LG Aimers
8,Feb

Sunjun Hwang
RAISE Lab

목차

데이터 전처리 방식

먼저, 데이터에 NaN 값이 포함되어 있어, 딥러닝을 진행하기에 매우 부적절해 보였음.
그래서 NaN 값을 처리하고자 해당 값을 전처리 하는 과정을 거쳤다.

1. 삭제(Remove)
2. 선형 보간(Linear Interpolation)
3. 다항 보간(Polynomial Interpolation)
4. 평균 대체(Mean Imputation)
5. KNN 대체(KNN Imputation)

삭제

결측값이 있는 행을 삭제
➔ 데이터 손실이 있으나, 데이터 수가 많을 때는 큰 영향을 미치지 않는다는 특징

물론, 제공 받은 데이터가 9만개의 데이터라 적은 편이기에 결과에 영향을 미칠 수도 있다고 생각은 했음.

```
## 1. 삭제 방법 (숫자형 변수에서만 적용)
delete_test_numeric = test_numeric.dropna()
delete_train_numeric = train_numeric.dropna()
```

선형 보간

앞뒤 값 기준으로 선형 보간, 하지만.. 앞뒤가 NaN이라면 NaN 처리가 불가능하다는 단점.
➔ 이를 해결하기 위해 선형 보간 방법을 거친 후 NaN이 남아 있는 부분은 삭제함.

삭제 방식처럼, 삭제하는 과정이 있으나, 삭제 방법처럼 아예 다 삭제하는 것이 아닌, 덜 삭제 한다는 점에서 차이

```
## 2. 선형 보간법 (숫자형 변수에서만 적용)
data_linear_interpolate_test = test_numeric.interpolate(method='linear')
data_linear_interpolate_train = train_numeric.interpolate(method='linear')
```

다항 보간

곡선을 반영한 보간, 데이터 패턴을 유지하지만 과적합이 가능하다는 단점이 있을 수 있음.

선형보간 방식과 마찬가지로 NaN을 해결하지 못하는 부분도 종종 있음. 해결하기 위해 삭제 시킴

```
## 3. 다항 보간법 (2차 다항식, 숫자형 변수만)
data_poly_interpolate_test = test_numeric.interpolate(method='polynomial', order=2)
data_poly_interpolate_train = train_numeric.interpolate(method='polynomial', order=2)
```

-> 다항 보간법 중, 다항식을 2차식으로 설정했음.

평균 대체

평균값으로 NaN을 채우는 방식입니다. 데이터의 변동성이 감소하며, 제일 많이 사용되는 방법임

앞에서 봤던 삭제, 선형 보간, 다항 보간 방식과 달리 삭제하는 과정은 없음.

```
## 4. 평균 대체 (숫자형 변수에서만 적용)
data_mean_fill_test = test_numeric.fillna(test_numeric.mean())
data_mean_fill_train = train_numeric.fillna(train_numeric.mean())
```

KNN 대체

주변 데이터를 참고하여 채움. 데이터의 패턴을 유지

경향성을 따라가는 느낌이며, 평균 대체와 마찬가지로 NaN이 남지 않음

```
## 5. KNN 대체 (숫자형 변수에서만 적용)
knn_imputer = KNNImputer(n_neighbors=5)
test_data_knn_fill_numeric = pd.DataFrame(knn_imputer.fit_transform(test_numeric), columns=test_numeric.columns)
train_data_knn_fill_numeric = pd.DataFrame(knn_imputer.fit_transform(train_numeric), columns=train_numeric.columns)
```

결과

데이터를 전처리 진행한 후, 모든 파일은 csv파일로 다시 저장을 해둠.

딥러닝이나, 고전적 방식이나 둘 다 해당 데이터로 학습을 시키고 좀 더 나은 방향으로 학습시킬 수 있는 방법을 찾고자 했음.

```
delete_test_df.to_csv( path_or_buf: "./dataset/test_remove.csv", index=False)
delete_train_df.to_csv( path_or_buf: "./dataset/train_remove.csv", index=False)

data_linear_interpolate_test.to_csv( path_or_buf: "./dataset/test_linear.csv", index=False)
data_linear_interpolate_train.to_csv( path_or_buf: "./dataset/train_linear.csv", index=False)

data_poly_interpolate_test.to_csv( path_or_buf: "./dataset/test_poly.csv", index=False)
data_poly_interpolate_train.to_csv( path_or_buf: "./dataset/train_poly.csv", index=False)

data_mean_fill_test.to_csv( path_or_buf: "./dataset/test_mean.csv", index=False)
data_mean_fill_train.to_csv( path_or_buf: "./dataset/train_mean.csv", index=False)

test_data_knn_fill.to_csv( path_or_buf: "./dataset/test_knn.csv", index=False)
train_data_knn_fill.to_csv( path_or_buf: "./dataset/train_knn.csv", index=False)
```

