CSE 574: Introduction to Machine Learning

Project 3: Classification

By

Manishkumarreddy Jarugu

#50206843

Introduction:

This project is to implement a number classification algorithm, that recognizes a 28 x 28 grayscale handwritten digit images and identify it as a digit among 0,1,2…,9. Here we have implemented following three models on MNIST data: (1) Logistic regression, (2) single hidden layer neural network, (3) convolutional neural network. And finally, (4) the trained models are tested on USPS data.

(1) Logistic regression:

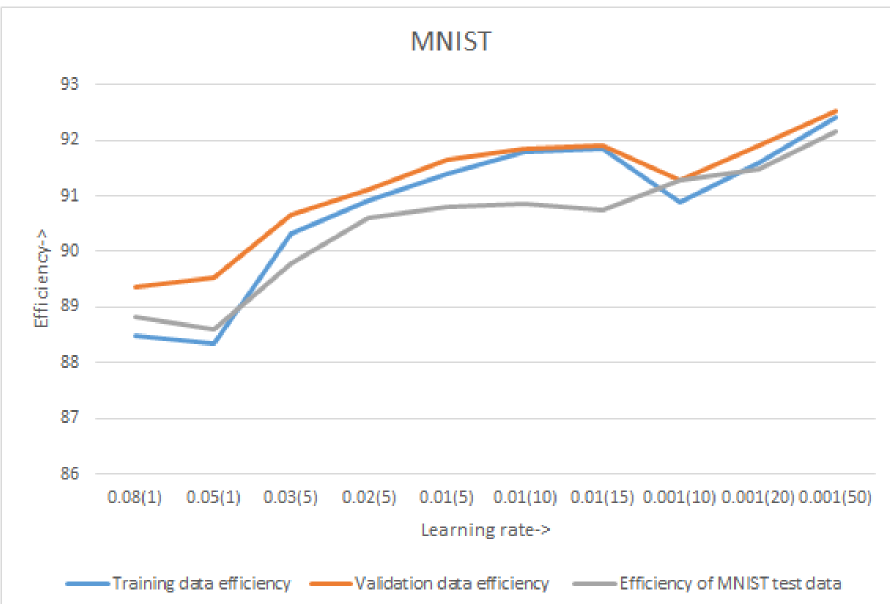
The logistic regression was implemented as mentioned in the appendix. First the data is being loaded and portioned into training, validation and test data. The model is trained on the training data set and being validated on the validated test data set and tested on the test set. The corresponding implementation of various equations in python is shown below:

```
w=np.random.random((len(inp_training_data[0]), N))
b=np.random.random((len(inp_training_data), N))
etta=0.001

for j in range(10):
    for x in range(0, len(inp_training_data)):
        ak=np.dot(inp_training_data[x],w)+b[x]
        y=np.exp(ak)/np.sum(np.exp(ak))
        inp=opt_training_data[x]
        tl=[]
        for xy in range(0,N):
            if(inp==xy):
                tl.append(1)
            else:
                tl.append(0)
        ttemp=np.array(tl)
        E=np.outer(inp_training_data[x],y - ttemp)
        w=w-E*etta

def computationLR(w1,inputt,outputt):
    sim = 0
    for x in range(0, len(inputt)):
        ak1 = np.dot(inputt[x], w1) + b[x]
        y1 = np.exp(ak1) / np.sum(np.exp(ak1))
        if(np.argmax(y1)==outputt[x]):
                sim+=1
    return sim
```

First the model is trained on MNIST data. The model efficiency is being improved by tuning the hyperparameters like learning rate and number of iterations. Changing both the parameters change the value of accuracy/efficiency. The different values for which the learning rate and the number of iterations are being changed are tabulated and shown in the below table:

Learning rate (η)	Number of iterations	Training data efficiency	Validation data efficiency	Efficiency of MNIST test data
0.08	1	88.496	89.36	88.82
0.05	1	88.348	89.53	88.6
0.03	5	90.334	90.65	89.8
0.02	5	90.918	91.11	90.62
0.01	5	91.408	91.66	90.8
0.01	10	91.802	91.84	90.87
0.01	15	91.86	91.92	90.74
0.001	10	90.884	91.28	91.29
0.001	20	91.602	91.91	91.47
0.001	50	92.422	92.54	92.17

A graph showing the efficiency/ accuracy curve for the MNIST data set is shown below:



The efficiency steadily increases as the number of iterations increases and as the learning rate decreases.

