**Pattern Recognition**

Mini-Project #4

Subject: Non-parametric methods

**Instructor: Dr. Yazdi**

**Hamze Ghaedi**

**STN: 9831419**

**Azar 99**

**Table of contents**

**Abstract…………………………………………………………………**

**PDF estimation (theory)…………………………………………**

**Problem #1…………………………………………………………….**

**Parazen PDF estimation………………………………….**

**Rectangular window……………………………….**

**Gaussian window……………………………………**

**Problem #2…………………………………………………………….**

**K-NN PDF estimation………………………………………**

**Problem #3…………………………………………………………….**

**K-NN Classifier………………………………………………..**

**Conclusion……………………………………………………………..**

**Appendix #1…………………………………………………………..**

**Codes for problem #1……………………………………..**

**Appendix #2…………………………………………………………..**

**Codes for problem #2……………………………………..**

**Appendix #3…………………………………………………………..**

**Codes for problem #3……………………………………..**

**Abstract**

Solving pattern recognition problems, usually leads to a decision-making process.

Bayes inference, is well-known and common method Applied to decision making problems, which we have used it so far.

Applying Bayes inference, needs the distribution of samples to be known but, usually it is not available!

So far, we have bypassed this issue by assuming a parametric model for the distributions (Gaussian, ...) which consists of some unknown parameters (mean and variance in the case of gaussian). Then the problem reduced to estimating these unknowns.

Another way to dealing with unknown distribution Is to estimate it directly without assuming any parametric model.

The latter, is called **Non-parametric method** which is the main subject of this mini project.

So, in this project, we are focusing on Non parametric methods (**K-NN** and **Parazen** **Window**) to

1. **Estimating unknown distributions**
2. **Addressing classification problem**

For MNIST handwritten digits dataset.

assuming Dn = {x1, x2, …, xn} be a set of n i.i.d samples each drawn from an unknown distribution p(x) our gola is to estimate p(x) based on the dataset Dn. we know that the probability of a given sample x falls in the region R is:

Because the samples are i.i.d, the probability of k samples out of n samples fall in the region R, follows binomial distribution as below:

and

Which mean if n samples are drawn from p(x), then n\*P samples fall in R in average, if we know the number of samples fall in R (let it be k) then we have:

In the other hand, assuming the volume of R is so small such that p(x) changes slowly in R we can substitute p(x’) by its value at a fixed-point x inside R, so

Which V is the volume of R. according to (1) and (2) we conclude that:

It can be proved that, if

The last equation approaches p(x).

**PDF Estimation (Theory)**

**Problem #1**

***Parazen Method***

**Parazen Method**

So, our PDF estimation methods is based on below equation:

In Parazen method, the volume V of the region R, assumed to be known thus, k, the number of samples falls in R, can be determined. For example, if we take R as a d dimensional hypercube, its volume become V = h^d which h is the length of edge of the hypercube. In order to calculate the k, we define:

thus, k can be calculated as follows:

So,

As discussed, convergence may be guaranteed if the volume approaches zero as number of samples goes to infinity thus, we may let for this purpose.

(h1 is an arbitrary initial value which there is no general rule for choosing it appropriately)

called Window Function and it may takes any shape. But generally, it would be better if it obeys distribution function rules (nonnegativity and integrated to 1). In Problem #1 we’ll study and compare two different window functions.

we take window function as follows:

So, it is a rectangular shape centered at origin. We can interpret the (\*) equation at previous page as linear combination of shifted phi functions, in other words it approximates the PDF as linear combination of shifted window functions.

So, by defining such as defined above, we are trying to approximate the PDF as sums of rectangular shapes.

Clearly, rectangular window cannot capture smoothly parts of the PDF well.

implementing bayes classifier based on Parazen method with rectangular window leads to overall accuracy of 83 %.

the following list demonstrates accuracies per different values of h1:

**Parazen Method with Rectangular Window**

|  |  |
| --- | --- |
| h | acc |
| 1.5 | 79.6% |
| 1.6 | **83%** |
| 1.7 | 81.27% |
| 1.8 | 76% |

**Parazen Method with Gaussian Window**

Instead of rectangular shape, we define as below

Which is the zero mean and unit variance gaussian distribution. So according to the (\*) equation,

Thus, the Parazen method approximates the unknown PDF by a linear combination of shifted gaussian windows.

