Pattern Recognition– Spring 2018
Computer Project 1 – Solutions
Due on: Mar 22 2018 11:59pm

Jianfeng Song
UIN:426009910
Jsong26@tamu.edu

Problem 1.

Problem 1.

$\varepsilon_{m_1} = \Phi\left(\frac{\sqrt{d}}{2\sigma}\right) \quad \varepsilon_{m_2} = \Phi\left(\frac{-\sqrt{d}}{2\sigma}\frac{1}{\sqrt{1-\rho}}\right)$

$\varepsilon_{m_3} = \Phi\left(\frac{-\sqrt{d}}{-2\sigma}\frac{1}{\sqrt{1+2\rho}}\right)$

Since they are all gaussian model.

$g(x) \mid Y=0 \sim a^T x + b \mid Y=0 \sim N(a^T\mu_0 + b, a^T\Sigma a)$

$g(x) \mid Y=1 \sim a^T x + b \mid Y=1 \sim N(a^T\mu_1 + b, a^T\Sigma a)$

then we have

$\varepsilon^0[\psi^*] = P(g(x) > 0 \mid Y=0) = \Phi\left(\frac{a^T\mu_0 + b}{\sqrt{a^T\Sigma a}}\right)$

$\varepsilon^1[\psi^*] = P(g(x) \leq 0 \mid Y=1) = \Phi\left(-\frac{a^T\mu_1 + b}{\sqrt{a^T\Sigma a}}\right)$

plug in a and b we have

$\varepsilon^0 = \Phi\left(\frac{-\frac{1}{2}\delta^2 + k}{\delta}\right)$

$\delta = \left[(\mu_1-\mu_0)^T \Sigma^{-1} (\mu_1-\mu_0)\right]^{\frac{1}{2}}$

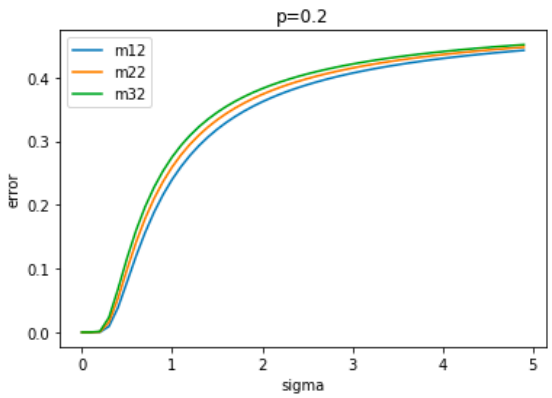
$\varepsilon^1 = \Phi\left(\frac{-\frac{1}{2}\delta^2 - k}{\delta}\right)$

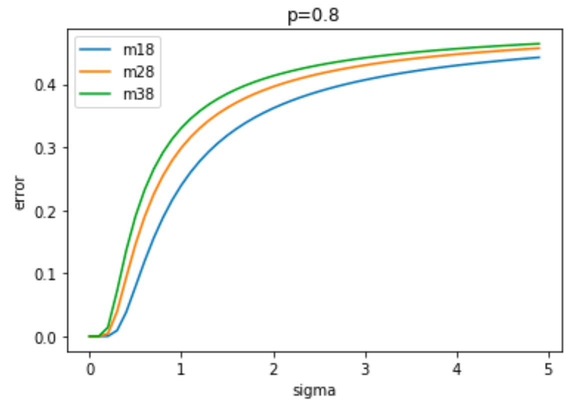
For m_1 $\delta = \left[(\mu_1-\mu_0)^T (\sigma^2 I)^{-1} (\mu_1-\mu_0)\right]^{\frac{1}{2}} = \sqrt{d}$

$\varepsilon_{m_1} = \Phi\left(-\frac{\sqrt{d}}{2\sigma}\right)$

$\cancel{\varepsilon_{m_2} = \Phi}$ $\varepsilon_{m_2} = \Phi\left(-\frac{\sqrt{d}}{2\sigma}\frac{1}{\sqrt{1+\rho}}\right)$

