Execution:-

1. SLP for IRIS

```python
import pandas as pd
df = pd.read_csv('https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data', header=None)
df.tail()
```

[1] ✓ 2.3s

	0	1	2	3	4
145	6.7	3.0	5.2	2.3	Iris-virginica
146	6.3	2.5	5.0	1.9	Iris-virginica
147	6.5	3.0	5.2	2.0	Iris-virginica
148	6.2	3.4	5.4	2.3	Iris-virginica
149	5.9	3.0	5.1	1.8	Iris-virginica

```python
df[:5]
```

[2] ✓ 0.9s

	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

```python
import matplotlib.pyplot as plt
import numpy as np
y = df.iloc[0:100, 4].values
y
```

[3] ✓ 0.6s

```
array(['Iris-setosa', 'Iris-setosa', 'Iris-setosa', 'Iris-setosa',
       'Iris-setosa', 'Iris-setosa', 'Iris-setosa', 'Iris-setosa',
       'Iris-setosa', 'Iris-setosa', 'Iris-setosa', 'Iris-setosa',
       'Iris-setosa', 'Iris-setosa', 'Iris-setosa', 'Iris-setosa',
       'Iris-setosa', 'Iris-setosa', 'Iris-setosa', 'Iris-setosa',
```

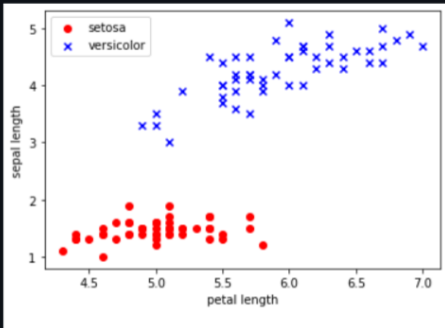
```python
y = np.where(y == 'Iris-setosa', -1, 1)
y
```

✓ 0.6s

```
array([-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,  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])
```

```python
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='versicolor')
plt.xlabel('petal length')
plt.ylabel('sepal length')
plt.legend(loc='upper left')
plt.show()
```

[6] ✓ 0.3s



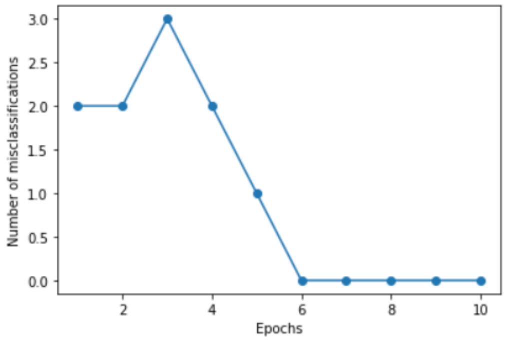
```python
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size = 0.2)

vector = np.vectorize(float)
X_train = vector(X_train)
X_train
```
[7] ✓ 1.2s

```
array([[5.5, 3.7],
       [5. , 3.5],
       [4.6, 1.4],
```

Perceptron:-

```python
import numpy as np

class Perceptron(object):
    def __init__(self, rate = 0.01, niter = 10):
        self.rate = rate
        self.niter = niter

    def fit(self, X, y):
        self.weight = np.zeros(1 + X.shape[1])
        self.errors = []

        for i in range(self.niter):
            err = 0
            for xi, target in zip(X, y):
                delta_w = self.rate * (target - self.predict(xi))
                self.weight[1:] += delta_w * xi
                self.weight[0] += delta_w
                err += int(delta_w ≠ 0.0)
            self.errors.append(err)
        return self

    def net_input(self, X):
        return np.dot(X, self.weight[1:]) + self.weight[0]

    def predict(self, X):
        return np.where(self.net_input(X) ≥ 0, 1, -1)
```
[8] ✓ 0.3s

