Shengwenxin_Ni_HW2

February 2, 2020

0.1 Import Libraries

[1137]:
```
import random
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
import seaborn as sns
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis
from sklearn.feature_selection import RFE
from sklearn import metrics
from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import NearestNeighbors
from sklearn.metrics import roc_auc_score
from sklearn.neighbors import KNeighborsClassifier
import copy
```

1 Question 1

1.1 Helper functions

produc_list: Generate a random list of X values with the desired length. Will be used in the following questions as well.

produce_y1: Calculate the value of y (in terms of probability)

[879]:
```
def produce_list(N,seed):
    random.seed(seed)
    x = []
    for i in range(N):
        x.append(random.uniform(-1,1))
    return x

def produce_y1(x1,x2):
```

```
    error = random.normalvariate(0,0.25)
    l_odds = x1 + x1**2 + x2 + x2 ** 2 + error
    prob = np.exp(l_odds) / (1 + np.exp(l_odds))
    return prob
```

1.2 Main Function

[1138]:
```
x1 = produce_list(200,317)
x2 = produce_list(200,828)

random.seed(317)

s1 = []
s2 = []
f1 = []
f2 = []

for i in range(len(x1)):
    if produce_y1(x1[i],x2[i]) >= 0.5:
        s1.append(x1[i])
        s2.append(x2[i])
    else:
        f1.append(x1[i])
        f2.append(x2[i])
```

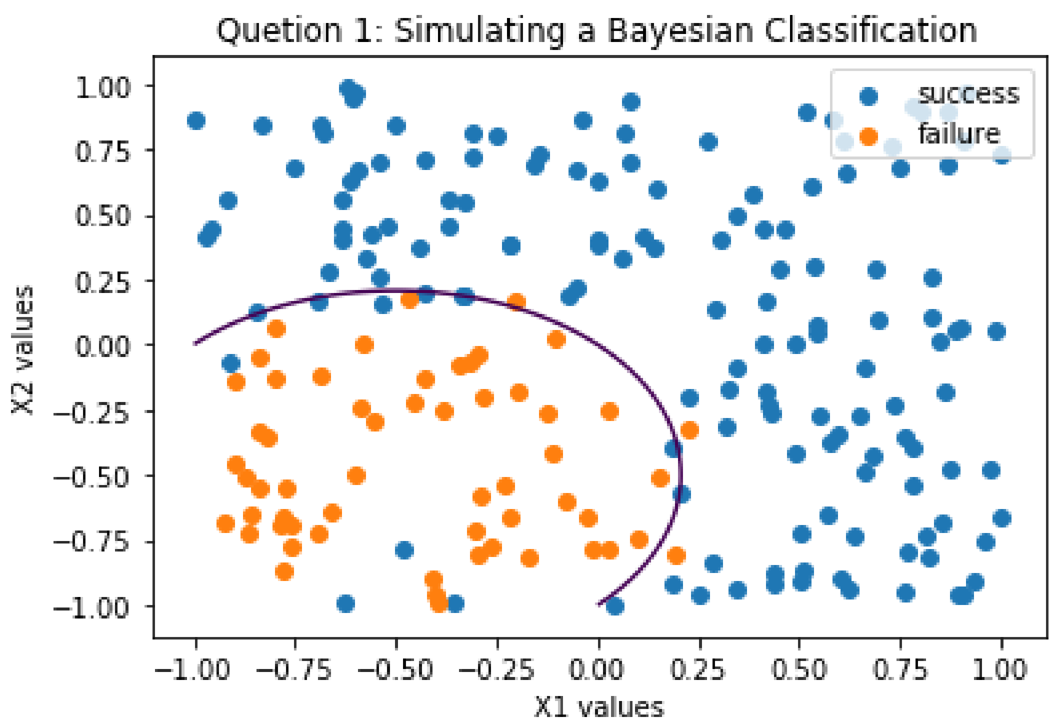
1.3 The Bayes Decision Boundary

Calculate the Bayes Decision Boundary based on the value of X1 and X2

[1139]:
```
x1 = np.linspace(-1,1,1000)
x2 = np.linspace(-1,1,1000)

Z = []
for i in x1:
    each = []
    for j in x2:
        each.append(i + i**2 + j + j ** 2)
    Z.append(each)
```

1.4 Generate the Graph

[1198]:
```
plt.scatter(s1,s2)
plt.scatter(f1,f2)
plt.contour(x1,x2,Z,[0])
plt.title('Quetion 1: Simulating a Bayesian Classification')
plt.xlabel('X1 values')
plt.ylabel('X2 values')
plt.legend(['success','failure'],loc='upper right')
plt.show()
```



Note: The solid purple line is the decision boundary.

2 Question 2

2.1 Helper Functions

produce_y2: Calculate the value of y (in terms of Boolean value)

err_rate: Calculate the error rate of a single round of simulation.

