LGAIMERS 3기

솔루션 발표

팀 프이 (김동규, 전현욱)

목차

대회 개요

대회 배경 및 주제

배경 및 주제

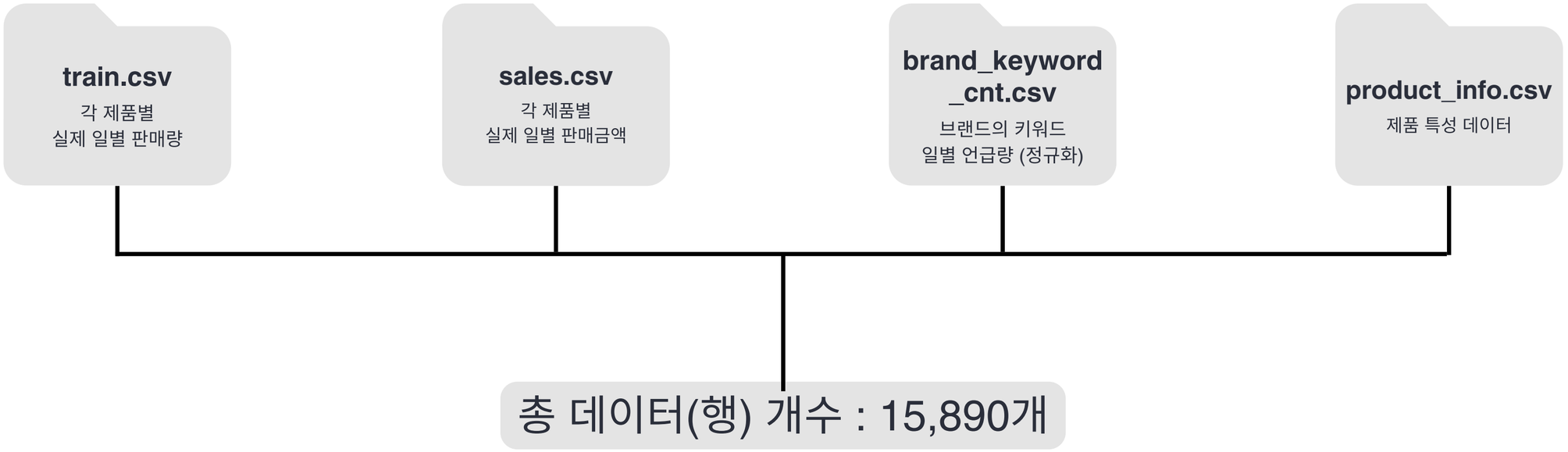
- 기업의 효율적인 재고 관리와 타켓 마게팅 전략을 세우기 위해 판매량 예측의 필요성 증대
- 온라인 판매 채널에서 수집되는 대규모 데이터를 활용해 판매 예측을 수행하는 AI 모델 개발

목표

- 온라인 채널 제품들의 실제 판매량을 예측 (23.04.05~ 23.04.25)
- 위 예측값의 PSFA 정확도를 최대화 하는 모델 개발

대이터 셋 요약

데이터 셋


train.csv
각 제품별
실제 일별 판매량
sales.csv
각 제품별
실제 일별 판매금액
brand_keyword
_cnt.csv
브랜드의 키워드
일별 언급량 (정규화)
product_info.csv
제품 특성 데이터
총 데이터(행) 개수 : 15,890개

대이터 셋 요약

train.csv

Features

- ID : 실제 판매되고 있는 고유 ID
- 제품 : 제품 코드
- 대분류 : 제품의 대분류 코드
- 중분류 : 제품의 중분류 코드
- 소분류 : 제품의 소분류 코드
- 브랜드 : 제품의 브랜드 코드
- 2022-01-01 ~ 2023-04-04 : 실제 일별 판매량

brand_keyword_cnt.csv

Features

- 브랜드 : 브랜드 코드
- 2022-01-01 ~ 2023-04-04 : 브랜드의 연관키워드 언급량을 정규화한 일별 데이터

sales.csv

Features

- ID : 실제 판매되고 있는 고유 ID
- 제품 : 제품 코드
- 대분류 : 제품의 대분류 코드
- 중분류 : 제품의 중분류 코드
- 소분류 : 제품의 소분류 코드
- 브랜드 : 제품의 브랜드 코드
- 2022-01-01 ~ 2023-04-04 : 실제 일별 총 판매금액

product_info.csv

Features

- 제품 : 제품 코드
- 제품특성 : 제품 특성 데이터(Text)

대회 개발 환경

Google Colab에서 진행

- OS: Ubuntu 22.04.2 LTS
- System RAM: 51.0 GB
- GPU(V100) RAM: 16.0 GB
- Python Version: Python 3.10.12
- ML/DL FrameWork: Torch 2.0.1+cu118

