HW2_classifier

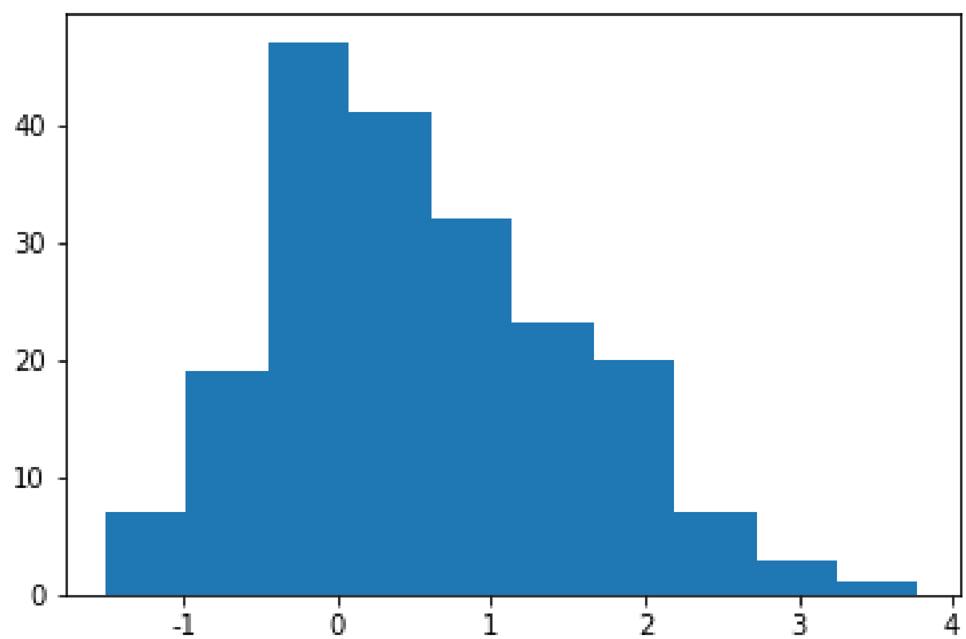
February 2, 2020

```
In [152]: import numpy as np
          import random
          import matplotlib.pyplot as plt
          import pandas as pd
          import seaborn as sns
```

0.1 PROBLEM 1: the bayes classifier

```
In [148]: random.seed(2020)
          #stimulate dataset
          X1 = np.array([random.uniform(-1,1) for i in range(200)])
          X2 = np.array([random.uniform(-1,1) for i in range(200)])
          e = np.array([np.random.normal(loc=0.0, scale=0.5, size=None) for i in range(200)])
          Y = X1 + X2 + X1**2 + X2**2 + e
```

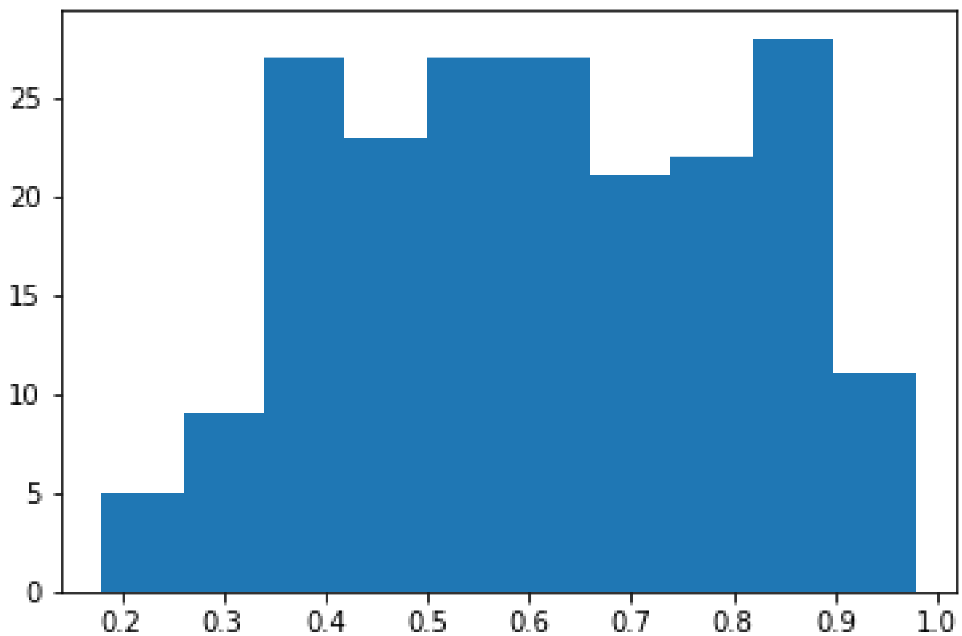
```
In [7]: plt.hist(Y)
        plt.show()
```



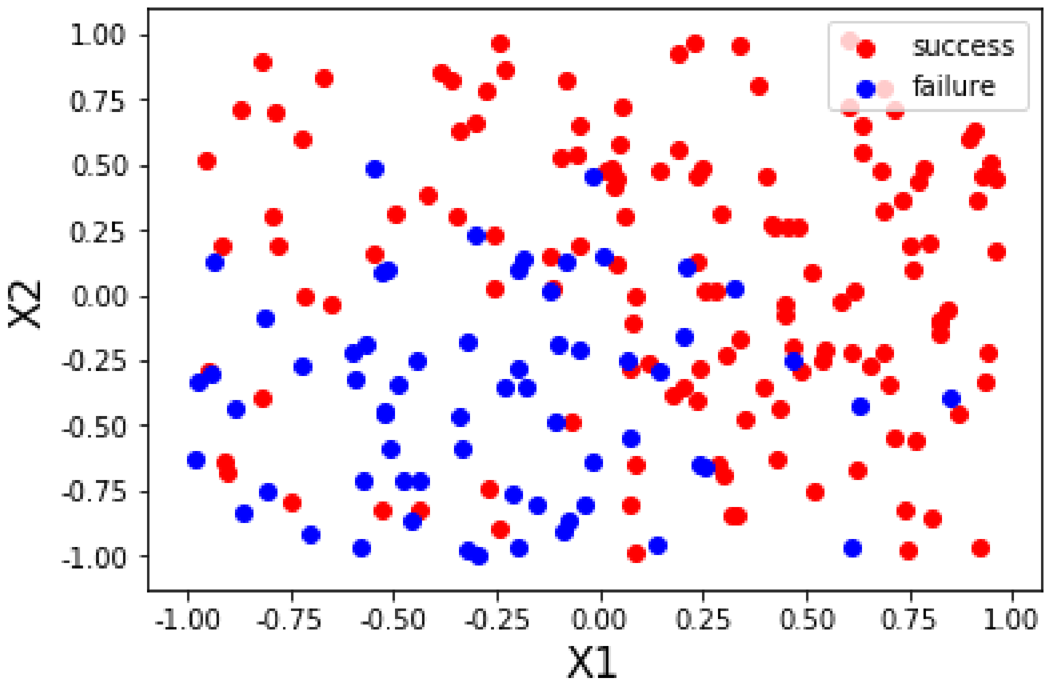
In [12]:
```python
# from log-odds to probability
def from_logodd_to_probability(y):
    return 1/(1/np.exp(y) + 1)
```

