📚 [LAB10] 지도학습 > 추천시스템 > BaselineOnly

📘 #01. 추천 시스템이란?

사용자가 좋아할 가능성이 높은 아이템을 예측하는 모델

- 넷플릭스 영화 추천
- 유튜브 영상 추천
- 쿠팡 상품 추천

📝 [1] 추천시스템의 핵심 데이터

추천 문제는 결국 **"사용자가 아직 보지 않은 아이템의 평점을 예측하는 문제"**임.

구성 요소	의미
사용자 (User)	서비스를 이용하는 사람
아이템 (Item)	영화, 상품, 음악 등
평점 (Rating)	사용자의 선호도 점수

📝 [2] 추천시스템의 유형

✏️ Recommender System

유저에 대한 인구통계학(demographic)적인 정보를 이용

나이/성별/인종 등을 고려하여 인구통계학적 집단(demographic stereotype/cluster)을 정의해 둔 후, 타깃 유저를 이 중 한 집단으로 분류

비슷한 인구통계학적 특성을 보이는 사람들은 취향도 비슷할 것이라 가정하는 것

> 오늘날에는 잘 사용하지 않음

✏️ Content Filtering (콘텐츠 기반 필터링 모델)

유저가 아닌 아이템의 내용(contents)에 집중하여 사용자가 좋아하는 콘텐츠를 분석하여 그와 유사한 콘텐츠를 추천해 주는 기술다.

영화의 경우 영화의 장르, 러닝타임, 감독, 주연 배우 등 영화에 대한 기본 정보를 활용함.

예를 들어 사용자가 “All I want for Christmas is you”라는 노래를 감상했다면, 이를 바탕으로 겨울과 관련된 노래를 추천해줄 수 있다.

✏️ Collaborative Filtering (협업 필터링 모델)

다른 사용자들로부터 취향 정보들을 모아 사용자의 관심사를 예측하는 방법.

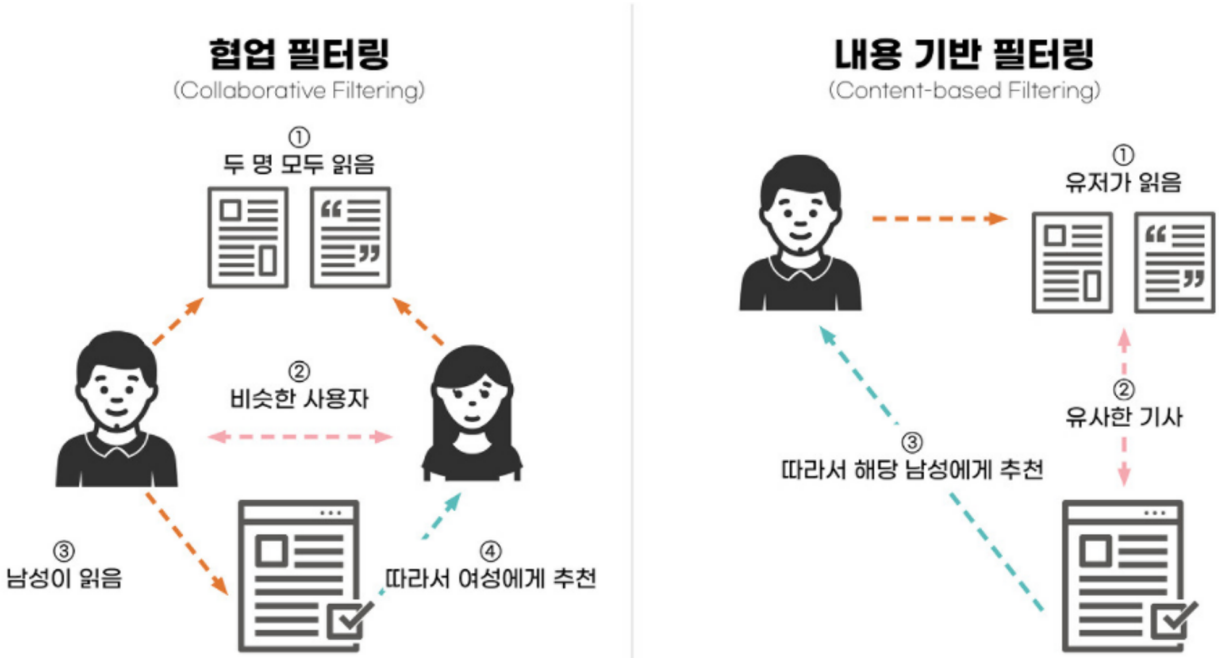
타깃 유저에 대한 데이터뿐만 아니라 다른 유저들에 대한 데이터도 적극적으로 활용한다.

