Wrap-up 리포트

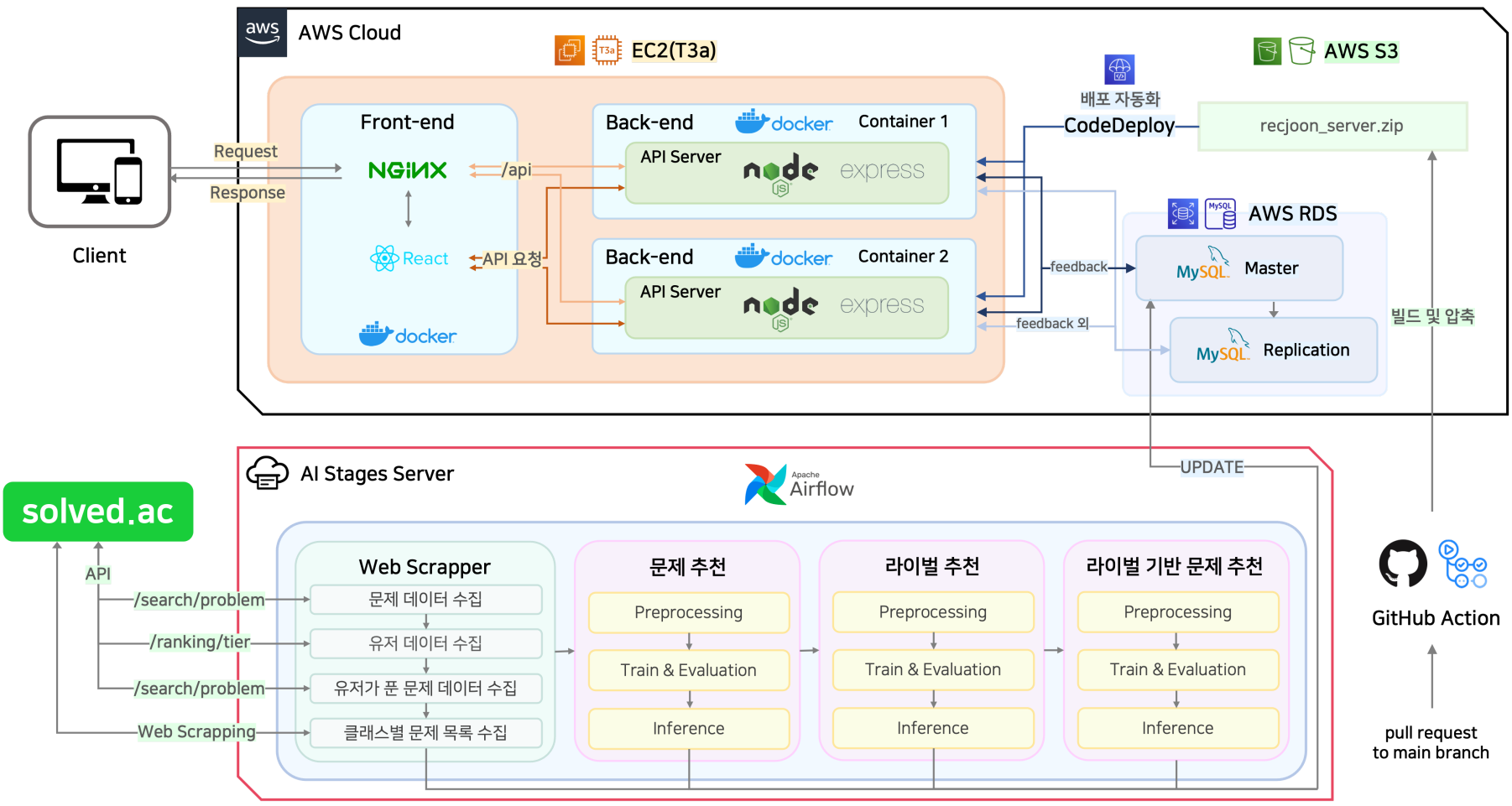
1. 프로젝트 개요

- 프로젝트 주제
    - 딥 러닝과 머신 러닝을 사용하여 Baekjoon Online Judge(BOJ)와 solved.ac 유저의 개인별 문제 풀이 이력을 바탕으로 본인의 수준에 맞는 문제와 라이벌을 추천하는 AI 모델을 개발하고 실제 웹 서비스로 배포합니다.

- 프로젝트 개요(프로젝트 구현 내용, 컨셉, 교육 내용과의 관련성 등)
    - 데이터 특성
        - 유저 정보
            - `handle`, `solved_count`, `user_class`, `tier`, `rating`, `rating_by_problems_sum`, `rating_by_class`, `rating_by_solved_count`, `exp`, `rival_count`, `reverse_rival_count`, `max_streak`, `rank`, `organization`
        - 문제 정보
            - `problem_id` , `title`, `tags`, `is_solvable`, `accepeted_user_count`, `level`, `average_tries`
        - 유저별 문제 풀이 이력
    - 데이터 정보
        - 사용자의 수 : 65,000명
        - 문제의 개수 : 약 23,000개
        - 사용자별 푼 문제의 총 개수 : 약 840만개
    - 구현 내용
        - 사용자별 문제 풀이 이력을 바탕으로 다음에 풀어볼만한 문제를 추천합니다.
        - 본인의 실력과 비슷한 라이벌을 추천해줍니다.
        - 본인의 라이벌과 비교하여 풀어볼만한 문제를 추천해줍니다.

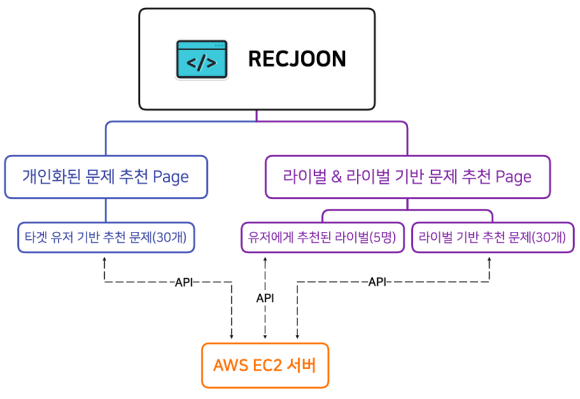
- 활용 장비 및 재료(개발 환경, 협업 tool 등)
    - GPU : v100
    - 코드 공유
        - GitHub
    - 개발 환경
        - AWS, JupyterLab, VS Code, PyCharm
    - 모델 성능 분석
        - Wandb
    - 피드백, 일정관리 및 의견 공유
        - Notion, Zoom, Jira

- 프로젝트 아키텍처

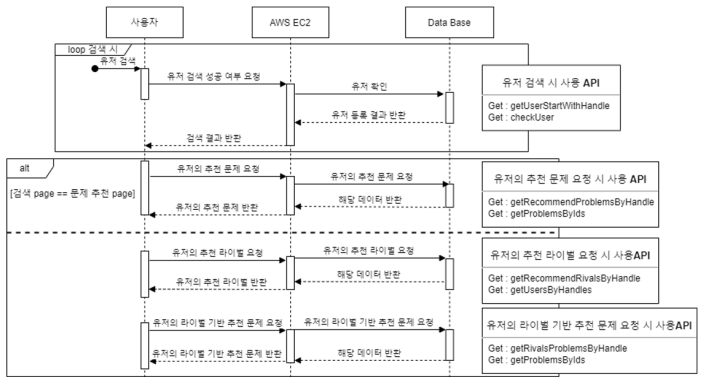


- 두 가지의 클라우드 컴퓨팅 사용
  - 웹 서버 운영을 위한 웹 서버와 데이터베이스를 위한 AWS
    - 프론트엔드와 백엔드를 Docker 컨테이너로 실행하기 위한 EC2 인스턴스 사용
      - NGINX를 통해 두 개의 백엔드 Docker 컨테이너를 사용한 blue-green deployment 기법으로 무중단 배포를 구현
    - AWS RDS를 통해 MySQL 데이터베이스 구축
      - 데이터를 쓰거나 수정할 수 있는 Master DB
      - Master DB를 복제한 오직 읽기 전용의 Replication DB
    - GitHub Action과 AWS CodeDeploy를 사용하여 CI&CD 자동화를 구현
  - 추천 모델 실행을 위한 AI Stages에서 제공하는 서버
    - Airflow를 통해 주기별로 정해진 DAG workflow에 따라서 실행
    - 용도에 따른 두 가지의 모듈
      - 데이터 수집을 위한 웹 스크레이핑 모듈
        - solved.ac API로부터 유저, 문제, 유저 문제 풀이 이력 데이터 수집
        - solved.ac로부터 클래스별 문제 목록 데이터를 웹 스크레이핑
      - 딥 러닝과 머신 러닝 모델을 학습하고 예측하는 모듈
        - 모델이 예측한 Inference 결과가 최종적으로 데이터베이스 테이블에 저장

- Sitemap



- UML Sequence Diagram



2. 프로젝트 팀 구성 및 역할

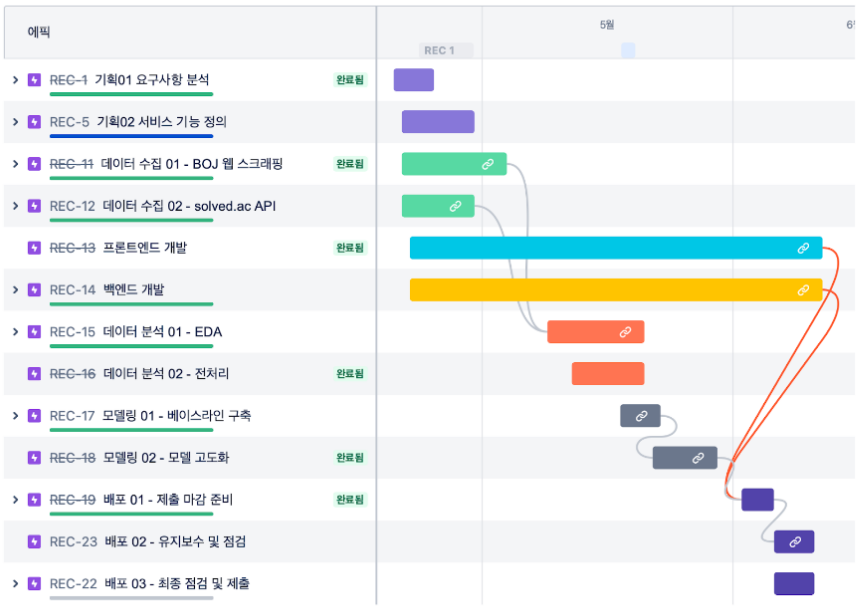
팀원	역할
김은선	라이벌 추천 모델링, 라이벌 기반 문제 추천 모델링, 테스크 자동화, 모델 실행 코드 모듈화

