Xu_Weijie_HW2

February 2, 2020

Name: Weijie Xu
CNetID: weijiexu
Student ID: 12245277

1 The Bayes Classifier

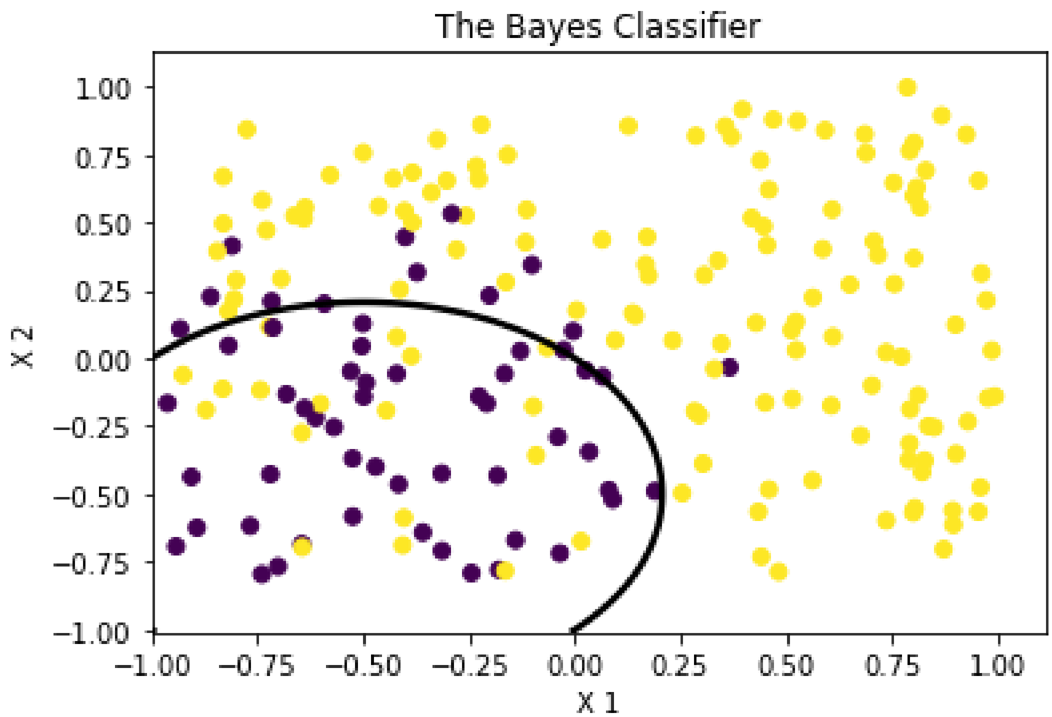
1.1 Problem 1

[1]:
```
import numpy as np
import pandas as pd
import random
import matplotlib.pyplot as plt
from statistics import mean
from sklearn.model_selection import train_test_split
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_curve, auc
```

[2]:
```
SEED = 970608
```

[3]:
```
np.random.seed(SEED)
x1 = np.random.uniform(-1, 1, 200)
x2 = np.random.uniform(-0.8, 1, 200)
X = np.stack((x1, x2), axis=-1)
y = x1 + x1 ** 2 + x2 + x2 ** 2 + np.random.normal(0, 0.5, 200)
odds = np.exp(y)
prob = odds / (1 + odds)
if_success = np.where(prob > 0.5, 1, 0)

X1 = np.linspace(-1, 1, 200)
X2 = np.linspace(-1.01, 1.01, 200)
X1, X2 = np.meshgrid(X1, X2)
y = X1 + X1 ** 2 + X2 + X2 ** 2
odds = np.exp(y)
prob = odds / (1 + odds)
```

```
plt.contour(X1, X2, prob, levels=[0.5], colors='black', linewidths=2.5)
plt.scatter(x1, x2, c=if_success)
plt.xlabel('X 1')
plt.ylabel('X 2')
plt.title('The Bayes Classifier')
plt.show()
```



2 Exploring Simulated Differences between LDA and QDA

2.1 Problem 2

As for the training error, when the decision boundary is linear, as illustrated in the table and the figure below, the performance of QDA is a little bit better than LDA. This is because the quadratic term in QDA allow the model to be more flexible to fit the training data.
However, as for the test error, the result below in Table 1 and Figure 1 shows that LDA performs, in turn, a bit better than QDA. This is probabily because the QDA model more flexible than necessary to fit the linear decision boundary in this scenario, which would lead the model to the problem of over-fitting.

[4]:
```
def calculate_lda_error(x1, x2, y):
    '''
    Calculate both the training error and the test error of lda model.
    '''
```

```
    X = np.stack((x1, x2), axis=-1)
    y = np.where(y >= 0, True, False)
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
 ↪train_size=0.7)

    clf = LinearDiscriminantAnalysis()
    clf.fit(X_train, y_train)
    train_error, test_error = 1 - clf.score(X_train, y_train), 1 - clf.
 ↪score(X_test, y_test)

    return train_error, test_error
```

[5]:
```
def calculate_qda_error(x1, x2, y):
    '''
    Calculate both the training error and the test error of qda model.
    '''

    X = np.stack((x1, x2), axis=-1)
    y = np.where(y >= 0, True, False)
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
 ↪train_size=0.7)

    clf = QuadraticDiscriminantAnalysis()
    clf.fit(X_train, y_train)
    train_error, test_error = 1 - clf.score(X_train, y_train), 1 - clf.
 ↪score(X_test, y_test)

    return train_error, test_error
```

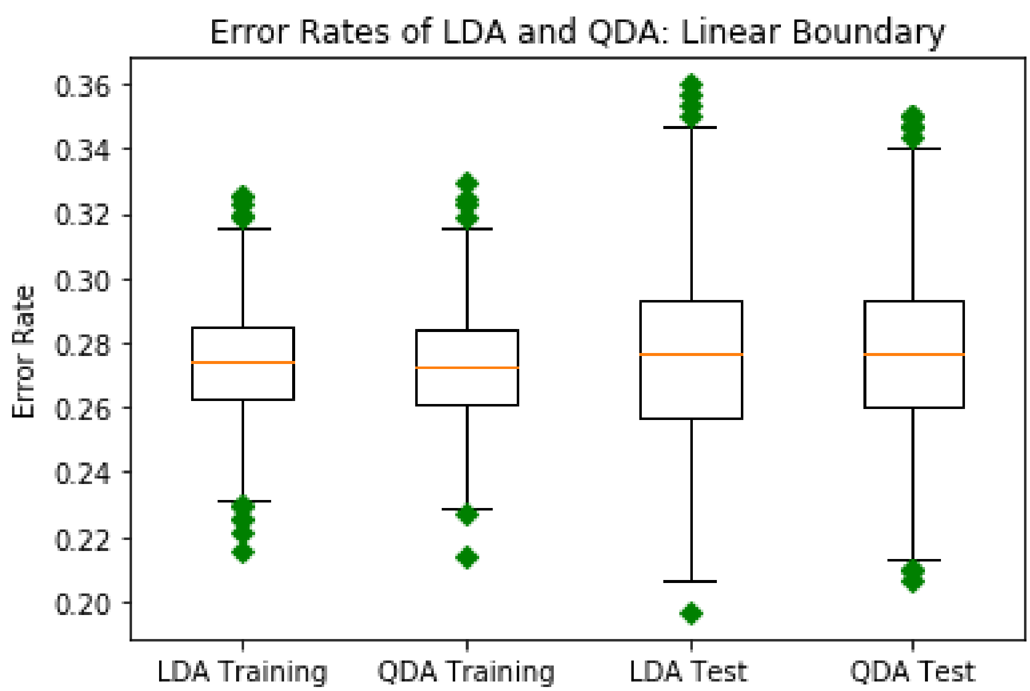
[6]:
```
np.random.seed(SEED)
train_error_lda, test_error_lda, train_error_qda, test_error_qda = [], [], [],␣
 ↪[]
for i in range(1000):
    x1 = np.random.uniform(-1, 1, 1000)
    x2 = np.random.uniform(-1, 1, 1000)
    y = x1 + x2 + np.random.normal(0, 1, 1000)
    train_error_lda.append(calculate_lda_error(x1, x2, y)[0])
    test_error_lda.append(calculate_lda_error(x1, x2, y)[1])
    train_error_qda.append(calculate_qda_error(x1, x2, y)[0])
    test_error_qda.append(calculate_qda_error(x1, x2, y)[1])
avg_train_lda = mean(train_error_lda)
avg_test_lda = mean(test_error_lda)
avg_train_qda = mean(train_error_qda)
avg_test_qda = mean(test_error_qda)
```