타깃 유저의 정보와 타 유저들의 정보가 협동(collaborate)하여 숨어있는 데이터를 찾아주는 방식

비슷한 취향을 가진 사용자들은 어떠한 아이템에 대해 비슷한 선호도를 가질 것이라는 가정하에 사용자와 아이템 간 상호작용 데이터를 활용한다.

만약 캐롤을 좋아하는 사람들이 공통적으로 판타지 영화에 대해 높은 선호도를 보인다면 사용자에게 “해리포터”를 추천해줄 수도 있을 것이다.

> 발전순으로 나열한 것이므로 지금은 Collaborative Filtering이 국룰~!!



📝 [2] Baseline 알고리즘

사용자가 컨텐츠에 부여한 평점을 기반으로 하는 가장 기본적 협업필터링 알고리즘.

추천에서 가장 기본이 되는 개념은 “편향(Bias)”임.

사람마다 평가 기준이 다름:

- 어떤 사람은 항상 후하게 줌
- 어떤 사람은 항상 짜게 줌
- 어떤 영화는 대체로 점수가 높음

이런 차이를 먼저 보정하는 것이 Baseline 모델임.

✏️ 📌 Baseline 평점 공식

$$\hat{r}_{ui} = \mu + b_u + b_i$$

기호	의미
μ	전체 평균 평점
b_u	사용자 편향
b_i	아이템 편향

✏️ 📌 해석

예시:

- 전체 평균 = 3.5
- 사용자 A 평균 = 3.0 (짜게 줌)
- 영화 평균 = 4.2 (인기 많음)

계산:

- 사용자 편향: 3.0 − 3.5 = −0.5
- 아이템 편향: 4.2 − 3.5 = 0.7

예측 평점:

$$3.5 - 0.5 + 0.7 = 3.7$$

즉, 개인 성향 + 영화 인기 정도를 반영한 점수임

✏️ 🎯 Baseline의 의미

- 추천의 출발점
- 복잡한 모델의 기본 구조
- 편향 보정의 개념 이해가 핵심

📘 #02. 준비작업

📝 [1] Surprise 패키지 설치

Surprise는 추천시스템 전용 파이썬 패키지임.

특징:

- 사용자–아이템 평점 데이터만 사용
- 추천 알고리즘 구현이 매우 간단
- sklearn과 유사한 구조

설치:

```
pip install --upgrade scikit-surprise
```

설치 에러시 아래 명령을 수행한 후 다시 시도

```
pip install --upgrade pip setuptools wheel
```

```
from hossam import *
from pandas import DataFrame, merge

from surprise import Dataset, Reader, BaselineOnly, accuracy
from surprise.model_selection import train_test_split, GridSearchCV
```

```
📦 아이티윌 이광호 강사가 제작한 라이브러리를 사용중입니다.
📚 자세한 사용 방법은 https://py.hossam.kr 을 참고하세요.
📧 Email: leekh4232@gmail.com
🎬 Youtube: https://www.youtube.com/@hossam-codingclub
📝 Blog: https://blog.hossam.kr/
🩹 Version: 0.5.4
현재 설치된 'hossam' 패키지 버전: 0.5.4

✅ 시각화를 위한 한글 글꼴(NotoSansKR-Regular)이 자동 적용되었습니다.
```

📝 [2] 데이터셋 가져오기

✏️ 분석대상 - 평점데이터

```
origin = load_data("ml100k-ratings")
origin.head()
```

```
943명의 사용자가 1,682편의 영화에 대해 남긴 100,000개의 평점 기록으로 구성된 명시적 평가 기반 추천 시스템 학습용 데이터셋
(출처: University of Minnesota)

컬럼명   의미
---------  ---------
user_id    사용자 ID
item_id    아이템 ID
```

```
rating      평점
timestamp   평가 시각
```

	user_id	item_id	rating	timestamp
0	196	242	3	881250949
1	186	302	3	891717742
2	22	377	1	878887116
3	244	51	2	880606923
4	166	346	1	886397596