내가 실험할 모델

1. ARIMA
2. SARIMA
3. RNN
4. LSTM
5. GRU
6. Transformer

전통적 모델

딥러닝 모델

모델별 성능 비교

Model	MSE	RMSE	MAE	MAPE(%)
ARIMA	0.1913	0.4374	0.3830	∞
SARIMA	0.1913	0.4373	0.3830	∞
RNN	0.2011	0.4485	0.4235	2531961800.00
LSTM	0.1952	0.4418	0.4080	2315424400.00
GRU	0.2208	0.4698	0.4612	3078133200.00
Transformer	0.1925	0.4387	0.3990	2163484400.00

- ✓ ARIMA와 SARIMA의 성능이 거의 동일
- ✓ 딥러닝 모델(RNN, LSTM, GRU, Transformer)이 ARIMA/SARIMA보다 MSE가 높은 경우가 많음
- ✓ Transformer가 딥러닝 모델 중 가장 낮은 MSE (0.1925)와 RMSE (0.4387)를 기록하여 안정적인 성능을 보임
- ✓ GRU는 가장 높은 MSE (0.2208)와 RMSE(0.4698)를 기록하며 성능이 가장 낮음
- ✓ MAPE가 ARIMA/SARIMA에서는 ∞(불가능)으로 나오고, 딥러닝에서는 과도하게 높은 값으로 계산됨. → MAPE 지표는 적절한 평가 기준이 아님.

전처리 방식별 성능 비교

ARIMA/SARIMA 모델 (Linear 전처리 기준)

Model	MSE	RMSE	MAE
ARIMA	0.1913	0.4374	0.3830
SARIMA	0.1913	0.4373	0.3830

✓ ARIMA와 SARIMA 모델이 전처리 방식에 관계없이 거의 동일한 성능을 보임.
✓ 시계열 모델로는 SARIMA가 더 유연하지만, 데이터에 계절성은 없기에 차이가 크지 않음.

LSTM 모델(전처리 방식별 성능 변화)

Model	MSE	RMSE	MAE
Linear	0.1952	0.4418	0.4080
Polynomial	0.1983	0.4453	0.4241
Mean	0.2119	0.4604	0.4496
KNN	0.1939	0.4403	0.4068

✓ Linear 보간과 KNN 대체가 가장 좋은 성능을 보였음.
✓ 평균 대체(Mean Imputation)는 가장 높은 MSE와 RMSE를 기록하여 예측성능이 저하됨

모델별 특징 요약

- **ARIMA/SARIMA**
  - 간단한 데이터에서 좋은 성능을 보이며, MSE와 RMSE가 안정적
  - 딥러닝보다 학습 시간이 짧고, 해석 가능성이 높다.
  - 전처리 방식에는 큰 영향을 받지 않았음.

- **RNN**
  - 데이터 패턴을 잘 학습하지 못하고, MAPE가 과도하게 높음
  - 장기 의존성을 고려하지 못해 성능이 저하됨.

- **LSTM**
  - RNN보다 좋은 성능을 보였으며, Linear 보간과 KNN 대체에서 최적의 결과를 냄
  - 장기 기억을 잘 학습하는 특징이 있음.

- **GRU**
  - LSTM보다 연산량이 적지만, Linear 전처리에서 가장 높은 MSE와 RMSE를 기록하여 성능이 낮음
  - 데이터가 적절히 학습되지 않았을 가능성이 있음.

- **Transformer**
  - 딥러닝 모델 중 가장 일관적인 예측 성능을 보였으며, 최적의 모델로 평가됨.
  - MSE와 RMSE 기준으로 모든 모델 중 가장 낮음.
  - 하지만, 다른 모델 대비 훈련 시간이 가장 길었음.

결론 – 최적의 모델 선택

- ✓ 전통적 시계열 모델(ARIMA, SARIMA)이 MSE와 RMSE에서 가장 안정적인 성능을 보였음.
- ✓ 딥러닝 모델 중 Transformer가 가장 우수한 성능을 보였으며, LSTM도 좋은 성능을 기록함.
- ✓ 데이터의 특성을 고려할 때, 딥러닝을 사용할 경우 Transformer 또는 LSTM이 최적의 선택임.
- ✓ 전처리 방식은 Linear Interpolation 또는 KNN 대체가 가장 효과적이었음

최종 선택

- 단순한 시계열 분석 → SARIMA 추천
- 복잡한 데이터 패턴 학습 → Transformer 추천
- 실시간 예측이 필요한 경우 → LSTM 추천

결론

- 딥러닝이 항상 시계열 모델보다 좋은 성능을 보이지는 않음
- 데이터의 특성(시계열인지 아닌지, 데이터의 패턴)에 따라 최적의 모델을 선택해야 함.
- 만약 복잡한 데이터 패턴이 존재하고, 장기적인 예측이 중요하다면 Transformer 모델을 활용하는 것이 가장 효과적.

Thanks!

Do you have any questions?

sunjun7559012@yonsei.ac.kr
010 -8240-7559 | https://sites.google.com/view/seonjunhwang