(2) Single Layer Neural Network:

The single layer neural network is implemented using the formulas given in the appendix 2.

The implementation of the single layer neural network in python is shown below:

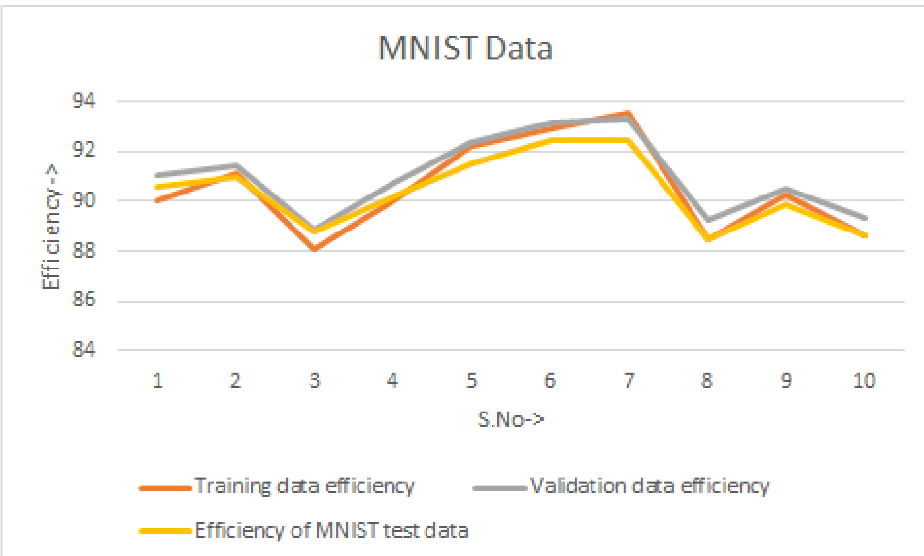
```
neural_w1=np.random.random((len(inp_training_data[0]), M))*0.1
neural_w2=np.random.random((M, N))*0.1
neural_b1=np.random.random((len(inp_training_data), M))
neural_b2=np.random.random((len(inp_training_data), N))
neural_etta=0.001
for i in range(10):
    print i
    for a in range(len(inp_training_data)):
        neural_zjh=inp_training_data[a].dot(neural_w1)+neural_b1[a]
        neural_zj=float(1)/(1+np.exp(-neural_zjh))
        neural_ak=neural_zj.dot(neural_w2)+neural_b2[a]
        neural_yk=np.exp(neural_ak)/np.sum(np.exp(neural_ak))
        neural_inp = opt_training_data[a]
        neural_tl=[]
        for xy in range(0,N):
            if(neural_inp==xy):
                neural_tl.append(1)
            else:
                neural_tl.append(0)
        neural_ttemp=np.array(neural_tl)
        delk=neural_yk-neural_ttemp
        h_delj=neural_zj.dot(1-neural_zj)
        delj=h_delj*(neural_w2.dot(delk))
        delE1=np.outer(inp_training_data[a],delj)
        delE2=np.outer(delk,neural_zj)
        neural_w1-=neural_etta*delE1
        neural_w2-=neural_etta*delE2.transpose()

def neuralSim(neural_w1,neural_w2,inp_training_data,opt_training_data,neural_b1,neural_b2):
    sim = 0
    for a in range(len(inp_training_data)):
        neural_zjh = inp_training_data[a].dot(neural_w1) + neural_b1[a]
        neural_zj = float(1) / (1 + np.exp(-neural_zjh))
        neural_ak = neural_zj.dot(neural_w2) + neural_b2[a]
        neuralMNIST _yk = np.exp(neural_ak) / np.sum(np.exp(neural_ak))
        if (np.argmax(neural_yk) == opt_training_data[a]):
            sim+=1
    return sim
```

Three parameters are being considered. Changing any of the one parameter change the value of accuracy. We change all the three parameters in such a way to increase the efficiency and the values are tabulated below:

S.No	Hidden Layers	Learning rate (η)	Number of iterations	Training data efficiency	Validation data efficiency	Efficiency of MNIST test data
1	50	0.01	1	90.05	91.02	90.57
2	50	0.01	10	91.138	91.41	90.98

3	100	0.05	1	88.06	88.84	88.83
4	100	0.01	1	90.024	90.72	90.21
5	100	0.01	10	92.238	92.37	91.55
6	100	0.001	20	92.888	93.13	92.49
7	100	0.001	50	93.574	93.34	92.45
8	800	0.01	1	88.496	89.26	88.5
9	800	0.01	10	90.252	90.5	89.85
10	1000	0.01	1	88.606	89.34	88.6

A graph depicting the above table is shown below:



Here there are three parameters to be considered: the number of hidden layers, learning rate and the number of iterations. Changing all the three parameters change the value of accuracy and the maximum accuracy observed for the set of results is approximately 92.5.

