**<PROJECT 최종보고서>**

**VOD 추천 시스템 개발**

**김지훈, 설재완, 최윤서**

**엄현상 교수님**

**임대섭 님(castis, 미래전략 알파팀)**

**Table of Contents**

**1. Abstract 3**

**2. Introduction 3**

**3. Background Study 4**

**A. 관련 접근방법/기술 장단점 분석 4**

**B. 프로젝트 개발환경 5**

**4. Goal/Problem & Requirements 5**

**5. Approach 5**

**6. Project Architecture 6**

**A. Architecture Diagram 6**

**B. Architecture Description 8**

**7. Implementation Spec 10**

**A. Input/Output Interface 10**

**B. Inter Module Communication Interface 10**

**C. Modules 10**

**8. Solution 12**

**A. Implementations Details 12**

**B. Implementations Issues 16**

**9. Results 17**

**A. Experiments 17**

**B. Result Analysis and Discussion 17**

**10. Division & Assignment of Work 18**

**11. Conclusion 18**

**◆ [Appendix] User Manual 19**

**1. 시청 로그 제공 19**

**2. Train set과 Test set으로 분리 19**

**3. Hyperparameter tuning 및 model training 19**

**4. 최종 추천 19**

# **Abstract**

우리는 VOD 서비스를 대상으로 하는 추천 시스템을 개발하는 프로젝트를 맡았다. 그런데 VOD 서비스는 별점과 같은 explicit feedback이 거의 없기 때문에 implicit feedback을 이용하여 개발해야 한다는 제한사항이 있었다. 그래서 알고리즘은 explicit feedback을 활용하는 추천 알고리즘을 그대로 가져다 사용하고, 대신 implicit feedback을 explicit feedback처럼 바꾸는 방법을 고안하기로 했다. 추천 알고리즘은 사용성과 실행 속도 및 예측 정확도 등을 고려하여 colaborative filtering + bias based 알고리즘을 사용하였다. implicit feedback을 explicit feedback으로 바꾸는 방법은 implicit feedback 중에서 시청 시간을 중심으로 하여 모든 사용자의 시청 시간의 중앙값을 기준으로 하는 환산 방식을 고안하여 사용하였다. 추천이 제대로 되었는지를 평가하는 방법은 추천 알고리즘의 결과를 보고 상위 20% 안에 실제 사용자가 본 컨텐츠가 얼마나 포함되어있는지를 계산하여 사용하였다. 그 결과 50%에 육박하는 정답률을 보이는 implicit -> explicit 환산 방식을 고안하는 데 성공하였다.

# **Introduction**

정보가 너무 많다. 제품을 구입하거나 문화생활을 즐길 때와 같이 여러가지 소비를 할 때, 우리는 수많은 회사의 제품과 서비스 사이를 지나면서 가장 적절한 하나를 선택하게 된다. 그러나 소비자가 직접 모든 정보를 조사하기에는 시장에 등록된 제품의 수가 너무 많다. 이런 이유로 소비자에게 상품을 추천해주는 서비스가 등장하게 되었다.

우리가 맡은 과제는 VOD 서비스를 대상으로 추천 시스템을 개발하는 것이다. 오늘날 셋탑박스를 활용한 인터넷 TV 서비스는 TV 송출 외에 TV 다시보기나 각종 영화, 애니메이션, 다큐멘터리 등을 원하는 시간에 볼 수 있게 하는 VOD 서비스도 같이 제공하고 있다. 문제는 세상에 존재하는 VOD 컨텐츠의 수가 너무나도 많다는 점이다. 자연히 VOD 추천 서비스의 필요성이 대두되었다.

Netflix와 같은 대규모 컨텐츠 기반시스템 회사들은 사용자가 매긴 별점 정보를 가지고 추천 시스템을 운용하고 있다. 그러나 국내의 VOD 서비스 회사들은 이러한 별점 데이터가 턱없이 부족하다. 따라서 별점 데이터를 사용하지 않는 다른 추천 시스템을 개발해야 한다.

우리는 사용자의 시청 시간이나 시청 패턴을 분석하여, 그것이 마치 별점인 것처럼 간주하는 추천 시스템을 개발하고자 한다. 추천 알고리즘 자체는 별점을 사용하는 기존의 알고리즘을 그대로 가져다 사용하므로, 사용자의 시청 시간이나 시청 패턴을 얼마나 별점으로 잘 환산하는지가 문제의 핵심이다.

우리의 프로젝트 진행은 다음과 같다. 먼저 VOD 시청 로그를 여러 각도에서 분석하고 여러가지 환산 방법을 고안해 낸다. 그 다음 각 환산 방법에 따라 시청 패턴을 별점 데이터로 환산한 후, 각각의 별점 데이터에 추천 알고리즘을 적용하여 나온 결과를 모두 비교하여 어떤 환산 방법이 가장 적합한지를 찾아낼 것이다. 이때 추천 알고리즘의 종류에 따라 결과가 달라질 수 있으므로, 추천 알고리즘 자체에 대한 이해도 동반하여야 한다.

# **Background Study**

## **관련 접근방법/기술 장단점 분석**

별점과 같은 사용자가 직접 자신의 선호도를 표시한 데이터를 explicit data라고 한다. 반대로 사용자가 직접 표현하지 않은 모든 데이터는 implicit data라고 한다. 우리는 implicit data를 분석하여 그걸 마치 explicit data인 것처럼 환산한 후, 해당 data를 기존의 explicit data를 활용한 추천 알고리즘에 적용하여 결과를 얻어낼 것이다.

Explicit data를 활용한 추천 알고리즘에는 크게 세 가지 방식이 있다.

1. Baseline

Bias-Based algorithm은 사용자와 영화의 평균적인 경향을 고려한 방식이다. 전체 rating의 평균, 각각의 user과 item들의 bias를 가지고 예측을 하는 모델이며, 개별 user와 각각의 item사이의 관계를 고려하지 못한다.

1. Collaborative filtering (CF)

Neighborhood algorithm에는 item-oriented 방식과 user-oriented 방식이 있다. User-oriented algorithm은 비슷한 선호도의 경향을 갖고 있는 user들을 묶는 방식을, Item-oriented algorithm은 비슷한 특성을 갖는 item들을 묶는 방식을 취한다. 우리의 VOD추천에서는 시간이 지날수록 사용자의 수가 증가하는데, User-based 접근방식에서는 이에 따르는 scalability를 감당할 수 없게 되므로 Item-Based 접근방식을 선택하였다.

