MAXIMUM LIKELIHOOD ESTMATION VS PARZEN WINDOW ESTIMATION

CS662 Project 2

Xiaobo Zhang

Section 1. Introduction

The goal of this project is to compare two density function estimation: maximum likelihood estimation and Parzen window estimation, under the size of training samples, variance of data and computation speed to determine which estimation works, which estimation not works and which is more advantage than the other. Section 2 will introduce maximum likelihood estimation and Parzen window estimation and how they are used for classification. Section 3 will experiment these two estimations with the size of training samples. Section 4 will experiment these two estimations with variance of data. Section 5 will experiment the computation speed of two estimations. Section 6 will conclude the result of this project.

Section 2. Introduction of maximum likelihood estimation and Parzen window estimation

According the Bayes decision rule:

$$Given\ x \epsilon X$$

$$find \underset{w_i \in \{w_1, w_2, \ldots w_n\}}{\operatorname{argmax}} \rho(w_i|x) Prob(w_i)$$

$$where\ \rho(w_i|x)\ is\ the\ distribution\ of\ X\ under\ the\ class\ w_i$$

, it uses probability density function $\rho(w_i|x)$ and priori probability $Prob(w_i)$ to calculate the posterior probability and determine which class the point(samples) belongs to. However, the probability density function is not given in real word, so we have to find some ways to make estimated parameters approximate to true parameters. In this project, there are two methods for estimation: parametric estimation and non-parametric estimation. Both methods will through observed data to construct an estimation.

For parametric estimation, the project uses maximum likelihood estimation to estimate the $\hat{\mu}$ and $\hat{\Sigma}$ as the parameters of density function for Bayes decision rule classification. The estimated $\hat{\mu}$ and $\hat{\Sigma}$ are:

$$Given\ trainnig\ data\ \{x_1, x_2, x_3, \ldots\ldots, x_n\} \epsilon\ X$$

$$\hat{\mu} = \frac{1}{\mathrm{N}} \sum_{x_i \in X} x_i \ \ and \quad \hat{\Sigma} = \frac{1}{\mathrm{N}} \sum_{x_i \in X} (x_i - \hat{\mu})(x_i - \hat{\mu})^T$$

Using estimated $\hat{\mu}$ and $\hat{\Sigma}$ as parameters in discrimination function g(x) for classification:

$$\begin{aligned} g(x) = \ & g_1(x) - \ g_2(x) \\ & = -\frac{1}{2}(x - \hat{\mu}_2)^T \hat{\Sigma}_2^{-1}(x - \hat{\mu}_2) - \frac{1}{2}(x - \ \hat{\mu}_1)^T \hat{\Sigma}_1^{-1}(x - \hat{\mu}_1) \\ & + \frac{1}{2}\ln\frac{|\hat{\Sigma}_2|}{|\hat{\Sigma}_1|} + \ln\frac{Prob(\widehat{w}_1)}{Prob(\widehat{w}_2)} \end{aligned}$$

It is same as project 1, choose class 1 when g(x) > 0, and choose class 2 when g(x) < 0. The only difference is changing the true parameter with estimated parameter.

For non-parametric estimation, this project uses Parzen window estimation to estimate the density of the test point (samples) p(x) for Bayes decision rule classification. Assume there is a point x_0 in the region R_n, and the estimated $\rho(x_0|w_i)$ will be:

$$\rho(x_0|w_i) = \frac{K_n}{N * V_n}$$

$where\ K_n\ is\ number\ of\ points\ in\ R_n\ belong\ to\ class\ i\ and\ V_n\ is\ the\ volumn\ of\ R_n$

The K_n is calculated by a window function φ:

$$\varphi(u) \begin{cases} 1 \quad if\ |u_1| < \frac{1}{2}\ and\ if\ |u_2| < \frac{1}{2}\ \ldots\ldots and\ |u_n| < \frac{1}{2} \\ \qquad\qquad\qquad 0\ else \end{cases}$$

If the window function is hypercube R^n, then K_n and V_n will be:

$$K_n = \sum_{x_i \epsilon \; all \; training \; data} \varphi(\frac{x_i - x_0}{h})$$

$$V_n = (h)^n$$

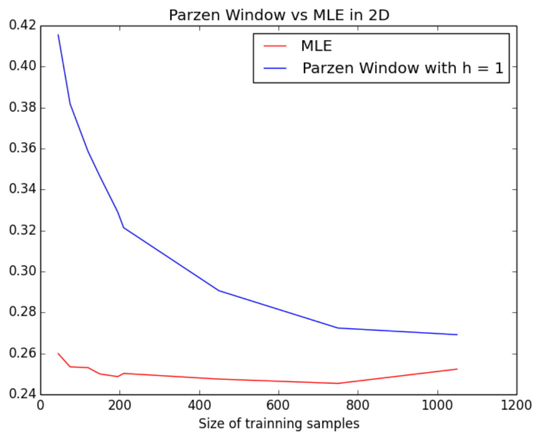
Using estimated density function $\rho(x_0|w_i)$ and prior probability $Prob(w_i)$ as parameters for Bayes decision classification.