[903]:
```python
def produce_y2(l):
    random.seed(426501)
    y = []
    for i in l:
        error = random.normalvariate(0,1)
        if i[0] + i[1] + error >= 0:
            y.append(True)
        else:
            y.append(False)
    return y


def err_rate(clf,x_train,y_train,x,y):
    clf.fit(x_train,y_train)
    df = pd.DataFrame({'X':x,'actual-Y':y})
    df['predicted-Y'] = clf.predict(x)
    df['error'] = (df['actual-Y'] != df['predicted-Y'])

    return (df['error']== True).sum()/df['error'].count()
```

2.2 Main Function

simulate-err: The function that produce a dataframe that keeps the resulte of the error rate of lda-test,lda-train,qda-test, qda-train, given the number of random data used in a single simulation. This function will be used in question 3&4 as well.

[1141]:
```python
def simulate_err(n,seed,func,include_train = True):
    lda_train = []
    lda_test = []
    qda_train = []
    qda_test = []

    for i in range(1000):

        x1 = produce_list(n, seed)
        seed += 1
        x2 = produce_list(n, seed)
        X = list(zip(x1, x2))
        Y = func(X)

        seed += 1
        x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size = 0.
↪3, random_state = seed)

        lda_clf = LinearDiscriminantAnalysis()
```

```
        qda_clf = QuadraticDiscriminantAnalysis()

        lda_test.append(err_rate(lda_clf,x_train,y_train,x_test,y_test))
        qda_test.append(err_rate(qda_clf,x_train,y_train,x_test,y_test))

        if include_train:
            lda_train.append(err_rate(lda_clf,x_train,y_train,x_train,y_train))
            qda_train.append(err_rate(qda_clf,x_train,y_train,x_train,y_train))

    df = { 'lda_test': lda_test,'qda_test': qda_test}

    if include_train:
        df['lda_train'] = lda_train
        df['qda_train'] = qda_train
    df = pd.DataFrame(df)

    return df
```

2.3 Statistical Table

[1142]:
```
df2 = simulate_err(1000,828,produce_y2)
df2.describe()
```

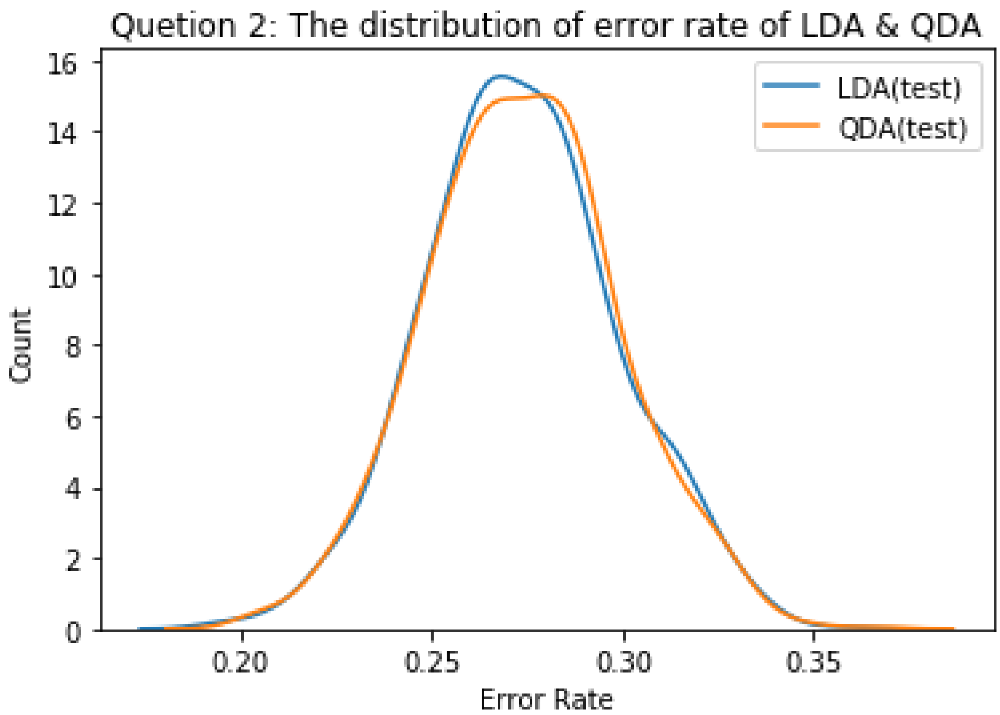
[1142]:

	lda_test	qda_test	lda_train	qda_train
count	1000.000000	1000.000000	1000.000000	1000.000000
mean	0.273490	0.273750	0.271691	0.271017
std	0.025289	0.025219	0.015979	0.015986
min	0.193333	0.200000	0.212857	0.215714
25%	0.256667	0.256667	0.261071	0.260000
50%	0.273333	0.273333	0.271429	0.271429
75%	0.290000	0.290000	0.282857	0.281429
max	0.356667	0.366667	0.330000	0.325714

