Skalierungstechniken für Machine Learning: Normalisierung und Standardisierung

Die Normalisierung und Standardisierung sind wichtige Vorverarbeitungsschritte in Data Science, insbesondere für Machine-Learning-Modelle. Beide Techniken haben das Ziel, die Werte der Features in einem Datensatz so zu transformieren, dass sie vergleichbarer und besser für Modelle geeignet sind.

Aspekt	Normalisierung	Standardisierung
Definition	Skalierung der Werte in einen Bereich (z. B. `[0, 1]`).	Zentrierung der Werte auf Mittelwert 0 und Skalierung auf Standardabweichung 1.
Formel	`x' = (x - x_min) / (x_max - x_min)`	`z = (x - μ) / σ`
Ergebnisbereich	Werte liegen in einem festen Bereich (z. B. `[0, 1]` oder `[-1, 1]`).	Verteilungszentrierte Werte, die oft zwischen `-3` und `+3` liegen.
Anwendungsbereich	Verwendung bei Algorithmen, die Distanzen oder Skalen berücksichtigen (z. B. k-NN, SVM).	Verwendung bei Algorithmen, die Normalverteilungen bevorzugen (z. B. PCA, lineare Regression).
Beispiel	Skalieren von Einkommen zwischen `0` und `1`.	Standardisieren von Noten, um sie vergleichbar zu machen.

Verbesserung der Modellleistung

Algorithmen mit Gradientenabstieg: Modelle wie lineare/logistische Regression oder neuronale Netze konvergieren schneller, wenn die Features ähnliche Skalen haben.
Skalensensitivität: Manche Modelle (z. B. k-NN, SVM, PCA) sind skalenempfindlich, da sie Abstände oder Ähnlichkeiten zwischen Datenpunkten berechnen.

Bessere Interpretierbarkeit

Standardisierung kann helfen, Daten verständlicher darzustellen, da alle Features dieselbe Skala haben und die Einheit der Standardabweichung benutzen.

Vermeidung von Verzerrungen

In Features mit großen Zahlenbereichen (z. B. Einkommen in Millionen vs. Alter in Jahren) könnten Modelle die großen Zahlenbereiche stärker gewichten, obwohl sie weniger wichtig sind.

Rücktransformation der Regressionsergebnisse nach Normalisierung

- Die unabhängige Variable X und die abhängige Variable y wurden normalisiert.
- Die Regression wurde auf den normalisierten Daten durchgeführt, wodurch die Koeffizienten β_{norm} und ein Intercept α_{norm} geschätzt wurden.

Rücktransformation bei Z-Score-Normalisierung

Falls die Normalisierung folgendermaßen durchgeführt wurde:

$$X_{\text{norm}} = \frac{X - \mu_X}{\sigma_X}, \quad y_{\text{norm}} = \frac{y - \mu_y}{\sigma_y},$$

wobei μ_X und σ_X den Mittelwert und die Standardabweichung von X und μ_y, σ_y die entsprechenden Werte von y darstellen, erfolgt die Rücktransformation der Koeffizienten wie folgt:

1. Rücktransformierter Koeffizient:

$$\beta = \beta_{\text{norm}} \cdot \frac{\sigma_y}{\sigma_X}$$

2. Rücktransformierter Intercept:

$$\alpha = \mu_y - \beta \cdot \mu_X$$

3. Das endgültige Modell in der ursprünglichen Skalierung lautet:

$$y = \beta \cdot X + \alpha$$

Rücktransformation bei Min-Max-Skalierung

Falls die Normalisierung folgendermaßen durchgeführt wurde:

$$X_{\text{norm}} = \frac{X - \min_X}{\max_X - \min_X}, \quad y_{\text{norm}} = \frac{y - \min_y}{\max_y - \min_y},$$

dann erfolgt die Rücktransformation der Koeffizienten wie folgt:

1. Rücktransformierter Koeffizient:

$$\beta = \beta_{\text{norm}} \cdot \frac{\max_y - \min_y}{\max_X - \min_X}$$

2. Rücktransformierter Intercept:

$$\alpha = \min_y + \alpha_{\text{norm}} \cdot (\max_y - \min_y) - \beta \cdot \min_X$$

3. Das endgültige Modell in der ursprünglichen Skalierung lautet:

$$y = \beta \cdot X + \alpha$$

Wichtig

Für die Rücktransformation ist es entscheidend, die während der Normalisierung verwendeten Parameter (μ_X, σ_X, $\min_X$, $\max_X$, etc.) zu speichern, da sie für die korrekte Berechnung notwendig sind.

Std-Scaler

```python
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression

# Beispiel-Daten
X = np.array([[1], [2], [3], [4], [5]])
y = np.array([3, 4, 2, 5, 6])

# Normalisierung mit StandardScaler
scaler_X = StandardScaler()
scaler_y = StandardScaler()

X_normalized = scaler_X.fit_transform(X)
y_normalized = scaler_y.fit_transform(y.reshape(-1, 1)).flatten()
```

```
# Regression auf normalisierten Daten
model = LinearRegression()
model.fit(X_normalized, y_normalized)

# Koeffizienten und Intercept aus der normalisierten Regression
beta_norm = model.coef_[0]
alpha_norm = model.intercept_

# Rücktransformation
beta = beta_norm * (scaler_y.scale_[0] / scaler_X.scale_[0])
alpha = scaler_y.mean_[0] - beta * scaler_X.mean_[0]

# Modell in der ursprünglichen Skalierung
print(f"y = {beta:.2f} * X + {alpha:.2f}")
```

MinMax-Scaler

```
from sklearn.preprocessing import MinMaxScaler

# Beispiel-Daten
X = np.array([[1], [2], [3], [4], [5]])
y = np.array([3, 4, 2, 5, 6])

# Normalisierung mit MinMaxScaler
scaler_X = MinMaxScaler()
scaler_y = MinMaxScaler()

X_normalized = scaler_X.fit_transform(X)
y_normalized = scaler_y.fit_transform(y.reshape(-1, 1)).flatten()

# Regression auf normalisierten Daten
model = LinearRegression()
model.fit(X_normalized, y_normalized)

# Koeffizienten und Intercept aus der normalisierten Regression
beta_norm = model.coef_[0]
alpha_norm = model.intercept_

# Rücktransformation
beta = beta_norm * (scaler_y.data_range_[0] / scaler_X.data_range_[0])
alpha = scaler_y.data_min_[0] + alpha_norm * scaler_y.data_range_[0] - beta
```

```
# Modell in der ursprünglichen Skalierung
print(f"y = {beta:.2f} * X + {alpha:.2f}")
```

Einfaches Zurückskalierenen der Vorhersagen

```
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression

# Beispiel-Daten
X = np.array([[111], [222], [333], [444], [555]])
y = np.array([33, 44, 22, 55, 66])

# Normalisierung
scaler_X = StandardScaler()
scaler_y = StandardScaler()

X_normalized = scaler_X.fit_transform(X)
y_normalized = scaler_y.fit_transform(y.reshape(-1, 1)).flatten()

# Regression auf normalisierten Daten
model = LinearRegression()
model.fit(X_normalized, y_normalized)

# Neue Daten für Prognose
X_new = np.array([[6], [7]])
X_new_normalized = scaler_X.transform(X_new)

# Prognose in normalisierter Skalierung
y_pred_normalized = model.predict(X_new_normalized)

# Rücktransformation der Prognose
y_pred_original = scaler_y.inverse_transform(y_pred_normalized.reshape(-1,

print(f"Vorhersage (ursprüngliche Skalierung): {y_pred_original.flatten()}"
```