```
In [429]:
           1 # Importing Libraries
              import numpy as np
              imnort nandas as nd
In [430]:
           1 #reading .csv file and storing it in "dataset" dataframe
            2 dataset=nd read csv("/home/ankit/Deskton/Tris csv")
           1 print(dataset.head()) # printing the whole dataset
In [431]:
           2 nrint(dataset['Species'] unique()) # printing the unique classes of iris da
             Id SepalLengthCm SepalWidthCm PetalLengthCm PetalWidthCm
          0
                           5.1
                                         3.5
                                                         1.4
              1
                                                                       0.2 Iris-setosa
          1
              2
                           4.9
                                         3.0
                                                         1.4
                                                                       0.2 Iris-setosa
          2
              3
                           4.7
                                          3.2
                                                         1.3
                                                                       0.2
                                                                            Iris-setosa
                                                                       0.2 Iris-setosa
          3
              4
                           4.6
                                         3.1
                                                         1.5
                                                                       0.2 Iris-setosa
              5
                           5.0
                                         3.6
                                                         1.4
          ['Iris-setosa'
                         'Iris-versicolor' 'Iris-virginica']
In [432]: 1 nrint(dataset shape) # size of the dataset
          (150.6)
In [4331:
           1 | #X= dataset.drop('Species',axis=1)
              #Y= dataset['Snecies']
In [434]: 1 #nrint(X)
In [435]: 1 #nrint(Y)
In [436]: 1 data = dataset to numpy() # storing iris dataset as numpy array
In [437]:
           1 for i in range(150):
                  data[i][0]=1 # adding 1st column as bias=1
In [438]: 1 nrint(data)
          [[1 5.1 3.5 1.4 0.2 'Iris-setosa']
           [1 4.9 3.0 1.4 0.2 'Iris-setosa']
           [1 4.7 3.2 1.3 0.2 'Iris-setosa']
           [1 4.6 3.1 1.5 0.2 'Iris-setosa']
           [1 5.0 3.6 1.4 0.2 'Iris-setosa'
           [1 5.4 3.9 1.7 0.4 'Iris-setosa']
           [1 4.6 3.4 1.4 0.3 'Iris-setosa']
           [1 5.0 3.4 1.5 0.2 'Iris-setosa']
           [1 4.4 2.9 1.4 0.2 'Iris-setosa']
           [1 4.9 3.1 1.5 0.1 'Iris-setosa']
           [1 5.4 3.7 1.5 0.2 'Iris-setosa']
           [1 4.8 3.4 1.6 0.2 'Iris-setosa
           [1 4.8 3.0 1.4 0.1 'Iris-setosa']
           [1 4.3 3.0 1.1 0.1 'Iris-setosa']
           [1 5.8 4.0 1.2 0.2 'Iris-setosa']
           [1 5.7 4.4 1.5 0.4 'Iris-setosa']
           [1 5.4 3.9 1.3 0.4 'Iris-setosa']
           [1 5.1 3.5 1.4 0.3 'Iris-setosa'
           [1 5.7 3.8 1.7 0.3 'Iris-setosa']
                1 2 0 1 5 0 2 17-2- ----
In [439]:
              from matplotlib import pyplot as plt
           3
           4
              for i in range(len(data)):
           5
                  if data[i][-1]=='Iris-setosa':
                      data[i][-1]=1 # relabeling Iris setosa class as "0"
           6
                  if data[i][-1]=='Iris-versicolor':
           7
           8
                      data[i][-1]=2 # relabeling Iris versicolor class as "1"
```

```
9
                    if data[i][-1]=='Iris-virginica':
            10
                        data[i][-1]=3 # relabeling Iris virginica class as "0"
                    data[i][0]=1 # initializing first column as 1 for hias
In [440]:
            1 #data
In [441]:
            1
               for i in range(len(data)):
            2
                    if data[i][-1]==1:
            3
                        plt.scatter(data[i,1],data[i,2],c='r')
            4
                    if data[i][-1]==2:
            5
                        plt.scatter(data[i,1],data[i,2],c='g')
            6
                    if data[i][-1]==3:
            7
                        plt.scatter(data[i,1],data[i,2],c='y')
            8
                    plt.xlabel('sepal length')
            9
                    plt.ylabel('sepal width')
           10
                               # a nlot between senal length and senal width
           11
              4.0
            width
             3.5
             3.0
              2.5
              2.0
                    4.5
                          5.0
                               5.5
                                    6.0
                                          6.5
                                               7.0
                                                     7.5
                                                          8.0
                                   sepal length