Implementing the classification schema for reduced MNIST handwritten digits based on Parazen Method with gaussian window achieves the overall accuracy of 93 %

The following tables demonstrates the accuracies per different values for h:

|  |  |
| --- | --- |
| h | acc |
| 0.1 | 93% |
| 0.5 | 92 % |
| 0.6 | 91% |
| 0.7 | 90% |

Generally, rectangular window is simpler to understand and easier to compute than gaussian window but it is less capable of capturing smoothness in the PDF than gaussian window.

Besides this, many phenomena obey gaussian distribution so, in general, it would be no surprise if gaussian window achieves better results than rectangular window as in the case of the problem #1.

in this problem, gaussian window reaches the overall accuracy of 93% which is 10% better than rectangular window

**Rectangular window vs Gaussian window**

**Appendix #1 contains codes and related explanations for problem #1**

Rectangular Window

Acc = 83%

Gaussian Window

Acc = 93%

**Problem #2**

***K-NN for PDF estimation***

|  |  |
| --- | --- |
| k | acc |
| 1 | **93%** |
| 2 | 92% |
| 3 | 91% |

In our main equation for estimating an unknown pdf, instead of V, we can adjust k, the number of points inside R, then determine the smallest volume V such that all k points fall inside it, this approach is called K Nearest Neighbor.

the volume V depends on the distance between the given sample, x, and the furthest point among its k Nearest points. (according to assignment#4 we called it Radius) so

Which is a constant factor.

(there is no general way to finding appropriate value for k)

Generally, our K-NN estimator finds the distance between a given test sample x and the kth smallest point in each class Then computes a value for the joint density p(x, wi) which is proportional to this distance.

After implementing K-NN pdf estimation method and establish a bayes classification for our dataset, the overall accuracy of 93 % achieved.

the table below, demonstrates accuracy for some different values of k

**K-Nearest Neighbor Method**

**Appendix #2 contains codes and related explanations for problem #2**

**Problem #3**

***KNN Classifier***

we can use KNN method to classify patterns by estimating posterior probability of classes directly:

Where k is the number nearest neighbors of x and ki of which turn out be labelled wi . Consequently, we must select the category most frequently represented in the cell.

Fitting a KNN classifier to a given dataset is relatively simple in compare with any other schema.

We must find K nearest neighbor of given sample x (based on a measure for distance) and determine the most frequently class among these K neighbors. Again, finding appropriate value for k is a challenging task.

In this problem, three KNN classifier for k = 1, 3, 5 along with Euclidean distance are fitted to MNIST handwritten digits. The results are listed below:

**Appendix #3 contains codes and related explanations for problem #3**

|  |  |
| --- | --- |
| k | acc |
| 1 | **93%** |
| 3 | 91% |
| 5 | 90% |

**KNN Classifier**

In this mini-project, the nonparametric approach (KNN and Parazen) to pattern recognition was studied.

KNN is an intuitive approach especially for classification. Parazen window is a flexible method for estimating an

Unknown PDFs. Both methods have meta parameters that must be taken carefully. **Both methods are based on same theories so, it would be no surprise if with appropriate values for their parameters (h and window function in Parazen and k and distance measure in KNN) both lead to same results as in the case of this project!**

**Conclusion**

**Appendix #1**

***Codes for Parazen estimation method***

#----------------Rectangular window---------------

def rect\_win(x,x\_i,h):

  n\_features = x.shape[0]

  for i in range(n\_features):

    dist = np.abs((x[i] - x\_i[i]) / h)

    if(dist > 0.5):

      return 0

  return 1

#-------------------------------------------------

Function below, imitates the notion of the rectangular window as discussed, it takes a training sample (x\_i), a given test sample (x) and the parazen window parameter (h), then returns 0 if given test sample (x) is out of the hypercube centered at x\_i and 1 otherwise.

