HW2

Jiaxuan Li

Question1

In [2]:

```
import random
import numpy as np
import math
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis
import pandas as pd
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
import scikitplot as skplt
```

In [35]:

```
np.random.seed(0)
```

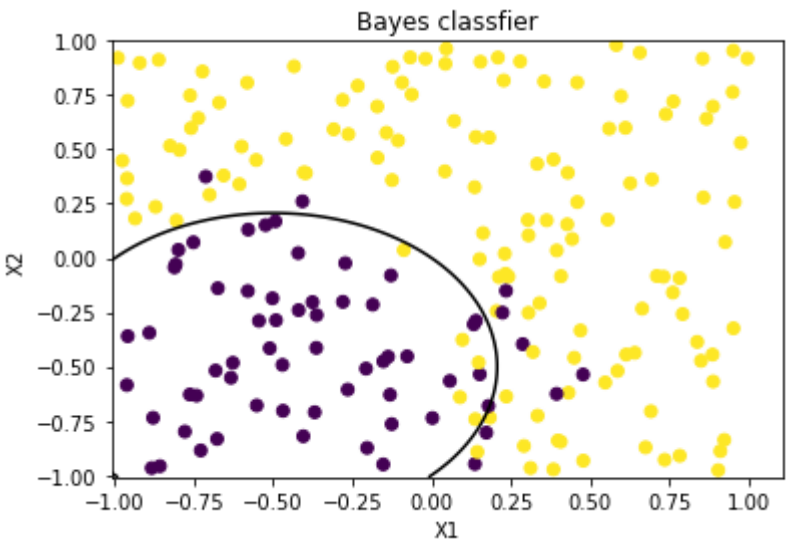
In [3]:

```
#set random seed
np.random.seed(0)
#simulate dataset
x1 = np.random.uniform(-1,1,200)
x2 = np.random.uniform(-1,1,200)
eps = np.random.normal(0,0.25,200)
#calculate y
y = x1+x1*x1+x2+x2*x2+eps
Y_exp = np.exp(y)
probability = Y_exp/(1+Y_exp)
label= np.where(probability>0.5,True,False)
#plot datapoints
plt.scatter(x1, x2, c=label)

#plot bayes decision boundary
x1 = np.arange(-1.01, 1.01, 0.01)
x2 = np.arange(-1.01, 1.01, 0.01)
X1, X2 = np.meshgrid(x1, x2)
y = X1 + X1 ** 2 + X2 + X2 ** 2
Y_exp = np.exp(y)
probability = Y_exp / (1 + Y_exp)
plt.contour(X1, X2, probability, levels=[0.5], colors='black')
plt.xlabel('X1')
plt.ylabel('X2')
plt.title('Bayes classfier')
```

Out[3]:

```
Text(0.5, 1.0, 'Bayes classfier')
```



Question2

The LDA method is better in test test because the Bayes decision boundary is linear. The QDA performs better in training set because QDA is more flexible than LDA and might overfit the train set. The result suggests that the train error is 0.274 for LDA and 0.273 for QDA, the test error is 0.2756 for LDA and 0.2761 for LDA. This is the same as the prediction, but the difference is really small.

In [3]:

```
lda_training = []
lda_test = []
qda_training = []
qda_test = []

for i in range(1000):
    #generate observations
    x1 = np.random.uniform(-1,1,1000)
    x2 = np.random.uniform(-1,1,1000)
    eps = np.random.normal(0,1,1000)
    #simulate y
    y_simulated = x1+x2+eps

    y= np.where(y_simulated>=0,True,False)
    X = np.column_stack((x1, x2))

    #split training and test dataset
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,shuf
fle=True)

    #estimate lda model
    clf_lda = LinearDiscriminantAnalysis()
    clf_lda.fit(X_train, y_train)
    LinearDiscriminantAnalysis()
    #calculate training and test error rate
    err_lda_train = 1 - clf_lda.score(X_train,y_train)
    err_lda_test = 1-clf_lda.score(X_test, y_test)
    lda_training.append(err_lda_train)
    lda_test.append(err_lda_test)
    #print(err_lda_train,err_lda_test)

    #estimate qda model
    clf_qda = QuadraticDiscriminantAnalysis()
    clf_qda.fit(X_train, y_train)
    QuadraticDiscriminantAnalysis()
    #calcuate training and test error rate
    err_qda_train = 1 - clf_qda.score(X_train,y_train)
    err_qda_test = 1-clf_qda.score(X_test, y_test)
    qda_training.append(err_qda_train)
    qda_test.append(err_qda_test)

#     print(err_qda_train,err_qda_test)
```

In [6]:

```
df = pd.DataFrame({'lda_training':lda_training,'lda_test':lda_test,'qda_trainin
g':qda_training,'qda_test':qda_test})
```

In [7]:

```
df.describe()
```

Out[7]:

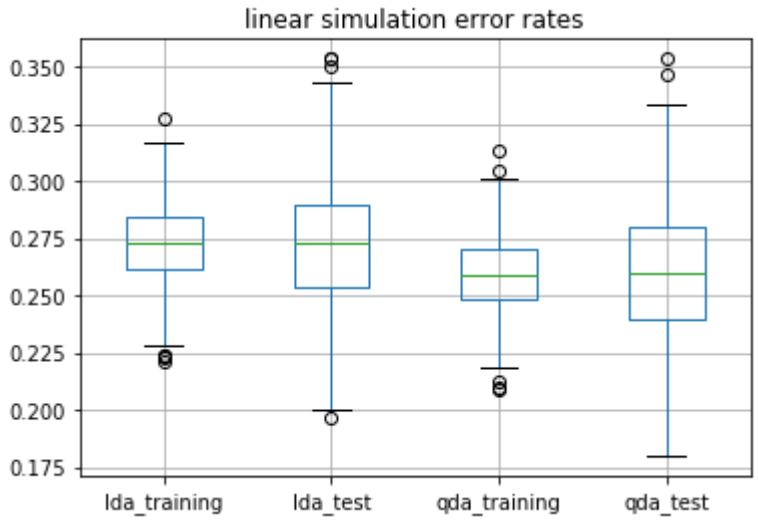
	lda_training	lda_test	qda_training	qda_test
count	1000.000000	1000.000000	1000.000000	1000.000000
mean	0.274297	0.275670	0.273069	0.276153
std	0.016442	0.025547	0.016503	0.025634
min	0.227143	0.156667	0.227143	0.176667
25%	0.262857	0.260000	0.261429	0.260000
50%	0.274286	0.276667	0.272143	0.276667
75%	0.285714	0.293333	0.284286	0.293333
max	0.331429	0.373333	0.332857	0.380000

In [11]:

```
df.boxplot()
plt.title('linear simulation error rates')
```

Out[11]:

```
Text(0.5, 1.0, 'linear simulation error rates')
```



Question 3

QDA is better in both training and test set because the Bayes decision boundary is quadratic. We can tell that the mean LDA train and test error rate are around 0.27, while the mean QDA train and test error rates are around 0.26

In [9]:

```
lda_training = []
lda_test = []
qda_training = []
qda_test = []
for i in range(1000):
    x1 = np.random.uniform(-1,1,1000)
    x2 = np.random.uniform(-1,1,1000)
    Y_decision = x1+x2+x1*x1+x2*x2
    eps = np.random.normal(0,1,1000)
    y_simulated = x1+x2+x1*x1+x2*x2+eps
    label = []
    for i in range(len(Y_decision)):
        if (y_simulated[i] >= 0):
            label.append(1)
        else:
            label.append(0)

    X = np.column_stack((x1, x2))
    y = label
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,shuf
fle=True)
    clf_lda = LinearDiscriminantAnalysis()
    clf_lda.fit(X_train, y_train)
    LinearDiscriminantAnalysis()
    err_lda_train = 1 - clf_lda.score(X_train,y_train)
    err_lda_test = 1-clf_lda.score(X_test, y_test)
    lda_training.append(err_lda_train)
    lda_test.append(err_lda_test)
    #print(err_lda_train,err_lda_test)

    clf_qda = QuadraticDiscriminantAnalysis()
    clf_qda.fit(X_train, y_train)
    QuadraticDiscriminantAnalysis()
    err_qda_train = 1 - clf_qda.score(X_train,y_train)
    err_qda_test = 1-clf_qda.score(X_test, y_test)
    qda_training.append(err_qda_train)
    qda_test.append(err_qda_test)
    #print(err_qda_train,err_qda_test)
```

In [10]:

```
df = pd.DataFrame({'lda_training':lda_training,'lda_test':lda_test,'qda_trainin
g':qda_training,'qda_test':qda_test})
df.describe()
```

Out[10]:

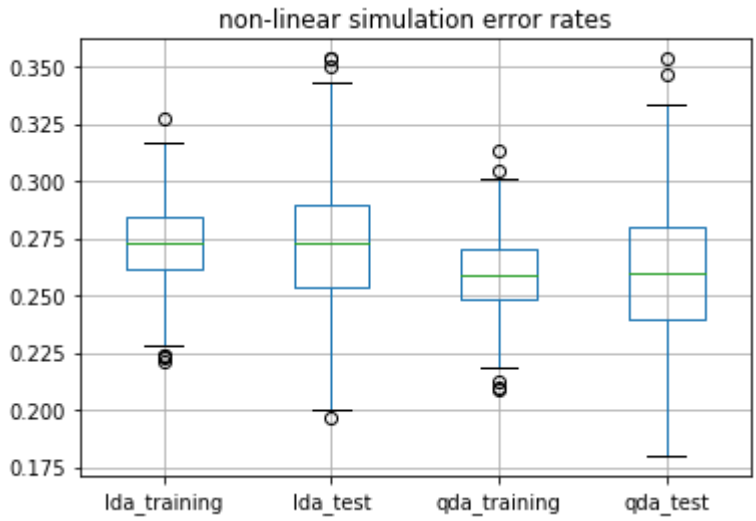
	lda_training	lda_test	qda_training	qda_test
count	1000.000000	1000.000000	1000.000000	1000.000000
mean	0.272536	0.273163	0.259271	0.260220
std	0.016824	0.026785	0.015826	0.026330
min	0.221429	0.196667	0.208571	0.180000
25%	0.261429	0.253333	0.248214	0.240000
50%	0.272857	0.273333	0.258571	0.260000
75%	0.284286	0.290000	0.270000	0.280000
max	0.327143	0.353333	0.312857	0.353333

In [12]:

```
df.boxplot()
plt.title('non-linear simulation error rates')
```

Out[12]:

```
Text(0.5, 1.0, 'non-linear simulation error rates')
```



Question4

the error rate of QDA relative to LDA decreases because QDA makes stronger assumption about the quadratic shape of the desicion boundary and has high variance, it benefits more when sample sizes increases.

