06-notebook

March 7, 2025

1 Unsupervised Machine Learning

- A class of machine learning algorithms.

- It **doesn't not have labeled outputs** or a human guiding the learning process. "In unsupervised learning, the learning algorithm is just shown the input data and asked to extract knowledge from this data."
- **While this kind of opinions are common in the community, it's never the entire story!**
- We still need to have some ideas about what to 'look for' in a dataset—more formally, we *often* need a **hypothesis**.
- The algorithm is **only given input data** without predefined answers.

- The goal is to **extract knowledge or patterns** from the input data.

1.1 Types

- **Transformations of the dataset**
  - Create a new representation of the data for better understanding.

  - Dimensionality reduction reduces features while trying to retain essential characteristics.

  - Example:
    * Reducing data to two dimensions for visualization.

    * Topic extraction identifies themes in text documents (e.g., tracking social media discussions on elections).
- **Clustering**
  - Groups similar data points into clusters.

  - Example: Grouping photos of the same person from a bunch of pictures without prior knowledge of identities.

1.2 Challenges

- No predefined labels to verify results. It's difficult—but not impossible—to evaluate performance.
- The algorithm's grouping may differ from human expectations (e.g., sorting faces by angle rather than identity).

1.3 Uses

- **Exploratory data analysis** to better understand data.

- **Preprocessing for supervised learning** to improve accuracy, reduce memory usage. These are often used even in supervised learning (but remain unsupervised in nature).

1.4 Preprocessing

Some preprocessing methods Scikit provides (please check the link for an excellent and elaborate guide!):

- **StandardScaler**
  - $X' = \frac{X-\mu}{\sigma}$ where μ and σ are mean and standard deviation of the feature.
  - Sets each feature's mean to 0 and variance to 1.

  - Brings features to the same magnitude but doesn't enforce specific minimum or maximum values.
- **RobustScaler**
  - Similar to StandardScaler but uses median and quartiles instead of mean and variance.

  - **Ignores outliers**, making it more robust to extreme values.
  - $X' = \frac{X-Q_2}{Q_3-Q_1}$
    * Q_2 = Median (50th percentile)

    * Q_1 = First quartile (25th percentile)

    * Q_3 = Third quartile (75th percentile)

    * $Q_3 - Q_1$ = Interquartile Range (IQR)
- **MinMaxScaler**
  - Scales all features between 0 and 1.

  - Ensures data is contained within a fixed range,
  - it's useful for models that require normalized input values
  - $X' = \frac{X-X_{\text{min}}}{X_{\text{max}}-X_{\text{min}}}$

1.4.1 Median & Interquartile Range (IQR)

Median:

- The middle value of a sorted dataset.

- If there are an even number of values, the median is the average of the two middle values.

- More robust to outliers than the mean.

Interquartile Range (IQR):

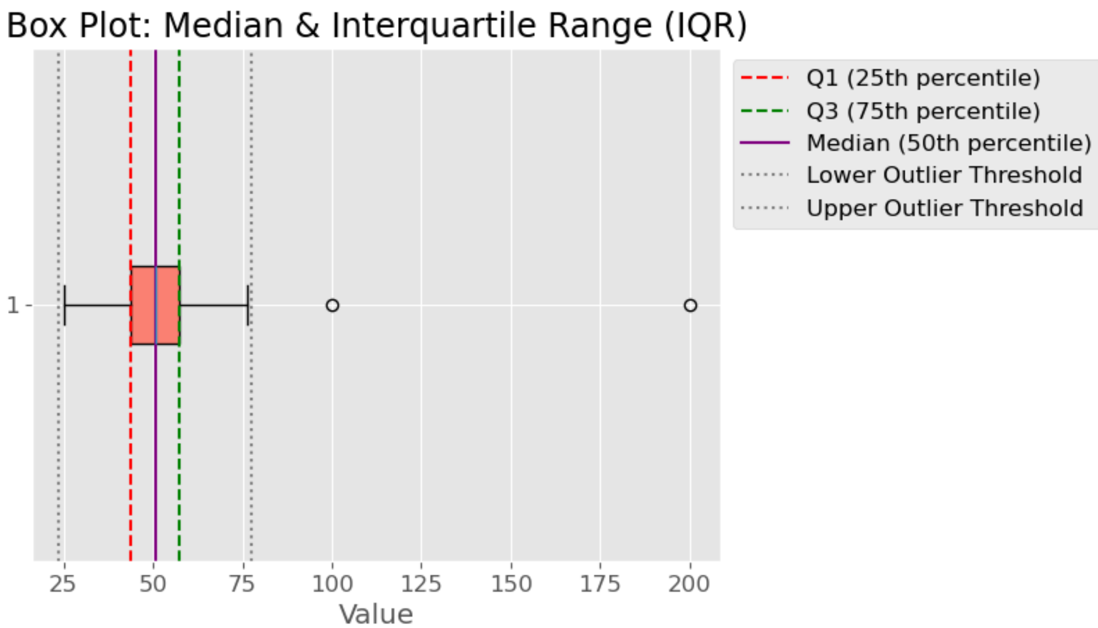
- Measures the spread of the middle 50% of data. IQR = Q3 - Q1 where

  - Q1 (First Quartile): 25th percentile (lower quartile).
  - Q3 (Third Quartile): 75th percentile (upper quartile).
- It's useful for detecting outliers (values below (Q1 - 1.5IQR) or above (Q3 + 1.5IQR) are considered outliers).