In [14]:
```python
Y_log = list(map(from_logodd_to_probability, Y))
```

In [15]:
```python
plt.hist(Y_log)
plt.show()
```

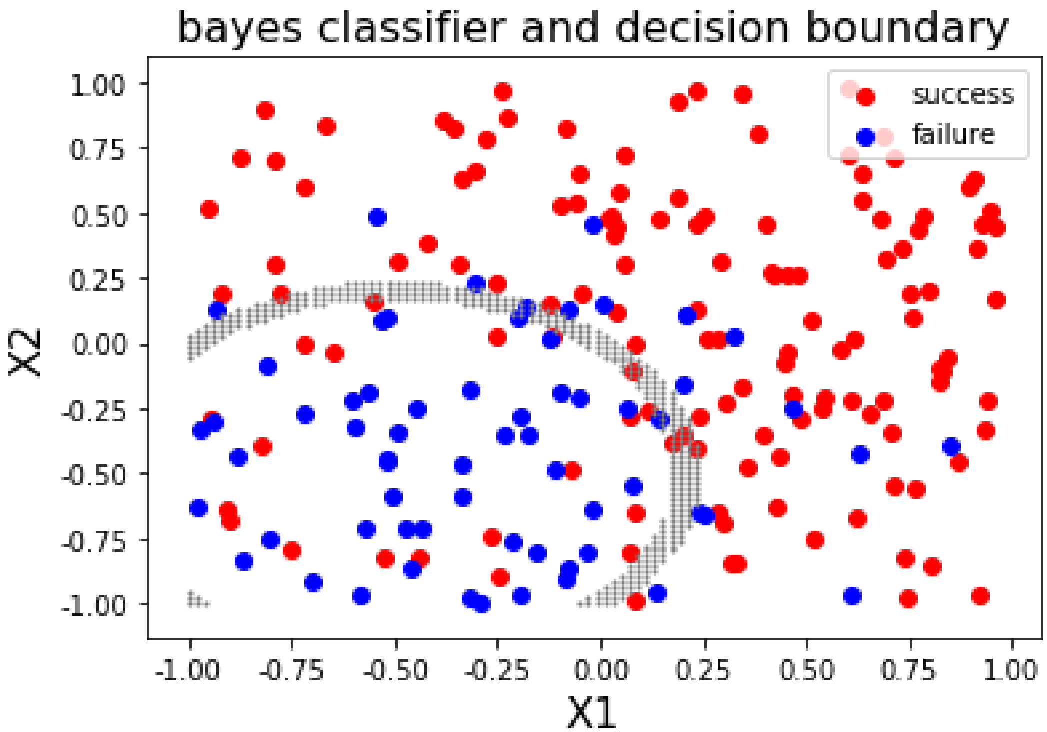


In [24]:
```python
plt.scatter(X1[np.array(Y_log)>0.5],\
            X2[np.array(Y_log)>0.5],c='r',label = 'success')
plt.scatter(X1[np.array(Y_log)<=0.5],\
            X2[np.array(Y_log)<=0.5],c='b',label = 'failure')
plt.xlabel('X1',size = 16)
plt.ylabel('X2',size = 16)
plt.legend()
plt.show()
```




```
In [149]: x1 = np.linspace(-1,1,100)
          x2 = np.linspace(-1,1,100)
          I = np.array([x1]*len(x1)).reshape((len(x1),len(x1)))
          J = I.T

In [150]: land = I+J+I**2+J**2

In [151]: #boundary
          plt.scatter(X1[np.array(Y_log)>0.5],\
                      X2[np.array(Y_log)>0.5],c='r',label = 'success')
          plt.scatter(X1[np.array(Y_log)<=0.5],\
                      X2[np.array(Y_log)<=0.5],c='b',label = 'failure')

          # when the pr = 0.5, Y = 0,
          #we can get the boundard formed by a set of pairs of (X1, X2)
          # AS I sample 10000 points on X1, X2 landscape,
          #the exactly 0 is a little hard to find.
          # So I set the range(-0.05,0.05)

          for i in range(len(x1)):
              for j in range(len(x2)):
                  if abs(land[i,j]) < 0.05:
                      plt.scatter(x1[i],x2[j],s=1,c='gray')
          plt.xlabel('X1',size = 16)
```

```
plt.ylabel('X2',size = 16)
plt.title('bayes classifier and decision boundary',size = 16)
plt.legend()
plt.show()
```



