Assignment 1 – k Nearest Neighbor

Name: Helena Julie Arpudaraj

UCF ID: 4735186

In this assignment I have programmed kNN method and applied to the MNIST data set of handwritten digits. The MNIST Training data set has 60,000 images and the Test data set has 10,000 images.

The kNN algorithm is as described below:

- Find the Euclidean distance for each test image with all 60,000 train images.
- Then find the K shortest distance images for each test image. K being (1,2,3… or 10)
- Match the labels for the shortest distance images from the Training images labels.
- Among the K matched labels, the label which gets the maximum vote(i.e., the one which is predicted the most times) is the predicted label for the test image.
- The test image label is then checked with the predicted label from train data. If it's the same then the prediction is right.
  The above algorithm is continued for each of the 10,000 test images.

The k={1,2,…10} nearest neighbors are determined using Euclidean distance formula as shown below:

$$D(a,b) = \sqrt{\sum_j (a_j - b_j)^2}$$

The accuracy for knn algorithm is calculated as below:

Accuracy % = (Number of Right Predictions/ Total Test Images)*100

I have downloaded the MNIST data set from http://yann.lecun.com/exdb/mnist/. (Links to an external site.)

I have programmed the whole assignment in Python and have not used any machine learning libraries.

Contents

kNN(k Nearest Neighbor) for k=1

Pseudo code:

Note: I have attached python code seperately

The kNN algorithm for k=1:

The below algorithm is continued for each of the 10,000 test images.

1. Find the Euclidean distance for each test image with all 60,000 train images.
2. Then find the shortest Euclidean distance images for each test image.
3. Match the label for the shortest distance images from the Training images labels set. This will be the predicted label for the test image.
4. Check the predicted label from train data with the test data label. If it’s the same then the prediction is right.

Accuracy % = (Number of Right Predictions/ Total Test Images)*100

Output:

1. Accuracy
   96.16%

2. Confidence Interval

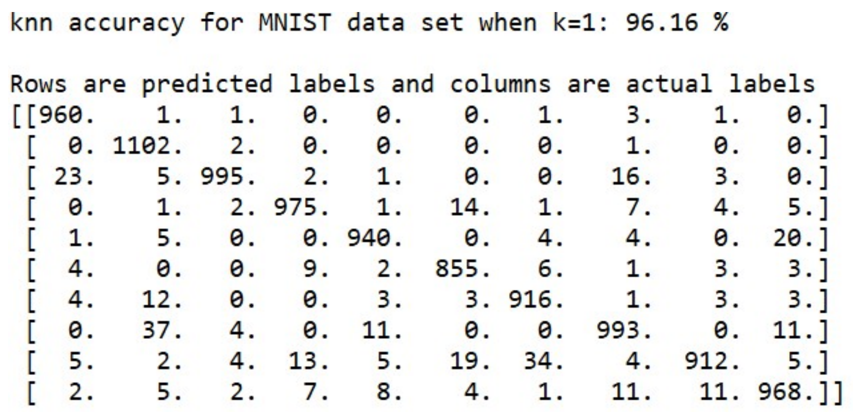
   Formula for standard deviation: $\hat{\sigma} = \sqrt{\frac{\hat{p}(1-\hat{p})}{n}}$
   n=10000
   p = 0.9616
   1-p= 0.0384

   Standard deviation = 0.0019

   Confidence Interval Lower bound = 0.9616-(1.96*0.0019) =0.957

   Confidence Interval Upper bound = 0.9616+(1.96*0.0019) =0.965

3. Confusion Matrix



```
knn accuracy for MNIST data set when k=1: 96.16 %

Rows are predicted labels and columns are actual labels
[[960.    1.    1.    0.    0.    0.    1.    3.    1.    0.]
 [  0. 1102.   2.    0.    0.    0.    0.    1.    0.    0.]
 [ 23.    5. 995.    2.    1.    0.    0.   16.    3.    0.]
 [  0.    1.    2. 975.    1.   14.    1.    7.    4.    5.]
 [  1.    5.    0.    0. 940.    0.    4.    4.    0.   20.]
 [  4.    0.    0.    9.    2.  855.    6.    1.    3.    3.]
 [  4.   12.    0.    0.    3.    3. 916.    1.    3.    3.]
 [  0.   37.    4.    0.   11.    0.    0.  993.    0.   11.]
 [  5.    2.    4.   13.    5.   19.   34.    4.  912.    5.]
 [  2.    5.    2.    7.    8.    4.    1.   11.   11.  968.]]
```

10 Fold Cross Validation

Note: I have attached python code seperately

Procedure and Pseudo code:

1. Shuffle the MNIST Training Dataset to ensure a uniform distribution of data
2. Divide the Training Dataset into 10 equal parts or 10 folds. Each fold will have (60,000/10) = 6000 images.
3. In this 10 folds, make 1 fold as test data set and the rest (60,000-6000)=54,000 images as the training data set
4. The 6,000 test images are classified using the knn-algorithm
5. The above knn algorithm procedure is continued to classify each of the 10 folds images. Each time a unique fold is taken as the test set and the remaining folds are taken as the training set.
6. The above 5 steps are carried for k={1,2,3,4,5,6,7,8,9,10} in kNN (the knn algorithm for k nearest neighbors is explained below)
7. Then find the optimal k(i.e., the k value in kNN which gives the highest accuracy)
8. Use this optimal k to classify the 10,000 test images in MNIST data set.

