📘 K-Nearest Neighbors (KNN) – Full Notes

1️⃣ Definition

K-Nearest Neighbors (KNN) is a **supervised machine learning algorithm** used for **classification and regression**, based on the idea that **similar data points exist close to each other**.

> **KNN predicts the label of a new sample by looking at the 'K' closest labeled points in training data.**

2️⃣ Real-Life Intuition

Imagine you moved to a new city and want to predict your personality type.
You look at **your 5 nearest neighbors (K=5)**—if **3 are extroverts** and **2 are introverts**, you say **you are extrovert**.

Example:

```
Nearest neighbors labels = [1, 1, 1, 0, 0]
Prediction = majority vote = 1
```

3️⃣ Why do we need KNN?

Reason	Explanation
Easy & intuitive	Based on similarity concept
Works without assumptions	Non-parametric → no underlying distribution
Good baseline model	Simple, interpretable
Real world is similarity-based	Example: recommendation systems, medical diagnosis

---

4️⃣ When did KNN come / why was it introduced?

KNN originated in **1951 (Fix & Hodges)** as a statistical method for **pattern classification**.
It was introduced because many datasets do **not fit linear equations** like regression or SVM — **we need similarity-based models**.

5️⃣ How KNN works (Step by Step)

Given: New data point x

1. **Choose K** (number of neighbors)
2. **Calculate distance** between x and all points
3. **Sort distances** in ascending order
4. **Pick first K nearest points**
5. **Majority vote** → classification
   or **mean of neighbors** → regression

6️⃣ Mathematical Explanation

Distance Formula Used

Most common distance: **Euclidean Distance**

d(x,y)=∑i=1n(xi−yi)2d(x,y)=\sqrt{\sum_{i=1}^{n}(x_i − y_i)^2}d(x,y)=i=1∑n(xi−yi)2

Other distances:

Metric	Formula	Use Case
Manhattan	(\sum	x_i - y_i

Minkowski	((\sum	x_i-y_i
Cosine similarity	(\frac{A·B}{	

---

7 Example Calculation

Assume we want to classify a new point `X = (3,4)` with 3 nearest neighbors (K=3)

Point	Class	Distance
(2,2)	0	√((3-2)²+(4-2)²) = √5
(4,4)	1	√((3-4)²+(4-4)²) = 1
(3,7)	1	3

Nearest 3 classes = `[1, 1, 0]`
Prediction = **1**

---

8 Hyperparameters

Parameter	Meaning
`K`	number of neighbors
`distance metric`	euclidean, manhattan, cosine
`weights`	uniform / distance weighted
`algorithm`	brute, kd_tree, ball_tree

How to Choose the Best K in KNN

A commonly used **rule of thumb** is:

K=NK = \sqrt{N}K=N

Where:

- **K** = number of nearest neighbors
- **N** = total number of training samples

Simple Example

If you have:

```
N = 100 samples
```

Then:

K=100=10K = \sqrt{100} = 10K=100=10

So you should start with **K = 10** and then test other values near it (like 9, 11, 12).

Why this rule exists

- If **K is too small**, model becomes **noisy & overfits**
  → sensitive to outliers
- If **K is too large**, model becomes **too simple / underfits**
  → loses important patterns

Choosing K around √N gives a **balanced starting point**.

Then you test 5–10 values near it using **cross-validation**.

Final Notes

- √N is **only a starting point**, not exact best value always
- You still tune K using **accuracy scores & validation curve**

One-line exam answer

> We usually select K by starting around √N (square root of total samples), because too small K causes overfitting and too large K causes underfitting, and √N gives a balanced choice.