[7]:
```
train_err_dict = {'LDA': avg_train_lda, 'QDA':avg_train_qda}
test_err_dict = {'LDA': avg_test_lda, 'QDA': avg_test_qda}
accurc_dict_p2 = {'Average Training Error': train_err_dict, 'Average Test␣
↪Error':test_err_dict}
pd.DataFrame(accurc_dict_p2)
```

[7]:

	Average Training Error	Average Test Error
LDA	0.273376	0.276830
QDA	0.272607	0.277437

[8]:
```
data = [train_error_lda, train_error_qda, test_error_lda, test_error_qda]
label = ['LDA Training', 'QDA Training', 'LDA Test', 'QDA Test']
plt.boxplot(data, 0, 'gD', labels=label)
plt.title('Error Rates of LDA and QDA: Linear Boundary')
plt.ylabel('Error Rate')
plt.show()
```



2.2 Problem 3

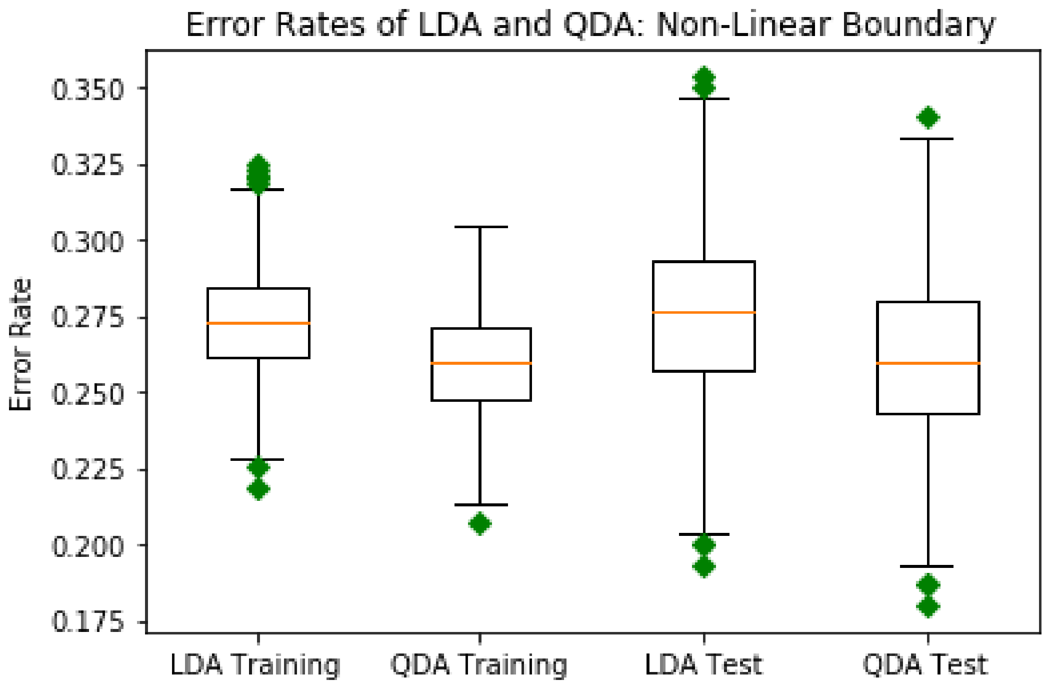
When the decision boundary is non-linear, as illustrated in the table and the figure below, the performance of QDA is better than LDA in terms of both training and test data. This is mainly because the LDA, which is lack of quadratic terms, is not flexible enough to account for this non-linear scenario.

[9]:
```
np.random.seed(SEED)
train_error_lda, test_error_lda, train_error_qda, test_error_qda = [], [], [],␣
 ↪[]
for i in range(1000):
    x1 = np.random.uniform(-1, 1, 1000)
    x2 = np.random.uniform(-1, 1, 1000)
    y = x1 + x1 ** 2 + x2 + x2 ** 2 + np.random.normal(0, 1, 1000)
    train_error_lda.append(calculate_lda_error(x1, x2, y)[0])
    test_error_lda.append(calculate_lda_error(x1, x2, y)[1])
    train_error_qda.append(calculate_qda_error(x1, x2, y)[0])
    test_error_qda.append(calculate_qda_error(x1, x2, y)[1])
avg_train_lda = mean(train_error_lda)
avg_test_lda = mean(test_error_lda)
avg_train_qda = mean(train_error_qda)
avg_test_qda = mean(test_error_qda)
```

[10]:
```
train_err_dict = {'LDA': avg_train_lda, 'QDA':avg_train_qda}
test_err_dict = {'LDA': avg_test_lda, 'QDA': avg_test_qda}
accurc_dict_p3 = {'Average Training Error': train_err_dict, 'Average Test␣
 ↪Error':test_err_dict}
pd.DataFrame(accurc_dict_p3)
```

[10]:
```
     Average Training Error  Average Test Error
LDA                0.273010            0.275823
QDA                0.259014            0.261143
```

[11]:
```
data = [train_error_lda, train_error_qda, test_error_lda, test_error_qda]
label = ['LDA Training', 'QDA Training', 'LDA Test', 'QDA Test']
plt.boxplot(data, 0, 'gD', labels=label)
plt.title('Error Rates of LDA and QDA: Non-Linear Boundary')
plt.ylabel('Error Rate')
plt.show()
```



2.3 Problem 4

The test error rate of both LDA and QDA will decrease with the increase of sample size. This is mainly because the variability and randomness will gradually decrease when we have larger sample size, and the pattern of our data will become more and more stable. Furthermore, according to the result presented in the table and the figure below, the test error of QDA decreases slightly more quickly than that of LDA. A possible reason is that the decision boundary of this scenario is non-linear. As a result, QDA would be more able to fit the data due to its higher flexibility. Therefore, the test rate of QDA will decrease in a faster pace than LDA.

