데이터 스케일링

2025. 05. 12(월)

학습 목표

01 데이터 스케일링의 필요성을 이해합니다.

02 주요 스케일링 기법을 파악하고 구현합니다.

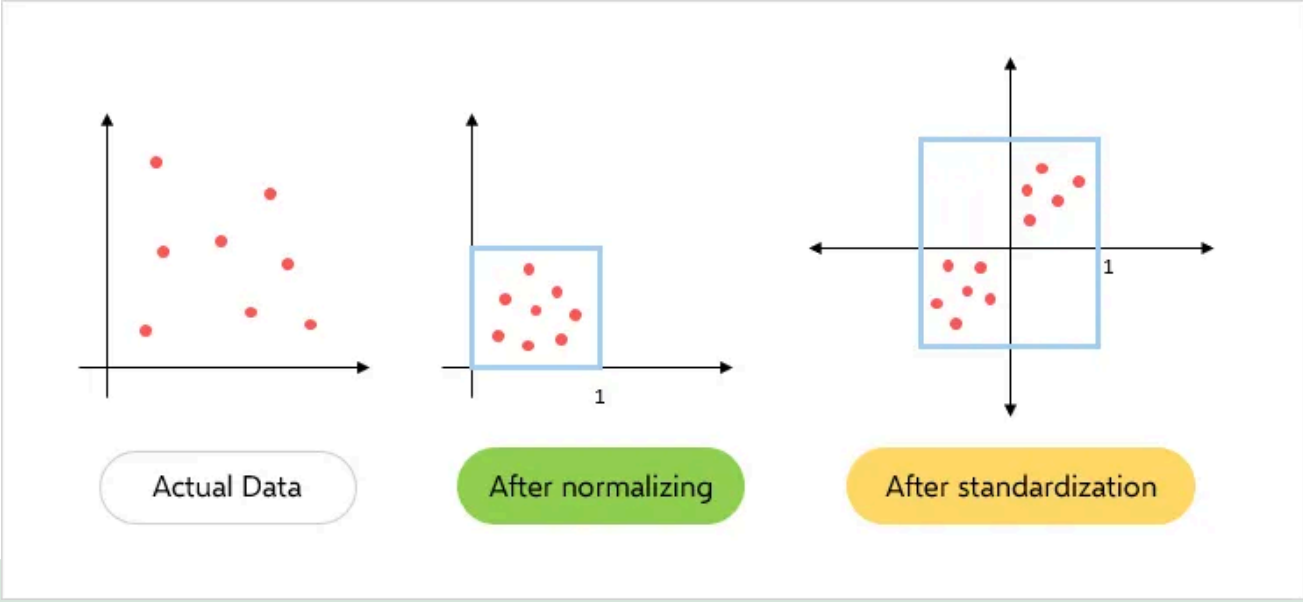
03 스케일링 과정에서 Data Leakage를 방지하는 방법을 학습

목차

스케일링 소개 및 필요성

스케일링이란?

- 머신러닝 전처리 단계 중 하나
- 데이터 Feature들의 값 범위(Scale)를 조정하여 일정한 수준으로 맞춰주는 작업



스케일링이 필요한 이유

- 모델 성능 향상:
  - 거리 기반 모델: KNN, K-Means, SVM 등은 Feature 간 거리에 민감하므로, Scale 차이가 크면 특정 Feature의 영향력이 과도해짐.
  - 경사하강법 기반 모델: 선형/로지스틱 회귀, 신경망 등에서 Feature 스케일이 다르면 최적 해를 찾는 경로가 비효율적이 되어 학습 속도가 느려지거나 수렴이 불안정해질 수 있음.
- Feature 중요도 해석 왜곡 방지: Scale 차이로 인한 분석 결과 왜곡 방지.

