EE 5841 Exam submission 2

by Sarala Ravindra

Question 4

MNIST data extraction and preprocessing

The MNIST dataset has 60000 train samples and 10000 test samples.The samples are hand written digits which are size-normalized and centered in a fixed-size image.Each image is 28 cross 28 in size.The below code extracts the MNIST data and is stored as train_images , test_images which are the train and test samples respectively and their corresponding labels are stored as train_lab and test_lab respectively.

In [9]:

```
%%time

import numpy as np

#  train and test data extraction

dataset_train = np.load("train_data.npz")

train_images = dataset_train['x']                       #train images
extraction
train_lab = dataset_train['y']                          #train labels extract
on
dataset_test = np.load("test_data.npz")

test_images = dataset_test['x']                         #test images extracti
n
test_lab = dataset_test['y']                            #test labels extracti
n

train_img = np.reshape(train_images,[60000,28*28])
train_lab = np.ravel(train_lab,order = 'C')
test_img = np.reshape(test_images,[10000,28*28])
```

```
Wall time: 433 ms
```

MLP Classifier implementation

In [75]:

```
%%time

#Import Libraries
import numpy as np
from sklearn.neural_network import MLPClassifier
```

```
# parameters on which the MLP classifier behaviour is studied
a =  [0.0001 , 0.001 , 0.01 , 0.1, 1, 10 ,100]
l_r =  [0.0001 , 0.001 , 0.01 , 0.1, 1, 10 ,100]

test_correct =  np.zeros((7,7))
train_correct =  np.zeros((7,7))

for ai in range(7):                                  # loops to fit model and c
eck performance for all the combination of parameters
    for li in range(7):

        # Create classification model
        clf = MLPClassifier(hidden_layer_sizes=(100, ), alpha=a[ai], learni
ng_rate_init=l_r[li])
        clf.fit(train_img,train_lab)

        # pridict test labels
        predict_lab_test = clf.predict(test_img)
        predict_lab_train = clf.predict(train_img)

        # calculate test and train accuracies
        for l in range(10000):
            if(predict_lab_test[l] == test_lab[l]):
                test_correct[ai,li] = test_correct[ai,li]+1

        for l in range(60000):
            if(predict_lab_train[l] == train_lab[l]):
                train_correct[ai,li] = train_correct[ai,li]+1

        print('For alpha  = ',a[ai],'and learning rate = ',l_r[li],'the tra
ining accuracy is',str.format('{0:.2f}',train_correct[ai,li]/600),'% and th
e testing accuracy is',str.format('{0:.2f}',test_correct[ai,li]/100),'%')
```

```
For alpha  =  0.0001 and learning rate =  0.0001 the training accuracy is 9
9.67 % and the testing accuracy is 95.19 %
For alpha  =  0.0001 and learning rate =  0.001 the training accuracy is 98
.20 % and the testing accuracy is 95.95 %
For alpha  =  0.0001 and learning rate =  0.01 the training accuracy is 89.
01 % and the testing accuracy is 88.63 %
For alpha  =  0.0001 and learning rate =  0.1 the training accuracy is 15.1
2 % and the testing accuracy is 14.91 %
For alpha  =  0.0001 and learning rate =  1 the training accuracy is 9.77 %
and the testing accuracy is 9.77 %
For alpha  =  0.0001 and learning rate =  10 the training accuracy is 9.91
% and the testing accuracy is 10.09 %
For alpha  =  0.0001 and learning rate =  100 the training accuracy is 9.86
% and the testing accuracy is 9.58 %
For alpha  =  0.001 and learning rate =  0.0001 the training accuracy is 99
.78 % and the testing accuracy is 95.59 %
For alpha  =  0.001 and learning rate =  0.001 the training accuracy is 97.
48 % and the testing accuracy is 94.78 %
For alpha  =  0.001 and learning rate =  0.01 the training accuracy is 86.5
5 % and the testing accuracy is 86.94 %
For alpha  =  0.001 and learning rate =  0.1 the training accuracy is 11.85
% and the testing accuracy is 12.19 %
For alpha  =  0.001 and learning rate =  1 the training accuracy is 9.75 %
and the testing accuracy is 9.74 %
For alpha  =  0.001 and learning rate =  10 the training accuracy is 10.22
% and the testing accuracy is 10.10 %
```

