Homework 2

Shao-Ting Chiu (UIN:433002162)

10/11/22

Table of contents

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 Enviromnent Setup

Third-party libraries

```python
%matplotlib inline
import sys # system information
import matplotlib # plotting
import scipy.stats as st # scientific computing
import pandas as pd # data managing
import numpy as np # numerical comuptation
import numba
import sklearn as sk
from numpy import linalg as LA
import scipy as sp
import scipy.optimize as opt
import sympy as sp
import matplotlib.pyplot as plt
from numpy.linalg import inv, det
from numpy.random import multivariate_normal as mvn
from numpy.random import binomial as binom
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA #problem 4.8
from sklearn.model_selection import train_test_split
# Matplotlib setting
plt.rcParams['text.usetex'] = True
matplotlib.rcParams['figure.dpi']= 300
np.random.seed(20221011)
```

Version

```python
print(sys.version)
print(matplotlib.__version__)
print(sp.__version__)
print(np.__version__)
print(pd.__version__)
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)
- $\epsilon_n = \frac{1}{2}\left(\Phi\left(\frac{a_n^T\hat{\mu}_0+b_n}{\|a_n\|}\right) + \Phi\left(-\frac{a_n^T\hat{\mu}_1+b_n}{\|a_n\|}\right)\right)$
  - $\mu_0 = (0, \dots, 0)$; $\hat{\mu}_0 = \frac{1}{N_0}\sum_{i=1}^{n} X_i I_{Y_i=0}$
  - $\mu_1 = (1, \dots, 1)$; $\hat{\mu}_1 = \frac{1}{N_1}\sum_{i=1}^{n} X_i I_{Y_i=1}$
  - $a_n = \hat{\mu}_1 - \hat{\mu}_0$
  - $b_n = -\frac{(\hat{\mu}_1-\hat{\mu}_0)^T(\hat{\mu}_1+\hat{\mu}_0)}{2}$[1]
- As shown in Figure 1, the error rate converges to optimal error as the sample size increases.

```python
def norml(v):
    return LA.norm(v, 2)

def get_an(hm0,hm1):
    return hm1 - hm0

def get_bn(hm0,hm1):
    return -float((hm1 - hm0).T @ (hm1+hm0))/2

def epsilon(hmu0, hmu1, mu0, mu1,p0=0.5):
    p1 = 1-p0
```

[1] All, in Example 3.4 there is a negative sign missing. The value of bn is -(m1-m0)^T(m1+m0)/2. — Ulisses on Slack

```
    an = get_an(hmu0, hmu1)
    bn = get_bn(hmu0, hmu1)
    epsilon0 = st.norm.cdf( (float(an.T* mu0) + bn)/norml(an))
    epsilon1 = st.norm.cdf(- (float(an.T*mu1)+ bn)/norml(an))
    return p0*epsilon0 + p1*epsilon1

class GaussianDataGen:
    def __init__(self, n, d, s=1, mu=0):
        self.n = n
        self.d = d
        self.mu = np.matrix(np.ones(d) * mu).T
        self.s = s
        self.cov = self.get_cov()

    def get_cov(self):
        return np.identity(self.d) * self.s

    def sample(self):
        data = np.random.normal(self.mu[0][0], self.s, size= (self.d, self.n))
        hmuV = np.mean(data, axis=1)
        return np.matrix(hmuV).T

def cal_eps(dg0, dg1, p0=0.5):
    hmuV0 = dg0.sample()
    hmuV1 = dg1.sample()
    mu0 = np.matrix(np.zeros(dg0.d)).T
    mu1 = np.matrix(np.ones(dg1.d)).T
    return epsilon(hmuV0, hmuV1, mu0, mu1,p0=0.5)
cal_eps_func = np.vectorize(cal_eps)

def exp_try_nd(n, d, s=1,M=1000):
    gX0 = GaussianDataGen(n=n, d=d, s= s,mu=0)
    gX1 = GaussianDataGen(n=n, d=d, s= s, mu=1)
    #eps = cal_eps_func([gX0]*M, gX1)
    eps = [cal_eps_func(gX0, gX1) for i in range(0,M)]
    return np.mean(eps)
exp_try_nd_func = np.vectorize(exp_try_nd)

def bayes_ncc(mu0, mu1):
    return st.norm.cdf(- norml(mu1-mu0)/2)


M = 1000
ns = np.arange(20,100, 10)
```

```
s = 1
dres = {2:[],5:[],8:[]}

for k in dres.keys():
    dres[k]= exp_try_nd_func(ns,k)