0.2 PROBLEM 2:

```
In [65]: from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
         from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis as QDA
```

```
In [155]: N = 1000
          indice = list(range(N))
          lda_train_error = []
          qda_train_error = []
          lda_test_error = []
          qda_test_error = []
          for time in range(1000):
              random.seed(time)
              #stimulate dataset
              X1 = np.array([random.uniform(-1,1) for i in range(1000)])
              X2 = np.array([random.uniform(-1,1) for i in range(1000)])
              e = np.array([np.random.normal(loc=0.0, scale=1, size=None) for i in range(1000)]
              Y = X1 + X2 + e#simulate Y
```

```
    Y = (Y>=0)
    split_point = int(len(X1)*0.7)
    indice = np.random.permutation(indice)
    train_x = np.array([X1[indice[:split_point]], X2[indice[:split_point]]]).T
    train_y = Y[indice[:split_point]]
    test_x = np.array([X1[indice[split_point:]], X2[indice[split_point:]]]).T
    test_y = Y[indice[split_point:]]

    # LDA training and testing:
    clf = LDA()
    clf.fit(train_x,train_y)
    y_predict = clf.predict(test_x)
    y_train_hat = clf.predict(train_x)
    lda_train_error.append\
    (sum(np.ones(len(y_train_hat))[y_train_hat!=train_y])/len(train_y))
    lda_test_error.append\
    (sum(np.ones(len(test_y))[y_predict!=test_y])/len(test_y))

    #QDA training and testing:
    clf = QDA()
    clf.fit(train_x,train_y)
    y_predict = clf.predict(test_x)
    y_train_hat = clf.predict(train_x)
    qda_train_error.append\
    (sum(np.ones(len(y_train_hat))[y_train_hat!=train_y])/len(train_y))
    qda_test_error.append\
    (sum(np.ones(len(test_y))[y_predict!=test_y])/len(test_y))
```

0.2.1 table of LDA and QDA error rate on training/testing set

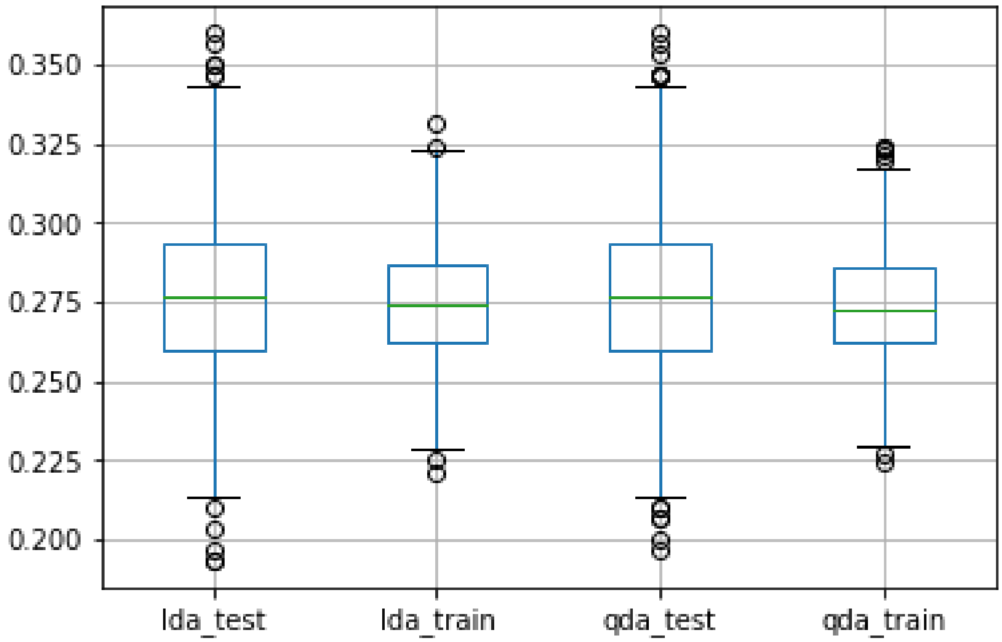
```
In [156]: df_result = pd.DataFrame({'lda_train':lda_train_error,\
                                     'lda_test':lda_test_error,'qda_train':qda_train_error,\
                                    'qda_test':qda_test_error})
          df_result.describe()
```

Out[156]:

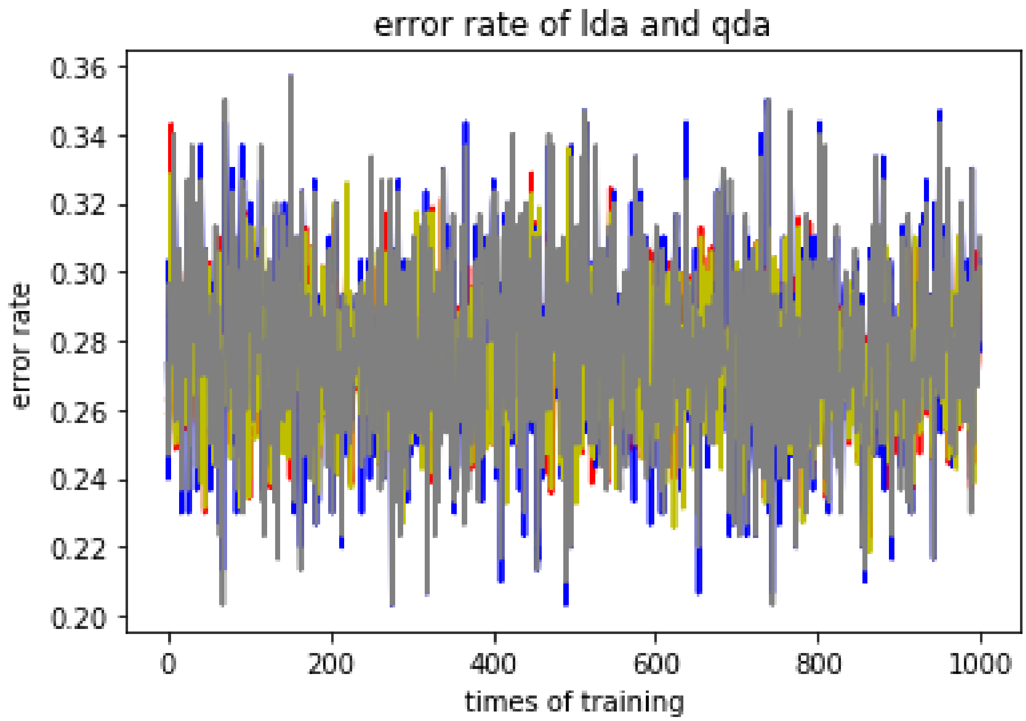
	lda_test	lda_train	qda_test	qda_train
count	1000.000000	1000.000000	1000.000000	1000.000000
mean	0.276400	0.274521	0.276987	0.273737
std	0.026020	0.017094	0.025937	0.016985
min	0.193333	0.221429	0.196667	0.224286
25%	0.260000	0.262857	0.260000	0.262857
50%	0.276667	0.274286	0.276667	0.272857
75%	0.293333	0.287143	0.293333	0.285714
max	0.360000	0.331429	0.360000	0.324286

0.2.2 graph of boxplot of LDA and QDA error rate on training/testing set

In [157]:
```
df_result.boxplot()
plt.show()
```



In [76]:
```
plt.plot(lda_train_error,c='r')
plt.plot(lda_test_error,c='b')
plt.plot(qda_train_error,c = 'y')
plt.plot(qda_test_error,c = 'gray')
plt.legend()
plt.xlabel('times of training')
plt.ylabel('error rate')
plt.title('error rate of lda and qda')
plt.show()
```



0.3 comparsion between lda and qda:

1. when the bayes decision boundary is linear, the classification performances of lda and qda do not show distinct difference.

