

# Data Management and Analysis Project2

Team03 오태양, 김해찬, 하창우, 김민영

## I. INTRODUCTION

본 프로젝트에서는, 온라인 소프트웨어 유통망 사이트 A의 data에 기반하여 프로젝트 #1의 방법론을 활용, DB를 구축하고, DB mining 및 Automated Recommendation System의 구현을 목표로 했다.

이를 위해 프로젝트를 크게 3단계로 구분했는데 PART I에서는 다양한 parameter들에 대해서 Decision Tree를 생성하고자 했고, PART II에서는 bundle간의 연관분석을 진행하여 그 분포를 살펴보고자 했으며, 마지막 PART III에서는 다양한 알고리즘을 사용하여 사용자들에게 bundle을 추천하는 추천시스템 구현하고 그 효과를 비교분석하고자 했다.

## II. PART I: 의사 결정 나무

Part I는 사이트 A에서 Best item으로 선정한 아이템 기준에 대한 의사결정나무를 만든다. 파이썬을 활용하며, 의사결정나무 생성에 적합한 데이터로 가공하기 위하여 mysql-connector-python 라이브러리를 통해 MySQL에 query를 실행하고 sklearn 라이브러리를 활용해 의사결정나무를 만들며 graphviz 라이브러리를 사용해서 결과를 시각화 한다. 그 이후엔 의사결정나무를 더 개선된 형태로 만들 수 있는지 여러 파라미터와 측정 기준을 변경하며 다른 형태의 의사결정나무를 생성해보았다.

### A. R 1-1 : ‘item’ 테이블에 ‘best\_item’ 열 추가하기

먼저 BEST item을 활용하기 위해 item 테이블에 ‘best\_item’이라는 새로운 열을 추가했다. 이 과정에서 코드를 반복 수행 시 이미 칼럼이 존재할 경우 오류가 생겼기에 try&except 문으로 예러를 잡아주었다. ‘best\_item’의 데이터형은 0 또는 1인 TINYINT(1)의 도메인으로 설정했다. NULL값은 없도록, 기본값은 0으로 설정했다. 그 다음엔 주어진 ‘best\_item\_list.txt’ 파일을 줄마다 참조하여 ‘item’ 테이블에 ‘item\_id’가 일치하는 아이템이 있다면 해당 튜플의 ‘best\_item’ 열의 값을 1로 수정하는 과정을 거쳤다.

## B. R 1-2 : item에 대한 추가 정보 획득

그 다음은 BEST item 선정 여부의 변수로 활용될 item들에 대한 정보를 수집하는 MySQL 쿼리를 실행하였다. Id, 베스트 아이템 선정 여부, 평가 점수, 스펙의 수, 태그의 수, 사용한 유저 수, 인당 평균 이용 시간, 작성된 리뷰 수, 리뷰의 추천 수 총합, 리뷰의 평균 본문 길이를 포함하는 총 10개의 column을 확인가능한 view이며, 각각의 정보를 모을 수 있는 ‘Item’, ‘item\_specs’, ‘tag’, ‘user\_item’, ‘review’ 테이블에서 가공을 거친 뒤 ‘item’ 테이블에 대해 LEFT OUTER JOIN을 하는 방법으로 view를 생성하였다. 이렇게 생성된 결과물은 ‘cursor’ 클래스의 fetchall 메소드로 데이터프레임으로 변환된 뒤, 데이터프레임의 to\_csv 메소드를 통해 ‘DMA\_project2\_team03\_part1.csv’에 저장되었다.

## C. R1-3 : Decision Tree 모델 학습

R1-3에서는 R1-2에서 저장한 아이템들의 정보를 입력값으로 하여 본격적으로 의사 결정 나무 모델을 학습시켰다. 우선 scikit-learn 모듈의 DecisionTreeClassifier 메소드로 모델을 적합 시키기 앞서 데이터프레임 형태로 존재하는 데이터를 학습에서 입력 값과 분류 값으로 사용할 수 있도록 가공하는 과정을 밟았다. 데이터프레임에서 각각 입력 값과 분류 값에 해당하도록 데이터 열을 나누어 저장하였으며, 데이터프레임을 to\_numpy 메소드를 사용해 배열로 2차원 배열로 변환하였다. 그 다음은 학습을 위해 Scikit Learn 모듈의 tree 오브젝트에서 DecisionTreeClassifier라는 메소드를 사용하여 Decision Tree를 생성하였다. Decision Tree의 Node impurity criterion은 gini, entropy 두 가지를 사용하였으며 그 외에 파라미터는 min-samples\_leaf=10, max\_depth=5라는 주어진 수치만을 고정으로 사용하여 Decision Tree를 학습시켰다. Impurity 지표인 Gini와 Entropy는 각각 다음과으로 나타내어진다.

$$Gini = 1 - \sum_j [p(j|t)]^2 \quad (1)$$

$$Entropy = - \sum_j p(j|t) \log_2 p(j|t) \quad (2)$$

표현값과 수학적인 특성은 다르지만 두 값 모두 하나의 노드에 여러 분류 값에 해당하는 경우가 섞여 있을 경우 커지는 특성을 보이며, 의사결정나무 생성은 매 분류 노드에서 이 값을 최소화하는 방향으로 분류 기준을 정하게 된다.

### 1. Decision Tree-Gini

Gini로 생성한 Decision Tree의 구체적인 파라미터들은 다음과 같다.

|                                 |             |
|---------------------------------|-------------|
| <b>DT gini parameter</b>        |             |
| <b>ccp_alpha</b>                | <b>0.0</b>  |
| <b>class_weight</b>             | <b>None</b> |
| <b>criterion</b>                | <b>gini</b> |
| <b>max_depth</b>                | <b>5</b>    |
| <b>max_features</b>             | <b>None</b> |
| <b>max_leaf_nodes</b>           | <b>None</b> |
| <b>min_impurity_decrease</b>    | <b>0.0</b>  |
| <b>min_impurity_split</b>       | <b>None</b> |
| <b>min_samples_leaf</b>         | <b>10</b>   |
| <b>min_samples_split</b>        | <b>2</b>    |
| <b>min_weight_fraction_leaf</b> | <b>0.0</b>  |
| <b>random_state</b>             | <b>None</b> |
| <b>splitter</b>                 | <b>best</b> |

여기서 중요하게 다룬 파라미터들은 ‘criterion’, ‘max\_depth’, ‘min\_samples\_leaf’, ‘min\_samples\_split’, ‘splitter’이다. 우선 ‘criterion’은 impurity를 측정할 방법을 의미하며 Gini 와 Entropy 두 방법이 있다. 첫 모델의 경우 gini를 선택했다. ‘max\_depth’는 분류가 몇 번까지 일어나는가를 정해주는 기준이며, 지나치게 많은 분류 단계를 거치며 과적합되는 것을 막기 위한 설정이다. 이어지는 ‘min\_samples\_leaf’, ‘min\_samples\_split’ 둘도 과적합을 방지하는 파라미터들로 각각 하나의 샘플이 가질 수 있는 최소 샘플의 수, 또는 스플릿이 일어날 수 있는 샘플의 크기 기준을 의미한다. ‘splitter’의 경우 기본으로 ‘best’ 값이 할당되는데, split을 할 때 node impurity criterion인 지표를 기준으로 얻어지는 이득을 항상 최대화할 것인가, 아니면 다른 방법을 사용할 것인가를 설정할 수 있다. 여기서는 ‘best’로 설정하였다. 생성한 Decision Tree(Gini)는 제출 파일에 첨부하였다.

