

MSEN660 HW1

Akshay Rao

October 2018

1 Assignment1

1.1 (a)

The mean and covariance matrices are

$$\Sigma_0 = \Sigma_1 = \begin{pmatrix} 1 & \rho \\ \rho & 1 \end{pmatrix}$$

$$\mu_0 = \begin{pmatrix} 0 & 0 \end{pmatrix}$$

$$\mu_1 = \begin{pmatrix} 1 & 1 \end{pmatrix}$$

The prior probabilities $P(Y=0)$ and $P(Y=1) = .5$. The optimal classifier in the Gaussian equal variance case is a hyperplane whose error is given by:

$$\epsilon^* = \phi\left(-\frac{1}{2} * \delta\right) \quad (1)$$

Where ϕ is the cdf of the Gaussian and δ is the Mahalanobis distance between the classes:

$$\delta^2 = (\mu_1 - \mu_0)^T \Sigma^{-1} (\mu_1 - \mu_0) \quad (2)$$

We get:

$$\Sigma^{-1} = \frac{1}{\sigma^2(1 - \sigma^2)} \begin{pmatrix} 1 & -\rho \\ -\rho & 1 \end{pmatrix} \quad (3)$$

from here we use (2) and (3) to get:

$$\delta^2 = \begin{pmatrix} 1 & 1 \end{pmatrix} \frac{1}{\sigma^2(1 - \rho^2)} \begin{pmatrix} 1 & -\rho \\ -\rho & 1 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

$$\delta^2 = \begin{pmatrix} 1 & 1 \end{pmatrix} \frac{1}{\sigma^2(1 - \rho)(1 + \rho)} \begin{pmatrix} 1 - \rho \\ 1 - \rho \end{pmatrix}$$

$$\delta^2 = \begin{pmatrix} 1 & 1 \end{pmatrix} \frac{1}{\sigma^2(1 + \rho)} \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

$$\delta^2 = \frac{2}{\sigma^2(1 + \rho)}$$

And so we have the optimal classification error by (1) :

$$\epsilon^* = \phi\left(-\frac{1}{\sqrt{2}\sigma\sqrt{1+\rho}}\right) \quad (4)$$

1.2 (b)

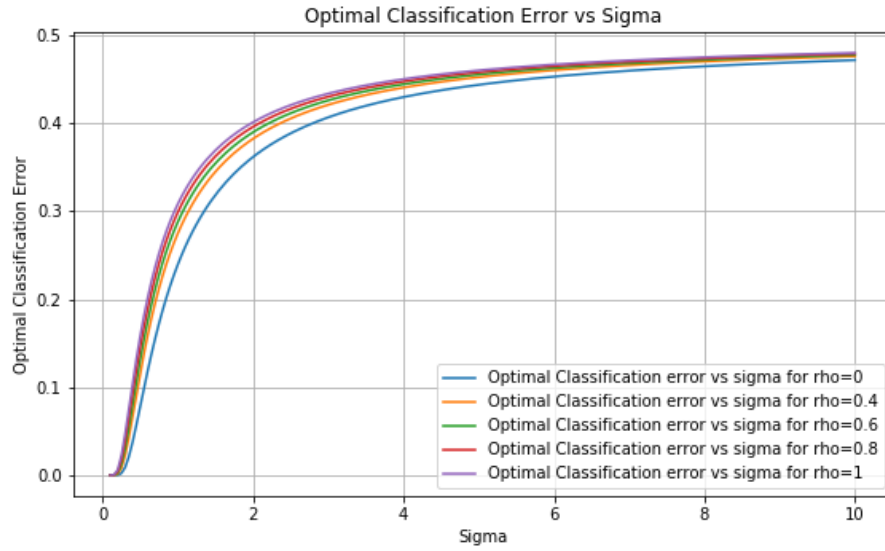


Figure 1: Optimal Classification Error for different ρ and σ

- We see that the optimal error quickly increases with σ . And then it asymptotically appears to be reaching $\frac{1}{2}$. This can be reasoned out.
- As the variance increases and reaches large values, the two classes become increasingly spread out and intermixed. This makes separation very difficult and so the optimal classification error slowly approaches the upper bound.
- The behavior of this plot would also depend on the separation of the means of the classes. Greater separation would have a smaller error and would require greater variance before the two distributions become difficult to separate.

1.3 (c)

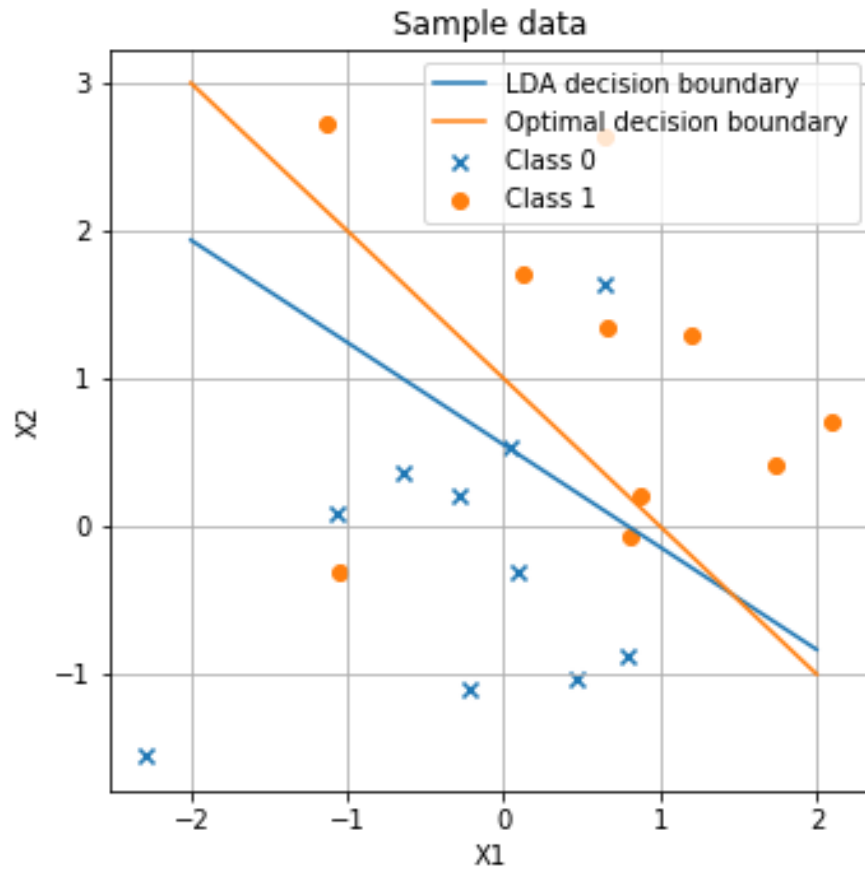


Figure 2: LDA decision boundary separating the two generated normal distributions for the classes

- We expect that the Optimal decision boundary separates the classes better. Though it is not immediately apparent from this plot.
- We also expect that as the training size increases the LDA decision boundary should approach that of the optimal classifier. This is because for the optimal classifier we know the ground truth (optimal classifier) and as we get more and more samples we can better understand the ground truth(LDA).

