HW2

Jiaxuan Li

Question1

In [97]:

```
import random
import numpy as np
import math
import matplotlib.pyplot as plt
```

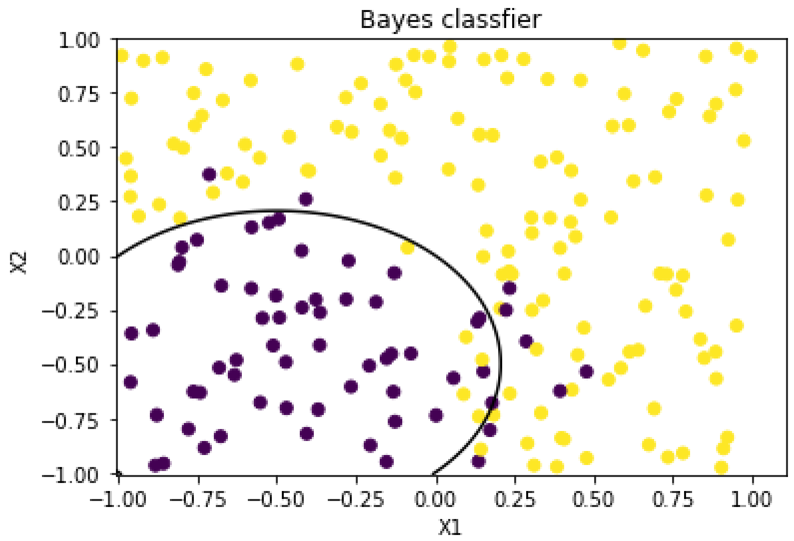
In [104]:

```
#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, prob, levels=[0.5], colors='black')
plt.xlabel('X1')
plt.ylabel('X2')
plt.title('Bayes classfier')
```

Out[104]:

```
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 has high variance so it performs better in training set.

In [115]:

```
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis
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)
    Y_decision = x1+x2
    eps = np.random.normal(0,1,1000)
    #simulate y
    y_simulated = x1+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
    #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 [116]:

```
#summarize simulation's error rates in tabular form
print(np.mean(lda_training),np.mean(lda_test),np.mean(qda_training),np.mean(qda_
test))
from tabulate import tabulate
print(tabulate([['LDA train', np.mean(lda_training)], ['LDA test', np.mean(lda_t
est)],['QDA train',np.mean(qda_training)],['QDA test',np.mean(qda_test)]], heade
rs=['dataset', 'Error rate']))
```

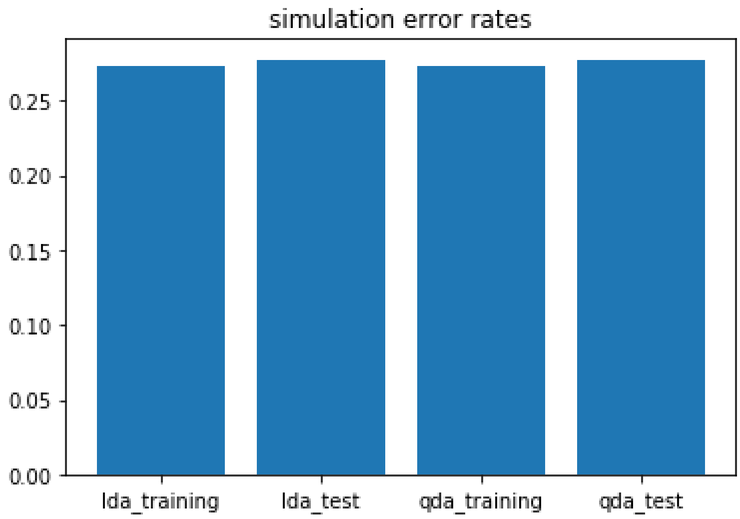
```
0.2738357142857143 0.2772766666666666 0.27306142857142857 0.2775
dataset      Error rate
---------  ------------
LDA train      0.273836
LDA test       0.277277
QDA train      0.273061
QDA test       0.2775
```

In [117]:

```
error_rate_list = [np.mean(lda_training),np.mean(lda_test),np.mean(qda_training
),np.mean(qda_test)]
name = ['lda_training','lda_test','qda_training','qda_test']
plt.bar(name,error_rate_list)
plt.title('simulation error rates')
```

Out[117]:

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



Question 3

QDA is better in both training and test set because the Bayes decision boundary is quadratic

In [122]:

```
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis
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 [123]:

```
print(np.mean(lda_training),np.mean(lda_test),np.mean(qda_training),np.mean(qda_
test))
```

```
0.2734785714285714 0.27475666666666665 0.25975714285714285 0.2615
```

In [124]:

```
#summarize simulation's error rates in tabular form
print(np.mean(lda_training),np.mean(lda_test),np.mean(qda_training),np.mean(qda_
test))
from tabulate import tabulate
print(tabulate([['LDA train', np.mean(lda_training)], ['LDA test', np.mean(lda_t
est)],['QDA train',np.mean(qda_training)],['QDA test',np.mean(qda_test)]], heade
rs=['dataset', 'Error rate']))
```

```
0.2734785714285714 0.27475666666666665 0.25975714285714285 0.2615
dataset      Error rate
---------  ------------
LDA train      0.273479
LDA test       0.274757
QDA train      0.259757
QDA test       0.2615
```

In [125]:

```
error_rate_list = [np.mean(lda_training),np.mean(lda_test),np.mean(qda_training
),np.mean(qda_test)]
name = ['lda_training','lda_test','qda_training','qda_test']
plt.bar(name,error_rate_list)
plt.title('simulation error rates')
```

Out[125]:

```
Text(0.5, 1.0, '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 [45]:

```
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis
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+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 [46]:

```
#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 [47]:

```
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))
```

```
0.2877 0.2880333333333333 0.27681 0.27742333333333336 0.275314666666
66665 0.27537933333333336 0.27538463333333335 0.27538890000000005
```

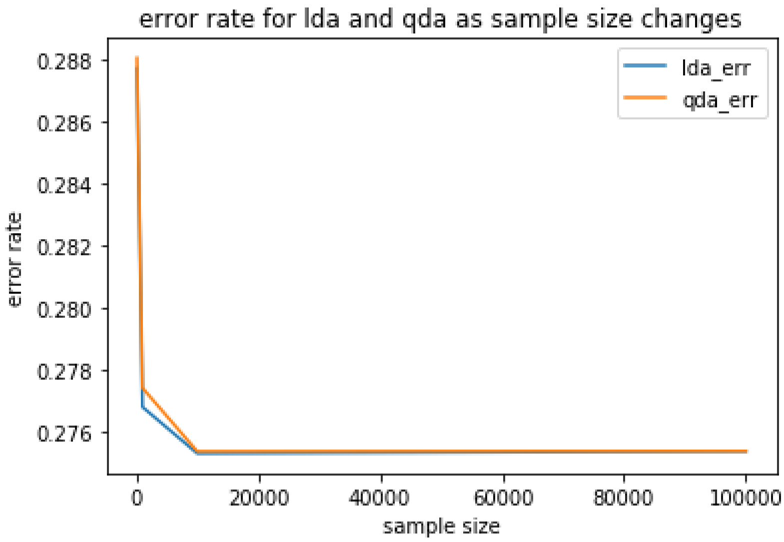
In [48]:

```
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]
```

```
[0.2877, 0.27681, 0.27531466666666665, 0.27538463333333335] [0.28803
33333333333, 0.27742333333333336, 0.27537933333333336, 0.27538890000
000005]
```

In [49]:

```
#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()
```



Question 5

the LDA model performs the best because it has the lowest error rate 0.263

In [111]:

```
import pandas as pd
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:5: FutureWarning: Method .as_matrix will be removed in a fu
ture version. Use .values instead.
  """
```

In [112]:

```
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
#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(random_state=0)
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 [15]:

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

In [91]:

```
#calculate error rate for logistic regression
cal_err(clf_lo)
```

Out[91]:

```
0.2657142857142857
```

In [94]:

```
#calculate error rate for lda
cal_err(clf_lda)
```

Out[94]:

```
0.2628571428571429
```

In [96]:

```
#calculate error rate for qda
cal_err(clf_qda)
```

Out[96]:

```
0.28
```

In [98]:

```
#calculate error rate for naive bayes
cal_err(gnb)
```

Out[98]:

```
0.30000000000000004
```

In [16]:

```
#calculate error rate for knn
for item in clf_knn_list:
    print(cal_err(item))
```

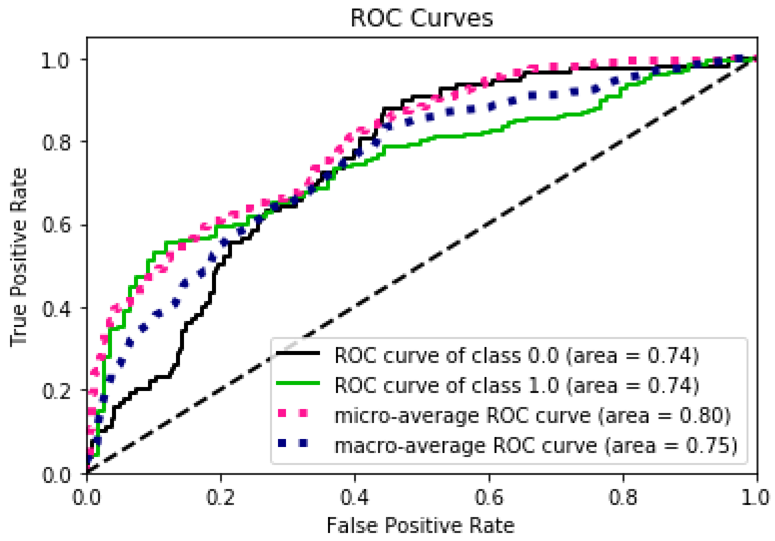
```
0.3085714285714286
0.41428571428571426
0.3371428571428572
0.3571428571428571
0.34571428571428575
0.3371428571428572
0.3342857142857143
0.3371428571428572
0.34285714285714286
0.3285714285714286
```

In [105]:

```
import scikitplot as skplt
import matplotlib.pyplot as plt
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()
```

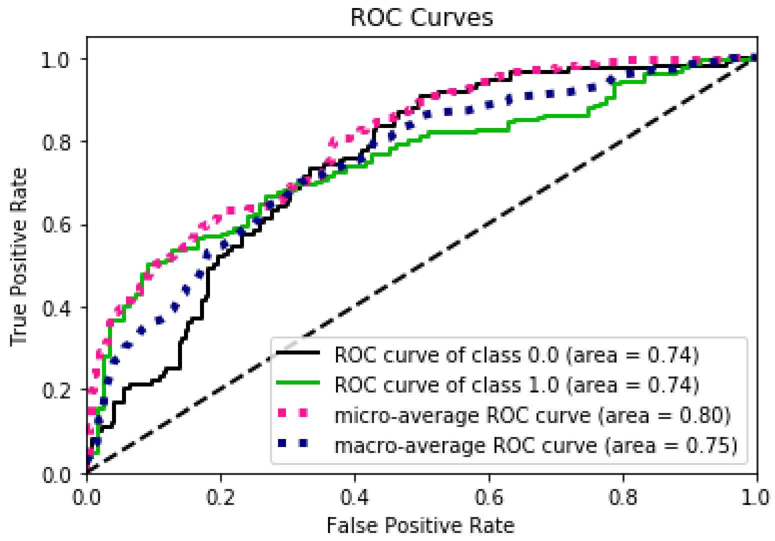
In [106]:

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



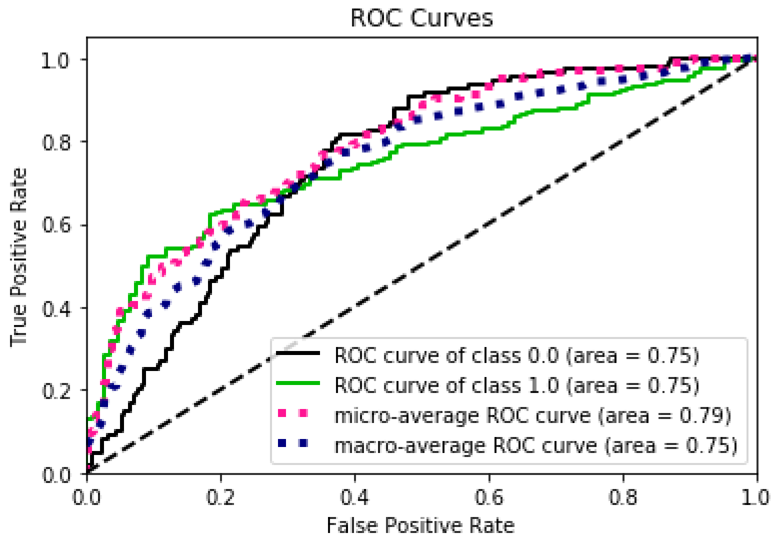
In [107]:

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



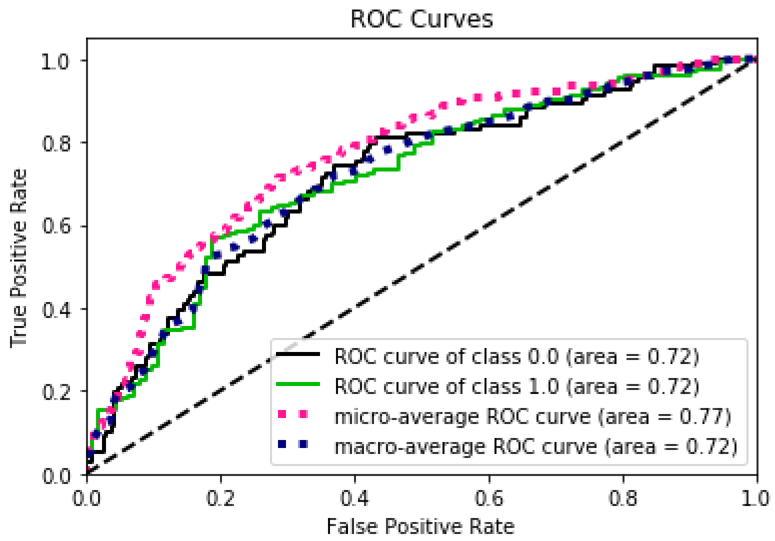
In [108]:

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



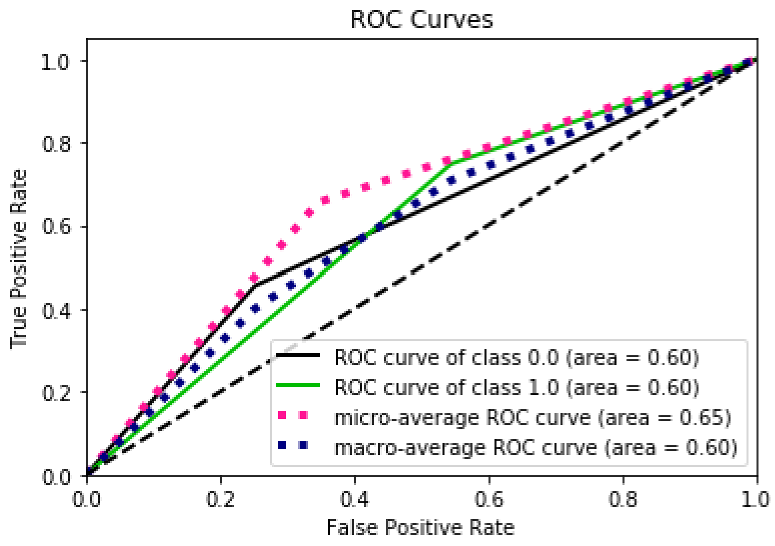
In [113]:

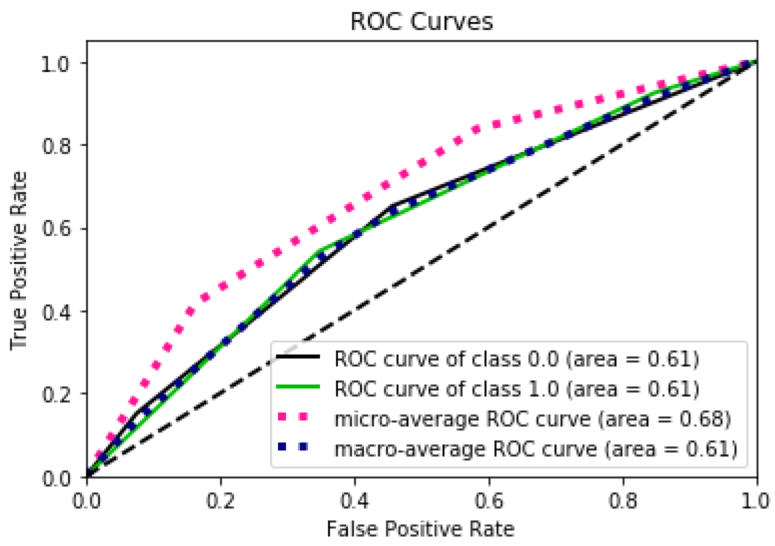
```
plot_roc(gnb)
```

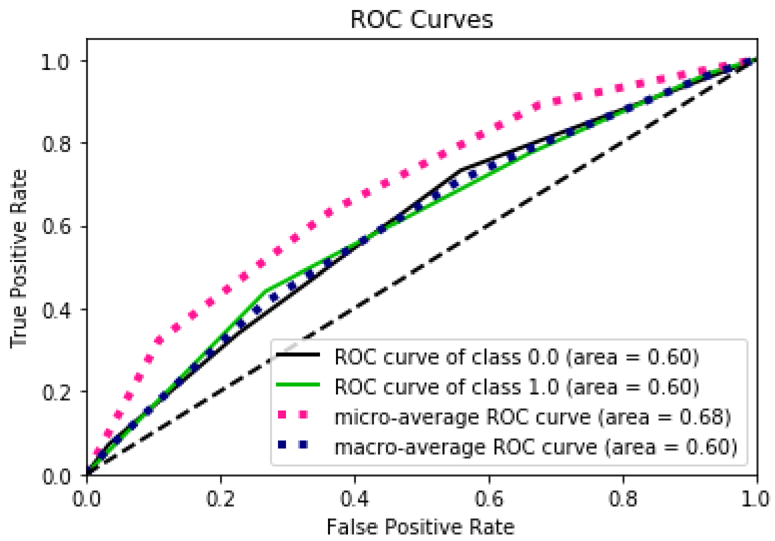


