Mithun Mohan Raj

Machine Learning assignment - 1

K Nearest Neighbors – Classification algorithm

Algorithm:

A case is classified by a majority vote of its neighbors, with the case being assigned to the class most common amongst its K nearest neighbors measured by a distance function. If K = 1, then the case is simply assigned to the class of its nearest neighbor. If k>1, then the case is assigned to the class of majority among its nearest neighbors**.(1)**

Pseudocode of k-nearest neighbor algorithm(k=1):

A. Load the MNIST datasets by creating a function that reads idx files.

    def read_idx(filename)

B. Change the datatype of the data as float64 in-order to make the calculations efficient.

C. Assign the k value as 1.

D. Taking one Testing image at a time, calculate the one nearest neighbor for the test image.

```
for x in range(int(len(test_data))):
 neighbors = getNeighbors(train_data,test_data[x], k,train_label)
```

E. Calculate the Euclidean distance between the training images and testing image and store the distances for the particular test image at a time.

```
length = len(testInstance)-1
for x in range(int(len(trainingSet))):
dist = euclideanDistance(testInstance, trainingSet[x],length)
```

F. Determine the training image that has the least Euclidean distance with the test image by sorting the distances in the ascending order.

```
distances.sort(key=operator.itemgetter(2))
```

G. Using the index of determined training image, find the training label corresponding to the training image.

H. Choose the determined training image as nearest neighbor for the given test image.

```
for x in range(k):
neighbors.append(distances[x][1])
```

I. The nearest neighbor is the predicted label for the test image and compare it with the corresponding test label.

```
Predicted_label = getNeighbors(train_data,test_data[x], k,train_label)
    if label_pred==test_label[x]:
        correct_v +=1
```

J. Increase the count by 1 if the predicted label is same as the test label.

K. Repeat the steps from D to J for all the test images taking one at a time.

L. Calculate the accuracy of the algorithm by taking the ratio between total number of correctly predicted labels(count) to the total number of test labels.

```
accuracy=correct_v/(int(len(test_label)))
```

DATASETS

Training Set-:

- The dataset contains 60,000 training image examples of the handwritten digits from 0 to 9.
- The size of each image is given as 28 * 28 pixels per image.

Test Set-:

- The dataset contains 10,000 training image examples of the handwritten digits from 0 to 9.
- The size of each image is given as 28 * 28 pixels per image.

Training Label-:

The dataset contains 60,000 training labels of the handwritten digits from 0 to 9.

Test Label-:

The dataset contains 10,000 training labels of the handwritten digits from 0 to 9.

Parameters

Accuracy:

"Accuracy" is the ratio of total number of correctly predicted labels to the total number of labels.

Formula used:

Accuracy in % = (Total number of correctly predicted labels/total number of labels) x 100

Euclidean distance:

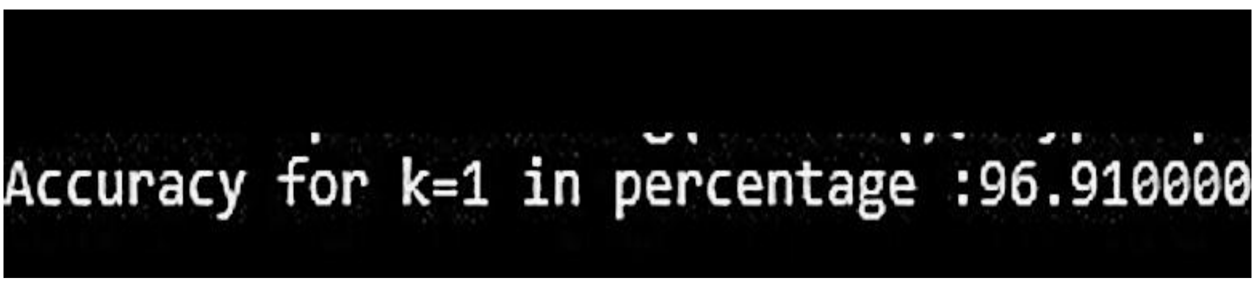
"Euclidean distance" is the straight-line distance between two corresponding pixels of training and testing image respectively and is evaluated using the euclidean norm.

