Module3

Machine Learning

Syllabus

1. **Similarity-based Learning:** Nearest-Neighbor Learning, Weighted K-Nearest-Neighbor Algorithm, Nearest Centroid Classifier, Locally Weighted Regression (LWR).
2. **Regression Analysis:** Introduction to Regression, Introduction to Linear Regression, Multiple Linear Regression, Polynomial Regression, Logistic Regression.
3. **Decision Tree Learning:** Introduction to Decision Tree Learning Model, Decision Tree Induction Algorithms.

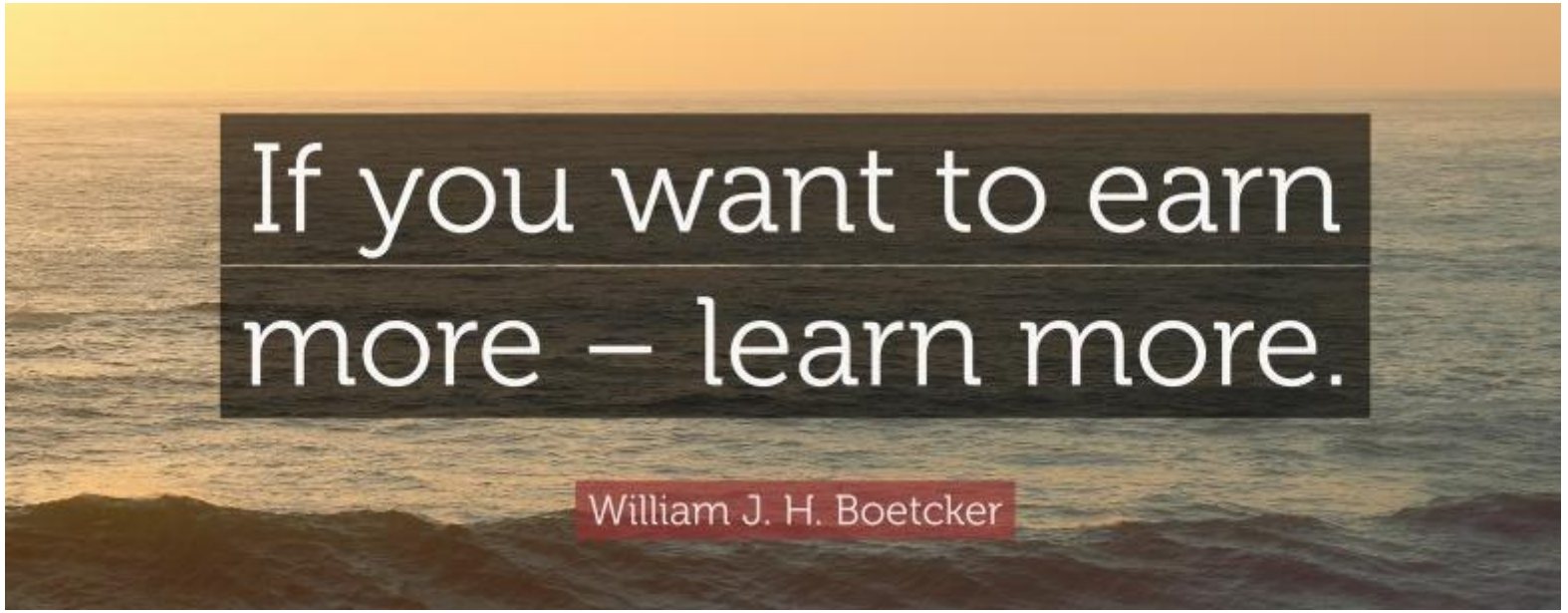
3.1 : Similarity-based Learning

1. Nearest-Neighbor Learning,
2. Weighted K-Nearest-Neighbor Algorithm,
3. Nearest Centroid Classifier,
4. Locally Weighted Regression (LWR).

Similarity based Learning

"**Any one who stops learning is old, whether at twenty or eight**"

….Henry Ford



What is similarity based learning ?