✏️ 분석결과 맵핑 데이터 - 영화 정보

```
metadata = load_data("ml100k-metadata")
metadata.head()
```

```
ml100k-ratings에 포함된 영화 제목, 공개시기, 장르 정보를 담고 있는 데이터 (출처: University of Minnesota)
```

	item_id	title	release_date	IMDb_URL	unknown	Action	Adventure	Animation	Children's
0	1	Toy Story (1995)	01-Jan-1995	http://us.imdb.com/M/title-exact?Toy%20Story%20(1995)	0	0	0	1	1
1	2	GoldenEye (1995)	01-Jan-1995	http://us.imdb.com/M/title-exact?GoldenEye%20(1995)	0	1	1	0	0
2	3	Four Rooms (1995)	01-Jan-1995	http://us.imdb.com/M/title-exact?Four%20Rooms%20(1995)	0	0	0	0	0
3	4	Get Shorty (1995)	01-Jan-1995	http://us.imdb.com/M/title-exact?Get%20Shorty%20(1995)	0	1	0	0	0
4	5	Copycat (1995)	01-Jan-1995	http://us.imdb.com/M/title-exact?Copycat%20(1995)	0	0	0	0	0

📘 #03. BaselineOnly 모델 적합

📝 [1] DataFrame을 Dataset 객체로 변환

scikit-surprise는 일반 DataFrame을 직접 학습하지 않고, 반드시 Reader를 통해 변환된 Dataset 객체만 학습 가능하다.

✏️ 평점의 범위 확인

```
rating_min = origin["rating"].min()
rating_max = origin["rating"].max()
print(f"Rating 범위: {rating_min} ~ {rating_max}")
```

```
Rating 범위: 1 ~ 5
```

✏️ surprise 라이브러리에서 사용할 수 있도록 데이터셋을 변환

```
# 평점의 범위를 지정하여 Reader 객체 생성
reader = Reader(rating_scale=(rating_min, rating_max))

# Dataset 객체를 생성 - 사용자 식별자, 아이템 식별자, 평점만으로 구성된 데이터 구조가 필요하다.
data = Dataset.load_from_df(origin[["user_id", "item_id", "rating"]], reader)

data
```

```
<surprise.dataset.DatasetAutoFolds at 0x3129be390>
```

📝 [2] BaselineOnly 모델의 주요 하이퍼 파라미터

하이퍼 파라미터 탐색은 원본 데이터에 대해서 수행한다.

파라미터명	핵심도	설명	기본값	GridSearchCV 권장값
bsl_options['method']	⭐⭐⭐	사용자·아이템 bias 추정 방법 (ALS 또는 SGD)	'als'	['als', 'sgd']
bsl_options['reg']	⭐⭐⭐	정규화 강도 (bias 과적합 억제)	10	[1, 5, 10, 15, 20]
bsl_options['learning_rate']	⭐⭐	SGD 사용 시 학습률 (method='sgd' 일 때만 적용)	0.005	[0.002, 0.005, 0.01, 0.02]
bsl_options['n_epochs']	⭐⭐	SGD 반복 횟수	20	[10, 20, 30, 50]
random_state	✅	SGD 초기화 시드 (재현성 확보용)	None	52

```
param_grid = {
    "bsl_options": {
        "method": ["als", "sgd"],
        "reg": [1, 5, 10, 15, 20],
        "learning_rate": [0.002, 0.005, 0.01],   # sgd일 때만 적용
        "n_epochs": [10, 20, 30]
```

```
    }
}

# GridSearchCV 객체를 생성하여 하이퍼파라미터 튜닝을 수행
gs = GridSearchCV(
    BaselineOnly,
    param_grid,
    measures=["rmse", "mae"],
    cv=5,
    n_jobs=-1
)

# GridSearchCV를 사용하여 최적의 하이퍼파라미터 조합을 찾는다.
# -> 원본 데이터 사용
gs.fit(data)

# 최적의 RMSE와 MAE 점수 및 해당 하이퍼파라미터 조합을 출력
print("Best RMSE:", gs.best_score["rmse"])
print("Best Params (RMSE):", gs.best_params["rmse"])

print("Best MAE:", gs.best_score["mae"])
print("Best Params (MAE):", gs.best_params["mae"])
```

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

```
Estimating biases using sgd...
Estimating biases using sgd...
Estimating biases using sgd...
Best RMSE: 0.9434998219063374
Best Params (RMSE): {'bsl_options': {'method': 'als', 'reg': 1, 'learning_rate': 0.002,
'n_epochs': 30, 'verbose': False}}
Best MAE: 0.7479280892973562
Best Params (MAE): {'bsl_options': {'method': 'als', 'reg': 1, 'learning_rate': 0.002,
'n_epochs': 30, 'verbose': False}}
```

📘 #04. 성능평가

surprise 패키지의 모델은 최적 파라미터를 찾은 후 전체 데이터를 사용하여 최적 모델로 다시 학습해야 한다.

sklearn의 GridSearchCV를 통해 얻는 best_estimator는 교차검증 후 train_set을 활용한 재학습까지 수행하지만, surprise 패키지는 재학습을 수행하지 않기 때문에 따로 진행해야 한다.

