과의 변화를 파악하고자 하는 시도는 일부 유의미한 듯이 보인다. 리뷰가 적은 Group1, 2에서는 추천결과가 3개 겹치는 경우가 적은 반면 리뷰가 많은 Group3, 4에서는 더 많이 나타났기 때문이다. 2개가 겹치는 경우는 그 반대로 나타났다. 리뷰를 많이 한다는 것은 그만큼 Input의 정보가 많이 실려있다는 의미고 모델링 결과도 더 비슷하게 나올 수 있다고 해석했다. 그러나 겹치는 수 가 2개에서 3개 사이에서 머물고 있으므로 좀 더 극명한 차이를 보지는 못하여 아쉬움이 있었다. 샘플 수를 더 늘려보면 차이가 더 눈에 띄게 나타날 것으로 예상한다.

시뮬레이션 결과 비교를 통해 발견한 흥미로운 점이 있었는데, 두 모델이 공통된 식당을 추천하는 경우에는 대부분 추천된 식당이 리뷰가 굉장히 많고 총평점이 매우 높았다는 것이다. 아래의 그림은 Tabelog.com 접속하여 직접 공통적으로 추천된 식당들의 정보를 캡처한 것이다.

두 모델에서 모두 추천되는 위와 같은 식당들은 일명 '유명' 맛집이긴 하지만 우리가 지향하는 '개인 맞춤형 맛집 추천'과는 다소 거리가 있다. 유명 맛집들은 누구에게나 추천해줄 법한 것들이

기 때문에 추천시스템 모델이 굳이 없어도 추천이 가능할 것이다. 그러나 두 모델에서 다르게 추천한 식당들은 조금 다른 결과를 보였 다.

DNN

NNMF

두 모델에서 다르게 추천하는 식당들은 비록 평점이 높지 않거나, 리뷰가 많이 없더라도 다른 요인들에 의해 추천된 것임을 확인했다. 가격대나 메뉴 카테고리가 유사한 식당들이 추천되기도 하였고, 평점관련 변수가 영향을 주더라도 그것이 단순히 총평점이 높은 쪽으로 추천되지 않음을 확인했다.

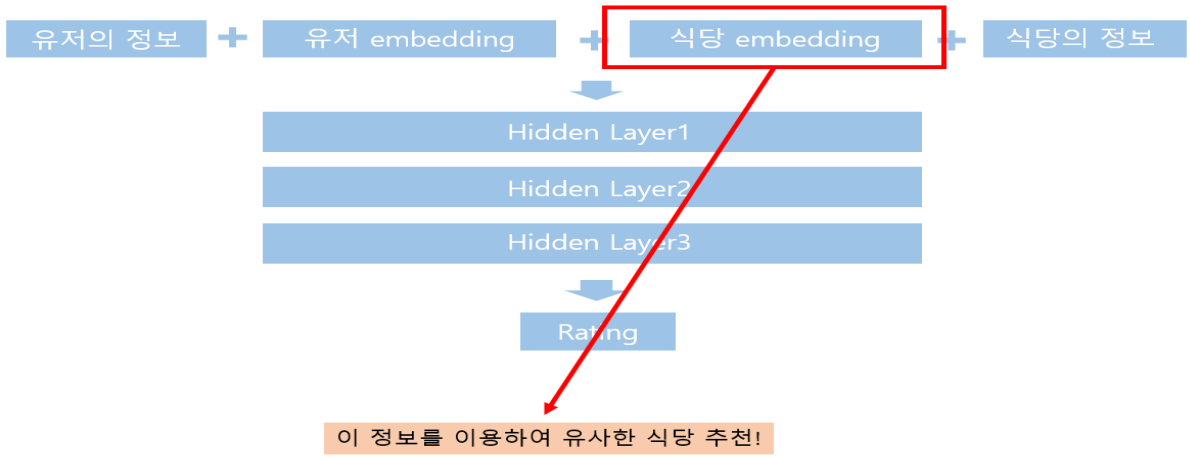
2.3 Item-to-Item Modeling

2.3.1 Distance with Embedded Values

(github : Daeryon– modeling – item_to_item_recommendation 부분)

지금까지의 알고리즘은 유저에게 식당을 추천하는 방식이었다면 지금부터는 특정 식당과 유사한 식당을 추천하는 방식에 대해 살펴보려 한다. 이 과정을 진행하면서 가장 큰 문제점은 식당과 가장 유사한 식당을 추천을 한 후 이를 평가할 마땅한 평가 지표가 존재하지 않는다는 점이었다. 그 이유는 모델링에서 설정할 target(Y변수)가 존재하지 않기 때문이다.

