DKT Wrap-up Report - Suggestify

1. 프로젝트 개요

1. 1. 배경

- 최근 들어서 데이터 사이언스를 이용해 자신의 학습을 상태를 파악하는 서비스가 배포되고 있다. Riiid의 산타토익과 같이, 수험자의 문제 풀이 데이터를 바탕으로 강점과 약점을 파악해 이를 보완할 다음 문제를 제시하는 방식이다. 시험을 통해서는 우리 개개인에 맞춤화된 피드백을 받기가 어렵고 따라서 무엇을 해야 성적을 올릴 수 있을지 판단하기 어렵다. 이럴 때 사용할 수 있는 것이 "지식 상태"를 추적하는 딥러닝 방법론인 DKT이다.
- DKT를 활용하면 우리는 학생 개개인에게 수학의 이해도와 취약한 부분을 극복하기 위해 어떤 문제들을 풀면 좋을지 추천이 가능하다. 대회에서는 학생 개개인의 이해도를 가리키는 지식 상태를 예측하는 일보다는, 주어진 문제를 맞출지 틀릴지 예측하는 것에 집중한다.
- Task: 각 학생이 푼 문제 리스트와 정답 여부가 담긴 데이터를 받아 userID별 마지막 문제를 맞출지 틀릴지 예측
- 성능 지표와 의미
  - AUC(Area Under the ROC Curve): ROC Curve 아래의 면적으로, threshold에 따라 변화하는 fpr(x),tpr(y) 값을 이용해 그린 그래프. 모델이 양성 샘플을 더 높은 점수로 예측하고 음성 샘플을 더 낮은 점수로 예측할 때 더 높은 값을 가지기 때문에, ACC만으로는 알 수 없는 세밀한 성능을 측정 가능.
  - ACC(Accuracy): 전체 데이터 중 정답 여부를 맞춘 데이터의 비율. 0.5가 넘는 값이면 1로 분류한 것으로 보고, 그 미만이라면 0으로 분류한 것으로 판단함.

1.2. 활용장비 및 재료

- 개발환경
  - OS: Linux-5.4.0-99-generic-x86_64-with-glibc2.31
  - GPU: Tesla V100-SXM2-32GB * 1
  - CPU cores: 8
- 협업 툴: github, wandb, notion, slack

1.3. 프로젝트 구조 및 사용 데이터셋의 구조도(연관도)

- 프로젝트 구조

```
📦level2-dkt-recsys-03
 ┣ 📂EDA
 ┣ 📂data
 ┣ 📂ensemble
 ┣ 📂Feature_Engineering
 ┗ 📂code
   ┣ 📂FM
   ┃ ┣ 📂models
   ┃ ┣ 📂submit
   ┣ 📂dkt
   ┃ ┣ 📂dkt
   ┣ 📂lgbm
   ┃ ┣ 📂outputs
   ┣ 📂lightgcn
   ┃ ┣ 📂lightgcn
   ┗ 📂xgb
```

- 데이터셋의 구조도

	userID	assessmentItemID	testId	answerCode	Timestamp	KnowledgeTag
0	0	A060001001	A060000001	1	2020-03-24 00:17:11	7224
1	0	A060001002	A060000001	1	2020-03-24 00:17:14	7225
2	0	A060001003	A060000001	1	2020-03-24 00:17:22	7225
3	0	A060001004	A060000001	1	2020-03-24 00:17:29	7225
4	0	A060001005	A060000001	1	2020-03-24 00:17:36	7225
...	...	...	...	...	...	...
2266581	7441	A030071005	A030000071	0	2020-06-05 06:50:21	438
2266582	7441	A040165001	A040000165	1	2020-08-21 01:06:39	8836
2266583	7441	A040165002	A040000165	1	2020-08-21 01:06:50	8836
2266584	7441	A040165003	A040000165	1	2020-08-21 01:07:36	8836
2266585	7441	A040165004	A040000165	1	2020-08-21 01:08:49	8836

2266586 rows × 6 columns

데이터 출처 : https://next.stages.ai/competitions/268/data/training

2. 프로젝트 팀 구성 및 역할

2.1 프로젝트 팀 구성

- 팀명 : Suggestify
- 팀 인원 : 6
- 팀 구성 : 김수빈, 박시우, 백승빈, 이재권, 이진민, 장재원

2.2 역할

이름	역할
김수빈	EDA, 데이터 전처리, Bert 실험 및 튜닝, Github Setting, DKT Baseline 튜닝
박시우	EDA, 데이터 전처리, feature engineering, LGBM feature 실험, GBDT, GRU 베이스라인 구축 및 실험
백승빈	EDA, 데이터 전처리, feature engineering, LQTR 구현 및 튜닝
이재권	EDA. 데이터 전처리, feature engineering, LightGCN 실험
이진민	EDA, 데이터 전처리, feature engineering, LightGBM feature 실험, FM, FFM 구현 및 실험
장재원	EDA, 데이터 전처리, feature engineering, LightGBM 고도화

3. 프로젝트 수행 절차 및 방법

먼저 EDA, Feature Engineering은 모두가 각자 진행한 뒤 자신이 얻은 Insight를 공유하였고 그 뒤 각자 하고 싶은 것들(Modeling, Hyper-Parameter Tuning, Data Augmentation 등)을 진행하였다.

프로젝트 진행 타임라인

4. 프로젝트 수행 결과

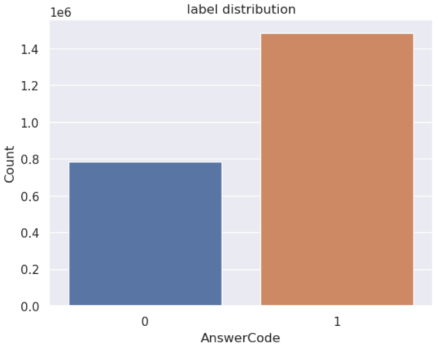
4.1. 탐색적 분석 및 전처리 (EDA & Pre-processing)

프로젝트에서는 Iscream 데이터셋을 사용하였다. 데이터셋에는 총 7,442명의 사용자가 문항을 풀었을 때의 정보가 담겨 있으며, 사용자가 푼 마지막 문항의 정답 여부(answercode)를 예측하는 binary classification 문제이다. 컬럼은 아래와 같이 총 6개의 정보가 포함되어 있다.

- `userID` 사용자의 고유번호. 총 7,442명의 고유 사용자가 있으며, train/test셋은 이 `userID`를 기준으로 90/10의 비율로 나누어져있다.
- `assessmentItemID` 문항의 고유번호. 총 9,454개의 고유 문항이 있습니다.
- `testId` 시험지의 고유번호. 총 1,537개의 고유한 시험지가 있다.
- `answerCode` 사용자가 해당 문항을 맞췄는지 여부에 대한 이진 데이터. 0은 사용자가 해당 문항을 틀린 것, 1은 사용자가 해당 문항을 맞춘 것이다.
- `Timestamp` 사용자가 해당문항을 풀기 시작한 시점의 데이터.
- `KnowledgeTag` 문항 당 하나씩 배정되는 태그, 일종의 중분류. 태그 자체의 정보는 비식별화 되어있지만, 문항을 군집화하는데 사용할 수 있다. 912개의 고유 태그가 존재한다.

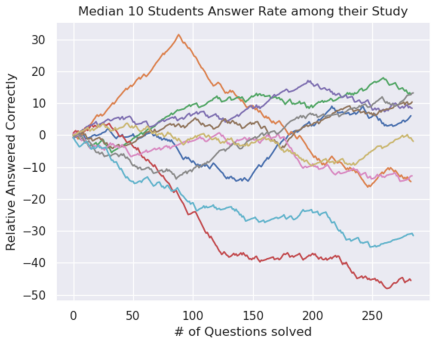
Feature 별로 EDA를 한 결과 아래와 같이 분석하였다.