In [442]:
               for i in range(len(data)):
             2
                    if data[i][-1]==1:
            3
                        plt.scatter(data[i,3],data[i,4],c='r')
            4
                    if data[i][-1]==2:
            5
                        plt.scatter(data[i,3],data[i,4],c='g')
            6
                    if data[i][-1]==3:
                        plt.scatter(data[i,3],data[i,4],c='y')
            8
                    plt.xlabel('petal length')
                    plt.ylabel('petal width')
            9
           10
               nlt show() # a nlot between netal length and netal width
              2.5
              2.0
            width
             1.5
            petal v
              0.5
              0.0
                                      4
                                                    6
                                   petal length
```

Implementing Forward Propagation MLP for Iris dataset

```
In [443]: 1 input_no_of_nodes = 4 # as features are 4-dimensional i.e. sepal length,pet
2 # and petal width
3 no_of_hidden_layers = 1 # just for simplicity , I am taking one hidden layer
```

```
no of classes = 3 # setosa, virginica and versicolor
            5
              # initializing input weights for output layer
            6
            8 weight1 = [] # initializing weights
            a
           10
           11 for k in range(3):
           12
                  w=[]
           13
                  for j in range(5):
           14
                       w.append(np.random.randn())
           15
                  weight1.append(w)
           16
           17 print("input weight vector for output layer:\n ",weight1)
           18
           19 # initializing input weights for hidden layer
           20
           21 weight2 = [] # initializing weights
           22
           23 bias= 1
           24
           25
           26
           27
              for k in range(4):
           28
                   w=[]
           29
                   for j in range(5):
           30
                       w.append(np.random.randn())
           31
                  weight2.append(w)
           32
           33 print("\ninput weight vector for hidden laver\\n " weight?)
          input weight vector for output layer:
            [[0.18204651248854653, -0.3195494800785854, 0.8506967596147637, -1.527883060]
          1839293, 0.8468775949912822], [-0.7105553426733896, 1.5427545553212112, -0.521
          2290329153958, 1.3199837146608846, 1.3027364165285165], [-1.2581738482758533,
          0.1954909533767402, -0.5293114051138316, 0.8982568826527799, -1.26789205684944
          4]]
          input weight vector for hidden layer:
            [[0.012079880287338243, -1.0994786150270182, 0.6706515214739192, 0.696574387]
          9098967, 0.874415795996476], [1.0953417415957751, 2.2584676519018334, 0.326125
          3541951244, 0.32904108755751343, 0.5200729353112302], [1.1755338637157928, 0.1
          0241462477627439, 0.9286554122455126, 0.6679682394666533, -0.223888796096033
          7], [0.309703588072365, -1.740283841682843, -0.2776661070260824, 1.73557043354
          72464, -0.9442540729403734]]
In [444]:
           1 #for dk(p), if class label belongs to class 1, we make 1st value as 1 and d
            2
              #others
            3
            4
              labels=[]
            5
              for i in range(150):
            6
                   if data[i][-1]==1:
            7
                       labels.append([1,0,0])
            8
                  if data[i][-1]==2:
            9
                       labels.append([0,1,0])
           10
                   if data[i][-1]==3:
                       lahels annend([0 0 11)
In [445]:
                       forward prop output=[]
            1
            2
            3
                   # Input on input layer -1 (hidden layer ) by using McCullouch Pit mode
            4
            5
                       u1=[]
            6
                       for j in range(4):
            7
                           sum=0
            8
                           for i in range(5):
            9
                               sum += weight2[j][i]*data[j][i]
           10
                           ul.append(sum)
           11
                       #print("Input on layer 1 (hidden layer) : ",u)
```

```
12
13
             # For output on layer 1, by using sigmoid function,
14
15
            v1=[]
16
            v1.append(1) # appending bias=1
17
18
            for i in range(4):
                sig=1/(1+ np.exp(-u1[i]))
19
20
                v1.append(sig)
21
            #print("output on layer 1 : ",v)
22
23
24
            # For input on layer 2 by using McCullouch Pit model,
25
26
            u2=[]
            for j in range(3):
27
28
                sum=0
29
                for i in range(5):
30
                    sum += weight1[j][i]*v1[i]
31
                u2.append(sum)
               #print("input of layer 2",u)
32
33
34
35
