통계학특강 Final Report

2015150017 최성웅

LSTM과 LightGBM을 사용한 판매량 예측 및 모델 성능 비교

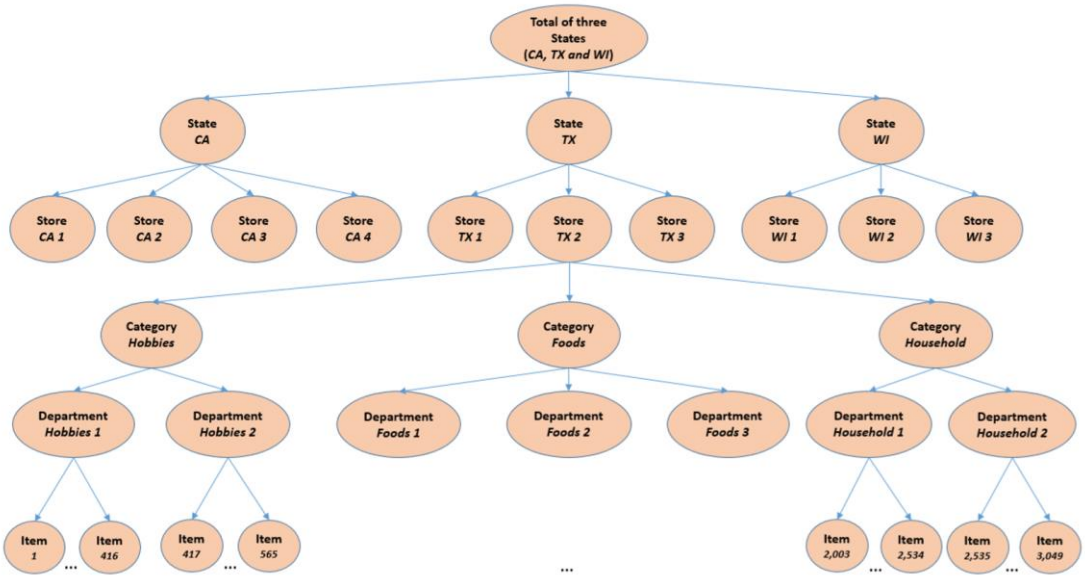
- 현재 진행중($10,000이상)인 Kaggle Competition –

1. Data Description

주어진 데이터를 활용하여 미래를 예측하는 것은 어느 분야에서든 중요하고 매력적인 일이다. 대규모 데이터의 확보와 분석 기법의 발전을 통한 예측이 다방면에서 진행되고 있으며, 작성자도 본 보고서를 통해서 **시계열 데이터 기반 예측을** 진행하였다. **LSTM**과 **LightGBM** 모형을 각각 사용하였고 최종 제출물의 정확도를 비교하였다.

Kaggle에서 진행 중인 **'M5 Forecasting – Accuracy' Competition**의 데이터를 분석에 사용하였으며 링크는 다음과 같다. (https://www.kaggle.com/c/m5-forecasting-accuracy)

본 데이터는 미국 세 개 주(California (CA), Texas (TX), and Wisconsin (WI))에 있는 월마트 총 10개 지점에서, 2011년 1월부터 2016년 6월까지 총 1913일 간 판매된 상품의 판매량에 대한 것이다. 각 상품은 세 개의 카테고리(Hobbies, Foods, Household)로 나뉘어져 있고, 카테고리는 다시 7개의 department(Hobbies_1 ~ Household_2)로 분화되어 있다. 총 상품의 개수는 3049개이고 모든 상품에 대한 **이후 28일 간의 판매량을 예측**하는 것이 목표이다. 데이터의 대략적인 구조를 다음과 같이 나타낼 수 있다.



