프로젝트 진행상황

(6/10 - 6/17)

AI0팀

Task, Pipeline

Task : **고객이 했던 주문정보를 보고 다음 주문을 예측하는 다중 레벨 분류**

Pipeline :
- **오토인코더로 주문 제품을 임베딩 벡터로 변환**
-**임베딩 벡터와 데이터셋으로 프레딕터 학습**

현재 진행상황

1. 오토인코더로 제품 인베딩 벡터 변환 - 완료
2. 프레딕터 CNN->GRU->(예측)구조 – 완료
3. XGBoost 와 비교
    - 결과 도출 후 평가 완료

XGBoost 사용

입력 데이터 : **주문당 최대** 145**개 제품 × 최근** 5**회 주문**
× 제품 임베딩 16 **형태**
- XGBoost**는** 2D **입력만 허용되며**, 3D **구조를 직접 처리할 수 없음**
- **멀티라벨 예측을 위해 약** 50,000**개 제품을 각각 이진 분류하는** OVR**구조**(145x5x16)**를 그대로 적용하면 모델 수가 과도하게 많아짐**.
- **학습 속도 및 메모리 사용량 문제로 중단**.

해결방안 : CNN**으로** (145, 64)**주문을** 128**차원으로 요약하듯이**
미리 고차원 구조를 1D **벡터로 요약 후** 2D **벡터로 입력**
- Top-K **후보 기반 예측으로 모델 수 감소**
- **약**30**만 사용자 ×** 128**차원 벡터 입력**, **다중 레이블**(Top-K) OVR **분류 구조로 변경하여 진행**

XGBoost 전처리 방식요약

미리 학습된 16**차원 제품 임베딩**(ev_final16.npy) **사용**

- **각 주문 내 제품 임베딩을** W @ ev + b (1-layer MLP) **적용 후**, Max Pooling
- **최근** 5**회 주문을 시계열 가중합** (λ=0.8)**으로 통합해**
  -> **유저 벡터**(seq_vec128.npy) **생성**
- **유저 벡터와 제품 벡터 간 내적 유사도 기반으로 상위** K**개 제품 추출** (K=300)
- **실제 주문**(product_id)**이 후보 안에 존재하면** 1, **없으면** 0
  -> CSR **라벨행렬**(Y_topk.npz) **생성**

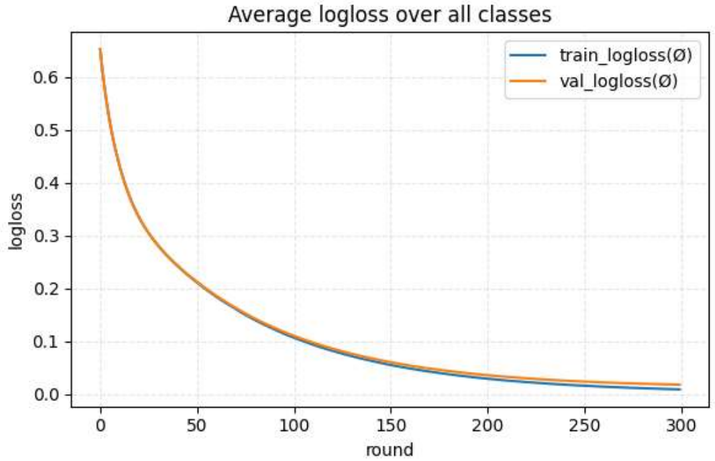
XGBoost 학습 설정

모델 구조 : One-vs-Rest, **총** 300**개 이진 분류기**
입력 X : seq_vec128.npy (304k × 128) → X_128.npz**로** CSR **저장**
라벨 Y : Top-K **후보** 300**개에 대한 다중 레이블** → Y_topk.npz
트리 설정 : tree_method = gpu_hist, predictor = gpu_predictor, max_depth = 6,
learning_rate = 0.1
클래스 불균형 보정 : scale_pos_weight = (#negative / #positive) per class
Early Stopping, patience = 30
평가 지표 : logloss (**각 클래스**), **최종적으로** F1-micro **전체 평균**

