MOHAMED FARHAN

ASSIGNMENT 1: K NEAREST NEIGHBOR- REPORT

DATASET:

I programmed the kNN classifier algorithm using the MNIST dataset which comprises of:

- Training images: 60000
- Test images: 10000

Both the training and test images sets contain images of handwritten digits from 0 (zero) to 9 (nine) such that each image is of 28*28 pixels (each pixel ranges between 0 and 255).

- Training labels: 60000
- Test labels: 10000

Both the training and test labels contain the labels (from 0 to 9) that correspond to the training and test images sets respectively.

PARAMETERS:

k

'k' is an input parameter set by the user. 'k' corresponds to number of the closest neighbors that need to be determined for a given test image.

The 'k' nearest neighbors are usually determined with the help of a distance metric such as Euclidean, Manhattan distance measured between the test and training images.

In our case, we program the KNN classifier with value of 'k' set to one (k=1).

EUCLIDIAN DISTANCE

Euclidian distance is an important parameter in order to determine 'k' nearest neighbors of a test image. It is defined as the distance measured along a straight line from point (x_i,y_i) to point (x_j,y_j) in Cartesian coordinate system.

It is calculated as follows:

Euclidean distance (train-image, test-image) = $\sqrt{\Sigma}$ ((test-image-pixel)– (train-image-pixel))^2

ACCURACY

Accuracy is an important parameter that measures the correctness of the algorithm. It is calculated by computing the ratio of correctly predicted test images to the total number of test images.

Accuracy % = Number of correctly predicted test images ÷ Number of total test images

PSEUDOCODE FOR STANDARD KNN CLASSIFIER

- Define value of k.
- Define a function 'find_neighbors', which calculates the 'k' closest neighbors of a test image using Euclidian distance between the training images and the corresponding test image.

```
for x in range(length of test image):
        distance = (instance1[x] - instance2[x])^2
return  √distance
```

- The Euclidian distances are sorted in the ascending order and stored in a list 'distances'
- The list 'neighbors' contains the training labels of the 'k' closest neighbors of a corresponding test image.

```
neighbors.append(training_image[x]training_labels[x])
```

- The function 'find_neighbors' returns the training labels of the 'k' closest neighbors of the particular test image
- Define a function 'majority', which determines the training label with maximum count value.
- The training labels and its count are stored in a dictionary named 'neighbor_count'.

```
neighbor_count={training_label:count}
```

- The function 'sort_Knearest', sorts the training labels stored in 'neighbor_count' in descending order with respect to the count value of each training label.

```
def sort_Knearest():
        return sort(neighbor_count) // sort in descending order
```

- The function 'majority' returns the training label with maximum count value.

```
neighbor_count= {}
if value in (neighbor_count):
        neighbor_count[value] += 1
```

```
                else:
                        neighbor_count[value] = 1
        return maximum(neighbor_count)
```

- The predicted label is compared with the test label to verify if the test image is predicted correctly.

```
correct_v=0
If pred_label==test_label[x]:
        correct_v +=1
```

- Accuracy is computed by calculating the number of correctly predicted labels against the number of test images.

```
accuracy=correct_v/number of test images
```

NOTE: If k=1, then the function 'majority' becomes redundant as only one closest neighbor will be determined for the corresponding test image. Therefore, only the single neighbor's training label will be returned by the function at all times.

RESULTS:

For standard kNN classifier algorithm with k=1

```
C:\Users\Admin\Desktop\ML>python trial.py
trial.py:20: DeprecationWarning: The binary mode
  return np.fromstring(f.read(),dtype=np.uint8).
Accuracy for k=1 in percentage :96.910000

C:\Users\Admin\Desktop\ML>
```

Accuracy of kNN algorithm for k=1 in % = 96.91

PROCEDURE FOR IMPLEMENTING 10 FOLD CROSS VALIDATION

- Shuffle training images and training labels.

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

shuffle(a)

train_shuffle=trainData[a]

label_shuffle=trainLabel[a]
```

- Divide the training and label datasets into K=10 subsets
- The first subset is taken to be the test set and the remaining 9 subsets are concatenated to form the training set.

```
for i in range(K=0 to K=10):
        testset =train_shuffle[(i*6000):(6000 * (i + 1)), :]
        testLabel =label_shuff[(i*6000):(6000*(i + 1)), :]

  for j in range(0 to 10):
    if(j!=i):
      part_append=train_shuff[(j * 6000):(6000*(j+1)),:]
      part_labelappend=label_shuff[(j * 6000):(6000*(j+1)),:]
      trainset_cv = np.append(trainset_cv,part_append)
      trainset_cvlabel=np.append(trainset_cvlabel,part_labelappend)
```

- For different values of 'k' (ranging from 1 to 10), calculate the Euclidian distance between the test image and the training image.