For alpha = 0.001 and learning rate = 100 the training accuracy is 9.86 % and the testing accuracy is 9.57 %
For alpha = 0.01 and learning rate = 0.0001 the training accuracy is 99.25 % and the testing accuracy is 95.02 %
For alpha = 0.01 and learning rate = 0.001 the training accuracy is 97.94 % and the testing accuracy is 95.37 %
For alpha = 0.01 and learning rate = 0.01 the training accuracy is 87.93 % and the testing accuracy is 87.15 %
For alpha = 0.01 and learning rate = 0.1 the training accuracy is 11.24 % and the testing accuracy is 11.35 %
For alpha = 0.01 and learning rate = 1 the training accuracy is 9.04 % and the testing accuracy is 8.92 %
For alpha = 0.01 and learning rate = 10 the training accuracy is 9.87 % and the testing accuracy is 9.80 %
For alpha = 0.01 and learning rate = 100 the training accuracy is 9.87 % and the testing accuracy is 9.80 %
For alpha = 0.1 and learning rate = 0.0001 the training accuracy is 99.72 % and the testing accuracy is 95.02 %
For alpha = 0.1 and learning rate = 0.001 the training accuracy is 98.53 % and the testing accuracy is 96.01 %
For alpha = 0.1 and learning rate = 0.01 the training accuracy is 86.43 % and the testing accuracy is 86.43 %
For alpha = 0.1 and learning rate = 0.1 the training accuracy is 11.24 % and the testing accuracy is 11.36 %
For alpha = 0.1 and learning rate = 1 the training accuracy is 11.24 % and the testing accuracy is 11.35 %
For alpha = 0.1 and learning rate = 10 the training accuracy is 9.06 % and the testing accuracy is 8.92 %
For alpha = 0.1 and learning rate = 100 the training accuracy is 10.44 % and the testing accuracy is 10.28 %
For alpha = 1 and learning rate = 0.0001 the training accuracy is 99.77 % and the testing accuracy is 95.97 %
For alpha = 1 and learning rate = 0.001 the training accuracy is 97.97 % and the testing accuracy is 96.52 %
For alpha = 1 and learning rate = 0.01 the training accuracy is 79.51 % and the testing accuracy is 79.26 %
For alpha = 1 and learning rate = 0.1 the training accuracy is 10.22 % and the testing accuracy is 10.10 %
For alpha = 1 and learning rate = 1 the training accuracy is 9.04 % and the testing accuracy is 8.92 %
For alpha = 1 and learning rate = 10 the training accuracy is 9.80 % and the testing accuracy is 9.86 %
For alpha = 1 and learning rate = 100 the training accuracy is 10.39 % and the testing accuracy is 10.18 %
For alpha = 10 and learning rate = 0.0001 the training accuracy is 99.75 % and the testing accuracy is 98.03 %
For alpha = 10 and learning rate = 0.001 the training accuracy is 97.38 % and the testing accuracy is 96.78 %
For alpha = 10 and learning rate = 0.01 the training accuracy is 88.41 % and the testing accuracy is 88.86 %
For alpha = 10 and learning rate = 0.1 the training accuracy is 9.94 % and the testing accuracy is 10.32 %
For alpha = 10 and learning rate = 1 the training accuracy is 10.04 % and the testing accuracy is 10.48 %
For alpha = 10 and learning rate = 10 the training accuracy is 10.21 % and the testing accuracy is 10.03 %
For alpha = 10 and learning rate = 100 the training accuracy is 10.94 % and the testing accuracy is 10.67 %
For alpha = 100 and learning rate = 0.0001 the training accuracy is 97.52 % and the testing accuracy is 97.10 %

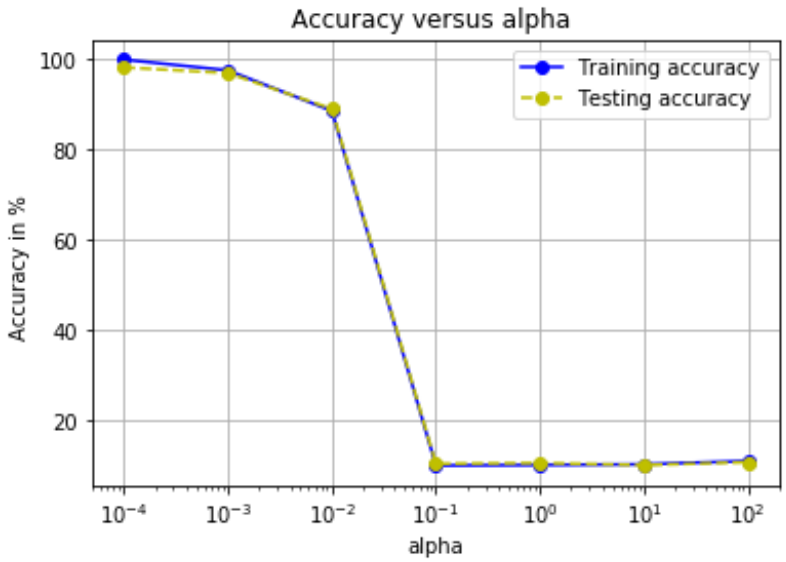
```
For alpha  =  100 and learning rate =  0.001 the training accuracy is 95.56
% and the testing accuracy is 95.37 %
For alpha  =  100 and learning rate =  0.01 the training accuracy is 86.93
% and the testing accuracy is 87.27 %
For alpha  =  100 and learning rate =  0.1 the training accuracy is 21.59 %
and the testing accuracy is 21.32 %
For alpha  =  100 and learning rate =  1 the training accuracy is 10.27 % a
nd the testing accuracy is 10.20 %
For alpha  =  100 and learning rate =  10 the training accuracy is 9.83 % a
nd the testing accuracy is 9.62 %
For alpha  =  100 and learning rate =  100 the training accuracy is 9.89 %
and the testing accuracy is 9.85 %
Wall time: 35min 34s
```

