책 GPT

-도서 추천 시스템-

16기 노연수, 민윤기

17기 강민채, 서지민, 우윤규

CONTENTS

1. 추천시스템이란?

2. DACON 도서 평점 예측
   - 데이터 전처리
   - 1st model
   - 2nd model

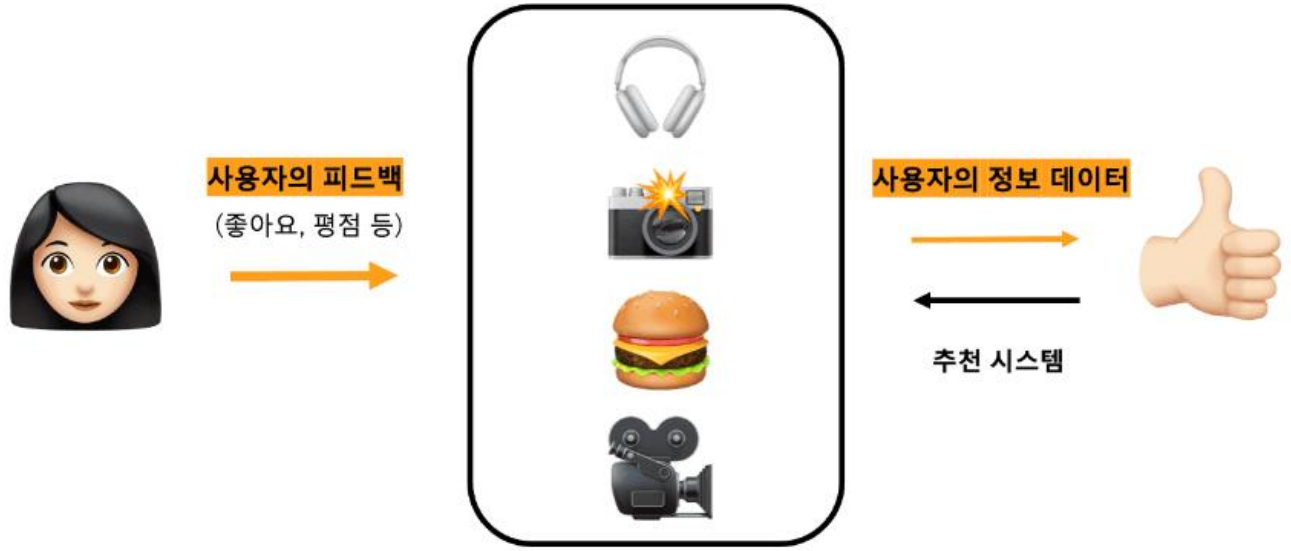
3. 컨텐츠 기반 모델
   - 모델링
   - 한계점 및 보완

4. 협업필터링 모델
   - 데이터 전처리
   - 클러스터링
   - 협업 필터링

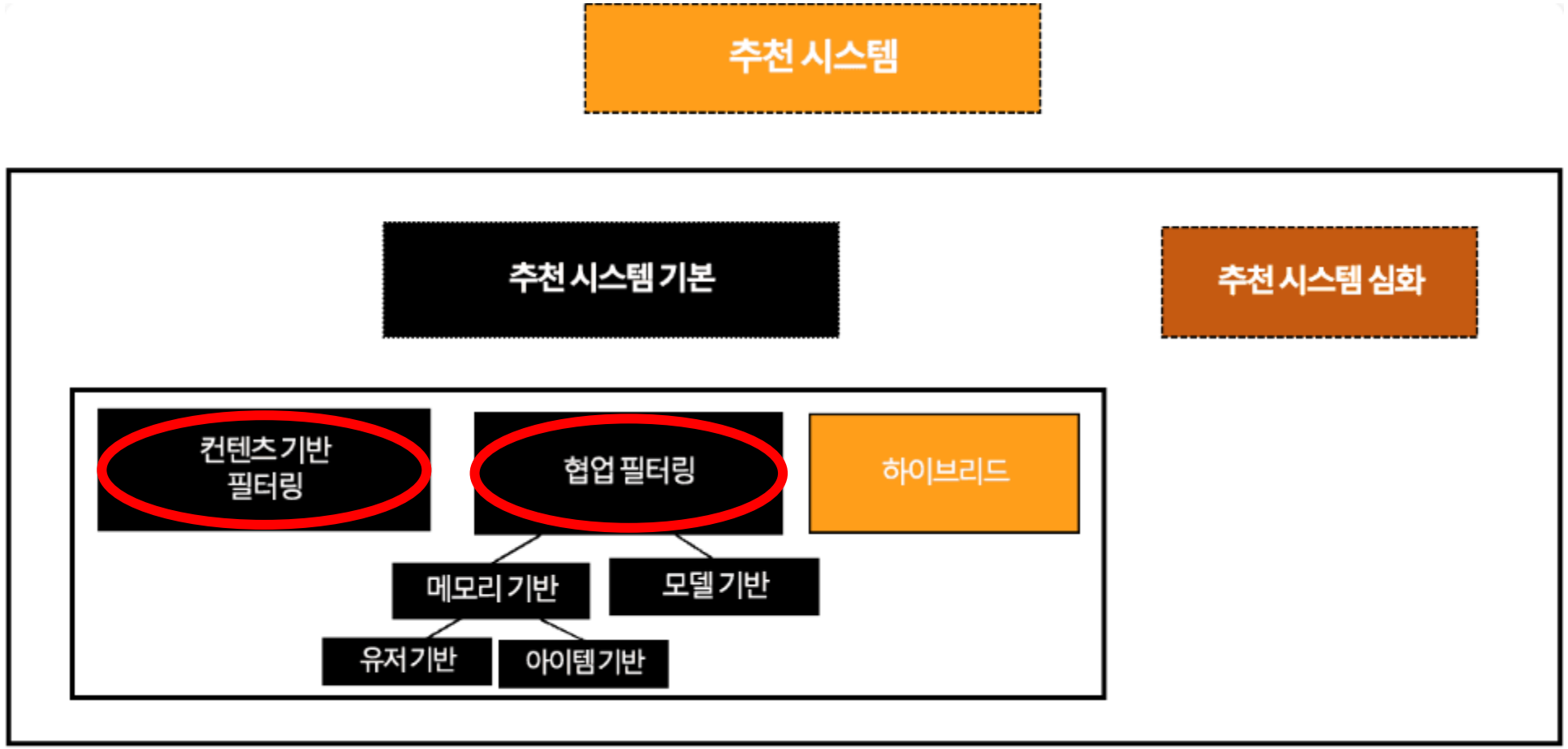
1. 추천시스템이란?

추천시스템이란?

- 사용자에게 상품을 제안하는 기술
- 과거 구매행동 분석을 통해 향후 구매 제품 예상



추천시스템 로드맵


추천 시스템
추천 시스템 기본
추천 시스템 심화
컨텐츠 기반 필터링
협업 필터링
하이브리드
메모리 기반
모델 기반
유저 기반
아이템 기반

2. DACON 도서 평점 예측

Task와 Performance index는?

[주제]

도서 추천 알고리즘 AI 모델 개발

[설명]

유저 정보와 도서 정보를 바탕으로,

유저가 부여한 도서 평점을 회귀 예측하는 AI 모델을 개발해야 합니다.

- Book-Rating : 유저가 도서에 부여한 평점 (0점 ~ 10점)
  - 단, 0점인 경우에는 유저가 해당 도서에 관심이 없고 관련이 없는 경우

