현재 진행상황

AI0팀

요약

- Task: 한 고객의 과거 주문 시퀀스를 받아와서 다음 주문을 예측하는 다중 레벨 분류
- 진행 pipeline

  -오토인코더로 각각의 제품을 임베딩 벡터로 변환한 파일 제작

  -EV파일과 데이터셋을 활용해 예측하는 모델(Predictor) 학습

  -Predictor와 XGBoost(Baseline ML모델) 성능 비교

진행상황

- 오토인코더로 제품 임베딩 벡터 생성 완료

TODO

- Predictor 훈련 및 성능 개선 (진행 중)

  -하이퍼파라미터 튜닝, 여러 가중치 추가

- XGBoost랑 성능 비교

Predictor 훈련 계획

- 직접 예측하는건 GRU를 활용
- GRU 입력에 (Batch, 145(제품수), 64(EV차원), N(주문수))를 입력하는것도 많다고 판단하여 CNN/Attention으로 차원 축소를 진행한 뒤 예측
- CNNGRU, AttentionGRU 두 모델 훈련 후 성능 평가

CNNGRU

1. CNN (OrderEncoderCNN)

- **목적**: 한 번의 주문에 담긴 여러 제품을 하나의 벡터로 요약
- **방법**:
  - 각 제품을 임베딩(embedding)한 후, 1D CNN으로 시퀀스(제품 나열)를 처리
  - CNN의 출력은 `AdaptiveMaxPool1d`로 압축되어, **주문 하나당 하나의 벡터**가 생성됨

  → 즉, 주문 내 여러 제품을 한 번에 요약하는 역할

2. GRU (OrderSequenceGRU)

- **목적**: 고객의 여러 주문(시퀀스)을 시계열 데이터로 처리하여, 전체적인 소비 패턴을 파악
- **방법**:
  - 위에서 CNN을 거친 각 주문의 벡터들을 **시간 순서대로** GRU에 입력
  - GRU는 시퀀스 정보를 요약해 마지막 은닉상태를 출력

  → 즉, 고객의 과거 여러 주문을 한 줄기로 요약하는 역할

CNNGRU

3. **최종 FC (PredictionHead)**

- **목적**: 다음번 주문에서 구매할 제품(여러 개 가능)을 예측
- **방법**:
  - GRU의 마지막 은닉상태를 Fully Connected(FC) 레이어에 통과
  - 출력 차원은 전체 상품 수와 같으며, 각 상품별로 예측 점수(로짓)를 산출

  → *이 고객이 어떤 상품을 주문할지 확률적으로 예측*

입출력 예시
-입력 :["Organic Strawberries"],
["Organic Baby Spinach", "Organic Hass Avocado", "Organic Strawberries"],
["Large Lemon", "Chicken Breast", "Organic Whole Milk", "Organic Strawberries"],
["Organic Whole Milk"],
["Organic Hass Avocado", "Organic Strawberries", "Organic Whole Milk"]
-출력 :["Organic Whole Milk", "Banana"]

CNNGRU 훈련

- **클래스 불균형 문제 해결 시도 (`pos_weight` 적용)**:
- 다음 장바구니 예측 문제의 특성상 '미구매' 상품이 '구매' 상품보다 훨씬 많아 클래스 불균형이 심각함. 이로 인해 모델이 단순히 '미구매'로 예측해도 BCE 손실은 낮게 나올 수 있지만, 실제 '구매' 상품을 예측하는 성능(특히 Recall 및 F1-macro)은 매우 낮았음.
- **시도: `pos_weight` 조정 및 `metrics_threshold=0.5`(제품을 구매로 판단할 확률 임계값)**
  - `--pos_weight=100.0`으로 `pos_weight` 값을 설정하고, `metrics_threshold`는 0.5로 고정하여 실험.
  - **결과**: Recall@K는 낮아졌지만(약 0.27~0.32), Precision@K는 약간 상승했고(약 0.02~0.03), F1-micro 점수도 이전보다는 개선됨(약 0.04~0.05). 하지만 F1-macro는 여전히 매우 낮음.
  - **분석**: `pos_weight`를 낮추니 Precision과 Recall 간의 극단적인 불균형이 다소 완화되었으나, 여전히 Precision이 매우 낮아 개선이 필요함

Pos_weight와 threshold 최적화 조합

- Grid Search를 통해 여러 조합을 시도했음
- 평가지표는 f1micro(precision + recall), precision@10(예측 상위 10개중 실제로 맞춘 비율),

recall@10(실제로 구매한 제품 중 얼마나 상위 10개에 들어갔는지 비율)을 사용함

- Pos_weight=5, threshold=0.4일때 가장 높음
- 2위의 pos_weight, threshold가 높아서, 더 높은 수치를 넣어서 Grid Search 진행 예정

```
N_ORDERS = 5
ORDER_HIDDEN = 128
BATCH_SIZE = 16
EPOCHS = 3
LEARNING_RATE = 1e-3
ATTN_HEADS = 4
N_LAYERS = 2
DROPOUT = 0.1
=== Grid Search 결과 요약 ===
```

	pos_weight	threshold	f1_micro	precision@10	recall@10
5	10	0.4	0.105563	0.129102	0.081887
19	25	0.6	0.104180	0.068416	0.164010
10	15	0.5	0.102892	0.118885	0.090692
9	15	0.4	0.102495	0.108809	0.096474
4	10	0.3	0.101659	0.137660	0.075124
15	20	0.6	0.101397	0.090794	0.126132
13	20	0.4	0.101320	0.102075	0.110605
0	5	0.3	0.100314	0.215606	0.032878
6	10	0.5	0.098071	0.138446	0.075928
8	15	0.3	0.095769	0.124224	0.085353
23	30	0.6	0.093565	0.058689	0.157434
22	30	0.5	0.092355	0.065590	0.156026
11	15	0.6	0.091750	0.104277	0.101381
14	20	0.5	0.089566	0.088571	0.090583
18	25	0.5	0.089523	0.062313	0.158914
17	25	0.4	0.085763	0.074391	0.139490
1	5	0.4	0.081889	0.242647	0.031489
12	20	0.3	0.075449	0.086305	0.127232
7	10	0.6	0.074973	0.151838	0.070645
21	30	0.4	0.071953	0.060414	0.134628
16	25	0.3	0.063714	0.070410	0.142501
20	30	0.3	0.056678	0.059677	0.160584
2	5	0.5	0.052465	0.232383	0.029571
3	5	0.6	0.044918	0.222160	0.040905

CNNGRU 개선사항

- 현재 모델은 "실제 구매될 상품 중 일부는 찾아내지만(적당한 Recall), 추천한 상품 대부분은 틀리는(매우 낮은 Precision)" 상태

->F1 점수를 실질적으로 향상시키기 위해서는 Precision과 Recall의 균형을 맞추는 것이 중요

- 학습률 스케줄러 도입 및 학습 안정화
- 모델 정규화 강화
- 위 방법으로도 개선이 미미할 경우 모델 변경 고려