EE5841 MACHINE LEARNING

Classification Project 1 - Report

--

Submitted by : Ashwini Nikumbh, Vrushaketu Mali, Ponkrshnan Thiagarajan

Date: 02-Mar-2020

Index

Introduction

This project aims at developing a framework to perform classification on a given dataset. Therefore, this project includes the implementation of several KNN classifiers on the MNIST data set. The MNIST database of handwritten digits has a training set of 60,000 examples and a test set of 10,000 examples. It is a subset of a larger set available from NIST. The digits have been size-normalized and centered in a fixed-size image of 28 x 28 pixels.

Import the libraries

Below imported libraries will be used in the various functions developed in the different codes of this project.

In [1]:
```
import numpy as np
import idx2numpy as id
import pandas as pd
import matplotlib.pyplot as plt

from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report
from sklearn.decomposition import PCA
from sklearn import preprocessing
from sklearn.preprocessing import MinMaxScaler
%matplotlib inline
```

Extracting the dataset

In [2]:
```
train_img_read = open('train-images.idx3-ubyte', 'rb')   # open the f
ile of training images
train_img = id.convert_from_file(train_img_read)

train_lab_read = open('train-labels.idx1-ubyte', 'rb')  # open the fi
le of training labels
train_lab = id.convert_from_file(train_lab_read)

test_img_read = open('t10k-images.idx3-ubyte', 'rb')    # open the fi
le of testing images
test_img = id.convert_from_file(test_img_read)

test_lab_read = open('t10k-labels.idx1-ubyte', 'rb')    # open the fi
le of testing labels
test_lab = id.convert_from_file(test_lab_read)

train=train_img.flatten()
X_train=train.reshape(60000,784)     # reshape the training data as 6
0000x784
test=test_img.flatten()
x_test=test.reshape(10000,784)       # reshape the testing data as 10
000x784
y_train=train_lab[:]
y_test=test_lab[:]
```

Part 1 - Implement a 1-nearest neighbor classifier

In this part, We are going to implement a 1-nearest neighbor classifier model that considers the image pixels in entire data set to be one long feature vector. To accomplish this, we are using the KNeighborsClassifier library which is imported from sklearn.neighbors. This is done without any scaling and the normalization on the pixel values. At the end of the function, the testing errors are presented for each digit in a table format.

In [4]:

```
Knn = KNeighborsClassifier(n_neighbors=1)             # Input the numb
er of nearest neighbors as 1
Knn.fit(X_train, y_train)                             # Fit KNN classi
fier model on training features and training labels
y_pred = Knn.predict(x_test)                          # Predict the cl
ass/label of test features


#Prediction of test error
test_error_recall=[]
Err_test=[]
for i in range(10):
    sample_test=y_test
    test_data=pd.DataFrame(sample_test)
    i1=test_data[test_data[0]==i]
    indx=list(i1.index)
    err=1-((y_pred[indx] ==i).sum()/(y_test==i).sum())
    test_error_recall.append(err)
    Err=(y_pred[indx] !=i).sum()
    Err_test.append(Err)
h=pd.DataFrame(test_error_recall, columns=['Percentage of testing err
or for each digit']) # % of test error for each digit
g=pd.DataFrame(Err_test, columns=['Test error for each digit'])   #Nu
mber of times the digits were wrongly classified


#Score
score=Knn.score(x_test,y_test)
print(classification_report(y_test, y_pred))  # Print the classificat
ion report of predicted and actual labels of test features
print('The Accuracy score is ',score)
# Print the score
```

```
              precision    recall  f1-score   support

           0       0.98      0.99      0.99       980
           1       0.97      0.99      0.98      1135
           2       0.98      0.96      0.97      1032
           3       0.96      0.96      0.96      1010
           4       0.97      0.96      0.97       982
           5       0.95      0.96      0.96       892
           6       0.98      0.99      0.98       958
           7       0.96      0.96      0.96      1028
           8       0.98      0.94      0.96       974
           9       0.96      0.96      0.96      1009

    accuracy                           0.97     10000
   macro avg       0.97      0.97      0.97     10000
weighted avg       0.97      0.97      0.97     10000

The Accuracy score is  0.9691
```

In [5]:
```
h    # Percentage of test error for each digit
###This table gives percentage of true negatives
###i.e for all datapoint that were actually positive, what percent ar
e classified incorrect
```

Out[5]:

	Percentage of testing error for each digit
0	0.007143
1	0.005286
2	0.038760
3	0.039604
4	0.038697
5	0.035874
6	0.014614
7	0.035019
8	0.055441
9	0.041625

In [6]:
```
g #Number of times the digits were wrongly classified
```

Out[6]:

	Test error for each digit
0	7
1	6
2	40
3	40
4	38
5	32
6	14
7	36
8	54
9	42

Results & conclusion of part 1

From the table of "Test error for each digit" we can see the digit 8 is having highest testing error, Which can also be predicted from the value of recall in classification report. Percentage recall for digit 8 is 94% which is lowest of all other digits. Minimum test error was obtained for digits 1 followed by digit 0, their recall score is 99% (highest amongst all the digi class)

The overall accuracy score is 0.9691

Precision

It test the ability of a classifier not to label a datapoint positive that was actually negative. For each class it is eqaul to the ratio of true positives to the sum of true and false positives. It can also be interpreted as , for all datapoint classified positive, what percent are correct?

Recall

It test the ability of a classifier to find all positive datapoint. For each class it is equal to the ratio of true positives to the sum of true positives and false negatives. It can also be interpreted as ,for all datapoint that were actually positive, what percent are classified correctly?. Percentage recall for digit 8 is 94% which is lowest amongst all the class of digits. Percentage recall for digits 0 and 1 is about 99% which is highest amongst all the class of digits.

F1 score ¶

It is a weighted harmonic mean of precision and recall.

Part 2 - Implement KNN leave-one-out approach for K from 1 to 20

In this part, We are going to implement KNN leave-one-out approach by defining a function and test the classifier for K values from 1 to 20. To accomplish this, we are using the KNeighborsClassifier library which is imported from sklearn.neighbors. Since it takes longer time to train the model on 60000 training images, we are randomly sampling 2000 points from the dataset.

In [7]:
```
# Shuffling and randomly sampling the training data

sample_train=np.column_stack((X_train, y_train))    # Combine trainin
g features and labels
np.random.shuffle(sample_train)                     # Shuffle the tot
al training data


dataset_train=pd.DataFrame(sample_train)
dataset_train=dataset_train.sample(2000)            # Choose 2000 ran
dom samples within training data
X_train1=np.array(dataset_train.drop([784], axis=1)) # Separate train
ing features from sampled training data
y_train1=np.array(dataset_train[784])                # Separate train
ing labels from sampled training data

# Shuffling and randomly sampling the testing data

sample_test=np.column_stack((x_test,y_test))       # Combine testing
 features and labels
np.random.shuffle(sample_test)                      # Shuffle the tot
al testing data


dataset_test=pd.DataFrame(sample_test)
dataset_test=dataset_test.sample(2000)               # Choose 2000 ra
ndom samples within testing data
x_test1=np.array(dataset_test.drop([784], axis=1))  # Separate testin
g features from sampled testing data
y_test1=np.array(dataset_test[784])                 # Separate testin
g labels from sampled testing data


k_test=X_train[1]
#k_test = np.array([k_test])
k_test.shape
err_tmp = np.zeros(10)

def leave_one_out(X_train,y_train):    # define the leave-one-out fun
ction with input arguments as training features and labels
    Err = np.empty([0, 10])            # create a blank matrix for er
ror values
    for K in range(1, 21):             # value of K goes from 1 to 20
        knn = KNeighborsClassifier(n_neighbors=K)  # Input the value
 of K from 1 to 20 as number of nearest neighbors
        err_tmp = np.zeros([1,10])     # create a blank matrix for er
ror values of each digit
        for j in range(len(X_train)):  # j goes from  0 to length of
 training features
            k_test=X_train[j]          # Select one K_test from train
ing features every time with its index equal to j
            k_test = np.array([k_test])
            k_valid=y_train[j]         # Select one K_valid from trai
ning labels every time with its index equal to j
            k_train_input=np.delete(X_train, j, 0)  # Separate K_test
from training features and define as training input
```

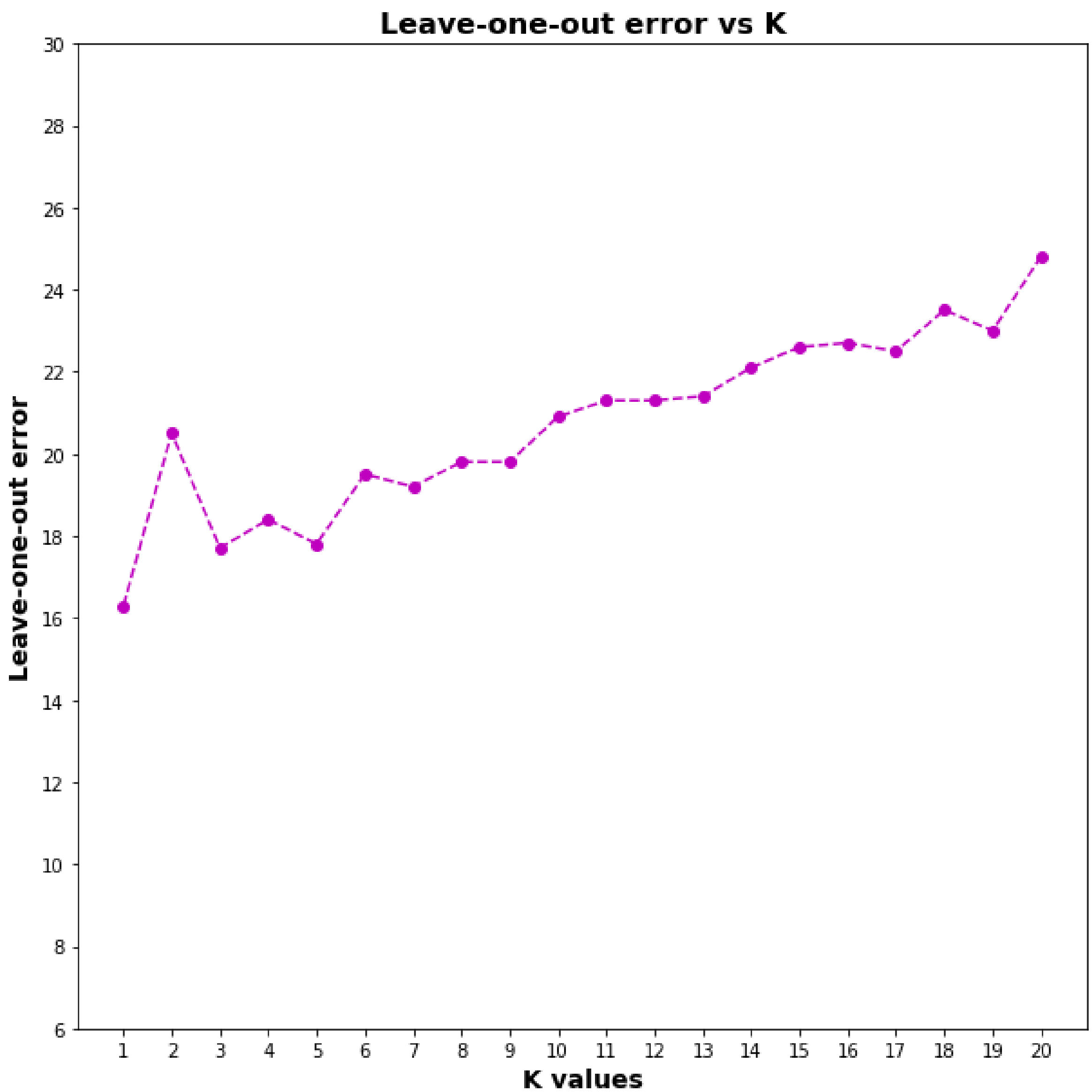
```
            k_train_outputs=np.delete(y_train, j, 0) # Separate K_val
id from training labels and define as training output
            knn.fit(k_train_input, k_train_outputs)  # Fit the KNN mo
del on training features and training labels
            pred_K = knn.predict(k_test)   # Predict the class of K_t
est
            err_tmp[0,k_valid] = err_tmp[0,k_valid] + (pred_K != k_va
lid)  # If predicted class is not equal to actual class

# then count error as 1
        Err = np.append(Err,err_tmp,axis=0)   # Fill the err matrix w
ith count of errors for all features
    return(Err)    # Return the error matrix

dp=leave_one_out(X_train1,y_train1)
d2=[]
for i in range(10):
    d1=dp[i]/sum(y_train1==i)
    d2.append(d1)
a=pd.DataFrame(d2)
a = a.applymap("{0:.2f}".format)
```

Results & conclusion of part 2

After running the leave-one-out function defined above, below code is developed for plotting the leave-one-out error for different K values (1 to 20).

