BAYES DECISION RULE

CS 662 Project 1

Xiaobo Zhang

Purdue University | CS 662

Bayes decision rule for classifying normal distribution data

Section 1. Introduction

This project will use the Bayes decision rule to experiment when Bayes decision rule is good at classify and when is not. The experiment will see the classification results under the effect of difference between the mean distance, sample size, feature vector dimension. Section 2 will introduce Bayes decision rule. Section 3 will experiment the Bayes decision rule with mean distance and vector dimension. Section 4 will experiment Bayes decision rule with sample size and vector dimension. Section 5 will experiment Bayes decision rule with space dimension. Section 6 will be the conclusion of the project.

Section 2. Introduction of Bayes decision rule

The decision rule is to find the most likely class w_i given the observation random variable x:

$$\underset{w_i \in \{w_1, w_2, \dots. w_n\}}{\text{argmax}} \quad Prob(w_i|x)$$

Since the $Prob(w_i|x)$ is posterior probability and hard to estimate, so according Bayes rule we transform $Prob(w_i|x)$ into:

$$Prob(w_i|x) = Prob(w_i|x)Prob(w_i)$$

the parameters $Prob(x|w_i)$ and $Prob(w_i)$ are able to calculate given enough data.

Therefore, decision rule for discrete feature vector will become:

$$Given\ x \epsilon X$$

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

for continuous feature vector the decision rule will become:

$$Given\ x \in X$$

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

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

In the next three sections, the experiment will assume the feature vector is continuous and the distribution of $\rho(w_i|x)$ is normally distributed. So

$$\rho(w_i|x) = \frac{1}{(2\pi)^{\frac{n}{2}}} \frac{1}{|\Sigma_i|^{\frac{1}{2}}} \exp\left(-\frac{1}{2}(x-\mu_i)^T \Sigma_i^{-1}(x-\mu_i)\right)$$

$$Prob(w_i|x) = \rho(w_i|x) * Prob(w_i)$$

$$= \frac{1}{(2\pi)^{\frac{n}{2}}} \frac{1}{|\Sigma_i|^{\frac{1}{2}}} \exp\left(-\frac{1}{2}(x-\mu_i)^T \Sigma_i^{-1}(x-\mu_i)\right) * Prob(w_i)$$

In order to make the computation easier, add ln at two side

$$g_i(x) = \ln Prob(w_i|x) = \ln \rho(w_i|x) * Prob(w_i)$$

$$g_i(x) = -\frac{1}{2}(x-\mu_i)^T \Sigma_i^{-1}(x-\mu_i) + \ln \frac{1}{(2\pi)^{\frac{n}{2}} |\Sigma_i|^{\frac{1}{2}}} + \ln Prob(w_i)$$

Then the most likely class w_i will be selected if:

$$\underset{i}{\operatorname{argmax}}\, g_i(x)$$

Foe two classes:

$$g(x) = g_1(x) - g_2(x)$$

$$= -\frac{1}{2}(x-\mu_2)^T \Sigma_2^{-1}(x-\mu_2) - \frac{1}{2}(x-\mu_1)^T \Sigma_1^{-1}(x-\mu_1)$$

$$+ \frac{1}{2} \ln \frac{|\Sigma_2|}{|\Sigma_1|} + \ln \frac{Prob(w_1)}{Prob(w_2)}$$

$g(x)$is discriminate function to classify w_1 and w_2. If $g(x) > 0$, then select w_1, $g(x) < 0$, then select w_2, and $g(x) = 0$, it means the point is on the boundary, selecting both w_1 and w_2 doesn't matter.

For special case $\Sigma_1 = \Sigma_2 = \Sigma$:

$$g(x) = g_1(x) - g_2(x) = \ldots$$

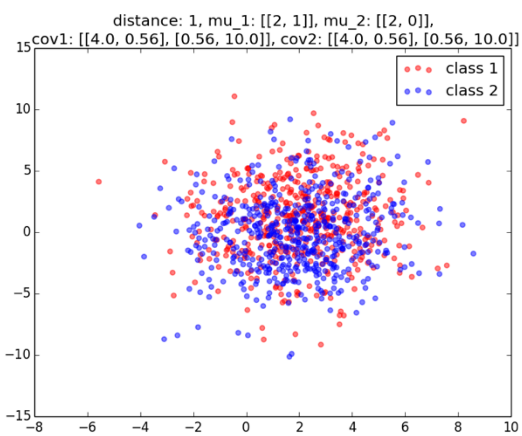
$$= (\mu_1 - \mu_2)^T \Sigma^{-1}(x) + \frac{\mu_2^T \Sigma^{-1} \mu_2 - \mu_1^t \Sigma^{-1} \mu_1}{2} + \ln \frac{Prob(w_1)}{Prob(w_2)}$$

The discriminate function is the threshold for this project to determine if the sample points belong to which class. It is very important in next three sections.

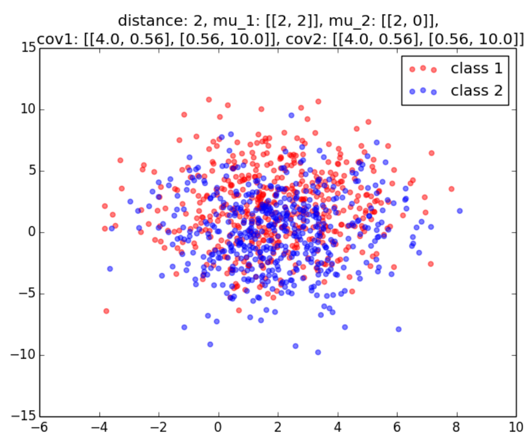
Section 3. Bayes decision rule with the mean distance

Before the experiment, there are two assumptions. First, the data is generated synthetically, and second the parameters, such as prior probability(p(w1), p(w2)), mean(μ_1, μ_2), and the covariance(Σ_1, Σ_2) are already known at the beginning of the experiment.