- SBL is a supervised learning techniques that predicts the class label of test instance by **gauging the similarity of this test instance** with training instances.
- It is also called as **Instance based learning** /Just in time learning when classifying a new instance. Instance is an entity or an example in the training data set.
- It considers only the **nearest instance or instances** to predict the class of unseen instances.
- It uses **distance metrics** to find the **similarity or dissimilarity** required for nearest neighbor classification.

Difference between Instance based and Model based Learning

Aspect	Instance-based Learning	Model-based Learning
Learning Approach	Lazy Learners	Eager Learners
Processing Time	Training instances are processed only during the testing phase.	Training instances are processed during the training phase.
Model Construction	No model is built before receiving a test instance.	A model is generalized using training instances before receiving a test instance.
Prediction Method	Predicts the class of the test instance directly from the training data.	Predicts the class of the test instance using the constructed model.
Testing Speed	Slow in the testing phase.	Fast in the testing phase.
Learning Strategy	Learns by making many local approximations.	Learns by creating a global approximation.

Examples

1. K Nearest neighbor (**KNN**)
2. Variants of Nearest Neighbor learning
3. Locally weighted Regression **(LWR)**
4. Learning Vector Quantization (**LVQ**)
5. Self Organization Map (**SOM**)
6. Radial Basis Function(**RBF**) networks

Nearest Neighbor Learning

- A widely used approach for similarity-based classification is **k-Nearest Neighbors (k-NN)**. It is a non-parametric method applicable to both classification and regression tasks. This algorithm is simple yet powerful, predicting the category of a test instance by considering the 'k' training samples that are closest to it. The test instance is assigned to the category with the highest probability based on the majority class among its nearest neighbors.
- A visual representation of this concept is shown in **Figure 4.1**. The figure illustrates two object classes, $\mathbf{C_1}$ and $\mathbf{C_2}$. Given a test instance **T**, its category is determined by analyzing the class of its **k = 3** nearest neighbors. Based on this approach, the test instance **T** is classified as belonging to $\mathbf{C_2}$.

Nearest Neighbor Learning

- A visual representation of this concept is shown in **Figure**.
- The figure illustrates two object classes, $\mathbf{C_1}$ and $\mathbf{C_2}$.
- Given a test instance **T**, its category is determined by analyzing the class of its **k = 3** nearest neighbors.
- Based on this approach, the test instance **T** is classified as belonging to $\mathbf{C_2}$.