사용한 라이브러리는 코드제출 파일에 자세하게 기록

```
1 !pip list
```

Package	Version
absl-py	1.4.0
aiohttp	3.8.5
aiosignal	1.3.1
alabaster	0.7.13
albumentations	1.3.1
altair	4.2.2
annotated-types	0.5.0
anyio	3.7.1
appdirs	1.4.4
argon2-cffi	23.1.0
argon2-cffi-bindings	21.2.0
array-record	0.4.1
arviz	0.15.1
astropy	5.3.2
astunparse	1.6.3
async-timeout	4.0.3
attrs	23.1.0
audioread	3.0.0
autograd	1.6.2
Babel	2.12.1
backcall	0.2.0
beautifulsoup4	4.11.2
bleach	6.0.0
blinker	1.4
blis	0.7.10
blosc2	2.0.0

데이터 분석

데이터 분석 방법

스케일링

- train.csv 파일의 일별 데이터에 스케일링 적용
- StandardScaler로 진행

그룹화

- 스케일링된 데이터를 각 범주 데이터별로 그룹으로 묶고 일별로 더함

범주 데이터:
['대분류', '중분류', '소분류', '브랜드']

```
1
2 # 대분류와 기준으로 그룹화하여 개수를 합산
3 big_grouped = train_data.groupby(["대분류"]).sum().reset_index()
4 big_grouped.head()
```

```
<ipython-input-7-cd1125b1082d>:2: FutureWarning: The default value of numeric_only in DataFrameGroupBy.sum is
  big_grouped = train_data.groupby(["대분류"]).sum().reset_index()
```

	대분류	2022-01-01	2022-01-02	2022-01-03	2022-01-04	2022-01-05	2022-01-06	2022-01-07	2022-01-08
0	B002-C001-0001	-644.668585	-814.659531	-809.953695	-601.581613	-65.972421	242.159825	239.527948	-74.019514
1	B002-C001-0002	-1579.309552	-2278.686137	-2339.700640	-1951.209608	-767.059821	-213.347378	-275.740877	-1117.988364
2	B002-C001-0003	-259.811012	-260.207453	-263.439624	-265.735887	-263.083525	-258.230495	-260.633613	-260.248205
3	B002-C001-0004	-33.473165	-35.056539	-32.480853	-31.046897	-29.028677	-31.562610	-29.480799	-27.636053
4	B002-C001-0005	-146.447020	-175.320747	-195.275983	-176.575959	-147.998581	-154.828800	-68.283497	-126.574850

데이터 groupby

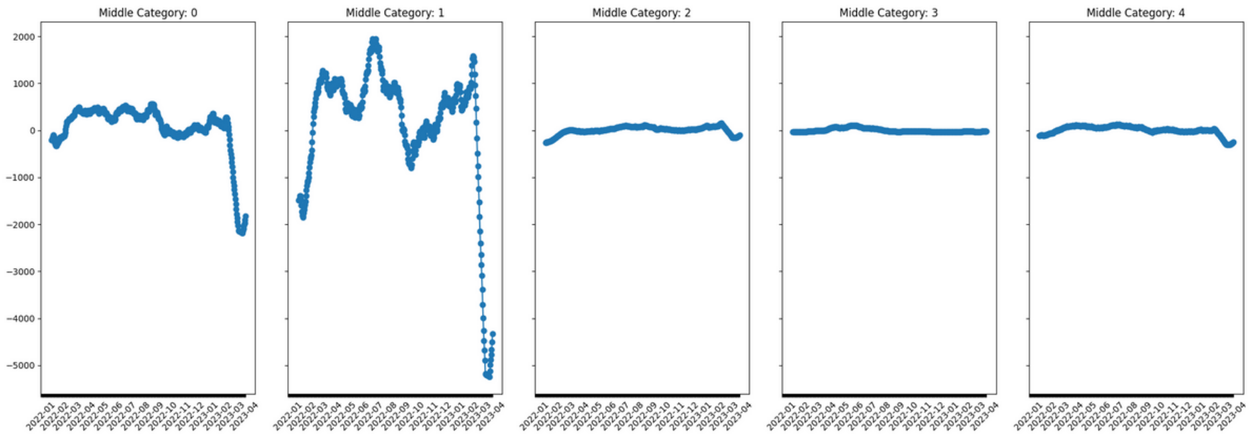
데이터 분석 방법

평균 이동 적용

- 25일 간의 단순 이동 평균을 적용
- 보다 뚜렷한 흐름 및 판매 변화량을 파악
- pandas의 `.rolling()` method 사용

시각화

- 그룹으로 묶인 데이터들의 matplotlib 라이브러리로 시각화를 진행
- 나머지 범주 데이터도 위와 같은 방식으로 시각화를 통해 데이터의 상관성을 파악