Formula used:

Euclidean distance (train-image, test-image) = √ Σ ((test-image-pixel)– (train-image-pixel))^2

RESULTS:

The following is image of the output terminal window for k-nearest neighbor algorithm for k = 1.



The above value for the "Accuracy" shows that k-nearest neighbor Algorithm for (k=1) predicts 96.91%(approximately 97%) of the test images accurately.

K-fold Cross-validation

Algorithm:

This approach involves randomly dividing the set of observations into k groups, or folds, of approximately equal size. The first fold is treated as a validation set, and the method is fit on the remaining k − 1 folds.**(2)**

Procedure of 10-Fold cross-validation:

1) Load the MNIST datasets by creating a function that reads idx files.

   def read_idx(filename):

2) Change the datatype of the data as float64 in-order to make the calculations efficient.
3) Shuffle the training datasets randomly.

   a=[i for i in range(0,60000)]

   shuffle(a)

   train_s=np.array(train_data[a])

   label_s=np.array(train_label[a])

4) Split the datasets into K=10 groups and perform the following steps.

   for i in range(10):

   A. For each group, assign one group as the test set and other groups as the training set.

      test_kfold =train_s[(i*6000):(6000 * (i + 1)), :]

      for j in range(10):

      if(j!=i):

      train=train_s[(j * 6000):(6000*(j+1)),:]

   B. Assign the k value as 1 to 10 and repeat the following steps for each k.

      k=[1,2,3,4,5,6,7,8,9,10]

      for i in k:

   C. Taking one Testing image at a time, calculate the Euclidean distance between the training images and test image.

      for x in range(int(len(test_data))):

```
neighbors = getNeighbors(train_data,test_data[x], k,train_label)

dist = euclideanDistance(testInstance, trainingSet[x], length)
```

D. Sort the training images corresponding to their Euclidean distances with the test image.
E. Determine the first k-training images that has the least Euclidean distances with the test image and arrange them in the ascending order of distances.
F. Using the indexes of determined training images, find the k-nearest training labels corresponding to the training image.
G. Choose the determined training images as k-nearest neighbors for the given test image.

```
for x in range(k):

    neighbors.append(distances[x][1])
```

H. Find one majority class of label among the k-nearest training labels

```
def find_majority(neighbors):
    value =neighbors[m,:]
    if value in tuple(countneighbours):
        countneighbours[tuple(value)] += 1
else:
        countneighbours[tuple(value)] = 1
        print(countneighbours)
```

I. The majority class of label is the predicted label for the test image and compare it with the corresponding test label.

```
label_pred=find_majority(neighbors)
    if label_pred==test_label[x]:
        correct_v +=1
```

J. Increase the count by 1 if the predicted label is same as the test label.
K. Repeat the steps from c to J for all the test images taking one at a time.
L. Calculate the accuracy of the algorithm for each “k” by taking the ratio between the total number of correctly predicted labels(count) to the total number of test labels.

accuracy=correct_v/(int(len(test_label)))

M. Print the calculated accuracy for each k (from 1 to 10)

5, Repeat the step 4 for each of the 10-fold groups and print the K x k (10*10)

Accuracies.

6, calculate the averages of accuracies for each k and find the maximum of averages.

Average(for each k)= (sum of accuracies for each fold)/10

7, The k value having the maximum average value of accuracies is the Optimal k value of the Algorithm.

Optimal k = k with maximum average value

DATASETS

Training Image dataset and training label dataset are shuffled randomly and used as datasets for each of the k-folds.

Datasets for each fold:

Test dataset:

1, one group of training image dataset is the test dataset.

2, the dataset contains 6000 training image examples of handwritten digits of 0 to 9.

3, Size of the each image is given as 28*28 pixels per image.

Training dataset:

1, The remaining group of training image dataset is training set.

2, The dataset contains 54,000 training image examples of handwritten digits of 0 to 9.