            # For output on layer 2(output layer), by using sigmoid function,
36
37
            v2=[]
38
39
            for i in range(3):
40
                sig=1/(1+ np.exp(-u2[i]))
41
                v2.append(sig)
               #print("output on layer 2 (output layer) : ",v)
42
43
            forward_prop_output.append(v2)
44
            yk=forward_prop_output
45
```

In [446]:

```
def forward_prop(p):
 1
            forward_prop_output=[]
 2
 3
 4
        # Input on input layer -1 (hidden layer ) by using McCullouch Pit mode
 5
 6
            u1=[]
            for j in range(4):
 7
 8
                sum=0
 9
                for i in range(5):
10
                    sum += weight2[j][i]*data[p][i]
11
                ul.append(sum)
12
             #print("Input on layer 1 (hidden layer) : ",u)
13
14
             # For output on layer 1, by using sigmoid function,
15
16
17
            v1.append(1) # appending bias=1
18
19
            for i in range(4):
20
                sig=1/(1+ np.exp(-u1[i]))
21
                v1.append(sig)
22
            #print("output on layer 1 : ",v)
23
24
25
            # For input on layer 2 by using McCullouch Pit model,
26
27
            u2=[]
28
            for j in range(3):
29
                sum=0
30
                for i in range(5):
31
                    sum += weight1[j][i]*v1[i]
                u2.append(sum)
32
33
               #print("input of layer 2",u)
```

```
34
           35
                       # For output on layer 2(output layer), by using sigmoid function,
           36
           37
           38
                       v2=[]
           39
                       for i in range(3):
           40
           41
                           sig=1/(1+ np.exp(-u2[i]))
           42
                           v2.append(sig)
           43
                          #print("output on layer 2 (output layer) : ",v)
           44
                       forward prop output.append(v2)
           45
                       yk=forward prop output
           46
           47
                       return vk
In [447]:
               #for dk(p), if class label belongs to class 1, we make 1st value as 1 and d
            2
               #others
            3
            4
               labels=[]
            5
               for i in range(150):
                   if data[i][-1]==1:
            6
                       labels.append([1,0,0])
            7
            8
                   if data[i][-1]==2:
            9
                       labels.append([0,1,0])
                   if data[i][-1]==3:
           10
                       lahels.annend([0.0.11)
           11
In [448]: 1 lahels
Out[448]: [[1, 0, 0],
            [1, 0, 0],
            [1, 0, 0],
            [1, 0, 0],
            [1, 0, 0],
            [1, 0, 0],
            [1, 0, 0],
            [1, 0, 0],
            [1, 0, 0],
            [1, 0, 0],
            [1, 0, 0],
            [1, 0, 0],
            [1, 0, 0],
            [1, 0, 0],
            [1, 0, 0],
            [1, 0, 0],
            [1, 0, 0],
            [1, 0, 0],
            [1, 0, 0],
```

Implementing Backpropagation MLP for iris dataset

```
def back prop(p):
In [449]:
           1
                   # computing delta w_ji(0) (learning rule) #weight2
            2
            3
            4
                   learning_rate = 0.5 #assuming
            5
            6
            7
                   forward prop output=[]
            8
            9
                   # Input on input layer -1 (hidden layer ) by using McCullouch Pit mode
           10
           11
                   u1=[]
                   for j in range(4):
           12
           13
                       sum=0
           14
                       for i in range(5):
                           sum += weight2[j][i]*data[p][i]
           15
           16
                       ul.append(sum)
```

```
17
            #print("Input on layer 1 (hidden layer) : ",u)
18
            # For output on layer 1, by using sigmoid function,
19
20
21
        v1=[]
22
       v1.append(1) # appending bias=1
23
24
        for i in range(4):
25
            sig=1/(1+ np.exp(-u1[i]))
26
            v1.append(sig)
27
        #print("output on layer 1 : ",v)
28
29
30
        # For input on layer 2 by using McCullouch Pit model,
31
       u2=[]
32
       for j in range(3):
33
34
            sum=0
35
            for i in range(5):
36
                sum += weight1[j][i]*v1[i]
37
            u2.append(sum)
38
            #print("input of layer 2",u)
39
40
41
        # For output on layer 2(output layer), by using sigmoid function,
42
43
       v2=[]
44
45
        for i in range(3):
46
            sig=1/(1+ np.exp(-u2[i]))
47
            v2.append(sig)
48
            #print("output on layer 2 (output layer) : ",v)
49
        forward_prop_output.append(v2)
50
       yk=forward_prop_output
51
52
53
54
55
        for k in range(3):
56
            for j in range(5):
57
                delta w kj 1 = (labels[p][k] - yk[0][k])*(yk[0][k])*(1-yk[0][k]
58
59
        delta_w_kj_1=learning_rate*delta_w_kj_1
60