0.4 PROBLEM 3:

when bayes decision boundary are non-linear, I guess the classificaiton performance of QDA can out perform that of LDA

```
In [158]: N = 1000
          indice = list(range(N))
          lda_train_error = []
          qda_train_error = []
          lda_test_error = []
          qda_test_error = []
          for time in range(1000):
              random.seed(time)
              #stimulate dataset
              X1 = np.array([random.uniform(-1,1) for i in range(1000)])
              X2 = np.array([random.uniform(-1,1) for i in range(1000)])
              e = np.array\
              ([np.random.normal(loc=0.0, scale=1, size=None) for i in range(1000)])
              Y = X1 + X2 + X1**2 + X2**2 + e## Y we got
```

```
Y = (Y>=0)# turn it into 'TRUE' and 'FALSE'
split_point = int(len(X1)*0.7)
indice = np.random.permutation(indice)
train_x = np.array([X1[indice[:split_point]], X2[indice[:split_point]]]).T
train_y = Y[indice[:split_point]]
test_x = np.array([X1[indice[split_point:]], X2[indice[split_point:]]]).T
test_y = Y[indice[split_point:]]

# LDA training and testing
clf = LDA()
clf.fit(train_x,train_y)
y_predict = clf.predict(test_x)
y_train_hat = clf.predict(train_x)
lda_train_error.append\
(sum(np.ones(len(y_train_hat))[y_train_hat!=train_y])/len(train_y))
lda_test_error.append
\(sum(np.ones(len(test_y))[y_predict!=test_y])/len(test_y))

#QDA training and testing
clf = QDA()
clf.fit(train_x,train_y)
y_predict = clf.predict(test_x)
y_train_hat = clf.predict(train_x)
qda_train_error.append\
(sum(np.ones(len(y_train_hat))[y_train_hat!=train_y])/len(train_y))
qda_test_error.append\
(sum(np.ones(len(test_y))[y_predict!=test_y])/len(test_y))
```

0.4.1 table of LDA and QDA error rate on training/testing set

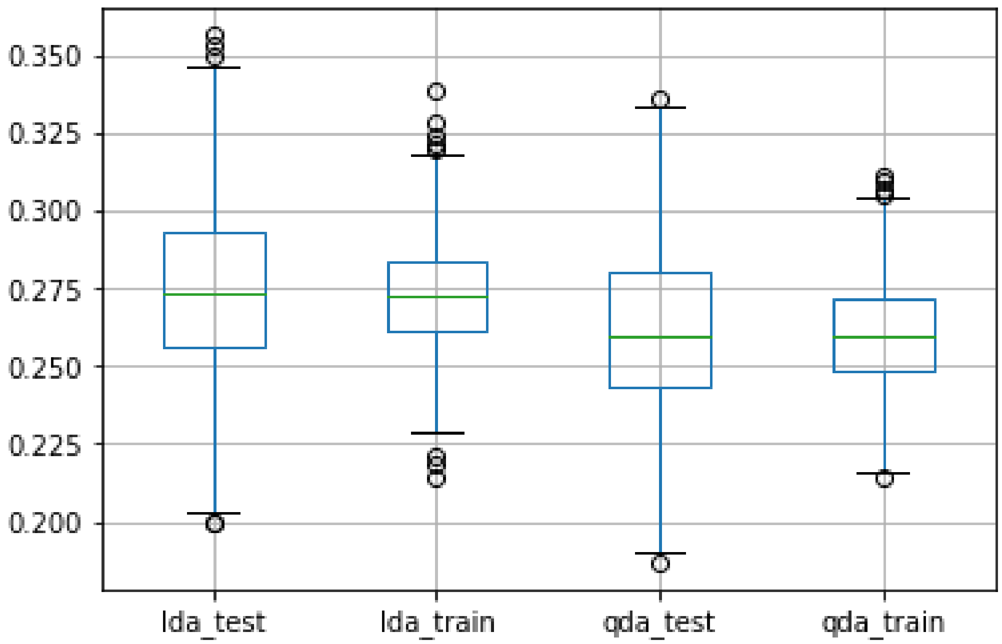
In [159]:
```
df_result = pd.DataFrame({'lda_train':lda_train_error,'lda_test':lda_test_error,\
                          'qda_train':qda_train_error,\
                         'qda_test':qda_test_error})
df_result.describe()
```

Out[159]:

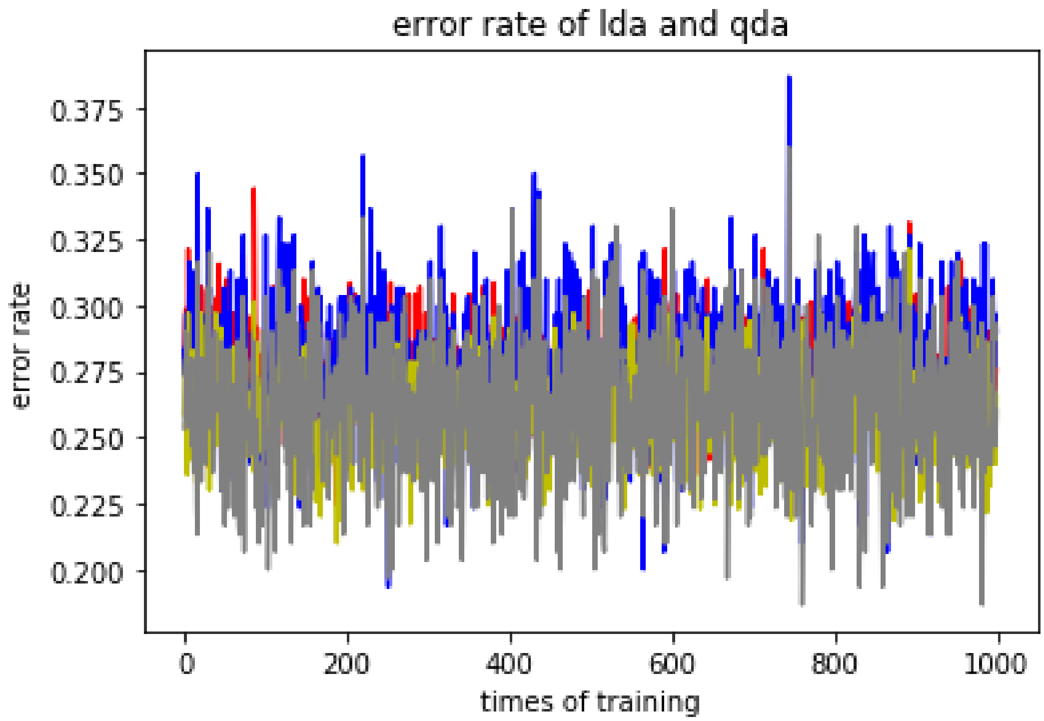
	lda_test	lda_train	qda_test	qda_train
count	1000.000000	1000.000000	1000.000000	1000.000000
mean	0.275083	0.273034	0.262077	0.259720
std	0.026058	0.017488	0.025492	0.016268
min	0.200000	0.214286	0.186667	0.214286
25%	0.256667	0.261429	0.243333	0.248571
50%	0.273333	0.272857	0.260000	0.260000
75%	0.293333	0.284286	0.280000	0.271429
max	0.356667	0.338571	0.336667	0.311429

0.4.2 graph of boxplot of LDA and QDA error rate on training/testing set

```
In [160]: df_result.boxplot()
          plt.show()
```




```
In [79]: plt.plot(lda_train_error,c='r')
         plt.plot(lda_test_error,c='b')
         plt.plot(qda_train_error,c = 'y')
         plt.plot(qda_test_error,c = 'gray')
         plt.legend()
         plt.xlabel('times of training')
         plt.ylabel('error rate')
         plt.title('error rate of lda and qda')
         plt.show()
```



0.4.3 comparsion between LDA and QDA

As our results show: when the bayes decision boundary is non linear, qda performs better than lda both on training and testing dataset. This is resulted from the fact that qda use the quadratic parameters to depict the non-linear boundary.

0.5 PROBLEM 4:

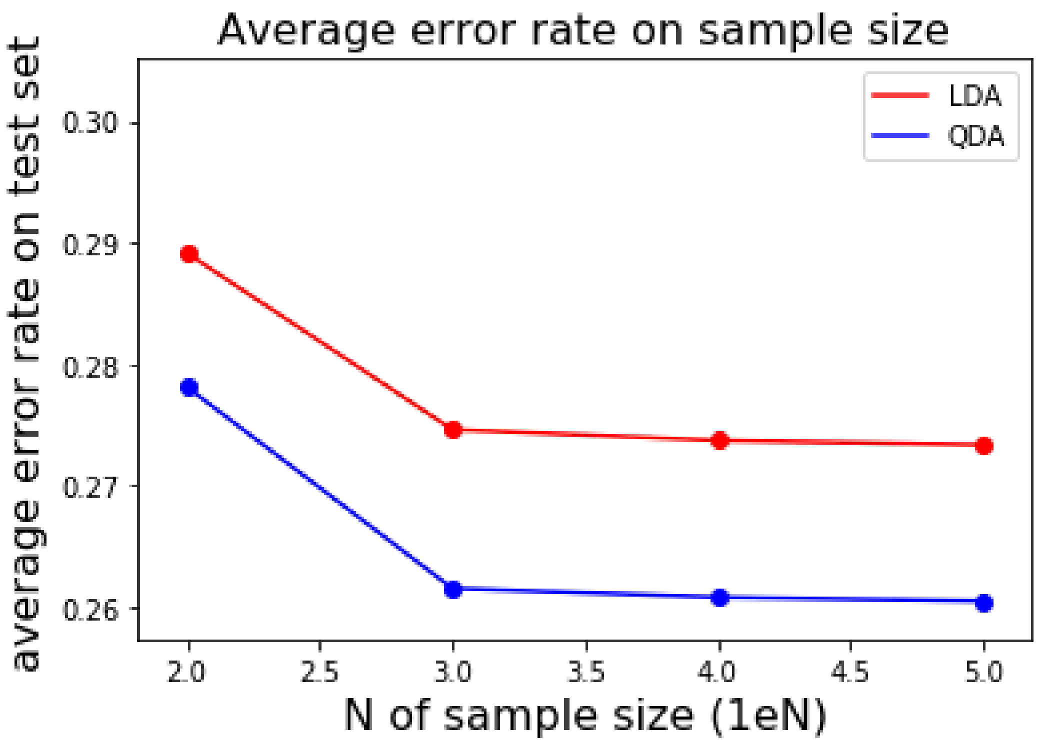
```
In [81]: N_list = [1e02,1e03,1e04,1e05]
         LDA_train_error = []
         LDA_test_error = []
         QDA_train_error = []
         QDA_test_error = []
         for N in N_list:
             N = int(N)
             indice = list(range(N))#sample size
             lda_train_error = []
             qda_train_error = []
             lda_test_error = []
             qda_test_error = []
             for time in range(1000):# number of experiment
                 random.seed(time)
                 #stimulate dataset
                 X1 = np.array([random.uniform(-1,1) for i in range(N)])#sample size
```

```
            X2 = np.array([random.uniform(-1,1) for i in range(N)])#sample size
            e = np.array\
            ([np.random.normal(loc=0.0, scale=1, size=None) for i in range(N)])#sample si
            Y = X1 + X2 + X1**2 + X2**2 + e
            Y = (Y>=0)

            split_point = int(len(X1)*0.7)
            indice = np.random.permutation(indice)
            train_x = np.array([X1[indice[:split_point]], X2[indice[:split_point]]]).T
            train_y = Y[indice[:split_point]]
            test_x = np.array([X1[indice[split_point:]], X2[indice[split_point:]]]).T
            test_y = Y[indice[split_point:]]

            #LDA training and testing
            clf = LDA()
            clf.fit(train_x,train_y)
            y_predict = clf.predict(test_x)
            y_train_hat = clf.predict(train_x)
            lda_train_error.\
            append(sum(np.ones(len(y_train_hat))[y_train_hat!=train_y])/len(train_y))
            lda_test_error.\
            append(sum(np.ones(len(test_y))[y_predict!=test_y])/len(test_y))

            #QDA training and testing
            clf = QDA()
            clf.fit(train_x,train_y)
            y_predict = clf.predict(test_x)
            y_train_hat = clf.predict(train_x)
            qda_train_error.\
            append(sum(np.ones(len(y_train_hat))[y_train_hat!=train_y])/len(train_y))
            qda_test_error.\
            append(sum(np.ones(len(test_y))[y_predict!=test_y])/len(test_y))
        LDA_train_error.append(np.mean(lda_train_error))
        QDA_train_error.append(np.mean(qda_train_error))
        LDA_test_error.append(np.mean(lda_test_error))
        QDA_test_error.append(np.mean(qda_test_error))
        print(N)
```

```
100
1000
10000
100000
```