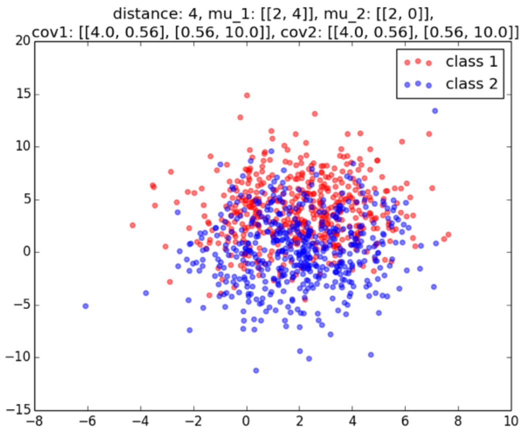
Case 1. Dimension = 2, $\Sigma_1 = \Sigma_2$, sample size = 1000 and p(w1) = p(w2)



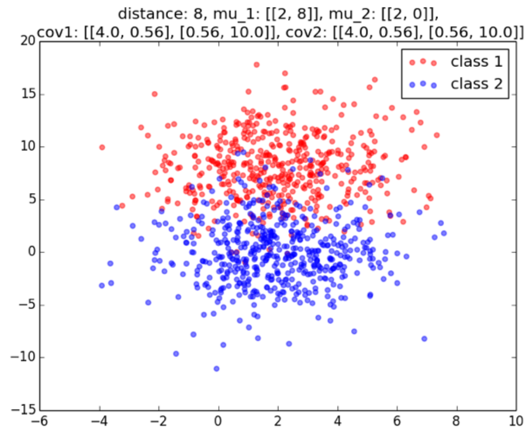
distance = 1



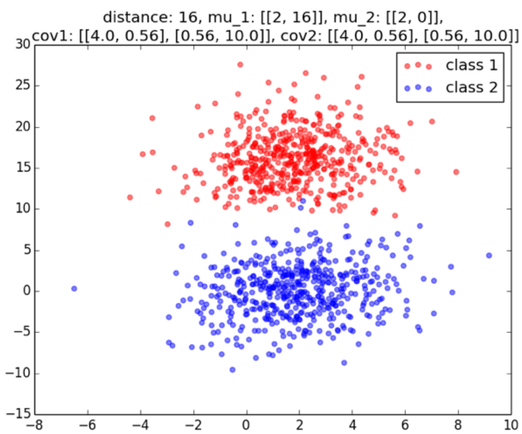
distance = 2



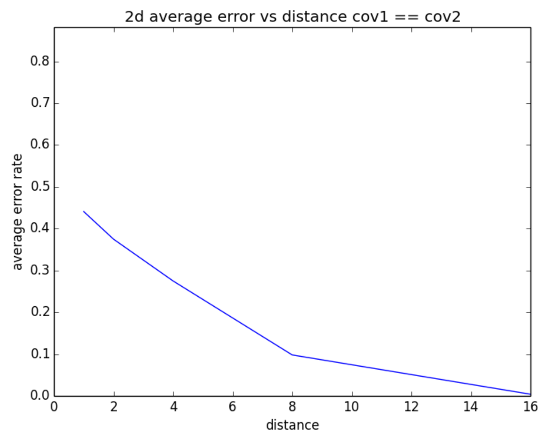
distance = 4



distance = 8



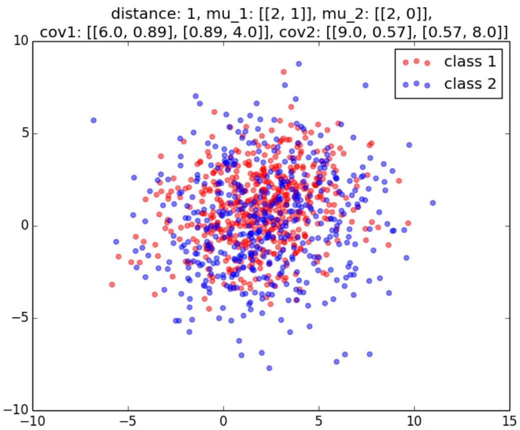
distance = 16



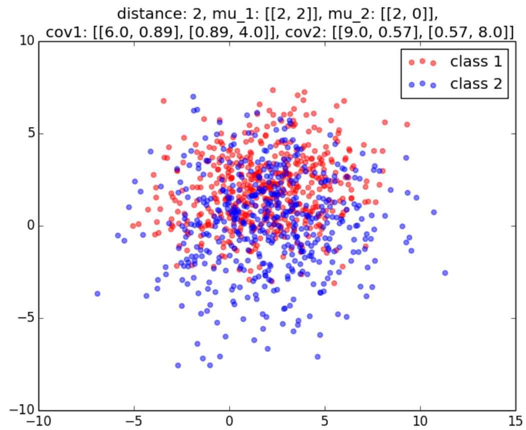
error rate vs distance

In two dimension with same covariance, the error rate of Bayes decision rule decrease when the distance of two mean point increase. Also, according the distribution, the group of two class will be more and more clear with increase of mean distance.

