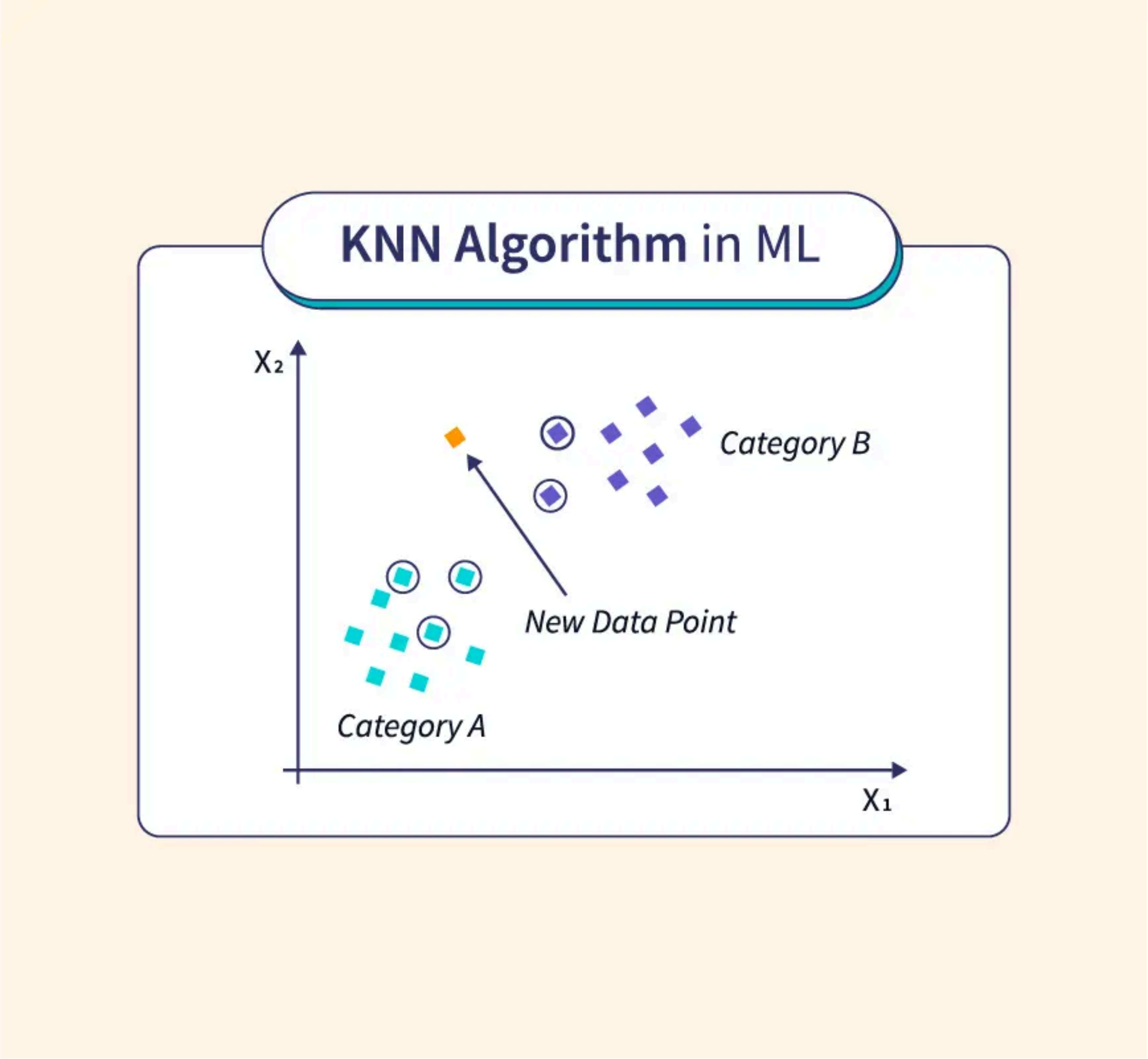
Modules 2: K-Nearest Neighbors (KNN)

🎯 Learning Objectives

- Understand how KNN works (Classification & Regression)
- Know how to measure distance and normalize data (Feature Scaling)
- Know how to choose appropriate K using Elbow Method
- Understand Decision Boundary
- Practice KNN in Python

What is KNN?