In [66]:
```
# Code for plotting the testing error vs K
m=[]
b=pd.DataFrame(dp)
for i in range(20):
    c=np.array(b.iloc[i])
    m.append(np.mean(c))
K = range(1, 21)
plt.figure(figsize=(10,10))
plt.plot(K, m, 'mo--')
plt.xlabel('K values', fontweight='bold', fontsize='14')
plt.ylabel('Leave-one-out error', fontweight='bold', fontsize='14')
plt.xticks(np.arange(1, 21, 1))
plt.yticks(np.arange(6, 32, 2))
plt.title('Leave-one-out error vs K', fontweight='bold', fontsize='1
6')
```

Out[66]: Text(0.5, 1.0, 'Leave-one-out error vs K')



Comments :

The above plot shows the graph of testing error at different K values. Here, the testing error for a single K value is calculated as the mean of error of all digits. From the general trend observed in the graph it can be infrred that

1. As K value increases the testing error increases when compared at specific number of sampled data points.
2. Thus, this confirms that K controls of degree of smoothing. Small K produces small regions of each classs whereas large K leads to fewer larger regions. Hence, increment in K causes bias to increases which is evident from the plot.
3. We are taking the value of K that returned the minimum testing error.
4. The lowest error is observed at K = 1, so the best value of K is 1.

Note :

As we are randomly sampling the training set, the best value of K for every run may change.

In [9]:
```
# Code for printing the results for best value of K

dp1 = pd.DataFrame(dp,index=['K=1','K=2','K=3','K=4','K=5','K=6','K=
7','K=8','K=9','K=10','K=11','K=12','K=13','K=14','K=15','K=16','K=1
7','K=18','K=19','K=20'])
m=[]
for i in range(20):    # i goes from 0 to 19
    c=np.array(dp1.iloc[i])   # combine the errors for K = i for all
 digits 0 to 9 in an array
    m.append(np.mean(c))    # Calculate the mean of all error for K =
i

print('The best value of K is',np.argmin(m)+1)   # Print the best val
ue of K as the one which gives min error for all digits
table=pd.DataFrame(dp1.iloc[np.argmin(m)])    # Print the table for t
esting error of all digits for best value of K
#table=pd.DataFrame((dp_best),index=['0'',1','2','3','4','5',
 '6','7','8','9'])
td_props = [
  ('font-size', '14px'),
  ('text-align', 'center')
  ]
styles = [
  dict(selector="td", props=td_props)
  ]
print('Table showing testing errors for all digits (in rows) for best
value of K in column label')
table
```

```
The best value of K is 1
Table showing testing errors for all digits (in rows) for best value
of K in column label
```

Out[9]:

	K=1
0	3.0
1	2.0
2	26.0
3	20.0
4	19.0
5	17.0
6	5.0
7	13.0
8	35.0
9	23.0

Comments :

Above table shows the values of testing errors for each digit (from 0 to 9) for best value of K (1 for this run) which is obtained from the above graph. The error value presented here represents the count of number of times the predicted digit is different from actual digit.

Note :

As we are randomly sampling the training set, the best value of K for every run changes which changes the values in above table.

Part 3 - Implementing a downsampling function

In this part, We are going to implement a function that downsamples the image by a factor of n. For example, if n is 4 then every 4th pixel of image is sampled in 784 dimension vector. In this function, we are repeating the KNN leave-one-out experiment. This function is then run for 4 different values of n, results of which are present in subsequent chapters. Since it takes longer time to train the model on 60000 training images, we are randomly sampling 2000 points from the dataset.

Notes :

1. As we are randomly sampling the training set, the best value of K for every run changes which changes the output values printed in every execution of downsampling function.
2. The value of query time printed for every run depends largly on the CPU processor and thus may vary for different processing units.
3. The values shown here are obtained when the program is run on intel(R)xenon(R) e3-1245 quad-core processor.

In [10]:
```
train=train_img.flatten()
test=test_img.flatten()
y_train=train_lab[:]
y_test=test_lab[:]

def downsample(n):                       # define a downsampling func
tion
    print('Factor of downsampling is',n) # Print the factor of downsa
mpling

    train_n=[]    # Create a matrix of trainin data
    test_n=[]     # Create a matrix of testing data

    for i in range(0,int(len(train)/n)):   # i goes from 0 to number
 of training points divided by sampling factor
        train_n.append(train[i*n])         # fill train_n matrix with
sampled pixels
    for j in range(0,int(len(test)/n)):    # j goes from 0 to number
 of testing points divided by sampling factor
        test_n.append(test[j*n])           # fill test_n matrix with
 sampled pixels

    train_n_array=np.array(train_n)
    train_n_rshape=train_n_array.reshape(60000,int(len(train_n_array)
/60000))   # Reshape the matrix of sampled train features
    train_n_combin=np.column_stack((train_n_rshape, y_train))    # Co
mbine the sampled train features and labels
    np.random.shuffle(train_n_combin)      # randomly shuffle the com
bined set
    train_n_data=pd.DataFrame(train_n_combin)
    train_n_sample=train_n_data.sample(2000)   # randomly choose 2000
data points from combined set
    X_train_n=np.array(train_n_sample.drop([train_n_sample.shape[1]-1
], axis=1))# Separate training features from random data
    y_train_n=np.array(train_n_sample[train_n_sample.shape[1]-1])  #
 Separate training labels from random data

    test_n_array=np.array(test_n)
    test_n_rshape=test_n_array.reshape(10000,int(len(test_n)/10000))
# Reshape the matrix of sampled test features
    test_n_combin=np.column_stack((test_n_rshape, y_test))     # Comb
ine the sampled test features and labels
    np.random.shuffle(test_n_combin)     # randomly shuffle the combi
ned set
    test_n_data=pd.DataFrame(test_n_combin)
    test_n_sample=test_n_data.sample(2000)    # randomly choose 2000
 data points from combined set
    X_test_n=np.array(test_n_sample.drop([test_n_sample.shape[1]-1],
axis=1)) # Separate testing features from random data
    y_test_n=np.array(test_n_sample[test_n_sample.shape[1]-1])   # Se
parate testing labels from random data

    k_one_sample_n=leave_one_out(X_train_n,y_train_n)  # Implement le
ave-one-out function on training features and labels
    k_one_sample_n_table=pd.DataFrame(k_one_sample_n)
    k_one_sample_n_format = k_one_sample_n_table.applymap("{0:.2f}".f
```

```
ormat)
    print(k_one_sample_n_format)

    m=[]
    for k in range(20):     # K goes from 0 to 19
        c=np.array(k_one_sample_n_table.iloc[k])    # combine the err
ors for K = i for all digits 0 to 9 in an array
        m.append(np.mean(c))                        # Calculate the m
ean of all errors for K = i
    print('The best value of K is',np.argmin(m)+1)  # Print the best
 value of K as the one which gives min error for all digits

    Knn = KNeighborsClassifier(n_neighbors=(np.argmin(m)+1))  # Imple
ment KNN classifier with K value as best value
    Knn.fit(train_n_rshape, y_train)                            # fit
the KNN model on training data
    y_pred_dn = Knn.predict(test_n_rshape)                          #
Predict the class of test feature
    print(classification_report(y_test, y_pred_dn)) # Print the class
ification report of predicted and actual labels of test features
    score=Knn.score(test_n_rshape,y_test)    # Calculate the score of
predictoion



    #Prediction of test error
    test_error_recall=[]
    Err_test=[]
    for i in range(10):
        sample_test=y_test
        test_data=pd.DataFrame(sample_test)
        i1=test_data[test_data[0]==i]
        indx=list(i1.index)
        err=1-((y_pred_dn[indx] ==i).sum()/(y_test==i).sum())
        test_error_recall.append(err)
        Err=(y_pred_dn[indx] !=i).sum()
        Err_test.append(Err)
    h1=pd.DataFrame(test_error_recall, columns=['Percentage of testin
g error for each digit']) # % of test error for each digit
    g1=pd.DataFrame(Err_test, columns=['Test error for each digit'])
#Number of times the digits were wrongly classified


    print('The accuracy score for n=',n,'is',score)                 #
Print the score
    print(h1)                                      # Print the score
    print(g1)
```

Part 3 case - 1 - Executing downsampling function at n = 2

Here, we are running the downsampling function for n = 2. The results and the conclusions are provided below.

In [11]:
```
%%time
downsample(2)
```

```
Factor of downsampling is 2
        0     1      2      3      4      5      6      7      8      
9
0    4.00  2.00  23.00  25.00  32.00  27.00   7.00  18.00  32.00  30.
00
1    1.00  2.00  18.00  20.00  19.00  37.00  15.00  21.00  42.00  62.
00
2    5.00  3.00  22.00  20.00  32.00  31.00   5.00  20.00  34.00  28.
00
3    6.00  3.00  25.00  20.00  27.00  33.00  12.00  18.00  39.00  34.
00
4    8.00  2.00  24.00  20.00  30.00  29.00   8.00  17.00  40.00  29.
00
5    8.00  2.00  31.00  21.00  23.00  30.00   8.00  18.00  40.00  31.
00
6    8.00  2.00  30.00  22.00  28.00  32.00   6.00  14.00  40.00  23.
00
7    8.00  2.00  33.00  22.00  23.00  35.00   8.00  20.00  42.00  30.
00
8    9.00  1.00  33.00  23.00  23.00  30.00   7.00  17.00  43.00  25.
00
9   10.00  1.00  36.00  22.00  24.00  30.00   7.00  18.00  42.00  28.
00
10  10.00  2.00  34.00  24.00  28.00  31.00   8.00  18.00  44.00  28.
00
11  10.00  2.00  41.00  24.00  28.00  31.00   8.00  20.00  43.00  30.
00
12  10.00  2.00  39.00  26.00  29.00  27.00   8.00  17.00  40.00  26.
00
13   9.00  1.00  40.00  24.00  29.00  30.00   9.00  19.00  43.00  31.
00
14  11.00  2.00  41.00  28.00  32.00  33.00   8.00  18.00  44.00  28.
00
15  11.00  2.00  40.00  27.00  32.00  33.00   9.00  19.00  47.00  32.
00
16  11.00  2.00  45.00  28.00  30.00  33.00   9.00  22.00  42.00  32.
00
17  11.00  2.00  45.00  29.00  31.00  35.00   9.00  22.00  43.00  33.
00
18  11.00  2.00  44.00  29.00  30.00  35.00   9.00  21.00  44.00  31.
00
19  11.00  2.00  46.00  30.00  29.00  35.00   9.00  21.00  44.00  31.
00
The best value of K is 1
              precision    recall  f1-score   support

           0       0.98      0.99      0.99       980
           1       0.96      1.00      0.98      1135
           2       0.98      0.96      0.97      1032
           3       0.95      0.95      0.95      1010
           4       0.97      0.95      0.96       982
           5       0.95      0.96      0.95       892
           6       0.98      0.98      0.98       958
           7       0.95      0.96      0.96      1028
           8       0.98      0.93      0.95       974
           9       0.94      0.95      0.95      1009
```

```
   accuracy                           0.96     10000
  macro avg       0.96      0.96      0.96     10000
weighted avg       0.96      0.96      0.96     10000

The accuracy score for n= 2 is 0.9641
   Percentage of testing error for each digit
0                                   0.007143
1                                   0.002643
2                                   0.039729
3                                   0.050495
4                                   0.046843
5                                   0.043722
6                                   0.020877
7                                   0.038911
8                                   0.066735
9                                   0.046581
   Test error for each digit
0                          7
1                          3
2                         41
3                         51
4                         46
5                         39
6                         20
7                         40
8                         65
9                         47
CPU times: user 23min 1s, sys: 405 ms, total: 23min 2s
Wall time: 23min 2s
```

Comments :

1. The first table shows the values of testing errors for all digits (in rows) for different values of K (in columns).
2. We are taking the value of K that returned the minimum testing error. Here, the best value of K is found out to be 1.
3. We are then training our KNeighborsClassifier using this best value of k, and then evaluating the performance using the classification_report function, the output of which you can see in table 2.
4. Table 2 shows the precision of classification for every digit from 0 to 9 evaluated at best value of K.
5. Thus, the accuracy score is coming out to be 96.41%.
6. The total time required for the execution of the program and print the results is 23 mins & 2 seconds.

Part 3 case - 2 - Executing downsampling function at n = 4

Here, we are running the downsampling function for n = 4. The results and the conclusions are provided below.

