Machine Learning

Data Visualization

Dept. SW and Communication Engineering

Prof. Giseop Noh (kafa46@hongik.ac.kr)

Lecture Goals

- Understand why visualization is essential in machine learning.
- Learn how to visualize data distributions, feature relationships, and model performance.
- Practice with matplotlib and seaborn to create intuitive plots.

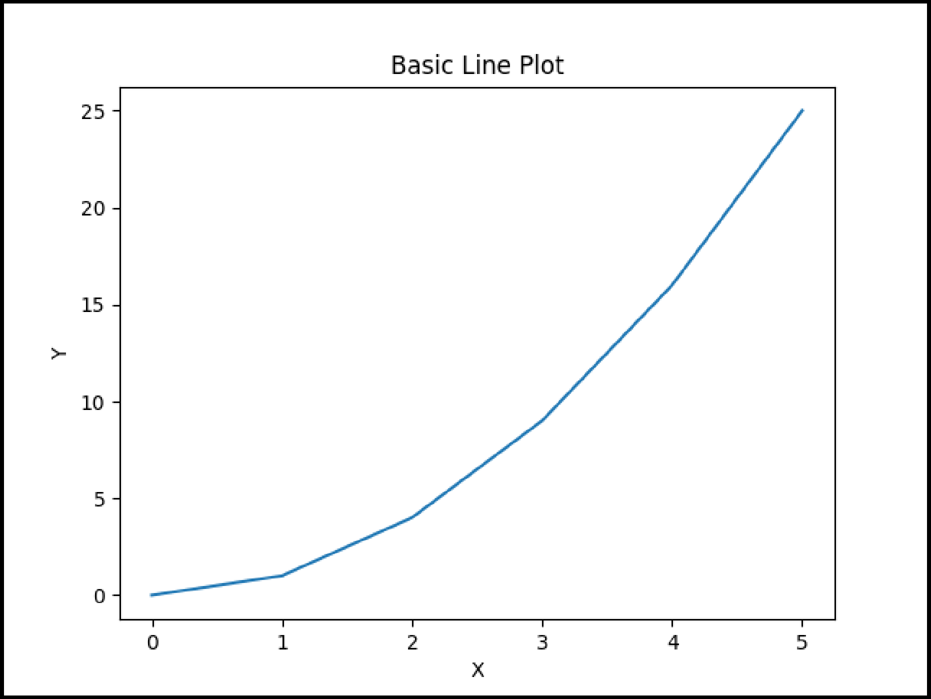
Why Visualization in Machine Learning?

- Visualization is not only for presentation
  - It is essential tool in every stage of ML:
  - Before training
    - Explore data, detect outliers, check feature distributions
  - During training
    - Monitor loss/accuracy curves to diagnose underfitting or overfitting
  - After training
    - Visualize predictions, confusion matrices, and decision boundaries

Basic Line Plot

- The most basic visualization connects the relationship between two variables with a straight line.
- Used to show changes over time or continuous relationships.

```
import matplotlib.pyplot as plt


x = [0, 1, 2, 3, 4, 5]
y = [0, 1, 4, 9, 16, 25]

# 기본 꺾은선 그래프
plt.plot(x, y)
plt.title("Basic Line Plot")
plt.xlabel("X")
plt.ylabel("Y")
plt.show()
```

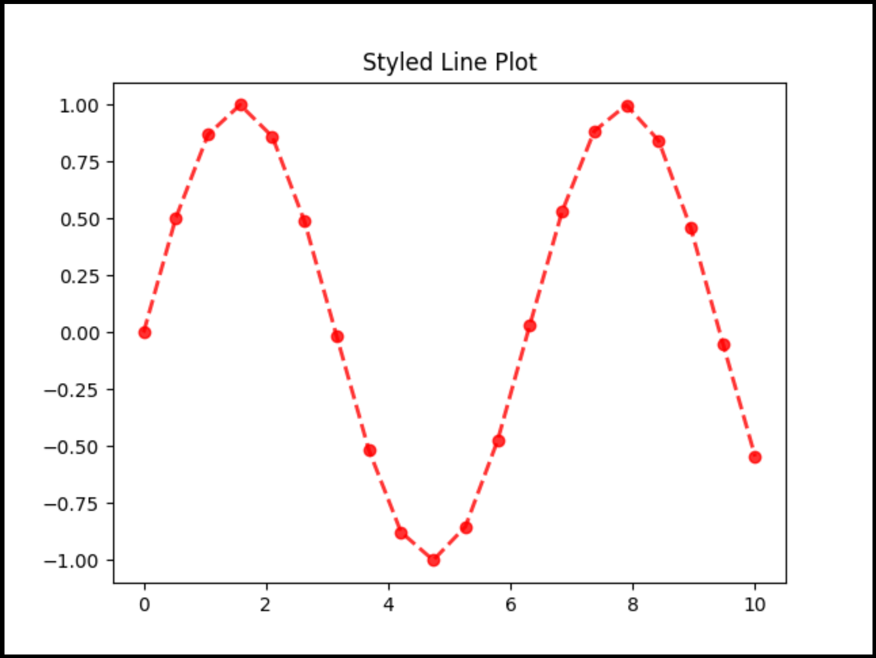


More Options

Option	Example	Description
color	"red", "g", "#FF5733"	Sets the line color
linestyle	"-", "--", ":", "-."	Sets the line style
marker	"o", "s", "^", "d"	Sets the marker shape for data points
linewidth	2, 3	Controls line thickness
alpha	0.5	Controls transparency (0–1)

Line Plot with Options

```
import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(0, 10, 20)
y = np.sin(x)
plt.plot(x, y,
         color="red",
         linestyle="--",
         marker="o",
         linewidth=2,
         alpha=0.8
)
plt.title("Styled Line Plot")
plt.show()
```



Visualizing Data Distribution

Necessity of Visualizing Data Distribution

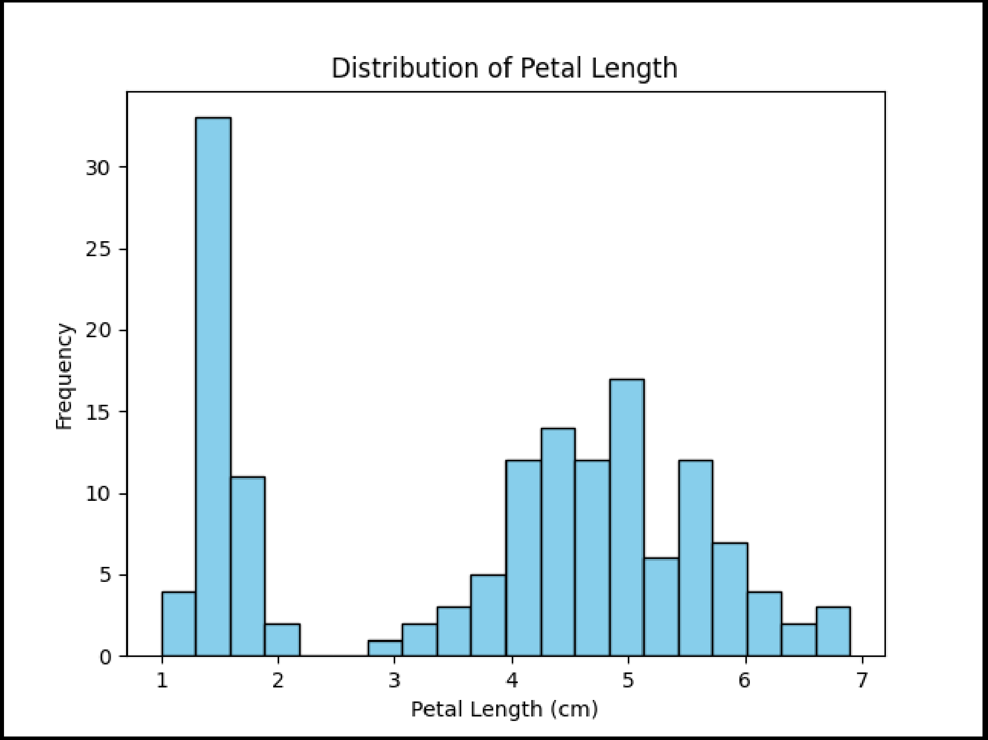
- **Understanding data distribution is a key step in machine learning workflows**
  - Visualization reveals whether data is balanced or skewed
  - Helps detect outliers, clusters, or gaps in values
  - Provides insight for preprocessing:
    - normalization, transformation, or outlier handling
  - Comparing distributions across classes shows if features are informative for classification
  - Reduce risks training on biased or poorly understood data, leading to weak generalization and misleading results