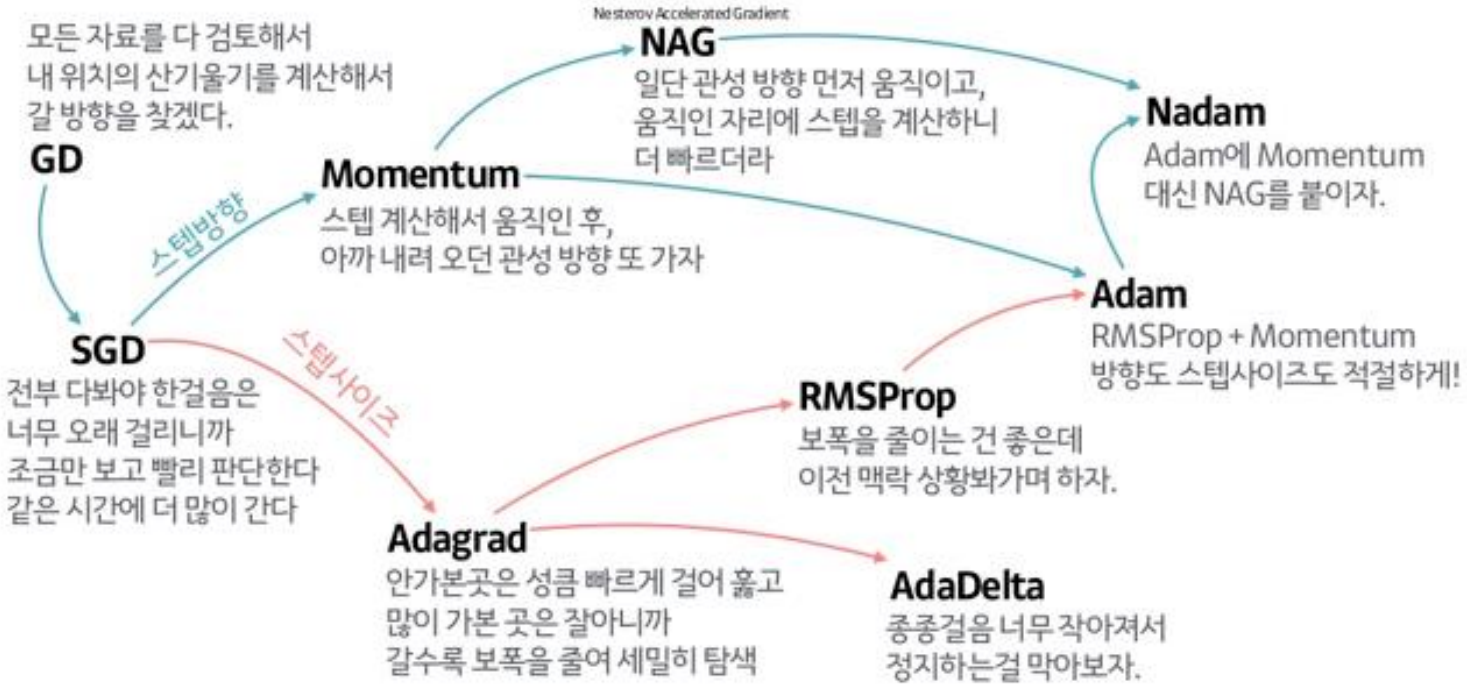
- 평가 산식 : RMSE (Root Mean Squared Error)

$$\mathrm{RMSE} = \sqrt{MSE} = \sqrt{\frac{\sum(\hat{y}-y)^2}{n}}$$

- 1st Model : Binary Classification ; BCELoss
- 2nd Model : Regression or Multi-Classification ; RMSE

Optimizer는?

- ADAM

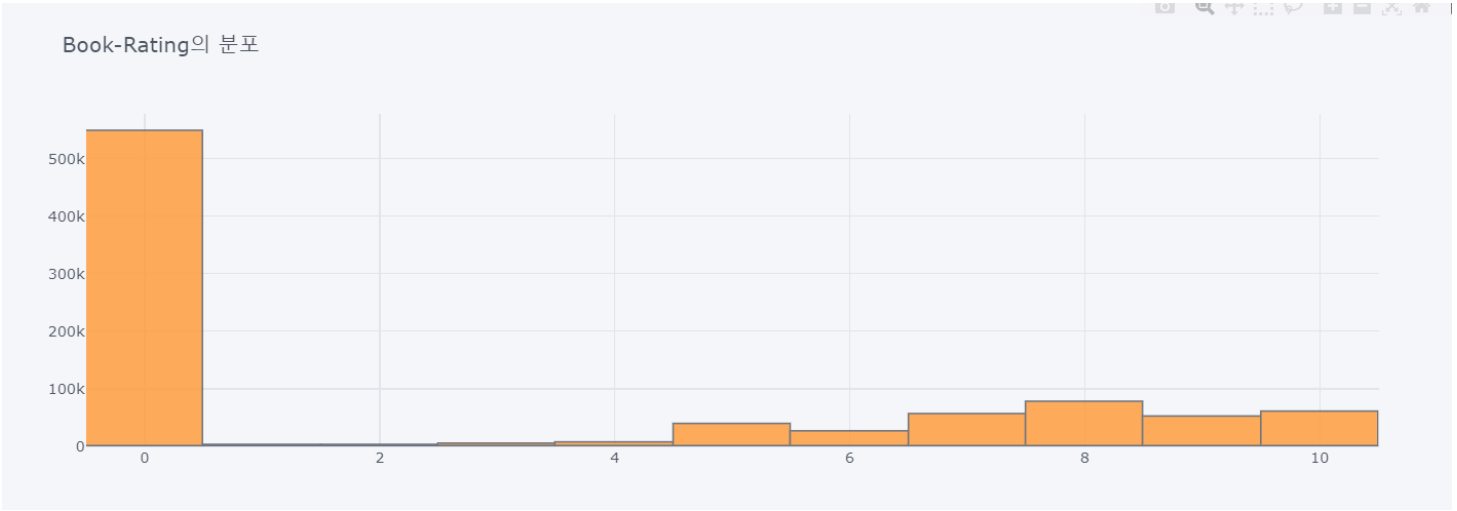


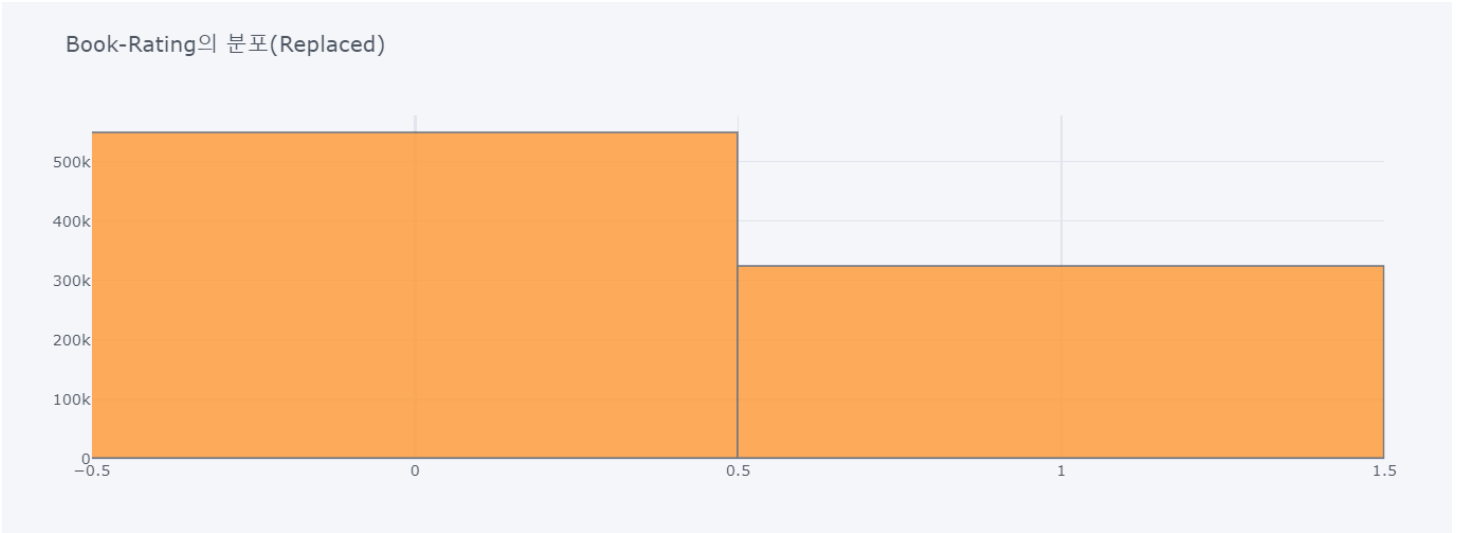
Optimizer 종류 // 출처 : https://www.slideshare.net/yongho/ss-79607172

