PERCEPTRON CLASSIFIER

AIM:- To implement a basic Perceptron classifier for binary classification on a diabetes dataset.

PROBLEM DESCRIPTION:- The task is to create a machine learning model using a Perceptron algorithm to predict the likelihood of diabetes occurrence in individuals based on certain health-related features. The dataset used contains information about patients, including attributes such as pregnancies, glucose levels, blood pressure, skin thickness, insulin levels, BMI, diabetes pedigree function, and age. The aim is to train a model that can accurately classify whether a patient has diabetes (Outcome = 1) or not (Outcome = 0) based on these features.

ALGORITHM:-

1. Importing Libraries:
   * numpy as np and pandas as pd are imported for numerical operations and data handling, respectively.

2) Defining the Perceptron Class:

* + Perceptron class is created to implement a perceptron model.

3) Initialization Function (\_\_init\_\_):

* + Initializes the perceptron with parameters:
    - learning\_rate: The learning rate used in the weight update.
    - epochs: The number of iterations over the training dataset.

4) Activation Function (activation):

* + Defines the activation function as the Heaviside step function using np.heaviside().

5) Fitting the Model (fit):

* + Initializes weights and bias as None.
  + Loops through epochs and within each epoch:
    - Loops through each sample in the training data.
    - Calculates the weighted sum (z) of input features and current weights plus bias.
    - Updates weights and bias using the perceptron learning rule:
      * self.weights = self.weights + self.learning\_rate \* (y[i] - y\_pred[i]) \* X[i]
      * self.bias = self.bias + self.learning\_rate \* (y[i] - y\_pred[i])
    - Returns the learned weights and bias.

6) Prediction Function (predict):

* + Calculates the weighted sum (z) of input features and learned weights plus bias.
  + Applies the activation function to the calculated z and returns the result.

7) Reading Data:

* + Reads a CSV file containing diabetes data using pd.read\_csv().

8) Splitting Data:

* + Splits the data into features X and target y using train\_test\_split() from sklearn.
  + Splits the data into training and testing sets with a test size of 50% and a specified random state.

9) Initializing Perceptron:

* + Creates an instance of the Perceptron class with a learning rate of 0.001 and 100 epochs.

10) Training the Perceptron:

* + Calls the fit method of the perceptron instance on the training data (X\_train, y\_train).

11) Making Predictions:

* + Uses the trained perceptron to predict outcomes on the test data (X\_test) using the predict method.

12) Evaluating Model Performance:

* + Calculates the accuracy score using accuracy\_score() from sklearn.metrics between predictions (pred) and actual test labels (y\_test).
  + Prints the accuracy score as a percentage.
  + Generates a classification report using classification\_report() to show precision, recall, F1-score, and support for each class based on the predictions and actual labels.

PSEUDOCODE:-

import numpy as np

import pandas as pd

class Perceptron:

def \_\_init\_\_(self, learning\_rate, epochs):

self.weights = None

self.bias = None

self.learning\_rate = learning\_rate

self.epochs = epochs

def activation(self, z):

return np.heaviside(z, 0)

def fit(self, X, y):

n\_features = X.shape[1]

self.weights = np.zeros((n\_features))

self.bias = 0

for epoch in range(self.epochs):

for i in range(len(X)):

z = np.dot(X, self.weights) + self.bias

self.weights = self.weights + self.learning\_rate \* (y[i] - y\_pred[i]) \* X[i]

self.bias = self.bias + self.learning\_rate \* (y[i] - y\_pred[i])

return self.weights, self.bias

def predict(self, X):

z = np.dot(X, self.weights) + self.bias

return self.activation(z)

diabetes\_df = pd.read\_csv(r"C:\Users\user\Desktop\ANJALI RAJ\Parceptron Classifier\diabetes.csv")

diabetes\_df.head()

from sklearn.model\_selection import train\_test\_split

X=diabetes\_df[['Pregnancies','Glucose','BloodPressure','SkinThickness','Insulin','BMI','DiabetesPedigreeFunction','Age']].values

y = diabetes\_df['Outcome'].values

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.5, random\_state=42)

perceptron = Perceptron(0.001, 100)

perceptron.fit(X\_train, y\_train)

pred = perceptron.predict(X\_test)

from sklearn.metrics import accuracy\_score

print("Accuracy Score : ",(accuracy\_score(pred, y\_test))\*100,"%")

from sklearn.metrics import classification\_report

report = classification\_report(pred, y\_test, digits=2)

print(report)

RESULT:-

The trained model correctly predicts the precision and accuracy.