The above code is writtten to output the training and testing accuracies for each combination of alpha and learning rate provided in the parameter list.**The results shows that alpha = 10 and learning rate = 0.0001 provides the best performance** (with test accuracy = 98.03 % and training accuracy = 99.75 %). Note the learning rate here is the learning rate_init parameter in the fuction used but it specicies the learning rate value for the above classifier.

Plots of training and test accuracies versus alpha and learning rate (when analysisng one parameter the other one is at its default value)

In [26]:

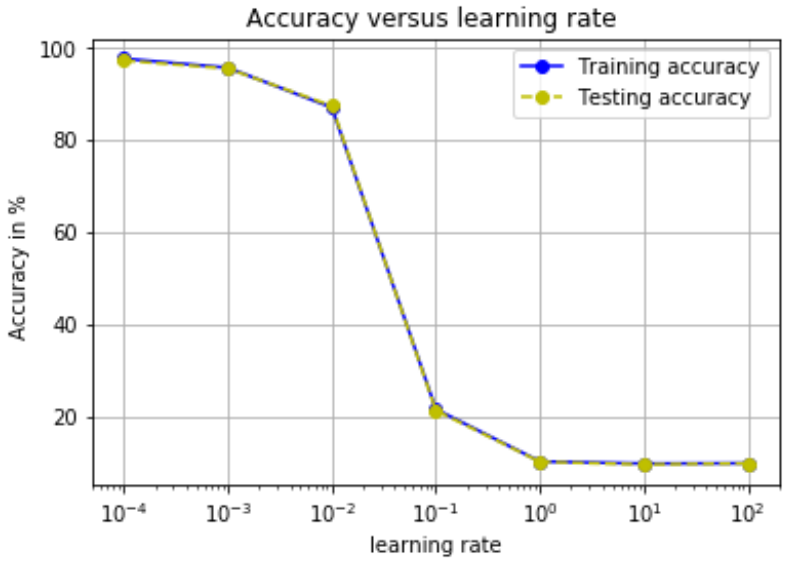
```
import matplotlib.pyplot as plt
# a =  [0.0001 , 0.001 , 0.01 , 0.1, 1, 10 ,100]
# l_r =  [0.0001 , 0.001 , 0.01 , 0.1, 1, 10 ,100]
train_percent= [99.75,97.38,88.41,9.94,10.04,10.21,10.94]
test_percent= [98.03,96.78,88.86,10.32,10.48,10.03,10.67]
plt.plot(a,train_percent,'-bo',a,test_percent,'--yo')

plt.legend(['Training accuracy','Testing accuracy'])
plt.xscale('log')
plt.xlabel('alpha')
plt.ylabel('Accuracy in % ')
plt.title('Accuracy versus alpha ')
plt.grid(True)
plt.show()
```



The parameters chosen to analyze the MLP classifier are alpha and the learning_rate_init .Alpha is the penalty parameter of l2 regulariser used in the classifier model while learning_rate_init is the rate at which the model updates it's weights during the learning.These proper choice of these two are important for the classifier performance.It is because alpha gives the tuning required to saves the classifier from being an over-fit model.**Higher values of alphas gives smaller weights on the features thereby decreasing the chances of model overfit (due to reduced variance) while the lower alpha values give larger weights on the features helping in dealing with bias related underfit of the model.** In simpler terms **too high the alpha the resuling model might be underfit and too low the alpha the resultuing model might be overfit**.Thus alpha must be wisely chosen so as to produce weights such that the model does not overfit or underfit the train data.

