

## 온라인 채널 제품 판매량 예측 AI 온라인 해커톤

작전명 성장

정수미(정파도), 이선택(selectionlee), 소지연(으엥)

---

## 목차

- 01 프로젝트 개요
- 02 데이터셋 소개
- 03 Baseline code 검토 및 수정
- 04 메타데이터 활용
- 05 모델링
- 06 결과
- 07 개발환경

## 프로젝트 개요



온라인 판매 채널에서 수집되는 대규모 데이터는 정확한 판매 예측을 수행하여  
효율적인 재고 관리와 타겟 마케팅 전략을 세우는 것이 중요함

특정 온라인 쇼핑몰의 일별 제품별 판매 데이터를 바탕으로  
향후 약 3주 간의 제품별 판매량을 예측하는 AI 모델을 개발

## 데이터셋 - 파일

### 1. train.csv

2022-01-01~2023-04-04의 일별 판매량

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

### 2. sample\_submission.csv

제출양식

ID : 실제 판매되고 있는 고유 ID

2023-04-05 ~ 2023-04-25 : 예측한 일별 판매량

## 데이터셋 - 메타데이터

### 1. sales.csv

2022-01-01~2023-04-04의  
일별 판매가격

train.csv와 동일한 양식

### 2. brand\_keyword\_cnt.csv

브랜드의 연관키워드 언급량을 정규화한  
일별 데이터

브랜드 : 브랜드 코드

### 3. product\_info.csv

제품 특성

제품 : 제품 코드

제품특성 : 제품 특성 데이터(Text)



train.csv



sales.csv



brand\_keyword\_cnt.csv



product\_info.csv



sample\_submission.csv

## Baseline code 검토 및 수정

### 1. make\_train\_data 함수 수정

```
def make_train_data(data, train_size=CFG['TRAIN_WINDOW_SIZE'], predict_size=CFG['PREDICT_SIZE']):
    # 'TRAIN_WINDOW_SIZE':90, 'PREDICT_SIZE':21 - 90일치로 학습 + 21일치 예측
    ...
    학습 기간 물력, 예측 기간 물력의 세트로 데이터를 생성
    data : 일별 판매량
    train_size : 학습에 활용할 기간
    predict_size : 추론할 기간
    ...
    num_rows = len(data)
    window_size = train_size + predict_size #111

    1) input_data = np.empty((num_rows + (len(data.columns) - window_size + 1), train_size, len(data.iloc[0, :4]) + 1))
    target_data = np.empty((num_rows + (len(data.columns) - window_size + 1), predict_size))

    for i in tqdm(range(num_rows)):
        encode_info = np.array(data.iloc[i, :4])
        2) sales_data = np.array(data.iloc[i, 4:])

        for j in range(len(sales_data) - window_size + 1):
            window = sales_data[j : j + window_size]
            temp_data = np.column_stack((np.tile(encode_info, (train_size, 1)), window[:train_size]))
            3) input_data[i * (len(data.columns) - window_size + 1) + j] = temp_data
            target_data[i * (len(data.columns) - window_size + 1) + j] = window[train_size:]

    return input_data, target_data
```

1) input\_data로 (15890 \* 353, 90, 5) 크기의 배열이 생성

→ 15890x463 크기의 train\_data가 data로 들어오기 때문에 463-111+1(=353)의 연산이 이루어짐

2) 463개의 행 데이터가 encode\_info와 sales\_data로 나누어 지면서 sales\_data의 크기는 459(=463-4)이 됨

