

Homework 2

Shao-Ting Chiu (UIN:433002162)

10/10/22

Table of contents

Homework Description	1
Computational Enviromnent Setup	2
Third-party libraries	2
Version	2
Problem 3.6 (Python Assignment)	3
Problem 4.2	5
(a)	5
(b)	6
(c)	7
Problem 4.3	7
(a)	7
(b)	9
Problem 4.4	10
(a)	10
(b)	12
(c)	14
Problem 4.8 (Python Assignment)	14
(a)	14
(b)	16
(c)	17
(d)	19
References	20

Homework Description

- Course: ECEN649, Fall2022
- Problems from the book:
 - 3.6 (10 pt)
 - 4.2 (10 pt)
 - 4.3 (10 pt)

4.4 (10 pt)

4.8 (20 pt)

- Deadline: Oct. 12th, 11:59 am

Computational Environment Setup

Third-party libraries

```
1 %matplotlib inline
2 import sys # system information
3 import matplotlib # plotting
4 import scipy.stats as st # scientific computing
5 import pandas as pd # data managing
6 import numpy as np # numerical computation
7 import numba
8 import sklearn as sk
9 from numpy import linalg as LA
10 import scipy as sp
11 import scipy.optimize as opt
12 import sympy as sp
13 import matplotlib.pyplot as plt
14 from numpy.linalg import inv, det
15 from numpy.random import multivariate_normal as mvn
16 from numpy.random import binomial as binom
17 from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA #problem 4.8
18 from sklearn.model_selection import train_test_split
19 # Matplotlib setting
20 plt.rcParams['text.usetex'] = True
21 matplotlib.rcParams['figure.dpi'] = 300
```

Version

```
1 print(sys.version)
2 print(matplotlib.__version__)
3 print(sp.__version__)
4 print(np.__version__)
5 print(pd.__version__)
6 print(sk.__version__)
```

3.8.14 (default, Sep 6 2022, 23:26:50)
[Clang 13.1.6 (clang-1316.0.21.2.5)]
3.3.1
1.6.2

1.19.1
1.1.1
1.1.2

Problem 3.6 (Python Assignment)

Using the synthetic data model in Section A8.1 for the homoskedastic case with $\mu_0 = (0, \dots, 0)$, $\mu_1 = (1, \dots, 1)$, $P(Y = 0) = P(Y = 1)$, and $k = d$ (independent features), generate a large number (e.g., $M = 1000$) of training data sets for each sample size $n = 20$ to $n = 100$, in steps of 10, with $d = 2, 5, 8$, and $\sigma = 1$. Obtain an approximation of the expected classification error $E[\epsilon_n]$ of the nearest centroid classifier in each case by averaging ϵ_n , computed using the exact formula (3.13), over the M synthetic training data sets. Plot $E[\epsilon_n]$ as a function of the sample size, for $d = 2, 5, 8$ (join the individual points with lines to obtain a smooth curve). Explain what you see.

- The formula in Braga-Neto (2020, 56, Eq. 3.13)

$$\begin{aligned}
 -\epsilon_n &= \frac{1}{2} \left(\Phi \left(\frac{a_n^T \hat{\mu}_0 + b_n}{\|a_n\|} \right) + \Phi \left(-\frac{a_n^T \hat{\mu}_1 + b_n}{\|a_n\|} \right) \right) \\
 * \mu_0 &= (0, \dots, 0); \hat{\mu}_0 = \frac{1}{N_0} \sum_{i=1}^n X_i I_{Y_i=0} \\
 * \mu_1 &= (1, \dots, 1); \hat{\mu}_1 = \frac{1}{N_1} \sum_{i=1}^n X_i I_{Y_i=1} \\
 * a_n &= \hat{\mu}_1 - \hat{\mu}_0 \\
 * b_n &= \frac{(\hat{\mu}_1 - \hat{\mu}_0)(\hat{\mu}_1 + \hat{\mu}_0)}{2}
 \end{aligned}$$

```

1 def hat_mu(m):
2     return np.mean(m, axis=0)
3
4 def get_an(hm0, hm1):
5     return hm1 - hm0
6
7 def get_bn(hm0, hm1):
8     return (hm1 - hm0)*(hm1+hm0).T/2
9
10 def epsilon(hmu0, hmu1, p0=0.5):
11     p1 = 1-p0
12     an = get_an(hmu0, hmu1)
13     bn = get_bn(hmu0, hmu1)
14     epsilon0 = st.norm.cdf((an*hmu0.T + bn)/LA.norm(an))
15     epsilon1 = st.norm.cdf(-(an*hmu1.T + bn)/LA.norm(an))
16     return (p0*epsilon0 + p1*epsilon1)[0][0]
17
18 class GaussianDataGen:
```

```

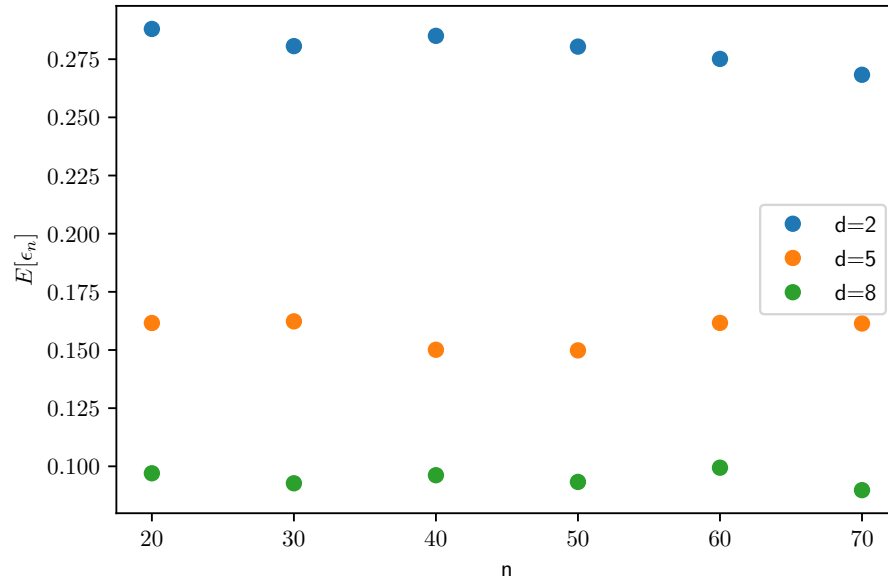
19     def __init__(self, n, d, s=1, mu=0):
20         self.n = n
21         self.d = d
22         self.mu = np.ones(d) * mu
23         self.s = s
24         self.cov = self.get_cov()
25
26     def get_cov(self):
27         return np.identity(self.d) * self.s
28
29     def sample(self):
30         hmuV = np.zeros(self.d)
31         for i in range(0,self.d):
32             hmuV[i] = np.mean(np.random.normal(self.mu[0], self.s, self.n))
33         return np.matrix(hmuV)
34
35     def cal_eps(dg0, dg1, p0=0.5):
36         hmuV0 = dg0.sample()
37         hmuV1 = dg1.sample()
38         return epsilon(hmuV0, hmuV1, p0=0.5)
39     cal_eps_func = np.vectorize(cal_eps)
40
41     def exp_try_nd(n, d, s=1,M=1000):
42         gX0 = GaussianDataGen(n=n, d=d, s= s,mu=0)
43         gX1 = GaussianDataGen(n=n, d=d, s= s, mu=1)
44         eps = cal_eps_func([gX0 for i in range(0,M)], gX1)
45         return np.mean(eps)
46     exp_try_nd_func = np.vectorize(exp_try_nd)
47
48
49     M = 1000
50     ns = np.arange(20,80, 10)
51     s = 1
52     dres = {2:[],5:[],8:[]}
53
54
55     for k in dres.keys():
56         dres[k] = exp_try_nd_func(ns,k,M)
57
58
59     fig, ax = plt.subplots()
60     for k in dres.keys():
61         ax.plot(ns, dres[k], 'o',label="d={}".format(k))
62     ax.set_xlabel("n")