## 2. Gini-그래프 분석

Gini 지표를 활용한 Decision Tree Model의 경우 ‘sum\_of\_recommend’(리뷰 추천 수)를 기준으로 가장 먼저 분류되었다. 즉 리뷰 추천수가 여러 아이템에서 best\_item으로 분류되기 위해 공통적으로 가지는 특징 중 하나임을 유추할 수 있다. 리뷰 추천수가 6.5 이상인 628개의 표본들 중에선 ‘num\_of\_specs’(스펙의 수)가 4.5개 이상이면 best\_item으로 분류되었다. 평가 수가 많은 제품 중에서 다수의 스펙을 가지는 경우, ‘steam-achievement’과 같은 best\_item 들이

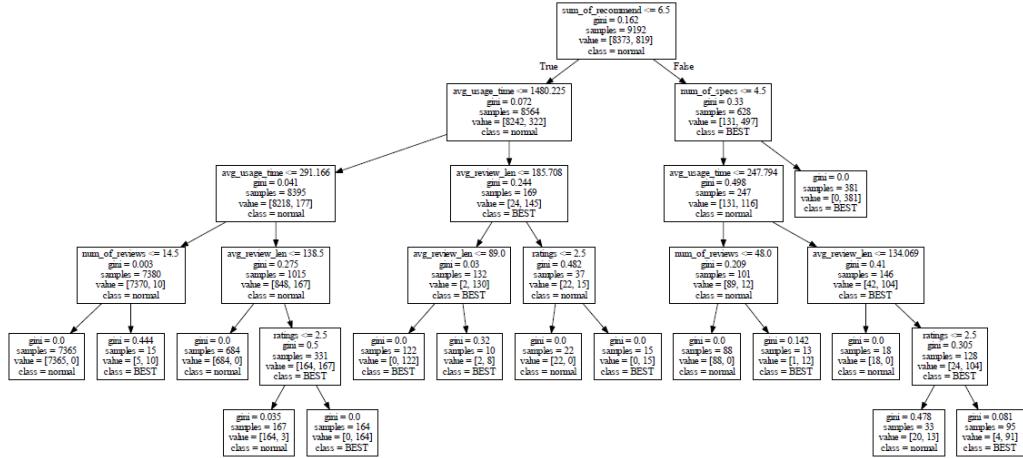


FIG. 1: Decision Tree - Gini

공통적으로 가지는 스페까지 가져야하기 때문임을 유추할 수 있었다. 스페이 적어도 여전히 247개의 아이템 중에 분류되지 않은 116개의 아이템이 있다. 이 경우 ‘total\_usage\_time’(총 이용 시간)으로 1차 분류되며 각 노드들은 리뷰의 수, 리뷰의 길이에 따라 분류되었다. 이를 직관적으로 해석해보면 리뷰의 총 추천수가 많으면 스페 수는 적은 아이템 중에선 사용시간이 적으면 리뷰 수가 많아야 best\_item으로 분류되고, 사용시간이 많으면 리뷰의 길이가 길고(성실한 리뷰의 개수) 아이템의 평점이 높을수록 best\_item으로 분류된다고 말할 수 있다. 리뷰 추천수가 부족한 경우엔 분류 값에서 normal : best의 비율이 8242 : 322뿐이므로, 다수의 리뷰 추천수가 낮은 normal 아이템 중에서 소수의 best 아이템을 찾아내는 과정으로 노드들이 이루어진다. 그 첫 번째 기준은 ‘avg\_usage\_time’(평균 이용 시간)이다. 이 경우 322개의 best\_item 중에선 반 정도밖에 안되는 145개의 아이템만이 해당했지만 normal item의 개수가 8242개에서 24개로 걸러지며 높은 gain을 얻을 수 있었다. 이는 평균 이용시간이 적을 경우 인지도가 낮을 것이며, 그 경우 best\_item으로 선정되기 어려울 것이라는 예측과 일치하는 결과이다. 평균 이용시간이 많은 경우 ‘avg\_review\_len’(리뷰의 평균 길이)가 짧을수록 best\_item 일 가능성성이 압도적으로 높았다. 이는 다수의 유저가 많은 시간을 쏟는 아이템의 경우 간략한 짧은 리뷰가 많으며, 이런 아이템들도 인지도와 총 사용시간 면에서 best\_item으로 선정될 수 있음을 설명했다. 나머지에 속하는 총 리뷰 추천수도 적고, 평균 사용 시간도 적은 8395개의 아이템 중에선 평균 사용 시간이 더 작을 경우 7380개의 아이템 중 10개만이 best\_item으로, 그마저도 리뷰의 수가 14.5개보다는 많을 경우에서 전부 걸러지며 best\_item 선정에 있어서 평균 사용시간과 리뷰의 수가 반영하는 인지도는 필요조건임을 확인할 수 있었다. 평균 사용시간이 1480시간보단 적고 291시간보다 많은 1015개의 아이템 중에선 리뷰의 길이가 길고 ‘ratings’(평가점수)가 높을 경우 best\_item으로 선정되는 비교적 명확한 분류 기준이 있었다. 즉 평균 사용 시간이 전체 중에서 상대적으로 적은 아이템일 경우 짧게 사용하면서도 긴 리

뷰와 좋은 평가 점수를 남길 정도로 강한 인상을 주는 아이템들이 best\_item으로 선정됨을 확인할 수 있었다.

결과적으로, min\_sample\_leaf=10, max\_depth=5라는 과적합을 방지하기 위한 파라미터를 걸어 줬음에도 하나의 잎 노드를 제외한 대부분의 잎 노드들이 depth=5로 가기 이전에 완전히 분류되거나, depth=5에서도 최우측 하나의 잎 노드만을 제외하고 낮은 gini impurity 값을 지닌 팬찮은 decision tree 모델이 생성되었음을 확인할 수 있었다. 이 경우 개선 방안은 더 나은 input 데이터를 활용해서 Depth=5 안에서 모든 leaf node가 impurity 값을 목표치 아래로 가지는 모델을 찾는 것이다.

### 3. Decision Tree-Entropy

Entropy로 생성한 Decision Tree의 구체적인 파라미터들은 다음과 같다.

| DT entropy parameter     |         |
|--------------------------|---------|
| ccp_alpha                | 0.0     |
| class_weight             | None    |
| criterion                | entropy |
| max_depth                | 5       |
| max_features             | None    |
| max_leaf_nodes           | None    |
| min_impurity_decrease    | 0.0     |
| min_impurity_split       | None    |
| min_samples_leaf         | 10      |
| min_samples_split        | 2       |
| min_weight_fraction_leaf | 0.0     |
| random_state             | None    |
| splitter                 | best    |

각 파라미터의 의미는 gini 파트에서 설명한 것과 동일한다. 이 부분에선 impurity criterion 만이 ‘entropy’로 변경되었다. 생성된 Decision Tree(Entropy)는 제출 파일에 첨부하였다.