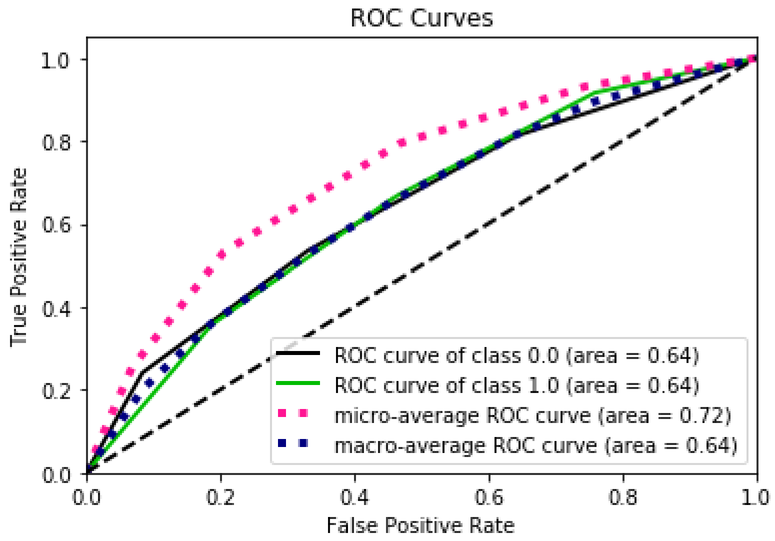
In [114]:

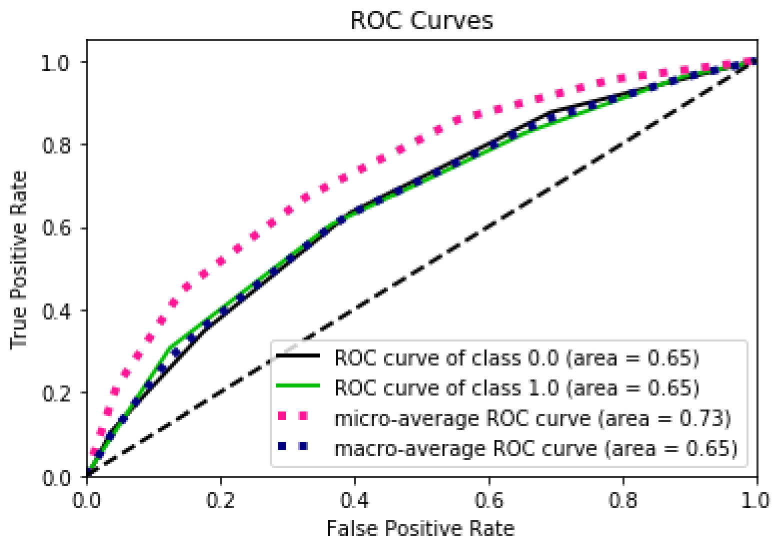
```
for item in clf_knn_list:
    plot_roc(item)
```

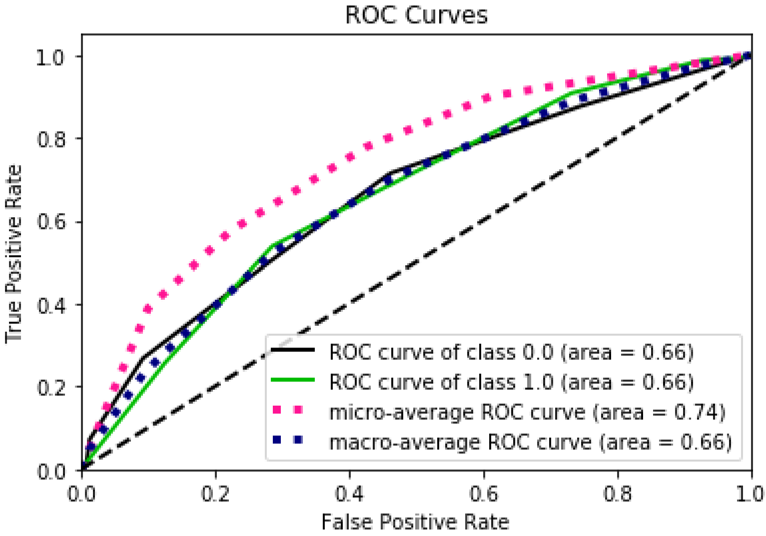

ROC Curves
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
ROC curve of class 0.0 (area = 0.60)
ROC curve of class 1.0 (area = 0.60)
micro-average ROC curve (area = 0.65)
macro-average ROC curve (area = 0.60)

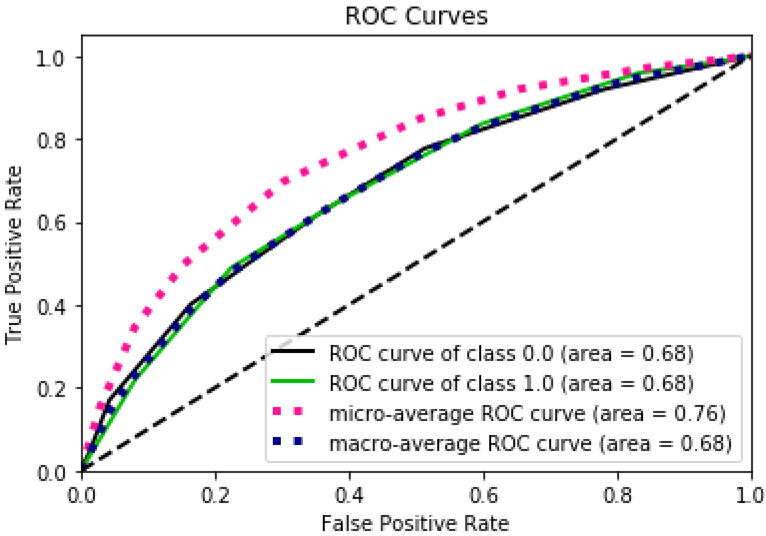

ROC Curves
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
ROC curve of class 0.0 (area = 0.61)
ROC curve of class 1.0 (area = 0.61)
micro-average ROC curve (area = 0.68)