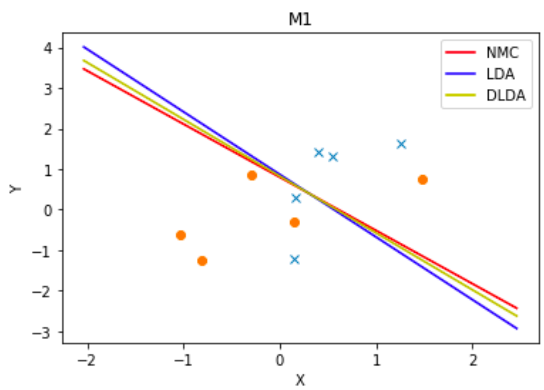
$\varepsilon_{m_3} = \Phi\left(\frac{\sqrt{d}}{-2\sigma}\frac{1}{\sqrt{1+2\rho}}\right)$

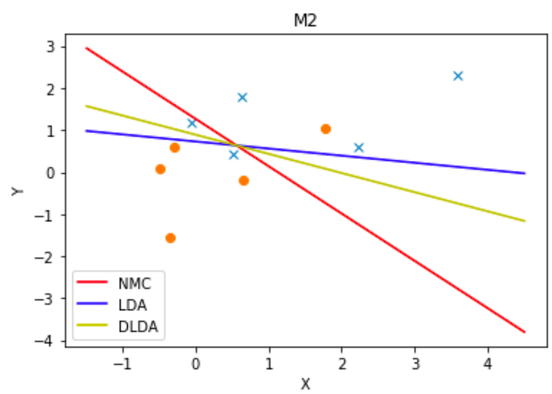




From the plot, we can see that all three errors increase as sigma increase because a larger variance means more overlapping between classes, so the error should increase as sigma increase. We can also see that as we increase the correlation between features the error also increase.

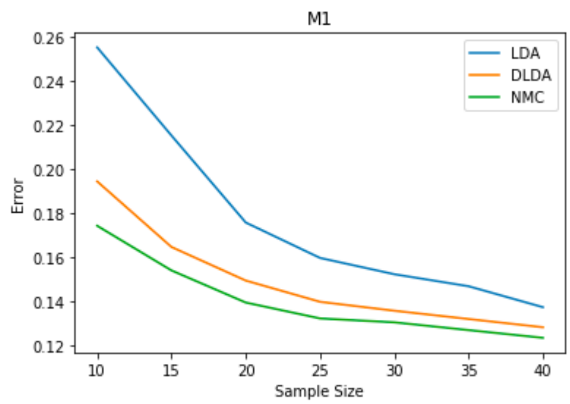
Problem 2.

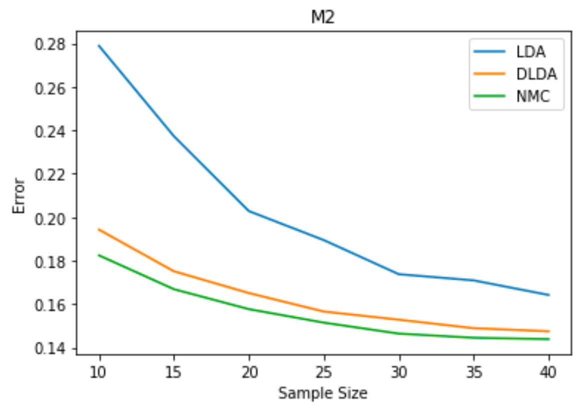


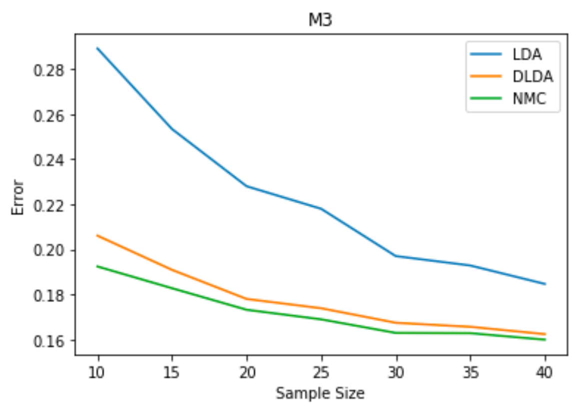


Error	LDA	DLDA	NMC
M1	0.25323838	0.23626709	0.23608276
M2	0.25111594	0.18238538	0.17878395

Problem 3.