이 과정에서 성능 평가를 위해 훈련/검증 데이터로 나누는 과정이 필요하다.

📝 [1] 훈련, 검증 데이터 분리

```python
# 데이터를 학습용과 테스트용으로 분할 (80% 학습, 20% 테스트)
train_data, test_data = train_test_split(data, test_size=0.2, random_state=52)

# 학습용과 테스트용 데이터의 크기를 출력
print(f"Trainset 크기: {train_data.n_ratings}개")
print(f"Testset 크기: {len(test_data)}개")
```

```
Trainset 크기: 80000개
Testset 크기: 20000개
```

📝 [2] 최적 모델 재학습

```python
# 최적 파라미터 추출
best_params = gs.best_params["rmse"]

# 모델 생성
best_model = BaselineOnly(**best_params)

# 전체 데이터 학습
best_model.fit(train_data)
```

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

```
<surprise.prediction_algorithms.baseline_only.BaselineOnly at 0x31e903e10>
```

📝 [3] 예측값 생성

```
predictions = best_model.test(test_data)
predictions[:5]  # 예측 결과의 일부를 출력
```

```
[Prediction(uid=303, iid=679, r_ui=2.0, est=3.1393275100473677, details={'was_impossible':
False}),
 Prediction(uid=308, iid=163, r_ui=4.0, est=3.6883481119038146, details={'was_impossible':
False}),
 Prediction(uid=327, iid=663, r_ui=4.0, est=3.5897839643819855, details={'was_impossible':
False}),
 Prediction(uid=912, iid=479, r_ui=4.0, est=4.106633675573525, details={'was_impossible':
False}),
 Prediction(uid=224, iid=329, r_ui=3.0, est=2.8119332194991884, details={'was_impossible':
False})]
```

📝 [4] 성능평가 지표 생성

- BaselineOnly는 구조상 큰 Gap이 발생하기 어렵기 때문에 과적합이 잘 발생하지 않는다.
- 하지만 모델 복잡도가 낮아 위험은 제한적
- 핵심 통제 변수는 reg 하나뿐

✏️ 과적합이 발생할 수 있는 경우

상황	설명
사용자 수 매우 많음	희소성 증가
특정 사용자 평점 수 매우 적음	bias 추정 불안정
`reg` 값이 너무 작음	bias가 과도하게 커짐

```
# Train 예측 (trainset 전체를 test 형식으로 변환)
train_predictions = best_model.test(train_data.build_testset())

# Test 예측
test_predictions = best_model.test(test_data)

# 성능 계산
train_rmse = accuracy.rmse(train_predictions, verbose=False)
train_mae  = accuracy.mae(train_predictions, verbose=False)

test_rmse = accuracy.rmse(test_predictions, verbose=False)
test_mae  = accuracy.mae(test_predictions, verbose=False)
```

```
# 일반화 오차 차이
rmse_gap = test_rmse - train_rmse
mae_gap  = test_mae  - train_mae

# 과적합 판정 기준 (RMSE 기준)
# 기준: test RMSE가 train RMSE보다 0.05 이상 크면 과적합 의심
if rmse_gap > 0.05:
    overfit_flag = "과적합 의심"
else:
    overfit_flag = "정상"

# 6️⃣ 성능평가표 생성
result_df = DataFrame({
    "Model": ["BaselineOnly"],
    "Train_RMSE": [train_rmse],
    "Test_RMSE": [test_rmse],
    "RMSE_Gap": [rmse_gap],
    "Train_MAE": [train_mae],
    "Test_MAE": [test_mae],
    "MAE_Gap": [mae_gap],
    "Overfitting": [overfit_flag]
})

result_df
```

	Model	Train_RMSE	Test_RMSE	RMSE_Gap	Train_MAE	Test_MAE	MAE_Gap	Overfitting
0	BaselineOnly	0.922	0.947	0.026	0.731	0.751	0.020	정상

📘 #05. TopN 추천

📝 [1] 아직 평가하지 않은 아이템에 대한 예측 수행

항목	타입	의미	비고
`uid`	str	사용자 ID (raw id)	내부 인덱스 아님
`iid`	str	아이템 ID (raw id)	내부 인덱스 아님
`true_rating`	float	실제 평점	anti-testset에서는 `None`
`est`	float	모델이 예측한 평점	추천 랭킹 기준
`details`	dict	예측 관련 부가 정보	알고리즘별로 다름

