Answer

February 2, 2020

[1]:
```
from sklearn.model_selection import train_test_split
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis,␣
↪QuadraticDiscriminantAnalysis
from sklearn.metrics import roc_curve, auc
import pandas as pd
import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt
import random
from math import exp
```

[2]:
```
random.seed(123)
```

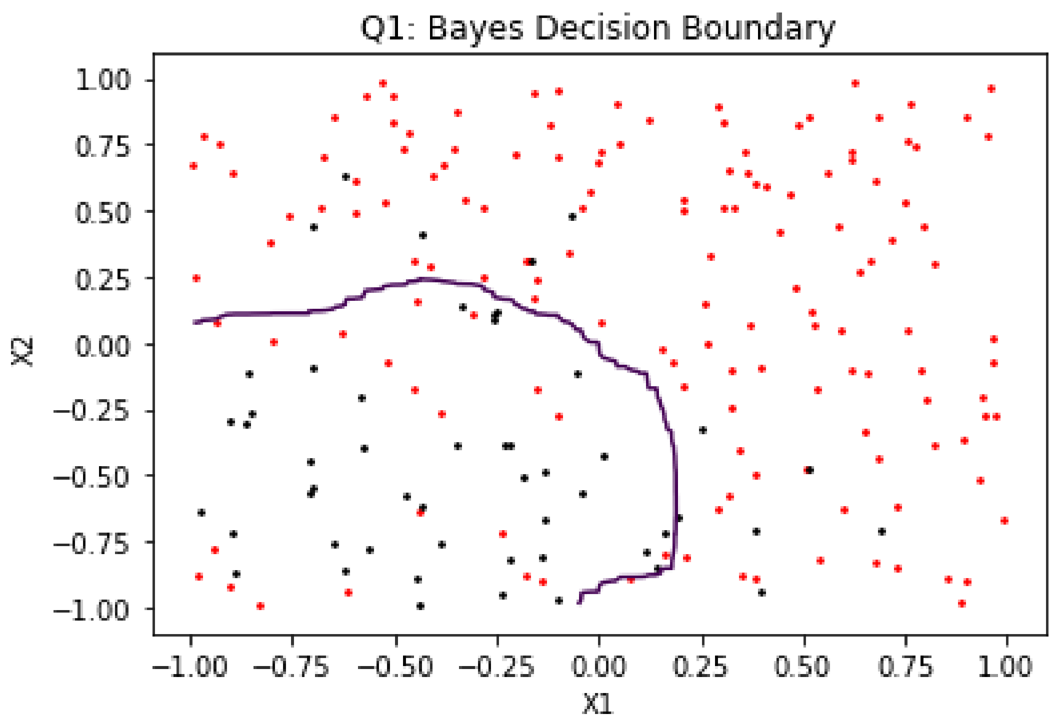
1 Decision Boundary

[3]:
```
X1 = np.random.uniform(-1,1,200)
X2 = np.random.uniform(-1,1,200)
e  = np.random.normal(0, 0.5, 200)
Y = X1 + X1**2 + X2 + X2**2 + e
prob = np.exp(Y)/ (1+np.exp(Y))
success = prob>0.5
```

[4]:
```
Z = []
for i in sorted(X1):
    each = []
    for j in sorted(X2):
        each.append(i + i**2 + j + j ** 2)
    Z.append(each)
```

[28]:
```
plt.scatter(X1[prob>0.5], X2[prob>0.5], color = 'red', s = 3)
plt.scatter(X1[prob<=0.5], X2[prob<=0.5], color = 'black', s=3)
plt.contour(sorted(X1),sorted(X2),Z,[0])
plt.title('Q1: Bayes Decision Boundary')
plt.xlabel('X1')
plt.ylabel('X2')
```

```
plt.show()
```



2 LDA vs QDA

[6]:
```
def simulation_1(n):
    X1, X2 = np.random.uniform(-1,1,n), np.random.uniform(-1,1,n)
    Response = (X1 + X2 + np.random.normal(0, 1, n)) >= 0
    data = pd.DataFrame({'x1': X1, 'x2':X2, 'R': Response})
    train, test = train_test_split(data, shuffle=False, test_size = 0.3)

    clf_l = LinearDiscriminantAnalysis()
    clf_l.fit(train[['x1', 'x2']].to_numpy(), np.array(train['R']))
    L_train = clf_l.score(train[['x1', 'x2']].to_numpy(), np.array(train['R']))
    L_test = clf_l.score(test[['x1', 'x2']].to_numpy(), np.array(test['R']))

    clf_q = QuadraticDiscriminantAnalysis()
    clf_q.fit(train[['x1', 'x2']].to_numpy(), np.array(train['R']))
    Q_train = clf_q.score(train[['x1', 'x2']].to_numpy(), np.array(train['R']))
    Q_test = clf_q.score(test[['x1', 'x2']].to_numpy(), np.array(test['R']))

    return L_train, L_test, Q_train, Q_test
```

[7]:
```
def simulation_n(n):
    X1, X2 = np.random.uniform(-1,1,n), np.random.uniform(-1,1,n)

    Response = (X1 + X1**2 + X2 + X2 **2 + np.random.normal(0, 1, n)) >= 0
    data = pd.DataFrame({'x1': X1, 'x2':X2, 'R': Response})
    train, test = train_test_split(data, shuffle=False, test_size = 0.3)

    clf_l = LinearDiscriminantAnalysis()
    clf_l.fit(train[['x1', 'x2']].to_numpy(), np.array(train['R']))
    L_train = clf_l.score(train[['x1', 'x2']].to_numpy(), np.array(train['R']))
    L_test = clf_l.score(test[['x1', 'x2']].to_numpy(), np.array(test['R']))

    clf_q = QuadraticDiscriminantAnalysis()
    clf_q.fit(train[['x1', 'x2']].to_numpy(), np.array(train['R']))
    Q_train = clf_q.score(train[['x1', 'x2']].to_numpy(), np.array(train['R']))
    Q_test = clf_q.score(test[['x1', 'x2']].to_numpy(), np.array(test['R']))

    return L_train, L_test, Q_train, Q_test
```

2.0.1 2. Linear

[8]:
```
LDA_train = []
LDA_test = []
QDA_train = []
QDA_test = []

for i in range(1000):
    ltr, lts, qtr, qts = simulation_1(1000)
    LDA_train.append(ltr)
    LDA_test.append(lts)
    QDA_train.append(qtr)
    QDA_test.append(qts)
```

[9]:
```
errors = pd.DataFrame({'LDA_train':LDA_train, 'LDA_test':LDA_test,
                       'QDA_train':QDA_train, 'QDA_test': QDA_test})
```

