<u>IDC410</u>

REPORT

<u>Implement Logistic Regression</u>

Assignment 2

Name: **Fayiz M**

Registration Number: **MS21078**

Aim;

- To investigate how different values of n and θ impact the ability for your logistic regression function to learn the coefficients, β, used to generate the output vector Y.
- To derive the derivation of the partial derivative of the cost function with respect to the parameters of the model.
- To perform some visualization

Theory;

Logistic regression is a type of statistical model often used for classification and predictive analytics. It estimates the probability of an event occurring, such as voting or not voting, based on a given dataset of independent variables. (Source: IBM)

Gradient descent is a common iterative optimization method utilized to minimize the cost function in machine learning, particularly in logistic regression models. This algorithm involves iteratively adjusting weights based on the gradient of the cost function. Beginning with random weight initialization, the algorithm calculates the gradient and updates the weights in the opposite direction of this gradient. This iterative process persists until the algorithm converges to the minimum cost value.

Methods and Code Snippets;

- Function to generate an m+1 dimensional data set, of size n, consisting of m continuous independent variables (X) and one dependent variable (Y)

```
import numpy as np

def generate_dataset(theta, n, m):
    beta = np.random.normal(size=m+1)
    beta /= np.linalg.norm(beta)
    X = np.random.normal(size=(n, m))
    X = np.hstack((np.ones((n, 1)), X))

    p = 1 / (1 + np.exp(-np.dot(X, beta)))
    Y = (p > 0.5).astype(int)
    flip = np.random.binomial(1, theta, size=n)
    Y = np.logical_xor(Y, flip).astype(int)

    return X, Y, beta
```

- function that learns the parameters of a logistic regression line

```
import numpy as np
#k: number of iteractions (epochs)
#tau: threshold on change in Cost function value from the previous to current iteration
#lamda:λ:the learning rate for Gradient Descent
def logistic_regression(X, Y, k, tau, learning_rate):
    mean_X = np.mean(X, axis=0)
    std_X = np.std(X, axis=0)
    std_X[std_X == 0] = 1e-8
    X_normalized = (X - mean_X) / std_X
    beta = np.random.normal(size=X_normalized.shape[1])
    sigmoid = lambda x: 1 / (1 + np.exp(-x))
    cost = np.inf

    for _ in range(k):
        p = sigmoid(X_normalized.dot(beta))
        gradient = X_normalized.T.dot(p - Y)
        beta -= learning_rate * gradient
        new_cost = -np.sum(Y * np.log(p) + (1 - Y) * np.log(1 - p))

        if np.abs(new_cost - cost) < tau:
            break

        cost = new_cost

    return beta, cost
```

- Example with a training set.

- To investigate how different values of n and theta impact the ability for your logistic regression function to learn the coefficients, β, used to generate the output vector Y.