✏️ 예측결과 생성

```
anti_testset = train_data.build_anti_testset()
predictions = best_model.test(anti_testset)
predictions[:5]  # 예측 결과의 일부를 출력
```

```
[Prediction(uid=234, iid=205, r_ui=3.5317375, est=3.4962269245104007,
details={'was_impossible': False}),
 Prediction(uid=234, iid=504, r_ui=3.5317375, est=3.311088596094799,
details={'was_impossible': False}),
 Prediction(uid=234, iid=73, r_ui=3.5317375, est=2.9809970216823714,
details={'was_impossible': False}),
 Prediction(uid=234, iid=475, r_ui=3.5317375, est=3.3435994193805505,
details={'was_impossible': False}),
 Prediction(uid=234, iid=294, r_ui=3.5317375, est=2.6108510945188326,
details={'was_impossible': False})]
```

✏️ 예측결과 데이터프레임 구성

```
pred_df = DataFrame(predictions,
                    columns=["user_id", "item_id", "true_rating", "pred_rating",
        "details"])

pred_df.head()
```

	user_id	item_id	true_rating	pred_rating	details
0	234	205	3.532	3.496	{'was_impossible': False}
1	234	504	3.532	3.311	{'was_impossible': False}
2	234	73	3.532	2.981	{'was_impossible': False}
3	234	475	3.532	3.344	{'was_impossible': False}
4	234	294	3.532	2.611	{'was_impossible': False}

📝 [3] 특정 사용자에 대한 상위 10개의 추천 영화 검색

✏️ 44번 사용자에 대한 Top 10 추천 데이터

```
N = 10
user_id = 44

topn_df = pred_df[pred_df["user_id"] == user_id]

topn_df = (
    topn_df[['user_id', 'item_id', 'pred_rating']]
    .sort_values(["pred_rating"], ascending=[False])
    .groupby("user_id")
```

```
    .head(N)
    .reset_index(drop=True)
)

topn_df
```

	user_id	item_id	pred_rating
0	44	408	4.423
1	44	169	4.385
2	44	483	4.371
3	44	12	4.339
4	44	357	4.268
5	44	114	4.254
6	44	134	4.244
7	44	178	4.238
8	44	657	4.223
9	44	174	4.216

✏️ 메타데이터와 병합하여 영화 정보 생성

```
movie_df = topn_df.merge(metadata, on="item_id", how="left")
movie_df
```

	user_id	item_id	pred_rating	title	release_date	IMDb_URL
0	44	408	4.423	Close Shave, A (1995)	28-Apr-1996	http://us.imdb.com/M/title-exact?Close%20Shave, %20A%20(1995)
1	44	169	4.385	Wrong Trousers, The (1993)	01-Jan-1993	http://us.imdb.com/M/title-exact?Wrong%20Trousers, %20The%20(1993)
2	44	483	4.371	Casablanca (1942)	01-Jan-1942	http://us.imdb.com/M/title-exact?Casablanca%20(1942)
3	44	12	4.339	Usual Suspects, The (1995)	14-Aug-1995	http://us.imdb.com/M/title-exact?Usual%20Suspects, %20The%20(1995)
4	44	357	4.268	One Flew Over the Cuckoo's Nest (1975)	01-Jan-1975	http://us.imdb.com/M/title-exact? One%20Flew%20Over%20the%20Cuckoo's%20Nest%20(1975)
5	44	114	4.254	Wallace & Gromit: The Best of Aardman	05-Apr-1996	http://us.imdb.com/Title?Wallace+ %26+Gromit%3A+The+Best+of+Aardman+Animation+(1996)

				Animation (1996)		
6	44	134	4.244	Citizen Kane (1941)	01-Jan-1941	http://us.imdb.com/M/title-exact?Citizen%20Kane%20(1941)
7	44	178	4.238	12 Angry Men (1957)	01-Jan-1957	http://us.imdb.com/M/title-exact?12%20Angry%20Men%20(1957)
8	44	657	4.223	Manchurian Candidate, The (1962)	01-Jan-1962	http://us.imdb.com/M/title-exact?Manchurian%20Candidate,%20The%20(1962)
9	44	174	4.216	Raiders of the Lost Ark (1981)	01-Jan-1981	http://us.imdb.com/M/title-exact?Raiders%20of%20the%20Lost%20Ark%20(1981)

📘 BaselineOnly의 특성

상황	예측 가능?	이유
기존 사용자 × 기존 영화	✓	b_u, b_i 존재
기존 사용자 × 신규 영화	⚠ 제한적	b_i 없음
신규 사용자 × 기존 영화	⚠ 제한적	b_u 없음
신규 사용자 × 신규 영화	거의 불가	둘 다 없음