- Answercode

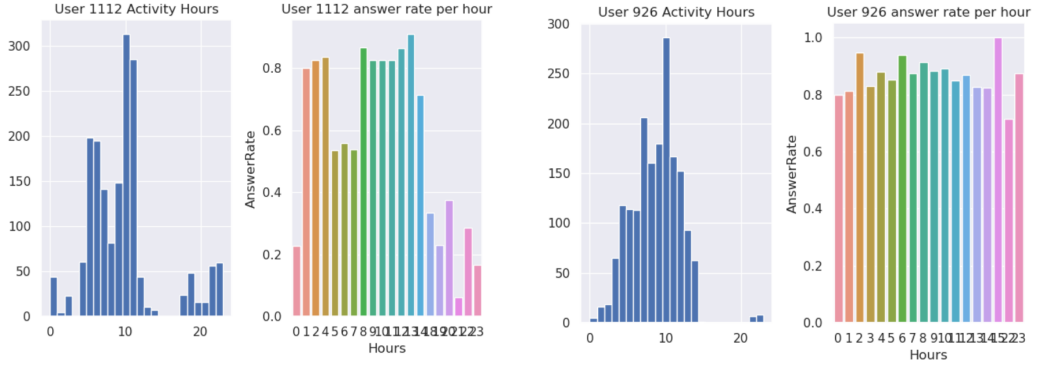


→ 정답을 맞추지 못하는 경우가 맞춘 경우의 절반으로, 학습하기 어려운 정도의 불균형은 없었다.

- User

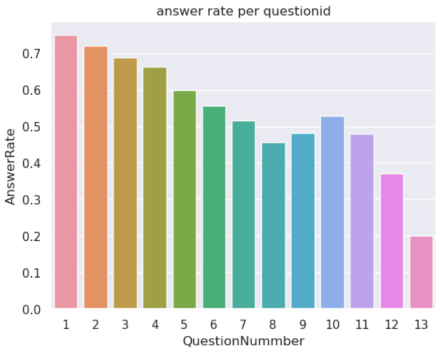


→ user별로 다른 정답패턴을 보인다.



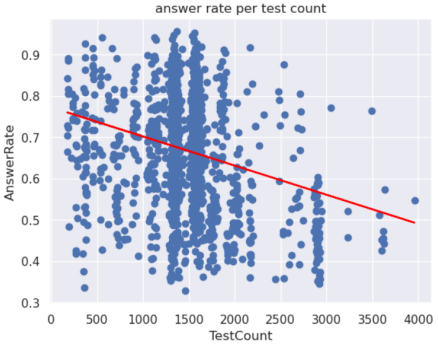
→ 유저마다 주 활동 시간대, 현 활동 시간대의 영향력 차이가 있다. 어떤 유저는 시간대가 정답률에 영향을 미치지만, 또 어떤 유저는 영향이 거의 없음을 확인할 수 있다.

- AssessmentId



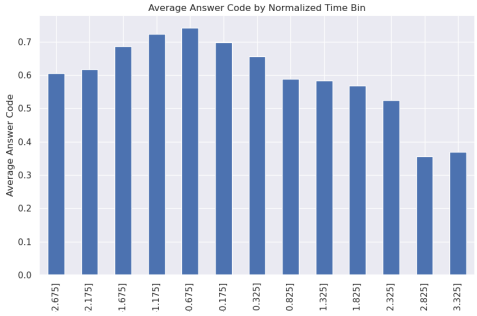
→ 문항 번호가 커질수록 정답률이 낮아지는 경향을 보인다

- Testid



→ 시험지 풀이 횟수가 많을수록 정답률이 낮아진다.

- Timestamp



→ 문제 풀이 소요 시간이 정답 여부에 많은 영향을 미친다.

- 기타
  - 같은 문제를 다시 풀었을때 보다 직전 문제를 맞췄을때 현재 문제의 정답률이 더 높다.

    맞췄던 동일한 문제를 풀었을 경우의 정답률 : 69%
    이전 문제를 맞춘 경우 다음 문제의 정답률 : 77%

4.2 Feature Engineering

4.2.1 Feature 추가

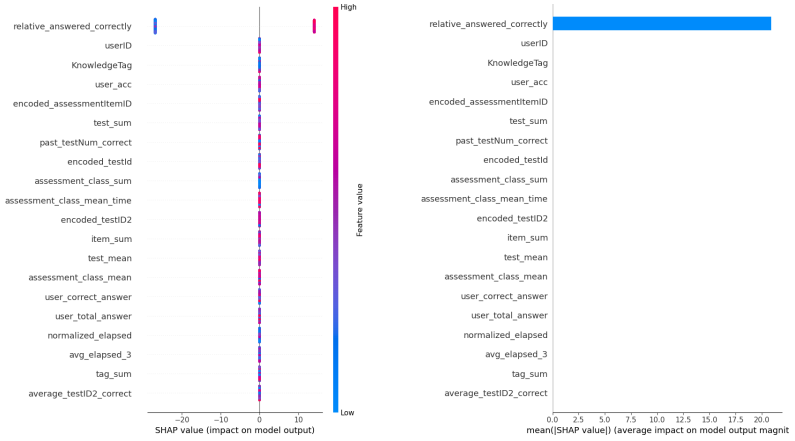
- 데이터셋 기본 변수
  - `KnowledgeTag` : 문제의 지식 태그를 나타내는 변수이다. 각 문제가 어떤 지식 영역에 속하는지를 표시하여 문제의 특성을 분류하는 데 사용된다.
  - `encoded_testId` : 시험 ID를 인코딩한 변수이다. 시험 ID를 숫자로 변환하여 데이터 분석 및 모델링에 활용할 수 있도록 한다.
  - `encoded_testID1` : assessmentItemID의 앞 3자리에서 추출된 값으로, 시험지의 대분류를 의미한다.
  - `encoded_testID2` : assessmentItemID의 중간 3자리에서 추출된 값으로, 시험지의 중분류를 의미한다.
  - `encoded_testNum` : assessmentItemID의 마지막 3자리에서 추출된 값으로, 시험지의 번호를 의미한다.
- 사용자 및 문항 누적 정보
  - `user_correct_answer, user_total_answer, user_acc` : 사용자의 문제풀이 수, 정답 수, 정답률을 누적하여 보여준다.
  - `past_count, past_correct, average_correct` : feature 별 문제풀이 수, 정답 수, 정답률을 누적하여 보여준다.
  - `past_user_count, past_user_correct, average_user_correct` : 사용자의 feature 별 문제풀이 수, 정답 수, 정답률을 누적하여 보여준다.
- 데이터셋 전체 통계
  - `test_sum, test_count, test_normalized_mean` : 사용자의 테스트 성적을 종합적으로 나타내는 지표들이다
  - `tag_sum, tag_count, tag_normalized_mean` : 사용자가 푼 문제의 태그 점수에 대한 정보를 종합적으로 나타내는 지표들이다.
  - `item_sum, item_count, item_normalized_mean` : 사용자가 푼 문제의 아이템 점수에 대한 정보를 종합적으로 나타내는 지표들이다.
- 시간 관련 정보
  - `normalized_elapsed`, `normalized_elapsed_user`, `normalized_elapsed_test`, `normalized_elapsed_user_test` : (사용자별, 문제별) 문제풀이시간을 정규화한 값이다. 이 값은 일정 기준에 따라 경과 시간을 조정하여 비교 가능한 형태로 변환한 것을 의미한다.
  - `hour` : 문제를 푼 시간을 의미한다.
  - `correct_per_hour` : 문제를 푼 시간당 정답률을 의미한다. 이는 사용자가 어떤 시간대에 문제를 잘푸는지 평가하기 위한 지표이다.
  - `hour_mode` : 사용자의 주 활동 시간이다. 이는 사용자가 주로 어떤 시간대에 학습을 진행하는지를 파악하는 데에 사용된다.
  - `month` : 월을 나타내는 단위이다. 이는 시간을 표현하거나 학습 기간을 분석하는 데에 사용된다.
  - `month_mean` : 월별로 평균 정답률을 나타내는 지표이다. 이는 월별 학습 성과를 비교하고 분석하는 데에 도움을 준다.
  - `week_num` : 주차를 나타내는 값이다. 이는 시간을 구분하거나 학습 기간 등을 분석하는 데에 사용된다.