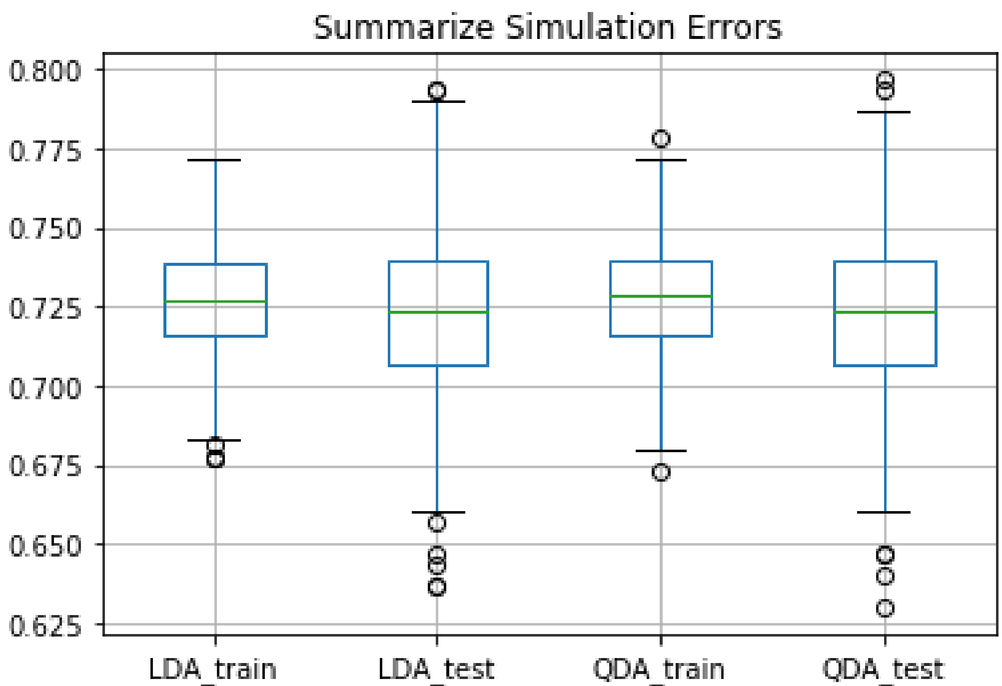
[10]:
```
errors.describe()
```

[10]:

	LDA_train	LDA_test	QDA_train	QDA_test
count	1000.000000	1000.000000	1000.000000	1000.000000
mean	0.727099	0.723980	0.727966	0.723483
std	0.016870	0.024971	0.016853	0.024626
min	0.677143	0.636667	0.672857	0.630000
25%	0.715714	0.706667	0.715714	0.706667
50%	0.727143	0.723333	0.728571	0.723333
75%	0.738571	0.740000	0.740000	0.740000

```
max       0.771429     0.793333     0.778571     0.796667
```

[11]:
```
errors.boxplot()
plt.title('Summarize Simulation Errors')
plt.show()
```



As we can observe, if the Bayes decision boundary is linear, it is hard to tell which one is better.

2.0.2 3. Non-linear

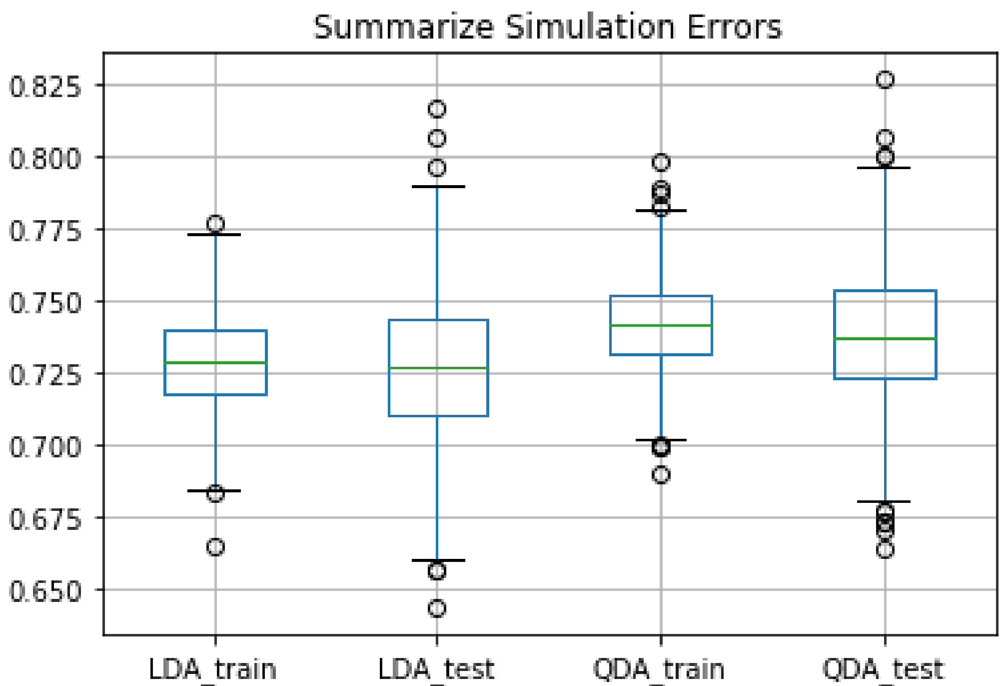
[12]:
```
LDA_train = []
LDA_test = []
QDA_train = []
QDA_test = []

for i in range(1000):
    ltr, lts, qtr, qts = simulation_n(1000)
    LDA_train.append(ltr)
    LDA_test.append(lts)
    QDA_train.append(qtr)
    QDA_test.append(qts)

errors = pd.DataFrame({'LDA_train':LDA_train, 'LDA_test':LDA_test,
                       'QDA_train':QDA_train, 'QDA_test': QDA_test})
```

[13]:
```
errors.describe()
```

[13]:

	LDA_train	LDA_test	QDA_train	QDA_test
count	1000.000000	1000.000000	1000.000000	1000.000000
mean	0.728113	0.725197	0.741630	0.738050
std	0.016152	0.024500	0.015247	0.023742
min	0.664286	0.643333	0.690000	0.663333
25%	0.717143	0.710000	0.731429	0.723333
50%	0.728571	0.726667	0.741429	0.736667
75%	0.740000	0.743333	0.751429	0.753333
max	0.777143	0.816667	0.798571	0.826667

[14]:
```
errors.boxplot()
plt.title('Summarize Simulation Errors')
plt.show()
```