```

```

63 ax.set_ylabel("$E[\\epsilon_n]$")
64 ax.legend();

```



Problem 4.2

A common method to extend binary classification rules to K classes, $K > 2$, is the *one-vs-one approach*, in which $K(K-1)$ classifiers are trained between all pairs of classes, and a majority vote of assigned labels is taken.

(a)

Formulate a multiclass version of parametric plug-in classification using the one-vs-one approach.

Let $\psi_{i,j}^*$ be a one-one classifiers that $i \neq j$, and $\{(i, j) | i \in [1, k], j \in [1, k], i \neq j\}$. For K classes, there are $K(K-1)$ classifiers; for each classifier $\psi_{i,j}^*$ and $x \in R^d$,

$$\psi_{i,j,n}^* = \begin{cases} 1, & D_{i,j,n}(x) > k_{i,j,n} \\ 0, & \text{otherwise} \end{cases} \quad (1)$$

where

- $D_{i,j,n}(x) = \ln \frac{p(x|\theta_{i,n})}{p(x|\theta_{j,n})}$

- $k_{ij,n} = \ln \frac{P(Y=j)}{P(Y=i)}$
- Noted that feature-label distribution is expressed via a family of PDF $\{p(x|\theta_i)|\theta \in \Theta \subseteq R^m\}$, for $i = 1, \dots, K$.

Let $\psi_{i,n}^* = \prod_{j \neq i} I_{\psi_{ij,n}^*}$, and the one-vs-one classifier is

$$\psi_n^*(x) = \arg \max_{k=1, \dots, K} \psi_{k,n}^* \quad (2)$$

(b)

Show that if the threshold $k_{ij,n}$ between classes i and j is given by $\ln \frac{\hat{c}_j}{\hat{c}_i}$, then the one-vs-one parametric classification rule is equivalent to the simple decision.

$$\psi_n(x) = \arg \max_{k=1, \dots, K} \hat{c}_k p(x|\theta_{k,n}), x \in R^d$$

(For simplicity, you may ignore the possibility of ties.)

$$\ln \frac{p(x|\theta_{i,n})}{p(x|\theta_{j,n})} > k_{ij,n} = \ln \frac{\hat{c}_j}{\hat{c}_i} \quad (3)$$

$$\ln p(x|\theta_{i,n}) - \ln p(x|\theta_{j,n}) > \ln \hat{c}_j - \ln \hat{c}_i \quad (4)$$

$$\hat{c}_i p(x|\theta_{i,n}) > \hat{c}_j p(x|\theta_{j,n}) \quad (5)$$

$$\psi_{ij,n}^* = \begin{cases} 1, & \hat{c}_i p(x|\theta_{i,n}) > \hat{c}_j p(x|\theta_{j,n}) \\ 0, & \text{otherwise} \end{cases} \quad (6)$$

$$(7)$$

Continue Equation 2. For given $i \in [1, k]$, let j_a , $a \in [1, k-1]$ be the order sequence of $1, \dots, k$ exclude i .