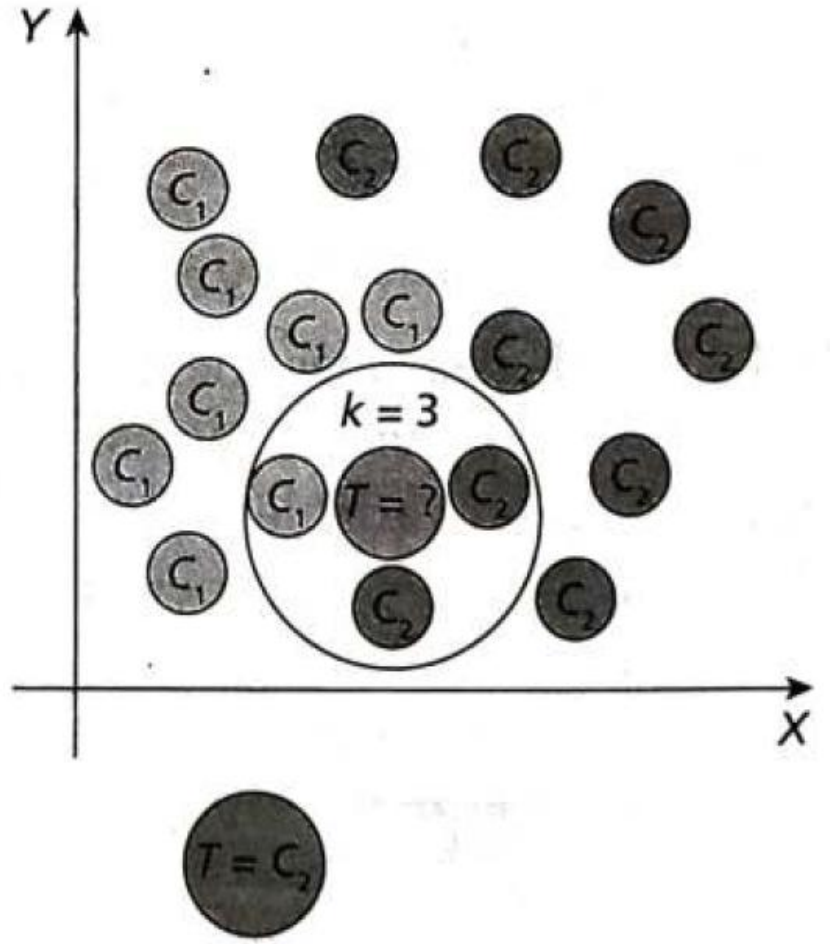


Figure 4.1: Visual Representation of *k*-Nearest Neighbor Learning

Algorithm : k-Nearest Neighbors (k-NN)

- **Inputs:**
  - Training dataset **T**
  - Distance metric **d**
  - Test instance **t**
  - Number of nearest neighbors **k**
- **Output:**
  - Predicted class or category of **t**

- **Prediction Process**

Prediction Process:

1. **Compute Distances:**
    - For each instance **i** in **T**, calculate the distance between the test instance **t** and every other instance **i** in the training dataset using a suitable distance metric.
    - **For continuous attributes:** Use the Euclidean distance formula:

$$\text{dist}((x_1, y_1), (x_2, y_2)) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

    - **For categorical (binary) attributes:** Use the Hamming distance:
        - If the values of two instances are the same, **d = 0**
        - Otherwise, **d = 1**

2. **Sort and Select Neighbors:**
    - Arrange the computed distances in ascending order.
    - Select the **k** nearest training instances to the test instance **t**.
3. **Predict the Class:**
    - If the target attribute is discrete, determine the class of **t** using **majority voting** among the **k** nearest neighbors.
    - If the target attribute is continuous, predict the value of **t** as the **mean** of the values of the **k** nearest neighbors.

Example : Student Performance Classification Using k-NN

- Consider a student performance training dataset consisting of 8 data instances, as shown in **Table.** This dataset represents the academic performance of individual students in a course, including their **CGPA**, **Assessment Scores**, and **Project Submissions** from previous semesters.
- The **independent attributes** in this dataset are:
  - **CGPA** (Cumulative Grade Point Average)
  - **Assessment Scores**
  - **Project Submitted**
- The **target variable** is **"Result"**, a discrete-valued variable that indicates whether a student **Passes** or **Fails** in the course. The goal is to classify whether a student will pass or fail based on their performance attributes using a **k-Nearest Neighbors (k-NN) algorithm**.

Table 4.2: Training Dataset T

S. No.	CGPA	Assessment	Project Submitted	Result
1	9.2	85	8	Pass
2	8.0	80	7	Pass
3	8.5	81	8	Pass
4	6.0	45	5	Fail
5	6.5	50	4	Fail
6	8.2	72	7	Pass
7	5.8	38	5	Fail
8	8.9	91	9	Pass

This dataset is used to predict whether a new student, based on their CGPA, assessment scores, and project submissions, will pass or fail using **k-NN classification**.

Solution: k-NN Classification Using Euclidean Distance

- Given a **test instance (6.1, 40, 5)** and a set of categories (**Pass, Fail**), also referred to as **classes**, we need to use the training dataset to classify the test instance using the **Euclidean distance metric**.
- The goal of classification is to assign a class (**Pass** or **Fail**) to an unknown instance.
  We set **k = 3** (using the k-nearest neighbors approach).

Step 1: Compute Euclidean Distance

- We calculate the **Euclidean distance** between the test instance **(6.1, 40, 5)** and each training instance in the dataset. The distances are shown in **Table 4.3**.