(3) Convolutional neural network:

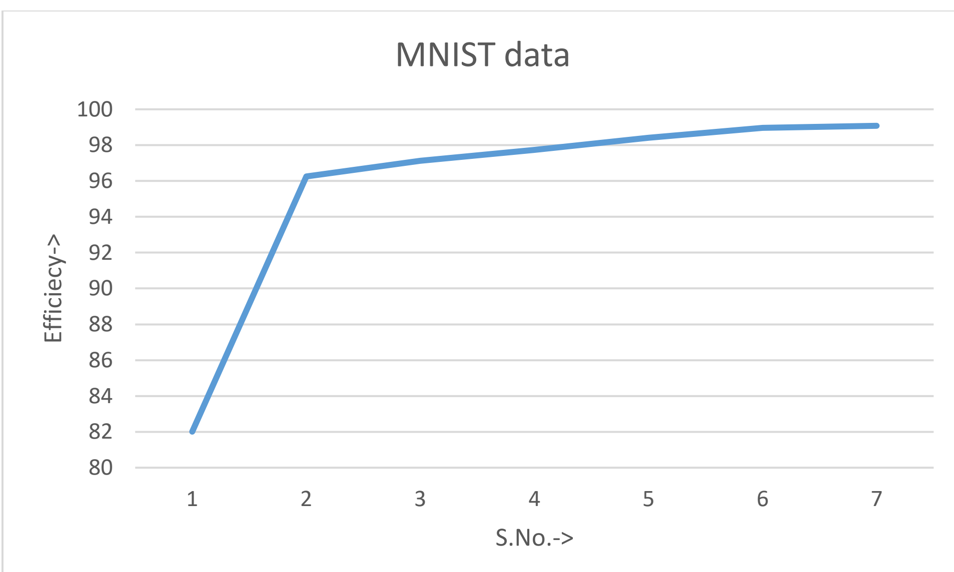
The dataset is trained on publicly available neural network package. We used tensor flow for this. Here there is only one parameter, that's being considered, number of iterations.

The implementation for this is done in python, which is almost same as the publicly available neural network package.

By changing the number of iterations, the efficiency / accuracy of the model was made to increase

S.no	No. of iterations	Efficiency of MNIST data
1	100	82.01
2	1000	96.26
3	1500	97.13
4	3000	97.73
5	5000	98.41
6	10000	98.97
7	20000	99.08

As the number of iterations increase, the value of the accuracy increases and the model could obtain a maximum efficiency of 99.08 while doing 20,000 iterations. A graph depicting the above table is as follow:



(4) Testing MNIST trained model on USPS test data:

The model obtained by training on MNIST data are being verified for the USPS data set. The USPS data images don't have the format of 28 x 28 pixels, so we are converting into the pixel range, normalize it and do the same set of evaluation on the USPS data set. The code for loading the USPS is shown below:

Code for loading USPS data:

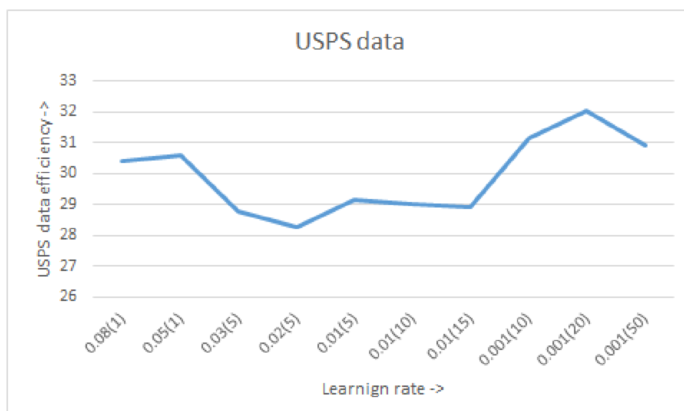
```
usps_test_x = []
usps_test_t = []
from PIL import Image
import glob
for i in range(10):
    for imgall in glob.glob('Numerals/'+str(i)+'/*.png'):
        img = Image.open(imgall)
        new_width = 28
        new_height = 28
        img = img.resize((new_width, new_height))
        pix = 1 - (np.array(list(img.getdata()))/255)
        usps_test_x.append(pix)
        usps_test_t.append(i)
```