First, we noticed that for all three models the error decreased as we increase sample size. We can also see that the error increases from m1 to m2 to m3 due to the increasing correlation of features, and NMC has the best error over LDA(Third) and DLDA(Second) in this test.

Code
Problem 1.

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Fri Mar 16 11:33:21 2018

@author: jianfengsong
"""
import numpy as np
import matplotlib.pyplot as plt
import scipy.integrate as integrate
import scipy.stats as ns
import math
import sympy as sym
from scipy.stats import norm
M1_2=list()
M1_8=list()
M2_2=list()
M2_8=list()
M3_2=list()
M3_8=list()
class fun():
    def norm_fun(d,fi,p,n):
        if n==1:
            M=np.sqrt(2)/(-2*fi)
#            print("1")
        if n==2:
            M=np.sqrt(2)/(-2*fi)/np.sqrt(1+p)
        if n==3:
            M=np.sqrt(2)/(-2*fi)/np.sqrt(1+2*p)
        return M
#fi=range(0,100)
r=np.arange(0,5,0.1)
for fi in r:
    M1_2.append(norm.cdf(fun.norm_fun(6,fi,0.2,1)))
    M1_8.append(norm.cdf(fun.norm_fun(6,fi,0.8,1)))
    M2_2.append(norm.cdf(fun.norm_fun(6,fi,0.2,2)))
    M2_8.append(norm.cdf(fun.norm_fun(6,fi,0.8,2)))
    M3_2.append(norm.cdf(fun.norm_fun(6,fi,0.2,3)))
    M3_8.append(norm.cdf(fun.norm_fun(6,fi,0.8,3)))

plt.figure(1)
plt.plot(r,M1_2,label='m12')
plt.plot(r,M2_2,label='m22')
```

```
plt.plot(r,M3_2,label='m32')
plt.xlabel('sigma')
plt.ylabel('error')
plt.title('p=0.2')
plt.legend()
plt.show()