Algorithm for knn:

- Find the Euclidean distance for each test image with all 54,000 train images.
- Finds the K shortest distance images for each test image. K being (1,2,3… or 10)
- Match the labels for the shortest distance images from the Training images labels.
- Among the K matched labels, the label which gets the maximum vote(i.e., the one which is predicted the most times) is the predicted label for the test image.
- Check the test image label with the predicted label from train data. If it's the same then the prediction is right.
- The above algorithm is continued for each of the 6,000 test images.

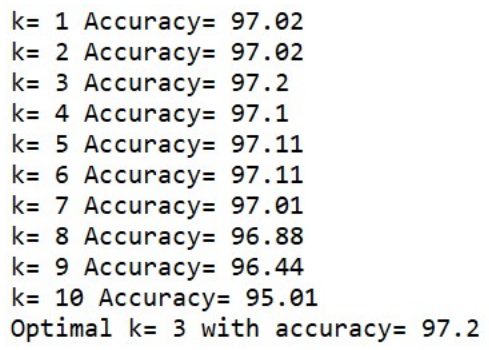
The k={1,2,…10} nearest neighbors are determined using Euclidean distance formula as shown below:

$$D(a,b)=\sqrt{\sum_j (a_j-b_j)^2}$$

The accuracy for knn algorithm is calculated as below:

Accuracy % = (Number of Right Predictions/ Total Test Images)*100

Output:



```
k= 1 Accuracy= 97.02
k= 2 Accuracy= 97.02
k= 3 Accuracy= 97.2
k= 4 Accuracy= 97.1
k= 5 Accuracy= 97.11
k= 6 Accuracy= 97.11
k= 7 Accuracy= 97.01
k= 8 Accuracy= 96.88
k= 9 Accuracy= 96.44
k= 10 Accuracy= 95.01
Optimal k= 3 with accuracy= 97.2
```

Applying Optimal k to classify MNIST Test Data Set

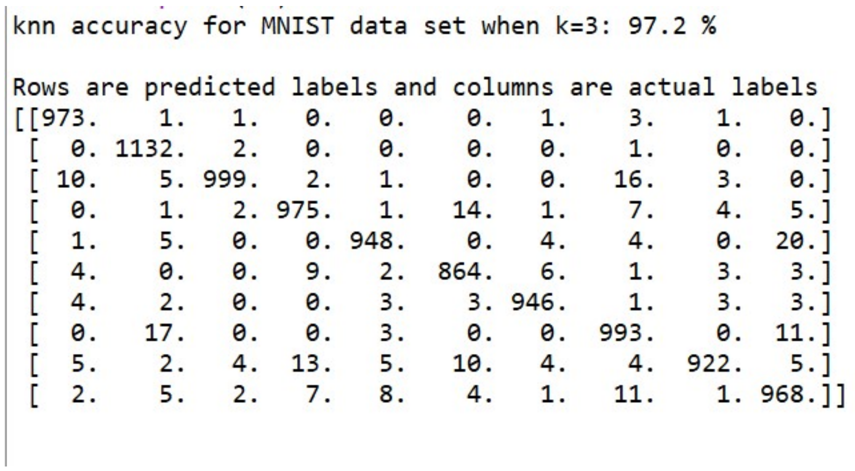
Note: I have attached python code seperately

The optimal k obtained from previous 10 fold cross validation procedure is 3. Applying kNN to classify the MNIST test images where k=3.

Output:

Accuracy = 97.2

Confusion Matrix for Optimal k=3



```
knn accuracy for MNIST data set when k=3: 97.2 %

Rows are predicted labels and columns are actual labels
[[973.    1.    1.    0.    0.    0.    1.    3.    1.    0.]
 [  0. 1132.   2.    0.    0.    0.    0.    1.    0.    0.]
 [ 10.    5.  999.    2.    1.    0.    0.   16.    3.    0.]
 [  0.    1.    2.  975.    1.   14.    1.    7.    4.    5.]
 [  1.    5.    0.    0.  948.    0.    4.    4.    0.   20.]
 [  4.    0.    0.    9.    2.  864.    6.    1.    3.    3.]
 [  4.    2.    0.    0.    3.    3.  946.    1.    3.    3.]
 [  0.   17.    0.    0.    3.    0.    0.  993.    0.   11.]
 [  5.    2.    4.   13.    5.   10.    4.    4.  922.    5.]
 [  2.    5.    2.    7.    8.    4.    1.   11.    1.  968.]]
```