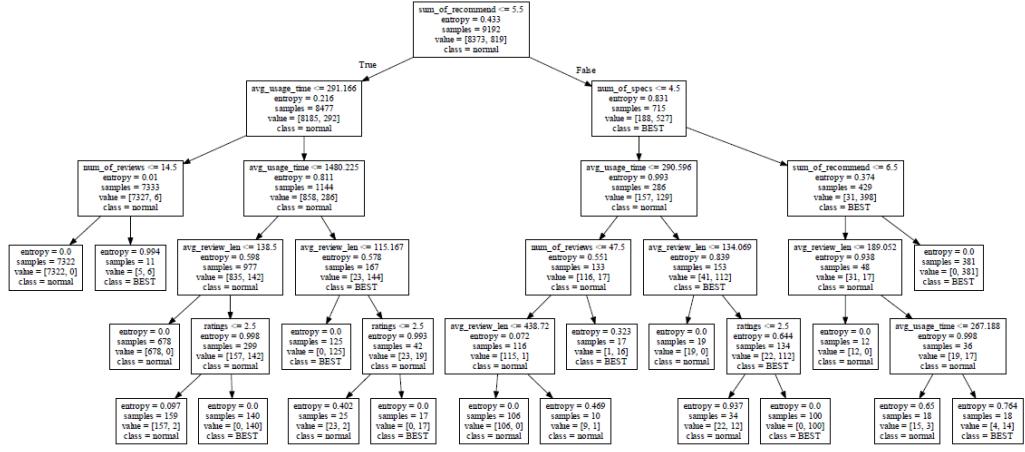


FIG. 2: Decision Tree - Entropy

## 4. Entropy-그래프 분석

Entropy 지표를 활용한 Decision Tree 모델의 경우에도 Gini지표로 생성한 것과 동일하게 'sum\_of\_recommend'(리뷰 추천 수)값을 기준으로 첫 분류가 이루어졌다. 첫 node에 의해 8373개의 아이템 중에 있는 819개의 best 아이템은 715개 중 527개가 best-item인 node와 8477개 중 292개만이 best-item인 node로 분류되었다. 두 모델의 공통점에서 리뷰 추천수가 best\_item 선정 여부에 미치는 영향이 큼을 유추 가능하다. 또한 여기서 리뷰 추천수가 큰 노드에선 gini 지표의 의사결정 나무 모델과 동일하게 num\_of\_specs(스펙의 수)로 두 번째 분류가 이루어졌다. 여기서 스펙의 수가 4.5보다 많을 경우 리뷰 추천 수로 다시 분류하며 얻어지는 381개의 best\_item이 속한 0의 entropy를 갖는 잎 노드는 gini지표에서 얻어지는 잎 노드와 같은 샘플을 가진 노드임을 확인 가능하다.

이번엔 entropy 의사 결정 나무 모델의 좌측을 관찰해보면, 'avg\_usage\_time'(평균 이용 시간)이 적고 'num\_of\_reviews'(리뷰의 수)도 부족한 경우에 얻어지는 7322개의 normal item이 속한 노드 또한 gini 지표에서 같은 조건을 확인할 수 있는 node이다. 그 주변을 확인해보면 상위 노드에서 사용시간을 기준으로 나누고 있는데, 이 또한 gini 지표에서 동일한 기준으로 나눴음을 확인할 수 있다. 그 아래로 이어지는 'avg\_review\_len'(평균 리뷰 길이)과 'ratings'(평가점수)의 기준도 gini 지표에서 동일하게 등장했던 분류의 지표임을 보아 리뷰 추천수가 적은 경우엔 'avg\_usage\_time', 'avg\_review\_len' 및 'ratings'가, 'sum\_of\_recommend'가 큰 경우엔 'num\_of\_specs'로 다수의 best\_item을 찾아내고, 그 외의 경우엔 'num\_of\_reviews', 'avg\_usage\_time', 'avg\_review\_len'와 'ratings'로 판단함을 확인할 수 있다. 전반적으로 gini 지표로 생성된 의사 결정 나무 모델과 유사하다. 앞서 나열된 지표들이 다른 방법으로 가공되어 입력 값으로 사용될 순 있어도 의사 결정 나무 형성에 있어서는 의미, 데이터면에서 필수적인

입력 값들임을 유추할 수 있다.

#### D. R 1-4 : Decision Tree Version

다른 버전의 Decision Tree를 만들어 보기 위해 3가지 관점에서 만들어져 있는 의사결정 나무들을 관찰했다. 첫째, input feature들이 의미 있는지를 확인했다. 빠질 것은 없는지, 더 들어갈 것은 없는지 여러 입력 값을 테스트해 보았다. 둘째, 과적합되거나 지나치게 덜 분류된 부분이 있는지, 혹시 있다면 학습 모델의 파라미터를 수정할 만한 부분이 있는지 확인했다. 셋째, 생성된 의사결정나무의 각 잎 노드까지의 논리가 일리가 있는지 확인했다. 각 과정은 다음과 같다.

##### 1. 입력 값

‘num\_of\_tags’ 제외 : 먼저 기존의 Gini와 Entropy를 impurity criterion으로 한 의사결정 나무 모델 중에선 Gini가 더 깔끔한 형태로 나무를 형성했고, 둘 중 어느 것을 택하여도 Best split에서 크게 변화가 없었기에 Gini를 impurity criterion으로 설정하고 시작했다. 기존의 입력 값들은 평가 점수, 스펙의 수, 태그의 수, 사용한 유저 수, 인당 평균 이용 시간, 작성된 리뷰 수, 리뷰의 추천 수 총합, 리뷰의 평균 본문 길이, 이렇게 총 8가지였다. 이 중에서 태그의 수는 의사결정나무 모델에 등장하지 않았다. 그래서 best\_item에 대해 태그의 수의 평균이 경향성이 아예 없는 것은 아닌지 확인하기 위해 MySQL에서 best\_item에 대해 num\_of\_tags의 sum을 구하는 쿼리를 실행했다. 그 경우 태그의 수에는 확실한 차이가 있었다. 그러나 tag의 수보다 더 유의미한 스플릿 기준들이 존재하므로 미뤄지며, 또한 tag의 수는 다양하게 분포하므로 이분법적이게 나누기엔 적합하지 않은 노드라고 판단하여 입력 값에서 제외해도 된다고 판단했다.

|   | best_item | avg(num_of_tags) |
|---|-----------|------------------|
| ▶ | 1         | 16.5507          |
|   | 0         | 7.7831           |

FIG. 3: 통계치

‘sum\_of\_recommend’ 보정 : ‘sum\_of\_recommend’의 값은 리뷰에서 그 아이템을 추천했는지, 아니면 추천하지 않았는지 여부의 평균을 낸 것이다. ‘Review’ 테이블에는 ‘helpful\_count’라는 리뷰에 대한 공감 정도를 표시하는 지표가 있다. 그 리뷰에 공감한 사람들 또한 그 리뷰를 작성한 것과 같은 감상을 가질 것이라는 판단 하에, 리뷰의 공감 수를 나타내는

‘helpful\_count’와 공감 비율을 나타내는 ‘helpful\_score’를 이용해 보정된 ‘sum\_of\_recommend’를 ‘weighted\_recommend’라는 새로운 입력 값으로 변화시켜 주었다. 즉, ‘helpful’ 점수가 높은 경우엔 그만큼 가중치를 주었다.

