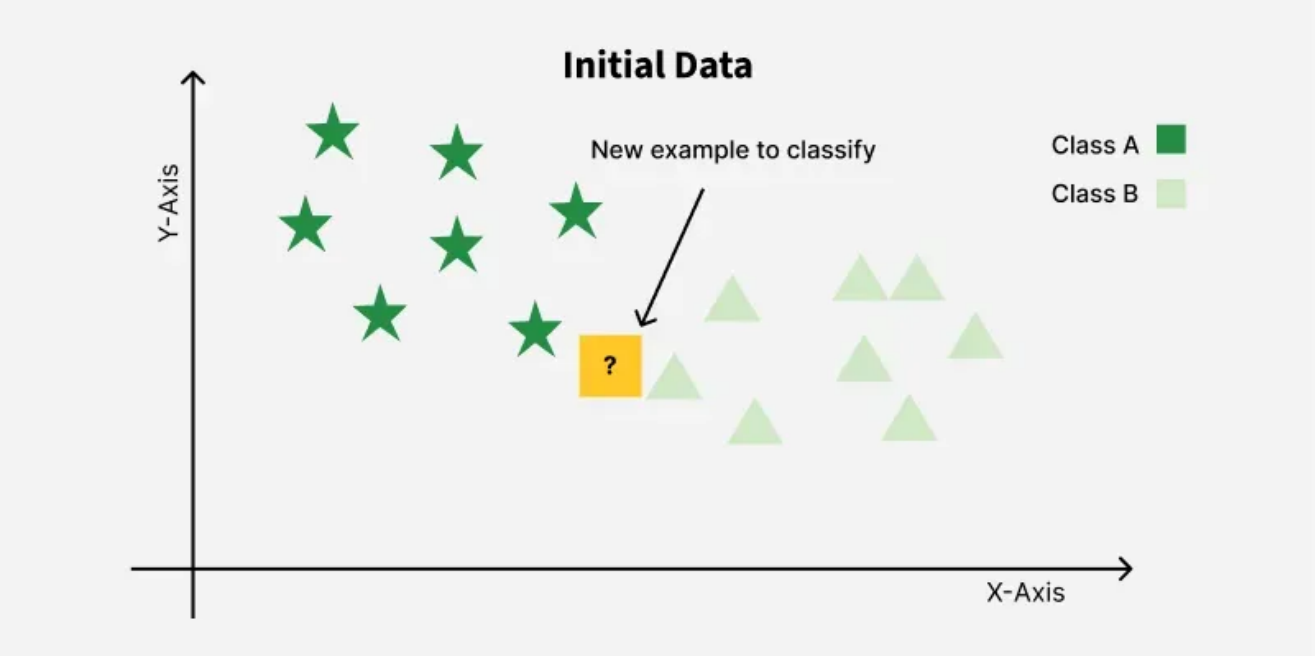
🧠 Understanding K-Nearest Neighbors (KNN) in 1D

Overview



The **K-Nearest Neighbors (KNN)** algorithm is a simple yet powerful **non-parametric model** used for both **classification** and **regression** tasks.
It makes predictions based on the **similarity (distance)** between data points — the idea is that **similar inputs have similar outputs**.

This document explains **how KNN behaves in one-dimensional (1D) cases**, such as when:

- You predict **Salary** based on **Age** (Regression), or
- You predict **Age Group** based on **Salary** (Classification).

🧠 Scenario: One Characteristic (1D Feature Space) - EstimatedSalary

Suppose your dataset now has **only one feature** — for example:

Age	EstimatedSalary
22	20,000
25	25,000
30	35,000
35	45,000
40	55,000
...	...

and you want to **predict salary** from **age** using **KNN Regression** (not classification).

⚙️ What Changes in the Algorithm

☑️ 1️⃣ Input and Output

- **Input (X)** → only one feature (`Age`), so it's **1-dimensional**.
  [
  X = [22, 25, 30, 35, 40, ...]
  ]
- **Output (y)** → a continuous value (`EstimatedSalary`).

So this becomes a **regression** problem, not classification.

☑️ 2️⃣ Distance Calculation (1D)

Since there's only one feature, the **distance** between two points is simply:

[
d(p, q) = |Age_p - Age_q|
]

No square roots or multidimensional space — it's just a **line**.

☑️ 3️⃣ Finding the k Nearest Neighbors

For a new input, say **Age = 28**, the KNN algorithm:

1. Finds the **k closest ages** in the training data.
   Example (if k = 3): ages 25, 30, 35.
2. Retrieves their corresponding salaries.
   Example: 25k, 35k, 45k.