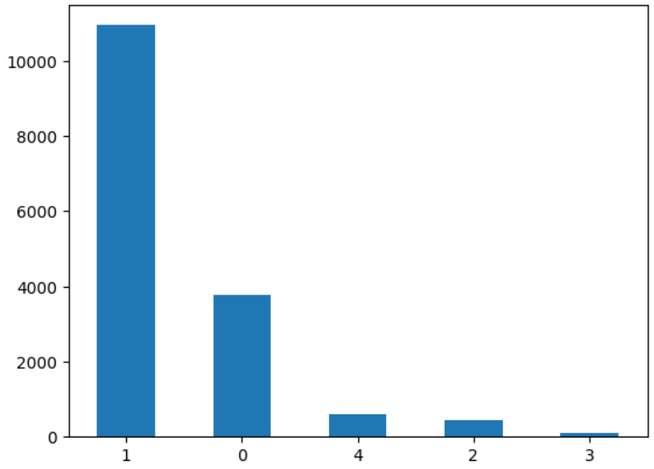
대분류 기준으로 본 평균 이동

데이터 분석 결과

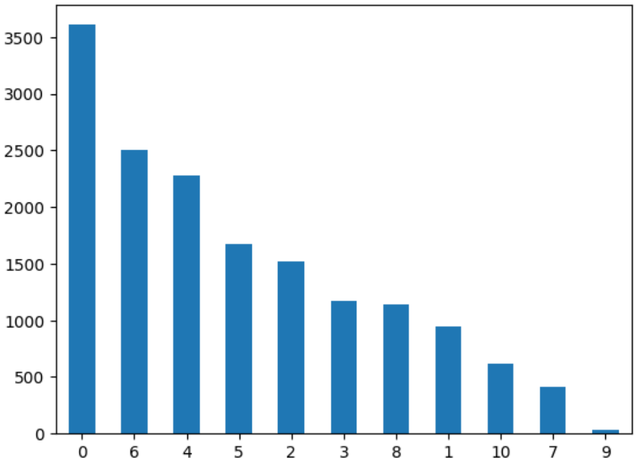
데이터 불균형

- 범주 데이터들의 분포를 파악
- 확인 결과 범주 데이터들이 불균형하게 분포

- 모델 학습에 사용할 학습 데이터와 평가 데이터를 분리할 때 불균형성을 고려하며 분리



대분류 개수 분포

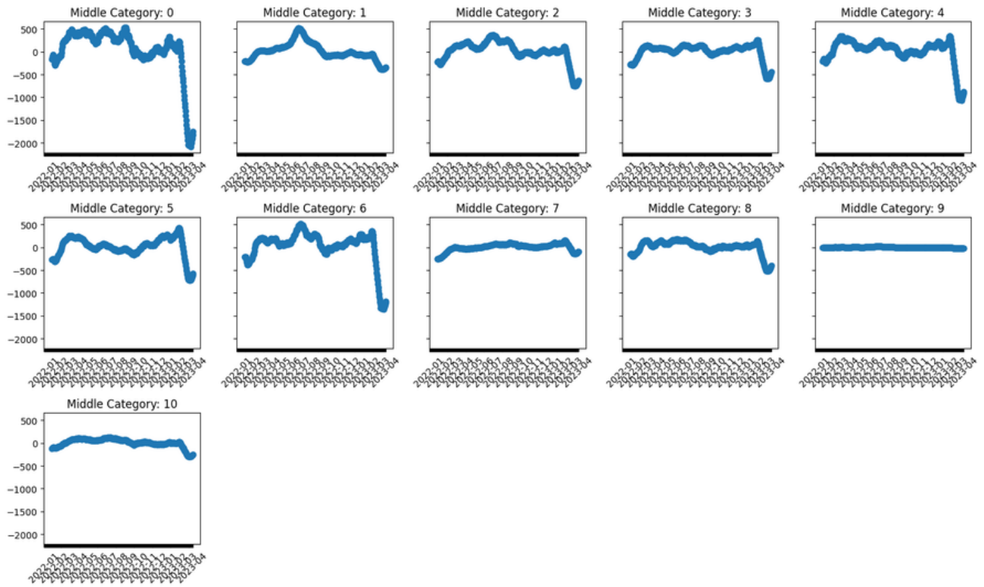


중분류 개수 분포

데이터 분석 결과

시계열 분석

- 범주 데이터 별 평균 이동 시각화로 분석
- 범주 데이터들 중에서 비슷한 양상(변화량)을 보인 데이터들이 존재했으나, 약간의 차이도 분명 존재했기 때문에 모든 범주 데이터를 사용하기로 결정



중분류 시계열 분석

데이터 전처리

결측치 처리

결측치 확인

- 결론부터 말하면 brand_keyword_cnt.csv 파일은 Best 모델에 사용하지 않지만, 여러 모델들을 실험하기 위해 해당 데이터를 전처리