Table 4.3: Euclidean Distance Computation

S. No.	CGPA	Assessment	Project Submitted	Result	Euclidean Distance
1	9.2	85	8	Pass	$\sqrt{(9.2-6.1)^2+(85-40)^2+(8-5)^2}$ = **45.2063**
2	8.0	80	7	Pass	$\sqrt{(8-6.1)^2+(80-40)^2+(7-5)^2}$ = **40.09501**

3	8.5	81	8	Pass	$\sqrt{(8.5-6.1)^2+(81-40)^2+(8-5)^2}$ = **41.17961**
4	6.0	45	5	Fail	$\sqrt{(6-6.1)^2+(45-40)^2+(5-5)^2}$ = **5.001**
5	6.5	50	4	Fail	$\sqrt{(6.5-6.1)^2+(50-40)^2+(4-5)^2}$ = **10.05783**
6	8.2	72	7	Pass	$\sqrt{(8.2-6.1)^2+(72-40)^2+(7-5)^2}$ = **32.13114**
7	5.8	38	5	Fail	$\sqrt{(5.8-6.1)^2+(38-40)^2+(5-5)^2}$ = **2.022375**
8	8.9	91	9	Pass	$\sqrt{(8.9-6.1)^2+(91-40)^2+(9-5)^2}$ = **51.23319**

Step 2: Selecting the Nearest Neighbors

- To classify the test instance, we sort the calculated Euclidean distances in **ascending order** and select the **three nearest training instances**. The selected **nearest neighbors** are displayed in **Table 4.4**.

Table 4.4: Nearest Neighbors

Instance	Euclidean Distance	Class
4	5.001	Fail
5	10.05783	Fail
7	2.022375	Fail

Thus, the **three nearest neighbors** are **instances 4, 5, and 7**, which have the **smallest Euclidean distances**.

Step 3: Classification by Majority Voting

- The final classification decision is made using **majority voting** among the three nearest neighbors.
- All three selected neighbors belong to the **Fail** class.
- Therefore, the **predicted class for the test instance** is:
- **Final Prediction: "Fail".**

Data Normalization and k-NN Classifier Performance

- **Data normalization or standardization** is necessary when different features in a dataset have varying ranges. This is essential for computing distances accurately and ensuring that no single feature dominates the calculation. **The goal is to bring all features within a common scale, providing equal influence**.
- For instance, consider a dataset where one feature has values in the range **[0,1]**, while another feature has values in the range **[0,100]**. In this case, the second feature will contribute more to the distance calculation, even for small variations, leading to biased results.

Factors Affecting k-NN Classifier Performance

- The **k-Nearest Neighbors (k-NN) classifier** is influenced by three key factors:
- **Number of nearest neighbors (k):**
  - A small **k** may lead to **overfitting** or unstable predictions.
  - A large **k** may introduce **irrelevant points** from other classes, reducing accuracy.
- **Choice of distance metric:**
  - Different metrics impact classification results and depend on the dataset's features.
- **Decision rule:**
  - The strategy used for assigning a class label based on **nearest neighbors** affects performance.

Suitability of k-NN in Different Data Spaces

- The **k-NN algorithm** works best in **low-dimensional spaces**.
- In **high-dimensional spaces**, distances become less meaningful, and nearest neighbors may not be close to each other, reducing classification effectiveness.

4.3 Weighted k-Nearest Neighbor (k-NN) Algorithm

- The **Weighted k-NN** is an extension of the standard **k-NN algorithm** that incorporates weighted distances when selecting neighbors. Traditional **k-NN** has limitations since its performance depends solely on the choice of **k nearest neighbors**, the **distance metric**, and the **decision rule**.
- The key concept behind **Weighted k-NN** is that **closer neighbors** to the test instance are assigned **higher weights** compared to those farther away. The weighting follows an **inverse proportionality to distance**, meaning that closer points have a stronger influence on classification decisions.

Weight Assignment in Weighted k-NN

- The selected **k nearest neighbors** can be assigned weights in two ways:
- **Uniform Weights:** All neighbors contribute equally to the classification decision.
- **Inverse Distance Weights:** Neighbors closer to the test instance have a greater influence, while those farther away contribute less.
- By using inverse distance weighting, **Weighted k-NN** improves accuracy by giving more importance to **closer** instances, reducing the impact of irrelevant distant points.