In [27]:

```
train_percent=[97.52,95.56,86.93,21.59,10.27,9.83,9.89]
test_percent= [97.1,95.37,87.27,21.32,10.2,9.62,9.85]
plt.plot(a,train_percent,'-bo',a,test_percent,'--yo')
plt.legend(['Training accuracy','Testing accuracy'])
plt.xscale('log')
plt.xlabel('learning rate')
plt.ylabel('Accuracy in % ')
plt.title('Accuracy versus learning rate ')
plt.grid(True)
plt.show()
```



Next coming to the learning rate parameter, it is the parameter that **controls the step-size taken in updating the model weights during the learning** from the training data.Smaller the learning step the finer the weight updates and better the accuracies while larger the steps faster the update and the testing accuracy relatively goes down.This effect can be seen in the above graph.Also smaller learning rate means it takes longer to come to the final weights of the model due to its smaller step-size and the larger steps does the learning faster but performs relatively poorly. So while choosing the learning rate we must also weigh the query time and the accuracies and make a choice based on the application.As our application here did not put any hard-time limitations I only considered the accuracies to choose the best learning rate for the model.

MLP with best choice of parameters

In [106]:

```
%%time

import time

#Import Libraries

from sklearn.neural_network import MLPClassifier
import matplotlib.pyplot as plt

# Create classification model
h =  [1 , 8,  64 , 100 , 1000 ]

test_correct =  np.zeros((5))

for hi in range(5):
    t1 = time.time()
    clf = MLPClassifier(hidden_layer_sizes= (h[hi],) , alpha = 10 ,max_iter
=500, learning_rate_init = 0.0001 )

    clf.fit(train_img,train_lab)
    t = time.time() - t1

    # pridict test labels
    predict_lab_test = clf.predict(test_img)

    # calculate test accuracy

    for l in range(10000):
        if(predict_lab_test[l] == test_lab[l]):
            test_correct[hi] = test_correct[hi] +1

    print('For ', h[hi],' nodes in hidden layers the Query time =' ,t,' and
testing accuracy is',str.format('{0:.2f}',test_correct[hi]/100),'%')
```

```
For  1  nodes in hidden layers the Query time = 214.03037095069885  and tes
ting accuracy is 39.28 %
For  8  nodes in hidden layers the Query time = 106.95877718925476  and tes
ting accuracy is 92.93 %
For  64  nodes in hidden layers the Query time = 149.85248827934265  and te
sting accuracy is 97.76 %
For  100  nodes in hidden layers the Query time = 201.21722531318665  and t
esting accuracy is 97.80 %
For  1000  nodes in hidden layers the Query time = 1884.0273506641388  and
testing accuracy is 97.97 %
Wall time: 42min 37s
```

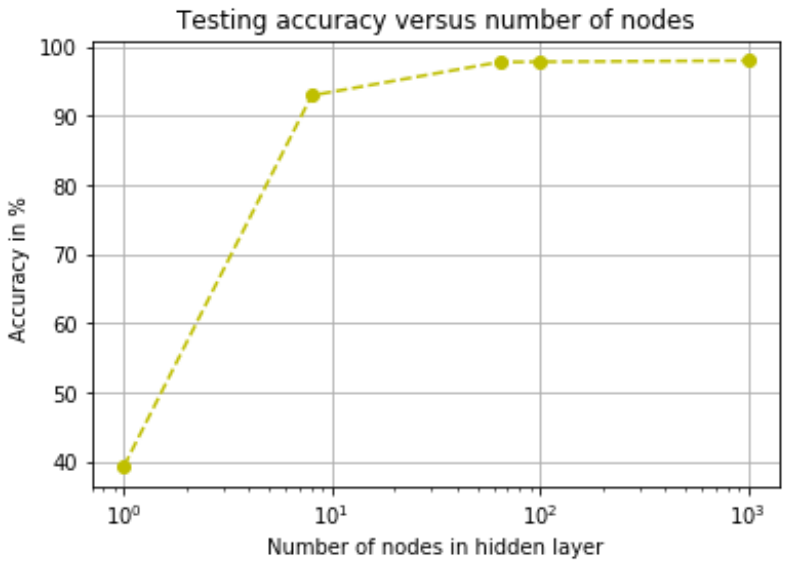
The above code is written to implement an MLP classifier with alpha =10 and learning_rate_init = 0.0001 and output the testing accuracies along with the query times for different number of nodes in the hidden layers.

