SHRI SAI ARAVIND. R
212223040197
EX. NO.4
DATE:

Implementation of MLP with Backpropagation for Multiclassification

Aim:

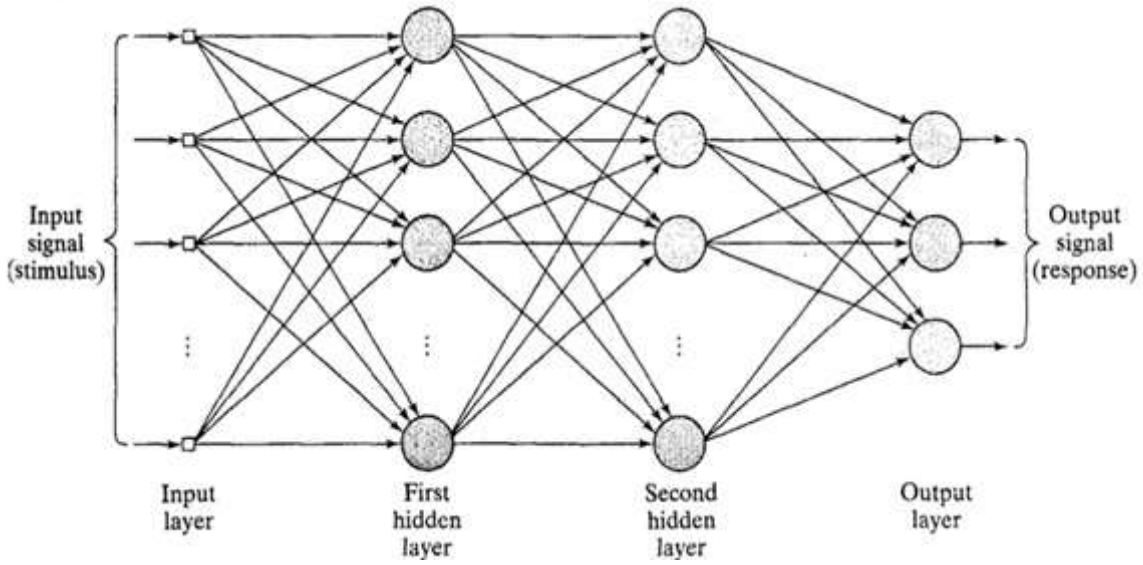
To implement a Multilayer Perceptron for Multi classification

Theory

A multilayer perceptron (MLP) is a feedforward artificial neural network that generates a set of outputs from a set of inputs. An MLP is characterized by several layers of input nodes connected as a directed graph between the input and output layers. MLP uses back propagation for training the network. MLP is a deep learning method. A multilayer perceptron is a neural network connecting multiple layers in a directed graph, which means that the signal path through the nodes only goes one way. Each node, apart from the input nodes, has a nonlinear activation function. An MLP uses backpropagation as a supervised learning technique. MLP is widely used for solving problems that require supervised learning as well as research into computational neuroscience and parallel distributed processing. Applications include speech recognition, image recognition and machine translation.
MLP has the following features:
Ø Adjusts the synaptic weights based on Error Correction Rule
Ø Adopts LMS
Ø possess Backpropagation algorithm for recurrent propagation of error
Ø Consists of two passes

```
(i)Feed Forward pass
        (ii)Backward pass
```

Ø Learning process –backpropagation
Ø Computationally efficient method



3 Distinctive Characteristics of MLP:
Ø Each neuron in network includes a non-linear activation function

$$y_j = \frac{1}{1 + \exp(-v_j)}$$

Ø Contains one or more hidden layers with hidden neurons
Ø Network exhibits high degree of connectivity determined by the synapses of the network
3 Signals involved in MLP are:
Functional Signal
*input signal
*propagates forward neuron by neuron thro network and emerges at an output signal
*F(x,w) at each neuron as it passes
Error Signal
*Originates at an output neuron
*Propagates backward through the network neuron
*Involves error dependent function in one way or the other
Each hidden neuron or output neuron of MLP is designed to perform two computations:
The computation of the function signal appearing at the output of a neuron which is expressed as a continuous non-linear function of the input signal and synaptic weights associated with that neuron
The computation of an estimate of the gradient vector is needed for the backward pass through the network
TWO PASSES OF COMPUTATION:
In the forward pass:
• Synaptic weights remain unaltered
• Function signal are computed neuron by neuron

$$y_j(n) = \varphi(v_j(n))$$

• Function signal of jth neuron is

where $v_j(n)$ is the induced local field of neuron j, defined by

$$v_j(n) = \sum_{i=0}^{m} w_{ji}(n) y_i(n)$$

If neuron j is in the first hidden layer of the network, $m = m_0$ and the index i the ith input terminal of the network, for which we write

$$y_i(n) = x_i(n)$$

$$y_j(n) = o_j(n)$$

If jth neuron is output neuron, the m=mL and output of j th neuron is
Forward phase begins with in the first hidden layer and end by computing ej(n) in the output layer

$$e_j(n) = d_j(n) - y_j(n),$$

In the backward pass,
• It starts from the output layer by passing error signal towards leftward layer neurons to compute local gradient recursively in each neuron