        #print(delta_w_kj_1)
61
62
63
        # computing delta w ji(0) (learning rule) #weight2
64
65
       learning_rate = 0.5 #assuming
66
67
68
        for k in range(3):
69
            for j in range(5):
70
                for l in range(5):
71
                    delta_w_{ji_0} = (labels[p][k] - yk[0][k])*yk[0][k]*(1-yk[0][k])
72
73
        delta_w_ji_0=learning_rate*delta_w_ji_0
        #print(delta_w_ji_0)
74
75
76
        # modifying weight1 and weight2 according to delta_w
77
        #print("previous weight1 matrix : ", weight1)
78
       temp1=0
79
        for k in range(3):
80
            for j in range(5):
81
                temp1 =weight1[k][j]
82
                weight1[k][j] =temp1+delta_w_kj_1
83
84
```

```
85
                   #print("modified input weight vector for output layer:\n ",weight1)
           86
           87
                   #print("Previous weight2 matrix : ", weight2)
           88
           89
                   temp2=0
           90
                   for k in range(4):
           91
                       for j in range(5):
           92
                           temp2=weight2[k][j]
           93
                           weight2[k][j] = temp2+ delta_w_ji_0
           94
           95
           96
                   #print("modified input weight vector for hidden layer:\n ",weight2)
           97
           98
                   return
           99
In [450]:
            1
              error1=0
            2
              for p in range(150):
            3
                  output1=forward prop(p)
            4
                   #print(output)
            5
                  for j in range(3):
            6
                      error1 += pow((labels[p][j] - output1[0][j]),2)
                   E1 = error1/2
                            #error after forward propagation initially
            2 print(F1)
          56.751504851834625
In [451]:
              for n in range(50):# number of iterations
           1
            2
                   error=0
            3
                   for i in range(150):
                       output=forward_prop(i)
            4
            5
                       #print(output)
            6
                       #error = 0
            7
                       for j in range(3):
            8
                           error += pow((labels[i][j] - output[0][j]),2)
            a
                       E = error/2
           10
                       #print(E)
                       #if E<0.1:
           11
           12
                            break
           13
                      back_prop(i)
           14
              print("Minimized error after 100 iteration: ", E) # Our objective was to Mi
           15
              # if error difference is not much then incearease the number of iteration b
          Minimized error after 100 iteration: 54.775238213652585
In [452]:
          1 print("Final input weight matrix for hidden layer: \n", weight2)
              nrint("\nFinal input weight matrix for output layer. \n" weight1)
          Final input weight matrix for hidden layer:
           [[-0.2431105734390827, \ -1.3546690687534335, \ 0.4154610677474975, \ 0.44138393418]
          347427, 0.6192253422700504], [0.8401512878693501, 2.003277198175414, 0.0709349
          0046870354, 0.07385063383109278, 0.26488248158481115], [0.9203434099893685, -
          0.15277582895014669, 0.673464958519087, 0.41277778574023155, -0.47907924982245
          27], [0.05451313434594388, -1.9954742954092584, -0.5328565607525014, 1.4803799
          79820831, -1.1994445266667888]]
          Final input weight matrix for output layer:
           [0.2529551252114036, -0.24864086735572838, 0.9216053723376181, -1.4569744474]
          610702, 0.9177862077141367], [-0.6396467299505352, 1.6136631680440703, -0.4503
          204201925382, 1.3908923273837437, 1.3736450292513755], [-1.1872652355529942,
          0.2663995660995988, -0.45840279239097464, 0.9691654953756343, -1.1969834441265
          85]]
```

Incorporating momentum factor

```
In [423]: 1 gamma=0.9
```

```
2 mom1=0
            3 mom2-0
In [424]:
               #Modifying little bit the definition of backpropagation for error learning
            1
            3
               def back_prop(p,mom1,mom2):
            4
                   # computing delta w_ji(0) (learning rule) #weight2
            5
            6
                   learning rate = 0.5 #assuming
            7
                   forward prop_output=[]
            8
            9
            10
                   # Input on input layer -1 (hidden layer ) by using McCullouch Pit mod
            11
            12
            13
                   for j in range(4):
            14
                        sum=0
            15
                        for i in range(5):
            16
                            sum += weight2[j][i]*data[p][i]
            17
                        ul.append(sum)
                    #print("Input on layer 1 (hidden layer) : ",u)
            18
            19
            20
