지역난방 시스템 열수요 예측 분석

본 프로젝트는 LSTM 기반 딥러닝 모델을 활용하여 지역난방 시스템의 1시간 후 열수요를 예측합니다. 기상 정보, 시간 데이터, 과거 열수요 패턴을 분석하여 정확한 예측을 목표로 합니다.

머신러닝 발표 4조

차한규, 박성진, 박정호

데이터 전처리 및 정규화

```
# 2. 결측치 처리
df.loc[df['si'] == -99.0, 'si'] = 0
df.loc[df['heat_demand'] < 0, 'heat_demand'] = np.nan
df.interpolate(method='linear', inplace=True)
```

```
C:\Users\Administrator\AppData\Local\Temp\ipykernel_23604\2705532261.py:4: FutureWarning: DataFrame.interpolate with o
bject dtype is deprecated and will raise in a future version. Call obj.infer_objects(copy=False) before interpolating
instead.
  df.interpolate(method='linear', inplace=True)
```

```
# 3. 시간 변수
df['hour'] = df.index.hour
df['hour_sin'] = np.sin(2 * np.pi * df['hour'] / 24)
df['hour_cos'] = np.cos(2 * np.pi * df['hour'] / 24)
df['weekday'] = df.index.weekday
df['is_weekend'] = (df['weekday'] >= 5).astype(int)
df.drop(columns=['hour', 'weekday'], inplace=True)
```

```
# 4. 원-핫 인코딩
df = pd.get_dummies(df, columns=['branch_id'], prefix='branch')
```

```
# 5. 정규화 (StandardScaler로 변경)
X = df.drop(columns=['heat_demand'])
y = df[['heat_demand']]
scaler_X = StandardScaler()
scaler_y = StandardScaler()
X_scaled = scaler_X.fit_transform(X)
y_scaled = scaler_y.fit_transform(y)
```

1 결측값 처리

선형 보간법을 적용하여 연속적 데이터 흐름을 보존했습니다.

2 파생 변수 생성

'hour'와 'weekday' 변수를 사인/코사인 함수로 변환하여 순환성을 표현했습니다.

3 범주형 변수 처리

'branch_id'에 원-핫 인코딩을 적용하여 각 지점의 특성을 구분했습니다.

4 정규화 방식

StandardScaler를 사용하여 이상치에 대한 민감도를 감소시켰습니다.

모델 구성 및 학습

모델 아키텍처

LSTM(32) → Dense(16, relu) → Dense(1) 구조를 채택했습니다.

시퀀스 설계

24시간 데이터를 입력으로 사용하여 시간적 패턴을 학습합니다.

하이퍼파라미터

에포크 10, 배치 사이즈 128로 설정했습니다.

학습 특성

시계열 데이터의 연속성을 유지하기 위해 shuffle=False로 설정했습니다. 이는 데이터의 시간적 순서를 보존하여 모델이 시간 패턴을 학습하는 데 중요합니다.
배치 정규화는 적용하지 않았으며, 최적화 알고리즘으로는 Adam을 사용했습니다. 손실 함수는 MSE를 채택했습니다.

예측 성능 및 분석

52.32

MAE

평균 절대 오차

62.71

RMSE

평균 제곱근 오차

19.98%

MAPE

평균 절대 백분율 오차

오차 발생 요인 분석

1. StandardScaler 적용으로 스케일링 범위가 확대되어 예측값의 불안정성 증가
2. 시계열 데이터의 급격한 변동과 노이즈 패턴
3. 공휴일, 기온 급변 등 복잡한 외부 요인 반영 부족

예측 정확도 향상 노력

다양한 실험 시도

스케일러 비교

MinMaxScaler와 StandardScaler의 성능 비교 실험을 진행했습니다.

모델 구조 최적화

LSTM 층 수와 뉴런 수를 다양하게 조정하여 성능을 비교했습니다.

과적합 방지 기법

Dropout, 조기 종료, Bidirectional LSTM 등 다양한 기법을 실험했습니다.

최종 결정 근거

복잡한 모델은 과적합이 발생했고, 단순한 모델은 예측 안정성이 떨어졌습니다. 현재 구조가 안정성과 정확도 사이의 최적 균형점으로 판단됩니다.

모델 복잡도 증가가 반드시 성능 향상으로 이어지지 않는 현상을 확인했습니다. 이는 데이터의 노이즈와 불규칙성에 기인합니다.

1. 데이터 불러오기 및 확인

1 데이터 로드

Pandas를 사용하여 CSV 파일에서 시계열 데이터를 불러옵니다.

2 기본 탐색

데이터 형태, 크기, 열 구성을 확인합니다.

3 결측치 확인

각 열의 결측값 수와 비율을 계산합니다.

4 데이터 타입 분석

각 변수의 데이터 타입과 메모리 사용량을 확인합니다.

코드는 데이터의 기본 특성을 파악하기 위한 필수적인 탐색 과정을 보여줍니다. 열수요 예측을 위한 기초 데이터 분석 단계입니다.