2.4 Graph (Distribution of error rates)

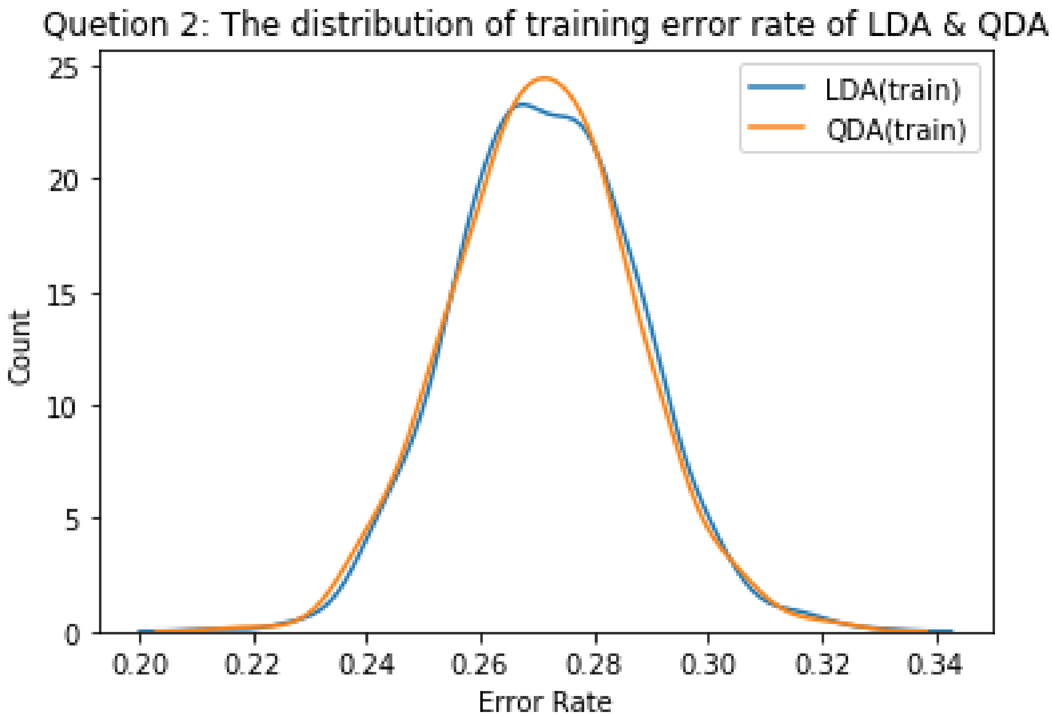
[1146]:
```
sns.distplot(df2['lda_test'],hist= False,label = 'LDA(test)')
sns.distplot(df2['qda_test'],hist= False,label = 'QDA(test)')

plt.title('Quetion 2: The distribution of testing error rate of LDA & QDA')
plt.xlabel('Error Rate')
plt.ylabel('Count')

plt.legend()
```

```
plt.show()
```



[1147]:

```
sns.distplot(df2['lda_train'],hist= False,label = 'LDA(train)')
sns.distplot(df2['qda_train'],hist= False,label = 'QDA(train)')

plt.title('Quetion 2: The distribution of training error rate of LDA & QDA')
plt.xlabel('Error Rate')
plt.ylabel('Count')

plt.legend()
plt.show()
```



2.5 Comments/ Conclusion

Given the results above, I believe that LDA and QDA perform equally on both the training set and the testing set. My conclusion can be supported by two illustrations (the distribution of training error/rate of LDA & QDA). We can see that in both graphs, two distribution lines (LDA and QDA) almost coincide.

3 Question 3

3.1 Helper Functions

produce_y3: Calculate the value of y (in terms of Boolean value)

[906]:
```
def produce_y3(l):
    random.seed(426501)
    y = []
    for i in l:
        error = random.normalvariate(0,1)
        if i[0] + i[0]**2 + i[1] + i[1]**2 + error >= 0:
            y.append(True)
        else:
```

```
            y.append(False)
    return y
```