## Baseline code 검토 및 수정

### 1. make\_train\_data 함수 수정

```
def make_train_data(data, train_size=CFG['TRAIN_WINDOW_SIZE'], predict_size=CFG['PREDICT_SIZE']):
    # 'TRAIN_WINDOW_SIZE':90, 'PREDICT_SIZE':21 - 90일치로 학습 + 21일치 예측
    ...
    학습 기간 물력, 예측 기간 물력의 세트로 데이터를 생성
    data : 일별 판매량
    train_size : 학습에 활용할 기간
    predict_size : 추론할 기간
    ...

    num_rows = len(data)
    window_size = train_size + predict_size #111

    1) input_data = np.empty((num_rows * (len(data.columns) - window_size + 1), train_size, len(data.iloc[0, :4]) + 1))
    target_data = np.empty((num_rows * (len(data.columns) - window_size + 1), predict_size))

    for i in tqdm(range(num_rows)):
        encode_info = np.array(data.iloc[i, :4])
        2) sales_data = np.array(data.iloc[i, 4:])

        for j in range(len(sales_data) - window_size + 1):
            window = sales_data[j : j + window_size]
            temp_data = np.column_stack((np.tile(encode_info, (train_size, 1)), window[:train_size]))
            3) input_data[i * (len(data.columns) - window_size + 1) + j] = temp_data
            target_data[i * (len(data.columns) - window_size + 1) + j] = window[train_size:]

    return input_data, target_data
```

3) (15890\*353, 90, 5) 크기의 input\_data에서  
실질적인 데이터 저장은 (15980\*349, 90, 5)  
만큼 이루어짐

→  $\text{len}(\text{sales\_data}) - \text{window\_size} + 1$   
즉,  $349 (= 459 - 111 + 1)$ 의 반복문을 돌기 때문

→ 0~348, 353~701, 706~1054, ... 에만 실질적인  
학습데이터 저장되면서,  
train set과 validation set를 구성하는 데 빈 데이터  
가 같이 사용하고 있음을 파악

\*target\_data도 동일한 방식으로 배열 생성 및 저장이  
이루어지고 있음

## Baseline code 검토 및 수정

### 1. make\_train\_data 함수 수정

```
[ ] def make_train_data(data, train_size=CFG['TRAIN_WINDOW_SIZE'], predict_size=CFG['PREDICT_SIZE']):
    ...
    학습 기간 물력, 예측 기간 물력의 세트로 데이터를 생성
    data : 알뜰 판매장
    train_size : 학습에 활용할 기간
    predict_size : 추론할 기간
    ...

    num_rows = len(data)
    window_size = train_size + predict_size
    # 대분류, 중분류, 소분류, 브랜드 column 제외
    input_data = np.empty((num_rows * ((len(data.columns)-4) - window_size + 1), train_size, len(data.iloc[0, :4]) + 1))
    target_data = np.empty((num_rows * ((len(data.columns)-4) - window_size + 1), predict_size))
    # 데이터를 저장하는 인덱스 변수 생성
    idx = 0
    for i in tqdm(range(num_rows)):
        encode_info = np.array(data.iloc[i, :4])
        sales_data = np.array(data.iloc[i, 4:])


        for j in range(len(sales_data) - window_size + 1):
            window = sales_data[j : j + window_size]
            temp_data = np.column_stack((np.tile(encode_info, (train_size, 1)), window[:train_size]))
            # 하나씩 증가시키며 빈 데이터 없이 순서대로 input data와 target data를 구성하도록 함
            input_data[idx] = temp_data
            target_data[idx] = window[train_size:]
            idx += 1
    print(idx)
    return input_data, target_data
```

- input\_data의 생성할 때, encode\_info의 크기(=4)만큼 줄어든 배열을 생성
- idx라는 변수를 두어 empty한 인덱스 없이 배열에 학습할 데이터를 저장