• it changes the synaptic weight by delta rule

The correction $\Delta w_{ji}(n)$ applied to $w_{ji}(n)$ is defined by the *delta rule*:

$$\Delta w_{ji}(n) = -\eta \frac{\partial \mathscr{E}(n)}{\partial w_{ji}(n)} \tag{4.12}$$

Algorithm:

1. Import the necessary libraries of python.
2. After that, create a list of attribute names in the dataset and use it in a call to the read_csv() function of the pandas library along with the name of the CSV file containing the dataset.
3. Divide the dataset into two parts. While the first part contains the first four columns that we assign in the variable x. Likewise, the second part contains only the last column that is the class label. Further, assign it to the variable y.
4. Call the train_test_split() function that further divides the dataset into training data and testing data with a testing data size of 20%. Normalize our dataset.
5. In order to do that we call the StandardScaler() function. Basically, the StandardScaler() function subtracts the mean from a feature and scales it to the unit variance.
6. Invoke the MLPClassifier() function with appropriate parameters indicating the hidden layer sizes, activation function, and the maximum number of iterations.
7. In order to get the predicted values we call the predict() function on the testing data set.
8. Finally, call the functions confusion_matrix(), and the classification_report() in order to evaluate the performance of our classifier.

Program:

```
#Include packages and builtin classes
import pandas as pd
import sklearn
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import classification_report, confusion_matrix

#Read the csv file to be considered for Multi-classification
df = pd.read_csv("Iris_data.csv")
df.head()
# Seperate the input features and target from the dataset
X = df.drop("species", axis = 1)
y = df["species"]

X
X.isnull().sum()
X = X.fillna(X.median(numeric_only=True))
X.isnull().sum()
```

```
y
#Transform the categorial into numerical values
le = LabelEncoder()
yenc = le.fit_transform(y)
yenc

#Split the data  for training  and testing
x_train, x_test, y_train, y_test = train_test_split(X,yenc, test_size=0.3, random_state=42)

# Normalizing input features
scaler = StandardScaler()
x_train_scaled = scaler.fit_transform(x_train)
x_test_scaled = scaler.transform(x_test)

x_train_scaled

x_test_scaled

#Define the MLP classifier
classifier = MLPClassifier(hidden_layer_sizes=(10,10,10), max_iter=1000)
#Train the classifier
classifier.fit(x_train_scaled, y_train)

predicitons = classifier.predict(x_test_scaled)
flowers = le.inverse_transform(predicitons)
res_df = pd.DataFrame({"Predicted Flowers":flowers, "Actual Flowers": le.inverse_transform(
res_df.tail(15)

# Evaluation of algorithm performance in classifying.
confusion_matrix(y_test, predicitons)

print(classification_report(y_test, predicitons))
from sklearn.metrics import accuracy_score

f"{accuracy_score(y_test, predicitons)*100:2f}"
```

Output:

Dataset

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

Labeled output set

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2])
```

Scaled Trian set

```
array([[-0.4134164 , -1.46135995, -0.09283739, -0.32149987],
       [ 0.55122187, -0.49173723,  0.72927683,  0.35364985],
       [ 0.67180165,  0.2354798 ,  0.96416661,  0.75873969],
       [ 0.91296121, -0.00692588,  0.31821972,  0.21861991],
       [ 1.63643991,  1.44750819,  1.31650128,  1.7039493 ],
       [-0.17225683, -0.24933155,  0.20077483,  0.08358997],
       [ 2.11875905, -0.00692588,  1.6101135 ,  1.16382952],
       [-0.29283662, -0.00692588,  0.37694217,  0.35364985],
       [-0.89573553,  1.20510252, -1.44345361, -1.40173942],
       [ 2.23933883, -0.49173723,  1.66883594,  1.02879957],
       [-0.05167705, -0.73414291,  0.14205239, -0.32149987],
       [-0.77515575,  0.96269684, -1.44345361, -1.40173942],
       [-1.01631531,  1.20510252, -1.50217605, -1.26670948],
       [-0.89573553,  1.93231955, -1.14984139, -1.13167953],
       [-1.01631531, -2.43098266, -0.21028228, -0.32149987],
       [ 0.55122187, -0.73414291,  0.61183194,  0.75873969],
       [-1.25747488,  0.96269684, -1.14984139, -1.40173942],
       [-1.01631531, -0.00692588, -1.32600872, -1.40173942],
       [-0.89573553,  0.72029116, -1.26728628, -0.99664959],
       [-0.29283662, -0.73414291,  0.20077483,  0.08358997],
       [-0.89573553,  0.96269684, -1.38473116, -1.40173942],
       [-0.17225683, -0.00692588,  0.20077483, -0.05143998],
       [ 2.23933883,  1.93231955,  1.66883594,  1.29885946],
       [-1.49863445,  0.47788548, -1.44345361, -1.40173942],
       [ 0.43064208, -0.24933155,  0.25949728,  0.08358997],
...
       [ 0.3100623 , -0.49173723,  0.08332994,  0.08358997],
       [-1.1368951 , -1.21895427,  0.37694217,  0.62370974],
       [-0.05167705,  2.41713091, -1.5608985 , -1.40173942],
       [-0.05167705, -0.97654859,  0.08332994, -0.05143998],
       [ 1.51586013, -0.00692588,  1.19905639,  1.16382952]])
```

Scaled Test set