팀원	역할
박정규	데이터 EDA, Front-end 개발, GCP로 Airflow 이전
이서희	데이터 EDA, 라이벌 추천 모델링, 라이벌 기반 문제 추천 모델링, 온오프라인 지표 개발
이선호	Back-end 개발, Front-end 디자인, CI & CD 자동화, 문제 추천 모델 전처리
진완혁	데이터 수집과 EDA, 문제 추천 모델링, 티어 필터링

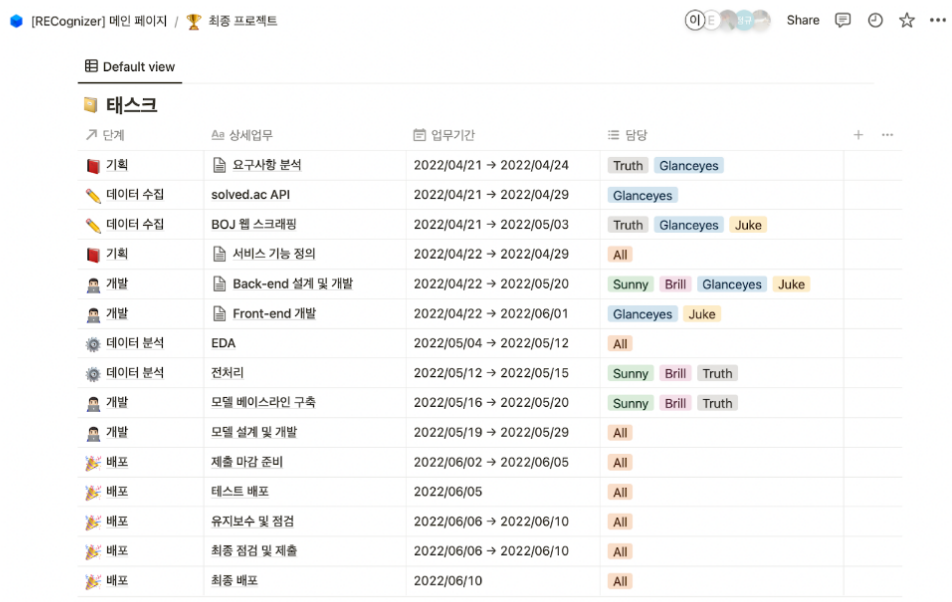
3. 프로젝트 수행 절차 및 방법

프로젝트 타임라인

Jira를 통한 협업 및 로드맵 구축



Notion을 활용한 태스크 및 실험 관리



EDA

저희는 solved.ac가 제공하는 API에서 받을 수 있는 Baekjoon Online Judge(BOJ)의 유저와 알고리즘 문제 데이터를 사용했습니다.

사용한 데이터

- **유저 데이터**

	id	handle	solved_count	user_class	tier	rating	rating_by_problems_sum	rating_by_class	rating_by_solved_count	exp	rival_count
0	1	koosaga	11075	10	31	3252	2802	250	175	14225816820	5
1	2	cki86201	6059	10	31	3231	2781	250	175	10292773001	0
2	3	mitnegativeinfinity	2188	10	31	3180	2730	250	175	6393679340	0
3	4	ainta	4111	10	31	3135	2685	250	175	6940261423	0
4	5	yclock	2670	10	31	3116	2666	250	175	4132785157	0
...	...	...	...	...	...	...	...	...	...	...	...
65168	65185	zpqmdh	0	0	0	0	0	0	0	0	0
65169	65186	zumge98	0	0	0	0	0	0	0	0	0
65170	65187	zxcv123	0	0	0	0	0	0	0	0	0
65171	65188	zxr83	0	0	0	0	0	0	0	0	0
65172	65189	zzzqw11	0	0	0	0	0	0	0	0	0

65173 rows × 15 columns

- 주요 column 소개
  - `handle` : 사용자 명 입니다.
  - `solved_count` : 사용자가 푼 문제 수입니다.
  - `user_class` : 사용자가 취득한 Class입니다. (Class : 각 단계 별 Class에 지정된 문제를 풀면 취득할 수 있는 Class입니다.)
  - `tier` : 브론즈 5 → 1, 브론즈 4 → 2, … ,루비 1→ 30, 마스터→31로 표현하는 사용자 티어입니다.
  - `rating` : 사용자의 레이팅입니다.
  - `rival_count` : 사용자의 라이벌 수입니다.
- 추가 주요 column

- `rank` : 사용자의 순위입니다.

- **문제 데이터**

	id	problem_id	title	tags	is_solvable	accepted_user_count	level	average_tries
0	1	1000	A+B	arithmetic,implementation,math	True	175152	1	2
1	2	1001	A-B	arithmetic,implementation,math	True	146736	1	1
2	3	1002	터렛	geometry,math	True	25716	7	4
3	4	1003	피보나치 함수	dp	True	34653	8	3
4	5	1004	어린 왕자	geometry	True	8438	8	2
...	...	...	...	...	...	...	...	...
22983	22984	25028	Fully Generate	dp,exponentiation_by_squaring,math	True	3	14	2
22984	22985	25029	Joyful KMP	NaN	True	3	22	1
22985	22986	25030	Polynomial Quine	gaussian_elimination,linear_algebra,math,numbe...	True	2	23	1
22986	22987	25031	Unifying Values	dp,prefix_sum	True	4	14	1
22987	22988	25032	Young Zebra	bfs,dfs,graphs,graph_traversal	True	4	16	1

  - 주요 column 소개
    - `problem_id` : 문제 ID 입니다.
    - `title` : 문제의 제목입니다.
    - `tags` : 문제의 유형 태그 목록입니다.
    - `level` : unrated → 0, 브론즈 5 → 1, … , 루비 1 → 30으로 표현하는 문제 난이도입니다.

- **유저별 푼 문제 데이터**

	id	handle	problems
0	1	koosaga	1000,1001,1002,1003,1004,1005,1006,1007,1008,1...
1	2	cki86201	1000,1001,1002,1003,1004,1005,1006,1007,1008,1...
2	3	mitnegativeinfinity	1000,1001,1019,1056,1067,1144,1150,1311,1372,1...
3	4	ainta	1000,1001,1002,1003,1004,1005,1007,1008,1009,1...
4	5	yclock	1000,1001,1002,1003,1004,1005,1007,1008,1009,1...
...	...	...	...
63871	63872	ohjcms1	2557,10718,15596
63872	63873	steven1010	2557
63873	63874	wwowwww	1000,1001,1008,1330,2438,2439,2557,2588,2739,2...
63874	63875	yijh061016	NaN
63875	63876	yoonkoo2001	NaN

  - 주요 column 소개
    - `handle` : 사용자 명 입니다.
    - `problems` : 사용자가 푼 문제의 목록입니다.

문제 데이터 분석

- **문제 데이터의 NA 값**

```
df_problems.isna().sum(axis=0)/22988
```

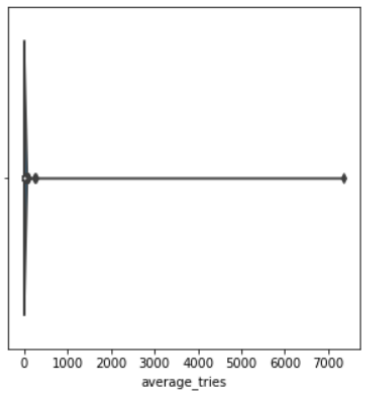
```
id                  0.000000
problem_id          0.000000
title               0.000044
tags                0.382678
is_solvable         0.000000
accepted_user_count 0.000000
level               0.000000
average_tries       0.000000
dtype: float64
```

  - tag속 NA값이 전체 중 38%의 비율을 차지했습니다. → 태그를 input으로 사용 시 전처리 필요합니다.

- **문제 데이터 속 연속형 변수 column 중 이상치 확인**

- average_tries

```
fig, ax= plt.subplots(figsize=[5,5])
ax= sns.boxplot(df_problems.average_tries)
ax= sns.violinplot(df_problems.average_tries)
```




```
df_problems[df_problems.average_tries == 7340]
```

	id	problem_id	title	tags	is_solvable	accepted_user_count	level	average_tries
9158	9159	10944	랜덤 게임~~	NaN	True	44	0	7340

- 문제 시도 횟수가 7,340인 이상치가 존재해서 제거한 후 진행하기로 결정했습니다.

- **풀 수 없는 문제 제외**

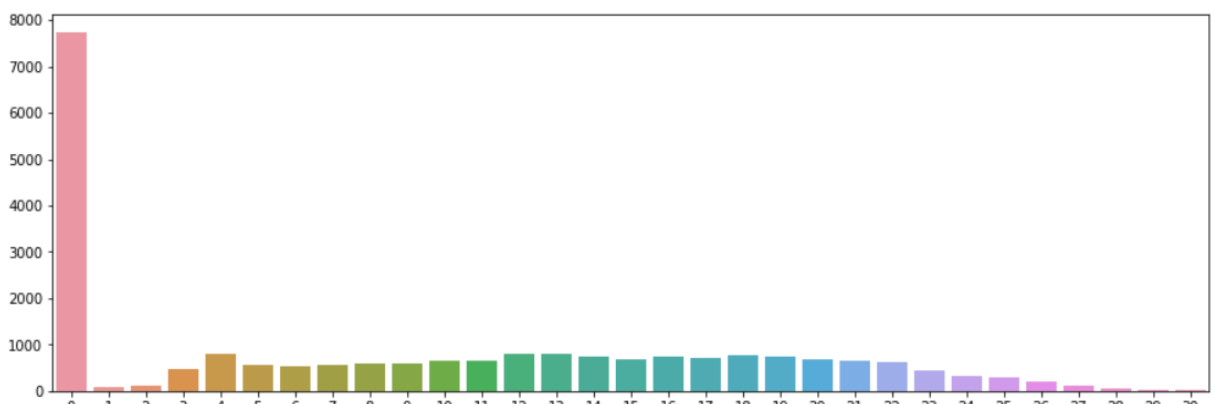
```
# is_solvable == False인 것들 확인
non_solvable = df_problems.loc[df_problems['is_solvable'] == False]
non_solvable
```

	id	problem_id	title	tags	is_solvable	accepted_user_count	level	average_tries
853	854	1861	경로 찾기	NaN	False	0	0	0
1031	1032	2046	이어달리기	NaN	False	0	0	0
1180	1181	2199	DNA 해독 2	ad_hoc	False	0	15	0
1595	1596	2627	트리회전	NaN	False	0	0	0
1607	1608	2639	주차장	NaN	False	0	0	0
...	...	...	...	...	...	...	...	...
22038	22039	24021	Zathras	NaN	False	0	0	0
22039	22040	24022	Seating Chart	NaN	False	0	0	0
22150	22151	24135	ロゴマーク (Logo)	NaN	False	0	0	0
22707	22708	24734	Eerie Shadows	NaN	False	0	0	0
22708	22709	24735	Getting Square	NaN	False	0	0	0