K-Nearest Neighbors is a **Supervised Learning** algorithm.

Principle: A new sample will be assigned the label of the majority of the nearest samples in the feature space.

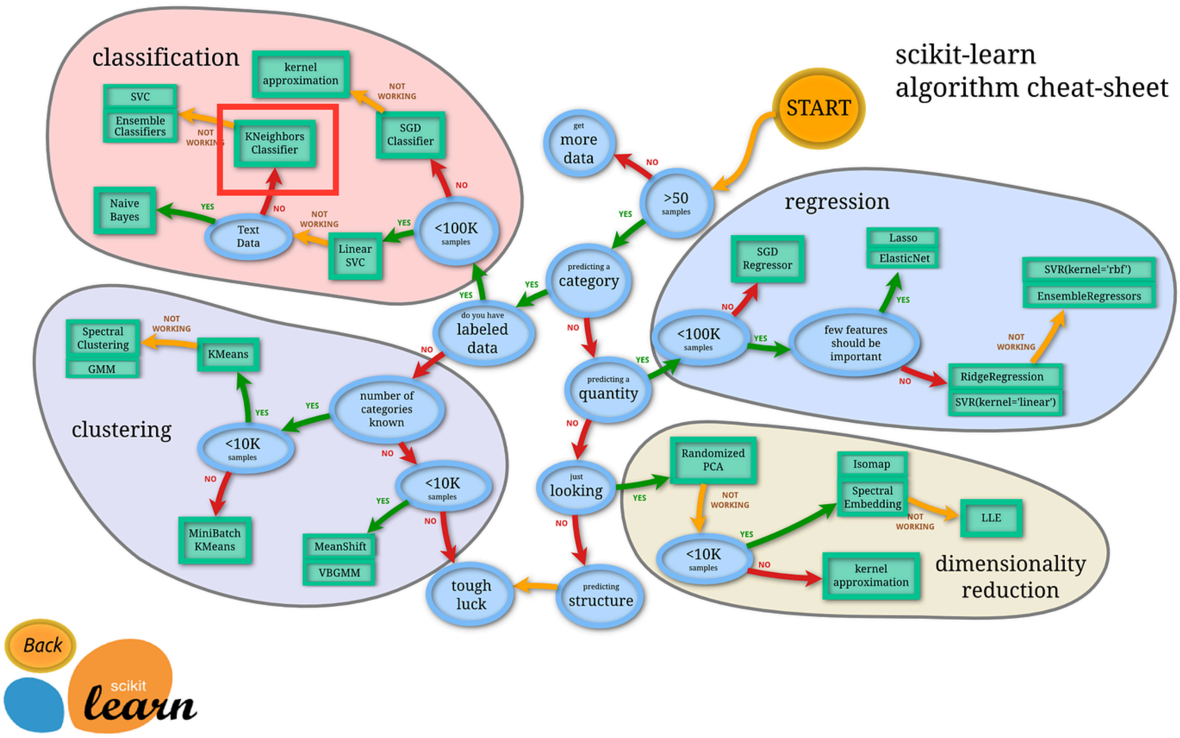
Characteristics

- **Non-parametric:** does not assume data distribution
- **Lazy learner:** does not learn in advance, only stores data → computes during prediction
- Also known as *instance-based / memory-based learning*