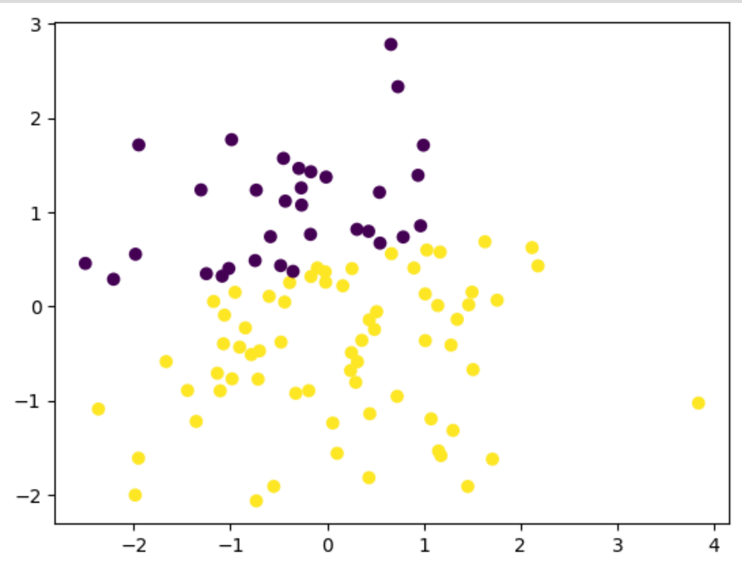
```python
#3D Scatter plot
num_samples = 20
n_values = np.linspace(50, 500, num_samples, dtype=int)
theta_values = np.linspace(0.1, 0.9, num_samples)
m = 3
k = 1000
tau = 1e-6
learning_rate = 0.01

n_mesh, theta_mesh = np.meshgrid(n_values, theta_values)
beta_diff = np.zeros_like(n_mesh, dtype=float)

for i in range(num_samples):
    for j in range(num_samples):
        n = n_values[i]
        theta = theta_values[j]

        X, Y, true_beta = generate_dataset(theta, n, m)
        estimated_beta, _ = logistic_regression(X, Y, k, tau, learning_rate)

        beta_diff[j, i] = np.linalg.norm(true_beta - estimated_beta)


fig = plt.figure(figsize=(10, 6))
ax = fig.add_subplot(111, projection='3d')
scatter = ax.scatter(n_mesh, theta_mesh, beta_diff, c=beta_diff, cmap='viridis')
ax.set_xlabel('n')
ax.set_ylabel('theta')
ax.set_zlabel('Beta Difference')
ax.set_title('Effect of n and theta on Beta Estimation')


cbar = fig.colorbar(scatter, ax=ax)
cbar.set_label('Beta Difference')

plt.show()
```

For further codes and visualizations, visit the below provided link for google Colab,

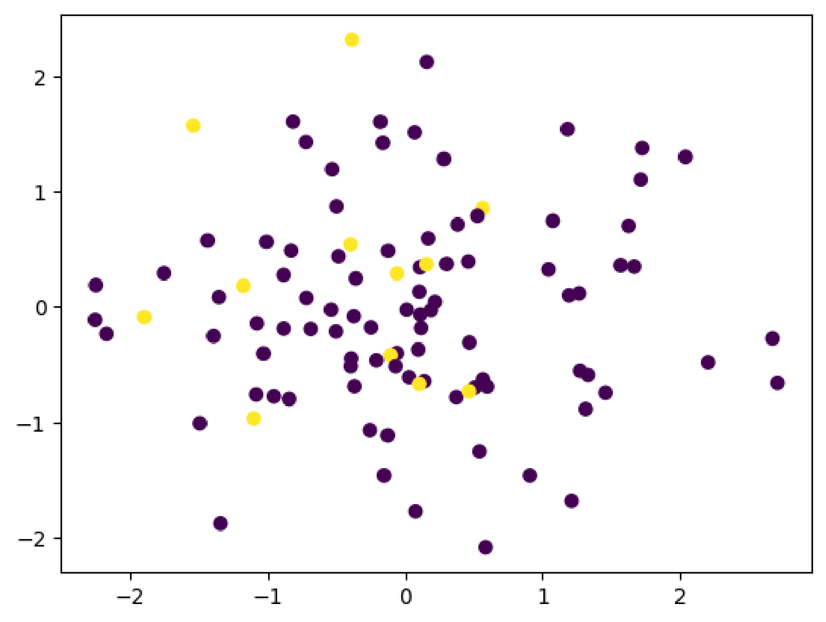
[Google colab](https://colab.research.google.com/drive/14SXuWbyM8yGomrTTcM-N57efmrtPvXQu?usp=chrome_ntp#scrollTo=if_rkmJVd2FK)

{ https://colab.research.google.com/drive/14SXuWbyM8yGomrTTcM-N57efmrtPvXQu?usp=chrome_ntp#scrollTo=if_rkmJVd2FK}