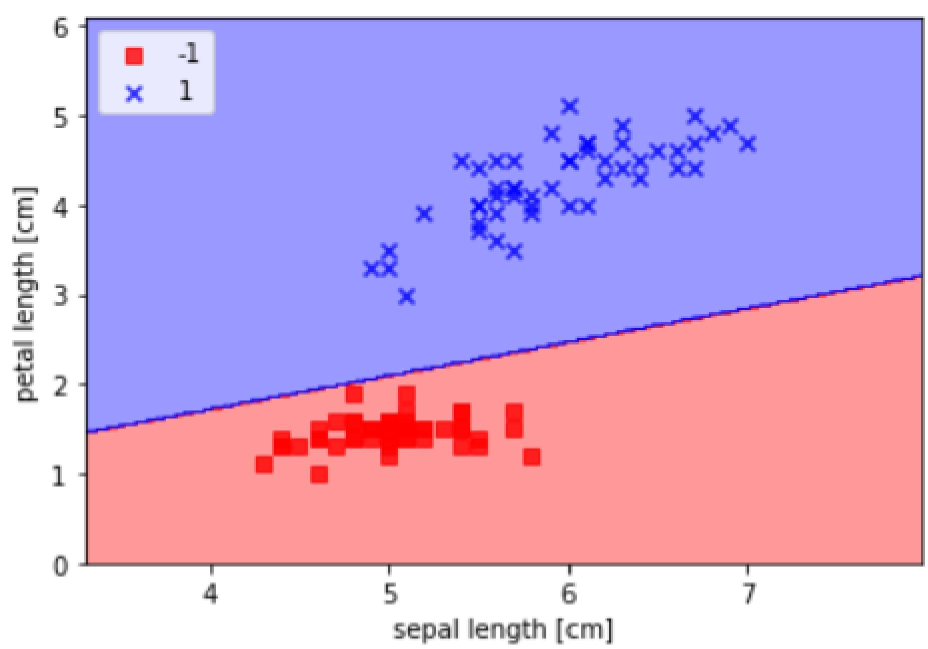
```python
pn = Perceptron(0.1, 10)
pn.fit(X, y)
plt.plot(range(1, len(pn.errors) + 1), pn.errors, marker='o')
plt.xlabel('Epochs')
plt.ylabel('Number of misclassifications')
plt.show()
```
[9] ✓ 0.2s



```python
from matplotlib.colors import ListedColormap
def plot_decision_regions(X, y, classifier, resolution=0.02):
   markers = ('s', 'x', 'o', '^', 'v')
   colors = ('red', 'blue', 'lightgreen', 'gray', 'cyan')
   cmap = ListedColormap(colors[:len(np.unique(y))])

   x1_min, x1_max = X[:,  0].min() - 1, X[:, 0].max() + 1
   x2_min, x2_max = X[:, 1].min() - 1, X[:, 1].max() + 1
   xx1, xx2 = np.meshgrid(np.arange(x1_min, x1_max, resolution),
   np.arange(x2_min, x2_max, resolution))
   Z = classifier.predict(np.array([xx1.ravel(), xx2.ravel()]).T)
   Z = Z.reshape(xx1.shape)
   plt.contourf(xx1, xx2, Z, alpha=0.4, cmap=cmap)
   plt.xlim(xx1.min(), xx1.max())
   plt.ylim(xx2.min(), xx2.max())

   for idx, cl in enumerate(np.unique(y)):
      plt.scatter(x=X[y == cl, 0], y=X[y == cl, 1],
      alpha=0.8, c=cmap(idx),
      marker=markers[idx], label=cl)
```
[10] 0.3s

```python
plot_decision_regions(X, y, classifier=pn)
plt.xlabel('sepal length [cm]')
plt.ylabel('petal length [cm]')
plt.legend(loc='upper left')
plt.show()
```
[11] 0.2s

