📚 [LAB10] 지도학습 > 추천시스템 > KNNBasic

📘 #01. KNNBasic 알고리즘 개요

이웃(K-Nearest Neighbors)을 기반으로 하는 전통적 협업필터링 알고리즘임.

비슷한 사용자 또는 비슷한 아이템을 찾아서 그들의 평점을 평균하여 예측함.

Baseline이 **편향 보정 모델**이라면,
KNNBasic은 **유사도 기반 모델**임.

📝 [1] KNNBasic의 기본 개념

두 가지 방식이 존재함.

구분	설명
User-based CF	나와 비슷한 사용자들이 준 평점을 참고
Item-based CF	내가 높게 준 아이템과 비슷한 아이템 참고

📝 [2] 예측 공식 (User-based)

$$\hat{r}_{ui} = \frac{\sum_{v \in N_k(u)} sim(u,v) \cdot r_{vi}} {\sum_{v \in N_k(u)} |sim(u,v)|}$$

기호	의미
$N_k(u)$	사용자 u와 가장 유사한 k명
sim(u, v)	사용자 u와 v의 유사도
r_{vi}	이웃 사용자 v가 아이템 i에 준 평점

📝 [3] 유사도(Similarity) 종류

방법	설명
Cosine	벡터 각도 기반
Pearson	평균 제거 후 상관계수
MSD	평균 제곱 차이 기반

📝 [4] 해석 예시

사용자 A가 영화 X를 아직 보지 않았다고 가정.

- A와 비슷한 사용자 3명 존재
- 그들이 X에 준 평점: 4, 5, 4
- 유사도 가중 평균 결과: 4.3

→ A의 예측 평점은 4.3

즉, **나와 비슷한 사람의 선택을 따른 결과임**

📝 [5] KNNBasic의 의미

- 협업필터링의 대표적 알고리즘
- 직관적 이해 가능
- 설명 가능성 높음 (왜 추천되었는지 설명 쉬움)
- 데이터가 희소하면 성능 저하 가능
- 편향 보정 기능은 기본적으로 없음

📝 [6] BaselineOnly vs KNNBasic 비교

구분	BaselineOnly	KNNBasic
핵심 개념	편향 보정	유사도 기반
공식 구조	$\mu + b_u + b_i$	유사도 가중 평균
개인 성향 반영	O	간접적
이웃 정보 활용	X	O
복잡도	낮음	중간
설명 가능성	중간	높음

📝 🔎 정리

- Baseline → “기본 성향 보정”
- KNNBasic → “비슷한 사람(또는 아이템)을 참고”

추천 시스템 이론에서 Baseline은 기준선, KNN은 본격적 협업필터링의 시작 단계임.

📘 #02. 준비작업

📝 [1] 패키지 참조

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

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

```
📦 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. KNNBAsic 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 0x13f947e90>
```

📝 [2] KNNBasic 모델의 주요 하이퍼 파라미터

필수 튜닝 대상은 k, sim_options['name'], user_based 3가지임.