## 메타 데이터 활용

### 1. brand\_keyword\_cnt.csv 활용

- 1) brand\_keyword\_cnt.csv를 읽어들이어, 각 브랜드별로 스케일링을 진행
    - 브랜드별로 언급량의 큰 차이가 보였기에, 이 과정을 거치지 않으면 어떤 브랜드는 0의 가중치를 받지만 다른 브랜드는 100이 넘는 가중치를 받을 수 있기 때문
    - 이를 통해, 해당 브랜드가 언급량이 많았던 시기와 적었던 시기를 수치적으로 구분할 수 있을 것이라 생각
  - 2) 가중치의 값이 0.3~0.5 사이의 값으로 조정되도록 시그모이드를 이용한 함수를 정의하여 각 브랜드 및 날짜별 가중치를 구함
  - 3) 가중치 배열과 train.csv의 브랜드를 label encoding 후 인코딩값으로 브랜드를 찾아 가중치값을 곱함
    - train.csv와 brand\_keyword\_cnt.csv 두 파일 모두 브랜드코드는 오름차순으로 정렬되어있기 때문
-  언급량 가중치가 적용된 판매량을 토대로 데이터를 구성하여, 어느정도 언급량을 감안한 판매량을 예측할 것이라고 생각

## 메타 데이터 활용

### 2. sales.csv 활용

- 1) train.csv와 sales.csv 데이터를 바탕으로 2023-01-01~2023-04-04 간 제품의 평균 판매금액을 구함
  - 2) 금액이 100원 이상인 제품 한에서만 (가격이 두자리수인 제품은 너무 싸다고(PMC 지점) 생각하여 가중치를 0으로 둠) 판매금액 역수로 가중치를 만듦
- 판매금액 역수로 가중치를 준 이유는 일반적으로 생각해봤을 때, 가격이 저렴할수록 사람들이 많이 구매한다고 생각하였기 때문

## 메타 데이터 활용

### 3. 가중치 반영

```
# train data에 가중치를 줌
for i in tqdm(range(len(train_data))):
    for j in range(4, len(train_data.columns)):
        mul = sales_weights[i] + sigmoid_df.loc[matching[i], sigmoid_df.columns[j-4]]
        w = train_data.loc[i, train_data.columns[j]] * mul
        train_data.loc[i, train_data.columns[j]] = round(train_data.loc[i, train_data.columns[j]] + w)
```

 브랜드 언급량 가중치와 판매 금액 가중치를 더한 후, 각 데이터마다 가중치들을 곱함

## 모델링

### 1. 모델 선정

#### LSTM

```
class LSTMModel(nn.Module): # LSTM Model
    def __init__(self, input_size=5, hidden_size=512, output_size=CFG['PREDICT_SIZE']):
        super(LSTMModel, self).__init__()
        self.hidden_size = hidden_size
        self.lstm = nn.LSTM(input_size, hidden_size, batch_first=True)
        self.layer_norm = nn.LayerNorm(hidden_size)
        self.fc = nn.Sequential(
            nn.Linear(hidden_size, hidden_size//2),
            nn.ReLU(),
            nn.Dropout(),
            nn.Linear(hidden_size//2, output_size)
        )

    def forward(self, x):
        lstm_out, _ = self.lstm(x)
        output = self.fc(lstm_out[:, -1, :])
        return output
```

- input size=5, hidden size=512,  
output size = 21

- 활성화함수로 ReLU 사용
- dropout을 통한 정규화

## 모델링

### 1. 모델 선정

#### GRU

```
class GRUModel(nn.Module): # GRU Model
    def __init__(self, input_size=5, hidden_size=512, output_size=CFG['PREDICT_SIZE']):
        super(GRUModel, self).__init__()
        self.hidden_size = hidden_size
        self.gru = nn.GRU(input_size, hidden_size, batch_first=True)
        self.fc = nn.Sequential(
            nn.Linear(hidden_size, hidden_size//2),
            nn.ReLU(),
            nn.Dropout(),
            nn.Linear(hidden_size//2, output_size)
        )

    def forward(self, x):
        gru_out, _ = self.gru(x)
        output = self.fc(gru_out[:, -1, :])
        return output
```