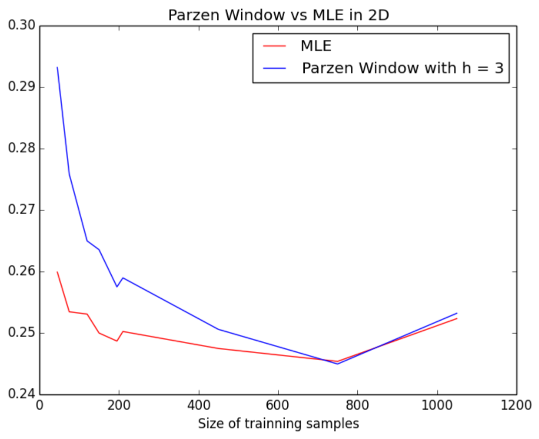
Section 3. Size of training samples

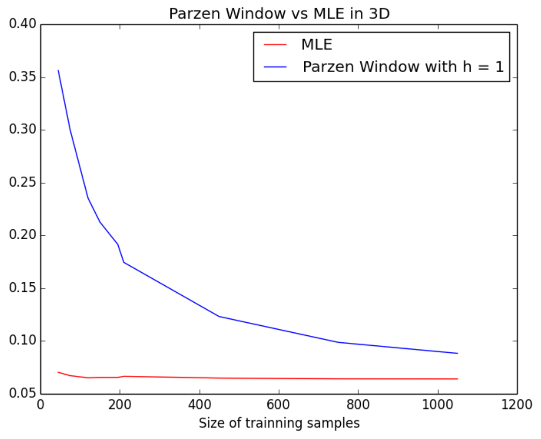
This experiment has several assumptions:

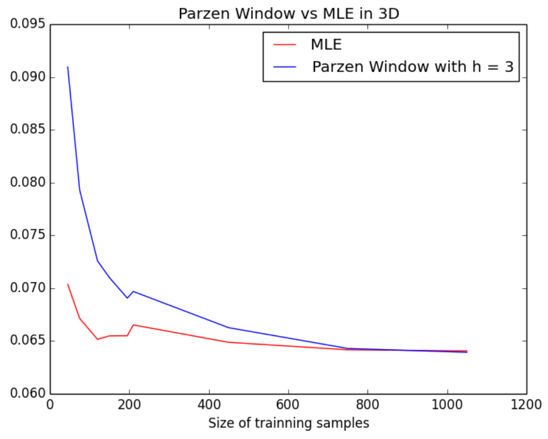
1. The data is multivariate Gaussian distribution.
2. The size of data is 3000.
3. $\Sigma_1 = \Sigma_2$.
4. The mean distance of two class is 3.
5. The variance of each dimension is less than 10.
6. The priori probability P(w1) = P(w2) = 0.5

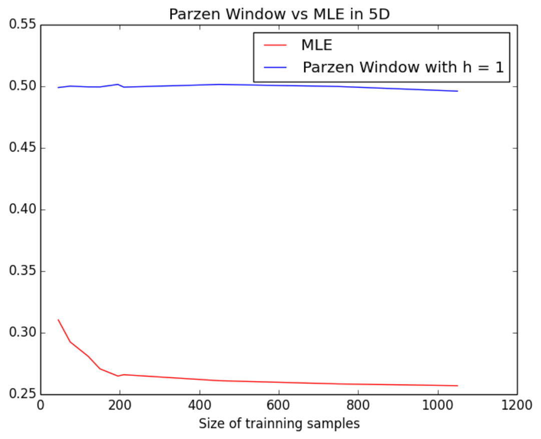
The experiment will compare the error rate for each method by different size of training samples and different dimension.