# Optimal error
opts = [bayes_ncc(np.zeros(k), np.ones(k)) for k in dres.keys()]

fig, ax = plt.subplots()
sts = ["-", "--", ":"]
for (i, k) in enumerate(dres.keys()):
    ax.plot(ns, dres[k], 'o',label="d={}".format(k))
    ax.axhline(y= opts[i], label="Optimal d = {}".format(k), color='k', linestyle=sts[i])

ax.set_xlabel("n")
ax.set_ylabel("$E[\\epsilon_n]$")
ax.legend();
```

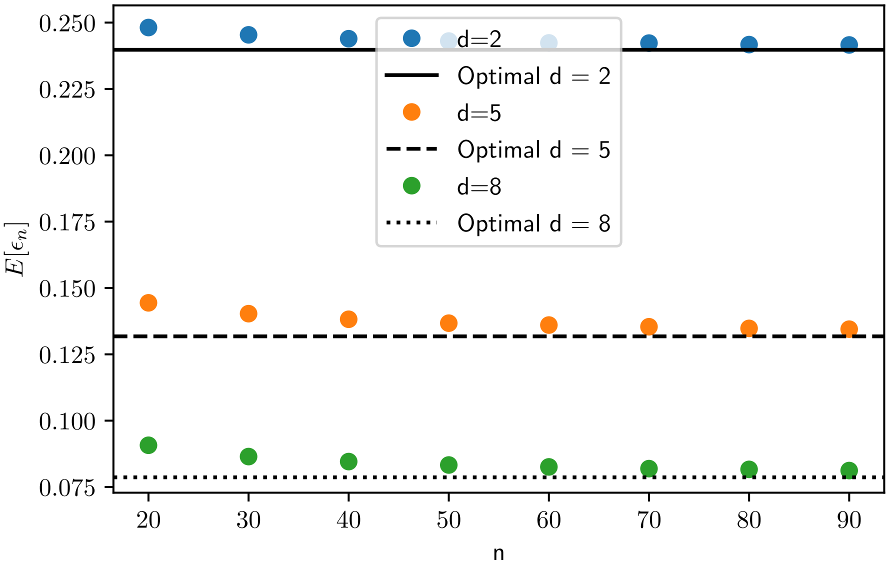

Figure 1: Error of En

Problem 4.2

A common method to extend binary classification rules to K classes, $K > 2$, is the *one-vs-one approach*, in which $\frac{K(K-1)}{2}$ classifiers are trained between all pairs of classes[2], 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 $x \in R^d$,

$$\psi^*_{ij,n} = \begin{cases} 1, & D_{ij,n}(x) > k_{ij,n} \\ 0, & \text{otherwise} \end{cases} \tag{1}$$

where

- $D_{ij,n}(x) = \ln \frac{p(x|\theta_{i,n})}{p(x|\theta_{j,n})}$
- $k_{ij,n} = \ln \frac{P(Y=j)}{P(Y=i)}$
- Noted that feature-label distribution is expressed via a familty of PDF $\{p(x|\theta_i)|\theta \in \Theta \subseteq R^m\}$, for $i = 1, \dots, K$.
- $\psi^*_{ij,n} = 1 \otimes \psi^*_{ji,n}$. These two are similar classifiers with inverted outcome.

Let $\psi^*_{i,n} = \sum_{j \neq i} \psi^*_{ij,n}$, and the one-vs-one classifier is

$$\psi^*_n(x) = \arg \max_{k=1,\dots,K} \psi^*_{k,n} \tag{2}$$

(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.)

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

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

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

[2] Also, in Problem 4.2, the number of classifiers is K(K-1)/2 not K(K-1) — Ulisses on TAMU Slack

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

$$= I_{\hat{c}_i p(x|\theta_{i,n}) > \hat{c}_j p(x|\theta_{j,n})} \tag{7}$$

Then,

$$\psi^*_{i,n} = \sum_{j\neq i} \psi^*_{ij,n} \tag{8}$$

$$= \sum_{j\neq i} I_{\hat{c}_i p(x|\theta_{i,n}) > \hat{c}_j p(x|\theta_{j,n})} \tag{9}$$

$$\psi^*_n(x) = \arg \max_{k=1,\ldots,K} \psi^*_{k,n} \tag{10}$$

$$= \arg \max_{k=1,\ldots,K} \sum_{j\neq i} \psi^*_{kj,n} \tag{11}$$

$$= \arg \max_{k=1,\ldots,K} \sum_{j\neq i} I_{\hat{c}_k p(x|\theta_{k,n}) > \hat{c}_j p(x|\theta_{j,n})} \tag{12}$$

$$\tag{13}$$

