**-3**

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LinearRegression

from sklearn.metrics import mean\_squared\_error,r2\_score

df=pd.read\_csv("Salary\_dataset.csv")

X=df[["YearsExperience"]]

y=df[["Salary"]]

X\_train,X\_test,y\_train,y\_test=train\_test\_split(X,y,*test\_size*=0.2,*random\_state*=42)

model=LinearRegression()

model.fit(X\_train,y\_train)

y\_pred=model.predict(X\_test)

mse=mean\_squared\_error(y\_test,y\_pred)

print("mse:",mse)

plt.scatter(X,y,*color*='blue',*label*='Actual')

plt.title("Linear Regression")

plt.xlabel("Year of Experience")

plt.ylabel("Salary")

plt.show()

**-4**

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import accuracy\_score, confusion\_matrix, classification\_report

# Load and preprocess

df = pd.read\_csv('MultipleFiles/Social\_Network\_Ads.csv')

X = df[['Age', 'EstimatedSalary']]

y = df['Purchased']

# Split and scale

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

scaler = StandardScaler()

X\_train = scaler.fit\_transform(X\_train)

X\_test = scaler.transform(X\_test)

# Train and predict

model = LogisticRegression()

model.fit(X\_train, y\_train)

y\_pred = model.predict(X\_test)

# Evaluation

print("Accuracy:", accuracy\_score(y\_test, y\_pred))

print("Confusion Matrix:\n", confusion\_matrix(y\_test, y\_pred))

print("Report:\n", classification\_report(y\_test, y\_pred))

# Decision boundary plot

def plot\_boundary(X, y, model):

x\_min, x\_max = X[:, 0].min()-1, X[:, 0].max()+1

y\_min, y\_max = X[:, 1].min()-1, X[:, 1].max()+1

xx, yy = np.meshgrid(np.arange(x\_min, x\_max, 0.01),

np.arange(y\_min, y\_max, 0.01))

grid = np.c\_[xx.ravel(), yy.ravel()]

zz = model.predict(grid).reshape(xx.shape)

plt.figure(figsize=(8,6))

plt.contourf(xx, yy, zz, alpha=0.3, cmap=plt.cm.Paired)

sns.scatterplot(x=X[:, 0], y=X[:, 1], hue=y, palette='coolwarm', edgecolor='k')

plt.xlabel('Age (scaled)')

plt.ylabel('Estimated Salary (scaled)')

plt.title('Logistic Regression Decision Boundary')

plt.show()

# Call the function

plot\_boundary(X\_test, y\_test, model)