                   # For output on layer 1, by using sigmoid function,
            21
                   v1=[]
            22
            23
                   v1.append(1) # appending bias=1
            24
            25
                   for i in range(4):
            26
                        sig=1/(1+ np.exp(-u1[i]))
            27
                       v1.append(sig)
            28
                   #print("output on layer 1 : ",v)
            29
            30
            31
                   # For input on layer 2 by using McCullouch Pit model,
            32
            33
                   u2=[]
                   for j in range(3):
            34
            35
                        sum=0
            36
                        for i in range(5):
            37
                            sum += weight1[j][i]*v1[i]
                        u2.append(sum)
            38
            39
                        #print("input of layer 2",u)
            40
            41
            42
                   # For output on layer 2(output layer), by using sigmoid function,
            43
            44
                   v2=[]
            45
                   for i in range(3):
            46
            47
                        sig=1/(1+ np.exp(-u2[i]))
            48
                        v2.append(sig)
            49
                        #print("output on layer 2 (output layer) : ",v)
            50
                   forward_prop_output.append(v2)
            51
                   yk=forward_prop_output
            52
            53
            54
                   for k in range(3):
            55
                        for j in range(5):
                            delta_w_kj_1 = (labels[p][k] - yk[0][k])*(yk[0][k])*(1-yk[0][k])
            56
            57
            58
                   delta_w_kj_1=learning_rate*delta_w_kj_1
            59
                   #print(delta_w_kj_1)
            60
            61
            62
                   # computing delta w ji(0) (learning rule) #weight2
            63
            64
                   learning_rate = 0.5 #assuming
            65
            66
```

In []: 1

```
67
                   for k in range(3):
           68
                       for j in range(5):
                            for l in range(5):
           69
                                delta_w_ji_0 = (labels[p][k] - yk[0][k])*yk[0][k]*(1-yk[0]
           70
           71
           72
                   delta_w_ji_0=learning_rate*delta_w_ji_0
           73
                   #print(delta_w_ji_0)
           74
           75
                   # modifying weight1 and weight2 according to delta_w
                   #print("previous weight1 matrix : ", weight1)
           76
           77
           78
                   # Here, I am using the concept of momentum factor
           79
                   mom1 += pow(gamma,i)*(delta_w_kj_1)
                   mom2 += pow(gamma,i)*(delta w ji 0)
           80
           81
           82
                   temp1=0
           83
                   for k in range(3):
           84
                       for j in range(5):
           85
                            temp1 =weight1[k][j]
           86
                            weight1[k][j] =temp1+(mom1 + delta_w_kj_1)
           87
           88
           89
                   #print("modified input weight vector for output layer:\n ",weight1)
           90
           91
                   #print("Previous weight2 matrix : ", weight2)
           92
           93
                   temp2=0
           94
                   for k in range(4):
           95
                       for j in range(5):
           96
                            temp2=weight2[k][j]
           97
                            weight2[k][j] = temp2+ (mom2 + delta_w_ji_0)
           98
           99
                   #print("modified input weight vector for hidden layer:\n ",weight2)
          100
          101
          102
                   return mom1, mom2
          103
In [425]:
            1
               for n in range(100):# number of iterations
            2
                   error=0
            3
                   for i in range(150):
            4
                       output=forward_prop(i)
            5
                       #print(output)
            6
                       #error = 0
            7
                       for j in range(3):
            8
                           error += pow((labels[i][j] - output[0][j]),2)
            9
                       E = error/2
           10
                       #print(E)
           11
                       #if E<0.1:
           12
                            break
           13
           14
                       back_prop(i,mom1,mom2)
           15
           16 print("Minimized error after 100 iteration: ", E) # Our objective was to Mi
           17
              # if error difference is not much then incearease the number of iteration b
```

9 of 9 07/10/20, 11:32 am

Minimized error after 100 iteration: 43.510786166850934