Classification with MNIST

The test images and train images and their corresponding labels are downloaded from the link http://yann.lecun.com/exdb/mnist/ .The images are reshaped for easy computation

In [3]:

```
##  Question 1

import numpy as np
import mnist

#  input data extraction

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

train_img = np.reshape(train_images,[60000,28*28])
test_img = np.reshape(test_images,[10000,28*28])

p_test_lab=np.zeros((10000))                             #to hold predicted labels o
test data
D=np.zeros((60000))

for s in range(10000):                                          # loop for 100
test samples

    sub = (train_img - test_img[s,:])
    D = np.einsum('ij,ij->i',sub,sub,dtype='float32')
    minval=np.argmin(D, axis=0)                                 # index of the
inimum distant point from the train set
    p_test_lab[s]=train_lab[minval]                           # predicted label.
for the test images

## Generate confusion matrix

##for a given label value ranging from 0 to 9
##check if the predicted label value is equal to the corresponding label
count = np.zeros([10,10])
for k_actual in range(10):
    for k_predicted in range(10):
        for l in range(10000):
            if( test_lab[l]==k_actual and p_test_lab[l]==k_predicted):
                count[k_actual,k_predicted]=count[k_actual,k_predicted]+1;
#confusion matrix
print('The confusion matrix for the classification is ')
print(count)
print('The correct prediction  is ',np.divide(np.trace(count),0.01*10000),'
%')
```

```
The confusion matrix for the classification is
[[  9.73000000e+02   1.00000000e+00   1.00000000e+00   0.00000000e+00
    0.00000000e+00   1.00000000e+00   3.00000000e+00   1.00000000e+00
    0.00000000e+00   0.00000000e+00]
 [  0.00000000e+00   1.12900000e+03   3.00000000e+00   0.00000000e+00
    1.00000000e+00   1.00000000e+00   1.00000000e+00   0.00000000e+00
    0.00000000e+00   0.00000000e+00]
 [  7.00000000e+00   6.00000000e+00   9.92000000e+02   5.00000000e+00
    1.00000000e+00   0.00000000e+00   2.00000000e+00   1.60000000e+01
    3.00000000e+00   0.00000000e+00]
 [  0.00000000e+00   1.00000000e+00   2.00000000e+00   9.70000000e+02
    1.00000000e+00   1.90000000e+01   0.00000000e+00   7.00000000e+00
    7.00000000e+00   3.00000000e+00]
 [  0.00000000e+00   7.00000000e+00   0.00000000e+00   0.00000000e+00
    9.44000000e+02   0.00000000e+00   3.00000000e+00   5.00000000e+00
    1.00000000e+00   2.20000000e+01]
 [  1.00000000e+00   1.00000000e+00   0.00000000e+00   1.20000000e+01
    2.00000000e+00   8.60000000e+02   5.00000000e+00   1.00000000e+00
    6.00000000e+00   4.00000000e+00]
 [  4.00000000e+00   2.00000000e+00   0.00000000e+00   0.00000000e+00
    3.00000000e+00   5.00000000e+00   9.44000000e+02   0.00000000e+00
    0.00000000e+00   0.00000000e+00]
 [  0.00000000e+00   1.40000000e+01   6.00000000e+00   2.00000000e+00
    4.00000000e+00   0.00000000e+00   0.00000000e+00   9.92000000e+02
    0.00000000e+00   1.00000000e+01]
 [  6.00000000e+00   1.00000000e+00   3.00000000e+00   1.40000000e+01
    5.00000000e+00   1.30000000e+01   3.00000000e+00   4.00000000e+00
    9.20000000e+02   5.00000000e+00]
 [  2.00000000e+00   5.00000000e+00   1.00000000e+00   6.00000000e+00
    1.00000000e+01   5.00000000e+00   1.00000000e+00   1.10000000e+01
    1.00000000e+00   9.67000000e+02]]
The correct prediction  is  96.91 %
```

The testing accuracy for each digit is presented as a confusion matrix above.The trace of this matrix gives the total accuracy of the classifier.The error can be computed by subtracting the total accuracy by total sample number.

It is important to observe that 1 nearest kNN classifier produced a good accuracy rate but it has a long query time.