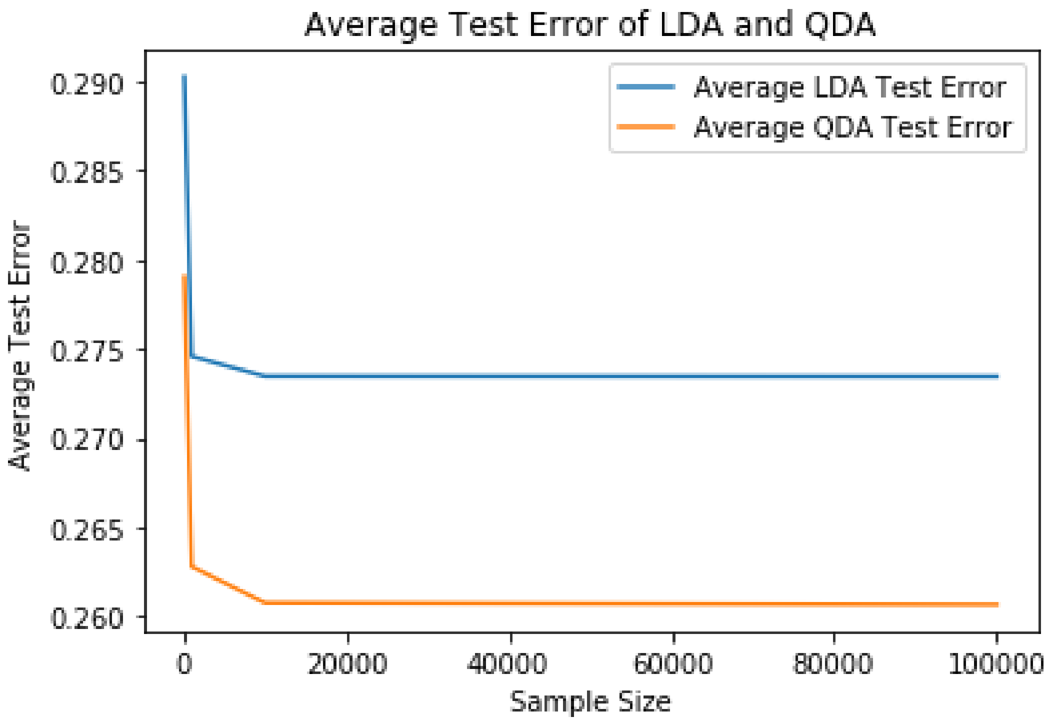
[12]:
```
np.random.seed(SEED)
sample_size = [1e02, 1e03, 1e04, 1e05]
sample_size_dict_ls = []
for n in sample_size:
    n = int(n)
    test_error_lda, test_error_qda = [], []
    sample_size_dict = {}
    for i in range(1000):
        x1 = np.random.uniform(-1, 1, n)
        x2 = np.random.uniform(-1, 1, n)
        y = x1 + x1 ** 2 + x2 + x2 ** 2 + np.random.normal(0, 1, n)
        test_error_lda.append(calculate_lda_error(x1, x2, y)[1])
        test_error_qda.append(calculate_qda_error(x1, x2, y)[1])
    avg_test_lda, avg_test_qda = mean(test_error_lda), mean(test_error_qda)
    sample_size_dict['Average LDA Test Error'] = mean(test_error_lda)
```

```
    sample_size_dict['Average QDA Test Error'] = mean(test_error_qda)
    sample_size_dict['QDA Error / LDA Error'] = avg_test_qda / avg_test_lda
    sample_size_dict_ls.append(sample_size_dict)
```

[13]:
```
accurc_dict_p4 = {'Sample Size = 100': sample_size_dict_ls[0],
                  'Sample Size = 1000': sample_size_dict_ls[1],
                  'Sample Size = 10000': sample_size_dict_ls[2],
                  'Sample Size = 100000': sample_size_dict_ls[3]}
accurc_table = pd.DataFrame(accurc_dict_p4)
accurc_table
```

[13]:
```
                        Sample Size = 100  Sample Size = 1000  \
Average LDA Test Error           0.290233            0.274577
Average QDA Test Error           0.279033            0.262813
QDA Error / LDA Error            0.961410            0.957158

                        Sample Size = 10000  Sample Size = 100000
Average LDA Test Error             0.273469              0.273454
Average QDA Test Error             0.260787              0.260681
QDA Error / LDA Error              0.953625              0.953289
```

[14]:
```
l1 = plt.plot(sample_size,accurc_table.loc['Average LDA Test␣
 ↪Error'],label='Average LDA Test Error')
l2 = plt.plot(sample_size,accurc_table.loc['Average QDA Test␣
 ↪Error'],label='Average QDA Test Error')
plt.title('Average Test Error of LDA and QDA')
plt.xlabel('Sample Size')
plt.ylabel('Average Test Error')
plt.legend()
plt.show()
```



3 Modeling Voter Turnout

3.1 Problem 5

There are two metrics that could be used here to evaluate the performance of the models: test error and AUC. That is, a good model is supposed to have a low test error and a high AUC value. Based on this criteria and the result presented in the table below, both QDA and Naive Bayes have relatively good performance in general. More specifically, as for the evaluation in terms of test rate, QDC demonstrates the best performance among all the models. On the other hand, as for the evaluation in terms of AUC, the Naive Bayes performs the best.

[15]:
```
df = pd.read_csv('mental_health.csv')
df.dropna(inplace=True)
df.head()
err_dict, auc_dict = {}, {}
```

[16]:
```
train, test = train_test_split(df, test_size=0.3, train_size=0.7,␣
 ↪random_state=SEED)
X_train, y_train = train.drop(['vote96', 'black', 'married'], axis=1),␣
 ↪train['vote96']
X_test, y_test = test.drop(['vote96', 'black', 'married'], axis=1),␣
 ↪test['vote96']
```

[17]:
```python
clf_lda = LinearDiscriminantAnalysis()
clf_lda.fit(X_train, y_train)
test_err_lda = 1 - clf_lda.score(X_test, y_test)
err_dict['LDA'] = test_err_lda
```

[18]:
```python
clf_qda = QuadraticDiscriminantAnalysis()
clf_qda.fit(X_train, y_train)
test_err_qda = 1 - clf_qda.score(X_test, y_test)
err_dict['QDA'] = test_err_qda
```

[19]:
```python
clf_log = LogisticRegression(random_state=0)
clf_log.fit(X_train, y_train)
test_err_log = 1 - clf_log.score(X_test, y_test)
err_dict['Logistics'] = test_err_log
```

```
/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)
```