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_absolute_error, mean_squared_error
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense
```

```python
# 1. 데이터 불러오기
df = pd.read_csv("train_heat.csv", nrows=10000)
df.columns = [col.split('.')[-1] for col in df.columns]
df['tm'] = pd.to_datetime(df['tm'], format='%Y%m%d%H')
df.set_index('tm', inplace=True)
df
```

tm	Unnamed: 0	branch_id	ta	wd	ws	rn_day	rn_hr1	hm	si	ta_chi	heat_demand
2021-01-01 01:00:00	1	A	-10.1	78.3	0.5	0.0	0.0	68.2	-99.00	-8.2	281
2021-01-01 02:00:00	2	A	-10.2	71.9	0.6	0.0	0.0	69.9	-99.00	-8.6	262
2021-01-01 03:00:00	3	A	-10.0	360.0	0.0	0.0	0.0	69.2	-99.00	-8.8	266
2021-01-01 04:00:00	4	A	-9.3	155.9	0.5	0.0	0.0	65.0	-99.00	-8.9	285
2021-01-01 05:00:00	5	A	-9.0	74.3	1.9	0.0	0.0	63.5	-99.00	-9.2	283
...	...	...	...	...	...	...	...	...	...	...	...
2022-02-21 12:00:00	9996	A	-1.1	207.9	1.6	0.0	0.0	79.3	1.14	-2.6	283
2022-02-21 13:00:00	9997	A	1.3	256.7	1.7	0.0	0.0	62.8	0.72	0.0	276
2022-02-21 14:00:00	9998	A	3.1	253.2	4.5	0.0	0.0	39.6	1.64	-0.3	263
2022-02-21 15:00:00	9999	A	2.4	291.3	3.8	0.0	0.0	41.4	1.77	-1.4	248
2022-02-21 16:00:00	10000	A	2.2	263.4	3.8	0.0	0.0	36.2	1.01	-1.1	231

10000 rows × 11 columns

2. 결측치 및 파생 변수 처리

```python
# 2. 결측치 처리
df.loc[df['si'] == -99.0, 'si'] = 0
df.loc[df['heat_demand'] < 0, 'heat_demand'] = np.nan
df.interpolate(method='linear', inplace=True)
```

```
C:\Users\Administrator\AppData\Local\Temp\ipykernel_23604\2705532261.py:4: FutureWarning: DataFrame.interpolate with o
bject dtype is deprecated and will raise in a future version. Call obj.infer_objects(copy=False) before interpolating
instead.
  df.interpolate(method='linear', inplace=True)
```

```python
# 3. 시간 변수
df['hour'] = df.index.hour
df['hour_sin'] = np.sin(2 * np.pi * df['hour'] / 24)
df['hour_cos'] = np.cos(2 * np.pi * df['hour'] / 24)
df['weekday'] = df.index.weekday
df['is_weekend'] = (df['weekday'] >= 5).astype(int)
df.drop(columns=['hour', 'weekday'], inplace=True)
```

```python
# 4. 원-핫 인코딩
df = pd.get_dummies(df, columns=['branch_id'], prefix='branch')
```

```python
# 5. 정규화 (StandardScaler로 변경)
X = df.drop(columns=['heat_demand'])
y = df[['heat_demand']]
scaler_X = StandardScaler()
scaler_y = StandardScaler()
X_scaled = scaler_X.fit_transform(X)
y_scaled = scaler_y.fit_transform(y)
```

결측치 처리 과정

1. 선형 보간법(interpolate)으로 연속 변수의 결측치 보완
2. 열수요(demand) 변수의 결측치를 중점적으로 처리
3. 전후 값을 기반으로 부드러운 연결성 확보

파생 변수 생성

시간적 특성을 활용하기 위해 다음 변수들을 추출했습니다:

- 시간(hour): 0-23 범위의 사인/코사인 변환
- 요일(weekday): 0-6 범위의 사인/코사인 변환
- 지점 ID: 원-핫 인코딩으로 범주화

사인/코사인 변환은 순환 변수의 연속성을 보존하는 중요한 기법입니다. 시간이 23시에서 0시로 넘어갈 때의 급격한 수치 변화를 방지합니다.

3. 정규화 및 시퀀스 생성

데이터 정규화

StandardScaler를 사용하여 평균 0, 표준편차 1로 변환했습니다. 이는 모델 학습 안정성을 높입니다.

훈련/검증 분할

시간적 순서를 고려하여 앞부분은 훈련, 뒷부분은 검증 데이터로 분할했습니다.

시퀀스 데이터 생성

24시간(WINDOW_SIZE=24)의 연속 데이터를 입력으로 1시간 후 예측값을 출력하는 시퀀스를 구성했습니다.

데이터셋 준비

X_train, y_train, X_val, y_val 형태로 모델 학습에 필요한 데이터셋을 완성했습니다.

