Shri Sai Aravind R
212223040197
Date
Experiment No. 2

Implementation of Perceptron for Binary Classification

AIM:

To implement a perceptron for classification using Python

EQUIPMENTS REQUIRED:

Hardware – PCs Anaconda – Python 3.7 Installation / Google Colab /Jupiter Notebook

RELATED THEORETICAL CONCEPT:

A Perceptron is a basic learning algorithm invented in 1959 by Frank Rosenblatt. It is meant to mimic the working logic of a biological neuron. The human brain is basically a collection of many interconnected neurons. Each one receives a set of inputs, applies some sort of computation on them and propagates the result to other neurons.
A Perceptron is an algorithm used for supervised learning of binary classifiers.Given a sample, the neuron classifies it by assigning a weight to its features. To accomplish this a Perceptron undergoes two phases: training and testing. During training phase weights are initialized to an arbitrary value. Perceptron is then asked to evaluate a sample and compare its decision with the actual class of the sample.If the algorithm chose the wrong class weights are adjusted to better match that particular sample. This process is repeated over and over to finely optimize the biases. After that, the algorithm is ready to be tested against a new set of completely unknown samples to evaluate if the trained model is general enough to cope with real-world samples.
The important Key points to be focused to implement a perceptron: Models have to be trained with a high number of already classified samples. It is difficult to know a priori this number: a few dozen may be enough in very simple cases while in others thousands or more are needed. Data is almost never perfect: a preprocessing phase has to take care of missing features, uncorrelated data and, as we are going to see soon, scaling.
Perceptron requires linearly separable samples to achieve convergence. The math of Perceptron.
If we represent samples as vectors of size n, where ‘n’ is the number of its features, a Perceptron can be modeled through the composition of two functions. The first one f(x) maps the input features ‘x’ vector to a scalar value, shifted by a bias ‘b’ f(x)=w.x+b
A threshold function, usually Heaviside or sign functions, maps the scalar value to a binary output:

$$t(x) = sgn(f(x)) = \begin{cases} 1 & \text{if } f(x) \geq 0 \\ -1 & \text{otherwise} \end{cases}$$

Indeed if the neuron output is exactly zero it cannot be assumed that the sample belongs to the first sample since it lies on the boundary between the two classes. Nonetheless for the sake of simplicity,ignore this situation.

ALGORITHM:

STEP 1: Importing the libraries
STEP 2:Importing the dataset
STEP 3:Plot the data to verify the linear separable dataset and consider only two classes
STEP 4:Convert the data set to scale the data to uniform range by using Feature scaling
STEP 4:Split the dataset for training and testing
STEP 5:Define the input vector ‘X’ from the training dataset
STEP 6:Define the desired output vector ‘Y’ scaled to +1 or -1 for two classes C1 and C2
STEP 7:Assign Initial Weight vector ‘W’ as 0 as the dimension of ‘X’ STEP 8:Assign the learning rate
STEP 9:For ‘N ‘ iterations ,do the following:
v(i) = w(i)*x(i)

```
    W (i+i)= W(i) + learning_rate*(y(i)-t(i))*x(i)<BR>
```

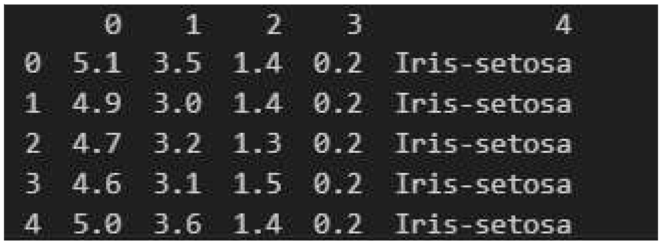
STEP 10:Plot the error for each iteration
STEP 11:Print the accuracy

PROGRAM:

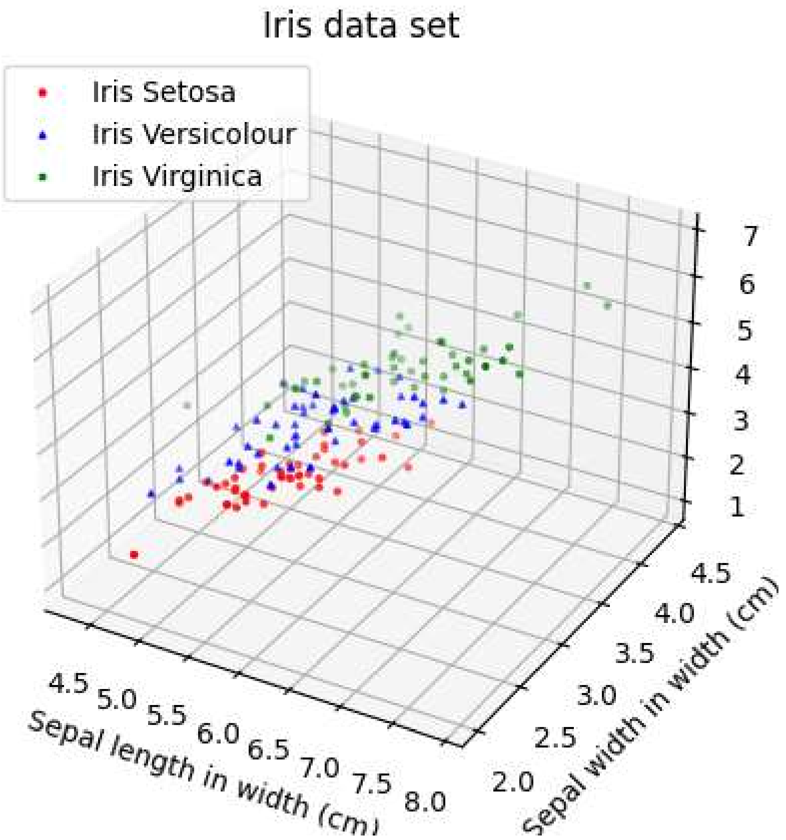
```
from sklearn.model_selection import train_test_split
import numpy as np
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.metrics import accuracy_score

class Perceptron:
  def __init__(self, input_size, learning_rate = 0.1):
    self.weights = np.zeros(input_size)
    self.bias = 0
    self.learning_rate = learning_rate
    self.misclassified = []

  def predict_probs(self, x):
    z = np.dot(x, self.weights) + self.bias
    return z

  def predict(self,x):
    probs = self.predict_probs(x)
    return np.where(probs>=0,1,-1)

  def fit(self, x_train, y_train, epochs):
    for epoch in range(epochs):
        err = 0
        for i in range(len(x_train)):
          x = x_train[i]
          y = y_train[i]

          y_pred = self.predict(x)
          error = y - y_pred

          self.weights += self.learning_rate*error*x
          self.bias += self.learning_rate*error

          err+=int(error!=0)
        self.misclassified.append(err)

# Start your main here ,read the iris data set
url = 'https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data'
df = pd.read_csv(url, header=None)
print(df.head())

y = df.iloc[:, 4].values
x = df.iloc[:, 0:3].values

import matplotlib.pyplot as plt
fig = plt.figure()
ax = plt.axes(projection='3d')
ax.set_title('Iris data set')
ax.set_xlabel("Sepal length in width (cm)")
ax.set_ylabel("Sepal width in width (cm)")
ax.set_zlabel("Petal length in width (cm)")

ax.scatter(x[:50, 0], x[:50, 1], x[:50, 2], color='red',
         marker='o', s=4, edgecolor='red', label="Iris Setosa")
ax.scatter(x[50:100, 0], x[50:100, 1], x[50:100, 2], color='blue',
         marker='^', s=4, edgecolor='blue', label="Iris Versicolour")
ax.scatter(x[100:150, 0], x[100:150, 1], x[100:150, 2], color='green',
         marker='x', s=4, edgecolor='green', label="Iris Virginica")
plt.legend(loc='upper left')
plt.show()

x = x[0:100, 0:2]
y = y[0:100]
plt.scatter(x[:50, 0], x[:50, 1], color='red', marker='o', label='Setosa')
plt.scatter(x[50:100, 0], x[50:100, 1], color='blue', marker='x',
          label='Versicolour')
plt.xlabel("Sepal length")
plt.ylabel("Petal length")
plt.legend(loc='upper left')
plt.show()

y = np.where(y == 'Iris-setosa', 1, -1)
x[:, 0] = (x[:, 0] - x[:, 0].mean()) / x[:, 0].std()
x[:, 1] = (x[:, 1] - x[:, 1].mean()) / x[:, 1].std()

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.25,random_state=0)

classifier = Perceptron(x_train.shape[1], learning_rate=0.01)
classifier.fit(x_train, y_train, epochs=10)
print("accuracy", accuracy_score(classifier.predict(x_test), y_test)*100)
```