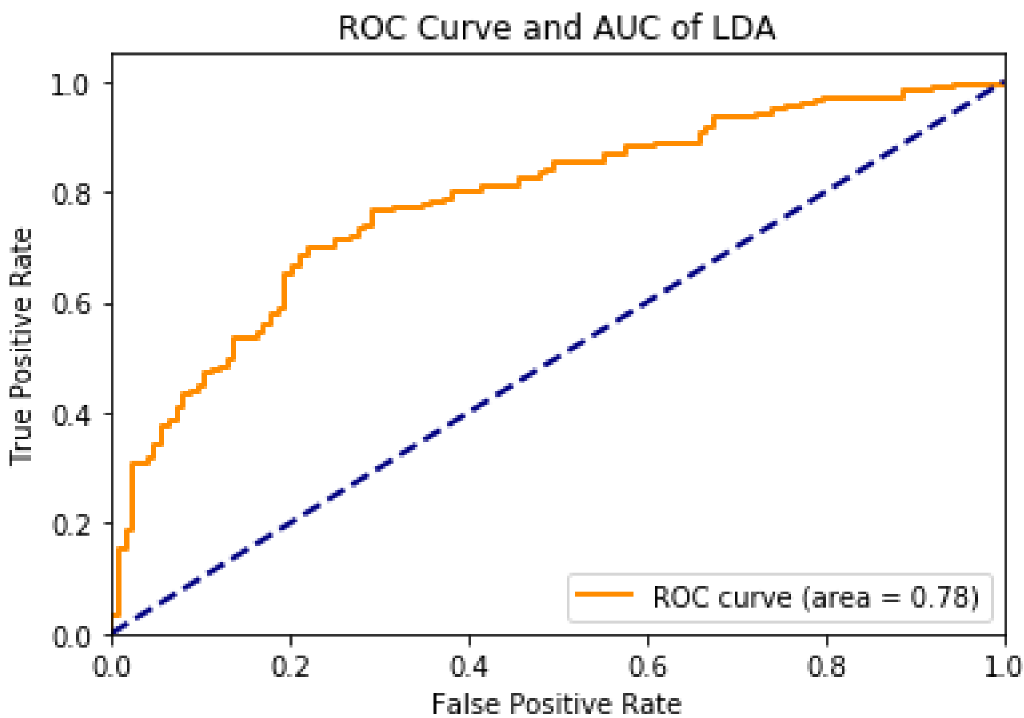
[20]:
```python
clf_nb = GaussianNB()
clf_nb.fit(X_train, y_train)
test_err_nb = 1 - clf_nb.score(X_test, y_test)
err_dict['Naive Bayes'] = test_err_nb
```

[21]:
```python
knn_models = []
best_knn_model = None
min_knn_test_err = 1
for k in range(1, 11):
    clf_knn = KNeighborsClassifier(n_neighbors=k, p=2)
    clf_knn.fit(X_train, y_train)
    test_err_knn = 1 - clf_knn.score(X_test, y_test)
    knn_models.append(clf_knn)
    err_dict['KNN (k = {})'.format(k)] = test_err_knn
```

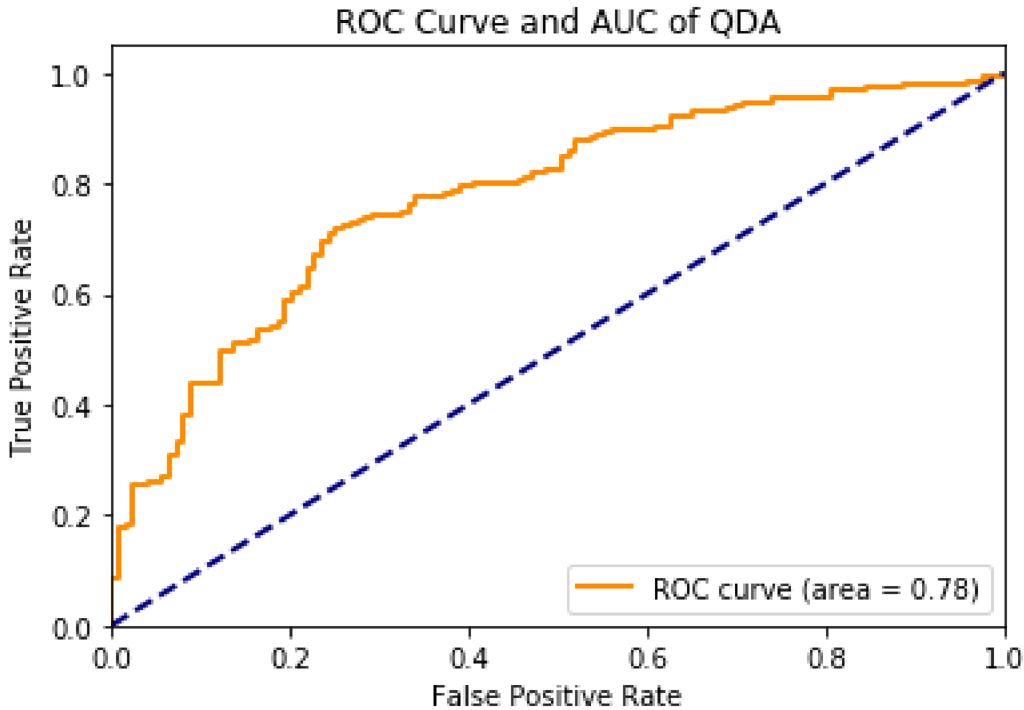
[22]:
```python
def compute_roc(model, model_name, auc_dict):
    '''
    Compute bothe roc curve and auc of different models.
    '''

    y_score = model.predict_proba(X_test)
    fpr, tpr, thresholds = roc_curve(y_test, y_score[:, 1])
    auc_lda = auc(fpr, tpr)
    auc_dict[model_name] = auc_lda

    plt.figure()
    lw = 2
```

```
    plt.plot(fpr, tpr, color='darkorange',
             lw=lw, label='ROC curve (area = %0.2f)' % auc_lda)
    plt.plot([0, 1], [0, 1], color='navy', lw=lw, 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('ROC Curve and AUC of {}'.format(model_name))
    plt.legend(loc="lower right")
    plt.show()
```

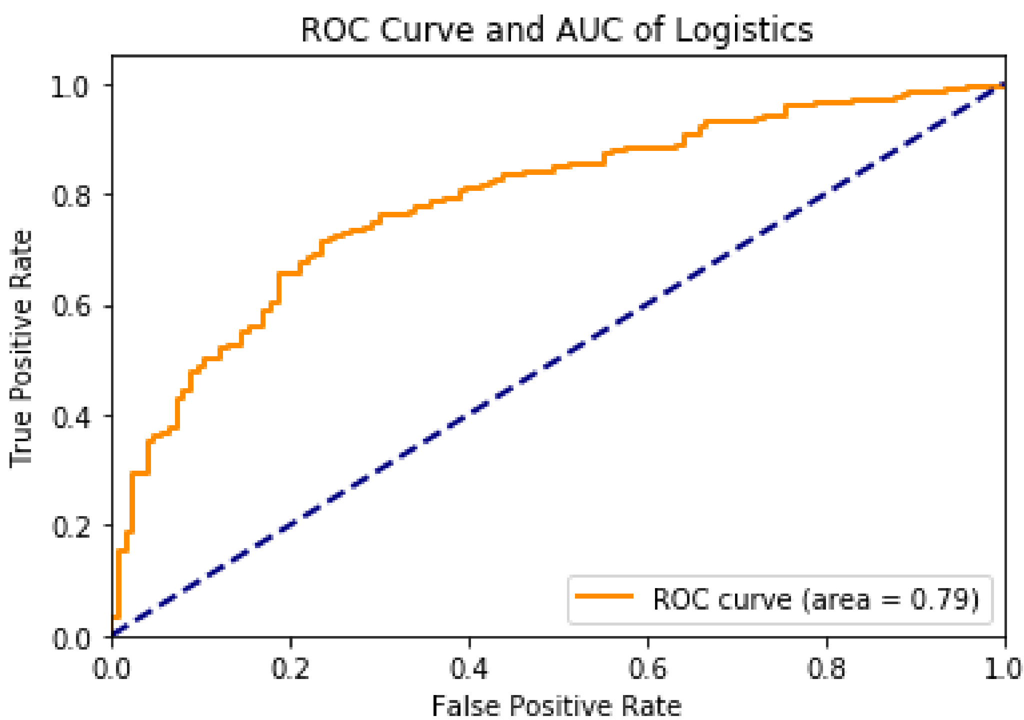
[23]:
```
compute_roc(clf_lda, 'LDA', auc_dict)
```



