스케일링

- `StandardScaler` **(표준화)**
  - 각 변수의 평균을 0, 표준편차를 1로 변환
  - 데이터가 정규분포를 따른다고 가정하며, 각 데이터 포인트가 평균으로부터 몇 표준편차만큼 떨어져 있는지를 나타내는 Z-score와 동일
- `MinMaxScaler` **(정규화)**
  - 각 변수의 최소값을 0, 최대값을 1로 변환
  - 모든 데이터가 0과 1 사이의 값으로 변환됨
- `RobustScaler`
  - `StandardScaler`와 유사하지만, 평균과 표준편차 대신 중앙값(median)과 사분위 범위(IQR)를 사용하여 스케일링
  - 이상치(outlier)의 영향을 훨씬 덜 받으며, 이상치가 많은 데이터에 더 적합

적용 가능한 상황

- **변수 간 스케일 차이가 클 때**: 거리 기반 알고리즘(KNN, SVM, K-Means 등), 경사 하강법 기반 알고리즘(선형 회귀, 로지스틱 회귀, 신경망 등)에서 모델이 변수들의 스케일 차이에 영향을 받지 않도록 하기 위해 필수적으로 사용됩니다.
- `StandardScaler`: 데이터가 정규분포에 가깝고, 이상치가 거의 없을 때 가장 일반적으로 사용됩니다.
- `MinMaxScaler`: 모든 변수를 동일한 범위(0~1)에 놓고 비교하고 싶을 때, 또는 이미지 처리에서 픽셀 값을 0~1 사이로 맞추거나, 신경망 모델에 데이터를 입력할 때 유용합니다.
- `RobustScaler`: 데이터에 이상치가 많아 `StandardScaler`의 성능이 저하될 우려가 있을 때 사용하면 더 안정적인 스케일링 결과를 얻을 수 있습니다.

Note: 트리 기반 모델(Decision Tree, Random Forest, Gradient Boosting 등)은 변수를 독립적으로 처리하므로 일반적으로 스케일링이 필요하지 않습니다.

주의사항

- 스케일러는 **학습 데이터(train data)에 대해서만 `fit()`과 `transform()`을 적용**하고, **테스트 데이터(test data)에 대해서는 `transform()`만 적용**해야 합니다.
- 이는 테스트 데이터의 정보(평균, 최소/최대 등)가 모델 학습 과정에 미리 노출되는 것을 방지(Data Leakage)하기 위함입니다.
- `fit()`은 스케일링에 필요한 통계량(평균, 표준편차, 최소/최대 등)을 계산하는 과정이고, `transform()`은 이를 적용하여 실제로 데이터를 변환하는 과정입니다.

예제 데이터 생성

```
import pandas as pd
from sklearn.preprocessing import StandardScaler, MinMaxScaler, RobustScaler

# 소득과 나이, 그리고 이상치를 포함한 데이터 생성
data = {'income': [50000, 52000, 48000, 150000, 51000, 49000, 1000000],
        'age': [25, 30, 28, 45, 26, 27, 35]}
df = pd.DataFrame(data)
```

1. StandardScaler

- **용도**: 평균 0, 표준편차 1을 갖도록 데이터를 변환
- **수식**: `x_scaled = (x - mean) / std`
- **코드 예시**

```
# 1. 스케일러 객체 생성
scaler_std = StandardScaler()

# 2. 데이터에 fit_transform 적용 (fit과 transform을 동시에)
df_std = scaler_std.fit_transform(df)

# 결과 확인 (Numpy 배열로 반환됨)
print("--- StandardScaler Result ---")
print(df_std)
print(f"\nMean: {df_std.mean(axis=0)}")
print(f"Std: {df_std.std(axis=0)}")

# Scikit-Learn을 사용 못할 경우 직접 연산 하는 방법
# ddof=0 으로 표준편차 설정을 해줘야 모수 표준편차로 계산됨 (기본 ddof=1은 표본
표준편차)
scaled_res = ((df - df.mean()) / df.std(ddof=0)).to_numpy()

# 일부 열만 스케일링하고 싶을 경우, cols를 지정한다.
cols = ['income', 'age']
scaled_res = ((df[cols] - df[cols].mean()) /
df[cols].std(ddof=0)).to_numpy()

# Scikit-Learn과 동일한 결과를 볼 수 있다.
print(scaled_res)
print(f"\nMean: {scaled_res.mean(axis=0)}")
print(f"Std: {scaled_res.std(axis=0)}")
```

- **결과 해석**
  - 변환된 데이터의 평균은 거의 0에 가깝고, 표준편차는 1이 됩니다.
  - `income`의 이상치(1,000,000) 때문에 다른 값들이 모두 음수로 크게 밀려나는 것을 볼 수 있습니다.
  - 이는 `StandardScaler`가 이상치에 민감함을 보여줍니다.