Plot comparing the test accuracy versus the number of nodes in hidden layer

In [107]:

```
import matplotlib.pyplot as plt

test_percent= np.divide(test_correct,100)
plt.plot(h,test_percent,'--yo')
plt.xscale('log')
```

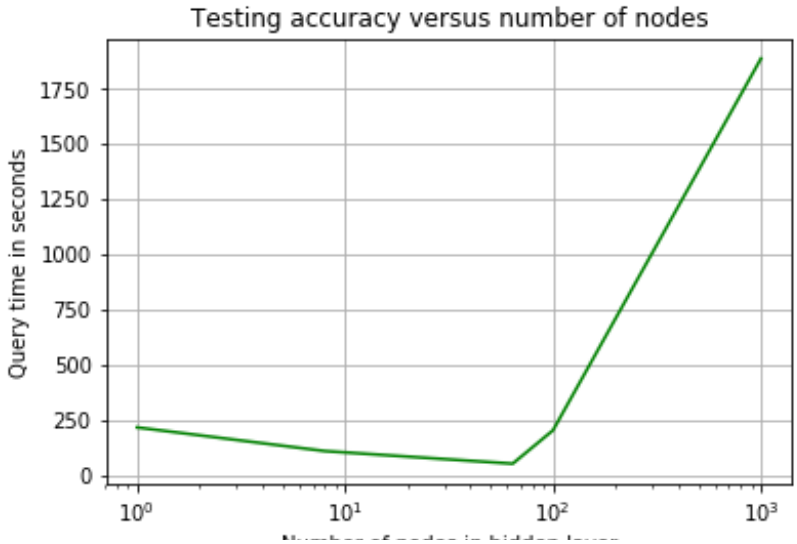
```
plt.xlabel('Number of nodes in hidden layer')
plt.ylabel('Accuracy in % ')
plt.title('Testing accuracy versus number of nodes')
plt.grid(True)
plt.show()
```



From the graph above we can observe **increase in the classifier testing accuracy with more nodes in the hidden layer**.This makes sense as the nodes in the layer increase*there is more sophisticate computation at each node.But the accuracy increse with respect to the number of nodes is not linear .The accuracy is as the node number goes from 64 to 100 is too small.Thus other parameters such as processing time must be weighed before the selection of the number of nodes.

In [30]:

```
import matplotlib.pyplot as plt

x = [1,8,64,100,1000]
y = [214,107, 50, 201, 1884]
plt.plot(x,y,'g')
plt.xscale('log')
plt.xlabel('Number of nodes in hidden layer')
plt.ylabel('Query time in seconds')
plt.title('Testing accuracy versus number of nodes')
plt.grid(True)
plt.show()
```



Number of nodes in hidden layer

As already mentioned before ,more the number of nodes in the hidden layer higher the compuation sophistication in each layer.Needless to say this sophistication also brings in delay in computation when the more number of nodes are used when not required .This can be explained from the steep increase in query time from 100 to 1000.Infact as the order of the number of nodes increase the time requirement goes up.The first half of the graph explains the fact that more the nodes involved faster the query (it can be observed from 1 to 64 of the x axis).From the graph it can be concluded that

1. More nodes in hidden layers results in less query time as long as the number of noder is in the order demanded by the classifier computation requirements.
2. if the number of nodes increase in high orders without the computation requirement the query time goes up

From the two graphs above 64 nodes in hidden layer seems to be a good choice for this problem(considering test accuracy and query time)

Comparision of MLP classsifier with other models

The best accuracies achieved for various classifiers implemented in my assignments for this course on the MNIST data is tabulated along with training time for each of them.

Classifier	**Number of training samples in thousands**	**Training time (approx)**	**Testing accuracy(%)**
MLP	60	57 sec	97.97
KNN	60	387 sec	96.91
Naive Bayes	60	13.5 sec	78.92
Logistic Regression	60	250 sec	84.4
RBF SVM	10	133 sec	94.95
linear SVM	10	132.1 sec	90.06

The accuracy achieved with the MLP classifier is the highest compared to the rest of the classifiers for the MNIST data set but it is not the best in terms of time but it is better than KNN which has almost same classification capacity but takes longer time to classify.KNN takes longer to train as it has to calculate the distance between the test sample and every train datapoint to determine the nearest neighbours which is a labourious process.The RBF SVM performs good in prediction but takes a lot of time to train as it requires to compute the radial functions.The logistic regression is the least performing in terms of accuracy as it is a linear model giving categorical data.From the table it can be seen the MLP is a good classifier with less training time requirement.**Weighing the classifiers on these two parameters the MLP classifier seems to be the best for the MNIST data set classification**.