Algorithm : Weighted k-NN

- **Inputs:**
  - Training dataset T
  - Distance metric d
  - **Weighting function w(i)**
  - Test instance t
  - Number of nearest neighbors K
- **Output:**
  - Predicted class or category
- **Prediction Process (for test instance t):**

Prediction Process (for test instance):

1. **Compute Distances:**
   - For each instance t' in the training dataset T, calculate the distance between the test instance t and t' using a specified distance metric.
   - **Continuous Attributes:** Use Euclidean distance:

$$\text{dist}((x_1, y_1), (x_2, y_2)) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

   - **Categorical Attributes (Binary):** Use Hamming Distance:
     - If the values of two instances are the same, $d = 0$. Otherwise, $d = 1$.

Prediction Process (for test instance):

2. **Sort and Select Neighbors:**
    - Arrange the distances in ascending order.
    - Select the first K nearest training instances to the test instance.

Prediction Process (for test instance):

3. **Class Prediction using Weighted Voting:**
    - Apply the weighting function $w(i)$ to the K selected nearest instances:
        - Compute the inverse of each distance for the selected K nearest instances.
        - Sum the inverses.
        - Compute each weight by dividing the inverse of a distance by the sum.
        - Each weight represents a vote for its corresponding class.
        - Sum the weights for each class.
        - The predicted class is the one with the highest total weight.

Example 4.2: Weighted k-NN Classification

- **Problem Statement :** Using the given training dataset from Table used for previous example, apply the **Weighted k-NN** algorithm to classify a test instance.

Solution:

Step 1: Compute Euclidean Distances

- Given test instance: (**CGPA = 7.6, Assessment = 60, Project Submitted = 8**)
- Classes: **Pass, Fail**
- Distance metric: **Euclidean distance**
- Assign $k = 3$ (number of nearest neighbors).

Table 4.5: Euclidean Distance Calculations

t= (7.6, 60, 8)

S.No	CGPA	Assessment	Project Submitted	Result	Euclidean Distance
1	9.2	85	8	Pass	$\sqrt{(9.2-7.6)^2+(85-60)^2+(8-8)^2} = 25.05115$
2	8.0	80	7	Pass	$\sqrt{(8-7.6)^2+(80-60)^2+(7-8)^2} = 20.02898$
3	8.5	81	8	Pass	$\sqrt{(8.5-7.6)^2+(81-60)^2+(8-8)^2} = 21.01928$
4	6.0	45	5	Fail	$\sqrt{(6-7.6)^2+(45-60)^2+(5-8)^2} = 15.38051$
5	6.5	50	4	Fail	$\sqrt{(6.5-7.6)^2+(50-60)^2+(4-8)^2} = 10.82636$
6	8.2	72	7	Pass	$\sqrt{(8.2-7.6)^2+(72-60)^2+(7-8)^2} = 12.05653$
7	5.8	38	5	Fail	$\sqrt{(5.8-7.6)^2+(38-60)^2+(5-8)^2} = 22.27644$
8	8.9	91	9	Pass	$\sqrt{(8.9-7.6)^2+(91-60)^2+(9-8)^2} = 31.04336$

Step 2: Select k Nearest Neighbors

- Sort the distances in ascending order and select the first 3 nearest training instances.

Table 4.6: Nearest Neighbors

Instance	Euclidean Distance	Class
4	15.38051	Fail
5	10.82636	Fail
6	12.05653	Pass

Step 3: Compute Weighted Voting for Classification

- Compute the inverse of each distance for the selected k nearest instances.