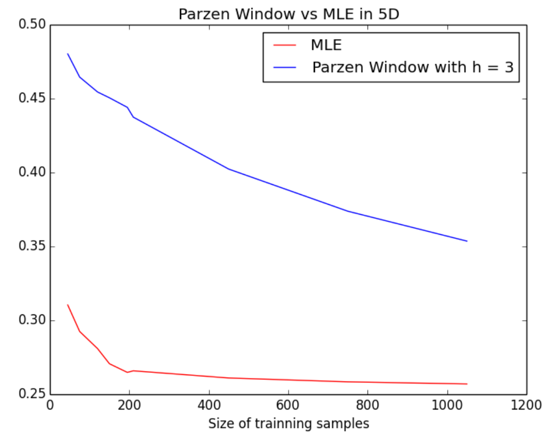


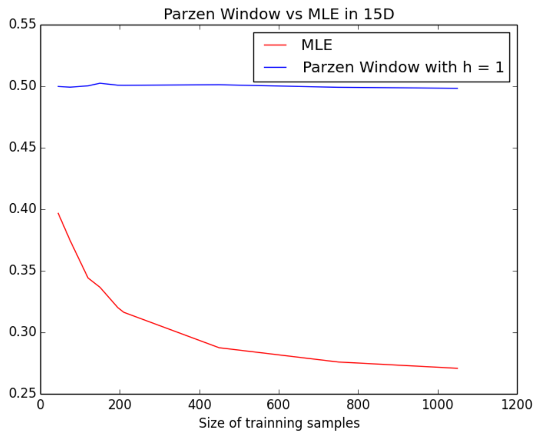


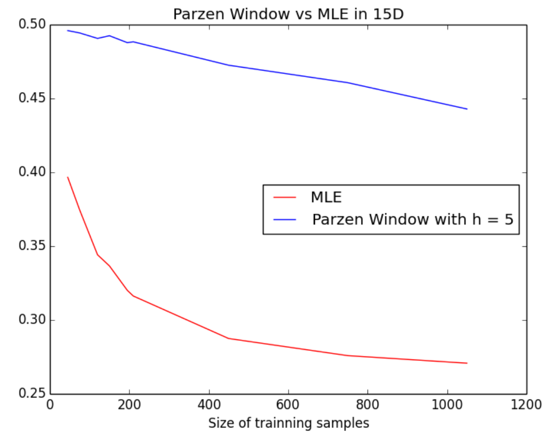












Given above graph, the maximum likelihood estimation has lower error rate than Parzen window estimation when the size of training samples is small. With the size of training samples increasing, the error rate of Parzen window estimation decrease faster than maximum likelihood estimation. It is because more and more training samples in the space makes the estimated density approximate to true density. When the size of training samples is really large, the error rate of two estimation methods will converge at some point.

There is an interesting thing is when the dimension increase, the error rate of Parzen window estimation always stays at 0.5 given the height of window h = 1. It perhaps the volume of the window is too small to hold enough points. If increasing the window size, the error rate will decrease with the size of training samples increase as usual. Choosing the right window size will influence the error rate a lot.

Section 4. Variance of data

For this experiment, there are several assumptions:

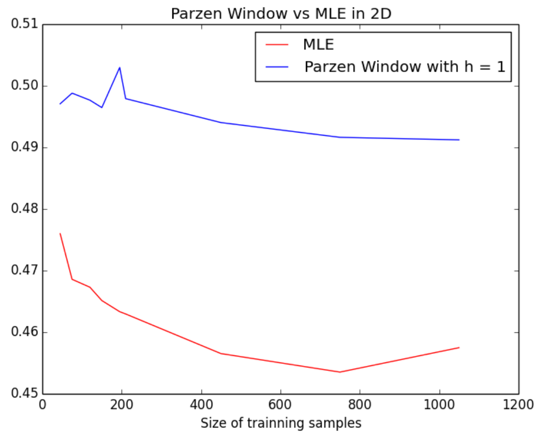
1. The data is multivariate Gaussian distribution.
2. The size of data is 3000.
3. $\Sigma_1 = \Sigma_2$.
4. The mean distance of two class is 3.
5. The priori probability P(w1) = P(w2) = 0.5

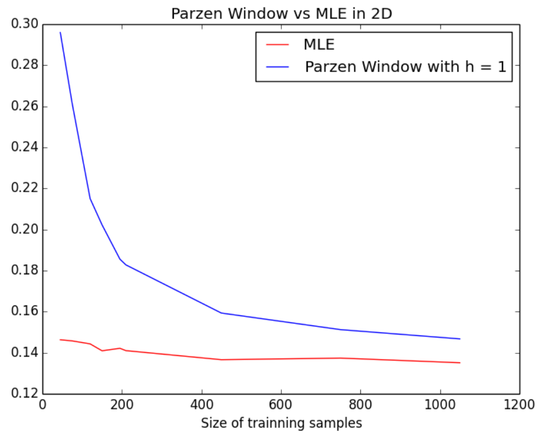
The experiment will compare if estimation methods work with different variance of data in 2D, 5D, 10D, 15D.