In [12]:
```
%%time
downsample(4)
```

```
Factor of downsampling is 4
        0     1      2      3      4      5      6      7      8      
9
0   11.00  7.00  40.00  53.00  37.00  41.00  15.00  23.00  66.00  43.
00
1    7.00  5.00  39.00  39.00  31.00  52.00  32.00  25.00  82.00  78.
00
2    8.00  5.00  42.00  44.00  38.00  50.00  20.00  26.00  67.00  45.
00
3    8.00  6.00  41.00  41.00  36.00  53.00  19.00  24.00  70.00  55.
00
4    8.00  5.00  43.00  41.00  45.00  52.00  21.00  27.00  68.00  38.
00
5    8.00  4.00  41.00  36.00  38.00  57.00  22.00  24.00  69.00  44.
00
6   10.00  4.00  47.00  36.00  41.00  53.00  23.00  23.00  69.00  37.
00
7   11.00  4.00  48.00  35.00  46.00  54.00  24.00  21.00  66.00  41.
00
8   13.00  4.00  43.00  39.00  43.00  54.00  22.00  22.00  62.00  38.
00
9   13.00  4.00  50.00  36.00  41.00  61.00  23.00  23.00  69.00  41.
00
10  13.00  5.00  52.00  34.00  43.00  65.00  20.00  22.00  70.00  39.
00
11  13.00  5.00  54.00  38.00  42.00  60.00  20.00  22.00  68.00  39.
00
12  13.00  5.00  53.00  39.00  42.00  62.00  19.00  24.00  68.00  38.
00
13  12.00  5.00  57.00  36.00  40.00  62.00  20.00  24.00  69.00  41.
00
14  12.00  6.00  56.00  39.00  43.00  63.00  20.00  25.00  68.00  40.
00
15  13.00  6.00  61.00  38.00  41.00  66.00  23.00  24.00  71.00  42.
00
16  13.00  6.00  60.00  41.00  44.00  68.00  21.00  25.00  70.00  41.
00
17  14.00  6.00  60.00  41.00  43.00  71.00  23.00  25.00  72.00  41.
00
18  15.00  6.00  63.00  43.00  45.00  67.00  24.00  26.00  72.00  39.
00
19  15.00  6.00  68.00  43.00  45.00  69.00  23.00  26.00  73.00  41.
00
The best value of K is 1
              precision    recall  f1-score   support

           0       0.96      0.98      0.97       980
           1       0.90      0.98      0.94      1135
           2       0.97      0.92      0.95      1032
           3       0.90      0.91      0.90      1010
           4       0.92      0.91      0.91       982
           5       0.92      0.88      0.90       892
           6       0.95      0.97      0.96       958
           7       0.93      0.93      0.93      1028
           8       0.93      0.86      0.89       974
           9       0.89      0.91      0.90      1009
```

```
    accuracy                           0.93     10000
   macro avg       0.93      0.92      0.93     10000
weighted avg       0.93      0.93      0.93     10000

The accuracy score for n= 4 is 0.926
   Percentage of testing error for each digit
0                                    0.019388
1                                    0.015859
2                                    0.076550
3                                    0.094059
4                                    0.089613
5                                    0.116592
6                                    0.033403
7                                    0.071012
8                                    0.142710
9                                    0.092170
   Test error for each digit
0                         19
1                         18
2                         79
3                         95
4                         88
5                        104
6                         32
7                         73
8                        139
9                         93
CPU times: user 12min, sys: 168 ms, total: 12min 1s
Wall time: 12min 1s
```

Comments :

1. The first table shows the values of testing errors for all digits (in rows) for different values of K (in columns).
2. We are taking the value of K that returned the minimum testing error. Here, the best value of K is found out to be 1.
3. We are then training our KNeighborsClassifier using this best value of k, and then evaluating the performance using the classification_report function, the output of which you can see in table 2.
4. Table 2 shows the precision of classification for every digit from 0 to 9 evaluated at best value of K.
5. Thus, the accuracy score is coming out to be 92.6%.
6. The total time required for the execution of the program and print the results is 12 mins & 1 seconds.

Part 3 case - 3 - Executing downsampling function at n = 8

Here, we are running the downsampling function for n = 8. The results and the conclusions are provided below.

In [13]:
```
%%time
downsample(8)
```

```
Factor of downsampling is 8
        0     1      2      3      4      5      6      7      8      
9
0   14.00  5.00  47.00  29.00  41.00  30.00  23.00  27.00  57.00  44.
00
1    9.00  2.00  44.00  32.00  31.00  42.00  31.00  33.00  80.00  85.
00
2    8.00  5.00  46.00  33.00  40.00  43.00  22.00  33.00  66.00  49.
00
3   11.00  6.00  51.00  30.00  29.00  41.00  18.00  29.00  64.00  55.
00
4   11.00  6.00  55.00  32.00  31.00  42.00  18.00  32.00  63.00  44.
00
5   12.00  5.00  57.00  33.00  35.00  43.00  21.00  33.00  63.00  45.
00
6   11.00  5.00  60.00  31.00  34.00  52.00  23.00  36.00  56.00  40.
00
7   12.00  4.00  58.00  33.00  33.00  53.00  23.00  32.00  55.00  35.
00
8   13.00  4.00  60.00  32.00  40.00  51.00  23.00  35.00  57.00  34.
00
9   13.00  4.00  59.00  32.00  39.00  52.00  23.00  34.00  55.00  36.
00
10  14.00  4.00  60.00  34.00  36.00  51.00  21.00  35.00  56.00  39.
00
11  14.00  4.00  63.00  33.00  37.00  55.00  24.00  35.00  58.00  40.
00
12  15.00  5.00  60.00  36.00  41.00  54.00  25.00  35.00  58.00  38.
00
13  15.00  4.00  63.00  35.00  42.00  58.00  25.00  34.00  56.00  36.
00
14  15.00  4.00  61.00  39.00  45.00  60.00  26.00  36.00  55.00  36.
00
15  15.00  4.00  63.00  39.00  47.00  59.00  26.00  40.00  59.00  36.
00
16  15.00  4.00  65.00  40.00  45.00  61.00  26.00  37.00  62.00  38.
00
17  16.00  4.00  64.00  41.00  46.00  62.00  29.00  37.00  66.00  38.
00
18  16.00  4.00  63.00  40.00  48.00  66.00  32.00  40.00  62.00  41.
00
19  16.00  4.00  62.00  42.00  48.00  63.00  30.00  39.00  65.00  37.
00
The best value of K is 1
              precision    recall  f1-score   support

           0       0.94      0.98      0.96       980
           1       0.91      0.98      0.95      1135
           2       0.95      0.93      0.94      1032
           3       0.89      0.89      0.89      1010
           4       0.91      0.89      0.90       982
           5       0.90      0.87      0.88       892
           6       0.94      0.96      0.95       958
           7       0.92      0.92      0.92      1028
           8       0.91      0.84      0.87       974
           9       0.86      0.89      0.87      1009
```

```
    accuracy                           0.91     10000
   macro avg       0.91      0.91      0.91     10000
weighted avg       0.91      0.91      0.91     10000

The accuracy score for n= 8 is 0.9147
   Percentage of testing error for each digit
0                                    0.023469
1                                    0.016740
2                                    0.072674
3                                    0.108911
4                                    0.113035
5                                    0.134529
6                                    0.043841
7                                    0.084630
8                                    0.156057
9                                    0.112983
   Test error for each digit
0                          23
1                          19
2                          75
3                         110
4                         111
5                         120
6                          42
7                          87
8                         152
9                         114
CPU times: user 7min 19s, sys: 116 ms, total: 7min 19s
Wall time: 7min 19s
```

Comments :

1. The first table shows the values of testing errors for all digits (in rows) for different values of K (in columns).
2. We are taking the value of K that returned the minimum testing error. Here, the best value of K is found out to be 1.
3. We are then training our KNeighborsClassifier using this best value of k, and then evaluating the performance using the classification_report function, the output of which you can see in table 2.
4. Table 2 shows the precision of classification for every digit from 0 to 9 evaluated at best value of K.
5. Thus, the accuracy score is coming out to be 91.47%.
6. The total time required for the execution of the program and print the results is 7 mins & 19 seconds.

Part 3 case - 4 - Executing downsampling function at n = 16

Here, we are running the downsampling function for n = 16. The results and the conclusions are provided below.

In [14]:
```
%%time
downsample(16)
```

```
Factor of downsampling is 16
        0      1      2      3      4      5      6      7       8
9
0   27.00  13.00  66.00  81.00  80.00  88.00  26.00  74.00   93.00  1
04.00
1   15.00   6.00  62.00  66.00  59.00  92.00  40.00  65.00  120.00  1
36.00
2   14.00   8.00  64.00  68.00  70.00  90.00  28.00  64.00  102.00  1
01.00
3   22.00   6.00  64.00  67.00  60.00  85.00  30.00  53.00  101.00  1
08.00
4   20.00   6.00  70.00  63.00  62.00  88.00  27.00  53.00  101.00  1
00.00
5   20.00   8.00  68.00  66.00  59.00  86.00  28.00  57.00  101.00  1
01.00
6   16.00   7.00  69.00  65.00  64.00  88.00  26.00  59.00  100.00
94.00
7   21.00   7.00  70.00  64.00  57.00  90.00  28.00  55.00  101.00  1
02.00
8   22.00   7.00  72.00  67.00  59.00  87.00  27.00  55.00  105.00
92.00
9   24.00   8.00  70.00  61.00  61.00  85.00  26.00  60.00  106.00
93.00
10  25.00   8.00  72.00  67.00  65.00  89.00  27.00  57.00  103.00
90.00
11  26.00   8.00  73.00  66.00  65.00  91.00  30.00  56.00  104.00
91.00
12  25.00   8.00  73.00  65.00  68.00  83.00  30.00  55.00  105.00
88.00
13  25.00   9.00  78.00  65.00  68.00  84.00  30.00  56.00  105.00
90.00
14  25.00   9.00  75.00  68.00  66.00  83.00  27.00  57.00  106.00
86.00
15  27.00   9.00  79.00  70.00  68.00  85.00  28.00  56.00  105.00
88.00
16  25.00   9.00  79.00  68.00  70.00  86.00  26.00  62.00  103.00
87.00
17  25.00  10.00  82.00  70.00  70.00  90.00  26.00  59.00  105.00
91.00
18  25.00   8.00  83.00  75.00  68.00  86.00  26.00  60.00  104.00
91.00
19  26.00   8.00  85.00  73.00  67.00  85.00  28.00  60.00  103.00
89.00
The best value of K is 7
              precision    recall  f1-score   support

           0       0.90      0.93      0.91       980
           1       0.85      0.96      0.90      1135
           2       0.90      0.82      0.86      1032
           3       0.76      0.79      0.77      1010
           4       0.78      0.77      0.77       982
           5       0.76      0.71      0.73       892
           6       0.85      0.91      0.88       958
           7       0.79      0.82      0.80      1028
           8       0.83      0.69      0.75       974
           9       0.72      0.73      0.73      1009
```

```
    accuracy                           0.81     10000
   macro avg       0.81      0.81      0.81     10000
weighted avg       0.81      0.81      0.81     10000

The accuracy score for n= 16 is 0.8146
   Percentage of testing error for each digit
0                                    0.073469
1                                    0.037004
2                                    0.181202
3                                    0.213861
4                                    0.233198
5                                    0.289238
6                                    0.091858
7                                    0.183852
8                                    0.312115
9                                    0.266601
   Test error for each digit
0                         72
1                         42
2                        187
3                        216
4                        229
5                        258
6                         88
7                        189
8                        304
9                        269
CPU times: user 4min 54s, sys: 118 ms, total: 4min 54s
Wall time: 4min 52s
```

Comments :

1. The first table shows the values of testing errors for all digits (in rows) for different values of K (in columns).
2. We are taking the value of K that returned the minimum testing error. Here, the best value of K is found out to be 7.
3. We are then training our KNeighborsClassifier using this best value of k, and then evaluating the performance using the classification_report function, the output of which you can see in table 2.
4. Table 2 shows the precision of classification for every digit from 0 to 9 evaluated at best value of K.
5. Thus, the accuracy score is coming out to be 81.46%.
6. The total time required for the execution of the program and print the results is 4 mins & 52 seconds.

Comparison of precision and query time for different downsampling factors

Comments:

1. As the downsampling factor (n) increases, accuracy decreases
2. Also, the query time reduces with increase in downsampling factor (n) Thus there is a clear tradeoff between the accuracy and query time with increase in down sampling factor.