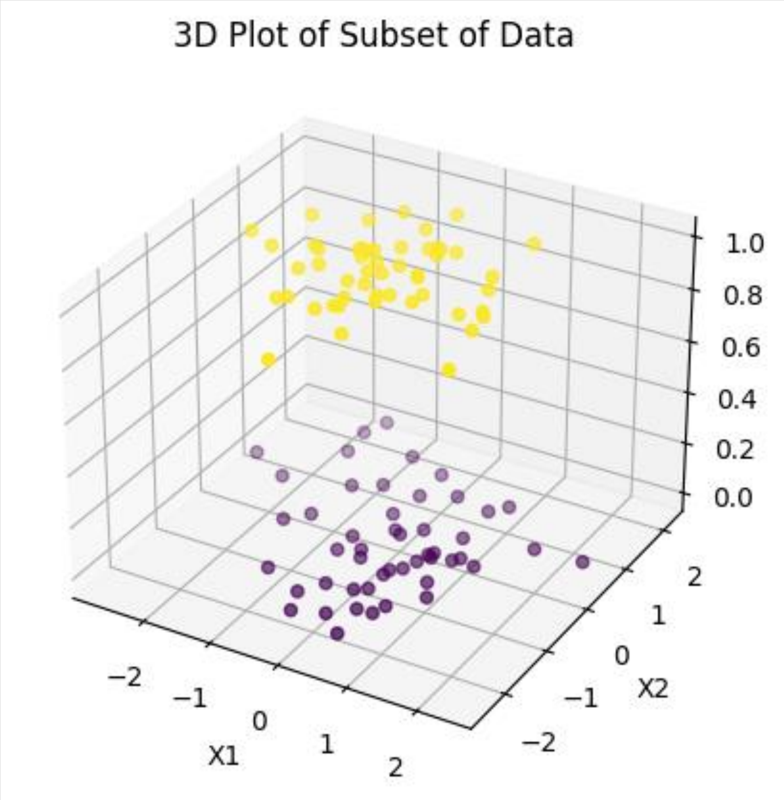
- Dataset without flipping



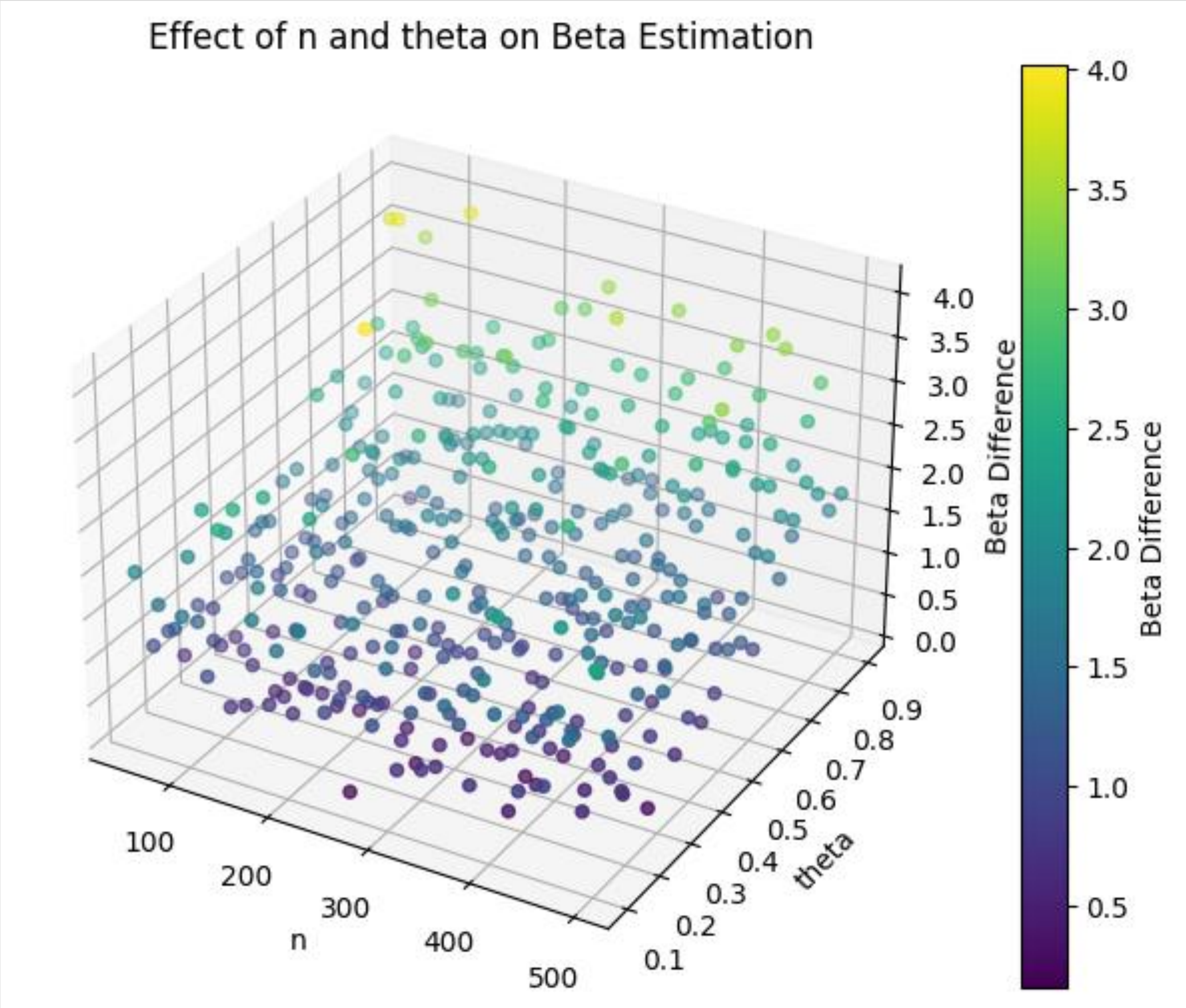
- Dataset with flipping



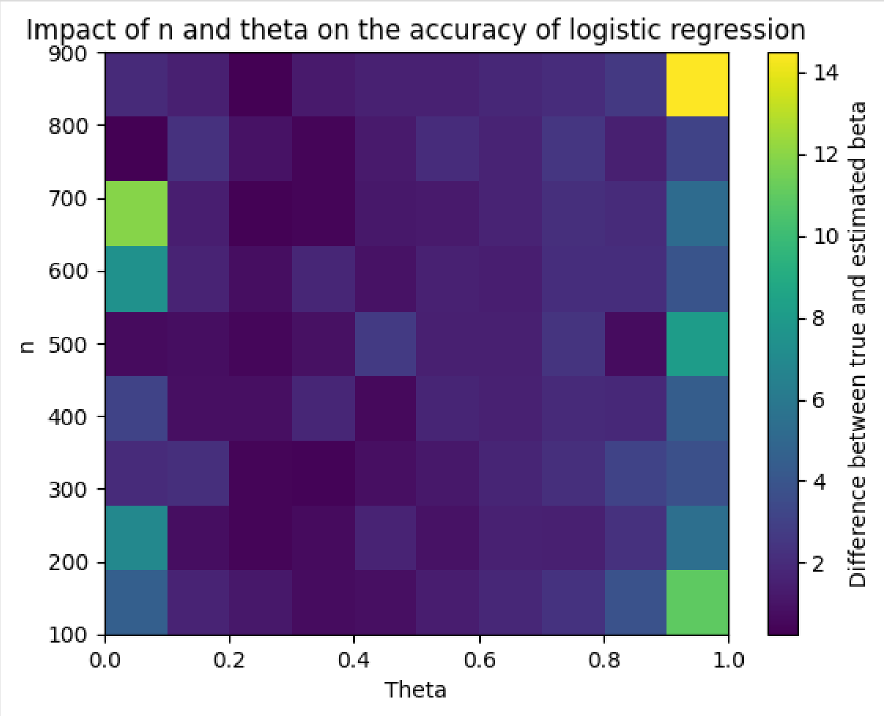
3D plot of subset of data



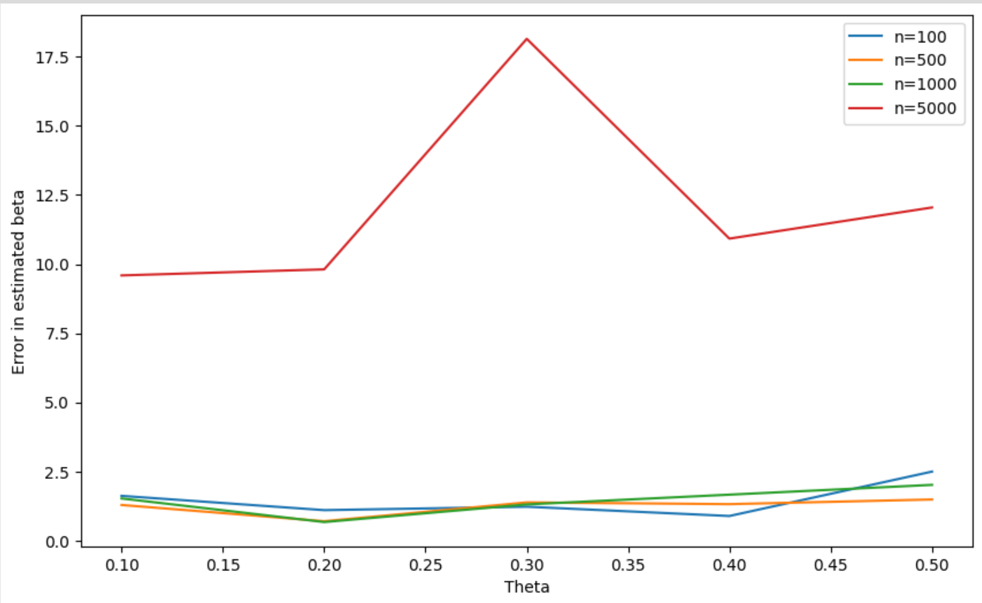
Visualization: 3-D Scatter plot;



Visualization: Heat Map;



Visualization: Line plots;



Observation and Inferences from the plots;

- As n increases, the logistic regression model has more data to learn from, which generally improves the accuracy of the estimated coefficients beta (can be observed from 3D scatter plot)
- θ is the noise level in the labels. As θ increases, the labels become noisier, making it harder for the logistic regression model to learn the true coefficients beta (Refer 3D Scatter plot)
- if n is too large, it may lead to overfitting, where the model learns the noise in the data instead of the underlying pattern (Refer line plot)