2. MinMaxScaler

- **용도**: 최소값 0, 최대값 1을 갖도록 데이터를 변환
- **수식**: `x_scaled = (x - min) / (max - min)`
- **코드 예시**

```
# 1. 스케일러 객체 생성
scaler_minmax = MinMaxScaler()

# 2. 데이터에 fit_transform 적용
df_minmax = scaler_minmax.fit_transform(df)

# 결과 확인
print("--- MinMaxScaler Result ---")
print(df_minmax)
print(f"\nMin: {df_minmax.min(axis=0)}")
print(f"Max: {df_minmax.max(axis=0)}")

# Scikit-Learn을 사용 못할 경우 직접 연산 하는 방법
scaled_res = ((df - df.min()) / (df.max()-df.min())).to_numpy()

# 일부 열만 스케일링하고 싶을 경우, cols를 지정한다.
cols = ['income', 'age']
scaled_res = ((df[cols] - df[cols].min()) / (df[cols].max()-
df[cols].min())).to_numpy()

# Scikit-Learn과 동일한 결과를 볼 수 있다.
print(scaled_res)
print(f"\nMin: {scaled_res.min(axis=0)}")
print(f"Max: {scaled_res.max(axis=0)}")
```

- **결과 해석**
  - 모든 값이 0과 1 사이로 변환됩니다.
  - 하지만 `income`의 이상치(1,000,000)가 1이 되고, 나머지 값들은 0에 매우 가깝게 압축되어 데이터의 분포를 제대로 표현하지 못하게 됩니다.
  - `MinMaxScaler` 역시 이상치에 매우 민감합니다.

3. RobustScaler

- **용도**: 중앙값 0, IQR 1을 갖도록 데이터를 변환
- **수식**: `x_scaled = (x - median) / IQR`
- **코드 예시**

```
# 1. 스케일러 객체 생성
scaler_robust = RobustScaler()

# 2. 데이터에 fit_transform 적용
df_robust = scaler_robust.fit_transform(df)

# 결과 확인
print("--- RobustScaler Result ---")
```

```
print(df_robust)

# Scikit-Learn을 사용 못할 경우 직접 연산 하는 방법

# 중앙값과 3분위수-1분위수를 통해 IQR 값을 구하고 진행
median = df.median()
iqr = df.quantile(0.75) - df.quantile(0.25)
scaled_res = ((df - median) / iqr).to_numpy()

# 일부 열만 스케일링하고 싶을 경우, cols를 지정한다.
cols = ['income', 'age']
scaled_res = ((df[cols] - df[cols].median()) / (df[cols].quantile(0.75)-
df[cols].quantile(0.25))).to_numpy()

# Scikit-Learn과 동일한 결과를 볼 수 있다.
print(scaled_res)
```

- **결과 해석**
  - `income`의 이상치(1,000,000)가 다른 값들에 미치는 영향이 `StandardScaler`나 `MinMaxScaler`에 비해 훨씬 적습니다.
  - 이상치를 제외한 나머지 값들의 스케일이 비교적 잘 유지되는 것을 볼 수 있습니다.
  - 이처럼 `RobustScaler`는 이상치가 존재할 때 훨씬 안정적인 변환 결과를 제공합니다.

`inverse_transform()`

- 상단의 모든 스케일러는 `inverse_transform()`을 통해 원상복구 시킬 수 있습니다.
- 그냥 변환할 경우, 지수표기법으로 결과가 나오는 경우도 있으니 확인 필요

```
# Numpy 설정을 바꾸는 방법
np.set_printoptions(suppress=True)   # 지수표기법 off
np.set_printoptions(precision=6)     # 소수점 자릿수 지정 (예: 6자리)

# Pandas DataFrame으로 변환 시 일반 표기법으로 보여줌 / astype은 변수별 적용이 안전
pd.DataFrame(scaler_robust.inverse_transform(df_robust)).astype("int")
```

장단점 및 대안

스케일러	장점	단점	선택 가이드
`StandardScaler`	가장 널리 사용되는 기본적인 스케일링 방법. 데이터의 분포를 크게 왜곡하지 않음.	이상치에 매우 민감하여, 이상치가 있는 경우 데이터 분포가 크게 왜곡될 수 있음.	데이터에 이상치가 거의 없고, 분포가 정규분포에 가까울 때 좋은 선택입니다. 많은 머신러닝 알고리즘의 기본 스케일러로 사용됩니다.

스케일러	장점	단점	선택 가이드
`MinMaxScaler`	모든 값을 0과 1 사이로 명확하게 제한함.	이상치에 매우 민감함. 이상치 하나 때문에 대부분의 데이터가 매우 좁은 범위에 압축될 수 있음.	데이터의 최소/최대값을 명확히 알고 있고, 그 범위가 크게 변하지 않을 때 유용합니다. 신경망에서 활성화 함수(e.g., Sigmoid)의 입력 범위와 맞출 때 자주 사용됩니다.
`RobustScaler`	이상치에 거의 영향을 받지 않아 안정적임(Robust).	`StandardScaler`에 비해 널리 사용되지는 않음. 변환된 값의 범위가 특정 범위로 제한되지 않음.	데이터에 이상치가 많다고 판단될 때 가장 먼저 고려해야 할 스케일러입니다. 이상치를 제거하지 않고 모델링을 진행할 때 특히 유용합니다.

대안: `MaxAbsScaler`는 각 변수를 최대 절대값으로 나누어 -1과 1 사이로 스케일링합니다. 데이터가 0을 중심으로 분포할 때 유용합니다. `Normalizer`는 각 샘플(행)의 유클리드 길이가 1이 되도록 스케일링하며, 이는 변수(열) 단위로 스케일링하는 다른 방법들과는 다릅니다.