Wang_Miaohan_HW2

February 2, 2020

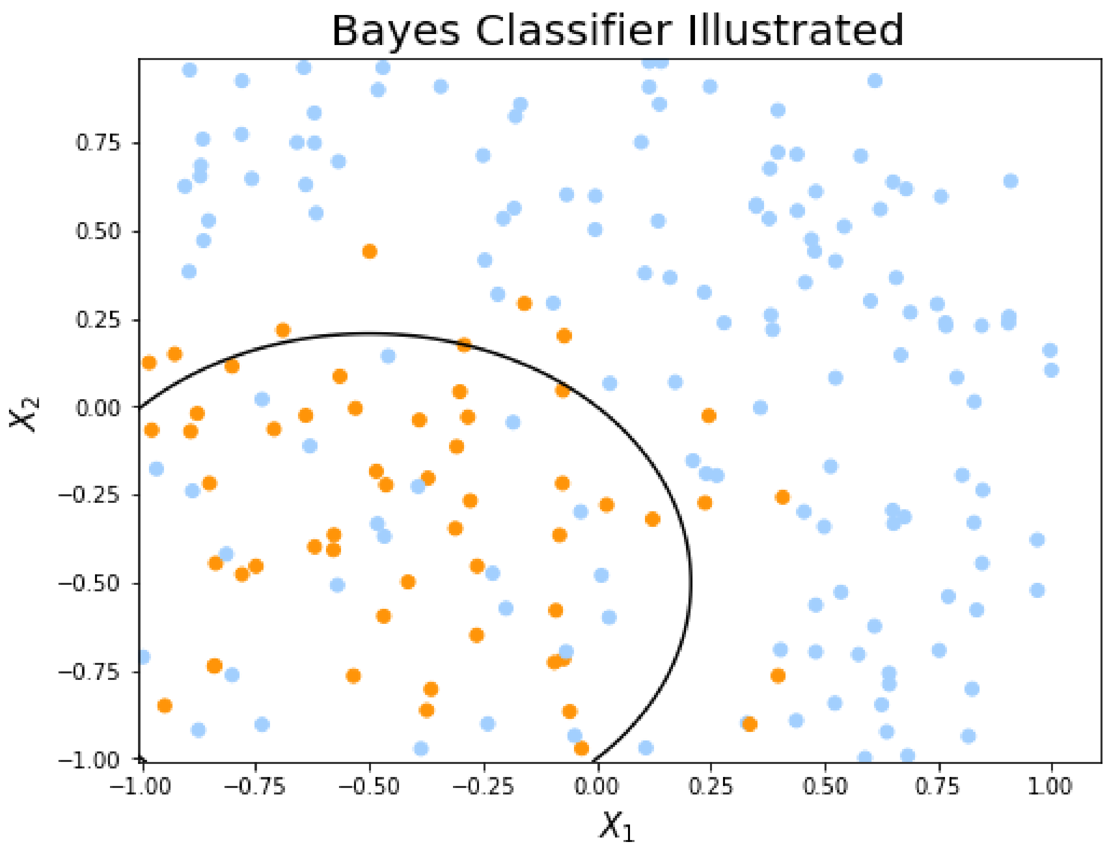
[1]:
```python
import random
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis
from sklearn.linear_model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

[2]:
```python
# auxiliary function for Q2, Q3, Q4
def simulation(size, rep, model, linearity, random_state=1020):
    random.seed(random_state)

    train_err = np.array([])
    test_err = np.array([])

    assert(model in ['lda', 'qda']), print("Please only put 'lda' or 'qda' in␣
↪the 'model' parameter.")

    for i in range(rep):
        X_1 = np.random.uniform(-1, 1, size)
        X_2 = np.random.uniform(-1, 1, size)
        X_data = np.stack((X_1, X_2), axis=-1)
        temp = X_1 + X_2 + np.random.normal(0, 1, size)
        if not linearity:
            temp += X_1**2 + X_2**2
        y_data = np.array(temp > 0)
        X_train, X_test, y_train, y_test = train_test_split(X_data, y_data,␣
↪test_size=0.3, random_state=random_state)

        mod = None
        if model == 'lda':
            mod = LinearDiscriminantAnalysis()
        elif model == 'qda':
            mod = QuadraticDiscriminantAnalysis()

        mod.fit(X_train, y_train)
        train_err = np.append(train_err, 1-mod.score(X_train, y_train))
        test_err = np.append(test_err, 1-mod.score(X_test, y_test))

    return train_err.mean(), test_err.mean()

# auxiliary function for Q5
def fit_model(split_dataset, model, k=0, random_state=1020):

    assert(model in ['log', 'lda', 'qda', 'nb', 'knn'])