XGBoost 성능평가, 요약

F1 스코어 결과

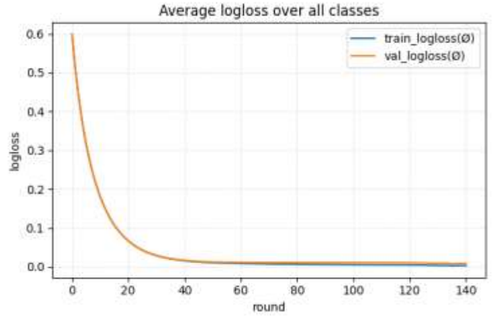
- F1-micro (threshold=0.4 기준) : 0.0119



요약

DL model 과 비슷한 구조로 진행됨

- (145, 5, 64) 주문 시퀀스를 (사용자, 128)로 미리 전처리 (CNN)
- 전처리한 데이터를 활용해 XGBoost 학습 (GRU + FC)
- ML model 역시 pos_weight를 활용했음
- DL 모델 대비 baseline으로 활용 가능



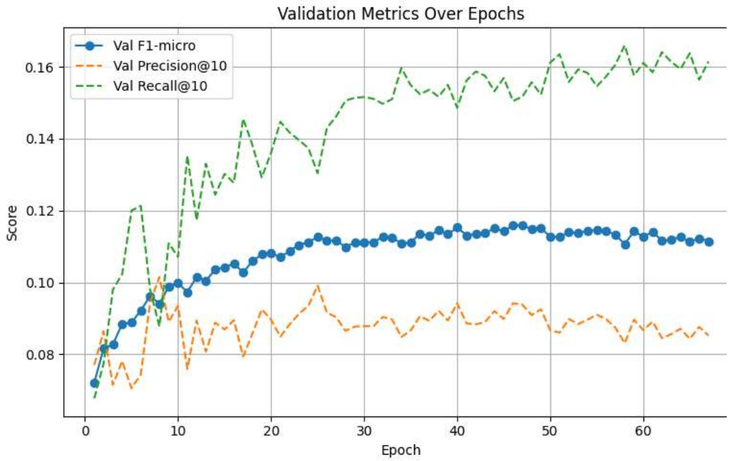
pos_weight 미활용시

f1micro = 0.0021

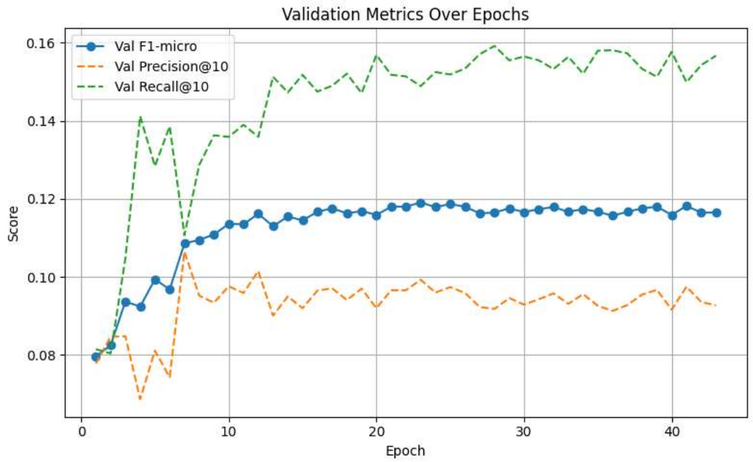
제품 성능을 높이기 위한 시도(predictor)

CNNGRU **미세조정**

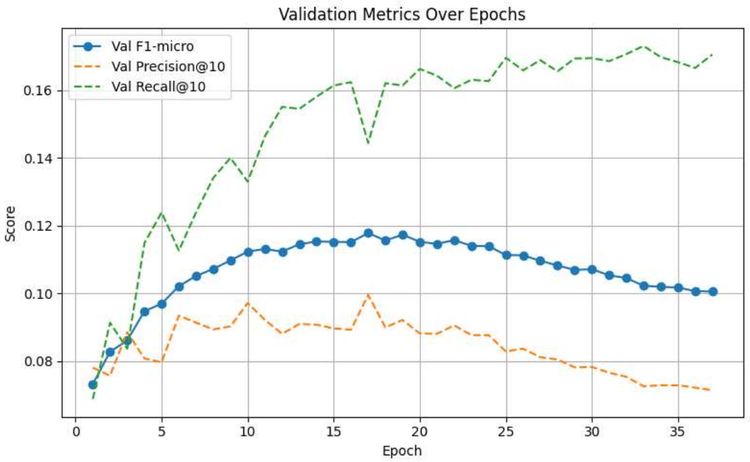
- CNNGRU_1 : cnn 128, kernel 4, pool 8, GRU 256, dropout 0.1, lr 0.0001
- CNNGRU_2 : cnn 128, kernel 4, pool 8, GRU 256, dropout 0.2, lr 0.001
- CNNGRU_3 : cnn 256, kernel 4, pool 8, GRU 256, dropout 0.1, lr 0.0005



