Name- Jaden Borges

Roll no.- 08

Experiment 2- Linear Regression: Parameter Estimation using OLS, MLE, and Gradient Descent.

Batch-1

28-1-2026

LR Using OLS

Step 1 - Imports

```
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
```

Step 2 - Input Data

```
x = np.array([10,20,30,50]).reshape(-1,1) # creating 1 column n decide no of rows
y = np.array([12,21,29,48])
```

Step 3 - Create Model And Fit Data

```
model = LinearRegression()
model.fit(x,y)
```

```
LinearRegression
LinearRegression()
```

Step 4 - Get Coeffiients And Print

```
w = model.coef_[0]
b = model.intercept_
print(f"The Slope is: {w}")
print(f"The Intercept is: {b}")
```

```
The Slope is: 0.8971428571428574
The Intercept is: 2.828571428571422
```

Step 5 - Make Predictions

```
y_pred = model.predict(x)
print(f"Prediction for training data: {y_pred}")
for xi,yi,ypi in zip(x.flatten(), y, y_pred):
    print(f"x: {xi}, Actual y: {yi}, Predicted y: {ypi}")
```

```
Prediction for training data: [11.8        20.77142857 29.74285714 47.68571429]
x: 10, Actual y: 12, Predicted y: 11.799999999999995
x: 20, Actual y: 21, Predicted y: 20.77142857142857
x: 30, Actual y: 29, Predicted y: 29.742857142857144
x: 50, Actual y: 48, Predicted y: 47.68571428571429
```

Step 6 - Error Calculation

```
mse = mean_squared_error(y, y_pred)
r2 = r2_score(y, y_pred)
print(f"Mean Squared Error: {mse}")
print(f"R2-score: {r2}")
```

```
Mean Squared Error: 0.1857142857142861
R2-score: 0.9989463019250253
```

LR Using MLE

Step 1 - Imports

```
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
from scipy.optimize import minimize
```

Step 2 - Input Data

```
x = np.array([10,20,30,50])
y = np.array([12,21,29,48])
```

Step 3 - Negative Log Likelihood Function

```
def neg_log_likelihood(params):
    w, b = params
    sigma2 = 1  # assume variance = 1
    y_pred = w * x + b
    nll = 0.5*np.sum((y - y_pred)**2 / sigma2 )
    return nll
```

```
#Initial Value for w and b
initial_guess = [0,0]
```

Step 4 - Minimize Negative Log Likelihood

```
result = minimize(neg_log_likelihood, initial_guess)
w_mle,b_mle = result.x
print(f"The MLE for slope is: {w_mle}")
print(f"The MLE for intercept is: {b_mle}")
```

```
The MLE for slope is: 0.8971428246616441
The MLE for intercept is: 2.828572315886469
```

Step 5 - Prediction

```
y_pred = w_mle * x + b_mle
print(f"Prediction for training data:")
for xi,yi,ypi in zip(x.flatten(), y, y_pred):
    print(f"x: {xi}, Actual y: {yi}, Predicted y: {ypi}")
```

```
Prediction for training data:
x: 10, Actual y: 12, Predicted y: 11.800000562502909
x: 20, Actual y: 21, Predicted y: 20.77142880911935
x: 30, Actual y: 29, Predicted y: 29.74285705573579
x: 50, Actual y: 48, Predicted y: 47.68571354896867
```

Step 6 - Error Calculation

```
mse = mean_squared_error(y, y_pred)
r2 = r2_score(y, y_pred)
print(f"Mean Squared Error: {mse}")
print(f"R2-score: {r2}")
```

```
Mean Squared Error: 0.18571428571451654
R2-score: 0.998946301925024
```

LR Using GD

Step 1 - Imports

```
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
```

Step 2 - Input Data

```
x = np.array([10,20,30,50])
y = np.array([12,21,29,48])
```