1.4 (d)

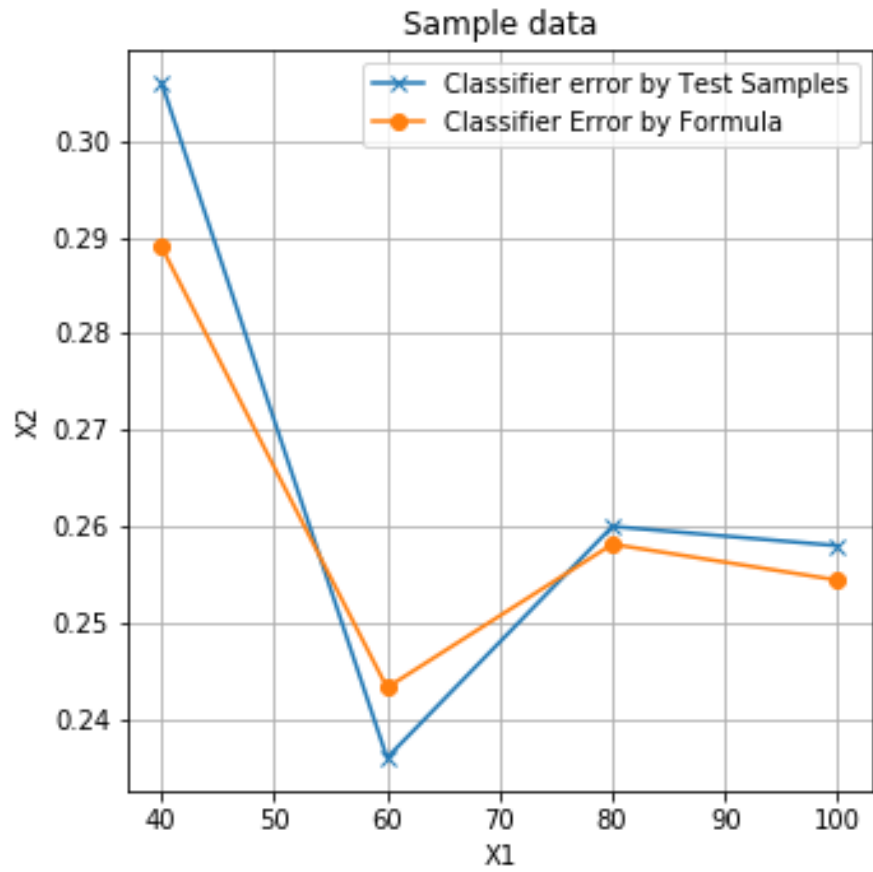


Figure 3: Classification error (y axis) vs training size (x axis)

Here we see that the classification error on the test set decreases as the training set size increases. This is to be expected. But it should be noted that the plot is very jumpy and erratic. This may have to do with the smaller training size.

2 Assignment2

2.1 (a)

Steps for pre-processing:

- We categorize the SFE into low (≤ 35) and high (≥ 45).
- Remove all predictors that do not have at least 60% non zero entries.
- Randomly split the data into training set (20%) and test set(80%).
- Resample until the training sample does not contain more than 55(%) from any one class.

| | C | N | Ni | Fe | Mn | Si | Cr | SFE |
|---|---------|--------|-------|----------|--------|--------|-------|-----|
| 0 | 0.04100 | 0.0540 | 8.10 | 70.87500 | 1.7100 | 0.3300 | 18.20 | 0.0 |
| 1 | 0.00400 | 0.0030 | 15.60 | 64.31700 | 0.0300 | 0.0200 | 17.50 | 1.0 |
| 2 | 0.07000 | 0.5400 | 16.13 | 54.67800 | 9.6400 | 0.4500 | 18.48 | 1.0 |
| 3 | 0.40000 | 0.0660 | 16.30 | 54.66000 | 9.6400 | 0.4500 | 18.48 | 0.0 |
| 4 | 0.00004 | 0.0001 | 12.00 | 71.99952 | 0.0001 | 0.0002 | 16.00 | 0.0 |

Figure 4: Sample of the training set after pre processing

2.2 (b)

- Now we use Welch's two sample t test to obtain T statistic and p value for each of the predictors. To apply this test we assume that the underlying distributions of the two classes are normal but it allows for unequal variances as opposed to student's t test.

$$t_{statistic} = \frac{\overline{X_1} - \overline{X_2}}{\sqrt{s^1/N_1 + s^2/N_2}}$$

- The predictors are then ordered in decreasing magnitude of t statistic value.
- The ranking of the predictors to be chosen depends upon the sampling for the training set. It would be prudent to run the sampling and ttest multiple times and take an average to obtain a ranking of the predictors. Although that is not performed here.

```
[Ttest_indResult(statistic=-3.6652244046944773, pvalue=0.0013486369196097205), 'Ni']
[Ttest_indResult(statistic=2.359906793878512, pvalue=0.027438300016957032), 'Fe']
[Ttest_indResult(statistic=2.1848241668704795, pvalue=0.04179478729923675), 'Si']
[Ttest_indResult(statistic=1.4943828075999175, pvalue=0.15926508688174165), 'C']
[Ttest_indResult(statistic=-1.0278205792662476, pvalue=0.31477403802244214), 'Cr']
[Ttest_indResult(statistic=0.80913241119481, pvalue=0.4267305733623522), 'Mn']
[Ttest_indResult(statistic=0.2829267844867566, pvalue=0.7800377202267232), 'N']
```

Figure 5: Sorted table of predictors with t statistics and p values after Welch's two sample t test

2.3 (c)

- The top two predictors are chosen and an LDA classifier is built using the training set.
- The LDA classifier is plotted along with the training data.