- 키워드 언급량에 결측치가 존재하는 브랜드 제품 확인
- 확인 결과, 35개의 브랜드에 결측치가 존재

```
# 결측치가 있는 행의 인덱스 찾기
missing_indices = df[df.isnull().any(axis=1)].index

missing_indices
```

✓ 0.0s

```
Index([  95,  246,  250,  303,  385,  440,  444,  466,  515,  647,  765,  811,
       1105, 1162, 1398, 1486, 1518, 1588, 1706, 1893, 1980, 1999, 2117, 2125,
       2298, 2328, 2349, 2430, 2471, 2495, 2529, 2711, 2855, 3142, 3149],
      dtype='int64')
```

결측치가 존재하는 브랜드 키워드 언급량의 index

결측치 처리

결측치 평균

결측치 대체값을 위해 해당 브랜드 키워드 언급량을
일별로 평균값을 구함

```
[4.150099159543218,
 4.353330219486435,
 5.4342538641511045,
 5.3508732162135635,
 5.489068538548581,
 5.497235631896844,
 5.121293894101893,
 4.25357116967571,
 4.620276290919558,
 6.066850589747004,
 5.838891149421135,
 5.54786084010347,
 5.346266585368138,
 5.271491355290851,
 4.308546669059622,
 4.585508526620189,
 6.147032044102208,
 6.255355432233124,
```

brand_keyword_cnt 평균값

결측치 처리

결측치 대체

- 데이터 분석에서 구한 시각화 비교를 바탕으로 결측치 대체
- 변화량이 유의미한 결측치에는 평균값
- 변화가 무의미한 결측치는 0
- `brand_keyword_cnt_preprocess.csv` 파일로 저장

```
# 결측치가 있는 행의 인덱스 찾기
missing_indices = df[df.isnull().any(axis=1)].index

# 결측치가 있는 행의 데이터를 values_column_0_scal_pos로 채움
for idx in missing_indices:
    brand_value = df.loc[idx, '브랜드']
    if brand_value in data:
        df.loc[idx, '2022-01-01':] = column_sums
    else:
        df.loc[idx, '2022-01-01':] = 0
df.iloc[missing_indices,:].head(35)
```

✓ 0.8s

	브랜드	2022-01-01	2022-01-02	2022-01-03	2022-01-04	2022-01-05	2022-01-06	2022-01-07	2022-01-08	2022-01-09	...	2023-03-26
95	B002-00117	0.000000	0.00000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000
246	B002-00296	0.000000	0.00000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000
250	B002-00302	4.150099	4.35333	5.434254	5.350873	5.489069	5.497236	5.121294	4.253571	4.620276	...	3.860724
303	B002-00366	0.000000	0.00000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000
385	B002-00460	4.150099	4.35333	5.434254	5.350873	5.489069	5.497236	5.121294	4.253571	4.620276	...	3.860724
440	B002-00533	4.150099	4.35333	5.434254	5.350873	5.489069	5.497236	5.121294	4.253571	4.620276	...	3.860724

결측치 평균값과 0값으로 대체

데이터 통합

평균값 계산

- sales.csv 파일에만 해당
- sales 데이터의 총합과 train의 데이터의 총합을 제품별로 단가 유추

```
[ ]  1 # 각 제품의 가격 추정
     2 estimated_prices = []
     3 for idx, row in sales_data.iterrows():
     4     corresponding_row = train_data.loc[idx]
     5     total_price = sum(row[1:])
     6     total_quantity = sum(corresponding_row[train_data.columns[1:]])
     7     if total_quantity == 0:
     8         estimated_prices.append(0)
     9     else:
    10         estimated_prices.append(total_price // total_quantity)
    11
```

```
 1 print(estimated_prices)
 2 rounded_prices = [round(price, -2) for price in estimated_prices]
 3 print(rounded_prices)
 4 # 추정된 가격을 train_data에 추가
 5 train_data.insert(1, '가격', rounded_prices)
```

```
[5488, 24338, 11750, 4005, 5092, 7650, 7205, 12040, 4787, 14650, 20416, 3893, 64031, 10809,
[5500, 24300, 11800, 4000, 5100, 7600, 7200, 12000, 4800, 14600, 20400, 3900, 64000, 10800,
```

제품 각각의 평균가격 반올림

데이터 통합

Feature 추가

- 유추한 단가를 100자리수로 반올림
- train.csv 파일에 새로운 열로 추가
- 열 이름: ‘가격’
- `new_train_round.csv` 파일로 저장