- If theta is too high, the model may learn from the noise instead of the underlying pattern, leading to poor generalization performance (refer 3D scatter plot)
- From the 3D scatter plot, we can infer that as the value θ increases, the beta difference also increases
- From the line plot, the least error in estimated beta can be observed at n=1000 and Theta=0.2
- Also in the line plot, the line n=5000 shows overfitting of data.

Derivations;

⇨ Derivation of the partial derivative of the cost function with respect to the parameters of the model (Logistic Regression model) ?

- Cost function for logistic regression is given by ;

$$J(\beta) = - \sum_{i=1}^{n} \left[y_i \log(P_i) + (1-y_i) \log(1-P_i) \right]$$

where ,

$$P_i = \frac{1}{1+e^{-x_i \beta}} \Rightarrow y_i = 1 \text{ given } x_i \ \& \ \beta$$

⇒ Derivation of gradient of cost function with respect to parameters β.

▷ derivative of P_i w.r.t β_j.

$$\frac{dP_i}{d\beta_j} = P_i (1-P_i) x_{ij}$$

▷ derivative of cost function w.r.t β_j ;

$$\frac{dJ(\beta)}{d\beta_j} = - \sum_{i=1}^{n} \left[y_i \frac{1}{P_i} \frac{dP_i}{\partial \beta_j} - (1-y_i) \frac{1}{1-P_i} \frac{\partial P_i}{\partial \beta_j} \right]$$

∴ on substitution we get ;

$$\frac{dJ(\beta)}{\partial \beta_j} = - \sum_{i=1}^{n} \left[y_i (1-P_i) x_{ij} - (1-y_i) P_i x_{ij} \right]$$

$$\Rightarrow \frac{\partial J(\beta)}{\partial \beta_j} = \sum_{i=1}^{n} (P_i - y_i) x_{ij}$$

$$\Rightarrow \frac{\partial J(\beta)}{\partial \beta} = X^T \cdot (P - y)$$

Here P : vector of predicted probabilities

y : vector of true labels

⇒ This gradient is used to update the coefficients β in the direction that minimizes the cost function

Here Learning rate (λ) : determines the step size in this direction.