K-Nearest Neighbour (KNN) – Complete Beginner-to-Advanced Revision Guide

---

1. Introduction

What is K-Nearest Neighbour?

K-Nearest Neighbour (KNN) is a **supervised, non-parametric, instance-based learning algorithm** used for **classification and regression**.

Instead of learning an explicit model, KNN **stores the training data** and makes predictions by looking at the **K closest data points** to a new input.

Key Intuition

> "Tell me who your neighbours are, and I'll tell you who you are."

- Similar points → similar labels
- Decision is made locally
- No training phase (lazy learning)

Real-World Applications

- Recommendation systems
- Handwritten digit recognition
- Image classification
- Medical diagnosis
- Anomaly detection

---

2. Mathematical Foundations

2.1 Feature Space Representation

Each data point is a vector:

$$ \mathbf{x} = (x_1, x_2, \dots, x_n) $$

Distance between points defines similarity.

2.2 Distance Metrics

Euclidean Distance (Most Common)

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

Manhattan Distance

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

Minkowski Distance

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

- $p = 2$ → Euclidean
- $p = 1$ → Manhattan

Cosine Distance (High-Dimensional Data)

$$ \text{cosine similarity} = \frac{x \cdot x'}{|x| |x'|} $$

$$ \text{cosine distance} = 1 - \text{cosine similarity} $$

---

3. KNN for Classification

Decision Rule

Given a query point x:

1. Compute distance to all training points
2. Select K nearest neighbours
3. Assign the most frequent class

$$ \hat{y} = \text{mode}{y_i : x_i \in N_K(x)} $$

Weighted KNN

Closer neighbours get more influence:

$$ w_i = \frac{1}{d(x, x_i)} $$

---

4. KNN for Regression

Prediction Rule

$$ \hat{y} = \frac{1}{K} \sum_{i \in N_K(x)} y_i $$

Weighted Regression

$$ \hat{y} = \frac{\sum w_i y_i}{\sum w_i} $$

---

5. Model "Training"

Why KNN Is Lazy Learning

- No parameter estimation
- No optimisation
- All computation happens at prediction time

Training cost: minimal
Prediction cost: expensive

---

6. Choosing the Value of K

Bias–Variance Tradeoff

- Small K → low bias, high variance
- Large K → high bias, low variance

Cross-Validation

Choose K that minimises validation error.

---

7. Step-by-Step Example

Example: 2D Classification

1. Plot data points
2. Choose K = 3
3. Compute distances
4. Take majority vote

---

8. Model Evaluation

Classification Metrics

- Accuracy
- Precision
- Recall
- F1-score
- ROC-AUC

Regression Metrics

- Mean Squared Error (MSE)

$$ \text{MSE} = \frac{1}{n} \sum (y - \hat{y})^2 $$

- Mean Absolute Error (MAE)

$$ \text{MAE} = \frac{1}{n} \sum |y - \hat{y}| $$

---

9. Interpretation

- Predictions depend on local neighbourhood
- No global model parameters
- Highly sensitive to feature scaling

---

10. Assumptions

- Similar points have similar labels
- Distance metric is meaningful
- Features are comparable

---

11. Common Pitfalls & Misconceptions

- ❌ Forgetting feature scaling
- ❌ Choosing K arbitrarily
- ❌ Using KNN on large datasets
- ❌ Curse of dimensionality
- ❌ Interpreting KNN as a model

---

12. Python Implementation

12.1 From Scratch (Classification)

```python
import numpy as np

class KNN:
    def __init__(self, k=3):
        self.k = k

    def fit(self, X, y):
        self.X_train = X
        self.y_train = y

    def predict(self, X):
        preds = []
        for x in X:
            distances = np.sqrt(np.sum((self.X_train - x) ** 2, axis=1))
            k_idx = np.argsort(distances)[:self.k]
            k_labels = self.y_train[k_idx]
            preds.append(np.bincount(k_labels).argmax())
        return np.array(preds)
```

12.2 Using scikit-learn

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

model = KNeighborsClassifier(n_neighbors=5, metric='euclidean')
model.fit(X_train, y_train)

y_pred = model.predict(X_test)
print(classification_report(y_test, y_pred))
```

12.3 Decision Boundary Visualization

```
import matplotlib.pyplot as plt

plt.scatter(X[:,0], X[:,1], c=y)
plt.title("KNN Decision Regions")
plt.show()
```

---

13. Computational Complexity

- Training: $O(1)$
- Prediction: $O(nd)$

Where: - n = number of samples - d = number of features

---

14. When NOT to Use KNN

- Large datasets
- High-dimensional spaces
- Real-time systems

---

15. Best Practices

- Always scale features

• Use odd K for binary classification
• Try distance weighting
• Use KD-tree / Ball-tree

---

16. Summary

KNN is: - Simple - Intuitive - Non-parametric - Powerful for local patterns

But: - Computationally expensive - Sensitive to noise and scale

---

End of Revision Guide