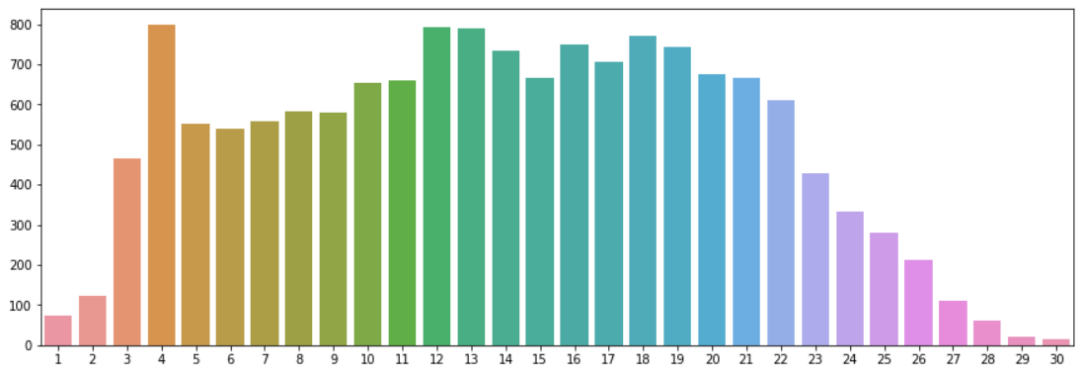
303 rows × 8 columns

- is_solvable = False인 문제(303개)를 데이터에서 제외했습니다.

- **문제 난이도 분포**



- Unrated(level : 0) 문제가 너무 많습니다. → level 0인 문제 제외 후 다시 분포 확인해 볼 필요가 있었습니다.



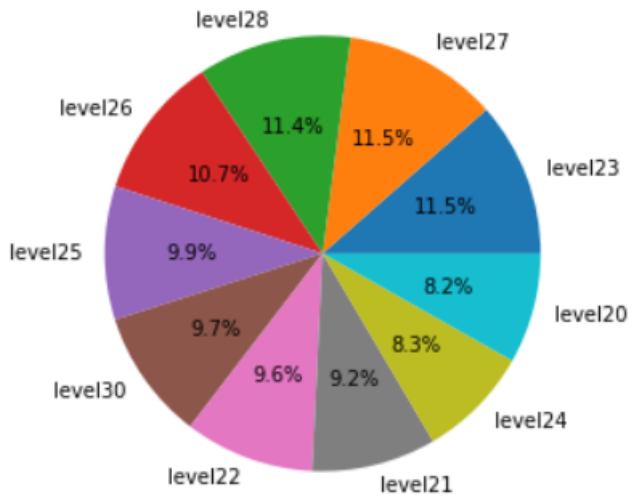
  - 중간 정도의 난이도를 가진 문제들이 대체로 많이 난이도가 매겨졌습니다.

- **각 문제의 태그가 각 난이도에 등장하는 분포**
  - 태그는 모두 184개 정도 존재합니다.

	tag	level1	level2	level3	level4	level5	level6	level7	level8	level9	...	level21	level22	level23	level24	level25	level26
0	math	0.652778	0.739837	0.732759	0.350438	0.324864	0.288355	0.313620	0.292096	0.302245	...	0.238908	0.240631	0.248619	0.250000	0.268908	0.263158
1	arithmetic	0.625000	0.528455	0.482759	0.161452	0.087114	0.031423	0.023297	0.022337	0.018998	...	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
2	implementation	0.569444	0.536585	0.596983	0.760951	0.758621	0.528651	0.433692	0.353952	0.281520	...	0.058020	0.061144	0.058011	0.108209	0.084034	0.070175
3	arbitrary_precision	0.125000	0.016260	0.015086	0.016270	0.007260	0.025878	0.021505	0.017182	0.008636	...	0.011945	0.009862	0.008287	0.003731	0.004202	0.017544
4	combinatorics	0.013889	0.008130	0.008621	0.008761	0.018149	0.011091	0.025090	0.037801	0.022453	...	0.046075	0.037475	0.066298	0.048507	0.071429	0.070175
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
179	suffix_tree	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000	0.000000	0.000000	0.003731	0.000000	0.017544
180	directed_mst	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000	0.000000	0.000000	0.003731	0.000000	0.011696
181	a_star	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000	0.000000	0.000000	0.000000	0.004202	0.000000
182	rb_tree	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000	0.000000	0.000000	0.000000	0.000000	0.005848
183	top_tree	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000	0.000000	0.000000	0.000000	0.000000	0.005848

- 이처럼 각 태그들은 각 난이도 별 등장하는 비율이 모두 상이합니다.

◦ 가장 대표적인 tag인 data_structures에 대한 상위 난위도에서의 분포입니다.



- 이처럼, 대표적이고 기본이 되는 tag는 난이도 별 등장 분포가 골고루 나오기도 합니다.

유저 데이터와 유저-문제 Interaction (유저가 문제를 푼 데이터)

- **유저가 많이 푼 tag**

	tag	tag_count
0	math	3838
1	implementation	3416
2	dp	2418
3	graphs	2262
4	data_structures	2221
5	string	1522
6	greedy	1344
7	bruteforcing	1304
8	graph_traversal	1239
9	sorting	1050

◦ 추후에 대표 tag를 선정해 Input으로 사용해도 좋을 것이라고 판단했습니다.

- **푼 문제 수가 0인 유저**

```
print('푼 문제 수가 0인 유저(1,297명) 중 rating이나 exp가 00이 아닌 유저들:',len(df_users[(df_users.solved_count == 0) & ((df_users.rating !=0)|(df_users.exp !=0))]))
```

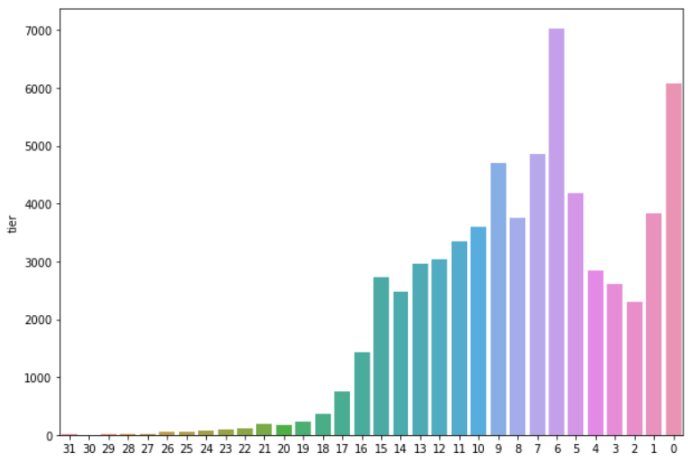
```
푼 문제 수가 0인 유저(1,297명) 중 rating이나 exp가 00이 아닌 유저들: 24
```

◦ 해당 유저들은 데이터를 받아오는 solved.ac에서 제재당한 유저일 가능성이 큽니다.

- 대표적인 solved.ac에서 사용자를 제재하는 경우 : 부정행위로 문제를 맞춘 경우
- 이와 비슷한 경우로 발생한 데이터로는 푼 문제 수가 0개가 아니지만 rating과 exp가 모두 0인 유저들이 데이터로 남아있습니다.

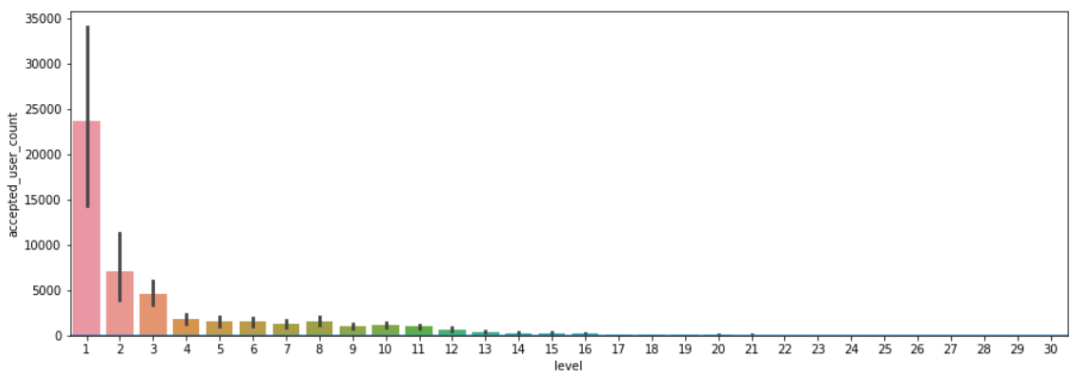
◦ 1297명은 푼 문제에 대한 정보가 없으므로, 데이터에서 제거하기로 판단했습니다.

- **유저의 Tier 분포**

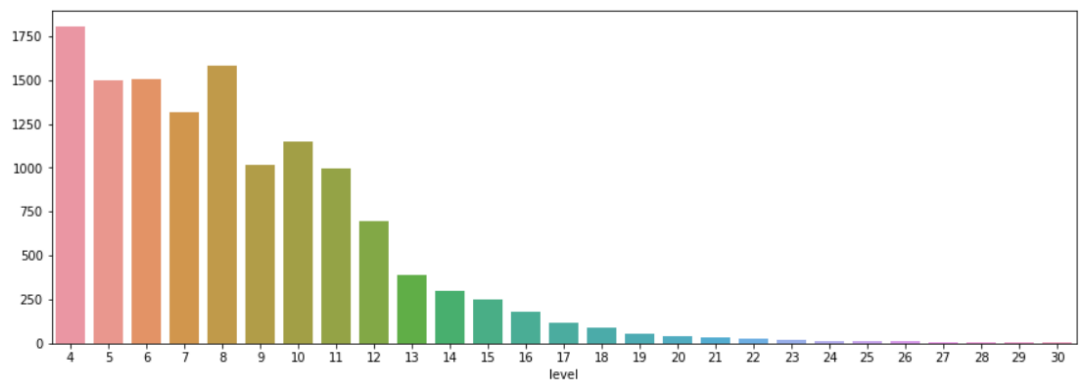


- 중하위권 tier인 6~7 사이에 가장 많은 유저가 분포되어 있습니다.

- **문제 난이도 별 맞힌 사용자 수 분포**



- 난이도가 1~3인 문제를 제외한 분포



- 대체로 입문 단계인 하위권 문제가 많이 풀렸고, 하위권 문제를 제외하여도 문제의 난이도가 높아질 수록 맞힌 유저의 수는 급격히 줄어듭니다.