3, Size of the each image is given as 28*28 pixels per image.

Test Label:

1, one group of training label dataset as a test labels.

2, The dataset contains 6000 training label examples of handwritten digits of 0 to 9

Training Label:

1, The remaining group of training label dataset as training labels.

2, The dataset contains 54,000 training label examples of handwritten digits of 0 to 9.

Parameters:

Optimal k-value:

It is the value of k for which average of accuracies for all 10-fold groups is maximum.

Formula used:

Optimum k-value = Maximum(k-Averages)

For each k, Average = (sum of accuracies/10)

Result:

The following image represents a table of accuracies for each of the 10-fold groups against k values.

k\K-FOLDS	K=1	K=2	K=3	K=4	K=5	K=6	K=7	K=8	K=9	K=10	sum	average	
k=1	97.4167	97.2	97.23333	97.13333	97.2	97.35	97.41667	97.43333	97.16667	97.38333	972.9333	97.29333	
k=2	97.4167	97.2	97.23333	97.13333	97.2	97.35	97.41667	97.43333	97.16667	97.38333	972.9333	97.29333	
k=3	97.3333	97.31667	97.48333	97.4	97.25	97.45	97.41667	97.51667	96.95	97.45	973.5667	97.35667	
k=4	97.4333	97.3	97.4	97.66667	97.38333	97.83333	97.53333	97.56667	97.21667	97.5	974.8333	97.48333	Optimal k
k=5	97.25	97.11667	97.38333	97.36667	97.15	97.58333	97.46667	97.45	96.88333	97.43333	973.0833	97.30833	
k=6	97.2167	97.31667	97.36667	97.41667	97.23333	97.71667	97.43333	97.41667	96.83333	97.48333	973.4333	97.34333	
k=7	97.1167	97.11667	97.15	97.31667	96.98333	97.48333	97.13333	97.26667	96.86667	97.3	971.7333	97.17333	
k=8	97.0167	97.11667	97.3	97.4	97	97.5	97.16667	97.16667	96.83333	97.28333	971.7833	97.17833	
k=9	97	96.91667	97.18333	97.28333	96.78333	97.4	96.8	96.98333	96.68333	97.15	970.1833	97.01833	
k=10	97	96.98333	97.15	97.35	96.81667	97.3	97	97.01667	96.7	97.1	970.4167	97.04167	

From the above table, we can infer that "k=4" has the maximum average value of accuracies as 97.4833

Hence the Optimal k-value for the algorithm is "4"

K-nearest neighbor for optimal-k

Pseudocode:

A. Load the MNIST datasets by creating a function that reads idx files.

   def read_idx(filename):

B. Change the datatype of the data as float64 in-order to make the calculations efficient.

C. Assign the k value as 4(optimal).

   K = 4

D. Taking one Testing image at a time, calculate the Euclidean distance between the training images and test image.

   for x in range(int(len(test_data))):

   neighbors = getNeighbors(train_data,test_data[x], k,train_label)

```
dist = euclideanDistance(testInstance, trainingSet[x], length)
```

E. Sort the training images corresponding to their Euclidean distances with the test image.
F. Determine the first k-training images that has the least Euclidean distances with the test image and arrange them in the ascending order of distances.

G. Using the indexes of determined training images, find the k-nearest training labels corresponding to the training image.

H. Choose the determined training images as k-nearest neighbor for the given test image.

```
for x in range(k):
    neighbors.append(distances[x][1])
```

I. Find one majority class of label among the k-nearest training labels.

```
def find_majority(neighbors):
    value =neighbors[m,:]
    if value in tuple(countneighbours):
        countneighbours[tuple(value)] += 1
else:
        countneighbours[tuple(value)] = 1
        print(countneighbours)
```

J. The majority class of label is the predicted label for the test image and compare it with the corresponding test label.

```
label_pred=find_majority(neighbors)
    if label_pred==test_label[x]:
```

correct_v +=1

K. Increase the count by 1 if the predicted label is same as the test label.
L. Repeat the steps from D to J for all the test images taking one at a time.
M. Calculate the accuracy of the algorithm by taking the ratio between total number of correctly predicted labels(count) to the total number of test labels.