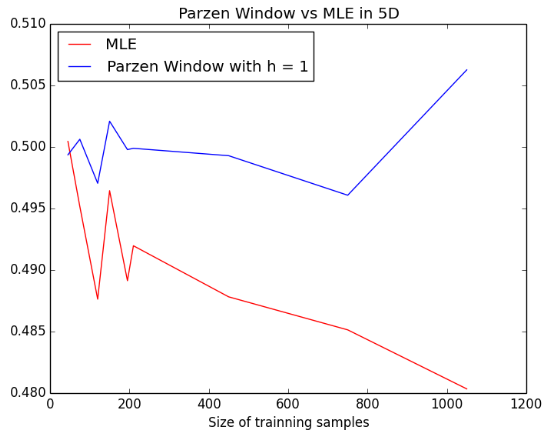
Case 1: variance between 100 and 1000 on each dimension(diagonal)

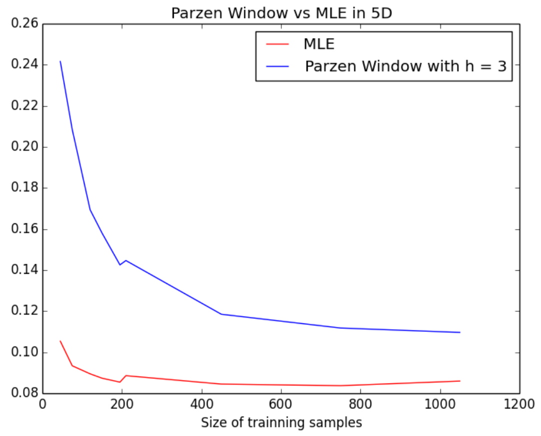
Case 2: variance between 0 and 5 on each dimension(diagonal)

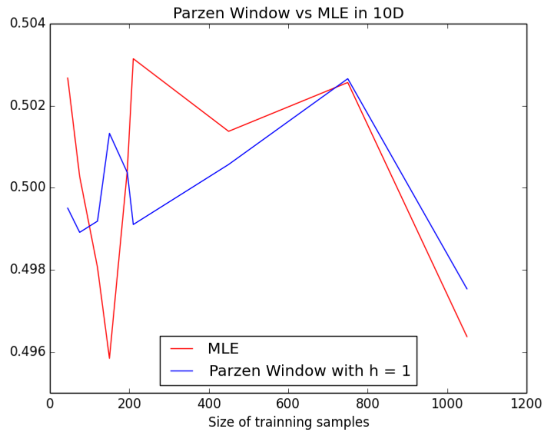
Case 1 Case 2

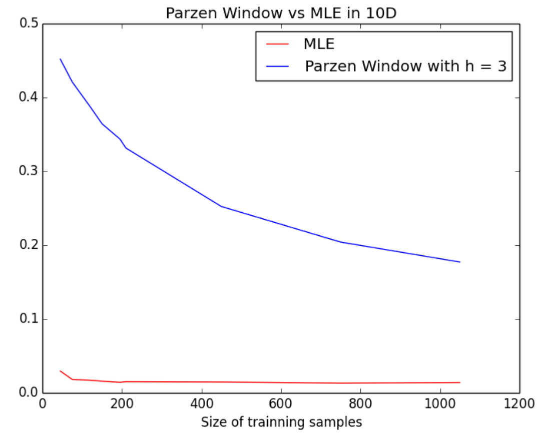

Parzen Window vs MLE in 2D
MLE
Parzen Window with h = 1
Size of trainning samples
0.45
0.46
0.47
0.48
0.49
0.50
0.51
0
200
400
600
800
1000
1200

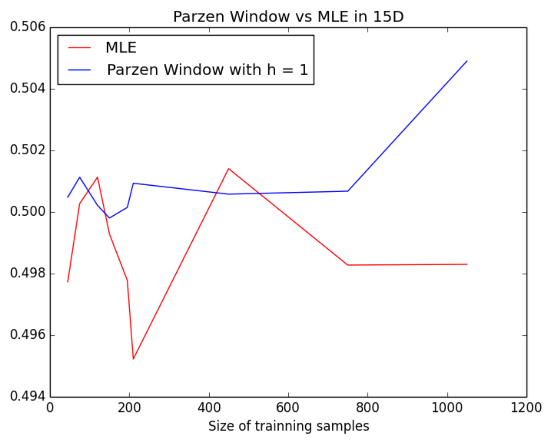

Parzen Window vs MLE in 2D
MLE
Parzen Window with h = 1
Size of trainning samples
0.12
0.14
0.16
0.18
0.20
0.22
0.24
0.26
0.28
0.30
0
200
400
600
800
1000
1200

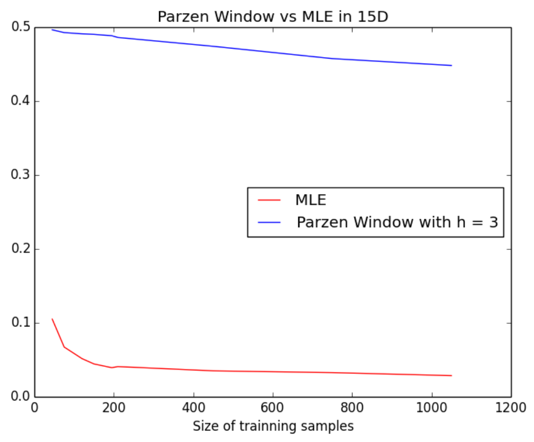

Parzen Window vs MLE in 5D
MLE
Parzen Window with h = 1
Size of trainning samples
0.480
0.485
0.490
0.495
0.500
0.505
0.510
0
200
400
600
800
1000
1200