- **유저와 유저의 문제 풀이에 관한 분석**
  - **가설 : 유저 Class 별 많이 푸는 tag가 있을 것이다.**

```
# 클래스 10인 유저가 많이 푼 태그 목록
class_num = 10

tmp_10 = class_user_tag(class_num)
tags_10_sum = pd.DataFrame(tmp_10[tmp_10.user_class == class_num].sum(axis=0)[unique_tags]).reset_index()
tags_10_sum.columns = ['tag', 'num']
tags_10_sum = tags_10_sum.sort_values('num', ascending=False).reset_index(drop=True)
tags_10_sum#.head()
```

```
100%|██████████| 19/19 [00:22<00:00,  1.20s/it]
```

	tag	num
0	math	15201
1	implementation	12845
2	data_structures	9368
3	dp	9159
4	graphs	8114
...	...	...
179	discrete_sqrt	5
180	rb_tree	3
181	pick	3
182	generating_function	2
183	a_star	0

184 rows × 2 columns

```
# 클래스 6인 유저가 많이 푼 태그 목록
class_num = 6

tmp_6 = class_user_tag(class_num)
tags_6_sum = pd.DataFrame(tmp_6[tmp_6.user_class == class_num].sum(axis=0)[unique_tags]).reset_index()
tags_6_sum.columns = ['tag', 'num']
tags_6_sum = tags_6_sum.sort_values('num', ascending=False).reset_index(drop=True)
tags_6_sum.head(10)
```

```
100%|██████████| 493/493 [02:30<00:00,  3.28it/s]
```

	tag	num
0	math	102784
1	implementation	100013
2	graphs	57938
3	dp	56555
4	data_structures	48240
5	graph_traversal	36531
6	arithmetic	35454
7	string	34259
8	bruteforcing	28813
9	sorting	26934

```
# 클래스 3인 유저가 많이 푼 태그 목록
class_num = 3

tmp_3 = class_user_tag(class_num)
tags_3_sum = pd.DataFrame(tmp_3[tmp_3.user_class == class_num].sum(axis=0)[unique_tags]).reset_index()
tags_3_sum.columns = ['tag', 'num']
tags_3_sum = tags_3_sum.sort_values('num', ascending=False).reset_index(drop=True)
tags_3_sum.head(10)
```

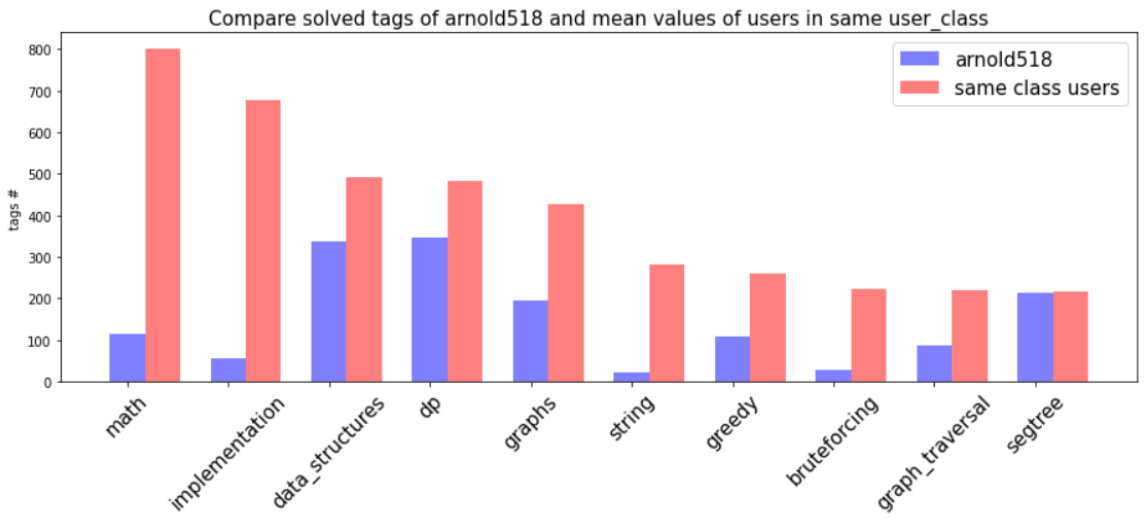
```
100%|██████████| 5326/5326 [10:01<00:00,  8.86it/s]
```

	tag	num
0	implementation	475529
1	math	414880
2	arithmetic	179009
3	dp	150415
4	graphs	149483
5	string	138131
6	data_structures	129951
7	graph_traversal	123421
8	bruteforcing	107981
9	sorting	107517

- 유저의 Class 별로 상위 10개의 tag 분포가 조금씩 상이한 것을 확인할 수 있다.

> 💡 유저 별 수준과 tag는 상관관계가 있다고 보았고, tag 정보를 유저가 푸는 문제를 파악하는데 사용할 수 있을 것으로 판단했습니다.

- **같은 Class 내 tag 별 평균 풀이 횟수와 한 유저의 tag 풀이 횟수와의 비교**
  - 유저 Class가 10인 유저를 무작위로 뽑은 후 같은 Class 내 tag 별 풀이 수 분포 비교



> 💡 무작위로 뽑은 arnold518 유저는 같은 Class 내 유저들에 비해 implementation, string, bruteforcing의 풀이 수가 현저히 적은 것을 발견할 수 있었습니다.
> → 유저에게 같은 Class 내 다른 유저들의 풀이 이력과 비교하여 많이 풀지 않은 tag 문제 위주로 추천하는 방법을 떠올려볼 수 있었습니다.

라이벌 데이터

```
print('rival 수가 0인 유저 비율:',sum(df_users.rival_count==0)/len(df_users))
print('reverse_rival 수가 0인 유저 비율:',sum(df_users.reverse_rival_count==0)/len(df_users))
print('reverse_rival과 rival 수가 0인 유저 비율:',sum((df_users.reverse_rival_count==0) & (df_users.rival_count==0))/len(df_users))
```

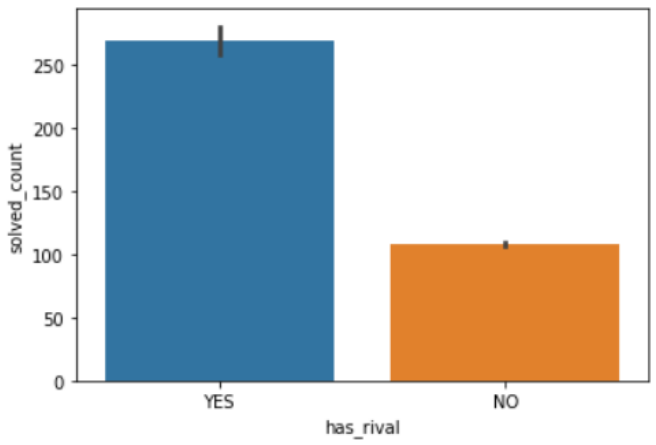
```
rival 수가 0인 유저 비율: 0.9186548938568476
reverse_rival 수가 0인 유저 비율: 0.8868119481495398
reverse_rival과 rival 수가 0인 유저 비율: 0.8689335587701171
```

💡 현재 전체 유저 중 약 87%가 라이벌 기능을 사용하고 있지 않습니다. 저희는 라이벌을 추천해주는 기능을 구현하려 하므로, 라이벌에 대한 데이터 분석을 통해 어떤 추천 전략을 사용해야 하는지 알아보려 했습니다.

- **가설 1 : 라이벌이 있는 유저들은 그렇지 않은 유저들에 비해 문제를 더 많이 푼다.**

```
df_check_rival = df_users.copy()
df_check_rival['has_rival'] = 'NO'

idx = df_check_rival[df_check_rival.rival_count >= 1].index
df_check_rival.loc[idx, 'has_rival'] = 'YES'
df_check_rival
```



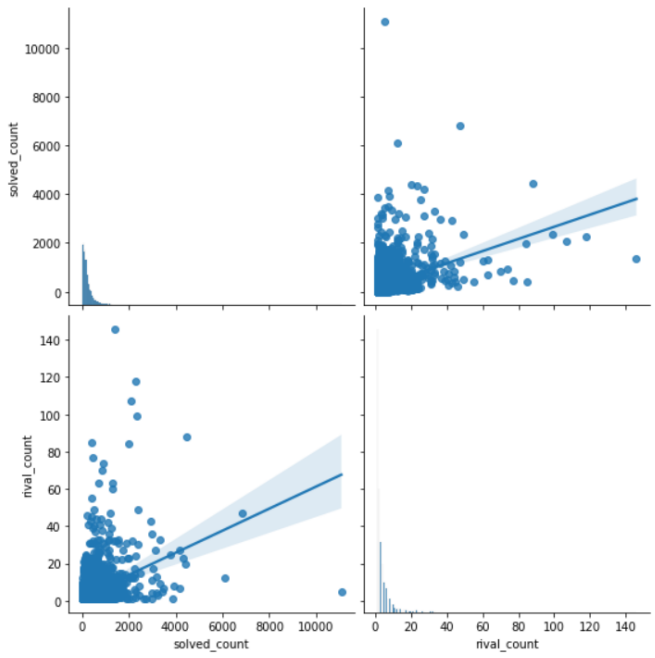
  - 라이벌이 있는 유저들이 그렇지 않은 유저들에 비해 2배 이상 문제를 더 많이 푸는 모습을 확인할 수 있습니다.
  - 라이벌의 여부와 문제 푸는 횟수는 약한 양의 상관관계를 보입니다.

```
df_check_rival['has_rival'] = np.where(df_check_rival['has_rival']=='YES', 1 ,0)
stats.pointbiserialr(df_check_rival['has_rival'], df_check_rival['solved_count']).correlation
```

```
0.199235382987701
```

💡 가설에 해당하는 사실을 확인할 수 있었습니다. 라이벌을 사용자들에게 추천을 해주어서 유저들에게 동기를 부여할 수 있다면, 개인 실력 향상과 백준 사이트의 활성화를 기대해볼 수 있을 것이라 생각했습니다.

- **가설 2 : 라이벌이 많을 수록 더 많은 문제를 푼다.**
  - 이 가설에 알아보기 위한 분석을 통해서 추천할 적절한 라이벌의 수를 알아볼 수 있을 것으로 생각했습니다.