1. Matrix factorization (MF)

Matrix Factorization Algorithm은 각각의 user과 item을 몇 차원의 수치화된 특징으로 나타낼 수 있다는 가정에서 출발한다. User x Item의 거대한 Matrix를 가정하고, 이를 n x m 크기라고 할 때, 이 matrix를 n x k와 k x m 크기의 matrix의 곱으로 나누어 n명의 user와 m개의 item을 각각 k개의 특징으로 표현한다. 이를 바탕으로 사용자의 선호도를 예측하는 model이 Matrix Factorization Algorithm이다.

Baseline과 Collaborative filtering을 합친 알고리즘이나 Baseline과 Matrix factorization을 합친 알고리즘도 활용 가능하다.

## **프로젝트 개발환경**

Explicit data를 활용한 추천 알고리즘 구현은 python을 활용하여 linux 환경에서 개발하였다. VOD 로그 데이터 분석은 팀원 간 커뮤니케이션을 원활히 하기 위해서 jupyter notebook을 활용하여 진행하였다. 많은 계산량을 요하는 training 모듈은 학부에서 제공하는 원격 리눅스 서버에서 실행하였다.

# **Goal/Problem & Requirements**

우리는 VOD 로그 데이터를 분석하여 사용자의 시청 시간이나 시청 패턴 등을 별점으로 환산하여야 한다. 그리고 나서 그렇게 환산되어 나온 데이터를 기존의 추천 알고리즘에 적용하여 결과를 얻는다. 이 과정에서 첫번째 문제는 VOD 로그 데이터를 어떤 방식으로 별점으로 환산해야 올바른 추천이 될 것인지를 알아내야 하는 것이다. 두번째 문제는 환산되어 나온 별점을 기존 알고리즘 중에서 어느 알고리즘에 적용하여야 더 나은 추천이 될지를 알아내야 하는 것이다. 세번째 문제는 추천되어 나온 결과가 과연 얼마나 제대로 된 추천인지 확인하는 방법을 고안해야 하는 것이다.

# **Approach**

먼저 Netflix prize를 받은 알고리즘에 대한 논문을 읽고 직접 구현한다. 그리고 나서 실제 Netflix data를 이용하여 예측 결과를 비교한다. 그 다음 사용성이나 실행 시간, 예측 정확도 등을 고려하여 그 중에서 가장 나은 한 가지 알고리즘을 정한다.

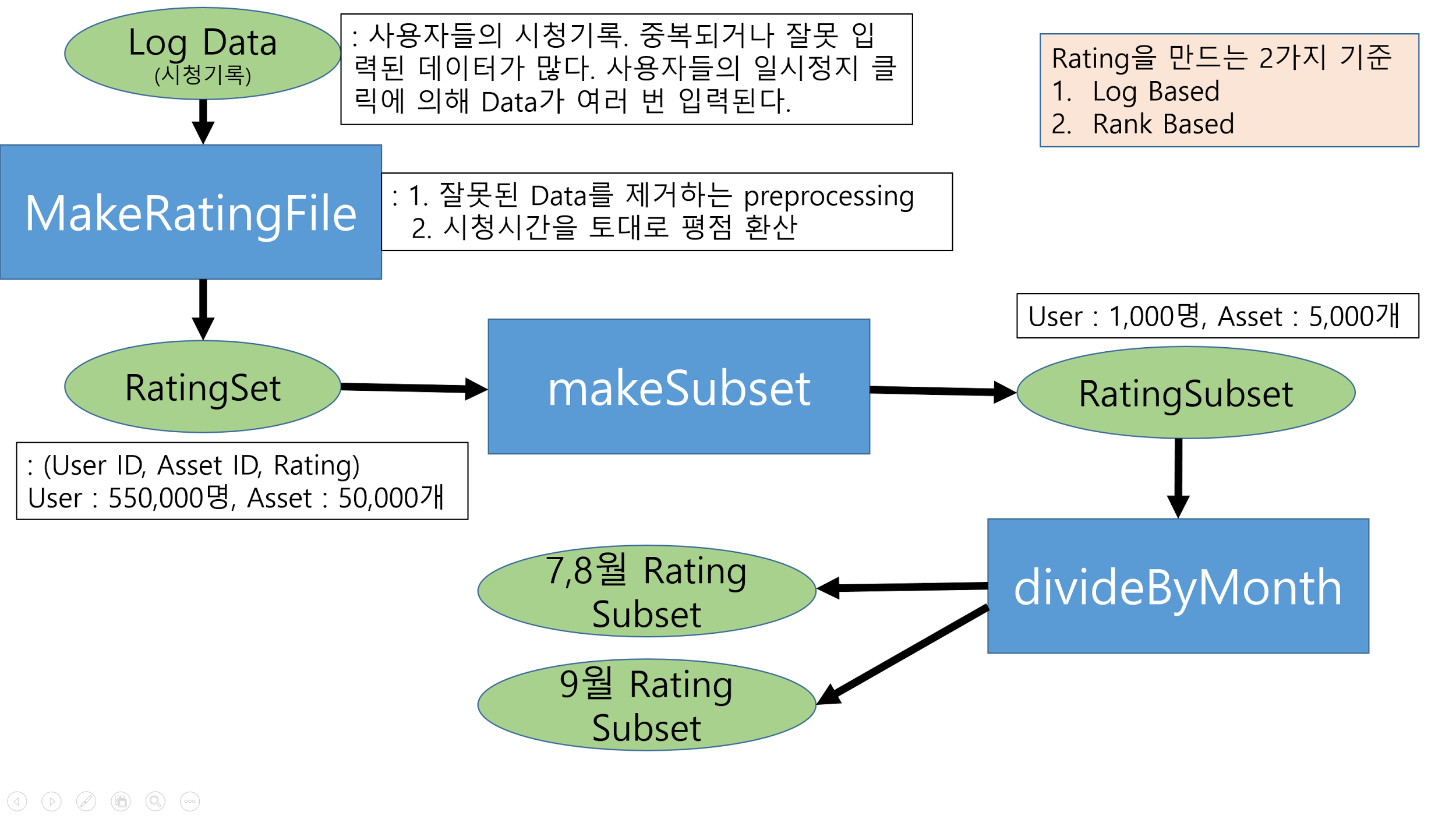
VOD 로그 데이터를 분석하여 올바르지 않은 데이터를 잘라낸다. 그 다음 시청 시간의 절대값 혹은 시청 시간과 각각의 컨텐츠 길이와의 비율 등 여러가지 기준을 통해 몇 가지 환산 방법을 고안한다. 각각의 환산 방법을 활용하여 나온 결과가 얼마나 올바르게 추천하는지를 비교하여 제일 나은 방법을 찾는다.

