데이터 변환

2025. 05. 12(월)

학습 목표

01 데이터 변환의 필요성과 목적을 이해합니다.

02 데이터 분포 정규화를 위한 변환 기법을 이해 및 구현합니다.

03 주요 변환 기법의 특징과 적용 시점을 비교합니다.

목차

데이터 변환 소개 및 필요성

데이터 변환 개요

- 데이터 변환: 데이터의 분포, 형태, 또는 관계를 변경하는 전처리 과정
- 주로 데이터의 비대칭성(왜도)을 완화하고 분포를 정규분포에 가깝게 만드는 데 사용
- (앞서 배운 스케일링은 데이터의 '범위'를 조정하는 반면, 변환은 데이터의 '분포'를 조정하는 데 초점)

데이터 변환의 필요성

- 모델 가정 충족: 선형 회귀, 가우시안 나이브 베이즈 등 일부 모델은 Feature가 정규분포를 따른다고 가정합니다. 변환을 통해 이 가정을 만족시켜 모델 성능을 높일 수 있습니다.
- 분산 안정화: 데이터 값의 크기에 따라 분산이 달라지는 문제를 완화합니다.
- 이상치 영향 감소: 극단적인 값들의 간격을 줄여 이상치의 영향을 줄입니다. (로그 변환 등)
- 선형성 개선: Feature와 Target 간의 비선형 관계를 선형적으로 변환하여 모델 학습을 용이하게 합니다.

주요 데이터 변환 기법

2.1. 로그 변환 (Log Transformation)

- 개념: 데이터 값에 로그 함수를 적용
- 목적: 주로 오른쪽으로 치우친 (Positive Skewed) 데이터의 비대칭성을 완화하고 분포를 정규분포에 가깝게 만듭니다.
- 효과: 큰 값들의 간격을 압축하여 분산을 안정화하고 이상치의 영향을 줄입니다.

- 수식:

$$X_{log} = \log(X)$$

  또는 0 또는 음수 값 처리를 위해

$$X_{log} = \log(X + 1)$$

  (NumPy에서는 `np.log1p()` 함수 제공)

- 주의: 데이터 값이 음수이거나 0인 경우 로그 변환을 직접 적용할 수 없습니다. $$X+1$$ 형태로 사용하거나 다른 변환 기법을 고려해야 합니다.

로그 변환 Python 예제

```
 import numpy as np
import pandas as pd
# 데이터 분포 시각화는 matplotlib/seaborn 필요 (여기서는 코드 실행만 예시)
# import matplotlib.pyplot as plt
# import seaborn as sns

# 오른쪽으로 치우친 데이터 (예: 소득, 판매량 등)
data = np.random.exponential(scale=100, size=200).astype(int) # 간결화
# 실제 데이터는 df['Column_Name'].values 형태로 사용
print("변환 전 데이터 (일부):", data[:10])
print("변환 전 평균:", np.mean(data).round(2))
print("변환 전 왜도:", pd.Series(data).skew().round(2))

# 로그 변환 적용 (np.log1p 사용 - log(x+1))
log_data = np.log1p(data)

print("변환 후 데이터 (일부):", log_data[:10].round(2))
print("변환 후 평균:", np.mean(log_data).round(2))
print("변환 후 왜도:", pd.Series(log_data).skew().round(2)) # 왜도가 감소함

# 시각화 예시 (코드 실행 안함)
# plt.figure(figsize=(12, 5))
# plt.subplot(1, 2, 1)
```

2.2. 제곱근 변환 (Square Root Transformation)

- 개념: 데이터 값에 제곱근 함수 ($$₩sqrt$$)를 적용
- 목적: 로그 변환과 유사하게 오른쪽으로 치우친 데이터의 비대칭성을 완화하고 분산을 안정화합니다.
- 효과: 로그 변환보다 변환 강도가 약합니다.
- 수식:

$$X_{sqrt} = \sqrt{X}$$

- 주의: 데이터 값이 음수인 경우 적용할 수 없습니다. 0 값은 0으로 변환됩니다.