plt.figure(2)
plt.plot(r,M1_8,label='m18')
plt.plot(r,M2_8,label='m28')
plt.plot(r,M3_8,label='m38')
plt.xlabel('sigma')
plt.ylabel('error')
plt.title('p=0.8')
plt.legend()
plt.show()
```

Problem 2.

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Mon Mar 19 17:35:16 2018

@author: jianfengsong
"""
import numpy as np
import matplotlib.pyplot as plt
import scipy.integrate as integrate
import scipy.stats as ns
import math
import sympy as sym
from sklearn import datasets
from sklearn.neighbors import NearestCentroid
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
import statistics as st
from scipy.stats import norm
#from mlpy import Dlda
cov_m1=np.array([[1,0],
            [0,1]])
cov_m2=np.array([[1,0.2],
             [0.2,1]])
m1=np.asarray([1,1]).reshape(2,1)
m0=np.asarray([0,0]).reshape(2,1)
def sample_set(sample_size,cov):
    x1= np.random.multivariate_normal([1,1], cov,sample_size)
```

```
    x0= np.random.multivariate_normal([0,0], cov,sample_size)
    return x1,x0
def take(x,y,sample,lab):
    for a in range(len(x)):
        sample.append(x[a])
        lab.append(y)

def sampledata(a):
    x1,x0=sample_set(5,cov_m1)
    x21,x20=sample_set(5,cov_m2)
    sample,lab=list(),list()
    sample_m2,lab_m2=list(),list()
    take(x1,1,sample,lab)
    take(x0,0,sample,lab)
    take(x21,1,sample_m2,lab_m2)
    take(x20,0,sample_m2,lab_m2)
    sam=np.asarray(sample)
    lab=np.asarray(lab)
    sam_m2=np.asarray(sample_m2)
    lab_m2=np.asarray(lab_m2)
    if a == 0 :
        return x1,x0,sam,lab,cov_m1
    if a == 1:
        return x21,x20,sam_m2,lab_m2,cov_m2
def cov_s(x1,x0,sam):
    cov1=np.cov(x1)
    cov0=np.cov(x0)
    covs=np.cov(sam)
    return cov1,cov0,covs
def mean(x1,x0):
    sumx=0
    sumy=0
    for b in x1:
        sumx=b+sumx
    sum1=sumx/len(x1) #sum1 is mean of u1(1,1)
    for c in x0:
        sumy=c+sumy
    sum0=sumy/len(x0) #sum0 is mean of u0(0,0)
    return sum1,sum0

###############################################################################
err_nmc,err_dlda,err_lda=list(),list(),list()
for a in range(2):###########   NMC
    x1,x0,sam,lab,m=sampledata(a)
    cov1,cov0,covs=cov_s(x1,x0,sam)   ###covariance of x1 x0 sam
    plt.figure(1+a)
```

```
    plt.title('M%i'%(1+a))
    plt.plot(x1[:,0],x1[:,1],'x')
    plt.plot(x0[:,0],x0[:,1],'o')
    sumx,sumy=0,0
    x_min, x_max = sam[:, 0].min() - 1, sam[:, 0].max() + 1
    x1_r=np.arange(x_min,x_max,0.1)
    for b in x1:
        sumx=b+sumx
    sum1=sumx/len(x1) #sum1 is mean of u1(1,1)
    for c in x0:
        sumy=c+sumy
    sum0=sumy/len(x0) #sum0 is mean of u0(0,0)
    mid=(sum1-sum0)/2
    slop=-(sum0[1]-sum1[1])/(sum0[0]-sum1[0])
    b=mid[1]-slop*mid[0]
    x2_nmc=slop*x1_r+b
    plt.plot(x1_r,x2_nmc,'r',label='NMC')
    mean_1,mean_0=mean(x1,x0)
    a_nmc=(mean_1-mean_0)
    b_nmc=(-1/2)*np.dot(a_nmc,(mean_1-mean_0).T)
    m1=np.asarray(mean_1).reshape(1,2)
    m0=np.asarray(mean_0).reshape(1,2)
    cdfn1=norm.cdf((np.dot(a_nmc,m0.T)+b_nmc)/np.sqrt(np.dot(np.dot(a_nmc,m),a_nmc.T)))
    cdfn2=norm.cdf(-(np.dot(a_nmc,m1.T)+b_nmc)/np.sqrt(np.dot(np.dot(a_nmc,m),a_nmc.T)))
    err_nmc.append(1/2*(cdfn1+cdfn2))

######## LDA
    lda = LinearDiscriminantAnalysis()
    lda.fit(sam, lab)
    cov1,cov0,covs=cov_s(x1,x0,sam)   ###covariance of x1 x0 sam
    pn1=(np.dot((x0-sum0).T,(x0-sum0))+np.dot((x1-sum1).T,(x1-sum1)))/(2*len(x0)-2)
    mean1=sum1
    mean0=sum0
    cov=(1/(2*len(x0)-2))*(np.matrix((x1-mean1)).T*np.matrix((x1-mean1))+np.matrix((x0-
mean0)).T*np.matrix((x0-mean0)))
    an1_ter=np.dot(cov**-1,(sum1-sum0))
    an1=an1_ter.T
    bn1=(-1/2)*np.dot(np.dot((sum1-sum0).T,cov**-1),(sum1-sum0))
    x2p=-bn1/an1[1]-an1[0]*x1_r/an1[1]
    plt.plot(x1_r,x2p.T,'b',label='LDA')

    #### DLDA
    sig_x0=1/2*((st.variance(x0[:,0]))+(st.variance(x1[:,0])))
    sig_x1=1/2*((st.variance(x0[:,1]))+(st.variance(x1[:,1])))
    sig=np.matrix([[sig_x0,0],
               [0,sig_x1]])
```

```
    an=np.dot(sig.I,(sum1-sum0))
    an2=an.T
    bn2=(-1/2)*np.dot(np.dot((sum1-sum0).T,sig.I),(sum1-sum0))
    x2_dlda=-bn2/an2[1]-an2[0]*x1_r/an2[1]
    plt.plot(x1_r,x2_dlda.T,'y',label='DLDA')
    plt.xlabel('X')
    plt.ylabel('Y')
    plt.title('M%i'%(a+1))
    plt.legend()
    plt.show()


err_dlda.append(1/2*(norm.cdf((np.dot(an2.T,m0.T)+bn2)/np.sqrt(np.dot(np.dot(an2.T,m),an2)))
+norm.cdf(-(np.dot(an2.T,m1.T)+bn2)/np.sqrt(np.dot(np.dot(an2.T,m),an2)))))

err_lda.append(1/2*(norm.cdf((np.dot(an1.T,m0.T)+bn1)/np.sqrt(np.dot(np.dot(an1.T,m),an1)))
+norm.cdf(-(np.dot(an1.T,m1.T)+bn1)/np.sqrt(np.dot(np.dot(an1.T,m),an1)))))
print(err_lda)
print(err_dlda)
print(err_nmc)
```