```
for x in range(length of test image)
        distance = (training_image[x] – test_image)^2
return  √distance
```

- Sort the training images with respect to the Euclidian distances (ascending order).
- Store the sorted training images in a list 'neighbors' and append its corresponding training labels

```
neighbors.append(training_image[x]training_labels[x])
```

- Define a function ‘majority’, which determines the training label with maximum count value.
- The training labels and its count are stored in a dictionary named ‘neighbor_count’.

```
neighbor_count={training_label:count}
```

- The function ‘sort_Knearest’, sorts the training labels stored in ‘neighbor_count’ in descending order with respect to the count value of each training label.

```
def sort_Knearest():
    return sort(neighbor_count)
```

- The function ‘majority’ returns the training label with maximum count value.

```
neighbor_count= {}
if value in (neighbor_count):
        neighbor_count[value] += 1
    else:
        neighbor_count[value] = 1
return maximum(neighbor_count)
```

- The predicted label is compared with the test label to verify if the test image is predicted correctly.

```
correct_v=0
If pred_label==test_label[x]:
    correct_v +=1
```

- Accuracy is computed by calculating the number of correctly predicted labels against the number of test images.

```
accuracy=correct_v/number of test images
```

RESULTS:

k/K	K=1	K=2	K=3	K=4	K=5	K=6	K=7	K=8	K=9	K=10	AVERAGE
k=1	97.38	97.1666	97.4333	97.41667	97.35	97.2	97.1333	97.416666	97.2	97.23333	97.29333
k=2	97.38	97.1666	97.4333	97.41667	97.35	97.2	97.1333	97.416666	97.2	97.23333	97.29333
k=3	97.45	96.95	97.51667	97.41667	97.45	97.25	97.4	97.333333	97.3166	97.48333	97.35667
k=4	97.5	97.2166	97.56667	97.53333	97.8333	97.383	97.6666	97.4	97.4333 333	97.3	97.48333 (OPTIMAL K)
k=5	97.43	96.8833	97.45	97.46667	97.5833	97.15	97.3666	97.38333	97.1166	97.25	97.30833
k=6	97.48	96.8333	97.41667	97.43333	97.7166	97.233	97.4166	97.36667	97.3166	97.216666	97.34333
k=7	97.3	968666	97.26667	97.13333	97.4833	96.983	97.3166	97.15	97.1166	97.116666	97.17333
k=8	97.28	96.8333	97.16667	97.16667	97.5	97	97.4	97.3	97.1166	97.016666	97.17833
k=9	97.15	96.6833	96.98333	96.8	97.4	96.783	97.2833	97.18333	96.9166	97	97.01833
k=10	97.1	96.7	97.01667	97	97.3	97.816	97.35	97.15	97.15	97	97.04167

For 10 Fold Cross Validation with values of k ranging from 1 to 10

(From the above table, we find the optimal value of k to be 4)

PSEUDOCODE FOR OPTIMAL k VALUE (k=4)

- Set value of k to 4 (optimal k value)
- Define a function 'find_neighbors', which calculates the 'k' closest neighbors of a test image using Euclidian distance between the training images and the corresponding test image.

```
for x in range(length of test image):
        distance = (instance1[x] - instance2[x])^2
return  √distance
```

- The Euclidian distances are sorted in the ascending order and stored in a list 'distances'
- The list 'neighbors' contains the training labels of the 'k' closest neighbors of a corresponding test image.

```
neighbors.append(training_image[x]training_labels[x])
```

- The function 'find_neighbors' returns the training labels of the 'k' closest neighbors of the particular test image
- Define a function 'majority', which determines the training label with maximum count value.
- The training labels and its count are stored in a dictionary named 'neighbor_count'.

```
neighbor_count={training_label:count}
```

- The function 'sort_Knearest', sorts the training labels stored in 'neighbor_count' in descending order with respect to the count value of each training label.

```
def sort_Knearest():
    return sort(neighbor_count) // sort in descending order
```

- The function 'majority' returns the training label with maximum count value.

```
neighbor_count= {}
if value in (neighbor_count):
        neighbor_count[value] += 1
    else:
        neighbor_count[value] = 1
return maximum(neighbor_count)
```

- The predicted label is compared with the test label to verify if the test image is predicted correctly.

```
correct_v=0
If pred_label==test_label[x]:
        correct_v +=1
```

- Accuracy is computed by calculating the number of correctly predicted labels against the number of test images.