- brand_keyword_cnt와 마찬가지로 Best 모델에는 사용하지 않지만 성능 테스트를 위해 사용

	가격	대분류	중분류	소분류	브랜드	2022-01-01	2022-01-02	2022-
0	5500	B002-C001-0002	B002-C002-0007	B002-C003-0038	B002-00001	0	0	
1	24300	B002-C001-0003	B002-C002-0008	B002-C003-0044	B002-00002	0	0	
2	11800	B002-C001-0003	B002-C002-0008	B002-C003-0044	B002-00002	0	0	
3	4000	B002-C001-0003	B002-C002-0008	B002-C003-0044	B002-00002	0	0	
4	5100	B002-C001-0001	B002-C002-0001	B002-C003-0003	B002-00003	0	0	

train 데이터 가격 열 추가

데이터 스케일링

스케일링

- MinMax, Standard, Robust, Normalizer 등 다양한 Scaler 비교 분석
- 학습 성능 및 최적화로 평가한 결과 Min Max Scaler가 가장 적합했음
- 행 기준으로 scaling
- `train_data.csv`의 시계열 데이터 스케일링

```
# 시계열 데이터 스케일링
sc_main = MinMaxScaler()

train_data.iloc[:, cat_size:] = sc_main.fit_transform(train_data.iloc[:,cat_size:]
```

모델링 및 학습

모델 정의

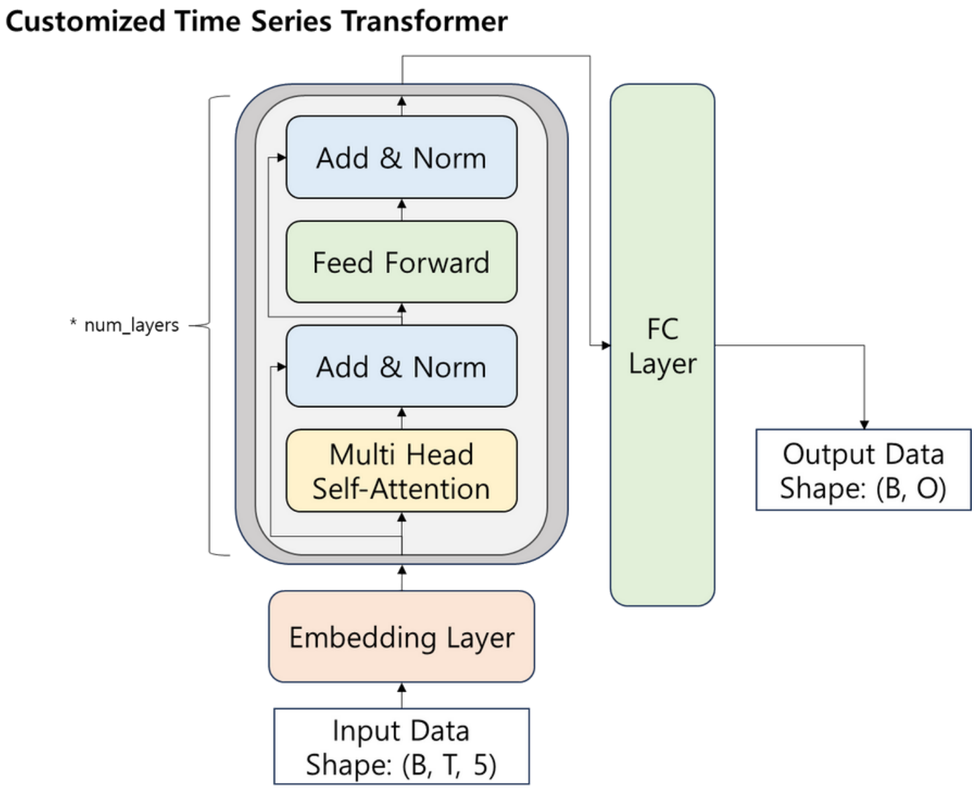
모델 선택 배경

배경

- 여러 시계열 모델을 바탕으로 실험적으로 평가
- 성능이 가장 괜찮았던 **Time Series Transformer** 선택
- 정확히는 Time Series Transformer를 사용한 것이 아닌 참고해서 Customize 해서 사용

평가 비교 모델

- LSTM (Baseline)
- Time Series Transformer (Customized)
- Informer (빠른 Attention 학습과 좋은 LSTF 성능)
- DLinear (간단한 linear층만 이용)