Classification accuracy with Confidence Interval

Formula for standard deviation: $\hat{\sigma} = \sqrt{\frac{\hat{p}(1-\hat{p})}{n}}$
n=10000
p = 0.972
1-p= 0.028

Standard deviation = 0.0016

Confidence Interval Lower bound = 0.972-(1.96*0.0016) =0.968

Confidence Interval Upper bound = 0.972+(1.96*0.0016) =0.975

Sliding Window

Code:

Note: I have attached python code seperately

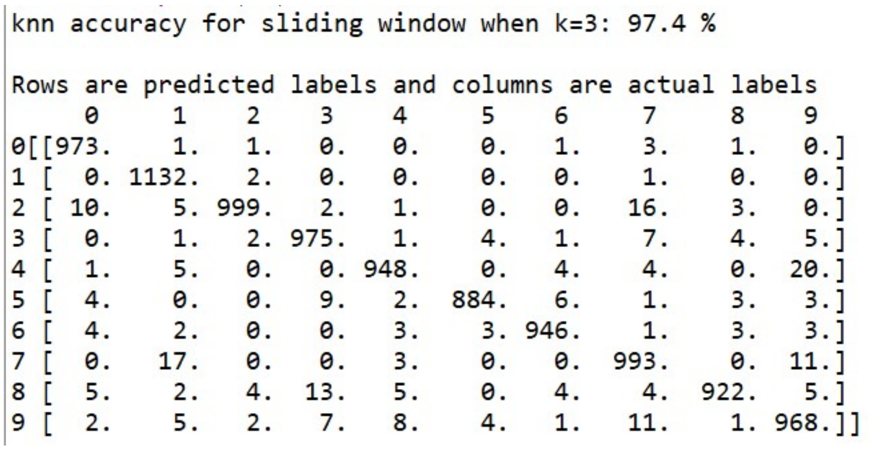
The pseudo code for algorithm is as follows:

1. For each training image add a row to the top and bottom of the image and a column to the left side and right side of image so that we get a 30X30 image.
2. To get 9 cropped image – first slide it on the top most row towards right 1 column at time to get 3 28X28 images, second slide down by one column and again slide towards right 1 column at a time get 3 28X28 images, again slide down by one column and again slide towards right 1 column at a time get 3 28X28 images. Now we have 9 cropped images for each training image.
3. Apply the kNN algorithm (k=3 Optimal k) to classify the MNIST Test data set by finding shortest Euclidean distance with the 9 cropped images of each training data.

   Output:-

Accuracy = 97.4

Confusion Matrix for Optimal k=3



```
knn accuracy for sliding window when k=3: 97.4 %

Rows are predicted labels and columns are actual labels
     0     1    2    3    4     5    6     7     8    9
0[[973.    1.   1.   0.   0.    0.   1.    3.    1.   0.]
1 [  0. 1132.   2.   0.   0.    0.   0.    1.    0.   0.]
2 [ 10.    5. 999.   2.   1.    0.   0.   16.    3.   0.]
3 [  0.    1.   2. 975.   1.    4.   1.    7.    4.   5.]
4 [  1.    5.   0.   0. 948.    0.   4.    4.    0.  20.]
5 [  4.    0.   0.   9.   2.  884.   6.    1.    3.   3.]
6 [  4.    2.   0.   0.   3.    3. 946.    1.    3.   3.]
7 [  0.   17.   0.   0.   3.    0.   0.  993.    0.  11.]
8 [  5.    2.   4.  13.   5.    0.   4.    4.  922.   5.]
9 [  2.    5.   2.   7.   8.    4.   1.   11.    1. 968.]]
```

Classification accuracy with Confidence Interval

1.

Formula for standard deviation: $\hat{\sigma} = \sqrt{\frac{\hat{p}(1-\hat{p})}{n}}$

n=10000

p = 0.974

1-p= 0.026

Standard deviation = 0.00159

Confidence Interval Lower bound = 0.972-(1.96*0.0016) =0.971

Confidence Interval Upper bound = 0.972+(1.96*0.0016) =0.977

Significance Testing and Difference Rule

The k-fold algorithm gave as accuracy=97.2% - p_0

The sliding window gave us accuracy=97.4% - p_1

Null Hypothesis H_0 : $p_0 = p_1$

Alternate Hypothesis H_a : $p_0 = p_1$ (This is true)

The k-fold algorithm confidence interval:

Confidence Interval Lower bound = 0.972-(1.96*0.0016) =0.968

Confidence Interval Upper bound = 0.972+(1.96*0.0016) =0.975

The sliding window algorithm confidence interval:

Confidence Interval Lower bound = 0.972-(1.96*0.0016) =0.971

Confidence Interval Upper bound = 0.972+(1.96*0.0016) =0.977

The confidence interval difference between to classifiers is 0.02 with accuracy difference of 0.2%.

Hence I have concluded that the sliding window classifier is more efficient.