CSC 635 Data Mining

## Assignment 3 Report

### Submitted to:

### Dr. Jamil Saquer

### Author(s):

### Rafail Islam

**K Nearest Neighbors**

**Introduction**

In this assignment, we worked with a data set of handwritten digits. The dataset has 785 attributes; the first attribute is the class labels which are the number digits from 0 to 9. Other attributes have pixel values from 0 to 255. Our task is to implement an algorithm that can classify new handwritten digits. For this assignment, we implement distance-based classifier-K nearest neighbors to classify the test data set of handwritten digits.

**Background**

Classification problems are very common in everyday life. In our previous assignment, we have implemented another classifier- decision trees. Unlike, decision trees, KNN is a simple algorithm that stores all available cases and classifies new cases based on a similarity measure (e.g., distance functions). KNN has been used in statistical estimation and pattern recognition already at the beginning of 1970’s as a non-parametric technique.

In KNN, the class of the new testing sample is determined by the majority vote of its neighbors. That being said, it finds the majority class within its K nearest neighbors. Find the similarity is the most important part of KNN algorithms, in this assignment, we are told to use Euclidean distance between two input vectors. However, I wrote a function that can be used for both Manhattan and Euclidean distances. If you chose p = 2, this function will return Euclidean distance in Figure 1.

# Evaluating distance between two tuples

**def** get\_distance**(**D**,**t**,** p **=** 2**):**

""" This function determine distace between two vectors

Parameters:

--------------

D : 1D array/list

input vector in train dataset

t : 1D array/list

input vector in test dataset

p : int (default=2, 2 for euclidean, 1 for manhattan )

type of distance

Returns:

--------------

distance : float32

distance between two input vectors

"""

# initialize distance

distance **=** 0.

**for** x **in** range**(**len**(**D**)-**1**):**

distance **+=** **(**abs**(**D**[**x**+**1**]-**t**[**x**]))\*\***p

distance **=** distance **\*\*(**1**/**p**)**

**return** distance

*Figure 1: Distance Function.*

**Implementation**

To implement KNN algorithms, I closely followed the steps given in the lecture slide. This algorithm does not build a model from the training data set. When we need to classify a new input vector, I calculated the distance of that input vector from the training data set. We kept only K numbers distances for the voting process. If any of the new instances get a smaller distance, we replaced that distance with the largest distance in our current distances list. Once we have done calculating the distances, we need to count the majority voting from the distance metrics. Instead of uniform voting, we used weighted voting to get better accuracy.

Apart from implementing strategies for getting better accuracy, finding the optimal value of K for a specific data set is also crucial. Hence, we planned to find the best value of K. Finding the best values of K, we implement a function where we used random samples from the training data set and determine the accuracy with the testing data set. We chose the value of k with higher accuracy.

# Find best value of k

**def** bestK**(**train**,**test**,**kmin**=**2**,**kmax**=**10**):**

# dictionary for storing value of K as key and Accuracy of correspoing value of K as value

k\_hist **=** **{}**

**for** k **in** range**(**kmin**,**kmax**+**1**):**

#train\_samples = sample(train, len(train)\*0.4)

# take random sample from train data set

# size of random sample = 30% of training data set

train\_samples **=** train**[**np**.**random**.**choice**(**train**.**shape**[**0**],** int**(**len**(**train**)\***0.3**),** replace**=False),** **:]**

\_**,**acc **=** classify**(**train\_samples**,**test**,**k**)**

k\_hist**[**k**]** **=** acc

# best value of k with maximum weight voting

best\_k **=** max**(**k\_hist**,**key**=**k\_hist**.**get**)**

# Plot k VS accuracy

lists **=** sorted**(**k\_hist**.**items**())**

x**,**y **=** zip**(\***lists**)**

#print(x,y)

plt**.**plot**(**x**,**y**)**

plt**.**xlabel**(**"Value of K"**)**

plt**.**ylabel**(**"Accuracy with random sample"**)**

plt**.**title**(**"K vs. Accuracy"**)**

plt**.**show**()**

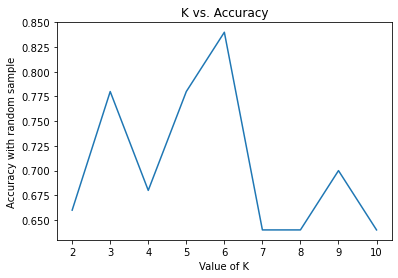
**return** best\_k

*Figure 2: Function for finding best value of k*.