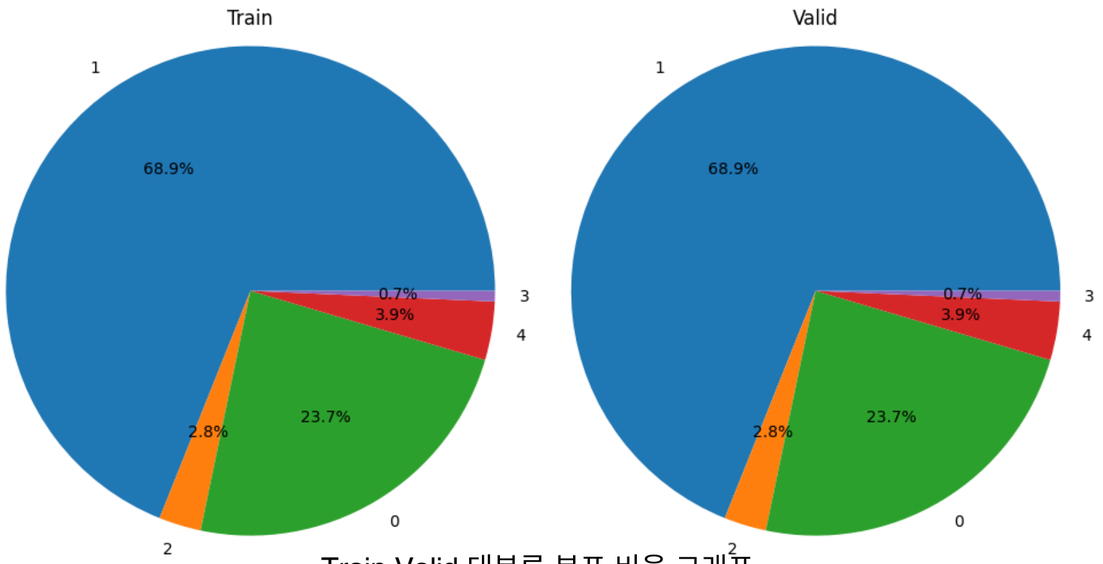
모델 데이터

구축 Pipeline

사용 파일:
`train.csv`,
`new_train_round.csv` (평가),
`brand_keyword_cnt_preprocess. csv` (평가)

데이터 구축 Process:

1. 시계열 데이터만 MinMax 스케일링
2. 범주 데이터 LabelEncoding
3. Window 사이즈 만큼 Stride(STEP_SIZE) 주기로 Input, Target 데이터 생성
4. 생성한 학습 데이터를 StratifiedKFold를 활용해 범주 데이터가 균등하게 분포하도록 학습 데이터와 평가 데이터로 분리(Validation 비율 : 0.2)



Train Valid 대분류 분포 비율 그래프

모델 데이터

모델 입출력 정의

Input 데이터 Shape : (Batch, Train Size, 5)
Target 데이터 Shape: (Batch, Predict Size)

Input Data Feature (3D Tensor)
- Index 0~3 : 인코딩된 범주데이터
  - [대분류, 중분류, 소분류, 브랜드] 순
- Index 4 : 스케일링된 시계열 데이터
  - [판매량]
- 총합 5개의 특징데이터를 가진 입력 데이터

Target Data Feature (2D Tensor)
- predict days의 스케일링된 판매량 데이터 (정답 data)

```
1 print(train_input.shape, train_target.shape,val_input.shape, val_targ
```

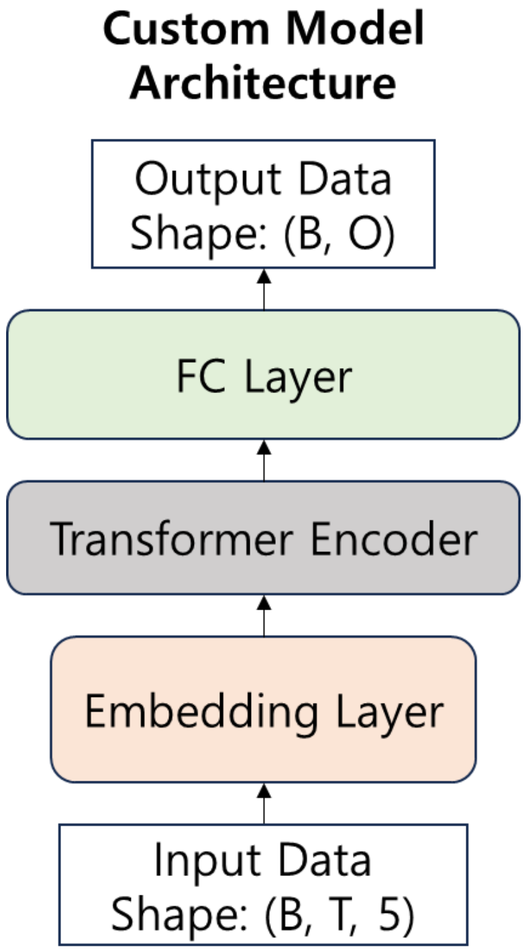
```
(572040, 126, 5) (572040, 21) (143010, 126, 5) (143010, 21)
```