먼저 소개할 알고리즘은 앞선 nnmf 모델에서 학습된 모델을 바탕으로 한다. Nnmf 모델에서 식당과 유저의 정보를 각각 하나의 카테고리로 여긴 후 임베딩하는 과정을 진행하였는데 이 임베딩된 정보를 이용한 것이다. 임베딩을 통해 특성이 유사한 객체들을 비슷한 정보를 갖도록 차원을 재배치한 것이므로 이 임베딩된 값을 이용하여 유사한 식당을 추천할 수 있다는 가정에서 시작하였다. 허나 한 가지 문제점은 이 임베딩된 값은 각각의 차원에 대해서 그 의미를 아는 것이 불가능하기 때문에 이를 무조건적으로 사용하기엔 위험하다고 판단을 하였고, 이 임베딩된 값이 의미가 있음을 증명한 뒤 이를 활용하려 한다.



이를 증명하는 방식은 다음과 같다. Item embedding된 값을 바탕으로 item 사이의 유사도 매트릭스를 구하고, 이를 통해 item based collaborative filtering(아이템 기반 협업 필터링)을 진행하는 것이다. 그리고 이를 전통적인 협업필터링(아이템 유저 매트릭스를 통해 유사도 매트릭스를 구하는 방법)과 결과를 비교함으로써 만일 전자의 성능이 후자의 성능과 비슷하거나 혹은 그 이상의 성능을 낸다면 이는 임베딩된 값이 item을 구분하는 충분한 근거가 된다고 생각하였다.

2) 정당성 확보

총 6108개의 식당에 대한 임베딩 정보(6108 X 300: 앞선 과정에서 item factor의 수를

	0	1	2	3	4	5	6	7	8	9
0	1.000000	0.186857	0.000000	0.000000	0.0	0.0	0.0	0.0	0.0	0.000000
1	0.186857	1.000000	0.102102	0.087202	0.0	0.0	0.0	0.0	0.0	0.000000
2	0.000000	0.102102	1.000000	0.046232	0.0	0.0	0.0	0.0	0.0	0.075008
3	0.000000	0.087202	0.046232	1.000000	0.0	0.0	0.0	0.0	0.0	0.102666
4	0.000000	0.000000	0.000000	0.000000	0.0	0.0	0.0	0.0	0.0	0.000000

	0	1	2	3	4	5	6
0	1.000000	0.185859	0.063571	0.184541	-0.057512	0.287358	-0.384533
1	0.185859	1.000000	0.036549	0.011444	0.066660	0.066992	-0.115676
2	0.063571	0.036549	1.000000	-0.021036	0.095420	0.098526	-0.018713
3	0.184541	0.011444	-0.021036	1.000000	0.034916	0.093578	-0.081725
4	-0.057512	0.066660	0.095420	0.034916	1.000000	-0.127311	0.077299

parameter tuning을 통해 300으로 설정)를 이용하여 코사인 유사도 매트릭스를 구한다. 또한, 비교를 위해 아이템 유저 매트릭스를 이용한 코사인 유사도 매트릭스 역시 구하였다.

위의 표는 기존의 유사도 매트릭스이고, 아래의 표는 임베딩 값을 기반으로 한 유사도 매트릭스이다. 기존의 유사도 매트릭스는 해당 식당에 대해 리뷰가 적은 경우 다른 식당과 유사도가 0이 되어버리는 한계를 갖고 있다. 반면, 임베딩된 값은 이런 문제점이 전혀 발생하지 않으며 굉장히 dense한 값을 갖게 된다. 그러나 한 가지 고려할 점은 임베딩 값에 음수가 존재하여 유사도 역시 음수 값이 존재한다는 점이다. 뒤에 아이템 기반 협업 필터링을 구현하는 과정에서 유사도에 음수가 존재하는 경우 가중 평균시 문제가 발생하기 때문에 min max scaling을 통해 음수값을 없애는 과정을 진행하였다.