#loading datasets------------------------

x\_train = pd.read\_csv("drive/MyDrive/mnist/Train\_Data.csv").to\_numpy()

y\_train = pd.read\_csv("drive/MyDrive/mnist/Train\_labels.csv").to\_numpy()

x\_test = pd.read\_csv("drive/MyDrive/mnist/Test\_Data.csv").to\_numpy()

y\_test = pd.read\_csv("drive/MyDrive/mnist/Test\_labels.csv").to\_numpy()

#----------------------------------------

Related datasets have been loaded form my google drive

import numpy as np

import pandas as pd

necessary libraries for reading and manipulating data are listed below:

1. NumPy for matrix manipulation
2. Pandas for reading csv

#----------------------Parazen----------------------

#computes p(x|wi)

def parazen\_method(x\_train,y\_train,x,h,window):

  n\_samples = x\_train.shape[0]

  n\_features = x\_train.shape[1]

  probs = np.zeros((10,1)) #place holder for p(x|wi)

  #V = h\*\* n\_features

  for i in range(n\_samples):

    x\_i = x\_train[i]

    y\_i = int(y\_train[i][0])

    probs[y\_i] += window(x\_i,x,h)

  return probs

#--------------------------------------------------

Function below, takes training points along with their labels, a given test sample x, parazen window parameter h and a window function handle (via window argument) then initiate a loop through training samples and At first, the corresponding label (y\_i) of the ith training point (x\_i) is determined then x\_i and the given test point (x) along with parameter h is injected to a given window function and the resulted value Is added to the corresponding likelihood of a class which y\_i determines. The function returns likelihood of classes for a given test sample x.

#----------------Gaussian window------------------

#zero mean and unit variance

def gaussian\_win(x,x\_i,h):

  euclid\_dist = np.dot(x - x\_i,x - x\_i) / (2\*h\*\*2)

  exp = np.exp(- euclid\_dist)

  exp /= np.pi

  return exp

#-------------------------------------------------

Function below, imitates the notion of the gaussian window as discussed, it takes a training sample (x\_i), a given test sample (x) and the parazen window parameter (h), then returns a value based on formulas discussed in problem #1.

def evaluate(y\_test,y\_pred):

  n\_samples = y\_pred.shape[0]

  n\_correct = 0;

  for i in range(n\_samples):

    if(y\_test[i]== y\_pred[i]):

      n\_correct += 1

  acc = n\_correct / n\_samples

  return acc

Function below, computes accuracy criteria for classification process by taking prediction labels (y\_pred) resulted form classifier and true labels (y\_test)

#-------------Bayesian Classifier-----------------

# p(wi|x) > p(wj|x) for all i != j --> choose wi

# p(wi|x) = p(x|wi)\*p(wi)/p(x)

# assuiming equal prior (p(wi) = 0.1 for (1 <= i <= 10)) then

# bayesian classifier becomes : p(x|wi) > p(x|wj) for all i != j --> choose wi

def bayes\_classifier(probs):

  return np.argmax(probs)

#-------------------------------------------------

Function below, takes a vector consists of probabilities of classes given a test sample and

Returns the index of the most probable of them.

**Review (Bayesian classification)**

**we want to assign a new pattern x to one of n given classes names**

**If be the probability of class i given test sample x, bayes classifier rule is as follows:**

#---------------------------Problem #1--------------------------------

#helper function---------------

def classify(h,window):

  n\_test\_samples = x\_test.shape[0]

  y\_pred = np.zeros((n\_test\_samples,1))

  for i in range(n\_test\_samples):

    probs = parazen\_method(x\_train,y\_train,x\_test[i],h,window)

    y\_pred[i] = bayes\_classifier(probs)

  acc = evaluate(y\_test,y\_pred)

  return acc

#------------------------------

#-------------------------------

#compute accuracy for h = 1.5,1.6,1.7,1.8 and rectangular window

rect\_acc = np.zeros((4,1))

rect\_acc[0] = classify(1.5,rect\_win)

rect\_acc[1] = classify(1.6,rect\_win)

rect\_acc[2] = classify(1.7,rect\_win)

rect\_acc[3] = classify(1.8,rect\_win)

#---------------------------------

#compute accuracy for h = 0.1,0.5,0.6,0.7 and rectangular window

gaus\_acc = np.zeros((4,1))

gaus\_acc[0] = classify(0.1,gaussian\_win)

gaus\_acc[1] = classify(0.5,gaussian\_win)

gaus\_acc[2] = classify(0.6,gaussian\_win)

gaus\_acc[3] = classify(0.7,gaussian\_win)

#---------------------------------

#----------------------------------------------------------------------------

The script below, put all previous functions together in order to solve the Problem #1.

At the first step, a helper function named classify is defined, this function takes a parameter h and a window function handle then initiates a loop through all test samples and returns the accuracy of classification.