A plot for the comparison of precision and quer time one of the runs is shown below.

NOTE: The below graph is representative and it is not meant to show the results of the current run. That is with each run the values presented in the graph might change which does not reflect in the graph.

In [47]:
```
N = np.array([2, 4, 8, 16])
T = np.array([1673, 939, 561, 335])
P = np.array([96.41, 93.32, 92.47, 81.66])
fig = plt.figure(figsize=(9,9))
plt.plot(N, P, 'ko-')
plt.xlabel('downsampling factor (N)',fontweight='bold', fontsize='12'
,color = 'blue')
plt.ylabel('Precision (%)',fontweight='bold', fontsize='12',color =
'blue')
plt.xticks(np.arange(0, 24, 4))
plt.yticks(np.arange(70, 102, 2))
plt.title('Precision vs downsampling factor',fontweight='bold', fonts
ize='16')
fig.text(.5, .05, "Fig: Comparison of Precision for different downsam
pling factor",fontweight='bold', fontsize='16',ha='center')

fig = plt.figure(figsize=(9,9))
plt.plot(N, T, 'ko-')
plt.xlabel('downsampling factor (N)',fontweight='bold', fontsize='12'
,color = 'blue')
plt.ylabel('Query Time (secs)',fontweight='bold', fontsize='12',color
= 'blue')
plt.xticks(np.arange(0, 24, 4))
plt.yticks(np.arange(300,2400, 300))
plt.title('Query Time vs downsampling factor',fontweight='bold', font
size='16')
fig.text(.5, .05, "Fig: Comparison of query time for different downsa
mpling factor",fontweight='bold', fontsize='16',ha='center')
```

Out[47]: Text(0.5, 0.05, 'Fig: Comparison of query time for different downsamp
ling factor')

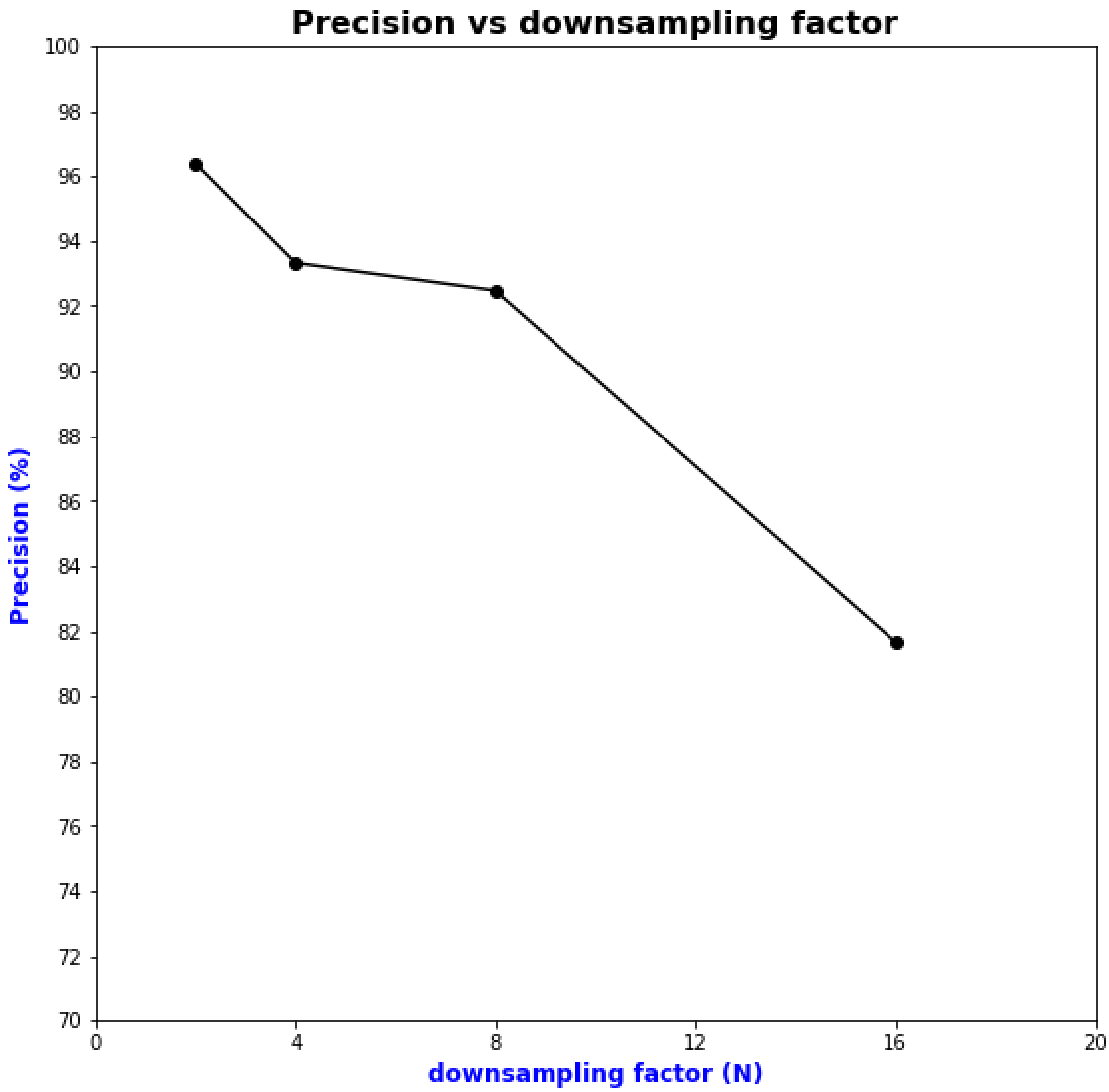


Fig: Comparison of Precision for different downsampling factor

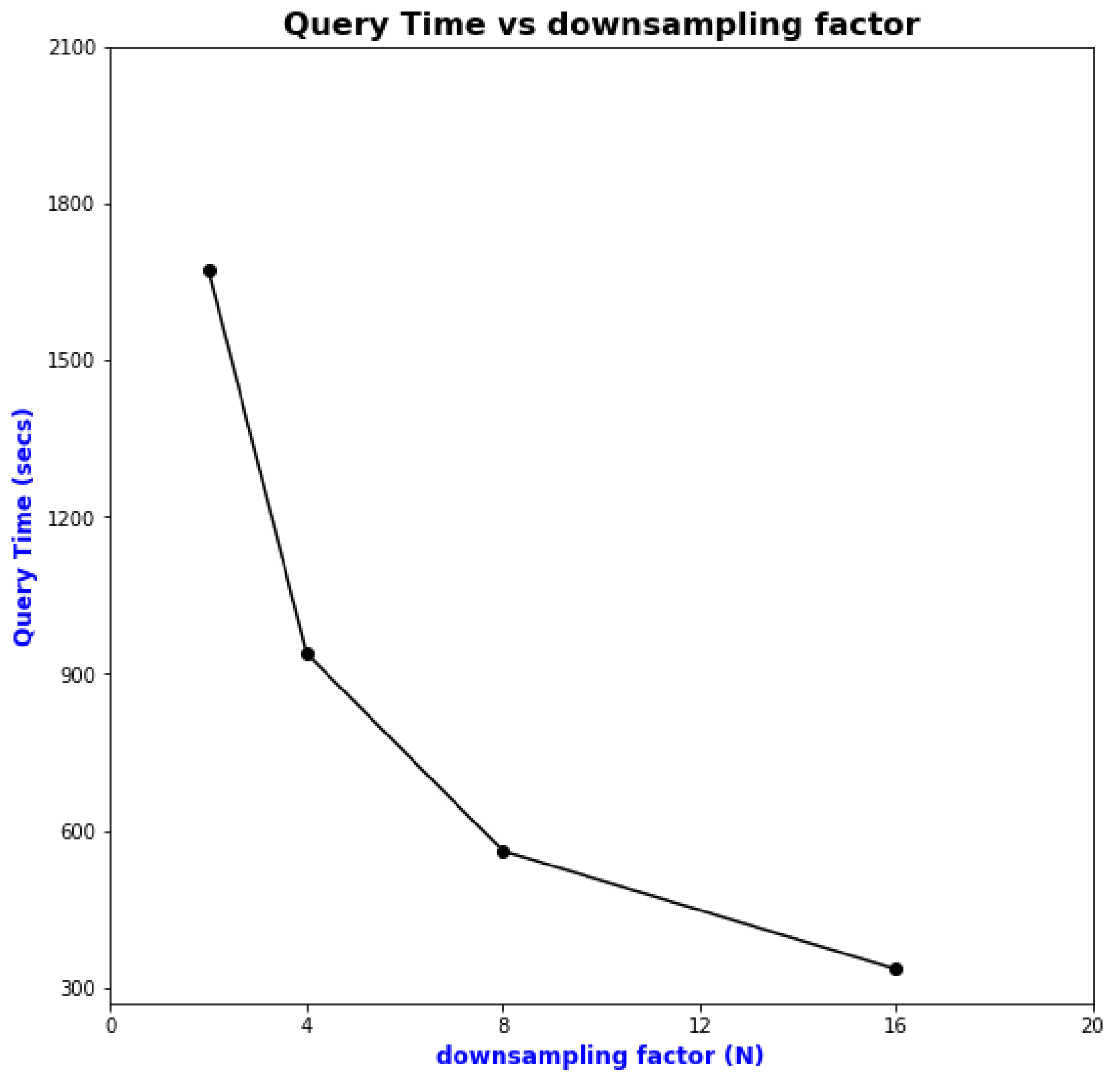


Fig: Comparison of query time for different downsampling factor

Part 4 - Implementing a smart downsampling function

In this part, We are going to implement a function that smart downsamples the image by binning nearby pixels. For example, if n is 4 then 28 x 28 image will be binned down to a 7 x 7 image by summing each 4x4 block in the image. In this function, we are repeating the KNN leave-one-out experiment. This function is run for 4 different values of n, results of which are present in subsequent chapters. Since it takes longer time to train the model on 60000 training images, we are randomly sampling 2000 points from the dataset.

Notes :

1. As we are randomly sampling the training set, the best value of K for every run changes which changes the output values printed in every execution of downsampling function.
2. The value of query time printed for every run depends largly on the CPU processor and thus may vary for different processing units.
3. The values shown here are obtained when the program is run on intel(R)xenon(R) e3-1245 quad-core processor.