데이터 전처리; Book-Rating

- 0 또는 1 (False or True)
- 첫번 째 모델에서 1 ~ 10을 1로 치환

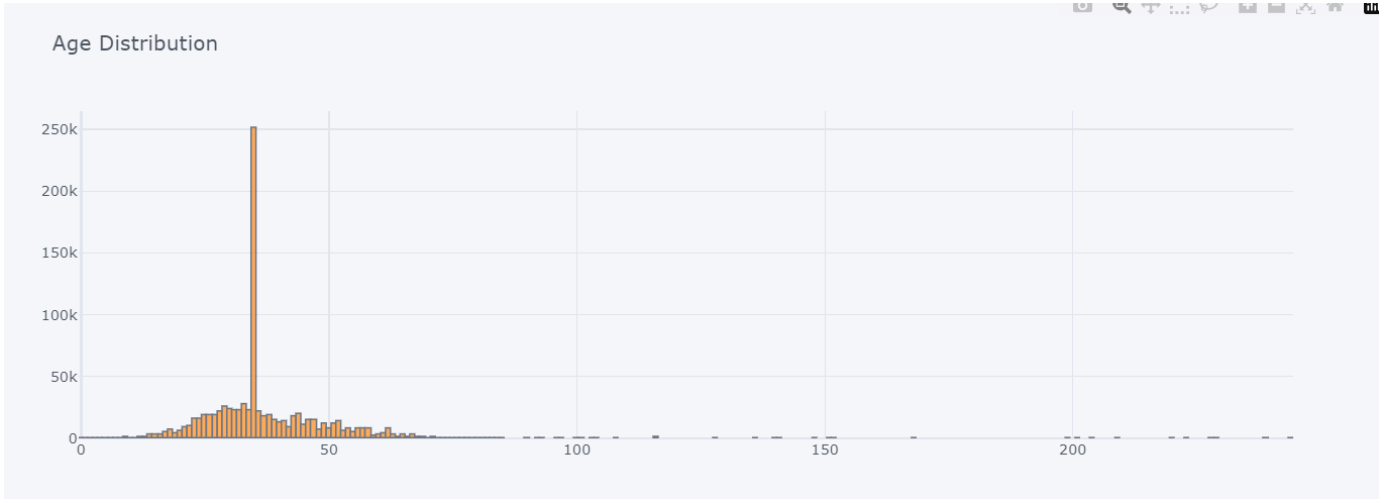
```
0      548804
1        1307
2        2019
3        4374
4        6462
5       38416
6       26670
7       55852
8       76971
9       50494
10      60024
Name: Book-Rating, dtype: int64
```





데이터 전처리; Age & Year-Of-Publication

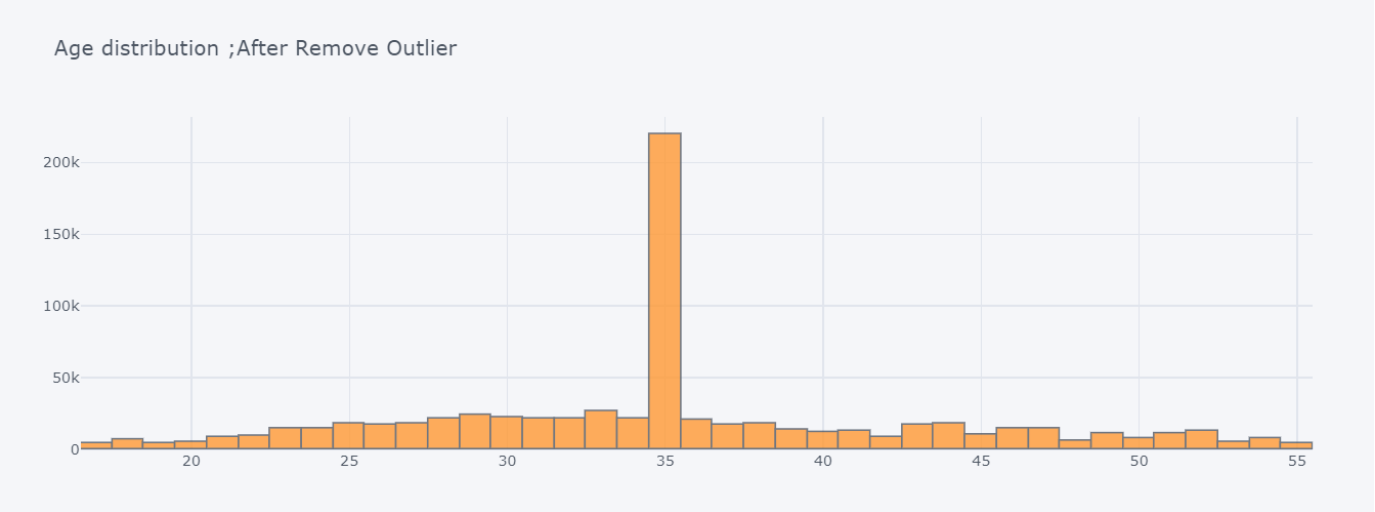
- Age 분포
- Year-Of-Publication 분포
- IQR을 활용하여 이상치 제거

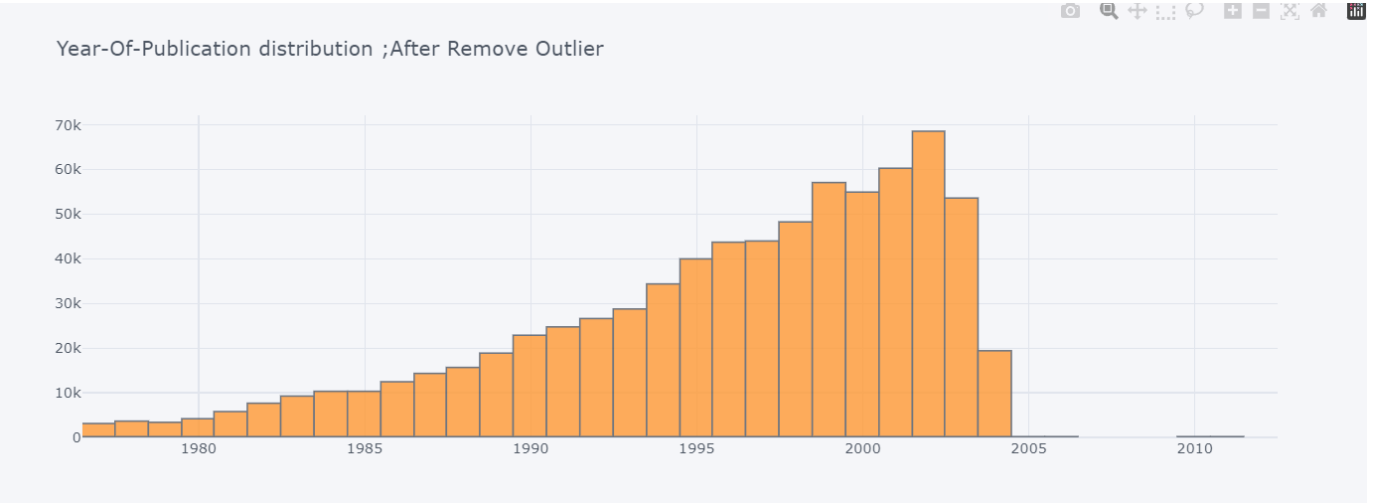



```
q1 = df_train_3['Age'].quantile(0.25)
q3 = df_train_3['Age'].quantile(0.75)
IQR = q3 - q1
upper = q3 + 1.5 * IQR
lower = q1 - 1.5 * IQR


# Year-Of-Publication
q1 = df_train_3['Year-Of-Publication'].quantile(0.25)
q3 = df_train_3['Year-Of-Publication'].quantile(0.75)
IQR = q3 - q1
upper_public = q3 + 1.5 * IQR
lower_public = q1 - 1.5 * IQR
```

데이터 전처리; Age & Year-Of-Publication

- 이상치 제거 후의 분포





데이터 전처리; Label Encoding & Test data

- User-ID, Book-Title, Publisher, Country, State, Town 를 Label Encoding
- 2가지 경우가 존재
  - Train data, Test data 둘 다 존재하는 경우
  - Test data에는 존재하지만, Train data에는 존재하지 않는 경우
- 2번째 경우를 모두 -1로 치환하고, 이를 독립적으로 예측

User-Idx	Title-Idx	Author-Idx	Publisher-Idx	Country-Idx	State-Idx	Town-Idx
0	0	0	0	0	0	0
0	1	1	1	0	0	0
0	2	2	1	0	0	0
0	3	3	2	0	0	0
0	4	4	3	0	0	0
...	...	...	...	...	...	...
83251	102893	46985	68	1	2	2
83252	32061	231	141	1	34	3823

CustomDataset & DataLodaer

- DL 모델에 적용시키기 위한 Custom Dataset을 다음과 같이 구성
- 구성된 Custom Dataset 기반으로 DataLoader을 만들어 DL 모델에 순차적으로 데이터를 제공
- 이러한 2개의 객체(Object)를 활용하여 Pytorch에서 효과적으로 학습 데이터를 모델에 제공가능

```
# 데이터셋 클래스 정의
class BookDataset(Dataset):
    def __init__(self, data):
        self.user_idx = data['User-Idx'].values
        self.Book_Title = data['Title-Idx'].values
        self.Book_Author =data['Author-Idx'].values
        self.Publisher = data['Publisher-Idx'].values
        self.Country= data['Country-Idx'].values
        self.State= data['State-Idx'].values
        self.Town= data['Town-Idx'].values
        self.rating = data['Attention'].values.astype(np.float32)

    def __len__(self):
        return len(self.rating)

    def __getitem__(self, idx):
        return self.user_idx[idx],self.Book_Title[idx], self.Book_Author[idx],self.Publisher[idx], self.Country[idx],
    self.State[idx], self.Town[idx],  self.rating[idx]
```

```
# DataLoader로 학습 데이터셋과 테스트 데이터셋 만들기
train_set = BookDataset(train_data)
test_set = BookDataset(test_data)
train_loader = DataLoader(train_set, batch_size=batch_size, shuffle=True)
test_loader = DataLoader(test_set, batch_size=batch_size, shuffle=False)
```

