WEEK - 08 - SPOT - MLP

Repeat Question 1 using a dataset with multiple classes.

Using Glass dataset:-

Index of /ml/machine-learning-databases/glass (uci.edu)

	1	1.52101	13.64	4.49	1.10	71.78	0.06	8.75	0.00	0.00.1	1.1
0	2	1.51761	13.89	3.60	1.36	72.73	0.48	7.83	0.00	0.00	1
1	3	1.51618	13.53	3.55	1.54	72.99	0.39	7.78	0.00	0.00	1
2	4	1.51766	13.21	3.69	1.29	72.61	0.57	8.22	0.00	0.00	1
3	5	1.51742	13.27	3.62	1.24	73.08	0.55	8.07	0.00	0.00	1
4	6	1.51596	12.79	3.61	1.62	72.97	0.64	8.07	0.00	0.26	1
...	...	...	...	...	...	...	...	...	...	...	...
208	210	1.51623	14.14	0.00	2.88	72.61	0.08	9.18	1.06	0.00	7
209	211	1.51685	14.92	0.00	1.99	73.06	0.00	8.40	1.59	0.00	7
210	212	1.52065	14.36	0.00	2.02	73.42	0.00	8.44	1.64	0.00	7
211	213	1.51651	14.38	0.00	1.94	73.61	0.00	8.48	1.57	0.00	7
212	214	1.51711	14.23	0.00	2.08	73.36	0.00	8.62	1.67	0.00	7

213 rows × 11 columns

Using MLP code from execution:-

loading data

```
#loading data
dataset = np.loadtxt('glass.conv.csv',delimiter=',')
no_of_columns = 9
#normalizing the input
dataset[:,:no_of_columns] = dataset[:,:no_of_columns]-dataset[:,:no_of_columns].mean(axis=0)
imax = np.concatenate((dataset.max(axis=0)
       *np.ones((1,no_of_columns+1)),np.abs(dataset.min(axis=0)
       *np.ones((1,no_of_columns+1)))),axis=0).max(axis=0)
dataset[:,:no_of_columns] = dataset[:,:no_of_columns]/imax[:no_of_columns]
```

[93] ✓ 0.1s Python

```
print(dataset.shape)
```

[94] ✓ 0.9s Python

```
(214, 10)
```

Changing targets to 2d array (if target is 3, it will now become [0 0 0 1 0 0 0])

```
# Split into training, validation, and test sets
target = np.zeros((np.shape(dataset)[0],7))
indices = np.where(dataset[:,no_of_columns]==0)
target[indices,0] = 1
indices = np.where(dataset[:,no_of_columns]==1)
target[indices,1] = 1
indices = np.where(dataset[:,no_of_columns]==2)
target[indices,2] = 1
indices = np.where(dataset[:,no_of_columns]==3)
target[indices,3] = 1
indices = np.where(dataset[:,no_of_columns]==4)
target[indices,4] = 1
indices = np.where(dataset[:,no_of_columns]==5)
target[indices,5] = 1
indices = np.where(dataset[:,no_of_columns]==6)
target[indices,6] = 1
```

[95] ✓ 0.9s Python

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

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

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

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

6] ✓ 0.1s Python

```
[0.60422959 0.59971295 0.4527668  1.         0.89022304 1.
 0.82994696 0.68066725 0.62471632] [-0.45329979 -0.60114818 -1.         -0.45492167 -0.97184025 -0.08700524
 -0.48761847 -0.05884016 -0.12585104]
```

Find the number of neurons

using sigmoid function

```
print("Using Sigmoid function")
acc = np.zeros(50)
err = np.zeros(50)
cm = []
net = []
for i in range(50):
    net.append(mlp(train,traint,i+1, outtype='logistic'))
    err[i] = net[i].earlystopping(train,traint,valid,validt,0.1)
    #err = net.mlptrain(train, traint, 0.25, 10000)
    cm.append(net[i].confmat(test,testt))
    acc[i]= np.trace(cm[i]) / np.sum(cm[i]) * 100
```

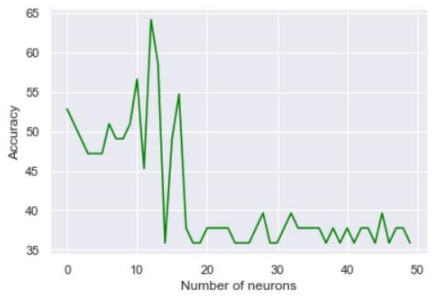
[114] ✓ 6.7s Python

```
Using Sigmoid function
No. of neurons in hidden layers =  1
Stopped, error =  14.84916789098454
Percentage Correct:  52.83018867924528
No. of neurons in hidden layers =  2
Stopped, error =  13.056349942005875
Percentage Correct:  50.943396226415096
No. of neurons in hidden layers =  3
Stopped, error =  13.676924693348425
Percentage Correct:  49.056603773584904
```