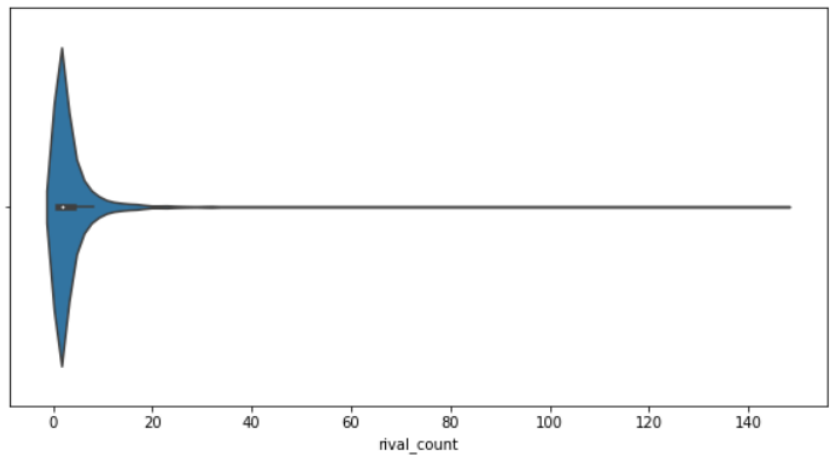
  - 라이벌 수와 푼 문제의 수 사이에는 약한 관계성이 있어보입니다.
    - 상관관계 : 약 0.39
    - p-value : 약 1.31

💡 라이벌의 수와 문제를 푸는 횟수는 약한 상관관계를 보입니다.
따라서, 적절히 큰 라이벌 수를 정하면 사용자의 더 많은 문제 풀이를 기대할 수 있을 것입니다.

- **라이벌을 가진 사용자들의 평균적인 라이벌 수**

```
plt.figure(figsize=(10,5))
sns.violinplot(x='rival_count', data=df_has_rival)
```

```
<AxesSubplot:xlabel='rival_count'>
```



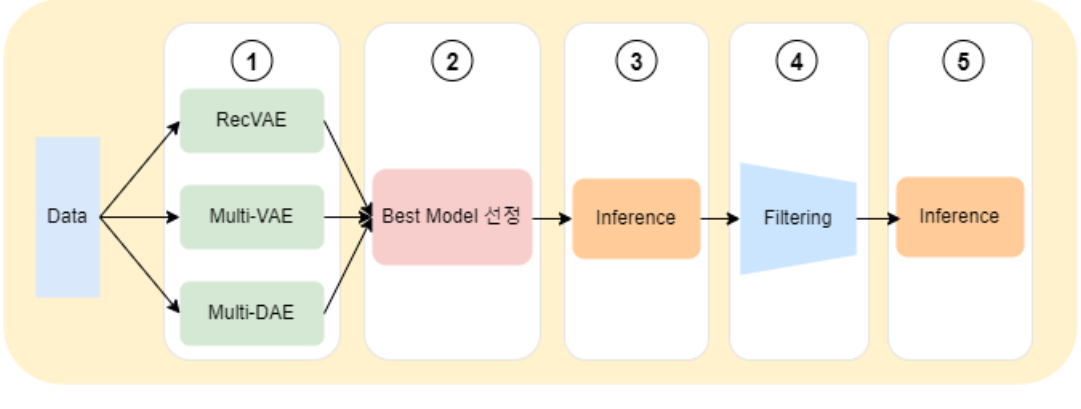
  - 100명 이상의 라이벌을 가진 사용자는 outside point로 보고 제외한 후 평균적인 라이벌의 수를 알아보았습니다.

```
# 평균적인 rival 수 확인
print("평균 rival 수:", round(np.mean(df_has_rival.rival_count), 2),'명')
```

```
평균 rival 수: 3.68 명
```

💡 outside point : 20명 이상
median : 약 2.5명
mean : 약 3.65명
→ 추천할 라이벌의 수를 3명보다는 조금 더 많은 5~6명으로 정하자는 결정이 나왔습니다.

문제 추천 모델



문제 추천은 위 그림의 5가지 과정으로부터 도출됩니다.

1. 데이터로부터 RecVAE, Multi-VAE, Multi-DAE 세 모델에 대해 각각 훈련을 진행합니다.
2. 훈련한 모델들 중 Recall @ 30 가 가장 높은 모델을 선정합니다.
3. Best Model로부터 1차 추론을 진행합니다.
4. 사용자의 티어와 문제의 레벨이 많은 차이가 나는 경우 Filtering을 거쳐 걸러냅니다.
   - 플래티넘 레벨 이상의 유저에 대해 x가 레벨이라면 $x - log_2 x$이상 $x + log_3 x$ 이하의 문제가 나오도록 했습니다.
5. 본인의 티어에 맞는 문제가 나오도록 최종적인 추론을 진행합니다.

실험한 모델명	출처
RecVAE(Variational AutoEncoder)	Ilya Shenbin, Anton Alekseev, Elena Tutubalina, Valentin Malykh, and Sergy I. Nikolenko. 2019. RecVAE: A New Variational Autoencoder for Top-N Recommendations with Implicit Feedback. ACM
Multi-VAE	Dawen Liang, Rahul G. Krishnan, Matthew D. Hoffman, Tony Jebara. 2018. Variational Autoencoders for Collaborative Filtering', WWW '18: Proceedings of the 2018 World Wide Web Conference
Multi-DAE(Denoising AutoEncoder)	Dawen Liang, Rahul G. Krishnan, Matthew D. Hoffman, Tony Jebara. 2018. Variational Autoencoders for Collaborative Filtering', WWW '18: Proceedings of the 2018 World Wide Web Conference

- 모델 설명
  - RecVAE
    - KL Divergence을 구할 때 새로운 인코더 아키텍처를 구성했습니다.
    - Prior Distribution에서 $p(z)$ 대신 $p(z|\varphi old, x)$를 사용했습니다.

        - Beta - VAE 논문에서 활용한 Beta Hyperparameter 적용했습니다.
        - Encoder와 Decoder를 번갈아 가면서 학습하는 Alternating Update 적용했습니다.
    - Multi-VAE와 Multi-DAE
        - Encoder와 Decoder를 통해 데이터를 재구성하는 AutoEncoder 모델입니다.
        - 문제의 태그에 관한 임베딩을 Encoder의 입력으로 같이 넣어서 문제에 관한 Side Information도 같이 학습할 수 있도록 하여 성능을 향상시키고자 했습니다.
        - Yifan Chen, and Maarten de Rijke. 2017. A Collective Variational Autoencoder for Top-N Recommendation with Side Information. ACM
- 지표 설명
    - Recall @ K : 유저의 관심 아이템 중 추천된 K개의 비율을 평가하는 지표.

모델별 성능 확인

모델명	Recall @ 30
RecVAE	0.7168
Multi-VAE	0.7153
Multi-DAE	0.7182

저희의 의도에 맞게 데이터가 업데이트 됨에 따라 Recall의 값이 가장 높은 모델을 선정하여 추론을 진행하도록 했습니다.

❗데이터의 업데이트에 따라 각 모델의 성능이 변경될 수 있습니다. (2022.06.06 기준)

라이벌 추천 모델

유저 정보와 아이템 정보를 활용해 타겟 유저와 유사한 유저 6명을 추천함으로써 알고리즘 학습에 동기를 부여하고자 했습니다.

실험한 모델명	출처
Collaborative MF(Matrix Factorization)	David Cortes. 2020. Cold-start recommendations in Collective Matrix Factorization
K-nearest neighbors	Altman, and Naomi S. 1992. An introduction to kernel and nearest-neighbor nonparametric regression

- **모델 설명**
    - 두 모델 모두 input feature로 user feature는 유저의 티어, 클래스, 문제 풀이 수, rating값을 활용했습니다.
    - **Collaborative MF:**
        - matrix factorization모델에 side infomation을 활용해 학습할 수 있는 모델로 유저 정보와 같은 변수들을 추가적으로 추천에 활용하고자 시도해보았습니다.
    - **K- nearest neighbors:**
        - KNN 알고리즘은 데이터의 특성 기반으로 제일 근접한 k개의 요소라벨을 참조하여 분류하는 알고리즘으로, feature를 기반으로 k명의 유사한 유저를 찾아내는데 적합하다고 판단했습니다.
        - user feature에 더해 item feature로는 각 유저가 난이도 별로 푼 문제 수를 활용했습니다.

- **지표 설명**

✔️ **라이벌 추천 지표**

- **문제 난이도 합**(라이벌 수:6 , 최대 난이도 합: 3000)

$$\frac{sum(\frac{|나의난이도합-라이벌난이도합|}{최대난이도합})}{라이벌수}$$

- **문제 클래스 보너스** (라이벌 수:6, 최대 클래스 보너스 점수: 3450)

$$\frac{sum(\frac{|나의클래스보너스점수-라이벌클래스보너스점수|}{최대클래스보너스점수})}{라이벌수}$$

- **푼 문제의 수 보너스** (라이벌 수:6, 최대 문제풀이 보너스 점수: 175)

$$\frac{sum(\frac{|나의문제풀이보너스점수-라이벌문제풀이보너스점수|}{최대문제풀이보너스점수})}{라이벌수}$$

- 위 산식을 통해 각 지표별로 나와 상대방의 차이가 어느정도 나는지를 산출 할 수 있습니다.
    - 세 지표의 평균 값이 0에 가까울 수록 유사유저를 잘 골라낸 모델입니다.

모델명	문제 난이도 합	문제 클래스 보너스	푼 문제의 수 보너스	평균
KNN	**0.00826**	**0.17212**	**0.19261**	**0.12433**
CMF	0.23473	0.12532	0.13017	0.16341

❗데이터의 업데이트에 따라 각 모델의 성능이 변경될 수 있습니다. (2022.06.06 기준)

라이벌 기반 문제 추천 모델

라이벌들에 비해 부족한 부분을 보완하고 실력을 향상시키기 위한 목적의 기능입니다.
라이벌들이 푼 문제들 중 내가 풀 문제들을 최대 30개 추천합니다. 5명의 라이벌들은 풀었지만 타겟 유저가 풀지 않은 문제가 30개보다 적은 경우 그대로 추천되지만, 만일 30개보다 많을 경우 사용자의 문제 풀이 패턴을 고려하여 문제를 필터링하기위해 아래 3가지 모델들을 활용하여 비교 및 적용하였습니다.