3.2 Main Function + Statistical Table

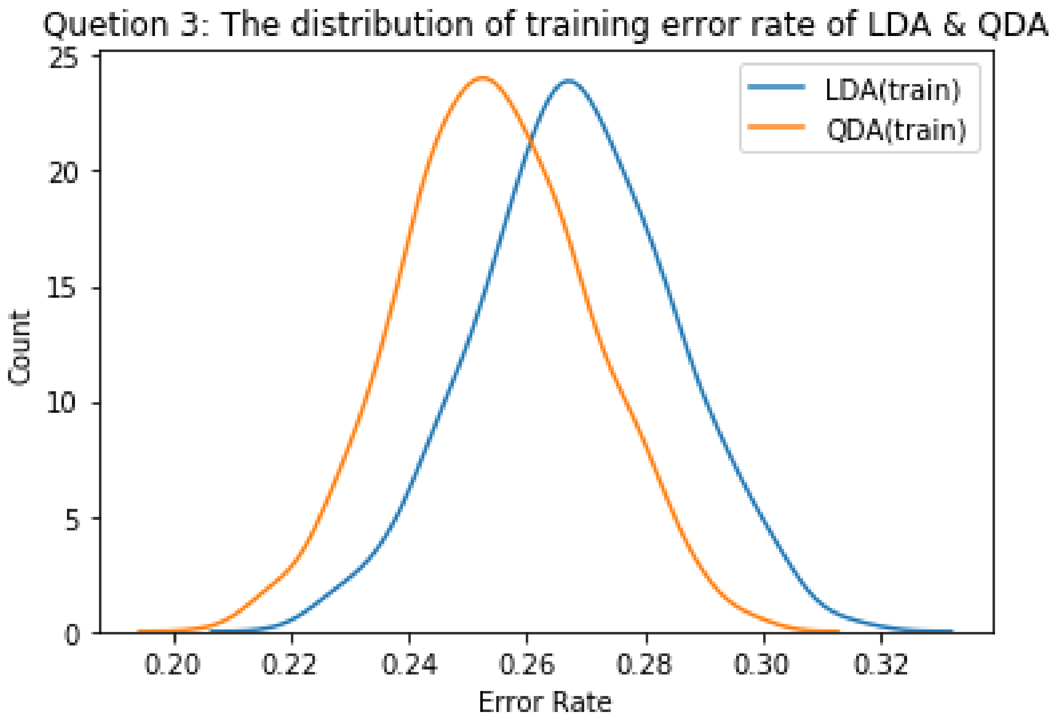
[907]:
```
df3 = simulate_err(1000,426501,produce_y3)
df3.describe()
```

[907]:

	lda_test	qda_test	lda_train	qda_train
count	1000.000000	1000.000000	1000.000000	1000.000000
mean	0.272270	0.259200	0.268303	0.254401
std	0.025291	0.024572	0.016778	0.016023
min	0.186667	0.200000	0.220000	0.207143
25%	0.253333	0.243333	0.257143	0.242857
50%	0.270000	0.256667	0.268571	0.254286
75%	0.290000	0.274167	0.280000	0.265714
max	0.363333	0.330000	0.318571	0.300000

3.3 Graph (Distribution of error rates)

[1150]:
```
sns.distplot(df3['lda_train'],hist= False,label = 'LDA(train)')
sns.distplot(df3['qda_train'],hist= False,label = 'QDA(train)')

plt.title('Quetion 3: The distribution of training error rate of LDA & QDA')
plt.xlabel('Error Rate')
plt.ylabel('Count')

plt.legend()
plt.show()
```



[1149]:
```
sns.distplot(df3['lda_test'],hist= False,label = 'LDA(test)')
sns.distplot(df3['qda_test'],hist= False,label = 'QDA(test)')

plt.title('Quetion 3: The distribution of testing error rate of LDA & QDA')
plt.xlabel('Error Rate')
plt.ylabel('Count')

plt.legend()
plt.show()
```

3.4 Comments/ Conclusion

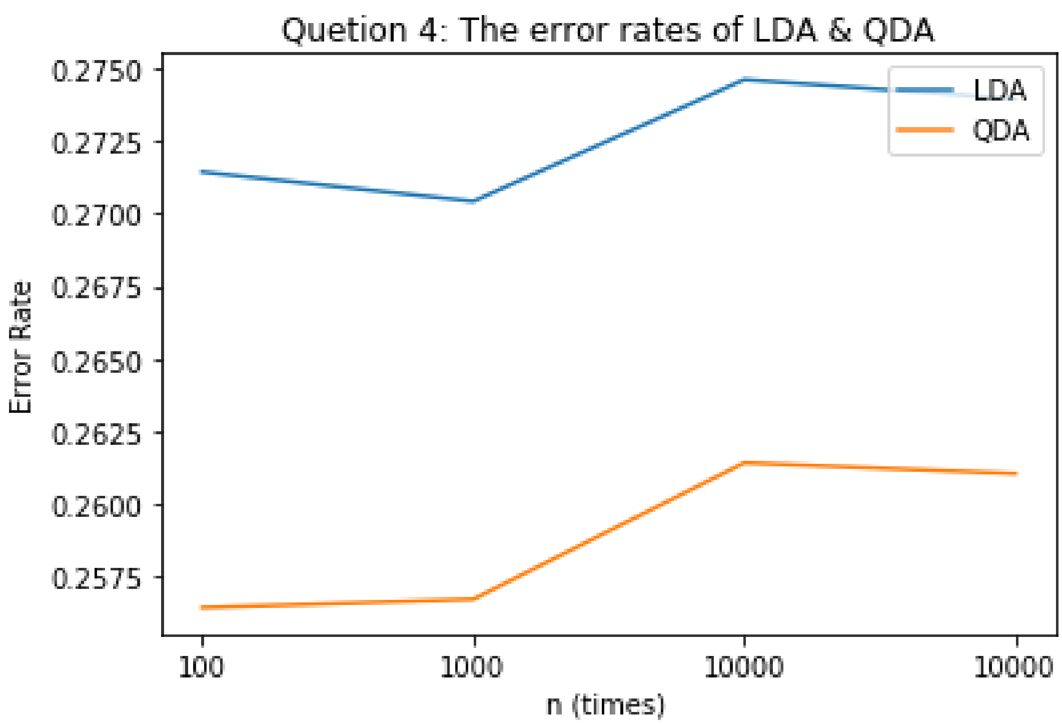
Given the results above, I believe that QDA slightly better than LDA on both the training set and the testing set. My conclusion can be supported by two illustrations (the distribution of training error/rate of LDA & QDA). We can see that in both graphs, the distribution line QDA is slightly left of that of LDA, which indicates a relative lower error rate.