CNNGRU_1



CNNGRU_2



CNNGRU_3

결과 : **학습률** 0.001, dropout 0.2 **에서 안정적인 성능**

제품 성능을 높이기 위한 시도(predictor)

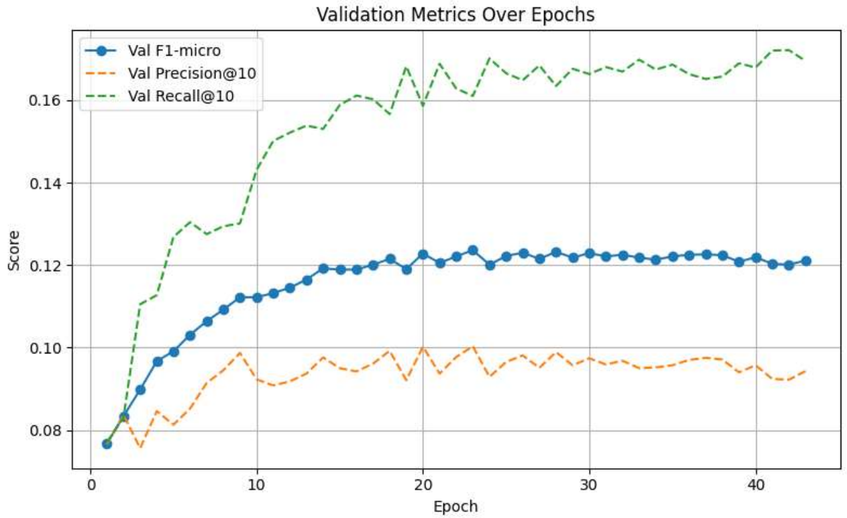
Feature **제거**

이전의 주문 특징을 넣으니 f1micro**가** 0.12**를 넘지 못함**

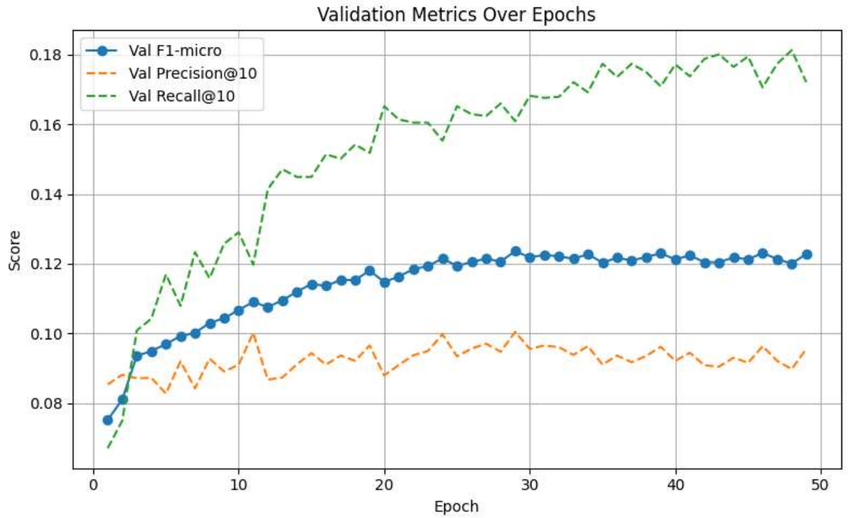
-> **제거 후 간단 조정**

CNN1 : cnn 256, kernel 5, pool 16, GRU 256, dropout 0.2, lr 0.0005