제곱근 변환 Python 예제

```
 import numpy as np
import pandas as pd

# 오른쪽으로 치우친 데이터 (음수 없음)
data = np.array([1, 4, 9, 16, 25, 100, 400]) # 간결화 예시
print("변환 전 데이터:", data)
print("변환 전 왜도:", pd.Series(data).skew().round(2))

# 제곱근 변환 적용
sqrt_data = np.sqrt(data)

print("변환 후 데이터:", sqrt_data)
print("변환 후 왜도:", pd.Series(sqrt_data).skew().round(2)) # 왜도가 감소함
```

2.3. 거듭제곱 변환 (Power Transformation)

- 개념: 데이터 분포를 정규 분포에 더 가깝게 만들기 위해 사용되는 일반적인 변환 기법
- 데이터에 적합한 최적의 변환 거듭제곱($$₩lambda$$)을 찾아 적용합니다.
- 주요 변환: Box-Cox 변환, Yeo-Johnson 변환

Box-Cox 변환

- 개념: 양수 데이터에만 적용 가능. 데이터에 적합한 최적의 $$₩lambda$$ 값을 찾아 다음과 같은 수식을 적용합니다.
- 수식:

$$y(\lambda) = \begin{cases} \frac{x^\lambda - 1}{\lambda} & \text{if } \lambda \neq 0 \\ \ln(x) & \text{if } \lambda = 0 \end{cases}$$

($$₩lambda = 0$$ 일 때는 자연 로그 변환과 같아짐)

- 장점: 데이터의 정규성 개선, 분산 안정화에 효과적
- 단점: 데이터 값이 반드시 양수여야 함 ($$X > 0$$), 변환된 결과의 해석이 어려움

Box-Cox 변환 Python 예제 (scipy.stats 또는 sklearn.preprocessing)

```
import numpy as np
from scipy import stats
import pandas as pd
from sklearn.preprocessing import PowerTransformer # sklearn 사용 시

# 양수 데이터 생성 (예: 가격, 거래량 등)
original_data = np.random.gamma(shape=5, scale=10, size=200) # 간결화
# 데이터에 0이 포함될 수 있다면 작은 값을 더하거나 Yeo-Johnson 사용
# original_data = original_data + 1 # 만약 데이터에 0이 있다면

print("변환 전 (일부):", original_data[:5].round(2))
print("변환 전 왜도:", pd.Series(original_data).skew().round(2))

# 방법 1: scipy.stats 사용 (최적 람다 값도 함께 반환)
transformed_data_scipy, best_lambda = stats.boxcox(original_data)
print(f"\nScipy Box-Cox 변환 (최적 람다: {best_lambda:.4f})")
print("변환 후 (일부):", transformed_data_scipy[:5].round(2))
print("변환 후 왜도:", pd.Series(transformed_data_scipy).skew().round(2))

# 방법 2: sklearn.preprocessing 사용 (Pipeline 등에 통합 편리)
# pt_cox = PowerTransformer(method='box-cox')
# transformed_data_sklearn = pt_cox.fit_transform(original_data.reshape(-1, 1)).flatten()
# print("\nSklearn Box-Cox 변환")
```

Yeo-Johnson 변환

- 개념: Box-Cox 변환을 보완하기 위해 개발되었으며, 음수, 0, 양수 값을 모두 포함하는 데이터에도 적용 가능합니다.
- 데이터 값의 부호에 따라 다른 수식을 적용하여 최적의 $$₩lambda$$ 값을 찾습니다.
- 수식: (Box-Cox보다 복잡하며, 데이터 부호에 따라 다름)
- 장점: 데이터 값 범위에 제약 없이 적용 가능, 정규성 개선에 효과적
- 단점: 변환된 결과의 해석이 Box-Cox 변환보다도 더 어려움

Yeo-Johnson 변환 Python 예제 (sklearn.preprocessing)