- 이전 시점 정보
  - `recent_sum`, `recent_mean` : 사용자의 최근 일련의 답변들의 점수 합계 및 평균을 나타내는 지표이다. 이는 사용자의 최근 학습 성과를 종합적으로 파악할 수 있게 도와준다.
  - `average_assessmentItemID_correct_shift` : 각각 사용자가 이전 단계에서 푼 문제의 정확한 답변 여부를 해당 문제의 ID별로 평균화한 값이다. 이는 사용자의 이전 단계에서 푼 문제의 특정 문제 ID에 대한 정확도를 측정하는 데에 유용하다.
  - `normalized_elapsed_shift`, `normalized_elapsed_user_shift`, `normalized_elapsed_test_shift`, `normalized_elapsed_user_test_shift` : 사용자의 이전 단계에서의 문제풀이시간을 정규화한 값이다. 이는 사용자의 학습 속도나 시간 관리 능력을 평가하는 데에 도움을 준다.

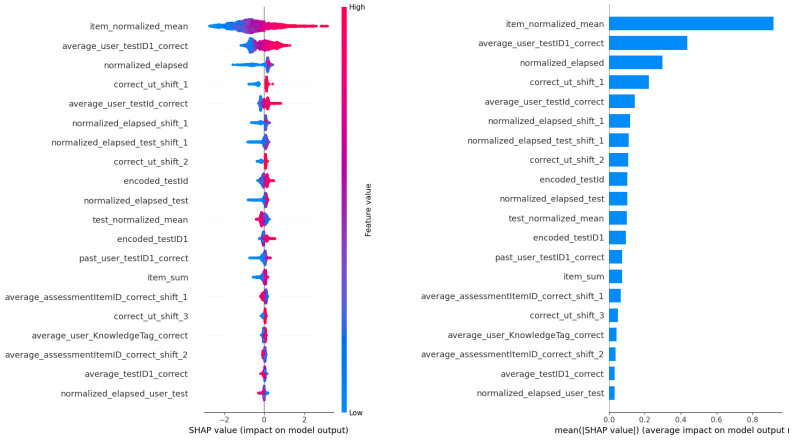
4.2.2 LGBM Feature Importance 비교 및 실험

- Feature Importance비교를 위해 Split, gain, SHAP(shapely value)를 사용하였다

1. 일부 Feature로 인해 과적합이 되는 현상
   → 미래 정보 누출(Look-ahead bias)되는 일부 Feature 제거



2. 일부 Feature의 영향력이 큰 현상 발생
   → 영향력이 큰 Feature들을 정규화시켜 다른 Feature들의 영향력 높임



3. Test data의 Feature engineering에서 test data만 사용하는 경우 데이터셋 전체 통계(Dataset Overall Statistics)로 인한 과적합 발생
   → Test data의 Feature engineering할때 train data 활용

4.3 모델 선정 및 평가

4.3.1 LightGBM

- 모델선정
  LGBM은 Leaf-wise 트리 성장 방식을 사용하여 메모리 사용량을 줄이고 속도를 향상시키는 경량화된 그래디언트 부스팅 기반의 트리 알고리즘이다. 이 알고리즘은 다른 부스팅 알고리즘에 비해 더 빠른 학습 속도와 자동적인 특성 선택 기능을 제공하여 다양한 피처들을 기반으로 좋은 성능을 보여줄 수 있기 때문에 이 모델을 선정하였다.
- 모델 실험

  모델에 가장 많은 영향을 주었던 Feature는 Dataset Overall Statistics(OS)이다. 그러나 이 Feature는 기존 TOFE(Test의 FE때 Test의 정보만 활용)에서는 과적합 현상을 보였다. 이를 해결하기 위해서 TTFE(Test의 FE때 Train과 Test의 모든 정보 활용)방법을 사용하였다. 다른 방법에 TTFE모델이 가장 좋은 성능을 보였다. 파라미터는 Random Search를 사용했으며 다음과 같이 설정하였다.
  Best parameters: {'num_leaves': 64, 'num_iterations': 200, 'min_data_in_leaf': 1, 'max_depth': 16, 'max_bin': 50, 'learning_rate': 0.05}

	Train	Valid	Test	(Public)	Test	(Private)
	ACC	ACC	AUC	ACC	AUC	ACC
TestOnlyFE	0.6200	0.5243	0.7599	0.6344	0.7890	0.6613
TestOnlyFE_RemoveOS	0.6049	0.5081	0.7836	0.7177	0.8024	0.7204
Train&TestFE	0.8638	0.8383	0.8198	0.7554	0.8406	0.7688
Train&TestFE_RemoveOS	0.8537	0.8314	0.8157	0.7527	0.8354	0.7554

4.3.2 Xgboost

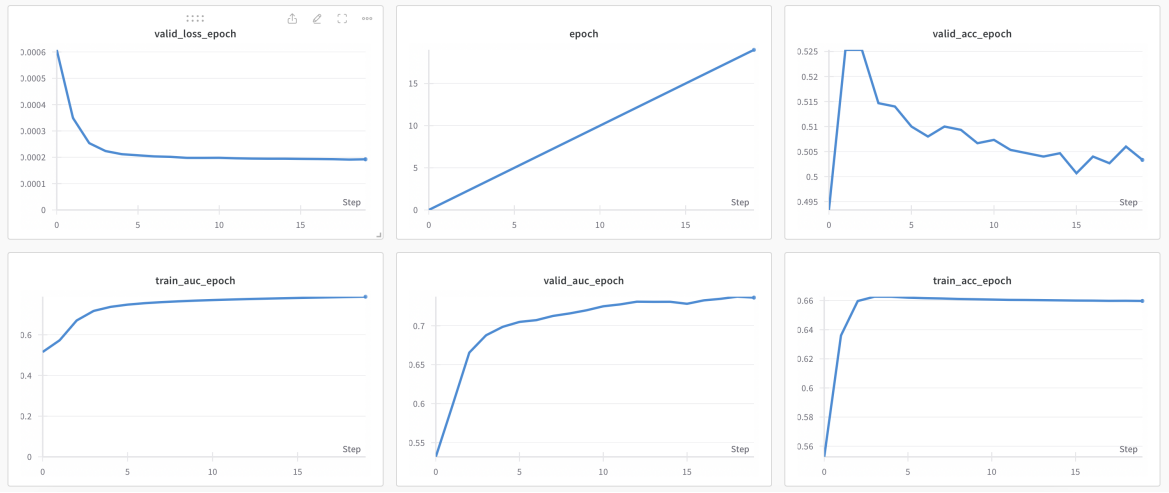
- 뛰어난 속도와 성능을 제공하며, 대용량 데이터셋에도 효과적이다.
- train, vaild auc 간의 차이가 너무 커지지 않으면서도 valid auc와 acc가 모두 향상된 경우에만 feature를 반영하는 방식으로 feature selection 과정을 거쳤다.
- 오버피팅 발생 시 valid auc만으로 성능을 알 수 없다고 생각해서 주어진 test data 파일에서 user의 마지막 문제 풀이 이전 문제 풀이에 대한 auc, acc를 확인해보면서 정상적으로 성능이 향상되고 있는지 확인했다. 최종적으로 private score를 확인해보니 이때 확인했던 값과 비슷함을 확인할 수 있었다.
- 기존 feature와 유사한 정보를 담은 feature가 추가되었을 때는 과적합이 발생하는 경향이 있다고 판단해서 최대한 겹치지 않는 정보를 가지도록 feature를 추가하거나 제거했다. 이 때 auc와 acc 간의 trade off가 발생하는 경우가 있어서, 각각 다른 feature를 학습한 모델들에 소프트 보팅을 적용해 보았더니 결과가 가장 좋았다.
- 하이퍼 파라미터는 optuna 라이브러리를 사용해 최적화하였다.
- Private 기준 AUC: 0.8543, ACC: 0.7769

4.3.3 Catboost

- 카테고리형 변수를 자동으로 처리하는 데 강점을 가진 그래디언트 부스팅 모델이다. lgbm, xgboost의 성능이 더 좋았기 때문에 최종적으로 사용하진 않았다.