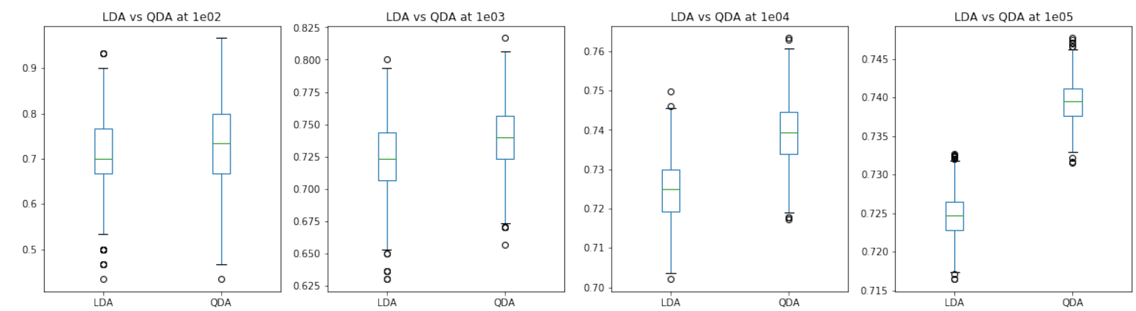


As we can observe, if the Bayes decision boundary is non-linear, QDA is slightly better than LDA on both sets.

2.0.3 4. Sample size

[15]:
```
np_lst = [1e02, 1e03, 1e04, 1e05]
lda_errors = dict()
qda_errors = dict()

for n in np_lst:
    lda_errors[n] = []
    qda_errors[n] = []
    for i in range(1000):
        ltr, lts, qtr, qts = simulation_1(int(n))
        lda_errors[n].append(lts)
        ltr, lts, qtr, qts = simulation_n(int(n))
        qda_errors[n].append(qts)
```

[16]:
```
plt.figure(figsize=(20,5))
name_lst = ["1e02", "1e03", "1e04", "1e05"]
for i, n in enumerate(np_lst):
    plt.subplot(1, 4, i+1)
    errors = pd.DataFrame({'LDA':lda_errors[n], 'QDA':qda_errors[n]})
    errors.boxplot(grid=False)
    plt.title(f'LDA vs QDA at {name_lst[i]}')
```



As sampel size n increases, the test error rate of QDA becomes better than LDA.

3 Modeling Voter Turnout

[17]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
```

[18]:
```
data = pd.read_csv('mental_health.csv')
data = data.dropna()
```

```
X_train, X_test, y_train, y_test = train_test_split(data.drop('vote96',axis=1),␣
↪data['vote96'], test_size=0.3)
X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

[18]:
```
((815, 7), (350, 7), (815,), (350,))
```

3.0.1 Train Models

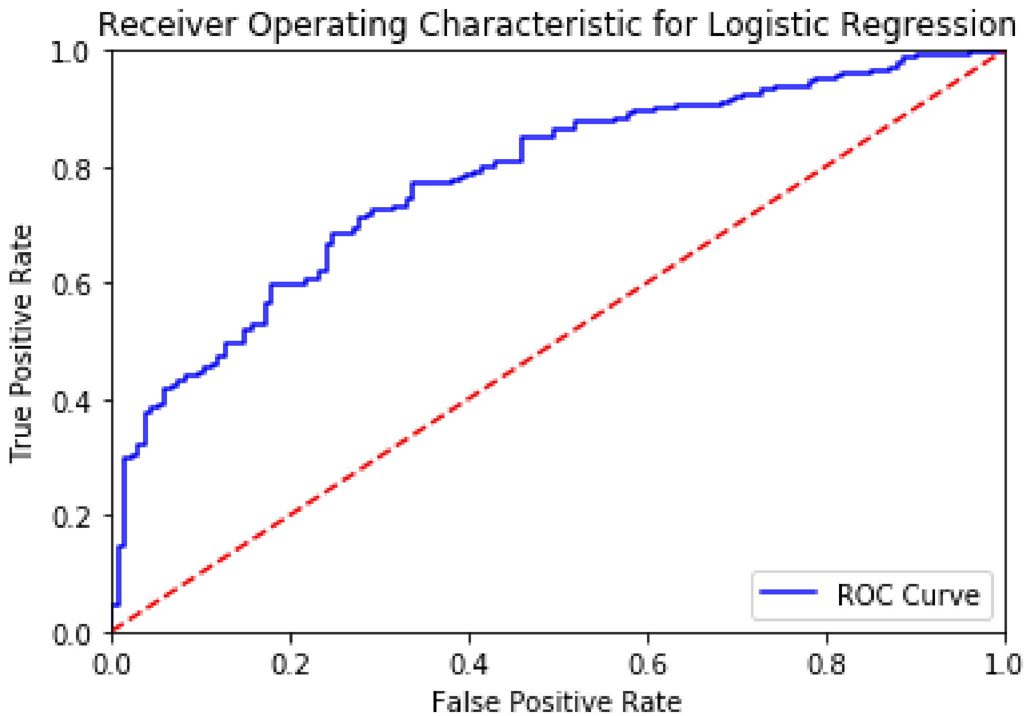
[19]:
```
## Logistic Regression
clf_log = LogisticRegression(solver='lbfgs').fit(X_train,y_train)
## LDA and QDA
clf_lda = LinearDiscriminantAnalysis().fit(X_train,y_train)
clf_qda = QuadraticDiscriminantAnalysis().fit(X_train,y_train)
## Naive Bayes
clf_nb = GaussianNB().fit(X_train,y_train)
## kNN
knn = [KNeighborsClassifier(n_neighbors=(i+1), metric='euclidean').
↪fit(X_train,y_train) for i in range(10)]
```

3.0.2 Results

[22]:
```
def model_performance(model, name):
    pred_label = model.predict(X_test)
    print("Error rate:", 1 - model.score(X_test,y_test))
    preds = model.predict_proba(X_test)[:,1]
    fpr, tpr, threshold = roc_curve(y_test, preds)
    roc_auc = auc(fpr, tpr)
    print('AUC = %0.2f' % roc_auc)

    plt.title(f'Receiver Operating Characteristic for {name}')
    plt.plot(fpr, tpr, 'blue', label = 'ROC Curve')
    plt.legend(loc = 'lower right')
    plt.plot([0, 1], [0, 1],'r--')
    plt.xlim([0, 1])
    plt.ylim([0, 1])
    plt.ylabel('True Positive Rate')
    plt.xlabel('False Positive Rate')
    plt.show()
```