Parzen Window vs MLE in 5D
MLE
Parzen Window with h = 3
Size of trainning samples
0.08
0.10
0.12
0.14
0.16
0.18
0.20
0.22
0.24
0.26
0
200
400
600
800
1000
1200


Parzen Window vs MLE in 10D
MLE
Parzen Window with h = 1
Size of trainning samples
0.496
0.498
0.500
0.502
0.504
0
200
400
600
800
1000
1200


Parzen Window vs MLE in 10D
MLE
Parzen Window with h = 3
Size of trainning samples
0.0
0.1
0.2
0.3
0.4
0.5
0
200
400
600
800
1000
1200





When the variance of data is large, the error rate of both estimation methods is very high and doesn't decrease a lot with the size of training samples increasing. For maximum likelihood estimation, it needs more training samples to make estimated parameters $\hat{\mu}$ and $\hat{\Sigma}$ close to true parameters. For Parzen window estimation, the window needs more points in the volume to let estimated density function close to true density function.

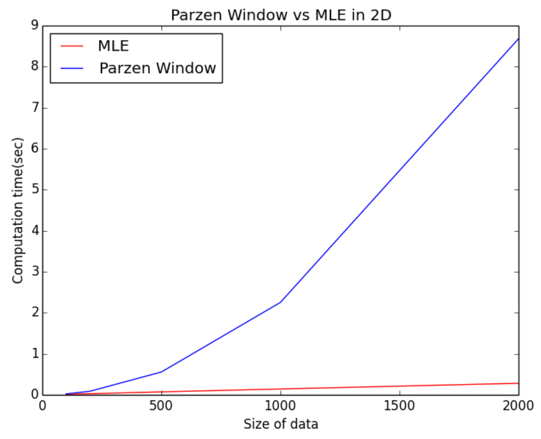
However, the error rate of both methods is much smaller when the variance of data is small. It probably because small variance data are more concrete, and the estimator can tell the entire distribution of the data without too many training samples and output parameters approximate to true parameters. The small variance data works for both estimation methods and it performs better on maximum likelihood estimation than Parzen window estimation when the size of training samples is small. The error rate of Parzen window estimation is not as low as the maximum likelihood estimation when the size of training samples is small, but the error rate decrease when the size of training samples increasing. It is because there are enough points (training samples) to tell entire distribution of data.

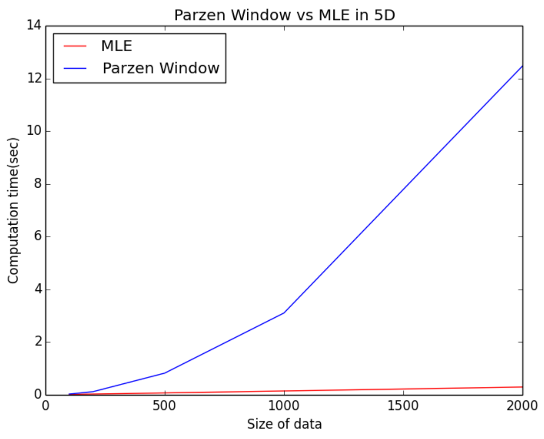
Section 5. Computation speed

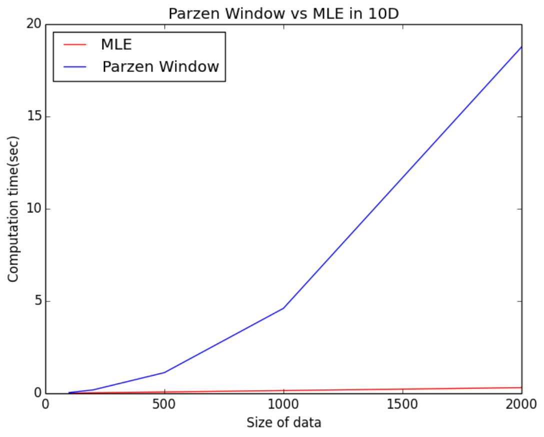
In this experiment, it has several assumptions:

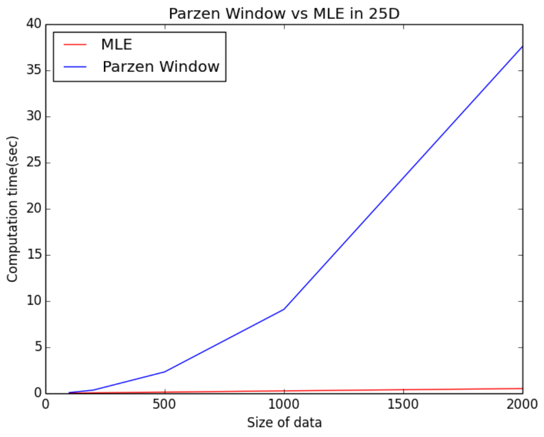
1. The data is multivariate Gaussian distribution.
2. The size of data is 3000.
3. The probability of training samples is 0.5 of entire data.
4. $\Sigma_1 = \Sigma_2$.
5. The mean distance between 2 class is 6.
6. The priori probability P(w1) = P(w2) = 0.5

The experiment will compare the computation time of each method in 2D, 5D, 10D and 25D.

