12/10/2025

Understanding the K-Nearest Neighbors:

(KNN Algorithm and the Effect of K on Accuracy)

Machine Learnig And Nueral Network

Student Name: Zohaib Afzal
STD ID : 24090596

University of Hertfordshire
Decemebr 10/12/2025

Table of Contents:

Table of Figures:

KNN Algorithm and the Effect of K on Accuracy

1. Introduction to KNN Algorithm

The **K-Nearest Neighbors (KNN)** algorithm is a machine-based learning algorithm that is mostly used for classification but can also be used for regression problems. It is also called as instance-based or lazy learning method because it does not make data distribution assumptions about the training data points rather it stores the whole dataset and performs computations only at the time of classification. It starts working by finding the value of "K" which is the best nearest data point to the given input data so that it can make predictions about the grouping of the individual data point (Ganga, et al., 2024)

2. Statistical Methods for Choosing k

2.1. Cross-Validation:

A reliable method to choose the best **k** is **k-fold cross-validation**. The dataset is split into *k* equal parts, and the model is trained on some folds and tested on the others. This process repeats until each fold has been used for testing. The **k** that gives the highest average accuracy across all folds is selected as the optimal value. (Ganga, et al., 2024).

2.2. Odd Values for k:

For classification tasks, using an odd value of k is often recommended. It prevent ties when counting which class appears most frequently among the nearest neighbors (Ganga, et al., 2024).

2.3. Visual Representations of KNN

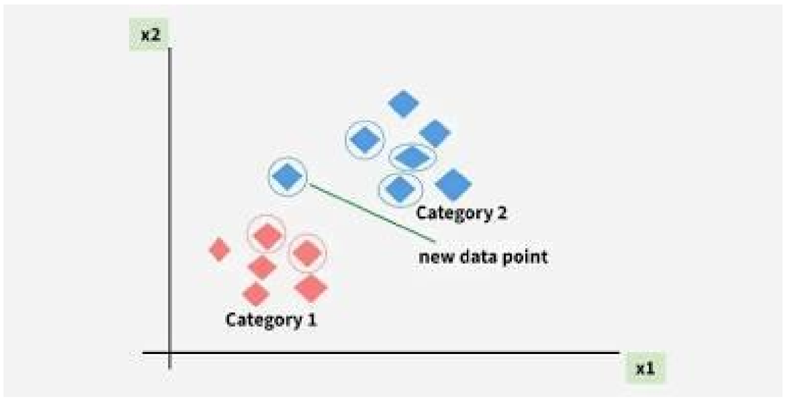


Figure 1: Visual Representation of KNN (K- Nearest Neighbors)

KNN makes predictions based on **distance** and **majority voting**, and it assigns the new point to the class that is most common among its closest neighbors.