macro-average ROC curve (area = 0.61)

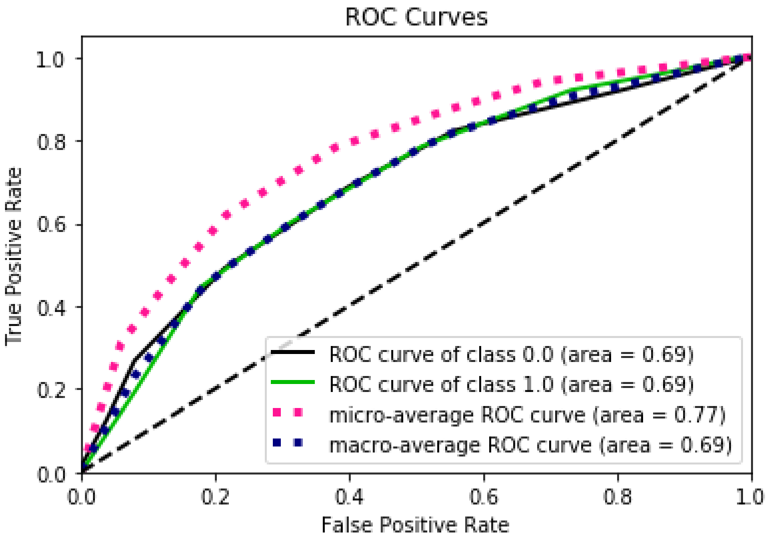

ROC Curves
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
ROC curve of class 0.0 (area = 0.60)
ROC curve of class 1.0 (area = 0.60)
micro-average ROC curve (area = 0.68)
macro-average ROC curve (area = 0.60)

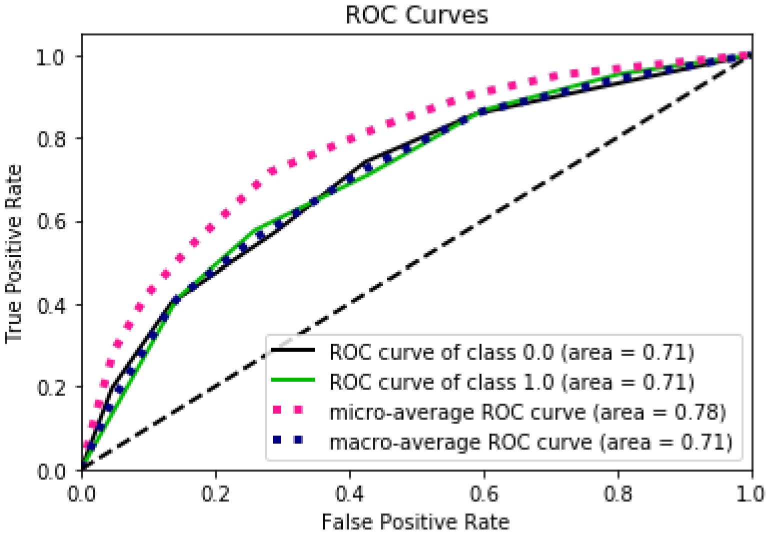

ROC Curves
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
ROC curve of class 0.0 (area = 0.64)
ROC curve of class 1.0 (area = 0.64)
micro-average ROC curve (area = 0.72)
macro-average ROC curve (area = 0.64)

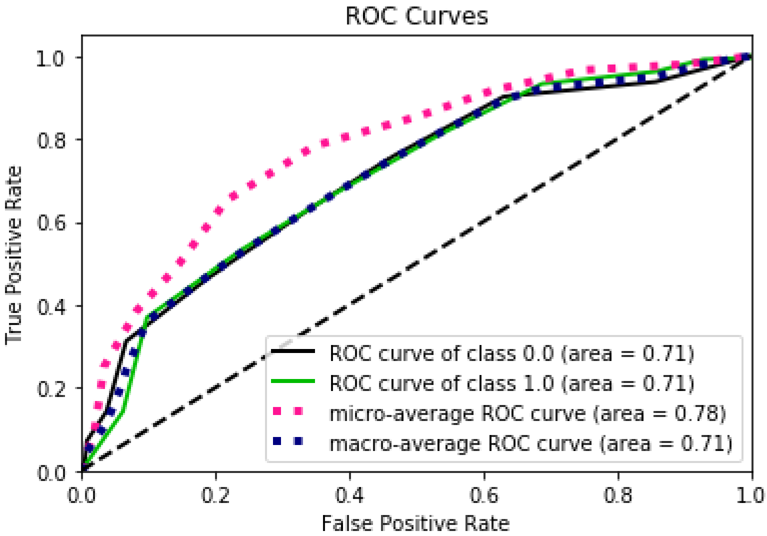

ROC Curves
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
ROC curve of class 0.0 (area = 0.65)
ROC curve of class 1.0 (area = 0.65)
micro-average ROC curve (area = 0.73)
macro-average ROC curve (area = 0.65)


ROC Curves
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
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
0.0
0.2
0.4
0.6
0.8
1.0
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
0.0
0.2
0.4
0.6
0.8
1.0
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
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)


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)