[24]:
```
compute_roc(clf_qda, 'QDA', auc_dict)
```

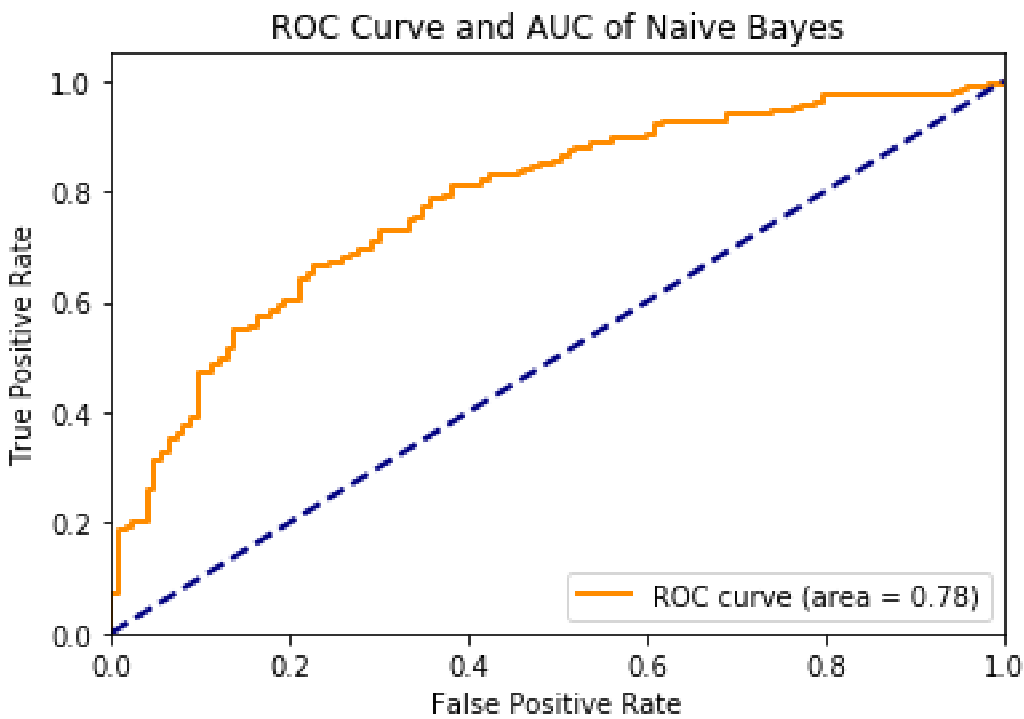


[25]:
```
compute_roc(clf_log, 'Logistics', auc_dict)
```



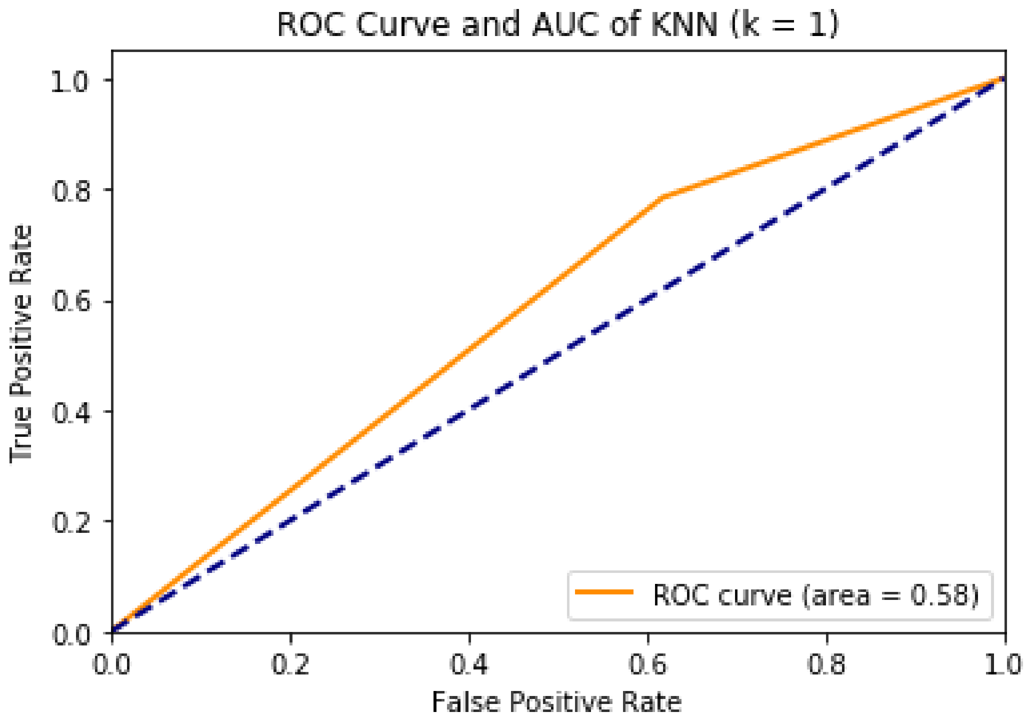
[26]:

```
compute_roc(clf_nb, 'Naive Bayes', auc_dict)
```

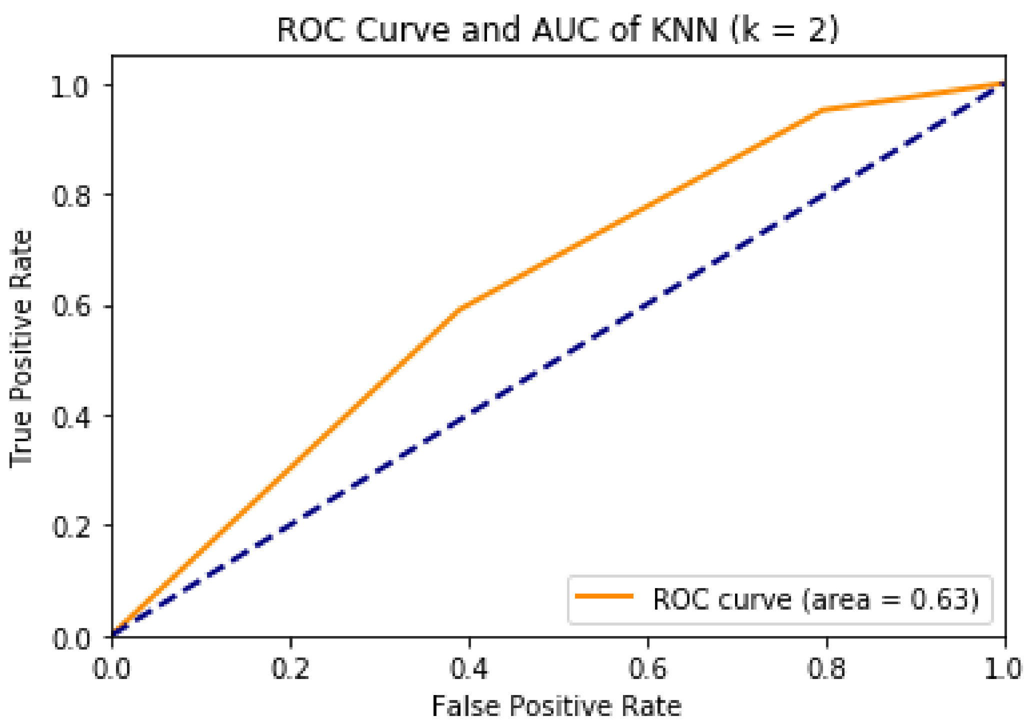


[27]:

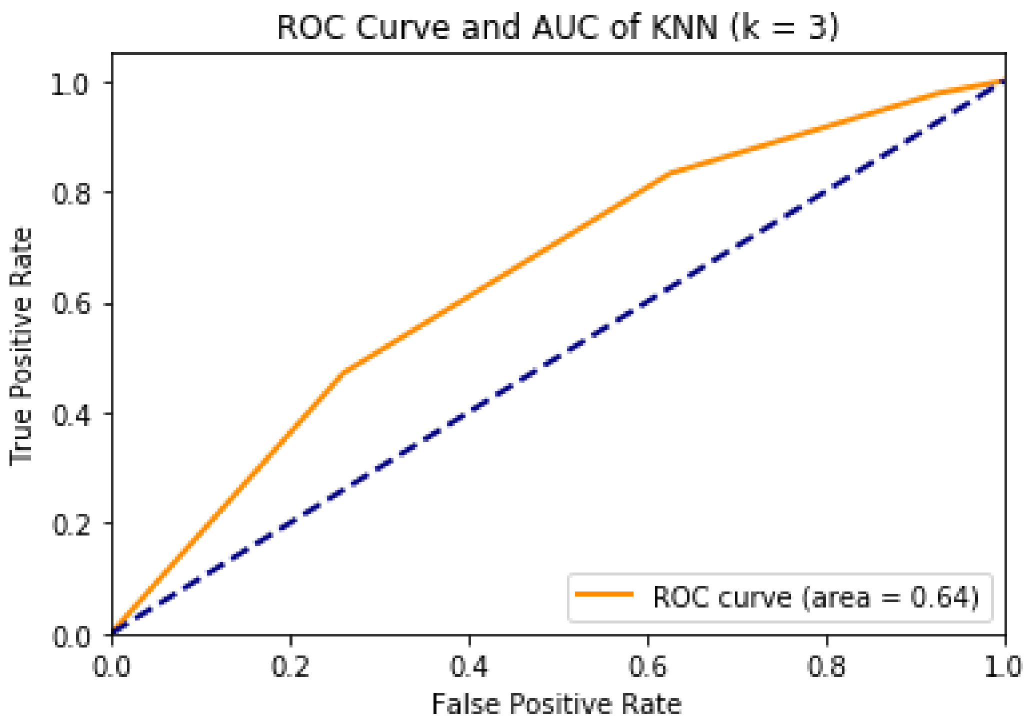
```
compute_roc(knn_models[0], 'KNN (k = 1)', auc_dict)
```



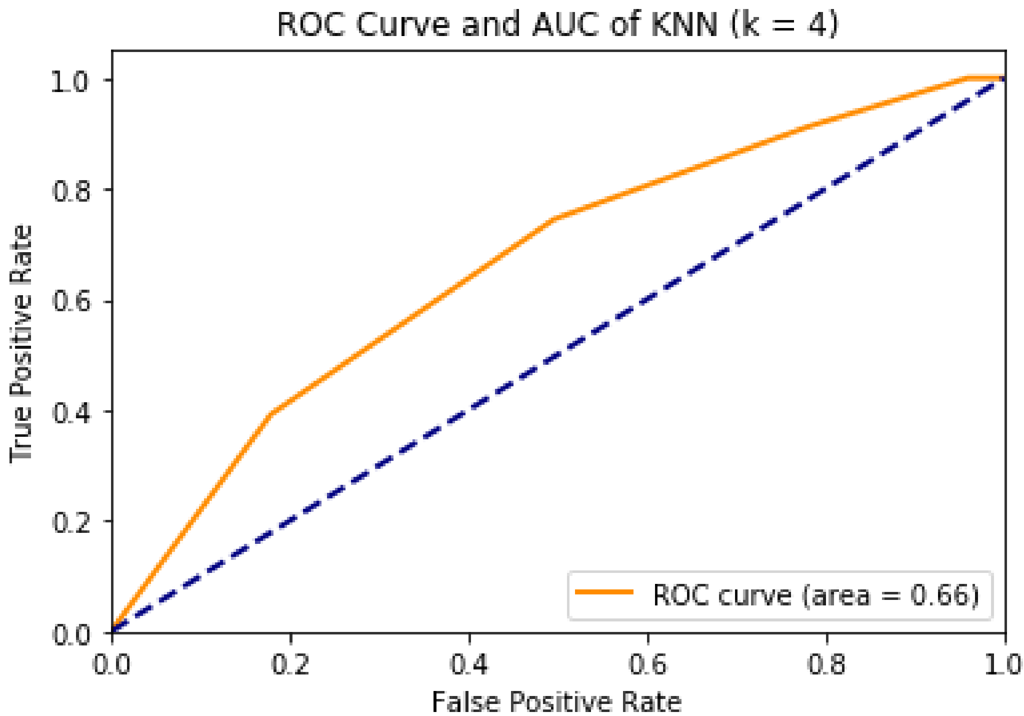
[28]:
```
compute_roc(knn_models[1], 'KNN (k = 2)', auc_dict)
```



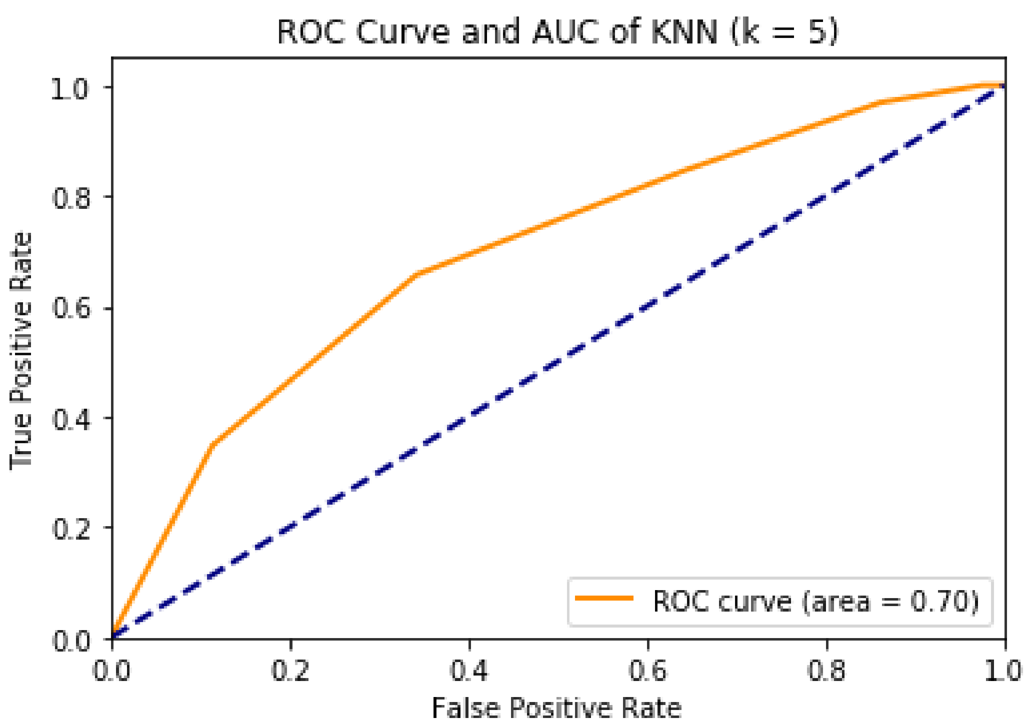
[29]:
```
compute_roc(knn_models[2], 'KNN (k = 3)', auc_dict)
```