<u>Logistic regression:</u>

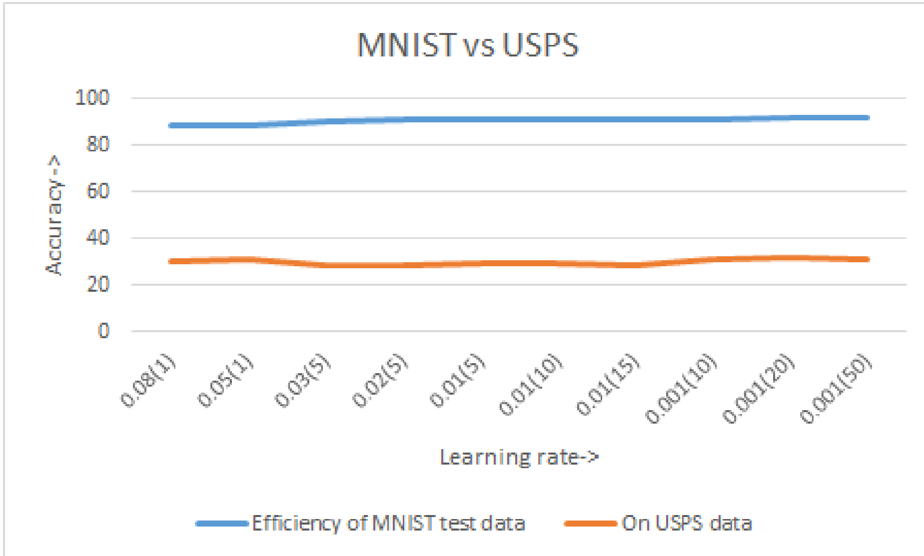
The MNIST trained data is tested on the USPS data, the different values that are obtained are shown below

The different values obtained for different iterations are shown below:

Learning rate (η)	Number of iterations	On USPS data
0.08	1	30.4265213261
0.05	1	30.6015300765
0.03	5	28.7614380719
0.02	5	28.2814140707
0.01	5	29.1364568228
0.01	10	28.9964498225
0.01	15	28.9014450723
0.001	10	31.1565578279
0.001	20	32.0316015801
0.001	50	30.9015450773



Comparing the test accuracies for the MNIST data and USPS data for logistic regression:

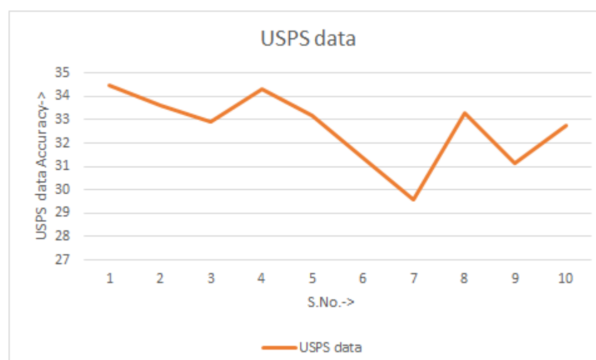


The USPS accuracy is very much less when compared to the accuracy of the MNIST test data supporting the “No free Lunch” theorem.

