Experiment

A python program to Implement Single layer perceptron

Aim:

To implement python program for the Single layer perceptron

Algorithm:

Step 1: Import necessary libraries:

* Import numpy for numerical operation

Step 2: Initialize the perceptron.

* Define the number of input feature (input_dim).
* Initialize weight (w) and bias (b) to zero or small random values.

Step 3: Define Activation function:

* Choose an activation function (eg. step function, sigmoid, Relu).
* User Defined function-sigmoid-func(x):
* Compute $1/(1+np.exp(-x))$ and return the value.

Step 4: Define Training Data:

* Define input features (x) and corresponding target lables (y).

Step 5: Define Learning Rate and number of Epochs,

* Choose a learning rate (alpha) and the number of training epochs.

Step 6: Training the perceptron:

* For each epoch:

  * For each input sample in the training data:

    » Compute the weighted sum of inputs (z) as the dot product of input features and weights plus bias (z = np.dot(x[i], w) + b).

    » Compute the error (error = y[i] - y_pred).

Step 7: Prediction:

* use the trained perceptrun to Predict the output for new input data.

Step 8: Evaluate the mode:

* Measure the Performance of the model using metrics Such as accuracy, precision, recall, etc.

Program:

```
import numpy as np
Import Pandas as pd
input_values = np.array([[0,0],[0,1],[1,1],[1,a]])
input_values.shape(4,2)
output = np.array([0,0,1,0])
output = output.reshape(4,1)
output.shape
weights = np.array([[0.1],[0.3]])
weights
bias = 0.2
def sigmoid_func(x):
        return 1/(1+np.exp(-x))
```

```
def der(x):
    return sigmoid_func(x)*(1-sigmoid_
                              func(x))

for epochs in range(15 000):
    input_arr = input_value
    weighted_sum = np.dot(input_arr, weights)+
    bias first_output = sigmoid_func(weighted_
      sum) error = first_output - output.

    tot_error = np.square(np.subtract(first_
                      output, output)).mean())

    first_der = error
    second_der = der(first_output)
    derivative = first_der * second_der
    t_input = input_value.T
    final_derivative = np.dot(t_input, derivative)
    weights = weights - (0.05 * final_derivative)
    for i in derivative
        bias = bias - (0.05 * i)
    print(weights)
    print(bias)
```

```
pred=np.array([1,0])
result = np.dot(pred weights)+bias
res=sigmoid_func(result)
print (res)
pred=np.array([1,1])
result=np.dot(pred, weights)+bias
res=sigmoid_func(result)
print(res)
pred=np.array([0,0])
result=np.dot(pred, weights)+bias
res=sigmoid_func(result)
print (res)
pred=np.array([0,1])
result=np.dot(pred weights)+bias
res=sigmoid_func(result)
print(res)
```

Result:

A [illegible] 30/10 Thus, the python program for implement single layer perception was executed successfully.