4 Question 4

4.1 Main Function + Graph (Seed = 828)

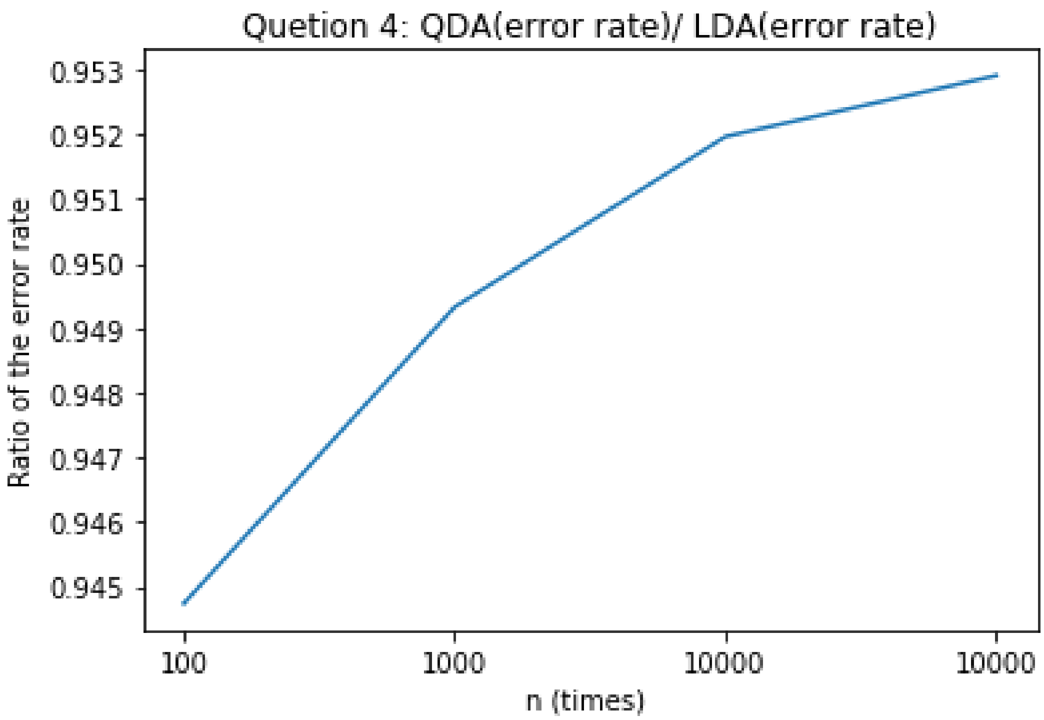
[825]:
```
n_list = [100,1000,10000,100000]

lda_list828 = []
qda_list828 = []

for n in n_list:
    df = simulate_err(n,828,produce_y3,False)
    lda_list828.append(df['lda_test'].mean())
    qda_list828.append(df['qda_test'].mean())
```

[1169]:
```
q4_xticks = [100,1000,10000,10000]
plt.xticks([0,1,2,3] ,q4_xticks)
plt.plot(lda_list828)
plt.plot(qda_list828)


plt.title('Quetion 4: The test error rates of LDA & QDA')
plt.xlabel('n (times) ')
plt.ylabel('Error Rate')
plt.legend(['LDA','QDA'],loc ='upper right')
plt.show()
```



[1170]:
```
ratio828 = []

for i in range(len(qda_list828)):
    ratio828.append(qda_list828[i]/lda_list828[i])
ratio828


plt.title('Quetion 4: QDA(error rate)/ LDA(error rate)')
plt.xlabel('n (times) ')
plt.xticks([0,1,2,3] ,q4_xticks)
plt.plot(ratio828)
```

```
plt.ylabel('Ratio of the test error rate')
plt.show()
```