분석에 사용되는 dataset은 크게 세 가지이며, 첫 번째는 **'calendar'**이다.

```
calendar.head()
```

	date	wm_yr_wk	weekday	wday	month	year	d	event_name_1	event_type_1	event_name_2	event_type_2	snap_CA	snap_TX	snap_WI
0	2011-01-29	11101	Saturday	1	1	2011	d_1	NaN	NaN	NaN	NaN	0	0	0
1	2011-01-30	11101	Sunday	2	1	2011	d_2	NaN	NaN	NaN	NaN	0	0	0
2	2011-01-31	11101	Monday	3	1	2011	d_3	NaN	NaN	NaN	NaN	0	0	0
3	2011-02-01	11101	Tuesday	4	2	2011	d_4	NaN	NaN	NaN	NaN	1	1	0
4	2011-02-02	11101	Wednesday	5	2	2011	d_5	NaN	NaN	NaN	NaN	1	0	1

'calendar'는 판매 기간 동안의 각 날짜에 대한 **연, 월, 일 정보**와 해당 날짜의 판매량에 영향을 미치는 **'event' 여부 및 종류**, 그리고 저소득 계층을 위한 할인 행사를 뜻하는 **'snap'에 대한 정보**를 담고 있다.

두 번째는 **'training sales data'**이다.

```
sales_train_validation.head(10)
```

	id	item_id	dept_id	cat_id	store_id	state_id	d_1	d_2	d_3	d_4	d_5	d_6	d_7	d_8	d_9	d_10
0	HOBBIES_1_001_CA_1_validation	HOBBIES_1_001	HOBBIES_1	HOBBIES	CA_1	CA	0	0	0	0	0	0	0	0	0	0
1	HOBBIES_1_002_CA_1_validation	HOBBIES_1_002	HOBBIES_1	HOBBIES	CA_1	CA	0	0	0	0	0	0	0	0	0	0
2	HOBBIES_1_003_CA_1_validation	HOBBIES_1_003	HOBBIES_1	HOBBIES	CA_1	CA	0	0	0	0	0	0	0	0	0	0
3	HOBBIES_1_004_CA_1_validation	HOBBIES_1_004	HOBBIES_1	HOBBIES	CA_1	CA	0	0	0	0	0	0	0	0	0	0
4	HOBBIES_1_005_CA_1_validation	HOBBIES_1_005	HOBBIES_1	HOBBIES	CA_1	CA	0	0	0	0	0	0	0	0	0	0
5	HOBBIES_1_006_CA_1_validation	HOBBIES_1_006	HOBBIES_1	HOBBIES	CA_1	CA	0	0	0	0	0	0	0	0	0	0
6	HOBBIES_1_007_CA_1_validation	HOBBIES_1_007	HOBBIES_1	HOBBIES	CA_1	CA	0	0	0	0	0	0	0	0	0	0
7	HOBBIES_1_008_CA_1_validation	HOBBIES_1_008	HOBBIES_1	HOBBIES	CA_1	CA	12	15	0	0	0	4	6	5	7	0
8	HOBBIES_1_009_CA_1_validation	HOBBIES_1_009	HOBBIES_1	HOBBIES	CA_1	CA	2	0	7	3	0	2	3	9	0	0
9	HOBBIES_1_010_CA_1_validation	HOBBIES_1_010	HOBBIES_1	HOBBIES	CA_1	CA	0	0	1	0	0	0	0	0	0	0

'training sales data'는 **상품명과 department id, category, store, state에 대한 정보**와 상품마다의 **일별 판매량 정보**를 담고 있다.