## 2. 의사결정나무 파라미터

우선 node impurity criterion은 gini를 사용했다. 또한 R1-3에서 얻은 의사결정나무 모델에서 max\_depth 파라미터는 5로 충분함을 확인했다. 다만, min\_samples\_leaf는 잎 노드가 normal:best 비율이 각각 5:5일 경우 이를 나누는 유의미한 기준이 있을 수 있다고 생각했으므로 5로 낮춰주었다. 또한, min\_split\_criterion=0.03으로 새로운 파라미터를 추가해주었는데, 이는 기존의 의사결정나무 모델에서 확인 가능한 다음과 같은 의미 없는 분류가 일어나지 않도록 node impurity 값이 충분히 작다면 분류를 더 진행하지 않도록 하는 과적합 방지 파라미터이다.

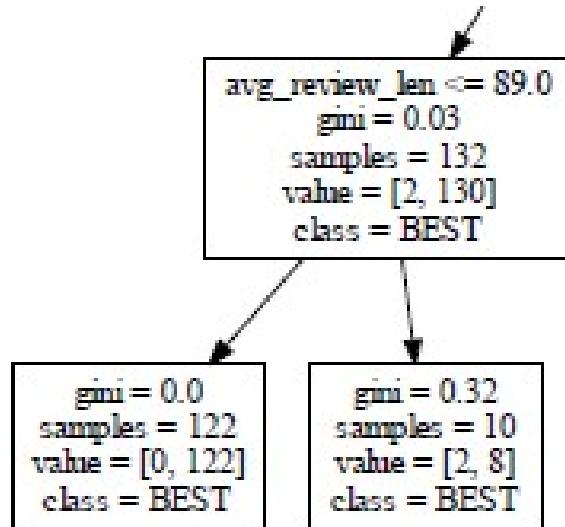


FIG. 4: gini-node

## 3. 잎 노드까지의 논리의 타당성

-‘Num\_of\_specs’ 입력 값 제거 : 기존의 의사결정나무에서 ‘num\_of\_specs’를 기준으로 best\_item 여부를 매우 높은 샘플 수로 가려낼 수 있음을 확인했다. 어떻게 이런 결과가 나오는지 item\_specs.csv 파일을 열어본 결과 specs가 많은 경우엔 대부분 ‘steam achievement’라는

|           |  |
|-----------|--|
| <b>유지</b> | <b>ratings, num_of_users, avg_usage_time, num_of_reviews, len_of_reviews</b> |
| <b>제외</b> | <b>num_of_tags, num_of_specs</b>   |
| <b>추가</b> | <b>x</b>   |
| <b>수정</b> | <b>sum_of_recommend -&gt;weighted_recommend</b>                              |

spec을 포함하고 있었다. 이는 best\_item으로 선정될 시 얻게 되는 spec 값으로, ‘num\_of\_specs’로 best\_item을 가려낸다는 것은 best\_item이기 때문에 best\_item이라는 식의 순환 논리적인 결과라고 판단했다. 즉, best\_item의 특징을 파악하는 데에 있어 논리적으로 무의미한 부분이며 gain이 지나치게 높은 좋은 split인 탓에 다른 split들의 사용을 막는다고 보고 ‘num\_of\_specs’가 의사결정나무를 더 성공적이게 만들에도 불구하고 입력 값에서 제외하기로 결정했다. 이로써 결과적인 입력 값은 다음과 같다.

| parameter          |      |
|--------------------|------|
| criterion          | Gini |
| Min_samples_leaf   | 5    |
| Max_depth          | 5    |
| Min_impurity_split | 0.03 |

#### 4. Decision Tree 결과 분석

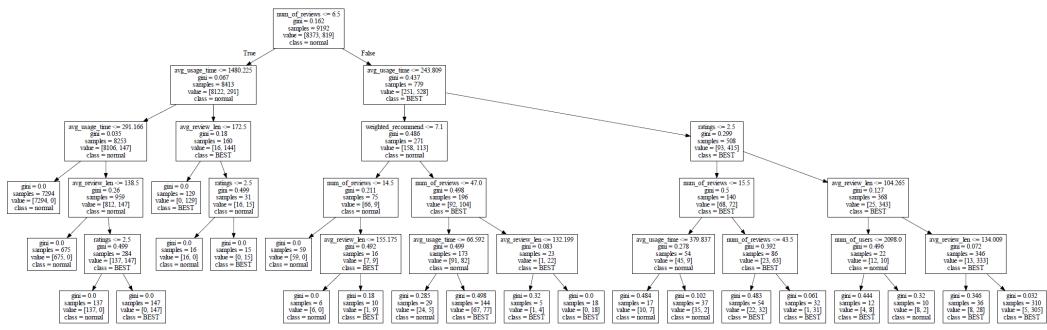


FIG. 5: New Decision Tree

‘weighted\_recommend’ 입력 값을 사용한 결과, 첫 분류는 ‘num\_of\_reviews’로 대체되었다. 일반적으로 best\_item으로 선정되기 위해선 인지도가 높고 사용자층이 많아야 가능하다는 관찰과 일관되었다. 그 다음 분류의 기준은 ‘avg\_usage\_time’이었다. 기준에 두번째 분류에서

압도적으로 좋은 gain을 보였던 ‘num\_of\_specs’의 기준이 사라졌기에 다른 분류 기준이 들어올 수 있었다. ‘Avg\_usage\_time’의 경우 사용 시간이 많은 쪽이 best 클래스의 노드로 분류되었다. 아이템의 best\_item 설정에 있어서 인지도와 얼마나 이용됐는지가 가장 중요함을 알 수 있다. 세 번째 분류부터는 평균 사용 시간, 평균 리뷰 길이, ‘weighted\_recommend’, ‘ratings’ 등이 다양하게 등장했다. 리뷰 수가 적은 아이템들의 경우 사용 시간이 적다면, 어느 정도의 사용 시간은 있으며 ‘ratings’, 즉 평가 점수가 높은 아이템들이 BEST로 얻어졌다. 특히 이 부분의 경우 0.0으로 통일된 gini값들을 보이며 훌륭하게 의사결정나무 가지가 형성되었음을 알 수 있었다. 리뷰 수가 적으며 평균 사용 시간은 많은 경우, R1-3에서 만들었던 의사결정나무 모델과 비슷하게 평균 리뷰 길이가 짧은 아이템들이 best\_item으로 높은 확률로 선정되었다. 리뷰 길이가 긴 경우엔 ‘ratings’로 분류할 경우 normal과 best\_item을 완전하게 분류할 수 있었다. 이로써 관찰한 리뷰수가 적은 쪽의 분류는 훌륭하게 이뤄졌음을 확인했다.