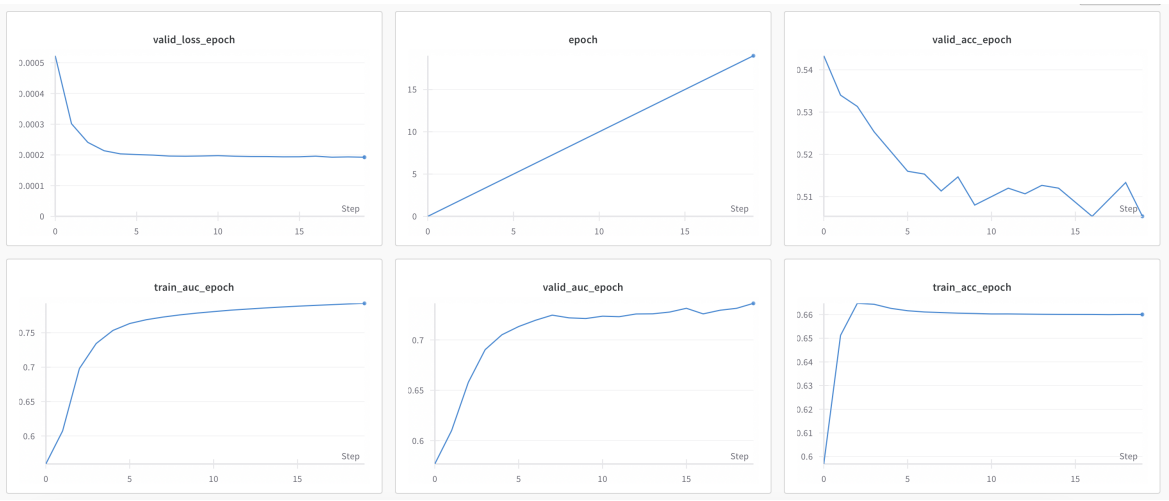
4.3.4 FM

- SVM과 Matrix Factorization의 장점을 합친 모델이며 sparse한 데이터에서 높은 성능을 가진다. 특성 간의 상호작용을 학습할 수 있지만 선형적으로 계산되기 때문에 조금은 한계점이 있을 수 있다고 생각했다. 코드의 문제인지 아직 파악은 못했지만 acc가 매우 작게 결과가 나왔었다. 하이퍼 파라미터는 learning rate를 조금 작게 설정하고 epochs를 늘렸을 때 가장 성능이 좋았다.



Train AUC	Train ACC	Valid AUC	Valid ACC
0.78727	0.65976	0.73612	0.50333

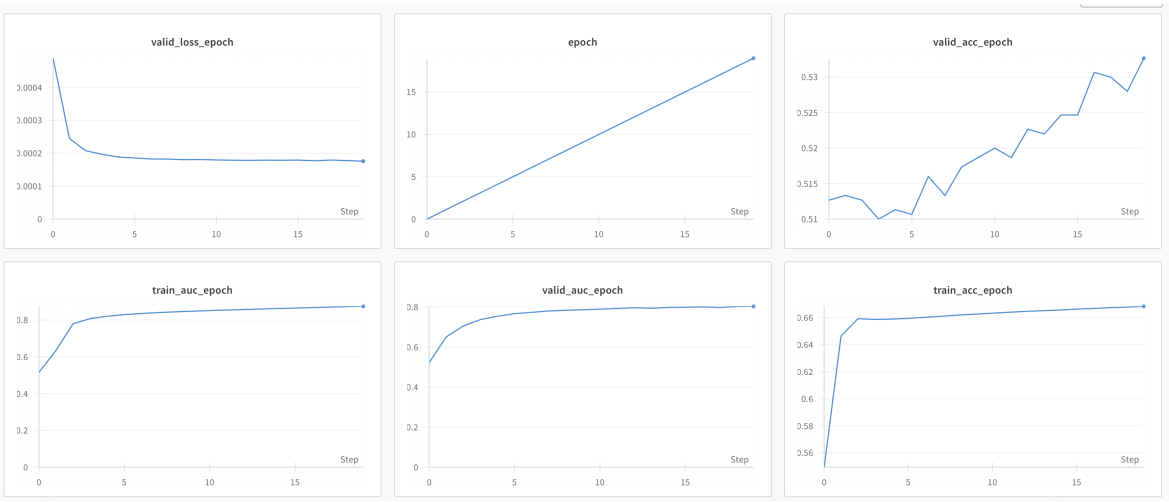
- 기본 feature들에서 user별 정답률을 binning하여 feature 추가한 결과 성능이 조금씩 올랐다.



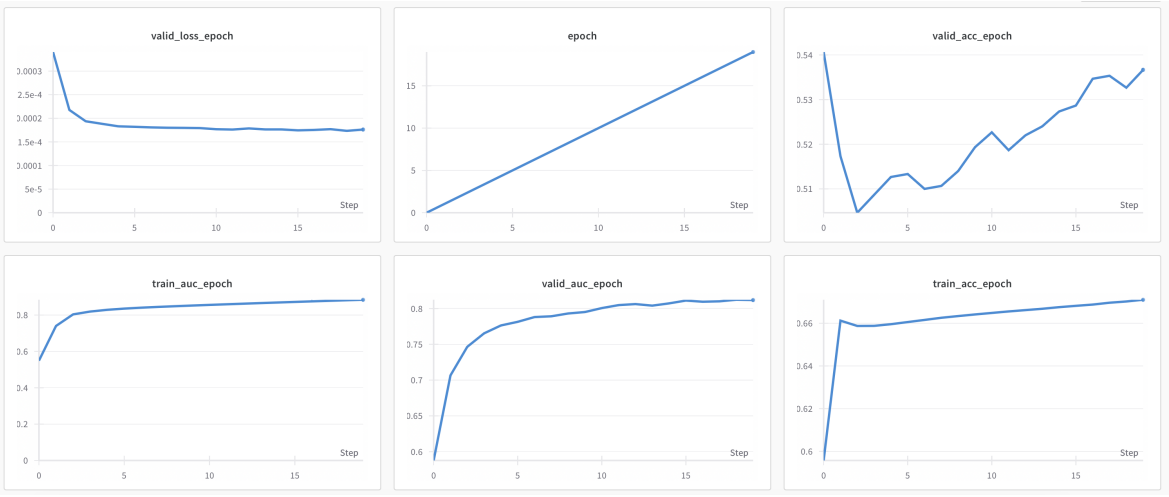
Train AUC	Train ACC	Valid AUC	Valid ACC
0.79285	0.66004	0.73635	0.50533

4.3.5 FFM

- 특성을 필드로 구분하고, 필드 간 상호작용을 잠재 벡터에 모델링 하므로, FM 보다 특성 간 상호작용을 더 섬세하게 모델링하여, 주로 카테고리 데이터에서 강점이 있을 것이라고 예상한다.
- FM과 동일한 조건으로 실험하였다.