Problem 3.

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Tue Mar 20 11:10:18 2018

@author: jianfengsong
"""
import numpy as np
import random as rd
import matplotlib.pyplot as plt
import scipy.integrate as integrate
import scipy.stats as ns
import math
import sympy as sym
from sklearn import datasets
from sklearn.neighbors import NearestCentroid
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
import statistics as st
from scipy.stats import norm
sample_size=[10,14,20,24,30,34,40]
sample_size1=[10,15,20,25,30,35,40]
```

```
mean0=[0,0,0,0,0,0]
mean1=[1,1,1,1,1,1]
m1=[[1,0,0,0,0,0],
    [0,1,0,0,0,0],
    [0,0,1,0,0,0],
    [0,0,0,1,0,0],
    [0,0,0,0,1,0],
    [0,0,0,0,0,1]]

m2=[[1,0.2,0,0,0,0],
    [0.2,1,0,0,0,0],
    [0,0,1,0.2,0,0],
    [0,0,0.2,1,0,0],
    [0,0,0,0,1,0.2],
    [0,0,0,0,0.2,1]]

m3=[[1,0.2,0.2,0,0,0],
    [0.2,1,0.2,0,0,0],
    [0.2,0.2,1,0,0,0],
    [0,0,0,1,0.2,0.2],
    [0,0,0,0.2,1,0.2],
    [0,0,0,0.2,0.2,1]]
m123=[m1,m2,m3]
def random_sample(samplesize,cov):
    x1,x1_lab=list(),list()
    x0,x0_lab=list(),list()
    x1_r=np.random.multivariate_normal(mean1, cov,samplesize//2)
    x0_r=np.random.multivariate_normal(mean0, cov,samplesize//2)
    for a in x1_r:
        x1.append(a.reshape(1,6))
    for b in x0_r:
        x0.append(b.reshape(1,6))
#    for b in range(samplesize):
#        k = rd.randint(0,1)
#        if k == 1 :
#            x_1= np.random.multivariate_normal(mean1, cov,1)
#            x1.append(x_1)
#            x1_lab.append(1)
#        if k == 0:
#            x_0= np.random.multivariate_normal(mean0, cov,1)
#            x0.append(x_0)
#            x0_lab.append(0)
    return x1,x0
def take(x,y,sample,lab):
    for a in range(len(x)):
        sample.append(x[a])
```