파라미터명	핵심도	설명	기본값	GridSearchCV 권장값
k	⭐⭐⭐	최근접 이웃 수 (참고할 사용자/아이템 개수)	40	[20, 30, 40, 50, 60, 80]
min_k	⭐⭐	예측에 필요한 최소 이웃 수	1	[1, 3, 5, 10]
sim_options['name']	⭐⭐⭐	유사도 계산 방식 (msd , cosine , pearson)	'msd'	['msd', 'cosine', 'pearson']
sim_options['user_based']	⭐⭐⭐	사용자 기반(True) / 아이템 기반(False) 선택	True	[True, False]
sim_options['min_support']	⭐⭐	유사도 계산에 필요한 최소 공통 평가 수	1	[1, 3, 5]
verbose	✅	학습 로그 출력 여부	False	[True, False]

```
%%time

# 하이퍼파라미터 그리드
param_grid = {
    "k": [30, 50],
    "min_k": [1, 3],
```

```
    "sim_options": {
        "name": ["msd", "cosine", "pearson"],
        "user_based": [True, False],
        "min_support": [1, 3, 5]
    }
}

# GridSearchCV 객체 생성 ---> 메모리 부족 에러 발생
# gs = GridSearchCV(
#     KNNBasic,
#     param_grid,
#     measures=["rmse", "mae"],
#     cv=5,
#     n_jobs=-1
# )

gs = RandomizedSearchCV(
    KNNBasic,
    param_grid,
    measures=["rmse", "mae"],
    cv=5,
    n_jobs=-1,
    random_state=52  # 재현성을 위해 시드 설정
)

# 학습
gs.fit(data)

# 최적 결과 출력
print("📌 Best RMSE Score:", gs.best_score["rmse"])
print("📌 Best Parameters:", gs.best_params["rmse"])
```

```
Computing the pearson similarity matrix...
Computing the pearson similarity matrix...
Computing the pearson similarity matrix...
Computing the pearson similarity matrix...
Computing the pearson similarity matrix...
Computing the cosine similarity matrix...
Computing the cosine similarity matrix...
Computing the cosine similarity matrix...
Computing the cosine similarity matrix...
Done computing similarity matrix.
Computing the cosine similarity matrix...
Done computing similarity matrix.
Done computing similarity matrix.
Done computing similarity matrix.
Done computing similarity matrix.
Done computing similarity matrix.
Done computing similarity matrix.
Done computing similarity matrix.
Done computing similarity matrix.
Done computing similarity matrix.
Computing the pearson similarity matrix...
Computing the pearson similarity matrix...
Computing the pearson similarity matrix...
Computing the pearson similarity matrix...
Computing the pearson similarity matrix...
Computing the pearson similarity matrix...
Computing the pearson similarity matrix...
Done computing similarity matrix.
```

```
Computing the pearson similarity matrix...
Done computing similarity matrix.
Computing the pearson similarity matrix...
Done computing similarity matrix.
Computing the pearson similarity matrix...
Done computing similarity matrix.
Done computing similarity matrix.
Done computing similarity matrix.
Done computing similarity matrix.
Done computing similarity matrix.
Done computing similarity matrix.
Done computing similarity matrix.
Computing the msd similarity matrix...
Computing the msd similarity matrix...
Computing the msd similarity matrix...
Done computing similarity matrix.
Done computing similarity matrix.
Computing the msd similarity matrix...
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the cosine similarity matrix...
Computing the cosine similarity matrix...
Done computing similarity matrix.
Computing the cosine similarity matrix...
Computing the cosine similarity matrix...
Done computing similarity matrix.
Done computing similarity matrix.
Computing the cosine similarity matrix...
Done computing similarity matrix.
Done computing similarity matrix.
Done computing similarity matrix.
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Computing the msd similarity matrix...
Done computing similarity matrix.
Done computing similarity matrix.
Computing the msd similarity matrix...
Computing the msd similarity matrix...
Computing the cosine similarity matrix...
Done computing similarity matrix.
Computing the cosine similarity matrix...
Done computing similarity matrix.
Computing the cosine similarity matrix...
Computing the cosine similarity matrix...
Computing the cosine similarity matrix...
Done computing similarity matrix.
Done computing similarity matrix.
Done computing similarity matrix.
Done computing similarity matrix.
Done computing similarity matrix.
Computing the pearson similarity matrix...
Computing the pearson similarity matrix...
Computing the pearson similarity matrix...
Computing the pearson similarity matrix...
Computing the pearson similarity matrix...
Done computing similarity matrix.
Done computing similarity matrix.
Done computing similarity matrix.
Computing the cosine similarity matrix...
Done computing similarity matrix.
Done computing similarity matrix.
Computing the cosine similarity matrix...
Computing the cosine similarity matrix...
Computing the cosine similarity matrix...
Computing the cosine similarity matrix...
Done computing similarity matrix.
```

```
Done computing similarity matrix.
Done computing similarity matrix.
Done computing similarity matrix.
Done computing similarity matrix.
📌 Best RMSE Score: 0.9744803530764052
📌 Best Parameters: {'k': 30, 'min_k': 3, 'sim_options': {'name': 'msd', 'user_based': True,
'min_support': 3}}
CPU times: user 3.01 s, sys: 183 ms, total: 3.19 s
Wall time: 9.21 s
```

📘 #04. 성능평가

📝 [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 = KNNBasic(**best_params)

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

```
Computing the msd similarity matrix...
Done computing similarity matrix.



<surprise.prediction_algorithms.knns.KNNBasic at 0x14fd1e950>
```

📝 [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=2.961052611202615, details={'actual_k': 30,
'was_impossible': False}),
 Prediction(uid=308, iid=163, r_ui=4.0, est=3.6921297070081267, details={'actual_k': 30,
'was_impossible': False}),
 Prediction(uid=327, iid=663, r_ui=4.0, est=3.7116288447244896, details={'actual_k': 30,
'was_impossible': False}),
 Prediction(uid=912, iid=479, r_ui=4.0, est=4.14022191559161, details={'actual_k': 30,
'was_impossible': False}),
 Prediction(uid=224, iid=329, r_ui=3.0, est=3.132067219144729, details={'actual_k': 30,
'was_impossible': False})]
```

📝 [4] 성능평가 지표 생성