세 번째는 **'sales prices'**이다.

```
sell_prices.head()
```

	store_id	item_id	wm_yr_wk	sell_price
0	CA_1	HOBBIES_1_001	11325	9.58
1	CA_1	HOBBIES_1_001	11326	9.58
2	CA_1	HOBBIES_1_001	11327	8.26
3	CA_1	HOBBIES_1_001	11328	8.26
4	CA_1	HOBBIES_1_001	11329	8.26

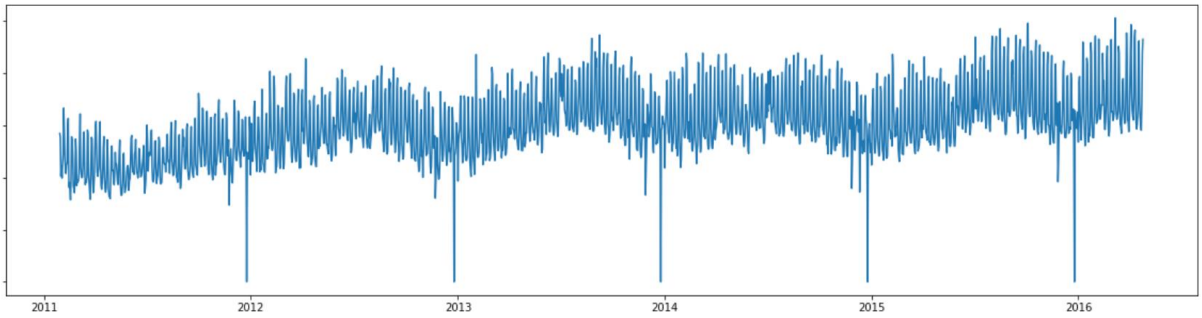
'sales prices'는 상품별 **주간 가격 정보**를 담고 있다.

판매량을 예측함에 있어 판매 기간 동안의 일별 정보뿐만 아니라 상품이 속한 카테고리, 지점, 주 등의 다양한 계층적 정보를 사용해야 하며 해당 정보들이 서로 다른 dataset에 담겨 있는 만큼 전처리 및 시각화를 통해 데이터에 대한 전반적인 insight를 도출하는 것이 중요하다고 판단하였다.

2. EDA

판매량의 state, store별 차이와 Monthly, Weekly seasonality 등을 확인하기 위해 다방면으로 EDA를 진행하였다. 분량상의 문제로 모든 시각화 결과를 첨부할 수 없으므로 중요한 부분에 대한 몇 가지 결과를 보고서에 담았다.

- 판매량 총합



상품 전체 판매량을 일별로 더하여 시간에 따라 그린 그래프이다. 판매량의 **추세가 시간에 따라 증가함**을 알 수 있고, 월마트의 유일한 휴일인 매년 크리스마스에는 판매량이 0인 것을 확인할 수 있다.

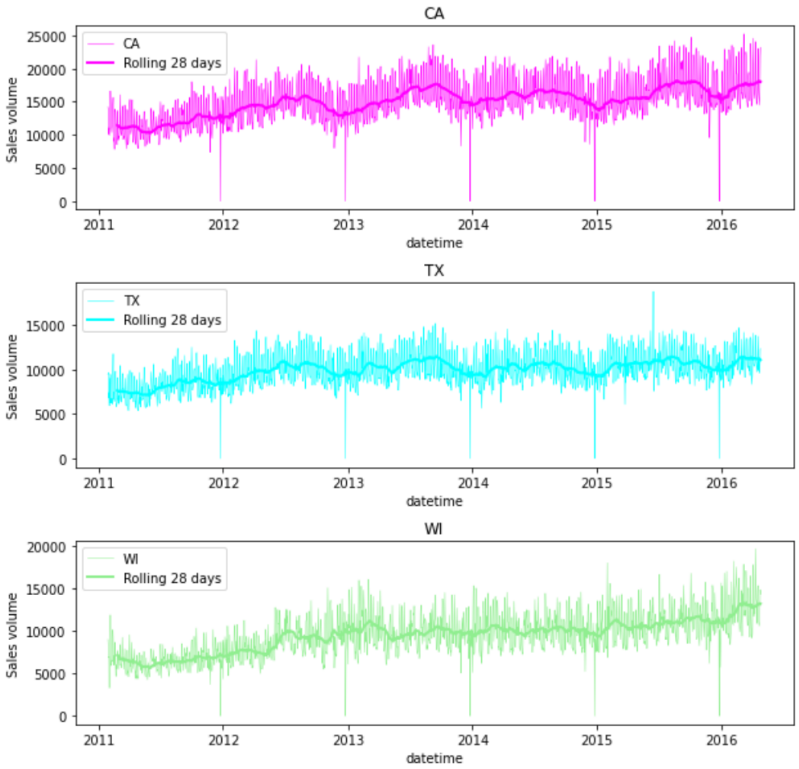
- 지점별 상품 판매가격

```
pd.DataFrame(data=price_df.groupby("store_id").sell_price.mean().round(3)).T
```