	restaurantId	reviewerId	ratingTotalScore	result_cf	result_new_cf
15	1	891	4.0	3.57359	3.61001
26	1	5543	4.2	3.46714	3.50772
27	1	8147	3.8	3.40417	3.45311
31	1	1273	3.5	3.45086	3.36373
42	2	7075	3.6	3.748	3.80581

Result_cf는 전통적인 협업필터링, result_new_cf는 임베딩 정보를 이용한 방법이다. 이를 이용하여 똑같이 최근 6개월 test 데이터에 대해 rmse를 구하니 다음과 같았다.

```
math.sqrt(sum((test_sample["ratingTotalScore"] - test_sample["result_cf"]) ** 2) / len(test_sample))
```

```
0.3468933471283824
```

```
math.sqrt(sum((test_sample["ratingTotalScore"] - test_sample["result_new_cf"]) ** 2) / len(test_sample))
```

```
0.3258939718618064
```

위의 결과에서 볼 수 있듯이 임베딩 정보를 이용한 방식의 성능이 더욱 우수함을 알 수 있다. 그리고 한 가지 더 주목할 점은 다음의 결과이다.

	restaurantId	reviewerId	ratingTotalScore	result_cf	result_new_cf
119	4	10710	3.9	0	3.46791
142	7	55	3.3	0	3.34219
154	8	3530	4.4	0	3.82991
195	9	13789	4.5	0	3.45142
228	11	3979	3.5	0	3.7

위의 값을 보면 전통 협업필터링이 문제점이 극명하게 드러나는 부분이다. 예측 평점이 0임을 알 수 있는데 이는 식당 내에 리뷰가 적고, 그 식당이 다른 식당들과 유사한 식당이 없을 경우 전혀 평점을 추천하지 못하게 된다. 하지만 이와 다르게 임베딩된 값을 이용할 경우 이런 부분 역시 고려가 가능하며 예측 평점 역시 우수한 결과값을 기록하였다.

앞의 과정을 바탕으로 nnmf 모델에서 item embedding 정보가 충분히 식당을 구별할 수 있는 정보임을 증명하였고, 이를 이용해서 식당을 입력하면 그와 가장 유사한 특성을 가진 식당을 추천하려 한다.

2.3.2 Deep Neural Network (with Selected Data)

(github : Woohyun – itemtoitem – itemtoitem_DNN_code 부분)

이 모델의 목적은 i번째 식당과 j번째 식당 간의 예측 Distance를 구하는 것이다. 즉 하나의 OBS는 i번째 식당과 j번째 식당의 정보가 된다. 앞서 User-to-Item의 DNN 모델과 방식은 유사하나 Model을 Train할 Data의 특성이 조금 다르다. 우선 아래 그림과 같이 User / Item Rating Matrix를 통해 Item 간의 Euclidian Distance Matrix를 구해준다.

	식당1	식당2	식당3	식당4	식당5
User1	3.5	0	0	4.1	4.1
User2	0	0	0	3.3	3.3
User3	0	5	0	0	0
User4	4.5	0	4.2	0	0
User5	3.4	5.0	0	0	0
User6	0	0	3.0	0	0
User7	0	0	3.6	3.0	0

User – Item Rating

	식당1	식당2	식당3	식당4	식당5
식당1	0	7.75	6.77	7.21	6.56
식당2	7.75	0	9.47	9.31	8.81
식당3	6.77	9.47	0	7.4	8.2
식당4	7.21	9.31	7.4	0	3
식당5	6.56	8.81	8.2	3	0

Item – Item Distance

Distance Matrix는 식당의 Unique한 개수의 차원만큼 만들어질 것이다. 모델링 중인 미나토구의 Item / Item Distance Matrix의 차원은 6094 X 6094다. 일반적인 Item-to-Item에서는 이 Matrix를 이용하여 i번째 식당과 가장 거리가 가까운 식당을 찾아서 추천해주는 방식일 것이다. (예를 들어 식당1을 입력하면 식당5가 추천됨) 그러나 이와 같은 방식은 리뷰가 적어서 Rating Vector가 Sparse한 식당에 대해서는 유의미하지 않은 결과를 도출하게 된다. 예를 들어 User에 의한 Rating이 한 자리 수인 경우에는 최소 거리에 의한 추천이 거의 어려워진다.