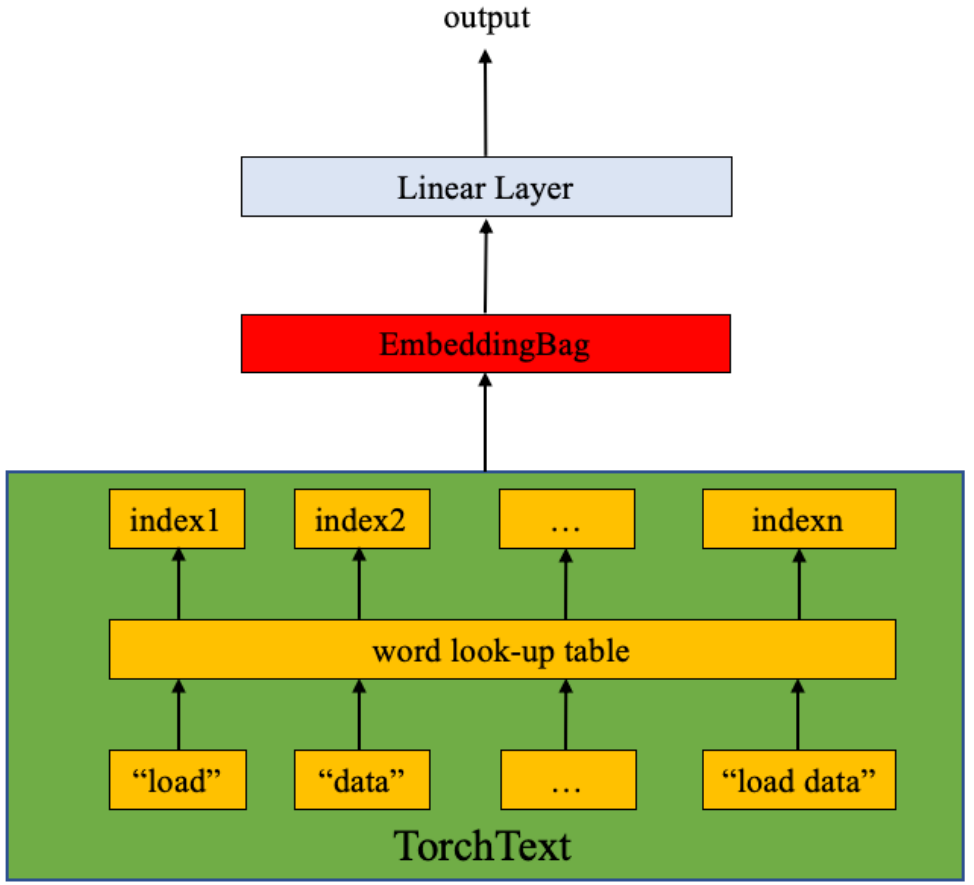
1st Model Structure

- 0 or 1

```python
# 모델 클래스 정의
class ContentBased(nn.Module):
    def __init__(self, num_users, num_items, num_author,num_publisher,num_country, num_state, num_town,
                 embedding_size = 16):
        super(ContentBased, self).__init__()
        self.user_embedding = nn.Embedding(num_users, embedding_size)
        self.item_embedding = nn.Embedding(num_items, embedding_size)
        self.author_embedding = nn.Embedding(num_author, embedding_size)
        self.publisher_embedding = nn.Embedding(num_publisher, embedding_size)
        self.country_embedding = nn.Embedding(num_country, embedding_size)
        self.state_embedding = nn.Embedding(num_state, embedding_size)
        self.town_embedding = nn.Embedding(num_town, embedding_size)

        self.numeric_layer = nn.Linear(2, embedding_size)

        self.fc1 = nn.Linear(embedding_size*8, 128)

        self.fc2 = nn.Linear(128, 64)

        self.fc3 = nn.Linear(64, 32)
        self.fc4 = nn.Linear(32, 16)
        self.fc5 = nn.Linear(16, 1)

        self.relu = nn.ReLU()
        self.dropout = nn.Dropout(0.2)
        self.sigmoid = nn.Sigmoid()
```

1st Model Structure

- Text To Vector (Text Embedding)
- 수치형 데이터
- 2가지 형태의 데이터를 벡터화 시키고, EmbeddingBag층을 활용하여 하나의 값으로 합침

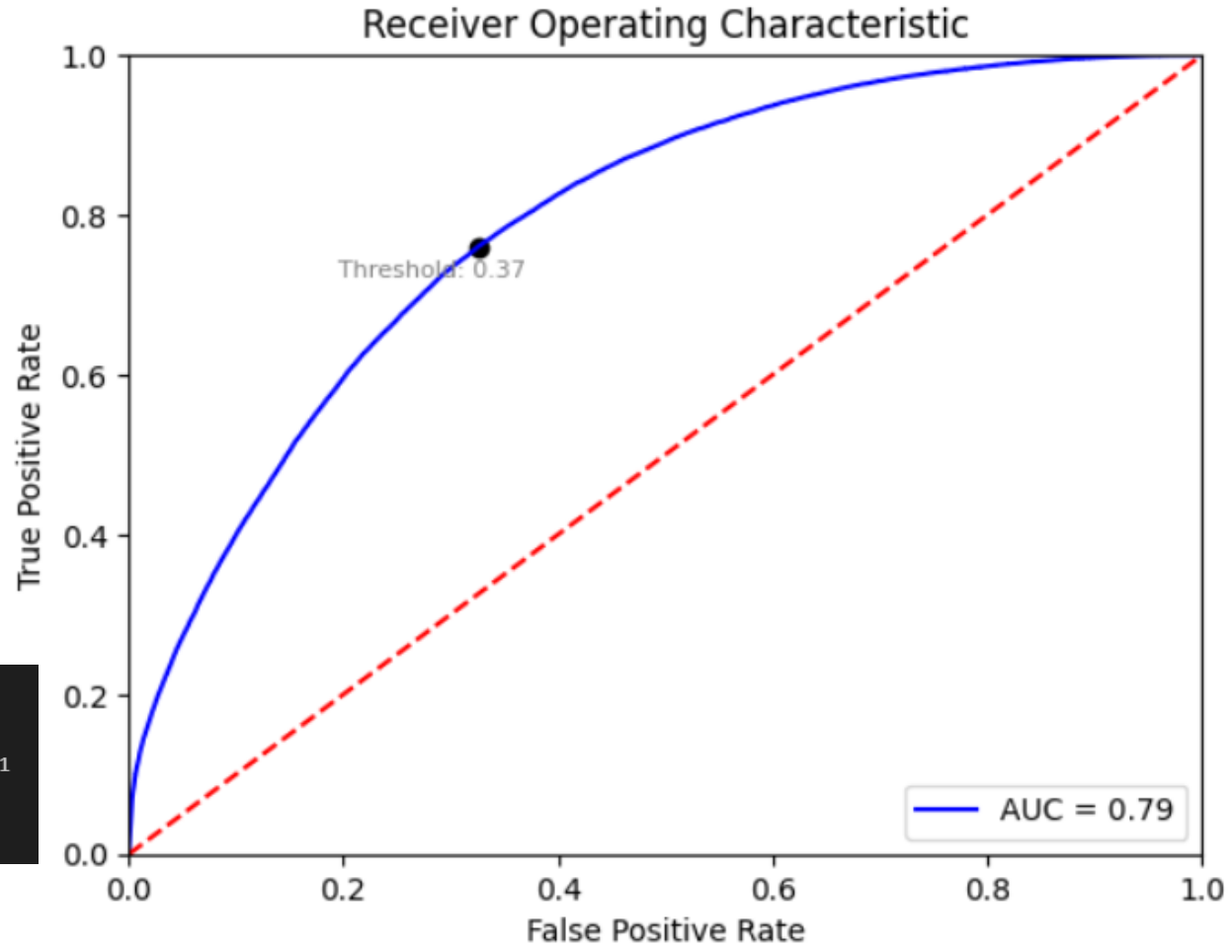


1st Model Performance

- ROC curve
- Loss function (BCELoss)

$$BCE = -\frac{1}{N}\sum_{i=0}^{N} y_i \cdot log(\hat{y}_i) + (1 - y_i) \cdot log(1 - \hat{y}_i)$$

```
Elipsed Time : 39.42240810394287
Epoch: 12/1000, Train Loss : 0.3312  Valid Loss: 0.7951,
 lowest_loss : 0.524065375328064, lowest_epoch : 1, epoch :11, ROC : 0.730501518224341

Early Stopped 12 epochs
```