3. KNN Pipeline:

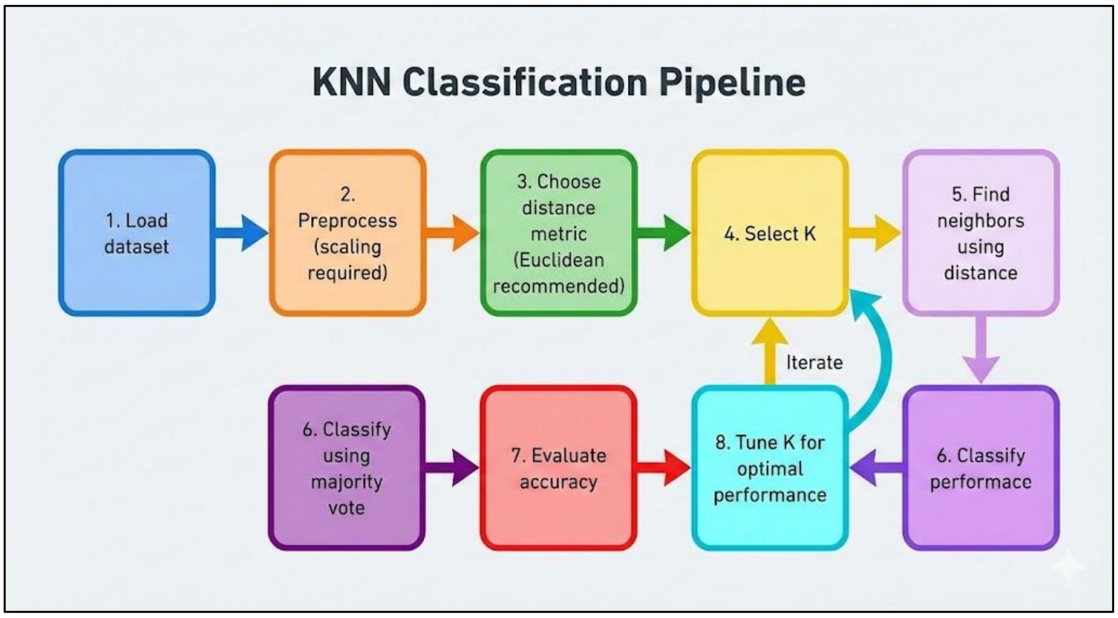

KNN Classification Pipeline
1. Load dataset
2. Preprocess (scaling required)
3. Choose distance metric (Euclidean recommended)
4. Select K
5. Find neighbors using distance
6. Classify performace
8. Tune K for optimal performance
Iterate
7. Evaluate accuracy
6. Classify using majority vote

Figure 2: KNN Classification Pipeline Workflow

3.1. Load Dataset

Import the dataset that we want to classify.

3.2. Preprocess the Data

Scale or normalize features.

It is important because KNN uses distance so large-scale features may dominate.

3.3. Choose Distance Metric

Examples:

i. **Euclidean distance (most common):**

Euclidean distance is also known as the straight-line distance between two points in a plane or space.

$$\text{distance}(x, X_i) = \sqrt{\sum_{j=1}^{d}(x_j - X_{i_j})^2}]$$

Figure 3: Euclidean Distance Formula

ii. **Manhattan distance:**

Manhattan distance, also known as L1 **distance** or **taxicab distance.** It measures the distance between two points by summing the absolute differences of their coordinates.

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

Figure 4: Manhattan Distance Formula

KNN depends on distance to measure similarity.

3.4. Select K (No of Neighbors)

Choose how many neighbors to consider when classifying a new point.

3.5. Find Neighbors

For each new data point:

- Calculate the distance to all training samples
- Pick the K closest points

3.6. Classify Using Majority Vote

Among the selected K neighbors:

- Count the class that appears the most
- Assign that class to the new point

3.7. Tune K for Better Accuracy

Try different values of K (e.g., 1–20).

Select the K that gives the best accuracy.

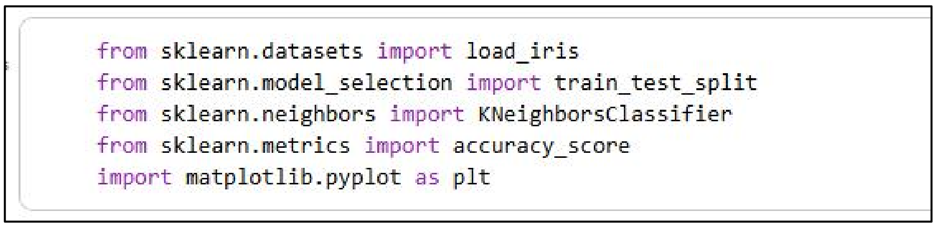
This step loops back to Step 4.

4. Implementing KNN from Scratch in Python:

4.1. Demonstration of code on a Dataset:

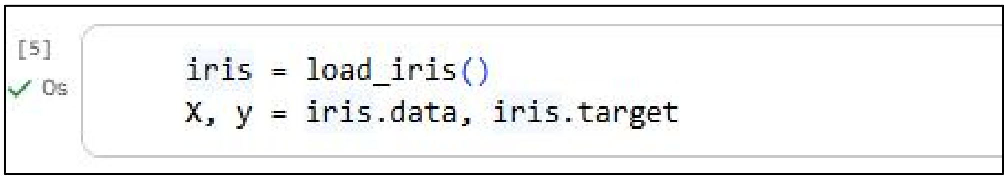
To demonstrate how value of *K* affects the performance, we consider the classic **Iris dataset** having 150 samples of iris flowers including 4 features and 3 classes. We can split it into training and test sets (e.g. 80% train, 20% test) and evaluate accuracy for K from 1 to 15.