In [165]:
```
plt.scatter(np.log10(np.array(N_list)),LDA_test_error,c='r')
plt.scatter(np.log10(np.array(N_list)),QDA_test_error,c='b')
plt.plot(np.log10(np.array(N_list)),LDA_test_error,c= 'r',label = 'LDA')
```

```
plt.plot(np.log10(np.array(N_list)),QDA_test_error,c= 'b',label = 'QDA')
plt.ylabel('average error rate on test set',size = 16)
plt.xlabel('N of sample size (1eN)',size = 16)
plt.title('Average error rate on sample size',size = 16)
plt.legend()
plt.show()
```



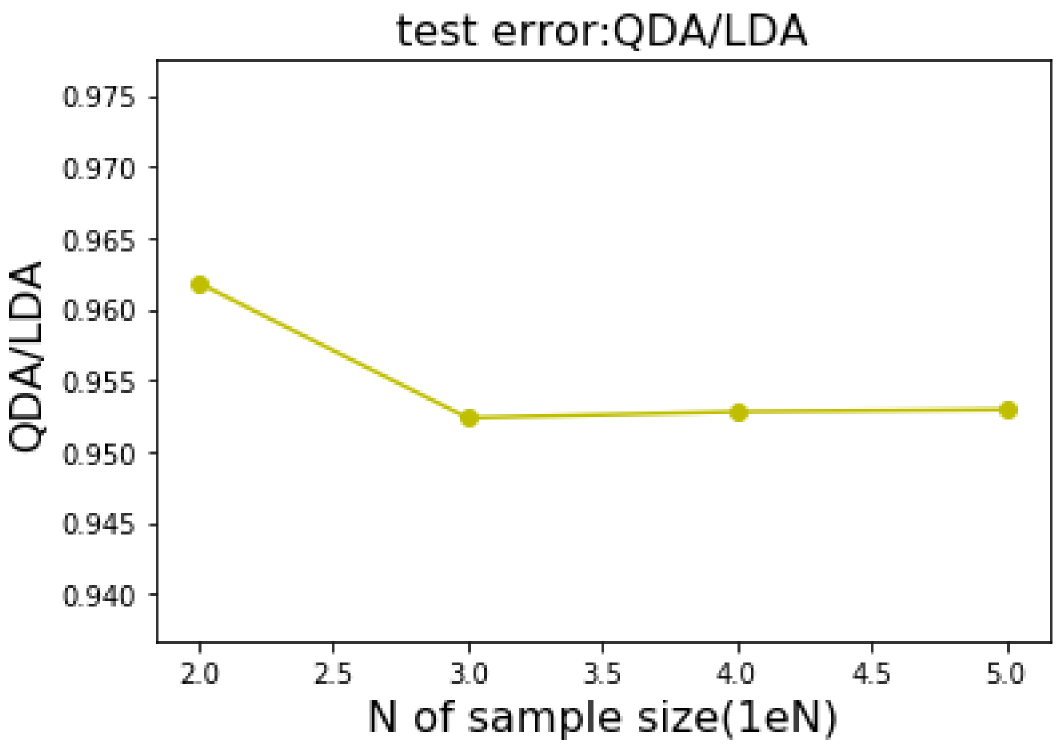
In [188]:
```
LDA_test_error
```

Out[188]:
```
[0.28913333333333335,
 0.27458333333333335,
 0.27368833333333326,
 0.27332443333333334]
```

In [189]:
```
QDA_test_error
```

Out[189]:
```
[0.27810000000000001, 0.26150000000000001, 0.260766, 0.26045996666666665]
```

In [166]:
```
plt.scatter(np.log10(np.array(N_list)),\
            np.array(QDA_test_error)/np.array(LDA_test_error),c= 'y')
plt.plot(np.log10(np.array(N_list)),\
         np.array(QDA_test_error)/np.array(LDA_test_error),c= 'y')
plt.ylabel('QDA/LDA',size = 16)
plt.xlabel('N of sample size(1eN)',size = 16)
plt.title('test error:QDA/LDA',size = 16)
plt.show()
```



AS graph"test error: QDA/LDA" shows, when bayes decision boundary is non linear and as sample size increases, the test error rate of QDA relative to LDA will decrease at first, and then it will become stable. The initial decrease is due to the fact that QDA is better than LDA to learn the non linear boundary, and QDA can perform better and learn more quickly than LDA. AS the sample size is sufficient for both classifiers to learn their best, they will give the best performance as they could, and then the test error rate for both classifiers will become constant and the relative performance of ODA against LDA will stabilize as well.