이 모델의 기본 원리는 리뷰가 적은(Sparse 한) 식당들 간의 거리는 유의미하지 않다고 가정하고 리뷰가 많은 식당들 간의 거리를 통해 모델을 Train 시키는 것이다. 이를 통해 리뷰가 적은 식당들 간의 거리를 예측하여 기존의 방식을 보완하는 것이 최종 목표이다. 이를 위해 먼저 User에 의한 평가가 70개 이상인 식당들을 select한 후 해당 식당들 간의 거리 Vector만을 Train set으로 사용한다. 이 방식은 앞서 살펴본 User-to-Item DNN 모델에서 User/Item-side Vector를 처리한 방식과 동일하다. 위 예시에서는 식당1과 식당4가 리뷰가 많은 Train 식당으로 select 되었다고 가정해보자. 편하게 지칭하기 위해 row 방향을 Item1방향, column방향을 Item2 방향이라 하겠다.

	식당1	식당2	식당3	식당4	식당5
식당1	0	7.75	6.77	7.21	6.56
식당2	7.75	0	9.47	9.31	8.81
식당3	6.77	9.47	0	7.4	8.2
식당4	7.21	9.31	7.4	0	3
식당5	6.56	8.81	8.2	3	0

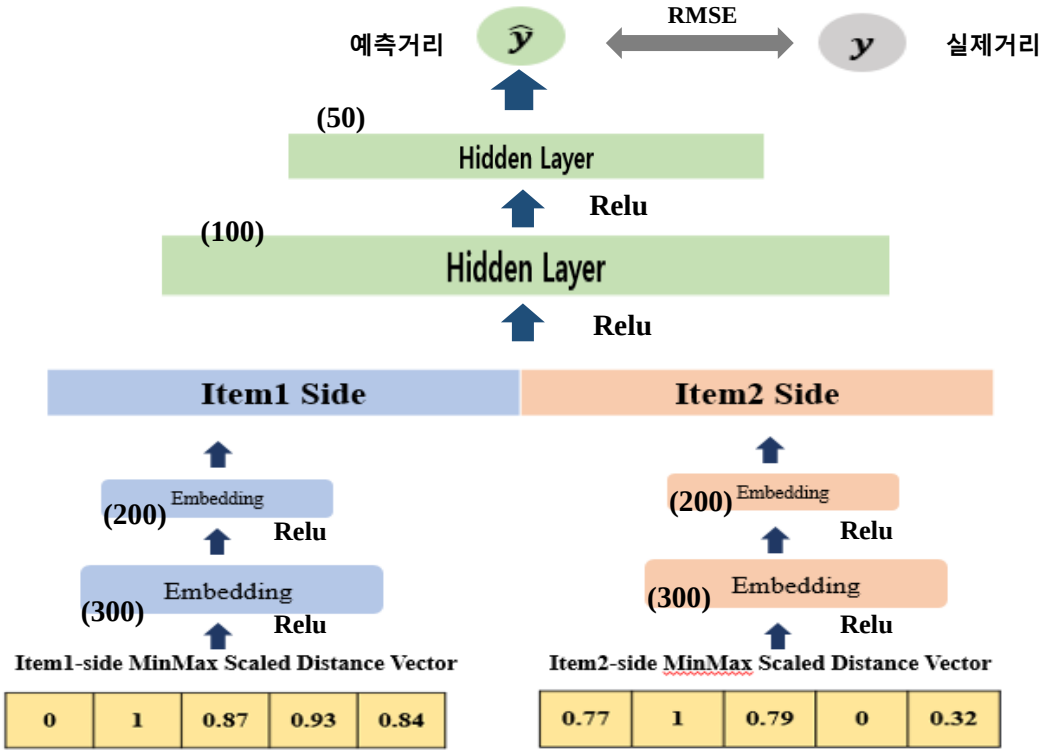
Item1-side MinMax Scaled Distance

0	1	0.87	0.93	0.84

Item2-side MinMax Scaled Distance

0.77	1	0.79	0	0.32

이후의 과정 또한 앞서 설명한 User-to-Item DNN 모델과 유사하다. 각 Distance Vector를 적절한 Output 차원으로 Embedding 시키고 Concat 시켜 Neural Net에 적합하는 것이다. 모델을 적합시킨 후에는 Predict의 과정만이 남는다. Sparse한 Rating Vector로 인해 거리 추정이 어려웠던 식당들을 위와 동일한 방식으로 Rating Vector Scaling 후 모델에 Input 해주면 학습된 Embedding을 통해 적절한 작은 차원의 Vector로 치환되고 보다 유의미한 거리 산출을 가능하게 할 것으로 기대한다.



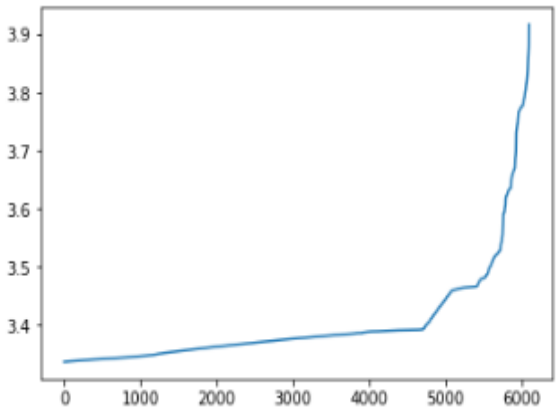