**Experimental Setup and Results**

I implemented this assignment in jupyter notebook and used python 3.8. I did not preprocess the data set: only converted into NumPy arrays for convenience. One important thing, I noticed while running this assignment. While data scaling provides better accuracy in KNN, it did not help in this assignment. That said, data scaling was not necessary here, as all the attributes have the same value range (eg. 0-255).

While finding the best value of k, we have seen that it did not give the same value always as we took random samples of the training data set. Figure 3 shows one instance of the best value of K. And we got 86.00% accuracy with k=6.



*Figure 3. Graph for choosing best value of K.*

However, we found that we got the best accuracy 88.00% with k=5.

**Conclusion**

While KNN is easy to implement, it has a big drawback. It is much slower than many other classifiers while predicting new instances. That’s why KNN is a lazy learner. Though, we implement a way to find the best value of k. It’s not consistent all the time due to its randomness nature.

**Code**

"""

hw3

Author: Rafail Islam

CSC 635

"""

#-----------------------------------import--------------------------------

**import** numpy **as** np

**import** pandas **as** pd

**import** matplotlib**.**pyplot **as** plt

**from** sklearn**.**metrics **import** accuracy\_score

**from** random **import** sample

#----------------------------------global variables------------------------

# Load data

train\_data **=** pd**.**read\_csv**(**"MNIST\_train.csv"**)**

test\_data **=** pd**.**read\_csv**(**'MNIST\_test.csv'**)**

# convert to numpy

train **=** train\_data**.**to\_numpy**()**

test **=** test\_data**.**to\_numpy**()**

#-------------------------------Functions--------------------------------------

# Evaluating distance between two tuples

**def** get\_distance**(**D**,**t**,** p **=** 2**):**

""" This function determine distace between two vectors

Parameters:

--------------

D : 1D array/list

input vector in train dataset

t : 1D array/list

input vector in test dataset

p : int (default=2, 2 for euclidean, 1 for manhattan )

type of distance

Returns:

--------------

distance : float32

distance between two input vectors

"""

# initialize distance

distance **=** 0.

**for** x **in** range**(**len**(**D**)-**1**):**

distance **+=** **(**abs**(**D**[**x**+**1**]-**t**[**x**]))\*\***p

distance **=** distance **\*\*(**1**/**p**)**

**return** distance

# Build KNN

**def** knn**(**D**,**t**,**k**):**

""" this function preditcs the class of a test sample by K-nearest Neighbours algorithm

Parametes:

-------------

D : 2D numpy arrays

training data set

t : 1D array/list

single test sample

k : int

number of nearest neighbours

Returns

-------------

label\_by\_voting : class type

predicted class label by uniform voting

label\_by\_weight\_voting: class type

predicted class label by weighted voting

"""

# empty list for tuples

N **=** **[]**

# list for storing best distances

dist **=** **[]**

# for each data in data set

**for** d **in** D**:**

# get distance

distance **=** get\_distance**(**d**,**t**)**

# append the label of the data in neighbour list

**if** len**(**N**)** **<** k**:**

N**.**append**(**d**[**0**])**

dist**.**append**(**distance**)**

**elif** distance **<** np**.**max**(**dist**):**

dist**.**remove**(**np**.**max**(**dist**))**

N**.**pop**(**np**.**argmax**(**dist**))**

dist**.**append**(**distance**)**

N**.**append**(**d**[**0**])**

# I did both uniform count voting and weight voting to see the progress

#--------count vote---------

#print(N)

#print(dist)

label\_by\_voting **=** max**(**set**(**N**),**key**=**N**.**count**)**

#--------count weighted vote------

label\_dic **=** **{}**

# get all unique class from the taining data set and make a dictionary

# where key is the class and value is weighted vote. Values are set to 0.0 initially.

**for** x **in** D**[:,**0**]:**

label\_dic**[**x**]=**0.0

#print(label\_dic)

# calculate distance weighted voting

**for** i **in** range**(**k**):**

label\_dic**[**N**[**i**]]** **+=** 1**/**dist**[**i**]\*\***2

#print(label\_dic)

# get key with maximum weighted voting

label\_by\_weight\_voting **=** max**(**label\_dic**,**key**=**label\_dic**.**get**)**

**return** label\_by\_voting**,** label\_by\_weight\_voting

# Accuracy function for derterming accuracy

**def** classify**(**train**,**test**,**k**):**

""" This function calculate accuracy of all test data over given train data set.