2nd Model Structure

```
class ContentBased(nn.Module):
    def __init__(self, num_users, num_items, num_author, num_publisher, num_country, num_state, num_town, embedding_size=32):
        super(ContentBased, self).__init__()
        self.user_embedding = nn.Embedding(num_users, embedding_size)
        self.item_embedding = nn.Embedding(num_items, embedding_size)
        self.author_embedding = nn.Embedding(num_author, embedding_size)
        self.publisher_embedding = nn.Embedding(num_publisher, embedding_size)
        self.country_embedding = nn.Embedding(num_country, embedding_size)
        self.state_embedding = nn.Embedding(num_state, embedding_size)
        self.town_embedding = nn.Embedding(num_town, embedding_size)

        self.numeric_layer = nn.Linear(2, embedding_size)

        self.fc1 = nn.Linear(embedding_size*8, 64)
        self.fc2 = nn.Linear(64, 32)
        self.fc3 = nn.Linear(32, 16)
        self.fc4 = nn.Linear(16, 1)

        self.relu = nn.ReLU()
        self.dropout = nn.Dropout(0.4)
```

2nd Model Performance

- Loss function (RMSE)

  - 평가 산식 : RMSE (Root Mean Squared Error)

$$\mathrm{RMSE} = \sqrt{MSE} = \sqrt{\frac{\sum(\hat{y}-y)^2}{n}}$$

```
Elipsed Time : 31.95119619369507
Epoch: 1/1000, Train Loss : 682.7021  Valid Loss: 11.3446,
 lowest_loss : 11.344611167907715, lowest_epoch : 0, epoch :0

Elipsed Time : 31.118008852005005
Epoch: 2/1000, Train Loss : 680.0192  Valid Loss: 11.8944,
 lowest_loss : 11.344611167907715, lowest_epoch : 0, epoch :1

Elipsed Time : 30.985981702804565
Epoch: 3/1000, Train Loss : 678.2783  Valid Loss: 12.6365,
 lowest_loss : 11.344611167907715, lowest_epoch : 0, epoch :2

Elipsed Time : 30.832452058792114
Epoch: 4/1000, Train Loss : 676.9686  Valid Loss: 13.4710,
 lowest_loss : 11.344611167907715, lowest_epoch : 0, epoch :3

Elipsed Time : 31.123528957366943
Epoch: 5/1000, Train Loss : 675.8062  Valid Loss: 14.0258,
 lowest_loss : 11.344611167907715, lowest_epoch : 0, epoch :4

Elipsed Time : 30.384843349456787
Epoch: 6/1000, Train Loss : 674.8483  Valid Loss: 14.3509,
 lowest_loss : 11.344611167907715, lowest_epoch : 0, epoch :5

Elipsed Time : 30.787959814071655
Epoch: 7/1000, Train Loss : 674.0937  Valid Loss: 14.9374,
 lowest_loss : 11.344611167907715, lowest_epoch : 0, epoch :6

Elipsed Time : 30.901458024978638
Epoch: 8/1000, Train Loss : 673.5336  Valid Loss: 15.7789,
 lowest_loss : 11.344611167907715, lowest_epoch : 0, epoch :7
```

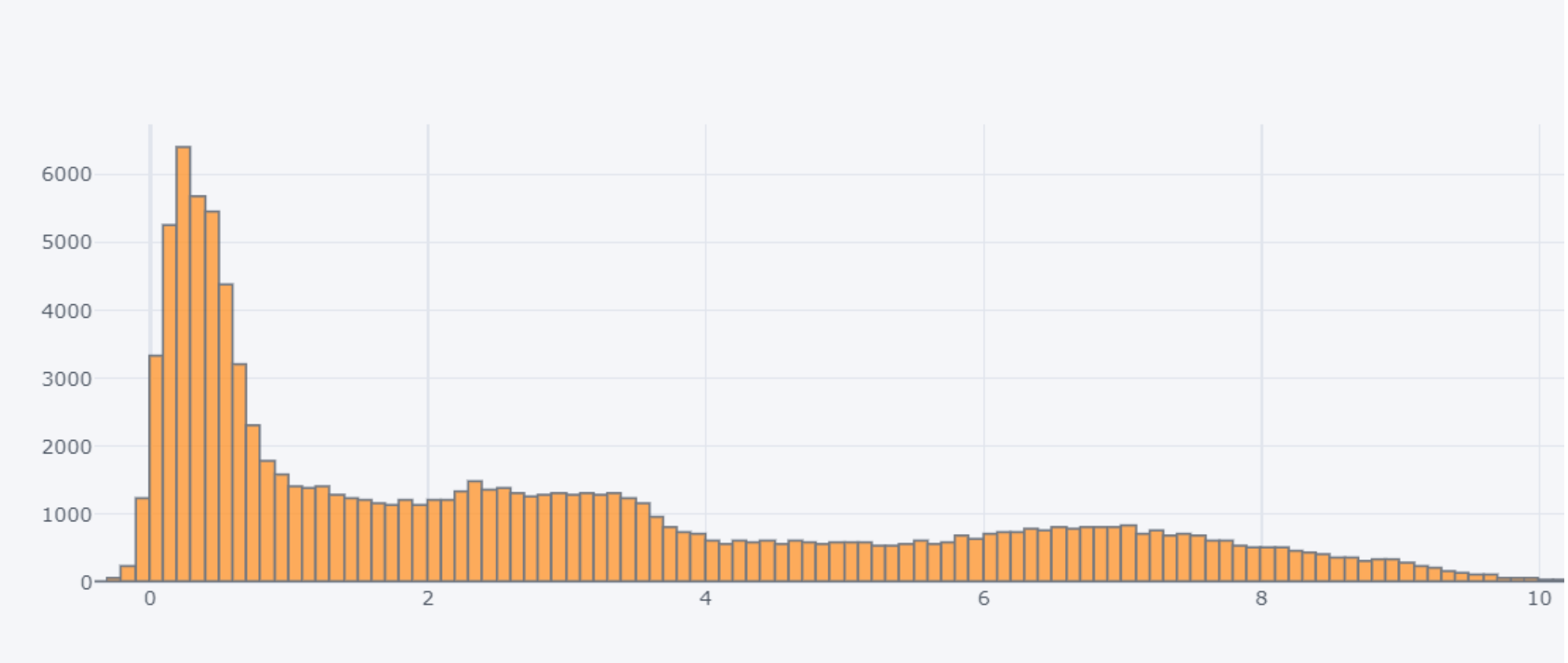
'-1' 치환

- '-1 ' 값을 치환하기 위해서 user-idx, publishment-idx, country-idx, state-idx 순으로 우열관계를 부여함.
- 부여된 우열관계를 기반으로 Group by 시킨 후, 우열관계에 따라서 그에 대한 Book-Rating의 중간값으로 대체

```
# 우열관계 바꿀려면 위에도 바꾸어야 함.
priority_dict = {'User-Idx': 0, 'Title-Idx': 1, 'Author-Idx': 2, 'Publisher-Idx':3, 'Town-Idx':4, 'State-Idx' : 5,
                 'Country-Idx': 6}
tf = a & b & c & d & e & f & g
afg = data_4_test.loc[tf,lst].values

if afg.shape[1] == 0:
    print('No data')
    pass

else:
    print(lst)
    lst = sorted(lst, key=lambda x: priority_dict[x])
    df_train_groupby = data.groupby(lst[0]).mean('Book-Rating').reset_index().copy()
    lst_score = []
    for ww in afg:
        try:
            lst_score.append(df_train_groupby.loc[ww[0] ==df_train_groupby[lst[0]].values]['Book-Rating'].values[0])
        except:
            something += 1
            lst_score.append(df_train_5['Book-Rating'].mean())
            print('Something is wrong')

    lst_score_all.append(lst_score)

    df_test_4_1.loc[tf,'Final'] = pd.DataFrame(lst_score).values.ravel()
    print(len(afg))
```