store_id	CA_1	CA_2	CA_3	CA_4	TX_1	TX_2	TX_3	WI_1	WI_2	WI_3
sell_price	4.414	4.446	4.388	4.422	4.375	4.367	4.387	4.458	4.442	4.418

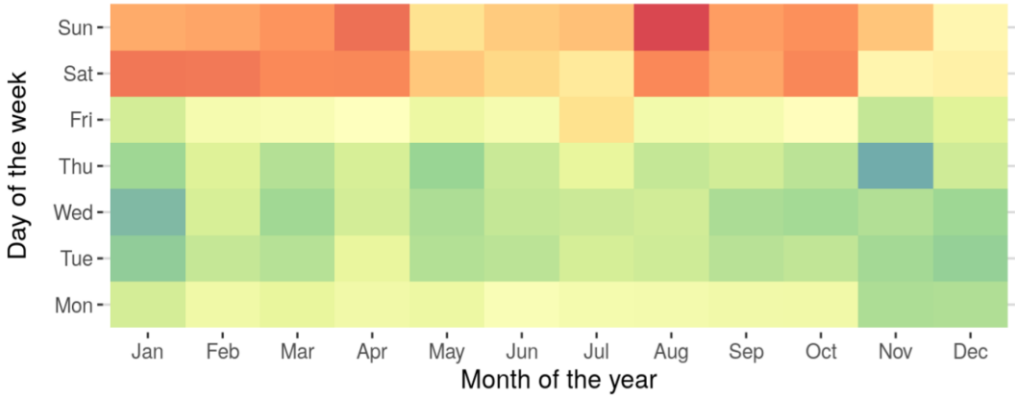
지점에 따른 평균 상품 판매 가격을 살펴본 결과, **유의미한 차이가 없는 것**으로 보였다. 실제로 이후 분석을 진행함에 있어 판매 가격을 고려하지 않는 경우의 정확도와 고려한 경우의 정확도에 큰 차이가 없는 것을 확인할 수 있었다.

- state별 판매량



판매량 총합과 유사하게 시간에 따른 판매량의 증가 추세를 확인할 수 있었다. 그리고 y축을 살펴보았을 때, 판매량 총합이 각 state에 따라 큰 차이를 보이고 있음을 확인할 수 있다. 더 세부적으로 시각화해본 결과, **state 뿐 아니라 store별 판매량 총합 또한 유의미한 차이**를 보이고 있는 것을 알 수 있었다.

- Monthly & Weekly Seasonality



판매량이 많을수록 색이 빨강에 가까워지는 heatmap을 사용하여 월별, 주별 판매량을 확인하였다. 그 결과, **2분기과 4분기의 판매량이 이외의 기간에 비하여 낮고 주말의 판매량이 평일 대비 많은 것**을 확인하였다.

위의 시각화 및 탐색 결과, 예측을 진행함에 있어 판매량의 **주, 월별 주기성 및 분기, 시간에 따른 증가 추세를 고려**해야 한다고 판단하였다. 또한, state와 store에 따라 판매량의 총합이 유의미한 차이를 보이고 있으므로 **상품이 속한 지점을 고려**해야 한다는 결론을 내렸다.