When Visualization is Used in Machine Learning

- **Visualization in ML Workflow**
  - Used from EDA
    - EDA: distributions, missing values, outliers
  - Training
    - Monitor learning (loss/accuracy curves)
  - Evaluation
    - Confusion matrix, ROC, decision boundaries
  - Interpretation
    - Feature importance & contributions
  - Essential: not optional → must-have tool at every stage

Histogram

```
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris

# Load dataset
iris = load_iris(as_frame=True)
df = iris.frame

# Plot histogram of petal length
plt.hist(
    df['petal length (cm)'],
    bins=20,
    color='skyblue',
    edgecolor='black'
)
plt.title("Distribution of Petal Length")
plt.xlabel("Petal Length (cm)")
plt.ylabel("Frequency")
plt.show()
```



Execution Result: Histogram

- The histogram: distribution of values for a single feature.
  - x-axis: the range of values (bins)
  - y-axis represents the frequency of samples in each bin.
- Interpretation
  - Peaks in the histogram: values are concentrated
  - Long tails or isolated bars: maybe outliers.
- In machine learning,
  - Useful for detecting skewness,
  - Checking that data needs normalization or transformation
  - Comparing feature distributions across classes.

Box Plot (1/3)

Necessity of Box Plot

- Compact summary: median, quartiles, outliers
- Easier group comparison than histograms
- Highlights variability, skewness, unusual values

Use in Machine Learning

- EDA: detect outliers, compare feature distributions
- Check need for normalization/transformations
- Assess model residuals → error distribution/bias

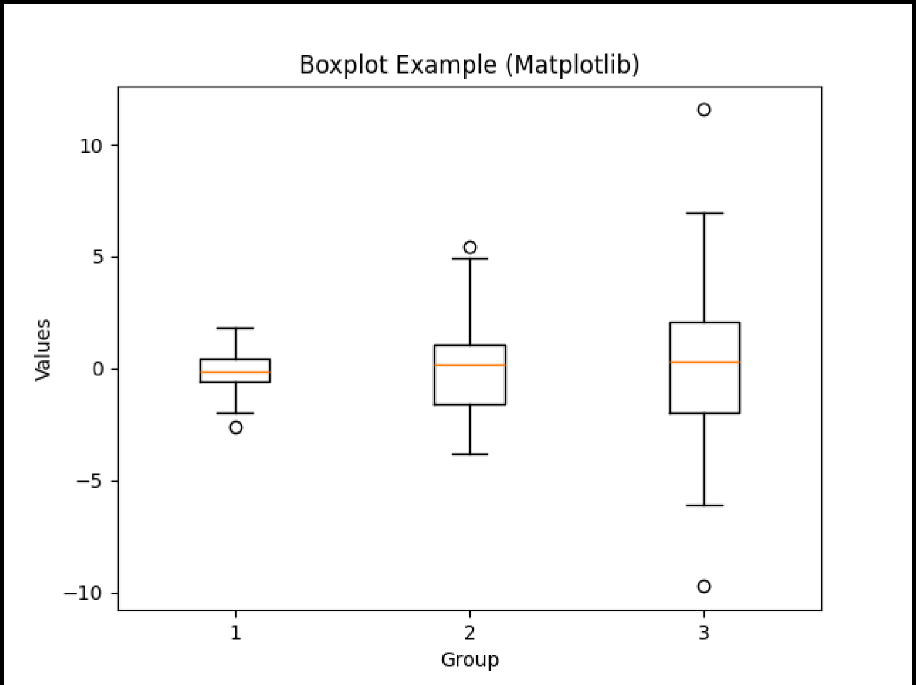
Box Plot (2/3)

```
import matplotlib.pyplot as plt
import numpy as np

# Example data
np.random.seed(42)
data = [np.random.normal(0, std, 100) for std in range(1, 4)]

# Basic boxplot
plt.boxplot(data)
plt.title("Boxplot Example (Matplotlib)")
plt.xlabel("Group")
plt.ylabel("Values")
plt.boxplot(data,
            notch=True,        # median 주변 신뢰구간 notch 추가
            vert=True,         # 세로(True)/가로(False) 방향
            patch_artist=True, # 박스 색칠 가능
            boxprops=dict(facecolor="lightblue", color="blue"),  # 박스 스타일
            medianprops=dict(color="red", linewidth=2))          # 중앙선 스타일
plt.title("Customized Boxplot")
plt.show()
```

Box Plot (3/3)

Execution Result: Boxplot

- The box shows the interquartile range (IQR = Q1 to Q3).



- The line inside the box: median
- The whiskers extend to the minimum and maximum values within 1.5 × IQR.
- Points outside the whiskers: potential outliers.

Comparing multiple boxes side by side: quick detection of group differences

Visualizing Feature Relationships

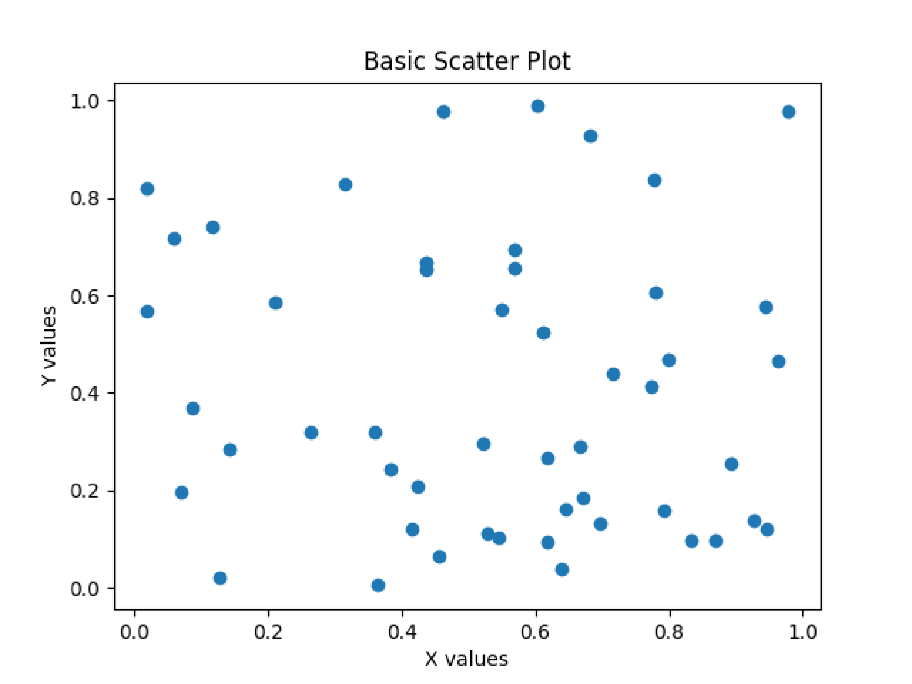
Scatter Plot (1/4)

- **Necessity of Scatter Plot**
  - Reveals relationships between two numerical variables
  - Detects linear/nonlinear trends, clusters, anomalies
  - Preserves individual data points → raw data insight

- **Use in Machine Learning**
  - EDA: check correlations, class separability
  - Identify predictive features via clusters
  - Visualize dimensionality reduction (PCA, t-SNE)

Scatter Plot (2/4)