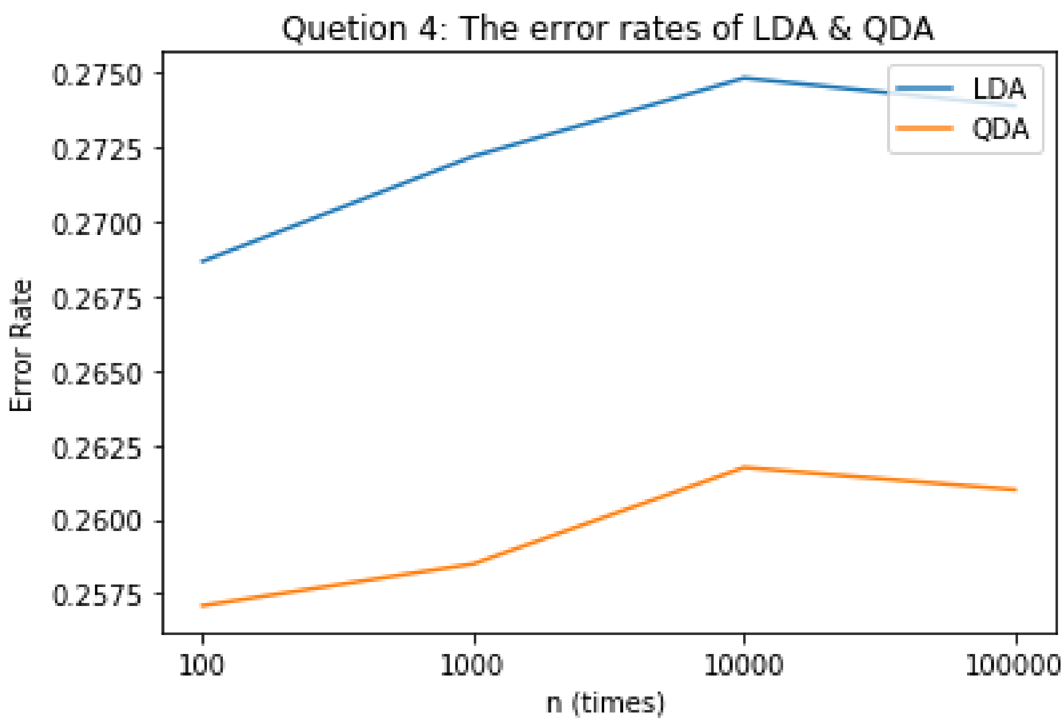


4.2 Main Function + Graph (Seed = 317)

[886]:
```
n_list = [100,1000,10000,100000]

lda_list = []
qda_list = []

for n in n_list:
    df = simulate_err(n,317,produce_y3,False)
    lda_list.append(df['lda_test'].mean())
    qda_list.append(df['qda_test'].mean())
```

[1171]:
```
q4_xticks = ['100','1000','10000','100000']
plt.xticks([0,1,2,3] ,q4_xticks)
plt.plot(lda_list)
plt.plot(qda_list)

plt.title('Quetion 4: The test error rates of LDA & QDA')
```

```
plt.xlabel('n (times) ')
plt.ylabel('Error Rate')
plt.legend(['LDA','QDA'],loc='upper right')
plt.show()
```



[1172]:
```
ratio = []
for i in range(len(qda_list)):
    ratio.append(qda_list[i]/lda_list[i])
ratio

plt.title('Quetion 4: QDA(error rate)/ LDA(error rate)')
plt.xlabel('n (times) ')
plt.xticks([0,1,2,3] ,q4_xticks)
plt.plot(ratio)
plt.ylabel('Ratio of the test error rate')
plt.show()
```

4.3 Comments/Conclusions

I tried the simulation twice with different random seeds. Two results above demonstrate that the ratio test error of QDA over LDA does not necessarily improve as n becomes larger. Instead, the results fluctuate but converge around 0.953.