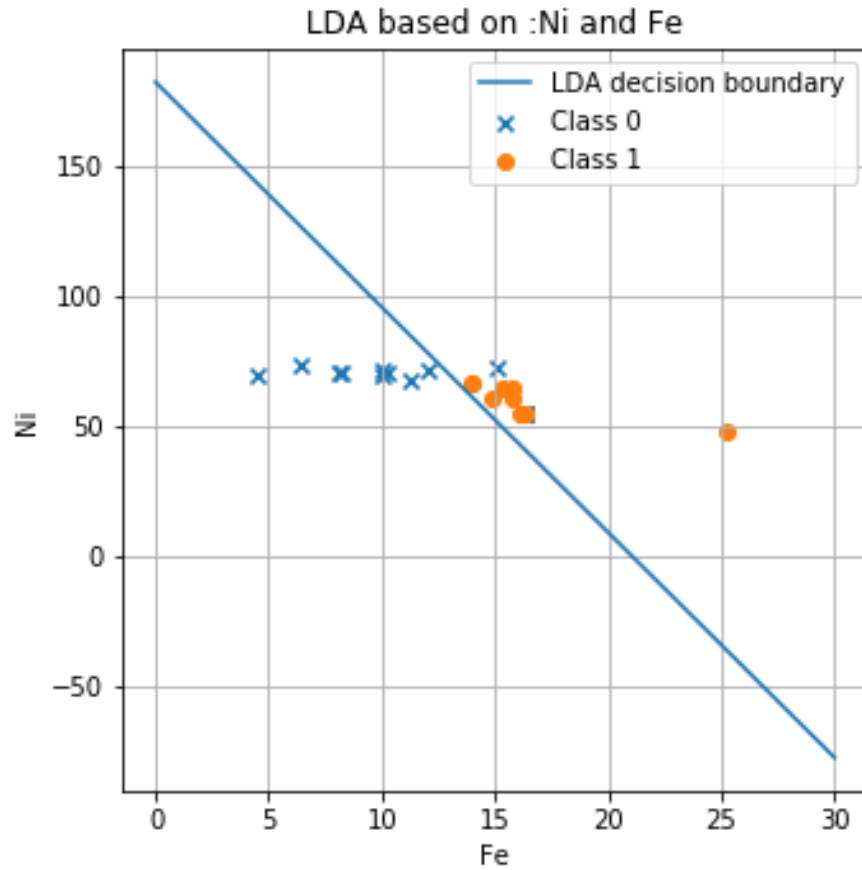


Figure 6: LDA decision boundary on two classes created using top two predictors

2.3.1 Estimated LDA error rate

- The error for this LDA classifier is estimated by predicting on the test data.
- We also construct a confusion matrix consisting of True and False Positives and Negatives

$$\begin{pmatrix} \text{True Positive} & \text{False Positive} \\ \text{False Negative} & \text{True Negative} \end{pmatrix}$$

- We can estimate the error of the LDA classifier by :

$$E = \frac{\text{False Positives} + \text{False Negatives}}{\text{Total Number of Data Points}}$$

- So for two predictors Ni and Fe we have:

| | Actual 1 | Actual 0 |
|-------------|----------|----------|
| Predicted 1 | 4 | 46 |
| Predicted 0 | 4 | 47 |

Figure 7: The confusion matrix

$$E_{2 \text{ predictors}} = \frac{46 + 4}{4 + 46 + 4 + 47}$$

$$E_{2 \text{ predictors}} = 0.4950$$

2.4 (d)

We now repeat this procedure for the top three, four and five predictors and estimate the classification error on the test set.

2.4.1 Three Predictors Ni,Fe and Si

| | Actual 1 | Actual 0 |
|-------------|----------|----------|
| Predicted 1 | 40 | 10 |
| Predicted 0 | 3 | 48 |

Figure 8: The confusion matrix

$$E_{3 \text{ predictors}} = \frac{10 + 3}{40 + 10 + 3 + 48}$$

$$E_{3 \text{ predictors}} = 0.1287$$

2.4.2 Four Predictors Ni,Fe,Si and C

| | Actual 1 | Actual 0 |
|-------------|----------|----------|
| Predicted 1 | 44 | 6 |
| Predicted 0 | 3 | 48 |

Figure 9: The confusion matrix

$$E_{4 \text{ predictors}} = \frac{6 + 3}{44 + 6 + 3 + 48}$$

$$E_{4 \text{ predictors}} = 0.0891$$

2.4.3 Four Predictors Ni,Fe,Si,C and Cr

| | Actual 1 | Actual 0 |
|-------------|----------|----------|
| Predicted 1 | 45 | 5 |
| Predicted 0 | 3 | 48 |

Figure 10: The confusion matrix

$$E_{5 \text{ predictors}} = \frac{5 + 3}{45 + 5 + 3 + 48}$$

$$E_{5 \text{ predictors}} = 0.0792$$

- We can see that as the number of predictors is increased we are able to reduce the LDA classification error.
- This shows how going into higher dimensional spaces (in this case, by incorporating more predictors) can help with separation of classes.
- It is worth noting that a decrease in classification error need not always occur. Here we are adding predictors based on the best individual predictor. This can ignore correlations between the predictors.

3 Code

The python code for all the problems is included below (as jupyter notebook pdf).

HW1_1

October 16, 2018

```
In [34]: import numpy as np
         from scipy import stats
         import matplotlib.pyplot as plt
         fig, ax = plt.subplots(figsize=[8,5])

         x=np.linspace(stats.norm.cdf(0.01),stats.norm.cdf(0.99),100)
         print(x)

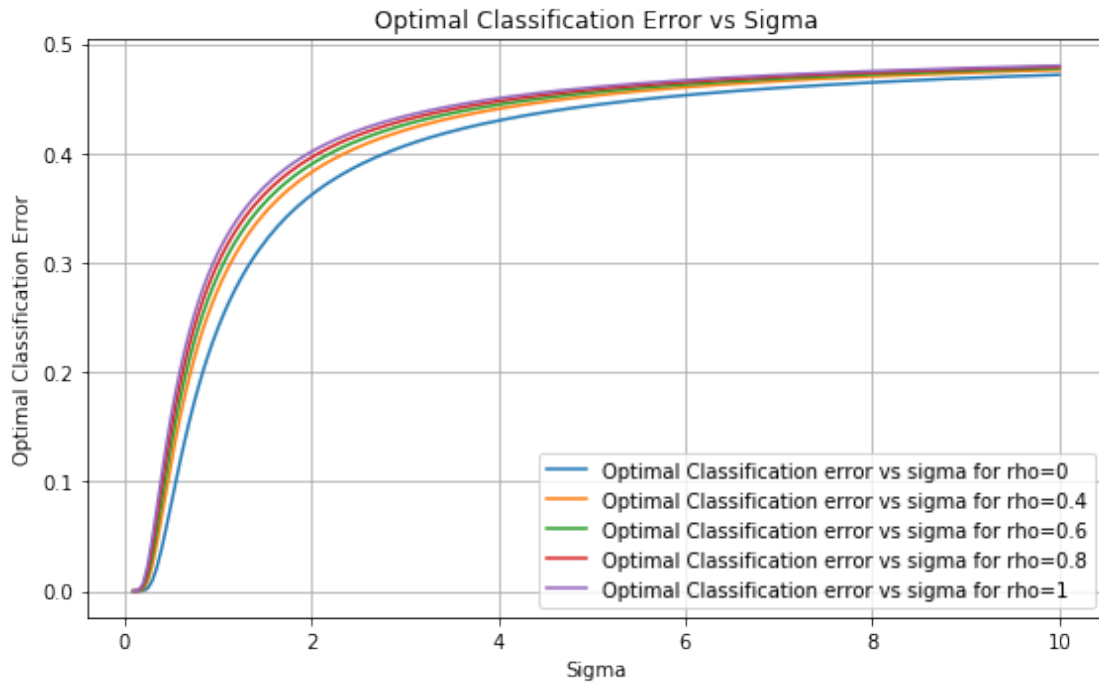
         rhoarray=list([0,0.4,0.6,0.8,1])
         sigmarr=np.linspace(.1,10,num=200)
         err=np.zeros((len(rhoarray),len(sigmarr)))
         #import pdb; pdb.set_trace()
         for i in range(0,len(rhoarray)):
             for j in range(0,len(sigmarr)):
                 err[i,j]=stats.norm.cdf(-1/(np.sqrt(2)*sigmarr[j]*np.sqrt(1+rhoarray[i])))
                 ax.plot(sigmarr,err[i,:],label='Optimal Classification error vs sigma for rho='+str(rhoarray[i]))
             ax.legend()
         plt.title('Optimal Classification Error vs Sigma')
         plt.ylabel('Optimal Classification Error')
         plt.xlabel('Sigma')
         fig.tight_layout()
         ax.legend()
         plt.grid(True)
         plt.show
         fig.savefig('hw1b.png')
```