   accuracy=correct_v/(int(len(test_data)))

N. Calculate the confusion matrix by comparing the classes in both predicted and actual Labels and by keeping the count on every set of class pairs.

   x1=pd.Series(y_actu, name='Actual')

   y1=pd.Series(y_pred, name='Predicted')

   df_confusion = pd.crosstab(x1,y1,margins=True)

   print(df_confusion)

Data-sets:

Data-sets section is same as that of k-nearest neighbor (k=1) section.

Parameters:

Confidence Interval:

A confidence interval is a bound on the estimate of a population variable. It is an interval statistic used to quantify the uncertainty on an estimate**.(3)**

Formula used:

95% Confidence Interval = $p \pm 1.96 \times \sigma$

P = success rate = accuracy/100

σ = $\sqrt{(p(1-p))/n}$, where n = number of samples.

Result:

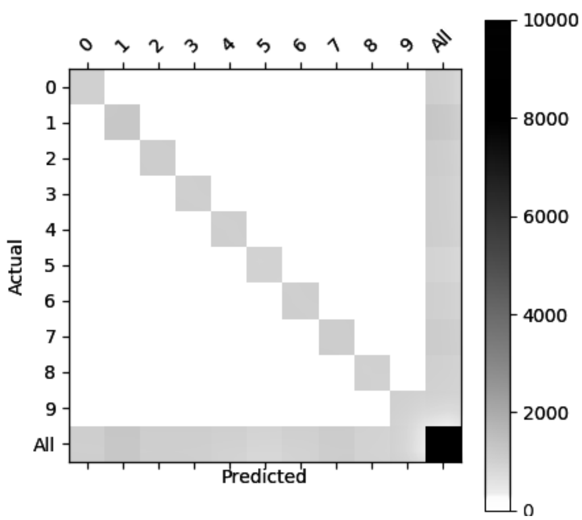
The accuracy for the algorithm with optimal k-value is “97.16%”

95% confidence interval = 0.9716 ± 1.96 x (0.16611)

= [0.96837,0.97482]

The following image represents the confusion matrix for the algorithm with the optimal k-value = 4

Predicted / Actual	0	1	2	3	4	5	6	7	8	9	All
0	973	1	1	0	0	1	3	1	0	0	980
1	0	1132	2	0	0	0	1	0	0	0	1135
2	10	5	995	2	1	0	0	16	3	0	1032
3	0	1	2	975	1	14	1	7	4	5	1010
4	1	5	0	0	948	0	4	4	0	20	982
5	4	0	0	9	2	864	6	1	3	3	892
6	4	2	0	0	3	3	946	0	0	0	958
7	0	17	4	0	3	0	0	993	0	11	1028
8	5	2	4	13	5	10	4	4	922	5	974
9	2	5	2	7	8	4	1	11	1	968	1009
All	999	1170	1010	1006	971	896	966	1037	933	1012	10000



Sliding window

Pseudocode:

A. Load the MNIST datasets by creating a function that reads idx files.

B. Change the datatype of the data as float64 in-order to make the calculations efficient.

C. Assign the k value as 4(optimal).

K = 4

for x in range(int(len(raw_test))):

D. Take one training image of size 28*28 at a time and pad the image with zeros so that the padded image size is 30 * 30.

for x in range(int(len(trainingset))):

sample = np.array(trainingset[x, :])

sample = np.pad(sample, pad_width=1, mode='constant', constant_values=1)

E. From the padded image, extract 9 images each of size 28*28 for every training image.

s1 = np.array(sample[0:28, 0:28]) s2 = np.array(sample[0:28, 1:29])

s3 = np.array(sample[0:28, 2:30]) s4 = np.array(sample[1:29, 0:28])

s5 = np.array(sample[1:29, 1:29]) s6 = np.array(sample[1:29, 2:30])

s7 = np.array(sample[2:30, 0:28]) s8 = np.array(sample[2:30, 1:29])

s9 = np.array(sample[2:30, 2:30])

F. Take one test image at a time and calculate the Euclidean distances between the 9 extracted image and the test image.

image = np.array([s1, s2, s3, s4, s5, s6, s7, s8, s9])

for i in range(9):

dist = euclideanDistance(testinstance, test,length)

G. Minimum of the 9 calculated distances is the Euclidean distance for that training image with the given test image.

```
min_dist = np.amin(distances)
```

H. Now repeat the steps from D to G to calculate Euclidean distances for all the training images to the given test image.