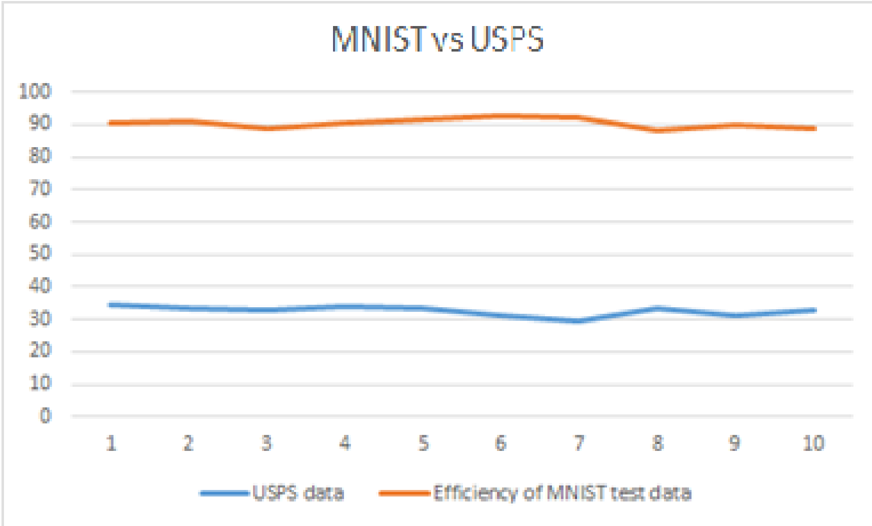
<u>Single Layer Neural Network:</u>

The MNIST trained single layer neural network model is tested on the USPS data, the values obtained for different parameters are shown below:

S.No.	Hidden Layers	Learning rate (η)	Number of iterations	USPS data
1	50	0.01	1	34.4867243362
2	50	0.01	10	33.591679584
3	100	0.05	1	32.8966448322
4	100	0.01	1	34.2967148357
5	100	0.01	10	33.196659833
6	100	0.001	20	31.3665683284
7	100	0.001	50	29.5764788239
8	800	0.01	1	33.3066653333
9	800	0.01	10	31.1415570779
10	1000	0.01	1	32.7316365818



The following graph shows a comparison between the MNIST test accuracies to the accuracy of USPS data set.



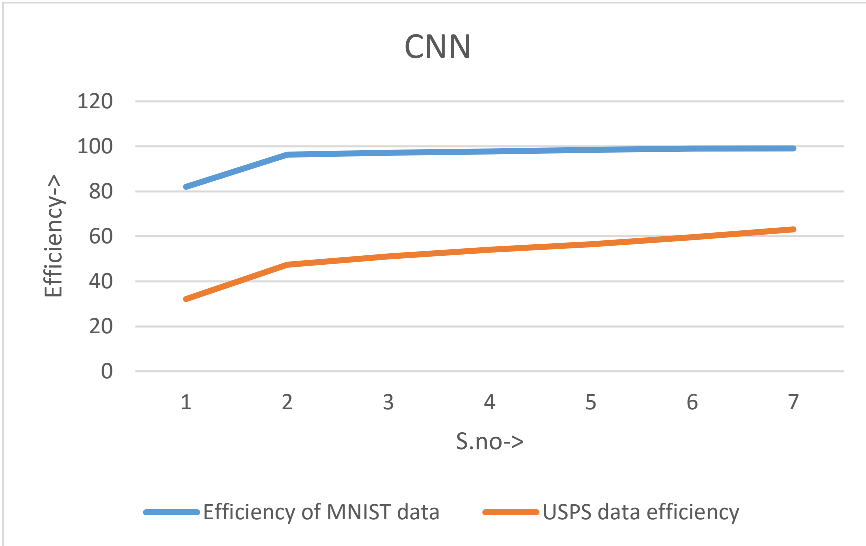
The above graph shows that the USPS test accuracy is much lower when compared to the test accuracy of the MNIST test data, supporting the “No free lunch” theorem.

<u>Convolutional Neural Network:</u>

The MNIST trained convolutional neural network model was tested on USPS data. The values are tabulated for different iterations.

S.no	No. of iterations	Efficiency of MNIST data	USPS data efficiency
1	100	82.01	32.13
2	1000	96.26	47.41
3	1500	97.13	51.06
4	3000	97.73	54.12
5	5000	98.41	56.53
6	10000	98.97	59.59
7	20000	99.08	63.11

A graph for the above values are plotted:



Thought the convolution neural network model has higher accuracy around 60 % for the USPS data, it is way too lower than the MNIST test accuracy of 99%

<u>Conclusion:</u>

In all the three implementations, the MNIST test accuracy is in the range of 90-99%. Whereas the USPS test data accuracy is in the range of 30-60%. This in turn supports the fact “that if an algorithm performs well on a certain class of problems then it necessarily pays for that with degraded performance on the set of all remaining problems” which is nothing but the NFL theorem