| | | | | | |
|------------|------------|------------|------------|------------|------------|
| 0.50398936 | 0.50737242 | 0.51075549 | 0.51413856 | 0.51752162 | 0.52090469 |
| 0.52428776 | 0.52767082 | 0.53105389 | 0.53443695 | 0.53782002 | 0.54120309 |
| 0.54458615 | 0.54796922 | 0.55135229 | 0.55473535 | 0.55811842 | 0.56150149 |
| 0.56488455 | 0.56826762 | 0.57165069 | 0.57503375 | 0.57841682 | 0.58179989 |
| 0.58518295 | 0.58856602 | 0.59194909 | 0.59533215 | 0.59871522 | 0.60209829 |
| 0.60548135 | 0.60886442 | 0.61224748 | 0.61563055 | 0.61901362 | 0.62239668 |
| 0.62577975 | 0.62916282 | 0.63254588 | 0.63592895 | 0.63931202 | 0.64269508 |
| 0.64607815 | 0.64946122 | 0.65284428 | 0.65622735 | 0.65961042 | 0.66299348 |
| 0.66637655 | 0.66975962 | 0.67314268 | 0.67652575 | 0.67990881 | 0.68329188 |
| 0.68667495 | 0.69005801 | 0.69344108 | 0.69682415 | 0.70020721 | 0.70359028 |
| 0.70697335 | 0.71035641 | 0.71373948 | 0.71712255 | 0.72050561 | 0.72388868 |
| 0.72727175 | 0.73065481 | 0.73403788 | 0.73742095 | 0.74080401 | 0.74418708 |

```

0.74757014 0.75095321 0.75433628 0.75771934 0.76110241 0.76448548
0.76786854 0.77125161 0.77463468 0.77801774 0.78140081 0.78478388
0.78816694 0.79155001 0.79493308 0.79831614 0.80169921 0.80508228
0.80846534 0.81184841 0.81523147 0.81861454 0.82199761 0.82538067
0.82876374 0.83214681 0.83552987 0.83891294]

```



```

In [4]: sigma=1
        rho=0.2
        u1=np.array([0,0])
        u2=np.array([1,1])
        cov=np.array([[sigma**2,rho*sigma**2],[rho*sigma**2,sigma**2]])
        print(cov)

#create sample two guassian distributions for each mean
        x1=np.random.multivariate_normal(u1,cov,10)
        x2=np.random.multivariate_normal(u2,cov,10)
        xs=np.concatenate((x1,x2),axis=0)

#designed lda classifier
        smean1=np.mean(x1,axis=0)
        smean2=np.mean(x2,axis=0)
        scov1=np.cov(x1.T,rowvar=True)
        scov2=np.cov(x2.T,rowvar=True)
        scov=(scov1+scov2)/2

```

```

scov_inv=np.linalg.inv(scov)
a_lda=scov_inv.dot((smean2-smean1))
b_lda=-0.5*((smean2-smean1).dot(scov_inv)).dot((smean1+smean2))
x1_lda=np.linspace(-2,2,num=20)
y_lda=-a_lda[0]/a_lda[1]*x1_lda-b_lda/a_lda[1]

#optimal classifier
cov_inv=np.linalg.inv(cov)
a_opt=cov_inv.dot((u2-u1))
b_opt=-0.5*((u2-u1).dot(cov_inv)).dot((u1+u2))
x1_opt=np.linspace(-2,2,num=20)
y_opt=-a_opt[0]/a_opt[1]*x1_opt-b_opt/a_opt[1]

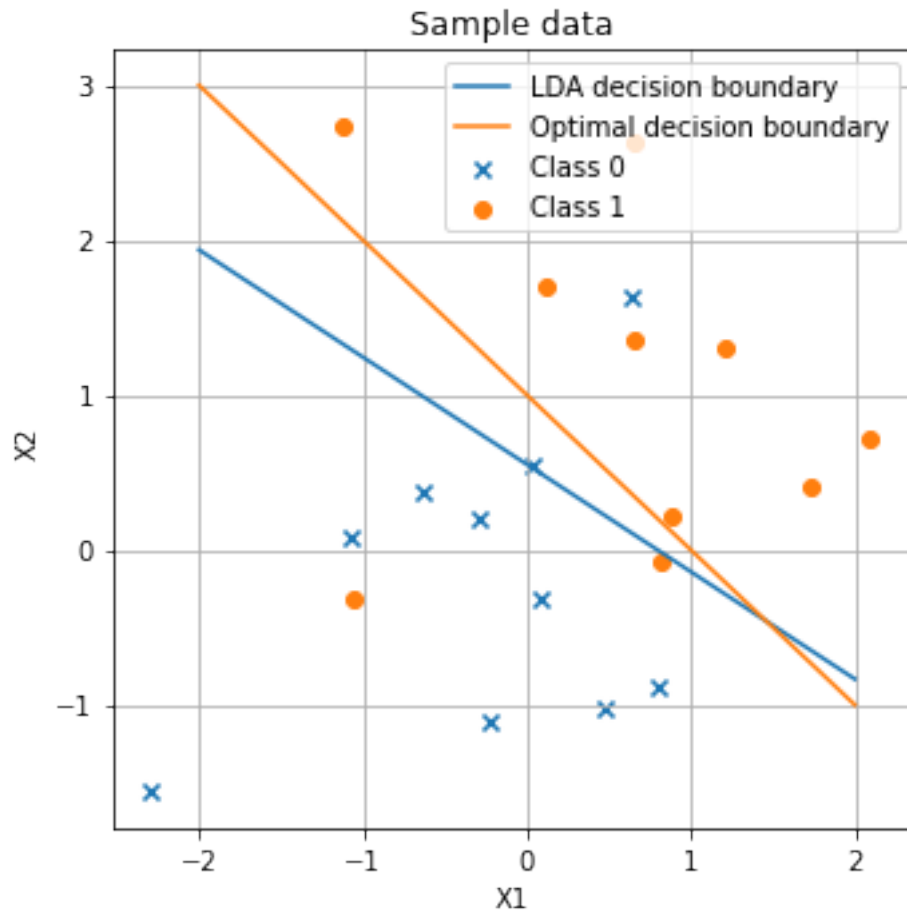
#import pdb; pdb.set_trace()
fig, ax = plt.subplots(figsize=[5,5])
plt.scatter(x1[:,0],x1[:,1],marker='x',label='Class 0')
plt.scatter(x2[:,0],x2[:,1],marker='o',label='Class 1')
plt.plot(x1_lda,y_lda,label='LDA decision boundary')
plt.plot(x1_opt,y_opt,label='Optimal decision boundary')
plt.title('Sample data')
plt.ylabel('X2')
plt.xlabel('X1')
fig.tight_layout()
ax.legend()
plt.grid(True)
plt.show
fig.savefig('hw1c.png')