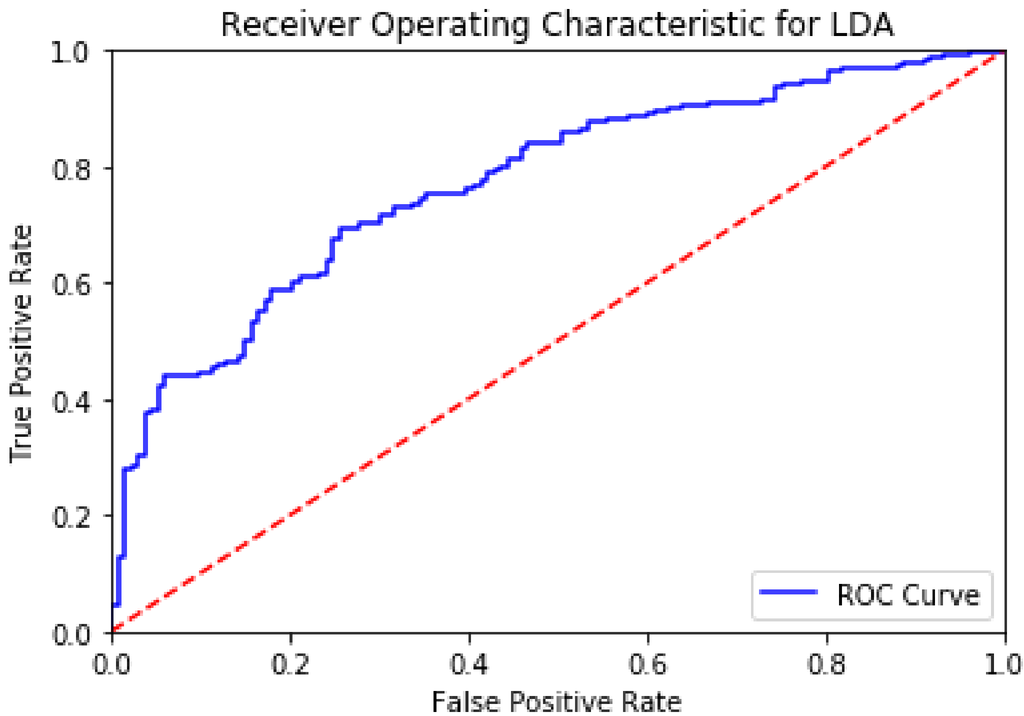
[23]:
```
model_performance(clf_log, 'Logistic Regression')
```

```
Error rate: 0.32571428571428573
AUC = 0.78
```



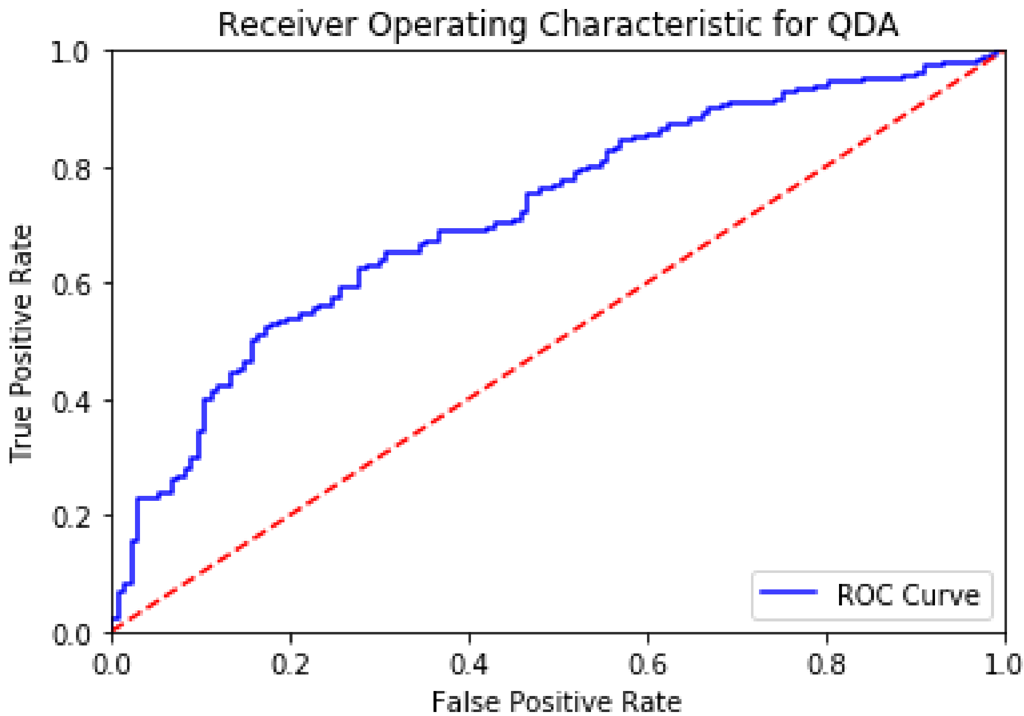
[24]:
```
model_performance(clf_lda, 'LDA')
```

```
Error rate: 0.33714285714285714
AUC = 0.77
```



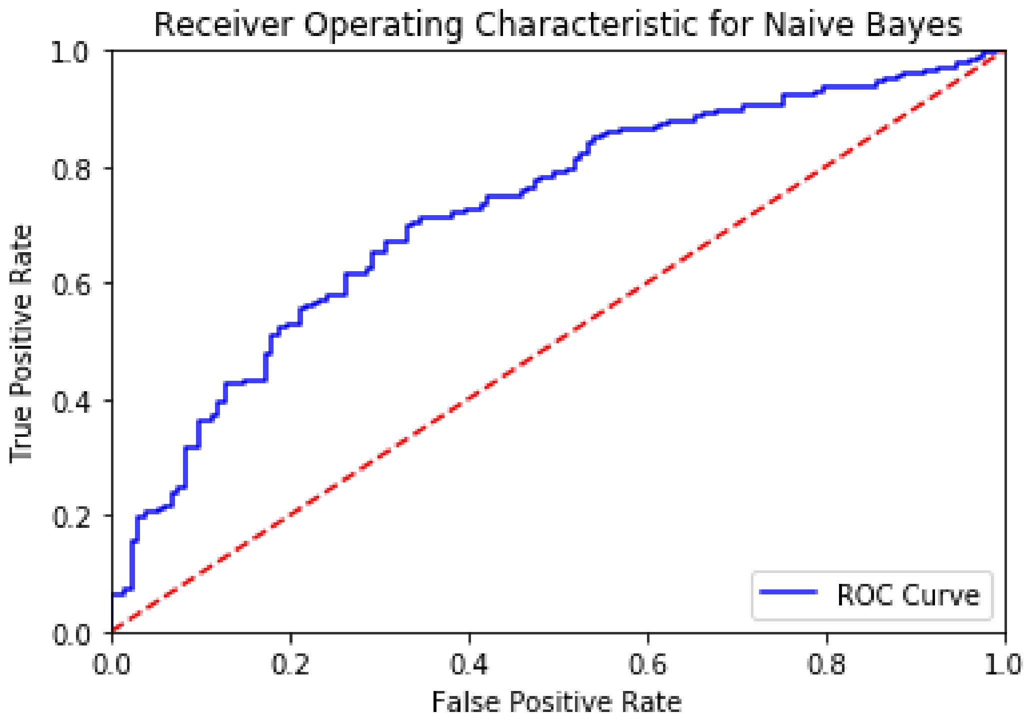
[25]:
```
model_performance(clf_qda, 'QDA')
```

```
Error rate: 0.31999999999999995
AUC = 0.72
```