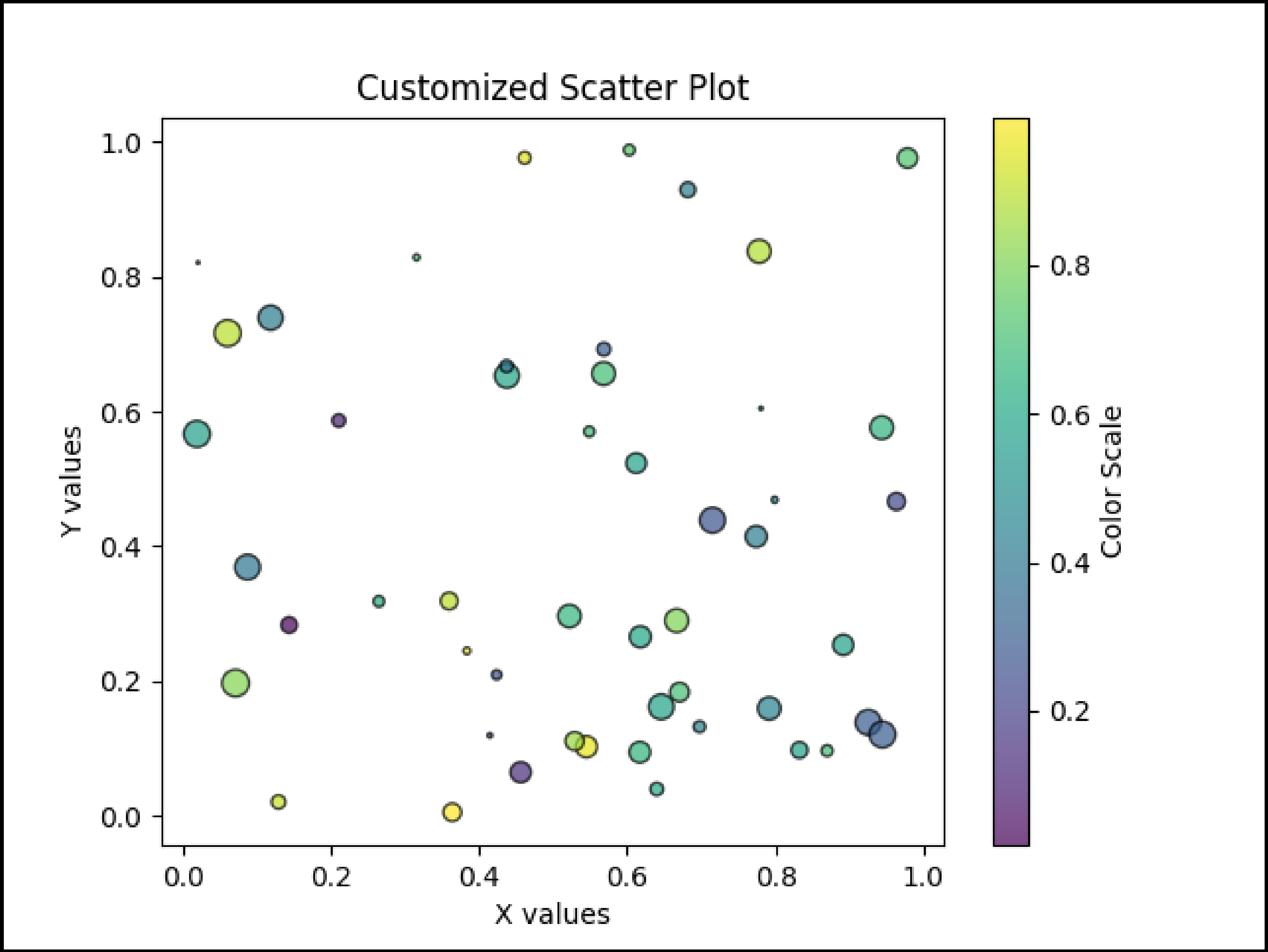
```python
import matplotlib.pyplot as plt
import numpy as np


# Example data
np.random.seed(0)
x = np.random.rand(50)   # 0~1 사이 난수 50개
y = np.random.rand(50)


# Scatter plot with options
colors = np.random.rand(50)   # Color array
sizes = 100 * np.random.rand(50)  # Size array


# Basic scatter plot
plt.scatter(x, y)
plt.title("Basic Scatter Plot")
plt.xlabel("X values")
plt.ylabel("Y values")
plt.show()


# Scatter Plot with Custom Style
# plt.scatter(x, y, c=colors, s=sizes, alpha=0.7, cmap="viridis", edgecolors="black")
# plt.title("Customized Scatter Plot")
# plt.xlabel("X values")
# plt.ylabel("Y values")
# plt.colorbar(label="Color Scale")
# plt.show()
```



Scatter Plot (3/4) - with Customization

Parameter	Description	Example
x, y	Data coordinates for points	plt.scatter(x, y)
c	Color(s) of points (single value or array)	plt.scatter(x, y, c="red")
cmap	Colormap when c is an array	plt.scatter(x, y, c=z, cmap="viridis")
s	Marker size (single value or array)	plt.scatter(x, y, s=50)
marker	Shape of markers (e.g., "o", "s", "^")	plt.scatter(x, y, marker="^")
alpha	Transparency (0 = invisible, 1 = solid)	plt.scatter(x, y, alpha=0.5)
edgecolors	Color of marker edges	plt.scatter(x, y, edgecolors="black")
linewidths	Thickness of marker edge	plt.scatter(x, y, edgecolors="black", linewidths=2)
label	Label for legend	plt.scatter(x, y, label="Group A")

Scatter Plot (4/4) - Customized Result



Pairplot (1/3)

Necessity of Pairplot

- Shows feature distributions (diagonal) + pairwise relationships (off-diagonal)
- Detects correlations, clusters, hidden patterns
- Provides a holistic view across multiple features

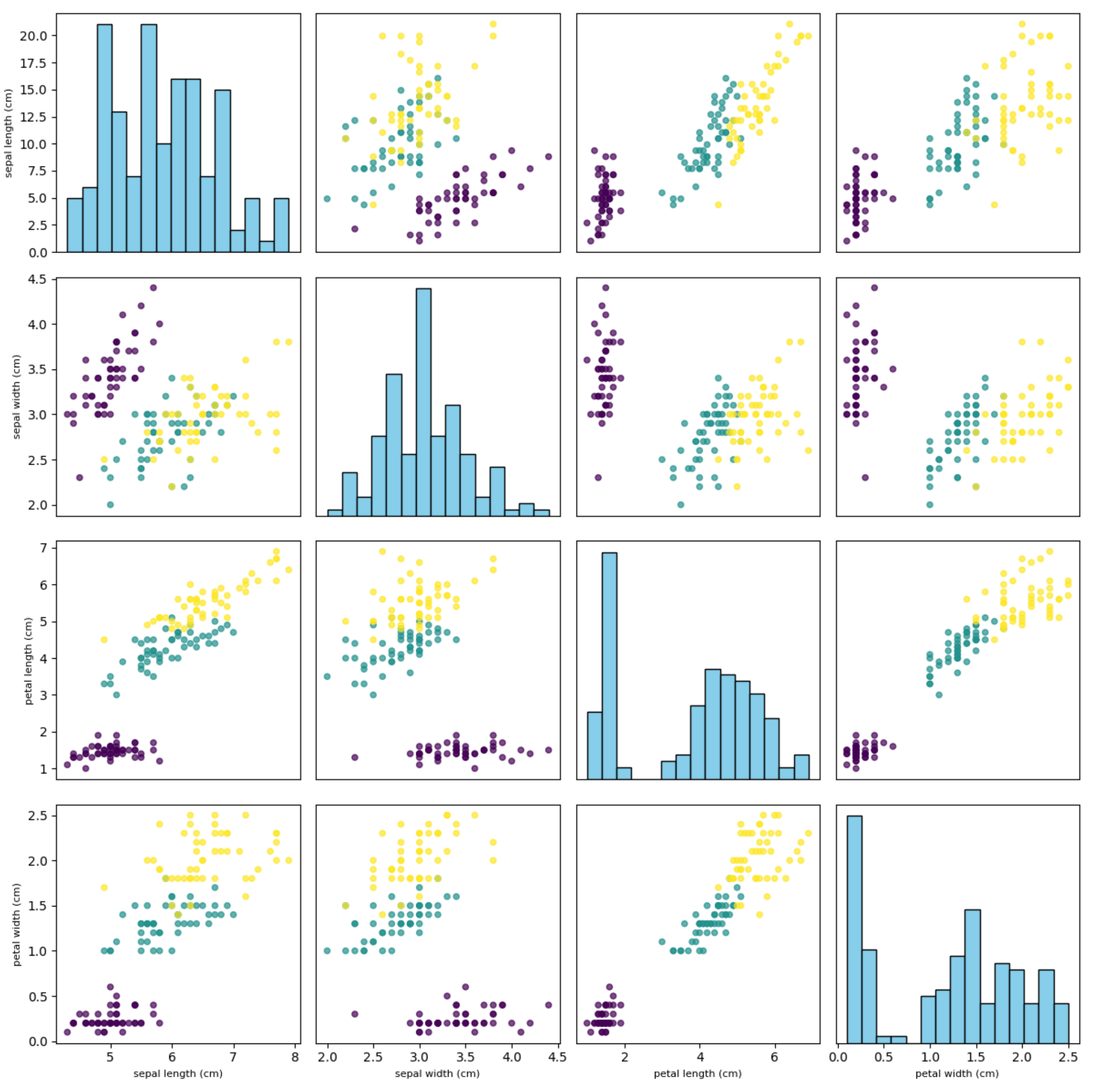
Use in Machine Learning

- EDA: evaluate feature interactions & class separability
- Useful for small/medium datasets
- Supports feature selection & dimensionality reduction
- Identifies redundant features & nonlinear relationships