In [36]:

```
def simulation_n(n):
    lda_training = []
    lda_test = []
    qda_training = []
    qda_test = []
    for i in range(1000):
        x1 = np.random.uniform(-1,1,n)
        x2 = np.random.uniform(-1,1,n)

        eps = np.random.normal(0,1,n)
        y_simulated = x1+x2+x1*x1+x2*x2+eps
        y = np.where(y_simulated>=0,True,False)
        X = np.column_stack((x1, x2))

        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
shuffle=True)
        clf_lda = LinearDiscriminantAnalysis()
        clf_lda.fit(X_train, y_train)
        LinearDiscriminantAnalysis()
        err_lda_train = 1 - clf_lda.score(X_train,y_train)
        err_lda_test = 1-clf_lda.score(X_test, y_test)
        lda_training.append(err_lda_train)
        lda_test.append(err_lda_test)
    #print(err_lda_train,err_lda_test)

        clf_qda = QuadraticDiscriminantAnalysis()
        clf_qda.fit(X_train, y_train)
        QuadraticDiscriminantAnalysis()
        err_qda_train = 1 - clf_qda.score(X_train,y_train)
        err_qda_test = 1-clf_qda.score(X_test, y_test)
        qda_training.append(err_qda_train)
        qda_test.append(err_qda_test)
    #print(err_qda_train,err_qda_test)
    return lda_test,qda_test
```

In [37]:

```
#simulate with different sizes
n1_lda,n1_qda = simulation_n(100)
n2_lda,n2_qda = simulation_n(1000)
n3_lda,n3_qda = simulation_n(10000)
n4_lda,n4_qda = simulation_n(100000)
```

In [44]:

```
n1_lda,n1_qda = simulation_n(100)
print(np.mean(n1_qda)/np.mean(n1_lda))
```

```
0.9473744422834914
```

In [38]:

```
print(np.mean(n1_lda),np.mean(n1_qda),np.mean(n2_lda),np.mean(n2_qda),np.mean(n3
_lda),np.mean(n3_qda),np.mean(n4_lda),np.mean(n4_qda))
```

```
0.288 0.2713 0.27448666666666666 0.2612233333333333 0.27335800000000
005 0.26037299999999997 0.27362366666666665 0.2607739333333333
```

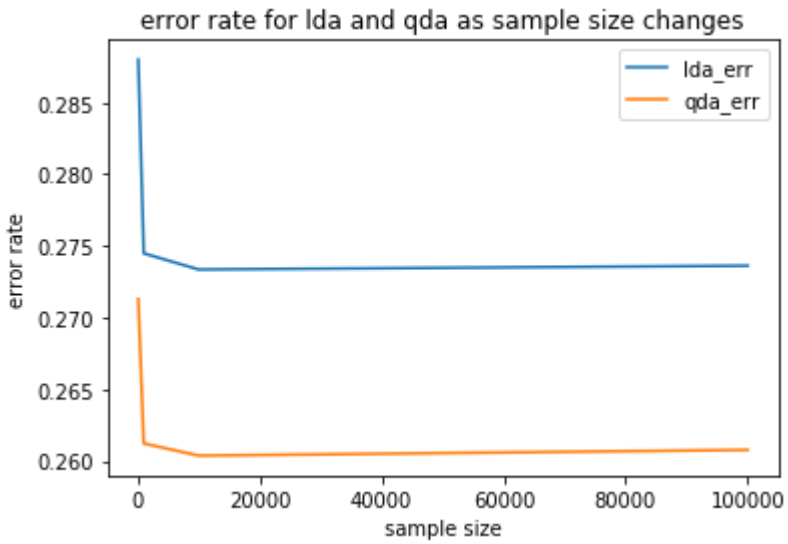
In [39]:

```
lda_err = [np.mean(n1_lda),np.mean(n2_lda),np.mean(n3_lda),np.mean(n4_lda)]
qda_err = [np.mean(n1_qda),np.mean(n2_qda),np.mean(n3_qda),np.mean(n4_qda)]
print(lda_err,qda_err)
size = [100,1000,10000,100000]
new = []
for i in range(len(lda_err)):
    new.append(qda_err[i]/lda_err[i])
print(new)
```

```
[0.288, 0.27448666666666666, 0.27335800000000005, 0.2736236666666666
5] [0.2713, 0.2612233333333333, 0.26037299999999997, 0.2607739333333
333]
[0.9420138888888889, 0.9516794987006048, 0.9524981891878047, 0.95303
86625913206]
```

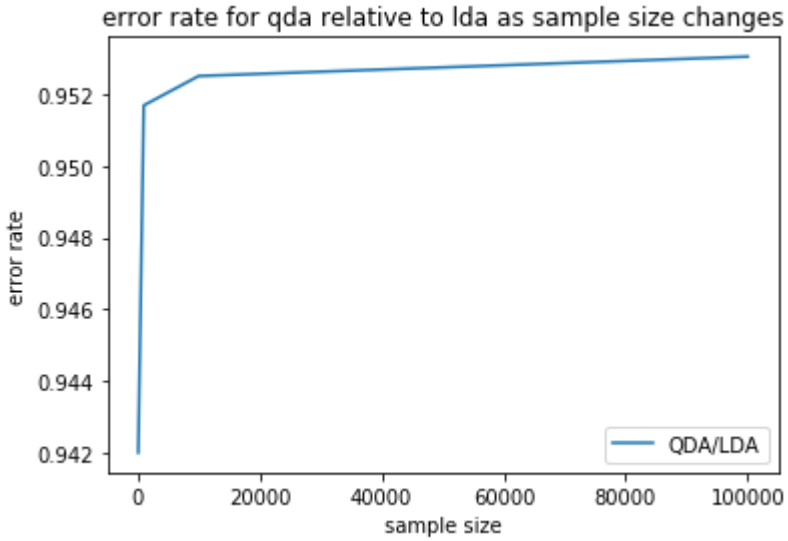
In [40]:

```
#plot error rate as it changes over different sample sizes
import matplotlib.pyplot as plt
plt.plot(size,lda_err,label = 'lda_err')
plt.plot(size,qda_err,label = 'qda_err')
plt.legend()
plt.xlabel('sample size')
# naming the y axis
plt.ylabel('error rate')
# giving a title to my graph
plt.title('error rate for lda and qda as sample size changes')
plt.show()
```



