📘 Chapter 3: Machine Learning

Mathematical Foundations + Numerical Applications + Python

🔷 1. Linear Regression — Mathematical Explanation

1️⃣ Model Definition

Linear regression assumes a **linear relationship** between input and output.

Single feature model:

$$\hat{y} = wx + b$$

Where:

- x: feature
- w: weight

Where:

- x: feature
- w: weight
- b: bias
- $\hat{y}$: predicted output

2️⃣ Objective Function (Loss Function)

We want predictions close to real values.

The most common loss is **Mean Squared Error (MSE)**:

$$J(w,b) = \frac{1}{n}\sum_{i=1}^{n}(y_i - (wx_i + b))^2$$

The goal is:

$$\min_{w,b} J(w,b)$$

Machine Learning Exercise — Electrical Engineering Application

You are given experimental measurements showing the relationship between **electric current** flowing in a conductor and the **measured electrical power loss.**

📊 Dataset

Current I (A)	1	2	3	4	5
Power P (W)	52	55	61	66	70

Because of temperature variation, noise, and measurement errors, the relationship is modeled using **linear regression**.

We assume the model:

$$\hat{P} = wI + b$$

❓ Questions

1. Write the **cost function** used in linear regression
2. Compute the **optimal values of w and b** using the **least squares method**
3. Predict the **power loss** when the current is **6 A**
4. Implement the solution using **Python**

🖍️ Mathematical Solution

1️⃣ Cost Function (Mean Squared Error)

$$J(w,b) = \frac{1}{n}\sum_{i=1}^{n}(P_i - (wI_i + b))^2$$

This function measures the **average squared electrical prediction error.**

2️⃣ Least Squares Formulas

$$w = \frac{n\sum I_i P_i - \sum I_i \sum P_i}{n\sum I_i^2 - (\sum I_i)^2}$$

$$b = \bar{P} - w\bar{I}$$

3️⃣ Compute Required Values

$$\sum I = 15, \quad \sum P = 304$$

$$\sum I^2 = 55, \quad \sum IP = 986$$

$$n = 5$$

🔷 Calculate w

$$w = \frac{5(986) - 15(304)}{5(55) - 15^2}$$

$$w = \frac{4930 - 4560}{275 - 225} = \frac{370}{50} = 7.4$$

🔷 Calculate b

$$\bar{I} = 3, \quad \bar{P} = 60.8$$

$$b = 60.8 - 7.4 \times 3 = 38.6$$

✅ Final Electrical Model

$$\boxed{\hat{P} = 7.4I + 38.6}$$

4️⃣ Prediction for $I = 6A$

$$\hat{P} = 7.4(6) + 38.6 = 83\ W$$

✔ **Predicted electrical power loss = 83 W**

python

```python
import numpy as np
# Electrical dataset
I = np.array([1, 2, 3, 4, 5])     # Current (A)
P = np.array([52, 55, 61, 66, 70])  # Power loss (W)
n = len(I)
# Least squares formulas
w = (n * np.sum(I * P) - np.sum(I) * np.sum(P)) / (n * np.sum(I**2) - np.sum(I)**2)
b = np.mean(P) - w * np.mean(I)
print("Slope w (W/A):", w)
print("Intercept b (W):", b)
# Prediction
I_new = 6
P_pred = w * I_new + b
print("Predicted power loss at 6 A:", P_pred, "W")
```

3️⃣ Gradients of the Cost Function

Partial derivative w.r.t. w

$$\frac{\partial J}{\partial w} = -\frac{2}{n} \sum I_i \left(P_i - (wI_i + b)\right)$$

Partial derivative w.r.t. b

$$\frac{\partial J}{\partial b} = -\frac{2}{n} \sum \left(P_i - (wI_i + b)\right)$$

4 Gradient Descent Update Rules

$$w := w - \alpha \frac{\partial J}{\partial w}$$

$$b := b - \alpha \frac{\partial J}{\partial b}$$

Where:

- α = learning rate
- Controls convergence speed and stability

1 Initialization (Iteration 0)

$$w^{(0)} = 0, \quad b^{(0)} = 0$$

Learning rate:

$$\alpha = 0.01$$

Number of samples:

$$n = 5$$

2 Predictions at Iteration 0

$$\hat{P}_i = wI_i + b = 0$$

So:

$$\hat{P} = [0,\ 0,\ 0,\ 0,\ 0]$$

3 Errors (Residuals)

$$e_i = P_i - \hat{P}_i$$

$$e = [52,\ 55,\ 61,\ 66,\ 70]$$

4 Compute Gradients

◆ Gradient with respect to w

$$\frac{\partial J}{\partial w} = -\frac{2}{n} \sum I_i (P_i - \hat{P}_i)$$

$$\sum I_i P_i = 1 \cdot 52 + 2 \cdot 55 + 3 \cdot 61 + 4 \cdot 66 + 5 \cdot 70$$

$$= 52 + 110 + 183 + 264 + 350 = 959$$

$$\frac{\partial J}{\partial w} = -\frac{2}{5} \times 959 = -383.6$$

🔷 Gradient with respect to b

$$\frac{\partial J}{\partial b} = -\frac{2}{n}\sum(P_i - \hat{P}_i)$$

$$\sum P_i = 52 + 55 + 61 + 66 + 70 = 304$$

$$\frac{\partial J}{\partial b} = -\frac{2}{5} \times 304 = -121.6$$

5️⃣ Parameter Update (End of Iteration 1)

🔷 Update w

$$w^{(1)} = w^{(0)} - \alpha\frac{\partial J}{\partial w}$$

$$w^{(1)} = 0 - 0.01(-383.6) = \boxed{3.836}$$

🔷 Update b

$$b^{(1)} = b^{(0)} - \alpha\frac{\partial J}{\partial b}$$

$$b^{(1)} = 0 - 0.01(-121.6) = \boxed{1.216}$$

✅ Result After One Iteration

$$\boxed{\hat{P} = 3.836I + 1.216}$$

➡️ Not yet optimal, but **closer to the real solution**

$(w \approx 7.4,\ b \approx 38.6)$

```python
import numpy as np
# Electrical dataset
I = np.array([1, 2, 3, 4, 5])      # Current (A)
P = np.array([52, 55, 61, 66, 70]) # Power loss (W)
n = len(I)
# Initialization
w = 0.0
b = 0.0
alpha = 0.01        # Learning rate
iterations = 1000
# Gradient Descent Loop
for _ in range(iterations):
    P_pred = w * I + b
    dw = (-2/n) * np.sum(I * (P - P_pred))
    db = (-2/n) * np.sum(P - P_pred)
```

```
    w = w - alpha * dw
    b = b - alpha * db
print("Final model:")
print("w =", w)
print("b =", b)
# Prediction
I_new = 6
P_new = w * I_new + b
print("Predicted power loss at 6 A:", P_new, "W")
```

✅ Expected Result (After Convergence)

$$w \approx 7.4, \quad b \approx 38.6$$

$$\hat{P}(6A) \approx 83\ W$$

✔ Same result as Least Squares, but obtained **iteratively**

1 Mathematical Theory: Linear Regression with Two Features

🔷 Model formulation

For **two input features** x_1, x_2:

$$\hat{y} = \theta_0 + \theta_1 x_1 + \theta_2 x_2$$

Where:

- $\hat{y}$: predicted output
- θ_0: bias (intercept)
- θ_1, θ_2: model parameters
- x_1, x_2: input features

🔷 Vector–matrix form

For m samples:

$$\mathbf{y} = \mathbf{X}\boldsymbol{\theta}$$

$$\mathbf{X} = \begin{bmatrix} 1 & x_{11} & x_{12} \\ 1 & x_{21} & x_{22} \\ \vdots & \vdots & \vdots \\ 1 & x_{m1} & x_{m2} \end{bmatrix}, \quad \boldsymbol{\theta} = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \end{bmatrix}$$

🔷 Cost function (Mean Squared Error)

$$J(\boldsymbol{\theta}) = \frac{1}{2m} \sum_{i=1}^{m} (\hat{y}_i - y_i)^2$$

🔷 Closed-form solution (Normal Equation)

$$\boldsymbol{\theta} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$$

✔ No iterations

✔ Exact solution

✖ Matrix inversion can be costly for large data

2 Electrical Engineering Example (Solved)

🔌 Problem statement

We want to **predict electrical power consumption** of a DC load using:

- x_1: Voltage V (Volts)
- x_2: Current I (Amperes)
- y: Power P (Watts)

Sample	Voltage (V)	Current (A)	Power (W)
1	10	1.0	9.8
2	12	1.2	14.3
3	14	1.5	20.1
4	16	1.7	26.8