Pairplot (2/3)

```
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.datasets import load_iris

# Load Iris dataset
iris = load_iris(as_frame=True)
df = iris.frame

features = ["sepal length (cm)", "sepal width (cm)",
            "petal length (cm)", "petal width (cm)"]
target = df["target"]

# Number of features
n = len(features)

fig, axes = plt.subplots(n, n, figsize=(12, 12))

for i in range(n):
    for j in range(n):
        ax = axes[i, j]
        if i == j:
            # Diagonal: histogram
            ax.hist(df[features[i]], bins=15, color="skyblue", edgecolor="black")
        else:
            # Off-diagonal: scatter plot
            scatter = ax.scatter(df[features[j]], df[features[i]],
                                 c=target, cmap="viridis", s=20, alpha=0.7)
        if i == n-1:
            ax.set_xlabel(features[j], fontsize=8)
        else:
            ax.set_xticks([])
        if j == 0:
            ax.set_ylabel(features[i], fontsize=8)
        else:
            ax.set_yticks([])

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

Pairplot (3/3)



- Diagonal plots: show the distribution of each feature (often histograms).
- Off-diagonal plots: show scatter plots of feature pairs.
- Cluster patterns: reveal whether certain features group naturally.
- Overlaps: suggest features may not fully distinguish between classes.

Visualizing Model Training

Loss Curves (1/3)

Necessity of Loss Curves

- Show how model error changes during training
- Detect convergence, overfitting, underfitting
- Evaluate optimization stability & hyperparameters

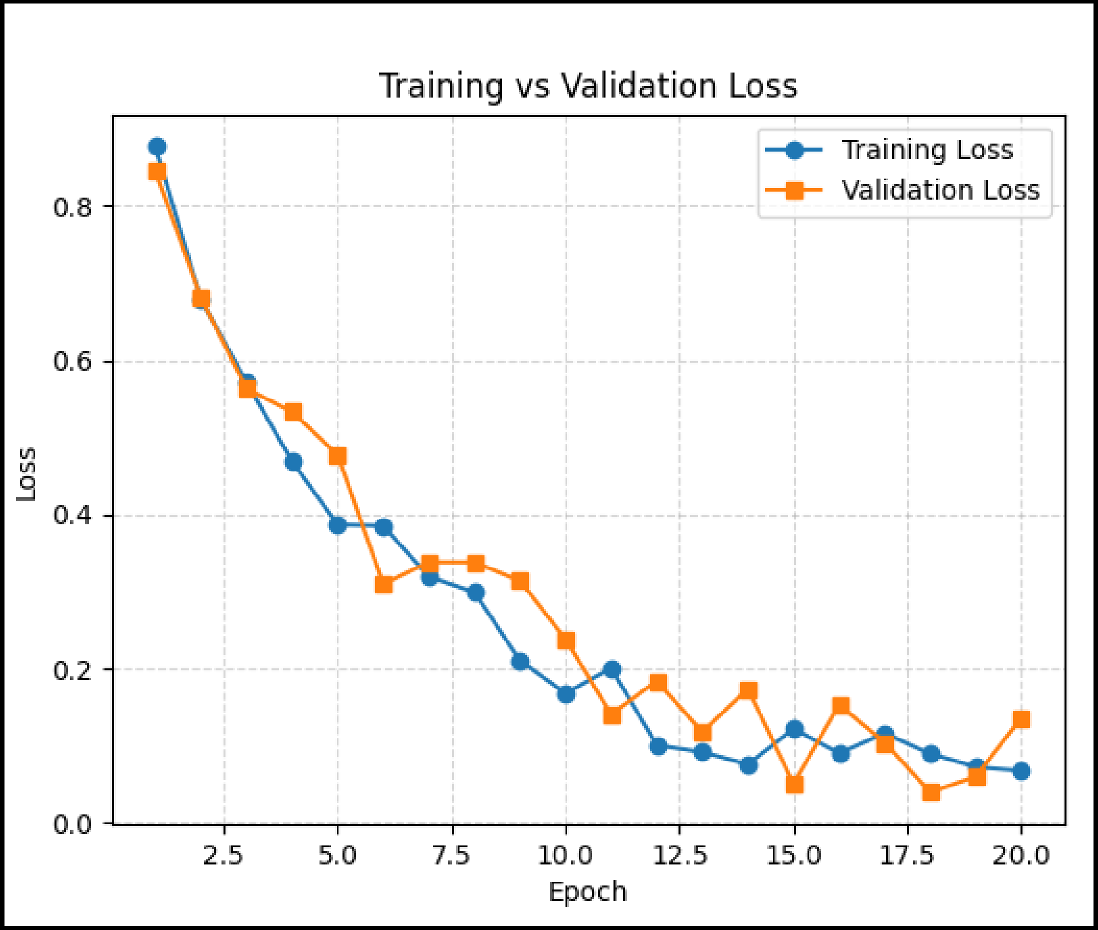
Use in Machine Learning

- Training: monitor loss/accuracy trends
- Validation: compare training vs validation loss
  - Both $\downarrow \rightarrow$ good learning
  - Train $\downarrow$ & Val $\uparrow \rightarrow$ overfitting
  - Both high $\rightarrow$ underfitting
- Guide adjustments: hyperparameters, model design, preprocessing

Loss Curves (2/3)

```
import matplotlib.pyplot as plt
import numpy as np

# Simulated loss values for 20 epochs
epochs = np.arange(1, 21)
train_loss = np.exp(-epochs/5) + 0.1 * np.random.rand(20)   # decreasing trend
val_loss = np.exp(-epochs/5) + 0.15 * np.random.rand(20)    # validation with noise

# Plot curves
plt.plot(epochs, train_loss, label="Training Loss", marker="o")
plt.plot(epochs, val_loss, label="Validation Loss", marker="s")