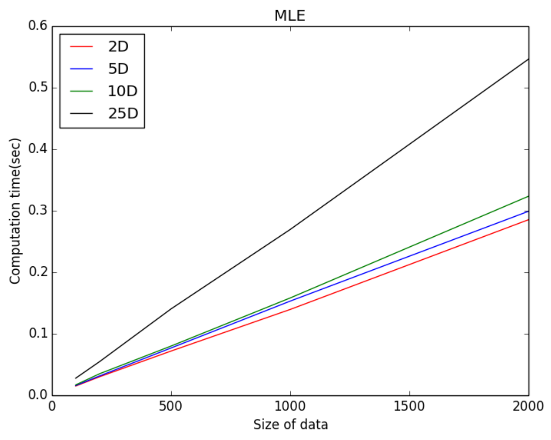


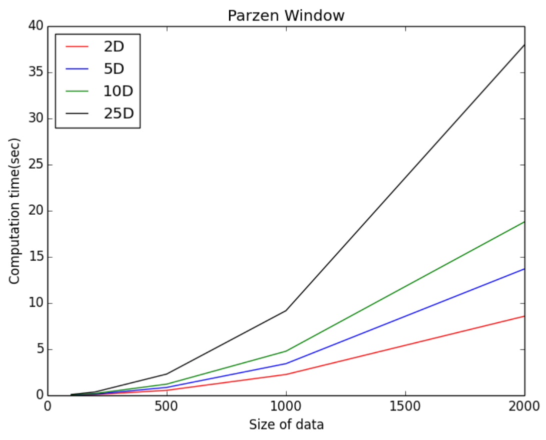






In each dimension, we can find that the computation cost of Parzen window estimation is higher than maximum likelihood estimation obviously. It is because each test point(sample) has to compute distance with all the training samples to get the density p(x). The computation cost will grow as the size of training samples increasing, but maximum likelihood estimation just uses estimated parameters $\hat{\mu}$ and $\hat{\Sigma}$, and discrimination function directly for classification. The size of training samples doesn't have strong influence to maximum likelihood estimation on computation cost, but the Parzen window estimation does.





For each estimation method, we can find that the computation time will increase with the size of data increasing. In addition, the computation cost of each estimation method will increase with dimension of the data increasing. According the graph, maximum likelihood estimation method grows linearly with the size of data increase, but for Parzen window estimation, it grows exponentially. So this experiment can conclude that the size of data highly influences the computation speed, especially for Parzen window estimation.

Section 6. Conclusion

This project experimented performance of two estimation methods: maximum likelihood estimation and Parzen window estimation, through the size of training samples, variance of data and computation speed. We can found that both estimation methods work good with small variance data, not good for large variance data. The maximum likelihood estimation works better than Parzen window estimation when the size of training samples is small, because little training samples cannot help Parzen window estimation figure out the distribution of data very well. Parzen window estimation will works well when distribution of data is clear, which means there are many training samples on the space. The maximum likelihood estimation is much faster than Parzen window estimation on computation speed. It is because Parzen window needs to scan all the training samples for just one test samples, but maximum likelihood estimation doesn't have to scan all training samples. Parzen window estimation needs a right window size h to make the classification good, the size of h shouldn't be too small, and shouldn't be too large either.

Reference:

1. Professor's lecture notes

2. Kernel density estimation via the Parzen-Rosenblatt window method
http://sebastianraschka.com/Articles/2014_kernel_density_est.html

mle.py

```python
import numpy as np

def estimate_mu(data):
    data =  np.array(data)
    return np.mean(data,axis=0)

def estimate_covariance(data):
    data =  np.array(data)
    return np.cov(data.T)
```

parzen.py

```python
#parzen.py
#function of parzen window
import math
import numpy as np

def window_function(sample,test_data,h):
    #print sample
    #print "x",test_data
    for x,x_i in zip(sample,test_data):

            if abs(x-x_i) > float(h)/2:
                return 0
        return 1

    def estimate_single_px(samples,test_data,h):
        inRegion = 0
        outRegion = 0

        for sample in samples:
            if window_function(sample,test_data,h) == 1:
                inRegion  += 1
            else:
                outRegion += 1

        return float(inRegion)/((inRegion+outRegion)*(h**len(test_data)))

    def get_px(samples,test_datas,h):
        px =[]

        for x in test_datas:
            x =np.array(x)

            val = estimate_single_px(samples,x,h)
            px.append(val)
        return px
```