$$\psi_{i,n}^* = 1 \text{ iff, } \hat{c}_i p(x|\theta_{i,n}) > \hat{c}_{j_1} p(x|\theta_{j_1,n}) \geq \hat{c}_{j_2} p(x|\theta_{j_2,n}) \geq \hat{c}_{j_{k-1}} p(x|\theta_{j_{k-1},n})$$

Therefore,

$$\psi_{1,n}^* = \begin{cases} 1, & \max_{k=1, \dots, K} \hat{c}_k p(x|\theta_{k,n}) = \hat{c}_1 p(x|\theta_1, n) \\ 0, & \text{otherwise} \end{cases} \quad (8)$$

$$\vdots \quad (9)$$

$$\psi_{K,n}^* = \begin{cases} 1, & \max_{k=1, \dots, K} \hat{c}_K p(x|\theta_{k,n}) = \hat{c}_K p(x|\theta_K, n) \\ 0, & \text{otherwise} \end{cases} \quad (10)$$

$$(11)$$

We can conclude that

$$\psi_n^*(x) = \arg \max_{k=1,\dots,K} \psi_{k,n}^* \quad (12)$$

$$= \arg \max_{k=1,\dots,K} \hat{c}_k p(x|\theta_k, n) \quad (13)$$

(c)

Applying the approach in items (a) and (b), formulate a multiclass version of Gaussian discriminant analysis. In the case of multiclass NMC, with all thresholds equal to zero, how does the decision boundary look like?

 Tip

- Think about 3-classes, 2-dimension
- Write the D of each classifier

$$D^*(x) = \frac{1}{2}(x - \mu_0)^T \Sigma_0^{-1}(x - \mu_0) - \frac{1}{2}(x - \mu_1)^T \Sigma_1^{-1}(x - \mu_1) + \frac{1}{2} \ln \frac{\det(\Sigma_0)}{\det(\Sigma_1)}$$

- replace $\mu_0 \rightarrow \hat{\mu}_0$; $\Sigma_0 \rightarrow \hat{\Sigma}_0$

Problem 4.3

Under the general Gaussian model $p(x|Y=0) \sim \mathcal{N}_d(\mu_0, \Sigma_0)$ and $p(x|Y=1) \sim \mathcal{N}_d(\mu_1, \Sigma_1)$, the classification error $\epsilon_n = P(\psi_n(X) \neq Y|S_n)$ of *any* linear classifier in the form

$$\psi_n(x) = \begin{cases} 1, & a_n^T x + b_n > 0, \\ 0, & \text{otherwise} \end{cases} \quad (14)$$

(examples discussed so far include LDA and its variants, and the logistic classifier) can be readily computed in terms of Φ (the CDF of a standard normal random variable), the classifier parameters a_n and b_n , and the distributional parameters $c = P(Y=1)$, μ_0 , μ_1 , Σ_0 , and Σ_1 .

(a)

Show that

$$\epsilon_n = (1 - c)\Phi\left(\frac{a_n^T\mu_0 + b_n}{\sqrt{a_n^T\Sigma_0a_n}}\right) + c\Phi\left(-\frac{a_n^T\mu_1 + b_n}{\sqrt{a_n^T\Sigma_1a_n}}\right)$$

Hint: the discriminant $a_n^T x + b_n$ has a simple Gaussian distribution in each class.

From Braga-Neto (2020, Eq. 2.34),

$$\epsilon^* = \underbrace{P(Y = 0)}_{=1-c} \epsilon^0[\psi^*] + \underbrace{P(Y = 1)}_{=c} \epsilon^1[\psi^*]$$

$$\epsilon^0[\psi^*] = P(a_n^T x + b_n > 0 | Y = 0) \quad (15)$$

$$(16)$$

Use the affine property of Gaussian distribution described in Braga-Neto (2020) [pp. 307. G4]¹.

- $a_n^T x + b_n | Y = 0 \sim N(a_n^T \mu_0 + b_n, \underbrace{a_n^T \Sigma_0 a_n}_{\sigma^2})$

$$\epsilon^0[\psi^*] = 1 - P(a_n^T x + b_n \leq 0 | Y = 0) \quad (17)$$

$$= 1 - \Phi\left(\frac{0 - (a_n^T \mu_0 + b_n)}{\sqrt{a_n^T \Sigma_0 a_n}}\right) \quad (18)$$

$$= 1 - \Phi\left(-\frac{a_n^T \mu_0 + b_n}{\sqrt{a_n^T \Sigma_0 a_n}}\right) \quad (19)$$

$$= \Phi\left(\frac{a_n^T \mu_0 + b_n}{\sqrt{a_n^T \Sigma_0 a_n}}\right) \quad (20)$$

Similarly,

$$\epsilon^1(\psi^*) = P(a_n^T x + b_n < 0 | Y = 1) \quad (21)$$

$$= \Phi\left(\frac{0 - (a_n^T \mu_1 + b_n)}{\sqrt{a_n^T \Sigma_1 a_n}}\right) \quad (22)$$

$$= \Phi\left(-\frac{a_n^T \mu_1 + b_n}{\sqrt{a_n^T \Sigma_1 a_n}}\right) \quad (23)$$

¹Any affine transformation $f(x) = AX + B$ of a Gaussian is a Gaussian. That is, A is a nongingular matrix, and B is a vector. If $X \sim N(\mu, \Sigma)$, $a^T X + b \sim N(A^T \mu + b, A^T \Sigma A)$. (ardianumam 2017)