리뷰 수가 많은 아이템들의 경우, 사용 시간이 적다면 ‘weighted\_recommend’가 높은지 여부로 113개의 best\_item 중에서 104개의 best\_item이 속하는 잎 노드를 분류해낼 수 있었다. 이 중에선 리뷰의 수가 47개보다 많을 경우 높은 확률로 best\_item이었고, 리뷰의 수가 보다 적은 경우엔 사용 시간이 많을 경우 대부분의 best\_item들이 그 잎 노드에 속하였다. 다만 이렇게 생긴 잎 노드는 67개의 normal과 77개의 best로 높은 impurity를 보였다. ‘weighted\_recommend’ 가 높지 못할 경우 리뷰의 수로 normal 7개에 best\_item 9개가 전부 속하는 잎 노드를 분류할 수 있었는데, 이 샘플은 표본수가 적지만 ‘min\_samples\_node’의 기준을 낮춰줘서 ‘avg\_review\_len’에 대해 분류가 한 번 더 일어났다. 여기까지 분류된 샘플들은 리뷰 길이가 길어야 best\_item 으로 분류되는 경우가 많다는 유의미한 정보를 얻었다. 리뷰 수가 많은 아이템들의 경우, 사용 시간까지 많다면 508개의 샘플 중 415개의 아이템이 best\_item인 잎 노드로 분류되며 대다수의 best\_item을 분류해낼 수 있었다. ‘num\_of\_specs’와 같은 순환 논리에 의한 대다수의 분류가 아니라는 점에서 의미 있는 개선이다. 이는 평균 점수를 기준으로 2.5보다 높을 경우 343개의 best\_item의 속하는 잎 노드로 분류되었다. 여기서 추가로 평균 리뷰 길이가 길 경우 333개의 best\_item이 속하는 잎 노드로 분류되었다. 이로써 대다수의 best\_item은 리뷰의 수가 많으며 평균 사용 시간이 충분히 많고, 평가는 2.5보다 좋으며 평균 리뷰 길이도 긴 편이라는 납득 가능한 논리를 도출해낼 수 있었다. 이 주요한 best\_item 잎 노드를 제외하면 리뷰 수가 많을수록, 혹은 유저 수를 기준으로 3~4번째 분류가 이뤄졌다.

결과적으로 R1-3에서 생성했던 의사결정나무와 비교하면 잎 노드들의 impurity 값은 눈에 띄게 개선되지 않았지만 논리적으로 더 많은 유의미한 결론을 얻을 수 있는 의사결정나무를 만들 수 있었다.

### III. PART II : 연관 분석

#### A. R2-1 : 상위 30개의 bundle 저장하기

유저와 번들 사이의 연관 분석에는 모든 번들이 아닌 상위 30개의 번들을 이용하기 위해서, 우선 각 번들의 점수를 표현하기 위한 ‘bundle\_id’, ‘bundle\_name’, ‘num\_item’, ‘num\_genre’, ‘score’라는 열들을 가진 ‘bundle\_score’라는 view를 만들었다. ‘bundle\_id’와 ‘bundle\_name’의 경우 ‘bundle’ 테이블에서 select문으로 얻을 수 있었다. ‘num\_item’과 ‘num\_genre’는 각각 ‘bundle\_item’, ‘bundle\_genre’ 테이블에서 ‘bundle\_id’에 대해 튜플의 수를 count했다. 이때, ‘num\_item’과 ‘num\_genre’에는 100씩 곱하여 가중치를 두었다. 번들에 포함된 아이템들의 유저수의 평균을 의미하는 ‘num\_user’는 bundle\_item 테이블과 user\_item 테이블에서 bundle\_id\*item\_id별로 이용자 수를 센 다음, 그것을 bundle\_id에 대해 합하여, 최종적으로는 곱하기 전의 ‘num\_item’ 값으로 나눠주어 얻었다. 최종적인 번들의 점수인 ‘score’는 위의 쿼리를 실행하여 얻은  $100 * \text{num\_item}$ ,  $100 * \text{num\_genre}$ , ‘num\_user’의 합으로 얻을 수 있다. 이렇게 얻어진 ‘score’라는 열을 view의 마지막 열에 추가하였다.

위에서 설명한 내용을 make\_bundle\_score라는 문자열 변수로 정의해 mysql-connector-python을 이용하여 저장하였다. 이를 pandas 라이브러리를 활용하여 score에 대해 내림차순으로 정렬하고, 그 중 상위 30개를 취해 Top30에 드는 번들의 목록을 포함한 bundle\_score라는 View를 만들었다.

#### B. R2-2 : Top30 bundle에 대한 각 user의 rating

각 user가 Top30 bundle에 매긴 평가점수를 담은 horizontal table인 ‘user\_bundle\_rating’은 ‘user\_item’, ‘bundle\_item’, ‘review’ 테이블에 담긴 정보를 통해 얻을 수 있다. 다만, ‘user\_item’ 테이블이 크기가 매우 크고, 최종 view에는 위에서 얻은 상위 30개의 bundle만을 활용할 것 이므로 ‘user\_item’에서 top30 bundle에 포함된 아이템들과 관련된 튜플만을 걸러내는 작업을 미리 수행했다. 그러기 위해 우선 top30 bundle\_id 리스트를 ‘top\_bundle\_list’라는 이름으로 python에 저장했다.

먼저 ‘user\_item’, ‘bundle\_item’, ‘review’ 테이블을 ‘user\_id’와 ‘item\_id’가 각각 같은 조건 으로 left join해서 하나의 view를 만들었다. 여기서 bundle\_name이 최종적인 key가 되므로 ‘bundle\_item’을 기준으로 left join하며 동시에 상위 30개에 해당하는 bundle만을 취하도록 했다. 또, ‘bundle’에 대해 join시켜 ‘bundle\_name’을 같이 얻었다. Left join 후, item 수와 recommend가 1인 수를 ‘user\_id’와 ‘bundle\_id’별로 센다. 여기서 count된 item 수가 5보다 클 때, 5로 바꿔주고, recommend의 수에는 5를 곱한다. 이 두 값을 더해 rating을 구할 수 있다. 같은 방법으로 이 문자열을 ‘make\_user\_bundle\_rating’에 저장한 후, mysql-connector-

python을 사용하여 저장했다. 이로써 user\_id, bundle\_id를 key로 하며 유저의 번들에 대한 rating을 저장한 view를 얻을 수 있었다. 여기서 rating 횟수가 충분히 많은 user에 대해서만 예측을 진행하고자, 20개 이상의 번들에 rating을 가진 user만을 추려내는 ‘partial\_user\_bundle\_rating’이라는 view를 생성하고 했다. 이를 위해서 ‘user\_bundle\_rating’ view에 대해 pandas 라이브러리를 활용하여 value\_count로 각각의 user\_id가 데이터프레임에 몇 번 등장하는지를 count로 나타내는 freq\_user라는 series를 만들고, 이 series중 count가 20 이상인 것만 취했다. 이렇게 구한 freq\_user에 대해 apply함수를 사용하여 각 행의 user\_id가 freq\_user에 포함되는 user\_id만을 다시 return하도록 했다.