CNN2 : cnn 256, kernel 4, pool 8, GRU 256, dropout 0.2, lr 0.0001



CNN1

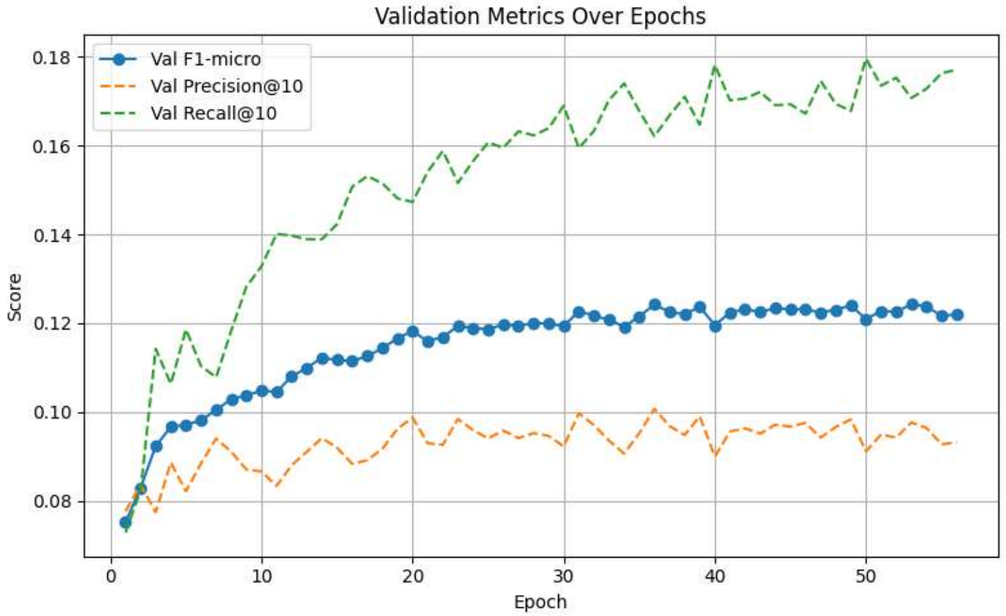


CNN2

결과 : CNN2(kernel4, pool8)**에서 안정적인 성능**

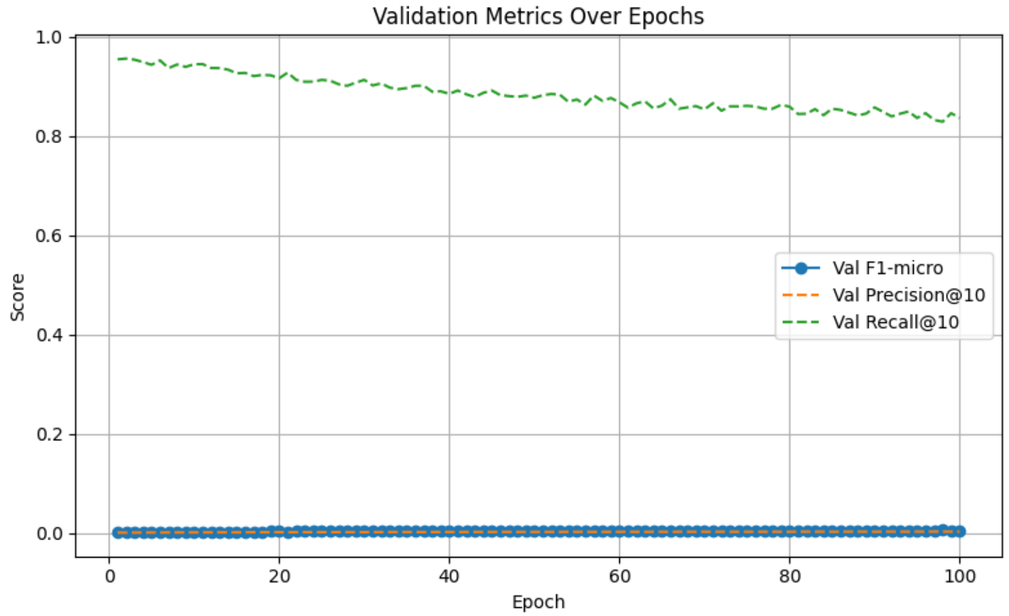
제품 성능을 높이기 위한 시도(predictor)

- **손실함수 비교** : CNN2**의 하이퍼파라미터에서 학습률만** 0.0005**로 변경**
- BCEWithLogitsLoss: **기본 기준**



BEC

- BPR Loss: **단독 사용 시 학습 실패** (F1 ≈ 0)