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib as mpl

plt.style.use("ggplot")
plt.rcParams["figure.dpi"] = 100
font = {"family": "sans-serif", "size": 12}

mpl.rc("font", **font)
```

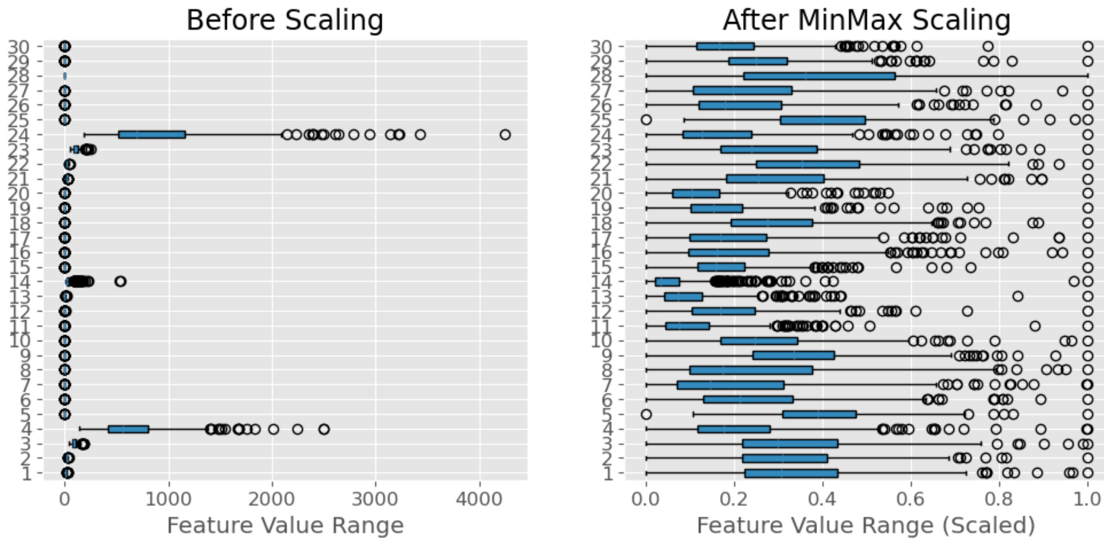
[2]:
```
# median and iqr
# sample data with some outliers
data = np.concatenate([np.random.normal(50, 10, 100), [100, 200]])

# box plot (more: https://en.wikipedia.org/wiki/Box_plot)
plt.boxplot(data, vert=False, patch_artist=True,␣
 ↪boxprops=dict(facecolor="salmon"))

# Calculate Q1, Q3, and IQR
Q1 = np.percentile(data, 25)
Q3 = np.percentile(data, 75)
IQR = Q3 - Q1

# Mark Q1, Q3, median, and outlier thresholds
median = np.median(data)
lower_whisker = Q1 - 1.5 * IQR
upper_whisker = Q3 + 1.5 * IQR

# Plot annotations
plt.axvline(Q1, color="r", linestyle="--", label="Q1 (25th percentile)")
plt.axvline(Q3, color="g", linestyle="--", label="Q3 (75th percentile)")
plt.axvline(median, color="purple", linestyle="-", label="Median (50th␣
 ↪percentile)")
plt.axvline(
    lower_whisker, color="gray", linestyle="dotted", label="Lower Outlier␣
 ↪Threshold"
)
plt.axvline(
    upper_whisker, color="gray", linestyle="dotted", label="Upper Outlier␣
 ↪Threshold"
```

```
)

# Labels and legend
plt.xlabel("Value")
plt.title("Box Plot: Median & Interquartile Range (IQR)")
plt.legend(loc="upper left", bbox_to_anchor=(1, 1));
```



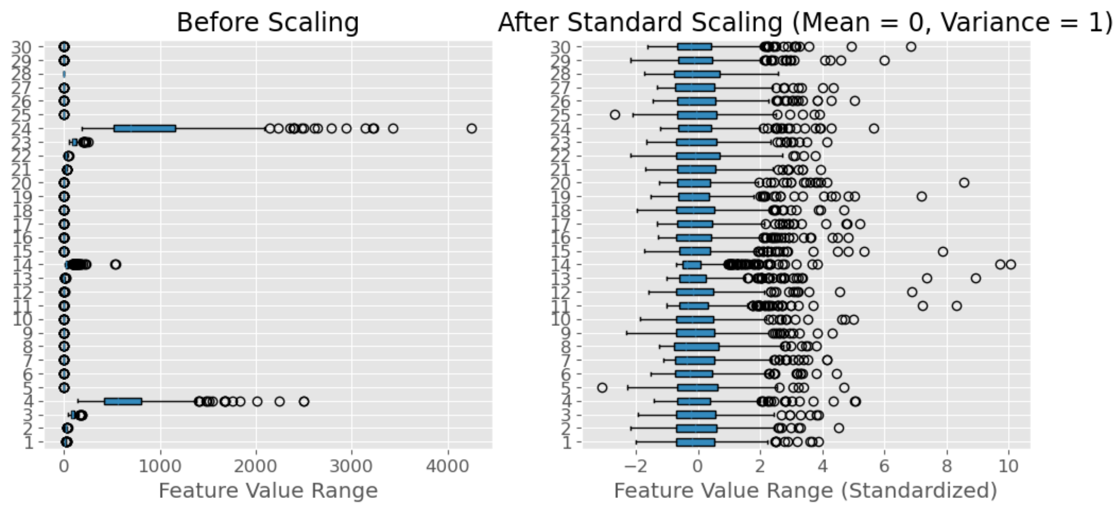
[3]:
```
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler, StandardScaler, RobustScaler