실험한 모델명	출처
BPR(Bayesian Personalized Ranking)	Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. 2009. BPR: Bayesian Personalized Ranking from Implicit Feedback
ALS(Alternating Least Squares) MF	Yifan Hu, Yehuda Koren, and Chris Volinsky. 2008. Collaborative Filtering for Implicit Feedback Datasets
Item-based CF(Collaborative Filtering)	Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. 2001. Item-based Collaborative Filtering Recommendation Algorithms

문제에 대한 전처리: is_solvable=True이고 level이 0이 아닌 문제들

사용한 주요 태그: `data_structures` `geometry` `graph` `greedy` `implementation` `math` `string` `dp`

- **모델 설명**
  - **Alternating Least Squares Matrix Factorization:**
    - mf모델은 유저의 문제풀이 이력이 담긴 행렬을 유저와 아이템의 잠재 벡터로 나누고, 두 벡터를 내적한 값이 원본 행렬과 같아지도록 복원하는 과정에서 유저의 문제 풀이 패턴을 학습할 수 있습니다.
    - 해당 학습과정을 거쳐 나온 행렬에서 추천대상이 되는 문제로 필터링을 거친 후, 풀 확률이 큰 문제들이 우선적으로 추천되도록 설계했습니다.
    - SGD의 오랜 학습 시간 단점을 극복하기 위해 ALS가 반영되었습니다.
  - **Bayesian Personalized Ranking:**
    - 앞서 설명한 MF와 비슷한 방식의 전처리를 진행하였습니다. 다만 기존의 MF의 최적화 기법은 랭킹을 고려하지 않은 방식입니다.
    - BPR은 Pairwise ranking loss를 최소화하여 MF의 임베딩을 학습하는 모델입니다.
    - 따라서 베이지안 추론에 기반한 최적화 기법을 반영한 BPR을 적용하여 문제에 대한 유저의 선호 강도를 반영하도록 하였습니다.
  - **Item-based Collaborative Filtering:**
    - 유저가 많이 푸는 문제의 유형과 난이도를 찾고, 그와 유사한 문제를 추천해주는 방식입니다. 구현이 간단하고 직관적이기에 해당 모델을 도입하였습니다.
    - 유저별 푼 문제들에 대해 주요 태그의 비율과 문제들의 평균 난이도를 구하여, 유저들이 많이 푸는 문제의 유형과 난이도를 찾습니다.
    - 각 유저별 라이벌들이 풀고 내가 안 푼 문제들의 태그와 난이도에 대해 정리하여, 코사인 유사도를 통해 위와 비슷한 상위 30문제를 선별합니다.
- **지표 설명**

✔️ **라이벌 기반 문제 추천 지표**

$$sum\left(\frac{\frac{(\text{내가 푼 문제의 난이도 합})}{(\text{내가 푼 문제 수})} - \frac{(\text{추천된 문제의 난이도 합})}{(\text{추천된 문제 수})}}{(\text{전체 유저 수})}\right)$$

  - 내가 푼 문제의 난이도 합 = 내가 푼 문제 중 상위 레벨 100 문제에 해당하는 난이도 합

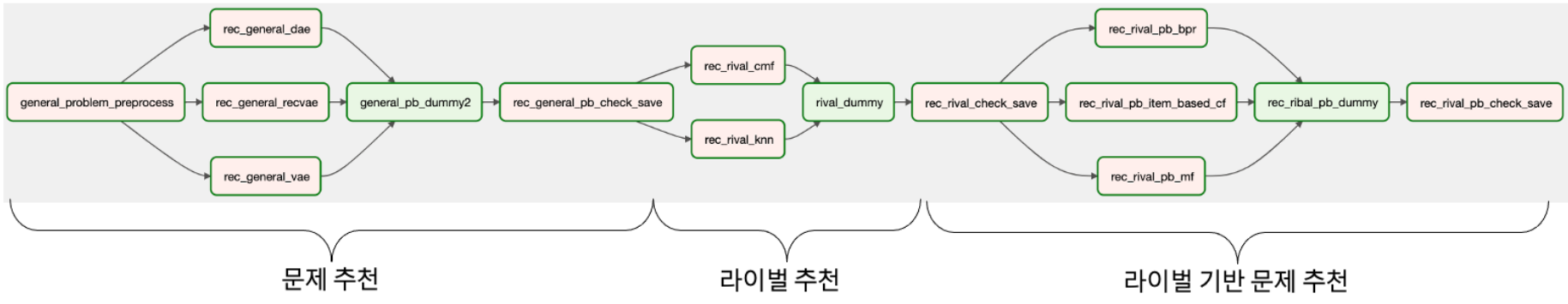
- **상위 레벨 100문제 선정 이유**
  - 라이벌보다 빠른 성장을 위해선 어떤 문제를 풀어야하는지 알려주기 위한 목적으로 추가된 기능이기 때문에, **라이벌이 푼 문제 중 높은 레벨의 문제를 집중적으로 추천한 모델을 기준 모델로 사용하기 위해** 선정
  - solved.ac의 rating 산출 기준 (난이도 순 상위 100개 문제로 점수 산출) 참고
- 추천된 문제와 타겟 유저가 푼 난이도 차이가 적을수록 추천이 잘된 모델이라고 판단하였습니다.

모델명	라이벌 기반 문제 추천 지표
ALS MF	**1.258**
BPR	3.036
Item-based CF	1.419

❗ 데이터의 업데이트에 따라 각 모델의 성능이 변경될 수 있습니다. (2022.06.06 기준)

Airflow를 통한 모델 선정 및 배치서빙

주기적으로 스크랩핑하는 데이터가 업데이트됨에 따라 각 테스크의 best 모델은 변할 수 있습니다. 복잡한 워크플로우 관리 및 각 테스크의 best 모델 선정 자동화를 위해 Airflow를 도입하였습니다. 아래 사진은 문제 추천, 라이벌 추천, 라이벌 기반 문제 추천에 대한 워크플로우 그래프입니다.



데이터베이스에서 업데이트된 데이터를 로드하여 위에서 설명한 각 테스크별 모델에 적용합니다. 각 테스크에서 성능이 가장 좋은 모델의 예측 결과를 데이터베이스에 저장하여 배치 서빙을 진행합니다.

5. 자체 평가 의견

- 잘한 부분
  - Google Analytics와 Google Tag Manage를 통해 이용자의 웹사이트 트래픽을 추적하여 비즈니스 관점의 발전을 시도한 점
  - 사용자 경험을 고려한 웹 사이트를 실제로 배포하여 다양한 사용자로부터 좋은 피드백을 받은 것
  - 문제 추천 뿐만이 아니라 라이벌 추천과 라이벌 기반 문제 추천으로 서비스를 확대한 것

- 아쉬운 부분
  - 데이터 수집의 한계로 인해 예상보다 더 길어진 데이터 업데이트 주기
  - 문제 추천에서 사용자에게 적합한 문제 티어에 관한 의문
    - ‘쉬운 문제가 나온다’, ‘어려운 문제가 출력된다’ 등 다양한 피드백에 관해 어떻게 대처할 것인지에 관한 고민

- 더 시도해 볼 만한 사항
  - 유저와 문제를 노드로 만들어서 GNN으로 학습시킨 노드 임베딩을 Multi-DAE와 Multi-VAE의 side information으로 사용
  - 22년 6월 15일자로 회수되는 AI stages 서버의 Airflow와 AI 모델 모듈을 GCP(Google Cloud Platform)으로 이전
  - 현재 활동하고 있는 유저를 라이벌로 추천해줄 수 있도록 구현
    - 최근 90일 동안 문제 풀이 제출 이력을 feature로 만들어서 라이벌 추천 모델에 사용
    - 티어와 문제 풀이 이력은 비슷하지만, 최근 문제 풀이 이력이 없는 유저를 추천하게 되는 현상 방지
  - 원활한 추천이 되지 않는 유저에 대한 알림 표시
  - 유저, 문제, 그리고 추천 데이터가 언제 업데이트 되었는지에 관한 안내
    - 푼 문제가 추천된다는 피드백 해결
  - 서비스 측정 지표 고도화

6. 개인 회고

김은선님 개인 회고

1. **이번 프로젝트에서의 나의 목표**

- EDA를 통해 모델링에 활용할 수 있는 인사이트 발굴, 라이벌 추천과 라이벌 기반 문제 추천 모델링을 통해 백준 사이트 활성화와 유저들의 실력향상에 기여하고자하였습니다.
- 더불어 에러가 발생한다면, 원인 파악과 해결방법에 대해 문서화 작업을 하고자하였습니다.

2. **목표를 달성하기 위해 시도한 점과 그 결과**

[EDA]

- 87%의 유저가 라이벌 기능을 사용하지 않는 것을 바탕으로, 라이벌 추천 서비스를 제공하면 라이벌 기능 활성화과 더불어 실력향상에도 도움이 될 것이라고 팀원들과 논의를 하였습니다.
- 더 설득력 있는 인사이트를 도출하기 위해, 라이벌이 있는 유저들은 라이벌이 없는 유저들에 비해 문제를 더 많이 푼다는 가설을 세우고 검증하였습니다. 라이벌이 있는 유저들이 없는 유저들에 비해 2배 이상 문제를 더 많이 풀고, 라이벌의 유무는 푼 문제 수와 약한 상관관계가 있음을 확인하였습니다. 이를 통해 라이벌을 등록할 수 있도록 적절한 라이벌을 추천해주면, 유저들이 문제를 더 많이 풀어 개인 실력 향상과 백준 사이트 활성화에 도움이 되리라 판단하였습니다.

- 또한 라이벌 수와 푼 문제 수간의 상관관계를 피어슨 상관계수 검정을 통해 파악한 결과, 0.385의 상관관계와 0에 아주 가까운 p-value 값을 통해 라이벌이 많을 수록 문제 푸는 횟수와 약한 상관관계를 보임을 확인하고, 적절히 큰 라이벌 수를 정하면 문제 푸는 횟수가 늘어날 것으로 판단하였습니다.

[Modeling]

- 최종 모델로 반영된 모델로는
  - 라이벌 추천: Collective Matrix Factorization
  - 라이벌 기반 문제 추천: Item based Collaborative Filtering, Bayesian Personalized Ranking이 있습니다.
- 라이벌 추천에서 팀원이 만든 기존 모델인 Matrix Factorization에서 side information을 활용한다면 성능이 더 올라갈 것으로 예상하고 관련 논문 및 자료들을 찾아보았습니다. 그 결과 Collective MF를 발견하였고, 유저와 아이템에 대한 side information을 추가하여 반영하였더니 MF보다 지표상 약 0.05~0.1 더 높은 성능을 보임을 확인하였습니다.
- 이번 프로젝트의 모델링에서 효율적으로 코드를 작성하고자 노력하였습니다. 예를 들어, Item based CF에서 필터링할 때와 코사인 유사도를 구할 때, 각 유저별 테스크를 수행하는 방식으로 약 6만5천명에 대해 작업을 하였더니 2시간 이상 소요되었습니다. 하지만 팀원분의 효율적인 코드를 참고하여, 유저별 각 테스크를 수행하는 것이 아닌 모든 유저에 대한 하나의 행렬 관점으로 보아 테스크를 수행하였더니 5분 내외로 시간을 단축할 수 있었습니다.

