Report of
Assignment 1: k Nearest Neighbor

Piyusha Jaisinghani (UCFID - 4736715)

Table of Contents

k-Nearest Neighbor classifier algorithm

k-Nearest Neighbor classifier algorithm is designed for the MNIST data set using python.

I. Structure of the python project

- Python Project name - knn_mnist

- knn_mnist python package has a python directory named knncode

- Inside knncode, a folder named dataset contains the MNIST dataset files.
  i. ./knn_mnist_09_10/knncode/dataset/t10k-images-idx3-ubyte
  ii. ./knn_mnist_09_10/knncode/dataset/t10k-labels-idx1-ubyte
  iii. ./knn_mnist_09_10/knncode/dataset/train-images-idx3-ubyte
  iv. ./knn_mnist_09_10/knncode/dataset/train-labels-idx1-ubyte

- Inside knncode, we have the following python files:
  i. **Data Reader** - This is the file used to load and read MNIST data(training images, training labels, test images, test labels) from the files present inside the dataset folder.

     parse_label() - To this method the path of the test and train label files is passed and it returns an array of the labels.

     parse_images() - To this method the path of test and train image files is passed and it converts the images into an array of floating points.
     Path : ./knn_mnist_09_10/knncode/datareader.py

  ii. **k-NN Classifier** - This is the file which has two classes Knnclassifier and MNISTPredictor and all the common code that is used to for k-cross fold validation, knn for optimal k and sliding window.

     class Knnclassifier - The object of the class initializes with two variables 1) dataset which is the training image set along with its labels and 2) k. The class has the following methods :-

     - *predict(self,point)* - This method takes one image(point) and calculates its euclidean distance with all the dataset images, sorts the distances ascending order and then takes out k closest points and their labels. It then predicts the label of the image(point) and return it.

     - *predictionslidingwindow(self,point)* - The method is used to predict labels for sliding window. It converts a 28*28 training image into 30*30 image by padding rows and columns of 0s at the top, bottom, left and right of 28*28 image. It then generates 9 sliced images(28*28) from 30*30 image by sliding over it and calculates the euclidean distance of a test image with all the 9 sliced images and

takes out the min distance out of 9. The padding and slicing and distance calculation is done for all the training image data set. It then sorts the distances ascending order and then takes out k closest points and their labels. It then predicts the label of the image(point) and return it.

class MNISTPredictor(Knnclassifier) - This class is the sub class of KnnClassifier class. It has two methods:-

- *distance(self, p1,p2)* - calls the method calEuclideanDistance to calculate distance between two images.

- *observe(self, values)* - call the method get_majority to get the majority labels out of k images.

calEuclideanDistance(point1, point2): - This method calculates the euclidean distance between two points.

```
def calEuclideanDistance(point1, point2):
        distance = np.sqrt(np.sum((point1 - point2) ** 2))
        return distance
```

get_majority(votes): - This method calculates the majority votes to predict the labels.

```
def get_majority(votes):
        counter = defaultdict(int)
        for vote in votes:
                counter[vote] += 1

        majority_count = max(counter.values())
        for key, value in counter.items():
                if value == majority_count:
                        return key
```

predict_test_set(predictor, test_set): - Method is used to predict labels for test images.

```
def predict_test_set(predictor, test_set):
    """Compute the prediction for every element of the test set."""
    predictions = [predictor.predict(test_set[i]) for i in range(len(test_set))]
    return predictions
```

evaluate_prediction(predictions, answers): - Method is used to evaluate predictions and calculate the accuracy.

```
def evaluate_prediction(predictions, answers):
        correct = sum(asarray(predictions) == asarray(answers))
        print("Total number of correct predictions :", correct)
        total = float(prod(len(answers)))
        print("Total number of labels :", total)
        return correct / total
```

confidenceinterval(classificationaccuracy,testImageSet): - Calculates and returns the confidence interval.

```
def confidenceinterval(classificationaccuracy,testImageSet ):
```

```
kaccuracy = classificationaccuracy/100
confidence_interval = []
confidence_interval_pos = kaccuracy+1.96* (sqrt((kaccuracy*(1-
kaccuracy))/len(testImageSet)))
confidence_interval.append(confidence_interval_pos)
confidence_interval_neg = kaccuracy-1.96* (sqrt((kaccuracy*(1-
kaccuracy))/len(testImageSet)))
confidence_interval.append(confidence_interval_neg)
return confidence_interval
```