Let $\psi^*_n(x) = \kappa$, that means $\psi^*_{\kappa,n}$ is the maximum among $\{\psi^*_{j,n} | j = (1, \ldots, K)\}$. Assume there is an $s \neq \kappa$ s.t. $\hat{c}_s p(x|\theta_{s,n}) > \hat{c}_\kappa p(x|\theta_{\kappa,n}) >$ the rests. Thus, $I_{\hat{c}_s p(x|\theta_{s,n}) > \hat{c}_\kappa p(x|\theta_{\kappa,n})} = 1$

$$\psi^*_{s,n} = \sum_{j\neq s} I_{\hat{c}_s p(x|\theta_{s,n}) > \hat{c}_j p(x|\theta_{j,n})} = \underbrace{\sum_{j\neq s; j\neq \kappa} I_{\hat{c}_s p(x|\theta_{s,n}) > \hat{c}_j p(x|\theta_{j,n})}}_{=a} + \underbrace{I_{\hat{c}_s p(x|\theta_{s,n}) > \hat{c}_\kappa p(x|\theta_{\kappa,n})}}_{=1}$$

$$\psi^*_{\kappa,n} = \sum_{j\neq \kappa} I_{\hat{c}_\kappa p(x|\theta_{\kappa,n}) > \hat{c}_j p(x|\theta_{j,n})} = \underbrace{\sum_{j\neq \kappa; j\neq s} I_{\hat{c}_\kappa p(x|\theta_{\kappa,n}) > \hat{c}_j p(x|\theta_{j,n})}}_{=a} + \underbrace{I_{\hat{c}_\kappa p(x|\theta_{\kappa,n}) > \hat{c}_s p(x|\theta_{s,n})}}_{=0}$$

where a is a nonnegative number. That means

$$\psi^*_{\kappa,n} < \psi^*_{s,n} \tag{14}$$

$\psi^*_{\kappa,n}$ is not the maximum. Equation 14 is contradict to the statement that $\psi^*_n(x) = \kappa$. In conclusion, $\hat{c}_k p(x|\theta_\kappa, n)$ is the maximum if $\psi_n(x) = k$.

(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?

For Gaussian discriminant analyis, the discriminant is defined as

$$\hat{D}^*_{ij}(x) = \frac{1}{2}(x-\hat{\mu}_i)^T\hat{\Sigma}_i^{-1}(x-\hat{\mu}_i) - \frac{1}{2}(x-\hat{\mu}_j)^T\hat{\Sigma}_j^{-1}(x-\hat{\mu}_j) + \frac{1}{2}\ln\frac{\det(\hat{\Sigma}_i)}{\det(\hat{\Sigma}_j)}$$

$$\psi^*_{ij,n}(x) = I_{\hat{D}^*_{ij}(x)>0} \tag{15}$$

$$\psi^*_n = \arg\max_{k=1,\dots,K} \hat{c}_k p(x|\hat{\mu}_{k,n}, \hat{\Sigma}_{k,n}) = \arg\max_{k=1,\dots,K} Normal(x; \hat{\mu}_{k,n}, \hat{\Sigma}_{k,n})$$

For NMC case, suppose there are 3 classes with $d = 2$.

- $K = 3$
- Number of classifiers: $\frac{3\cdot 2}{2} = 3$
  - $\psi^*_{1,2}, \psi^*_{1,3}, \psi^*_{2,3}$

$$boundary = \begin{cases} \hat{\Sigma}^{-1}(\hat{\mu}_1 - \hat{\mu}_2)\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + (\hat{\mu}_2 - \hat{\mu}_1)^T\hat{\Sigma}^{-1}\left(\frac{\hat{\mu}_1+\hat{\mu}_2}{2}\right) = 0 \\ \hat{\Sigma}^{-1}(\hat{\mu}_1 - \hat{\mu}_3)\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + (\hat{\mu}_3 - \hat{\mu}_1)^T\hat{\Sigma}^{-1}\left(\frac{\hat{\mu}_1+\hat{\mu}_3}{2}\right) = 0 \\ \hat{\Sigma}^{-1}(\hat{\mu}_2 - \hat{\mu}_3)\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + (\hat{\mu}_3 - \hat{\mu}_2)^T\hat{\Sigma}^{-1}\left(\frac{\hat{\mu}_2+\hat{\mu}_3}{2}\right) = 0 \end{cases} \tag{16}$$