```
        lab.append(y)
def mean(x1,x0):
    sumx=0
    sumy=0
    for b in x1:
        sumx=b+sumx
    sum1=sumx/len(x1) #sum1 is mean of u1(1,1)
    for c in x0:
        sumy=c+sumy
    sum0=sumy/len(x0) #sum0 is mean of u0(0,0)
    return sum1,sum0
########################################
h=0
for m in m123:
    h=h+1
    dif_lda,dif_dlda,dif_nmc=list(),list(),list()
    for a in sample_size:
        print (a)
        lda_err=0
        dlda_err=0
        nmc_err=0
        for b in range(100):
            x1,x0=random_sample(a,m)
            mean_1,mean_0=mean(x1,x0)
            m1=np.asarray(mean1).reshape(1,6)
            m0=np.asarray(mean0).reshape(1,6)
            ############# LDA
            cov=(1/(len(x0)+len(x1)-2))*(np.matrix((x1-mean_1)).T*np.matrix((x1-
mean_1))+np.matrix((x0-mean_0)).T*np.matrix((x0-mean_0)))
            a_lda_ter=np.dot(cov**-1,(mean_1-mean_0).T)
            a_lda=a_lda_ter.T
            b_lda=(-1/2)*np.dot(a_lda,(mean_1-mean_0).T)
            cdf1=norm.cdf((np.dot(a_lda,m0.T)+b_lda)/np.sqrt(np.dot(np.dot(a_lda,m),a_lda.T)))
            cdf2=norm.cdf(-(np.dot(a_lda,m1.T)+b_lda)/np.sqrt(np.dot(np.dot(a_lda,m),a_lda.T)))
            err_lda=1/2*(cdf1+cdf2)
            lda_err=lda_err+err_lda
            ############ DLDA
            x0_r=np.asarray(x0).reshape(len(x0),len(x0[0][0]))
            x1_r=np.asarray(x1).reshape(len(x1),len(x1[0][0]))

            sig_x0=1/2*((st.variance(x0_r[:,0]))+(st.variance(x1_r[:,0])))
            sig_x1=1/2*((st.variance(x0_r[:,1]))+(st.variance(x1_r[:,1])))
            sig_x2=1/2*((st.variance(x0_r[:,2]))+(st.variance(x1_r[:,2])))
            sig_x3=1/2*((st.variance(x0_r[:,3]))+(st.variance(x1_r[:,3])))
            sig_x4=1/2*((st.variance(x0_r[:,4]))+(st.variance(x1_r[:,4])))
            sig_x5=1/2*((st.variance(x0_r[:,5]))+(st.variance(x1_r[:,5])))
```

```
            covd=np.matrix([[sig_x0,0,0,0,0,0],
                         [0,sig_x1,0,0,0,0],
                         [0,0,sig_x2,0,0,0],
                         [0,0,0,sig_x3,0,0],
                         [0,0,0,0,sig_x4,0],
                         [0,0,0,0,0,sig_x5]])
            a_dlda_ter=np.dot(covd**-1,(mean_1-mean_0).T)
            a_dlda=a_dlda_ter.T
            b_dlda=(-1/2)*np.dot(a_dlda,(mean_1-mean_0).T)

cdfd1=norm.cdf((np.dot(a_dlda,m0.T)+b_dlda)/np.sqrt(np.dot(np.dot(a_dlda,m),a_dlda.T)))
            cdfd2=norm.cdf(-
(np.dot(a_dlda,m1.T)+b_dlda)/np.sqrt(np.dot(np.dot(a_dlda,m),a_dlda.T)))
            err_dlda=1/2*(cdfd1+cdfd2)
            dlda_err=dlda_err+err_dlda
            ################# NMC
            a_nmc=(mean_1-mean_0)
            b_nmc=(-1/2)*np.dot(a_nmc,(mean_1-mean_0).T)

cdfn1=norm.cdf((np.dot(a_nmc,m0.T)+b_nmc)/np.sqrt(np.dot(np.dot(a_nmc,m),a_nmc.T)))
            cdfn2=norm.cdf(-
(np.dot(a_nmc,m1.T)+b_nmc)/np.sqrt(np.dot(np.dot(a_nmc,m),a_nmc.T)))
            err_nmc=1/2*(cdfn1+cdfn2)
            nmc_err=nmc_err+err_nmc
        dif_lda.append(lda_err/100)
        dif_dlda.append(dlda_err/100)
        dif_nmc.append(nmc_err/100)
    lda=np.asarray(dif_lda).reshape(7,1)
    dlda=np.asarray(dif_dlda).reshape(7,1)
    nmc=np.asarray(dif_nmc).reshape(7,1)
    sample_size2=np.asarray(sample_size1).reshape(7,1)
    plt.figure(h)
    plt.title('M%i'%h)
    plt.plot(sample_size2,lda,label='LDA')
    plt.plot(sample_size2,dlda,label='DLDA')
    plt.plot(sample_size2,nmc,label='NMC')
    plt.xlabel('Sample Size')
    plt.ylabel('Error')
    plt.legend()
    plt.show()
```