Combining together,

$$\epsilon^* = \underbrace{P(Y=0)}_{=1-c} \epsilon^0[\psi^*] + \underbrace{P(Y=1)}_{=c} \epsilon^1[\psi^*] \quad (24)$$

$$= (1-c)\epsilon^0[\psi^*] + c\epsilon^1[\psi^*] \quad (25)$$

$$= (1-c)\Phi\left(\frac{a_n^T \mu_0 + b_n}{\sqrt{a_n^T \Sigma_0 a_n}}\right) + c\Phi\left(-\frac{a_n^T \mu_1 + b_n}{\sqrt{a_n^T \Sigma_1 a_n}}\right) \quad (26)$$

(b)

Compute the errors of the NMC, LDA, and DLDA classifiers in Example 4.2 if $c = 1/2$,

$$\mu_0 = \begin{bmatrix} 2 \\ 3 \end{bmatrix}, \mu_1 = \begin{bmatrix} 6 \\ 5 \end{bmatrix}, \Sigma_0 = \begin{bmatrix} 1 & 1 \\ 1 & 2 \end{bmatrix}, \text{ and } \Sigma_1 = \begin{bmatrix} 4 & 0 \\ 0 & 1 \end{bmatrix}$$

Which classifier does the best?

As shown in Table 1, NMC has the lowest Bayes error.

```

1 def epsilon_general(an, bn, c, mu0, mu1, sig0, sig1):
2
3     e0 = st.norm.cdf(\
4         float(an.T @ mu0 + bn)/\
5         np.sqrt(float(an.T @ sig0 @ an)))
6     e1 = st.norm.cdf(\
7         -float(an.T @ mu1 + bn)/\
8         np.sqrt(float(an.T @ sig1 @ an)) )
9
10    return (1-c)*e0 + c*e1
11
12    truth = {
13        "c": 0.5,
14        "mu0": np.matrix([[2],[3]]),
15        "mu1": np.matrix([[6],[5]]),
16        "sig0": np.matrix([[1,1],[1,2]]),
17        "sig1": np.matrix([[4,0],[0,1]]),
18    }
19
20    meth = {
21        "NMC":{
22            "an": np.matrix([[4],[2]]),
23            "bn": -24
24        },

```

Table 1: Bayes Errors of NMC, LDA and DLDA

	Method	Bayes Error
0	NMC	0.084775
1	LDA	0.085298
2	DLDA	0.087606

```

25     "LDA":{
26         "an": 3/7*np.matrix([[5], [3]]),
27         "bn": -96/7
28     },
29     "DLDA":{
30         "an": 2/5*np.matrix([[6], [5]]),
31         "bn": -88/5
32     }
33 }
34
35 berrors = np.zeros(len(meth.keys()))
36
37 for (i,k) in enumerate(meth.keys()):
38     berrors[i] = epsilon_general(**meth[k], **truth)
39
40 pd.DataFrame({"Method": list(meth.keys()),\
41     "Bayes Error": berrors}).sort_values(\
42     ["Bayes Error"], ascending=[1])

```

Problem 4.4

Even in the Gaussian case, the classification error of quadratic classifiers in general require numerical integration for its computation. In some special simple cases, however, it is possible to obtain exact solutions. Assume a two-dimensional Gaussian problem with $P(Y = 1) = \frac{1}{2}$, $\mu_0 = \mu_1 = 0$, $\Sigma_0 = \sigma_0^2 I_2$, and $\Sigma_1 = \sigma_1^2 I_2$. For definiteness, assume that $\sigma_0 < \sigma_1$.

(a)

Show that the Bayes classifier is given by

$$\psi^*(x) = \begin{cases} 1, & \|x\| > r^*, \\ 0, & \text{otherwise,} \end{cases} \quad \text{where } r^* = \sqrt{2 \left(\frac{1}{\sigma_0^2} - \frac{1}{\sigma_1^2} \right)^{-1} \ln \frac{\sigma_1^2}{\sigma_0^2}} \quad (27)$$

In particular, the optimal decision boundary is a circle of radius r^* .

The inverted Σ_1 and Σ_2 are²

$$\Sigma_0 = \sigma_0^2 I_2 = \begin{bmatrix} \sigma_0^2 & 0 \\ 0 & \sigma_0^2 \end{bmatrix} \quad (29)$$

$$\Sigma_0^{-1} = \frac{1}{\sigma_0^4} \begin{bmatrix} \sigma_0^2 & 0 \\ 0 & \sigma_0^2 \end{bmatrix} = \sigma_0^{-2} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = \sigma_0^{-2} I_2 \quad (30)$$

$$\Sigma_1^{-1} = \sigma_1^{-2} I_2 \quad (31)$$

Use the derivation in Braga-Neto (2020, 74),

$$A_n = \begin{bmatrix} a_{11} & a_{12} \\ a_{12} & a_{22} \end{bmatrix} = \frac{-1}{2} \Sigma_1^{-1} - \Sigma_0^{-1} = \frac{-1}{2} (\sigma_1^{-2} - \sigma_0^{-2}) \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad (32)$$

$$b_n = \begin{bmatrix} b_{n,1} \\ b_{n,2} \end{bmatrix} = \Sigma_1^{-1} \underbrace{\mu_1}_{=0} - \Sigma_0^{-1} \underbrace{\mu_0}_{=0} \quad (33)$$

$$= \begin{bmatrix} 0 \\ 0 \end{bmatrix} \quad (34)$$

$$c = -\frac{1}{2} \ln \frac{|\Sigma_1|}{|\Sigma_0|} = \frac{-1}{2} \ln \frac{\sigma_1^4}{\sigma_0^4} = -\ln \frac{\sigma_1^2}{\sigma_0^2}$$