Using classify helper function, the classification process is conducted for numerous values of h and different window functions.

print("Rectangular window with h = 1.5 :")

print(" acc = ",rect\_acc[0][0],"%")

print("Rectangular window with h = 1.6 :")

print(" acc = ",rect\_acc[1][0],"%")

print("Rectangular window with h = 1.7 :")

print(" acc = ",rect\_acc[2][0],"%")

print("Rectangular window with h = 1.8 :")

print(" acc = ",rect\_acc[3][0],"%")

print("Gaussian window with h = 0.1 :")

print(" acc = ",gaus\_acc[0][0],"%")

print("Gaussian window with h = 0.5 :")

print(" acc = ",gaus\_acc[1][0],"%")

print("Gaussian window with h = 0.6 :")

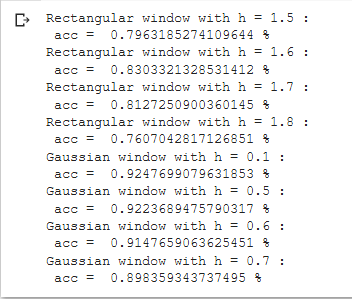
print(" acc = ",gaus\_acc[2][0],"%")

print("Gaussian window with h = 0.7 :")

print(" acc = ",gaus\_acc[3][0],"%")

the script below, prints accuracies resulted from classify function per different configurations

**The printed list is depicted below.**



**Appendix #2**

***Codes for KNN estimation method***

#-------------------top3\_Neighbour\_points----------------

#returns top 3 nearset points to a given point x among training points

def top3\_Neighbour\_points(x\_train,y\_train,x):

  n\_samples = x\_train.shape[0]

  dists = np.zeros((10,3)) #placeholder for top 3 smallest distances in each of 10 classes

  dists.fill(float('inf')) #fills all dists elements with float infinity (\*\*) (Line 23)

  for i in range(n\_samples):

    x\_i = x\_train[i]

    y\_i = int(y\_train[i])

    dist = np.linalg.norm(x - x\_i)

    if(dists[y\_i][2] <= dist): # (\*\*) initial values of dists must be greater than any possible distance ! (Line 17)

      continue

    else:

      dists[y\_i][2] = dist

      if(dists[y\_i][1] > dist):

        dists[y\_i][1],dists[y\_i][2] = dists[y\_i][2],dists[y\_i][1] #swap

      if(dists[y\_i][0] > dist):

        dists[y\_i][0],dists[y\_i][1] = dists[y\_i][1],dists[y\_i][0] #swap

  return dists

#----------------------------------------------------------

After importing necessary libraries and loading datasets.

Function top3\_Neighbour\_points receives training set and a given test sample

Then finds the distances of top 3 nearest point to x in each class.

Finally returns (10 \* 3) matrix each row contains distances to 3NN of x in one of 10 classes.

#--------------------------evaluate--------------------------

def evaluate(y\_pred,y\_test):

  n\_samples = y\_test.shape[0]

  n\_corrects = 0

  for i in range(n\_samples):

    if(y\_pred[i] == y\_test[i]):

      n\_corrects += 1

  acc = n\_corrects / n\_samples

  return acc

#------------------------------------------------------------

Same as problem #1

#---------------------bayes\_classifier----------------------

#decides based on MAP

def bayes\_classifier(probs):

  preds = np.zeros((3,1))

  preds[0][0] = np.argmax(probs[:,0])

  preds[1][0] = np.argmax(probs[:,1])

  preds[2][0] = np.argmax(probs[:,2])

  return preds

#------------------------------------------------------------

Bayes\_classifier function is just like it’s counterpart in problem 1 but it repeats the operation for 3 set of probabilities corresponding to k = 1, 2, 3

#--------------------PROBLEM #2-----------------------------

n\_test\_samples = x\_test.shape[0]

y\_pred = np.zeros((n\_test\_samples,3))

#loop through all test points---------------

for i in range(n\_test\_samples):

  top3\_dists = top3\_Neighbour\_points(x\_train,y\_train,x\_test[i]) #find top 3 NN

  probs = KNN\_estimator(top3\_dists) #computes probabilities based on top 3 NN

  preds = bayes\_classifier(probs) #MAP prediction

  y\_pred[i] = preds.reshape((3,)) #accuracy evaluation

#-------------------------------------------

In the script below, a loop through all test samples is initiated and for each test samples the distance between the sample and top 3 nearest of its neighbors in each class is determined then based on these distances the likelihood of each class give test sample x is evaluated for k = 1, 2, 3, finally using these likelihoods (assuming equal priori) a Bayesian classifier Is established