Table 4.7: Inverse Distance Calculation

Instance	Euclidean Distance	Inverse Distance	Class
4	15.38051	0.06502	Fail
5	10.82636	0.092370	Fail
6	12.05653	0.08294	Pass

- **Find the sum of the inverse distances:**

$$0.06502 + 0.092370 + 0.08294 = 0.24033$$

- **Compute the weight for each class by dividing the inverse distance by the sum.**

Table 4.8: Weight Calculation

Instance	Euclidean Distance	Inverse Distance	Weight (Inverse Distance / Sum)	Class
4	15.38051	0.06502	0.270545	Fail
5	10.82636	0.092370	0.384347	Fail
6	12.05653	0.08294	0.345109	Pass

- Add the weights for each class:
  - Fail: $0.270545 + 0.384347 = 0.654892$
  - Pass: 0.345109
- Final Prediction: Since **Fail (0.654892)** > **Pass (0.345109)**, the test instance is classified as '**Fail**'.

Final Prediction:

✅ The test instance (7.6, 60, 8) is classified as 'Fail'.

4.4 Nearest Centroid Classifier

- An alternative to **k-NN** classifiers for similarity-based classification is the **Nearest Centroid Classifier**.
- Also known as the **Mean Difference Classifier**, this method assigns a test instance to the class whose **centroid (mean) is closest** to it.

Algorithm : Nearest Centroid Classifier

- **Inputs:**
  - Training dataset **T**
  - Distance metric **d**
  - Test instance **t**
- **Output:**
  - Predicted class or category
- **Steps:**
  - Compute the **mean (centroid) for each class**.
  - Calculate the distance between the **test instance and the centroid** of each class using the **Euclidean Distance**.
  - Assign the test instance to the class with the **smallest distance**.

Example 4.3: Nearest Centroid Classification

- Consider the dataset shown in **Table 4.9**, which contains two features, **x** and **y**, and the target classes labeled as **'A'** or **'B'**.
- The objective is to predict the class of a given test instance using the **Nearest Centroid Classifier**.

Table 4.9: Sample Dataset

x	y	Class
3	1	A
5	2	A
4	3	A
7	6	B
6	7	B
8	5	B

Solution:

Step 1: Compute Centroids

The centroids for each class are calculated as follows:

- **Centroid of Class A**

$$\left(\frac{3+5+4}{3}, \frac{1+2+3}{3}\right) = \left(\frac{12}{3}, \frac{6}{3}\right) = (4, 2)$$

- **Centroid of Class B**

$$\left(\frac{7+6+8}{3}, \frac{6+7+5}{3}\right) = \left(\frac{21}{3}, \frac{18}{3}\right) = (7, 6)$$

Now, given a **test instance (6,5)**, we determine its class.

Solution:

Step 2: Compute Euclidean Distances

We calculate the Euclidean distance between the test instance **(6,5)** and each centroid:

- **Distance from Centroid of Class A (4,2):**

$$\text{Euc\_Dist}[(6,5);(4,2)] = \sqrt{(6-4)^2 + (5-2)^2}$$

- **Distance from Centroid of Class B (7,6):**

$$\text{Euc\_Dist}[(6,5);(7,6)] = \sqrt{(6-7)^2 + (5-6)^2}$$

$$= \sqrt{1+1} = \sqrt{2} \approx 1.414$$

Final Prediction:

Since the test instance **(6,5)** is **closer to Class B** than to Class A (**1.414 < 3.6**), it is classified as '**B**'.

4.5 Locally Weighted Regression (LWR)

- Locally Weighted Regression (LWR) is a non-parametric supervised learning algorithm that **performs local regression by incorporating the regression model with the nearest neighbor approach**.
- It is also known as a **memory-based method** since it requires **training data during prediction** but **only considers data points that are close to the test instance**.
- Using the nearest neighbors' approach, LWR selects instances closest to a test instance and fits a **linear function to those selected neighbors**.

4.5 Locally Weighted Regression (LWR)

- The key idea is to approximate the linear functions of the selected neighbors in such a way that the error is minimized.
- Unlike standard linear regression, where the prediction line is strictly linear, LWR allows the prediction to follow a curve.

Ordinary Linear Regression

In standard linear regression, a linear relationship is established between the input variable x and the output variable y. Given a training dataset T, the hypothesis function $h_\beta(x)$ predicts the target output as a linear equation:

$$h_\beta(x) = \beta_0 + \beta_1 x$$

where β_0 represents the intercept and β_1 is the coefficient of x.