평점 수 70 개 이상으로 Select 된 두 Item(식당들)

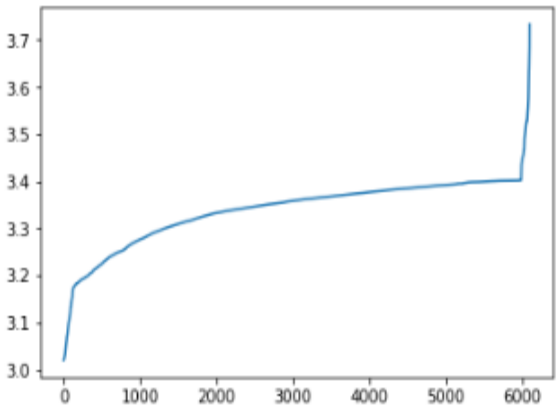
2.2.3 모델 Simulation

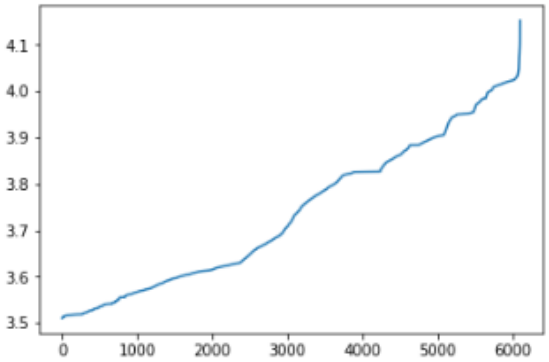
앞서 언급했듯, Item-to-Item 모델은 정답 Target이 없기 때문에 마땅한 평가 지표가 없다. 그러나 User-to-Item처럼 시뮬레이션은 해볼 수 있다. Embedding 값을 이용한 추천 모델에서는 결과로 나오는 새로운 유사도 행렬에서 가장 유사한 Item을 추천하게 된다. Selected Data를 활용한 DNN 모델에서는 User-to-Item과 유사하게 입력 Item의 변수를 고정시키고 다른 Item들의 값을 넣어가며 Output되는 예측 거리가 가장 짧은 Item을 추천한다. 이러한 방식으로 구현한 시뮬레이션 결과는 다음과 같다. (예측 dist는 log 변환된 결과이다.) 그 아래의 그림은 샘플링된 몇 Item의 예측결과(dist)를 Line Plot을 그린 것이다.

```
item_recommnad('/tokyo/A1301/A130103/13001507/')
```

	restaurantId	dist
5994	/tokyo/A1316/A131602/13166110/	3.070941
70	/tokyo/A1301/A130103/13012715/	3.070958
1502	/tokyo/A1306/A130603/13204376/	3.070997
3906	/tokyo/A1308/A130802/13171549/	3.071006
5117	/tokyo/A1314/A131402/13111693/	3.071007







Ⅲ. 결론

3.1 의의 및 향후 연구과제

이번 분석의 의의로 맛집 추천시스템에 전통적으로 사용되어왔던 Collaborative Filtering 이나 Contents Based Filtering이 아닌 Deep Learning을 사용하여 추천 모델을 구축했다는 점을 우선 말할 수 있겠다. Data의 형태가 일반적인 Predict 모델과 다르기 때문에 이를 Neural Network의 Input으로 어떻게 전처리하여 활용할 수 있는지에 대한 접근법도 제시했다.