3. LSTM

3-1. Pepropessing

```
sales.head()
```

	0	1	2	3	4	5	6	7	8	9	...	30480	30481	30482	30483	30484	30485	30486	30487	30488	30489
d_1	0	0	0	0	0	0	0	12	2	0	...	0	14	1	0	4	0	0	0	0	0
d_2	0	0	0	0	0	0	0	15	0	0	...	0	11	1	0	4	0	0	6	0	0
d_3	0	0	0	0	0	0	0	0	7	1	...	0	5	1	0	2	2	0	0	0	0
d_4	0	0	0	0	0	0	0	0	3	0	...	0	6	1	0	5	2	0	2	0	0
d_5	0	0	0	0	0	0	0	0	0	0	...	0	5	1	0	2	0	0	2	0	0

5 rows × 30490 columns

LSTM 모델의 input data를 만들기 위해, index를 시간 정보로 둘 수 있도록 'training sales data'를 transpose하고 전처리한 결과이다. index는 총 판매기간인 1913일을 나타내고, 0부터 30489의 컬럼명은 각 상품을 나타낸다. 이후 위에서 언급한 학습 진행 시 고려해야 할 점들을 반영하며 분석을 진행하였다.

- event 정보 포함

판매량을 예측함에 있어 해당 날짜의 event 존재 여부와 그 종류를 고려할 수 있도록 **'before_event'와 'event_type' feature**를 input data에 추가하였다. 'before_event'는 event가 있는 날짜의 전 날일 경우 1의 값을 가지고, 'event_type'은 해당 event의 종류를 숫자 형태로 담고 있다. event_type 변수는 이후 one hot encoding을 통해 가지고 있는 범주의 개수만큼 나누어 주었다. 구체적인 코드는 다음과 같다.

```
for i in range(2, 1969):
    if ((calendar['event_name_1'][i] != 0) | (calendar['event_name_2'][i] != 0)):
        before_event[i-1] = 1
```

```
before_event = pd.DataFrame(before_event, columns = ['before_event'])
```

```
before_event.index = calendar.d
```

```
before_event_train, before_event_test = before_event[:1913], before_event[1913:1941]
```

> 'before event' feature 생성

- 분기, 월, 요일 등의 시간 정보

month	year
10	2011
10	2011
10	2011
10	2011
10	2011

> 연, 월 정보를 담은 feature

quarter	mday
4	5
4	6
4	7
4	5
4	6

> 분기, 요일 정보를 담은 feature

index 날짜에 따라 **연, 월, 분기, 요일 정보** 등을 담은 feature를 생성 후 이전에 생성한 input data에 추가하였다.

3_2. Model fit

```
model.summary()
#model.fit(X_train, y_train, epochs = 32, batch_size = 64)
```

```
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_4 (LSTM)                (None, 14, 64)            7822336
_________________________________________________________________
dropout_4 (Dropout)          (None, 14, 64)            0
_________________________________________________________________
lstm_5 (LSTM)                (None, 14, 128)           98816
_________________________________________________________________
dropout_5 (Dropout)          (None, 14, 128)           0
_________________________________________________________________
lstm_6 (LSTM)                (None, 256)               394240
_________________________________________________________________
dropout_6 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_2 (Dense)              (None, 30490)             7835930
=================================================================
Total params: 16,151,322
Trainable params: 16,151,322
Non-trainable params: 0
_________________________________________________________________
```

세 개의 LSTM layer를 쌓고 각 layer 사이에 0.3 비율의 Dropout layer를 추가하였다. LSTM layer의 노드 수를 64, 128, 256으로 증가시켰고, 총 30490개의 상품에 대한 예측 판매량을 출력해야 하므로 마지막 출력층의 노드는 30490으로 설정하였다.