Step 3 - Initialization

```
w,b = 0,0
alpha = 0.001
n_iter = 30000
n = len(x)
```

```
for i in range(n_iter):
  y_pred = w * x.flatten() + b
  dw = (-2/n) * np.sum(x.flatten() * (y - y_pred))
  db = (-2/n) * np.sum(y - y_pred)
  w -= alpha * dw
  b -= alpha * db
print(f"The Slope is: {w}")
print(f"The Bias is: {b}")
```

```
The Slope is: 0.8971429705949376
The Bias is: 2.8285674071043445
```

LR Using GD With Single Parameter

Step 1 - Imports

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

Step 2 - Input Data

```
x = np.array([1,2,3])
y = np.array([2,3,5])
n = len(x)
```

Step 3 - Calculate Loss Function

```
def loss(w1):
  w0 = np.mean(y) - w1 * np.mean(x)
  y_pred = w0 + w1 * x
  return np.sum((y - y_pred)**2)
```

Step 4 - Calulate Gradient Of J wrt w1

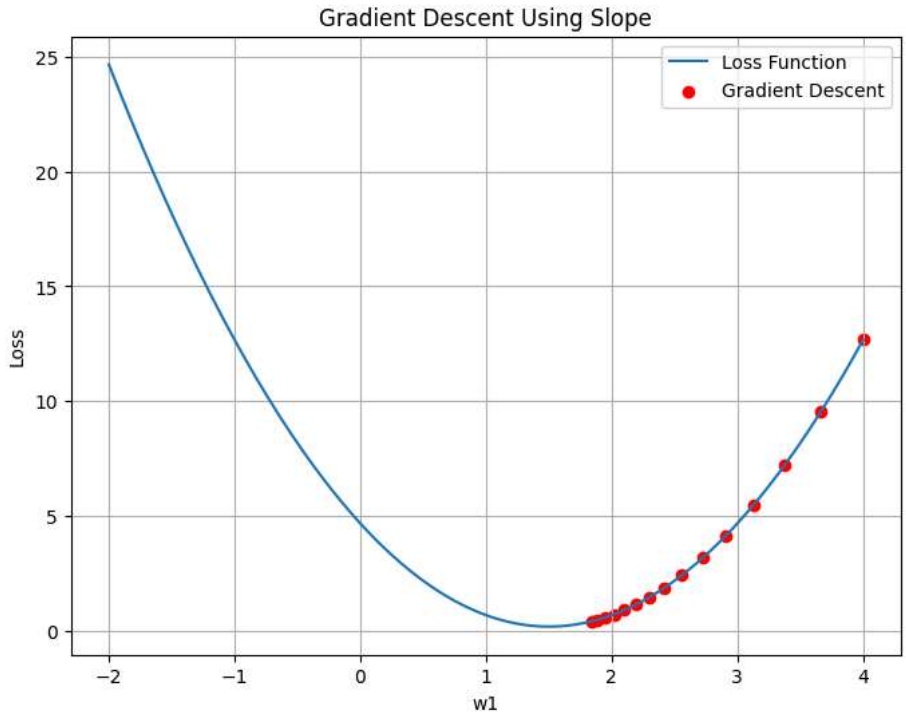
```
def gradient(w1):
  w0 = np.mean(y) - w1 * np.mean(x)
  y_pred = w0 + w1 * x
  return -2/n *np.sum(x*(y-y_pred))
```

Step 5 - Gradient Descent

```
lr = 0.1
wl = 4
iter = 15
w1_values = []
loss_values = []

for i in range(iter):
    w1_values.append(wl)
    loss_values.append(loss(wl))
    grad = gradient(wl)
    wl -= lr * grad
```

Step 6 - Plot Loss Function And GD

```
w_space = np.linspace(-2,4,200)
loss_space = [loss(w) for w in w_space]

plt.figure(figsize=(8,6))
plt.plot(w_space, loss_space, label='Loss Function')

plt.scatter(w1_values, loss_values, color='red', label='Gradient Descent')
plt.xlabel('w1')
plt.ylabel('Loss')
plt.title('Gradient Descent Using Slope')
plt.legend()
plt.grid(True)
plt.show()
```



Start coding or generate with AI.