### C. R2-3: horizontal table 생성

이용자를 행으로, top30에 드는 번들을 열로 가지는 Horizontal table을 만들기 위해서 pandas 라이브러리를 활용하였다. (freq\_user의 수 \* top30 bundle item의 수)의 크기를 가지고, 모든 값이 False인 데이터프레임 ‘horizontal’을 생성했다. 여기서는 다뤄야할 데이터가 많으므로 Boolean 자료형을 사용하여 0, 1을 구분하도록 했다.

테이블의 빈 칸을 채우기 위해 top\_bundle\_list에 대하여 반복문을 실행했다. 목록에 속하는 top\_bundle에 대해 유저가 매긴 점수를 series로 매 loop마다 저장하였으며, 그 series의 rating 값을 1로 바꿔주며 user\_id와 bundle\_id에 맞게 채워주었다. 즉, column 방향으로 채워 나갔다. 같은 열에 존재하는지 여부는 pandas의 Boolean indexing을 활용하였다. 이로써, Top30 bundle에 대해서 최소 20개 이상의 아이템을 이용한 유저들에 대해 어떤 유저가 어떤 번들을 사용했는지에 대한 정보를 담은 데이터프레임을 얻었다.

### D. R2-4 : 연관분석

연관분석을 통해 알고자 하는 것은 어떤 번들을 사용한 이력이 있을 때 다른 번들을 사용하는 경우가 있는지 rule을 찾아내는 것이다. 연관 분석의 frequent\_rule\_generation에서 활용할 frequent\_itemset을 만들기 위해서 Mlxtend 모듈의 apriori를 사용하였고 frequent의 min\_support는 0.35로 지정했다. ‘Rules’ 변수에는 frequent\_itemset이라는 apriori를 활용하며 rule의 타당성 평가 척도는 ‘Lift’로 하라는 연관규칙을 지정하였고, min\_threshold는 2가 되게 했다.

### 1. 연관분석 결과 평가

먼저 pandas 라이브러리 DataFrame.describe()를 사용하여 생성된 룰들의 대략적인 분포를 확인했다. 아래 표는 이를 소수 셋째 자리까지 나타낸 수치이다. 총 446544개의 룰이 rules에 생성되었고, itemset 들의 길이는 4에서 13 사이로 나타났다.

|       | antecedent | consequent | support | confidence | lift   | leverage | conviction |
|-------|------------|------------|---------|------------|--------|----------|------------|
| count | 446544     | 446544     | 446544  | 446544     | 446544 | 446544   | 446544     |
| mean  | 0.418      | 0.418      | 0.377   | 0.904      | 2.166  | 0.203    | inf        |
| std   | 0.027      | 0.027      | 0.020   | 0.057      | 0.036  | 0.010    | NaN        |
| min   | 0.350      | 0.350      | 0.350   | 0.747      | 2.115  | 0.185    | 2.57E+00   |
| 25%   | 0.399      | 0.399      | 0.358   | 0.865      | 2.140  | 0.194    | 4.45E+00   |
| 50%   | 0.420      | 0.420      | 0.373   | 0.912      | 2.150  | 0.202    | 6.54E+00   |
| 75%   | 0.438      | 0.438      | 0.390   | 0.947      | 2.177  | 0.210    | 1.06E+01   |
| max   | 0.469      | 0.469      | 0.457   | 1.000      | 2.314  | 0.243    | inf        |

각 rule의 support 값은 평균 0.377, 표준편차 0.020으로 주로 0.37 주위에서 균형 있게 분포함을 볼 수 있었다. confidence값의 평균은 0.904로, X라는 itemset을 이용했을 때 Y를 이용하는 확률이 0.904라는 높은 수치를 보이는 rule을 다수 얻었음을 알 수 있었다. lift값은 평균 2.166 근처에 분포하며 metric='lift'로 설정함으로써 양의 상관관계를 가진 rule들을 얻었음을 알 수 있었다. 이를 더 구체적으로 확인하기 위해 전체 rule에서 lift값의 분포를 pandas code를 사용하여 저장하고 그 결과를 보았다.

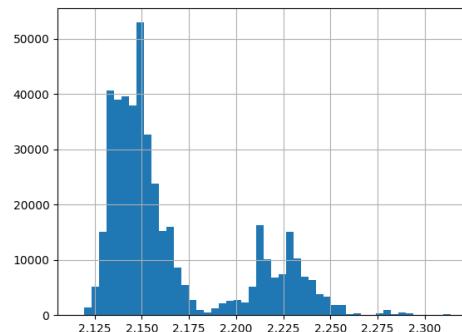


FIG. 6: Lift histogram

Fig 6. 히스토그램에서 lift값이 몇 그룹으로 나뉘어 분포하고 있음을 볼 수 있었다.

|       | antecedent | consequent | support | confidence | lift  | leverage | conviction |
|-------|------------|------------|---------|------------|-------|----------|------------|
| count | 2592       | 2592       | 2592    | 2592       | 2592  | 2592     | 2592       |
| mean  | 0.395      | 0.395      | 0.356   | 0.903      | 2.287 | 0.200    | 8.367      |
| std   | 0.020      | 0.020      | 0.003   | 0.046      | 0.010 | 0.002    | 4.936      |
| min   | 0.364      | 0.364      | 0.354   | 0.832      | 2.276 | 0.198    | 3.774      |
| 25%   | 0.376      | 0.376      | 0.354   | 0.858      | 2.280 | 0.199    | 4.397      |
| 50%   | 0.392      | 0.392      | 0.354   | 0.903      | 2.283 | 0.200    | 6.351      |
| 75%   | 0.413      | 0.413      | 0.357   | 0.946      | 2.291 | 0.201    | 10.800     |
| max   | 0.425      | 0.425      | 0.364   | 0.974      | 2.314 | 0.205    | 21.828     |

여기서 주의 깊게 볼 부분은 lift가 특히 높은 rule들이다. 즉 그냥 그 bundle을 사용할 확률보다 X itemset에 해당하는 번들을 이용했을 때 이용할 확률이 매우 높은 경우를 잡아낸 rule들이다. Lift가 2.27이상인 rule은 총 2592개가 있었는데, 이 rule들은 특히나 강한 상관관계를 보여주는 rule들로, 유의미한 정보를 얻을 수 있는 rule들이라고 판단했다. 이 rule들 중 일부를 csv파일로 만들어 그 예시 또한 case1, case2, case3과 같이 확인해보았다. 결과는 table I, II, III에 제시하였다.

| Antecedents                                      | Consequents  |
|--|--|
| 'Half-Life Complete'                             | 'Borderlands Take Over Your Life Bundle'           |
| 'Sid Meiers Civilization V: Complete'            | 'Sid Meiers Civilization Anthology'                |
| 'Grand Theft Auto V & Megalodon Shark Cash Card' | 'Source Multiplayer Pack'                          |
| 'Borderlands Triple Pack'                        | 'Grand Theft Auto V & Great White Shark Cash Card' |

TABLE I: Case1) confidence = 0.97, lift = 2.28

| Antecedents                                      | Consequents  |
|--|--|
| 'Grand Theft Auto V & Megalodon Shark Cash Card' | 'Eidos Anthology'                                  |
| 'Half-Life Complete'                             | 'Grand Theft Auto V & Whale Shark Cash Card'       |
| 'Borderlands Triple Pack'                        | 'Grand Theft Auto V & Great White Shark Cash Card' |
| 'Sid Meiers Civilization V: Complete'            | 'Valve Complete Pack'                              |
|  | 'Source Multiplayer Pack'                          |
|  | 'Sid Meiers Civilization Anthology'                |
|  | 'Borderlands Take Over Your Life Bundle'           |