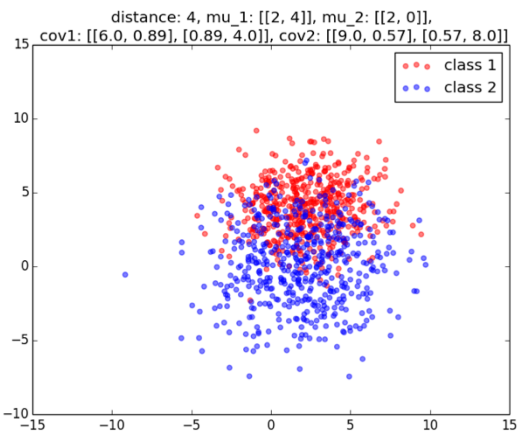
Case 2. Dimension = 2, $\Sigma_1 \neq \Sigma_2$, sample size = 1000 and p(w1) = p(w2)



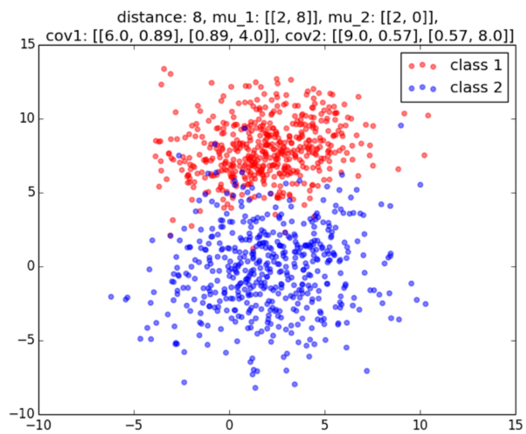
distance = 1



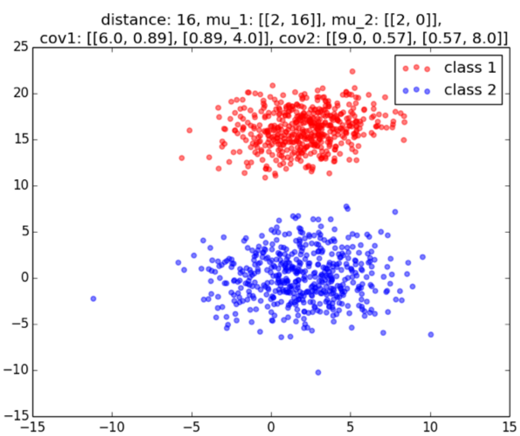
distance = 2



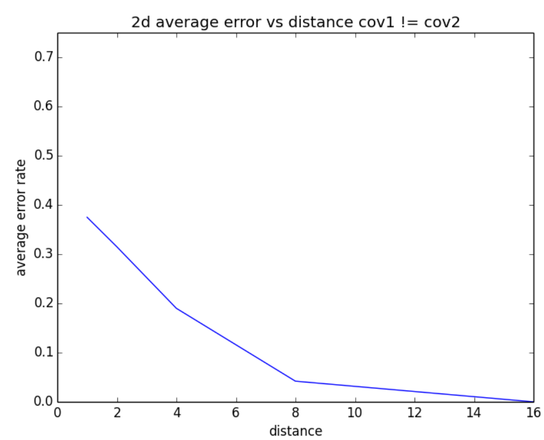
distance = 4



distance = 8



distance = 16

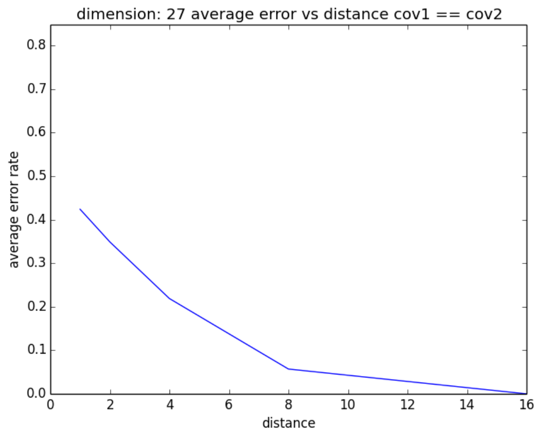


error rate vs distance

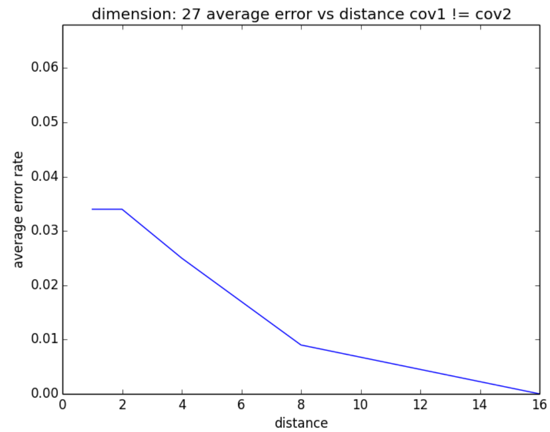
In two dimension with different covariance, the error rate of Bayes decision rule is also decrease when the distance of two mean points increase. The group of two class will become clear as case 1 more and more clear with increase of mean distance.

Case 3. Dimension = 25, $\Sigma_1 = \Sigma_2$, sample size = 1000 and p(w1) = p(w2)

Case 4. Dimension = 25, $\Sigma_1 \neq \Sigma_2$, sample size = 1000 and p(w1) = p(w2)



case 3



case 4

For high dimension feature vector at both case 3 and case 4, the error rate of Bayes decision rule still decreases with the increase of mean distance.