The accuracy varies very heavily with respect to the number of neurons

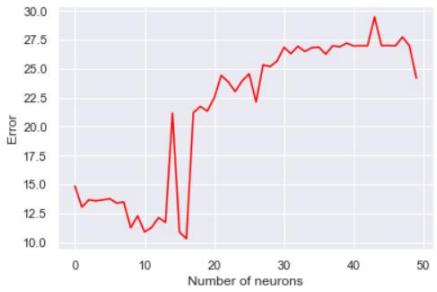
```
plt.xlabel("Number of neurons")
plt.ylabel("Accuracy")
plt.plot(acc, color='green')
plt.show()
```

[115] ✓ 0.5s Python




```
plt.xlabel("Number of neurons")
plt.ylabel("Error")
plt.plot(err, color='red')
plt.show()
```

[116] ✓ 0.7s Python




```
n = np.argmax(acc)
print("Number of neurons for maximum accuracy =", n)
print("Accuracy = ", acc[n])
```

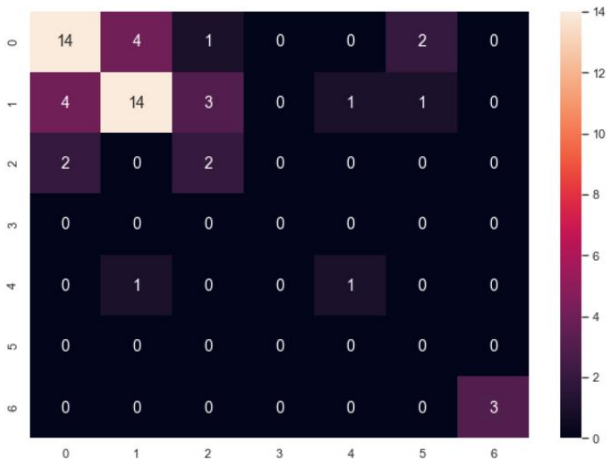
[117] ✓ 0.1s Python

```
Number of neurons for maximum accuracy = 12
Accuracy =  64.15094339622641
```

```
print("Using sigmoid function")
displayConfusionMatrix(cm[n],plt)
plt.show()
```

[119] ✓ 1.6s Python

```
Using sigmoid function
```



The model performs okish to classify multi class data.

```
print("Classification report using sigmoid activation function")
printClassificationReport(net[n], test, testt)
```

[120] ✓ 0.1s Python

```
Classification report using sigmoid activation function
              precision    recall  f1-score   support

           0       0.67      0.70      0.68        20
           1       0.61      0.74      0.67        19
           2       0.50      0.33      0.40         6
           4       0.50      0.50      0.50         2
           5       0.00      0.00      0.00         3
           6       1.00      1.00      1.00         3

    accuracy                           0.64        53
   macro avg       0.55      0.55      0.54        53
weighted avg       0.60      0.64      0.62        53
```

Checking performance with other activation functions:-

softmax activation function

```
print("Using Softmax activation function")
softnet = mlp(train,traint,n, outtype='softmax')
softnet.earlystopping(train,traint,valid,validt,0.15)
softcm = softnet.confmat(test, testt)
```

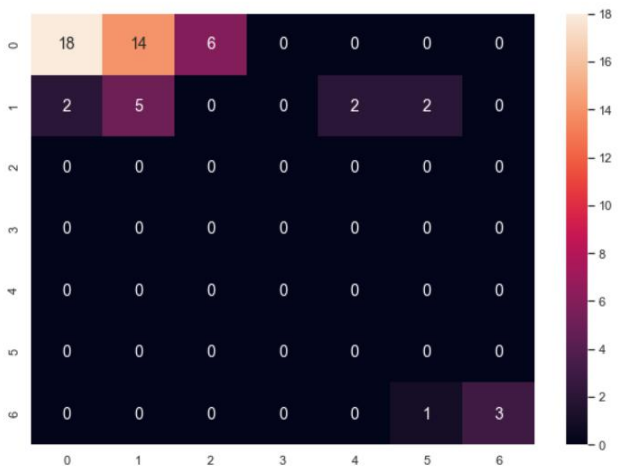
[144] ✓ 0.8s Python

```
Using Softmax activation function
No. of neurons in hidden layers =  12
Stopped, error =  15.30869622180197
Percentage Correct:  49.056603773584904
```

```
print("Using softmax function")
displayConfusionMatrix(softcm,plt)
plt.show()
```

[145] ✓ 2.7s Python

```
Using softmax function
```




```
print("Classification report using Softmax activation function")
printClassificationReport(softnet, test, testt)
```

[146] ✓ 0.1s Python

```
Classification report using Softmax activation function
              precision    recall  f1-score   support

           0       0.47      0.90      0.62        20
           1       0.45      0.26      0.33        19
           2       0.00      0.00      0.00         6
           4       0.00      0.00      0.00         2
           5       0.00      0.00      0.00         3
           6       0.75      1.00      0.86         3

    accuracy                           0.49        53
   macro avg       0.28      0.36      0.30        53
weighted avg       0.38      0.49      0.40        53
```

