→ Building Infrastructure

→ Optimizing ML algorithm

→ Integration.

LINEAR REGRESSION WITH NUMPY & PYTHON

Build → Univariate Linear Regression.

TASK 1&2 Import Libraries and load Data

G to PLOT → Graph Gallery

Univariate ⇔ Linear Regression with one variable.

Data → Bike sharing data to decide which city to expand in.

Read Data.

data = pd.read_csv('name')

first five → data.head()

data.info ← To get more insight.

TASK-3 VISUALISE THE DATA

We will use scatter plot as only 2 variables.

y axis for 'to predict'

ax = sns.scatterplot(x = "Population", y = "Profit", data = data)

ax.set_title("Profit in $10000 vs City Population")

We will find a line of best fit. (Linear Regression)

TASK-4 Compute Cost J(θ)

Our Linear Regression powered by Gradient Descent.

Difference btw levels predicted

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2$$

$J(\theta)$: Cost / Error

$\frac{1}{2m}$: normalisation of data.

$h_\theta(x^{(i)})$: Estimate Profit on x

$y^{(i)}$: training example Profit

2 is mathematical convenience & will cancel out when we compute.

m is length of data

h_θ is the hypothesis.

$h_\theta(x) = \theta^T x = \theta_0 + \theta_1 x_1$

Implementing Cost function

```
def cost_function(X, y, theta):
    m = len(y)                                  # dot of X & theta
    y_pred = X.dot(theta)                       # function name; Dot product of θ and x.
    error = (y_pred - y)**2
    return 1/(2*m) * np.sum(error)
```

`m = data.Population.values.size` — Total data ka size

X can be population but we need to add another dimension to accomodate the θ_0 intercept term. So, θ_0 is another feature of X.

```
X = np.append(np.ones((m,1)), data.Population.values.reshape(m,1), axis=1)
```

- `np.ones((m,1))` → first row of all 1
- redefine the data with m rows & 1 column
- `axis=1` → changing to columns

```
y = data.Profit.values.reshape(m,1)
theta = np.zeros((2,1))
```

All zeroes

(2,1) → 2 rows, 1 column

We added an extra dimension to our input or feature matrix (x) to accomodate the intercept term
Explanation and sets it to all ones QUIZ QUESTION

$h_\theta(x)$ = predicted value of y at x.

$y = mx + c$ → Eq. of line.

$h_\theta(x) = \theta_1 x_1 + \theta_0$ ← Intercept term i.e where does line intersect y-axis.

So, I need to find values of θ_1 & θ_0

That's why My Initial theta is all zero to begin with. And my theta only have 2 rows & 1 columns i.e 2 boxes one represents θ_0 and other θ_1

θ_0	0
θ_1	0

Case I normal X . Θ.

Case II appended X . Θ.

Case I $x \cdot \theta_0 + x \cdot \theta_1$

wrong equation

Case II $x_0 \theta_0 + x_1 \theta_1$

we have appended all 1 → actual population values.

Overall Equ → $\theta_0 + x_1 \theta_1$, which is exactly what we want and that's why we appended and changed X

__/__/____

~~BASIC IMAGE CLASSIFICATION (TENSORFLOW)~~

TASK-5 GRADIENT DESCENT

Reducing the errors / cost function.

Formula Used → Batch Gradient Descent.

learning rate.

```
def gradient_descent(x, y, theta, alpha, iterations):
    m = len(y)
    costs = []          # to track all cost history to see how errors changed.
    for i in range(iterations):
        y_pred = x.dot(theta)
        error = np.dot(x.transpose().(y_pred - y))
        theta = theta - alpha * 1/m * error
        costs.append(cost_function(x, y, theta))
    return theta, costs
```

+ ki dimension → (m, 2)
use transpose kiya → (2, m)
multiply (2, m) x (m, 1)
result → (2, 1) theta
jo hamara theta tha.

iska dimension hai (m, 1)

```
theta, costs = gradient_descent(x, y, theta, alpha=0.01, iterations=2000)
print("h(x) = {} + {}x1".format(sls.round(theta[0,0],2), sls.round(theta[1,0],2)))
```

intercept term

sls.round(theta[1,0],2)

TASK-6 Visualise the cost function

```
from mpl_toolkits.mplot3d import Axes3D
theta_0 = np.linspace(-10,10,100)
```

creates a list of uniform numbers from to How many

```
theta_1 = np.linspace(-1,4,100)
cost_values = np.zeroes((len(theta_0), len(theta_1)))
for i in range(len(theta_0)):
    for j in range(len(theta_1)):
        t = np.array([theta_0[i], theta_1[j]])
        cost_values[i,j] = cost_function(x,y,theta)

fig = plt.figure(figsize=(12,8))
```

Size of graph.

```
ax = fig.gca(projection='3d')
```

3d to 2D plotting

```
surf = ax.plot_surface(theta_0, theta_1, cost_values,
                       cmap='viridis')
```

colour map

diversion colour map.

```
fig.colorbar(surf, shrink=0.5, aspect=5)
plt.xlabel("$\Theta_0$")
plt.ylabel("$\Theta_1$")
ax.set_zlabel("$J(\Theta)$")
```

cost function

__/__/____

ax.view_init (30, 330) Rotating for better view.

TASK-7 PLOTTING THE CONVERGENCE

Task ⇒ Plot the costs found above.

```
plt.plot(costs)
plt.xlabel("Iterations")
plt.ylabel("$J(\Theta)$")
plt.title("Values of the Cost function over Iteration of Gradient Descent")
```

TASK-8 Training Data with Linear Regression fit.

Important
If is a problem we mostly forget But Hard to Debug

Theta.shape → (2,1)

Theta → array([[-3.7], [1.182]])

Theta = np.squeeze(Theta) → Remove extra dim.

Range of values

x_value = [x for x in range(5,25)]

y_value = [(x * Theta[1] + Theta[0]) for x in x_value]

The equation i.e $y = h_\theta(x) = \theta^T x = \theta_0 + \theta_1 x$ | loop for all values of x

__/__/____

Display line

```
sns.lineplot (x-value, y-value)
plt.xlabel ("name")
plt.ylabel ("name")
```

TASK-9 Inference using optimised θ value or using line to make predictions.

```
def predict [x, theta]
    y-pred = np.dot (theta.transpose(), x)
    return y-pred.
```

Our first value of y to predict profit. (1 because θ_0 was multiplied by 1. Actual x.)

```
y-pred-1 = predict (np.array([1, 4]) theta) * 10000
print ("The estimated profit is" y-pred-1)

y-predpred-2 = predict (np.array([1, 8.3]) theta) * 10000
```