```

```

[[1.  0.2]
 [0.2 1. ]]

```



In [36]: *#part d*

```
sigma=1
rho=0.2
u1=np.array([0,0])
u2=np.array([1,1])
cov=np.array([[sigma**2,rho*sigma**2],[rho*sigma**2,sigma**2]])
print(cov)

nlist=np.linspace(40,100,num=4)

test_error=np.zeros(len(nlist))
err_lda=np.zeros(len(nlist))
fig, ax = plt.subplots(figsize=[5,5])

# generate test set
x1_test=np.random.multivariate_normal(u1,cov,250)
x2_test=np.random.multivariate_normal(u2,cov,250)
```

```

for t in range(0,len(nlist)) :
    #import pdb; pdb.set_trace()
    #create sample two guassian distributions for each mean
    x1=np.random.multivariate_normal(u1,cov,int(nlist[t]/2))
    x2=np.random.multivariate_normal(u2,cov,int(nlist[t]/2))

    #train on training set
    smean1=np.mean(x1,axis=0)
    smean2=np.mean(x2,axis=0)
    scov1=np.cov(x1.T,rowvar=True)
    scov2=np.cov(x2.T,rowvar=True)
    scov=(scov1+scov2)/2
    scov_inv=np.linalg.inv(scov)
    a_lda=scov_inv.dot((smean2-smean1))
    b_lda=-0.5*((smean2-smean1).dot(scov_inv)).dot((smean1+smean2))
    err1=0

    #estimate error on test set
    for i in range(0,len(x1_test)):
        g1=a_lda.dot(x1_test[i])+b_lda
        if (g1>0):
            err1+=1
    err2=0
    for i in range(0,len(x2_test)):
        g2=a_lda.dot(x2_test[i])+b_lda
        if (g2<=0):
            err2+=1

    test_error[t]=(err1+err2)/500

    #obtaining error by formula for LDA
    err_lda[t]=0.5*(stats.norm.cdf((a_lda.dot(u1)+b_lda)/np.sqrt(a_lda.dot(scov.dot(a

# print('Test error : ',test_error)

#import pdb; pdb.set_trace()

plt.plot(nlist,test_error,marker='x',label='Classifier error by Test Samples')
plt.plot(nlist,err_lda,marker='o',label='Classifier Error by Formula')
plt.hold(True)
plt.title('Sample data')
plt.ylabel('Classification Error on Test Set')
plt.xlabel('Training sample size (n)')
fig.tight_layout()
ax.legend()
plt.grid(True)

```

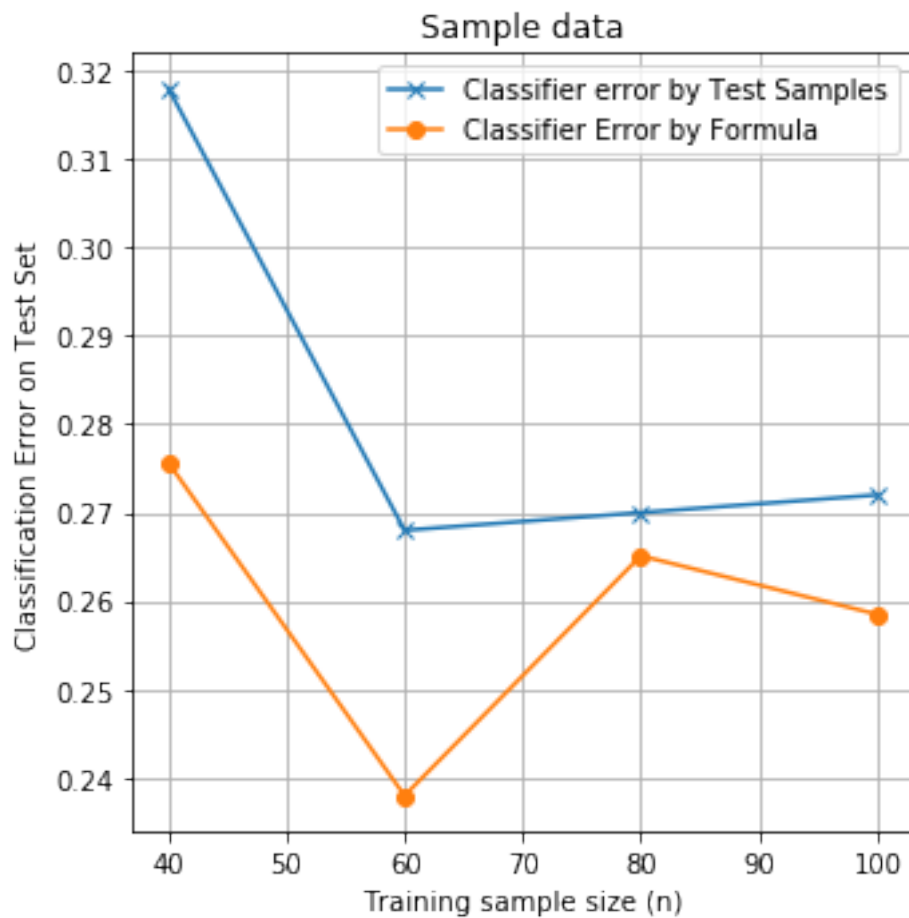
```
plt.show
fig.savefig('hw1d.png')
```

```
[[1.  0.2]
 [0.2 1.  ]]
```

C:\Users\aksha\Anaconda3\lib\site-packages\ipykernel_launcher.py:60: MatplotlibDeprecationWarning: Future behavior will be consistent with the long-time default: plot commands add elements without first clearing the Axes and/or Figure.