```python
# 6. 시퀀스 생성
WINDOW_SIZE = 24
def create_sequences(X, y, window_size):
    Xs, ys = [], []
    for i in range(len(X) - window_size):
        Xs.append(X[i:i+window_size])
        ys.append(y[i+window_size])
    return np.array(Xs), np.array(ys)

X_seq, y_seq = create_sequences(X_scaled, y_scaled, WINDOW_SIZE)
split = int(0.8 * len(X_seq))
X_train, X_test = X_seq[:split], X_seq[split:]
y_train, y_test = y_seq[:split], y_seq[split:]
```

```python
# 7. 모델 구성 (단순화)
model = Sequential([
    LSTM(32, input_shape=(WINDOW_SIZE, X_train.shape[2])),
    Dense(16, activation='relu'),
    Dense(1)
])
model.compile(optimizer='adam', loss='mse')
```

```
C:\Users\Administrator\anaconda3\Lib\site-packages\keras\src\layers\rnn\rnn.py:199: UserWarning: Do not pass an `input
_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the fi
rst layer in the model instead.
  super().__init__(**kwargs)
```

4. 모델 구성 및 학습 결과

```
# 8. 학습
history = model.fit(X_train, y_train, epochs=10, batch_size=128, shuffle=False, verbose=1)
```

```
Epoch 1/10
63/63 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - loss: 0.4869
Epoch 2/10
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 0.2511
Epoch 3/10
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 0.0860
Epoch 4/10
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 0.1443
Epoch 5/10
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 0.0696
Epoch 6/10
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 0.0797
Epoch 7/10
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 0.0635
Epoch 8/10
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 0.0549
Epoch 9/10
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 0.0556
Epoch 10/10
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 0.0543
```

모델 구조

```
model = Sequential([    LSTM(32, input_shape=(WINDOW_SIZE, X_train.shape[2])),
Dense(16, activation='relu'),    Dense(1)])
```

간결한 구조로 과적합을 방지하고 일반화 성능을 높였습니다. 복잡한 모델보다 안정적인 예측을 제공합니다.

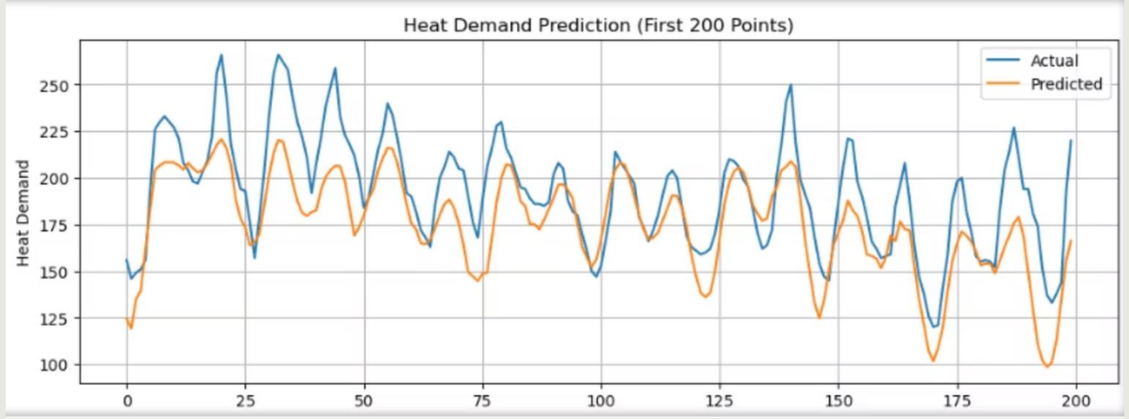
학습 과정 설정

- optimizer: Adam(학습률=0.001)
- loss: MSE(평균제곱오차)
- 에포크: 10
- 배치 크기: 128
- 조기 종료: 사용하지 않음

시계열 데이터 특성을 고려하여 shuffle=False로 설정했습니다. 시간 순서를 보존하는 것이 중요합니다.

5. 예측 결과 시각화 코드 및 출력

```python
# 11. 예측 시각화
plt.figure(figsize=(12, 4))
plt.plot(y_test_inv[:200], label='Actual')
plt.plot(y_pred_inv[:200], label='Predicted')
plt.title('Heat Demand Prediction (First 200 Points)')
plt.xlabel('Time Index')
plt.ylabel('Heat Demand')
plt.legend()
plt.grid(True)
plt.show()
```



시각화 코드

Matplotlib을 활용하여 실제값과 예측값을 비교하는 그래프를 생성했습니다.

성능 평가

MAE, RMSE, MAPE 지표를 계산하여 모델의 예측 정확도를 정량적으로 평가했습니다.

개선 방향

시각화 결과를 바탕으로 추가 특성 공학과 모델 구조 조정 방향을 도출했습니다.

패턴 분석

그래프를 통해 모델이 전반적인 추세는 잘 포착하지만 급격한 변동은 놓치는 경향을 확인했습니다.