TABLE II: Case2) confidence = 0.94, lift = 2.28

| Antecedents                                  | Consequents  |
|--|--|
| 'Half-Life Complete'                         | 'Eidos Anthology', 'Grand Theft Auto V & Megalodon Shark Cash Card', 'Sid Meiers Civilization V: Complete' |
| 'Grand Theft Auto V & Whale Shark Cash Card' | 'Grand Theft Auto V & Great White Shark Cash Card'   |
| 'Borderlands Triple Pack'                    | 'Valve Complete Pack'  |
| 'Sid Meiers Civilization Anthology'          | 'Source Multiplayer Pack', 'Borderlands Take Over Your Life Bundle'  |

TABLE III: Case3) confidence = 0.92, lift = 2.28

위의 경우들은 각각 Antecedents에 해당하는 번들을 이용한 이력이 있을 경우 'Consequents'에 해당하는 번들 또한 이용할 확률이 매우 높음을 의미한다고 볼 수 있다. 즉 유저에 대해서 타겟 마케팅을 하는 데에 제시할 수 있는 아이템으로 이 정보를 활용할 수 있을 것이다.

#### IV. PART III : 추천시스템

Part III에서는 추천시스템을 통해 사용자에게 번들을 추천하는 시스템을 구현한다. 번들은 아이템의 묶음이지만 추천시스템 방식과 용어 통일을 위해 앞으로 아이템으로 지칭한다. Surprise 모듈[1]을 사용하여 추천시스템을 구현하였다. Part II에서 작성한 UBR(User\_Bundle\_rating) 파일을 사용하여 trainset, testset을 구성하였으며 각각 사용자 기반, 아이템 기반, 행렬분해 방법을 사용하여 추천시스템을 구현하였다.

##### A. R3-1 : get\_top\_n 함수 작성

NBCF 방식의 추천 시스템으로 추천된 아이템들 중 top n개의 아이템을 추려내는 함수를 작성하였다. User-based, item-based 두 경우에 대해 작성하였으며, 결과값이 저장된 딕셔너리에서 추정 평가 값이 높은 순서대로 정렬한 뒤 top n개만을 남기는 방법으로 함수를 구현했다.

##### B. R3-2 : User-based Recommendation

Neighborhood-based Collaborative Filtering에서 사용자 기반 추천 시스템 방식은 특정 사용자에 대해서 유사한 특성을 보이는 사용자의 기록을 통해서 추천 정도를 결정한다. 여기서 k명의 neighborhood는 k명의 유사한 사용자가 된다. 유사한 정도를 파악하는 유사도 함수에는 cosine 척도와 Pearson 상관 계수가 있다. 우선 R-3-2에서 요구한 사용자 기반 추천 시스템을 두 가지 경우에 대해 구현했다. 첫째는 KNNBasic 알고리즘과 cosine 유사도 함수를 사용하는 방식, 둘째는 KNNWithMeans 알고리즘과 pearson 함수를 사용하였다. KNNBasic 알고리즘은 해당 사용자와 유사한 사용자 k명(이하)를 찾아 유사한 사용자들이

아이템 사용에 평가한 값을 가중치를 곱한 뒤 합하여 정규화한 값으로 사용자의 아이템 평가 값을 추정한다. 여기서 유사한 사용자를 판단하는 기준으로 넣어준 유사도함수를 사용한다. KNNWithMeans 알고리즘은 KNNBasic과 달리 평가 값을 바로 가중합하지 않고 평균을 뺀 값을 가중합 한 뒤, 해당 사용자의 평균값을 더하는 형식으로 평가 값을 추정한다. 즉, 이 알고리즘은 사용자가 rating을 줄 때의 개인 편차를 고려하여 반영할 수 있는 KNNBasic보다 개선된 방법이라고 볼 수 있다.[2]

$$KNNBasic : \hat{r}_{ui} = \frac{\sum_{v \in N_i^K(u)} sim(u, v) \cdot r_{vi}}{\sum_{v \in N_i^K(u)} sim(u, v)} \quad (3)$$

$$KNNWithMeans : \hat{r}_{ui} = \frac{\sum_{v \in N_i^K(u)} sim(u, v) \cdot (r_{vi} - \mu_v)}{\sum_{v \in N_i^K(u)} sim(u, v)} + \mu_v \quad (4)$$

유사도 함수 cosine은 사용자 u,v 의 평가값을 벡터로 나타냈을 때 내적과 벡터의 크기를 이용해 cosine 값을 구하는 방법이다. Pearson 또한 사용자 u,v의 평가값 벡터의 상관계수를 구하는 방법인데, 이 방법은 cosine과 유사하지만 사용자의 평균을 뺀 값을 벡터로 사용하여 평균 점수에 대한 사용자별 개인 편차를 제거한 방식이다.

Requirement 3-2의 User-based Recommendation 결과에 해당하는 추천 결과 top-10 item 은 제출 파일에 첨부하였다. 이 두 방법 외에 성능이 높은 추천시스템을 찾기 위해 우선 random seed를 고정한 후, KNNBasic, KNNWithMeans, KNNBaseline 세 가지 알고리즘에 대해 각각 네 가지 유사도함수 pearson, pearson\_baseline, cosine, msd를 사용하여 총 12가지 경우에 대해 실험하였다. KNNBaseline 은 KNNWithMeans 와 유사한 알고리즘으로 평균 대신 baseline을 구하여 rating에서 뺀 뒤, 그 값을 활용해 점수를 예측한 뒤에 해당 사용자의 baseline을 다시 더해주는 방식이다. baseline이란 해당 사용자가 기본적으로 일반적이게 주는 평가 값으로 사용자의 점수 편향을 의미한다. 일반적으로 x점 이상 주는 사용자라 할 때 x가 baseline이 된다.

$$KNNBaseline : \hat{r}_{ui} = \frac{\sum_{v \in N_i^K(u)} sim(u, v) \cdot (r_{vi} - b_{ui})}{\sum_{v \in N_i^K(u)} sim(u, v)} + b_{ui} \quad (5)$$

pearson\_baseline 유사도 함수는 위와 마찬가지로 pearson 유사도 함수에서 평균 부분을 baseline 으로 바꾼 함수이며, msd 유사도 함수는 mean squared difference 로 사용자 u,v가 모두 사용한 아이템에 대해 평가 값의 제곱합을 모두 사용한 아이템수로 나누어 준 값이다.