여기서 추천이 얼마나 올바르게 되었는지는 실제 두 달치 VOD 로그 데이터를 입력으로 받아서 나온 추천 결과가 그 다음 한 달치 VOD 시청 기록과 얼마나 일치하는지를 보고 판단한다.

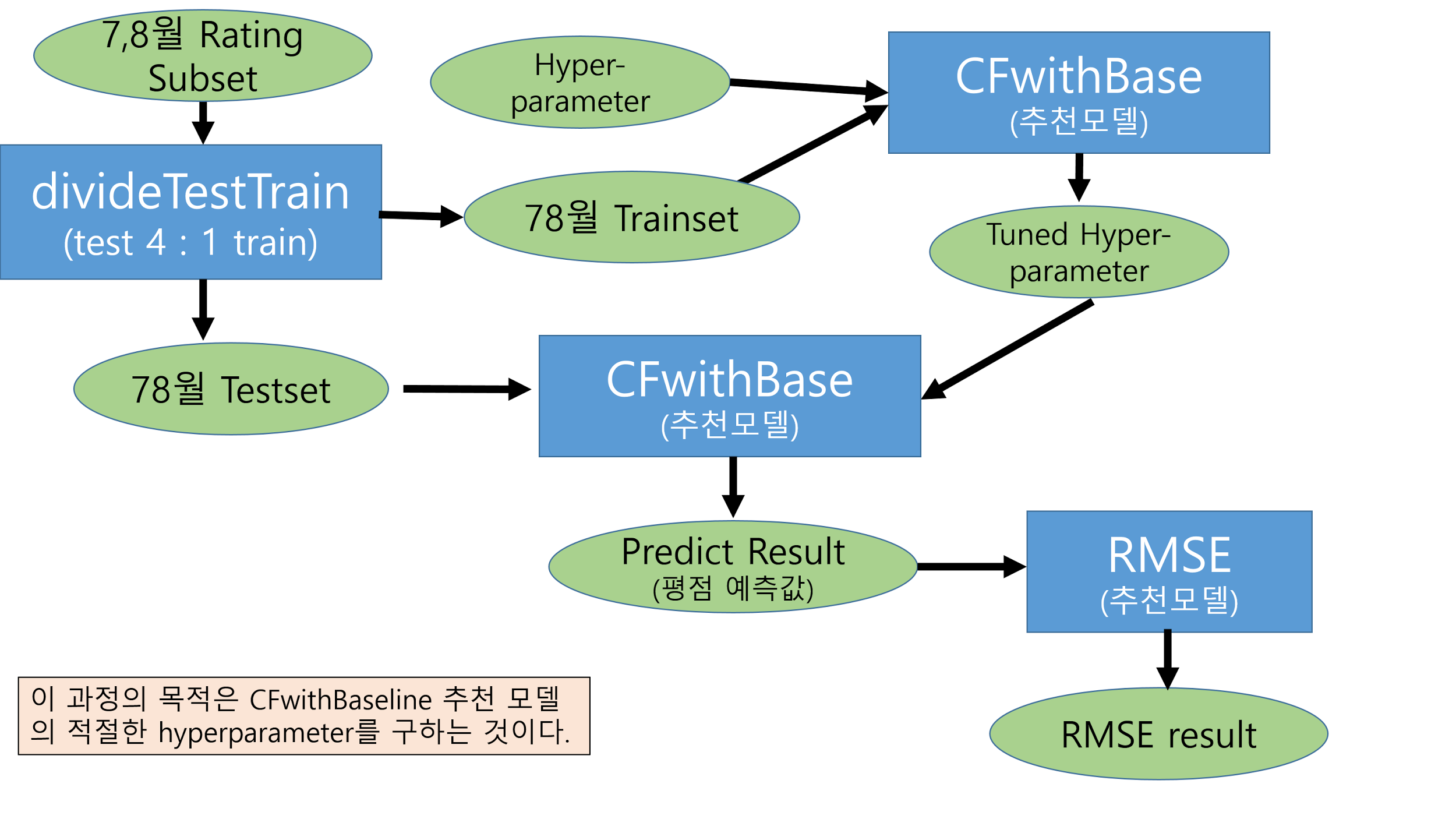
# **Project Architecture**

## **Architecture Diagram**

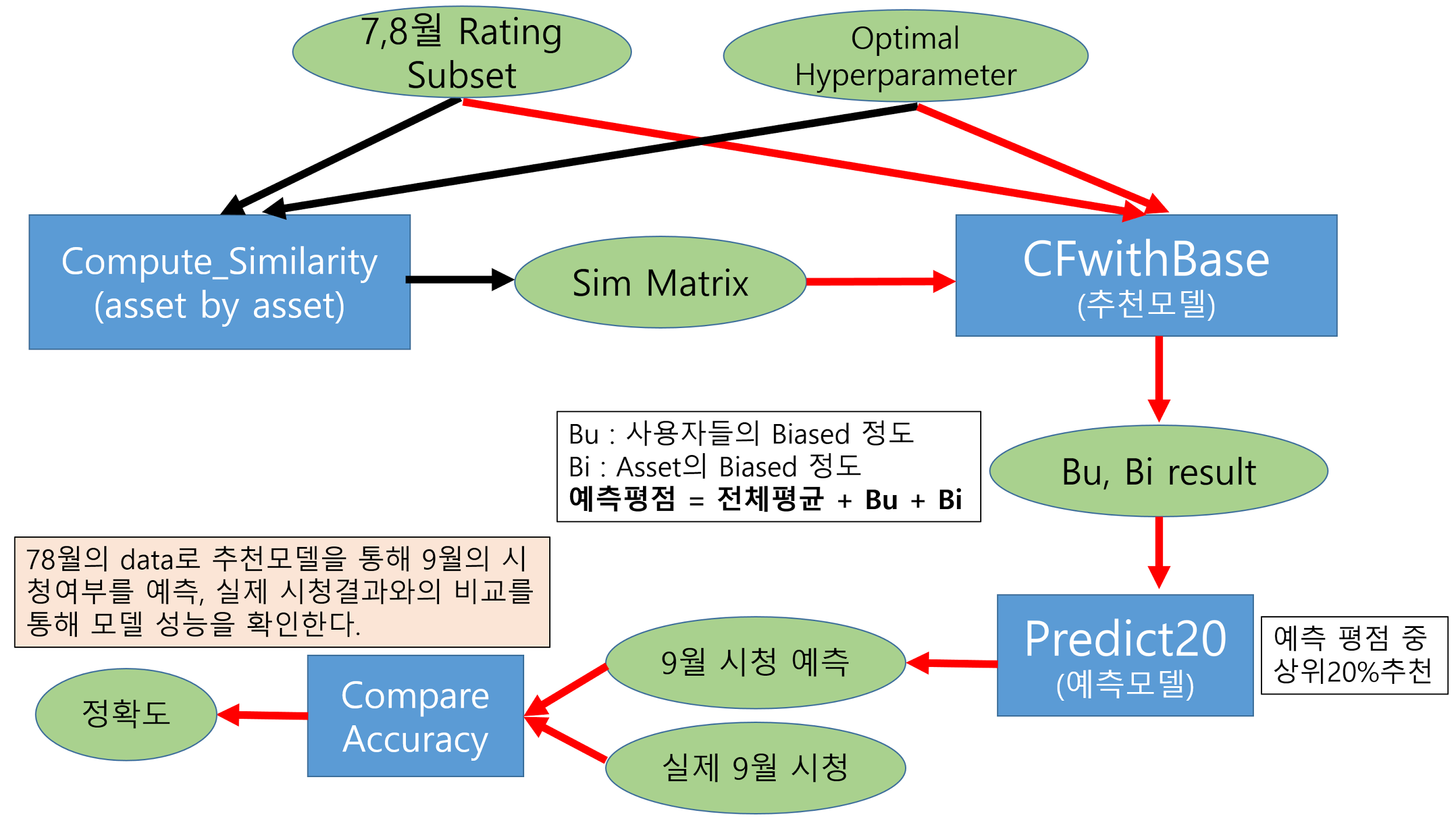
1. Log Data의 Preprocessing, 평점 Data의 생성, Subset 추출



1. 추천 모델에 필요한 HyperParmeter의 튜닝 과정



(3) 예측 평점을 통해 9월의 시청여부 예측, 실제 기록과의 비교



## **Architecture Description**

1. **Log Data의 Preprocessing, 평점 Data의 생성, Subset 추출**

**<1> MakeRatingFile** (Preprocessing + 평점 환산)

**Preprocessing** : Implicit Data(Log Data)에서의 중복된 Data를 제거하고, 잘못된 log들을 제거한다. 잘못된 Log는 Asset의 Runtime이 0으로 되어있거나, 분과 초가 잘못되어 있는 Data 등이 있었다.

**평점 환산** : Preprocessing을 통해 필터링된 Data를 (UserID, AssetID)로 groupby하여 시청시간의 합을 구한다. 각 Asset 별로 이 시청시간의 합의 Median을 구한다. RAW RATING은 각 시청자의 해당 Asset에 대한 시청시간의 합을 앞에서 구한 Median 값으로 나눈 값이다. 이 RAW RATING을 실제 알고리즘에 사용할 평점으로 환산해야 한다. 평점의 환산 기준은 두가지가 있다.

**(1)** **by Rank** : 전체 (UserID, AssetID, RAW RATING)의 Data set을 정렬한 뒤, 1~5까지의 평점을 구간을 나누어 등수대로 부여하였다.

**(2) by Log** : 전체 (UserID, AssetID, RAW RATING)의 Data set에서 RAW RATING의 Log값을 구하였다. RAW RATING의 분포가 0.02~40 정도이기 때문에, Log값을 취하면 -2에서 2 사이의 값이 나온다. 