Chapter 3: Classification and Clusturing

Classification in Machine Learning:

Definition:
Classification is a supervised learning technique where the goal is to predict the categorical label of new data points based on historical data. The model is trained on labeled data, meaning that each training instance has both input features and a known label (target value). The model learns the relationship between the input features and the labels so that it can predict the label for unseen data.

Key Points:

- **Supervised learning**: The model is trained with labeled data.
- **Output**: The output is discrete, meaning it belongs to a finite set of categories or classes.
- **Examples**:
    - **Spam detection**: Classifying emails as "spam" or "not spam".
    - **Medical diagnosis**: Predicting whether a patient has a certain disease (e.g., "positive" or "negative").
    - **Image recognition**: Identifying objects in an image (e.g., "cat", "dog", "car").

Algorithms:

- Decision Trees
- k-Nearest Neighbors (k-NN)
- Support Vector Machines (SVM)
- Logistic Regression
- Naive Bayes
- Neural Networks

Algorithm 1: K-Nearest Neighbours(k-NN)

Definition of k-Nearest Neighbors (k-NN):

k-Nearest Neighbors (k-NN) is a **supervised machine learning algorithm** used for **classification** and **regression tasks**. It is a simple, instance-based, and non-parametric algorithm that classifies data points or predicts outcomes based on the similarity to its **k nearest neighbors** in the feature space.

Key Characteristics of k-NN:

1. **Lazy Learning**:
   - k-NN does not build a model during training.
   - It stores the entire training dataset and performs computation only at prediction time.
2. **Instance-Based**:
   - Predictions are made by comparing the new data point with existing data points in the training set.
3. **Similarity Measure**:
   - Typically uses **distance metrics** such as Euclidean, Manhattan, or Minkowski distance to find the closest neighbors.
4. **Versatile**:
   - Can handle both **classification** (categorical outputs) and **regression** (continuous outputs).

How k-NN Works:

1. **Training Phase**:
   - No explicit training; the training data is simply stored.
2. **Prediction Phase**:
   - **Step 1**: Compute the distance between the new data point and all points in the training dataset.
   - **Step 2**: Identify the **k closest neighbors** (smallest distances).
   - **Step 3**:
     - For **classification**: Perform a **majority vote** among the k neighbors to assign a class label.
     - For **regression**: Compute the **average target value** of the k neighbors.
3. **Output**:
   - A predicted class label (classification) or a continuous value (regression).

k-NN Classification Algorithm:

The k-Nearest Neighbors (k-NN) algorithm is a simple, instance-based learning algorithm used for both classification and regression tasks. For classification, the algorithm assigns a class label based on the majority vote of its k nearest neighbors.

Algorithm for k-NN Classification:

1. **Input**:
   - Training dataset $D = \{(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)\}$, where x_i represents the features and y_i represents the class label of the i-th instance.
   - A new data point x_{new} that needs to be classified.
   - A value of k, the number of nearest neighbors to consider.
2. **Steps**:
   1. **Compute the distance** between the new data point x_{new} and all the points in the training dataset D. Typically, **Euclidean distance** is used:

      $$d(x_{new}, x_i) = \sqrt{(x_{new1} - x_{i1})^2 + (x_{new2} - x_{i2})^2 + ... + (x_{newm} - x_{im})^2}$$
   2. **Sort the distances** from the new data point to all points in the training dataset.
   3. **Select the k nearest neighbors** based on the sorted distances.
   4. **Vote on the class label** of the k nearest neighbors:
      - Count the frequency of each class label among the k neighbors.
      - Assign the class label that appears most frequently to x_{new}.
3. **Output**:
   - The predicted class label for the new data point x_{new}.

k-NN Regression Algorithm:

In k-NN regression, instead of assigning a class label, the algorithm predicts a continuous value by averaging the outputs (target values) of the k nearest neighbors.