C:\Users\aksha\Anaconda3\lib\site-packages\matplotlib__init__.py:911: MatplotlibDeprecationWarning: mplDeprecation)

C:\Users\aksha\Anaconda3\lib\site-packages\matplotlib\rcsetup.py:156: MatplotlibDeprecationWarning: mplDeprecation)



Hw1_2

October 16, 2018

```
In [24]: import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

#import excel table
SFE_data=pd.read_excel('SFE_Dataset.xlsx')
col=SFE_data.columns
print(col)

#preprocessing

#remove columns with zero elements

nonzero_ratio=np.zeros(len(col))
counter=0
for col_i in col:
    n_zeros=0
    #print('column : ',col_i)
    #import pdb; pdb.set_trace()
    for j in range(0,len(SFE_data[col_i])):
        if(SFE_data[col_i][j]==0):
            n_zeros+=1
    nonzero_ratio[counter]=1-n_zeros/len(SFE_data[col_i])
    counter+=1
print('non zero ratio : ',nonzero_ratio)

drop_cols=[]
for i in range(0,len(SFE_data.columns)-1):
    if (nonzero_ratio[i]<.6):
        #print('non zero ratio',nonzero_ratio[i])
        drop_cols.append(col[i])

SFE_data_cleancols=SFE_data.drop(drop_cols,axis=1)
new_col=SFE_data_cleancols.columns
print(len(new_col),len(col))
#import pdb; pdb.set_trace()
```



```

Index(['C', 'N', 'P', 'S', 'V', 'Ni', 'Nb', 'Al', 'Ti', 'Fe', 'Hf', 'Mo', 'Mn',
      'Co', 'Si', 'Cr', 'Cu', 'SFE'],
      dtype='object')
non zero ratio : [0.89006342 0.60887949 0.36997886 0.40169133 0.00422833 0.75052854
 0.01268499 0.09513742 0.01479915 1.          0.00211416 0.39957717
 0.80338266 0.00634249 0.60887949 0.8372093  0.05496829 1.          ]
8 18

```

In [25]: *#SFE into high and low, remove rows*

```

#high and low SFE
SFE=SFE_data_cleancols['SFE']
non_SFE=[]
for i in range(0,len(SFE)):
    if(SFE[i]<=35):
        SFE_data_cleancols['SFE'][i]=0
    elif(SFE[i]>=45):
        SFE_data_cleancols['SFE'][i]=1
    elif(SFE[i]<45 and SFE[i]>35):
        non_SFE.append(i)
SFE_data_cleanrows=SFE_data_cleancols.drop(non_SFE)
SFE=SFE_data_cleanrows['SFE']
#print(SFE)
#SFE_data_cleanrows.head
#SFE_data_cleancols.head
#import pdb; pdb.set_trace()

# for resetting the index after removing the columns or rows
SFE_data_cleanrows=SFE_data_cleanrows.reset_index(drop=True)

[m,n]=SFE_data_cleanrows.shape
col=SFE_data_cleanrows.columns

#clean rows that have any zeros
zer=[]
ncol=len(col)
for i in range(0,m):
    #import pdb; pdb.set_trace()
    n_zeros=0
    for j in range (0,ncol-1):
        if(SFE_data_cleanrows.at[i, col[j]]==0):
            n_zeros+=1
    if(n_zeros>0):
        zer.append(i)

#import pdb; pdb.set_trace()

```

```

In [26]: SFE_data_cleanrows2=SFE_data_cleanrows.drop(zero)
         # for resetting the index after removing the columns or rows
         SFE_data_cleanrows2=SFE_data_cleanrows2.reset_index(drop=True)

In [27]: # randomly sample into training data and then reject data with over 55% of any one
         [m,n]=SFE_data_cleanrows2.shape
         import random
         import time
         random.seed(time.time())
         okay_sample_flag=0
         while(okay_sample_flag==0):
             train_set=SFE_data_cleanrows2.sample(frac=0.2,random_state=random.randint(1,100))
             test_set=SFE_data_cleanrows2.drop(train_set.index)

             train_set=train_set.reset_index(drop=True)
             test_set=test_set.reset_index(drop=True)

             #remove samples for more than 55%
             n_ones=0
             for i in range(0,len(train_set)):
                 if (train_set.at[i,'SFE']==1):
                     n_ones+=1
             if(n_ones/len(train_set)>.45 and n_ones/len(train_set)<.55):
                 okay_sample_flag+=1
         # import pdb; pdb.set_trace()

In [36]: train_set.head()

```

| | C | N | Ni | Fe | Mn | Si | Cr | SFE |
|---|---------|--------|-------|----------|--------|--------|-------|-----|
| 0 | 0.04100 | 0.0540 | 8.10 | 70.87500 | 1.7100 | 0.3300 | 18.20 | 0.0 |
| 1 | 0.00400 | 0.0030 | 15.60 | 64.31700 | 0.0300 | 0.0200 | 17.50 | 1.0 |
| 2 | 0.07000 | 0.5400 | 16.13 | 54.67800 | 9.6400 | 0.4500 | 18.48 | 1.0 |
| 3 | 0.40000 | 0.0660 | 16.30 | 54.66000 | 9.6400 | 0.4500 | 18.48 | 0.0 |
| 4 | 0.00004 | 0.0001 | 12.00 | 71.99952 | 0.0001 | 0.0002 | 16.00 | 0.0 |

```

Out[36]:

In [28]: from scipy import stats

         col=train_set.columns

         ttest=[]
         for i in range (0,len(col)-1):
             ttest.append([stats.ttest_ind(train_set[train_set['SFE']==0][col[i]], train_set[t

In [29]: ttest_sorted=sorted(ttest,key=lambda x:x[:,0][1])
         print(ttest_sorted[0][1])
         col1=ttest_sorted[0][1]
         col2=ttest_sorted[1][1]
         col3=ttest_sorted[2][1]
         col4=ttest_sorted[3][1]
         col5=ttest_sorted[4][1]

```

Ni

```
In [50]: for i in range(0, len(ttest_sorted)):
         print(ttest_sorted[i][0:3])
```

```
[Ttest_indResult(statistic=-3.6652244046944773, pvalue=0.0013486369196097205), 'Ni']
[Ttest_indResult(statistic=2.359906793878512, pvalue=0.027438300016957032), 'Fe']
[Ttest_indResult(statistic=2.1848241668704795, pvalue=0.04179478729923675), 'Si']
[Ttest_indResult(statistic=1.4943828075999175, pvalue=0.15926508688174165), 'C']
[Ttest_indResult(statistic=-1.0278205792662476, pvalue=0.31477403802244214), 'Cr']
[Ttest_indResult(statistic=0.80913241119481, pvalue=0.4267305733623522), 'Mn']
[Ttest_indResult(statistic=0.2829267844867566, pvalue=0.7800377202267232), 'N']
```