In [16]:
```
train=train_img.flatten()
X_train=train.reshape(60000,784)
y_train=train_lab[:]
x_test=test.reshape(10000,784)
y_test=test_lab[:]


def smart_downsample(n):   # define a smart downsampling function
    print('Factor of smart downsampling is',n)  # Print the factor of smart downsampling

    a1=[]
    f1=[]
    a2=[]
    f2=[]

    for k in range(len(X_train)):  # k goes from 0 to length of training features
        X_t=X_train[k].reshape(28,28)   # Reshape every training feature in 28x28 matrix
        for i in range(int((X_t.shape[0])/n)):  # i goes from 0 to number of training points divided by downsampling factor
            for j in range(int(X_t.shape[0]/n)): # j goes from 0 to number of training points divided by downsampling factor
                f1=X_t[i*n:(i+1)*n,j*n:(j+1)*n].sum()  # create a downsampled f2 matrix for training features
                a1.append(f1)    # fill the matrix a1 with downsampled training data
    bin_n=np.array(a1).reshape(60000,int(len(X_t)/n)*int(len(X_t)/n)) # reshape the a1 matrix to 28/n*28/n

    train_bin_n_combin=np.column_stack((bin_n, y_train))   # Combine the downsampled train features with training labels
    np.random.shuffle(train_bin_n_combin)     # randomly shuffle the combined matrix
    train_bin_n_data=pd.DataFrame(train_bin_n_combin)
    train_bin_n_sample=train_bin_n_data.sample(2000)   # randomly choose 2000 data points from combined data points
    X_train_bin_n=np.array(train_bin_n_sample.drop([train_bin_n_sample.shape[1]-1], axis=1))  # separate training features from downsampled datapoints
    y_train_bin_n=np.array(train_bin_n_sample[train_bin_n_sample.shape[1]-1])  # separate training labels from downsampled datapoints


    for k in range(len(x_test)):   # k goes from 0 to number of test features
        X_ts=x_test[k].reshape(28,28)  # reshape each test feature in 28*28 matrix
        for i in range(int((X_ts.shape[0])/n)):  # i goes from 0 to 28/n
            for j in range(int(X_ts.shape[0]/n)):  # j goes from 0 to 28/n
                f2=X_ts[i*2:(i+1)*n,j*n:(j+1)*n].sum()   # create a downsampled f2 matrix for testing features
                a2.append(f2)    # fill the matrix a2 with downsample
```

```
d testing data
    bin_n_ts=np.array(a2).reshape(10000,int(len(X_ts)/n)*int(len(X_ts
)/n))  # reshape the a2 matrix to 28/n*28/n

    test_bin_n_combin=np.column_stack((bin_n_ts, y_test))   # Combine
the downsampled test features with test labels
    np.random.shuffle(test_bin_n_combin)   # randomly shuffle the com
bined matrix
    test_bin_n_data=pd.DataFrame(test_bin_n_combin)
    test_bin_n_sample=test_bin_n_data.sample(2000)   # randomly choos
e 2000 data points from combined data points
    X_test_bin_n=np.array(test_bin_n_sample.drop([test_bin_n_sample.s
hape[1]-1], axis=1)) # separate testing features from downsampled dat
apoints
    y_test_bin_n=np.array(test_bin_n_sample[test_bin_n_sample.shape[1
]-1])  # separate testing labels from downsampled datapoints


    k_one_sample_bin_n=leave_one_out(X_train_bin_n,y_train_bin_n)  #
 Implement leave-one-out function on downsampled training data
    k_one_sample_bin_n_table=pd.DataFrame(k_one_sample_bin_n)
    k_one_sample_bin_n_format = k_one_sample_bin_n_table.applymap("
{0:.2f}".format)
    print(k_one_sample_bin_n_format)

    m1=[]
    for k in range(20):    # K goes from 0 to 19
        c1=np.array(k_one_sample_bin_n_table.iloc[k]) # combine the e
rrors for K 0 to 19 for all digits 0 to 9 in an array
        m1.append(np.mean(c1))    # Calculate the mean of errors of a
ll digits for each K
    print('The best value of K is',np.argmin(m1)+1)   # Print the bes
t value of K as the one which gives min error for all digits


    Knn = KNeighborsClassifier(n_neighbors=(np.argmin(m1)+1))   # Imp
lement KNN classifier with best K value
    Knn.fit(bin_n, y_train)                                    # fit
 the KNN model on binned training data
    y_pred_bin = Knn.predict(bin_n_ts)                             # P
redict the class of binned test feature
    print(classification_report(y_test, y_pred))              # Print
the classification report of predicted and actual labels of test feat
ures
    score_bin=Knn.score(bin_n_ts,y_test)

    #Prediction of test error
    test_error_recall=[]
    Err_test=[]
    for i in range(10):
        sample_test=y_test
        test_data=pd.DataFrame(sample_test)
        i1=test_data[test_data[0]==i]
        indx=list(i1.index)
        err=1-((y_pred_bin[indx] ==i).sum()/(y_test==i).sum())
        test_error_recall.append(err)
        Err=(y_pred_bin[indx] !=i).sum()
```

```
        Err_test.append(Err)
    h2=pd.DataFrame(test_error_recall, columns=['Percentage of testin
g error for each digit']) # % of test error for each digit
    g2=pd.DataFrame(Err_test, columns=['Test error for each digit'])
#Number of times the digits were wrongly classified



    print(h2)
    print(g2)


    print('The accuracy score for n=',n,'is',score_bin)
# Print the score
    return
```

Part 4 case - 1 - Executing smart downsampling function at n = 2

Here, we are running the smart downsampling function for n = 2. The results and the conclusions are provided below.

In [17]:
```
%%time
smart_downsample(2)   # Run smart donwsample function at n = 2
```

```
Factor of smart downsampling is 2
       0     1      2      3      4      5      6      7      8
9
0   6.00  2.00  18.00  13.00  20.00  25.00   6.00  16.00  26.00  29.0
0
1   2.00  1.00  13.00  10.00  14.00  33.00  15.00  20.00  34.00  51.0
0
2   3.00  2.00  16.00  13.00  17.00  30.00  12.00  18.00  26.00  27.0
0
3   2.00  3.00  15.00  11.00  17.00  28.00  15.00  16.00  28.00  36.0
0
4   2.00  3.00  13.00  16.00  21.00  21.00  12.00  16.00  29.00  28.0
0
5   2.00  3.00  15.00  14.00  21.00  29.00  11.00  17.00  32.00  28.0
0
6   2.00  3.00  15.00  15.00  24.00  26.00  10.00  17.00  33.00  26.0
0
7   2.00  2.00  20.00  12.00  24.00  31.00  14.00  16.00  32.00  31.0
0
8   2.00  2.00  20.00  14.00  23.00  29.00  12.00  17.00  32.00  27.0
0
9   2.00  2.00  24.00  12.00  23.00  31.00  15.00  16.00  34.00  31.0
0
10  2.00  2.00  25.00  12.00  26.00  31.00  16.00  17.00  31.00  27.0
0
11  2.00  2.00  25.00  11.00  29.00  36.00  16.00  19.00  28.00  31.0
0
12  3.00  3.00  29.00  13.00  26.00  33.00  16.00  18.00  29.00  29.0
0
13  3.00  3.00  29.00  12.00  27.00  37.00  16.00  18.00  30.00  29.0
0
14  3.00  3.00  29.00  14.00  31.00  36.00  15.00  17.00  30.00  30.0
0
15  3.00  2.00  29.00  15.00  26.00  37.00  16.00  17.00  30.00  32.0
0
16  3.00  2.00  28.00  14.00  27.00  41.00  16.00  18.00  31.00  30.0
0
17  3.00  2.00  28.00  14.00  24.00  39.00  15.00  19.00  32.00  32.0
0
18  3.00  2.00  27.00  15.00  30.00  35.00  17.00  19.00  32.00  32.0
0
19  3.00  2.00  30.00  14.00  28.00  38.00  15.00  19.00  32.00  33.0
0
The best value of K is 1
              precision    recall  f1-score   support

           0       0.98      0.99      0.99       980
           1       0.97      0.99      0.98      1135
           2       0.98      0.96      0.97      1032
           3       0.96      0.96      0.96      1010
           4       0.97      0.96      0.97       982
           5       0.95      0.96      0.96       892
           6       0.98      0.99      0.98       958
           7       0.96      0.96      0.96      1028
           8       0.98      0.94      0.96       974
           9       0.96      0.96      0.96      1009
```

```
    accuracy                           0.97     10000
   macro avg       0.97      0.97      0.97     10000
weighted avg       0.97      0.97      0.97     10000

   Percentage of testing error for each digit
0                                    0.009184
1                                    0.003524
2                                    0.028101
3                                    0.043564
4                                    0.033605
5                                    0.030269
6                                    0.014614
7                                    0.036965
8                                    0.046201
9                                    0.039643
   Test error for each digit
0                          9
1                          4
2                         29
3                         44
4                         33
5                         27
6                         14
7                         38
8                         45
9                         40
The accuracy score for n= 2 is 0.9717
CPU times: user 41min 28s, sys: 58 s, total: 42min 26s
Wall time: 14min 46s
```

Comments :

1. The first table shows the values of testing errors for all digits (in rows) for different values of K (in columns).
2. We are taking the value of K that returned the minimum testing error. Here, the best value of K is found out to be 1.
3. We are then training our KNeighborsClassifier using this best value of k, and then evaluating the performance using the classification_report function, the output of which you can see in table 2.
4. Table 2 shows the precision of classification for every digit from 0 to 9 evaluated at best value of K.
5. Thus, the accuracy score is coming out to be 97.17%.
6. The total time required for the execution of the program and print the results is 14 mins & 46 seconds.

Part 4 case - 2 - Executing smart downsampling function at n = 4

Here, we are running the smart downsampling function for n = 4. The results and the conclusions are provided below.

In [18]:
```
%%time
smart_downsample(4)   # Run smart donwsample function at n = 4
```

```
Factor of smart downsampling is 4
       0     1      2      3      4      5      6      7      8
9
0   4.00  3.00  20.00  36.00  34.00  25.00  11.00  21.00  39.00  27.0
0
1   2.00  3.00  10.00  28.00  23.00  29.00  20.00  20.00  56.00  52.0
0
2   1.00  4.00  16.00  37.00  31.00  27.00  10.00  22.00  40.00  24.0
0
3   4.00  4.00  16.00  32.00  25.00  23.00  10.00  21.00  43.00  31.0
0
4   3.00  4.00  22.00  34.00  36.00  21.00   9.00  18.00  39.00  26.0
0
5   5.00  4.00  20.00  29.00  28.00  22.00  11.00  18.00  41.00  25.0
0
6   4.00  4.00  24.00  32.00  36.00  23.00  10.00  17.00  39.00  23.0
0
7   3.00  4.00  26.00  30.00  36.00  23.00  11.00  17.00  38.00  21.0
0
8   4.00  2.00  28.00  30.00  41.00  23.00  12.00  19.00  39.00  21.0
0
9   4.00  2.00  27.00  30.00  38.00  25.00  13.00  21.00  42.00  21.0
0
10  5.00  2.00  28.00  28.00  39.00  23.00  12.00  19.00  43.00  24.0
0
11  5.00  2.00  29.00  31.00  42.00  24.00  12.00  19.00  44.00  22.0
0
12  5.00  2.00  28.00  30.00  39.00  25.00  13.00  22.00  44.00  22.0
0
13  5.00  2.00  30.00  27.00  38.00  25.00  12.00  21.00  45.00  20.0
0
14  7.00  2.00  28.00  31.00  39.00  26.00  12.00  23.00  45.00  21.0
0
15  7.00  2.00  33.00  32.00  43.00  27.00  13.00  23.00  44.00  22.0
0
16  8.00  2.00  31.00  28.00  45.00  27.00  13.00  23.00  45.00  22.0
0
17  8.00  2.00  32.00  28.00  45.00  26.00  14.00  23.00  45.00  22.0
0
18  8.00  2.00  33.00  26.00  48.00  26.00  16.00  23.00  45.00  24.0
0
19  8.00  2.00  34.00  28.00  47.00  28.00  16.00  24.00  46.00  25.0
0
The best value of K is 6
              precision    recall  f1-score   support

           0       0.98      0.99      0.99       980
           1       0.97      0.99      0.98      1135
           2       0.98      0.96      0.97      1032
           3       0.96      0.96      0.96      1010
           4       0.97      0.96      0.97       982
           5       0.95      0.96      0.96       892
           6       0.98      0.99      0.98       958
           7       0.96      0.96      0.96      1028
           8       0.98      0.94      0.96       974
           9       0.96      0.96      0.96      1009
```

```
    accuracy                           0.97     10000
   macro avg       0.97      0.97      0.97     10000
weighted avg       0.97      0.97      0.97     10000

   Percentage of testing error for each digit
0                                    0.155102
1                                    0.569163
2                                    0.904070
3                                    0.791089
4                                    0.870672
5                                    0.983184
6                                    0.938413
7                                    0.798638
8                                    0.120123
9                                    0.850347
   Test error for each digit
0                        152
1                        646
2                        933
3                        799
4                        855
5                        877
6                        899
7                        821
8                        117
9                        858
The accuracy score for n= 4 is 0.3043
CPU times: user 3min 35s, sys: 91.1 ms, total: 3min 35s
Wall time: 3min 34s
```

Comments :

1. The first table shows the values of testing errors for all digits (in rows) for different values of K (in columns).
2. We are taking the value of K that returned the minimum testing error. Here, the best value of K is found out to be 6.
3. We are then training our KNeighborsClassifier using this best value of k, and then evaluating the performance using the classification_report function, the output of which you can see in table 2.
4. Table 2 shows the precision of classification for every digit from 0 to 9 evaluated at best value of K.
5. Thus, the accuracy score is coming out to be 30.43%.
6. The total time required for the execution of the program and print the results is 3 mins & 34 seconds.

Part 4 case - 3 - Executing smart downsampling function at n = 7

Here, we are running the smart downsampling function for n = 7. The results and the conclusions are provided below.