```
accuracy=correct_v/number of test images
```

- Create a confusion matrix using the predicted labels and true labels

```
y_pred.append(label_pred)
y_actu.append(test_label[x])
x1=pd.Series(y_actu, name='Actual')
y1=pd.Series(y_pred, name='Predicted')
df_confusion = pd.crosstab(x1,y1,margins=True)
```

CONFIDENCE INTERVAL

A confidence interval gives the estimation for a population variable. It is an interval statistic used to quantify the uncertainty on an estimate.

FORMULA:

95% Confidence Interval=p ± (1.96 * σ)

P=accuracy/100

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

RESULTS:

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

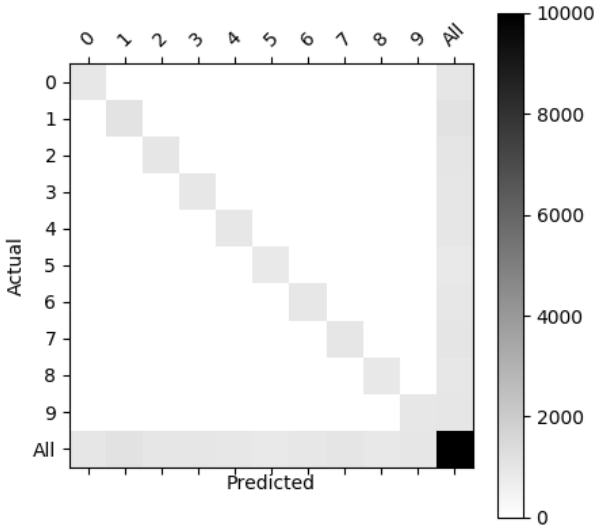
= [0.96843, 0.97476]

Accuracy and confusion matrix for kNN algorithm with the optimal k-value = 4

97.16

Predicted	0	1	2	3	4	5	6	7	8	9	All
Actual											
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

Accuracy for optimal k-value = 4 is 97.16%



Confusion matrix plot for optimal k-value=4

SLIDING WINDOW FOR OPTIMAL k-VALUE

- Set value of k to 4 (Optimal k value).
- Training images of size 28*28 is padded with zeros. The new training image size is 30*30.

  training_image_new = pad_zeros(old_training_image, pad_width=1)

- 9 images each of size 28*28 are extracted from a single training image with new size of 30*30

  i1 = training_image_new [0:28, 0:28])
  i2 = training_image_new [0:28, 1:29])
  i3 = training_image_new [0:28, 2:30])
  i4 = training_image_new [1:29, 0:28])
  i5 = training_image_new [1:29, 1:29])
  i6 = training_image_new [1:29, 2:30])
  i7 = training_image_new [2:30, 0:28])
  i8 = training_image_new [2:30, 1:29])
  i9 = training_image_new [2:30, 2:30])

- Euclidian distance is calculated for each of the extracted 9 images and the corresponding test image

```
for x in range(length of test image)
        distance = (extracted_image[x] – test_image)²
return  √distance
```

- Sort the training images with respect to the Euclidian distances (ascending order).
- Store the sorted training images in a list 'neighbors' and append its corresponding training labels

```
neighbors.append(training_image[x]training_labels[x])
```

- Define a function 'majority', which determines the training label with maximum count value.
- The training labels and its count are stored in a dictionary named 'neighbor_count'.

```
neighbor_count={training_label:count}
```

- The function 'sort_Knearest', sorts the training labels stored in 'neighbor_count' in descending order with respect to the count value of each training label.

```
def sort_Knearest():
    return sort(neighbor_count)
```

- The function 'majority' returns the training label with maximum count value.

```
neighbor_count= {}
if value in (neighbor_count):
        neighbor_count[value] += 1
    else:
        neighbor_count[value] = 1
return maximum(neighbor_count)
```

- The predicted label is compared with the test label to verify if the test image is predicted correctly.

```
correct_v=0
If pred_label==test_label[x]:
    correct_v +=1
```

- Accuracy is computed by calculating the number of correctly predicted labels against the number of test images.

accuracy=correct_v/number of test images

CONFIDENCE INTERVAL

A confidence interval gives the estimation for a population variable. It is an interval statistic used to quantify the uncertainty on an estimate.

FORMULA:

95% Confidence Interval=p ± (1.96 * σ)

P=accuracy/100

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

RESULTS:

95% confidence interval = 0.9772 ± 1.96 x (0.001492)

= [0.97423, 0.98012]

Accuracy and confusion matrix for optimal k-value (k=4) using sliding window

Accuracy for k=4 in sliding window (optimal k) 97.72

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	989	1	0	1	7	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	10	4	0	3	0	0	1000	0	11	1028
8	5	2	4	4	5	10	4	4	931	5	974
9	2	5	2	7	8	4	1	0	1	979	1009
All	999	1170	1010	1006	971	896	966	1037	933	1012	10000

Accuracy for optimal k-value (k=4) using sliding window technique is 97.72%

DIFFERENCE RULE

- **Sliding window technique is more efficient than the standard kNN classifier algorithm because of its higher accuracy as shown in above results.**
- **In particular the goal is testing the difference between two population proportions.**