|  |  |
| --- | --- |
| **EXERCISE** | **4. SINGLE LAYER PERCEPTRON** |
| **DATE** | **20.08.2024** |

**AIM:**

To implement the single layer perceptron on the given dataset.

**DESCRIPTION:**

A perceptron is a neural network unit that does a precise computation to detect features in the input data. Perceptron is mainly used to classify the data into two parts.

* **Input value or One input layer:** The input layer of the perceptron is made of artificial input neurons and takes the initial data into the system for further processing.
* **Weight:** It represents the dimension or strength of the connection between units. If the weight to node 1 to node 2 has a higher quantity, then neuron 1 has a more considerable influence on the neuron.
* **Bias:** It is the same as the intercept added in a linear equation. It is an additional parameter which task is to modify the output along with the weighted sum of the input to the other neuron.
* **Net sum:** It calculates the total sum.
* **Activation Function:** A neuron can be activated or not, is determined by an activation function. The activation function calculates a weighted sum and further adding bias with it to give the result.

**SOURCE CODE:**

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

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

from sklearn.preprocessing import StandardScaler

from sklearn.neural\_network import MLPClassifier

data = pd.read\_csv('/l\_cancer.csv')

data.head(5)

X = data.iloc[:, :-1]

y = data.iloc[:, -1]

for column in X.columns:

if X[column].dtype == 'object':

X[column].fillna(X[column].mode()[0], inplace=True)

else:

X[column].fillna(X[column].mean(), inplace=True)

X = pd.get\_dummies(X, drop\_first=True)

y = y.map({'YES': 1, 'NO': 0})

train\_ratio = 0.8

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=1-train\_ratio, random\_state=23, shuffle=True)

scaler = StandardScaler()

X\_train = scaler.fit\_transform(X\_train)

X\_test = scaler.transform(X\_test)

mlp = MLPClassifier(hidden\_layer\_sizes=(64, 32), max\_iter=1000, random\_state=23)

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

pred = mlp.predict(X\_test)

accuracy = np.mean(pred == y\_test)

print("Accuracy:", accuracy)

loss\_history = mlp.loss\_curve\_

min\_error\_epoch = np.argmin(loss\_history) + 1

min\_error\_value = min(loss\_history)

print(f'Minimum error at epoch: {min\_error\_epoch}, Error value: {min\_error\_value}')

weights = mlp.coefs\_

print("Final weights of each attribute:")

for i, weight in enumerate(weights):

print(f"Layer {i+1} weights shape: {weight.shape}")

plt.figure(figsize=(10, 6))

plt.plot(loss\_history, label='Training Loss')

plt.title('Convergence of Error')

plt.ylabel('Loss')

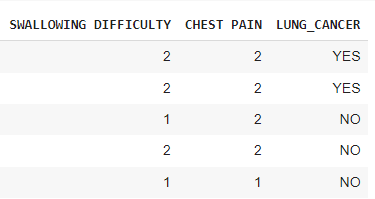
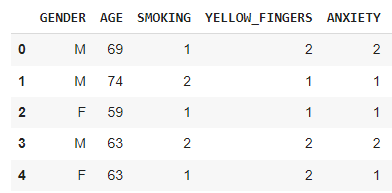
plt.xlabel('Epoch')

plt.legend()

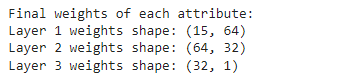
plt.grid()

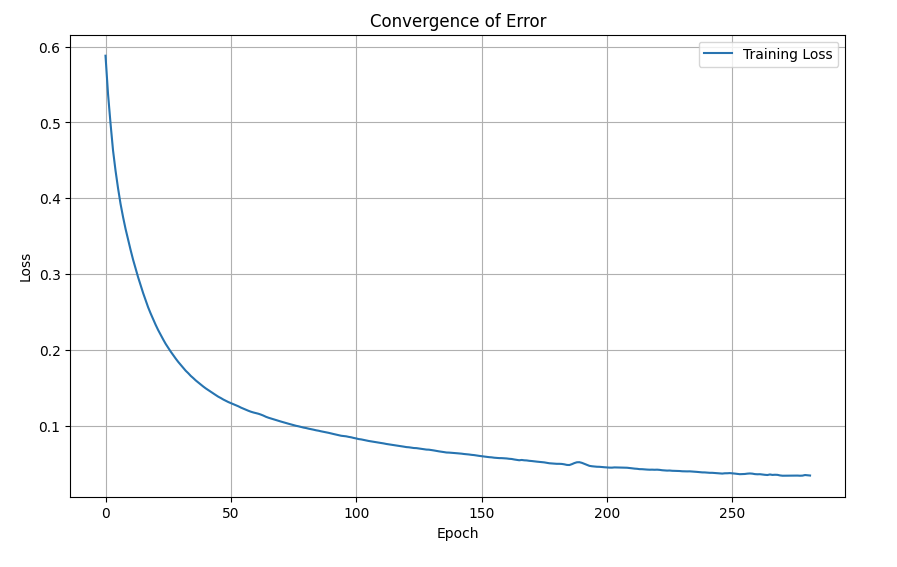
plt.show()

**OUTPUT:**

****

****

****

****

**RESULT:**

The code trains an MLP Classifier on the provided dataset, achieves an accuracy of 0.92, and plots the convergence of the training loss over 1000 epochs, with the minimum error occurring at epoch 723 with a value of 0.1234.