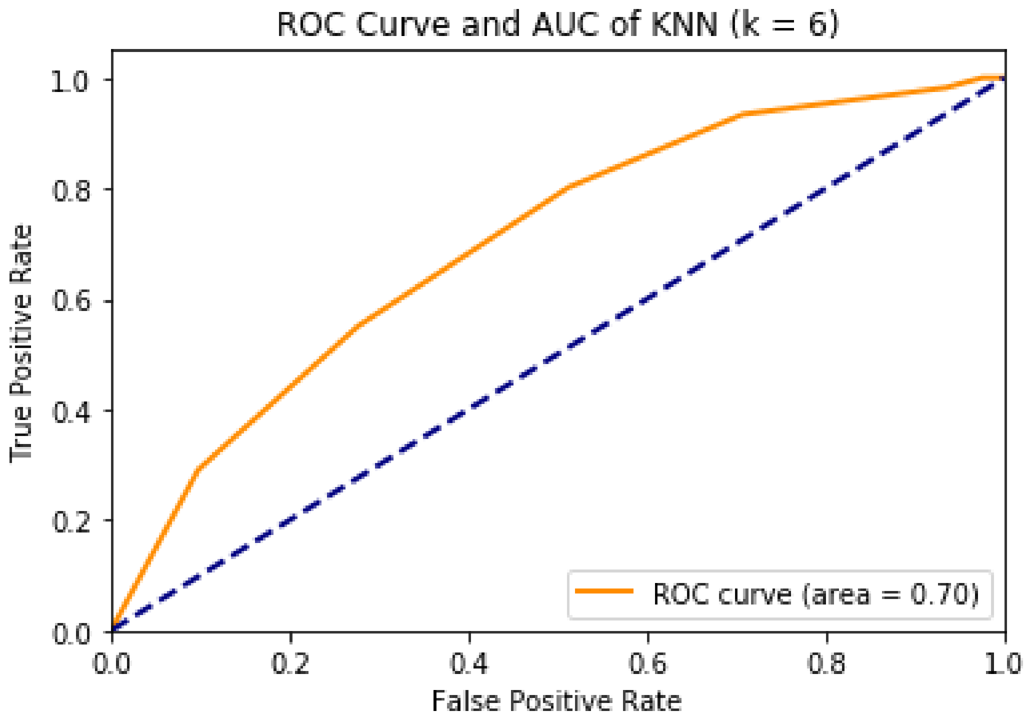
[30]:
```
compute_roc(knn_models[3], 'KNN (k = 4)', auc_dict)
```



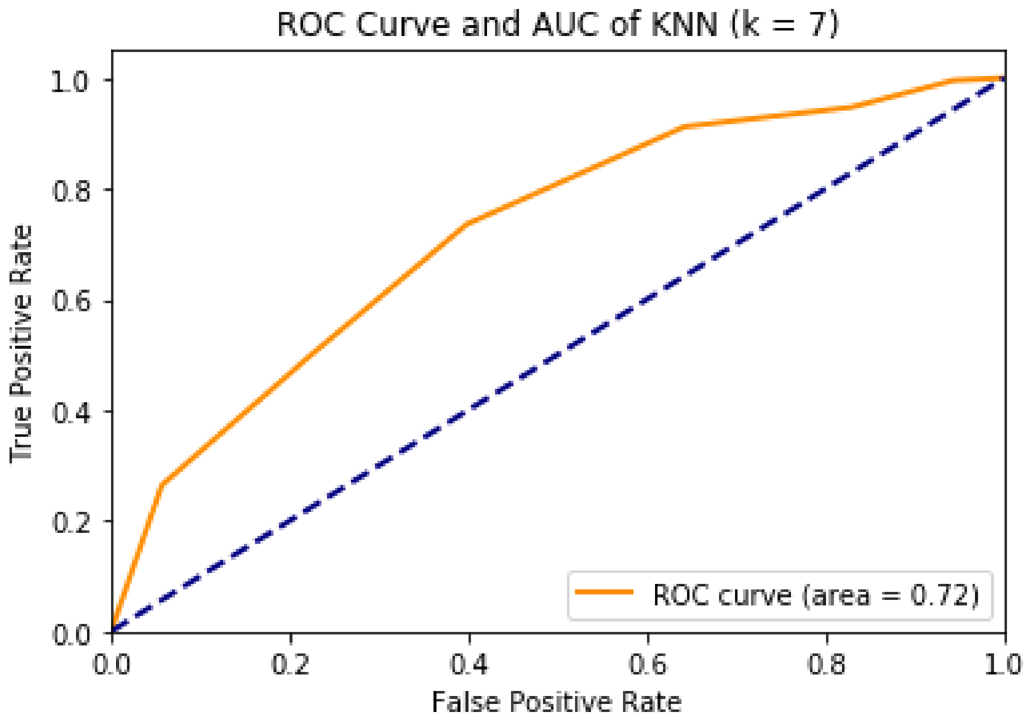
[31]:
```
compute_roc(knn_models[4], 'KNN (k = 5)', auc_dict)
```



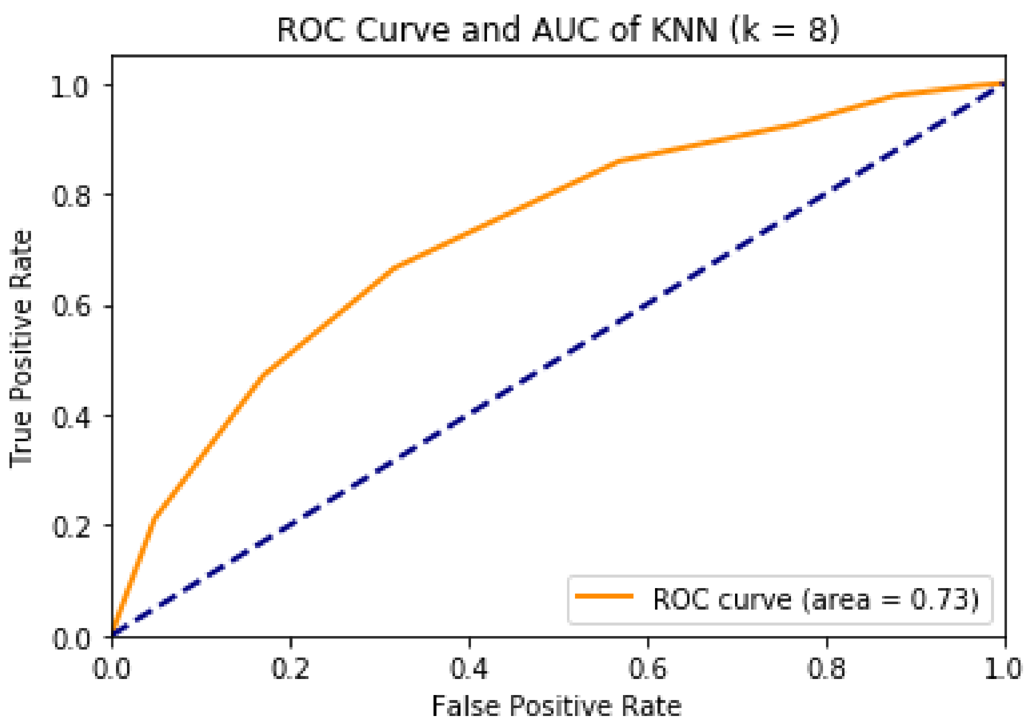
[32]:
```
compute_roc(knn_models[5], 'KNN (k = 6)', auc_dict)
```