Import Libraries:



```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
```

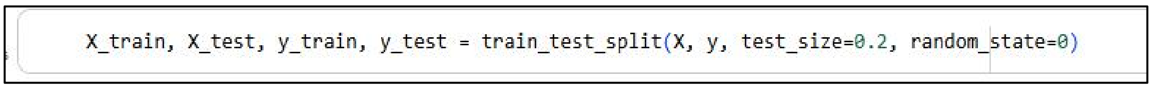
Load datasets:



```
iris = load_iris()
X, y = iris.data, iris.target
```

- X contains the input features.
- y contains the class labels.

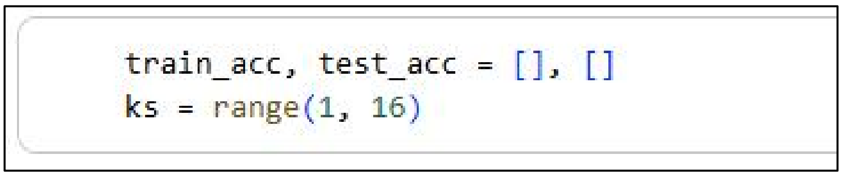
Training:



```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
```

- 80% data = Training
- 20% data = Testing
- random_state=0 ensures consistency.

Testing Values for K:



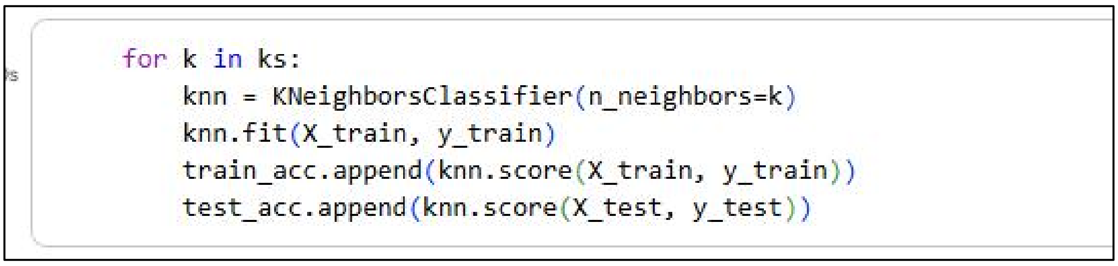
```
train_acc, test_acc = [], []
ks = range(1, 16)
```

- Values of K tested = 1 to 15
- Two lists store accuracies:

  A. train_acc → Training accuracy

  B. test_acc → Testing accuracy

Building Values and Training Model:

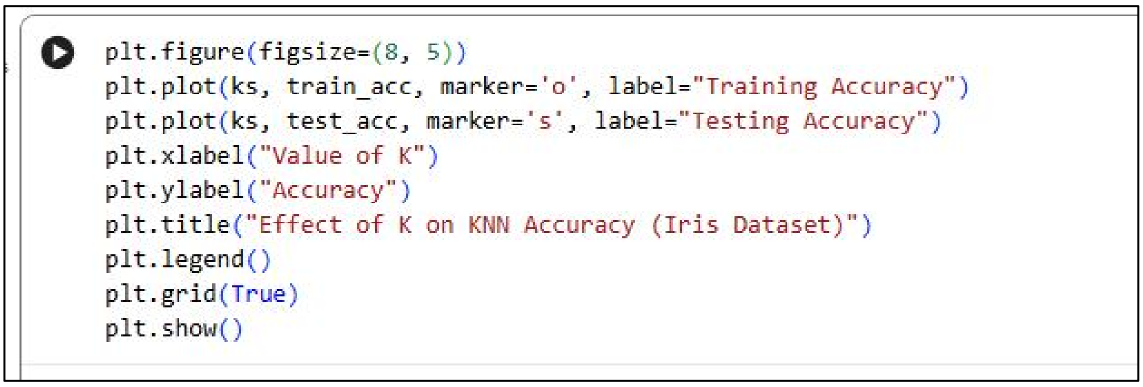


```
for k in ks:
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train, y_train)
    train_acc.append(knn.score(X_train, y_train))
    test_acc.append(knn.score(X_test, y_test))
```

For each K:

Build and Train the model. Measure accuracy on training set and testing set. Store all accuracy values for plotting.