**5**

**import pandas as pd**

**import matplotlib.pyplot as plt**

**import seaborn as sns**

**from sklearn.model\_selection import train\_test\_split**

**from sklearn.svm import SVC**

**from sklearn.preprocessing import LabelEncoder**

**from sklearn.metrics import classification\_report, confusion\_matrix**

**import numpy as np**

**# Load the dataset**

**df = pd.read\_csv('IRIS.csv')**

**# Encode species labels**

**le = LabelEncoder()**

**df['species\_encoded'] = le.fit\_transform(df['species'])**

**# Define features and target**

**X = df.drop(['species', 'species\_encoded'], axis=1)**

**y = df['species\_encoded']**

**# Train/test split**

**X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)**

**# Train SVM**

**svm\_model = SVC(kernel='linear')**

**svm\_model.fit(X\_train, y\_train)**

**# Predict**

**y\_pred = svm\_model.predict(X\_test)**

**# Classification report**

**print("Classification Report:\n")**

**print(classification\_report(y\_test, y\_pred, target\_names=le.classes\_))**

**# Confusion matrix visualization**

**conf\_matrix = confusion\_matrix(y\_test, y\_pred)**

**plt.figure(figsize=(6, 4))**

**sns.heatmap(conf\_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=le.classes\_, yticklabels=le.classes\_)**

**plt.title("Confusion Matrix")**

**plt.xlabel("Predicted")**

**plt.ylabel("Actual")**

**plt.tight\_layout()**

**plt.show()**

**# ======================**

**# 2D Decision Boundary**

**# ======================**

**# Select 2 features for plotting**

**X\_2d = df[['sepal\_length', 'sepal\_width']]**

**y\_2d = df['species\_encoded']**

**# Fit SVM with 2D data**

**svm\_2d = SVC(kernel='linear')**

**svm\_2d.fit(X\_2d, y\_2d)**

**# Create meshgrid for plotting**

**h = 0.02**

**x\_min, x\_max = X\_2d.iloc[:, 0].min() - 1, X\_2d.iloc[:, 0].max() + 1**

**y\_min, y\_max = X\_2d.iloc[:, 1].min() - 1, X\_2d.iloc[:, 1].max() + 1**

**xx, yy = np.meshgrid(np.arange(x\_min, x\_max, h),**

**np.arange(y\_min, y\_max, h))**

**# Predict on mesh**

**Z = svm\_2d.predict(np.c\_[xx.ravel(), yy.ravel()])**

**Z = Z.reshape(xx.shape)**

**# Plot decision boundary**

**plt.figure(figsize=(8, 6))**

**plt.contourf(xx, yy, Z, cmap=plt.cm.coolwarm, alpha=0.8)**

**sns.scatterplot(data=df, x='sepal\_length', y='sepal\_width', hue='species', palette='deep', edgecolor='k')**

**plt.title('SVM Decision Boundary (sepal length vs sepal width)')**

**plt.xlabel('Sepal Length')**

**plt.ylabel('Sepal Width')**

**plt.show()**

**-6**

import numpy as np

*class* HebbianNetwork:

*def* \_\_init\_\_(*self*, *input\_size*, *output\_size*, *learning\_rate*=0.01):

*self*.weights = np.random.rand(*input\_size*, *output\_size*)

*self*.learning\_rate = *learning\_rate*

*def* train(*self*, *inputs*, *outputs*):

        for i in range(len(*inputs*)):

            input\_vector = *inputs*[i]

            output\_vector = *outputs*[i]

*self*.weights += *self*.learning\_rate \* np.outer(input\_vector, output\_vector)

*def* predict(*self*, *input\_vector*):

        return np.dot(*input\_vector*, *self*.weights)

# Example usage

# Define input and output patterns

inputs = np.array([[1, 0], [0, 1], [1, 1], [0, 0]])

outputs = np.array([[1, 0], [0, 1], [1, 1], [0, 0]])

# Initialize the Hebbian network

network = HebbianNetwork(*input\_size*=2, *output\_size*=2)

# Train the network

network.train(inputs, outputs)

# Test the network

test\_input = np.array([1, 0])

predicted\_output = network.predict(test\_input)

print(*f*"Predicted output for input {test\_input}: {predicted\_output}")

**7**

import numpy as np

import matplotlib.pyplot as plt

from scipy.stats import norm, gaussian\_kde

# Generate data

X = np.concatenate([

np.random.normal(2, 1, 200),

np.random.normal(-1, 0.8, 600)

])

# Initialize parameters

mu1, mu2 = 1, 0

sigma1, sigma2 = 1, 1

pi1, pi2 = 0.5, 0.5

log\_likelihoods = []

# EM Algorithm

for \_ in range(20):

# E-step

r1 = pi1 \* norm.pdf(X, mu1, sigma1)

r2 = pi2 \* norm.pdf(X, mu2, sigma2)

gamma1 = r1 / (r1 + r2)

gamma2 = 1 - gamma1

# M-step

N1, N2 = gamma1.sum(), gamma2.sum()

mu1 = np.sum(gamma1 \* X) / N1

mu2 = np.sum(gamma2 \* X) / N2

sigma1 = np.sqrt(np.sum(gamma1 \* (X - mu1)\*\*2) / N1)

sigma2 = np.sqrt(np.sum(gamma2 \* (X - mu2)\*\*2) / N2)

pi1, pi2 = N1 / len(X), N2 / len(X)

# Log-likelihood

log\_likelihood = np.sum(np.log(

pi1 \* norm.pdf(X, mu1, sigma1) + pi2 \* norm.pdf(X, mu2, sigma2)

))

log\_likelihoods.append(log\_likelihood)

# Log-likelihood plot

plt.plot(log\_likelihoods)

plt.title("Log-Likelihood over Epochs")

plt.xlabel("Epoch")

plt.ylabel("Log-Likelihood")

plt.grid(True)

plt.show()

# Final density plot

X\_sorted = np.sort(X)

mix\_density = pi1 \* norm.pdf(X\_sorted, mu1, sigma1) + pi2 \* norm.pdf(X\_sorted, mu2, sigma2)

plt.plot(X\_sorted, gaussian\_kde(X)(X\_sorted), label="KDE", color="green")

plt.plot(X\_sorted, mix\_density, label="GMM", color="red")

plt.title("Density Estimation")

plt.xlabel("X")

plt.ylabel("Density")

plt.legend()

plt.grid(True)

plt.show()