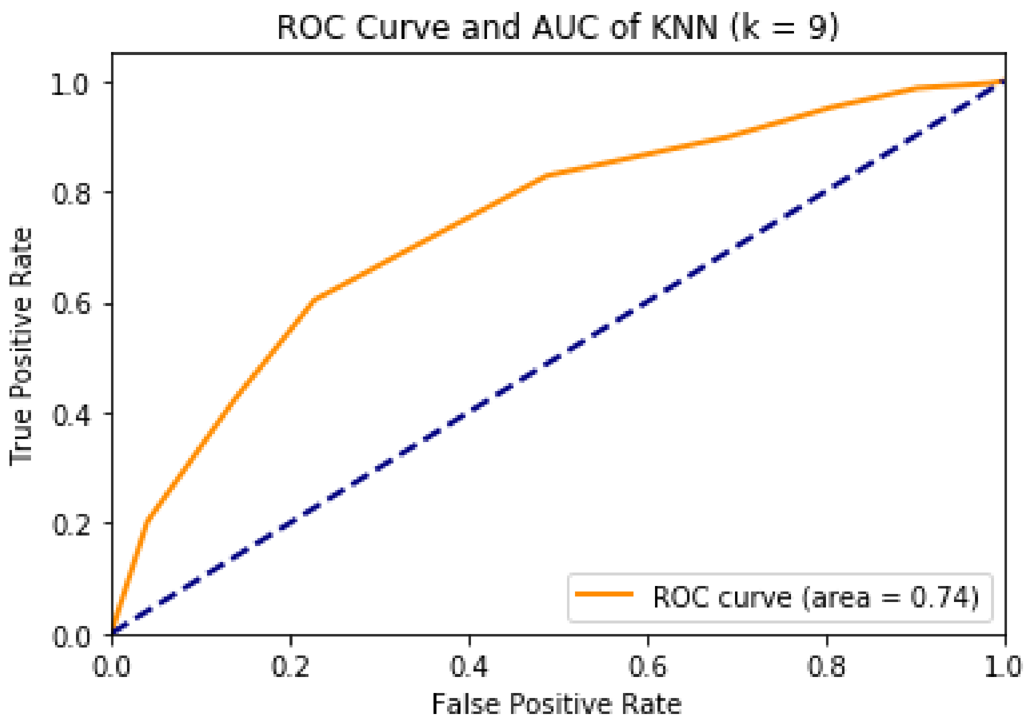
[33]:
```
compute_roc(knn_models[6], 'KNN (k = 7)', auc_dict)
```



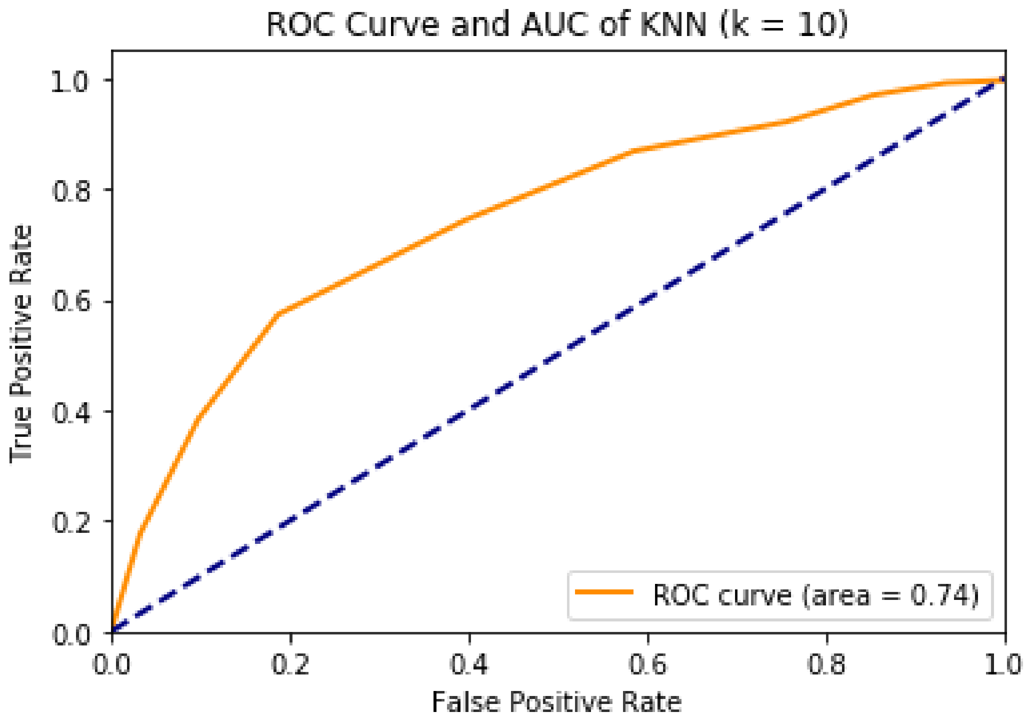
[34]:
```
compute_roc(knn_models[7], 'KNN (k = 8)', auc_dict)
```



[35]:
```
compute_roc(knn_models[8], 'KNN (k = 9)', auc_dict)
```



[36]:
```
compute_roc(knn_models[9], 'KNN (k = 10)', auc_dict)
```



[37]:
```
accurc_dict = {'Test Error': err_dict, 'AUC': auc_dict}
```

[38]:
```
pd.DataFrame(accurc_dict).sort_values(by=['Test Error'])
```

[38]:

	Test Error	AUC
QDA	0.265714	0.775151
Naive Bayes	0.268571	0.779091
KNN (k = 7)	0.282857	0.721124
LDA	0.291429	0.782243
KNN (k = 8)	0.291429	0.725099
KNN (k = 10)	0.291429	0.742935
Logistics	0.300000	0.786612
KNN (k = 6)	0.308571	0.700996
KNN (k = 9)	0.308571	0.738924
KNN (k = 5)	0.325714	0.695731
KNN (k = 3)	0.328571	0.644533
KNN (k = 4)	0.340000	0.662548
KNN (k = 1)	0.357143	0.583127
KNN (k = 2)	0.402857	0.626894