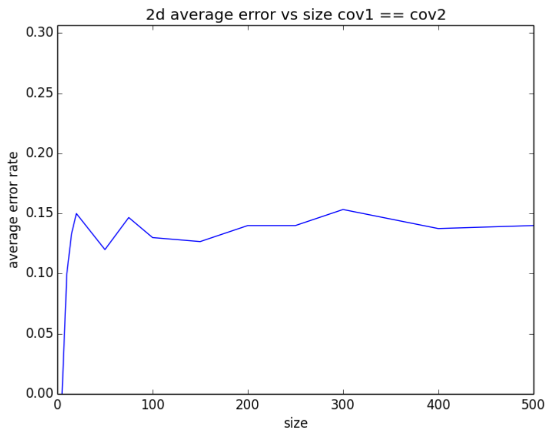
Given the all four cases, it easily finds that no matter what the dimension of the feature vector is, the mean distance of two class increase, the error rate of Bayes decision rule will decrease and goes to 0 at the end. It probably because when the mean distance of two class is small, the two class will very likely goes to the other class or over the boundary (g(x) =0), but when the distance becomes larger, the variable of two class is far away from the boundary (g(x) = 0), and the error rate decrease. So Bayes decision rules is pretty good at classifying data when the mean distance of two class is separated far away in any dimension.

Section 4. Bayes decision with the number of points classified

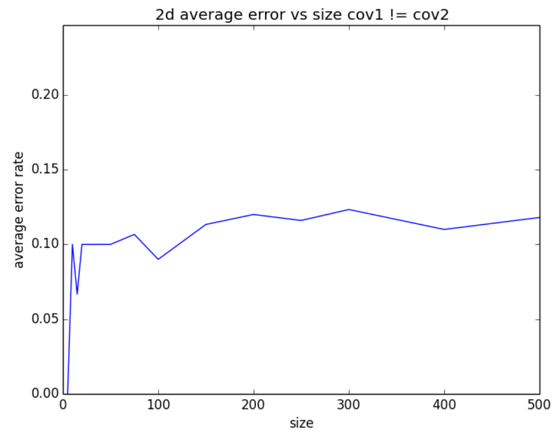
Assume the mean distance = 4, Σ is constructed randomly

Case 1. Dimension = 2, $\Sigma_1 = \Sigma_2$, and p(w1) = p(w2)

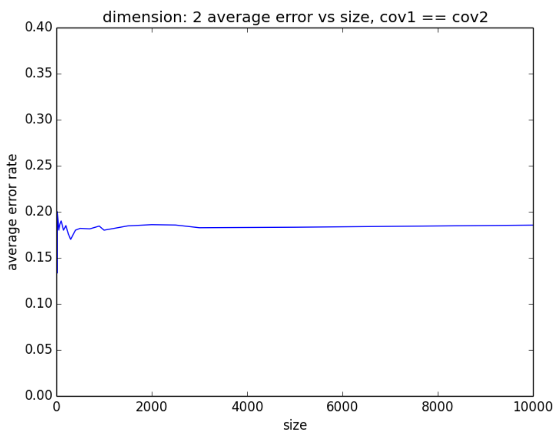
Case 2. Dimension = 2, $\Sigma_1 \neq \Sigma_2$, and p(w1) = p(w2)



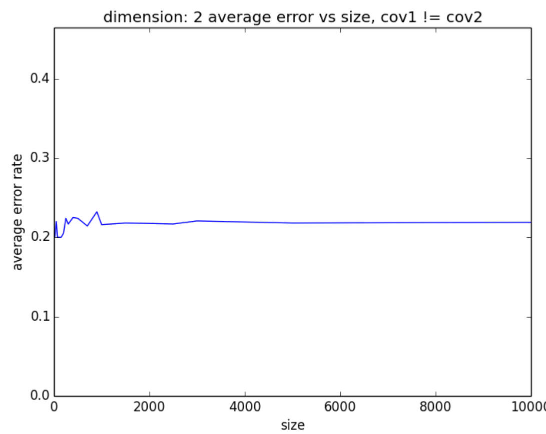
case 1 (small sample size)



case 2 (small sample size)



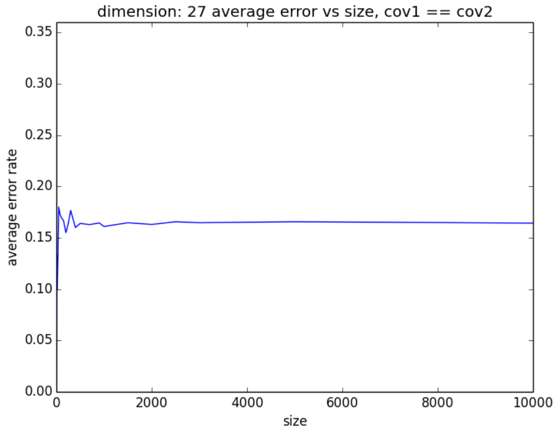
case 1(large sample size)



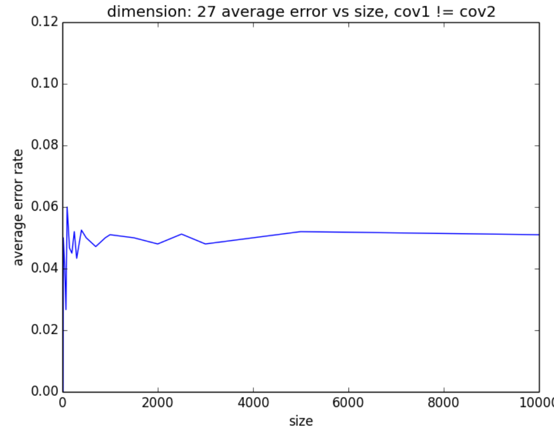
case 2(large sample size)