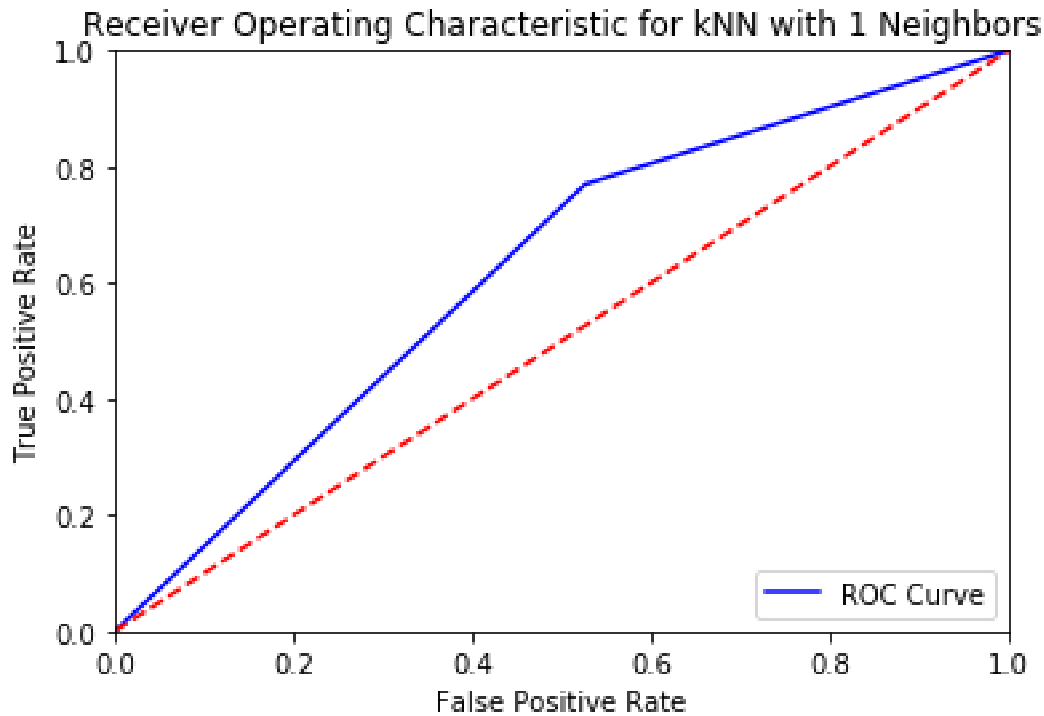
[26]:
```
model_performance(clf_nb, 'Naive Bayes')
```

```
Error rate: 0.30285714285714284
AUC = 0.72
```

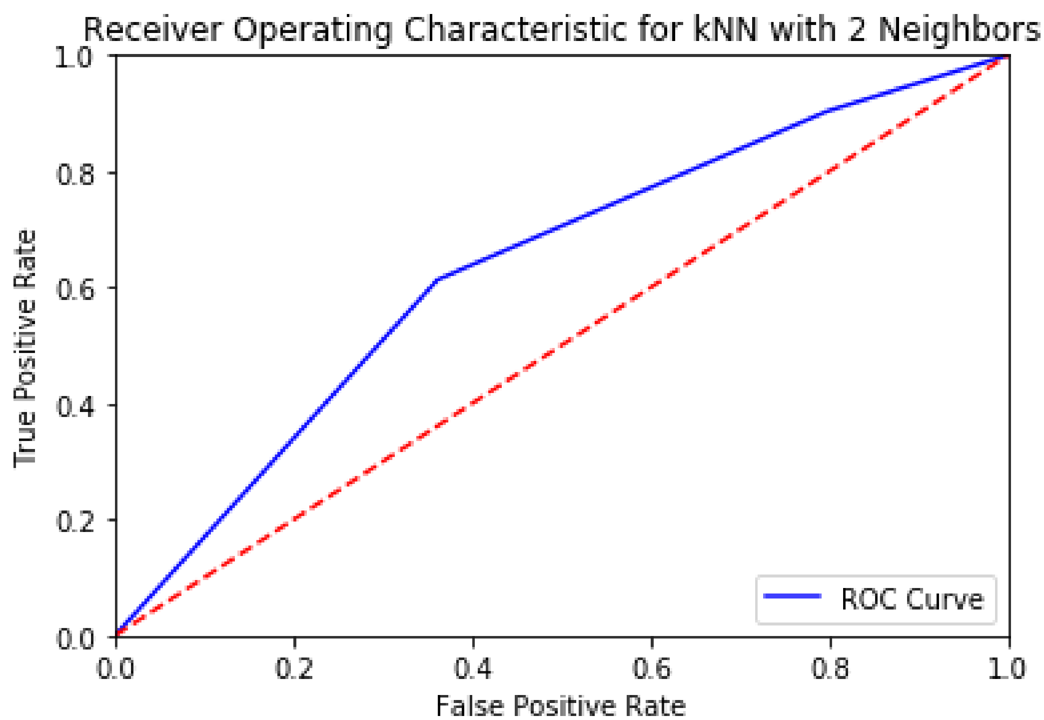


[27]:
```
for i in range(10):
    model_performance(knn[i], f'kNN with {i+1} Neighbors')
```