여기에 3을 더해주면 1에서 5까지의 평점이 생성되며, 대부분의 RAW RATING이 1근처에 있기 때문에 3 근처를 평균으로 하는 정규분포 형태의 평점분포가 구해진다.

두 가지 평점 기준 중 9월에 대한 시청예측을 더 잘하는 평점 환산 기준을 택하는 것이 이번 연구의 핵심 목표이다. 따라서 (2), (3)번 과정을 두 평점 Data에 대해 실행한 뒤 예측 결과를 비교하였다.

**<2> makeSubset, divideByMonth**

**makeSubset :** 전체 Data가 너무 크기 때문에 예측 모델을 돌리기에 유의미한 정도의 크기의 subset을 뽑는 모듈이다. 시청건수가 높은 User 1000명, 시청건수가 높은 Asset 5000개에 대한 subset을 추출하였다.

**DivideByMonth :** Subset을 다시 7,8월에 시청한 Subset과 9월에 시청한 Subset으로 나눈다. 이 과정은 이후에 7,8월의 Data만을 통해 9월의 시청여부를 예측하기 위해 필요한 과정이다.

1. **추천 모델에 필요한 HyperParmeter의 튜닝 과정**

**divTestTraining** **, CFwithBase, RMSE : (중간발표 때 구현됨)**

CFwithBase 모듈을 돌리기 위해서는 몇 가지 HyperParameter가 필요하다. HyperParameter에는 Learning Rate, Lambda(Regularization Parameter) 등이 있다. 적절한 HyperParameter 값을 구하기 위해 7,8월의 평점 Data를 Train, Test의 4:1 비율로 나누었다. 이후 CFwithBase 추천모델에서 Train Set을 가지고 Test Set에 대한 평점예측을 한 뒤, RMSE 값을 구하고, RMSE값이 낮은 HyperParameter의 조합을 선택한다.

1. **예측 평점을 통해 9월의 시청여부 예측, 실제 기록과의 비교 과정**

**<1> ComputeSimilarity, CFwithBase : (중간발표 때 구현됨)**

먼저 ComputeSimilarity 모듈을 통해 Asset간의 Similarity Matrix를 구한다. 이 Matrix와 앞에서 구했던 Optimal HyperParameter, 그리고 7,8월의 Rating Dataset을 가지고 CFwithBase 추천모듈을 돌려 Bi, Bu값을 구한다.

