**Experiment 3**

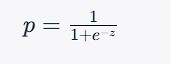
**Aim of Experiment**

Implement Logistic Regression

**Theory / Algorithm / Conceptual Description**

Logistic regression is a statistical method used for binary classification problems where the goal is to predict a binary outcome (e.g., yes/no, 0/1) based on one or more predictor variables. It is a type of generalized linear model that uses a logistic function (also known as the sigmoid function) to transform a linear combination of the predictor variables into a probability value between 0 and 1.

The logistic function is defined as follows:

where $p$ is the predicted probability of the positive outcome, $z$ is

the linear combination of the predictor variables and their coefficients, and $e$ is the mathematical constant known as the Euler's number.

The linear combination $z$ is given by:



where $\beta\_0$ is the intercept or bias term, $\beta\_1, \beta\_2, ..., \beta\_p$ are the coefficients or weights assigned to the predictor variables $x\_1, x\_2, ..., x\_p$ respectively.

To fit a logistic regression model, we use maximum likelihood estimation to find the values of the coefficients that maximize the likelihood of the observed data. This involves finding the set of coefficients that make the predicted probabilities as close as possible to the actual binary outcomes in the training data.

Once the model is trained, we can use it to predict the probability of the positive outcome for new data points by plugging in the values of the predictor variables into the logistic function and computing the predicted probability. We can then use a threshold (usually 0.5) to convert the predicted probabilities into binary predictions.

Logistic regression has many extensions, such as multinomial logistic regression for multi-class classification problems and ordinal logistic regression for ordered categorical outcomes. It is widely used in many fields, such as healthcare, finance, marketing, and social sciences.

**Program**

|  |
| --- |
| import pandas as pd from sklearn.model\_selection import train\_test\_split from sklearn.preprocessing import LabelEncoder import numpy as np  import matplotlib.pyplot as plt    df = pd.read\_csv('diabetes.csv')      df.shape    # le = LabelEncoder()  # df['age'] = le.fit\_transform(df['age'])  # df['income'] = le.fit\_transform(df['income'])  # df['student'] = le.fit\_transform(df['student'])  # df['credit\_rating'] = le.fit\_transform(df['credit\_rating'])  # df['buys\_computer'] = le.fit\_transform(df['buys\_computer'])     1. = df.iloc[:, :-1] 2. = df.iloc[:, -1]   X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, Y, test\_size=0.2, random\_state=42)    def sigmoid(z):  return 1.0/(1 + np.exp(-z))    def gradients(X, y, y\_hat):    m = X.shape[0]    dw = (1/m)\*np.dot(X.T, (y\_hat - y)) return dw    def train(X, y,epochs, lr):  m, n = X.shape w = np.zeros((n,1)) y = y.reshape(m,1)  x = X for epoch in range(epochs): for i in range(m): |

|  |
| --- |
| xb = X  yb = y  y\_hat = sigmoid(np.dot(xb, w)) dw = gradients(xb, yb, y\_hat) w -= lr\*dw    return w    w = train(X\_train.values, y\_train.values, epochs=1000, lr=0.01)    print(w)    def predict(X):    preds = sigmoid(np.dot(X, w))    pred\_class = []  pred\_class = [1 if i > 0.5 else 0 for i in preds]    return np.array(pred\_class)    def accuracy(y, y\_hat): accuracy = np.sum(y == y\_hat) / len(y) return accuracy    predict(X\_test.values)    accuracy(y\_test.values, predict(X\_test.values))    # Logistic Regression    ## Importing the libraries    import numpy as np  import matplotlib.pyplot as plt import pandas as pd    ## Importing the dataset    dataset = pd.read\_csv('Social\_Network\_Ads.csv')  X = dataset.iloc[:, :-1].values y = dataset.iloc[:, -1].values    ## Splitting the dataset into the Training set and Test set    from sklearn.model\_selection import train\_test\_split |