confusionmatrix(predictions, actuallabels): - calculates the confusion matrix using the total number of correct predictions and which image was predicted wrongly.

```
def confusionmatrix(predictions, actuallabels):
    row = {}
    for i in range(len(actuallabels)):
      x = str(actuallabels[i]) + str(predictions[i])
      key = "group_{0}".format(x)
      if key in row:
         row["group_{0}".format(x)] = row["group_{0}".format(x)] + 1
      else:
         row["group_{0}".format(x)] = 1

    labelrows = []
    for x in range(0, 10):
      for y in range(0, 10):
         j = str(x) + str(y)
        p = "group_{0}".format(j)
        if p in row:
            labelrows.append(row["group_{0}".format(j)])
       else:
            labelrows.append(0)

    cm = reshape(labelrows, (10, 10))
    return cm
```

Path : ./knn_mnist_09_10/knncode/kNNClassifier.py

iii. **10-Fold Validation** - This file is used to perform kFoldValidation on the training set of MNIST dataset to find the optimal k.
Path : ./knn_mnist_09_10/knncode/kfoldvalidation.py

iv. **k-NN for Optimal K** - This file is used to run the k-NN classifier on MNIST dataset using the optimal k.
Path : ./knn_mnist_09_10/knncode/knnforoptimalk.py

v. **k-NN with Sliding Window** - This file is used to run the k-NN classifier with sliding window on the training image on MNIST dataset using the optimal k.
Path : ./knn_mnist_09_10/knncode/slidingwindow.py

II. MNIST dataset

MNIST dataset is a collection of handwritten digits from 0 to 9. We are using following 4 files to extract the MNIST data.

- ./knn_mnist_09_10/knncode/dataset/t10k-images-idx3-ubyte
- ./knn_mnist_09_10/knncode/dataset/t10k-labels-idx1-ubyte
- ./knn_mnist_09_10/knncode/dataset/train-images-idx3-ubyte
- ./knn_mnist_09_10/knncode/dataset/train-labels-idx1-ubyte

It consists of 60,000 training images with their labels and 10,000 test images with their labels. Each image is a 28*28 matrix.

10-fold cross validation to find out optimal k

I. Steps for 10-fold cross validation are:-

1. Shuffled the training data set which is a 2D array that has 60,000 training images and their labels.
2. After shuffling divided the shuffled training sets into 10 different fold arrays.
3. Each fold was then fed to the MNISTPredictor and kNNClassifier. Labels of each image from one fold is predicted based on the each image in 9 folds i.e. 54,000 images and so on. We repeat the process until all the images in all folds are predicted for a particular value of k.
4. For a particular value of k we calculate the accuracy of each fold and then average of all the 10 folds gives us the accuracy for that value of k in Table 1.
5. Repeated steps 3 and 4 for k = 1 to 10.
6. Below are the logs with detailed results received for each fold accuracy per k.

II. 10- fold accuracy for all Ks from 1 to 10

Table 1

10- fold cross validateon	
k-values	**Accuracy in %**
1	97.26333333333332
2	97.28333333333333
3	97.41000000000002
4	97.45333333333333
5	97.29166666666668
6	97.17666666666667
7	97.12166666666666
8	97.09033333333335