**8**

import numpy as np

class McCullochPittsNeuron:

def \_\_init\_\_(self, weights, threshold):

self.weights = weights

self.threshold = threshold

def activate(self, inputs):

summation = np.dot(inputs, self.weights)

return 1 if summation >= self.threshold else 0

# OR Gate

or\_weights = np.array([1, 1])

or\_threshold = 1

or\_neuron = McCullochPittsNeuron(or\_weights, or\_threshold)

print("OR Gate:")

or\_inputs = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])

for input\_pair in or\_inputs:

print(f"Input: {input\_pair}, Output: {or\_neuron.activate(input\_pair)}")

# AND Gate

and\_weights = np.array([1, 1])

and\_threshold = 2

and\_neuron = McCullochPittsNeuron(and\_weights, and\_threshold)

print("\nAND Gate:")

and\_inputs = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])

for input\_pair in and\_inputs:

pri**nt(f"Input: {input\_pair}, Output: {and\_neuron.activate(input\_pair)}")**

**9**

**import numpy as np**

**class Perceptron:**

**def \_\_init\_\_(self, input\_size, lr=0.1, epochs=10):**

**self.weights = np.zeros(input\_size + 1) # +1 for bias**

**self.lr = lr**

**self.epochs = epochs**

**def activation(self, x):**

**return 1 if x >= 0 else 0**

**def predict(self, x):**

**x = np.insert(x, 0, 1) # Add bias input**

**weighted\_sum = np.dot(self.weights, x)**

**return self.activation(weighted\_sum)**

**def train(self, X, y):**

**for \_ in range(self.epochs):**

**for xi, target in zip(X, y):**

**xi = np.insert(xi, 0, 1) # Add bias input**

**prediction = self.activation(np.dot(self.weights, xi))**

**self.weights += self.lr \* (target - prediction) \* xi**

**# OR gate dataset**

**X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])**

**y = np.array([0, 1, 1, 1])**

**# Train the perceptron**

**model = Perceptron(input\_size=2)**

**model.train(X, y)**

**# Test predictions**

**print("Testing OR gate:")**

**for xi in X:**

**print(f"Input: {xi}, Predicted: {model.predict(xi)}")**

**10**

**import numpy as np**

**import matplotlib.pyplot as plt**

**from mpl\_toolkits.mplot3d import Axes3D**

**# Generate 3D dataset**

**np.random.seed(42)**

**X = np.random.multivariate\_normal(mean=[5, 10, 15], cov=[[3, 2, 1], [2, 3, 2], [1, 2, 3]],**

**size=100)**

**# Standardize data (centering)**

**X\_centered = X - np.mean(X, axis=0)**

**# Compute covariance matrix & eigen decomposition**

**cov\_matrix = np.cov(X\_centered.T)**

**eigenvalues, eigenvectors = np.linalg.eig(cov\_matrix)**

**# Sort eigenvectors by largest eigenvalues**

**sorted\_indices = np.argsort(eigenvalues)[::-1]**

**top\_2\_eigenvectors = eigenvectors[:, sorted\_indices[:2]]**

**# Transform data to 2D space**

**X\_pca = X\_centered @ top\_2\_eigenvectors**

**# Visualization**

**fig = plt.figure(figsize=(12, 5))**

**# 3D Scatter Plot**

**ax1 = fig.add\_subplot(121, projection='3d')**

**ax1.scatter(X[:, 0], X[:, 1], X[:, 2], c='blue', alpha=0.5)**

**ax1.set\_title("Original 3D Data")**

**ax1.set\_xlabel("Feature 1")**

**ax1.set\_ylabel("Feature 2")**

**ax1.set\_zlabel("Feature 3")**

**# 2D PCA Scatter Plot**

**ax2 = fig.add\_subplot(122)**

**ax2.scatter(X\_pca[:, 0], X\_pca[:, 1], c='red', alpha=0.5)**

**ax2.set\_title("PCA Reduced Data (2D)")**

**ax2.set\_xlabel("PC1")**

**ax2.set\_ylabel("PC2")**

**plt.show()**

**# Explained variance**

**explained\_variance = eigenvalues / np.sum(eigenvalues)**

**print("Explained Variance Ratios:", explained\_variance)**