Bi,Bu는 각 영화와 사용자가 전체평균평점으로부터 얼마나 Biased 되었는지에 대한 값을 Vector처럼 가지고 있는 값이다. 예측평점은 평균평점 + Bi + Bu값으로 구할 수 있다.

**<2> Predict20, ComputeAccuracy**

**Predict20** : 약 80명의 사용자에 대해 각 사용자의 모든 Asset에 대한 예측 평점을 구한다. 그리고 각 사용자마다 높은 평점을 부여할 것으로 예측되는 상위 20프로의 Asset을 기록한다. 이 Asset들은 9월달에 해당 사용자가 시청할 것으로 예측된다.

**ComputeAccuracy :** 앞에서 구한 상위 20프로의 Asset의 집합 가운데 실제 9월에 시청된 Asset이 얼마나 포함되어 있는지 구한다. 이 값이 높을수록 미래를 예측하기에 좋은 평점 기준이라고 결론내릴 수 있다.

# **Implementation Spec**

## **Input/Output Interface**

사용자는 시청기록을 최초입력으로 제공해야한다. 시청기록은 user id, asset id, 시청 시간 등의 정보를 포함해야한다. 이 시청기록을 우리의 모듈 MakeRatingFile에 input으로 주면 output으로 user가 asset에 내린 환산된 평점이 나온다. 환산된 평점은 필요에 따라 다른 여러 explicit feedback을 바탕으로 하는 알고리즘에 적용이 가능하다. 우리의 경우는 collaborative filtering with bias model에 적용을 하였다. explicit rating을 divideTestTrain모듈에 적용하여 train set과 test set을 얻을수 있으며 train set과 test set을 이용하여 적절한 hyper parameter(n과 lambda)를 구할 수 있다. 이 과정은 사용자에 의해 행해져야 한다. 적절한 hyper parameter가 정해지면 다시 CFwithbase.py를 이용해서 item bias, user bias, rating mean을 구할 수 있으며 compute\_sim.py를 이용해서 similarity matrix를 구할 수 있다. 마지막으로 CFwithbase\_pred.py와 구해진 bias, similarity matrix를 이용하면 특정 user에 대해서 asset을 추천할 수 있다. 즉 log data제공 -> MakeRatingFile 모듈 -> hyper parameter tuning -> CFwithbase.py -> CFwithbase\_pred.py모듈의 과정을 거쳐 특정 user에 대해서 asset추천을 할 수 있다.

## **Inter Module Communication Interface**

MakeRatingFile 모듈은 log data를 input으로 받아서 user id, asset id, rating의 csv형식의 환산된 rating정보를 output으로 낸다. divideTestTrain 모듈은 csv형식의 환산된 rating정보를 input으로 받아서 random하게 8:2의 비율로 train set과 test set으로 나누어 2개의 file을 output으로 낸다., compute\_sim 모듈은 divideTestTrain모듈에 의하여 생성된 파일 중 train set file을 input으로 받아서 similarity matrix file을 output으로 낸다. CFwithbase 모듈은 train set과 similarity matrix을 input으로 받아서 item bias, user bias, rating mean을 각각 file output으로 낸다. 마지막으로 CFwithbase\_pred는 item bias, user bias, mean, similarity matrix와 user id를 input으로 받아서 최종 추천 asset의 이름을 콘솔에 출력해준다.

## **Modules**

1. 별점 환산 모듈
2. Data Preprocessing + Rating 변환

Log Data(Implicit Data)에는 잘못된 Data가 매우 많다. 따라서 평점으로 변환하기 이전에 이 Data를 제거하는 preprocessing 과정이 필요하다.

Rating으로의 기본적인 변환 과정은 다음과 같다.

1. (UserID, AssetID)로 시청기록을 groupby, 시청시간의 합을 구한다.
2. (UserID, AssetID, DurationSum)에서 RAW\_RATING (= DurationSum / Asset의 시청시간 중앙값)를 구한다.
3. RAW\_RATING이 평점계산의 틀이 된다.

<평점 환산 기준>

1. **by Rank** : RAW\_RATING의 크기에 따라 (UserID, AssetID, RAW\_RATING) 를 sorting하고, 구간을 나누어1에서 5까지의 균등한 값을 평점으로서 부여한다.

2. **by Log** : RAW\_RATING의 분포는 0.02~40이다. 그리고 대부분 1 근처에서 분포하고 있다. 따라서 Log값을 취하면 0을 평균으로 하며 -2에서 2 값 사이에서 정규분포의 형태를 띠는 값을 구할 수 있다. 여기에 3을 더해주면 1~5까지의 평점이 구해진다.

1. **makeSubset :** 전체 Data(User 550,000, Asset 50,000)에서 Subset(User 1000, Asset 1000)을 뽑았다.
2. **divideByMonth** : Subset을 7,8월의 기록에 해당하는 것과 9월의 시청기록에 해당하는 것으로 나누었다.
3. 추천 알고리즘 모듈

Netflix data를 활용해 실험해 본 결과 여러 추천 알고리즘 중에 bias-based와 colaborative filtering을 같이 사용하는 것이 예측 정확도와 실행 시간에 있어서 가장 좋았다. 따라서 우리의 추천 모듈은 colaborative filtering과 bias-based 알고리즘을 같이 사용하였다.

먼저 bias-based 모델을 이용하여 user bias와 item bias를 training시킨다. 이 부분에서 overfitting을 막기 위해 regularization parameter가 들어간다. 이는 사용자가 튜닝해야 하는 hyper paramter이다.

그 다음 각각의 컨텐츠에 매겨진 별점을 각 컨텐츠를 사용자 수 만큼의 factor를 갖는 벡터로 만든다. 만들어진 벡터를 normalize 한 후 이를 이용하여 모든 컨텐츠의 모든 컨텐츠에 대한 cosine similiarity를 구하고 matrix로 저장한다. 여기서 시청 횟수가 특별히 적은 사용자가 similiarity 값을 왜곡하는 것을 방지하기 위해 reliability parameter가 들어간다. 이 또한 사용자가 튜닝해야 하는 hyper paramter이다.

similiarity matrix와 user/item bias가 만들어졌으면 이를 이용하여 내가 원하는 사용자와 컨텐츠 쌍에 대해 예측 별점을 만들어 낼 수 있다.