Question 2

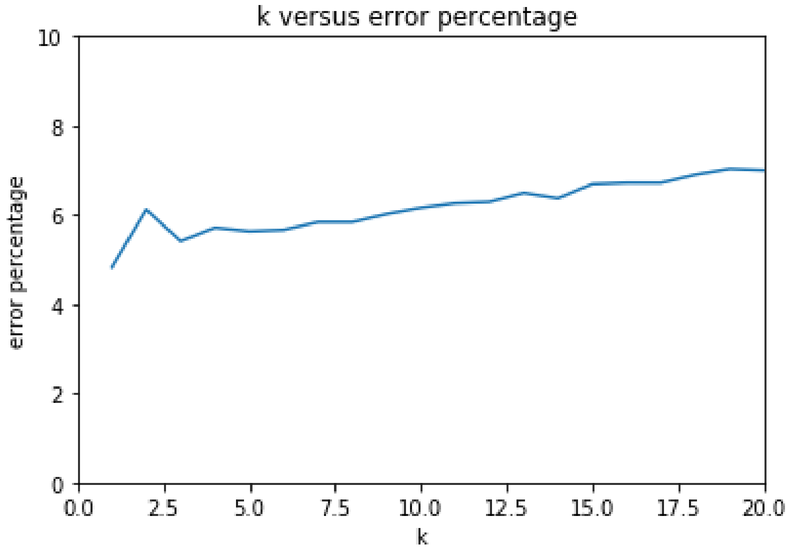
In [5]:

```
%%time

import random
from scipy.stats import mode
import numpy as np
a = 10000
ran_index = np.zeros(a)
for r_int in range(a):
    ran_index[r_int] = random.randint(0,60000)
    r_i=(ran_index).astype(int)
```

```
    samp_train_img = train_img[r_i]
    samp_train_lab = train_lab[r_i]
# generate distance matrix
D=np.zeros([a,a])

for s in range(a):                                          # loop for a leave
ones

    sub = (samp_train_img - samp_train_img[s,:])
    D[:,s] = np.einsum('ij,ij->i',sub,sub,dtype='float32')

# Cross-validation to determine best k in range 1 to 20

e = np.zeros((20))                              # error count for k values rang
ng from 1 to 20

for k in range(1,21):                           # loop for k
    te_img = np.zeros([1,28*28])                # to store test and train for c
ross validation
    tr_img = np.zeros([a-1,28*28])
    predict_label = np.zeros((10000))
    for leave_one in range(10000):              # leave-one method for every sa
mple in test

        #te_img = samp_train_img[leave_one,:]
        te_lab = samp_train_lab[leave_one]
        #tr_img = np.delete(samp_train_img,leave_one, 0)
        tr_lab = np.delete(samp_train_lab,leave_one, 0)
        Dist = np.zeros(1000-1)
        Dist = np.delete(D[leave_one,:],leave_one,0)
        #sort distance matrix from min to max value indices
        D_index_sort = np.argsort(Dist, axis=0)
        min_ind = np.zeros([k])             # to store min index values for
given k
        min_ind = D_index_sort[0:k]
        predict_label = mode(tr_lab[min_ind])[0][0]   # generate predict la
bel  np.argmax(np.bincount

        # error counter for k
        if(predict_label != te_lab):
            e[k-1]=e[k-1]+1
print(e)
print('Best value of k is',np.argmin(e)+1)

import matplotlib.pyplot as plt

k=np.arange(1,21)
plt.plot(k,np.divide(e[k-1],100))
plt.axis([0,20,0,10])
plt.xlabel('k')
plt.ylabel('error percentage')
plt.title('k versus error percentage ')
plt.show()

print('Best value of k is',np.argmin(e),'with prediction error = ',np.divid
e(e[np.argmin(e)],100),'%')
```

```
[ 483.  612.  541.  571.  563.  566.  584.  584.  602.  616.  626.  629.
  649.  637.  669.  672.  672.  690.  703.  700.]
Best value of k is 1
```

```
Best value of k is 1
```




```
Best value of k is 0 with prediction error =  4.83 %
CPU times: user 6min 24s, sys: 428 ms, total: 6min 24s
Wall time: 6min 27s
```

Due to time contraints the 60,000 train images are randomly sampleddown to 10,000 samples.This train data is used inthe leave-one out process which is done to determine the best value of k.