🔷 Step 1: Build matrix X

$$\mathbf{X} = \begin{bmatrix} 1 & 10 & 1.0 \\ 1 & 12 & 1.2 \\ 1 & 14 & 1.5 \\ 1 & 16 & 1.7 \end{bmatrix}$$

$$\mathbf{y} = \begin{bmatrix} 9.8 \\ 14.3 \\ 20.1 \\ 26.8 \end{bmatrix}$$

🔷 Step 2: Apply normal equation

$$\boldsymbol{\theta} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$$

Solving gives approximately:

$$\theta_0 = -4.12, \quad \theta_1 = 1.25, \quad \theta_2 = 5.90$$

🔷 Final regression model

$$\boxed{\hat{P} = -4.12 + 1.25V + 5.90I}$$

🔷 Interpretation (very important in EE)

- $1.25V$: Power increases linearly with voltage
- $5.90I$: Current has a **stronger influence**
- Bias term absorbs losses & non-idealities

3️⃣ Python Implementation (From Scratch + Sklearn)

🐍 Method 1: Using NumPy (Normal Equation)

```
import numpy as np
# Data
V = np.array([10, 12, 14, 16])
I = np.array([1.0, 1.2, 1.5, 1.7])
P = np.array([9.8, 14.3, 20.1, 26.8])
# Design matrix
X = np.column_stack((np.ones(len(V)), V, I))
y = P.reshape(-1, 1)
```

```
# Normal equation
theta = np.linalg.inv(X.T @ X) @ X.T @ y
```

```
print("Theta values:")
print(theta)
# Prediction
V_new = 15
I_new = 1.6
P_pred = theta[0] + theta[1]*V_new + theta[2]*I_new

print("Predicted Power:", P_pred.item(), "W")
```

🐍 Method 2: Using Scikit-Learn

```
from sklearn.linear_model import LinearRegression
import numpy as np

# Data
X = np.array([
    [10, 1.0],
    [12, 1.2],
    [14, 1.5],
    [16, 1.7]
])

y = np.array([9.8, 14.3, 20.1, 26.8])

# Model
model = LinearRegression()
model.fit(X, y)
```

```
print("Intercept:", model.intercept_)
print("Coefficients:", model.coef_)

# Prediction
prediction = model.predict([[15, 1.6]])
print("Predicted Power:", prediction[0], "W")
```

🔷 3. Train / Test Split — Mathematical Meaning

📊 Given Dataset

Measured relationship between **current** and **power loss** in a power line:

Index	Current I (A)	Power P (W)
1	1	52
2	2	55
3	3	61
4	4	66
5	5	70

We assume a linear ML model:

$$\hat{P} = wI + b$$

🎯 Objective of Train/Test Split

- **Train set** → learn w and b
- **Test set** → evaluate prediction performance on unseen data

This mimics **real electrical systems**, where models must generalize to new operating conditions.

1️⃣ Train / Test Split (Manual)

We choose:

🔷 Training data (60%)

I	P
1	52
2	55
3	61

🔷 Test data (40%)

I	P
4	66
5	70

2 Mathematical Training (Least Squares on Train Set)

Training values

$$\sum I = 6, \quad \sum P = 168$$

$$\sum I^2 = 14, \quad \sum IP = 345$$

$$n = 3$$

◆ Compute w

$$w = \frac{n\sum IP - \sum I \sum P}{n\sum I^2 - (\sum I)^2}$$

$$w = \frac{3(345) - 6(168)}{3(14) - 6^2} = \frac{1035 - 1008}{42 - 36} = \frac{27}{6} = \boxed{4.5}$$

◆ Compute b

$$\bar{I} = 2, \quad \bar{P} = 56$$

$$b = 56 - 4.5 \times 2 = \boxed{47}$$

✅ Trained Electrical Model

$$\boxed{\hat{P} = 4.5I + 47}$$

3 Testing Phase (Unseen Data)

◆ Predictions

For $I = 4A$:

$$\hat{P} = 4.5(4) + 47 = 65\ W$$

For $I = 5A$:

$$\hat{P} = 4.5(5) + 47 = 69.5\ W$$

◆ Prediction Errors

I	Actual P	Predicted $\hat{P}$	Error
4	66	65	−1
5	70	69.5	−0.5

4 Test Error (Mean Squared Error)

$$MSE = \frac{1}{2}[(66 - 65)^2 + (70 - 69.5)^2]$$

$$MSE = \frac{1}{2}(1 + 0.25) = \boxed{0.625}$$

✔ Small error → **good generalization**

```python
import numpy as np
# Full dataset
I = np.array([1, 2, 3, 4, 5])     # Current (A)
P = np.array([52, 55, 61, 66, 70])  # Power loss (W)
# Train / Test split
I_train = np.array([1, 2, 3])
P_train = np.array([52, 55, 61])
I_test = np.array([4, 5])
P_test = np.array([66, 70])
# Training using least squares
n = len(I_train)
w = (n * np.sum(I_train * P_train) - np.sum(I_train) * np.sum(P_train)) / \
    (n * np.sum(I_train**2) - np.sum(I_train)**2)
b = np.mean(P_train) - w * np.mean(I_train)
print("Trained model:")
print("w =", w)
print("b =", b)
# Testing
P_pred = w * I_test + b
print("Predicted power:", P_pred)
# Mean Squared Error
mse = np.mean((P_test - P_pred)**2)
print("Test MSE:", mse)
```

🔷 4. Classification — K-Nearest Neighbors (KNN)

Problem

In an electrical engineering lab, we have data about different **electrical loads** based on **voltage (V)** and **current (I)** measurements. Each load can be classified as:

- **Resistive (R)** → label 0
- **Inductive (L)** → label 1
- **Capacitive (C)** → label 2

Voltage (V)	Current (A)	Load Type
10	2	0 (R)
12	2.2	0 (R)
5	5	1 (L)
6	5.5	1 (L)
3	8	2 (C)
4	7.5	2 (C)

We measure a **new load** with **Voltage = 7 V** and **Current = 5 A.**

We want to **classify this new load** using **KNN** with **K = 3.**

Step 1: Mathematical Explanation

1. **Distance formula:** For KNN, we use **Euclidean distance** between points:

$$d = \sqrt{(V_{\text{new}} - V_i)^2 + (I_{\text{new}} - I_i)^2}$$

Where (V_i, I_i) are training points.

2. **Compute distance to each training point:**

- Distance to (10, 2):

$$d = \sqrt{(7-10)^2 + (5-2)^2} = \sqrt{(-3)^2 + 3^2} = \sqrt{9+9} = \sqrt{18} \approx 4.243$$

- Distance to (12, 2):

$$d = \sqrt{(7-12)^2 + (5-2.2)^2} = \sqrt{(-5)^2 + 2.8^2} = \sqrt{25 + 7.84} = \sqrt{32.84} \approx 5.73$$

- Distance to (5, 5):

$$d = \sqrt{(7-5)^2 + (5-5)^2} = \sqrt{2^2 + 0^2} = \sqrt{4} = 2$$

- Distance to (6, 5.5):

$$d = \sqrt{(7-6)^2 + (5-5.5)^2} = \sqrt{1^2 + (-0.5)^2} = \sqrt{1 + 0.25} = \sqrt{1.25} \approx 1.118$$

- Distance to (3, 8):

$$d = \sqrt{(7-3)^2 + (5-8)^2} = \sqrt{4^2 + (-3)^2} = \sqrt{16+9} = \sqrt{25} = 5$$

- Distance to (4, 7.5):

$$d = \sqrt{(7-4)^2 + (5-7.5)^2} = \sqrt{3^2 + (-2.5)^2} = \sqrt{9 + 6.25} = \sqrt{15.25} \approx 3.905$$

Step 2: Identify K nearest neighbors

K = 3, so select **3 smallest distances:**

1. 1.118 → (6, 5.5) → Load = 1 (L)
2. 2 → (5, 5) → Load = 1 (L)
3. 3.905 → (4, 7.5) → Load = 2 (C)

Step 3: Majority Voting

- Load 1 (L) → 2 neighbors
- Load 2 (C) → 1 neighbor

Prediction: Load Type = Inductive (L, label 1)

✅ **Mathematical solution complete.**

Step 4: Python Implementation

```python
import numpy as np
from sklearn.neighbors import KNeighborsClassifier
# Training data: [Voltage, Current]
X_train = np.array([
    [10, 2],
    [12, 2.2],
    [5, 5],
    [6, 5.5],
    [3, 8],
    [4, 7.5]
])
# Labels: 0=R, 1=L, 2=C
y_train = np.array([0, 0, 1, 1, 2, 2])
# New measurement
X_test = np.array([[7, 5]])
# KNN classifier
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train, y_train)
# Prediction
predicted_label = knn.predict(X_test)
load_types = {0: "Resistive", 1: "Inductive", 2: "Capacitive"}
print(f"Predicted Load Type: {load_types[predicted_label[0]]}")
```

Output:

```pgsql
Predicted Load Type: Inductive
```

🔷 5. K-Means Clustering — Mathematics

Example Context: Electrical Engineering

Problem:

Suppose we have a **set of electrical devices** in a smart grid, and we have measured their **power consumption (kW) and voltage (V)**. We want to **cluster these devices into groups** with similar characteristics using **K-Means**.

