Day49_Feature_Scaling_in_ML

July 18, 2025

Today we are learning about: Feature Scaling in Machine Learning

Feature scaling is one of the most important preprocessing steps in any machine learning pipeline. It transforms your data so that all features have similar scales, which helps models converge faster and perform better.

1 Why Feature Scaling?

Many machine learning algorithms are **distance-based** (like KNN, SVM) or **gradient-based** (like Logistic Regression, Neural Networks). These models are sensitive to the scale of the features.

If you don't scale features, a feature with a large scale will dominate the learning process.

2 Types of Feature Scaling

There are two most common types:

2.1 Normalization (Min-Max Scaling)

- Also called **Min-Max Scaling**.
- Scales the data to a fixed range of **0 to 1**.
- Useful when you **do not** assume a Gaussian (normal) distribution.

Formula:

$$x_{scaled} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

2.2 Example

[5]:
```
import numpy as np
from sklearn.preprocessing import MinMaxScaler
import pandas as pd

# Sample data
data = np.array([[50], [20], [30], [80], [100]])
```

[4]:
```
# Min-Max Scaler
scaler = MinMaxScaler()
```

```
scaled_data = scaler.fit_transform(data)

# Display
df = pd.DataFrame({'Original': data.flatten(), 'MinMax Scaled': scaled_data.
 ↪flatten()})
print(df)
```

```
   Original  MinMax Scaled
0        50          0.375
1        20          0.000
2        30          0.125
3        80          0.750
4       100          1.000
```

2.3 Standardization (Z-score Scaling)

- Also called **Z-score normalization**.
- Transforms the data so that it has a **mean of 0** and a **standard deviation of 1**.
- Useful when data **follows Gaussian distribution**.

Formula:

$$x_{scaled} = \frac{x - \mu}{\sigma}$$

Where:

- μ is the mean
- σ is the standard deviation

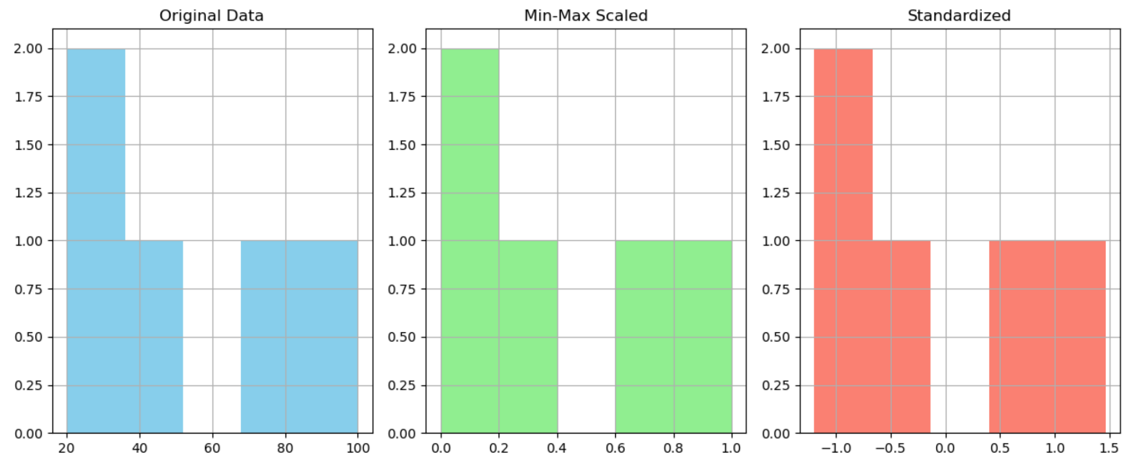
2.4 Example:

[7]:
```
from sklearn.preprocessing import StandardScaler

# Standard Scaler
std_scaler = StandardScaler()
standardized_data = std_scaler.fit_transform(data)

# Display
df_std = pd.DataFrame({'Original': data.flatten(), 'Standardized':␣
 ↪standardized_data.flatten()})
print(df_std)
```

```
   Original  Standardized
0        50     -0.199557
1        20     -1.197342
2        30     -0.864747
3        80      0.798228
4       100      1.463418
```

3 Visualization

[8]:
```python
import matplotlib.pyplot as plt

plt.figure(figsize=(12,5))

# Original Data
plt.subplot(1,3,1)
plt.title("Original Data")
plt.hist(data, bins=5, color='skyblue')
plt.grid()

# Min-Max Scaled
plt.subplot(1,3,2)
plt.title("Min-Max Scaled")
plt.hist(scaled_data, bins=5, color='lightgreen')
plt.grid()

# Standardized
plt.subplot(1,3,3)
plt.title("Standardized")
plt.hist(standardized_data, bins=5, color='salmon')
plt.grid()

plt.tight_layout()
plt.show()
```



4 Notes:

Method	Range	When to Use
Normalization	[0, 1]	When you don't assume normal distribution
Standardization	Mean = 0, SD = 1	When data is Gaussian or you need outlier robustness

5 Summary

- Feature scaling ensures all features contribute equally to the model.
- Normalization is good for non-normal distributions.
- Standardization is good when the data is normally distributed.