Case 3. Dimension = 27, $\Sigma_1 = \Sigma_2$, and p(w1) = p(w2)

Case 4. Dimension = 27, $\Sigma_1 \neq \Sigma_2$, and p(w1) = p(w2)



case 3



case 4

From the all above cases, the error rate of Bayes decision rule will go stable as the sample size increase. However, when the sample size is very small, the error rate of Bayes decision rule is not very stable. In above four cases, the average error rate will be close to some value after the size greater than 2000. It is because small sample size is not able to fully represent the entire distribution of the data of the class, so the error rate will not true at the beginning, but the error rate be stable when the size increase. So the Bayes decision rule works good when the sample size is large, and the result will stay in some range mostly under 0.5.

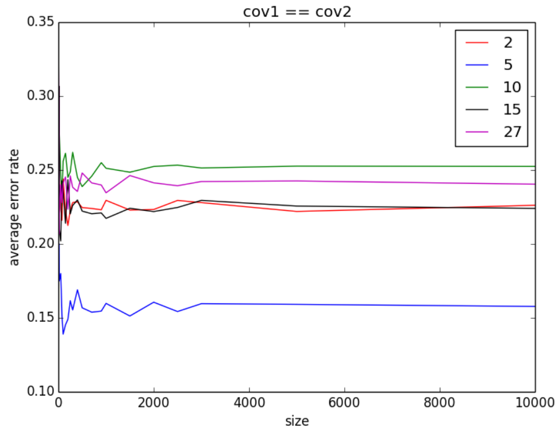
Section 5. Bayes decision rule with space dimension

Case 1. Σ is diagonal matrix, $\Sigma_1 = \Sigma_2$, dimension = 2,5,10,15,27, p(w1) = p(w2)

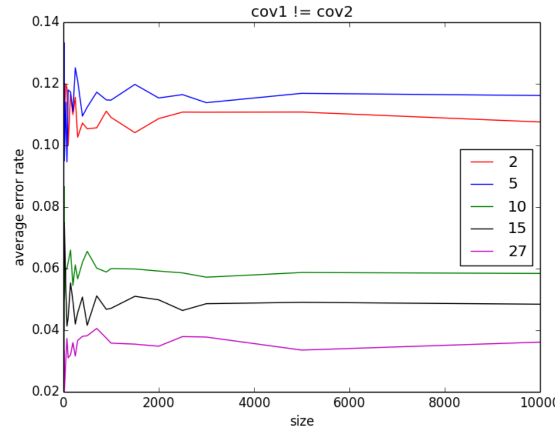
Case 2. Σ is diagonal matrix, $\Sigma_1 \neq \Sigma_2$, dimension = 2,5,10,15,27, p(w1) = p(w2)

Case 3. Σ is not diagonal matrix, $\Sigma_1 = \Sigma_2$, dimension = 2,5,10,15,27, p(w1) = p(w2)

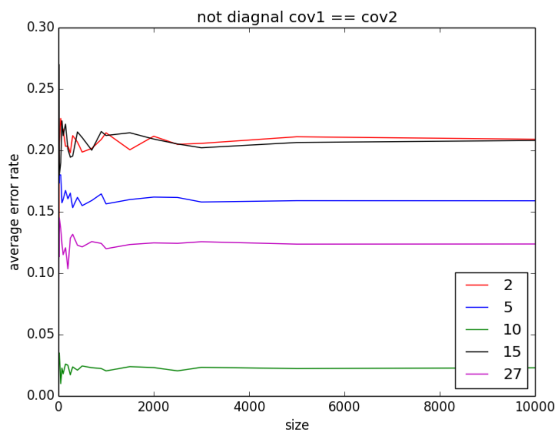
Case 4. Σ is not diagonal matrix, $\Sigma_1 \neq \Sigma_2$, dimension = 2,5,10,15,27, p(w1) = p(w2)



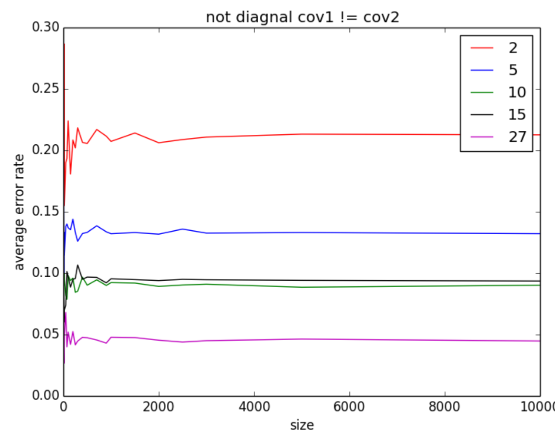
case 1



case 2



case 3



case 4

Based on observation, the error rate has no direct relation with space dimension. According the above chart, some low dimension's error rate is smaller than high dimension error rate, but some are not. For example, the error rate of dimension 10 is smaller than dimension 27 in casa 3. It is mostly because of construction of the covariance matrix is random, which will probably make the sample points are concentrated together in some dimension and hard to classify, so the error rate will be vary. If the covariance matrix doesn't have such condition, the error rate will be smaller, and vice versa.

