k-Nearest Neighbors

By Trần Minh Dương - Learning Support

Overview

Imagine you just moved to a new city and are looking for a good restaurant to try. You ask a few locals where they usually eat. If most of them recommend the same place, you're likely to trust their judgment and go there.

This is the core idea behind k-Nearest Neighbors (k-NN)—it makes predictions based on the **votes** of its closest "neighbors."

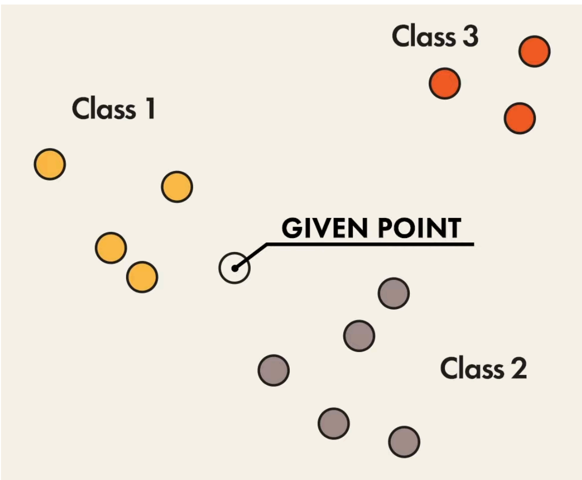
Watch this short video 👉🏻 K Nearest Neighbors | Intuitive explained

How k-NN Works:

1. **Store all data points**: k-NN memorizes all training data without creating a fixed mathematical model (parametric models)
2. **Find the k closest points**: Given a new input, k-NN calculates the distance between this point and all existing data points.
3. **Vote for classification**: In classification, the majority class among the k nearest neighbors determines the new data point's label.
4. **Average for regression**: In regression, k-NN predicts the value by averaging the outputs of the k nearest neighbors.

Algorithm Steps

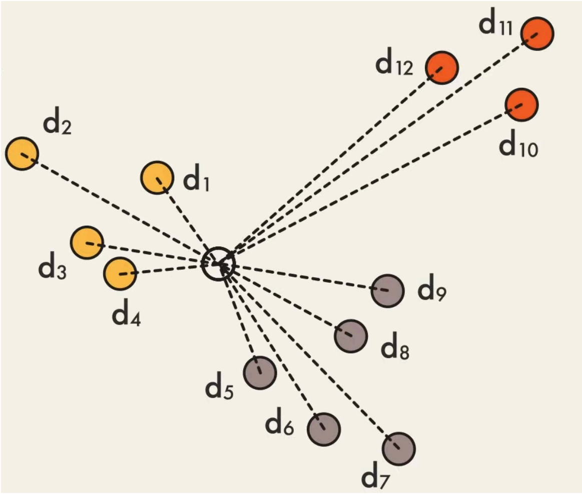
1. Initialize



Load the dataset, including their labels (classes)

Enter data for the new input (query instance)

2. Calculate the distance



Compute the distance between a query instance and all training examples using a chosen metric. Common metrics include:

- **Euclidean Distance**:

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

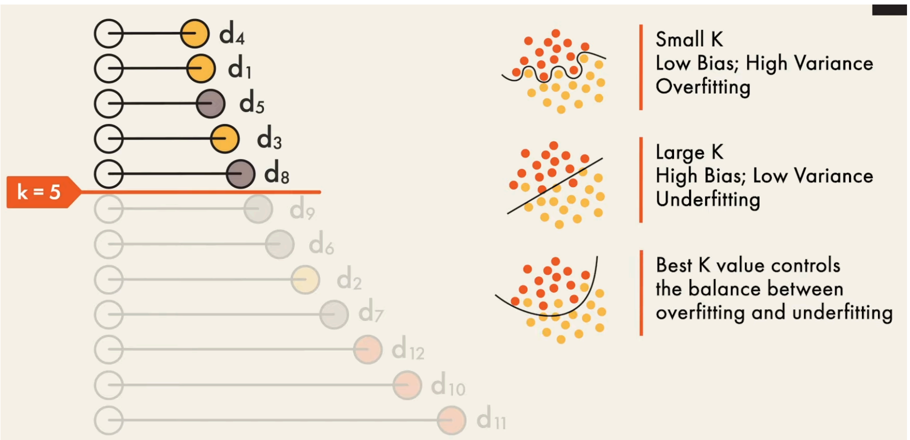
- **Manhattan Distance**:

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

- **Minkowski Distance** (generalized form):

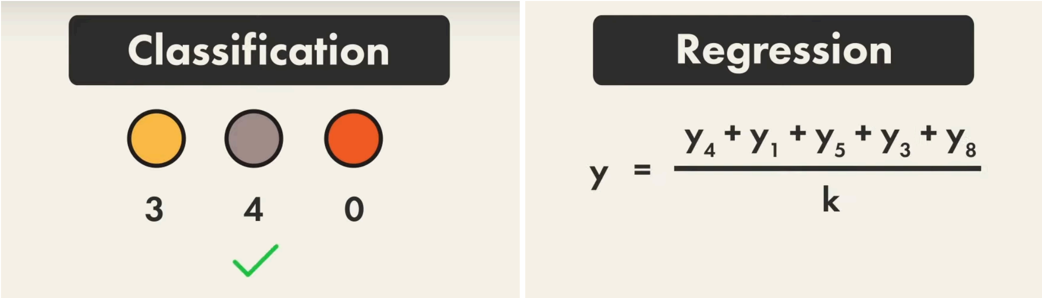
$$d(\mathbf{x}, \mathbf{y}) = \left(\sum_{i=1}^{n} |x_i - y_i|^p \right)^{1/p}$$

3. Sort distance and select k neighbors



Sort the distances in ascending order and identify the `k` training examples closest to the query instance.