5 Question 5

5.1 Split Data (into training vs testing)

[1173]:
```
df5 = pd.read_csv('mental_health.csv').dropna()
y = copy.copy(df5['vote96'])
x = copy.copy(df5[X_col])
x_train, x_test, y_train, y_test = train_test_split(np.array(x),np.
 ↪array(y),test_size = 0.3,shuffle = True)
```

5.2 Helper Functions:

q5: Input a model and get a dictionary that contains the error rate and the information related to ROC.

plot_roc: Plot the graph of ROC

plot_area: Plot the graph of AUC.

plot_error: Plot the graph of the error rate

[1183]:
```
def q5(clf, X_train,Y_train,X_test, Y_test,name):

    clf.fit(X_train,Y_train)
    Y_predicted = clf.predict(X_test)
    df= pd.DataFrame({'Y_actual':Y_test, 'Y_predicted': Y_predicted})
    df['error'] = (df['Y_actual'] != df['Y_predicted'])

    y_predict_prob = clf.predict_proba(x_test)[:,1]
    df['score'] = y_predict_prob
    fpr, tpr, thresholds = metrics.roc_curve(df['Y_actual'], df['score'])

    error_rate = (df['error']== True).sum()/df['error'].count()
    area = roc_auc_score(df['Y_actual'], df['score'])

    dic = {'fpr':fpr,'tpr':tpr,'error rate':error_rate,'area':area,'name':name}

    return dic

def plot_roc(dic_list):
    legend = []
    plt.figure(figsize=(15,8))
    for dic in dic_list:
        fpr = dic['fpr']
        tpr = dic['tpr']
        plt.plot(fpr,tpr)
        legend.append(dic['name'])

    plt.legend(legend,loc='lower right')
    plt.plot([0, 1], [0, 1], color='navy', linestyle='--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Question 5: ROC for each model')
    plt.show()

def plot_error(dic_list):
    name = []
    err = []
    plt.figure(figsize=(15,8))
    for dic in dic_list:
        name.append(dic['name'])
```

```
        err. append(dic['error rate'])
    plt.bar(name,err)
    plt.xlabel('Model')
    plt.ylabel('Error Rate')
    plt.title('Question 5: Error Rate for each model')

def plot_area(dic_list):
    name = []
    area = []
    plt.figure(figsize=(15,8))
    for dic in dic_list:
        name.append(dic['name'])
        area.append(dic['area'])
    plt.bar(name,area)
    plt.xlabel('Model')
    plt.ylabel('AUC')
    plt.title('Question 5: AUC for each model')
```

5.3 Logistic Regression Model

[1199]:
```
logi = LogisticRegression()
logi_dic = q5(logi,x_train,y_train,x_test,y_test,'LOGI')
```

```
/Users/nishengwenxin/opt/anaconda3/lib/python3.7/site-
packages/sklearn/linear_model/logistic.py:432: FutureWarning: Default solver
will be changed to 'lbfgs' in 0.22. Specify a solver to silence this warning.
  FutureWarning)
```

5.4 Linear Discriminant Model

[1185]:
```
lda = LinearDiscriminantAnalysis()
lda_dic = q5(lda,x_train,y_train,x_test,y_test,'LDA')
```

5.5 Quadratic Regression Model

[1186]:
```
qda = QuadraticDiscriminantAnalysis()
qda_dic = q5(qda,x_train,y_train,x_test,y_test,'QDA')
```

5.6 Gaussian Naive Bayes Model

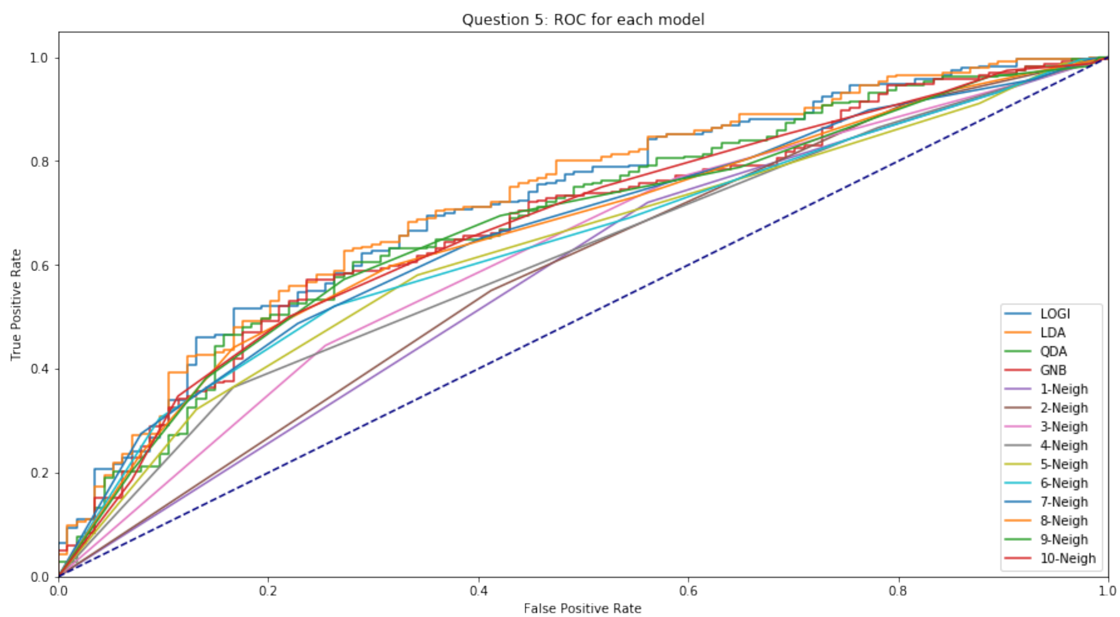
[1187]:
```
gnb = GaussianNB()
gnb_dic = q5(gnb,x_train,y_train,x_test,y_test,'GNB')
```