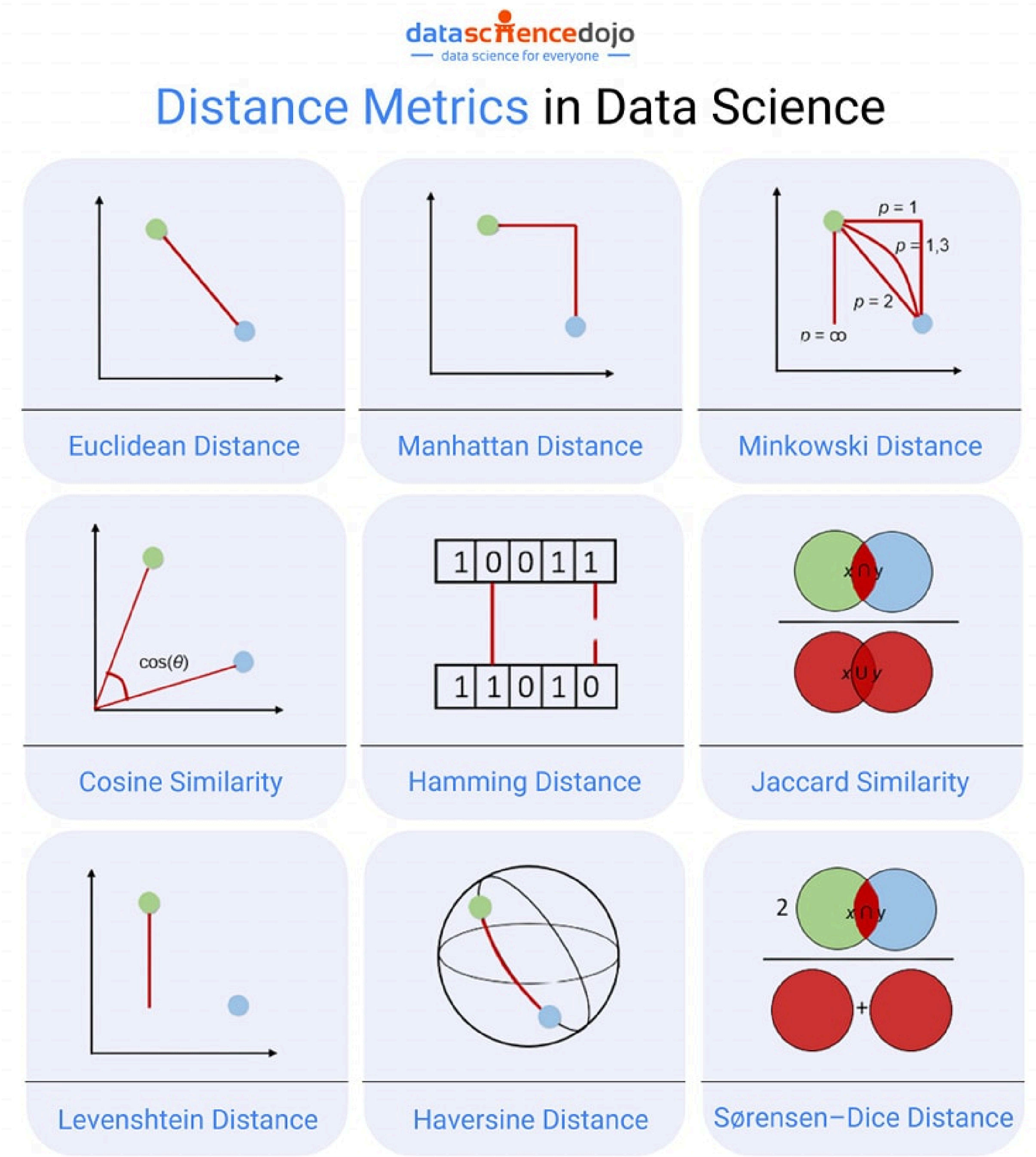
> 💡 **Conceptual Example**
> A new image looks like both a **cat** and a **dog** → KNN compares features with labeled cat & dog images, calculates **distance**, sees which is closest → classifies by majority vote.

How It Works

- **Choose K** (number of nearest neighbors)
- **Calculate distance** between new point and all training points
- **Sort** by distance in ascending order
- **Select K nearest points**
- **Classification/Prediction:**
  - **Classification:** choose the majority class
  - **Regression:** average the values



Distance Metrics Formulas



▼ Euclidean Distance

$$d(x, y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

Parameters:

- x, y : two data points

- x_i, y_i : feature values at position i
- n : number of features
- $d(x, y)$: linear distance

Example:

$x = (2, 3)$, $y = (5, 7)$

$$d = \sqrt{(2-5)^2 + (3-7)^2} = \sqrt{25} = 5$$

Use case: measures proximity in continuous space (L2 norm)

▼ Manhattan Distance

$$d(x, y) = \sum_{i=1}^{n} |x_i - y_i|$$

Parameters:

- x_i, y_i : feature values at position i
- n : number of dimensions

Example:

$x = (2, 3)$, $y = (5, 7)$

$$d = |2-5| + |3-7| = 3 + 4 = 7$$

Use case: suitable for data with outliers; measures along "city block" distance (L1)

