**Name: Rina Kailas Mali**

**Roll No: 95**

**Practical No: 13.2**

**Practical Name: Classification Of Iris Dataset By Applying Artificial Neural Network**

**With Back-Propogation Algorithm**

# classification of iris data set by aplying artificial neural network using Back-propogation algorithm  
**import** numpy **as** np  
**import** pandas **as** pd  
**from** sklearn.datasets **import** load\_iris  
**from** sklearn.model\_selection **import** train\_test\_split  
**import** matplotlib.pyplot **as** plt  
  
# load dataset  
data = load\_iris()  
  
# Get features and target  
x = data.data  
y = data.target  
print(**"Y="**, y)  
  
y = pd.get\_dummies(y).values  
print(y[:3])  
  
# split data into train and test data  
x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size=20, random\_state=4)  
  
# initialize variable  
learning\_rate = 0.1  
iteration = 6000  
N = y\_train.size  
  
# number of input features  
input\_size = 4  
  
# number of hidden layers neurons  
hidden\_size = 2  
  
# mo. of neurons at output layers  
output\_size = 3  
results = pd.DataFrame(columns=[**"mse"**, **"accuracy"**])  
  
# initialize weights  
np.random.seed(10)  
# initialiizing weight for the hidden layers  
W1 = np.random.normal(scale=0.5, size=(input\_size, hidden\_size))  
print(**"weight 1"**, W1)  
  
# initializing weight for the output layers  
W2 = np.random.normal(scale=0.5, size=(hidden\_size, output\_size))  
print(**"weight 2"**, W2)  
  
  
**def** sigmoid(x):  
 **return** 1/(1 + np.exp(-x))  
  
  
**def** mean\_squared\_error(y\_pred, y\_true):  
 **return** (((y\_pred - y\_true) \*\* 2).sum()) / (2 \* y\_pred.size)  
  
  
**def** accuracy(y\_pred, y\_true):  
 acc = y\_pred.argmax(axis=1) == y\_true.argmax(axis=1)  
 **return** acc.mean()  
  
  
**for** itr **in** range(iteration):  
  
 # feedforward propagation  
 # on hidden layer  
 Z1 = np.dot(x\_train, W1)  
 A1 = sigmoid(Z1)  
  
# on output layer  
 Z2 = np.dot(A1, W2)  
 A2 = sigmoid(Z2)  
  
# calculating error  
 mse = mean\_squared\_error(A2, y\_train)  
 acc = accuracy(A2, y\_train)  
 results = results.\_append({**"mse"**: mse, **"accuracy"**: acc}, ignore\_index=**True**)  
  
# backpropagation  
 E1 = A2 - y\_train  
 dw1 = E1 \* A2 \* (1 - A2)  
  
 E2 = np.dot(dw1, W2.T)  
 dw2 = E2 \* A1 \* (1 - A1)  
  
# weight updates  
 W2\_update = np.dot(A1.T, dw1) / N  
 W1\_update = np.dot(x\_train.T, dw2) / N  
  
 W2 = W2 - learning\_rate \* W2\_update  
 W1 = W1 - learning\_rate \* W1\_update  
  
results.mse.plot(title=**"Mean squared Error"**)  
  
results.accuracy.plot(title=**"Accuracy"**)  
  
# feedforward  
Z1 = np.dot(x\_test, W1)  
A1 = sigmoid(Z1)  
  
Z2 = np.dot(A1, W2)  
A2 = sigmoid(Z2)  
  
acc = accuracy(A2, y\_test)  
print(**"Accuracy: {}"**.format(acc))

**OUTPUT:**

C:\Users\sejal\MCA-I\_ML\Scripts\python.exe C:/Users/sejal/PycharmProjects/MCA-I\_ML/nural\_network\_Backpropa\_algo.py

Y= [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 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]

[[ True False False]

[ True False False]

[ True False False]]

weight 1 [[ 0.66579325 0.35763949]

[-0.77270015 -0.00419192]

[ 0.31066799 -0.36004278]

[ 0.13275579 0.05427426]]

weight 2 [[ 0.00214572 -0.08730011 0.21651309]

[ 0.60151869 -0.48253284 0.51413704]]