Train AUC	Train ACC	Valid AUC	Valid ACC
0.87479	0.66847	0.80363	0.53267



Train AUC	Train ACC	Valid AUC	Valid ACC
0.88418	0.671	0.81185	0.53667

4.3.6 LSTM

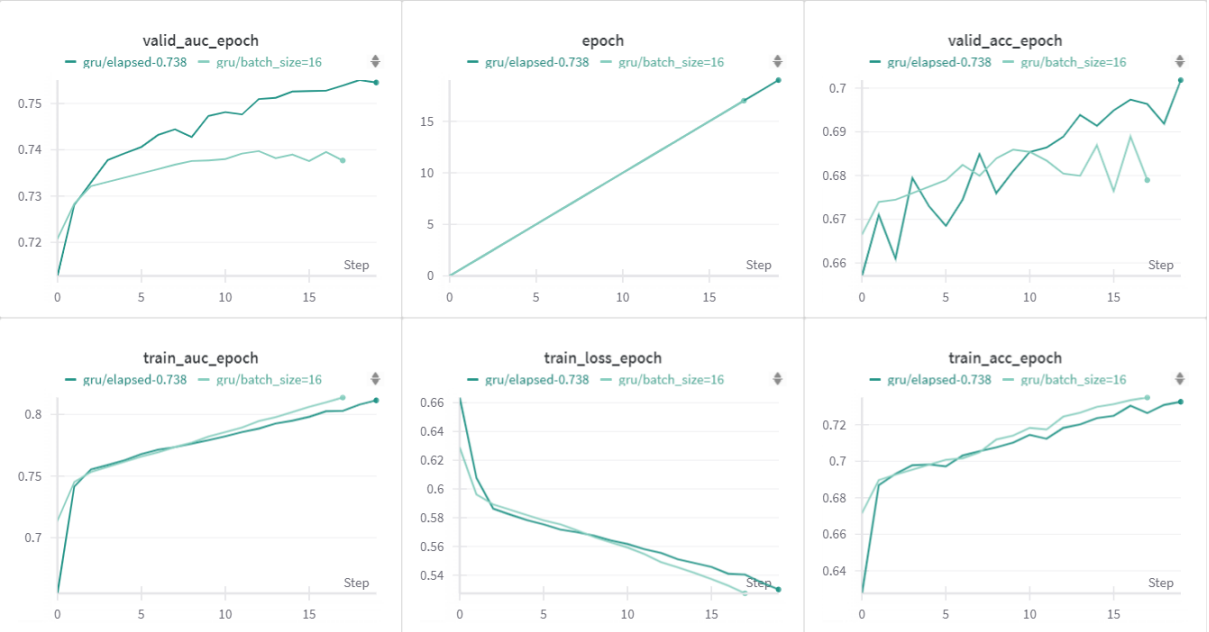
- 장단기 기간을 모두 기억해서 그런지 안정적으로 학습하는 모습을 보였다.

4.3.7 LSTMATTN

- 베이스라인 내 sequential 모델 중 가장 빠르게 학습 성능이 수렴하는 모습을 보였다.

4.3.8 GRU

- GRU는 LSTM에 비해 간단한 구조로 연산량도 적고 학습 속도도 더 빠르다. LGBM에서 feature selection을 할 때 최근의 학생의 정답 여부가 그 이후 정답 여부에 영향을 많이 미친다는 것을 알게되었고, 그에 따라 오히려 장기 기억이 크게 필요가 없다는 생각이 들어 더 간단한 모델을 시도하는 것이 좋겠다고 생각해서 선정하게 되었다.
- layer의 경우 1개일 때, batch size는 16일 때의 성능이 가장 좋았다.
- 하이퍼 파라미터를 수정해서 정말 간단한 형태로 만들고, 기존 베이스라인 내에서는 카테고리형 변수만을 임베딩해서 사용하기 때문에 예측에 도움이 될 연속형 변수 정보가 있을 것이라고 판단해서 사용자의 누적 정확도와 문제 풀이 시간 정보를 넣어주었다. 그 이전에 비해 아래의 그래프와 같이 성능이 크게 향상되었다.

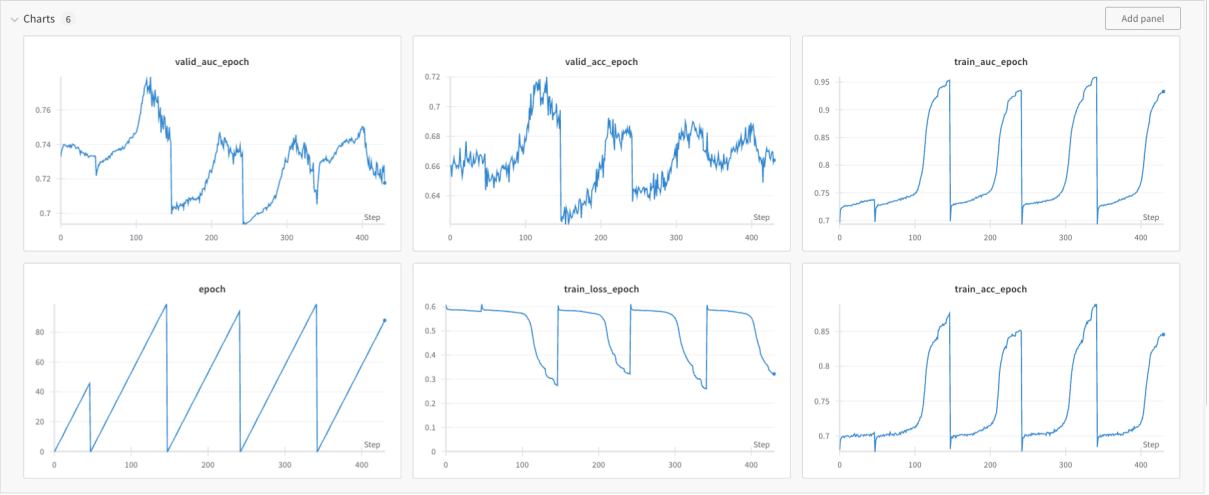


- Private 기준 AUC: 0.7702, ACC: 0.7016

4.3.9 GRUATTN

- 기본 GRU에서 입력시퀀스에 대해 중요한 정보에 집중할 수 있도록 하여 성능을 향상한다.
- GRU에 비해 확실히 빠르게 성능이 수렴하는 만큼 그래프를 보았을 때 불안정적으로 학습한다고 판단해서 최종적으로는 사용하지 않았다.

4.3.10 BERT



- 문항 간의 관계를 파악하는 Attention 기법을 unidirectional하게 수행하는 기존 Transformer 모델과 다르게, 이를 bidirectional로 파악한다. 이를 통해 문항 간의 더욱 심층적인 관계를 확인할 수 있을 것이라는 가설을 바탕으로 연구가 진행되었다.
- 기존 모델 BaseLine을 기반으로, Data Augmentation 및 K-Fold를 적용했다. Data Augmentation을 통해 Training Data의 수를 5358 → 24890으로 증가시켰다. 그리고 K-Fold를 통해 5개의 Best AUC 모델을 추출하고 이를 Ensemble해 결과를 도출하는 Process로 학습했다.
- Feature Engineering과 Hyperparameter Tuning은 따로 수행하지 않았다.
- 학습 결과 Public AUC 0.7378, ACC 0.6828로 미세한 성능 향상을 이루었으나, GBDT 모델 대비 성능이 낮다고 판단, 최종 Ensemble 과정에서는 제외했다.

K(Fold Num)	epoch(Max 100)	AUC	ACC	Public AUC	Public ACC
1	17	0.7402	0.6657		
2	73	0.7792	0.7187	0.7203	0.7151
3	65	0.7467	0.6873		
4	71	0.7456	0.6901		
5	59	0.7504	0.6833		
Average		0.7525	0.6890	0.7525	0.6890

- Private 기준 AUC: 0.7690, ACC: 0.7043

4.3.11 LQTR

- 지난 Riiid 대회의 우승 솔루션 중 하나인 LQTR을 구현해보았다. LQTR은 Transformer모델에 LSTM과 DNN을 합친 것으로 Positional embedding을 제거한 Transformer모델로 문제 간 특징을 학습하고 LSTM을 통해 시퀀스의 패턴을 학습한다. 처음에는 원래 모델과 동일하게 DNN을 사용했으나 결과가 너무 빠르게 수렴하는 것을 확인하고 모델의 복잡도를 낮추기위해 단일 fc layer로 교체하였다.
- sampling을 통해 hyper parameter 실험한 결과 hidden_dim=256, out_dim=128, n_layer=1, n_head=2, max_seq_len=100에서 가장 성능이 좋은 것을 확인할 수 있었다. drop_out은 성능에 큰 영향을 주지 않아 처음 설정인 0.2로 고정했다.
- 모델들의 성능을 비교해보니 머신러닝 모델들이 AI모델들보다 우수한 성능을 내는 것을 확인할 수 있었다. AI모델들이 성능이 그리 좋지 못한 이유가 데이터가 적기 때문일 수도 있다는 생각이 들어 Data augmentation을 해보았다. Data augmentation을 실행해본 결과, 일반화 성능은 좋아지지만 전체적인 지표가 나빠져 적용하지 않았다.