Printing Output:



```
plt.figure(figsize=(8, 5))
plt.plot(ks, train_acc, marker='o', label="Training Accuracy")
plt.plot(ks, test_acc, marker='s', label="Testing Accuracy")
plt.xlabel("Value of K")
plt.ylabel("Accuracy")
plt.title("Effect of K on KNN Accuracy (Iris Dataset)")
plt.legend()
plt.grid(True)
plt.show()
```

This plots two curves:

- Training and Testing accuracy vs. K
- Shows how accuracy changes with the increase in k.

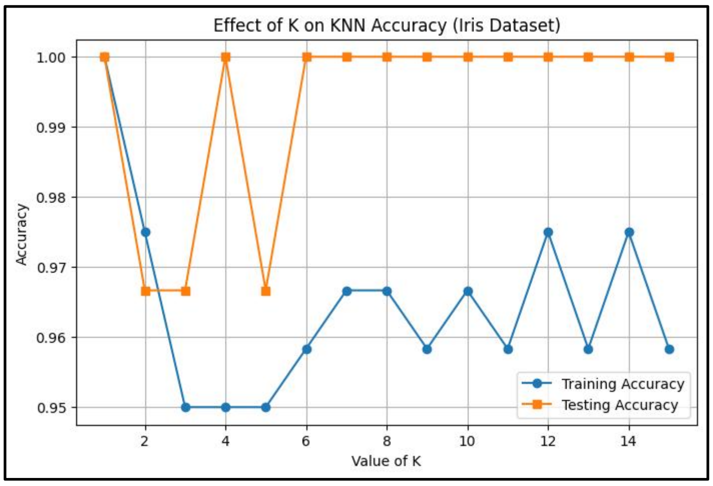


Figure 5: Output (Effect of K on KNN)

4.2. Key Observations:

- **K = 1 gives 100% training accuracy**
  - This is **overfitting**: too accurate on training data, but not perfect on test data.
- **Training accuracy decreases as K increases**
  - When K increases, decision boundaries become smoother.
  - This reduces overfitting.
- **Testing accuracy becomes stable for K between 6 and 15**
  - Testing accuracy remains around **97–100%**, which is very good.
  - Higher K makes the model more general and less noisy.
- **Best performance happens at K = 6 to 15**
  - Testing accuracy stays at **100%** for many values.
  - This means the model demonstrates good generalization performance.

5. Visualizing Decision Boundaries

The process shows how **the value of K in the K-Nearest Neighbors algorithm affects model performance** on the Iris dataset. Instead of visualizing decision boundaries, it focuses on how **accuracy changes** as K increases from 1 to 15. Both **training** and **testing accuracy** are plotted, it allows us to see the trade-off between overfitting and generalization.

5.1. Training Accuracy Region (Blue Line)

The blue curve represents how well the model performs on the data on which it was trained.

- At **K = 1**, training accuracy reaches **100%**, meaning the model memorizes every training point.
- As K increases, training accuracy drops gradually to around **95–97%**, because model becomes smoother and less sensitive to individual training instances.

This decreasing trend reflects that **larger K values reduce overfitting** by smoothing the decision boundaries.

5.2. Testing Accuracy Region (Orange Line)

The orange curve shows the performance on unseen test data.

- Unlike the training curve, testing accuracy stays **high and stable**, mostly between **97% and 100%**.
- For moderate K values (6 to 15), the testing accuracy becomes extremely stable and even reaches **100%** for many K values.

This indicates that **larger K values help the model generalize better**, preventing it from fitting noise.

6. How the Value of K Affects Model Behavior

6.1. The Role of K in Bias-Variance Tradeoff

The choice of **k** is critical in KNN, controlling **model complexity** and sensitivity to noise. **K=1** makes the model highly flexible: **low bias** but **high variance**, as it fits complex boundaries and is sensitive to outliers. A **large K** smooths predictions, ignoring local patterns, resulting in **high bias** and **low variance** (Eda Kavlakoglu, 2023), (Cornelissen, R., 2022).