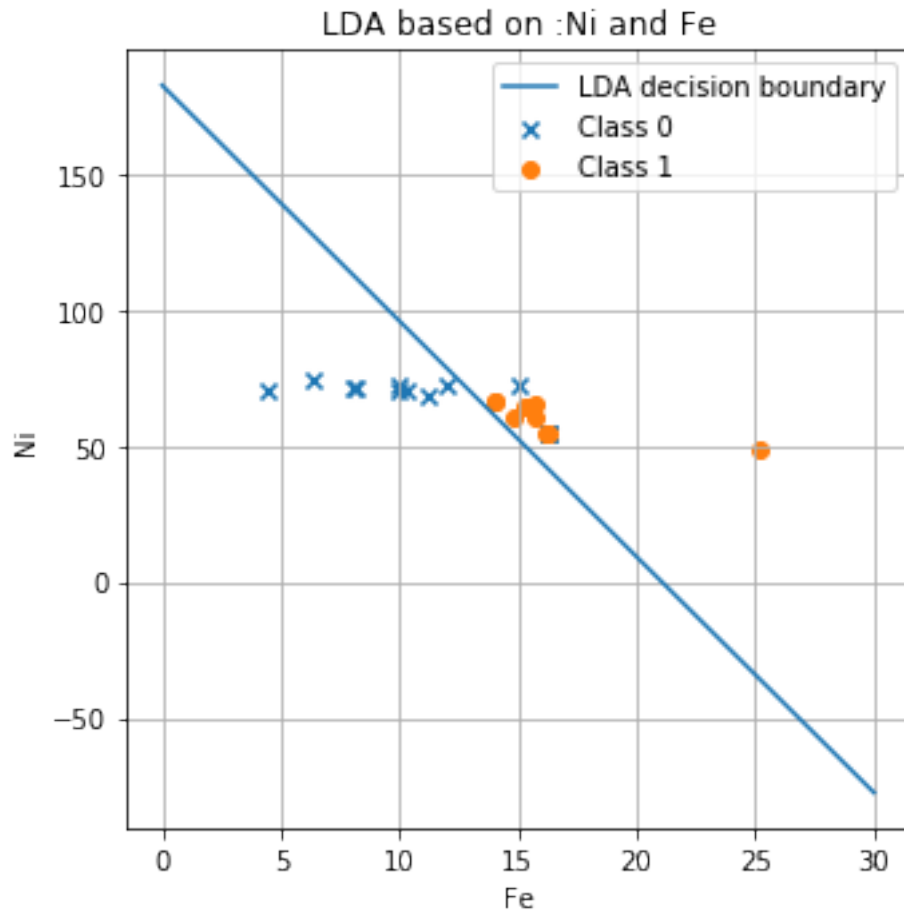
```
In [121]: train_set_0=train_set[train_set['SFE']==0]
         train_set_1=train_set[train_set['SFE']==1]

         train_set_0=train_set_0.reset_index(drop=True)
         train_set_1=train_set_1.reset_index(drop=True)

         smean1=np.array([float(train_set_0.mean(0)[col1]),float(train_set_0.mean(0)[col2])])
         smean2=np.array([float(train_set_1.mean(0)[col1]),float(train_set_1.mean(0)[col2])])
         cov=train_set.cov()
         #import pdb; pdb.set_trace()

         scov=np.zeros((2,2))
         scov=[[float(cov[col1][col1]),float(cov[col1][col2])],[float(cov[col2][col1]),float(cov[col2][col2])]
         scov_inv=np.linalg.inv(scov)
         a_lda=scov_inv.dot((smean2-smean1))
         b_lda=-0.5*((smean2-smean1).dot(scov_inv)).dot((smean1+smean2))
         x1_lda=np.linspace(0,30,num=20)
         y_lda=-a_lda[0]/a_lda[1]*x1_lda-b_lda/a_lda[1]
```

```
In [74]: fig, ax = plt.subplots(figsize=[5,5])
         plt.scatter(train_set_0[col1],train_set_0[col2],marker='x',label='Class 0')
         plt.scatter(train_set_1[col1],train_set_1[col2],marker='o',label='Class 1')
         plt.plot(x1_lda,y_lda,label='LDA decision boundary')
         plt.title('LDA based on :'+ col1 +' and '+ col2)
         plt.ylabel(col1)
         plt.xlabel(col2)
         fig.tight_layout()
         ax.legend()
         plt.grid(True)
         plt.show
         fig.savefig('hw2c.png')
```



```
In [122]: #estimating error on test set
test_set_0=test_set[test_set['SFE']==0]
test_set_1=test_set[test_set['SFE']==1]

test_set_0=test_set_0.reset_index(drop=True)
test_set_1=test_set_1.reset_index(drop=True)

errtest_set_0=np.zeros((len(test_set_0),2))
errtest_set_1=np.zeros((len(test_set_1),2))
for i in range (0,len(test_set_0)):
    errtest_set_0[i][0]=test_set_0[col1][i]
    errtest_set_0[i][1]=test_set_0[col2][i]
for i in range (0,len(test_set_1)):
    errtest_set_1[i][0]=test_set_0[col1][i]
    errtest_set_1[i][1]=test_set_0[col2][i]
    #[test_set[test_set['SFE']==0][col1],test_set[test_set['SFE']==0][col2]]
    #test_set_1[i]=[test_set[test_set['SFE']==1][col1],test_set[test_set['SFE']==1][
#import pdb; pdb.set_trace()
```

```

err1=0
TP=0
FP=0
TN=0
FN=0
g1=0
g2=0
for i in range(0,len(errtest_set_0)):
    g1=a_lda.dot(errtest_set_0[i])+b_lda
    if (g1>0):
        err1+=1
        FN+=1
    if (g1<=0):
        TN+=1
err2=0
for i in range(0,len(errtest_set_1)):
    g2=a_lda.dot(errtest_set_1[i])+b_lda
    if (g2<=0):
        err2+=1
        FP+=1

    if (g2>0):
        TP+=1
err=(err1+err2)/(len(errtest_set_0)+len(errtest_set_1))
conf_mat=np.array(([TP,FP],[FN,TN]))
print(err)

```

0.49504950495049505

```

In [105]: err
          print(conf_mat)

```

```

[[ 4 46]
 [ 4 47]]

```

```

In [107]: pd.DataFrame(conf_mat,[' Predicted 1','Predicted 0'],['Actual 1','Actual 0'])

```

```

Out[107]:
          Actual 1  Actual 0
Predicted 1         4         46
Predicted 0         4         47

```

```

In [119]: #now repeat for the top 3,4 and 5 predictors
          #first we do top 3 predictors

```

```

train_set_0=train_set[train_set['SFE']==0]

```

```

train_set_1=train_set[train_set['SFE']==1]

train_set_0=train_set_0.reset_index(drop=True)
train_set_1=train_set_1.reset_index(drop=True)

#clean train set to only contain columns of our interest
train_set_clean=train_set[['col1,col2,col3]]
train_set_0_clean=train_set_0[['col1,col2,col3]]
train_set_1_clean=train_set_1[['col1,col2,col3]]

smean1=np.array(train_set_0_clean.mean(0))
smean2=np.array(train_set_1_clean.mean(0))
cov=train_set_clean.cov()
#import pdb; pdb.set_trace()
cov_inv=np.linalg.inv(cov)
a_lda=cov_inv.dot((smean2-smean1))
b_lda=-0.5*((smean2-smean1).dot(cov_inv)).dot((smean1+smean2))
x1_lda=np.linspace(0,40,num=20)
y_lda=-a_lda[0]/a_lda[1]*x1_lda-b_lda/a_lda[1]