Algorithm for k-NN Regression:

1. **Input:**
   - Training dataset $D = \{(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)\}$, where x_i represents the features and y_i represents the continuous target value of the i-th instance.
   - A new data point x_{new} whose target value y_{new} needs to be predicted.
   - A value of k, the number of nearest neighbors to consider.
2. **Steps:**
   1. **Compute the distance** between the new data point x_{new} and all points in the training dataset D.
   2. **Sort the distances** from the new data point to all points in the training dataset.
   3. **Select the k nearest neighbors** based on the sorted distances.
   4. **Calculate the average target value** of the k nearest neighbors:

      $$y_{new} = \frac{1}{k} \sum_{i=1}^{k} y_i$$

   5. **Output** the predicted target value y_{new}.
3. **Output:**
   - The predicted target value for the new data point x_{new}.

k-NN Classification Example (Complete Walkthrough)

We will use the dataset provided and apply the **k-Nearest Neighbors (k-NN)** algorithm to classify a new data point based on **height** and **weight** features.

Dataset:

Height (cm)	Weight (kg)	Class
150	55	A
160	60	A
170	65	B
180	70	B

We need to classify a new data point:

- **Height = 165 cm**
- **Weight = 62 kg**

We will use **k** = 3 for the classification.

Step 1: Calculate Euclidean Distance

The formula for **Euclidean distance** between two points $p_1 = (x_1, y_1)$ and $p_2 = (x_2, y_2)$ is:

$$d(p_1, p_2) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Let's compute the distances from the new point (165, 62) to each point in the training dataset.

Step 1: Calculate Euclidean Distance

The formula for **Euclidean distance** between two points $p_1 = (x_1, y_1)$ and $p_2 = (x_2, y_2)$ is:

$$d(p_1, p_2) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Let's compute the distances from the new point (165, 62) to each point in the training dataset.

1. **Distance to (150, 55):**

$$d(150, 55) = \sqrt{(165 - 150)^2 + (62 - 55)^2} = \sqrt{15^2 + 7^2} = \sqrt{225 + 49} = \sqrt{274} \approx 16.55$$

2. **Distance to (160, 60):**

$$d(160, 60) = \sqrt{(165 - 160)^2 + (62 - 60)^2} = \sqrt{5^2 + 2^2} = \sqrt{25 + 4} = \sqrt{29} \approx 5.39$$

3. **Distance to (170, 65):**

$$d(170, 65) = \sqrt{(165 - 170)^2 + (62 - 65)^2} = \sqrt{(-5)^2 + (-3)^2} = \sqrt{25 + 9} = \sqrt{34} \approx 5.83$$

4. **Distance to (180, 70):**

$$d(180, 70) = \sqrt{(165 - 180)^2 + (62 - 70)^2} = \sqrt{(-15)^2 + (-8)^2} = \sqrt{225 + 64} = \sqrt{289} \approx 17$$

Step 2: Find the k Nearest Neighbors

Now that we have the distances, we need to find the **3 nearest neighbors** (since we chose $k = 3$).

- Distance to (160, 60): 5.39
- Distance to (170, 65): 5.83
- Distance to (150, 55): 16.55
- Distance to (180, 70): 17

Step 2: Find the k Nearest Neighbors

Now that we have the distances, we need to find the **3 nearest neighbors** (since we chose $k = 3$).

- Distance to (160, 60): 5.39
- Distance to (170, 65): 5.83
- Distance to (150, 55): 16.55
- Distance to (180, 70): 17

The **3 nearest neighbors** are:

1. **(160, 60)** with Class A (distance = 5.39)
2. **(170, 65)** with Class B (distance = 5.83)
3. **(150, 55)** with Class A (distance = 16.55)

Step 3: Majority Vote

Now, we will apply the **majority vote** rule to determine the class of the new data point. The classes of the 3 nearest neighbors are:

- **Class A:** 2 occurrences
- **Class B:** 1 occurrence

Since **Class A** appears more frequently among the k nearest neighbors, the new data point **(165, 62)** is classified as **Class A**.

Final Result:

- **Predicted Class: Class A**

Summary of Steps:

1. **Calculate distances** between the new point and all training points.
2. **Find the k nearest neighbors** (in this case, $k = 3$).
3. **Perform majority voting** among the neighbors.
4. The predicted class for the new point is **Class A** based on the majority vote.