Problem 4.3

> Under the general Gaussian model $p(x|Y=0) \sim \mathcal{N}_d(\mu_0, \sum_0)$ and $p(x|Y=1) \sim \mathcal{N}_d(\mu_1, \sum_1)$, the classification error $\epsilon_n = P(\psi_n(X) \neq Y|S_n)$ of *any* linear classifier in the form
>
> $$\psi_n(x) = \begin{cases} 1, & a_n^T x + b_n > 0, \\ 0, & \text{otherwise} \end{cases} \tag{17}$$
>
> (examples discussed so far include LDA and its variants, and the logistic classifier) can be readily computed in terms of Φ (the CDF

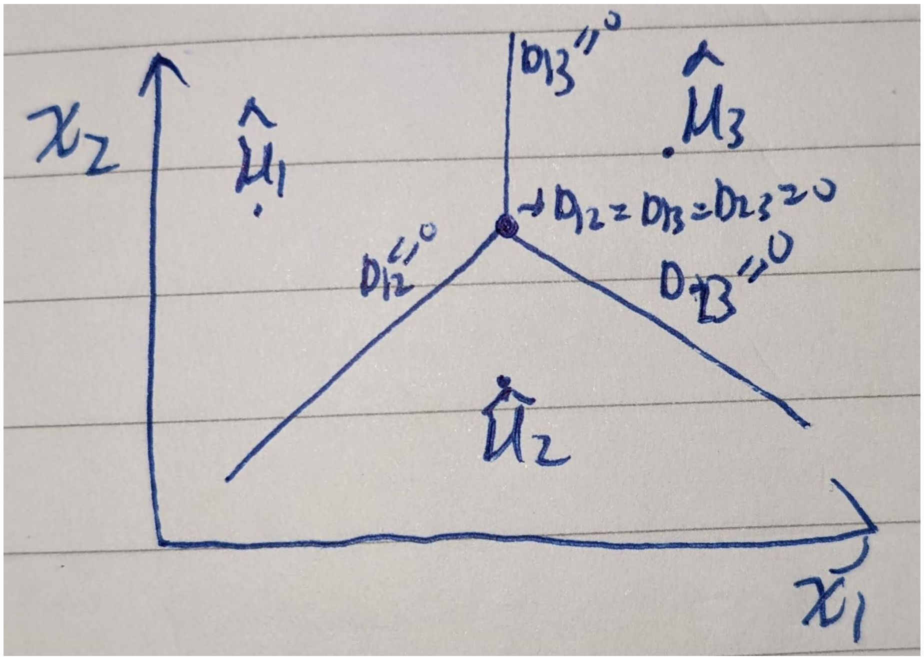

Figure 2: Homoskedastic cases

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

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

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

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

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

$$\begin{align}\epsilon^0[\psi^*] = P(a_n^T x + b_n > 0 | Y = 0) \tag{18}\\ \tag{19}\end{align}$$

Use the affine property of Gaussian distribution described in Braga-Neto (2020) [pp. 307. G4][3].

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

$$\begin{align}\epsilon^0[\psi^*] &= 1 - P(a_n^T x + b_n \leq 0 | Y = 0) \tag{20}\\ &= 1 - \Phi\left(\frac{0 - (a_n^T \mu_0 + b_n)}{\sqrt{a_n^T \Sigma_0 a_n}}\right) \tag{21}\\ &= 1 - \Phi\left(-\frac{a_n^T \mu_0 + b_n}{\sqrt{a_n^T \Sigma_0 a_n}}\right) \tag{22}\\ &= \Phi\left(\frac{a_n^T \mu_0 + b_n}{\sqrt{a_n^T \Sigma_0 a_n}}\right) \tag{23}\end{align}$$

Similarly,

$$\begin{align}\epsilon^1(\psi^*) &= P(a_n^T x + b_n < 0 | Y = 1) \tag{24}\\ &= \Phi\left(\frac{0 - (a_n^T \mu_1 + b_n)}{\sqrt{a_n^T \Sigma_1 a_n}}\right) \tag{25}\\ &= \Phi\left(-\frac{a_n^T \mu_1 + b_n}{\sqrt{a_n^T \Sigma_1 a_n}}\right) \tag{26}\end{align}$$

Combining together,

$$\begin{align}\epsilon^* &= \underbrace{P(Y = 0)}_{=1-c} \epsilon^0[\psi^*] + \underbrace{P(Y = 1)}_{=c} \epsilon^1[\psi^*] \tag{27}\\ &= (1 - c)\epsilon^0[\psi^*] + c\epsilon^1[\psi^*] \tag{28}\\ &= (1 - c)\Phi\left(\frac{a_n^T \mu_0 + b_n}{\sqrt{a_n^T \Sigma_0 a_n}}\right) + c\Phi\left(-\frac{a_n^T \mu_1 + b_n}{\sqrt{a_n^T \Sigma_1 a_n}}\right) \tag{29}\end{align}$$

[3] Any affine transformation $f(x) = AX + B$ of a Gaussian is a Guassian. 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)

(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.