I. Sort the training images corresponding to their Euclidean distances with the test image.

```
distances_main.sort(key=operator.itemgetter(2))
```

J. Determine the first k-training images that has the least Euclidean distances with the test image and arrange them in the ascending order of distances.

K. Using the indexes of determined training images, find the k-nearest training labels corresponding to the training image.

L. Choose the determined training images as k-nearest neighbors for the given test image.

```
for x in range(k):
        neighbors.append(distances_main[x][1])
```

M. Find one majority class of label among the k-nearest training labels

```
def find_majority(neighbors):
        value =neighbors[m,:]
        if value in tuple(countneighbours):
            countneighbours[tuple(value)] += 1
    else:
            countneighbours[tuple(value)] = 1
            print(countneighbours)
```

N. The majority class of label is the predicted label for the test image and compare it with the corresponding test label.

O. Increase the count by 1 if the predicted label is same as the test label.

```
if label_pred==test_label[x]:
        correct_v +=1
```

P. Repeat the steps from D to O for all the test images taking one at a time.

Q. Calculate the accuracy of the algorithm by taking the ratio between total number of correctly predicted labels(count) to the total number of test labels.

```
accuracy=correct_v/(int(len(raw_test)))
```

O. Calculate the confusion matrix by comparing the classes in both predicted and actual Labels and and by keeping the count on every set of class pairs.

```
x1=pd.Series(y_actu, name='Actual')

y1=pd.Series(y_pred, name='Predicted')

df_confusion = pd.crosstab(x1,y1,margins=True)
print(df_confusion)
```

Datasets:

Datasets used are same as that of used in optimal k-nn standard section.

Parameters:

Confidence Interval:

A confidence interval is a bound on the estimate of a population variable. It is an interval statistic used to quantify the uncertainty on an estimate.

Formula used:

95% Confidence Interval = p ± 1.96 x σ

P = success rate = accuracy/100

σ = $\sqrt{(p(1-p))/n}$

n = number of samples.

Result:

The accuracy for the sliding window algorithm with optimal k-value is “97.94%”

95% Confidence Interval = 0.9794 $\pm$ 1.96 x 0.00142= [0.97661,0.98218]

Accuracy for optimal k in sliding window is : 97.94

Pred	0	1	2	3	4	5	6	7	8	9	All
Actu											
0	973	1	1	0	0	1	3	1	0	0	980
1	0	1132	2	0	0	0	1	0	0	0	1135
2	10	5	1000	2	1	0	0	11	3	0	1032
3	0	1	2	988	1	0	1	8	4	5	1010
4	1	5	0	0	958	0	4	4	0	10	982
5	4	0	0	9	2	864	6	1	3	3	892
6	4	2	0	0	3	3	946	0	0	0	958
7	0	4	4	0	3	0	0	1006	0	11	1028
8	5	2	4	4	5	0	4	4	941	5	974
9	2	5	2	7	0	4	1	0	1	977	1009
All	999	1170	1010	1006	971	896	966	1037	933	1012	10000

The above image represents the confusion matrix for the Sliding window algorithm with the optimal k-value = 4

8, The sliding window algorithm is a better algorithm than a standard algorithm by the difference rule.

References:

(1)https://www.analyticsvidhya.com/blog/2018/03/introduction-k-   neighbours-algorithm-clustering/

(2) https://machinelearningmastery.com/k-fold-cross-validation/

(3) https://machinelearningmastery.com/confidence-intervals-for-machine-learning/