#--------------------KNN\_estimator-------------------------

def KNN\_estimator(NNpoints):

  probs = np.zeros((10,3))

  #common constants have been eliminated

  for i in range(10):

    probs[i][0] = 1 / NNpoints[i][0]

    probs[i][1] = 1 / NNpoints[i][1]

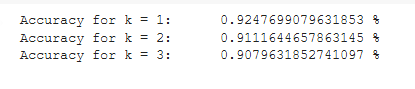
    probs[i][2] = 1 / NNpoints[i][2]

  return probs;

#-----------------------------------------------------------

As discussed in text, in KNN pdf estimator, the likelihood of each class given test sample x is proportional to the radius of volume V enspheres top k Nearest Neighbors of the given sample.

NOTE: constant factors are omitted.



#----------------------------RESULTS-------------------

acck1 = evaluate(y\_pred[:,0],y\_test)

acck2 = evaluate(y\_pred[:,1],y\_test)

acck3 = evaluate(y\_pred[:,2],y\_test)

print("Accuracy for k = 1:\t",acck1,"%")

print("Accuracy for k = 2:\t",acck2,"%")

print("Accuracy for k = 3:\t",acck3,"%")

#------------------------------------------------------

Printing the results

**Appendix #3**

***Codes for KNN Classifier***

#-------------------------------------------------------------------------------

def KNN(x\_train,y\_train,x):

  n\_samples = x\_train.shape[0]

  dists = np.zeros((n\_samples,1))

  for i in range(n\_samples):

    dists[i,0] = np.linalg.norm(x - x\_train[i]) #euclidean norm

  t = np.argsort(dists[:,0]) #sort indices based on calculated distances ascendingly

  y\_pred = np.zeros((3,1)) #reserve three placeholder for k = 1, 3, 5

  y\_pred[0] = int(y\_train[int(t[0])]) # k = 1 Nearest Neighbor

#-------------------------(K = 3)------------------------------

  #counting occurance of classes in top 3 nearset neighbors

  freq = np.zeros((10,1))

  k = 3

  for i in range(k): # k = 3

    freq[int(y\_train[t[i]])] += 1

  #sort class indices base on number of occurance

  freq\_arg = np.argsort(freq[:,0]) #ascending

  freq\_arg = freq\_arg[::-1] #turns to descending

  y\_pred[1] = freq\_arg[0] #pick majory

#-----------------------------------------------------------------

#------------------------(K = 5)----------------------------------

  freq = np.zeros((10,1))

  k = 5

  for i in range(k):

    freq[int(y\_train[t[i]])] += 1

  freq\_arg = np.argsort(freq[:,0])

  freq\_arg = freq\_arg[::-1]

  y\_pred[2] = freq\_arg[0]

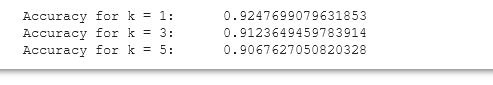
#-----------------------------------------------------------------

  return y\_pred #end KNN function

#-------------------------------------------------------------------------------

KNN function is a brute force implementation of a KNN classifier. It takes a test sample x

Along with training dataset then finds distances between x and all training samples then sorts the indices of training samples based on their corresponding distances ascendingly finally picks the label of majority class among top k sorted training samples as predicted label for x



#------------------------------------PROBLEM 3----------------------------

n\_test\_samples = x\_test.shape[0]

y\_pred = np.zeros((n\_test\_samples,3))

for i in range(n\_test\_samples):

  y\_pred[i] = KNN(x\_train,y\_train,x\_test[i]).reshape((3,))

print("Accuracy for k = 1:\t",evaluate(y\_pred[:,0],y\_test))

print("Accuracy for k = 3:\t",evaluate(y\_pred[:,1],y\_test))

print("Accuracy for k = 5:\t",evaluate(y\_pred[:,2],y\_test))

#-------------------------------------------------------------------------

Script below, initiates a loop through all test samples and makes predict for each of them using KNN method defined in previous pages. KNN returns three predictions corresponding to K = 1, 3, 5. The results of classifications are stored in y\_pred which is a matrix each of its rows corresponds to a test samples and has three columns each one contains classification result for k = 1, 3, 5 respectively.