Errors:

K small	K large
Overfitting	Underfitting
Highly sensitive to noise	Too generalized

🔟 Advantages & Disadvantages

✔ Advantages

- Simple to understand & implement
- Non-parametric (no assumption)
- Works well with small, clean datasets
- Flexible for classification & regression

❌ Disadvantages

- Slow prediction for large data
- Sensitive to irrelevant features & scaling

- Memory expensive
- Bad with high dimensional data

---

11 Need for Feature Scaling

Because distance depends on value magnitude:

$X' = \frac{X - mean}{std}$

Example:
Height range: 150-200
Weight range: 40-80
Weight becomes less important → wrong results

So use: **StandardScaler or MinMaxScaler**

---

12 Applications

Field	Use
Healthcare	cancer detection
Finance	credit score risk
Recommendation systems	movie similarity
Handwritten digits	MNIST dataset

---

13 Python Code

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
```

```
X_test = scaler.transform(X_test)

knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)
```

1️⃣4️⃣ When not to use KNN

🚫 Very large datasets
🚫 High-dimensional datasets
🚫 Real-time prediction required

🧠 Summary in One Line

> **KNN predicts using similarity (distance) to nearest neighbors instead of learning a mathematical model.**

🔥 Different Algorithms Used in KNN

When KNN predicts, it must **search for the K nearest points**.
Doing this on large datasets can be slow, so sklearn gives different search algorithms:

1️⃣ Brute Force (Exhaustive Search)

Definition

Compares the distance from the query point to **every other point** in the dataset.

Time Complexity

O(n×d)O(n \times d)O(n×d)

`n` = number of samples
`d` = number of dimensions

When to Use

- Small datasets
- Low dimensions
- Very accurate results

Example

```
KNeighborsClassifier(algorithm='brute')
```

2️⃣ KD-Tree (K-Dimensional Tree)

Definition

A **binary tree structure** that divides data space into halves recursively.
Used for **fast nearest neighbor search**.

Time Complexity

Average:

O(logn)O(\log n)O(logn)

Worst case:

O(n)O(n)O(n)

When to Use

Requirements	Reason
Numeric data	KD tree works only for numeric values
Low dimensions (d < 30)	Performance decreases when dimensions increase

Example

```
KNeighborsClassifier(algorithm='kd_tree')
```

3️⃣ Ball Tree

Definition

Divides space into **hyperspheres (balls)** instead of splitting along axes (like KD-tree).
Better for **high-dimensional and complex data**.

Time Complexity

Better performance than KD-tree when dimensions are high.

When to Use

Condition	Why
High-dimensional data	handles spherical clusters better
Mixed type distributions	more stable

Example

```
KNeighborsClassifier(algorithm='ball_tree')
```

4️⃣ Auto (Default)

Definition

Automatically selects the best algorithm among:

```
kd_tree → ball_tree → brute
```

depending on data size & dimension.

Example

```
KNeighborsClassifier(algorithm='auto')
```

🏆 Which algorithm is best?

Data Type	Best Algorithm
Small dataset	brute
Low dimensions (d < 30)	kd_tree
High dimensions	ball_tree
Not sure	auto

🎯 Why do we need these algorithms?

Because KNN is **slow during prediction**

O(n) per predictionO(n) \text{ per prediction}O(n) per prediction

To improve **speed / runtime**, we need efficient searching structures (KD tree, Ball tree).

🔍 Summary Table

Algorithm	Works How	Best Use	Weakness
Brute	checks every point	small dataset	slow for large data
KD-tree	axis split tree	low-d high speed	bad when dims high
Ball-tree	spherical clustering	high-dimensional	complex
Auto	auto selection	general usage	none

🧠 KNN Parameters Recap

```
KNeighborsClassifier(
    n_neighbors=5,
    weights='uniform',
    metric='minkowski',
    algorithm='auto'
)
```

🎓 Interview Question

> **Q: Why is KNN called a lazy learner?**
> Because it does **not train a model** — it stores data and **waits until prediction time**, where it performs heavy computation.

🔥 Final Answer (Simple)

KNN is called a lazy learner because it does not build a model during training. Instead, it waits until prediction time and then performs all computations (like finding distances) to make a prediction.