1. 평가 모듈

7월과 8월의 로그 데이터로부터 나온 parameter와 similiarity matrix를 이용하여 모든 컨텐츠에 대해 별점을 예측한다. 그 다음 별점이 높은 순서로 20%를 뽑는다. 이 예측 목록 안에 사용자가 실제로 9월에 시청한 컨텐츠가 얼마나 포함되어 있는지 정답률을 계산한다. 정답률이 높을 수록 더 좋은 환산 방법이 된다.

# **Solution**

## **Implementations Details**

i) Rating( Explicit Data) 환산 모듈

1. Preprocessing( 잘못된 Data 제거) + Rating 환산 과정
2. 중복된 Data : (USER\_ID, ASSET\_ID, EVENT\_TIME)이 동일한 서로 다른 log data가 존재한다는 것을 확인했고, 이를 제거하였다.

결과 : 23,451,732 -> 21,234,059 log records  **(9.46% records dropped)**

(2) (USER\_ID, ASSET\_ID)로 groupby하여 각 사용자가 각 컨텐츠를 본 기록을 모두 합쳤다.

결과 : 21,234,059 -> 10,252,942 log records (51.71% records reduced)

(3) Asset\_Meta에서 run\_time이 0인 asset들에 대한 log record 제거

결과 : 10,252,942 -> 10,236,116 log\_records **(0.16% records dropped)**

(4) 잘못된 Watched\_Time이 입력된 Data 제거 : log data record는 일괄적으로 minute 단위로 입력되는데, 중간중간에 second 단위로 입력되어 혼란을 초래하는 log data들이 있다. 이를 일일이 구분해 낼 수 없어 (Watch\_Time/Asset의 길이)의 값이 너무 크면 제거하였다. 이 값을 기준으로 sorting한 뒤, 10,000,000 개의 record만 남겼다.

결과 : 10,236,116 -> 10,000,000 log records **(2.31% records dropped)**

(5) 한 asset을 40번 이상 시청했거나 0.02번 이하로 시청한 record는 잘못 찍힌 로그에 의한 의미 없는 시청기록이라고 판단하여 제거하였다. 그 이유는 분에 해당하는 기록이 초로 입력된 데이터가 있기 때문이고, 1/60이 약 0.02이기 때문이다.

결과 : 10,000,000 -> 9746638 log records **(2.5% records dropped)**

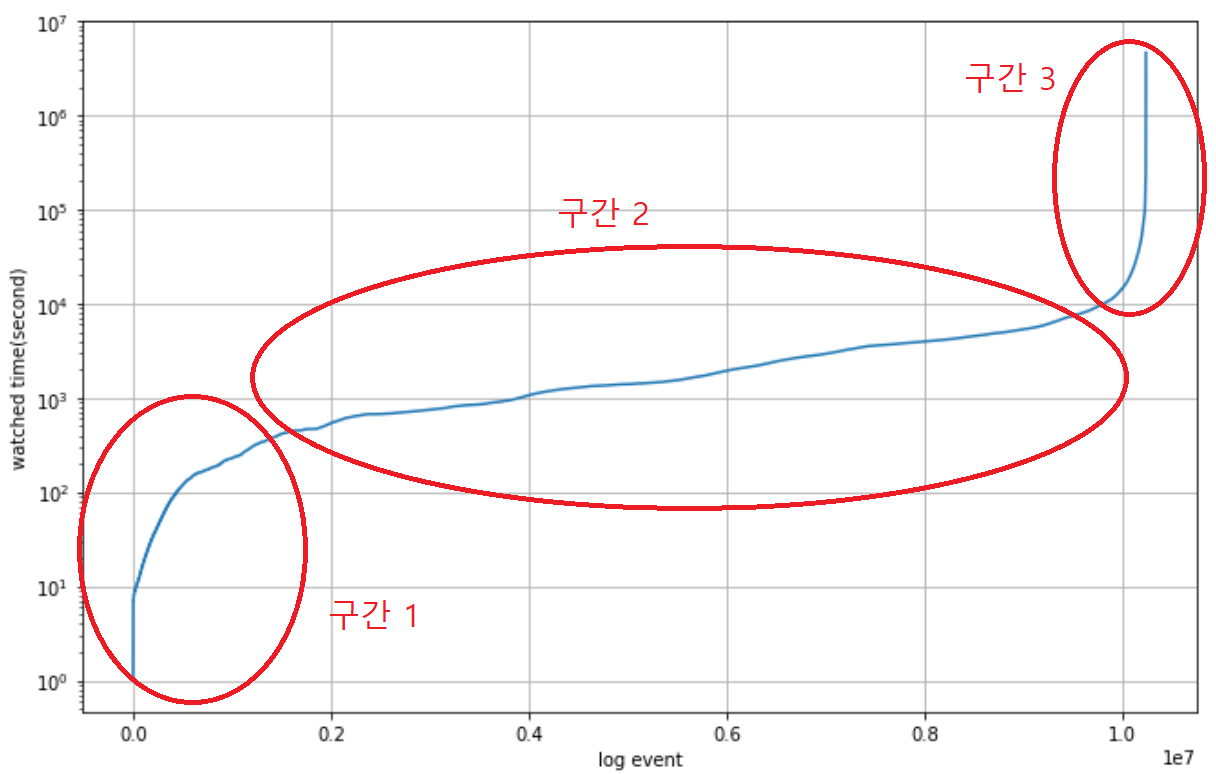
2. Rating의 환산

1번 (preprocessing) 과정을 통해 (Asset, User)단위로 record를 grouping하였다. 따라서 하나의 record는 (USER\_ID, ASSET\_ID, 시청시간의 합)의 묶음이다. 이를 이제 별점으로 환산해야 하는데 환산 방법의 선택 이유와 과정은 아래와 같다.

1. **상대시간 vs 절대시간**

시청시간의 합이란 시청자가 해당 Asset을 7~9월 동안 시청한 시간의 총 합이다. 이를 편의상 Duration이라고 하겠다. 이 논문에서 상대시간이라 함은 Asset의 길이에 대한 시청시간의 상대적인 비율을 의미한다. 상대시간 변환 방법에서는 Duration을 Asset의 runtime으로 나누고, 이 시청 비율을 가지고 평점을 만드는 것이다. 하지만 이 방법을 사용할 경우, 일부 시청길이가 짧은 asset들(동요, 어린이 학습 만화)에 대해 2회 이상 시청한 사용자가 너무 많기 때문에, 1회 온전히 시청한 사용자들의 평점 이 너무 낮게 나오는 현상이 발생한다. 따라서 상대시간 변환방법은 사용하지 않았다.