10- fold cross validateon	
9	97.00833333333334
10	96.91500000000002

III. Code Snippet

The complete code is present at **./knncode/kfoldvalidation.py** below is the part of code

```
ks = [1,2,3,4,5,6,7,8,9,10]
all_k_all_fold = []
for k in ks:

    # generating test data and training data from 10 folds
    fold_10_accuracy_each_k = []
    for fold in range(len(foldedArray)):
        training_set = []
        training_set = foldedArray[:]
        test_labels = []
        testing_set = []
        train_set_fold_copy = training_set[fold]
        test_set = train_set_fold_copy[:]
        print("size of test_set---->", len(test_set))

        for i in range(len(test_set)):
            test_labels.append(test_set[i][1])
            testing_set.append(test_set[i][0])

        print("size of test labels :", len(test_labels))

        del training_set[fold]
        print("size of training_set---->", len(training_set))
        fold_training_set = list(itertools.chain.from_iterable(training_set))
        predictor = knn.MNISTPredictor(fold_training_set, k)
        one_train_predictions = knn.predict_test_set(predictor, testing_set)
     one_train_set_accuracy = knn.evaluate_prediction(one_train_predictions, test_labels)
        fold_10_accuracy_each_k.append(one_train_set_accuracy)

    print("Accuracies for all folds for k = ",k," is :", fold_10_accuracy_each_k)
    all_k_all_fold.append((k,fold_10_accuracy_each_k))
    average_accuracy = sum(fold_10_accuracy_each_k) / 10
    print("Average_accuracy for k = ",k," is :", average_accuracy)
    accuracy_in_percent = average_accuracy * 100
    k_accuracy.append((k,accuracy_in_percent))

print("Accuracy for each fold of every k : ",all_k_all_fold)
print("Accuracies for all k ranging from 1 to 10 is :", k_accuracy)
k_accuracy.sort(key=lambda val: val[1])
print("optimal k :", k_accuracy[-1][0])
```

IV. Logs

Logs are zipped with the source code.

Optimal K

Based on the results in Table 1, optimal k is **4** with the maximum accuracy of **97.45%** approximately

K-nn with k = 4 to classify all the images in MNIST test data set

The labels of all the test images in the MNIST test data set is predicted using 60000 training images for k=4.

I. Code Snippet

The complete code is present at **./knncode/knnforoptimalk.py** below is the part of code

```
from numpy import *
import numpy as np
import knncode.kNNClassifier as knn

optimal_k = 4
#Get the training images and labels
import knncode.datareader as datareader
print("loading training images")
trainImagesSet = datareader.parse_images("train-images-idx3-ubyte")
print("size of train image set---->", size(trainImagesSet))
print("shape of train image set---->", shape(trainImagesSet))

print("loading training labels")
trainLabelsSet = datareader.parse_labels("train-labels-idx1-ubyte")
print("size of train image set---->", size(trainLabelsSet))
print("shape of train image set---->", shape(trainLabelsSet))

print("loading test images")
testImagesSet = datareader.parse_images("t10k-images-idx3-ubyte")
print("size of test image set---->", size(testImagesSet))
print("shape of test image set---->", shape(testImagesSet))

print("loading test labels")
testLabelsSet = datareader.parse_labels("t10k-labels-idx1-ubyte")
print("size of test image set---->", size(testLabelsSet))
print("shape of test image set---->", shape(testLabelsSet))


# Shuffling the training Image Set
dataset = []
for i in range(len(trainImagesSet)):
    dataset.append((trainImagesSet[i, :, :], trainLabelsSet[i]))
```

```
#Copy of training image set
copyOfTrainingDataSet = dataset[:]
np.random.shuffle(copyOfTrainingDataSet)

#running knn for optimal K
predictor = knn.MNISTPredictor(copyOfTrainingDataSet, optimal_k)
optimal_k_predictions = knn.predict_test_set(predictor, testImagesSet)
print("Predictions for Optimal k (4) is : ",optimal_k_predictions )


#Accuracy and error calculations
optimal_k_accuracy = knn.evaluate_prediction(optimal_k_predictions, testLabelsSet)
optimal_k_classification_accuracy  = optimal_k_accuracy*100
print(" Classification accuracy for optimal K (4) :", optimal_k_classification_accuracy)
optimal_k_classification_error = (1-optimal_k_accuracy)*100
print(" Classification error for optimal K (4) :", optimal_k_classification_error)

#confidence interval for optimal k
ci = knn.confidenceinterval(optimal_k_classification_accuracy,testImagesSet)
print("Confidence Interval for optimal k (4) : ",ci)


#Confusion matrix for optimal k predictions
cm = knn.confusionmatrix(optimal_k_predictions,testLabelsSet)
print("confusion matrix for optimal-K (4) : ",cm)
```