Submitted solution

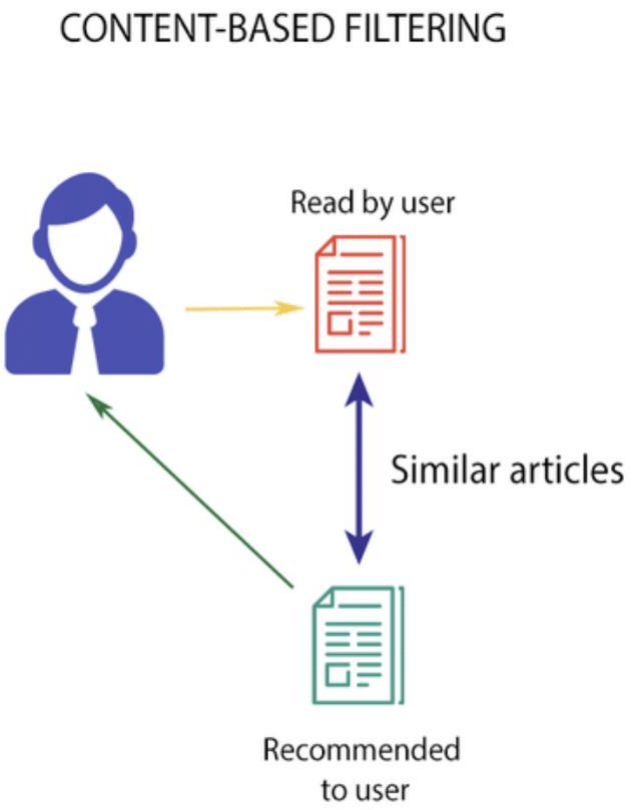
- 정답의 Book-Rating 분포(0 클래스 제외)



3. 콘텐츠 기반 모델

콘텐츠 기반 필터링

- 콘텐츠만을 활용하여 추천하는 알고리즘
- 사용자가 좋아하는 아이템과 유사한 콘텐츠를 추천하는 방식
- 다른 사용자들의 정보가 필요 X
- 아이템 설명만 있다면 유명하지 않은 아이템도 추천 가능
- 주관성이 개입될 수 있고 다양한 취향을 반영하기 어려움



전처리

```
train['Age'].value_counts().sort_index()
```

```
0.0      495
1.0      361
2.0      278
3.0      128
4.0      250
        ...
228.0     48
229.0     11
237.0      2
239.0    116
244.0      7
Name: Age, Length: 137, dtype: int64
```

[Age]

5세 미만&100세 이상 제거

```
train['Location'].value_counts()
```

```
toronto, ontario, canada      12262
n/a, n/a, n/a                 11161
```

[Location]

'n/a, n/a, n/a' 결측치 제거

```
df[df['Year-Of-Publication']==-1].count()
```

```
User-ID                11177
Book-Rating            11177
Age                    11177
Book-Title             11177
Book-Author            11177
Year-Of-Publication    11177
Publisher              11177
state                  11177
dtype: int64
```

[Year-Of-Publication]

출판 연도 -1 제거

모델링

랜덤추출로 데이터 크기 줄이기

중복되는 제목 제거

TF-IDF를 활용하여 벡터화

코사인 유사도 계산 후 유사한 책 제목 추천

모델링: TF-IDF란?

- 다른 문서에는 등장하지 않지만 특정 문서에서만
  자주 등장하는 단어를 찾아서 문서 내 단어의 가중치를 계산하는 방법

- TF(d, t) : 특정 문서 d에서의 특정 단어 t의 등장 횟수
  DF(t) : 특정 단어 t가 등장한 문서의 수
  IDF(d, t) : DF(t)에 반비례하는 수
  TF-IDF(d, t) = TF(d, t)*IDF(d, t)

- 즉 해당 문서 내 특정 단어 빈도가 높을수록,
  전체 문서 중 특정 단어를 포함한 문서가 적을수록 값이 높

Book-Title
Where Angels Walk: True Stories of Heavenly Vi...
Cry to Heaven
Call of the Wild (A Watermill Classic)
Day of Confession
Ruby Ann's Down Home Trailer Park Bbqin' Cookbook

```
# 중복되는 제목 제거하기
df = df.drop_duplicates(subset= 'Book-Title')
df.reset_index(drop=True, inplace=True)
# 도서 제목에 대한 TF-IDF 벡터화
vectorizer = TfidfVectorizer(stop_words='english')
item_tfidf = vectorizer.fit_transform(df['Book-Title'])
```

모델링

```
user_input = input("도서명을 입력하세요: ")
```

```
도서명을 입력하세요: Without a Doubt
```

```
recommend_books(user_input)
```

	Book-Title	Book-Author	Publisher	Year-Of-Publication
	Shadow of a Doubt	William Coughlin	St. Martin's Press	1995.0
	The Salmon of Doubt	DOUGLAS ADAMS	Ballantine Books	2003.0
	Reasonable Doubt	Philip Friedman	Ivy Books	1997.0
	The Salmon of Doubt: Hitchhiking the Galaxy One Last Time	DOUGLAS ADAMS	Harmony	2002.0
	Legacy of the Sword (Chronicles of the Cheysuli, Book 3)	Jennifer Roberson	Daw Books	1992.0
	Mixed Blessings	Danielle Steel	Delacorte Press	1992.0
	Flabbergasted	Ray Blackston	Fleming H. Revell Company	2003.0
	Buddha of Suburbia	Hanif Kureishi	Faber Paperbacks	2000.0
	Murder in Havana (Truman, Margaret, Capital Crimes Series.)	Margaret Truman	Fawcett Books	2002.0

한계점

- 해당 데이터는 책 리뷰, 장르 등 책에 대한 추가적인 정보 없이 책 제목만 있어 콘텐츠 기반 필터링만으로 추천시스템을 구현하기에는 어려움이 있음.

- 따라서, 다른 사용자와 같은 추가 정보를 활용하는 협업 필터링 모델, 하이브리드 추천시스템을 적용하는 것이 효과적임.

하이브리드 모델

- 이와 같은 한계점을 개선하기 위해
  협업 필터링과 컨텐츠 베이스 필터링을 결합한 간단한
  하이브리드 모델 생성

→ 유사한 사용자가 추천하는 책들 중 (협업 필터링)
가장 유사한 제목을 가진 책들 선별 (컨텐츠 베이스)

모델링

User-Item 간 행렬 생성
& 사용자 간의 코사인 유사도 계산

입력된 책을 높게 평가한 사용자를 찾아서
그 사용자와 가장 유사한 사용자들을 찾는 함수

유사한 사용자들이 높게 평가한 책을 찾는 함수

사용자로부터 책 제목을 입력 받고,
해당 책에 대한 추천을 출력하는 코드

하이브리드 모델 생성

```
def recommend_books2(book_title, user_sim_df, n_books=10):
    similar_users = find_similar_users(book_title, user_sim_df)
    recommended_books = df[df['User-ID'].isin(similar_users) & (df['Book-Title'] != book_title)]
    recommended_books = recommended_books.sort_values(by='Book-Rating', ascending=False)['Book-Title'].unique()[:n_books]
    return recommended_books
```

```
book_title = input("Enter a book title: ")
try:
    recommendations = recommend_books2(book_title, user_sim_df)
    print("Recommended books for you:")
    for book in recommendations:
        print(book)
except:
    print("Sorry, the book title you entered is not in our database. Please try again.")
```

```
Enter a book title: Daniel's Gift
Recommended books for you:
Miracle Girls #1
Her Father's House
Too Many to Mourn
The Last Command (Star Wars: The Thrawn Trilogy, Vol. 3)
The Virgin Suicides
Once upon a Rose
Chicken Soup for the Mother &amp;amp; Daughter Soul: Stories to Warm the Heart and Honor the Relationship
What the IRS Doesn't Want You to Know: A Cpa Reveals the Tricks of the Trade (What the IRS Doesn't Want You to Know)
This Year It Will Be Different: And Other Stories : A Christmas Treasury
Out on a Limb
```

유사한 사용자가 추천하는 책들을 추천해주는

recommend_books2 함수 생성

하이브리드 모델 생성

```python
def hybrid_recommendation(book_title, user_sim_df, n_books=10):

    recommended_books = recommend_books2(book_title, user_sim_df, n_books)
    similar_books = recommend_books(book_title)

    hybrid_recommendations = []
    for book in similar_books:
        if book in recommended_books:
            hybrid_recommendations.append(book)

    return hybrid_recommendations[:n_books]
```