Linear activation function

```
print("Using Linear activation function")
signet = mlp(train,traint,n, outtype='linear')
signet.earlystopping(train,traint,valid,validt,0.15)
sigcm = signet.confmat(test, testt)
```

[153] ✓ 0.3s Python

```
Using Linear activation function
No. of neurons in hidden layers =  12
Stopped, error =  15.038230534429555
Percentage Correct:  49.056603773584904
```

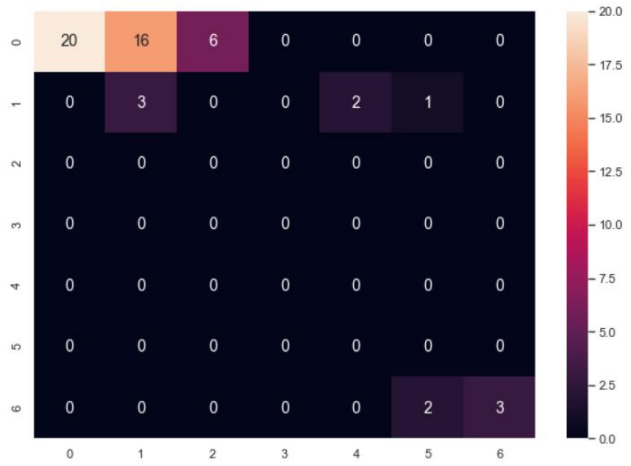
```
print("Using linear activation function")
displayConfusionMatrix(sigcm,plt)
plt.show()
```

[154] ✓ 1.3s Python


```
print("Using linear activation function")
displayConfusionMatrix(sigcm,plt)
plt.show()
```

[154] ✓ 1.3s Python


```
Using linear activation function
```




```
print("Classification report using linear activation function")
printClassificationReport(signet, test, testt)
```

[155] ✓ 0.1s Python

```
Classification report using linear activation function
              precision    recall  f1-score   support

           0       0.48      1.00      0.65        20
           1       0.50      0.16      0.24        19
           2       0.00      0.00      0.00         6
           4       0.00      0.00      0.00         2
           5       0.00      0.00      0.00         3
           6       0.60      1.00      0.75         3

    accuracy                           0.49        53
   macro avg       0.26      0.36      0.27        53
weighted avg       0.39      0.49      0.37        53
```

Tanh activation function

```
print("Using tanh activation function")
tanhnet = mlp(train,traint,n, outtype='tanh')
#tanhnet.mlptrain(train, traint, 0.25, 20000)
tanhnet.earlystopping(train,traint,valid,validt,0.001)
tanhcm = tanhnet.confmat(test, testt)
```

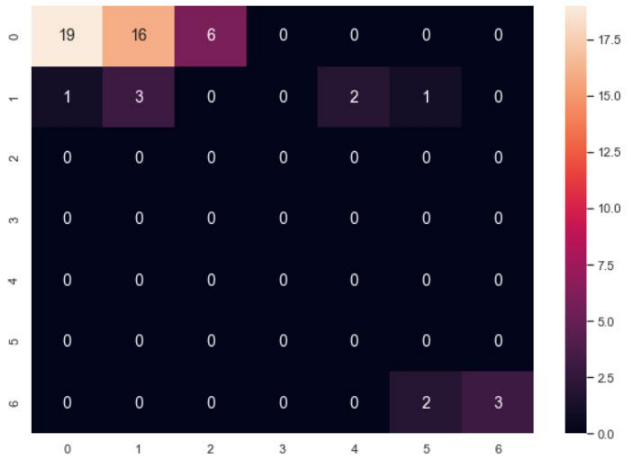
[171] ✓ 0.3s Python

```
Using tanh activation function
No. of neurons in hidden layers =  12
Stopped, error =  15.501622909344988
Percentage Correct:  47.16981132075472
```

```
print("Using tanh activation function")
displayConfusionMatrix(tanhcm,plt)
plt.show()
```

[172] ✓ 1.1s Python

```
Using tanh activation function
```




```
print("Classification report using tanh activation function")
printClassificationReport(tanhnet, test, testt)
```

[173] ✓ 0.1s Python

```
Classification report using tanh activation function
              precision    recall  f1-score   support

           0       0.46      0.95      0.62        20
           1       0.43      0.16      0.23        19
           2       0.00      0.00      0.00         6
           4       0.00      0.00      0.00         2
           5       0.00      0.00      0.00         3
           6       0.60      1.00      0.75         3

    accuracy                           0.47        53
   macro avg       0.25      0.35      0.27        53
weighted avg       0.36      0.47      0.36        53
```

As we can see, all the activation functions worked poorly, but Sigmoid gave the best results amongst all. Using example of more data and perhaps more number of layers, could have significantly improved the accuracy.