```
cv_rmse = gs.best_score["rmse"]

train_pred = best_model.test(train_data.build_testset())
test_pred = best_model.test(test_data)

train_rmse = accuracy.rmse(train_pred, verbose=False)
train_mae  = accuracy.mae(train_pred, verbose=False)

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

rmse_gap_train = test_rmse - train_rmse
rmse_gap_cv    = test_rmse - cv_rmse
mae_gap        = test_mae - train_mae

# 과적합 판정 기준
if rmse_gap_train > 0.05:
    overfit_flag = "과적합 의심"
else:
    overfit_flag = "정상"

result_df = DataFrame({
    "Model": ["KNNBasic"],
    "CV_RMSE": [cv_rmse],
    "Train_RMSE": [train_rmse],
    "Test_RMSE": [test_rmse],
    "RMSE_Gap(Test-Train)": [rmse_gap_train],
    "RMSE_Gap(Test-CV)": [rmse_gap_cv],
    "Train_MAE": [train_mae],
    "Test_MAE": [test_mae],
    "MAE_Gap": [mae_gap],
    "Overfitting": [overfit_flag]
})
```

```
result_df
```

	Model	CV_RMSE	Train_RMSE	Test_RMSE	RMSE_Gap(Test-Train)	RMSE_Gap(Test-CV)	Train_MAE	Test_MAE	MAE_Gap
0	KNNBasic	0.974	0.744	0.978	0.234	0.003	0.579	0.772	0.193

📘 #05. TopN 추천

📝 [1] 아직 평가하지 않은 아이템에 대한 예측 수행

✏️ 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=4.139000452121666, details={'actual_k': 30,
'was_impossible': False}),
 Prediction(uid=234, iid=504, r_ui=3.5317375, est=3.7569486475889007, details={'actual_k':
30, 'was_impossible': False}),
 Prediction(uid=234, iid=73, r_ui=3.5317375, est=3.250778140875676, details={'actual_k': 30,
'was_impossible': False}),
 Prediction(uid=234, iid=475, r_ui=3.5317375, est=3.6258093141506746, details={'actual_k':
30, 'was_impossible': False}),
 Prediction(uid=234, iid=294, r_ui=3.5317375, est=3.0665961793224272, details={'actual_k':
30, '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	4.139	{'actual_k': 30, 'was_impossible': False}
1	234	504	3.532	3.757	{'actual_k': 30, 'was_impossible': False}
2	234	73	3.532	3.251	{'actual_k': 30, 'was_impossible': False}
3	234	475	3.532	3.626	{'actual_k': 30, 'was_impossible': False}
					{'actual_k': 30, 'was_impossible': False}

4	234	294	3.532	3.067	

📝 [3] 특정 사용자에 대한 상위 10개의 추천 영화 검색

✏️ 35번 사용자에 대한 Top 10 추천 데이터

```
N = 10
user_id = 35

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	35	1293	5.000
1	35	603	4.648
2	35	114	4.580
3	35	50	4.552
4	35	22	4.528
5	35	313	4.500
6	35	318	4.478
7	35	272	4.476
8	35	64	4.475
9	35	923	4.464

✏️ 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_U
0	35	1293	5.000	Star Kid (1997)	16-Jan-1998	http://us.imdb.com/M/title-exact?imdb-title-120478

1	35	603	4.648	Rear Window (1954)	01-Jan-1954	http://us.imdb.com/M/title-exact?Rear%20Window%20(1954)
2	35	114	4.580	Wallace & Gromit: The Best of Aardman Animation (1996)	05-Apr-1996	http://us.imdb.com/Title?Wallace+%26+Gromit%3A+The+Best+of+Aardman+Animation+(1996)
3	35	50	4.552	Star Wars (1977)	01-Jan-1977	http://us.imdb.com/M/title-exact?Star%20Wars%20(1977)
4	35	22	4.528	Braveheart (1995)	16-Feb-1996	http://us.imdb.com/M/title-exact?Braveheart%20(1995)
5	35	313	4.500	Titanic (1997)	01-Jan-1997	http://us.imdb.com/M/title-exact?imdb-title-120338
6	35	318	4.478	Schindler's List (1993)	01-Jan-1993	http://us.imdb.com/M/title-exact?Schindler's%20List%20(1993)
7	35	272	4.476	Good Will Hunting (1997)	01-Jan-1997	http://us.imdb.com/M/title-exact?imdb-title-119217
8	35	64	4.475	Shawshank Redemption, The (1994)	01-Jan-1994	http://us.imdb.com/M/title-exact?Shawshank%20Redemption,%20The%20(1994)
9	35	923	4.464	Raise the Red Lantern (1991)	01-Jan-1991	http://us.imdb.com/M/title-exact?Da%20Hong%20Deng%20Long%20Gao%20Gao%20Gua%20(19