From the plot above,the best value of k is found to be 1 with error percent equal to 4.83.

Question 3

In [23]:

```
%%time
#------------------Question 3-------------------

import time
from scipy.stats import mode
k=1

train_lab_3 = np.reshape(samp_train_lab,[10000])

n = 4,7,14,2
for ind in range(4):
    start_timer = time.time()                                        #star
timer
    e = 0
    train_img_down = np.zeros((10000,int(784/n[ind])))
    for sample in range(int(784/n[ind])):
        train_img_down[:,sample]=samp_train_img[:,sample*n[ind]]      #down
sampledimage

    a=10000
    D=np.zeros([a,a])
    for s in range(a):                                               # loo
for a leave ones

        sub = (train_img_down - train_img_down[s,:])
        D[:,s] = np.einsum('ij,ij->i',sub,sub,dtype='float64')
```

```
    predict_label = np.zeros([10000-1])
    for leave_one in range(10000):                                   #
leave-one method for every sample in test

        te_lab = samp_train_lab[leave_one]
        tr_lab = np.delete(train_lab_3,leave_one, 0)

        Dist = np.zeros(1000-1)
        Dist = np.delete(D[leave_one,:],leave_one,0)

        #predict label generation
        D_index_sort = np.argsort(Dist, axis=0)
        min_ind = np.zeros([1,k])                                   # to s
re min index values for given k
        min_ind = D_index_sort[0:k]
        predict_label = mode(tr_lab[min_ind])[0][0]                 # gene
ate predict label

        # error counter for k
        if(predict_label != te_lab):
            e = e+1
    end_timer = time.time()
    print('query time for n=',n[ind], "=", end_timer-start_timer)
    print('Error  =  ', e/100,'for in n=',n[ind])
```

```
query time for n= 4 = 77.76203417778015
Error  =   9.48 for in n= 4
query time for n= 7 = 52.517000913619995
Error  =   18.49 for in n= 7
query time for n= 14 = 31.86644434928894
Error  =   30.85 for in n= 14
query time for n= 2 = 150.3340916633606
Error  =   5.34 for in n= 2
CPU times: user 5min 11s, sys: 1.21 s, total: 5min 12s
Wall time: 5min 12s
```

The train images of length of 28*28 are sampled down by factor n. The leave one out process is carried outfor different n values with the best value of k determined in the previous question which is equal to 1. As the sampling factor value increases the query time decreases due to lesser amount of image content to deal in computation but the prediction error also increases for the same reason.This is evident from the result above.

Question 4 & 5

In [12]:

```
%%time


import time
from scipy.stats import mode
k=1
train_img_4 = np.reshape(samp_train_img,[10000,28,28])     #reshape the tra
in data to an array
train_lab_4 = np.reshape(samp_train_lab,[10000])
```

```
n = 2,4,7,14,1                                              #n=1 for questio
5
for ind in range(5):
    start_timer = time.time()
    train_smart=np.zeros((10000,n[ind],n[ind]))
    e = 0
    for sample in range(10000):
        for x in range(n[ind]):
            for y in range(n[ind]):
                c=int(28/n[ind])
                train_smart [sample,x,y] =
np.sum(train_img_4[sample,x*c:x*c+c,y*c:y*c+c])

    train_img_down = np.reshape(train_smart,[10000,n[ind]*n[ind]])

    a=10000
    D=np.zeros([a,a])
    for s in range(a):                                        # loop for a
leave ones
        sub = (train_img_down - train_img_down[s,:])
        D[:,s] = np.einsum('ij,ij->i',sub,sub,dtype='float64')

    predict_label = np.zeros([10000-1])
    for leave_one in range(10000):                          # leave-one method
for every sample in test

        te_lab = samp_train_lab[leave_one]
        tr_lab = np.delete(train_lab_4,leave_one, 0)

        Dist = np.zeros(1000-1)
        Dist = np.delete(D[leave_one,:],leave_one,0)

        #predict label generation
        D_index_sort = np.argsort(Dist, axis=0)
        min_ind = np.zeros([1,k])                             # to store min i
dex values for given k
        min_ind = D_index_sort[0:k]
        predict_label = mode(tr_lab[min_ind])[0][0]           # generate
predict label

        # error counter for k
        if(predict_label != te_lab):
            e = e+1
    end_timer = time.time()
    print('query time for n=',n[ind], "=", end_timer-start_timer)
    print('Error=  ', e/100,'percent for n=',28/n[ind])
```

```
query time for n= 2 = 11.898622989654541
Error=   50.48 percent for n= 14.0
query time for n= 4 = 17.362730741500854
Error=   20.64 percent for n= 7.0
query time for n= 7 = 31.956013679504395
Error=   6.09 percent for n= 4.0
query time for n= 14 = 89.7149646282196
Error=   4.27 percent for n= 2.0
query time for n= 1 = 10.082106351852417
Error=   72.11 percent for n= 28.0
CPU times: user 2min 39s, sys: 1.5 s, total: 2min 41s
```

```
Wall time: 2min 41s
```