In [41]:

```
plt.plot(size,new,label = 'QDA/LDA')
plt.legend()
plt.xlabel('sample size')
# naming the y axis
plt.ylabel('error rate')
# giving a title to my graph
plt.title('error rate for qda relative to lda as sample size changes')
plt.show()
```



Question 5

the LDA model performs the best because it has the lowest error rate 0.26 and highest roc

In [15]:

```
df = pd.read_csv('/Users/lijiaxuan/Downloads/problem-set-2-master/mental_health.csv')
df = df.dropna()
print(df)
X = df.as_matrix(columns=df.columns[1:])
y = df['vote96'].values
```

```
      vote96  mhealth_sum   age  educ  black  female  married    inc
10
0        1.0          0.0  60.0  12.0      0       0      0.0   4.81
49
2        1.0          1.0  36.0  12.0      0       0      1.0   8.82
73
3        0.0          7.0  21.0  13.0      0       0      0.0   1.73
87
7        0.0          6.0  29.0  13.0      0       0      0.0  10.69
98
11       1.0          1.0  41.0  15.0      1       1      1.0   8.82
73
...      ...          ...   ...   ...    ...     ...      ...
...
2822     1.0          2.0  37.0  14.0      0       0      1.0   5.88
49
2823     1.0          2.0  30.0  12.0      0       1      1.0   3.47
74
2828     1.0          1.0  40.0  12.0      0       1      0.0   1.73
87
2829     1.0          2.0  73.0   6.0      0       0      1.0   2.27
37
2830     1.0          4.0  47.0  12.0      0       0      0.0   3.47
74

[1165 rows x 8 columns]

/Users/lijiaxuan/anaconda3/lib/python3.7/site-packages/ipykernel_lau
ncher.py:4: FutureWarning: Method .as_matrix will be removed in a fu
ture version. Use .values instead.
  after removing the cwd from sys.path.
```

In [16]:

```
#split dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,shuffle=
True)
#train different models
clf_lo = LogisticRegression()
clf_lo.fit(X_train, y_train)
clf_lda = LinearDiscriminantAnalysis()
clf_lda.fit(X_train, y_train)
clf_qda = QuadraticDiscriminantAnalysis()
clf_qda.fit(X_train, y_train)
gnb = GaussianNB()
gnb.fit(X_train,y_train)

n_neighbors = range(1,11)
clf_knn_list = []
for i in n_neighbors:
    clf_knn = KNeighborsClassifier(n_neighbors = i,metric = 'euclidean')
    clf_knn.fit(X_train, y_train)
    clf_knn_list.append(clf_knn)
```

```
/Users/lijiaxuan/anaconda3/lib/python3.7/site-packages/sklearn/linea
r_model/logistic.py:432: FutureWarning: Default solver will be chang
ed to 'lbfgs' in 0.22. Specify a solver to silence this warning.
  FutureWarning)
```

In [17]:

```
def cal_err(function):
    return 1-function.score(X_test,y_test)
```

In [18]:

```
#calculate error rate for logistic regression
print('logistic regression:',cal_err(clf_lo))
print('LDA:',cal_err(clf_lda))
print('QDA:',cal_err(clf_qda))
print('Nayes Classifier:',cal_err(gnb))
for i in range(len(clf_knn_list)):
    print('KNN',i+1,cal_err(clf_knn_list[i]))
```

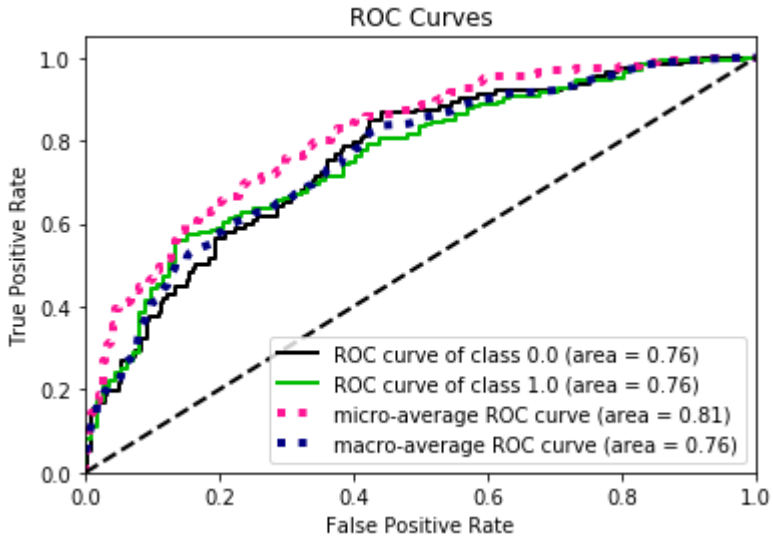
```
logistic regression: 0.2828571428571428
LDA: 0.26
QDA: 0.2914285714285715
Nayes Classifier: 0.2857142857142857
KNN 1 0.34571428571428575
KNN 2 0.4228571428571428
KNN 3 0.36571428571428577
KNN 4 0.37428571428571433
KNN 5 0.3285714285714286
KNN 6 0.3514285714285714
KNN 7 0.3028571428571428
KNN 8 0.3085714285714286
KNN 9 0.2828571428571428
KNN 10 0.30000000000000004
```