II. The confusion matrix

Below is the confusion matrix for optimal k i.e. k = 4

```
[[ 973    1    1    0    0    1    3    1    0    0 ]
 [   0 1132    2    0    0    0    1    0    0    0 ]
 [  10    5  995    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    0    0    0 ]
 [   0   17    4    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 ]]
```

III. The classification accuracy

The classification accuracy calculated by dividing the correct labels predicted by total number of test labels for **optimal k =4 is 97.16 %**

Total number of correct predictions = 9716
Total number of test labels = 10000
Accuracy = (Total number of correct predictions/ Total number of test labels)*100
= (9716/10000) *100

Accuracy = 97.16%

Error = 100 - Accuracy = 2.84%

IV. The confidence interval

Confidence interval is calculated by estimating the standard deviation for the accuracy 0.9716 as explained below:-

$$\text{Standard deviation} = \sqrt{((\text{accuracy}*(1-\text{accuracy})/\text{number of test images})}$$

Here the accuracy is the accuracy of optimal k i.e. 4.
Aiming for 95% confidence i.e. using the z-score 1.96, the confidence interval is calculated using the below formula:-

$$CI = \text{accuracy} \pm 1.96 * \text{standard deviation}$$

Confidence interval calculated for accuracy 97.16% is **[0.9748558095629812, 0.9683441904370188].**

V. Logs

Logs are zipped with the source code.

Sliding window in k-NN classifier

I. Steps for sliding window in k-NN classifier:-

1. Each 28*28 training image is converted into 30*30 image(say padded image) by adding an extra row of 0s in the top and bottom of the array and an extra column of 0s in the left and right of the array that represents the training image.
2. Extract 9 28*28 images(say sliced image) out of the above padded image and calculate the euclidean distance of each of the 9 sliced images with each test image.
3. The sliced image (out of 9 sliced images) having the least distance with the test image is then placed in distances array.
4. Using the k = 4 we then find the 4 nearest neighbors for a particular image to predict the label of test image.

II. Code Snippet

```
from numpy import *
import numpy as np
import knncode.kNNClassifier as knn

optimal_k = 4
#Get the training images and labels
import knncode.datareader as datareader

def predict_test_set(predictor, test_set):
    """Compute the prediction for every element of the test set."""
    predictions = [predictor.predictionslidingwindow(test_set[i]) for i in range(len(test_set))]
    return predictions

print("loading training images")
trainImagesSet = datareader.parse_images("train-images-idx3-ubyte")
print("size of train image set---->", size(trainImagesSet))
print("shape of train image set---->", shape(trainImagesSet))

print("loading training labels")
trainLabelsSet = datareader.parse_labels("train-labels-idx1-ubyte")
print("size of train image set---->", size(trainLabelsSet))
print("shape of train image set---->", shape(trainLabelsSet))

print("loading test images")
testImagesSet = datareader.parse_images("t10k-images-idx3-ubyte")
print("size of test image set---->", size(testImagesSet))
print("shape of test image set---->", shape(testImagesSet))

print("loading test labels")
testLabelsSet = datareader.parse_labels("t10k-labels-idx1-ubyte")
print("size of test image set---->", size(testLabelsSet))
print("shape of test image set---->", shape(testLabelsSet))


# Shuffling the training Image Set
dataset = []
```

```
for i in range(len(trainImagesSet)):
    dataset.append((trainImagesSet[i, :, :], trainLabelsSet[i]))

#Copy of training image set
copyOfTrainingDataSet = dataset[:]
#np.random.shuffle(copyOfTrainingDataSet)

testdataset = []
for i in range(len(testImagesSet)):
    testdataset.append((testImagesSet[i, :, :], testLabelsSet[i]))

test_labels = []
test_images = []
for i in range(len(testdataset)):
    test_labels.append(testdataset[i][1])
    test_images.append(testdataset[i][0])

#Sliding Window code
sliding_win_predictor = knn.MNISTPredictor(copyOfTrainingDataSet, optimal_k)
slinding_win_predictions = predict_test_set(sliding_win_predictor, test_images)
print("predictions of sliding window for k = 4 : ", slinding_win_predictions)

#Accuracy and error calculations
sliding_win_accuracy = knn.evaluate_prediction(slinding_win_predictions, test_labels)
sliding_win_classification_accuracy  = sliding_win_accuracy*100
print(" Sliding window classification accuracy for optimal K (4) :",
sliding_win_classification_accuracy)
sliding_win_classification_error = (1-sliding_win_accuracy)*100
print(" Sliding Window classification error for optimal K (4):", sliding_win_classification_error)

#confidence interval for sliding window
sliding_win_ci = knn.confidenceinterval(sliding_win_classification_accuracy,test_images)
print("Sliding Window confidence Interval for optimal k (4): ",sliding_win_ci)

#Confusion matrix for sliding window predictions
sliding_win_cm = knn.confusionmatrix(slinding_win_predictions,test_labels)
print("confusion matrix for sliding window : ",sliding_win_cm)
```