Max sequence length	window	Train AUC	Train ACC	Valid AUC	Valid ACC
100	50	0.7295	0.71463	0.71463	0.73606
150	50	0.72982	0.76215	0.71716	0.74031
100	100	0.72478	0.76215	0.7112	0.7439
150	100	0.73449	0.78016	0.70941	0.74094

- GRU기반 모델에서 시간과 정답률에 대한 feature를 추가했더니 성능이 올랐다는 결과가 있어서 LQTR에도 문제를 푼 시간과 유저의 전체 정답률 feature를 추가했다.
- Train ACC: 0.76726, Train AUC: 0.83163, Val ACC: 0.70896, VAL AUC 0.77637로 성능을 소폭 올릴 수 있었다.

4.3.12 LightGCN

- 모델 선정

  Knowledge Tracing에서는 문제에 대한 사용자의 정답 여부를 예측한다. 이는 추천 시스템에서 아이템에 대한 사용자의 선호도를 예측하는 작업과 유사하기 때문에 Knowlege Tracing에 추천 모델을 활용 수 있다.

  LightGCN은 NGCF에서 feature transformation과 nonlinear activation을 제거하여 학습의 효율성과 성능을 높인 CF 모델이다. 그래프 구조를 통해 사용자와 문제 간의 정답 여부에 대한 관계를 학습할 수 있기 때문에 사용자와 문제 간의 정답 여부 만을 이용하였음에도 좋은 예측 성능을 보였다.

- Validation 설정

  Test는 사용자가 가장 최근에 푼 문제 하나에 대한 정답 여부를 예측하는 것으로 설계되어있다. 따라서 Validation도 사용자가 가장 최근에 푼 문제 만을 포함하도록 재구성하였다. Test데이터와 개수가 동일하도록 Train 데이터에 포함된 사용자의 10%인 744명의 사용자가 가장 최근에 푼 문제로 Validation을 구성하였다.

- Hyperparameter 실험

  LightGCN의 Hyperparameter인 n_layers와 hidden_dim에 대한 최적의 값을 찾기 위해 실험을 진행하였다. n_layers는 그래프에서 정보가 전파되는 층의 개수를 나타내고, hidden_dim은 사용자와 아이템의 임베딩 크기를 나타낸다. Validation의 AUROC를 기준으로 모델의 학습 성능을 확인하였고, 20 epoch 동안 AUROC가 개선되지 않으면 early stopping이 실행되도록 설계하였다.

	Train AUC	Train ACC	Valid AUC	Valid ACC	Epoch	Time
n_layers=1 hidden_dim=64	0.8852	0.8223	0.8150	0.7446	727	1h 21m
n_layers=2	0.8884	0.8254	0.8142	0.7366	1187	3h 30m
n_layers=3	0.8853	0.8225	0.8117	0.7325	1695	5h 7m
hidden_dim=32	0.8744	0.8118	0.8048	0.7379	946	1h 33m
hidden_dim=128	0.8064	0.7681	0.7546	0.6962	42	0h 16m

n_layers의 경우에는, n_layers가 증가함에 따라 소요 시간도 증가하였지만 성능은 오히려 하락하였다. 정보 전파 층이 늘어나면서 연관이 적은 정보도 포함됨에 따라 약간의 오버피팅이 발생하는 것으로 판단된다.

hidden_dim의 경우에는, 32로 설정했을 때 임베딩의 크기가 작아서 학습 성능이 조금 떨어졌고, 128로 설정했을 때 임베딩의 크기가 너무 커서 학습이 제대로 이루어지지 않았다.

따라서, n_layers = 1, hidden_dim = 64인 LightGCN 모델을 채택하였다.

- Test 결과

	Public AUC	Public ACC	Private AUC	Private ACC
LightGCN	0.7794	0.6909	0.8145	0.7581

4.4 최종 모델 선정 및 결과

Ensemble 1 : LGBM_TTFE(0.5) + XGBoost(0.5)

Ensemble 2 : LGBM_Ensemble(0.5) [TTFE, RemoveOS, RemoveOS] + XGBoost(0.5)

Ensemble 3 : LGBM_TTFE(0.5) + Voting(0.5) [LGBM(0.25) + XGBoost(0.1) + CatBoost(0.05) + FFM(0.025) + LQTR(0.025) + LightGCN(0.025) + GRU(0.0125) + BERT(0.0125)]

	Public AUC	Public ACC	Private AUC	Private ACC
Ensemble 1	0.8160	0.7634	0.8475	0.7661
Ensemble 2	0.8208	0.7581	0.8401	0.7500
Ensemble 3	0.8168	0.7608	0.8435	0.7554

최종적으로 Public AUC 기준 가장 높았던 Ensemble 2, Public ACC 기준 가장 높았던 Ensemble1을 제출했다.

5. 자체 평가 의견

5.1 잘한 점들

- 데이터에 대해 충분한 이해를 거친 이후 feature engineering과 feature selection을 통해 자신이 세운 가설을 검증해보는 시간을 적절히 가진 것 같다.
- 팀원들 각자 해보고 싶은 모델들을 선정해서 develop해보는 시간을 가졌다.
- 다른 모델을 작업하고 있더라도 실험을 해보면서 얻은 insight를 공유했기 때문에 그로부터 또 새로운 아이디어가 생겨 적용해볼 수 있었던 것 같다.

5.2 아쉬웠던 점들

- 딥러닝 모델들이 성능이 좋지 않았는데, 앙상블에서도 성과가 없어 아무것도 쓰이지 못했다.
- 처음 합을 맞춰봐서 세팅, 그라운드 룰 정하는 것에 많은 시간이 소요돼서 나중에는 모델링 할 시간이 부족했다.
- 처음에는 깃허브(issue, pull request, 코드리뷰 등)를 잘 활용하면서 해보자고 했으나 그렇게 되지 못했던 것이 아쉽다.
- 6명이 비슷하게 제출 횟수를 쓰진 못한 것 같다. 다들 모델 성능을 local에서 개선하는 것도 중요하지만 제출해서 확인하면서 제출 기회를 아깝지 않게 사용하는 것도 좋을 것 같다.

5.3 프로젝트를 통해 배운 점 또는 시사점

- 기존의 모델들에서 구조들을 상황에 맞게 바꾸면서 여러가지 시도들을 해봐야겠다고 느꼈다.
- 딥러닝 모델을 develop하는 시간이 너무 적었기 때문에 최종 성능이 아쉬웠던 것 같다. 초반에 각자 베이스라인과 데이터를 이해하는 시간을 좀 줄이더라도 feature engineering을 최대한 빨리 취합한 이후에 바로 각자 하고 싶은 딥러닝 모델을 해보는 게 좋을 것 같다.
- 특히 이번 대회의 1등 팀의 결과를 확인해보았을 때, 탄탄한 모델 아키텍쳐가 관건이었다. 이 부분을 다음 프로젝트에 반영할 수 있었으면 좋겠다.
- 이번 대회가 원래는 public 순위와 private 순위의 shaking이 있는 편이라고 들었는데 그렇지 않았다. 그게 가능했던 이유는 최대한 오버피팅을 막으면서도 성능 개선을 하려고 주의했기 때문이라고 생각한다. 다음 대회에서의 overfitting 방지의 중요성을 깨달았다.

6. 개인 회고

6.1 김수빈

가장 먼저 BaseLine의 동작을 이해해보고자 했다. 전반적인 코드의 구조를 먼저 이해하고, 모듈들 간의 관계들을 이해하는데 많은 시간을 들였다. 그리고 Bert를 기반으로 하나의 모델을 깊이있게 파고자 했다.