# Add labels and title
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.title("Training vs Validation Loss")
plt.legend()
plt.grid(True, linestyle="--", alpha=0.5)
plt.show()
```

Loss Curves (3/3)



- Training loss decreases steadily if the model is fitting the data.
- Validation loss indicates generalization to unseen data.
- Gap between training and validation loss reveals overfitting.
- Flat curves suggest poor learning or too low learning rate.
- Rapid oscillations may indicate instability or too high learning rate.

Accuracy Curves

Necessity of Accuracy Curves

- Track classification performance over time
- Complement loss curves with an interpretable metric
- Detect convergence, overfitting, underfitting

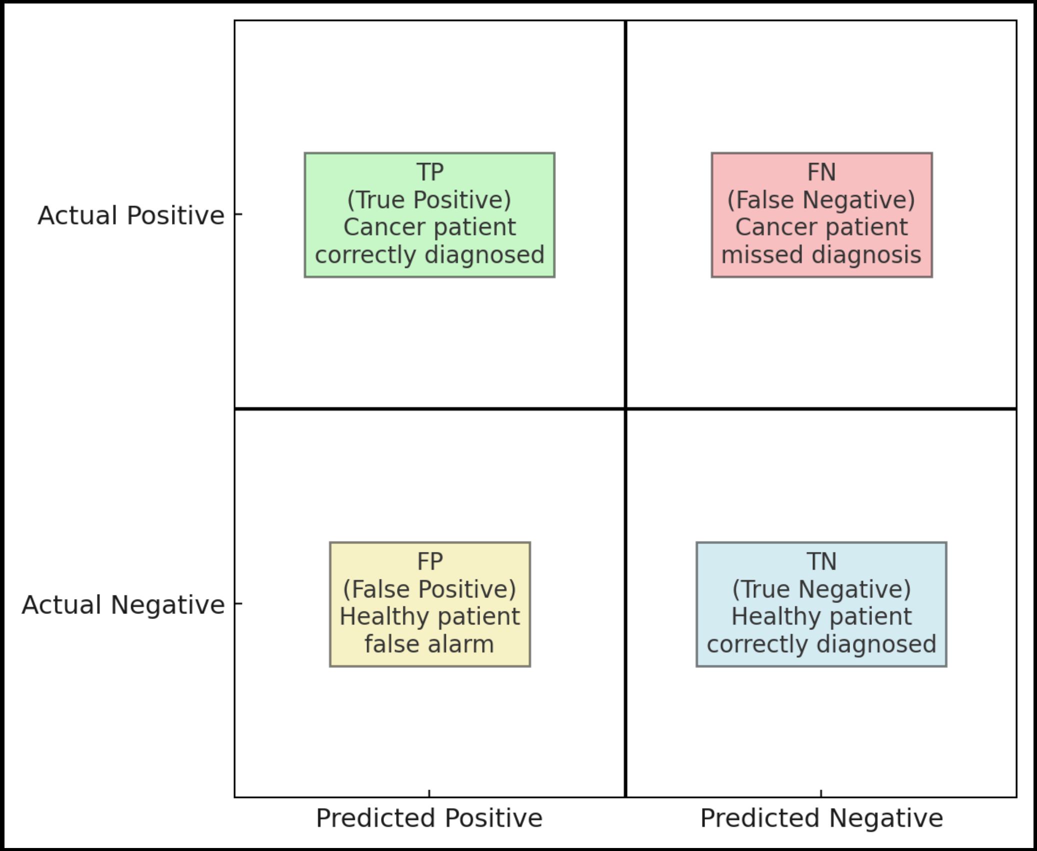
Use in Machine Learning

- Monitor training & validation accuracy during model training
- Compare train vs validation performance
  - Both $\uparrow$ $\rightarrow$ good learning
  - Train $\uparrow$ & Val $\downarrow$ $\rightarrow$ overfitting
  - Both low $\rightarrow$ underfitting
- Compare models & training strategies for better generalization

Accuracy Curves (1/3)

Accuracy (Evaluation Metric)

- Measures proportion of correctly predicted samples, Simple & intuitive metric
- Limitation: can be misleading with imbalanced datasets



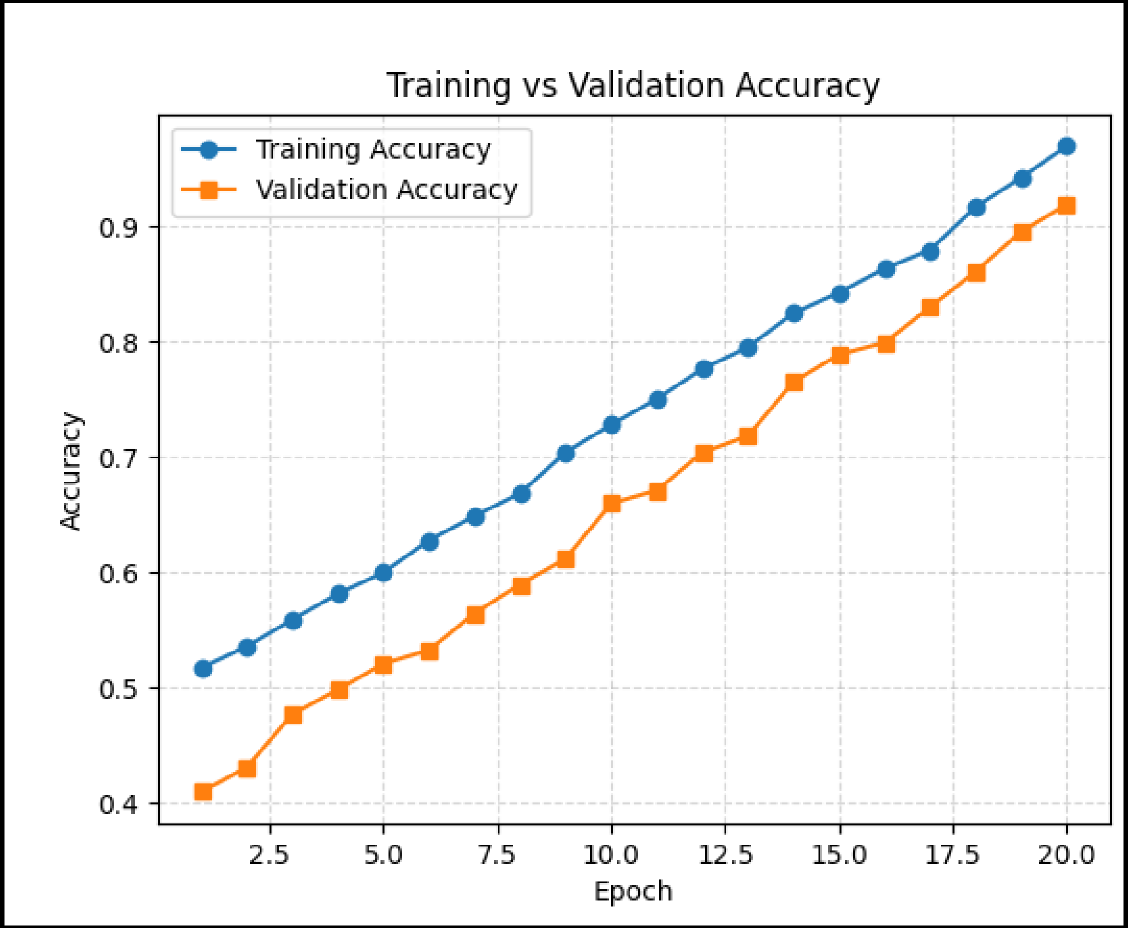
$$Accuracy = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Predictions}}$$

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

Accuracy Curves (2/3)

```
import matplotlib.pyplot as plt
import numpy as np


# Simulated accuracy values for 20 epochs
epochs = np.arange(1, 21)
train_acc = np.linspace(0.5, 0.95, 20) + 0.02 *
np.random.rand(20)  # increasing trend
val_acc = np.linspace(0.4, 0.9, 20) + 0.03 *
np.random.rand(20)     # validation with noise


# Plot curves
plt.plot(epochs, train_acc, label="Training Accuracy", marker="o")
plt.plot(epochs, val_acc, label="Validation Accuracy", marker="s")


# Add labels and title
plt.xlabel("Epoch")
plt.ylabel("Accuracy")
plt.title("Training vs Validation Accuracy")
plt.legend()
plt.grid(True, linestyle="--", alpha=0.5)
plt.show()
```

Accuracy Curves (3/3)



- Training accuracy increases as the model learns patterns in the training data.
- Validation accuracy reflects performance on unseen data.
- Parallel rise of both curves indicates effective learning.
- Large gap between curves suggests overfitting.
- Flat or low curves point to underfitting or insufficient model complexity.

Visualizing Model Performance

Confusion Matrix (1/6)

Confusion Matrix

- Compares predicted labels vs actual labels
- Shows both correct predictions and types of errors
- Provides deeper insight than accuracy alone

Importance in Machine Learning

- Essential for imbalanced datasets
- Reveals class-specific performance
- Helps identify where the model makes mistakes

Confusion Matrix (2/6)

Necessity of Confusion Matrix

- Provides detailed view beyond accuracy
- Shows predictions by true vs predicted categories
- Identifies specific error types hidden in accuracy

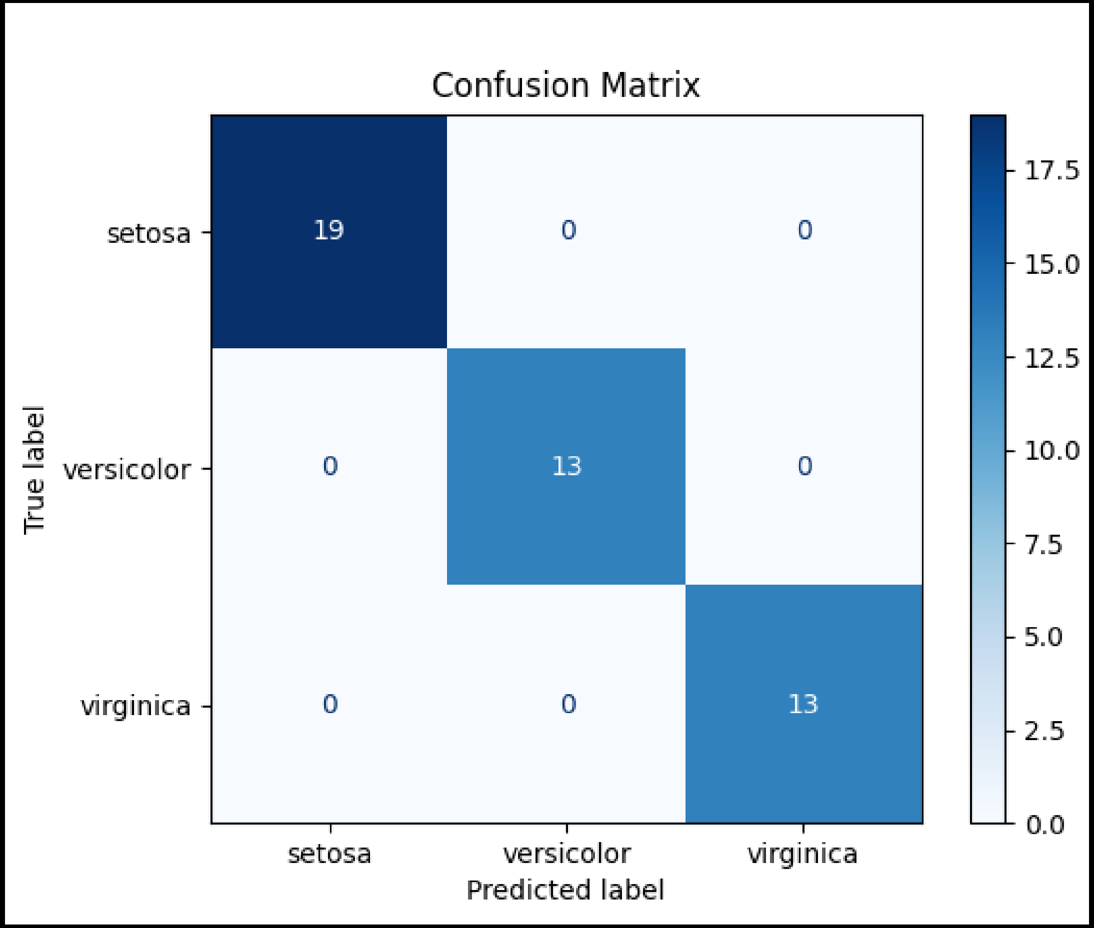
Use in Machine Learning

- Evaluate model performance on test data
- Critical for imbalanced datasets
- Basis for calculating Precision, Recall, F1-score
- Gives a complete picture of classifier behavior

Confusion Matrix (3/6)