6.2. Overfitting vs Underfitting:

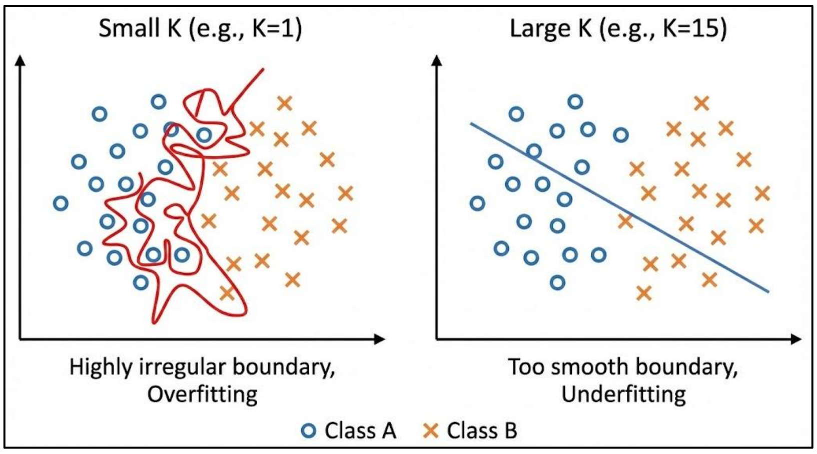

Small K (e.g., K=1)
Large K (e.g., K=15)
Highly irregular boundary, Overfitting
Too smooth boundary, Underfitting
Class A
Class B

Figure 6: Over-fitting Vs Under-fitting Model

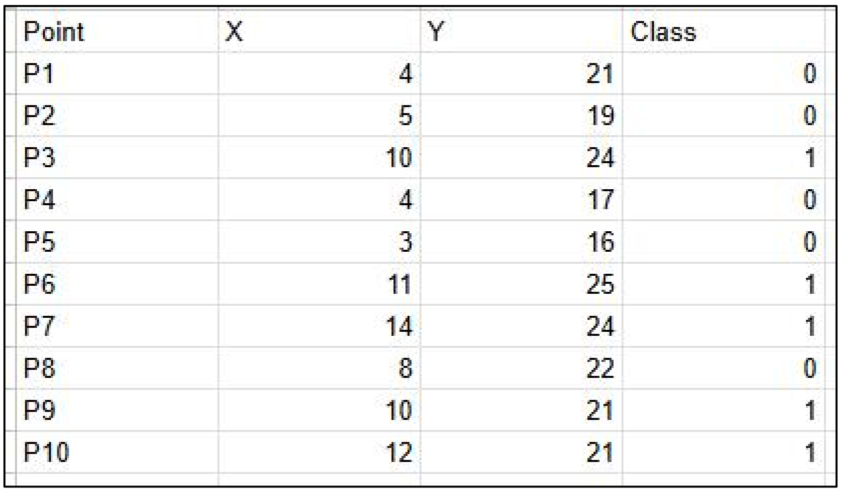
- **Left Side: Small K (e.g., K = 1)**
  - The decision boundary (red line) is **highly irregular**.
  - The classifier tries to perfectly fit every data point even the noise.
  - This leads to **overfitting**:
    1. The model memorizes the training data.
    2. It performs well on training data but poorly on new and unseen data.
    3. The boundary fluctuates a lot and becomes too complex.

- **Right Side: Large K (e.g., K = 15)**
  - The decision boundary becomes **very smooth** and almost straight.
  - Too many neighbors are considered, so the model becomes overly generalized.
  - This results in **underfitting**:
    1. The model fails to capture important patterns.
    2. Classification becomes too simple.

7. KNN Classification Numerical Example

- **Training Data:** The pairs (x, y) are the features, and class is the label for the training set.
- **Query Point:** The point to be classified is the **mean** of the x and y values.
- **K Value:** We will use **K = 3**.

- **Distance Metric:** We will use **Euclidean Distance**.



Point	X	Y	Class
P1	4	21	0
P2	5	19	0
P3	10	24	1
P4	4	17	0
P5	3	16	0
P6	11	25	1
P7	14	24	1
P8	8	22	0
P9	10	21	1
P10	12	21	1

Figure 7: KNN Classification Numerical

- **Data Setup**

  The training set consists of 10 points in a 2D feature space (X, Y):

- **Calculate the Query Point (Mean)**

  First, we find the average values for X and Y to define our point to be classified.

