24/03/25 Lab-03 __/__/____

1) Linear Regression:

→ import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.axes as ax
from matplotlib.animation import FuncAnimation

→ url = " https ---- "
data = pd.read_csv(url)
data

data = data.dropna()

train_input = np.array(data.x[0:500]).reshape(500,1)
train_output = np.array(data.y[0:500]).reshape(500,1)

test_input = np.array(data.x[500:700]).reshape(199,1)
test_output = np.array(data.y[500:700]).reshape(199,1)

→ class LinearRegression:
def __init__(self):
self.parameters = {}

def forward_propogation(self, train_input):
m = self.parameters['m']
c = self.parameters['c']
predictions = np.multiply(m, train_input) + c

return predictions

```
def cost_function(self, predictions, train_output):
    cost = np.mean(-train_output - predictions)**2)

    return cost

def backword_propogation(self, train_input,
train_output, predictions):
    derivatives = {}
    df = (predictions - train_output)
    dm = 2 * np.mean(np.multiply(train_input,
                                 df))

    dc = 2 * np.mean(df)
    derivatives['dm'] = dm
    derivatives['dc'] = dc
    return derivatives

def update_parameters(self, derivatives, learning_rate
    self.parameters['m'] = self.parameters['m'] -
learning_rate * derivatives['dm']
    self.parameters['c'] = self.parameters['c'] -
learning_rate * derivatives['dc']

def train(self, train_input, train_output,
learning_rate, iters):
    self.parameters['m'] = np.random.uniform
(0,1) * -1
    self.parameters['c'] = np.random.uniform
```

__/__/____

```
ani = FuncAnimation(fig, update, frames=iter,
interval=200, blit=True)

ani.save("linear_regression_A.gif", writer='ffmp

plt.xlabel('Input')
plt.ylabel('Output')
plt.title('Linear Regression')
plt.legend()
plt.show()

return self.parameters, self.los
```

output →

Iteration 1, Loss = 1228.859017670112
Iteration 2, Loss = 10.40095 2157878854
Iteration 3, Loss = 8.132130341952354

__/__/____

8) Multiple linear Regression:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

data = {
"Feature1": [1,2,3,4,5,6,7,8,9,10],
"Feature2": [2,3,5,7,11,13,17,19,23,29],
"Feature3": [3,6,9,12,15,18,21,24,27,30],
"Target": [5,9,15,22,31,41,53,66,80,96]
}
df = pd.DataFrame(data)

X = df.drop(columns=["Target"]).values
y = df["Target"].values.reshape(-1,1)

X = np.linalg.solve
X = np.hstack((np.ones((X.shape[0],1)), X))

beta = np.linalg.solve(X.T @ X + 0.01 * np.identity
(X.shape[1]), X.T @ y)

Y_pred = X @ beta

mse = np.mean((y - y_pred)**2)
total_variance = np.sum((y - np.mean(y))**2)
explained_variance = np.sum((y_pred - np.mean(y)
**2)
```

r2 = explained_variance / total_variance

```
print("Model coefficients:", beta[1:].flatten())
print("Intercept:", beta[0][0])
print("Mean Squared Error:", mse)
print("R-Squared score:", r2
```

```
plt.scatter(y, y_pred, color='blue')
plt.plot(y, y, color='red', linestyle='--')
plt.xlabel("Actual values")
plt.ylabel("predicted values")
plt.title("Actual vs predicted values")
plt.show()
```

3 → Logistic Regression

```
import numpy as np
import matplotlib.pyplot as plt

def sigmoid(z):
    return 1/(1+np.exp(-z))

def compute_cost(X, y, theta):
    m = len(y)
    h = sigmoid(X @ theta)
    cost = (-1/m) * np.sum(y * np.log(h) +
                           (1-y) * np.log(1-h))
    return cost
```

__/__/____

```
def gradient_descent(X, y, theta, alpha,
iterations):
    m = len(y)
    cost_history = []

    for _ in range(iterations):
        gradient = (1/m) * X.T @ (sigmoid(X
                   @ theta) - y)
        theta -= appha * gradient
        cost_history.append(compute_cost(X,
            y, theta))
    return theta, cost_history

def predict(X, theta):
    return (sigmoid(X @ theta) >= 0.5).astype(int)

np.random.seed(42)
X = np.random.rand(100, 1) * 10
y = (X > 5).astype(int).ravel()

X_b = np.c_[np.ones((X.shape[0], 1)), X]
theta = np.zeros(X_b.shape[1])
alpha = 0.1
iteration = 1000

theta, cost_history = gradient_descent(X_b, y,
    theta, alpha, iteration)

y_pred = predict(X_b, theta)
```

```
accuracy = np.mean(y_pred == y)
print("Accuracy: ", accuracy: .2f(y))
plt.scatter(x, y, color='blue', label='Actual Data')

plt.scatter(X, y_pred, colur='red', marker='x',
    label='predicted labels')

plt.xlabel("Feature")
plt.ylabel("Class (0 or 1)")
plt.legend()
plt.show()
```

output:

Accuracy: 0.97.

Sohail B
24/3/25