X, y = load_breast_cancer(return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=43)


scaler = MinMaxScaler().fit(X_train)
X_train_scaled = scaler.transform(X_train)

# Plot before and after scaling
fig, axes = plt.subplots(1, 2, figsize=(12, 5))

axes[0].boxplot(X_train, vert=False, patch_artist=True)
axes[0].set_title("Before Scaling")
axes[0].set_xlabel("Feature Value Range")

axes[1].boxplot(X_train_scaled, vert=False, patch_artist=True)
```

```
axes[1].set_title("After MinMax Scaling")
axes[1].set_xlabel("Feature Value Range (Scaled)")

pass;
```

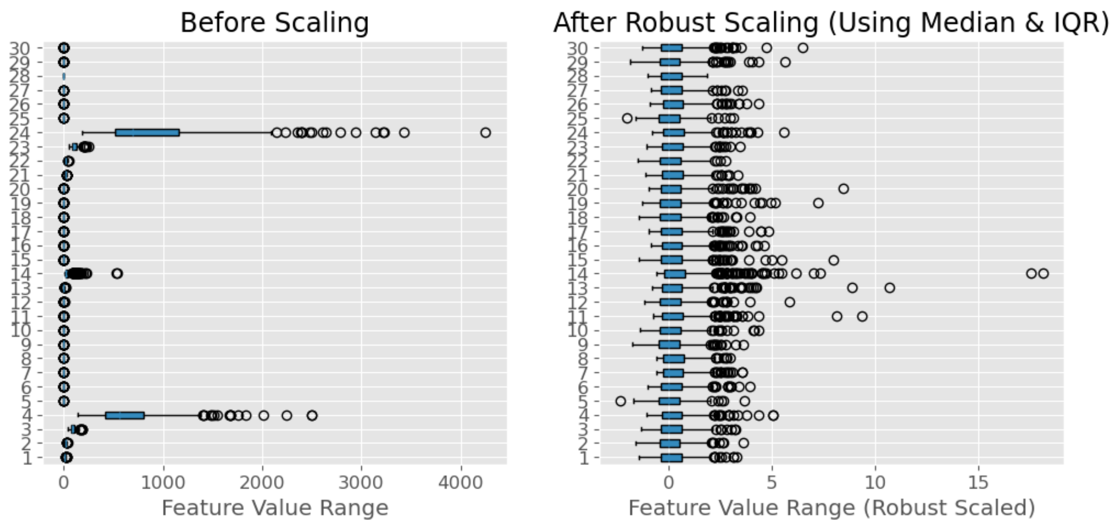


[4]:

```
# Scale data
scaler = StandardScaler().fit(X_train)
X_train_scaled = scaler.transform(X_train)

# Plots before and after scaling
fig, axes = plt.subplots(1, 2, figsize=(12, 5))

axes[0].boxplot(X_train, vert=False, patch_artist=True)
axes[0].set_title("Before Scaling")
axes[0].set_xlabel("Feature Value Range")

axes[1].boxplot(X_train_scaled, vert=False, patch_artist=True)
axes[1].set_title("After Standard Scaling (Mean = 0, Variance = 1)")
axes[1].set_xlabel("Feature Value Range (Standardized)")

pass;
```



[5]:
```python
# Scale data using RobustScaler
scaler = RobustScaler().fit(X_train)
X_train_scaled = scaler.transform(X_train)

# Plots before and after scaling
fig, axes = plt.subplots(1, 2, figsize=(12, 5))

axes[0].boxplot(X_train, vert=False, patch_artist=True)
axes[0].set_title("Before Scaling")
axes[0].set_xlabel("Feature Value Range")

axes[1].boxplot(X_train_scaled, vert=False, patch_artist=True)
axes[1].set_title("After Robust Scaling (Using Median & IQR)")
axes[1].set_xlabel("Feature Value Range (Robust Scaled)")

pass;
```


Before Scaling
Feature Value Range
After Robust Scaling (Using Median & IQR)
Feature Value Range (Robust Scaled)