In [19]:
```
%%time
smart_downsample(7)   # Run smart donwsample function at n = 7
```

```
Factor of smart downsampling is 7
        0      1      2      3      4      5      6      7       8
9
0   63.00  19.00  54.00  61.00  63.00  77.00  19.00  50.00   76.00
72.00
1   35.00  15.00  44.00  79.00  46.00  84.00  41.00  46.00  112.00  1
25.00
2   45.00  14.00  49.00  61.00  60.00  78.00  29.00  51.00   82.00
81.00
3   45.00  15.00  51.00  53.00  50.00  79.00  26.00  45.00   83.00
77.00
4   52.00  14.00  49.00  51.00  48.00  74.00  23.00  44.00   77.00
77.00
5   47.00  12.00  53.00  51.00  54.00  75.00  26.00  45.00   75.00
76.00
6   47.00  12.00  55.00  50.00  53.00  75.00  23.00  40.00   69.00
73.00
7   50.00  12.00  55.00  53.00  54.00  82.00  22.00  39.00   77.00
75.00
8   48.00  10.00  60.00  54.00  56.00  74.00  23.00  41.00   76.00
75.00
9   50.00  10.00  61.00  56.00  52.00  71.00  21.00  43.00   70.00
78.00
10  49.00  12.00  65.00  56.00  50.00  77.00  23.00  39.00   70.00
79.00
11  49.00  12.00  66.00  53.00  51.00  79.00  21.00  38.00   69.00
76.00
12  49.00  12.00  67.00  54.00  49.00  78.00  21.00  41.00   69.00
76.00
13  49.00  16.00  66.00  54.00  52.00  80.00  22.00  40.00   71.00
76.00
14  51.00  14.00  66.00  56.00  49.00  81.00  21.00  44.00   71.00
76.00
15  51.00  15.00  63.00  57.00  47.00  83.00  21.00  44.00   70.00
76.00
16  52.00  15.00  67.00  56.00  49.00  79.00  21.00  45.00   70.00
76.00
17  55.00  13.00  66.00  57.00  48.00  82.00  22.00  42.00   68.00
75.00
18  54.00  13.00  67.00  56.00  49.00  82.00  20.00  43.00   69.00
77.00
19  57.00  15.00  67.00  58.00  52.00  85.00  20.00  44.00   62.00
73.00
The best value of K is 7
              precision    recall  f1-score   support

           0       0.98      0.99      0.99       980
           1       0.97      0.99      0.98      1135
           2       0.98      0.96      0.97      1032
           3       0.96      0.96      0.96      1010
           4       0.97      0.96      0.97       982
           5       0.95      0.96      0.96       892
           6       0.98      0.99      0.98       958
           7       0.96      0.96      0.96      1028
           8       0.98      0.94      0.96       974
           9       0.96      0.96      0.96      1009
```

```
     accuracy                           0.97     10000
    macro avg       0.97      0.97      0.97     10000
 weighted avg       0.97      0.97      0.97     10000

   Percentage of testing error for each digit
0                                     0.608163
1                                     0.993833
2                                     0.653101
3                                     0.287129
4                                     0.996945
5                                     0.408072
6                                     0.998956
7                                     1.000000
8                                     0.728953
9                                     1.000000
   Test error for each digit
0                          596
1                         1128
2                          674
3                          290
4                          979
5                          364
6                          957
7                         1028
8                          710
9                         1009
The accuracy score for n= 7 is 0.2265
CPU times: user 1min 6s, sys: 76 ms, total: 1min 6s
Wall time: 1min 4s
```

Comments :

1. The first table shows the values of testing errors for all digits (in rows) for different values of K (in columns).
2. We are taking the value of K that returned the minimum testing error. Here, the best value of K is found out to be 7.
3. We are then training our KNeighborsClassifier using this best value of k, and then evaluating the performance using the classification_report function, the output of which you can see in table 2.
4. Table 2 shows the precision of classification for every digit from 0 to 9 evaluated at best value of K.
5. Thus, the accuracy score is coming out to be 22.65%.
6. The total time required for the execution of the program and print the results is 1 mins & 4 seconds.

Part 4 case - 4 - Executing smart downsampling function at n = 14

Here, we are running the smart downsampling function for n = 14. The results and the conclusions are provided below.

In [20]:
```
%%time
smart_downsample(14)   # Run smart donwsample function at n = 14
```

```
Factor of smart downsampling is 14
         0      1       2       3       4       5       6       7      
8  \
0   108.00  41.00  115.00  145.00  138.00  151.00  105.00  110.00  14
7.00   
1    67.00  29.00   92.00  125.00  132.00  143.00  134.00  115.00  17
0.00   
2    56.00  27.00  103.00  131.00  147.00  152.00  117.00  112.00  16
7.00   
3    56.00  29.00   98.00  128.00  137.00  144.00  106.00  101.00  16
3.00   
4    61.00  29.00  101.00  133.00  128.00  143.00  104.00  100.00  16
2.00   
5    51.00  28.00  102.00  131.00  127.00  142.00  106.00   98.00  15
9.00   
6    49.00  29.00  100.00  133.00  129.00  145.00  106.00   97.00  16
4.00   
7    56.00  26.00  101.00  134.00  128.00  147.00  100.00   95.00  16
4.00   
8    57.00  25.00  106.00  135.00  119.00  142.00  107.00   91.00  16
5.00   
9    54.00  26.00  102.00  130.00  124.00  137.00  105.00   89.00  16
1.00   
10   62.00  25.00   99.00  132.00  127.00  134.00  106.00   88.00  16
2.00   
11   62.00  25.00  100.00  131.00  124.00  136.00  105.00   88.00  16
1.00   
12   57.00  27.00   96.00  133.00  123.00  136.00  106.00   92.00  15
9.00   
13   53.00  26.00  104.00  125.00  125.00  135.00  107.00   85.00  16
1.00   
14   54.00  24.00  103.00  128.00  126.00  135.00  106.00   86.00  16
0.00   
15   52.00  26.00  108.00  133.00  116.00  138.00  108.00   82.00  15
9.00   
16   52.00  24.00  101.00  136.00  118.00  140.00  103.00   84.00  16
2.00   
17   56.00  22.00  103.00  134.00  115.00  141.00  103.00   82.00  15
8.00   
18   53.00  24.00  102.00  134.00  118.00  145.00  103.00   85.00  16
0.00   
19   59.00  23.00  103.00  141.00  113.00  141.00  102.00   88.00  16
1.00   

         9  
0   158.00  
1   186.00  
2   176.00  
3   166.00  
4   163.00  
5   170.00  
6   167.00  
7   167.00  
8   171.00  
9   168.00  
10  168.00  
11  168.00  
```

```
12  166.00
13  171.00
14  170.00
15  173.00
16  168.00
17  168.00
18  167.00
19  167.00
The best value of K is 18
              precision    recall  f1-score   support

           0       0.98      0.99      0.99       980
           1       0.97      0.99      0.98      1135
           2       0.98      0.96      0.97      1032
           3       0.96      0.96      0.96      1010
           4       0.97      0.96      0.97       982
           5       0.95      0.96      0.96       892
           6       0.98      0.99      0.98       958
           7       0.96      0.96      0.96      1028
           8       0.98      0.94      0.96       974
           9       0.96      0.96      0.96      1009

    accuracy                           0.97     10000
   macro avg       0.97      0.97      0.97     10000
weighted avg       0.97      0.97      0.97     10000

   Percentage of testing error for each digit
0                                    0.894898
1                                    0.906608
2                                    0.469961
3                                    1.000000
4                                    0.889002
5                                    1.000000
6                                    0.239040
7                                    1.000000
8                                    1.000000
9                                    1.000000
   Test error for each digit
0                        877
1                       1029
2                        485
3                       1010
4                        873
5                        892
6                        229
7                       1028
8                        974
9                       1009
The accuracy score for n= 14 is 0.1594
CPU times: user 42.9 s, sys: 87 ms, total: 43 s
Wall time: 41 s
```

Comments :

1. The first table shows the values of testing errors for all digits (in rows) for different values of K (in columns).
2. We are taking the value of K that returned the minimum testing error. Here, the best value of K is found out to be 18.
3. We are then training our KNeighborsClassifier using this best value of k, and then evaluating the performance using the classification_report function, the output of which you can see in table 2.
4. Table 2 shows the precision of classification for every digit from 0 to 9 evaluated at best value of K.
5. Thus, the accuracy score is coming out to be 15.94%.
6. The total time required for the execution of the program and print the results is 41 seconds.

Part 5 - Running a smart downsampling function at n = 28

In [21]:
```
%%time
smart_downsample(28)   # Run smart donwsample function at n = 28
```

```
Factor of smart downsampling is 28
         0       1       2       3       4       5       6       7       
8  \
0   152.00  133.00  182.00  167.00  176.00  175.00  185.00  165.00  1
69.00
1   123.00   91.00  168.00  160.00  179.00  173.00  191.00  176.00  1
72.00
2   111.00   79.00  169.00  170.00  175.00  178.00  187.00  182.00  1
74.00
3   116.00   76.00  171.00  177.00  181.00  173.00  185.00  181.00  1
70.00
4   117.00   83.00  169.00  177.00  184.00  174.00  184.00  181.00  1
70.00
5   115.00   78.00  173.00  173.00  185.00  173.00  179.00  180.00  1
68.00
6   116.00   73.00  166.00  172.00  179.00  171.00  180.00  172.00  1
68.00
7   116.00   64.00  159.00  165.00  176.00  175.00  184.00  171.00  1
68.00
8   121.00   63.00  153.00  165.00  178.00  174.00  188.00  167.00  1
67.00
9   117.00   63.00  157.00  169.00  182.00  174.00  187.00  167.00  1
64.00
10  107.00   63.00  156.00  170.00  182.00  175.00  183.00  167.00  1
64.00
11  113.00   65.00  160.00  172.00  181.00  172.00  179.00  162.00  1
66.00
12  115.00   60.00  161.00  173.00  176.00  173.00  175.00  157.00  1
63.00
13  111.00   58.00  160.00  174.00  177.00  174.00  176.00  159.00  1
60.00
14  108.00   57.00  158.00  174.00  179.00  172.00  175.00  162.00  1
62.00
15  108.00   62.00  166.00  177.00  181.00  169.00  177.00  164.00  1
64.00
16  103.00   65.00  164.00  177.00  179.00  171.00  175.00  165.00  1
62.00
17  106.00   62.00  167.00  179.00  180.00  173.00  175.00  168.00  1
61.00
18  108.00   56.00  165.00  178.00  180.00  173.00  174.00  168.00  1
59.00
19  108.00   60.00  166.00  178.00  178.00  172.00  175.00  164.00  1
58.00

         9  
0   175.00  
1   203.00  
2   192.00  
3   188.00  
4   181.00  
5   185.00  
6   185.00  
7   178.00  
8   179.00  
9   175.00  
10  167.00  
11  166.00  
```

```
12  167.00
13  167.00
14  165.00
15  163.00
16  161.00
17  161.00
18  158.00
19  159.00
The best value of K is 15
              precision    recall  f1-score   support

           0       0.98      0.99      0.99       980
           1       0.97      0.99      0.98      1135
           2       0.98      0.96      0.97      1032
           3       0.96      0.96      0.96      1010
           4       0.97      0.96      0.97       982
           5       0.95      0.96      0.96       892
           6       0.98      0.99      0.98       958
           7       0.96      0.96      0.96      1028
           8       0.98      0.94      0.96       974
           9       0.96      0.96      0.96      1009

    accuracy                           0.97     10000
   macro avg       0.97      0.97      0.97     10000
weighted avg       0.97      0.97      0.97     10000

   Percentage of testing error for each digit
0                                    0.590816
1                                    0.271366
2                                    0.852713
3                                    0.888119
4                                    0.872709
5                                    0.931614
6                                    0.933194
7                                    0.874514
8                                    0.921971
9                                    0.911794
   Test error for each digit
0                        579
1                        308
2                        880
3                        897
4                        857
5                        831
6                        894
7                        899
8                        898
9                        920
The accuracy score for n= 28 is 0.2037
CPU times: user 46.1 s, sys: 79.2 ms, total: 46.2 s
Wall time: 44.3 s
```

Comments :

1. The first table shows the values of testing errors for all digits (in rows) for different values of K (in columns).
2. We are taking the value of K that returned the minimum testing error. Here, the best value of K is found out to be 15.
3. We are then training our KNeighborsClassifier using this best value of k, and then evaluating the performance using the classification_report function, the output of which you can see in table 2.
4. Table 2 shows the precision of classification for every digit from 0 to 9 evaluated at best value of K.
5. Thus, the accuracy score is coming out to be 20.37%.
6. The total time required for the execution of the program and print the results is 44.3 seconds.

Chapter 6 - Feature transformation methods

6.1 What is a feature transformation?

Feature transformation is a function that transforms features from one representation to another.