Separation of each class:-



The core of the MLP network:-

```
import numpy as np

class mlp:
    """A Multi-Layer Perceptron"""

    def __init__(
        self, inputs, targets, nhidden, beta=1, momentum=0.9, outtype="logistic"
    ):
        """Constructor"""
        self.nin = np.shape(inputs)[1]
        self.nout = np.shape(targets)[1]
        self.ndata = np.shape(inputs)[0]
        self.nhidden = nhidden

        self.beta = beta
        self.momentum = momentum
        self.outtype = outtype

        # Initialise network
        self.weights1 = (
            (np.random.rand(self.nin + 1, self.nhidden) - 0.5) * 2 / np.sqrt(self.nin)
        )
        self.weights2 = (
            (np.random.rand(self.nhidden + 1, self.nout) - 0.5)
            * 2
            / np.sqrt(self.nhidden)
        )

    def earlystopping(self, inputs, targets, valid, validtargets, eta, niterations=100):

        valid = np.concatenate((valid, -np.ones((np.shape(valid)[0], 1))), axis=1)

        old_val_error1 = 100002
        old_val_error2 = 100001
        new_val_error = 100000

        count = 0
        while ((old_val_error1 - new_val_error) > 0.001) or (
            (old_val_error2 - old_val_error1) > 0.001
        ):
            count += 1
            print(count)
            self.mlptrain(inputs, targets, eta, niterations)
            old_val_error2 = old_val_error1
            old_val_error1 = new_val_error
            validout = self.mlpfwd(valid)
            new_val_error = 0.5 * np.sum((validtargets - validout) ** 2)

        print("Stopped", new_val_error, old_val_error1, old_val_error2)
        return new_val_error

    def mlptrain(self, inputs, targets, eta, niterations):
        """Train the neural network"""
        # Add the inputs that match the bias node
        inputs = np.concatenate((inputs, -np.ones((self.ndata, 1))), axis=1)
        change = range(self.ndata)

        updatew1 = np.zeros((np.shape(self.weights1)))
        updatew2 = np.zeros((np.shape(self.weights2)))

        for n in range(niterations):

            self.outputs = self.mlpfwd(inputs)
```

```python
            error = 0.5 * np.sum((self.outputs - targets) ** 2)
            if np.mod(n, 100) == 0:
                print("Iteration: ", n, " Error: ", error)

            # Different types of output neurons and their activation functions
            if self.outtype == "linear":
                deltao = (self.outputs - targets) / self.ndata
            elif self.outtype == "logistic":
                deltao = (
                    self.beta
                    * (self.outputs - targets)
                    * self.outputs
                    * (1.0 - self.outputs)
                )
            elif self.outtype == "softmax":
                deltao = (
                    (self.outputs - targets)
                    * (self.outputs * (-self.outputs) + self.outputs)
                    / self.ndata
                )
            else:
                print("error")

            # hidden network delta
            deltah = (
                self.hidden
                * self.beta
                * (1.0 - self.hidden)
                * (np.dot(deltao, np.transpose(self.weights2)))
            )

            updatew1 = (
                eta * (np.dot(np.transpose(inputs), deltah[:, :-1]))
                + self.momentum * updatew1
            )
            updatew2 = (
                eta * (np.dot(np.transpose(self.hidden), deltao))
                + self.momentum * updatew2
            )
            self.weights1 -= updatew1
            self.weights2 -= updatew2

    def mlpfwd(self, inputs):
        """Run the network forward"""

        self.hidden = np.dot(inputs, self.weights1)
        self.hidden = 1.0 / (1.0 + np.exp(-self.beta * self.hidden))
        self.hidden = np.concatenate(
            (self.hidden, -np.ones((np.shape(inputs)[0], 1))), axis=1
        )

        outputs = np.dot(self.hidden, self.weights2)

        # Different types of output neurons
        if self.outtype == "linear":
            return outputs
        elif self.outtype == "logistic":
            return 1.0 / (1.0 + np.exp(-self.beta * outputs))
        elif self.outtype == "softmax":
            normalisers = np.sum(np.exp(outputs), axis=1) * np.ones(
                (1, np.shape(outputs)[0])
            )
            return np.transpose(np.transpose(np.exp(outputs)) / normalisers)
        else:
            print("error")
```

```python
    def confmat(self, inputs, targets):
        """Confusion matrix"""

        # Add the inputs that match the bias node
        inputs = np.concatenate((inputs, -np.ones((np.shape(inputs)[0], 1))), axis=1)
        outputs = self.mlpfwd(inputs)

        nclasses = np.shape(targets)[1]

        if nclasses == 1:
            nclasses = 2
            outputs = np.where(outputs > 0.5, 1, 0)
        else:
            # 1-of-N encoding
            outputs = np.argmax(outputs, 1)
            targets = np.argmax(targets, 1)

        cm = np.zeros((nclasses, nclasses))
        for i in range(nclasses):
            for j in range(nclasses):
                cm[i, j] = np.sum(
                    np.where(outputs == i, 1, 0) * np.where(targets == j, 1, 0)
                )
        output = cm
        print("Confusion matrix is:")
        print(cm)
        print("Percentage Correct: ", np.trace(cm) / np.sum(cm) * 100)
        return output
```

2) XOR MLP IMPLEMENTATION

```python
import numpy as np
import mlp
xordata = np.array([[0,0,0],[0,1,1],[1,0,1],[1,1,0]])

q = mlp.mlp(xordata[:,0:2],xordata[:,2:3],2,outtype='logistic')
q.mlptrain(xordata[:,0:2],xordata[:,2:3],0.25,501)
q.confmat(xordata[:,0:2],xordata[:,2:3])
```

```
[2]   ✓ 0.9s
...  Iteration:  0  Error:  0.5695708881870863
     Iteration:  100  Error:  0.47075890231399664
     Iteration:  200  Error:  0.255898635745315
     Iteration:  300  Error:  0.015897228160342988
     Iteration:  400  Error:  0.006786036804098226
     Iteration:  500  Error:  0.0042563267936051915
     Confusion matrix is:
     [[2. 0.]
      [0. 2.]]
     Percentage Correct:  100.0
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         2
           1       1.00      1.00      1.00         2

    accuracy                           1.00         4
   macro avg       1.00      1.00      1.00         4
weighted avg       1.00      1.00      1.00         4
```

3. IRIS DATASET:-

```python
# replacing class names with numbers
def preprocessIris(infile,outfile):

    stext1 = 'Iris-setosa'
    stext2 = 'Iris-versicolor'
    stext3 = 'Iris-virginica'
    rtext1 = '0'
    rtext2 = '1'
    rtext3 = '2'

    fid = open(infile,"r")
    oid = open(outfile,"w")

    for s in fid:
        if s.find(stext1)>-1:
            oid.write(s.replace(stext1, rtext1))
        elif s.find(stext2)>-1:
            oid.write(s.replace(stext2, rtext2))
        elif s.find(stext3)>-1:
            oid.write(s.replace(stext3, rtext3))
    fid.close()
    oid.close()
```