```python
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

# Load dataset
iris = load_iris(as_frame=True)
X = iris.data
y = iris.target

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Train a simple classifier
clf = LogisticRegression(max_iter=200)
clf.fit(X_train, y_train)

# Predictions
y_pred = clf.predict(X_test)

# Confusion Matrix
cm = confusion_matrix(y_test, y_pred, labels=clf.classes_)

# Display with Matplotlib
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=iris.target_names)
disp.plot(cmap="Blues")
plt.title("Confusion Matrix")
plt.show()
```

Confusion Matrix (4/6)

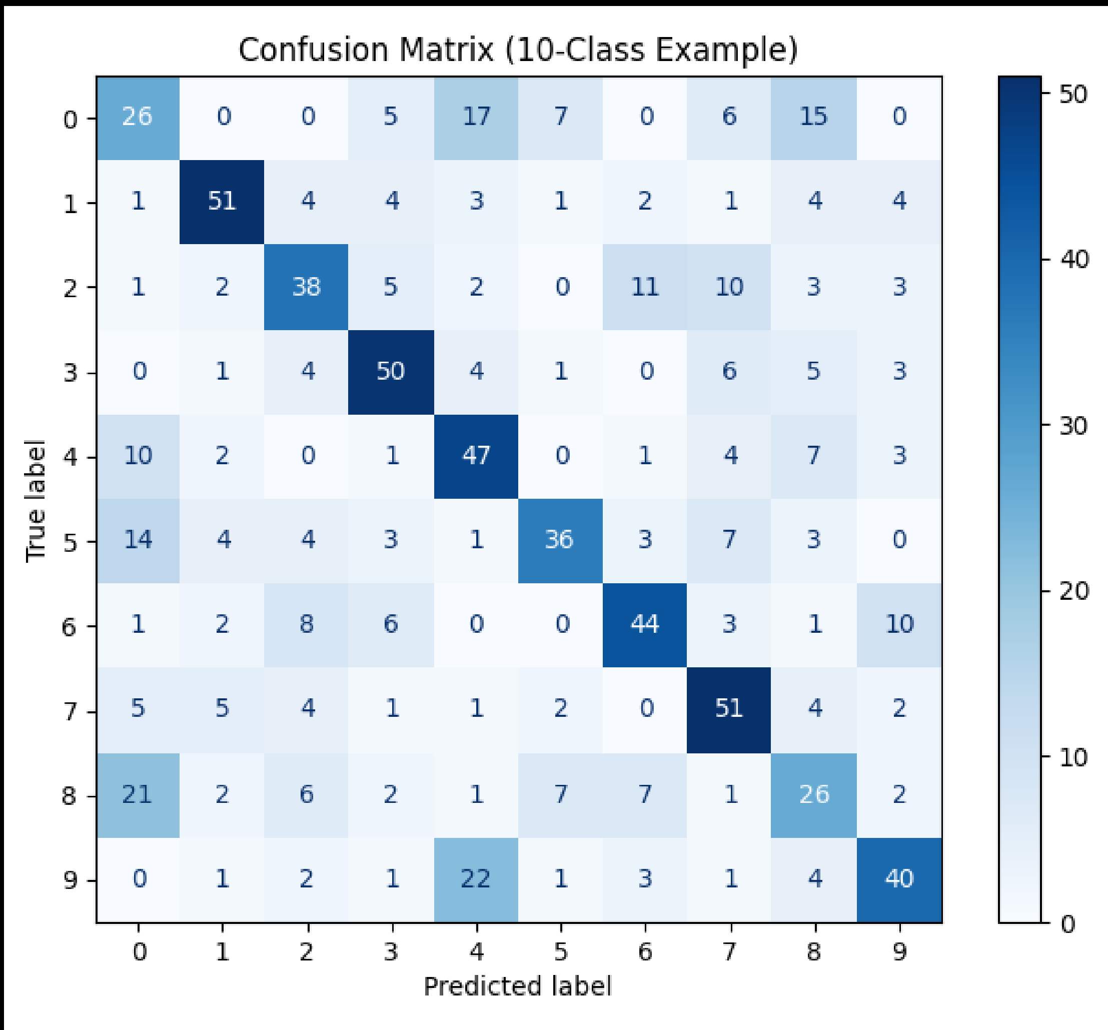


- Diagonal values → correctly classified samples.
- Off-diagonal values → misclassified samples.
- Rows → true class labels.
- Columns → predicted class labels.
- Helps to detect class-specific weaknesses (e.g., frequent misclassification between two similar classes).

Confusion Matrix (5/6) - Imbalance Dataset Example

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay, classification_report

# 1) Create a 10-class dataset
X, y = make_classification(
    n_samples=3000,
    n_features=15,
    n_informative=10,
    n_redundant=2,
    n_classes=10,
    n_clusters_per_class=1,
    random_state=42
)

# 2) Train-test split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.25, stratify=y, random_state=42
)

# 3) Train a classifier
clf = LogisticRegression(max_iter=2000, multi_class="multinomial", solver="lbfgs")
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)

# 4) Compute confusion matrix
cm = confusion_matrix(y_test, y_pred)

# 5) Plot confusion matrix
fig, ax = plt.subplots(figsize=(8, 6))
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=np.arange(10))
disp.plot(ax=ax, cmap="Blues", colorbar=True, values_format="d")
plt.title("Confusion Matrix (10-Class Example)")
plt.show()

# 6) Print classification report
print("Classification Report (per-class precision/recall/F1):")
print(classification_report(y_test, y_pred))
```

Confusion Matrix (6/6) - Imbalance Dataset Results



- With 10 classes, the confusion matrix is a 10×10 grid.
- Diagonal values → correctly classified samples for each class.
- Off-diagonal values → misclassifications, showing which classes get confused with each other.
- This helps students see that accuracy alone may not explain which classes are “harder” to classify.
- The classification report complements the confusion matrix by providing precision, recall, and F1-score for each class, which is crucial when some classes are systematically harder to predict.