6.2 Why do we need a feature transformation?

Some of the reasons for implementing the feauture transformation are as below:
1.Data types are not suitable to be fed into a machine learning algorithm, e.g. text, categories
2.Feature values may cause problems during the learning process, e.g. data represented in different scales
3.We want to reduce the number of features to plot and visualize data, speed up training or improve the accuracy of a specific model

6.3 In this project, we are going to discuss two feature transformation methods which are:

1. Standard Scalar or Feature Standardization
2. MinMax Scalar or Unity normalization or Feature normalization

6.3.1 Standard Scalar or Feature Standardization

Standardization typically means rescaling the data to have a mean of 0 and a standard deviation of 1 (unit variance). The distribution of pixel values often follows a Normal or Gaussian distribution, e.g. bell shape.This distribution may be present per image, per mini-batch of images, or across the training dataset. As such, there may be benefit in transforming the distribution of pixel values to be a standard Gaussian: that is both centering the pixel values on zero and normalizing the values by the standard deviation. The result is a standard Gaussian of pixel values with a mean of 0.0 and a standard deviation of 1.0. As with centering, the operation can be performed per image, per mini-batch, and across the entire training dataset, and it can be performed globally across channels or locally per channel. Standardization may be preferred to normalization and centering alone and it results in both zero-centered values of small input values, roughly in the range -3 to 3, depending on the specifics of the dataset.For consistency of the input data, it may make more sense to standardize images per-channel using statistics calculated per mini-batch or across the training dataset, if possible.

Standardization (or Z-score normalization) is the process of rescaling the features so that they'll have the properties of a Gaussian distribution with, $\mu=0$ and $\sigma=1$ where μ is the mean and σ is the standard deviation from the mean; standard scores (also called z scores) of the samples are calculated as follows:

$z=(x-\mu)/\sigma$

Steps to implement Feature Standardization

1. From sklearn.preprocessing library import StandardScaler module
2. Create a scaling object of StandardScaler module.
3. Pass the training and testing data (features) to fit this scaling object to scale down values considering mean=0 and standard deviation=1 (Gaussian Normal distribution).

6.3.1.1 Code for using standard scalar method with KNN leave-one-out experiment

Below mentioned code implements the standard scalar feature transformation method and here it is applied to the same dataset on which we have run the part 1 to 5.

In [33]:
```
%%time
from sklearn import preprocessing                           #From sklear
n.preprocessing library import StandardScaler module
std_scale=preprocessing.StandardScaler().fit(X_train)  #Create a scal
ing object of StandardScaler module.
X_train_std=std_scale.transform(X_train)                       #Pass the
training and testing data (features) to fit this scaling object
X_test_std=std_scale.transform(x_test)


train_std_combin=np.column_stack((X_train_std, y_train))
np.random.shuffle(train_std_combin)
train_std_data=pd.DataFrame(train_std_combin)
train_std_sample=train_std_data.sample(2000)
X_train_s=np.array(train_std_sample.drop([train_std_sample.shape[1]-1
], axis=1))
y_train_s1=np.array(train_std_sample[train_std_sample.shape[1]-1])
y_train_s = y_train_s1.astype(int)


test_std_combin=np.column_stack((X_test_std, y_test))
np.random.shuffle(test_std_combin)
test_std_data=pd.DataFrame(test_std_combin)
test_std_sample=test_std_data.sample(2000)
X_test_s=np.array(test_std_sample.drop([test_std_sample.shape[1]-1],
axis=1))
y_test_s=np.array(test_std_sample[test_std_sample.shape[1]-1])



err_tmp = np.zeros(10)
from sklearn.neighbors import KNeighborsClassifier

def leave_one_out(X_train,y_train):
    Err = np.empty([1, 10])
    for K in range(1, 20):
        knn = KNeighborsClassifier(n_neighbors=K)
        err_tmp = np.zeros([1,10])
        for j in range(len(X_train)):
            k_test=X_train[j]
            k_test = np.array([k_test])
            k_valid=y_train[j]
            k_train_input=np.delete(X_train, j, 0)
            k_train_outputs=np.delete(y_train, j, 0)
            knn.fit(k_train_input, k_train_outputs)
            pred_K = knn.predict(k_test)
            err_tmp[0,k_valid] = err_tmp[0,k_valid] + (pred_K != k_va
lid)
            #if(j%100==0):
                #print(j)
        Err = np.append(Err,err_tmp,axis=0)

    return(Err)


k_one_sample_s_table=pd.DataFrame(leave_one_out(X_train_s,y_train_s))
```

```
print(dp)

m=[]
for i in range(20):
    c=np.array(k_one_sample_s_table.iloc[i])
    m.append(np.mean(c))
print('The best value of K is',np.argmin(m)+1)

## KNN classification algorithm
Knn = KNeighborsClassifier(n_neighbors=1)
Knn.fit(X_train_std, y_train)
y_pred = Knn.predict(X_test_std)

Err=[]                                                  #Prediction of t
est error for each digit
for i in range(10):
    total_pred=(y_pred==i).sum()
    total_n=(y_test==i).sum()
    test_err=abs(total_n-total_pred)
    Err.append(test_err)
h=pd.DataFrame(Err, columns=['Test error for each digit'])
score=Knn.score(X_test_std,y_test)

print(classification_report(y_test, y_pred))
print('The accuracy score is',score)
```

```
[[ 3.  2. 26. 20. 19. 17.  5. 13. 35. 23.]
 [ 3.  1. 28. 17. 10. 24.  9. 17. 50. 46.]
 [ 3.  1. 32. 22. 17. 14.  4. 17. 43. 24.]
 [ 5.  1. 31. 21. 14. 17.  2. 12. 43. 38.]
 [ 3.  2. 31. 22. 19. 15.  3. 16. 40. 27.]
 [ 4.  2. 35. 22. 15. 18.  5. 21. 40. 33.]
 [ 5.  2. 37. 23. 19. 16.  5. 19. 40. 26.]
 [ 4.  2. 40. 21. 18. 20.  5. 18. 41. 29.]
 [ 4.  2. 38. 22. 20. 18.  5. 19. 41. 29.]
 [ 5.  2. 40. 23. 17. 20.  7. 19. 44. 32.]
 [ 5.  2. 39. 25. 23. 22.  7. 19. 42. 29.]
 [ 6.  2. 42. 20. 22. 21.  8. 20. 41. 31.]
 [ 6.  2. 40. 23. 25. 23.  8. 18. 41. 28.]
 [ 6.  2. 45. 23. 23. 23.  8. 20. 44. 27.]
 [ 7.  2. 45. 23. 24. 26.  8. 20. 44. 27.]
 [ 7.  2. 46. 23. 23. 25.  9. 20. 43. 29.]
 [ 7.  2. 43. 23. 24. 26.  9. 20. 41. 30.]
 [ 7.  2. 45. 24. 25. 27. 10. 23. 42. 30.]
 [ 7.  2. 48. 24. 23. 25.  9. 22. 40. 30.]
 [ 8.  2. 52. 24. 26. 29. 10. 23. 42. 32.]]
The best value of K is 1
              precision    recall  f1-score   support

           0       0.95      0.98      0.97       980
           1       0.96      0.99      0.98      1135
           2       0.96      0.94      0.95      1032
           3       0.91      0.94      0.93      1010
           4       0.95      0.94      0.95       982
           5       0.92      0.91      0.92       892
           6       0.97      0.97      0.97       958
           7       0.94      0.93      0.93      1028
           8       0.94      0.90      0.92       974
           9       0.91      0.92      0.92      1009

    accuracy                           0.94     10000
   macro avg       0.94      0.94      0.94     10000
weighted avg       0.94      0.94      0.94     10000

The accuracy score is 0.9434
CPU times: user 2h 49min 30s, sys: 4min 20s, total: 2h 53min 51s
Wall time: 45min 17s
```

6.3.1.2 Results of implementing standard scalar method

Comments:

1. From the results it is seen that there is no improvement in accuracy as compared to downsampling with n=2.
2. The wall time increases by 2 folds with this method of scaling.

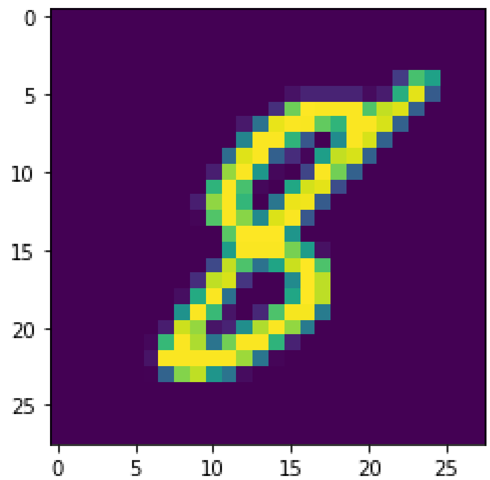
This shows that feature standardization doesnt improve the results this problem.

The original feature and the standardized feature as plotted below.

Original feature

In [34]:
```
image = np.array(X_train[202].reshape(28,28,1)).squeeze()
plt.imshow(image)
plt.show
```

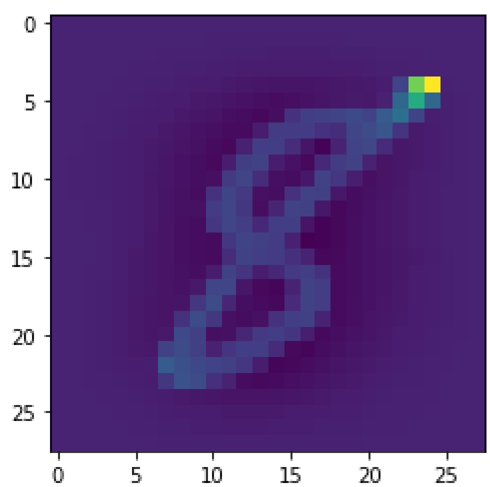
Out[34]: <function matplotlib.pyplot.show(*args, **kw)>



Standardized feature

In [35]:
```
image_std = np.array(X_train_std[202].reshape(28,28,1)).squeeze()
plt.imshow(image_std)
plt.show
```

Out[35]: <function matplotlib.pyplot.show(*args, **kw)>



6.3.2 MinMax Scalar or Unity normalization or Feature normalization

Normalization is a technique of “feature scaling” which is used to make all of the raw data values fit into a range between 0 and 1. It works better for cases in which the standardization might not work so well. Generally, each image data is a vector of pixel values (integers) which ranges between 0 and 255. Normalization divide a vector by its length and transforms it into a range between 0 and 1. If the distribution is not Gaussian or the standard deviation is very small, the min-max scaler works better. To normalize a variable to a range between 0 and 1 we need the lowest value and the highest value of the measurements on the variable and then use a simple formula to use on each measurement:

Xnorm=(X-Xmin)/(Xmax-Xmin)

Steps to implement Feature normalization

1. From sklearn.preprocessing library import MinMaxScaler module
2. Create a scaling object of MinMaxScalar module.
3. Pass the training and testing data (features) to this scaling object to scale down their magnitudes from (0 to 255) to (0-1). Thus MinMaxScalar scale down the numbrs to same scale (0-1)

6.3.2.1 Code for using MinMax scalar method with KNN leave-one-out experiment

Below mentioned code implements the MinMax scalar feature transformation method and here it is applied to the same dataset on which we have run the part 1 to 5.