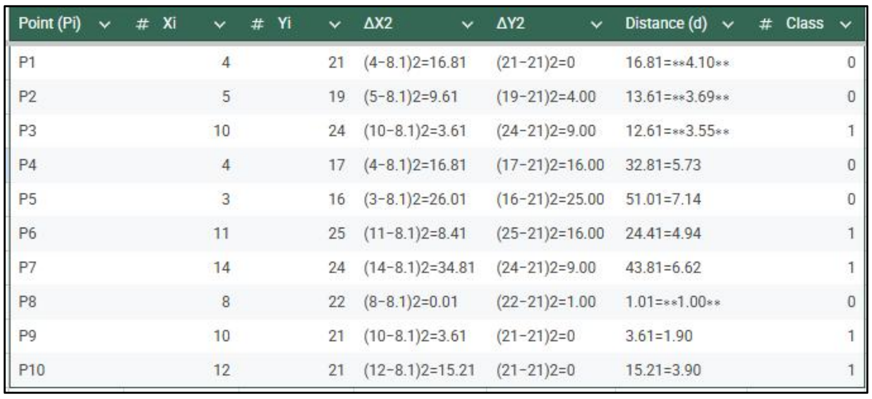
  bar{X} = 81 / 10 = 8.1

  bar{Y} = 210 / 10 = 21.0

  Query Point (Q): (8.1, 21.0)

- **Calculate Euclidean Distance (K=3)**

  We calculate the distance between the Query Point (8.1, 21.0) and every point in the dataset.

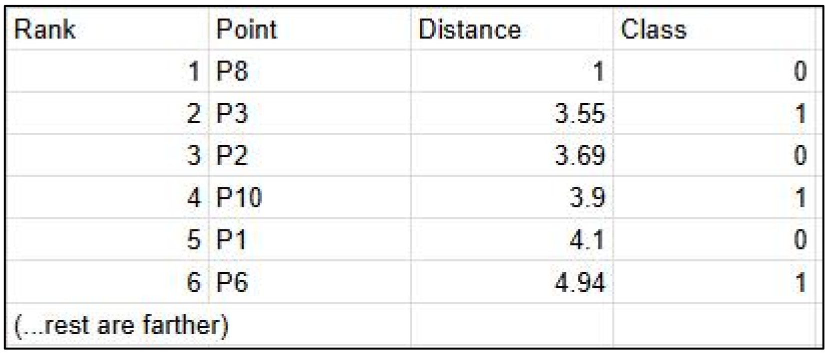


Point (Pi)	# Xi	# Yi	ΔX2	ΔY2	Distance (d)	# Class
P1	4	21	(4−8.1)2=16.81	(21−21)2=0	16.81=**4.10**	0
P2	5	19	(5−8.1)2=9.61	(19−21)2=4.00	13.61=**3.69**	0
P3	10	24	(10−8.1)2=3.61	(24−21)2=9.00	12.61=**3.55**	1
P4	4	17	(4−8.1)2=16.81	(17−21)2=16.00	32.81=5.73	0
P5	3	16	(3−8.1)2=26.01	(16−21)2=25.00	51.01=7.14	0
P6	11	25	(11−8.1)2=8.41	(25−21)2=16.00	24.41=4.94	1
P7	14	24	(14−8.1)2=34.81	(24−21)2=9.00	43.81=6.62	1
P8	8	22	(8−8.1)2=0.01	(22−21)2=1.00	1.01=**1.00**	0
P9	10	21	(10−8.1)2=3.61	(21−21)2=0	3.61=1.90	1
P10	12	21	(12−8.1)2=15.21	(21−21)2=0	15.21=3.90	1

Figure 8: Calculating Euclidean Distance

- **Find the K=3 Nearest Neighbors**

  We sort the points by distance (smallest to largest) and select the top 3.



Rank	Point	Distance	Class
1	P8	1	0
2	P3	3.55	1
3	P2	3.69	0
4	P10	3.9	1
5	P1	4.1	0
6	P6	4.94	1
(...rest are farther)			

Figure 9: Evaluating Distance for K

- **Majority Vote and Classification**

  The **K=3 Neighbors** are:

  - P8: Class **0**
  - P3: Class **1**
  - P2: Class **0**

  We count the votes:

  - **Class 0:** 2 votes
  - **Class 1:** 1 vote

  The new Query Point (8.1, 21.0) is classified as Class 0.