Smart sampling reduces the image size without much loss in the image information compared to the regular sampling done in the previous question.comparing the results ofquestion 3 and 4 we can see that the error for a given n reduces with smart sampling compared to the regular sampling. please note for factor n image is binneddown to 28/n size ,say for n = 4 an image reduced to 28/4=7. As the sampling factor value increases the query time decreases due to lesser amount of image content to deal in computation but the prediction error also increases for the same reason.But the smartsampler performs better compared to the regular sampler.

The above cell also includes result for reducing 28*28 image to 1 pixel and it corresponds to n = 1. The error for smart sampling image down by 28 is 72.11%

Question 6

In [22]:

```
%%time
import time

from scipy.stats import mode
k=1
train_img_4 = np.reshape(samp_train_img,[10000,28,28])     #reshape the tra
in data to an array
train_lab_4 = np.reshape(samp_train_lab,[10000])

n = 10,20,30,40,50,60

for ind in range(6):
    print(n[ind])
    train_img_down=np.zeros((10000,n[ind]*n[ind]))
    train_img_4_2d=np.reshape(train_img_4,[10000,28*28])
    e = 0
    k=n[ind]
    data = train_img_4_2d.astype("float64")
    data -= np.mean(data, axis=0)
    U, S, V = np.linalg.svd(data, full_matrices=False)
    train_img_down=U[:,:k].dot(np.diag(S)[:k,:k])
    a=10000
    D=np.zeros([a,a])
    for s in range(a):                                          # loop for a
leave ones
        sub = (train_img_down - train_img_down[s,:])
        D[:,s] = np.einsum('ij,ij->i',sub,sub,dtype='float64')

    predict_label = np.zeros([10000-1])
    for leave_one in range(10000):                          # leave-one method
for every sample in test

        te_lab = samp_train_lab[leave_one]
        tr_lab = np.delete(train_lab_4,leave_one, 0)

        Dist = np.zeros(1000-1)
        Dist = np.delete(D[leave_one,:],leave_one,0)

        #predict label generation
        D_index_sort = np.argsort(Dist, axis=0)
```

```
        D_index_sort = np.argsort(Dist, axis=0)
        min_ind = np.zeros([1,k])                          # to store min i
dex values for given k
        min_ind = D_index_sort[0:k]
        predict_label = mode(tr_lab[min_ind])[0][0]        # generate
predict label

        # error counter for k
        if(predict_label != te_lab):
            e = e+1                                    #e[k-1]=e[k-1]+1
    print('Error=  ', e/100,'percent for n=',n[ind])
```

```
10
Error=   9.62 percent for n= 10
20
Error=   6.07 percent for n= 20
30
Error=   6.02 percent for n= 30
40
Error=   6.59 percent for n= 40
50
Error=   7.49 percent for n= 50
60
Error=   8.04 percent for n= 60
CPU times: user 2min 45s, sys: 2.38 s, total: 2min 47s
Wall time: 2min 37s
```

PCA implementation to improve the classifier performance involves the feature dimension reduction by generating a non-alligned axis along which the variance of the data is more.Here the image after PCA implementation mostly consists of only those features whose 'value' is more in the determination of the class ofa sample given.In the above program the PCA implementation is achieved using the singularvalue decomposition of the image matrix.the feature matix is eigen values are computed and a new tranformation isdetermined to obtain the non-alligned axis vector.

note thta the query time of the classifier increases with almost the same error prediction percent.