14일간의 판매량을 Input하여 15일째의 판매량을 예측하는 LSTM 모델을 32 epoch까지 학습한 결과, 마지막 loss는 다음과 같다.

```
loss: 0.1726 - acc: 0.9349

loss: 0.1055 - acc: 0.9828
```

이후 학습된 모델을 통해 1913일 이후 총 28일 간의 판매량을 예측하였다. 1913일부터의 이전 14일간 판매량을 이용하여 1914일째의 판매량을 예측하면, 1914일 예측 판매량을 포함한 이전 14일간의 판매량을 이용하여 1915일의 판매량을 예측하는 방식으로 진행하였다. 구현 코드는 다음과 같다.

```
X_test = []
X_test.append(inputs[0:timesteps])
X_test = np.array(X_test)
pred= []

for j in range(timesteps,timesteps + 28):
    predicted = model.predict(X_test[0,j - timesteps:j].reshape(1, timesteps, 30491))
    testInput = np.column_stack((np.array(predicted, feature_plus[0][1913 + j - timesteps]))
    X_test = np.append(X_test, testInput).reshape(1, j + 1,30491)
    predict = sc.inverse_transform(testInput)[:,0:30490]
    pred.append(predicted_stock_price)
```

코드 중 feature_plus는 앞서 만든 판매량 이외의 정보, 즉 event, 분기, 요일 등의 feature를 결합한 데이터 프레임이다. 모델 실행을 통해 출력되는 28일간의 예측 판매량을 competition의 submission 형식에 따라 만든 후 실제 제출한 결과는 다음과 같다.

Competition 규칙에 따라 RRMSE(regularized root mean squared error)가 낮을수록 결과가 좋은 것인데 크게 좋은 결과를 얻지 못했다. 이는 앞서 예측 진행 시 고려해야 할 점으로 언급했던 state, store 등, 상품이 속한 그룹에 대한 정보를 포함하지 못했기 때문이라고 판단하였다.

- state, store 및 snap, sell_prices 정보 포함

따라서, store_id에 따라 dataset을 총 10개(CA_1 ~ WI_3)로 나눈 후, 위와 동일한 방식으로 학습을 진행해보았다. Dataset을 store별로 나누었으므로 상위 계층인 state의 snap 정보와 store별 주간 상품 평균 판매가를 input data의 feature에 추가할 수 있었다.

snap_WI	sell_price
1.0	3.97
1.0	3.97
0.0	3.97
1.0	4.34
1.0	4.34

> WI_3의 snap 정보와 주간 평균 상품 판매가 정보를 담은 feature 추가

10개의 학습 결과를 다시 submission 형식에 맞춰 하나로 합친 후 실제 제출한 결과는 다음과 같다. 확실히 모든 상품을 한 번에 입력하여 state와 store 정보를 고려하지 못했을 때 보다 나은 결과를 보였다.

4. LightGBM

LSTM 모델 사용시, feature 간의 계층이 존재하기 때문에 예측 시 고려해야 할 정보를 모두 포함하지 못하고 store별로 학습을 진행한 후 결과를 결합해야 하는 번거로움이 존재했다. 상품 간 판매량에서 유의미한 차이가 store보다 더 세부적인, 즉 category나 department 별로 나타났다면 더 많은 횟수의 학습을 진행해야 했을 것이며 그 횟수가 아주 커질 경우 실제로 예측을 진행하기는 불가능에 가깝다. 때문에, 앙상블 모델 중 하나인 LightGBM 모델을 이용하여 한 번의 학습에 모든 정보를 담고, 모든 상품의 판매량에 대한 예측을 진행하고자 하였다.