Ordinary Linear Regression

The cost function used in linear regression aims to minimize the error between the predicted value $h_\beta(x)$ and the actual value y. It is defined as:

$$J(\beta) = \frac{1}{2}\sum_{i=1}^{m}(h_\beta(x_i) - y_i)^2$$

where m denotes the number of instances in the training dataset.

Locally Weighted Linear Regression

In LWR, the cost function is modified to include weights for the nearest neighbors, ensuring that points closer to the test instance have a greater influence. The updated cost function is:

$$J(\beta) = \frac{1}{2}\sum_{i=1}^{m} w_i(h_\beta(x_i) - y_i)^2$$

where w_i represents the weight assigned to each x_i.

Locally Weighted Linear Regression

The weight function is computed using a Gaussian kernel, which assigns higher weights to instances closer to the test instance. For instances farther away, the weight approaches zero but never becomes exactly zero. The weight function is given by:

$$w_i = e^{-\frac{(x-x_i)^2}{2\tau^2}}$$

where τ is the bandwidth parameter that determines the rate at which w_i decreases with distance from x_i.

Example 4.4: Locally Weighted Regression with a Sample Dataset

Consider a simple example with four instances as shown in Table 4.10, and apply locally weighted regression.

Table 4.10: Sample Dataset

S.No.	Salary (in lakhs)	Expenditure (in thousands)
1	5	25
2	1	5
3	2	7
4	1	8

Solution

Using a linear regression model, assume we have computed the parameters:

$$\beta_0 = 4.72, \quad \beta_1 = 0.62$$

For a test instance with $x = 2$, the predicted expenditure y' is:

$$y' = \beta_0 + \beta_1 x = 4.72 + 0.62 \times 2 = 5.96$$

Applying the nearest neighbor model, we select the $k = 3$ closest instances.

Table 4.11: Euclidean Distance Calculation

S.No.	x = Salary (in lakhs)	y = Expenditure (in thousands)	Euclidean Distance
1	5	25	$\sqrt{(5-2)^2} = 3$
2	1	5	$\sqrt{(1-2)^2} = 1$
3	2	7	$\sqrt{(2-2)^2} = 0$
4	1	8	$\sqrt{(1-2)^2} = 1$

Instances **2, 3, and 4** are the closest to the test instance based on the Euclidean distance.

The mean expenditure for these three closest instances is:

$$\frac{5+7+8}{3} = \frac{20}{3} = 6.67$$

Now, we compute the weights for the closest instances using the **Gaussian kernel**:

$$w_i = e^{-\frac{(x - x_i)^2}{2\tau^2}}$$

Weight Calculation for Closest Instances

Weight of Instance 2: $$w_2 = e^{-\frac{(1-2)^2}{2(0.4)^2}} = e^{-3.125} = 0.043$$

Weight of Instance 3: $$w_3 = e^{-\frac{(2-2)^2}{2(0.4)^2}} = e^{0} = 1$$

Weight of Instance 4: $$w_4 = e^{-\frac{(1-2)^2}{2(0.4)^2}} = e^{-3.125} = 0.043$$

Predicted Outputs for the Three Closest Instances

For each of the three nearest neighbors, we compute their predicted output:

- Predicted output for Instance 2:

$$y_2' = h_\beta(x_2) = 4.72 + 0.62 \times 1 = 5.34$$

- Predicted output for Instance 3:

$$y_3' = h_\beta(x_3) = 4.72 + 0.62 \times 2 = 5.96$$

- Predicted output for Instance 4:

$$y_4' = h_\beta(x_4) = 4.72 + 0.62 \times 1 = 5.34$$

Error Calculation

The cost function is computed as:

$$J(\beta) = \frac{1}{2} \sum_{i=1}^{m} w_i (h_\beta(x_i) - y_i)^2$$

Substituting the values:

$$J(\beta) = \frac{1}{2} \left[0.043(5.34 - 5)^2 + 1(5.96 - 7)^2 + 0.043(5.34 - 8)^2 \right]$$

$$J(\beta) = 0.6953$$

Now, we need to adjust this cost function to minimize the error and obtain optimal values for β.