▼ Minkowski Distance (generalized)

$$d(x, y) = \left(\sum_{i=1}^{n} |x_i - y_i|^p \right)^{1/p}$$

Parameters:

- p : order of norm (1=L1, 2=L2,...)
- x_i, y_i : feature values

Example:

$x = (2, 3)$, $y = (5, 7)$, $p = 3$

$$d = (|2-5|^3 + |3-7|^3)^{1/3} = (27+64)^{1/3} \approx 4.32$$

Use case: generalized formula, adjusts sensitivity to distant points using p

▼ Hamming Distance

$$d(x, y) = \frac{\text{Number of different positions}}{n}$$

Parameters:

- x, y : two vectors/strings of equal length
- n : number of elements

Example:

$x = [1, 0, 1, 1]$, $y = [1, 1, 1, 0]$

Different at 2/4 positions → $d = \frac{2}{4} = 0.5$

Use case: used for discrete or binary data

Feature Scaling (Normalization)

⚠️ **Note:** KNN is sensitive to scale → **must normalize data.**

▼ Z-score Standardization

$$z = \frac{x - \mu}{\sigma}$$

Parameters:

- x : original value
- μ : mean
- σ : standard deviation

Example:

$x = 180$, $\mu = 170$, $\sigma = 10$

$$z = \frac{180 - 170}{10} = 1$$

Use case: transforms data to mean 0, standard deviation 1

▼ Min–Max Normalization

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

Parameters:

- x : original value
- x_{min}, x_{max} : min, max of feature
- x' : value in [0,1]

Example:

$x = 70$, $x_{min} = 50$, $x_{max} = 100$

$$x' = \frac{70 - 50}{100 - 50} = 0.4$$

Use case: preserves ratio between values, bounds data to [0,1]

Decision Boundary

📌 **Decision Boundary in KNN**

- KNN creates **non-linear classification boundaries**
- Boundary depends on K and data distribution:
- **Small K** → curved boundary, sensitive to noise
- **Large K** → smooth boundary, more stable but may underfit

---

Elbow Method – Choosing Optimal K

$$\text{Error Rate}(K) = 1 - \text{Accuracy}(K)$$

Parameters:

- K : number of neighbors to test
- $\text{Accuracy}(K)$: accuracy with K

Example:

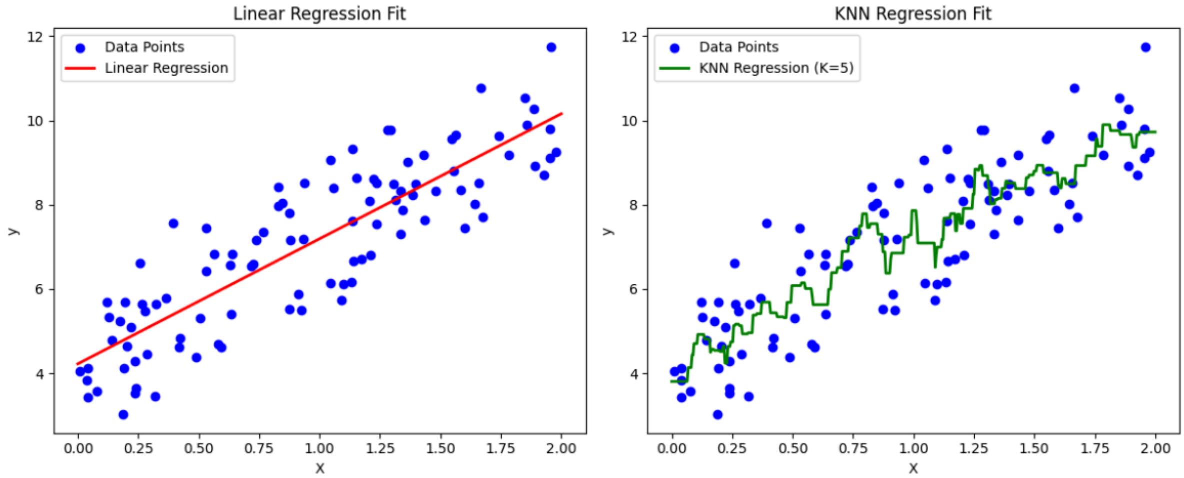
If $K = 1 \Rightarrow 85\%$, $K = 3 \Rightarrow 90\%$, $K = 15 \Rightarrow 89\%$

→ "Elbow" around **K=3–5** (accuracy increase slows down)

Use case: choose K at the elbow point where error reduction slows

---

Regression with KNN



KNN can also be used for **regression** (instead of choosing a class, we take the average of target values).