주요 스케일링 기법

2.1. 표준화 (Standardization)

- 개념: 데이터의 평균을 0, 표준편차를 1로 변환
- Z-점수 정규화라고도 불림
- 데이터 분포를 정규분포 형태로 만드는 효과

$$z = \frac{x - \mu}{\sigma}$$

- x: 원래 값
- μ: 평균 (mean)
- σ: 표준편차 (standard deviation)

표준화 Python 예제 (StandardScaler)

```
 import pandas as pd
from sklearn.preprocessing import StandardScaler

data = {'F1': [10, 20, 30, 40, 50], 'F2': [1, 2, 3, 4, 5]}
df = pd.DataFrame(data)
print("Original:\n", df)

# StandardScaler 객체 생성 및 적용
scaler = StandardScaler()
df_scaled = scaler.fit_transform(df) # 학습(fit)과 변환(transform) 동시 수행
df_scaled = pd.DataFrame(df_scaled, columns=df.columns)

print("\nStandardized:\n", df_scaled)
print("\nMean after scaling:\n", df_scaled.mean().round(2)) # 소수점 둘째 자리까지 표시
print("\nStd Dev after scaling:\n", df_scaled.std().round(2)) # 소수점 둘째 자리까지 표시
```

2.2. Min-Max 스케일링 (Min-Max Scaling)

- 개념: 데이터를 특정 범위 (주로 0~1 사이)로 조정
- Normalization이라고도 불리지만, 엄밀히는 표준화와 구분됨 (범위 vs 분포)
- 모든 Feature가 동일한 고정 범위에 있도록 만듦

$$X_{scaled} = \frac{X - X_{min}}{X_{max} - X_{min}}$$

- X: 원래 값
- X_{min}: Feature의 최솟값
- X_{max}: Feature의 최댓값

Min-Max 스케일링 Python 예제 (MinMaxScaler)

```
import pandas as pd
from sklearn.preprocessing import MinMaxScaler

data = {'F1': [10, 20, 30, 40, 50], 'F2': [1, 2, 3, 4, 5]}
df = pd.DataFrame(data)
print("Original:\n", df)

# MinMaxScaler 객체 생성 및 적용
scaler = MinMaxScaler()
df_scaled = scaler.fit_transform(df) # 학습(fit)과 변환(transform) 동시 수행
df_scaled = pd.DataFrame(df_scaled, columns=df.columns)

print("\nMin-Max Scaled:\n", df_scaled)
print("\nMin values after scaling:\n", df_scaled.min())
print("\nMax values after scaling:\n", df_scaled.max())
```

2.3. Robust 스케일링 (Robust Scaling)

- 개념: 이상치(Outlier)의 영향을 최소화하며 스케일링
- 평균/표준편차 대신 중앙값(Median)과 사분위수(Quartile) 사용
- 이름처럼 "강건한(Robust)" 스케일링 방법
- 작동 원리:
  - 중앙값 (Median): 데이터 정렬 시 가운데 값 (이상치 영향 적음)
  - 사분위수 (Quartile): Q1(25%), Q3(75%)
  - IQR (Interquartile Range): $Q_3 - Q_1$

$$x_i' = \frac{x_i - \text{median}(x)}{\text{IQR}(x)} \quad (\text{sklearn 기준})$$

Robust 스케일링 Python 예제 (RobustScaler)

```
 import numpy as np
from sklearn.preprocessing import RobustScaler
import pandas as pd

# Sample data (이상치 포함)
data = np.array([[10, 10], [20, 20], [30, 30],
                 [40, 40], [50, 50], [1, 100], [2, 200], [1000, 5]]) # 이상치를 좀 더 극적으로 추가
df = pd.DataFrame(data, columns=['feature_1', 'feature_2'])
print("Original Data (with outlier):\n", df)

# RobustScaler 객체 생성 및 적용
robust_scaler = RobustScaler()
scaled_data = robust_scaler.fit_transform(df) # 학습(fit)과 변환(transform) 동시 수행
scaled_df = pd.DataFrame(scaled_data, columns=['feature_1', 'feature_2'])

print("\nRobust Scaled Data:\n", scaled_df)
print("\nMedian after scaling:\n", scaled_df.median()) # 중앙값은 0 근처가 됨
```