```python
def epsilon_general(an, bn, c, mu0, mu1, sig0, sig1):

    e0 = st.norm.cdf(\
        float(an.T @ mu0 + bn)/\
        np.sqrt(float(an.T @ sig0 @ an)))
    e1 = st.norm.cdf(\
        -float(an.T @ mu1 + bn)/\
        np.sqrt(float(an.T @ sig1 @an)) )

    return (1-c)*e0 + c*e1

truth = {
    "c": 0.5,
    "mu0": np.matrix([[2],[3]]),
    "mu1": np.matrix([[6],[5]]),
    "sig0": np.matrix([[1,1],[1,2]]),
    "sig1": np.matrix([[4,0],[0,1]]),
}

meth = {
    "NMC":{
        "an": np.matrix([[4],[2]]),
        "bn": -24
    },
    "LDA":{
        "an": 3/7*np.matrix([[5], [3]]),
        "bn": -96/7
    },
    "DLDA":{
        "an": 2/5*np.matrix([[6],[5]]),
        "bn": -88/5
    }
}

berrors = np.zeros(len(meth.keys()))
```

Table 1: Bayes Errors of NMC, LDA and DLDA

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

```

for (i,k) in enumerate(meth.keys()):
    berrors[i] = epsilon_general(**meth[k], **truth)

pd.DataFrame({"Method": list(meth.keys()),\
    "Bayes Error": berrors}).sort_values(\
    ["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

$$\psi^*(x) = \begin{cases} 1, & \|x\| > r^*, \\ 0, & \text{otherwise} \end{cases}, \qquad \text{where } r^* = \sqrt{2\left(\frac{1}{\sigma_0^2} - \frac{1}{\sigma_1^2}\right)^{-1} \ln \frac{\sigma_1^2}{\sigma_0^2}} \tag{30}$$

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

The inverted Σ_1 and Σ_2 are[4]

[4]
$$\begin{bmatrix} a & b \\ c & d \end{bmatrix}^{-1} = \frac{1}{ad - bc} \begin{bmatrix} d & -b \\ -c & a \end{bmatrix} \tag{31}$$

$$\Sigma_0 = \sigma_0^2 I_2 = \begin{bmatrix}\sigma_0^2 & 0 \\ 0 & \sigma_0^2\end{bmatrix} \tag{32}$$

$$\Sigma_0^{-1} = \frac{1}{\sigma_0^4}\begin{bmatrix}\sigma_0^2 & 0 \\ 0 & \sigma_0^2\end{bmatrix} = \sigma_0^{-2}\begin{bmatrix}1 & 0 \\ 0 & 1\end{bmatrix} = \sigma_0^{-2} I_2 \tag{33}$$

$$\Sigma_1^{-1} = \sigma_1^{-2} I_2 \tag{34}$$

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

$$A_n = \begin{bmatrix}a_{11} & a_{12} \\ a_{12} & a_{22}\end{bmatrix} = \frac{-1}{2}\Sigma_1^{-1} - \Sigma_0^{-1} = \frac{-1}{2}(\sigma_1^{-2} - \sigma_0^{-2})\begin{bmatrix}1 & 0 \\ 0 & 1\end{bmatrix} \tag{35}$$

$$b_n = \begin{bmatrix}b_{n,1} \\ b_{n,2}\end{bmatrix} = \Sigma_1^{-1}\underbrace{\boldsymbol{\mu}_1}_{=0} - \Sigma_0^{-1}\underbrace{\boldsymbol{\mu}_0}_{=0} \tag{36}$$

$$= \begin{bmatrix}0 \\ 0\end{bmatrix} \tag{37}$$

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

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

$$D(x) = a_{11}x_1^2 + 2a_{12}x_1x_2 + a_{22}x_2^2 + b_1x_1 + b_2x_2 + c = 0 \tag{38}$$

$$a_{11}(x_1^2 + x_2^2) = \ln\frac{\sigma_1^2}{\sigma_0^2} \tag{39}$$

$$x_1^2 + x_2^2 = 2(\frac{1}{\sigma_0^2} - \frac{1}{\sigma_1^2})^{-1}\ln\frac{\sigma_1^2}{\sigma_0^2} \tag{40}$$

$$r^* = \sqrt{x_1^2 + x_2^2} = \sqrt{2(\frac{1}{\sigma_0^2} - \frac{1}{\sigma_1^2})^{-1}\ln\frac{\sigma_1^2}{\sigma_0^2}} \tag{41}$$

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

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

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

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

Part I: Definition of errors

$$\begin{aligned}
\epsilon^0[\psi^*] &= P(D^*(X) > k^*|Y=0) && (42)\\
&= P(\|x\| > r^*|Y=0) && (43)
\end{aligned}$$

$$\begin{aligned}
\epsilon^1[\psi^*] &= P(D^*(X) \le k^*|Y=1) && (44)\\
&= P(\|x\| \le r^*|Y=1) && (45)
\end{aligned}$$

Part II: PDF of 2D Gaussian

$$\begin{aligned}
p(x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}) &= \frac{1}{\sqrt{(2\pi)^2\sigma^4}} \exp(-\frac{1}{2}x^T\Sigma^{-1}x) && (46)\\
&= \frac{1}{\sqrt{(2\pi)^2\sigma^4}} \exp(-\frac{1}{2}\frac{x_1^2 + x_2^2}{\sigma^2}) && (47)\\
&= \frac{1}{2\pi\sigma^2} \exp(-\frac{x_1^2 + x_2^2}{2\sigma^2}) && (48)\\
& && (49)
\end{aligned}$$

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:

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

Part III: Integration

$$\begin{aligned}
\epsilon^0[\psi^*] &= \int_{\theta=0}^{\theta=2\pi} \int_{r=r^*}^{\infty} \frac{1}{2\pi\sigma_0^2} \exp(-\frac{r^2}{2\sigma_0^2}) r dr d\theta && (50) \\
&= \frac{1}{2\pi\sigma_0^2} \int_{\theta=0}^{\theta=2\pi} \int_{r=r^*}^{\infty} \exp(-\frac{r^2}{2\sigma_0^2}) r dr d\theta && (51) \\
&= \frac{1}{2\pi\sigma_0^2} \int_{\theta=0}^{\theta=2\pi} \sigma_0^2 \exp(-\frac{r_*^2}{2\sigma_0^2}) d\theta && (52) \\
&= \exp(-\frac{r_*^2}{2\sigma_0^2}) && (53) \\
&= \exp\left(-\frac{1}{\sigma_0^2(\sigma_0^{-2} - \sigma_1^{-2})} \ln \frac{\sigma_1^2}{\sigma_0^2}\right) && (54) \\
&= \exp\left(\frac{-1}{(1 - \frac{\sigma_0^2}{\sigma_1^2})} \ln \frac{\sigma_1^2}{\sigma_0^2}\right) && (55) \\
&= \exp\left(-(1 - \frac{\sigma_0^2}{\sigma_1^2})^{-1} \ln \frac{\sigma_1^2}{\sigma_0^2}\right) && (56)
\end{aligned}$$

$$\begin{aligned}
\epsilon^1[\psi^*] &= \int_{\theta=0}^{\theta=2\pi} \int_{r=0}^{r=r^*} \frac{1}{2\pi\sigma_1^2} \exp(-\frac{r^2}{2\sigma_1^2}) r dr d\theta && (57) \\
&= 1 - \exp(-\frac{r_*^2}{2\sigma_1^2}) && (58) \\
&= 1 - \exp\left(-\frac{1}{\sigma_1^2(\sigma_0^{-2} - \sigma_1^{-2})} \ln \frac{\sigma_1^2}{\sigma_0^2}\right) && (59) \\
&= 1 - \exp\left(-\frac{1}{(\frac{\sigma_1^2}{\sigma_0^2} - 1)} \ln \frac{\sigma_1^2}{\sigma_0^2}\right) && (60) \\
&= 1 - \exp\left(-\frac{\frac{\sigma_0^2}{\sigma_1^2}}{(1 - \frac{\sigma_0^2}{\sigma_1^2})} \ln \frac{\sigma_1^2}{\sigma_0^2}\right) && (61) \\
&= 1 - \exp\left(-\frac{\sigma_0^2}{\sigma_1^2}(1 - \frac{\sigma_0^2}{\sigma_1^2})^{-1} \ln \frac{\sigma_1^2}{\sigma_0^2}\right) && (62)
\end{aligned}$$

Part IV: Combining together

$$
\begin{align}
\epsilon^* &= P(Y=0)\epsilon^0[\psi^*] + P(Y=1)\epsilon^1[\psi^*] \tag{63}\\
&= \frac{1}{2}\epsilon^0 + \frac{1}{2}\epsilon^1 \tag{64}\\
&= \frac{1}{2}\exp\left(-(1-\frac{\sigma_0^2}{\sigma_1^2})^{-1}\ln\frac{\sigma_1^2}{\sigma_0^2}\right) + \frac{1}{2} - \frac{1}{2}\exp\left(-\frac{\sigma_0^2}{\sigma_1^2}(1-\frac{\sigma_0^2}{\sigma_1^2})^{-1}\ln\frac{\sigma_1^2}{\sigma_0^2}\right) \tag{65}\\
&= \frac{1}{2}\exp\left(-(1-\frac{\sigma_0^2}{\sigma_1^2})^{-1}\ln\frac{\sigma_1^2}{\sigma_0^2}\right) + \frac{1}{2} \tag{66}\\
&- \frac{1}{2}\exp\left(-(\frac{\sigma_0^2}{\sigma_1^2}-1)(1-\frac{\sigma_0^2}{\sigma_1^2})^{-1}\ln\frac{\sigma_1^2}{\sigma_0^2}\right)\exp\left(-(1-\frac{\sigma_0^2}{\sigma_1^2})^{-1}\ln\frac{\sigma_1^2}{\sigma_0^2}\right) \tag{67}\\
&= \frac{1}{2} + \frac{1}{2}\left[1-\exp\left(-\underbrace{(\frac{\sigma_0^2}{\sigma_1^2}-1)(1-\frac{\sigma_0^2}{\sigma_1^2})^{-1})}_{=-1}\ln\frac{\sigma_1^2}{\sigma_0^2}\right)\right]\exp\left(-(1-\frac{\sigma_0^2}{\sigma_1^2})^{-1}\ln\frac{\sigma_1^2}{\sigma_0^2}\right) \tag{68}\\
&= \frac{1}{2} + \frac{1}{2}\left[1-\exp\left(\ln\frac{\sigma_1^2}{\sigma_0^2}\right)\right]\exp\left(-(1-\frac{\sigma_0^2}{\sigma_1^2})^{-1}\ln\frac{\sigma_1^2}{\sigma_0^2}\right) \tag{69}\\
&= \frac{1}{2} + \frac{1}{2}\left[1-\frac{\sigma_1^2}{\sigma_0^2}\right]\exp\left(-(1-\frac{\sigma_0^2}{\sigma_1^2})^{-1}\ln\frac{\sigma_1^2}{\sigma_0^2}\right) \tag{70}\\
&= \frac{1}{2} - \frac{1}{2}\left(\frac{\sigma_1^2}{\sigma_0^2}-1\right)\exp\left(-(1-\frac{\sigma_0^2}{\sigma_1^2})^{-1}\ln\frac{\sigma_1^2}{\sigma_0^2}\right) \tag{71}\\
& \tag{72}
\end{align}
$$