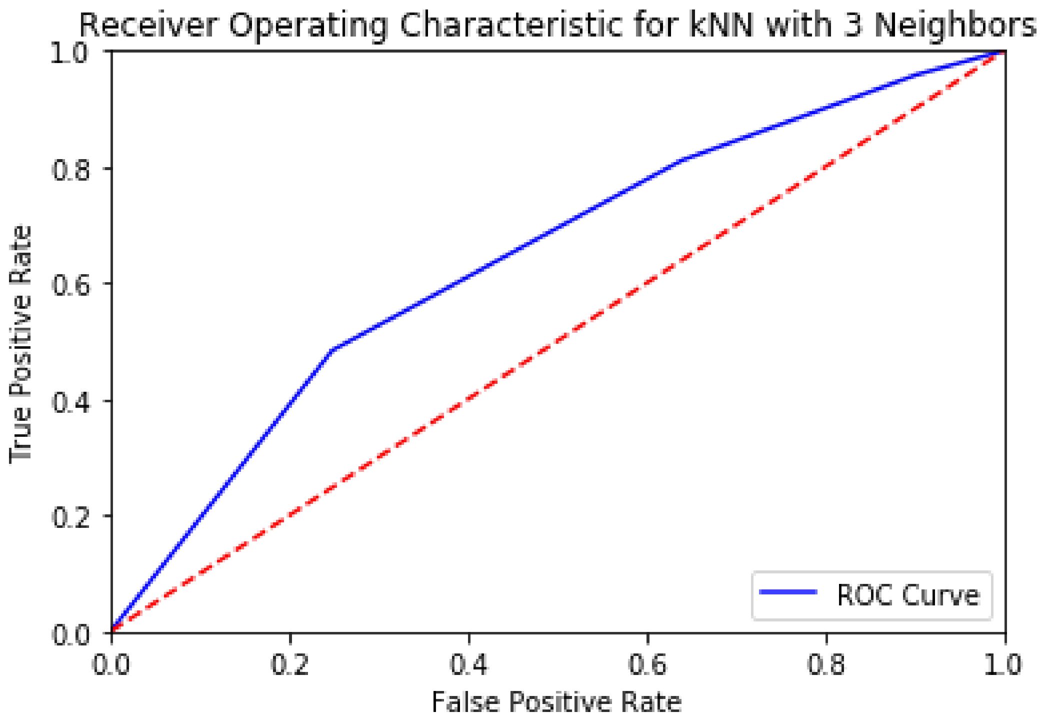
```
Error rate: 0.34285714285714286
AUC = 0.62
```



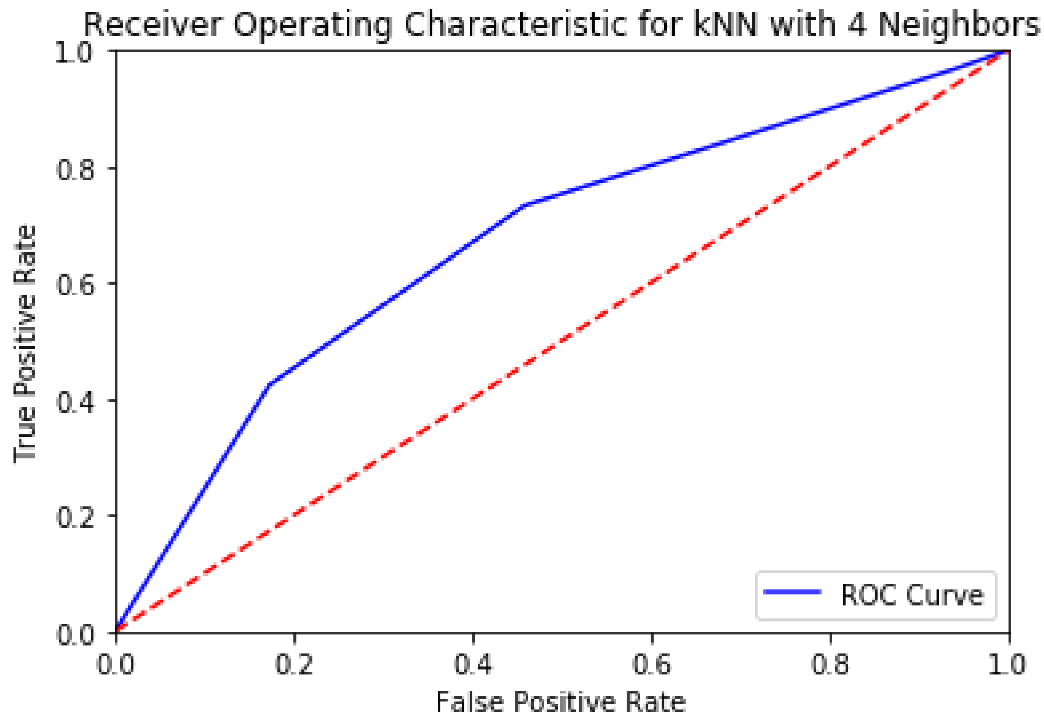

```
Error rate: 0.37714285714285711
AUC = 0.63
```