Section 5. Conclusion

After the experiment of Bayes decision rule with mean distance, sample size, and space dimension, we can conclude the mean distance will affect the result of the average error rate under condition of both low dimension and high dimension. The sample size will affect the result of the average error rate in both low dimension and high dimension, especially when the sample size is small, the error rate is not true value. But the value of error rate will be close to some value when the sample size is large (at least greater than 1000). The value of average error rate varies by covariance matrix. Sometimes, the value is really small and pretty good, but sometimes the error rate will close to 0.5 or larger. It depends on covariance matrix makes the point together in some dimension.

```
import numpy as np
from scipy.stats import norm
from numpy import matrix
from numpy.linalg import linalg
import scipy.spatial.distance as distance
import scipy.stats as stats
import matplotlib.pyplot as plt
import math
import random

'''
this function is used to create covariance matrix for mulitple dimension
'''
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)

'''
this function is used to generate random points according the probablity
```

```
'''
def generate_random_point(p,size):
    temp = []
    for i in range(size):
        if random.random() < p:
            temp.append(0)
        else:
            temp.append(1)
    size1 = temp.count(0)
    size2 = temp.count(1)
    return size1,size2


'''
this function is used to compute descrimante function gx
'''
def compute_gx(mu_1,mu_2,cov_1,cov_2,w1,w2,x):

    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



    print v[0][0]+const
    const = math.log(float(w1)/w2)
    gx = (mu_1-mu_2)*cov_1.I*x.T+(mu_2 * cov_1.I * mu_2.T - mu_1 * cov_1.I *
mu_1.T) /2
    #print "gx",gx
    v  = gx.tolist()
    print v[0][0]+const
```

```
'''
part1_B is doing given fixed distance find the error rate with sample size in 2D
with covariance equal and not equal
'''

def part1_B(equal):
    w1 = 0.5
    w2 = 0.5

    avg_error = []

    if equal:
        #cov_1 = matrix(([5,1.2],[1.2,3]))
        cov_1 = create_variance_matrix(2)
        cov_2 = cov_1
    else:
        #cov_1 = matrix(([5,1.2],[1.2,3]))
        #cov_2 = matrix(([2,1.5],[1.5,2]))
        cov_1 = create_variance_matrix(2)
        cov_2 = create_variance_matrix(2)

    print cov_1
    print cov_2

    for size in
[5,10,15,20,50,75,100,150,200,250,300,400,500,600,700,800,900,1000,1200,150
0,3000]:
        print "size: ",size

        for distance in [4]:
            print "distance: ",distance
            mu_1 = matrix([2,distance])
            mu_2 = matrix([2,0])
```

```
    errors = []

    for time in range(10):


        size1,size2 = generate_random_point(w1,size)

        #p_X,p_Y = np.random.multivariate_normal(mu_1.tolist()[0], cov_1,
int(w1*size)).T
        #n_X,n_Y = np.random.multivariate_normal(mu_2.tolist()[0], cov_2,
int(w2*size)).T
        p_X,p_Y = np.random.multivariate_normal(mu_1.tolist()[0], cov_1,
size1).T
        n_X,n_Y = np.random.multivariate_normal(mu_2.tolist()[0], cov_2,
size2).T
        #print "cov_1",cov_1
        #print "cov_2",cov_2

        #px = plt.scatter(p_X,p_Y ,color='red',alpha = 0.5,label= "class 1")
        #py = plt.scatter(n_X,n_Y,color='blue',alpha = 0.5,label= "class 2")
        #print "distance: ",distance, ", mu_1: ",mu_1.tolist(),"mu_2:
",mu_2.tolist(),"cov1: ",cov_1.tolist(),"cov2: ",cov_2.tolist()

        #title = "distance: ",str(distance), ", mu_1: ",str(mu_1.tolist()),", mu_2:
",str(mu_2.tolist()),",\n cov1: ",str(cov_1.tolist()),", cov2: ",str(cov_2.tolist())
        #title = ''.join(title)
            #print title
        #plt.title(title)
        #plt.legend(['class 1','class 2'])
        #plt.show()

        #plt.show()

        pos = 0
        neg = 0
        eq = 0
```

```
error = 0
for i in range(len(p_X)):
   #point = create_random_point(2)
   #print "point",point

   #pt = matrix(point)
   pt = matrix((p_X[i],p_Y[i]))
   #print pt
   val =  compute_gx(mu_1,mu_2,cov_1,cov_2,w1,w2,pt)

   #print "val",val
   if val > 0:
      #pos += 1
      pass
   elif val < 0:
      neg += 1
      error += 1
   else:
      eq +=  1

for i in range(len(n_X)):

   #pt = matrix(point)
   pt = matrix((n_X[i],n_Y[i]))
   val =  compute_gx(mu_1,mu_2,cov_1,cov_2,w1,w2,pt)
   #print "val",val
   if val > 0:
      #pos += 1
      error += 1
      pos += 1
   elif val < 0:
      pass
   else:
      eq +=  1

errors.append(error)
```

```
            print errors
            avg = sum(errors)/float(len(errors))
            print avg
            print float(avg)/size
            avg_error.append(float(avg)/size)
    print avg_error

plt.plot([5,10,15,20,50,75,100,150,200,250,300,400,500,600,700,800,900,1000,12
00,1500,3000],avg_error)
    plt.xlabel("size")
    plt.ylabel("average error rate")
    plt.ylim([0,max(avg_error)*2])
    if equal:
        plt.title("2d average error vs size cov1 == cov2")
    else:
        plt.title("2d average error vs size cov1 != cov2")
    plt.show()


'''
part1_A is doing given fixed sample size find
the error rate with different mean distance 2d

'''

def part1_A(equal):
    w1 = 0.5
    w2 = 0.5
    #size = 1000
    if equal:
        #cov_1 = matrix(([5,1.4],[1.4,3]))
        cov_1 = create_variance_matrix(2)
        cov_2 = cov_1
    else:
        #cov_1 = matrix(([5,1.4],[1.4,3]))
        #cov_2 = matrix(([2,1.1],[1.1,2]))
```

```
      cov_1 = create_variance_matrix(2)
      cov_2 = create_variance_matrix(2)

   print cov_1
   print cov_2

   for size in [1000]:
      print "size: ",size
      avg_error = []


      for distance in [1,2,4,8,16]:
         print "distance: ",distance
         mu_1 = matrix([2,distance])
         mu_2 = matrix([2,0])


         #cov_1 = matrix(([5,0],[0,3]))

         errors = []

         for time in range(10):

            size1,size2 = generate_random_point(w1,size)

            #p_X,p_Y = np.random.multivariate_normal(mu_1.tolist()[0], cov_1,
int(w1*size)).T
            #n_X,n_Y = np.random.multivariate_normal(mu_2.tolist()[0], cov_2,
int(w2*size)).T
            p_X,p_Y = np.random.multivariate_normal(mu_1.tolist()[0], cov_1,
size1).T
            n_X,n_Y = np.random.multivariate_normal(mu_2.tolist()[0], cov_2,
size2).T
            #print "cov_1",cov_1
            #print "cov_2",cov_2

            px = plt.scatter(p_X,p_Y,color='red',alpha = 0.5,label= "class 1")
```

```
        py = plt.scatter(n_X,n_Y,color='blue',alpha = 0.5,label= "class 2")

        title = "distance: ",str(distance), ", mu_1: ",str(mu_1.tolist()),", mu_2: ",str(mu_2.tolist()),",\n cov1: ",str(cov_1.tolist()),", cov2: ",str(cov_2.tolist())
        title = ''.join(title)

        plt.title(title)
        plt.legend(['class 1','class 2'])
        plt.show()



        pos = 0
        neg = 0
        eq = 0

        error = 0
        for i in range(len(p_X)):
            #point = create_random_point(2)
            #print "point",point

            #pt = matrix(point)
            pt = matrix((p_X[i],p_Y[i]))

            val =  compute_gx(mu_1,mu_2,cov_1,cov_2,w1,w2,pt)

            #print "val",val
            if val > 0:
                #pos += 1
                pass
            elif val < 0:
                neg += 1
                error += 1
            else:
                eq +=  1

        for i in range(len(n_X)):
            #point = create_random_point(2)
```

```
            #print "point",point

            #pt = matrix(point)
            pt = matrix((n_X[i],n_Y[i]))

            val =  compute_gx(mu_1,mu_2,cov_1,cov_2,w1,w2,pt)
            #print "val:",val
            if val > 0:
                #pos += 1
                error += 1
                pos += 1
            elif val < 0:
                pass
            else:
                eq +=  1

        errors.append(error)

    print errors
    avg = sum(errors)/float(len(errors))
    print avg
    print float(avg)/size
    avg_error.append(float(avg)/size)
print avg_error
plt.plot([1,2,4,8,16],avg_error)
plt.xlabel("distance")
plt.ylabel("average error rate")
plt.ylim([0,max(avg_error)*2])
if equal:
    plt.title("2d average error vs distance cov1 == cov2")
else:
    plt.title("2d average error vs distance cov1 != cov2")
plt.show()


'''
```