```

In [120]: *#estimating error on test set*

```

test_set_0=test_set[test_set['SFE']==0]
test_set_1=test_set[test_set['SFE']==1]

test_set_0=test_set_0.reset_index(drop=True)
test_set_1=test_set_1.reset_index(drop=True)

#consider only predictors of our importance
test_set_clean=test_set[['col1,col2,col3]]
test_set_0_clean=test_set_0[['col1,col2,col3]]
test_set_1_clean=test_set_1[['col1,col2,col3]]

errtest_set_0=np.array(test_set_0_clean)
errtest_set_1=np.array(test_set_1_clean)

#import pdb; pdb.set_trace()
err1=0
TP=0
FP=0
TN=0
FN=0
g1=0
g2=0
for i in range(0,len(errtest_set_0)):
    g1=a_lda.dot(errtest_set_0[i])+b_lda

```

```

        if (g1>0):
            err1+=1
            FN+=1
        if (g1<=0):
            TN+=1
err2=0
for i in range(0,len(errtest_set_1)):
    g2=a_lda.dot(errtest_set_1[i])+b_lda
    if (g2<=0):
        err2+=1
        FP+=1

    if (g2>0):
        TP+=1
err=(err1+err2)/(len(errtest_set_0)+len(errtest_set_1))
print(err)
conf_mat=np.array(([TP,FP],[FN,TN]))

```

0.12871287128712872

```
In [110]: pd.DataFrame(conf_mat,[' Predicted 1','Predicted 0'],['Actual 1','Actual 0'])
```

```
Out[110]:
```

| | Actual 1 | Actual 0 |
|-------------|----------|----------|
| Predicted 1 | 40 | 10 |
| Predicted 0 | 3 | 48 |

```
In [117]: #now we do for top 4 predictors
```

```

train_set_0=train_set[train_set['SFE']==0]
train_set_1=train_set[train_set['SFE']==1]

train_set_0=train_set_0.reset_index(drop=True)
train_set_1=train_set_1.reset_index(drop=True)

#clean train set to only contain columns of our interest
train_set_clean=train_set[[col1,col2,col3,col4]]
train_set_0_clean=train_set_0[[col1,col2,col3,col4]]
train_set_1_clean=train_set_1[[col1,col2,col3,col4]]

smean1=np.array(train_set_0_clean.mean(0))
smean2=np.array(train_set_1_clean.mean(0))
cov=train_set_clean.cov()
#import pdb; pdb.set_trace()
cov_inv=np.linalg.inv(cov)
a_lda=cov_inv.dot((smean2-smean1))
b_lda=-0.5*((smean2-smean1).dot(cov_inv)).dot((smean1+smean2))

```

```

x1_lda=np.linspace(0,40,num=20)
y_lda=-a_lda[0]/a_lda[1]*x1_lda-b_lda/a_lda[1]

In [118]: #estimating error on test set
test_set_0=test_set[test_set['SFE']==0]
test_set_1=test_set[test_set['SFE']==1]

test_set_0=test_set_0.reset_index(drop=True)
test_set_1=test_set_1.reset_index(drop=True)

#consider only predictors of our importance
test_set_clean=test_set[[col1,col2,col3,col4]]
test_set_0_clean=test_set_0[[col1,col2,col3,col4]]
test_set_1_clean=test_set_1[[col1,col2,col3,col4]]

errtest_set_0=np.array(test_set_0_clean)
errtest_set_1=np.array(test_set_1_clean)

#import pdb; pdb.set_trace()
err1=0
TP=0
FP=0
TN=0
FN=0
g1=0
g2=0
for i in range(0,len(errtest_set_0)):
    g1=a_lda.dot(errtest_set_0[i])+b_lda
    if (g1>0):
        err1+=1
        FN+=1
    if (g1<=0):
        TN+=1
err2=0
for i in range(0,len(errtest_set_1)):
    g2=a_lda.dot(errtest_set_1[i])+b_lda
    if (g2<=0):
        err2+=1
        FP+=1

    if (g2>0):
        TP+=1
err=(err1+err2)/(len(errtest_set_0)+len(errtest_set_1))
conf_mat=np.array(( [TP,FP] , [FN,TN] ))
print(err)

```

0.0891089108910891


```
In [113]: pd.DataFrame(conf_mat,[' Predicted 1','Predicted 0'],['Actual 1','Actual 0'])
```

```
Out[113]:
```

| | Actual 1 | Actual 0 |
|-------------|----------|----------|
| Predicted 1 | 44 | 6 |
| Predicted 0 | 3 | 48 |

```
In [114]: #now we do for top 5 predictors
```

```
train_set_0=train_set[train_set['SFE']==0]
train_set_1=train_set[train_set['SFE']==1]

train_set_0=train_set_0.reset_index(drop=True)
train_set_1=train_set_1.reset_index(drop=True)

#clean train set to only contain columns of our interest
train_set_clean=train_set[['col1,col2,col3,col4,col5]]
train_set_0_clean=train_set_0[['col1,col2,col3,col4,col5]]
train_set_1_clean=train_set_1[['col1,col2,col3,col4,col5]]

smean1=np.array(train_set_0_clean.mean(0))
smean2=np.array(train_set_1_clean.mean(0))
cov=train_set_clean.cov()
#import pdb; pdb.set_trace()
cov_inv=np.linalg.inv(cov)
a_lda=cov_inv.dot((smean2-smean1))
b_lda=-0.5*((smean2-smean1).dot(cov_inv)).dot((smean1+smean2))
x1_lda=np.linspace(0,40,num=20)
y_lda=-a_lda[0]/a_lda[1]*x1_lda-b_lda/a_lda[1]
```

```
In [115]: #estimating error on test set
```

```
test_set_0=test_set[test_set['SFE']==0]
test_set_1=test_set[test_set['SFE']==1]

test_set_0=test_set_0.reset_index(drop=True)
test_set_1=test_set_1.reset_index(drop=True)

#consider only predictors of our importance
test_set_clean=test_set[['col1,col2,col3,col4,col5]]
test_set_0_clean=test_set_0[['col1,col2,col3,col4,col5]]
test_set_1_clean=test_set_1[['col1,col2,col3,col4,col5]]

errtest_set_0=np.array(test_set_0_clean)
errtest_set_1=np.array(test_set_1_clean)
```

```

#import pdb; pdb.set_trace()
err1=0
TP=0
FP=0
TN=0
FN=0
g1=0
g2=0
for i in range(0,len(errtest_set_0)):
    g1=a_lda.dot(errtest_set_0[i])+b_lda
    if (g1>0):
        err1+=1
        FN+=1
    if (g1<=0):
        TN+=1
err2=0
for i in range(0,len(errtest_set_1)):
    g2=a_lda.dot(errtest_set_1[i])+b_lda
    if (g2<=0):
        err2+=1
        FP+=1

    if (g2>0):
        TP+=1
err=(err1+err2)/(len(errtest_set_0)+len(errtest_set_1))
conf_mat=np.array(([TP,FP],[FN,TN]))
print(err)

```

0.07920792079207921

In [116]: pd.DataFrame(conf_mat,[' Predicted 1','Predicted 0'],['Actual 1','Actual 0'])

Out[116]:

| | Actual 1 | Actual 0 |
|-------------|----------|----------|
| Predicted 1 | 45 | 5 |
| Predicted 0 | 3 | 48 |