plotting.py

```
import matplotlib.pyplot as plt

def plot(title, parzen, mle,xtitle,ytitle,x_attr,h):
    plt.clf()
    plt.title(title)
    plt.xlabel(xtitle)
    plt.plot(x_attr,mle,label="MLE",color = 'r')
    plt.plot(x_attr,parzen,label="Parzen Window with "+str(h),color = 'b')
    plt.legend(loc='best')
    #plt.ylim(0,0.8)
    plt.show()
```

testing_mle.py

```
import mle
import numpy as np
import random
import parzen
from numpy import matrix
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt
from matplotlib.mlab import bivariate_normal
import math
from numpy.linalg import linalg
import time


def compute_gx(mu_1,mu_2,cov_1,cov_2,w1,w2,x):

    mu_1 =matrix(mu_1)
    mu_2 = matrix(mu_2)
    cov_1 = matrix(cov_1)
    cov_2 = matrix(cov_2)


```

```
        const = 0.5 * math.log(linalg.det(cov_2)/linalg.det(cov_1)) + math.log(float(w1)/w2)

        #print "const",const
        #print "xx",(0.5* (x-mu_2) * cov_2.I * (x-mu_2).T ) - (0.5* (x-mu_1) * cov_1.I * (x-mu_1).T )
        gx = (0.5 * (x-mu_2) * cov_2.I * (x-mu_2).T ) - (0.5* (x-mu_1) * cov_1.I * (x-mu_1).T )
        #print gx
        v = gx.tolist()
        return v[0][0]+const



    def testing(train_samples_1,test_samples_1,train_samples_2,test_samples_2):

        now = time.time()


        estimate_mu_1 = mle.estimate_mu(train_samples_1)
        estimate_cov_1 = mle.estimate_covariance(train_samples_1)

        estimate_mu_2 = mle.estimate_mu(train_samples_2)
        estimate_cov_2 = mle.estimate_covariance(train_samples_2)

        w1 = len(train_samples_1)/float(len(train_samples_1)+len(train_samples_2))
        w2 = len(train_samples_2)/float(len(train_samples_1)+len(train_samples_2))

        error = 0
        for d in test_samples_1:
            val = compute_gx(estimate_mu_1 ,estimate_mu_2,estimate_cov_1,estimate_cov_2,w1,w2,d)
            if val > 0:
                pass
            else:
                error += 1

        for d in test_samples_2:
            val = compute_gx(estimate_mu_1 ,estimate_mu_2,estimate_cov_1,estimate_cov_2,w1,w2,d)
            if val > 0:
```

```python
                error += 1
            else:
                pass

        res =  float(error)/float(len(test_samples_1)+len(test_samples_2))

        return res
        #errors.append(res)
        #times.append(time.time()-now)

        #print errors
        #print "error rate:",sum(errors)/len(errors)
        #print "average_time:",sum(times)/len(times)
        #return sum(errors)/len(errors)
        #return sum(times)/len(times)
```

testing_parzen.py

```python
import numpy as np
import random
import parzen
import time
from numpy import matrix


def classify(x,train_samples_1,train_samples_2,w1,w2,h):
    px_1 = parzen.estimate_single_px(train_samples_1,x,h)
        px_2 = parzen.estimate_single_px(train_samples_2,x,h)
        if px_1*w1 > px_2*w2:
            return 1
        else:
            return 2


    def testing(train_samples_1,test_samples_1,train_samples_2,test_samples_2,h):
        errors = []
        times = []

        now = time.time()

        w1 = len(train_samples_1)/float(len(train_samples_1)+len(train_samples_2))
        w2 = len(train_samples_2)/float(len(train_samples_1)+len(train_samples_2))
        error = 0
    for x in test_samples_1:
        label = classify(x,train_samples_1,train_samples_2,w1,w2,h)
        if label == 1:
            pass
        else:
            error +=1

    for x in test_samples_2:
        label = classify(x,train_samples_1,train_samples_2,w1,w2,h)
        if label == 2:
            pass
        else:
            error +=1


    res = error/float(len(test_samples_1)+len(test_samples_2))
    return res
    #errors.append(res)
    #times.append(time.time()-now)

    #print errors
    #print "error rate:",sum(errors)/len(errors)
    #print "average_time:",sum(times)/len(times)
    #return sum(errors)/len(errors)
    #return sum(times)/len(times)
```