- input size=5, hidden size=512,  
output size = 21

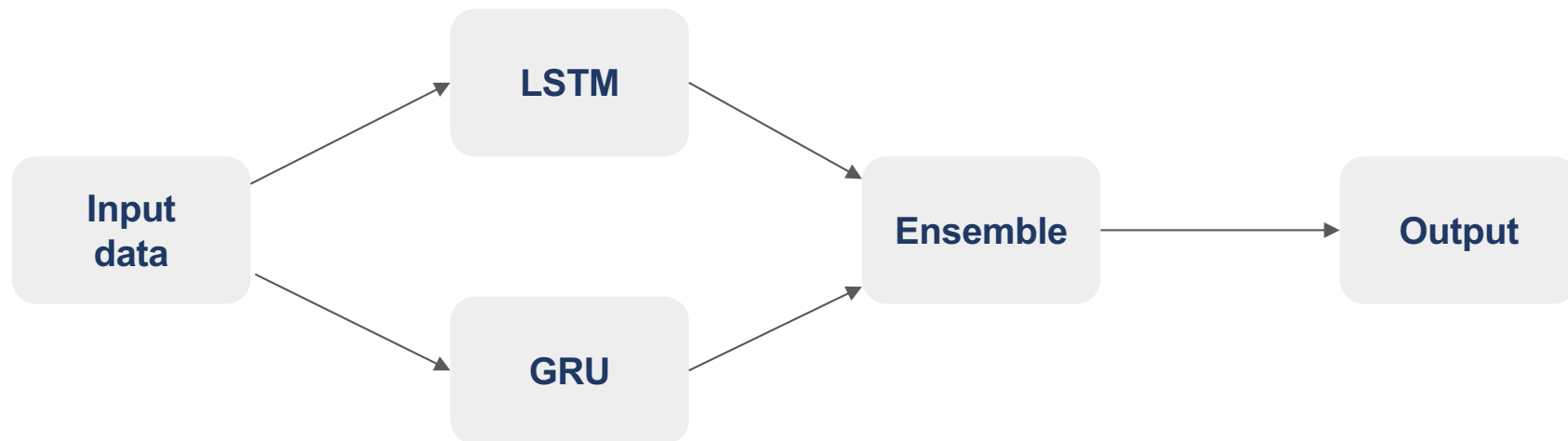
- 활성화함수로 ReLU 사용  
- dropout을 통한 정규화

## 모델링

### 2. 앙상블

#### LSTM, GRU Model Stacking Ensemble

➡ 단일 모델로만 학습을 진행했을 때 성능이 높지 않다고 판단하여 LSTM, GRU 두 모델을 stacking ensemble



## 모델링

### 3. 하이퍼 파라미터 조정

- 학습 시 val loss가 갑자기 커지는 구간이 존재하여 val loss가 커지기 전까지 epoch을 조정하여 학습을 진행
- 최적의 하이퍼 파라미터를 찾기 위해 optuna, grid search를 진행하였으나, RAM 용량 문제로 인해 따로 하이퍼 파라미터를 조정하지 않음

\*epoch 제외 baseline에서 주어진 하이퍼 파라미터 그대로 진행

 epoch=7일 때 리더보드 점수가 가장 높은 것을 확인

## 결과

앙상블 모델과 메타 데이터를 이용한 가중치 적용 후 private score 0.55 이상의 성능을 보임

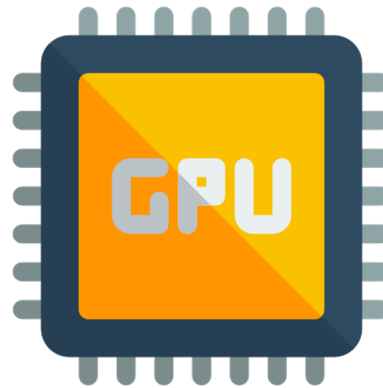
방법	score
기본 baseline	0.49294
LSTM + GRU Stacking Ensemble	0.53502
브랜드 언급량 가중치 반영	0.53844
판매금액 가중치 반영	0.54904
최종	0.56304(public) 0.55397(private)



개발 환경



Google Colab



GPU Nvidia Tesla T4 사용  
(colab 기본 제공)



python 3.10.12

# 감사합니다

작전명 성장

정수미(정파도), 이선택(selectionlee), 소지연(으엥)