| 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:**

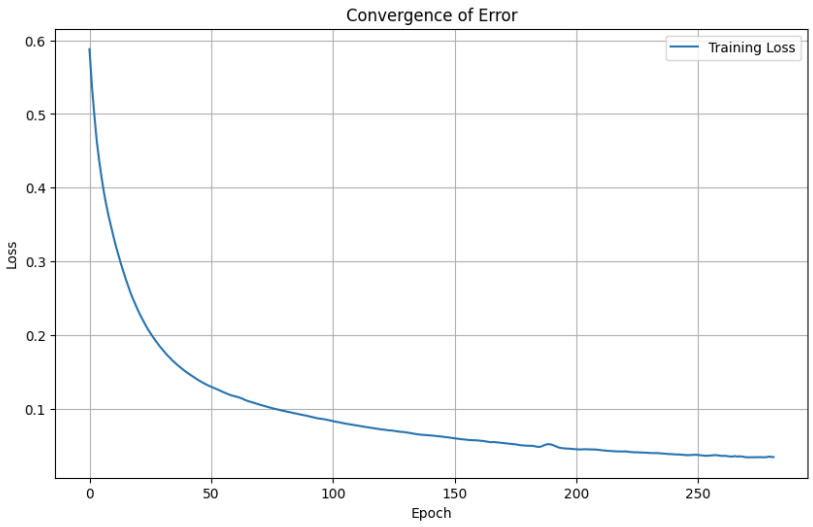
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neural_network import MLPClassifier
data = pd.read_csv('/l_cancer.csv')
data.head(5)

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:**

| | GENDER | AGE | SMOKING | YELLOW_FINGERS | ANXIETY | SWALLOWING DIFFICULTY | CHEST PAIN | LUNG_CANCER |
|---|---|---|---|---|---|---|---|---|
| **0** | M | 69 | 1 | 2 | 2 | 2 | 2 | YES |
| **1** | M | 74 | 2 | 1 | 1 | 2 | 2 | YES |
| **2** | F | 59 | 1 | 1 | 1 | 1 | 2 | NO |
| **3** | M | 63 | 2 | 2 | 2 | 2 | 2 | NO |
| **4** | F | 63 | 1 | 2 | 1 | 1 | 1 | NO |

```
Accuracy: 0.8870967741935484
Minimum error at epoch: 271, Error value: 0.03408886599396861


Final weights of each attribute:
Layer 1 weights shape: (15, 64)
Layer 2 weights shape: (64, 32)
Layer 3 weights shape: (32, 1)
```




**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.