In [19]:

```
def plot_roc(function):
    y_true = y_test
    y_probas = function.predict_proba(X_test)
    skplt.metrics.plot_roc(y_true, y_probas)
    plt.show()
```

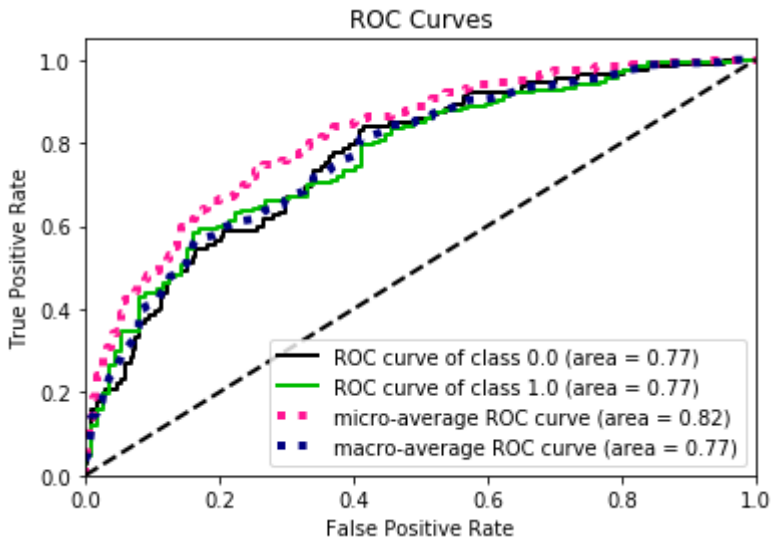
In [20]:

```
#plot roc for logistic regression
plot_roc(clf_lo)
```



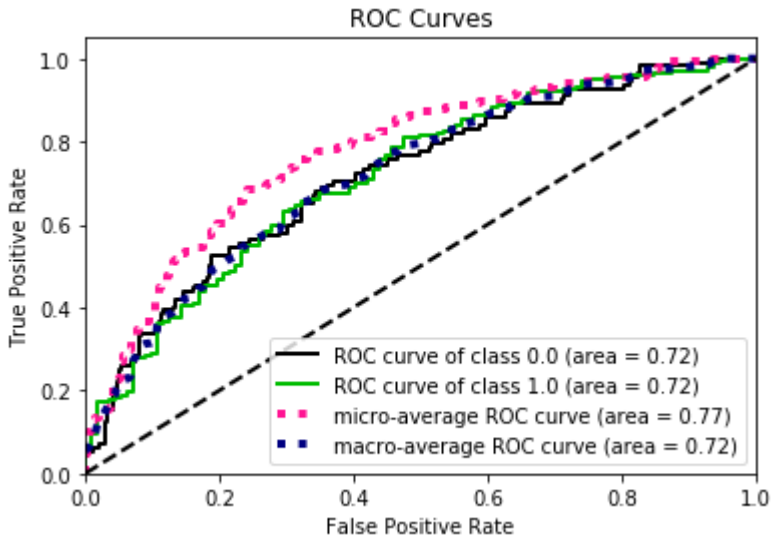
In [21]:

```
#plot roc for lda
plot_roc(clf_lda)
```



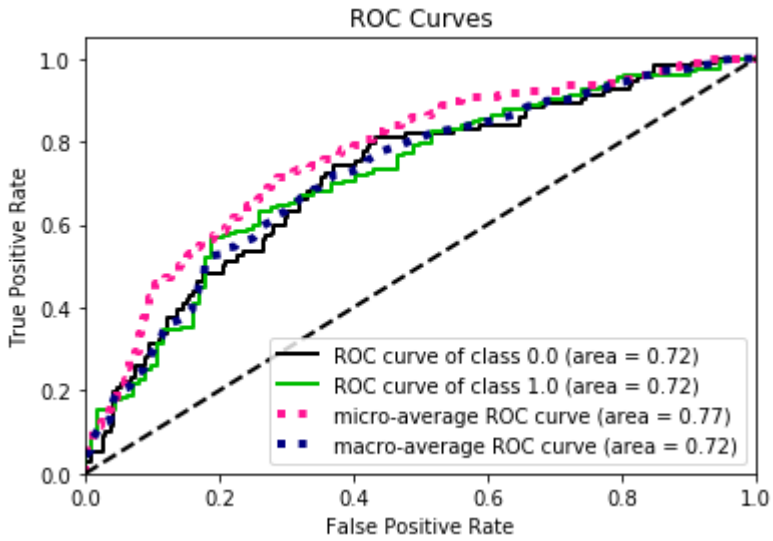
In [22]:

```
#plot roc for qda
plot_roc(clf_qda)
```