Step 1: Sample Data

Let's say we have 8 devices with the following measurements:

Device	Power (kW)	Voltage (V)
D1	1.2	220
D2	1.5	230
D3	0.8	210
D4	5.0	400
D5	4.8	390
D6	5.2	410
D7	0.9	215
D8	1.3	225

We want to **cluster these devices into 2 clusters** (K=2). Likely clusters: low-power devices vs high-power devices.

Step 2: Mathematical K-Means Procedure

1. **Initialization:** Pick 2 initial centroids (randomly, for example D1 and D4):

$$\mathbf{C}_1 = (1.2, 220), \quad \mathbf{C}_2 = (5.0, 400)$$

2. **Distance Calculation:** Use Euclidean distance:

$$d(\mathbf{x}, \mathbf{C}) = \sqrt{(P - P_c)^2 + (V - V_c)^2}$$

Where P = Power, V = Voltage.

3. **Assign Clusters:** Assign each device to the nearest centroid.

Step 2.1: Distances to C1 and C2

For D2:

$$d(D2, C1) = \sqrt{(1.5 - 1.2)^2 + (230 - 220)^2} = \sqrt{0.09 + 100} = \sqrt{100.09} \approx 10.005$$

$$d(D2, C2) = \sqrt{(1.5 - 5)^2 + (230 - 400)^2} = \sqrt{12.25 + 28900} = \sqrt{28912.25} \approx 170.06$$

→ D2 → Cluster 1

Repeat for all devices:

Device	Cluster
D1	1
D2	1
D3	1
D4	2
D5	2
D6	2
D7	1
D8	1

4. **Update Centroids:** Compute new centroid as **mean of cluster points.**

$$\mathbf{C}_1 = \left(\frac{1.2 + 1.5 + 0.8 + 0.9 + 1.3}{5}, \frac{220 + 230 + 210 + 215 + 225}{5}\right) = (1.14, 220)$$

$$\mathbf{C}_2 = \left(\frac{5.0 + 4.8 + 5.2}{3}, \frac{400 + 390 + 410}{3}\right) = (5.0, 400)$$

Notice **C2 stayed the same**, C1 slightly updated.

5. **Repeat Assignment:** Recalculate distances → cluster assignments.

   After convergence (no changes in cluster), we have:

- **Cluster 1 (Low Power Devices):** D1, D2, D3, D7, D8
- **Cluster 2 (High Power Devices):** D4, D5, D6

Step 3: Python Implementation

```
# Import libraries
import numpy as np
from sklearn.cluster import KMeans

# Device data: Power (kW), Voltage (V)
X = np.array([
    [1.2, 220],
    [1.5, 230],
    [0.8, 210],
    [5.0, 400],
    [4.8, 390],
    [5.2, 410],
    [0.9, 215],
    [1.3, 225]
])
```

```
# Create KMeans model with 2 clusters
kmeans = KMeans(n_clusters=2, random_state=0)
kmeans.fit(X)
# Cluster labels
labels = kmeans.labels_
centroids = kmeans.cluster_centers_
# Output results
for i, label in enumerate(labels):
    print(f"Device D{i+1} → Cluster {label+1}")
print("\nCentroids:")
print(centroids)
```

Expected Output:

```
Device D1 → Cluster 1
Device D2 → Cluster 1
Device D3 → Cluster 1
Device D4 → Cluster 2
Device D5 → Cluster 2
Device D6 → Cluster 2
Device D7 → Cluster 1
Device D8 → Cluster 1
```

```
Centroids:
[[ 1.14 220. ]
 [ 5.   400. ]]
```

Step 4: Interpretation

- **Cluster 1:** Low-power devices (residential loads).
- **Cluster 2:** High-power devices (industrial loads).
- K-Means helped automatically group devices based on power & voltage, which can be useful for **load management, fault detection, or demand response** in electrical grids.