This is the basic procedure for **k-Nearest Neighbors** classification.

k-NN Regression Example

Let's apply the **k-Nearest Neighbors (k-NN)** algorithm for **regression.** In k-NN regression, instead of using majority voting to predict a class label, we use the average (or sometimes the weighted average) of the values of the **k nearest neighbors** to predict the output for the new data point.

Dataset:

Height (cm)	Weight (kg)	Price ($)
150	55	200
160	60	250
170	65	300
180	70	350

We need to predict the price for a new data point with:

- **Height = 165 cm**
- **Weight = 62 kg**

We'll use **k** = 3 for the prediction.

Step 1: Calculate Euclidean Distance

Just like classification, we first calculate the **Euclidean distance** between the new point and all the points in the training dataset.

The formula for Euclidean distance is the same:

$$d(p_1, p_2) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

1. **Distance to (150, 55):**

$$d(150, 55) = \sqrt{(165 - 150)^2 + (62 - 55)^2} = \sqrt{15^2 + 7^2} = \sqrt{225 + 49} = \sqrt{274} \approx 16.55$$

2. **Distance to (160, 60):**

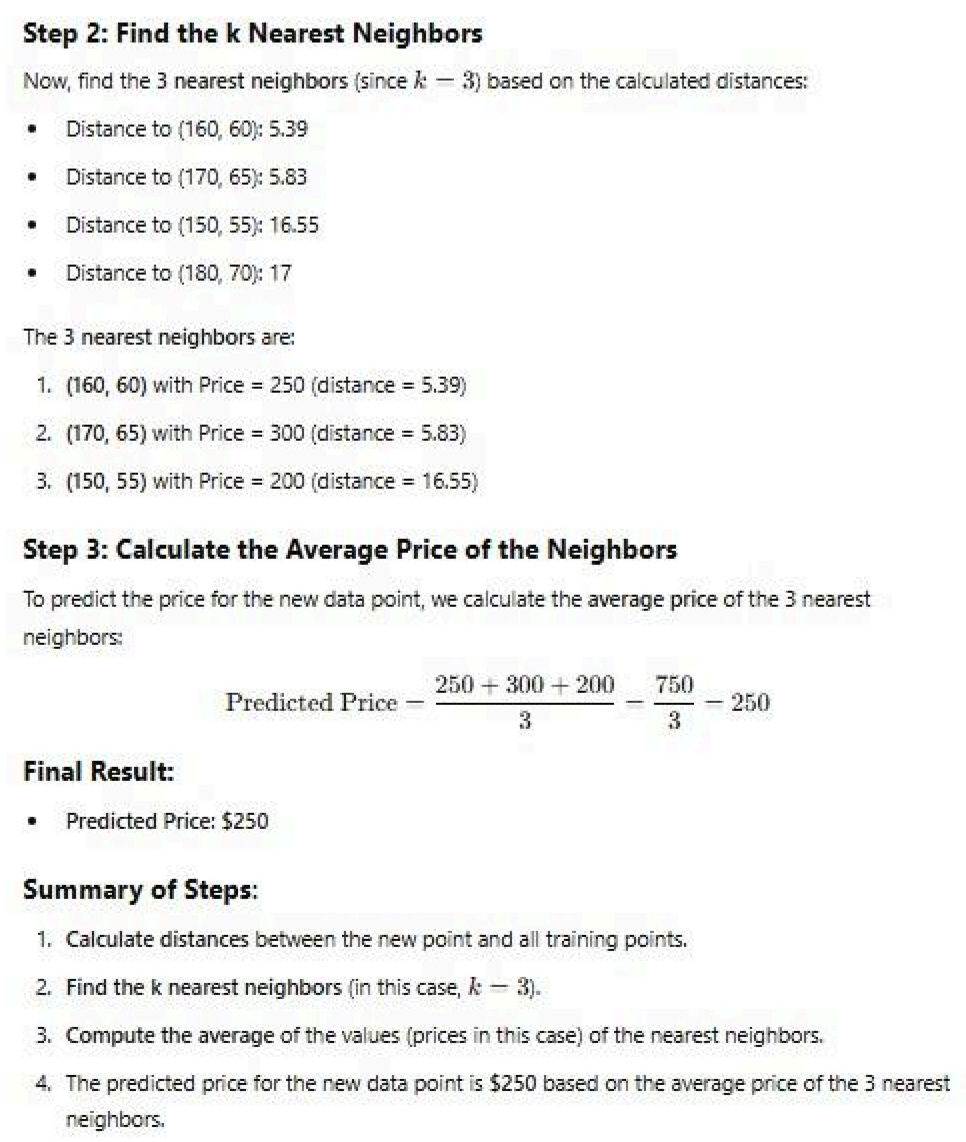
$$d(160, 60) = \sqrt{(165 - 160)^2 + (62 - 60)^2} = \sqrt{5^2 + 2^2} = \sqrt{25 + 4} = \sqrt{29} \approx 5.39$$