	id	item_id	dept_id	store_id	cat_id	state_id	d	sales	date	wm_yr_wk	...	month	year	event_name_1	event_type_1	event_name_2	event_type_2	snap_CA	snap_TX	snap_WI	sell_price
0	HOBBIES_1_002_CA_1_validation	1	0	0	0	0	d_250	0.0	2011-10-05	11136	...	10	2011	0	0	0	0	1.0	1.0	1.0	3.97
1	HOBBIES_1_002_CA_1_validation	1	0	0	0	0	d_251	0.0	2011-10-06	11136	...	10	2011	0	0	0	0	1.0	1.0	1.0	3.97
2	HOBBIES_1_002_CA_1_validation	1	0	0	0	0	d_252	0.0	2011-10-07	11136	...	10	2011	0	0	0	0	1.0	1.0	0.0	3.97
3	HOBBIES_1_004_CA_1_validation	3	0	0	0	0	d_250	0.0	2011-10-05	11136	...	10	2011	0	0	0	0	1.0	1.0	1.0	4.34
4	HOBBIES_1_004_CA_1_validation	3	0	0	0	0	d_251	4.0	2011-10-06	11136	...	10	2011	0	0	0	0	1.0	1.0	1.0	4.34

LightGBM 모델에 적용하기 위해 전처리를 거친 input data는 위와 같다. **3049개의 상품에 대한 정보를 순서대로 1913일씩 정렬**하였다. 각 행은 store, state, department와 같은 **그룹 정보** 뿐 아니라 **날짜에 따른 event, snap, sell prices 등의 정보**까지 모두 포함하고 있다. Feature 중 명목형인 것은 one hot encoding을 적용하지 않고 data type을 **category로 변환**하였다. 그 이유는 가지를 넓고 얕게 뻗는, 즉 균형 트리 분할을 하는 다른 트리 기반 모델과 달리, 정보의 손실을 줄이기 위해 가지를 좁고 깊게 뻗는 LightGBM 모델의 특성 때문이다. feature가 category type 일 경우, LigthGBM은 해당 feature의 모든 범주에 대해 가지를 뻗는 것이 아니라 범주를 이분화하여 두 개의 가지를 뻗은 후 손실 값이 큰 범주에서 또 다시 가지를 이분화하는 식으로 학습을 진행하는데, 이 과정이 LightGBM의 특성에 잘 부합되어 성능을 향상시킨다.

위의 dataset 중 'sales' 변수가 target variable이며, 해당 변수의 주기성과 추세를 반영하기 위해 **moving average와 lag**를 반영한 feature를 추가하였다.

lag_7	lag_28	rmean_7_7	rmean_28_7	rmean_7_28	rmean_28_28
0.0	0.0	0.000000	0.142857	0.178571	0.285714
0.0	0.0	0.000000	0.000000	0.142857	0.285714
0.0	1.0	0.000000	0.142857	0.142857	0.321429
0.0	0.0	0.000000	0.142857	0.142857	0.321429
1.0	1.0	1.428571	1.428571	1.607143	1.785714

weekly seasonality와 monthly seasonality를 고려하기 위해 각각 7차 차분, 28차 차분을 적용한 'lag_7'과 'lag_28' feature를 추가하였고, 증가하는 추세를 고려하기 위해 7일과 28일간의 이동평균을 구한 값에 대해 다시 한 번 더 7일과 28일 간의 이동평균을 구한 'rmean7_7' ~ 'rmean28_28' feature를 추가하였다. 여기서 한 달의 기간을 30이나 31일이 아닌 28일로 설정한 것은 이후 예측해야 하는 기간이 28일이기 때문이며 실제로 30일과 31일을 기준으로 이동평균 값을 구한 결과, 28일로 설정했을 때 보다 낮은 정확도를 보였다. 위의 feature를 추가한 코드는 다음과 같다.

```
dayLags = [7, 28]
lagSalesCols = [f"lag_{dayLag}" for dayLag in dayLags]
for dayLag, lagSalesCol in zip(dayLags, lagSalesCols):
    ds[lagSalesCol] = ds[["id","sales"]].groupby("id")["sales"].shift(dayLag)

windows = [7, 28]
for window in windows:
    for dayLag, lagSalesCol in zip(dayLags, lagSalesCols):
        ds[f"rmean_{dayLag}_{window}"] = ds[["id", lagSalesCol]].groupby("id")[lagSalesCol].transform(lambda x: x.rolling(window).mean())
```

...