|  |
| --- |
| X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.25, random\_state = 0)    print(X\_train)    print(y\_train)    print(X\_test)    print(y\_test)    ## Feature Scaling    from sklearn.preprocessing import StandardScaler sc = StandardScaler()  X\_train = sc.fit\_transform(X\_train)  X\_test = sc.transform(X\_test)    print(X\_train)    print(X\_test)    ## Training the Logistic Regression model on the Training set    from sklearn.linear\_model import LogisticRegression classifier = LogisticRegression(random\_state = 0) classifier.fit(X\_train, y\_train)    ## Predicting a new result    print(classifier.predict(sc.transform([[30,87000]])))    ## Predicting the Test set results    y\_pred = classifier.predict(X\_test)  print(np.concatenate((y\_pred.reshape(len(y\_pred),1), y\_test.reshape(len(y\_test),1)),1))    ## Making the Confusion Matrix    from sklearn.metrics import confusion\_matrix, accuracy\_score cm = confusion\_matrix(y\_test, y\_pred) print(cm)  accuracy\_score(y\_test, y\_pred)    ## Visualising the Training set results    from matplotlib.colors import ListedColormap X\_set, y\_set = sc.inverse\_transform(X\_train), y\_train  X1, X2 = np.meshgrid(np.arange(start = X\_set[:, 0].min() - 10, stop = X\_set[:, 0].max() + 10, step =  0.25), |
| np.arange(start = X\_set[:, 1].min() - 1000, stop = X\_set[:, 1].max() + 1000, step = 0.25)) plt.contourf(X1, X2, classifier.predict(sc.transform(np.array([X1.ravel(), X2.ravel()]).T)).reshape(X1.shape),  alpha = 0.75, cmap = ListedColormap(('salmon', 'dodgerblue'))) plt.xlim(X1.min(), X1.max()) plt.ylim(X2.min(), X2.max()) for i, j in enumerate(np.unique(y\_set)):  plt.scatter(X\_set[y\_set == j, 0], X\_set[y\_set == j, 1], c = ListedColormap(('salmon', 'dodgerblue'))(i), label = j)  plt.title('Logistic Regression (Training set)') plt.xlabel('Age')  plt.ylabel('Estimated Salary') plt.legend() plt.show()    ## Visualising the Test set results    from matplotlib.colors import ListedColormap X\_set, y\_set = sc.inverse\_transform(X\_test), y\_test  X1, X2 = np.meshgrid(np.arange(start = X\_set[:, 0].min() - 10, stop = X\_set[:, 0].max() + 10, step =  0.25), np.arange(start = X\_set[:, 1].min() - 1000, stop = X\_set[:, 1].max() + 1000, step = 0.25)) plt.contourf(X1, X2, classifier.predict(sc.transform(np.array([X1.ravel(), X2.ravel()]).T)).reshape(X1.shape),  alpha = 0.75, cmap = ListedColormap(('salmon', 'dodgerblue'))) plt.xlim(X1.min(), X1.max()) plt.ylim(X2.min(), X2.max()) for i, j in enumerate(np.unique(y\_set)): plt.scatter(X\_set[y\_set == j, 0], X\_set[y\_set == j, 1], c = ListedColormap(('salmon', 'dodgerblue'))(i), label = j)  plt.title('Logistic Regression (Test set)') plt.xlabel('Age')  plt.ylabel('Estimated Salary') plt.legend() plt.show() |

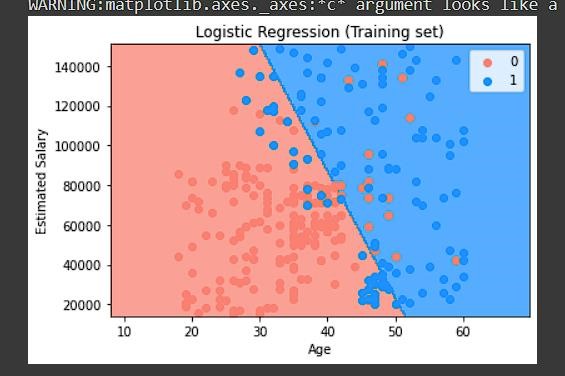
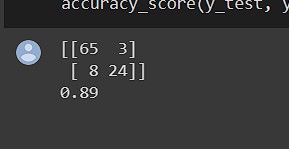
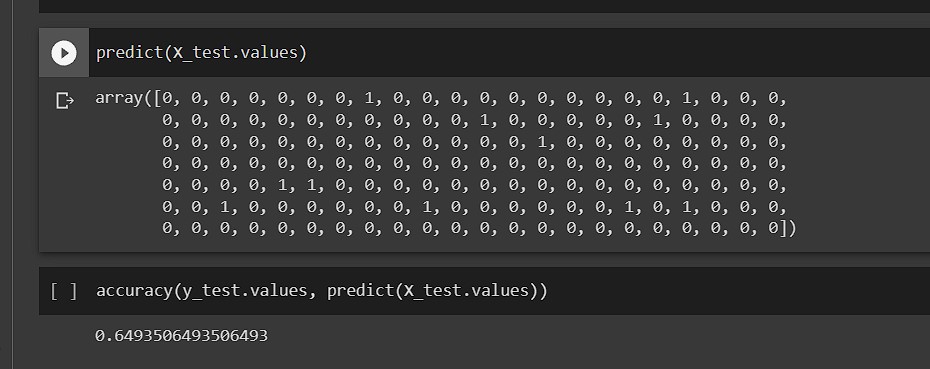
**Output**

|  |
| --- |
|  |

I

n

built functions



**Conclusion:** Logistic Regression was implemented from scratch and using the Sci Kit Learn Library