8. Applications of KNN in machine learning:

The KNN algorithm has been utilized within a variety of applications, largely within classification (Eda Kavlakoglu, 2023) . Some of these include:

8.1. Data Preprocessing

Many datasets contain missing values. KNN can fill in these missing entries by finding similar data points and estimating the most likely value. This process is called ***missing data imputation*.**

8.2. Recommendation Systems

a) KNN helps group users based on similar behavior or preferences.
b) Websites use it to recommend products, videos, or content based on users with similar activity patterns.
c) However, KNN can be slow for large-scale recommendation systems since it must compare each user with all others.

8.3. Healthcare

In healthcare, KNN can support medical predictions, such as estimating the risk of heart attacks or prostate cancer. It does this by analyzing gene expression patterns and finding similarities among patients. _(Cornelissen, R., 2022)

9. Advantages of KNN:

9.1 Easy to Implement

a) KNN is very simple and beginner-friendly.
b) It does not require complex formulas or advanced training steps.
c) You only need to store the dataset and calculate distances.
d) Because of its simplicity, it is often one of the first algorithms taught in machine learning.

9.2. Adapts Easily to New Data

a) KNN automatically adapts when new data is added.
b) There is no need to retrain the entire model.
c) The algorithm simply includes new points when finding nearest neighbors.
d) This makes it suitable for continuously changing or growing datasets.

9.3. Few Hyper parameters to Tune

Compared to other machine learning algorithms, KNN requires very little tuning. We mainly need to choose:

- the value of **k** (how many neighbors to consider)
- the **distance metric** (like Euclidean or Manhattan distance)

10. Disadvantages of KNN:

10.1 Does not scale well:

a) KNN stores the entire dataset, making it slow on large datasets.
b) High memory usage increases computational cost.
c) Distance calculations become time-consuming as data grows.

10.2 Curse of dimensionality:

a) KNN performs poorly with high-dimensional data.
b) Too many features reduce the effectiveness of distance-based classification.
c) More features can increase classification errors, especially with small samples.

10.3 Prone to overfitting:

a) With a very small k (e.g., k = 1), KNN may memorize the training data and overfit.
b) High-dimensional data increases overfitting risk.

c) Very large k can cause under fitting by over smoothing class boundaries.

11. Conclusion:

The K-Nearest Neighbors algorithm is powerful in its simplicity. Its behavior depends on the choice of *k*: small values of *k* make the model flexible but prone to overfitting, while large *k* values make it stable but makes it under fit (Eda Kavlakoglu, 2023) . KNN's intuitive approach (classify by "neighborhood voting") is easy to implement (as shown in the code examples) and understand, yet can achieve strong performance on many tasks. When using KNN, always remember to preprocess (normalize) the data and experiment with *k* to find the best solution for your problem (Ganga, et al., 2024) . Through code, visualization and practical numerical, we can see exactly how KNN partitions space and how those partitions change with *k* providing both insight and practical guidance for improving classification accuracy.

12. References:

1. IBM Developer (2022) 'Think: What is the k-nearest neighbors (KNN) algorithm?', *IBM Blog*. Available: https://www.ibm.com/think/topics/knblogs (Accessed: 3/12/25).
2. Ganga, D. and Hammoudeh, M. (2024). *K-Nearest-Neighbours,* GeeksforGeeks. [Online]. Available: https://www.geeksforgeeks.org/machine-learning/k-nearest-neighbours/ (Accessed: 1/12/25).
3. Cornelissen, R. et al. (2018). *Lecture 02: K-Nearest neighbors* CS 4780, Cornell University. Available: https://www.cs.cornell.edu (Accessed: 2/12/25).
4. Scikit-learn (n.d.) *K Nearest Neighbors*. Available at: https://scikit-learn.org/stable/modules/neighbors.html (Accessed: 3/12/2025).

Appendix : https://github.com/ZohaibAfzal295/Understanding-the-knn-Algorithm-and-effect-of-k-on-accuracy