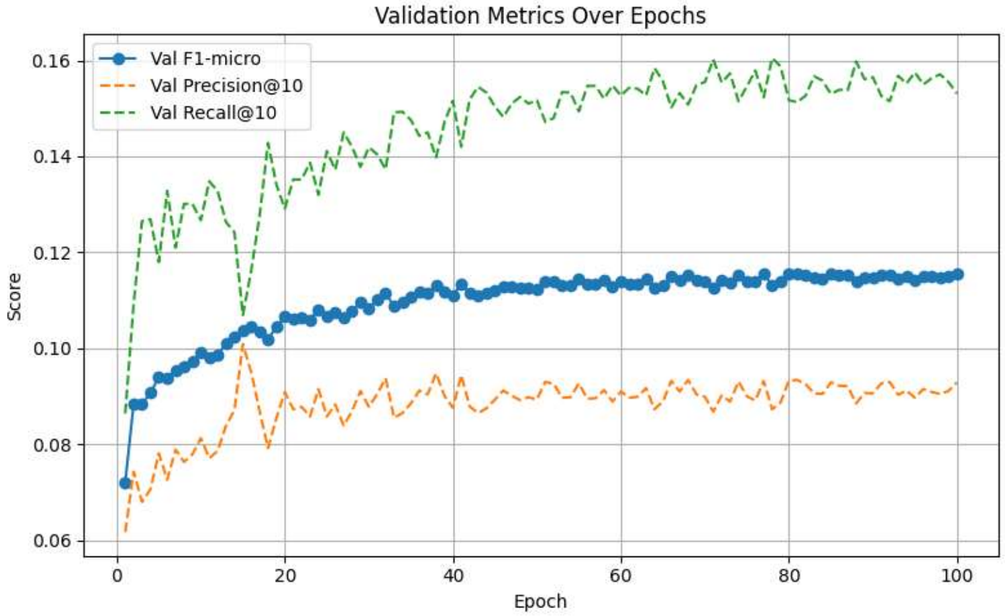


BPR

MarginRankingLoss : BPR로 학습이 안되는걸 확인해 비슷할거라 여겨 진행 X

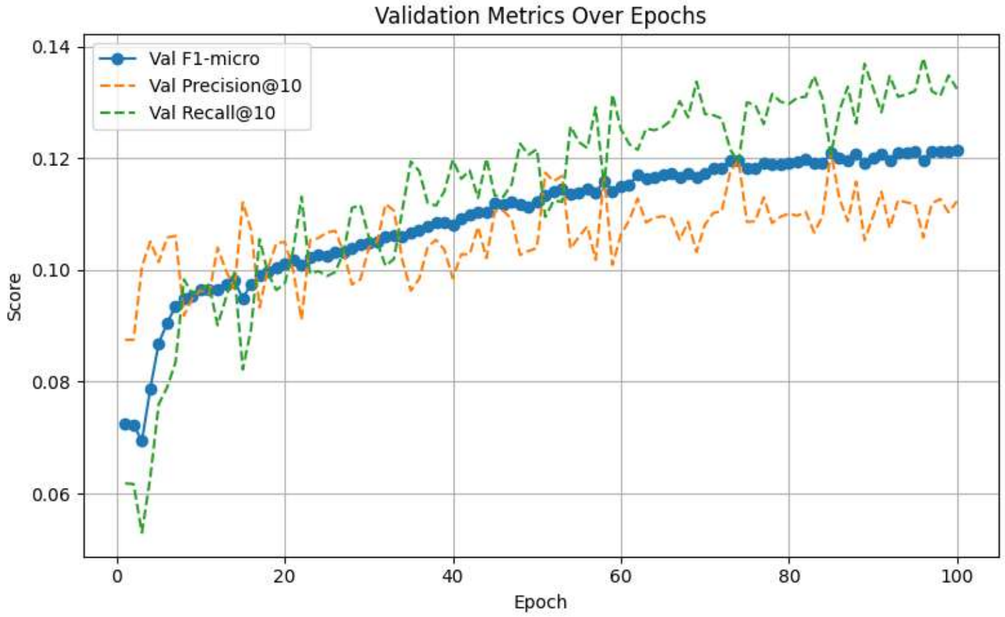
제품 성능을 높이기 위한 시도(predictor)

- BCE + BPR : BCE **단독보다 개선이 되지 않음**



BCE + BPR

- BCE + Margin : Recall**이 많이 감소했지만**
  Precision**이** 0.11 **수준으로 최고점**
  -> **정밀도는 높아졌지만**, Recall>Precision**의 중요도로 보는 게 모델의도에 맞다고 봄**



BCE + Margin

제품 성능을 높이기 위한 시도(predictor)

결론 요약

XGBoost f1micro = 0.012
CNNGRU f1micro = 0.121

Best **구조** : CNN -> GRU **기반 모델**

Best loss: BCEWithLogitsLoss **단독**
-BPR **단독**/Ranking **단독** -> **학습 실패**
-BPR/Ranking **혼합** -> **성능 하락**
혼합 손실 실험 결과 :
혼합은 가능하지만 BCE **단독이 가장 안정적인 성능을 보여줌**