유사한 제목을 가진 책을 추천해주는

recommend_books 함수

+

유사한 사용자가 추천해주는 책을 추천하는 recommend_books2

➡ hyprid_recommendation 함수

```python
book_title = input("Enter a book title: ")
try:
    recommendations = hybrid_recommendation(book_title, user_sim_df)
    print("Recommended books for you:")
    for book in recommendations:
        print(book)
except:
    print("Sorry, the book title you entered is not in our database. Please try again.")
```

```
Enter a book title: Daniel's Gift
                                        Book-Title        Book-Author  #
0                                  Daniel'S Bride  Linda Lael Miller
1                               The Greatest Gift        Danny Leigh
2                                   Daniel Martin        John Fowles
3                                  SERPENT'S GIFT   Helen Elaine Lee
4                          Gabriel's Gift : A Novel     Hanif Kureishi
5                    The Devil and Daniel Silverman    Theodore Roszak
6                        The Macgregors: Daniel-Ian       Nora Roberts
7                        The Macgregors: Daniel-Ian       Nora Roberts
8                    The Perfect Gift (Avon Romance)     Christina Skye
9   God Cares: Vol 1 The Message of Daniel for You...  C. Mervyn Maxwell
10          Secrets of Happiness (Secrets Gift Books)  J. Donald Walters

    Year-Of-Publication                       Publisher
0                1992.0                          Pocket
1                2004.0              Faber and Faber Ltd
2                1999.0                             LGF
3                1995.0                         Scribner
4                2001.0                         Scribner
5                2002.0                      Consortium
6                1999.0                      Silhouette
7                1999.0                      Silhouette
8                1999.0                            Avon
9                1981.0  Pacific Press Pub. Association
10               2003.0      Crystal Clarity Publishers
```

4. 협업 필터링 기반 모델

협업필터링이란?

- 사용자의 구매 이력 및 평점을 통한 행동 양식을 기반으로 추천

    1. User-Based: 특정 사용자의 구매패턴, 평점을 활용해 유사한 사용자를 찾아 추천 리스트 생성

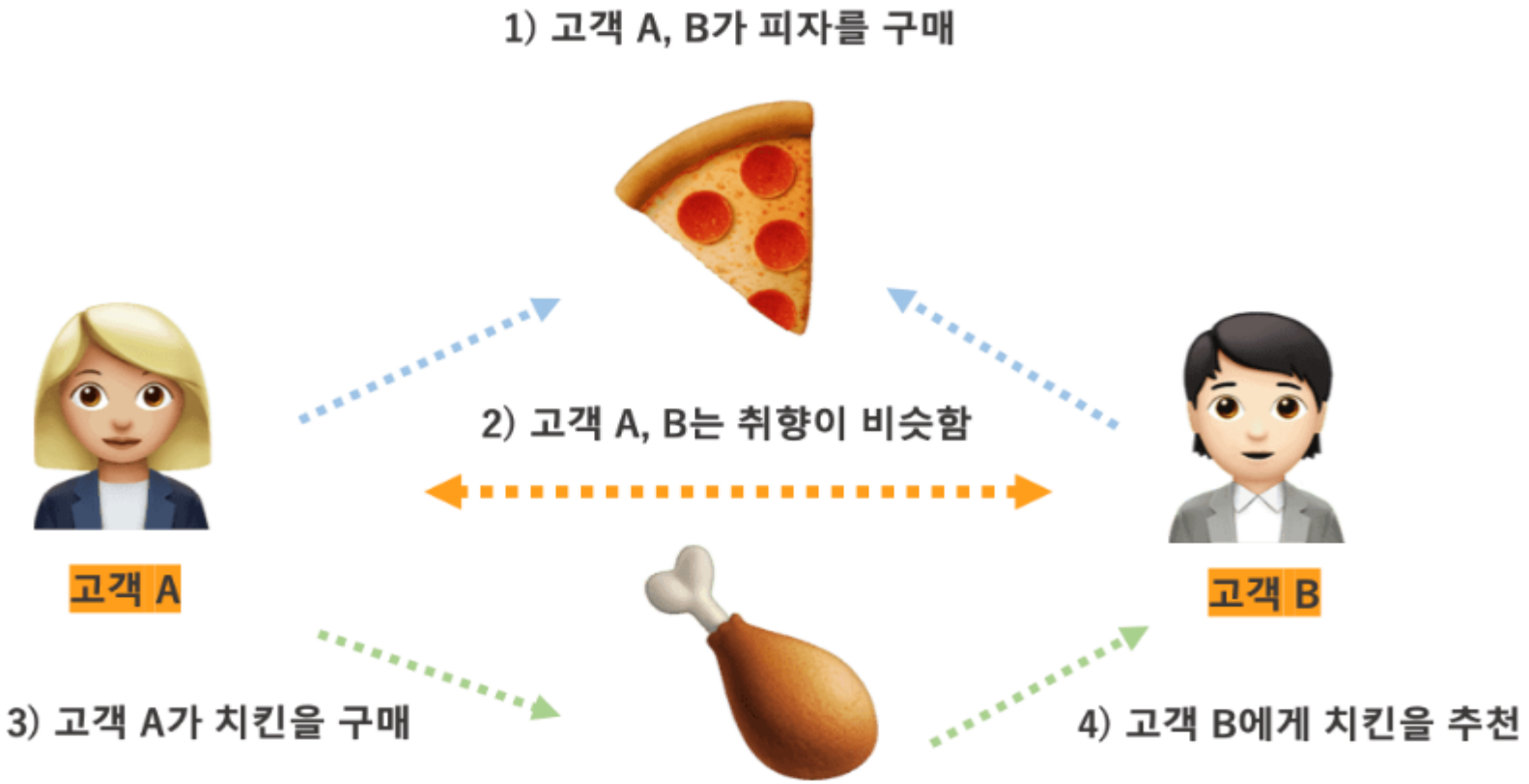
    2. Item-Based: 사용자들이 아이템에 대해 부여한 평점 간의 유사도를 계산해 추천 리스트 생성

- 대표적인 유사도 계산 Metric: **코사인 유사도**

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}}$$

  - Step1. 나와 **비슷한 성향**을 가진 사람 찾기
  - Step2. 그 사람의 **또 다른 소비를 파악**하여 나에게 추천

협업필터링이란?


1) 고객 A, B가 피자를 구매
2) 고객 A, B는 취향이 비슷함
고객 A
고객 B
3) 고객 A가 치킨을 구매
4) 고객 B에게 치킨을 추천