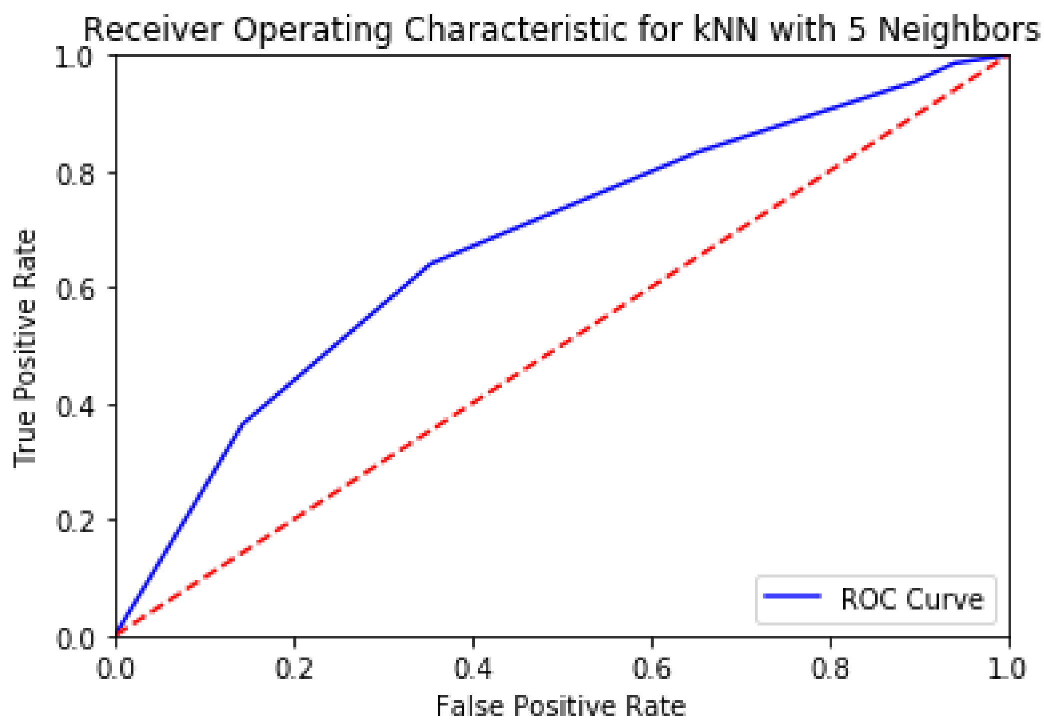
```
Error rate: 0.36
AUC = 0.64
```



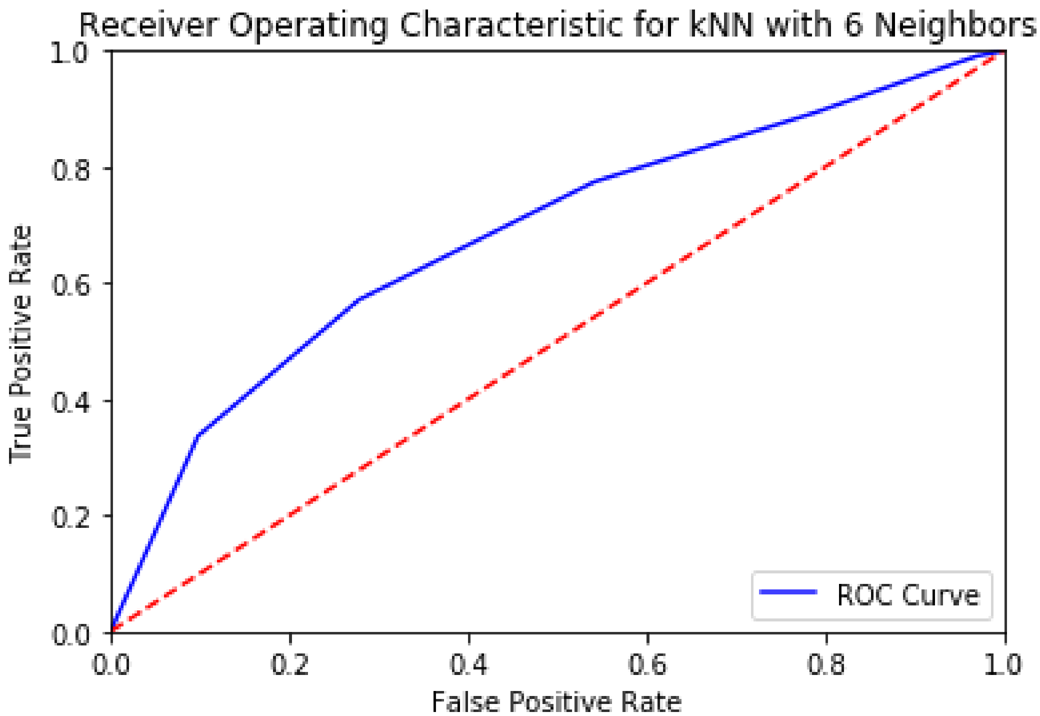

```
Error rate: 0.33999999999999997
AUC = 0.67
```



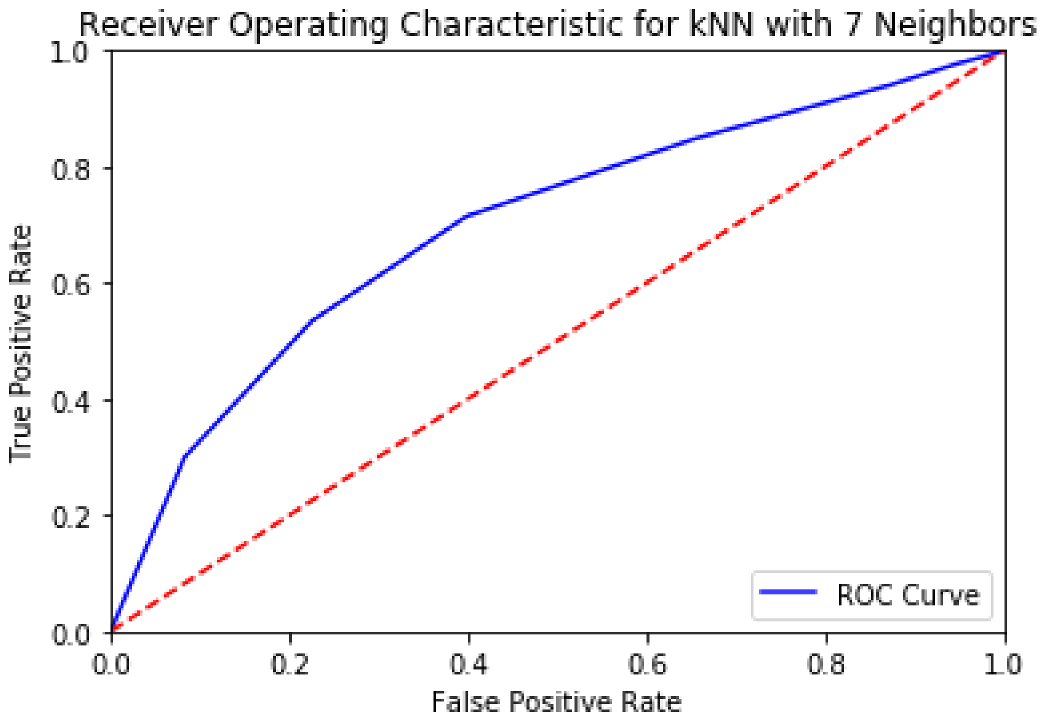

```
Error rate: 0.3514285714285714
AUC = 0.67
```