3. **Distance to (170, 65):**

$$d(170, 65) = \sqrt{(165 - 170)^2 + (62 - 65)^2} = \sqrt{(-5)^2 + (-3)^2} = \sqrt{25 + 9} = \sqrt{34} \approx 5.83$$

4. **Distance to (180, 70):**

$$d(180, 70) = \sqrt{(165 - 180)^2 + (62 - 70)^2} = \sqrt{(-15)^2 + (-8)^2} = \sqrt{225 + 64} = \sqrt{289} \approx 17$$



Step 2: Find the k Nearest Neighbors

Now, find the 3 nearest neighbors (since $k = 3$) based on the calculated distances:

- Distance to (160, 60): 5.39
- Distance to (170, 65): 5.83
- Distance to (150, 55): 16.55
- Distance to (180, 70): 17

The 3 nearest neighbors are:

1. **(160, 60)** with Price = 250 (distance = 5.39)
2. **(170, 65)** with Price = 300 (distance = 5.83)
3. **(150, 55)** with Price = 200 (distance = 16.55)

Step 3: Calculate the Average Price of the Neighbors

To predict the price for the new data point, we calculate the **average price** of the 3 nearest neighbors:

$$\text{Predicted Price} = \frac{250 + 300 + 200}{3} = \frac{750}{3} = 250$$

Final Result:

- **Predicted Price: $250**

Summary of Steps:

1. **Calculate distances** between the new point and all training points.
2. **Find the k nearest neighbors** (in this case, $k = 3$).
3. **Compute the average** of the values (prices in this case) of the nearest neighbors.
4. The predicted price for the new data point is **$250** based on the average price of the 3 nearest neighbors.

k-NN Classification

```
# Import necessary modules
from sklearn.neighbors import KNeighborsClassifier
import numpy as np

# Training data: Features (Height, Weight) and Labels (Class A or B)
X_train = np.array([[150, 55], [160, 60], [170, 65], [180, 70]])  # Feature matrix
y_train = np.array(['A', 'A', 'B', 'B'])  # Class labels

# Test data: New data point to classify
X_test = np.array([[165, 62]])

# Initialize the k-NN classifier with k=3
```

```
knn_classifier = KNeighborsClassifier(n_neighbors=3)

# Train the classifier using the training data
knn_classifier.fit(X_train, y_train)

# Predict the class for the test data
predictions = knn_classifier.predict(X_test)

# Print the predicted class for the test point
print("Predicted class:", predictions[0])  # Expected output: 'A'
```

k-NN Regression

```
# Import necessary modules
from sklearn.neighbors import KNeighborsRegressor
import numpy as np

# Training data: Features (Height, Weight) and Target values
X_train = np.array([[150, 55], [160, 60], [170, 65], [180, 70]])  # Feature matrix
y_train = np.array([200, 250, 300, 350])  # Target values (e.g., prices, weights)

# Test data: New data point to predict the target value
X_test = np.array([[165, 62]])

# Initialize the k-NN regression with k=3
knn_regressor = KNeighborsRegressor(n_neighbors=3)

# Train the regressor using the training data
knn_regressor.fit(X_train, y_train)

# Predict the target value for the test data
predictions = knn_regressor.predict(X_test)

# Print the predicted value for the test point
print("Predicted value:", predictions[0])  # Expected output: ~266.67
```