각각의 경우에 대해 데이터를 5등분하여 각 1/5마다 한번씩 test set으로 나머지 4/5 만큼을 trainset 으로 하여 5-fold cross validation 을 진행하였다. 정확도는 5회 진행한 유효성 검사의 평균값을 사용하였으며, 지표로는 RMSE(Root mean square error)를 사용하였다. RMSE 값은 다음과 같다.

|                  | KNNBasic     | KNNWithMeans | KNNBaseline  |
|------------------|--------------|--------------|--------------|
| Pearson          | <b>1.026</b> | <b>1.034</b> | <b>1.01</b>  |
| Cosine           | <b>1.016</b> | <b>1.028</b> | <b>1.004</b> |
| MSD              | <b>1.037</b> | <b>1.052</b> | <b>1.025</b> |
| Pearson_baseline | <b>0.958</b> | <b>0.974</b> | <b>0.951</b> |

총 12가지 경우에서 평균 RMSE가 가장 낮은 우수한 값을 보인KNNBaseline 알고리즘과 Pearson\_baseline 유사도 함수의 조합을 best model 로 선정하였다.

### C. R3-3 : Item-based Recommendation

다음으로 R-3-3에서는 R-3-2의 User-based Recommendation에서와 유사한 방법으로 Item-based Recommendation을 구현했다. 마찬가지로 KNNBasic, KNNWithMeans 알고리즘을 사용했으며 각각에 대해 cosine, pearson 유사도함수를 사용했다. 대신 이번엔 아이템 기반으로 한 추천시스템을 구현하였다. 같은 알고리즘이지만, 차이점은 특정 유저에 대한 유사한 사용자를 찾는 것이 아니라 해당 사용자가 평가한 아이템 중 유사한 k개의 아이템을 찾아 그 아이템들에 대한 사용자들의 평가 값을 이용해 특정 아이템에 대해 높은 점수를 매길 것이라고 추정되는 유저를 찾는다는 점이다. Requirement 3-3의 Item-based Recommendation 결과에 해당하는 추천 결과 top-10 user는 제출 파일에 첨부하였다.

마찬가지로, KNNBasic, KNNWithMeans, KNNBaseline 알고리즘에 대해 유사도함수 pearson, pearson\_baseline, cosine, msd 를 사용하여 총 12가지 경우에 대해 실험하였으며, KNNWithMeans, Pearson 을 유사도함수로 사용하는 방법을 best model 로 선정하였다.

|                  | KNNBasic     | KNNWithMeans | KNNBaseline  |
|------------------|--------------|--------------|--------------|
| Pearson          | <b>1.621</b> | <b>1.021</b> | <b>1.021</b> |
| Cosine           | <b>1.593</b> | <b>1.081</b> | <b>1.081</b> |
| MSD              | <b>1.426</b> | <b>1.057</b> | <b>1.057</b> |
| Pearson_baseline | <b>1.680</b> | <b>1.058</b> | <b>1.056</b> |

#### D. R3-4 : Matrix-based Recommendation

마지막으로, 행렬분해방식을 사용하여 추천시스템을 구현하였다. 사용자-아이템 평가 값 행렬이 연관성이 높아 행렬의 rank값이 행의 수에 비해 적다는 관찰에 기반한 방법으로, SVD(singular value decomposition)을 사용하여 사용자가 평가하지 않은 아이템의 평가 값을 추정한다. R-3-4에서 요구한 SVD, SVD++ 방식을 사용해 구현하였다. SVD를 사용할 때 bias=false로 하여 baseline을 사용하지 않게 설정해주었다. SVD++는 SVD에서 보완된 방법으로, 사용자가 아이템을 평가했는지에 여부를 0 또는 1로 고려하여 implicit rating이라는 값을 추가해 보완한 방법이다.[2] SVD, SVD++ 방식에 대해 각각 n\_factors와 n\_epoch(Gradient Descent 방식의 반복 회수)를 주어진 조건에 맞게 바꿔가며 top-5 bundle 추천 결과를 총 네 가지를 얻을 수 있었다. 결과는 제출 파일에 첨부하였다.

이후, 성능을 높인 추천시스템을 찾기 위해 SVD, SVD++, NMF 세 가지 방식에 대해 n\_factors와 n\_epochs를 늘려가며 실험하였다. NMF는 Non-negative Matrics Factorization의 약자로 SVD에서 보완된 방법이며 항상 분해되는 매트릭스인 UV의 값들이 항상 양수인 값을 가지도록 제한을 걸어준 알고리즘이다. n\_factors와 n\_epochs는 새로운 NMF 방식과 기존의 방식을 비교하기 위해 요구조건에 나와있는 (100, 50), (200, 100)값을 사용하였으며, n\_factors가 200일 때 n\_epochs가 부족하여 추정이 덜 진행되었다 판단하여 충분히 추정이 진행되는 경우를 확인하기 위해 n\_epochs=200인 것을 추가하였다. 결과적으로 총 3가지 parameter 경우와 총 세 가지 알고리즘 SVD, SVD++, NMF를 사용하여 9가지 경우에 대해 실험하였으며 R3-2, R3-3에서와 마찬가지로 5-fold cross validation으로 얻은 RMSE값을 평균으로 하여 모델의 유효성을 평가하였다. 행렬 계산에 시간이 많이 소요되어 큰 parameter 값들에 대해서는 실행하지 못했다. 아래 실험결과를 토대로 선정한 best model 은 SVD++, (n\_factors, n\_epochs) = (200, 200)이다.

| (n_factors, n_epochs) | SVD    | SVD++  | NMF    |
|-----------------------|--------|--------|--------|
| 100, 50               | 0.9433 | 0.9337 | 0.9298 |
| 200, 100              | 0.9442 | 0.9294 | 0.9415 |
| 200, 200              | 0.9428 | 0.9277 | 0.9511 |

#### V. SUMMARY

본 프로젝트는 Decision tree, 연관 분석, 추천시스템 구축과 같은 3개의 파트로 구성되었다. 먼저 PART I: Decision Tree에서는 제시된 조건하에서 Gini 와 Entropy를 기준으로 DT를

완성한 다음, 3단계에 걸쳐 새로운 조건하에서 DT를 완성했다. 그 결과 기존의 DT들에서는 발견할 수 없었던, 세부적인 규칙을 일부 파악할 수 있었다.

PART II: 연관분석에서는 data 전처리를 통해 가공 후, rating을 함으로써 연관분석을 진행했다. 이를 바탕으로 번들 간의 rule들을 찾아내고, 그 타당성을 분석하였다. 그 결과 lift에 따른 rule의 분포를 확인할 수 있었으며, 특히 lift 값이 큰 rule들의 경우에 대해 추가적인 분석을 진행할 수 있었다.

PART III: Recommendation system에서는 User-based와 Item-based, Matrix-based 세 가지 방식으로 접근했다. 앞의 2가지 방식의 경우 KNNBasic과 KNNWithMeans algorithm 외에도 KNNBaseline algorithm을 추가로 적용하여 RMSE 측면에서 더 나은 recommendation model을 선정하는데 성공했다. 또한 Matrix-based 방식에서는 SVD, SVD++ algorithm에 더해 NMF 방식을 추가로 적용, factor와 epoch 수 또한 다양화하여 실험했고 그 결과를 비교 분석, 최선의 model을 선정했다.

---

- [1] N. Hug, Journal of Open Source Software 5, 2174 (2020), URL <https://doi.org/10.21105/joss.02174>.
- [2] C. C. Aggarwal, *Recommender Systems - The Textbook* (Springer, 2016), ISBN 978-3-319-29659-3.