절대시간은 Duration을 그대로 평점에 반영하는 것이다. 이 Duration을 Asset의 시청길이의 중앙값으로 나누고, 이것을 RAW\_RATING이라고 했다. 이렇게 구한 RAW\_RATING의 분포를 그리자 아래 그림과 같은 분포가 이루어졌다.



이렇듯 분포가 일정하지 않기 때문에, 구간1, 3에는 적은 범위의 평점 (ex) 1~1.5점, 4.5~5점), 구간 2에는 넓은 범위의 평점을 부여해야 한다.

**(2) LOG 변환 방법 vs RANK 변환 방법**

위에서 언급한 조건을 만족하기 위해서 두가지 변환 방법을 구상해 보았다.

<1> LOG 변환 방법에서는 위의 RAW\_RATING 값에 대해 log값을 취한다. 이 경우에 약 -2~2의 값이 나오게 되고, 여기에 3을 더하면 1에서 5점 사이의 평점으로 변환하는 결과를 얻게 된다. Log값을 취한 이유는 급격한 기울기를 완화하는 효과를 얻기 위해서이다.

<2> RANK 변환 방법에서는 구간별로 평점 범위를 부여하고, 각 범위 내에서 등수대로 평점을 부여하는 방식이다. 여기서는 구간 1에서 1~1.5점을, 2에서 1.5~4.5점을, 3에서 4.5~5점을 부여하였다.

이후 두가지 평점 변환 모델의 비교를 통해 무엇이 더 좋은 평점 전환 방법인지 분석하는 것이 이 논문의 중요한 목표 중 하나이다. 그 비교 기준은 어떤 평점방식을 통해 7,8월의 DATA를 가지고 9월달의 시청여부를 더 잘 예측할 수 있는지이다.

ii) 추천 알고리즘 및 평가 모듈

이 모듈에서는 hyper paramter로 n과 lambda를 받는다. n은 asset간의 similairy를 계산할 때 이용되는 hyper parameter이다. 먼저 asset1과 asset2를 시청한 user를 기준으로 rating vector를 구 하고 두 vector간 cosine similarity를 구한다. n은 공통시청 user의 수에 따라 구해진 cosine similarity를 수용하는 정도에 대한 hyper parameter이다. lambda는 train set에 대한 over fitting을 막기 위한 hyper paramter로 lambda가 0일 경우 극단적으로 train set에 대해서 fitting이 되어 test set에 대해서는 잘 예측하지 못하는 경우가 생길 수 있다.

우리의 추천알고리즘 모듈의 이름은 CFwithbase와 CFwithbase\_pred이다. CFwithbase는 lambda, learning rate, trainset을 input으로 받고 training을 진행한다. baseline model에서와 같은 방법으로 parameter를 tuning하며 최종으로 item bias, user bias parameter를 output으로 내놓는다. CFwithbase\_pred는 user id, item bias, user bias, similarity matrix, 78월 시청기록을 input으로 받는다. similarity matrix와 user bias, item bias를 이용하여 해당 유저가 모든 asset에 대해서 내릴 평점을 예측한 후, 예측한 평점이 높은 순서대로 20%의 평점에 대한 asset을 추려낸다. 해당 유저가 9월에 본 시청 기록과 앞에서 추려낸 asset을 이용하면 정답률을 구할 수 있다.

iii) 데모용 프로그램

데모를 시연하기 위하여 필요한 데이터는 user bias, item bias, rating mean(CFwithbase.py를 통해 구할 수 있음), 7,8월 시청기록, 9월 시청기록, similarity matrix이다. 데모를 시연하기 전에 미리 train을 완료하여 위의 것들을 준비한 후 show\_78log.py, CFwithbase\_pred.py, show\_9log.py를 실행한다. show\_78log.py는 7,8월 user id를 입력으로 받아서 해당 user가 7,8월에 어떤 asset들을 시청하였는지 asset의 이름을 콘솔에 출력해준다. 존재하지 않는 user id를 입력할 경우 error message를 띄워준다. CFwithbase\_pred.py에서 user id를 받아서 user bias, item bias, rating mean, similarity matrix를 이용하여 해당 유저에게 실제 asset 10개를 추천해 주고 그 이름을 콘솔에 출력해준다. show\_9log.py에서는 user id를 받아서 해당 유저가 9월달에 시청한 asset의 이름을 출력해준다. 즉 특정 user id에 대하여 show\_78log.py, CFwithbase\_pred.py, show\_9log.py를 실행하면 7,8월에 유저가 본 asset들, 우리의 추천 asset, 실제 유저가 9월에 본 asset을 보여줌으로써 결과를 확인할 수 있다.

## **Implementations Issues**

크게 두가지 측면에서 issue가 있었다. 첫번째는 속도의 문제, 두번째는 통계적 기법 등을 이용한 별점의 환산문제였다.

추천알고리즘의 가장 큰 문제는 실행시간이 너무 길다는 것이었다. input으로 사용한 log data는 2000만 line이 넘어가는 거대한 데이터이며 특히 CF model에서 asset간의 similarity를 구하려면 asset의 수의 제곱에 비례한 계산시간이 필요하다. asset이 50,000개일 때 12,00,00,000개 pair에 대한 similarity를 구해야하는 것이다.

이 과정에서 python의 기본 자료형인 list를 사용하는 것 보다 numpy 패키지를 사용하는 것이 속도향상에 큰 도움이 되었다. 뿐만 아니라 거대한 iteration을 수행할 때 index에 대한 dictionary를 만들어 활용하면 iteration횟수를 줄일 수 있어 큰 도움이 됐다. 또한 pandas라는 대형 data를 다루는 패키지를 이용하면 file input을 읽어들이거나 특정 data를 추출하는데에 있어 실행시간을 크게 줄일 수 있었다.

두번째는 별점 환산방식을 고안하는 일이다. explicit feedback을 이용한 추천 알고리즘이 있지만 implicit feedback을 이용한 추천방법은 없다. 우리는 explicit feedback을 이용한 추천방법이 있는 것에 착안하여 implicit feedback을 explicit feedback으로 변환하기 위한 여러 방법들을 시도해 보았다. 하지만 단순 시청 기록으로 해당 유저가 asset에 대하여 긍정적인 생각을 가지고 있는지 부정적인 생각을 가지고 있는지 판단하는 것은, 실제 사용자의 가족 구성원과 생활 방식과 그에 따른 시청 패턴을 연구하지 않는 이상 단순한 어림짐작일 뿐이다.