User-to-Item에서는 DNN with Entity Embedding과 NNMF의 다른 두 알고리즘으로 접근하여 각각의 결과를 비교하면서 문제 해결의 방법이 유일하지 않음을 보였다. 최종 평가에서는 결과값을 앙상블한 것이 가장 적은 오차를 보이면서, 두 모델이 긍정적인 방향으로 종합될 수 있음을 보이기도 하였다. 또한 단순히 모델링으로 끝나지 않고 직접 시뮬레이션 함수를 만들어 두 모델이 추천하는 과정 및 결과의 해서하고 비교하였다. 고정된 샘플 유저를 뽑아서 두 모델이 적절한 다양성을 가지면서도 어긋난 방향으로는 가지 않음을 보이기도 했다.

Item to Item에서도 지금까지 거의 유일하게 사용되오던 Item간 Distance Matrix에서만 머물지 않고 Embedding된 값으로 계산하는 Distance, 리뷰가 많은 Item간의 Distance Vector를 활용한 DNN등 창의적인 접근법을 제시한 바 있다.

시간적인 문제 등으로 인해 해결하지 못한 문제도 있다. 우선 5개의 도시를 병합한 All City Model을 만들지 못한 것을 들 수 있겠다. 최초에는 Minato 구 만으로 기본 모델을 구축하고 이후 상위 5개 구 전체를 포함하는 모델을 만들 계획이었다. 그러나 Rating Matrix를 만들고 신경망의 Input을 구성하는 와중에 데이터의 Size가 너무 커져서 결국 Minato 구 모델, Shinjuku 구 모델 등, 상위 5개 도시를 각각 모델링하며 마무리하게 되었다. 가장 obs가 많았던 Minato 구를 기준으로 모든 파라미터를 튜닝함으로써 다른 도시에 최적화된 튜닝을 하지 못하기도 했다. 좀 더 빨리 대용량 데이터의 문제에 직면했다면 Spark를 사용하여 해결할 수도 있었을 문제라고 생각한다. 이후 기회가 된다면 지금까지의 모델링과정을 바탕으로 All City Model을 구현하고자 한다.

Item-to-Item에서는 알고리즘의 특성상 실제 y(실제 Distance)가 없기 때문에, 모델간 비교가 어렵지만 그럼에도 어떠한 평가 Metric을 제안하지 못한 것이 한계라고 할 수 있다. 이를 평가하기 위해서는 두 알고리즘을 사용한 Beta버전의 Item-to-Item 서비스를 만들어서 만족도 설문조사를 통해 가능할 것이다. 사실 서비스 이용 후기를 통한 만족도 조사는 User-to-Item 모델에서도 반드시 시행하여 모델의 직접적인 평가 Metric으로 활용해볼 필요가 있다.

3.2 참고문헌

1) Rim Fakhfakh , Anis Ben Ammar , Chokri Ben Amar

Deep Learning-based Recommendation: Current Issues and Challenges (2017)

(http://thesai.org/Downloads/Volume8No12/Paper_9-Deep_Learning_based_Recommendation.pdf)

2) SHUAI ZHANG , LINA YAO, AIXIN SUN, YI TAY

Deep Learning based Recommender System: A Survey and New Perspectives(2018)

(https://arxiv.org/pdf/1707.07435.pdf **)**

3) **https://github.com/entron/entity-embedding-rossmann/ (Code reference)**

4) https://arxiv.org/ftp/arxiv/papers/1503/1503.07475.pdf **(Matrix factorization)**

5) https://nipunbatra.github.io/blog/2017/neural-collaborative-filtering.html **(keras embedding code)**