Regression Prediction Formula

$$\hat{y} = \frac{1}{K} \sum_{i=1}^{K} y_i$$

Parameters:

- K : number of nearest neighbors
- y_i : target value of K nearest neighbors
- $\hat{y}$: predicted value

Example:

5 nearest neighbors have values [4,5,6,5,4]

$$\hat{y} = (4 + 5 + 6 + 5 + 4)/5 = 4.8$$

Evaluation Formula (Mean Squared Error)

$$MSE = \frac{1}{m} \sum_{i=1}^{m} (y_i - \hat{y}_i)^2$$

Parameters:

- y_i : actual value
- $\hat{y}_i$: predicted value
- m : number of samples

Example:

Actual [3,3], predicted [2.8,3.2]:

$$MSE = \frac{(3 - 2.8)^2 + (3 - 3.2)^2}{2} = 0.04$$

Parameters in scikit-learn

```
KNeighborsClassifier(
    n_neighbors=5,
    weights='distance',
    metric='minkowski',
    p=2,
    algorithm='auto',
    n_jobs=-1
)
```

Parameter	Description & Example
`n_neighbors`	Number of neighbors (K). K=3 → consider 3 nearest points
`weights`	`'uniform'` : all points equal weight `'distance'` : closer points have more influence
`metric`	Distance calculation method (`'euclidean'` , `'manhattan'` , `'minkowski'` , `'hamming'`)
`p`	Minkowski order: 1=L1, 2=L2
`algorithm`	Neighbor search method: `'auto'` , `'kd_tree'` , `'ball_tree'` , `'brute'`
`n_jobs`	Number of CPU cores for parallel processing (-1 uses all)

Practical Examples

▼ Classification – Iris Dataset

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

# Load data
X, y = load_iris(return_X_y=True)

# Split data
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42
)

# Feature scaling
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Train model
knn = KNeighborsClassifier(
    n_neighbors=5,
    metric='euclidean',
```

```
    weights='distance'
)
knn.fit(X_train, y_train)

# Predict and evaluate
y_pred = knn.predict(X_test)
print("Accuracy:", accuracy_score(y_test, y_pred))
```

▼ Regression – Boston Housing

```
from sklearn.datasets import load_boston
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import mean_squared_error

# Load data
X, y = load_boston(return_X_y=True)

# Split data
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# Feature scaling
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Train model
knn_reg = KNeighborsRegressor(
    n_neighbors=5,
    weights='distance',
    metric='minkowski',
    p=2
)
knn_reg.fit(X_train, y_train)
```

```
# Predict and evaluate
y_pred = knn_reg.predict(X_test)
print("MSE:", mean_squared_error(y_test, y_pred))
```

Advantages & Disadvantages

✅ **Advantages**

- Simple, easy to understand, no training required
- Effective with non-linear data
- Easy to extend to regression
- No assumptions about data distribution

⚠️ **Disadvantages**

- Slow computation with large datasets
- Sensitive to feature scale & noise
- Need to choose K appropriately
- Performance degrades with high dimensions (curse of dimensionality)

Summary

Component	Content
Algorithm Type	Supervised, Non-parametric, Lazy learner
Applications	Classification, Regression
Distance Metrics	Euclidean, Manhattan, Minkowski, Hamming
Need Normalization?	Yes (very important)
Choose K	Elbow Method
Default Metric	Minkowski (p=2 → Euclidean)

Component	Content
Regression Evaluation	MSE
Library	`sklearn.neighbors`

Formula Notation Table

Symbol	Meaning
x_i, y_i	Feature value at position i of two points
n	Number of dimensions (feature dimension)
p	Minkowski order
μ, σ	Mean & standard deviation
x_{min}, x_{max}	Min, max value of feature
K	Number of neighbors
m	Number of samples
MSE	Mean Squared Error
$d(x, y)$	Distance between two points
$\hat{y}$	Predicted value