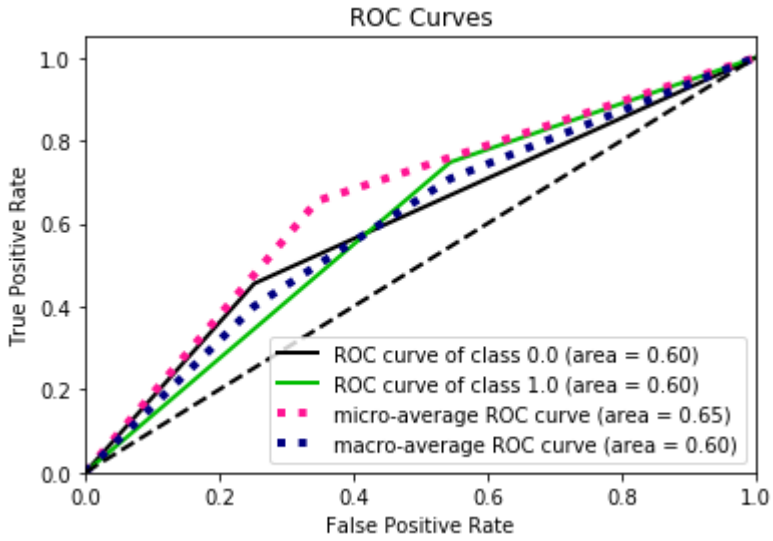
In [23]:

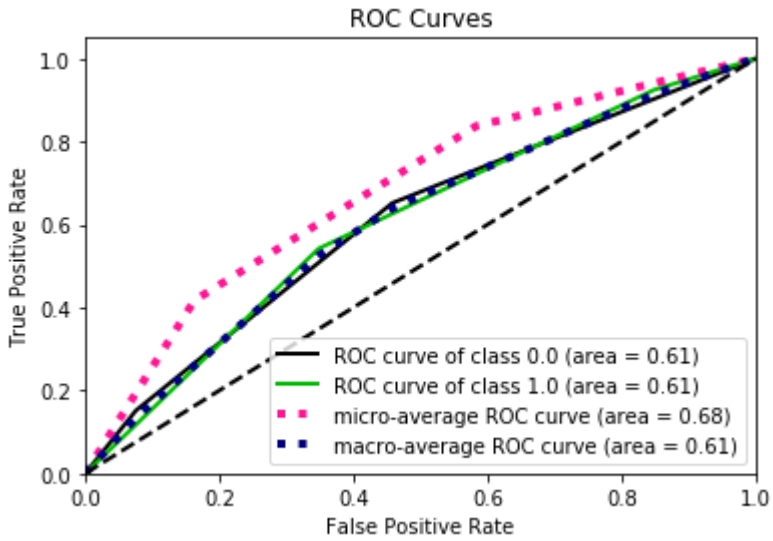
```
plot_roc(gnb)
```

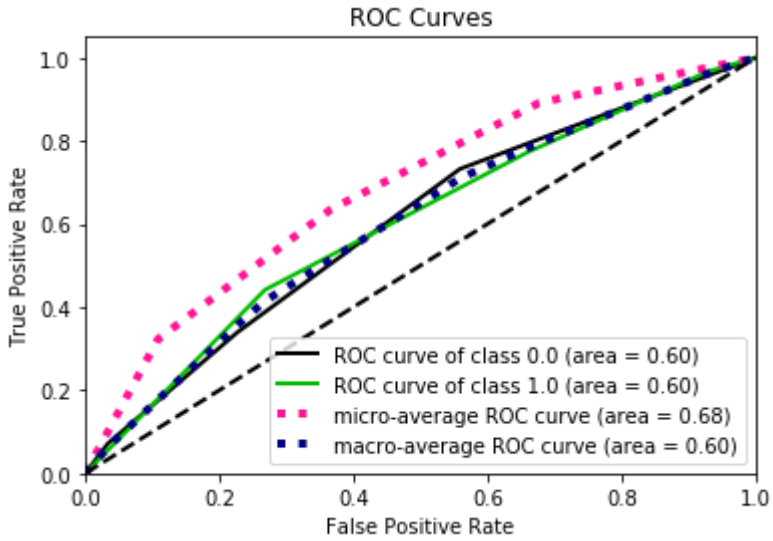


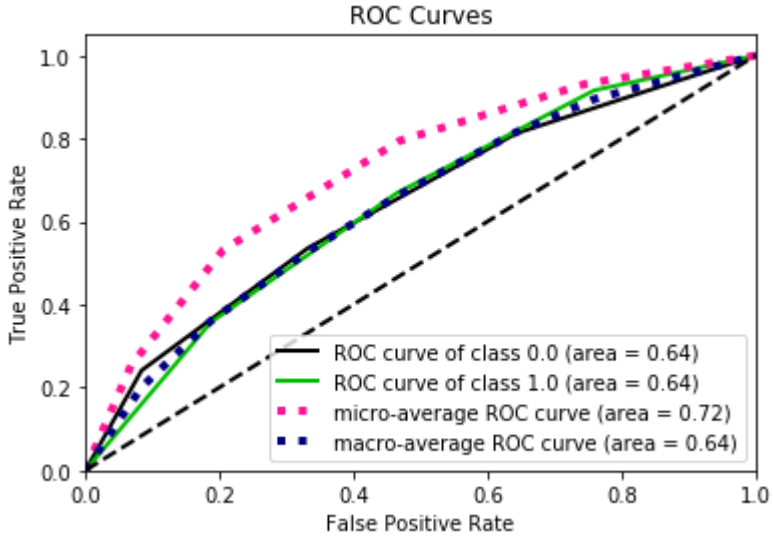
In [24]:

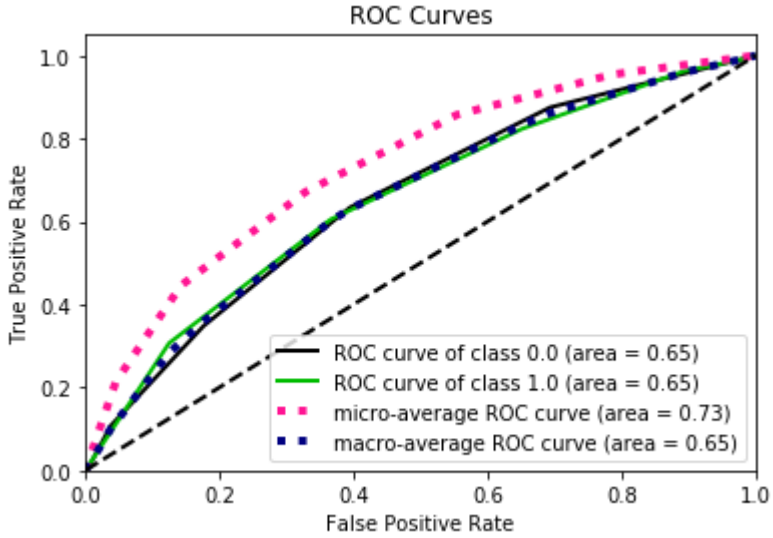
```
for item in clf_knn_list:
    plot_roc(item)
```

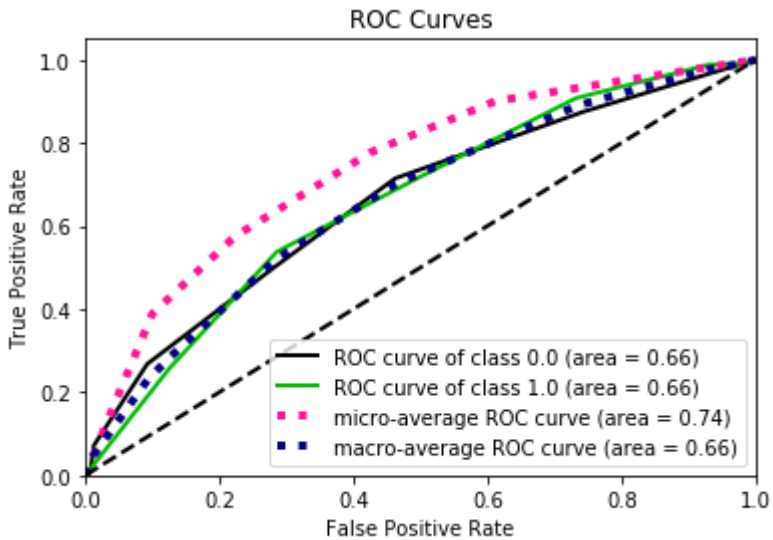

ROC Curves
True Positive Rate
False Positive Rate
ROC curve of class 0.0 (area = 0.60)
ROC curve of class 1.0 (area = 0.60)
micro-average ROC curve (area = 0.65)
macro-average ROC curve (area = 0.60)

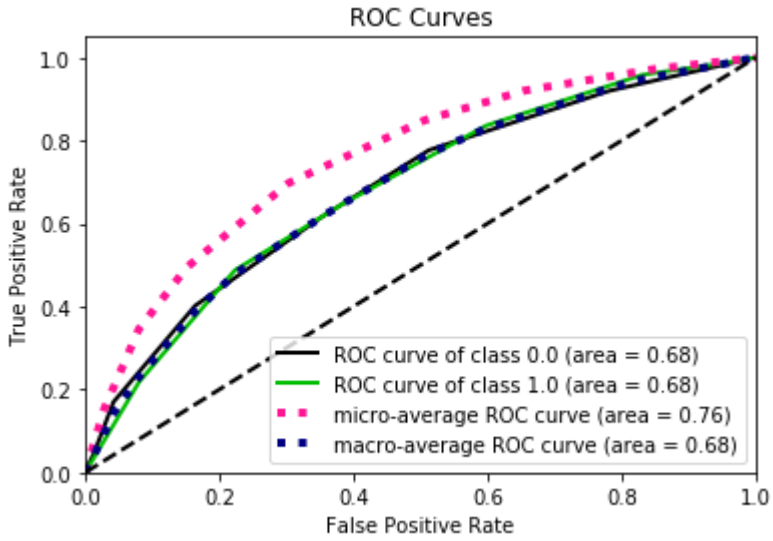

ROC Curves
True Positive Rate
False Positive Rate
ROC curve of class 0.0 (area = 0.61)
ROC curve of class 1.0 (area = 0.61)
micro-average ROC curve (area = 0.68)
macro-average ROC curve (area = 0.61)