0.6 PROBLEM 5:

In [93]:
```
df_data = pd.read_csv('mental_health.csv')
```

In [94]:
```
df_data.describe()
```

Out[94]:

	vote96	mhealth_sum	age	educ	black \
count	2613.000000	1414.000000	2828.000000	2820.000000	2832.000000
mean	0.682357	2.869165	45.556931	13.250709	0.141243
std	0.465649	3.066242	17.100132	2.927512	0.348333
min	0.000000	0.000000	18.000000	0.000000	0.000000
25%	0.000000	1.000000	32.000000	12.000000	0.000000
50%	1.000000	2.000000	42.000000	13.000000	0.000000
75%	1.000000	4.000000	57.000000	16.000000	0.000000
max	1.000000	16.000000	89.000000	20.000000	1.000000

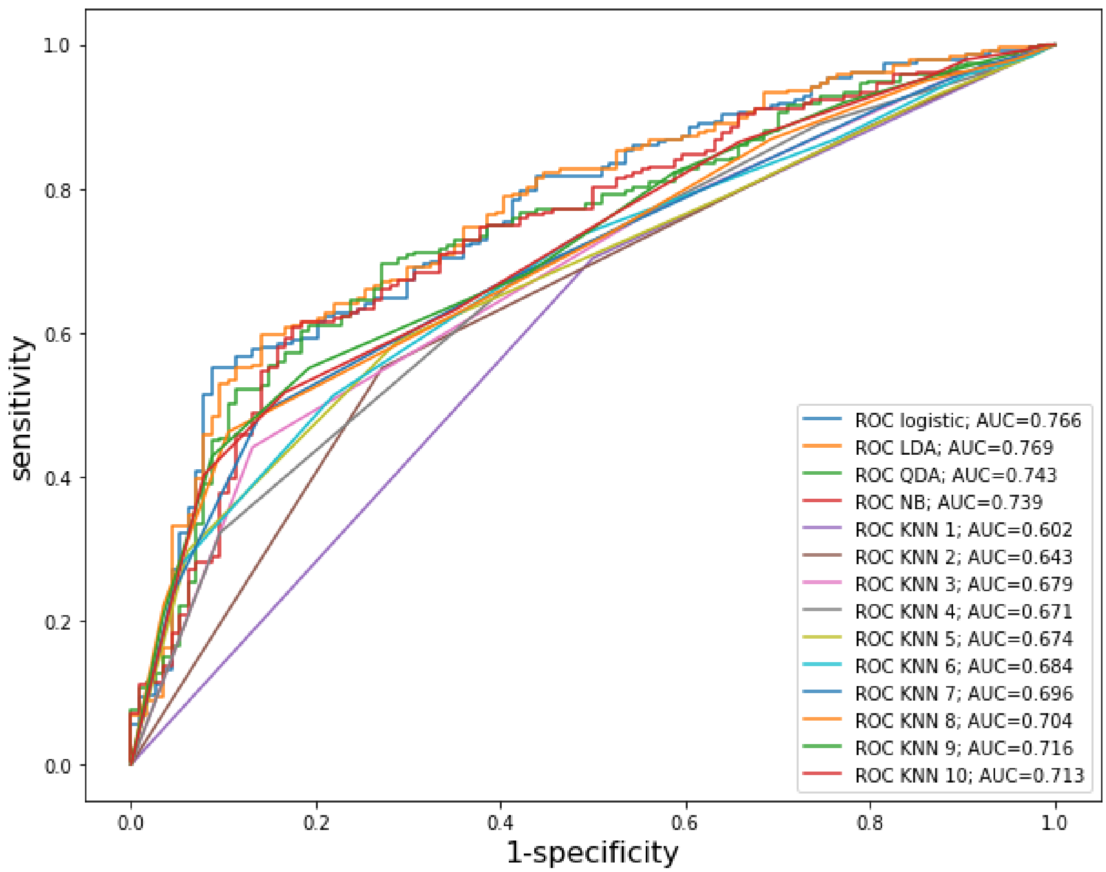
```
             female      married        inc10
count   2832.000000  2831.000000  2503.000000
mean       0.564972     0.475450     4.576070
std        0.495848     0.499485     3.608336
min        0.000000     0.000000     0.053500
25%        0.000000     0.000000     2.006200
50%        1.000000     0.000000     3.477400
75%        1.000000     1.000000     5.884900
max        1.000000     1.000000    14.879600
```

In [95]:
```
df_data.isnull().sum()
```

Out[95]:
```
vote96          219
mhealth_sum    1418
age               4
educ             12
black             0
female            0
married           1
inc10           329
dtype: int64
```

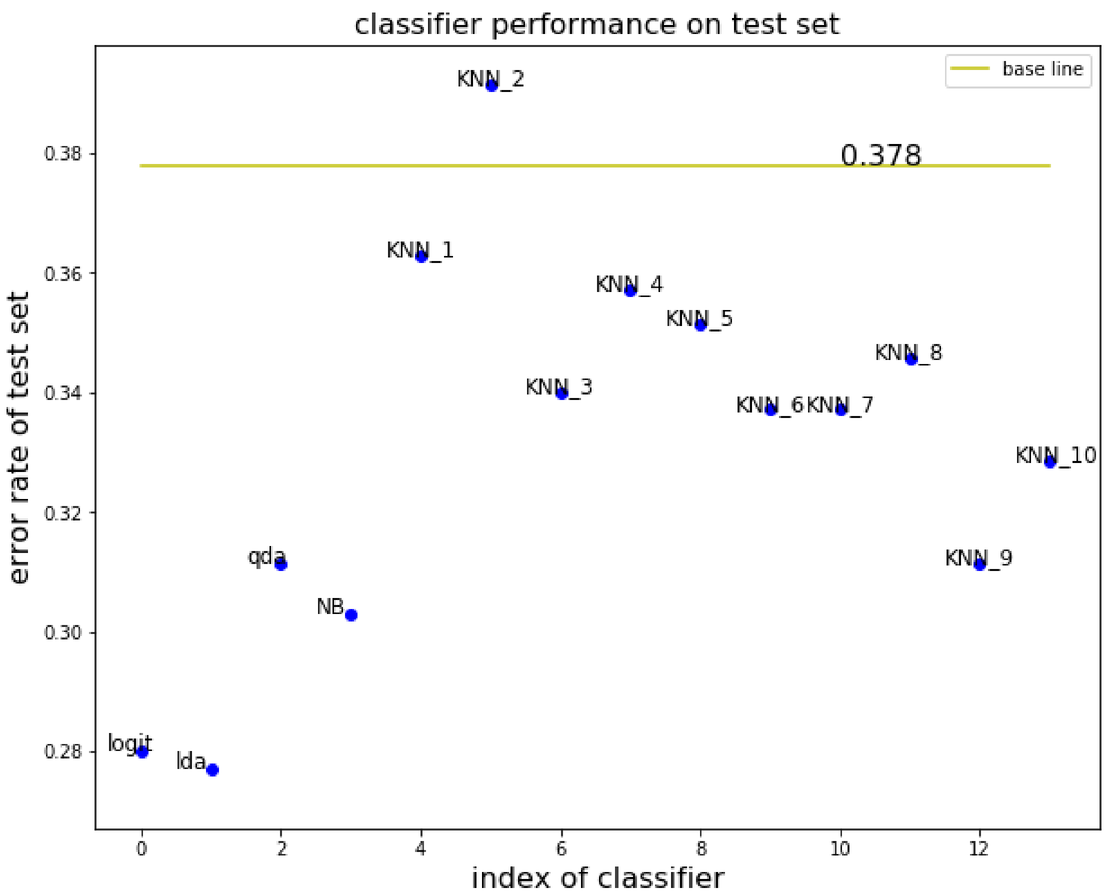
In [186]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import auc
from sklearn.metrics import roc_curve
import copy
plt.figure(figsize=(10,8))
#fig, ax = plt.subplots()
df_data.dropna(how='any',inplace = True)
y = copy.copy(df_data['vote96'])
x = copy.copy\
(df_data[['mhealth_sum','age','educ','black','female','married','inc10']])
error_rate = []