☑️ 4️⃣ Prediction Rule (Regression)

Then it computes the **average** of the target values of those neighbors:

[
\hat{y} = \frac{1}{k} \sum_{i=1}^{k} y_i
]

In our example:

[
\hat{y} = \frac{25000 + 35000 + 45000}{3} = 35,000
]

☑ So the predicted salary for Age = 28 is **35,000**.

☑ 5 Visualization (1D)

When you plot this:

- The **x-axis** represents `Age`.
- The **y-axis** represents `Salary`.
- The KNN regression prediction forms a **piecewise curve** (non-linear) following the pattern of your data.

You can visualize:

- The original data points as dots.
- The KNN-predicted curve (average of local neighbors).

📊 Summary — What Happens in 1D

Aspect	Explanation
Type	KNN Regression
Input dimension	1 (Age)
Output	Continuous value (Salary)
Distance metric	Absolute difference in ages
Decision boundary	Not applicable (no classification)
Prediction	Mean of nearest neighbors' salaries

💬 Intuitive Analogy

Think of it like asking:

> "What's the average salary of people around my age?"

KNN looks at people with similar ages (the **nearest neighbors**) and averages their salaries to guess yours.

🧩 If You Had Used Classification Instead

If the target were still **binary** (like *Purchased = Yes/No*),
and you only gave **Age**, then:

- KNN would find people with **similar ages**,
- Count how many purchased vs. didn't purchase,
- Predict the **majority class** among those neighbors.

So:

- 1 feature → model learns a **threshold** along a 1D line
  (e.g., users under 30 don't buy, over 30 do).

Would you like me to create a **Python example** (with graph) that shows how KNN regression works with a single feature like `Age → Salary`? It would illustrate how the curve adapts to the data.

Excellent 👏 — now we're flipping the problem:

Instead of predicting **Salary** (a continuous value), you want to predict **Age group or class** (a categorical variable) based on **Salary** — that's a **KNN classification problem** with **one feature**.

Let's unpack this carefully 👇

🧠 Scenario: One Input Feature — Age

Suppose your dataset looks like this:

EstimatedSalary	AgeGroup
20,000	0 (Young)
25,000	0
30,000	0
50,000	1 (Adult)
55,000	1
70,000	2 (Senior)
80,000	2

You train a **KNN Classifier** to learn the relationship between `Salary` and `AgeGroup`.

⚙️ Step-by-Step — How KNN Works Here

1️⃣ Feature and Labels

- **X (input)** → `[20,000], [25,000], [30,000], [50,000], [55,000], [70,000], [80,000]`
- **y (target)** → `[0, 0, 0, 1, 1, 2, 2]`

Each training example is a single number — the **salary**.

So your feature space is **1D** (a straight line).

2️⃣ Distance Calculation

KNN calculates distance in **1D**, which is just:

[
d(p, q) = |Salary_p - Salary_q|
]

So if you want to classify a new example,
say `Salary = 60,000`,
the model finds its **k nearest salaries** in the dataset.

Example with **k = 3**:

- Neighbors: 55,000 (class 1), 70,000 (class 2), 50,000 (class 1)
- Their classes: `[1, 2, 1]`

---

3 Majority Voting (Classification Rule)

The algorithm counts the most frequent class among the k nearest neighbors.

[
\text{Predicted class} = \text{mode of neighbors' classes}
]

In this case:

Neighbor Salary	Class
55,000	1
70,000	2
50,000	1

Class **1** appears twice → **Predicted class = 1 (Adult)**.

---

4 Visualization (1D Classification)

You can imagine the **x-axis = Salary**
and the **y-axis = Predicted Age Group**.

Since we only have one dimension, the model forms **segments** (intervals) along the salary line — each segment corresponds to a different predicted class:

```
|------|----------|---------------|---------------|
Young  Adult                  Senior
20000  40000      60000         80000
```

At each new salary value, KNN checks which neighboring salaries dominate and assigns that class.

Summary

Aspect	Explanation
Type	KNN Classification
Input dimension	1 (Salary)
Target	Discrete class (Age group)
Distance metric	Absolute salary difference
Decision boundary	Salary thresholds where the predicted class changes
Prediction	Majority vote among nearest salaries

Intuitive Analogy

Think of it like:

> "If people around your salary are mostly adults, the model assumes you are too."

The model doesn't build an explicit formula — it simply looks at the **local neighborhood** in the feature space and classifies based on the **majority**.

Note on Visualization

If you plot it:

- The **x-axis**: Salary
- The **y-axis**: Predicted class (0 = Young, 1 = Adult, 2 = Senior)

The graph would show **step-like regions**, where each region corresponds to the dominant class — that's the **decision boundary** in 1D.