[Airflow를 통한 테스크 자동화]

- 모델의 실시간 예측 시간이 오래 걸려서 배치 서빙을 진행하고자 하였습니다. 또한 주기별로 스크랩핑되는 데이터의 업데이트로 인해 테스크별 성능이 제일 좋은 모델이 변할 수도 있다고 판단하여 airflow를 통한 테스크 자동화를 구축하였습니다.
- 워크플로우를 작업하던 중, 주어진 시간내에 빠르게 작업을 하기 위해서는 병렬 처리가 필요하였습니다. Airflow에서의 병렬처리는 LocalExecutor를 사용해야했는데, Airflow의 기본 DB는 sqlite로, SequentialExecutor만 사용 가능하였습니다. 때문에 Mysql로 기본 DB를 설정하여 LocalExecutor를 사용하도록 바꾸는 과정이 작업하였습니다. 하지만 생각보다 해당 작업에서 설치 및 데이터 연결의 오류를 많이 겪었다. 팀원과 함께 블로그를 많이 참고하고, 며칠 동안의 여러 시도 끝에 해결할 수 있었습니다. 같은 실수를 반복하지 않도록, 에러가 난 이유와 해결 방법을 따로 문서화해서 정리하였습니다.
- 라이벌 기반 문제 추천 모델링 중 크기가 맞지 않은 에러를 발견했습니다. 해당 원인을 파악하기 위해 트랙깅한 결과, Primary Key의 칼럼이 다른 행의 데이터와 매칭이 안될 경우, 새로운 유저에 대한 정보가 업데이트되지 않았음을 확인하였습니다. 때문에 테이블의 Primary Key는 모델의 예측 결과 순서가 바뀌어도 변하지 않는 칼럼으로 설정하였고, 초기 스키마 설계의 중요성을 깨닫게 되었습니다.

3. **한계 및 아쉬운 점**

- Airflow의 크고 작은 오류들을 해결하느라 모델링에 시간을 더 투자하지 못한 점이 아쉬웠습니다.
  - 라이벌 추천 부분에서 lightgcn의 유저 및 태그 임베딩의 결과를 내적하고 코사인 유사도를 반영하여 비슷한 유저들을 찾으려는 시도를 하였지만, 추천 결과가 다른 모델들에 비해 좋지 않았음을 확인하였습니다. 이를 보완하고 싶었지만 반영하지 못하였고, 강화 학습을 이용한 추천시스템과 같이 다양한 모델의 실험을 하지 못한 것이 아쉬웠습니다.

박정규님 개인 회고

1. **이번 프로젝트에서의 목표**

우리 팀은 공개적인 데이터를 사용하여 사용자의 수요를 이끌 수 있는 하나의 서비스까지 완성하는 것이 목표였습니다. 따라서 적절한 공개 데이터를 통해 사용자가 관심을 가질 만한 주제를 잡고, 사용자가 서비스를 사용할 수 있도록 사이트까지 완성하는 것이 목표였습니다.

2. **목표를 달성하기 위해 시도한 점과 그 결과**

- **EDA : 데이터 속에서 발견할 수 있는 예외적인 상황 분석**

  지니고 있는 도메인 지식과 데이터를 받아오는 solved_ac 서비스의 이해를 바탕으로 받아온 데이터 중 예외적인 상황에 대해 이해하려 하고, 또 처리 방법에 대해 고민해보았습니다.

- **front-end 개발**

  사용자가 서비스를 이용할 수 있도록, 하나의 사이트를 만들어 제공하기로 한 것이 이번 프로젝트의 목표였습니다.

  따라서, React를 사용하여 사이트의 전체적인 설계와 사용자가 원할히 서비스를 사용할 수 있도록 사이트 내에서의 데이터 흐름 처리, 사이트 내에서 작동하는 기능들(검색어 자동 완성, 페이지 분리, 유저 검색 및 검색한 유저의 추천 결과들의 데이터 전역 처리, 사용자의 편의를 고안한 스포일러 버튼 상태의 전역 처리, 유저 피드백 수용 등)의 구현을 맡았습니다.

정말 아쉽고, 팀원에게 정말 미안하게도 EDA에서 예외적인 상황에 대한 예외 처리에 대해서는 마무리 짓지 못하였고, front-end 개발에 있어서는 기능적으로 유저가 사이트를 돌아다니며 사용할 기능들의 상태를 전역적으로 처리하여 그 편의를 개선하는 것에는 성공하였습니.

3. **최종 프로젝트를 진행하며 아쉬운 점과 느낀 점.**

- **맡은 역할에만 너무 집중한 것이 아니었나에 대한 미안함.**

  front-end 개발을 맡다보니 다른 팀원이 맡은 역할에 조금 다른 결의 일을 담당하게 되었어서 다른 팀원들의 진척 사항과 어려움을 함께 보는 것에 대해 조금 소홀했던 것 같다.

  돌이켜보면 지금의 기능들을 좀 더 빠르게 구현할 수 있지 않았을까에 대한 아쉬움과, 그랬다면 다른 팀원들의 일을 함께 부담할 수 있지는 않았을까에 대한 아쉬움이 남는다.

- **역할 분담의 중요성**

  정말 누구 하나 빠졌다면 이런 결과물이 나올 수 있었을까에 대한 생각이 깊게 남았다. 정말 다들 맡은 바 너무나 완벽하게 해주었어서, 짧은 기한 내에 이리 멋진 결과물을 낼 수 있었던 것 같다.

  5명이 모두 각자 맡은 바에 최선을 다하고, 또 도울 일과 상의할 일이 생기면 마다하지 않는 모습을 보며 팀 프로젝트라는 것에 대해 배울 수 있었던 것 같다.

이서희님 개인 회고

1. 이번 프로젝트에서의 목표:

서비스 출시(?)를 위해 필요한 기획 및 개발, 서빙, 추후 분석까지 모든 프로세스를 팀원들과 함께 경험하는 것을 메인 목표로 잡았습니다. 또한 모델링 뿐 만 아니라 모델링 이후의 작업들(지표 측정)을 경험하는 것을 부가적인 목표로 잡았습니다.

2. 시도한 점

A. EDA

- 크롤링한 데이터들의 결측치, 분포, 데이터 타입 등을 분석하고 그에 맞는 전처리 코드를 작성하여 활용했습니다.

B. Modeling

- **라이벌 추천 모델**
  - **Rule based + cosine:** 써니가 생각한 룰을 기반으로 만들어진 모델에서 나온 후보 라이벌을 코사인 유사도를 활용해 최종 5명의 라이벌로 추출하도록 했습니다. 생각한 라이벌 기준으로 엄격하게 필터링할 수 있다는 장점이 있지만 새로 들어오는 데이터의 특성을 반영하거나 학습할 수 없다는 단점이 있어 최종모델로 선정하지 않았습니다.
  - **K-means + similarity**: kmeans로 유저를 클러스터링한 후 클러스터링 내의 유저들 중 코사인유사도 및 knn으로 최종 5명을 선정하는 방식으로 실험 수행했습니다. 그러나 지표 성능이 좋지 않았을 뿐 만 아니라 클러스터링 역시 다른 티어와 클래스의 유저가 많이 섞여있는 양상이 보였기에 최종모델에서 제외했습니다.
  - **DBSCAN + similarity:** 마찬가지로 유저를 클러스터링 한 후 코사인 및 knn으로 최종 5명을 선정했습니다. kmeans보다는 정량적 지표가 좋았으나, 클러스터가 잘 못 잡힌 유저의 경우 완전히 특성이 다른 유저가 추천된다는 불안정성이 있어 최종모델에서 제외했습니다.
  - **Matrix Factorization + similarirty**: 문제풀이이력 혹은 문제 유형 풀이 이력과 유저의 interaction data를 통해 학습한 유저 임베딩으로 타겟 유저와 유사한 유저들을 추천하도록 했습니다. 문제 풀이 이력은 이전의 알고리즘 보다 잘 반영하나. 그 외의 유저 정보들은 반영하지 못해 추천 결과가 불안정해 제외했습니다.
  - **K-Nearest Neighbors:** 유저 feature를 하나씩 추가하고 제외하며 최적의 feature 조합을 찾았고, 이에 더해 유저의 문제풀이 이력을 feature로 활용해 모델을 설계했습니다. 고차원의 feature를 직접적으로 반영한만큼 정성적으로도 정량적으로도 가장 성능이 좋았기에 최종모델로 채택했습니다.
- **라이벌 기반 문제 추천 모델**
  - **Matrix Factorization:** 문제 풀이 이력을 기반으로 학습한 아이템임베딩을 활용해 유저가 푼 문제와 가장 유사한 라이벌 문제들을 추천하도록 수행했습니다.

C. Evaluation

- **라이벌 추천 모델 지표**
  - 추천 성능을 평가할 라벨이 없는 상황이었기 때문에 추후 모델 선정 시 추천의 결과를 정량적으로 평가할 지표를 개발할 필요가 있었습니다.
  - 이를 위해 현재 solved.ac에서 사용중인 평가산식을 고려해 지표를 설정했고 최종 모델 선정기준으로 활용했습니다.
- **라이벌 기반 문제 추천 모델 지표**
  - 마찬가지로 라이벌이 푼 문제 중 내가 풀만한 문제를 추천해줘야하는 상황에서 추천 성능을 평가할 지표가 필요했습니다.
  - 라이벌 기반 문제 추천 기능의 목적에 맞게 모델을 평가할 수 있는 지표를 설정해 모델 선정기준으로 활용했습니다.

D. etc

- **모델 후시 분석:**
  - 라이벌 추천 모델: 정성적 평가를 위해 streamlit을 활용해 각 유저에게 추천된 라이벌 정보와 유저정보를 한번에 봄으로써 신속한 평가를 수행할 수 있도록 했습니다.
  - 라이벌 기반 문제 추천 모델: 정성적 평가를 위해 추천된 문제들의 난이도와 유저 정보를 볼 수 있도록 구조화하여 결과를 확인했습니다.
- **서비스 성과 분석:**
  - 서비스 배포 후 기능을 사용자들이 잘 활용하고 있는지 평가하기 위해 GA와 tag manager를 활용해 성과를 측정할 수 있도록 했습니다.
  - 주요 성과 지표는 라이벌 추천, 라이벌 문제 추천, 일반 문제 추천 버튼 클릭 전환율로 잡아 측정했습니다.

3. 회고

- 모델링 프로젝트를 한 적은 있지만 이 모델을 실제로 서비스에 활용한 프로젝트는 처음이라서 기분이 남달랐습니다. 특히 사람들에게 추천결과가 보여질 것이라고 생각하니 추천결과의 품질에 대해서 더 생각할 수 있었던 것 같습니다. 7월 30일까지는 올라오는 피드백을 보고 모델링 고도화를 계속 진행해보면 더 의미있는 프로젝트가 될것같다는 생각이 듭니다.
- 모델링도 중요하지만 추천 결과를 어떻게 평가해야할지 그리고 서비스가 잘 돌아가고 있는지는 어떻게 알 수 있을지에 대해서도 생각하고 생각한 점들을 프로젝트에 적용해 볼 수 있어 좋았습니다.
- 프로젝트를 수행하며 실제로 스타트업은 이렇게 일하지 않을까라는 생각이 많이 들었습니다. 특히 이렇게 제대로 된 협업을 경험할 수 있도록 열심히 참여하신 팀원분들께도 감사했습니다.