```
array([[ 0.3100623 , -0.49173723,  0.49438706, -0.05143998],
       [-0.17225683,  1.93231955, -1.26728628, -1.26670948],
       [ 2.23933883, -0.97654859,  1.78628083,  1.43388941],
       [ 0.18948252, -0.24933155,  0.37694217,  0.35364985],
       [ 1.15412078, -0.49173723,  0.5531095 ,  0.21861991],
       [-0.53399618,  0.96269684, -1.38473116, -1.13167953],
       [-0.29283662, -0.24933155, -0.15155983,  0.08358997],
       [ 1.27470056,  0.2354798 ,  0.72927683,  1.43388941],
       [ 0.43064208, -1.9461713 ,  0.37694217,  0.35364985],
       [-0.05167705, -0.73414291,  0.0246075 , -0.05143998],
       [ 0.79238143,  0.47788548,  0.72927683,  1.02879957],
       [-1.25747488, -0.00692588, -1.44345361, -1.53676936],
       [-0.4134164 ,  1.20510252, -1.50217605, -1.40173942],
       [-1.1368951 ,  0.2354798 , -1.38473116, -1.53676936],
       [-0.89573553,  1.93231955, -1.38473116, -1.26670948],
       [ 0.55122187,  0.72029116,  0.49438706,  0.4886798 ],
       [ 0.79238143, -0.00692588,  1.14033394,  1.29885946],
       [-0.29283662, -1.21895427,  0.0246075 , -0.18646992],
       [-0.17225683, -0.49173723,  0.37694217,  0.08358997],
       [ 0.67180165, -0.49173723,  1.02288906,  1.29885946],
       [-1.37805466, -0.00692588,  0.25949728, -1.40173942],
       [ 0.3100623 , -0.00692588,  0.61183194,  0.75873969],
       [-1.01631531,  0.96269684, -1.32600872, -1.13167953],
       [ 0.67180165, -0.49173723,  1.02288906,  1.16382952],
       [ 2.4804984 ,  1.93231955,  1.49266861,  1.02879957],
...
       [-0.05167705, -0.73414291,  0.72927683,  0.89376963],
       [ 0.18948252,  0.96269684,  0.37694217,  0.4886798 ],
       [ 1.033541  ,  0.2354798 ,  0.49438706,  0.35364985],
       [-0.53399618,  2.17472523, -1.50217605, -1.13167953],
       [-0.53399618,  1.68991387, -1.38473116, -1.40173942]])
```

MLP Classifier property

```
MLPClassifier(hidden_layer_sizes=(10, 10, 10), max_iter=1000)
```

Predicted vs Actual Labels sample

	Predicted Flowers	Actual Flowers
30	Iris-setosa	Iris-setosa
31	Iris-setosa	Iris-setosa
32	Iris-versicolor	Iris-versicolor
33	Iris-setosa	Iris-setosa
34	Iris-setosa	Iris-setosa
35	Iris-virginica	Iris-virginica
36	Iris-versicolor	Iris-versicolor
37	Iris-setosa	Iris-setosa
38	Iris-setosa	Iris-setosa
39	Iris-setosa	Iris-setosa
40	Iris-virginica	Iris-virginica
41	Iris-versicolor	Iris-versicolor
42	Iris-versicolor	Iris-versicolor
43	Iris-setosa	Iris-setosa
44	Iris-setosa	Iris-setosa

Confusion Matrix

```
array([[18,  1,  0],
       [ 0, 13,  0],
       [ 0,  1, 12]], dtype=int64)
```

Classification Report

```
              precision    recall  f1-score   support

           0       1.00      0.95      0.97        19
           1       0.87      1.00      0.93        13
           2       1.00      0.92      0.96        13

    accuracy                           0.96        45
   macro avg       0.96      0.96      0.95        45
weighted avg       0.96      0.96      0.96        45
```

Accuracy Score

```
'95.555556'
```

Result:

Thus, MLP is implemented for multi-classification using python.