Parameters

------------

train : 2D numpy arrays

training data set

test : 2D array/list

test data set

k : int

number of nearest neighbours

Returns

------------

accuracy\_voting : float

accuracy by uniform voting

accuracy\_weight\_voting : float

accuracy by distace weight voting

"""

prd\_voting **=** **[]**

prd\_weight\_voting **=** **[]**

**for** t **in** test**:**

clss1**,** clss2**=** knn**(**train**,**t**,**k**)**

prd\_voting**.**append**(**clss1**)**

prd\_weight\_voting**.**append**(**clss2**)**

#print(" vot : %.1f%%" %(accuracy\_score(test[:,0],np.asarray(prd\_voting) )\*100) )

#print(" w\_vot : %.1f%%" %(accuracy\_score(test[:,0],prd\_weight\_voting)\*100) )

# evaluate voting accuracy using accuracy\_score metrics

accuracy\_voting **=** accuracy\_score**(**test**[:,**0**],**np**.**asarray**(**prd\_voting**)** **)**

accuracy\_weight\_voting **=**accuracy\_score**(**test**[:,**0**],**prd\_weight\_voting**)**

**return** accuracy\_voting **,** accuracy\_weight\_voting

# Find best value of k

**def** bestK**(**train**,**test**,**kmin**=**2**,**kmax**=**10**):**

""" This function determine best value of k from the given range.

Parameters

------------

train : 2D array

tarin data set

test : 2D array

test data set

kmin : int (default: 2)

minimum value of k

kmax : int (default: 10)

maximum value of k

Returns

------------

best\_k : int

best value of k

"""

# dictionary for storing value of K as key and Accuracy of correspoing value of K as value

k\_hist **=** **{}**

**for** k **in** range**(**kmin**,**kmax**+**1**):**

#train\_samples = sample(train, len(train)\*0.4)

# take random sample from train data set

# size of random sample = 30% of training data set

train\_samples **=** train**[**np**.**random**.**choice**(**train**.**shape**[**0**],** int**(**len**(**train**)\***0.3**),** replace**=False),** **:]**

\_**,**acc **=** classify**(**train\_samples**,**test**,**k**)**

k\_hist**[**k**]** **=** acc

# best value of k with maximum weight voting

best\_k **=** max**(**k\_hist**,**key**=**k\_hist**.**get**)**

# Plot k VS accuracy

lists **=** sorted**(**k\_hist**.**items**())**

x**,**y **=** zip**(\***lists**)**

#print(x,y)

plt**.**plot**(**x**,**y**)**

plt**.**xlabel**(**"Value of K"**)**

plt**.**ylabel**(**"Accuracy with random sample"**)**

plt**.**title**(**"K vs. Accuracy"**)**

plt**.**show**()**

**return** best\_k

# Main function to call all the necessary functions

**def** main**():**

# finding best value of k

# Note: uncomment the line below to find best value of k. But I am not sure. If this is optimal. Dr. Saquer

# said that you may use random sample to find best value of k. However, It does not give same value of K

# all the time as it is randomized.

k **=** bestK**(**train**,**test**)**

# I found k=5 works better in this case. We can use K=5 to get maximum accuracy.

#k = 5

**print(**"K ="**,**k**)**

missClassified **=** 0

**for** t **in** test**:**

clss1**,** clss2 **=** knn**(**train**,**t**,**k**)**

**print(**"Desired class: "**,**t**[**0**],**" computed class: "**,**clss2**)**

**if** t**[**0**]** **!=** clss2 **:**

missClassified **+=** 1

#print("Desired class: ",t[0]," computed class: ",clss2)

accuracy **=** **(** 1 **-** missClassified**/**len**(**test**)** **)** **\*** 100

**print(**"Accuracy rate: %.1f%%" **%**accuracy**)**

**print(**"Number of misclassified test samples: "**,** missClassified**)**

**print(**"Total number of test samples: "**,**len**(**test**))**

main**()**

#-------------------------------------End of Functions--------------------------------