모델

모델 구조

크게 3개의 Part로 나뉨

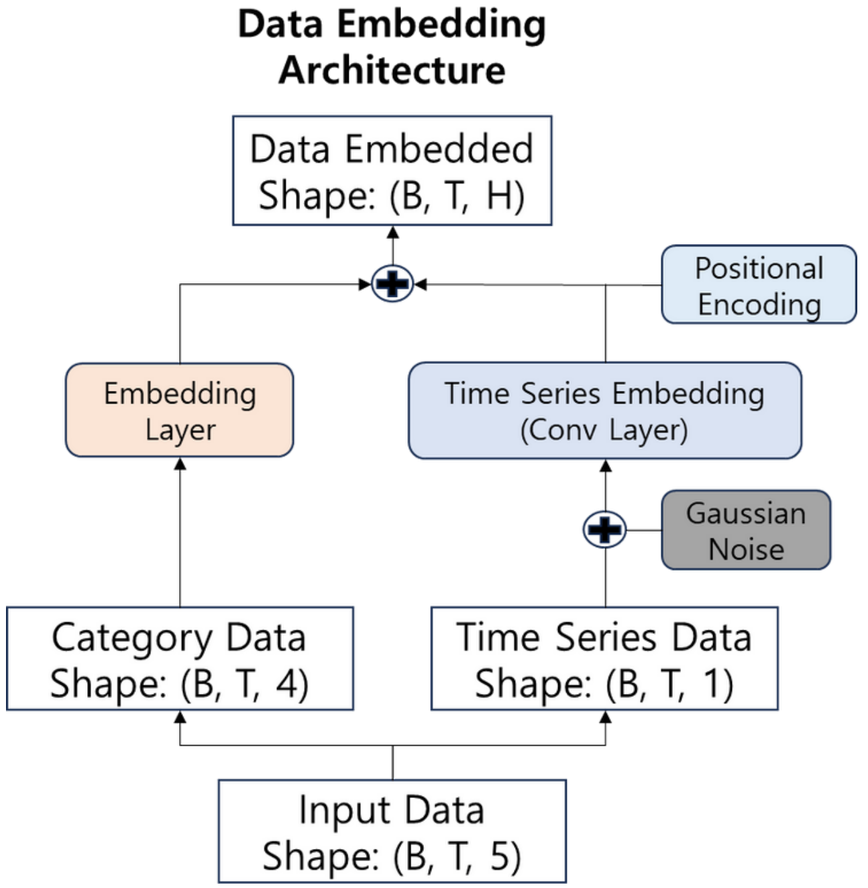
1. Data Embedding Layer
2. Transformer Encoder
3. Fully-Connected Layer



모델

임베딩 레이어

1. Category와 Time Series 데이터로 분리
2. Category Data는 Embedding Layer를 거쳐 H(hidden size) 차원만큼 매핑
3. Time Series Data는 먼저 학습 시에만 가우시안 노이즈를 추가해서 과적합 방지
4. 그 후에 ConvLayer를 통해 H차원만큼 임베딩
5. Positional Encoding을 통해 위치 정보 추가 (sin, cos)
6. 범주 데이터, 시계열 데이터 모두 더하고 Dropout 적용