[20] ✓ 0.3s

```python
import numpy as np
preprocessIris('iris.data.csv','iris_proc.data')

#loading data
iris = np.loadtxt('iris_proc.data',delimiter=',')
#normalizing the input
iris[:,:4] = iris[:,:4]-iris[:,:4].mean(axis=0)
imax = np.concatenate((iris.max(axis=0)*np.ones((1,5)),np.abs(iris.min(axis=0)*np.ones((1,5)))),axis=0).max(axis=0)
iris[:,:4] = iris[:,:4]/imax[:4]

#sample output
print (iris[0:5,:])
```

[21] ✓ 0.4s

```
[[-0.36142626  0.33135215 -0.7508489  -0.76741803  0.        ]
 [-0.45867099 -0.04011887 -0.7508489  -0.76741803  0.        ]
 [-0.55591572  0.10846954 -0.78268251 -0.76741803  0.        ]
 [-0.60453809  0.03417533 -0.71901528 -0.76741803  0.        ]
 [-0.41004862  0.40564636 -0.7508489  -0.76741803  0.        ]]
```

```python
# Split into training, validation, and test sets
target = np.zeros((np.shape(iris)[0],3));
indices = np.where(iris[:,4]==0)
target[indices,0] = 1
indices = np.where(iris[:,4]==1)
target[indices,1] = 1
indices = np.where(iris[:,4]==2)
target[indices,2] = 1

# Randomly order the data
order = np.arange(np.shape(iris)[0])

np.random.shuffle(order)
iris = iris[order,:]
target = target[order,:]

train = iris[::2,0:4]
traint = target[::2]
valid = iris[1::4,0:4]
validt = target[1::4]
test = iris[3::4,0:4]
testt = target[3::4]

print (train.max(axis=0), train.min(axis=0))
```

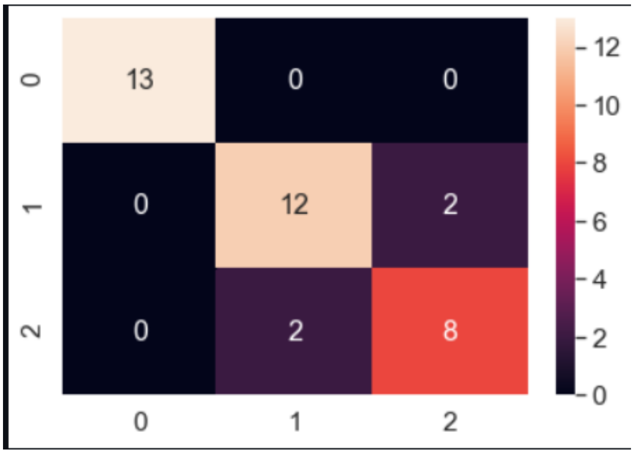
[36] ✓ 0.6s

```
[1.         0.55423477 0.93633277 1.        ] [-0.75040519 -0.78306092 -0.87818336 -0.8442623 ]
```

```python
# Train the network
import MLP as mlp
net = mlp.mlp(train,traint,5,outtype='logistic')
net.earlystopping(train,traint,valid,validt,0.1)
cm = net.confmat(test,testt)
```

[38] ✓ 0.8s

```
1
Iteration:  0  Error:  27.890093397433485
2
Iteration:  0  Error:  0.161357486453l9908
3
Iteration:  0  Error:  0.0757257709096788
Stopped 0.8519153765474519 0.7991526000943426 0.6804791234401425
Confusion matrix is:
[[10.  0.  0.]
 [ 0. 12.  1.]
 [ 0.  1. 13.]]
Percentage Correct:  94.5945945945946
```



Performance metrics:-

```
from sklearn.metrics import classification_report
targets=testt
inputs = np.concatenate((test, -np.ones((np.shape(test)[0], 1))), axis=1)
nclasses = np.shape(targets)[1]
output = net.mlpfwd(inputs)
if nclasses == 1:
    nclasses = 2
    output = np.where(output > 0.5, 1, 0)
else:
    # 1-of-N encoding
    output = np.argmax(output, 1)
    targets = np.argmax(targets, 1)


print(classification_report(targets, output))
```

[54] ✓ 0.4s

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        10
           1       0.92      0.92      0.92        13
           2       0.93      0.93      0.93        14

    accuracy                           0.95        37
   macro avg       0.95      0.95      0.95        37
weighted avg       0.95      0.95      0.95        37
```