5.7 Nearest k-neighbors

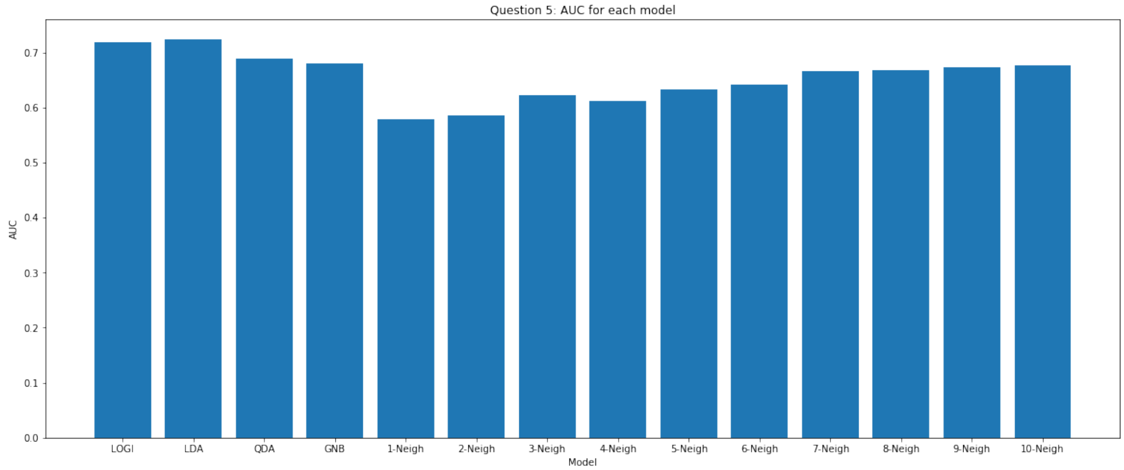
[1181]:
```
num = 1
dic_neigh = []
while num <= 10:
    name = str(num) + '-Neigh'
    clf = KNeighborsClassifier(n_neighbors = num)
    dic = q5(clf,x_train,y_train,x_test,y_test,name)
    dic_neigh.append(dic)
    num += 1
```

5.8 Graph - ROC & AUC

[1188]:
```
dic_list = [logi_dic,lda_dic,qda_dic,gnb_dic] + dic_neigh
plot_roc(dic_list)
```

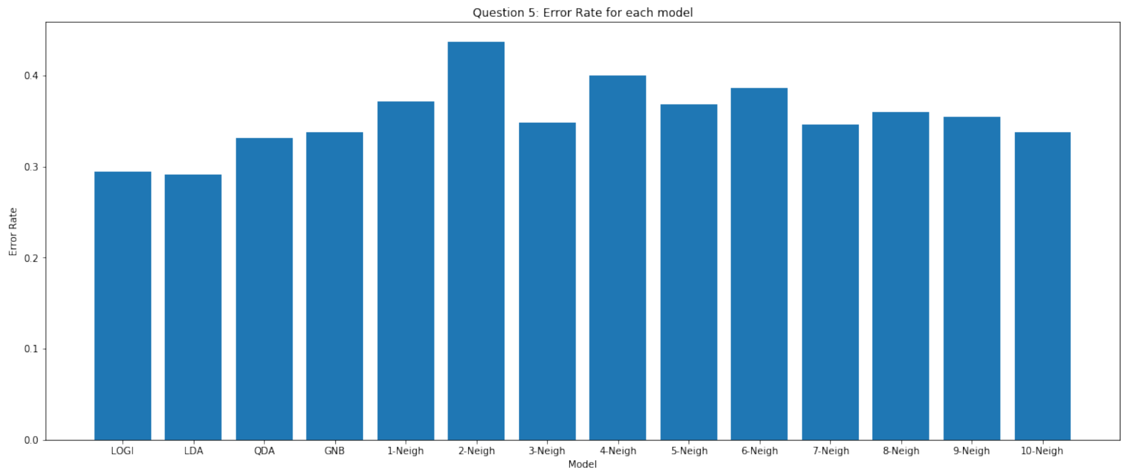


[1190]:
```
plot_area(dic_list)
```



5.9 Graph - Error Rate

[1191]:
```
plot_error(dic_list)
```



5.10 Comments/Conclusions

LDA performs the best, if the best is defined as giving the most accurate predictions. This statement can be supported by the fact that:

1. LDA model has the smallest error rate.
2. LDA model has the greatest area under the curve.