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

# 음수, 0, 양수 데이터를 포함하는 데이터 생성
data = np.random.normal(loc=5, scale=10, size=200) # 정규 분포 형태지만, 왜도를 가진 데이터로 변경
# data = np.concatenate([data, [-50, 0, 100]]) # 극단적인 값 추가 예시
print("변환 전 (일부):", data[:5].round(2))
print("변환 전 왜도:", pd.Series(data).skew().round(2))


# Yeo-Johnson 변환 적용 (sklearn PowerTransformer 사용)
pt_yj = PowerTransformer(method='yeo-johnson')
yeojohnson_data = pt_yj.fit_transform(data.reshape(-1, 1)).flatten() # sklearn은 2D array 입력 필요

print("\nYeo-Johnson 변환")
print("변환 후 (일부):", yeojohnson_data[:5].round(2))
print("변환 후 왜도:", pd.Series(yeojohnson_data).skew().round(2))
print("Sklearn 람다:", pt_yj.lambdas_.round(4)) # sklearn으로 찾은 람다 값
```

변환 기법 선택 가이드

변환 기법 선택 가이드

어떤 변환 기법을 사용할지는 주로 데이터의 분포 형태와 값의 범위(양수만인지, 음수/0 포함인지)에 따라 결정됩니다.

- 로그 변환 (Log Transformation):
  - 강한 오른쪽 왜도를 가지는 데이터에 가장 일반적으로 시도됨
  - 데이터 값이 모두 양수일 때 사용 (0 또는 음수 포함 시 log(x+1) 고려)
  - 해석이 비교적 직관적임
- 제곱근 변환 (Square Root Transformation):
  - 로그 변환과 유사하게 오른쪽 왜도 완화에 사용
  - 변환 강도가 로그 변환보다 약함
  - 데이터 값이 모두 양수일 때 사용
  - 해석이 비교적 직관적임

- 거듭제곱 변환 (Power Transformation - Box-Cox, Yeo-Johnson):
  - 데이터 분포를 가장 효과적으로 정규분포에 가깝게 만들 수 있는 최적의 변환을 자동으로 찾아줌
  - Box-Cox: 데이터 값이 모두 양수일 때만 사용 가능
  - Yeo-Johnson: 데이터 값이 음수, 0, 양수 모두 포함될 때 사용 가능
  - 변환된 결과의 해석은 어려움

결론 및 실전 팁

- 데이터 분포 확인 필수: 변환을 결정하기 전에 히스토그램, QQ Plot, 왜도(skewness) 값 등을 확인하여 데이터의 분포 형태를 파악해야 합니다.
- 모델 요구사항 고려: 사용하려는 모델이 데이터의 정규성을 강하게 요구하는지 확인합니다.
- Data Leakage 방지: 스케일링과 마찬가지로, 데이터 변환 시에도 Train 데이터로 `fit()` 하고 Train/Test 데이터에 `transform()` 해야 합니다. (Scikit-learn의 `PowerTransformer`도 이 규칙을 따름)
- 성능 비교: 여러 변환 기법을 적용해보고 모델 성능을 비교하여 최종 선택을 합니다.
- 변환 조합: 경우에 따라 스케일링과 변환을 함께 적용하기도 합니다. (순서 주의: 보통 변환 먼저 -> 스케일링 나중)

요약 정리

- 데이터 변환은 주로 데이터의 분포를 조정하여 모델의 가정을 충족시키고 성능을 향상하는 기법입니다.
- 로그 변환과 제곱근 변환은 오른쪽으로 치우친 데이터를 정규화하고 분산을 안정화하는 데 유용합니다.
- 거듭제곱 변환 (Box-Cox, Yeo-Johnson)은 데이터에 적합한 최적의 변환을 찾아 데이터 분포를 정규분포에 가깝게 만드는데 효과적입니다. 데이터 값의 범위에 따라 적절한 거듭제곱 변환을 선택해야 합니다.
- 변환 시에도 Data Leakage를 방지하기 위해 Train 데이터로 `fit` 후 Train/Test 데이터에 `transform` 해야 합니다.