```
import  numpy as  np
import  pandas  as  pd
import seaborn as sns
from sklearn.preprocessing import LabelEncoder
data=pd.read_csv('/content/drive/MyDrive/iris (1).csv')
data.columns=['Sepal_len_cm','Sepal_wid_cm','Petal_len_cm','Petal_wid_cm','Type']
data.head(10)
```

	Sepal_len_cm	Sepal_wid_cm	Petal_len_cm	Petal_wid_cm	Type
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0
5	5.4	3.9	1.7	0.4	0
6	4.6	3.4	1.4	0.3	0
7	5.0	3.4	1.5	0.2	0
8	4.4	2.9	1.4	0.2	0
9	4.9	3.1	1.5	0.1	0

```
def activation_func(value):
    return((np.exp(value)-np.exp(-value))/(np.exp(value)+np.exp(-value)))
```

```
def perceptron_train(in_data,labels,alpha):
    X=np.array(in_data)
    y=np.array(labels)
    weights=np.random.random(X.shape[1])
    original=weights
    bias=np.random.random_sample()
    for key in  range(X.shape[0]):
        a=activation_func(np.matmul(np.transpose(weights),X[key]))
        yn=0
        if a>=0.7:
             yn=1
        elif a<=(-0.7):
             yn=-1
        weights=weights+alpha*(yn-y[key])*X[key]
        print('Iteration '+str(key)+': '+str(weights))
    print('Difference: '+str(weights-original))
    return weights
```

```
def perceptron_test(in_data,label_shape,weights):
    X=np.array(in_data)
    y=np.zeros(label_shape)
    for key in  range(X.shape[1]):
        a=activation_func((weights*X[key]).sum())
        y[key]=0
        if a>=0.7:
          y[key]=1
        elif a<=(-0.7):
          y[key]=-1
    return y
```

```
def score(result,labels):
    difference=result-np.array(labels)
    correct_ctr=0
    for elem in  range(difference.shape[0]):
        if difference[elem]==0:
            correct_ctr+.1
    score=correct_ctr*100/difference. size
    print('Score.'+str(score))
```

```
divider = np.random.rand(len(data)) < 0.70
d_train=data[divider]
d_test=data[~divider]
```

```
# Dividing d_train  into data  and  labels/targets

d_train_y=d_train['Type']
d_train_X=d_train.drop(['Type'],axis=1)

# Dividing d_train  into data  and  labels/targets

d_test_y=d_test['Type']
d_test_X=d_test.drop(['Type'],axis=1)

# Learning rate

alpha = 0.01

# Train

weights = perceptron_train(d_train_X, d_train_y, alpha)

# Test

result_test=perceptron_test(d_test_X,d_test_y.shape,weights)
```

```
Iteration 50: [2.11489686 1.77681228 0.53878592 0.17757496]
Iteration 51: [2.11489686 1.77681228 0.53878592 0.17757496]
Iteration 52: [2.11489686 1.77681228 0.53878592 0.17757496]
Iteration 53: [2.11489686 1.77681228 0.53878592 0.17757496]
Iteration 54: [2.11489686 1.77681228 0.53878592 0.17757496]
Iteration 55: [2.11489686 1.77681228 0.53878592 0.17757496]
Iteration 56: [2.11489686 1.77681228 0.53878592 0.17757496]
Iteration 57: [2.11489686 1.77681228 0.53878592 0.17757496]
Iteration 58: [2.11489686 1.77681228 0.53878592 0.17757496]
Iteration 59: [2.11489686 1.77681228 0.53878592 0.17757496]
Iteration 60: [2.11489686 1.77681228 0.53878592 0.17757496]
Iteration 61: [2.11489686 1.77681228 0.53878592 0.17757496]