(c)

Compare the optimal classifier to the QDA classifier in Braga-Neto (2020, Example 4.3). Compute the error of the QDA classifier and compare to the Bayes error. (Given $\sigma_0^2 = 2$ and $\sigma_1^2 = 8$)[5]

Part I: Optimal Error of Example 4.3

```python
def berror_two(sig0, sig1):
    assert sig1 > sig0
    rat = sig1/sig0
    return 0.5 - 0.5*(rat-1)*np.exp(-((1-rat**-1)**-1)*np.log(rat))


pd.DataFrame({"Optimal Error": [berror_two(2, 8)]})
```

[5] For Problem 4.3(c), please assume sigma_0^2 = 2 and sigma_1^2 = 8. — Ulisses (TAMU Slack)

	Optimal Error
0	0.263765

Part II: QDA Error

Use the result in Problem 4.4 (b) and let $\hat{r}$ be the boundary of the QDA in Braga-Neto (2020, Example 4.3):

- ϵ^0 is [6]

$$
\begin{aligned}
\epsilon^0 &= \int_{\theta=0}^{\theta=2\pi} \int_{r=\hat{r}}^{\infty} \frac{1}{2\pi\hat{\sigma}_0^2} \exp(-\frac{r^2}{2\hat{\sigma}_0^2}) r dr d\theta && (73)\\
&= \exp(-\frac{\hat{r}^2}{2\hat{\sigma}_0^2}) && (74)\\
&= \exp\left(-\frac{\frac{32}{9}\ln 2}{2\cdot\frac{2}{3}}\right) && (75)\\
&\approx 0.157 && (76)
\end{aligned}
$$

- ϵ^1 is [7]

$$
\begin{aligned}
\epsilon^1 &= 1 - \exp\left(-\frac{\hat{r}^2}{2\hat{\sigma}_1^2}\right) && (77)\\
&= 1 - \exp\left(-\frac{\frac{32}{9}\ln 2}{2\cdot\frac{8}{3}}\right) && (78)\\
&\approx 0.370 && (79)
\end{aligned}
$$

Since $k_n = 0$ is assumed, the error of LDA is[8]

$$
\begin{aligned}
\epsilon_{LDA} &= \frac{1}{2}(\epsilon_{LDA}^0 + \epsilon_{LDA}^1) && (80)\\
&= \frac{1}{2}(0.157 + 0.370) && (81)\\
&= \underline{0.264} && (82)
\end{aligned}
$$

Conclusion

The QDA error is larger than the optimal error.

[6] Via WolframAlpha

[7] Via WolframAlpha

[8] Via WolframAlpha

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 (SFE ≤ 35) and high (SFE ≥ 45), excluding the middle values.

(a)

Apply the preprocessing steps in c01_matex.py to obtain a data matrix of dimensions 123(number of sample points) × 7(number of features), as described in Braga-Neto (2020, sec. 1.8.2). Define low (SFE ≤ 35) and high (SFE ≥ 45) labels for the data. Pick the first 50% of the sampe point s to be the training data and the remaining 50% to be test data[9].

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

```python
print(SFE.shape)
SFE.head(4)
```

```
(123, 8)
```

[9] 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 (TAMU Slack)

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

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

```python
print(train.shape)
train.head(4)
```

```
(61, 8)
```

```python
print(test.shape)
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.

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

Table 5: Results of T-test analysis

	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

```python
df_low = get_SFE_low(train)
df_high = get_SFE_high(train)

ttest = st.ttest_ind(df_low, df_high)

tdf = pd.DataFrame({
    "Features": df_low.keys(),
    "T-statistics": ttest.statistic,
    "P-Values": ttest.pvalue
})

tdf = tdf.sort_values(["T-statistics"], ascending=[0])
tdf
```

(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 6. Because the train data (Figure 3) is inseparable with single boundary. This implies that we need extra informative feature to make this two classes separatable.

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

```
def get_data_with_features(data, features):
    X = data[features].values
    Y = data.SFE > 45
    return X, Y

def get_loss(X, Y, clf):
    loss = sk.metrics.zero_one_loss(Y, clf.predict(X))
    return loss

X, Y = get_data_with_features(train, features)
X_test, Y_test = get_data_with_features(test, features)

clf = LDA(priors=(0.5,0.5))
clf.fit(X,Y.astype(int))
a = clf.coef_[0]
b = clf.intercept_[0];