2.4. MaxAbs 스케일링 (MaxAbs Scaling)

- 개념: 각 Feature의 절댓값이 가장 큰 값을 기준으로 스케일링
- 데이터를 -1과 1 사이 범위로 조정 ($$ ₩frac|} $$)
- 데이터의 중심을 0으로 옮기지는 않음
- 특징:
  - 데이터의 부호 유지
  - **희소 데이터 (Sparse Data)**에 적합 (0 값을 그대로 유지)
  - Outlier에 Min-Max만큼 민감하지는 않으나 Robust보다는 덜 강건

MaxAbs 스케일링 Python 예제 (MaxAbsScaler)

```
 import numpy as np
from sklearn.preprocessing import MaxAbsScaler
import pandas as pd

# Sample data (양수/음수 포함 가능, 희소 데이터에도 적합)
data = np.array([[-10, 1], [0, 5], [20, 0], [-5, -3]])
df = pd.DataFrame(data, columns=['feature_1', 'feature_2'])
print("Original Data:\n", df)

# MaxAbsScaler 객체 생성 및 적용
scaler = MaxAbsScaler()
scaled_data = scaler.fit_transform(df) # 학습(fit)과 변환(transform) 동시 수행
scaled_df = pd.DataFrame(scaled_data, columns=['feature_1', 'feature_2'])

print("\nMaxAbs Scaled Data:\n", scaled_df)
print("\nMax Absolute values after scaling:\n", scaled_df.abs().max()) # 절댓값의 최댓값이 1이 됨
```

스케일링 적용 시점 및 주의사항

언제 스케일링을 적용해야 할까?

- 거리 기반 알고리즘: KNN, K-Means, DBSCAN, SVM (커널 방법 포함), PCA 등
- 경사하강법 기반 알고리즘: 신경망 (NN), 선형 회귀 (Linear Regression), 로지스틱 회귀 (Logistic Regression) 등
- Feature별 스케일이 크게 다를 때
- Feature Engineering 과정에서 새로운 Feature를 만들었는데 기존 Feature와 Scale 차이가 클 때

언제 불필요하거나 해가 될까?

- 트리 기반 알고리즘: Decision Tree, Random Forest, Gradient Boosting (XGBoost, LightGBM) 등은 Feature의 절대적인 Scale에 덜 민감합니다. (분기 기준은 Scale에 영향을 받지 않음)
- Feature의 Scale 자체가 의미 있는 정보인 경우: 예를 들어, 온도나 금액 데이터에서 값 자체의 크기가 중요한 의미를 가질 때 스케일링이 정보 손실을 야기할 수 있습니다.
- 이미 같은 스케일을 가진 Feature들: 모든 Feature가 이미 0~1 또는 -1~1 같은 유사한 범위에 있다면 추가적인 스케일링이 불필요할 수 있습니다.

Data Leakage 방지

Data Leakage란?

- 모델 학습 과정에서 테스트 데이터나 검증 데이터의 정보가 직간접적으로 모델 학습에 영향을 미치는 현상
- 스케일링 과정에서 가장 흔하게 발생
- 모델 성능을 과대평가하게 만들어 실제 배포 시 성능 하락의 주요 원인

잘못된 스케일링 예시 (Data Leakage 발생)

```
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
import numpy as np # 예시를 위해 numpy 사용

# 예시 데이터 생성
data = np.random.rand(100, 5) * 100 # 0~100 사이의 임의 데이터
labels = np.random.randint(0, 2, 100) # 임의 라벨

# 데이터를 훈련 세트와 테스트 세트로 분할
X_train, X_test, y_train, y_test = train_test_split(data, labels, test_size=0.3, random_state=42)

# 🚨 Data Leakage 발생 시점! 🚨
# 테스트 데이터 정보가 포함된 전체 데이터로 fit() 또는 fit_transform() 사용
scaler_leak = StandardScaler()
# X_all = np.vstack((X_train, X_test)) # train과 test를 합치는 것은 Data Leakage를 유발
# scaler_leak.fit(X_all) # 이렇게 하면 테스트 데이터 정보가 fit에 포함!
# scaled_X_train_leak = scaler_leak.transform(X_train)
# scaled_X_test_leak = scaler_leak.transform(X_test)

# 🚨 더 심각한 Data Leakage! 🚨
# 테스트 데이터로 별도의 scaler를 fit_transform 하는 경우
# scaler_test_leak = StandardScaler()
```