우리가 선택한 방식은 데이터에서 답을 찾는 것이었다. 사람들이 각각의 컨텐츠를 본 시청시간의 중앙값을 기준으로 하면, 정규분포와 비슷한 모양을 보이는 분포를 구할 수 있었다.

# **Results**

## **Experiments**

i) regularization parameter에 따른 정답률 (환산 기준 : log, n = 10)

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| λ | 0.1 | 0.3 | 1 | 3 | 10 | 30 | 50 | 70 | 100 |
| 정답률(%) | 46.23 | 46.27 | 46.3 | 46.41 | 46.65 | 46.99 | 47.21 | 47.24 | 47.17 |

ii) reliability parameter에 따른 정답률 (환산 기준 : log, λ = 70)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| n | 0 | 5 | 10 | 15 | 20 |
| 정답률(%) | 41.39 | 39.76 | 47.24 | 40.16 | 40.29 |

iii) 환산 기준에 따른 정답률 (λ = 70, n = 10)

|  |  |  |
| --- | --- | --- |
| 환산 기준 | log | rank |
| 정답률(%) | 47.24 | 29.66 |

## **Result Analysis and Discussion**

User/item bias에 대한 regularization parameter는 70이 제일 좋은 것으로 나왔다. 이는 user 또는 item bias가 크지 않는게 좋다는 의미다. Reliability parameter는 10이 제일 좋은 것으로 나왔다. 이는 시청 횟수가 10보다 작은 사용자에 대해서는 similarity를 계산할 때 절반보다 낮게 신뢰하겠다는 의미이다. log 기준으로 별점을 환산하는 것이 rank 기준으로 하는 것보다 정답률이 높았다. 어느 정도 사용자의 성향을 반영하는 데 성공했다고 볼 수 있다. 프로젝트를 계속 진행한다면 더욱 좋은 성과를 낼 수 있을 것이다.

# **Division & Assignment of Work**

|  |  |
| --- | --- |
| 항목 | 담당자 |
| 기존 알고리즘 공부 | 전원 |
| Raw log data pre-processing | 최윤서 |
| CFwithbase Module | 김지훈 |
| Compute similarity matrix module | 설재완 |
| Demo program | 전원 |

# **Conclusion**

여기까지 우리는 Implicit data를 이용해 추천 시스템을 운용하는 한 가지 가능성을 제시했다. 그 결과 사용자의 성향을 반영한 결과가 나왔다. 이는 implicit data를 적절히 explicit data로 환산한 후 explicit data를 이용하는 알고리즘을 적용할 수 있음을 보여준다. Implicit data를 explicit data로 환산하는 적절한 방법만 있으면 된다. 환산하는 방법은 우리의 방법 외에도 더욱 많은 방법이 있을 것이다. 우리가 받은 log data에는 asset들을 분류하는 더 큰 기준인 series정보와 시청 날짜를 나타내는 event time정보도 있었는데, 이 두 정보도 환산에 활용한다면 더욱 좋은 환산방법을 만들어 낼 수도 있을 것이다.

현실세계에는 Explicit data가 없는 경우가 많다. 음악 스트리밍 사이트에서 음악의 추천, 사이드바 광고에서 광고의 추천 등의 상황에서도 explicit feedback이 있는 경우보다 없는 경우가 더 많다. 우리의 프로젝트를 발전시켜 나간다면, 다양한 상황에서 implicit feedback을 이용하여 보편적으로 쓸 수 있는 추천 알고리즘을 만드는 것이 가능할 것이다.

# **[Appendix] User Manual**

## **시청 로그 제공**

사용자는 시청 로그를 최초의 input으로 제공해야한다. 시청로그는 user를 식별할 수 있는 id, asset을 식별할 수 있는 id, 특정 user가 특정 asset을 시청한 시간(duration)을 포함해야한다. MakeRatingFile모듈에 시청 로그를 input으로 주면 output으로 환산된 rating이 나온다. 환산된 rating의 양식은 user\_id, asset\_id, rating순서의 csv파일이다.

* + - 1. **Train set과 Test set으로 분리**

환산된 rating을 train set과 test셋으로 분리해야한다. 이는 이후에 hyper parameter tuning과 model training을 위해 필요한 과정이다. 1에서 얻은 rating을 divideTestTrain모듈에 input으로 주면 임의로 80%의 rating정보는 train set에, 20%의 rating정보는 test set에 분리되어 2개의 file output이 나온다.

* + - 1. **Hyper parameter tuning 및 model training**

최적의 추천을 위해 hyper parameter인 lambda와 n을 설정해야한다. lambda는 model이 train set에 over fitting되는 것을 방지하기 위한 것이고 n은 asset 사이의 similarity 를 구할 때, 공통 user수에 따라 차등되는 similarity를 적용하기위한 것이다. compute\_sim.py 모듈에 2에서 얻은 train set과 hyper parameter n을 input으로 주면 output으로 similarity matrix가 나온다. CFwithbase.py 모듈에 2에서 얻은 train set과 hyper paramter lambda를 input으로 주면 item bias, user bias, rating mean이 output file로 나온다. similarity matrix, item bias, user bias, rating mean과 test set을 rmse.py에 input으로 주면 test set에 대한 rmse가 나온다. n과 lambda를 적절히 바꾸면서 test set에 대한 rmse가 가장 낮게 나오는 n과 lambda를 설정하면 된다. 그리고 그 n과 lambda를 이용하여 마지막으로 compute\_sim.py모듈을 이용해서 similarity matrix를 만들고 CFwithbase.py 모듈을 이용해서 train 시키면 최종 item bias, user bias, rating mean이 나온다.

**4. 최종 추천**

3에서 구한 similarity matrix, item bias, user bias, rating mean과 CFwithbase\_pred.py모듈을 이용하면 최종 추천을 할 수 있다. CFwithbase\_pred.py모듈에 command line argument로 위의 4개의 파일을 입력하고 실행하면 ‘Enter USER ID: ’라는 문구가 콘솔에 출력되는데 이 때 user id를 입력하면 해당 user에 대한 추천 asset 10개의 이름이 콘솔에 출력된다. 만약 user id가 존재하지 않는다면 ‘USER ID does not exist’라는 에러 메시지가 출력된다.