III. The confusion matrix

Below is the confusion matrix for sliding window with the optimal k i.e. k = 4

```
[[ 976    0    1    0    0    1    1    1    0    0 ]
 [  0 1131    2    0    1    0    1    0    0    0 ]
 [  3    2 1019    0    0    0    0    7    1    0 ]
 [  0    1    2  999    0    3    0    2    3    0 ]
 [  0    4    0    0  969    0    3    2    0    4 ]
 [  2    2    0    3    1  869    3    1    0    1 ]
 [  4    4    0    0    4    4  942    0    0    0 ]
 [  0    5    2    0    1    0    0 1016    0    4 ]
 [  3    0    2    2    3    5    2    1  951    5 ]
 [  3    4    1    3    5    6    0    6    0  981 ]]
```

IV. The classification accuracy

The classification accuracy for the sliding window is calculated by dividing the correct labels predicted by total number of test labels for **optimal k =4 is 98.53 %**

Total number of correct predictions = 9853
Total number of test labels = 10000
Accuracy = (Total number of correct predictions/ Total number of test labels)*100
= (9853/10000) *100
Accuracy = 98.53%

Error = 100 - Accuracy = 1.47%

V. The confidence interval

Confidence interval is calculated by estimating the standard deviation for the accuracy 0.9853 as explained below:-

$$\text{Standard deviation} = \sqrt{((\text{accuracy}*(1-\text{accuracy}))/\text{number of test images})}$$

Here the accuracy is the accuracy of optimal k i.e. 4.
Aiming for 95% confidence i.e. using the z-score 1.96, the confidence interval is calculated using the below formula:-

$$CI = \text{accuracy} \pm 1.96 * \text{standard deviation}$$

Confidence interval calculated for accuracy 98.53% is **[0.987658842696239, 0.982941157303761]**

V. Logs

Logs are attached with the source code.

Performance comparison of two k-NN Classifiers

Performance of k-NN classifier with optimal k (accuracy_1) = 0.9716
Performance of k-NN classifier with sliding window and optimal k (accuracy_2) = 0.9853

Difference of these accuracies = accuracy_2 - accuracy_1
= 0.9853 - 0.9716 = 0.013699999999999934

Mean of these accuracies (accuracy_3)= (accuracy_2 + accuracy_1)/Samples
= (0.9853 + 0.9716)/20000
accuracy_3 = 0.000097845

Hypothesis = $1.96 * \sqrt{((accuracy\_3*(1-accuracy\_3)/samples)}$

$= 1.96 * \sqrt{((0.000097845 *(1-0.000097845)/20000)}$

= 0.00013708475004337893

p-value = 0.00013708475004337893

From above, we can conclude that p-value is less than 0.05.
The performance of the k-NN classifier increased by improving the distance metric using the sliding window.

Classification accuracy of standard classifier = 97.16%
Classification accuracy of sliding window classifier = 98.53%
Increase = 98.53 - 97.16 = 1.37
Percent increase = (1.37/97.16) * 100 = 1.41004 %

The classification accuracy of sliding window classifier has increased by **1.41%** from the standard classifier and hence the sliding window classifier is significantly better than the standard k-NN Classifier based on the 0.2% difference rule.