ROC Curve

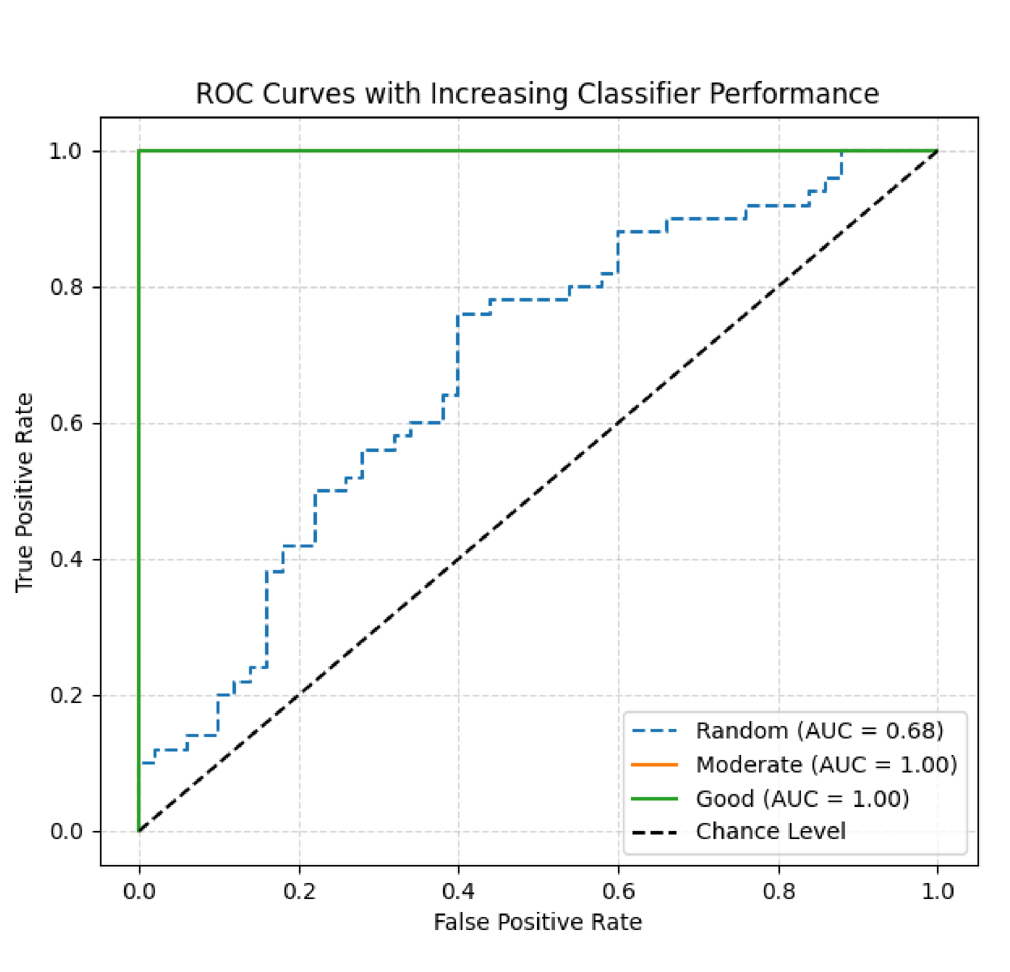
- **Necessity of ROC (Receiver Operating Charateristic) Curve**
  - Shows trade-off: Sensitivity (TPR) vs False Positive Rate (FPR)
  - Evaluates performance across thresholds
  - Useful for imbalanced datasets
  - Helps compare false positives vs false negatives

- **When ROC Curve is Used in Machine Learning**
  - After training binary classifiers (extendable to multiclass)
  - Evaluate class separation under varying thresholds
  - Widely used for model comparison
  - Closer to top-left or higher AUC → better model

ROC Curve

```python
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve, auc
import numpy as np

# Example binary labels
y_true = np.array([0]*50 + [1]*50)  # 50 negatives, 50 positives

# Simulated prediction scores
y_score_random   = np.random.rand(100)                # random predictions
y_score_medium   = np.linspace(0.2, 0.8, 100)         # moderate separation
y_score_good     = np.concatenate([np.linspace(0,0.3,50), np.linspace(0.7,1,50)])  # good separation

# Compute ROC curves
fpr_r, tpr_r, _ = roc_curve(y_true, y_score_random)
fpr_m, tpr_m, _ = roc_curve(y_true, y_score_medium)
fpr_g, tpr_g, _ = roc_curve(y_true, y_score_good)

# Compute AUC
auc_r = auc(fpr_r, tpr_r)
auc_m = auc(fpr_m, tpr_m)
auc_g = auc(fpr_g, tpr_g)

# Plot
plt.figure(figsize=(7,6))
plt.plot(fpr_r, tpr_r, label=f"Random (AUC = {auc_r:.2f})", linestyle="--")
plt.plot(fpr_m, tpr_m, label=f"Moderate (AUC = {auc_m:.2f})")
plt.plot(fpr_g, tpr_g, label=f"Good (AUC = {auc_g:.2f})")
plt.plot([0,1],[0,1],"k--", label="Chance Level")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curves with Increasing Classifier Performance")
plt.legend()
plt.grid(True, linestyle="--", alpha=0.5)
plt.show()
```

ROC Curve



- X-axis: False Positive Rate (FPR).
- Y-axis: True Positive Rate (TPR).
- Diagonal line: random guessing baseline (AUC = 0.5).
- Curve closer to the top-left corner → stronger classifier.
- AUC value quantifies the overall ability:
  - 0.0 ~ 0.5 = no discrimination (random).
  - 0.7 ~ 0.8 = acceptable.
  - 0.8 ~ 0.9 = excellent.
  - 0.9 ~ 1.0 = outstanding.

Advanced Visualizations

Decision Boundary

Decision Boundary in Machine Learning

- A surface that separates classes in the feature space
- 2D: appears as a line
- 3D: a plane or curved surface
- Higher dimensions: a $(d-1)$-dimensional hypersurface
- Cannot be directly visualized in higher dimensions, but critical for understanding model decisions

Decision Boundary

Necessity of Decision Boundary

- Visualizes how a classifier separates classes in feature space
- Provides intuitive picture of model predictions
- Helps detect overfitting, underfitting, or poor separation

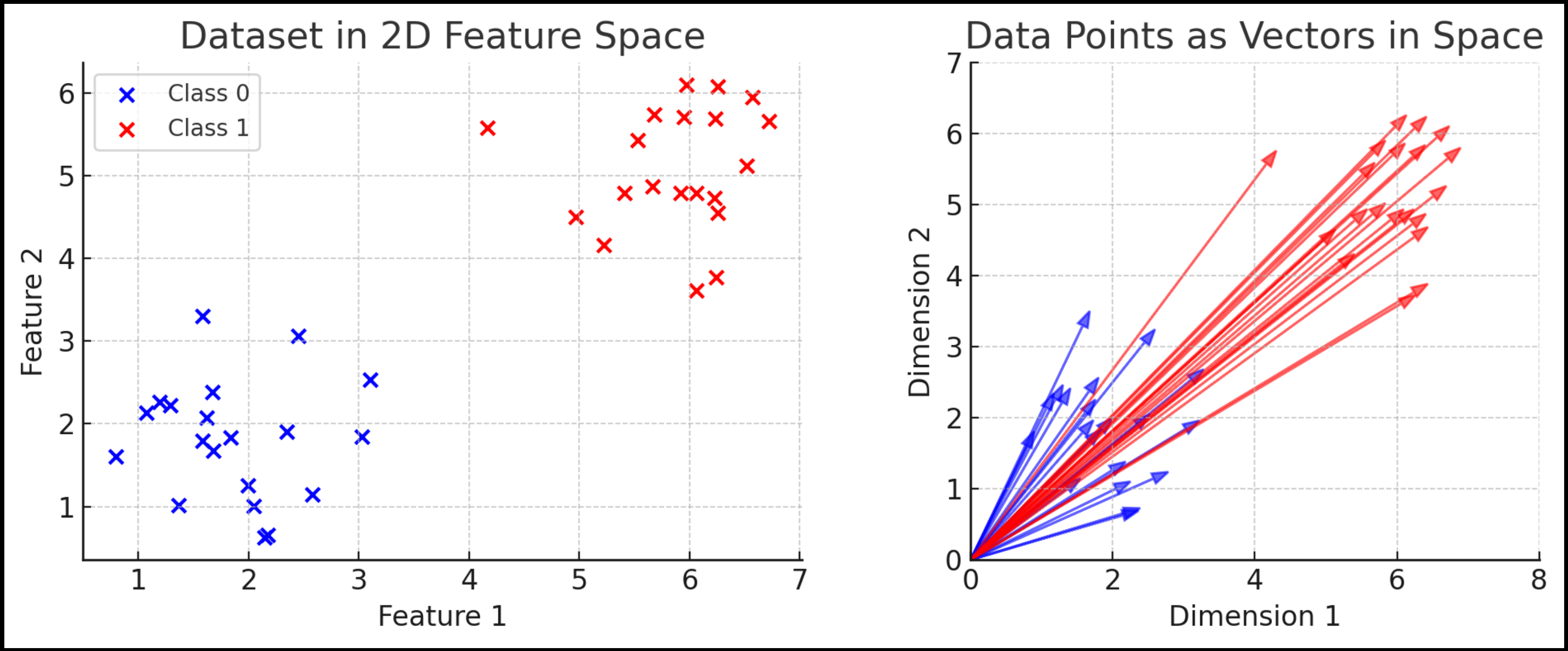
When Decision Boundary is Used in Machine Learning

- Mainly for classification tasks with 2–3 features (visualizable)
- Useful for teaching & model interpretation
- Shows how algorithms (e.g., Logistic Regression, SVM, Decision Tree) divide the input space
- Used to compare classifiers and evaluate partitioning

Data as Vectors

Each data point can be represented as a vector in an n-dimensional space, where n is the number of features.