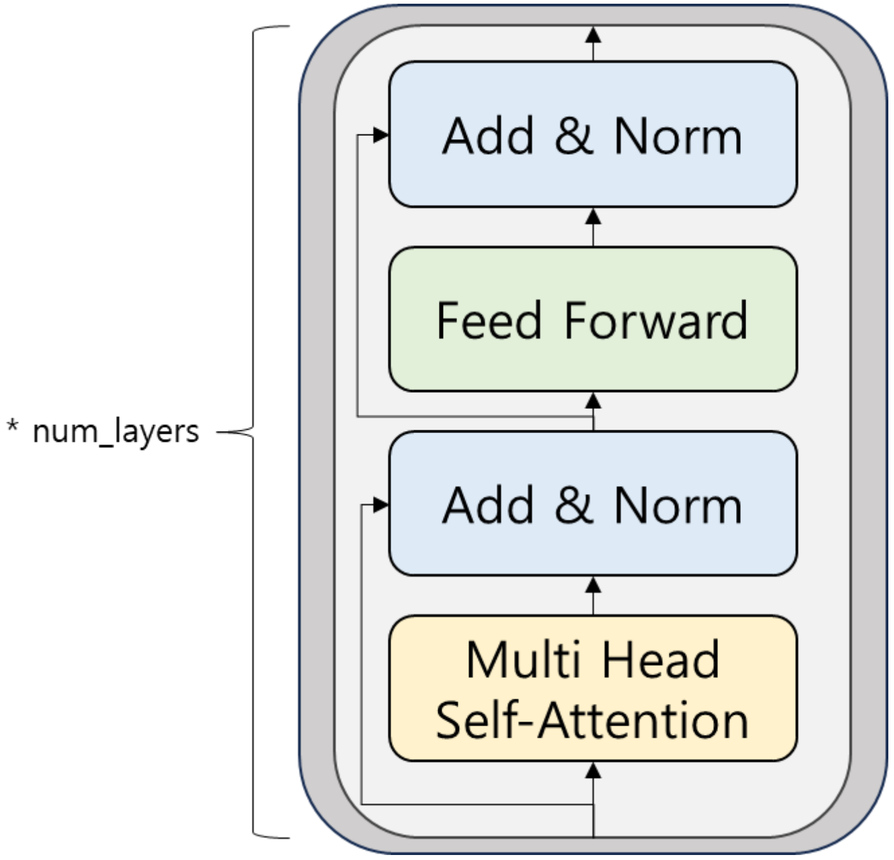


모델

트랜스포머 + FC

Transformer의 Encoder 파트만 가져와서 사용

1. Data Embedding Layer를 거친 Data가 Input으로 시작
2. Norm은 Layer Normalization 사용
3. Activation Fuction은 ReLU 사용
4. Transformer Encoder를 거친 Output을 FC Layer를 거쳐 최종 Output 생성
5. FC Layer에는 Activation Function 사용 X, 단순 projection



학습 설정

최적화

Loss Function 및 Optimizer 정보

- Loss: MSE
- Optimizer: AdamW
- Scheduler: MultiplicativeLR
- Gradient Clipping
- 최적화가 잘 이루어지는 지를 확인하기 위해 HuberLoss와 PSFA Loss도 확인 (학습에 사용 안함)

```
num_epochs = CFG.EPOCHS

optimizer = optim.AdamW(model.parameters(), lr=CFG.LEARNING_RATE)

total_steps = len(train_loader) * num_epochs

criterion = nn.MSELoss()
huberloss = nn.HuberLoss()

```

학습 설정

하이퍼 파라미터

- train window size: 126
- epochs: 10
- learning rate: 1e-4
- (transformer) num heads: 4
- hidden size: 512
- (transformer) num layers: 4

```
class CFG:
    TRAIN_WINDOW_SIZE = 126
    PREDICT_SIZE = 21
    EPOCHS = 10
    LEARNING_RATE = 1e-4
    BATCH_SIZE = 256
    SEED = 41
```

학습 파라미터 및 Config

```
# Hyper-parameters
num_time_steps = CFG.TRAIN_WINDOW_SIZE
num_heads = 4
hidden_size = 512
num_layers = 4
num_output_days = CFG.PREDICT_SIZE
```

모델 파라미터

모델 평가

Window Size

- Window Size 에 따른 Valiation 결과를 비교
- MSE Loss와 PSFA의 값을 비교하여 좋은 window size를 탐색
- 실험 결과 126이 가장 성능이 좋았음

Window Size	MSE	Validation PSFA
96	0.0188	0.5645
126	0.0179	0.5749
192	0.0185	0.5689
336	0.0201	0.5621

모델 평가

학습 데이터

- 여러 학습 데이터를 모델에 대입해서 학습 후 평가
- 가격의 경우 5개의 범주 데이터와 1개의 시계열 데이터
- 가격 + 키워드의 경우 5개의 범주 데이터와 2개의 시계열 데이터
- MSE Loss는 비슷했으나 Valid PSFA는 기본 모델이 가장 높았음

학습 데이터	MSE	Valid PSFA
기본(train.csv)	0.0179	0.5749
가격(new_train_round.csv)	0.0175	0.5549
가격 + 키워드 (brand_keyword_cnt.csv)	0.0179	0.5569

결론

- 시계열 데이터 특성 상 이상치에 영향을 덜 주기 위해서 모델 일반화에 초점을 두고 학습 전략을 설계
- 여러 모델을 시도해서 가장 성능이 높았던 Time Series Transformer 모델 선택
- 다양한 데이터 입력을 넣었을 때, Basic하게 넣는 방법의 성능이 가장 높았음
- Window Size 역시 126이 가장 성능이 높았음
- 최종 모델
  - Time Series Transformer
  - Data: `train.csv`
  - Input Shape: [B, T, 5]
  - Output Shape: [B, O]
  - Public Score: 0.56224
  - Private Score: 0.55444

참고자료

Paper:

- Wen, Q., Zhou, T., Zhang, C., Chen, W., Ma, Z., Yan, J., & Sun, L. (2022). Transformers in Time Series: A Survey. International Joint Conference on Artificial Intelligence.
- Zhou, H., Zhang, S., Peng, J., Zhang, S., Li, J., Xiong, H., & Zhang, W. (2020). Informer: Beyond Efficient Transformer for Long Sequence Time-Series Forecasting. AAAI Conference on Artificial Intelligence.
- Zhang, K., Zuo, W., Chen, Y., Meng, D., & Zhang, L. (2016). Beyond a Gaussian Denoiser: Residual Learning of Deep CNN for Image Denoising. *IEEE Transactions on Image Processing, 26*, 3142-3155.

GitHub:

- TST Review: https://github.com/qingsongedu/time-series-transformers-review
- Dlinear: https://github.com/cure-lab/LTSF-Linear
- Informer: https://github.com/zhouhaoyi/Informer2020