This function is used to do given fixed sample size find the error rate with mean distance in N-d
'''

```
def part2_A(d,equal):
    mu_1 = [0]* d
    mu_2 = [0]* d

    w1 = 0.5
    w2 = 0.5


    mean_2 = matrix(mu_2)

    if equal:
        cov_1 = create_variance_matrix(d)
        cov_2 = cov_1
    else:
        cov_1 = create_variance_matrix(d)
        cov_2 = create_variance_matrix(d)

    print cov_1
    print cov_2
    print "dimesion: ", d
    for size in [1000]:
        print "size: ",size
        avg_error = []
        for distance in [1,2,4,8,16]:
            print "distance: ",distance
            mu_1[1] = distance
            #print mu_1
            mean_1 = matrix(mu_1)

            errors = []
            for time in range(10):
                size1,size2 = generate_random_point(w1,size)
```

```
            #p_X = np.random.multivariate_normal(mean_1.tolist()[0], cov_1,
int(w1*size))
            #n_X = np.random.multivariate_normal(mean_2.tolist()[0], cov_2,
int(w2*size))

            p_X = np.random.multivariate_normal(mean_1.tolist()[0], cov_1, size1)
            n_X = np.random.multivariate_normal(mean_2.tolist()[0], cov_2, size2)

            pos = 0
            neg = 0
            eq = 0

            error = 0
            #classify possitive point
            for i in range(len(p_X)):
               #point = create_random_point(2)
               #print "point",point

               #pt = matrix(point)
               pt = matrix((p_X[i]))
               #print pt
               val =  compute_gx(mu_1,mu_2,cov_1,cov_2,w1,w2,pt)

               #print "val",val
               if val > 0:
                  #pos += 1
                  pass
               elif val < 0:
                  neg += 1
                  error += 1
               else:
                  eq +=  1

            #classify negative pointe
            for i in range(len(n_X)):
               #point = create_random_point(2)
               #print "point",point
```

```
            #pt = matrix(point)
            pt = matrix((n_X[i]))
            val =  compute_gx(mu_1,mu_2,cov_1,cov_2,w1,w2,pt)
            #print "val",val
            #determine erroor
            if val > 0:
               #pos += 1
               error += 1
               pos += 1
            elif val < 0:
               pass
            else:
               eq +=  1

         errors.append(error)

      print errors
      avg = sum(errors)/float(len(errors))
      print avg
      print float(avg)/size
      avg_error.append(float(avg)/size)
   print avg_error
   plt.plot([1,2,4,8,16],avg_error)
   plt.xlabel("distance")
   plt.ylabel("average error rate")
   plt.ylim([0,max(avg_error)*2])
   if equal:
      tit = "dimension: ",str(d)," average error vs distance cov1 == cov2"
      tit = "".join(tit)
      plt.title(tit)
   else:
      tit = "dimension: ",str(d)," average error vs distance cov1 != cov2"
      tit = "".join(tit)
      plt.title(tit)
      #plt.title("dimension 27: average error vs distance cov1 != cov2")
   plt.show()
```

```
'''
This funcion is doing given fixed distance finding error rate with different sample
size in N-D

'''
c_index = 0
def part2_B(d,equal):
    mu_1 = [0]* d
    mu_2 = [0]* d

    global c_index
    colors = ['r', 'b', 'g', 'k', 'm']

    w1 = 0.5
    w2 = 0.5
    #size = 10000

    mean_2 = matrix(mu_2)
    if equal:
        cov_1 = create_variance_matrix(d)
        cov_2 = cov_1
    else:
        cov_1 = create_variance_matrix(d)
        cov_2 = create_variance_matrix(d)

    print cov_1
    print cov_2
    print "dimesion: ", d
    avg_error = []
    for size in
[5,10,15,20,50,75,100,150,200,250,300,400,500,700,900,1000,1500,2000,2500,3
000,5000,10000]:
        print "size: ",size
        for distance in [4]:
            print "distance: ",distance
            mu_1[1] = distance
```

```
        #print mu_1
        mean_1 = matrix(mu_1)

        errors = []
        for time in range(10):

            size1,size2 = generate_random_point(w1,size)

            p_X = np.random.multivariate_normal(mean_1.tolist()[0], cov_1, size1)
            n_X = np.random.multivariate_normal(mean_2.tolist()[0], cov_2, size2)
            #p_X = np.random.multivariate_normal(mean_1.tolist()[0], cov_1,
int(w1*size))
            #n_X = np.random.multivariate_normal(mean_2.tolist()[0], cov_2,
int(w2*size))

            pos = 0
            neg = 0
            eq = 0

            error = 0
            #classify possitive point
            for i in range(len(p_X)):
                #point = create_random_point(2)
                #print "point",point

                #pt = matrix(point)
                pt = matrix((p_X[i]))
                #print pt
                val =  compute_gx(mu_1,mu_2,cov_1,cov_2,w1,w2,pt)

                #print "val",val
                if val > 0:
                    #pos += 1
                    pass
                elif val < 0:
                    neg += 1
                    error += 1
```

```
                else:
                    eq +=  1

            #classify negative pointe
            for i in range(len(n_X)):

                #pt = matrix(point)
                pt = matrix((n_X[i]))
                val =  compute_gx(mu_1,mu_2,cov_1,cov_2,w1,w2,pt)
                #print "val",val

                #determine erroor
                if val > 0:
                    #pos += 1
                    error += 1
                    pos += 1
                elif val < 0:
                    pass
                else:
                    eq +=  1

            errors.append(error)

        print errors
        avg = sum(errors)/float(len(errors))
        print avg
        print float(avg)/size
        avg_error.append(float(avg)/size)
    print avg_error

plt.plot([5,10,15,20,50,75,100,150,200,250,300,400,500,700,900,1000,1500,2000,
2500,3000,5000,10000],avg_error,label=str(d),color = colors[c_index])
    #plt.xlabel("size")
    #plt.ylabel("average error rate")
    #plt.ylim([0,max(avg_error)*2])
    c_index += 1
    if equal:
```

```
        tit = "dimension: ",str(d)," average error vs size, cov1 == cov2"
        tit = "".join(tit)
        plt.title(tit)
    else:
        tit = "dimension: ",str(d)," average error vs size, cov1 != cov2"
        tit = "".join(tit)
        plt.title(tit)
    #plt.show()


if __name__ == "__main__":
    #part1_A(True)
    #part1_A(False)
    #part1_B(True) #doing 27d with same covariance and different covariance
    #part1_B(False)


    #part2_A(2,True) #doing 27d
    #part2_A(5,True) #doing 27d
    #part2_A(10,True) #doing 27d
    #part2_A(27,True) #doing 27d
    #part2_A(2,False) #doing 27d
    #part2_A(5,False) #doing 27d
    #part2_A(10,False) #doing 27d
    #part2_A(27,False) #doing 27d
    #part2_B(2,True)
    #part2_B(5,True)
    #part2_B(10,True)
    #part2_B(15,True)
    #part2_B(27,True)
    part2_B(2,False)
    part2_B(5,False)
    part2_B(10,False)
    part2_B(15,False)
    part2_B(27,False)
    plt.xlabel("size")
    plt.ylabel("average error rate")
```

```
plt.title("not diagnal cov1 != cov2")
plt.legend(loc='best')
plt.show()
```