test.py

```
import testing_mle
import testing_parzen
import plotting
from matplotlib.mlab import bivariate_normal
import matplotlib.pyplot as plt
import random
from numpy import matrix
import numpy as np

def create_variance_matrix(d):
    cov = []
    for i in range(d):
        vec = []
        for j in range(d):
            if i == j:
                vec.append(random.randint(1, 10))
                #vec.append(1)
            else:
                vec.append(0)
        cov.append(vec)

    for i in range(d):
        for j in range(i+1,d,1):
            if i != j:
                #cov[i][j] = random.randint(1, 3)
                cov[i][j] = round(random.random(),2)
                cov[j][i] = cov[i][j]
    return matrix(cov)

def generate_random_point(dimesion,distance):
    p1 = []
    p2 = []
    for t in range(dimesion):
        x = random.random()*10-10
        p1.append(x)
        p2.append(x)

    l = len(p1)
    i = int(random.random()*l)
    p2[i] += distance
    return p1,p2

def generate_normal_data(mean,cov,size):
    data = np.random.multivariate_normal(mean, cov, size)
    return data

def generate_possion_data(size,dimension):
    data = np.random.poisson(10,size=(size,dimension))
    return data

def generate_uniform_data(start,end,size,dimesion):
    data = np.random.uniform(start,end,size=(size,dimesion))

    return data

def sampling(data_1,data_2,prob):
    train_1 = []
    test_1 = []
    train_2 = []
    test_2 = []
    for d in data_1:
        if random.random() < prob:
```

```python
            train_1.append(d)
        else:
            test_1.append(d)

    for d in data_2:
        if random.random() < prob:
            train_2.append(d)
        else:
            test_2.append(d)
    return train_1,test_1,train_2,test_2


#testing methods by computation time
dimensions = [2,5,10,25]
sizes = [100,200,500,1000,2000]

for d in dimensions:
    parzen = []
    mle = []

    for size in sizes:
        print d,size
        parzen.append(testing_parzen.testing(d,size,0.5,1))

        mle.append(testing_mle.testing(d,size,0.5))
    print "d"
    title = "Parzen Window vs MLE in "+str(d)+"D"
    title = str(title);
    xtitle = "Size of data"
    ytitle = "Computation time(sec)"
    x_attr = sizes

    plotting.plot(title, parzen, mle,xtitle,ytitle,x_attr)


colors = ['r', 'b', 'g', 'k', 'm']
i = 0
for d in dimensions:
    parzen = []
    mle = []

    for size in sizes:
        print d,size
        parzen.append(testing_parzen.testing(d,size,0.5,1))

        #mle.append(testing_mle.testing(d,size,0.5))
    plt.plot(sizes,parzen,label=str(d)+"D",color=colors[i])

    #plt.plot(sizes,mle,label=str(d)+"D",color=colors[i])

    i+= 1

title = "Parzen Window"
#title = "MLE"
plt.title(title)
xtitle = "Size of data"
ytitle = "Computation time(sec)"
plt.xlabel(xtitle)
plt.ylabel(ytitle)
plt.legend(loc='best')

plt.show()



#testing by trainning sampling size and variance of data
dimensions = [10,15,25]
size = 1500
probs = [0.03,0.05,0.08,0.1,0.13,0.14,0.3,0.5,0.7]
for d in dimensions:

    #generate parameter
    mean_1,mean_2 = generate_random_point(d,3)
    cov_1 = create_variance_matrix(d)
    cov_2 = cov_1

    #generate data
    data_1 = generate_normal_data(mean_1,cov_1,size)
    #data_1 = generate_uniform_data(-10,10,size,d)
    data_2 = generate_normal_data(mean_2,cov_2,size)
    #data_2 = generate_uniform_data(-10,10,size,d)

    parzen_1 = []
    parzen_3 = []

    mle = []

    for prob in probs:
        print d,size*prob
        parzen_errors_1 = []
        parzen_errors_3 = []

        mle_errors = []
        for t in range(10):
            train_samples_1,test_samples_1,train_samples_2,test_samples_2 = sampling(data_1,data_2,prob)
            parzen_errors_1.append(testing_parzen.testing(train_samples_1,test_samples_1,train_samples_2,test_samples_2,1))
            parzen_errors_3.append(testing_parzen.testing(train_samples_1,test_samples_1,train_samples_2,test_samples_2,3))
            mle_errors.append(testing_mle.testing(train_samples_1,test_samples_1,train_samples_2,test_samples_2))
        parzen_1.append(sum(parzen_errors_1)/len(parzen_errors_1))
        parzen_3.append(sum(parzen_errors_3)/len(parzen_errors_3))

        mle.append(sum(mle_errors)/len(mle_errors))
    print parzen_1
    print parzen_3
    print mle

    title = "Parzen Window vs MLE in "+str(d)+"D"
    title = str(title);
    xtitle = "Size of trainning samples"
    ytitle = "error rate"
    x_attr = [x*size for x in probs]

    plotting.plot(title, parzen_1, mle,xtitle,ytitle,x_attr,"h = 1")
    plotting.plot(title, parzen_3, mle,xtitle,ytitle,x_attr,"h = 3")
```