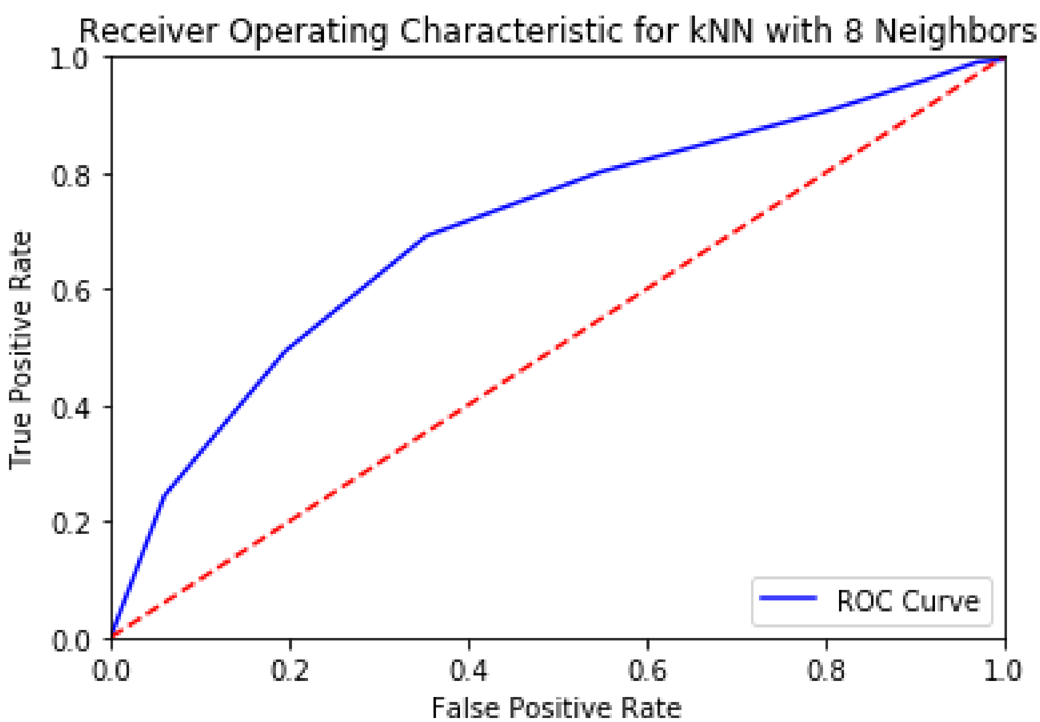
```
Error rate: 0.34571428571428575
AUC = 0.68
```



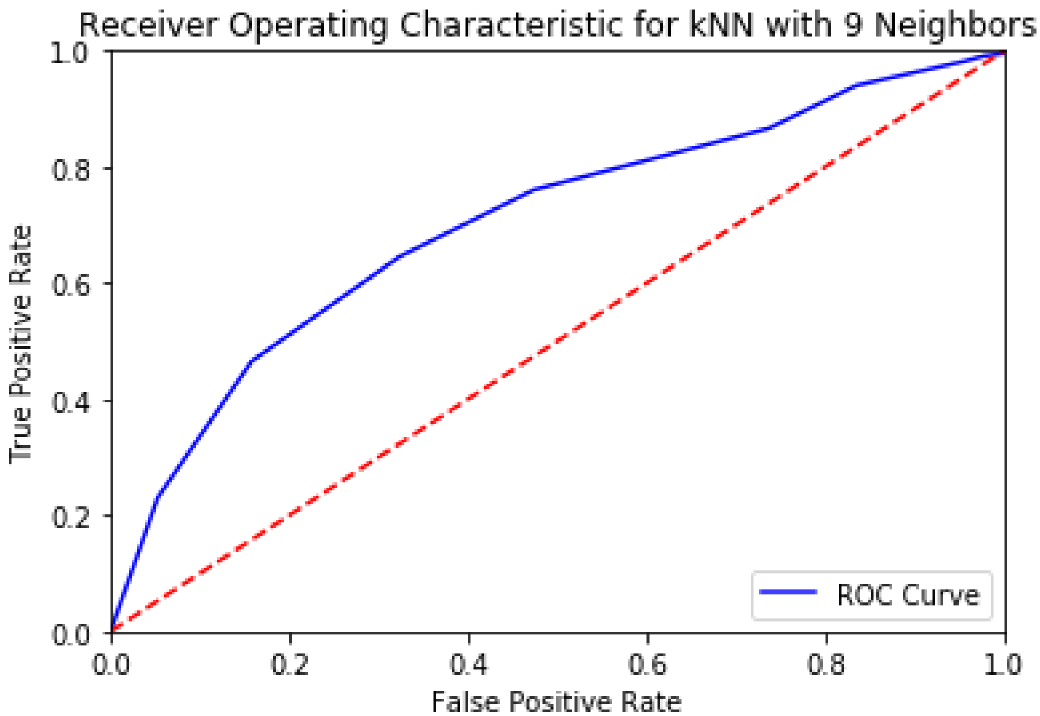

```
Error rate: 0.34285714285714286
AUC = 0.70
```



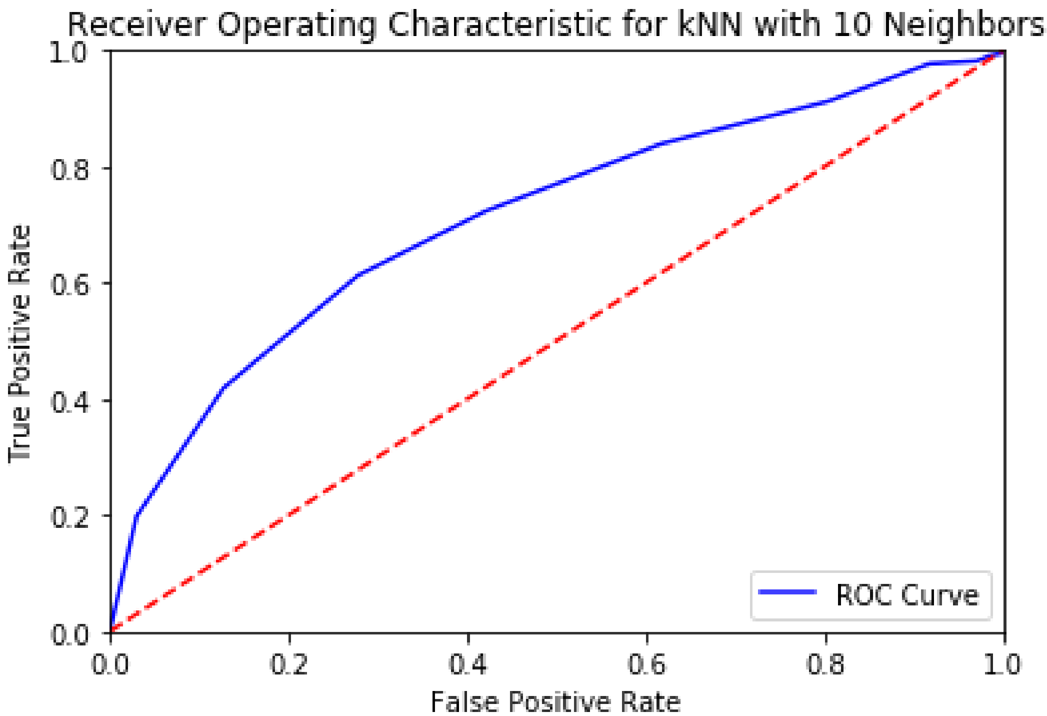

```
Error rate: 0.3314285714285714
AUC = 0.70
```



```
Error rate: 0.3628571428571429
AUC = 0.70
```




```
Error rate: 0.3342857142857143
AUC = 0.71
```



Judging by both error rate and AUC, logistic regression and LDA work the best.

```
[ ]:
```