According to Braga-Neto (2020, Eq. 4.26), the 2-dimensional QDA decision boundary is

$$D(x) = a_{11}x_1^2 + 2a_{12}x_1x_2 + a_{22}x_2^2 + b_1x_1 + b_2x_2 + c = 0 \quad (35)$$

$$a_{11}(x_1^2 + x_2^2) = \ln \frac{\sigma_1^2}{\sigma_0^2} \quad (36)$$

$$x_1^2 + x_2^2 = 2\left(\frac{1}{\sigma_0^2} - \frac{1}{\sigma_1^2}\right)^{-1} \ln \frac{\sigma_1^2}{\sigma_0^2} \quad (37)$$

$$r^* = \sqrt{x_1^2 + x_2^2} = \sqrt{2\left(\frac{1}{\sigma_0^2} - \frac{1}{\sigma_1^2}\right)^{-1} \ln \frac{\sigma_1^2}{\sigma_0^2}} \quad (38)$$

2

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix}^{-1} = \frac{1}{ad-bc} \begin{bmatrix} d & -b \\ -c & a \end{bmatrix} \quad (28)$$

Noted that $\left(\frac{1}{\sigma_0^2} - \frac{1}{\sigma_1^2}\right) > 0$ because $\sigma_0 < \sigma_1$

For any point $\|x_j\| > r^*$, the discriminant (D) is larger than 0, and $\psi^*(x_j) = 1$.

(b)

Show that the corresponding Bayes error is given by

$$\epsilon^* = \frac{1}{2} - \frac{1}{2} \left(\frac{\sigma_1^2}{\sigma_0^2} - 1 \right) e^{-(1 - \frac{\sigma_0^2}{\sigma_1^2})^{-1} \ln \frac{\sigma_1^2}{\sigma_0^2}}$$

In particular, the Bayes error is a function only of the ratio of variances $\frac{\sigma_1^2}{\sigma_0^2}$, and $\epsilon^* \rightarrow 0$ as $\frac{\sigma_1^2}{\sigma_0^2} \rightarrow \infty$.

Hint: use polar coordinates to solve the required integrals analytically.

Part I: Definition of errors

$$\epsilon^0[\psi^*] = P(D^*(X) > k^* | Y = 0) \quad (39)$$

$$= P(\|x\| > r^* | Y = 0) \quad (40)$$

$$\epsilon^1[\psi^*] = P(D^*(X) \leq k^* | Y = 1) \quad (41)$$

$$= P(\|x\| \leq r^* | Y = 1) \quad (42)$$

Part II: PDF of 2D Gaussian

$$p(x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}) = \frac{1}{\sqrt{(2\pi)^2 \sigma^4}} \exp\left(-\frac{1}{2} x^T \Sigma^{-1} x\right) \quad (43)$$

$$= \frac{1}{\sqrt{(2\pi)^2 \sigma^4}} \exp\left(-\frac{1}{2} \frac{x_1^2 + x_2^2}{\sigma^2}\right) \quad (44)$$

$$= \frac{1}{2\pi\sigma^2} \exp\left(-\frac{x_1^2 + x_2^2}{2\sigma^2}\right) \quad (45)$$

$$(46)$$

Use the polar coordination, $x_1 = r \cos \theta$ and $x_2 = r \sin \theta$. $x_1^2 + x_2^2 = r^2$. We can transform 2D gaussian into polar coordination:

$$p(r, \theta) = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{r^2}{2\sigma^2}\right)$$

Part III: Integration

$$\epsilon^0[\psi^*] = \int_{\theta=0}^{\theta=2\pi} \int_{r=r^*}^{\infty} \frac{1}{2\pi\sigma_0^2} \exp\left(-\frac{r^2}{2\sigma_0^2}\right) r dr d\theta \quad (47)$$

$$= \frac{1}{2\pi\sigma_0^2} \int_{\theta=0}^{\theta=2\pi} \int_{r=r^*}^{\infty} \exp\left(-\frac{r^2}{2\sigma_0^2}\right) r dr d\theta \quad (48)$$

$$= \frac{1}{2\pi\sigma_0^2} \int_{\theta=0}^{\theta=2\pi} \sigma_0^2 \exp\left(-\frac{r_*^2}{2\sigma_0^2}\right) d\theta \quad (49)$$

$$= \exp\left(-\frac{r_*^2}{2\sigma_0^2}\right) \quad (50)$$

$$= \exp\left(-\frac{1}{\sigma_0^2(\sigma_0^{-2} - \sigma_1^{-2})} \ln \frac{\sigma_1^2}{\sigma_0^2}\right) \quad (51)$$

$$= \exp\left(\frac{-1}{(1 - \frac{\sigma_0^2}{\sigma_1^2})} \ln \frac{\sigma_1^2}{\sigma_0^2}\right) \quad (52)$$

$$= \exp\left(-\left(1 - \frac{\sigma_0^2}{\sigma_1^2}\right)^{-1} \ln \frac{\sigma_1^2}{\sigma_0^2}\right) \quad (53)$$

$$\epsilon^1[\psi^*] = \int_{\theta=0}^{\theta=2\pi} \int_{r=0}^{r=r^*} \frac{1}{2\pi\sigma_1^2} \exp\left(-\frac{r^2}{2\sigma_1^2}\right) r dr d\theta \quad (54)$$

$$= 1 - \exp\left(-\frac{r_*^2}{2\sigma_1^2}\right) \quad (55)$$

$$= 1 - \exp\left(-\frac{1}{\sigma_1^2(\sigma_0^{-2} - \sigma_1^{-2})} \ln \frac{\sigma_1^2}{\sigma_0^2}\right) \quad (56)$$