4. Predict



- **Classification**: Majority vote among neighbors.
- **Regression**: Average of neighbors' target values.

Pros and Cons

Advantages	Disadvantages
No training phase	Expensive for large data
Simple to implement	Sensitive to noise
Handles multi-class data	Requires feature scaling

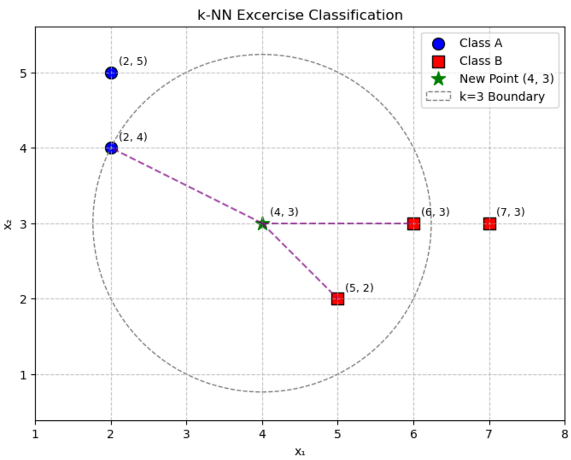
Exercise

Task: Predict the class label for a new data point using k-NN with `k=3` and Euclidean distance.

Dataset:

Feature 1 (x_1)	Feature 2 (x_2)	Class (Labels)
2	4	A
2	5	A
5	2	B
6	3	B
7	3	B
4	3	???

Solution



Step 1: Compute Distances

Compute the distance fron (4,3) to other points using **Euclidean Distance**:

Feature 1 (x_1)	Feature 2 (x_2)	Distance
2	4	$\sqrt{(4-2)^2+(3-4)^2}=\sqrt{4+1}=2.24$
2	5	$\sqrt{(4-2)^2+(3-5)^2}=\sqrt{4+4}=2.82$
5	2	$\sqrt{(4-5)^2+(3-2)^2}=\sqrt{1+1}=1.41$
6	3	$\sqrt{(4-6)^2+(3-3)^2}=\sqrt{4+0}=2.00$
7	3	$\sqrt{(4-7)^2+(3-3)^2}=\sqrt{9+0}=3.00$

Step 2: Identify Nearest Neighbors

Sorting the distances: $1.41 \rightarrow 2.00 \rightarrow 2.24 \rightarrow 2.82 \rightarrow 3.00$

Which correpsonds to points: $(5,2) \rightarrow (6,3) \rightarrow (2,4) \rightarrow (2,5) \rightarrow (7,3)$

Hence, k = 3 nearest samples are: **(5,2), (6,3), (2,4)**.

Step 3: Majority Vote

- (5,2) - Class B
- (6,3) - Class B
- (2,4) - Class A

$\Rightarrow$ Majority class = **B**

Code Example

In [1]:

```
import numpy as np
# Step 1: Initialize
points = [
    [2,4,'A'],
    [2,5,'A'],
    [6,3,'B'],
    [7,3,'B'],
    [5,2,'B']
]
newpoint = (4,3)
k = 3
```

```python
# Step 2: Calculate distance
for p in points:
    x1 = p[0]
    x2 = p[1]
    p.append(np.sqrt((newpoint[0]-x1)**2 + (newpoint[1]-x2)**2))

for p in points:
    print(f"Point:({p[0]},{p[1]}), Class:{p[2]}, Distance:{p[3]:.2f}")
```

```
Point:(2,4), Class:A, Distance:2.24
Point:(2,5), Class:A, Distance:2.83
Point:(6,3), Class:B, Distance:2.00
Point:(7,3), Class:B, Distance:3.00
Point:(5,2), Class:B, Distance:1.41
```

In [2]:
```python
# Step 3: Sort and select neighbors
points_sorted = sorted(points,key=lambda p: p[3]) #sort by distance
neighbors = points_sorted[:k]

for p in neighbors:
    print(f"Point:({p[0]},{p[1]}), Class:{p[2]}, Distance:{p[3]:.2f}")
```

```
Point:(5,2), Class:B, Distance:1.41
Point:(6,3), Class:B, Distance:2.00
Point:(2,4), Class:A, Distance:2.24
```

In [3]:
```python
# Step 4: Predict
classes = [n[2] for n in neighbors] #Extract classes
votes = {x:classes.count(x) for x in classes}

print("Votes:",votes)

majority_vote = max(votes,key=votes.get)

print(f"The point {newpoint[0],newpoint[1]} belongs to class:",majority_vote)
```

```
Votes: {'B': 2, 'A': 1}
The point (4, 3) belongs to class: B
```

Tip: Choose an odd value for k to avoid tie votes

This document was created in Jupyter Notebook by Trần Minh Dương (tmd).

If you have any questions or notice any errors, feel free to reach out via Discord at @tmdhoctiengphap or @ICT-Supporters on the USTH Learning Support server.

Check out my GitHub repository for more projects: GalaxyAnnihilator/MachineLearning .