x_train, x_test, y_train, y_test = train_test_split\
(np.array(x),np.array(y),test_size = 0.3,shuffle = True)

# logistic regression
clf = LogisticRegression(random_state=0).fit(x_train, y_train)
y_predict = clf.predict(x_test)
error_rate.append(sum(np.ones(len(y_test))[y_predict!=y_test])/len(y_test))
y_predict_prob = clf.predict_proba(x_test)[:,1]
fpr, tpr,_ = roc_curve(y_test, y_predict_prob)
auc = roc_auc_score(y_test,  y_predict_prob)
plt.plot(fpr,tpr,label = 'ROC logistic; AUC={}'.format(round(auc,3)))
```

```python
# LDA
clf = LDA().fit(x_train, y_train)
y_predict = clf.predict(x_test)
error_rate.append(sum(np.ones(len(y_test))[y_predict!=y_test])/len(y_test))
y_predict_prob = clf.predict_proba(x_test)[:,1]
fpr, tpr,_ = roc_curve(y_test, y_predict_prob)
auc = roc_auc_score(y_test,  y_predict_prob)
plt.plot(fpr,tpr,label = 'ROC LDA; AUC={}'.format(round(auc,3)))

#QDA
clf = QDA().fit(x_train, y_train)
y_predict = clf.predict(x_test)
error_rate.append(sum(np.ones(len(y_test))[y_predict!=y_test])/len(y_test))
y_predict_prob = clf.predict_proba(x_test)[:,1]
fpr, tpr,_ = roc_curve(y_test, y_predict_prob)
auc = roc_auc_score(y_test,  y_predict_prob)
plt.plot(fpr,tpr,label = 'ROC QDA; AUC={}'.format(round(auc,3)))

#NB
clf = GaussianNB().fit(x_train, y_train)
y_predict = clf.predict(x_test)
error_rate.append(sum(np.ones(len(y_test))[y_predict!=y_test])/len(y_test))
y_predict_prob = clf.predict_proba(x_test)[:,1]
fpr, tpr,_ = roc_curve(y_test, y_predict_prob)
auc = roc_auc_score(y_test,  y_predict_prob)
plt.plot(fpr,tpr,label = 'ROC NB; AUC={}'.format(round(auc,3)))

#KNN 1-10
for i in range(1,11):
    clf = KNeighborsClassifier(n_neighbors=i,p=2).fit(x_train, y_train)
    y_predict = clf.predict(x_test)
    error_rate.append(sum(np.ones(len(y_test))[y_predict!=y_test])/len(y_test))
    y_predict_prob = clf.predict_proba(x_test)[:,1]
    fpr, tpr,_ = roc_curve(y_test, y_predict_prob)
    auc = roc_auc_score(y_test,  y_predict_prob)
    plt.plot(fpr,tpr,label = 'ROC KNN {}; AUC={}'.format(i, round(auc,3)))
plt.xlabel('1-specificity',size = 16)
plt.ylabel('sensitivity', size = 16)
plt.legend()
plt.show()
```




```
In [192]: name_to_index = ['logit','lda','qda','NB','KNN_1',\
                          'KNN_2','KNN_3','KNN_4','KNN_5',\
                          'KNN_6','KNN_7','KNN_8','KNN_9','KNN_10']
```

```
In [198]: plt.figure(figsize=(10,8))
          for i in range(len(name_to_index)):
              plt.scatter(i,error_rate[i],c='b')
              plt.text(i-0.5,error_rate[i],name_to_index[i],color = 'black',size = 12)
          plt.plot(range(len(error_rate)),[0.378]*len(error_rate),c= 'y',label = 'base line')
          plt.xlabel('index of classifier',size = 16)
          plt.ylabel('error rate of test set',size =16)
          plt.title('classifier performance on test set',size = 16)
          plt.text(10,0.378,'0.378',size = 16)
          plt.legend()
          plt.show()
```



I think the LDA classifier give the best performance. I would define a good classifier as 1. have relative low error rate on test set; 1. have relative high AUC score.

This definition is reasonable because low error rate suggests this classifier learn the data well and can generalize the learning pattern to unseen data. Moreover, high AUC score suggests that as the false positive rate increases, the true positive rate will increase in a even faster speed, and this model can give its performace(highest true positive rate) at a relative low error level(low false positive rate). Based on these criteria I set, LDA is our best classifier as LDA classifier owns the lowest test error rate: 0.300 and the highest AUC: 0.779 on our dataset.