...

```
# Remove all rows with NaN value
ds.dropna(inplace = True)
```

이후 gridsearch를 이용하여 hyper parameter의 tuning을 진행하였고, 설정한 hyper parameter에 따라 LightGBM 모델의 학습을 진행했다.

```
params = {
          "objective" : "poisson", #poisson regression
          "metric" :"rmse",
          "force_row_wise" : True, #Learning direction (relatively small columns)
          "learning_rate" : 0.075,
          "sub_row" : 0.75,
          "bagging_freq" : 1, #0 means disable bagging; k means perform bagging at every k iteration
          "lambda_l2" : 0.1, #L2 regularization
          "metric": ["rmse"],
          'verbosity': 1, # Info
          'num_iterations' : 1200,
          'num_leaves': 128,
          "min_data_in_leaf": 100,
         }
```

```
# Train LightGBM model
m_lgb = lgb.train(params, trainData, valid_sets = [validData], verbose_eval = 20)
```

```
[1120]  valid_0's rmse: 2.33106
[1140]  valid_0's rmse: 2.33032
[1160]  valid_0's rmse: 2.32939
[1180]  valid_0's rmse: 2.32871
[1200]  valid_0's rmse: 2.32726
```

학습을 진행한 코드 및 num_iterations : 1200의 결과로 출력된 최종 rmse 값들은 다음과 같다.

위에서 학습된 lgb model을 이용하여 예측 시점 직전 28일간의 데이터를 이용하여 29일째의 판매량을 예측하는 방식으로 전체 예측을 진행하였다. 이후 출력값을 competition 규칙에 맞는 submission 형식으로 수정 후 실제 제출한 결과는 다음과 같다.

앞서 LSTM 모델을 이용하여 전체 상품의 판매량을 input한 예측 결과물과 store별로 세분화하여 진행한 예측 결과물보다 좋은 점수를 보이는 것을 확인할 수 있다.

5. Summary

딥러닝 기반 모델인 LGBM을 사용한 결과에 비해 앙상블 모델 중 하나인 LigthGBM을 사용한 결과가 더 좋은 이유를 분석해본 결과, 시계열 데이터에 대한 이해가 부족했기 때문이라고 생각한다. 수업에서 배운 내용과 동일하게 validation set 설정 후, valid acc / loss를 train acc / loss와 비교하며 과적합 문제가 발생하는지를 확인하려고 했지만 연속적인 시간에 따라 나타나는 판매량을 train과 validation set으로 나누어 학습을 진행하는 과정에서 많은 어려움이 있었다. 학습을 진행할수록 validation loss가 증가하거나 상식 밖의 값을 가지는 것을 확인했는데, 이는 판매량 데이터에 주기와 추tp가 있기 때문이라고 추정하였다. 때문에, 차분과 이동평균을 사용하여 판매량 데이터를 전처리한 후 학습을 진행해보았지만 정확도가 좋지 않았다.

시계열 분석 관련 강의를 수강하지 못하여 비정상 시계열 데이터를 정상 시계열 데이터로 만들어 주는 과정에 있어 Autocorrelation plot과 같은 지표를 이해하는 데에 어려움이 있었고, 가지고 있는 지식 내에서 가장 최선의 LSTM 모델 적합 결과를 도출하기 위해 분기, 월, 요일 등의 시간 정보를 feature로 추가하여 주기나 추세로 인한 문제점을 해결해보려 하였다.

결과적으로, Kaggle Competition에 처음 참여한 입장에서, 예상했던 것보다 좋은 결과를 얻음과 함께 실제로 모델을 제작해보고 시행착오를 겪는 과정에서 많은 것을 배울 수 있었다. 특히 보고서의 서두에서 말한 것처럼, 많지 않은 feature를 이용하여 미래를 예측하는 경험을 몸소 체험해 봄으로써 딥러닝의 활용과 그 가치를 확인할 수 있었다.