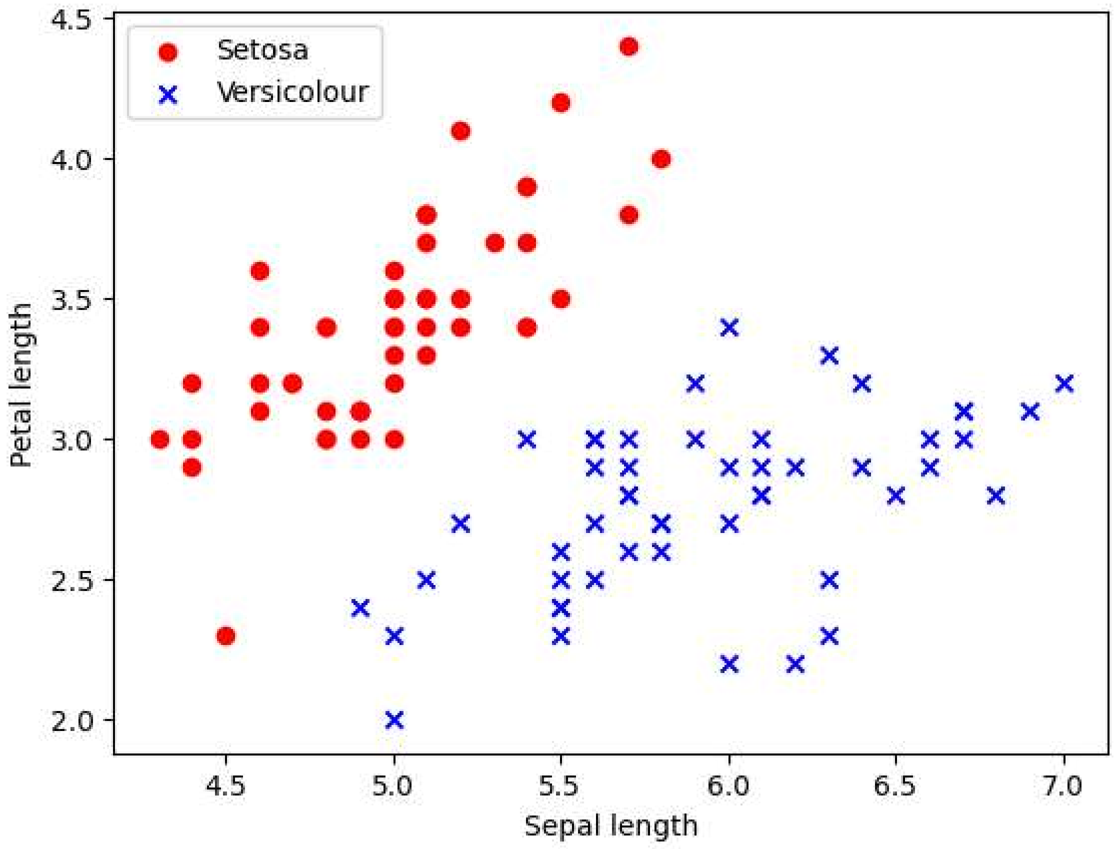
```
plt.plot(range(1, len(classifier.misclassified) + 1),classifier.misclassified, marker='o')
plt.xlabel('Epoch')
plt.ylabel('Errors')
plt.show()
```