Iteration 62: [2.11489686 1.77681228 0.53878592 0.17757496]
Iteration 63: [2.11489686 1.77681228 0.53878592 0.17757496]
Iteration 64: [2.11489686 1.77681228 0.53878592 0.17757496]
Iteration 65: [2.11489686 1.77681228 0.53878592 0.17757496]
Iteration 66: [2.11489686 1.77681228 0.53878592 0.17757496]
Iteration 67: [2.05189686 1.74381228 0.47878592 0.15257496]
Iteration 68: [1.99389686 1.71681228 0.42778592 0.13357496]
Iteration 69: [1.92289686 1.68681228 0.36878592 0.11257496]
Iteration 70: [1.85989686 1.65781228 0.31278592 0.09457496]
Iteration 71: [1.79489686 1.62781228 0.25478592 0.07257496]
Iteration 72: [1.74589686 1.60281228 0.20978592 0.05557496]
Iteration 73: [1.67289686 1.57381228 0.14678592 0.03757496]
Iteration 74: [1.60089686 1.53781228 0.08578592 0.01257496]
Iteration 75: [ 1.53689686  1.51081228  0.03278592 -0.00642504]
Iteration 76: [ 1.47889686  1.48281228 -0.01821408 -0.03042504]
Iteration 77: [ 1.41489686  1.45081228 -0.07121408 -0.05342504]
Iteration 78: [ 1.34989686  1.42081228 -0.12621408 -0.07142504]
Iteration 79: [ 1.27289686  1.38281228 -0.19321408 -0.09342504]
Iteration 80: [ 1.20389686  1.35081228 -0.25021408 -0.11642504]
Iteration 81: [ 1.14789686  1.32281228 -0.29921408 -0.13642504]
Iteration 82: [ 1.07089686  1.29481228 -0.36621408 -0.15642504]
Iteration 83: [ 1.00789686  1.26781228 -0.41521408 -0.17442504]
Iteration 84: [ 0.94089686  1.23481228 -0.47221408 -0.19542504]
Iteration 85: [ 0.86889686  1.20281228 -0.53221408 -0.21342504]
Iteration 86: [ 0.80689686  1.17481228 -0.58021408 -0.23142504]
Iteration 87: [ 0.74289686  1.14681228 -0.63621408 -0.25242504]
Iteration 88: [ 0.67089686  1.11681228 -0.69421408 -0.26842504]
Iteration 89: [ 0.59689686  1.08881228 -0.75521408 -0.28742504]
Iteration 90: [ 0.51789686  1.05081228 -0.81921408 -0.30742504]
Iteration 91: [ 0.45389686  1.02281228 -0.87521408 -0.32942504]
Iteration 92: [ 0.32789686  0.96681228 -0.97721408 -0.35942504]
Iteration 93: [ 0.14489686  0.88881228 -1.14521408 -0.40142504]
Iteration 94: [-0.08610314  0.79881228 -1.32821408 -0.47042504]
Iteration 95: [-0.27510314  0.69681228 -1.49621408 -0.54242504]
Iteration 96: [-0.46710314  0.60381228 -1.66121408 -0.59642504]
Iteration 97: [-0.66810314  0.51081228 -1.82921408 -0.66842504]
Iteration 98: [-0.87510314  0.41781228 -1.98221408 -0.73742504]
Iteration 99: [-1.04910314  0.33681228 -2.13521408 -0.79442504]
Iteration 100: [-1.25310314  0.24081228 -2.31221408 -0.86342504]
Iteration 101: [-1.45410314  0.14181228 -2.48321408 -0.93842504]
Iteration 102: [-1.65510314  0.05181228 -2.63921408 -1.00742504]
Iteration 103: [-1.84410314 -0.02318772 -2.78921408 -1.06442504]
Iteration 104: [-2.03910314 -0.11318772 -2.94521408 -1.12442504]
Iteration 105: [-2.22510314 -0.21518772 -3.10721408 -1.19342504]
Iteration 106: [-2.40210314 -0.30518772 -3.26021408 -1.24742504]
Difference: [-2.782 -0.895 -3.289 -1.346]
```

```
# Test

result_test=perceptron_test(d_test_X,d_test_y.shape,weights)

# Calculate score

score(result_test,d_test_y)
```

```
Score.0.0
```