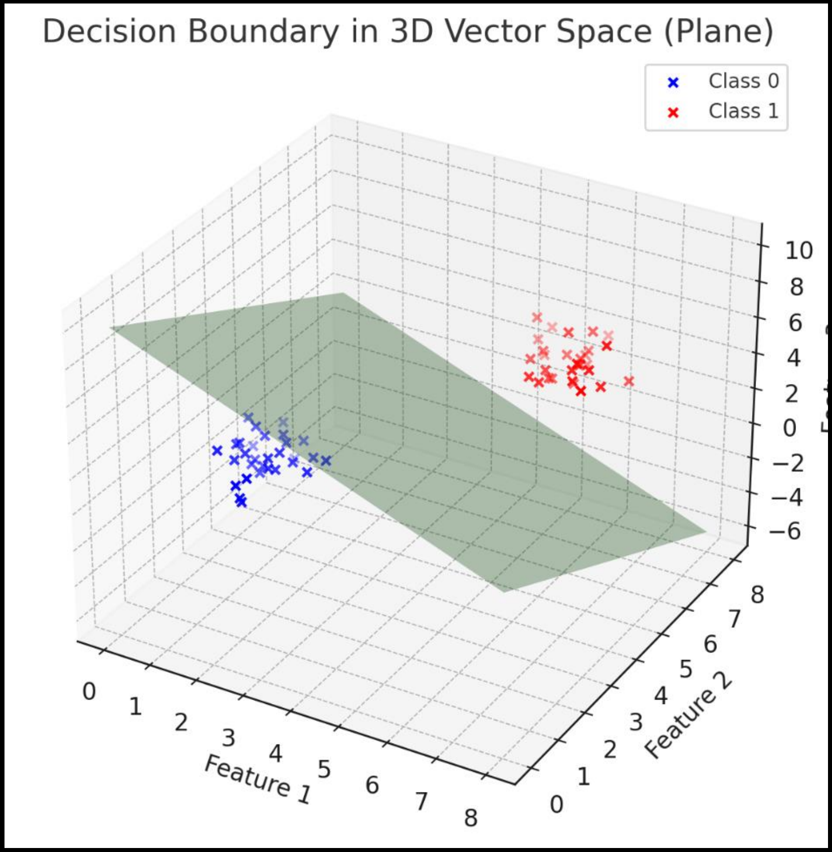
For example, if a sample has 100 features, it corresponds to a vector in $\mathbb{R}^{100}$.



Decision Boundary as a hypersurface

- A classifier divides this space into regions.

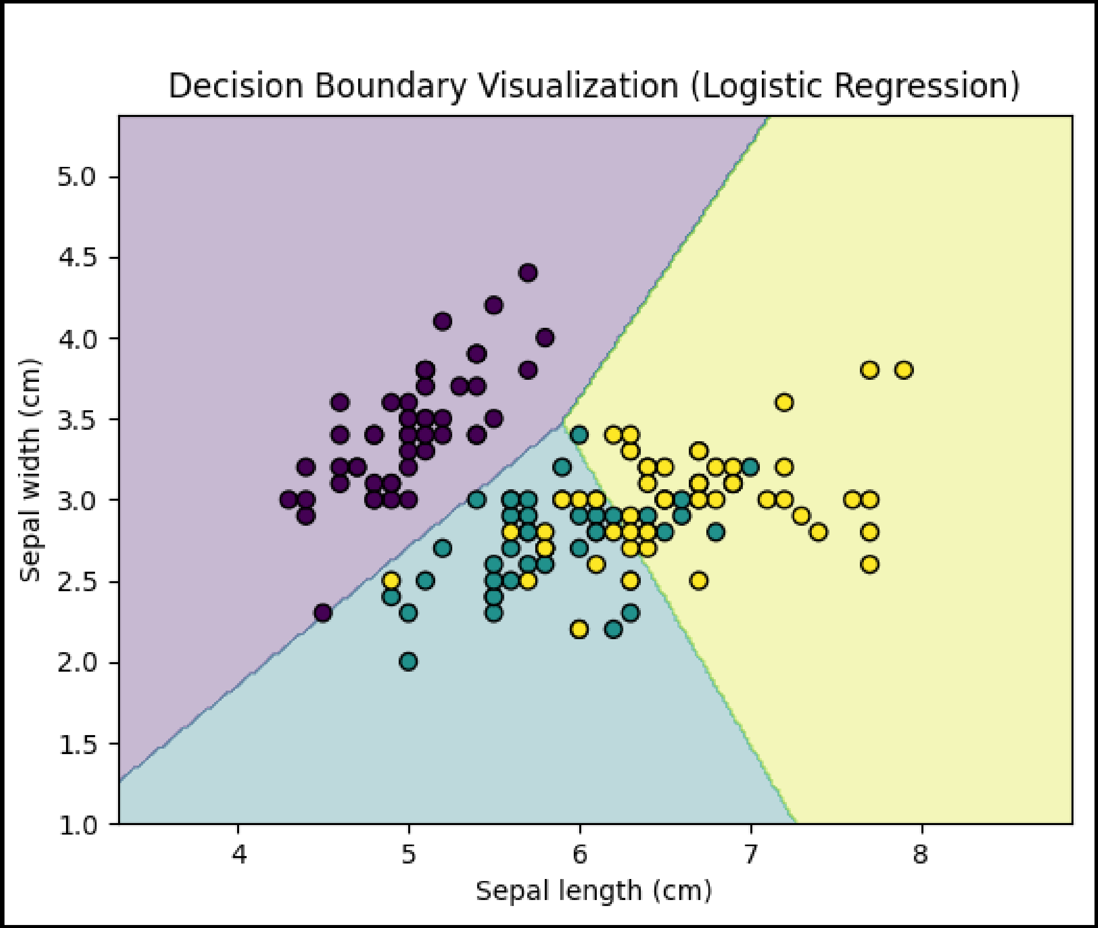
- The boundary between regions is a hypersurface where the model is uncertain.
  - Linear models (e.g., logistic regression, linear SVM) → boundaries are hyperplanes.
  - Nonlinear models (e.g., decision trees, kernel SVM, neural networks) → boundaries can be highly curved and complex.



Decision Boundary Example

```python
import matplotlib.pyplot as plt
import numpy as np
from sklearn.datasets import load_iris
from sklearn.linear_model import LogisticRegression

# Load dataset
iris = load_iris()
X = iris.data[:, :2]   # Use only first two features (sepal length, sepal width)
y = iris.target

# Train classifier
clf = LogisticRegression(max_iter=200)
clf.fit(X, y)

# Create mesh grid
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.02),
                     np.arange(y_min, y_max, 0.02))

# Predict class for each grid point
Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)

# Plot decision boundary
plt.contourf(xx, yy, Z, alpha=0.3, cmap=plt.cm.viridis)
plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.viridis, edgecolor="k", s=40)

plt.xlabel("Sepal length (cm)")
plt.ylabel("Sepal width (cm)")
plt.title("Decision Boundary Visualization (Logistic Regression)")
plt.show()
```

Execution Result: Decision Boundary



- Colored regions → areas where the model predicts a specific class.
- Scatter points → actual data samples, colored by their true labels.
- Smooth or linear boundaries → indicate simple models (e.g., logistic regression).
- Irregular boundaries → indicate more complex models (e.g., decision trees).
- Misclassified points appear inside the wrong region, showing model weaknesses.

Thank you!