OUTPUT:

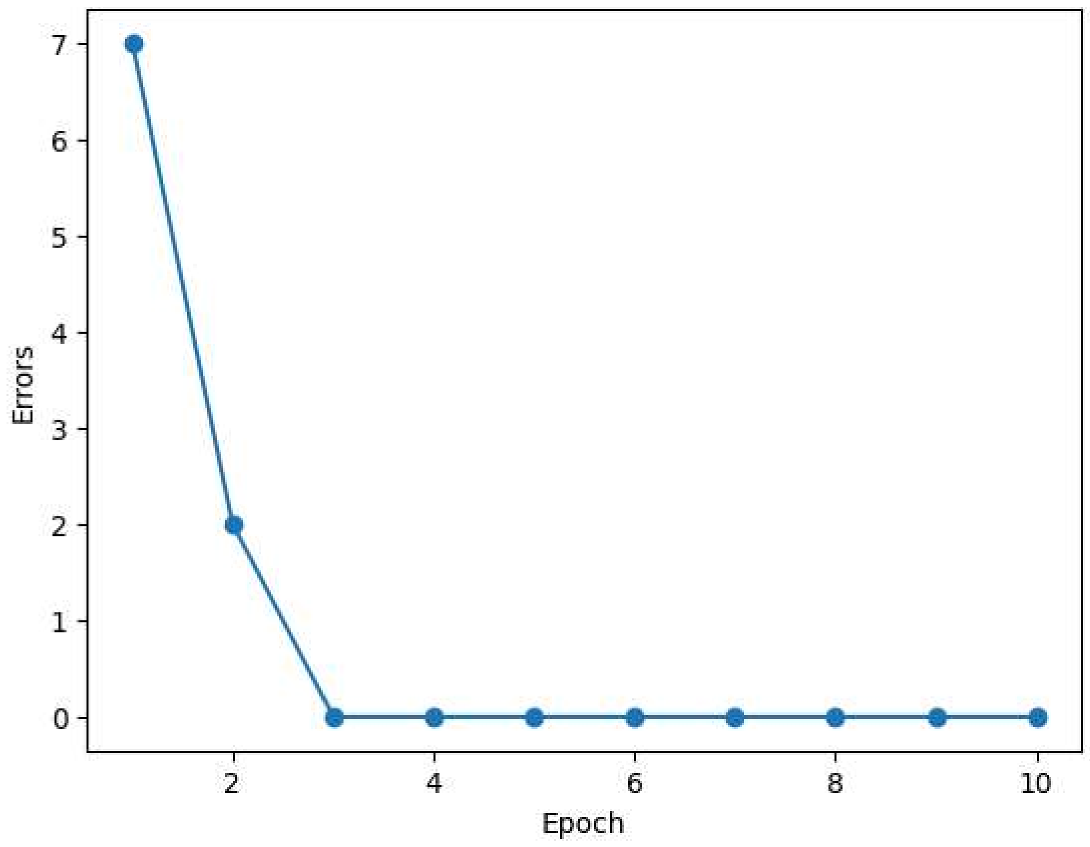


```
     0    1    2    3            4
0  5.1  3.5  1.4  0.2  Iris-setosa
1  4.9  3.0  1.4  0.2  Iris-setosa
2  4.7  3.2  1.3  0.2  Iris-setosa
3  4.6  3.1  1.5  0.2  Iris-setosa
4  5.0  3.6  1.4  0.2  Iris-setosa
```





```
accuracy 100.0
```



RESULT:

Thus, a single layer perceptron model is implemented using python to classify Iris data set.