전처리

1. 거주 지역 단순화

: 상위 60% state 추출한 후 나머지 주는 Others로 분류 → 20여개의 state로 범주화

2. 연령 이상치 처리 및 범주화

: 120세 이상 Outliers 제거

: 10세 미만/90세 이상은 각각 두 그룹으로 분류 & 10세~ 90세 사이의 유저들은 5세 단위로 범주화

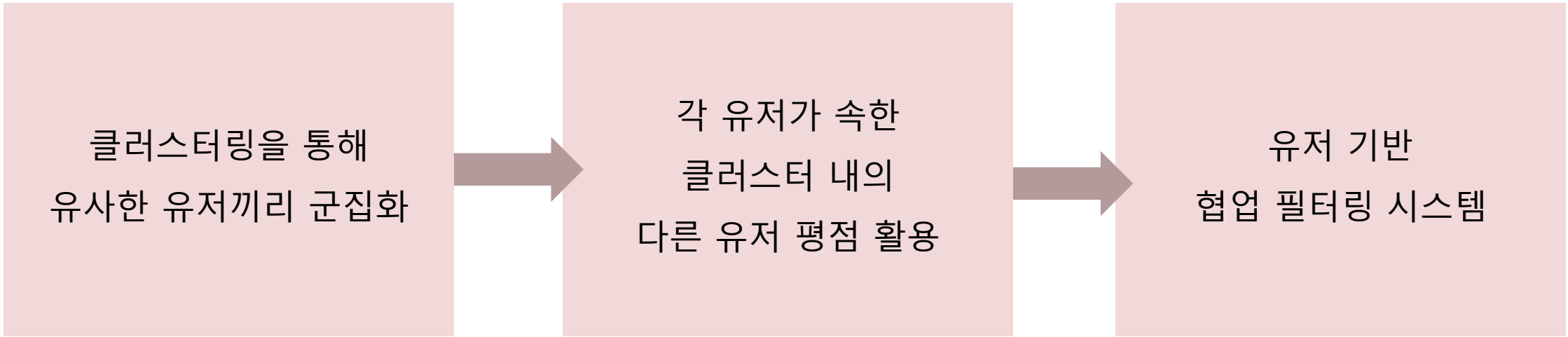
3. 출판연도 이상치 처리 및 범주화

: -1의 값을 갖는 Outlier 제거

: 1980년 이전/2000년 이후는 각각 두 그룹으로 분류 & 1980~2000년은 5년 단위로 범주화

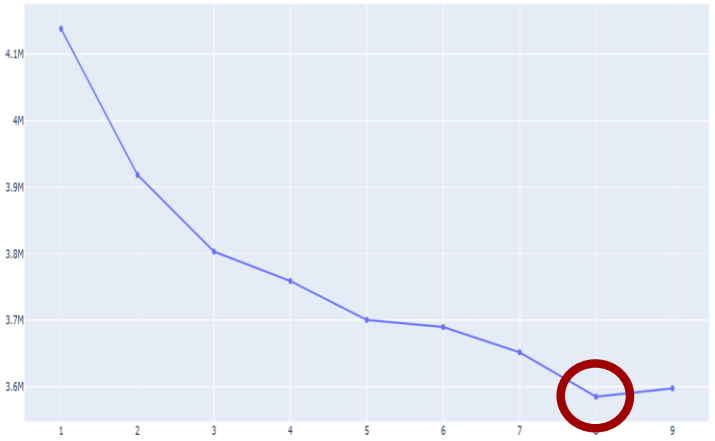
클러스터링

- STEPS


클러스터링을 통해
유사한 유저끼리 군집화
각 유저가 속한
클러스터 내의
다른 유저 평점 활용
유저 기반
협업 필터링 시스템

클러스터링

- K-Modes 클러스터링　범주형 변수 군집화 기법 / 클러스터 중심치를 mode로 계산



K=8로 지정

User-ID

Book-Rating

Age

State

Book-Title

Book-Author

Year-Of-Publication

Publisher

최종 변수 조합

```
df['clusters'].value_counts()

0    216816
1    162129
3     77455
2     76405
6     45281
4     42367
5     38513
7     29458
```

클러스터링 결과

클러스터링

- 한 유저가 여러 개의 클러스터에 속하는 문제 발생

  ➔ 각 유저 당 한 개의 클러스터 지정(최빈값)

- 각 클러스터별 특징 확인(도서 취향)

각 클러스터별 **평균 평점이 가장 높은 도서** 5권 확인

Cluster 0

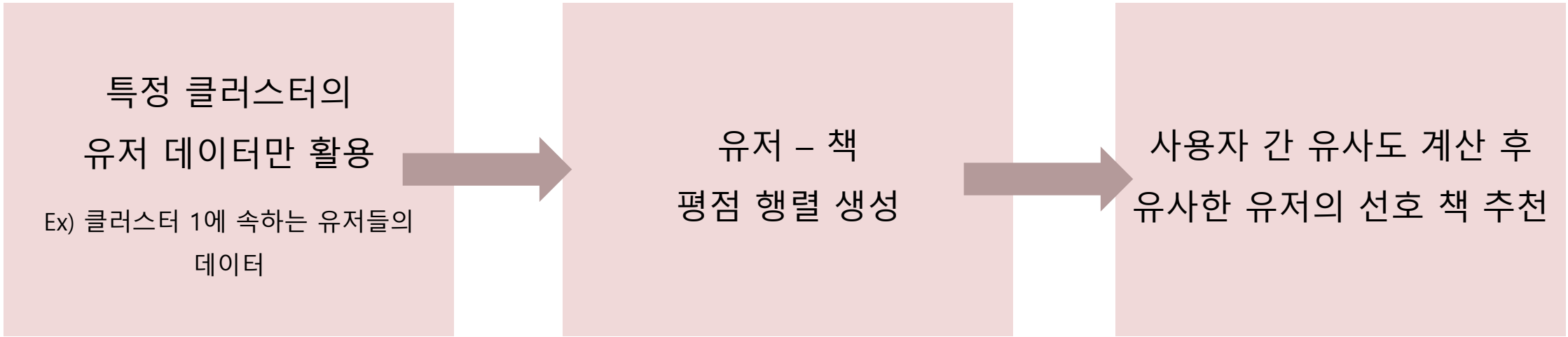
```
cluster0.groupby('Book-Title')['Book-Rating'].mean().nlargest(5)
```

```
Book-Title
"Follow Me!"                                                                  10.0
"I Can't" Said the Ant                                                        10.0
"Soleil De Soufre" Et Autres Nouvelles                                        10.0
"The House of Cthulhu" and Other Tales                                        10.0
'Isms: a dictionary of words ending in -ism, -ology, and -phobia,: With some similar terms, arranged in subject order  10.0
```

Cluster 0	Cluster 1	Cluster 2	Cluster 3	Cluster 4	Cluster 5	Cluster 6	Cluster 7
SF, 공포물	에세이	전기, 기행문	전략 가이드	자연	로맨스	모험	소설

협업 필터링

- STEPS


특정 클러스터의
유저 데이터만 활용
Ex) 클러스터 1에 속하는 유저들의 데이터
유저 – 책
평점 행렬 생성
사용자 간 유사도 계산 후
유사한 유저의 선호 책 추천

협업 필터링

- 유저 – 책 평점 행렬 생성

```
rating_matrix = df1.pivot_table(index='User-ID', columns='Book-Title', values='Book-Rating')
```

Book-Title	Deceived	Earth Prayers From around the World: 365 Prayers, Poems, and Invocations for Honoring the Earth	Final Fantasy Anthology: Official Strategy Guide (Brady Games)	Flight of Fancy: American Heiresses (Zebra Ballad Romance)	Garfield Bigger and Better (Garfield (Numbered Paperback))	God's Little Promise Book	Goosebumps Monster Edition 1: Welcome to Dead House, Stay Out of the Basement, and Say Cheese and Die!	LA Gallinita Roja/the Little Red Hen	Mystery Mile	Q-Space (Star Trek The Next Generation, Book 47)	...	seaQuest 2	sed &amp: awk (A Nutshell handbook)	stardust	teach yourself...C++	them (Modern Library)	wet sand, raven tracks
user_id																	
USER_00003	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0
USER_00077	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0
USER_00136	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0
USER_00207	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0
USER_00209	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
USER_91794	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0
USER_91895	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0
USER_91905	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0
USER_92054	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0
USER_92078	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0

협업 필터링

- 유사도 계산

```
from sklearn.metrics.pairwise import cosine_similarity

def create_similarity_matrix(rating_matrix):
    ratings = rating_matrix.values

    similarity_matrix = cosine_similarity(ratings)

    similarity_matrix = pd.DataFrame(similarity_matrix, index=rating_matrix.index, columns=rating_matrix.index)

    return similarity_matrix
```

협업 필터링

- 유사한 유저의 선호 책 목록에서 추천

①
```
def get_recommendations(user_id, rating_matrix, similarity_matrix, top_n=5):
    user_ratings = rating_matrix.loc[user_id]
    user_similarity = similarity_matrix.loc[user_id]
```

②
```
    # 유사도가 가장 높은 사용자 찾기
    similar_users = user_similarity.sort_values(ascending=False).index[1:]

    recommended_books = []
    target = []
    for user in similar_users:
        other_user_ratings = rating_matrix.loc[user]
        unrated_books = other_user_ratings[other_user_ratings == 0].index
        rated_books = other_user_ratings[other_user_ratings > 0].index
```

③
```
    # 그 사용자가 높은 평점을 준 책을 목록에 추가
    recommended_books.append(rated_books)

    point=df1[df1['User-ID']==user_id]['Book-Title']
    point=point.tolist()
    recommended_books=recommended_books[0]
    recommended_books=recommended_books.tolist()
    for i in range(len(point)):
      if point[i] in recommended_books:
        recommended_books.remove(point[i])
    target.append(recommended_books)
```

④
```
    # 지정 추천 개수까지 반복
    if len(recommended_books) >= top_n:
        break

# 상위 n개 선택
top_books = recommended_books[:top_n]

return top_books
```

협업 필터링

- 추천 결과

```
user_id = 'USER_00059'
recommendations = get_recommendations(user_id, rating_matrix, similarity_matrix, top_n=5)
print(recommendations)
```

Aurora Quest (Earthblood #3) (Earthblood, No 3)' SF

'Berserker' SF

'Death of Sleep' SF

'Earthblood (Gold Eagle Super Bolan)' SF

'Excavation' SF, 스릴러

클러스터 3에 속한
00059번 유저에게 책 추천

미국에 거주하는 40대 초반 유저

- 영어 서적 추천
- SF 장르 책 위주로 추천

Reference

*** Understanding PyTorch Loss Functions: The Maths and Algorithms**

(https://towardsdatascience.com/understanding-pytorch-loss-functions-the-maths-and-algorithms-part-2-104f19346425)

*** 자습해도 모르겠던 딥러닝, 머리속에 인스톨 시켜드립니다.**

https://www.slideshare.net/yongho/ss-79607172

*** 제2회 코스포 x 데이콘 도서 추천 알고리즘 AI경진대회**

https://dacon.io/competitions/official/236093/overview/description

Thank you !

Any Question?