이선호님 개인 회고

학습 목표

개인 학습 목표

AI 모델 기반의 웹 서비스를 구축하기 위한 기본적인 Back-end 설계와 API 서버 개발을 학습합니다. 또한 실제 서비스를 배포할 때 발생하는 문제를 예방하기 위한 사례를 익히고 개선 방안을 적용해봅니다.

공동 학습 목표

BOJ 유저별 실력과 문제 풀이 이력에 기반한 새로운 문제 추천 모델과 solved.ac 라이벌을 추천해주는 모델을 개발하고 고도화합니다. 또한 이를 웹 서비스로 배포하여 전반적인 워크플로우를 익히고 실제 Product Serving 과정을 경험합니다.

학습 목표를 위한 노력과 시도

개인 학습 측면

Back-end 구축을 위한 노력

비동기에 최적화되고 Client에서 오는 많은 요청과 전달해야 할 응답을 처리할 수 있도록 API 서버를 Node.js로 개발했습니다. 또한 데이터베이스를 모델 학습과 예측 결과를 수정하거나 입력할 때 또는 피드백을 작성할 때 사용하는 쓰기 용도의 Master DB, API 서버에서 GET 요청을 처리하는 오직 읽기 용도의 복제 DB 두 가지로 나누어 관리했습니다. 이는 실제로 Airflow에서 모델을 학습하고 예측 결과를 한꺼번에 업로드할 때 waiting table for metadata lock 현상이 종종 발생할 경우 웹 서비스까지 영향을 받아 클라이언트에서 데이터를 받아오지 못하는 현상을 해결하고자 한 시도입니다.

GitHub Action과 AWS CodeDeploy를 통한 CI & CD 자동화

서버의 코드를 그대로 SSH 연결하여 관리하는 것보다는 Git을 통한 협업과 소프트웨어 구성 관리를 준수하고자 했습니다. 그래서 수정사항이 안정화되면 main branch에 pull request를 날리고 merge하여 GitHub Action에 의해 S3 bucket으로 자동으로 빌드되어 배포되는 과정을 구축했습니다. 이후 S3 bucket에 저장된 코드를 AWS CodeDeploy를 통해 EC2로 자동 배포되도록 했습니다. 그러나 배포 과정에서 docker 컨테이너가 다시 빌드되면서 클라이언트와 웹 서버와의 연결이 끊기는 현상이 발생하는 현상을 목격하고, NGINX 프록시 서버와 번갈아 사용하는 두 개의 docker 컨테이너를 통해 빌드 과정에서 서버가 중단되는 일이 없게끔 조치했습니다.

공동 학습 측면

Autoencoder 계열 모델 실험

데이터 수집의 한계로 인해 유저의 문제 풀이 순차 이력을 모델링에 사용하지 못해 non-sequential 모델 중에 autoencoder 계열의 모델에 주목했습니다. 이는 이전에 있었던 MovieRec 대회에서 가장 좋은 성능을 보였고, 실제로 진완혁님이 진행한 실험 결과에서 autoencoder 계열의 모델이 다른 모델(Catboost, DeepFM)보다 정량적•정성적 평가 모두 좋게 나왔던 근거가 뒷받침되었습니다. 그래서 이전 대회에서 가장 좋은 성능을 보였던 EASE 모델을 실험했지만, 진완혁님이 사용한 RecVAE의 모델 성능이 가장 좋게 나와서 이를 주 모델로 택하기로 결정했습니다. 이 과정에서 사용한 데이터 전처리 코드를 미리 팀원에게 공유했습니다.

Front-end 레이아웃 디자인과 기능 추가

BOJ와 solved.ac를 모두 회원제로 운영되고 있는 상황에서 문제와 라이벌을 추천받을 수 있는 서비스까지 회원제로 운영되는 건 사용자에게 피로감을 유발할 수 있다는 생각이 들었습니다. 그래서 비회원제로 운영하면서 사용자가 적절한 문제를 손쉽게 추천받을 수 있는 방법을 고민했고, 그에 관한 해답으로 간단하게 유저 핸들을 검색하면 Netflix 스타일의 carousel 슬라이드 디자인으로 여러 section으로 나눠 해당 유저의 문제 또는 라이벌을 추천하는 서비스를 고안했습니다. 이에 관해 박정규님과 오프라인으로 만나서 논의하며 최종 레이아웃의 개요를 잡았습니다.

BOJ에는 사용자의 문제 풀이 취향에 따라 미리 티어와 알고리즘 태그를 보이게 할 수 있는 옵션이 있습니다. 그래서 이를 벤치마킹하여 이번 웹사이트에서도 스포일러 스위치를 추가하여 사용자가 문제에 관한 데이터를 간단히 훑어볼 수 있도록 작업했습니다.

한계와 개선방향

실제 서비스를 배포하여 다양한 사용자로부터 피드백을 받았을 때, 대체로 사용자에게 친화적인 서비스를 제공한다는 점에서 긍정적인 평가를 받은 것 같습니다. 그러나 데이터 수집에 대한 제약으로 인해 데이터 자동 수집 주기가 길어져서 실시간으로 유저가 푼 문제 이력이 학습 과정에서 추천 결과에 반영되지 못하고, Inference 시 직전에 푼 문제가 출력 결과에서 제외되지 못하는 점이 아쉽습니다. 또한 MLflow를 사용해서 각 모델 학습 결과에 관해 템플릿으로 관리하지 못하는 점, 그리고 Autoencoder 기반의 모델 뿐만이 아니라 좀 더 다양한 모델에 관해 실제 서비스에 적용하지 못한 점이 아쉬운데, 앞으로 약 한 달동안 시범 서비스를 운영하면서 사용자의 피드백을 반영하여 부족한 부분을 개선해 나가고자 합니다.

진완혁님 개인 회고

데이터 수집, EDA

- 사용자가 풀었던 전체 문제를 고려한 Non-Sequential 모델과 최근에 풀었던 문제를 고려한 Sequential 모델을 구성하려했다.
- 최근에 풀었던 60개 문제를 활용하려 했으나 그 부분에 대해 BOJ에서는 수집이 원칙적으로 불가능하여 solved.ac API를 활용한 Sequential 모델에 집중하게 됐다.
- 데이터 분포와 이상값들을 확인하고 사용가능한 데이터로 정제를 진행했다. 평균 시도 횟수가 너무 많거나, 태그가 없는 것 또는 채점이 불가능한 문제는 제외했다.
- 문제의 정보(레벨, 태그, 시도횟수 등)을 활용하여 유사한 문제를 구해봤다. 코사인 유사도와 유클리드 거리를 이용해봤는데, 정성적으로 확인해봤을 때 유클리드 거리를 이용하는 것이 좀 더 유사한 문제가 나오는 것을 확인했다. 후에 우리 프로젝트에 적용해볼 만한 사항으로 생각된다.
- Negative Sampling을 위해 여러가지 방법을 고민하고 시도했다.

모델 구성 과정

나는 문제 추천 모델링에 집중했다. 양질의 결과를 내기 위해 여러 모델들을 실험해봤다.

- DeepFM
  - DeepCTR 라이브러리의 DeepFM을 활용했다. 범주형과 수치형 데이터의 구분이 중요하기 때문에 추가적인 데이터 변형을 진행했다.
  - 정성적인 성능을 확인해보니 다소 본인의 티어에 맞지 않게 과도하게 높거나 낮은 티어의 문제가 많이 나왔다.
  - 불안정한 성능으로 인해 활용 모델로 채택되지 못 했다.

- GMF
  - Matrix Factorization Model의 상호작용을 일반화하여 모델의 성능을 향상시킨 GMF를 적용해봤다.
  - MF에 비교하여 내적을 일반화하고 활성함수를 추가했는데, DeepFM보다는 정성적으로 훨씬 나은 성능을 보였지만, 일부 유저에 비해 굉장히 불안정한 추론 결과를 보였다.
  - 활용 모델로 채택되지 못 했다.

- CatBoost
  - DKT 대회에서 좋은 성능을 냈던 머신러닝 알고리즘인 CatBoost를 적용해보았다.
  - CatBoost는 Categorical한 데이터에서 더욱 뛰어난 성능을 내기 때문에 우리의 데이터를 범주화하여 학습과 추론을 진행했다.
  - GMF와 유사하게 나쁘지 않은 성능을 냈지만 일부 유저에 대해 굉장히 불안정해 채택되지 못 했다.

- Multi-VAE, Multi-DAE
  - 수집한 데이터의 가장 큰 문제점은 Negative Sample이 없다는 것이었다.
  - 이 문제로 인해 인코더와 디코더를 통해 데이터를 재구성하는 Auto Encoder 모델을 적용했다.
  - 성능을 향상시키기 위해 Side Information을 적용시키기 위해 문제의 tag 정보를 학습에 활용했다.
  - 정량적, 정성적으로도 뛰어난 성능을 내어 최종적으로 사용될 모델로 채택되었다.

- RecVAE
  - Multi-VAE의 성능을 향상시키기 위해 새로운 인코더 아키텍처를 구성한 RecVAE도 활용했다.
  - Multi-VAE와 인풋과 아웃풋 형태가 완전히 동일하기 때문에 원활하게 바로 적용할 수 있었다.
  - 다른 AE모델보다 뛰어난 성능을 내어 최종적으로 사용될 모델로 채택되었다.

회고

실제 서비스를 운영할 수 있도록 프로젝트를 진행하다보니 다른 팀원들이 맡은 역할에 대해 굉장히 멋있기도 하고 신기하기도 했다.

직접 데이터를 수집하고, EDA, 전처리 후 생성한 모델이 실제 서비스에 적용되었다고 생각하니 굉장히 뿌듯하기도 하고 의미가 있는 것 같다.

Airflow를 통해 태스크를 자동화하는 파트도 공부를 진행했는데, 이번에는 다른 능력 좋은 팀원이 맡아주셨고 후에 꼭 공부한 내용을 적용해볼 수 있도록 해보겠다!

현재는 AE기반의 모델들이 주요 사용모델로 채택되었지만, 추가적으로 성능 고도화를 위해 여러 논문을 읽어보고 다른 알고리즘을 적용해보면서 성능 향상을 이끌어 내봐야겠다.
사용자들로부터 피드백을 받을 수 있게 구성하였기 때문에 수렴 가능한 의견들을 적용시켜 더욱 양질의 서비스를 만들것이다.

실제 회사에서 진행해볼만한 프로세스를 경험한 것 같아 굉장히 많은 도움이 된 것 같다.