$$= 1 - \exp\left(-\frac{1}{(\frac{\sigma_1^2}{\sigma_0^2} - 1)} \ln \frac{\sigma_1^2}{\sigma_0^2}\right) \quad (57)$$

$$= 1 - \exp\left(-\frac{\frac{\sigma_0^2}{\sigma_1^2}}{(1 - \frac{\sigma_0^2}{\sigma_1^2})} \ln \frac{\sigma_1^2}{\sigma_0^2}\right) \quad (58)$$

$$= 1 - \exp\left(-\frac{\sigma_0^2}{\sigma_1^2} \left(1 - \frac{\sigma_0^2}{\sigma_1^2}\right)^{-1} \ln \frac{\sigma_1^2}{\sigma_0^2}\right) \quad (59)$$

Part IV: Combining together

$$\epsilon^* = P(Y = 0)\epsilon^0[\psi^*] + P(Y = 1)\epsilon^1[\psi^*] \quad (60)$$

$$= \frac{1}{2}\epsilon^0 + \frac{1}{2}\epsilon^1 \quad (61)$$

$$= \frac{1}{2} \exp\left(-\left(1 - \frac{\sigma_0^2}{\sigma_1^2}\right)^{-1} \ln \frac{\sigma_1^2}{\sigma_0^2}\right) + \frac{1}{2} - \frac{1}{2} \exp\left(-\frac{\sigma_0^2}{\sigma_1^2} \left(1 - \frac{\sigma_0^2}{\sigma_1^2}\right)^{-1} \ln \frac{\sigma_1^2}{\sigma_0^2}\right) \quad (62)$$

$$= \frac{1}{2} \exp\left(-\left(1 - \frac{\sigma_0^2}{\sigma_1^2}\right)^{-1} \ln \frac{\sigma_1^2}{\sigma_0^2}\right) + \frac{1}{2} \quad (63)$$

$$- \frac{1}{2} \exp\left(-\left(\frac{\sigma_0^2}{\sigma_1^2} - 1\right) \left(1 - \frac{\sigma_0^2}{\sigma_1^2}\right)^{-1} \ln \frac{\sigma_1^2}{\sigma_0^2}\right) \exp\left(-\left(1 - \frac{\sigma_0^2}{\sigma_1^2}\right)^{-1} \ln \frac{\sigma_1^2}{\sigma_0^2}\right) \quad (64)$$

$$= \frac{1}{2} + \frac{1}{2} \left[1 - \exp\left(-\underbrace{\left(\frac{\sigma_0^2}{\sigma_1^2} - 1\right) \left(1 - \frac{\sigma_0^2}{\sigma_1^2}\right)^{-1}}_{=-1} \ln \frac{\sigma_1^2}{\sigma_0^2}\right) \right] \exp\left(-\left(1 - \frac{\sigma_0^2}{\sigma_1^2}\right)^{-1} \ln \frac{\sigma_1^2}{\sigma_0^2}\right) \quad (65)$$

$$= \frac{1}{2} + \frac{1}{2} \left[1 - \exp\left(\ln \frac{\sigma_1^2}{\sigma_0^2}\right) \right] \exp\left(-\left(1 - \frac{\sigma_0^2}{\sigma_1^2}\right)^{-1} \ln \frac{\sigma_1^2}{\sigma_0^2}\right) \quad (66)$$

$$= \frac{1}{2} + \frac{1}{2} \left[1 - \frac{\sigma_1^2}{\sigma_0^2} \right] \exp\left(-\left(1 - \frac{\sigma_0^2}{\sigma_1^2}\right)^{-1} \ln \frac{\sigma_1^2}{\sigma_0^2}\right) \quad (67)$$

$$= \frac{1}{2} - \frac{1}{2} \left(\frac{\sigma_1^2}{\sigma_0^2} - 1 \right) \exp\left(-\left(1 - \frac{\sigma_0^2}{\sigma_1^2}\right)^{-1} \ln \frac{\sigma_1^2}{\sigma_0^2}\right) \quad (68)$$

$$(69)$$

(c)

Compare the optimal classifier to the QDA classifier in Example 4.3. (Braga-Neto 2020, 74) Compute the error of the QDA classifier and compare to the Bayes error.

Problem 4.8 (Python Assignment)

Apply linear discriminant analysis to the stacking fault energy (SFE) dataset (see Braga-Neto (2020, sec. A8.4)), already mentioned in Braga-Neto (2020, ch. 1). Categorize the SFE values into two classes, low ($\text{SFE} \leq 35$) and high ($\text{SFE} \geq 45$), excluding the middle values.

(a)

Apply the preprocessing steps in `c01_matex.py` to obtain a data matrix of dimensions $123(\text{number of sample points}) \times 7(\text{number of features})$, as described in Braga-Neto (2020, sec. 1.8.2). Define low ($\text{SFE} \leq 35$) and high ($\text{SFE} \geq 45$) labels for the data.

