

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.



PDF Estimation (Theory)

assuming $D_n = \{x_1, x_2, \dots, x_n\}$ be a set of n i.i.d samples each drawn from an unknown distribution $p(x)$ our goal is to estimate $p(x)$ based on the dataset D_n . we know that the probability of a given sample x falls in the region R is:

$$P = \int_R p(x') dx'$$

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:

$$P_k = \binom{n}{k} P^k (1 - P)^{n-k}$$

and

$$E\{k\} = n * P$$

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:

$$P \approx \frac{k}{n} \quad (1)$$

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

$$P = \int_R p(x') dx' = p(x)V \quad (2)$$

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

$$\frac{k}{n} = P = \int_R p(x') dx' \approx p(x)V$$

$$p(x) \approx \frac{k}{nV}$$

It can be proved that, if

$$k \rightarrow \infty, V \rightarrow 0, \frac{k}{n} \rightarrow 0 \quad \text{while} \quad n \rightarrow \infty$$

The last equation approaches $p(x)$.

Problem #1

Parazen Method

Parazen Method

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

$$p(X) \approx \frac{k}{nV}$$

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:

$$\phi(u) = \begin{cases} 1 & |u_i| < \frac{1}{2} \text{ for } 1 < i < d \\ 0 & \text{otherwise} \end{cases}$$

thus, k can be calculated as follows:

$$k = \sum_{i=1}^n \phi(x - x_i)$$

So,

$$p_n(x) \approx \frac{1}{n} \sum_{i=1}^n \frac{1}{V_n} \phi\left(\frac{x - x_i}{h_n}\right) \quad (*)$$

As discussed, convergence may be guaranteed if the volume approaches zero as number of samples goes to infinity thus, we may let $V = h_n^d$ and $h_n = \frac{h_1}{\sqrt[n]{n}}$ for this purpose.

(h_1 is an arbitrary initial value which there is no general rule for choosing it appropriately)

$\phi(x)$ is 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.

Parazen Method with Rectangular Window

we take window function ϕ as follows:

$$\phi(u) = \begin{cases} 1 & |u_i| < \frac{1}{2} \text{ for } 1 < i < d \\ 0 & \text{otherwise} \end{cases}$$

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 ϕ 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 h_1 :

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 ϕ as below

$$\phi(u) = \frac{1}{\sqrt{2\pi}} e^{-\frac{u^2}{2}}$$

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

$$p_n(x) \approx \frac{1}{n} \sum_{i=1}^n \frac{1}{V_n} \frac{1}{\sqrt{2\pi}} e^{\left(\frac{(x-x_i)^T(x-x_i)}{2h_n^2} \right)}$$

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%

Rectangular window vs Gaussian window

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

<p>Gaussian Window</p> <p>Acc = 93%</p>	<p>Rectangular Window</p> <p>Acc = 83%</p>
---	--

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

Problem #2

K-NN for PDF estimation

K-Nearest Neighbor Method

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

$$V \sim R_k(x)$$
$$p_{knn}(x) = \frac{\alpha}{R_k(x)}$$

Which α 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 k th smallest point in each class Then computes a value for the joint density $p(x, w_i)$ 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	acc
1	93%
2	92%
3	91%

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

Problem #3

KNN Classifier

KNN Classifier

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

$$p(x, w_i) \approx \frac{k_i}{nV}$$
$$p(w_i|x) \approx \frac{p(x, w_i)}{p(x)} = \frac{\frac{k_i}{nV}}{\frac{k}{nV}} = \frac{k_i}{k}$$

Where k is the number nearest neighbors of x and k_i of which turn out be labelled w_i . 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:

k	acc
1	93%
3	91%
5	90%


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

Conclusion

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

KNN is an intuitive approach especially for classification. Parzen 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 Parzen and k and distance measure in KNN) both lead to same results as in the case of this project!**



Appendix #1

Codes for Parazen estimation method

1

necessary libraries for reading and manipulating data are listed below:

- 1) NumPy for matrix manipulation
- 2) Pandas for reading csv

```
import numpy as np
import pandas as pd
```

2

Related datasets have been loaded from my google drive

```
#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()
#-----
```

3

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.

```
#-----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
#-----
```


4

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.

```
#-----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
#-----
```

5

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 i th 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 .

```
#-----Parazen-----
#computes  $p(x|w_i)$ 
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  $p(x|w_i)$ 

    #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
#-----
```

Review (Bayesian classification)

we want to assign a new pattern x to one of n given classes names w_1, w_2, \dots, w_n

If $p(w_i|x)$ be the probability of class i given test sample x , bayes classifier rule is as follows:

$$p(w_i|x) > p(w_j|x) \quad \text{for all } i \neq j \rightarrow \text{choose } w_i$$

6

Function below, takes a vector consists of probabilities of classes given a test sample and Returns the index of the most probable of them.

```
#-----Bayesian Classifier-----  
#  $p(w_i|x) > p(w_j|x)$  for all  $i \neq j \rightarrow$  choose  $w_i$   
#  $p(w_i|x) = p(x|w_i) * p(w_i) / p(x)$   
# assuming equal prior ( $p(w_i) = 0.1$  for  $(1 \leq i \leq 10)$ ) then  
# bayesian classifier becomes :  $p(x|w_i) > p(x|w_j)$  for all  $i \neq j \rightarrow$  choose  $w_i$   
  
def bayes_classifier(probs):  
    return np.argmax(probs)  
  
#-----
```

7

Function below, computes accuracy criteria for classification process by taking prediction labels (y_pred) resulted from classifier and true labels (y_test)

```
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
```

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.

```
#-----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, prints accuracies resulted from classify function per different configurations

```
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 printed list is depicted below.

```
➤ Rectangular window with h = 1.5 :
   acc =  0.7963185274109644 %
Rectangular window with h = 1.6 :
   acc =  0.8303321328531412 %
Rectangular window with h = 1.7 :
   acc =  0.8127250900360145 %
Rectangular window with h = 1.8 :
   acc =  0.7607042817126851 %
Gaussian window with h = 0.1 :
   acc =  0.9247699079631853 %
Gaussian window with h = 0.5 :
   acc =  0.9223689475790317 %
Gaussian window with h = 0.6 :
   acc =  0.9147659063625451 %
Gaussian window with h = 0.7 :
   acc =  0.898359343737495 %
```

Appendix #2

Codes for KNN estimation method

1

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.

```
#-----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 o
f 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
#-----
```

2

Bayes_classifier function is just like it's counterpart in problem 1 but it repeats the operation for 3 set of probabilities corresponding to $k = 1, 2, 3$

```
#-----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  
#-----
```

3

Same as problem #1

```
#-----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  
#-----
```

4

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.

```
#-----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;
#-----
```

5

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

```
#-----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
#-----
```


Printing the results

```
#-----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,"%")  
#-----
```

```
Accuracy for k = 1:      0.9247699079631853 %  
Accuracy for k = 2:      0.9111644657863145 %  
Accuracy for k = 3:      0.9079631852741097 %
```

Appendix #3

Codes for KNN Classifier

1

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

```
#-----
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 occurrence of classes in top 3 nearest 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 occurrence
    freq_arg = np.argsort(freq[:,0]) #ascending
    freq_arg = freq_arg[::-1] #turns to descending
    y_pred[1] = freq_arg[0] #pick majority

#-----

#----- (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
#-----
```

2

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.

```
#-----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))
#-----
```

```
Accuracy for k = 1:      0.9247699079631853
Accuracy for k = 3:      0.9123649459783914
Accuracy for k = 5:      0.9067627050820328
```