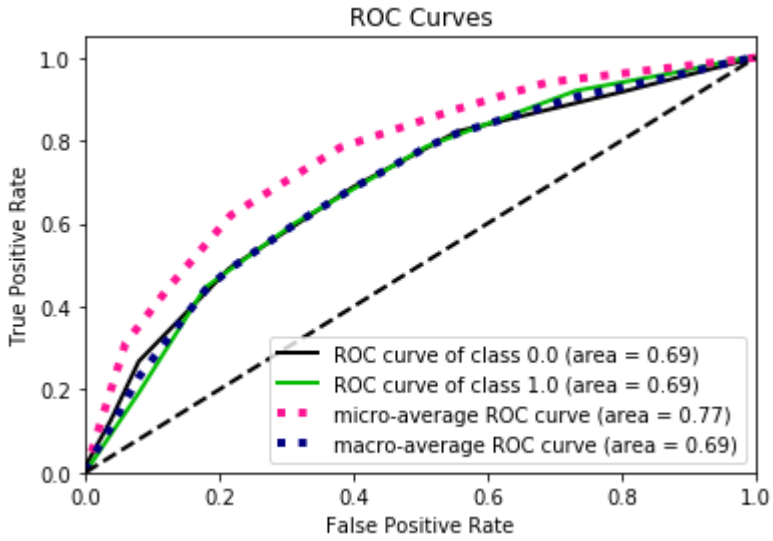

ROC Curves
True Positive Rate
False Positive Rate
ROC curve of class 0.0 (area = 0.60)
ROC curve of class 1.0 (area = 0.60)
micro-average ROC curve (area = 0.68)
macro-average ROC curve (area = 0.60)

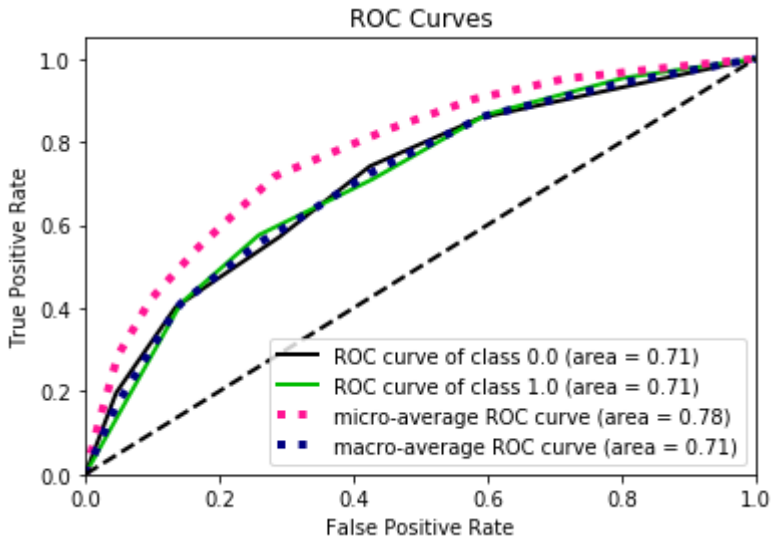

ROC Curves
True Positive Rate
False Positive Rate
ROC curve of class 0.0 (area = 0.64)
ROC curve of class 1.0 (area = 0.64)
micro-average ROC curve (area = 0.72)
macro-average ROC curve (area = 0.64)

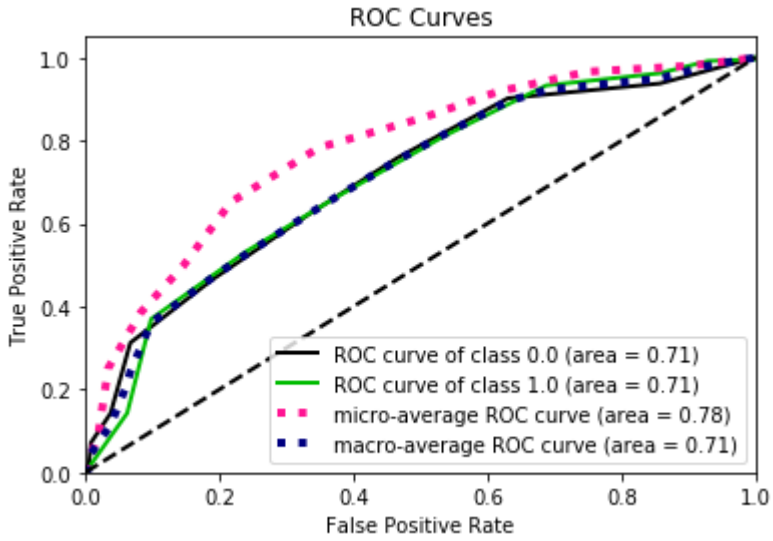

ROC Curves
True Positive Rate
False Positive Rate
ROC curve of class 0.0 (area = 0.65)
ROC curve of class 1.0 (area = 0.65)
micro-average ROC curve (area = 0.73)
macro-average ROC curve (area = 0.65)


ROC Curves
True Positive Rate
False Positive Rate
ROC curve of class 0.0 (area = 0.66)
ROC curve of class 1.0 (area = 0.66)
micro-average ROC curve (area = 0.74)
macro-average ROC curve (area = 0.66)


ROC Curves
True Positive Rate
False Positive Rate
ROC curve of class 0.0 (area = 0.68)
ROC curve of class 1.0 (area = 0.68)
micro-average ROC curve (area = 0.76)
macro-average ROC curve (area = 0.68)


ROC Curves
True Positive Rate
False Positive Rate
ROC curve of class 0.0 (area = 0.69)
ROC curve of class 1.0 (area = 0.69)
micro-average ROC curve (area = 0.77)
macro-average ROC curve (area = 0.69)


ROC Curves
True Positive Rate
False Positive Rate
ROC curve of class 0.0 (area = 0.71)
ROC curve of class 1.0 (area = 0.71)
micro-average ROC curve (area = 0.78)
macro-average ROC curve (area = 0.71)


ROC Curves
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
ROC curve of class 0.0 (area = 0.71)
ROC curve of class 1.0 (area = 0.71)
micro-average ROC curve (area = 0.78)
macro-average ROC curve (area = 0.71)