# Error
err_train = get_loss(X,Y, clf)
err_test = get_loss(X_test, Y_test, clf)
```

```
def plot_swe_predict(ax, X, Y, clf, title=""):
    ax.scatter(X[~Y,0],X[~Y,1],c='blue',s=32,label='Low SFE')
    ax.scatter(X[Y,0],X[Y,1],c='orange',s=32,label='High SFE')
    left,right = ax.get_xlim()
    bottom,top = ax.get_ylim()
    ax.plot([left,right],[-left*a[0]/a[1]-b/a[1],-right*a[0]/a[1]-b/a[1]],'k',linewidth=2
    ax.set_title(title)
    ax.set_xlim(left,right)
    ax.set_ylim(bottom,top)
    ax.set_xlabel(var1)
    ax.set_ylabel(var2)
    ax.legend();

fig43tr, ax43tr = plt.subplots()
ax43tr = plot_swe_predict(ax43tr, X, Y, clf, title="Train error {}".format(err_train));
```

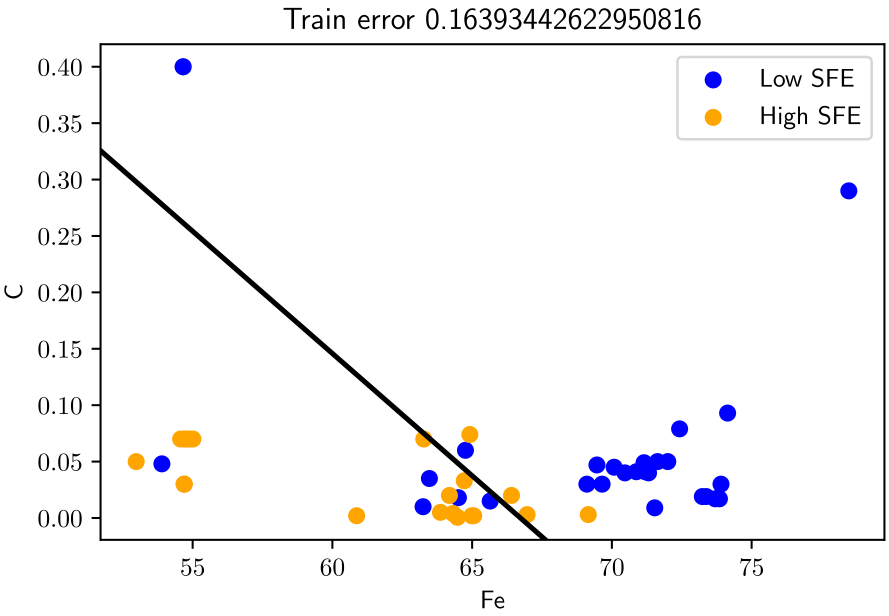

Figure 3: Train data with LDA

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

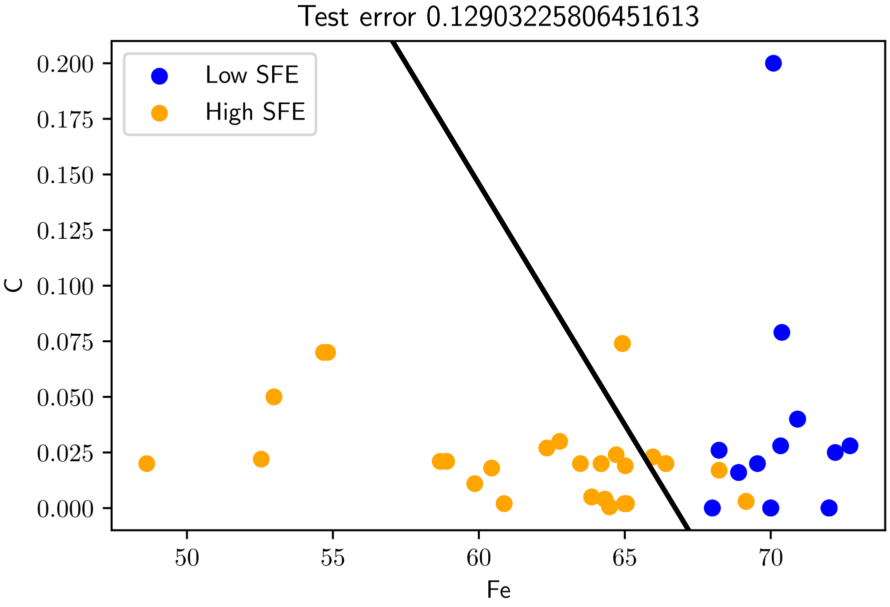

Figure 4: Test data with LDA

(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 6, 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.

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

    clf = LDA(priors=(0.5,0.5))
    clf.fit(X,Y.astype(int))

    # Error
```

Table 6: 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

```
    err_trains[j] = get_loss(X,Y, clf)
    err_tests[j] = get_loss(X_test, Y_test, clf)

pd.DataFrame({
    "Number of Features": n_features,
    "Training Error Rate": err_trains,
    "Testing Error Rate": err_tests
})
```

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.