올바른 스케일링 방법

```
 from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
import numpy as np # 예시를 위해 numpy 사용

# 예시 데이터 생성
data = np.random.rand(100, 5) * 100 # 0~100 사이의 임의 데이터
labels = np.random.randint(0, 2, 100) # 임의 라벨

# 데이터를 훈련 세트와 테스트 세트로 분할 (이 과정이 가장 먼저 이루어져야 함!)
X_train, X_test, y_train, y_test = train_test_split(data, labels, test_size=0.3, random_state=42)

# ✅ 올바른 방법 ✅
scaler = StandardScaler()

# 1. 반드시 훈련 데이터(X_train)로만 Scaler를 학습 (fit)
scaler.fit(X_train)

# 2. 학습된 Scaler로 훈련 데이터를 변환 (transform)
scaled_X_train = scaler.transform(X_train)

# 3. 학습된 (X_train으로 fit된) Scaler로 테스트 데이터를 변환 (transform)
# 테스트 데이터의 통계량은 스케일링에 영향을 주지 않음
```

어떤 스케일러를 선택할까?

스케일러 선택 가이드

어떤 스케일러를 사용할지는 데이터의 특성, 모델의 종류, 이상치의 존재 여부 등을 고려하여 결정해야 합니다.

- StandardScaler:
  - 가장 일반적으로 사용됨
  - Feature가 정규분포에 가깝거나 이상치의 영향이 크지 않을 때 적합
  - 거리 기반 모델, 경사하강법 기반 모델 (신경망 등)에 널리 사용
- MinMaxScaler:
  - Feature 범위를 0~1 또는 특정 범위로 엄격하게 제한하고 싶을 때
  - 이미지 처리 등 Feature의 상대적인 크기보다 절대적인 범위가 중요할 때
  - 이상치(Outlier)에 매우 민감하므로 이상치 처리가 선행되거나 RobustScaler 고려

스케일러 선택 가이드 (계속)

- RobustScaler:
  - 데이터에 이상치가 많을 때 가장 효과적
  - 이상치의 영향을 최소화하면서 Scale 조정
  - Feature의 절대적인 범위보다 중앙값으로부터의 거리가 중요할 때
- MaxAbsScaler:
  - **희소 데이터 (Sparse Data)**에 적합 (0 값을 유지해야 할 때)
  - 데이터의 부호 유지 필요 시
  - 이상치에 Standard/MinMax보다 덜 민감하지만 Robust보다는 취약

결론 및 실전 팁

- 데이터 탐색: 스케일링 전에 Feature별 분포, 범위, 이상치 여부를 확인합니다.
- 모델 고려: 사용할 모델의 특성을 파악하고 적합한 스케일러를 선택합니다.
- Data Leakage 방지: Train/Test 분할 -> Train에 fit -> Train/Test에 transform 순서 철저히 준수!
- 다양한 시도: 여러 스케일러를 적용해보고 모델 성능을 비교하여 최적의 스케일러를 선택하는 것이 좋습니다.
- Pipeline 활용: Scikit-learn의 Pipeline을 사용하면 전처리 단계와 모델 학습을 묶어서 Data Leakage를 방지하고 코드를 간결하게 관리할 수 있습니다.

요약 정리

- 데이터 스케일링은 거리/경사하강법 기반 모델 성능 향상 및 학습 안정화에 필수입니다.
- 가장 중요한 주의사항은 스케일링 과정에서 Data Leakage를 방지하는 것입니다. Train 데이터로만 `fit()`하고 Train/Test 데이터 모두에 `transform()`을 적용해야 합니다.
- 데이터 특성(분포, 이상치)과 모델 종류를 고려하여 적절한 스케일러를 선택하는 것이 중요합니다.