In [67]:
```
%%time
scaler = MinMaxScaler()  #Creating a scaling object named scaler
X_train_nor = scaler.fit_transform(X_train)   #apply the opject in fe
atures to be scaled
X_test_nor = scaler.fit_transform(x_test)

train_nor_combin=np.column_stack((X_train_nor, y_train))
np.random.shuffle(train_nor_combin)
train_nor_data=pd.DataFrame(train_nor_combin)
train_nor_sample=train_nor_data.sample(2000)
X_train_n=np.array(train_nor_sample.drop([train_nor_sample.shape[1]-1
], axis=1))
y_train_n1=np.array(train_nor_sample[train_nor_sample.shape[1]-1])
y_train_n=y_train_n1.astype(int)



test_nor_combin=np.column_stack((X_test_nor, y_test))
np.random.shuffle(test_nor_combin)
test_nor_data=pd.DataFrame(test_nor_combin)
test_nor_sample=test_nor_data.sample(2000)
X_test_n=np.array(test_nor_sample.drop([test_nor_sample.shape[1]-1],
axis=1))
y_test_n=np.array(test_nor_sample[test_nor_sample.shape[1]-1])



err_tmp = np.zeros(10)
from sklearn.neighbors import KNeighborsClassifier

def leave_one_out(X_train,y_train):
    Err = np.empty([1, 10])
    for K in range(1, 20):
        knn = KNeighborsClassifier(n_neighbors=K)
        err_tmp = np.zeros([1,10])
        for j in range(len(X_train)):
            k_test=X_train[j]
            k_test = np.array([k_test])
            k_valid=y_train[j]
            k_train_input=np.delete(X_train, j, 0)
            k_train_outputs=np.delete(y_train, j, 0)
            knn.fit(k_train_input, k_train_outputs)
            pred_K = knn.predict(k_test)
            err_tmp[0,k_valid] = err_tmp[0,k_valid] + (pred_K != k_va
lid)
            #if(j%100==0):
                #print(j)
        Err = np.append(Err,err_tmp,axis=0)

    return(Err)


k_one_sample_s_table=pd.DataFrame(leave_one_out(X_train_n,y_train_n))
print(dp)

m=[]
```

```
for i in range(20):
    c=np.array(k_one_sample_s_table.iloc[i])
    m.append(np.mean(c))
print('The best value of K is',np.argmin(m)+1)

# KNN classification agorithm
Knn = KNeighborsClassifier(n_neighbors=1)
Knn.fit(X_train_nor, y_train)
y_pred = Knn.predict(X_test_nor)
score=Knn.score(X_test_nor,y_test)

print('The accuracy score for is',score)
print(classification_report(y_test, y_pred))
```

```
[[ 3.  2. 26. 20. 19. 17.  5. 13. 35. 23.]
 [ 3.  1. 28. 17. 10. 24.  9. 17. 50. 46.]
 [ 3.  1. 32. 22. 17. 14.  4. 17. 43. 24.]
 [ 5.  1. 31. 21. 14. 17.  2. 12. 43. 38.]
 [ 3.  2. 31. 22. 19. 15.  3. 16. 40. 27.]
 [ 4.  2. 35. 22. 15. 18.  5. 21. 40. 33.]
 [ 5.  2. 37. 23. 19. 16.  5. 19. 40. 26.]
 [ 4.  2. 40. 21. 18. 20.  5. 18. 41. 29.]
 [ 4.  2. 38. 22. 20. 18.  5. 19. 41. 29.]
 [ 5.  2. 40. 23. 17. 20.  7. 19. 44. 32.]
 [ 5.  2. 39. 25. 23. 22.  7. 19. 42. 29.]
 [ 6.  2. 42. 20. 22. 21.  8. 20. 41. 31.]
 [ 6.  2. 40. 23. 25. 23.  8. 18. 41. 28.]
 [ 6.  2. 45. 23. 23. 23.  8. 20. 44. 27.]
 [ 7.  2. 45. 23. 24. 26.  8. 20. 44. 27.]
 [ 7.  2. 46. 23. 23. 25.  9. 20. 43. 29.]
 [ 7.  2. 43. 23. 24. 26.  9. 20. 41. 30.]
 [ 7.  2. 45. 24. 25. 27. 10. 23. 42. 30.]
 [ 7.  2. 48. 24. 23. 25.  9. 22. 40. 30.]
 [ 8.  2. 52. 24. 26. 29. 10. 23. 42. 32.]]
The best value of K is 1
The accuracy score for is 0.9691
              precision    recall  f1-score   support

           0       0.98      0.99      0.99       980
           1       0.97      0.99      0.98      1135
           2       0.98      0.96      0.97      1032
           3       0.96      0.96      0.96      1010
           4       0.97      0.96      0.97       982
           5       0.95      0.96      0.96       892
           6       0.98      0.99      0.98       958
           7       0.96      0.96      0.96      1028
           8       0.98      0.94      0.96       974
           9       0.96      0.96      0.96      1009

    accuracy                           0.97     10000
   macro avg       0.97      0.97      0.97     10000
weighted avg       0.97      0.97      0.97     10000

CPU times: user 2h 26min 18s, sys: 3min 28s, total: 2h 29min 47s
Wall time: 40min 26s
```

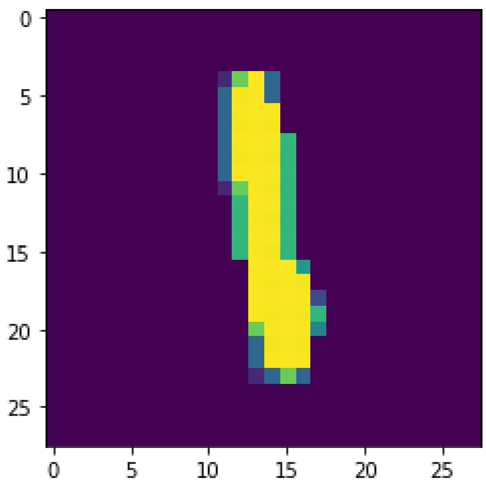
6.3.2.2 Results of implementing MinMax scalar method

1. Accuracy and wall time are better than the standardization presented in the previous section though this method performs poorer than downsampling with n=2.

Original feature

In [68]:
```
image = np.array(X_train[200].reshape(28,28,1)).squeeze()
plt.imshow(image)
plt.show
```

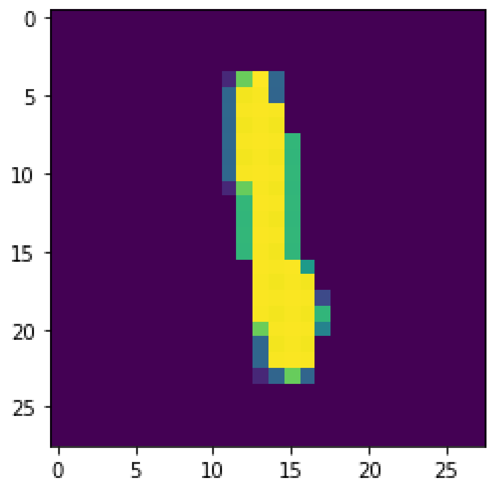
Out[68]:
```
<function matplotlib.pyplot.show(*args, **kw)>
```



Normalized feature

In [69]:
```
image_nor = np.array(X_train_nor[200].reshape(28,28,1)).squeeze()
plt.imshow(image_nor)
plt.show
```

Out[69]:
```
<function matplotlib.pyplot.show(*args, **kw)>
```



6.3.2.3 Conclusion of Standard Scalar Method

6.4 Principal Component Analysis

1. Principal Component Analysis (PCA) is a dimension-reduction tool that can be used to reduces the dimensionality of the large dataset and retains the most variation in the data set. Rather than training a classifier on very high-dimensional data, we can instead train the classifier on the lower-dimensional dataset, which will filter out the random noise in the inputs feature. A linear mapping A is obtained from PCA which transforms the feature vector x to a new low dimensional feature vector z given by [1].
$$z = A^T X$$

The following objective function is used to obtain the optimal transformation A^*.
$$A^* = \arg\max tr(A^T \Sigma A)$$
$$s.t A^T A = I$$

where,

$$\Sigma = \frac{1}{n}\Sigma_i^n (x_i - \mu)(x_i - \mu)^T$$

where n is the number of sample and μ is the sample mean vector.

In PCA analysis, the crucial part is to select the number of principal components. In this work, 392 principal components are used. Sklearn library is used to implement PCA. 2000 training and 2000 test samples are used to find the best value of K.

In [46]:

```
%%time
train=train_img.flatten()
X_train=train.reshape(60000,784)
test=test_img.flatten()
x_test=test.reshape(10000,784)
y_train=train_lab[:]

y_test=test_lab[:]



# Applying PCA function on training
# and testing set of X component
pca = PCA(int(784/2))
X_train_p = pca.fit_transform(X_train)
X_test_p = pca.transform(x_test)


# sampling the training data

train_pca_combin=np.column_stack((X_train_p, y_train))
np.random.shuffle(train_pca_combin)
train_pca_data=pd.DataFrame(train_pca_combin)
train_pca_sample=train_pca_data.sample(2000)
X_train_pca=np.array(train_pca_sample.drop([train_pca_sample.shape[1]
-1], axis=1))
y_train_pca1=np.array(train_pca_sample[train_pca_sample.shape[1]-1])
y_train_pca = y_train_pca1.astype(int)


# sampling the testing data

test_pca_combin=np.column_stack((X_test_p, y_test))
np.random.shuffle(test_pca_combin)
test_pca_data=pd.DataFrame(test_pca_combin)
test_pca_sample=test_pca_data.sample(2000)
X_test_pca=np.array(test_pca_sample.drop([test_pca_sample.shape[1]-1
], axis=1))
y_test_pca=np.array(test_pca_sample[test_pca_sample.shape[1]-1])


#K leave 1 out



err_tmp = np.zeros(10)

def leave_one_out(X_train,y_train):
    Err = np.empty([1, 10])
    for K in range(1, 20):
        knn = KNeighborsClassifier(n_neighbors=K)
        err_tmp = np.zeros([1,10])
        for j in range(len(X_train)):
            k_test=X_train[j]
            k_test = np.array([k_test])
            k_valid=y_train[j]
```

```
            k_train_input=np.delete(X_train, j, 0)
            k_train_outputs=np.delete(y_train, j, 0)
            knn.fit(k_train_input, k_train_outputs)
            pred_K = knn.predict(k_test)
            err_tmp[0,k_valid] = err_tmp[0,k_valid] + (pred_K != k_va
lid)
            #if(j%100==0):
                #print(j)
        Err = np.append(Err,err_tmp,axis=0)

    return(Err)


k_one_sample_pca_table=pd.DataFrame(leave_one_out(X_train_pca,y_train
_pca))
print(k_one_sample_pca_table)

m=[]
for i in range(20):
    c=np.array(k_one_sample_pca_table.iloc[i])
    m.append(np.mean(c))
print('The best value of K is',np.argmin(m)+1)



# KNN classification agorithm
Knn = KNeighborsClassifier(n_neighbors=(np.argmin(m)+1))
Knn.fit(X_train_p,y_train)
y_pred = Knn.predict(X_test_p)


score=Knn.score(X_test_p,y_test)

print(classification_report(y_test, y_pred))
print('The accuracy score for is',score)
```

```
       0    1     2     3     4     5     6     7     8     9
0    0.0  0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0
1    7.0  1.0  24.0  23.0  21.0  30.0  10.0  10.0  33.0  14.0
2    3.0  1.0  22.0  15.0  12.0  34.0  18.0  15.0  48.0  30.0
3    4.0  2.0  25.0  16.0  19.0  29.0   9.0  11.0  35.0  16.0
4    2.0  2.0  25.0  15.0  16.0  29.0  12.0  14.0  40.0  20.0
5    7.0  2.0  23.0  17.0  19.0  28.0  12.0  16.0  40.0  22.0
6    6.0  2.0  25.0  19.0  19.0  30.0  12.0  16.0  40.0  21.0
7    8.0  2.0  29.0  20.0  23.0  30.0  10.0  14.0  37.0  18.0
8    6.0  2.0  30.0  19.0  22.0  31.0  10.0  17.0  40.0  19.0
9    7.0  2.0  29.0  20.0  23.0  31.0   8.0  16.0  41.0  19.0
10   7.0  2.0  34.0  19.0  21.0  35.0  10.0  17.0  42.0  21.0
11   7.0  2.0  35.0  22.0  25.0  35.0  10.0  15.0  43.0  20.0
12   6.0  2.0  40.0  19.0  23.0  37.0  10.0  17.0  44.0  18.0
13   8.0  2.0  38.0  20.0  27.0  35.0  11.0  16.0  46.0  19.0
14   7.0  2.0  42.0  20.0  24.0  37.0  11.0  18.0  45.0  20.0
15   7.0  2.0  42.0  20.0  30.0  38.0  11.0  15.0  46.0  19.0
16   7.0  2.0  43.0  24.0  27.0  40.0  11.0  15.0  46.0  17.0
17   9.0  2.0  42.0  22.0  27.0  41.0  11.0  16.0  45.0  15.0
18   9.0  1.0  44.0  22.0  26.0  44.0  11.0  15.0  47.0  15.0
19  11.0  2.0  43.0  23.0  26.0  46.0  11.0  16.0  47.0  17.0
The best value of K is 1
              precision    recall  f1-score   support

           0       0.98      0.99      0.99       980
           1       0.97      0.99      0.98      1135
           2       0.98      0.96      0.97      1032
           3       0.96      0.96      0.96      1010
           4       0.97      0.96      0.97       982
           5       0.95      0.97      0.96       892
           6       0.98      0.99      0.98       958
           7       0.96      0.96      0.96      1028
           8       0.98      0.95      0.96       974
           9       0.96      0.96      0.96      1009

    accuracy                           0.97     10000
   macro avg       0.97      0.97      0.97     10000
weighted avg       0.97      0.97      0.97     10000

The accuracy score for is 0.9691
CPU times: user 57min 25s, sys: 1min 24s, total: 58min 50s
Wall time: 14min 59s
```

Conclusion of PCA

1. It is observed that PCA reduces wall time when compared to downsampling, standardization and normalization
2. PCA also improves accuracy as compared to the other three methods.

It is thus concluded that PCA is a better feature transformation for this problem.

Reference

[1] Watanabe, Kenji, Takumi Kobayashi, and Toshikazu Wada. "Semi-Supervised Feature Transformation for Tissue Image Classification." PloS one 11.12 (2016).