Table 2: Filtered data

	C	N	Ni	Fe	Mn	Si	Cr	SFE
0	0.004	0.003	15.6	64.317	0.03	0.02	17.5	51.6
1	0.020	0.009	15.6	64.188	0.03	0.03	17.6	54.6
2	0.020	0.002	14.0	66.409	0.03	0.01	17.1	50.3
3	0.005	0.001	15.6	63.866	0.19	0.01	17.7	52.8

Pick the first 50% of the sample points to be the training data and the remaining 50% to be test data³.

```

1 # Setting
2 def get_SFE_low(df):
3     df_ = df[df["SFE"]<=35]
4     return df_.loc[:, df_.columns!='SFE']
5 def get_SFE_high(df):
6     df_ = df[df["SFE"]>=45]
7     return df_.loc[:, df_.columns!='SFE']
8 # Load data
9 SFE_data = pd.read_table("data/Stacking_Fault_Energy_Dataset.txt")
10 # pre-process the data
11 f_org = SFE_data.columns[:-1] # original features
12 n_org = SFE_data.shape[0] # original number of training points
13 p_org = np.sum(SFE_data.iloc[:, :-1]>0)/n_org # fraction of nonzero components for each fe
14 f_drp = f_org[p_org<0.6] # features with less than 60% nonzero co
15 SFE1 = SFE_data.drop(f_drp,axis=1) # drop those features
16 s_min = SFE1.min(axis=1)
17 SFE2 = SFE1[s_min!=0] # drop sample points with any zero values
18 SFE = SFE2[(SFE2.SFE<35)|(SFE2.SFE>45)] # drop sample points with middle responses
19 train, test = train_test_split(SFE, test_size=0.5, shuffle=False)

1 print(SFE.shape)
2 SFE.head(4)

(123, 8)

1 print(train.shape)
2 train.head(4)

```

³

All, for the last problem, please make sure you are dividing the data 50% - 50% for training and testing, the values in the book are incorrect. — Ulisses [[link to the Slack dialogue](#)]

Table 3: Train data

	C	N	Ni	Fe	Mn	Si	Cr	SFE
0	0.004	0.003	15.6	64.317	0.03	0.02	17.5	51.6
1	0.020	0.009	15.6	64.188	0.03	0.03	17.6	54.6
2	0.020	0.002	14.0	66.409	0.03	0.01	17.1	50.3
3	0.005	0.001	15.6	63.866	0.19	0.01	17.7	52.8

Table 4: Test data

	C	N	Ni	Fe	Mn	Si	Cr	SFE
295	0.07	0.40	16.13	54.818	9.64	0.45	18.48	65.0
296	0.07	0.54	16.13	54.678	9.64	0.45	18.48	53.0
297	0.04	0.04	9.00	70.920	1.20	0.40	18.20	30.4
298	0.04	0.04	9.00	70.920	1.20	0.40	18.20	25.7

(61, 8)

```
1 print(test.shape)
2 test.head(4)
```

(62, 8)

(b)

Using the function `ttest_ind` from the `scipy.stats` module, apply Welch's two-sample t-test on the training data, and produce a table with the predictors, T statistic, and p -value, ordered with largest absolute T statistics at the top.

```
1 df_low = get_SFE_low(train)
2 df_high = get_SFE_high(train)
3
4 ttest = st.ttest_ind(df_low, df_high)
5
6 tdf = pd.DataFrame({
7     "Features": df_low.keys(),
8     "T-statistics": ttest.statistic,
9     "P-Values": ttest.pvalue
10 })
11
12 tdf = tdf.sort_values(["T-statistics"], ascending=[0])
13 tdf
```


	Features	T-statistics	P-Values
3	Fe	5.934069	1.663380e-07
0	C	1.640007	1.063253e-01
5	Si	1.182468	2.417634e-01
6	Cr	0.039704	9.684632e-01
4	Mn	-0.209783	8.345594e-01
1	N	-0.955007	3.434710e-01
2	Ni	-8.878685	1.820603e-12

(c)

Pick the top two predictors and design an LDA classifier. (This is an example of *filter feature selection*, to be discussed in Chapter 9.). Plot the training data with the superimposed LDA decision boundary. Plot the testing data with the superimposed previously-obtained LDA decision boundary. Estimate the classification error rate on the training and test data. What do you observe?

Both train and test data has higher error rate compared to the result in Table 5. Because the train data (Figure 1) is inseparable with single boundary. This implies that we need extra informative feature to make this two classes separatable.

```

1 features = list(tdf["Features"][0:2])
2 var1, var2 = features

1 def get_data_with_features(data, features):
2     X = data[features].values
3     Y = data.SFE > 45
4     return X, Y
5
6 def get_loss(X, Y, clf):
7     loss = sk.metrics.zero_one_loss(Y, clf.predict(X))
8     return loss
9
10 X, Y = get_data_with_features(train, features)
11 X_test, Y_test = get_data_with_features(test, features)
12
13 clf = LDA(priors=(0.5,0.5))
14 clf.fit(X,Y.astype(int))
15 a = clf.coef_[0]
16 b = clf.intercept_[0];
17
18 # Error

```

```

19 err_train = get_loss(X,Y, clf)
20 err_test = get_loss(X_test, Y_test, clf)

1 def plot_swe_predict(ax, X, Y, clf, title=""):
2     ax.scatter(X[~Y,0],X[~Y,1],c='blue',s=32,label='Low SFE')
3     ax.scatter(X[Y,0],X[Y,1],c='orange',s=32,label='High SFE')
4     left,right = ax.get_xlim()
5     bottom,top = ax.get_ylim()
6     ax.plot([left,right],[-left*a[0]/a[1]-b/a[1],-right*a[0]/a[1]-b/a[1]],'k',linewidth=2)
7     ax.set_title(title)
8     ax.set_xlim(left,right)
9     ax.set_ylim(bottom,top)
10    ax.set_xlabel(var1)
11    ax.set_ylabel(var2)
12    ax.legend();
13
14 fig43tr, ax43tr = plt.subplots()
15 ax43tr = plot_swe_predict(ax43tr, X, Y, clf, title="Train error {}".format(err_train));