이번 프로젝트에서 긍정적인 부분이 있었다면, K-Fold 및 Data Augmentation 등을 다른 Deep Learning 기반 DKT 모델에 맞게 적용한 것, 깃허브를 통해 코드를 관리하고, 버전을 컨트롤한 것이 떠오른다. BaseLine 모델의 구조가 굉장히 복잡한 편이다. 모듈과 모듈 간이 서로 밀결합되어있는 형태라, 모듈 간의 결합도(Coupling)가 상당히 높은 편이라고 느꼈었다. 이런 점에 있어서 베이스라인을 수정하는 과정이 상당히 힘든 과정이었다. 코드 간의 재사용성을 높인 측면에서는 현재 Baseline이 좋은 편이라고 생각하지만, 새로운 기능을 하나 구현하기 위해서 전체적인 Baseline을 모두 뜯어고쳐야했고, 그 과정에서 많은 시간이 소요되었다. 만일 다음에는 이러한 경우, Baseline을 깔끔히 포기하고 새로이 Baseline을 구성하는 것도 생각해보는걸로 하겠다.

그리고 이번 프로젝트를 진행하면서 유독 서버가 터진 적이 많았다. 그럴때마다 구현하고 있던 코드들이 모두 소실되고, 허무하게 포기하고, 좌절하는 경험이 많았었다. 그래서 그러한 이유로 깃을 관리하는 것에 더욱 집착을 했던 기억이 난다. 코드를 수정한 뒤 깃 커밋 그리고, 오류가 나면 롤백 및 폐기, 깃을 통해서 버전을 관리해가면서 서버가 터지더라도 당황하지 않고 이를 대응할 수 있게 되었다. 이 점은 다음 프로젝트 및 최종 프로젝트에서도 가져가보기로 했다. 그리고 Feature의 수가 많아짐에 따라서, Feature 관리 및 Data Version을 관리하는 여러가지 Tool들을 알게 되었다. (Feast, Data Version Control) 이런 도구들을 다음 프로젝트때 반영해보는 계획을 세우게 되었다.

아쉬운 점을 떠올리면, 프로젝트가 끝난 후, 다른 리포트를 확인했을 때, 문제 풀이 순서에 매우 중요한 패턴이 내재되어 있었는데, 앞의 문제를 푼 결과가 뒤 문제의 결과와의 관계성이 상당히 짙었다는 것이다. 그런데, Bert 기반 모델은 Bidirectional Task이므로, 단방향인 Transformer보다도 더 낮은 성능을 보였다는 내용이었다. 이런 내용을 빠르게 알았다면, 집착 없이 빠르게 포기했을 텐데 하는 생각도 들었다.

가장 아쉬운 점은 딥러닝 모델의 구조를 바꾸지는 못한 채, 곁가지들만 건드리고 프로젝트가 끝나버린 느낌이 너무 강했다. 가설을 바탕으로 딥러닝 모델의 전반적인 아키텍쳐를 건드려보는 경험은 결국 다음 프로젝트로 넘기게 되었다. 이 부분을 보완하기 위해서는 EDA부터 데이터 전처리에 이르는 순차적인 과정들을 착실히 밟고, 모델 아키텍쳐 설계에 Task를 집중해보자는 생각이 들었다.

가설을 세우고, 이를 적용해보고, 포기하거나 개선하거나를 결정하는 과정을 Agile하게 하는 방법에 대해서는 아직 숙제이다. 이 부분을 포함해보고자 한다.

6.2 박시우

lv1에서 아쉬웠던 점을 기반으로 학습 목표를 세워 달성하고자 하였다. 모델을 더 다양하게 사용해보고 eda를 통해 얻은 인사이트를 적용한 feature engineering을 통해 성능을 개선해보고 싶었다.

노력한 점

1. 베이스라인으로 제공된 것 외의 다양한 모델을 적용해보기 위해서, Xgboost, Catboost, Tabnet, GRU, GRUATTN 모델을 사용해 보았다.
2. 가정을 통한 실험, 결과 해석을 반복하며 data handling을 해보면서 모델의 학습 성능을 개선하였다. 기존 모델의 구조를 바꿔보기 보다는 feature engineering을 통해 인풋 데이터를 추가하거나 제거하면서 모델의 성능을 높였다.

   xgboost → lgbm을 위해 팀원들과 함께 만들었던 feature가 90개 이상이었기 때문에, validation auc가 오르면서도 train auc와의 차이가 크지 않도록 feature selection을 진행하였다. 서로 다른 피쳐를 앙상블한 모델로 private score 기준 auc 0.8543, acc 0.7769을 달성하였다.

   catboost → lgbm에 활용했던 feature와 유사하게 사용하였다.

   tabnet → lgbm에 활용했던 feature와 유사하게 사용하였다. 오버피팅 문제가 심해서 최종적으로는 사용하지 않았다.

   GRU, GRUATTN → GRUATTN은 오버피팅이 빠르게 발생하는 문제가 있어서 가벼운 GRU 모델을 사용해서 선택과 집중을 하기로 했다. GRU에 기존 베이스라인에는 categorical 데이터만을 embedding하여 학습하기 때문에 시간 정보와 사용자에 대한 정보가 필요하다고 생각했다. user의 누적 정확도와 문제 풀이 시간을 넣어 학습한 결과 확실히 accuracy가 높아지는 모습을 보였다.
3. 팀원들과 활발한 협업을 진행하기 위해서 혼자 실험을 하면서 얻은 인사이트를 공유하고 문제가 있을 때는 함께 의논하여 해결하려고 노력했다.

   GRU 모델에서 feature engineering을 해서 개선된 결과를 공유해서 다른 딥러닝 모델에서도 팀원들이 똑같이 적용해서 성능을 개선하였다.

아쉬운 점

1. 모델의 구조를 바꿔보는 창의적인 시도를 해보지 못했다. 프로젝트 주제에 맞게 기존 모델 구조를 변형해보는 시도를 해봤어도 좋았을 것 같다.
2. 트리 모델의 성능을 딥러닝 모델의 성능이 넘게 만들지 못했다. 1번과 연결되는 내용으로, 모델 구조를 바꿔 봤으면 가능하지 않았나 싶다.

6.3 백승빈

프로젝트를 진행하며 시도한 것

지난 프로젝트의 결과가 좋지 않았던 이유 중 하나가 정보 부족이었다는 생각이 들어 이번 프로젝트에는 지난 기수들의 결과와 다른 유사한 대회들의 솔루션을 보며 인사이트를 얻으려고 했다.

- 모델 구현 : 지난 Riiid 대회의 우승 솔루션이었던 LQTR을 구현하였다. 처음에는 원래 모델과 동일하게 DNN을 사용했으나 결과가 너무 빠르게 수렴하는 것을 보고 복잡도를 낮추기위해 단일 fc layer로 교체하였다.
- Data augmentation : AI모델들의 성능을 높이기위해 Data augmentation을 진행했다. 역시 지난 Riiid 대회 솔루션의 일부였던 window 방식과 다른 팀원분의 아이디어였던 시퀀스의 끝은 고정한채로 시작시점만 바꾸는 방법 2가지를 시도했다. 두번째 방식은 overfitting이 심하여 폐기되었다. 동일한 시퀀스가 여러 번 출연하게 그런 것이라 예상된다.
- hyper parameter 튜닝 자동화: 이번 프로젝트에서는 ray를 이용하여 hyper parameter 튜닝을 자동화시키는 작업을 해보았다. 직접 여러 조합을 시도하는 것보다 꼼꼼하게 실험해볼 수 있고 한번에 여러 실험을 진행하는 동안 다른 작업을 미리 할 수 있어서 시간 절약 측면에서도 이점을 볼 수 있었다.