```

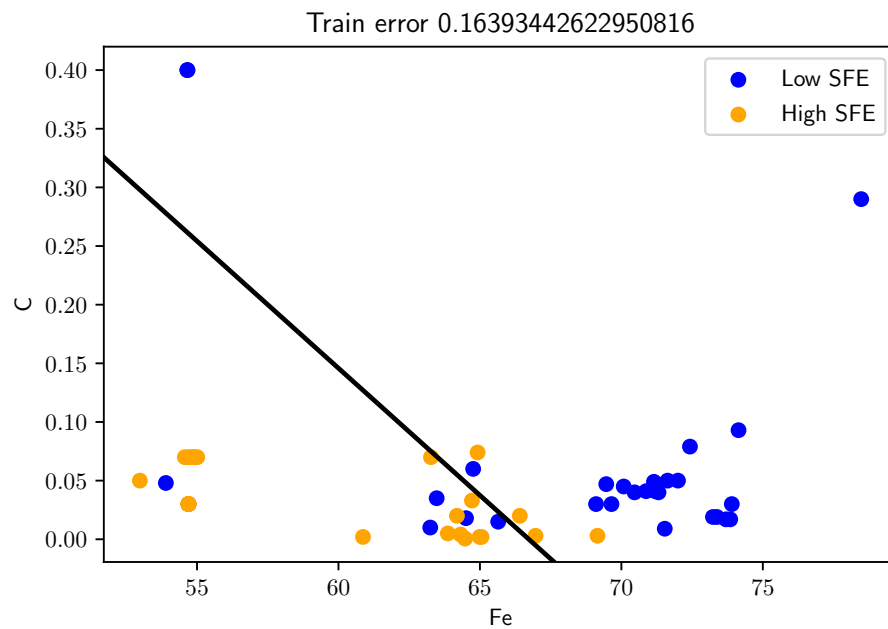


Figure 1: Train data with LDA

```

1 fig43t, ax43t = plt.subplots()
2 ax43t = plot_swe_predict(ax43t, X_test, Y_test, clf, title="Test error {}".format(err_test))

```

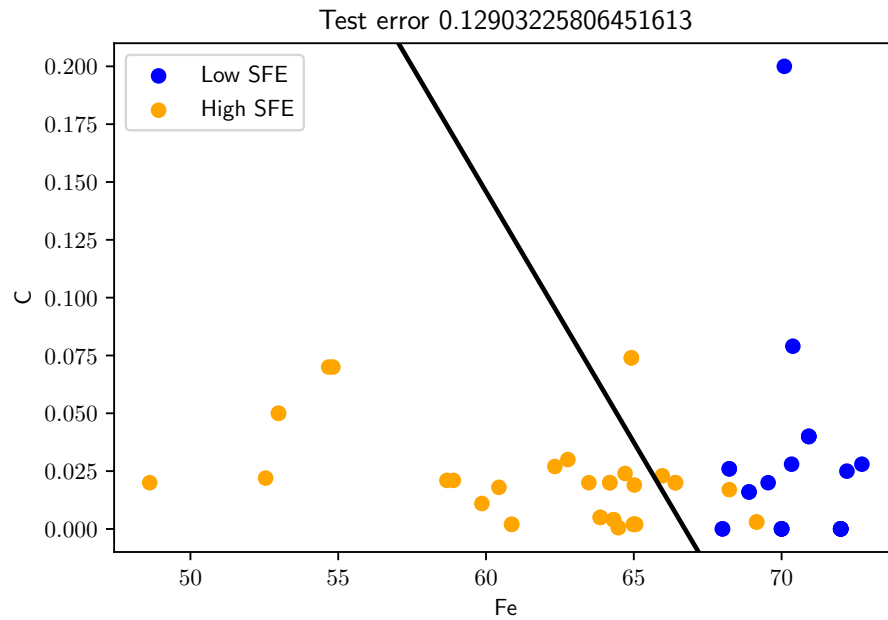


Figure 2: Test data with LDA

(d)

Repeat for the top three, and five predictors. Estimate the errors on the training and testing data (there is no need to plot the classifiers). What can you observe?

As shown in Table 5, 3 features can get the lowest testing error because the approach gathers those informative features. For 4 and 5 features, the test error rate increases due to their t -values close to 0.

```

1 n_features = [2,3,4,5]
2 err_trains = np.zeros(len(n_features))
3 err_tests = np.zeros(len(n_features))
4 for (j,i) in enumerate(n_features):
5     fs = list(tdf["Features"][0:i])
6     X, Y = get_data_with_features(train, fs)
7     X_test, Y_test = get_data_with_features(test, fs)
8

```

Table 5: Surveying different number of features

	Number of Features	Training Error Rate	Testing Error Rate
0	2	0.163934	0.129032
1	3	0.131148	0.080645
2	4	0.081967	0.161290
3	5	0.081967	0.145161

```

9     clf = LDA(priors=(0.5,0.5))
10    clf.fit(X,Y.astype(int))
11
12    # Error
13    err_trains[j] = get_loss(X,Y, clf)
14    err_tests[j] = get_loss(X_test, Y_test, clf)
15
16    pd.DataFrame({
17        "Number of Features": n_features,
18        "Training Error Rate": err_trains,
19        "Testing Error Rate": err_tests
20    })

```

References

- ardianumam. 2017. “Understanding Multivariate Gaussian, Gaussian Properties and Gaussian Mixture Model.” *Ardian Umam Blog*.
- Braga-Neto, Ulisses. 2020. *Fundamentals of Pattern Recognition and Machine Learning*. Springer.