프로젝트를 진행하며 배운 것

- 시간 부족의 문제로 시퀀셜 모델쪽에 feature engineering을 충분히 해보지 못했는데 data augmentation에 시간을 덜 쓰고 이 부분에 신경을 썼다면 모델의 성능을 더욱 끌어올릴 수 있었을 것 같아 아쉽다. 어떠한 실험이 도움이 안된다면 깔끔하게 미련을 버리고 다른 실험을 진행하는 것이 필요한 것 같다.

다음 프로젝트 때 개선할 점

- 단순히 존재하는 모델을 구현하는 것 뿐만 아니라 현재의 문제에 맞게 잘 변형시키는 것을 시도할 필요를 느꼈다. 모델 구조에 대해 특정한 부분이 주는 이점을 정확하게 파악하고 이를 바탕으로 모델을 어떻게 변화시켜 더 상황에 적합하게 만들 수 있는지 고민해봐야할 것 같다.
- 프로젝트를 진행하면서 서로 맡고 있는 작업들을 확인하기 편리하도록 Workflow 차트를 만들었는데 생각보다 이용을 잘 하지않게 되었다. 피어세션 때 각자 자신이 수행한 작업들에 대해 공유했지만 진행하다보니 조금씩 작업들이 겹치는 부분이 생겼다. 지금 현재 Workflow 차트 파일의 위치가 접근성이 나빠서 이용을 잘 안하게 되는 것 같은데 다음 프로젝트 때는 이러한 점을 보완해 서로의 작업들을 파악하는 것이 더욱 용이해졌으면 좋겠다.

6.4 이재권

이번 프로젝트에서 잘한 점과 시도한 것

- 모델의 일반화에 도움이 되도록 Validation 데이터를 구성하였다. 베이스라인 코드에서는 Train 데이터 전체에 대한 비율로 Validation 데이터를 구성하였다. Test 항목과 동일하게 가장 최근에 푼 문제 만으로 Validation 데이터를 재구성하였다. 그 결과, Validation 결과와 Test 결과 사이의 간극을 좁힐 수 있었다.
- 저번 프로젝트에서 EDA와 Feature Engineering의 중요성을 느꼈었다. 그래서 이번 프로젝트에는 EDA와 Feature Engineering에 많은 시간을 투자하였고, ML 모델들의 성능 향상을 위해 여러 파생 변수도 생성하였다.
- 원활한 협업을 위해 팀원들과 함께 프로젝트 전반적인 계획을 세우고, 개별 계획을 계속 공유하면서 프로젝트를 진행하였다. 지속적인 공유를 통해 서로 어떤 것을 진행하고 있는지 확인할 수 있었고, 효과적으로 협업 할 수 있었다.

이번 프로젝트에서 아쉬웠던 점과 다음 프로젝트에서 시도해볼 것

- 이번 프로젝트에서는 GNN 기반 모델을 사용하는 데에 있어 사용자와 아이템의 특성을 반영하지 않고 협업 필터링 방법만 활용하였다. 협업 필터링 방법은 학습이 효율적이고 성능도 좋았지만, 더 많은 데이터를 활용하여 정확도를 높이지 못한 점이 아쉬웠다. 다음 프로젝트에는 사용자와 아이템의 특성을 반영한 Heterogeneous Graph 혹은 Knowlege Graph를 사용해보고 싶다.
- 이번 프로젝트에는 EDA, Feature Engineering, LightGBM 모델링을 함께 진행을 하였다. 저번 프로젝트에서 EDA와 Feature Engineering의 중요성을 느꼈고, ML 모델이 좋은 성능을 보일 것이라는 판단으로 그렇게 계획하였다. 그 결과, LightGBM에서 좋은 성과가 나왔지만 생각보다 작업 효율이 나오지 않았고, 다음 프로젝트에는 각자 데이터를 이해하고 공유한 후에 Feature Engineering과 모델링을 분업하는 방식으로 더 효율적이고 효과적인 협업을 진행해보고 싶다.

6.5 이진민

시도하였던 것들

- 일단 저번 대회때 새로운 feature들을 거의 만들어 내지 못해서 feature engineering에 많은 시도들을 해봤었다.
- feature_selection을 활용하며 LightGBM모델 feature 실험
- 저번 대회때 k-fold를 사용하지 못했어서 아쉬움이 많이 남아 k-fold를 구현했다.
- 머신러닝 모델들과 앙상블을 할 때 조금의 성능향상을 기대하고 FM, FFM을 만들고 feature들을 실험하였다.

마주한 한계 및 아쉬웠던 점

- 모델링에 적은 시간을 할당해서 좋은 성능을 내는 딥러닝 모델들을 만들지 못했던 점
- 깃허브의 적극적인 활용이 제대로 되지 않았던 점
- 시퀀스 특성이 있는 데이터이지만 새로운 시퀀스 모델링을 시도하지 않았던 점
- k-fold 를 구현하긴 했었지만 많이 활용하지 않았던 점

- 처음 만나는 팀원들이어서 어떻게 진행할 것인지를 정하는 것에 많은 시간이 소요되기도 했고 EDA, Feature Engineering에 너무 많은 시간을 할당했던 점
- 생각보다 데이터 전처리에 많은 시간이 소요되었던 점
- FM, FFM이 앙상블에 좋은 결과를 내지 못했던 점
- LightGBM 오버피팅을 해결하지 못했던 점

다음에 시도해볼 것들 및 느낀점

- 데이터의 특성들을 먼저 파악한 뒤 가설을 세운 뒤 데이터와 좀 더 상성이 잘 맞을 것 같은 모델들을 먼저 시도해봐야겠다고 생각했다.
- 저번 대회때는 많은 feature들을 새로 만들진 않았었는데 이번 대회때 시도해본 결과 잘 만든 feature들이 모델 성능에 많은 영향을 미친다는 것을 느꼈고 좋은 아이디어로 feature들을 많이 만들고 실험해봐야겠다고 느꼈다.
- 기존에 있는 모델들에서 주어진 데이터에 맞게 변형하면서 모델링을 해봐야겠다고 생각했다.

6.6 장재원

지난 프로젝트에서 가설 설정과 분석 과정에 있어 아쉬움을 느꼈다. 이를 보완하고자 이번에는 EDA를 통해 초기에 가설을 설정하고, 이를 실험을 통해 검증하는 과정을 거쳤다.

이번에 시도한 것들

- EDA와 FE를 통한 다양한 가설 설정
  EDA를 진행하며 데이터의 다양한 특징들에 근거하여 여러 가설들을 세웠다. (예: 문제의 난이도가 정답률에 영향을 미칠 것이다, 유저별로 지식 수준이 다를 것이다 등). 가설 검정 과정도중에 'Dataset Overall Statistics'가 과적합을 일으키는 현상을 발견하였고 이를 해결하기위해 'Test Data가 작아서 발생하는 문제다'라는 새로운 가설을 세운 후, dataset을 수정하여 문제를 해결하였다.
- 모델 성능 향상
  LGBM의 성능 향상을 위해 FE 과정에서 추가한 다양한 피처들에 대해 Feature Importance를 비교하고, 이를 바탕으로 Feature Selection을 진행하였다. 이후에는 하이퍼파라미터 튜닝을 통해 모델의 성능을 더욱 높였다.

앞으로 시도할 것들

- 각 모델의 성능 향상을 FE비교
  LGBM에서 사용한 피처들이 다른 모델에서도 높은 성능을 보여주지는 않았다. 이를 통해 각 모델마다 적합한 FE가 필요하다는 사실을 깨달았다. 다음번에는 같은 Feature들이 다른 모델에 대해서 어떠한 영향을 미치는지 그리고 각 모델에는 어떠한 Feature들이 적합한지 비교해봐야겠다.
- 협업툴을 활용한 이슈 공유
  우리 팀은 Github, Notion, Slack과 같은 다양한 협업툴을 사용하였다. 하지만 이러한 다양한 협업툴을 사용하다 보니, 이슈 공유가 여러 툴에서 이루어지게 되어 의사소통이 원활하게 이루어지지 않았다. 다음번에는 협업툴별로 역할을 명확히 부여해야겠다.