    X_train, X_test, y_train, y_test = split_dataset

    title = 'ROC with '

    mod = None
    if model == 'log':
        mod = LogisticRegression(random_state=random_state)
        title += 'Logistic Regression'
    elif model == 'lda':
        mod = LinearDiscriminantAnalysis()
        title += 'LDA'
    elif model == 'qda':
        mod = QuadraticDiscriminantAnalysis()
        title += 'QDA'
    elif model == 'knn':
        mod = KNeighborsClassifier(n_neighbors = k)
        title += str(k) + ' Nearest Neighbor(s)'
    elif model == 'nb':
        mod = GaussianNB()
        title += 'Naive Bayes'

    mod.fit(X_train, y_train)

    fpr, tpr, threshold = metrics.roc_curve(y_test, mod.predict(X_test))
    roc_auc = metrics.auc(fpr, tpr)

    plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' % roc_auc)
    plt.title(title, fontsize=15)
    plt.legend(loc = 'lower right')
    plt.plot([0, 1], [0, 1],'r--')
    plt.xlim([0, 1])
    plt.ylim([0, 1])
    plt.ylabel('Sensitivity')
    plt.xlabel('1 - Specificity')
    plt.show()

    return 1-mod.score(X_train, y_train), 1-mod.score(X_test, y_test), roc_auc
```

1 Question 1

[3]:
```
# setting random seed
random.seed(1020)

# simulate dataset
X1 = np.random.uniform(-1, 1, 200)
X2 = np.random.uniform(-1, 1, 200)

# calculating Y and probabilty derived from Y
Y = X1 + X1**2 + X2 + X2**2 + np.random.normal(0, 0.5, 200)
prob = np.exp(Y) / (1 + np.exp(Y))
```

[4]:
```
plt.figure(figsize=(8,6))

# scatter plot of X1 against X2
categories = np.array([1 if p > 0.5 else 0 for p in prob])
plt.scatter(X1, X2, c=np.where(prob >  0.5, '#a2cffe', '#ff9408'))
plt.xlabel('$X_1$', fontsize=15)
plt.ylabel('$X_2$', fontsize=15)
plt.title('Bayes Classifier Illustrated', fontsize=20)

# drawing bayesian decision boundary
a = np.arange(-1.01, 1, 0.01)
b = np.arange(-1.01, 1, 0.01)
A, B = np.meshgrid(a, b)
C = A + A**2 + B + B**2
prob_C = np.exp(C) / (1 + np.exp(C))
plt.contour(A, B, prob_C, levels=[0.5], colors='black');
```



2 Question 2

We expect the LDA to work better than QDA in both test error since the data we generated is strictly linear, but the train error of QDA might be better than LDA since it might take noises as quadratic relationships, resulting in overfitting the data. First we simulate 1000 times the LDA and QDA models of $f(X) = X_1 + X_2 + \epsilon$, we obtain the average train and test error rates in the following table:

```
[5]: LDA_err_Q2 = simulation(size=1000, rep=1000, model='lda', linearity=True)
     QDA_err_Q2 = simulation(size=1000, rep=1000, model='qda', linearity=True)
     df2 = pd.DataFrame({'LDA': LDA_err_Q2, 'QDA': QDA_err_Q2}, index=['Train␣
      ↪Error', 'Test Error'])

     # Error Comparison of LDA and QDA (Linear Case)
     df2
```

```
[5]:                   LDA       QDA
     Train Error  0.274826  0.272263
```

```
Test Error   0.276997  0.276873
```

We observe that with $f(X) = X_1 + X_2 + \epsilon$, train error rate of LDA is higher than that of QDA and test error rate of LDA is slightly lower than that of QDA. This conforms with our expectation. Since the data given by $f(X)$ here is linear, which fulfills assumptions of LDA that all classes shares the same covariance matrix. LDA would perform better than QDA when the relationshop in the data is indeed linear.

3 Question 3

We would expect QDA to perform better than LDA in both train and test error rates given the relationship in the data is quadratic. Here, we simulate 1000 times the LDA and QDA models of $f(X) = X_1 + X_1^2 + X_2 + X_2^2 + \epsilon$, we obtain the average train and test error rates in the following table:

[6]:
```
LDA_err_Q3 = simulation(size=1000, rep=1000, model='lda', linearity=False)
QDA_err_Q3 = simulation(size=1000, rep=1000, model='qda', linearity=False)
df3 = pd.DataFrame({'LDA': LDA_err_Q3, 'QDA': QDA_err_Q3}, index=['Train␣
↪Error', 'Test Error'])

# Error Comparison of LDA and QDA (Quadratic Case)
df3
```

[6]:
```
                  LDA       QDA
Train Error  0.273210  0.258609
Test Error   0.274353  0.261380
```

When we switch to $f(X) = X_1 + X_1^2 + X_2 + X_2^2 + \epsilon$, the performances of QDA is better than that of LDA, same as what we expected. QDA has both a lower train error rate and a lower test error rate. This is because the data here has a quadratic relationship, so that QDA captures the perculiarities of the data much better than LDA. Hence QDA performs better in both training and test scenarios. The LDA model suffers from unequal variance across $f(X)$, yielding a less satisfactory performance than in the linear case.

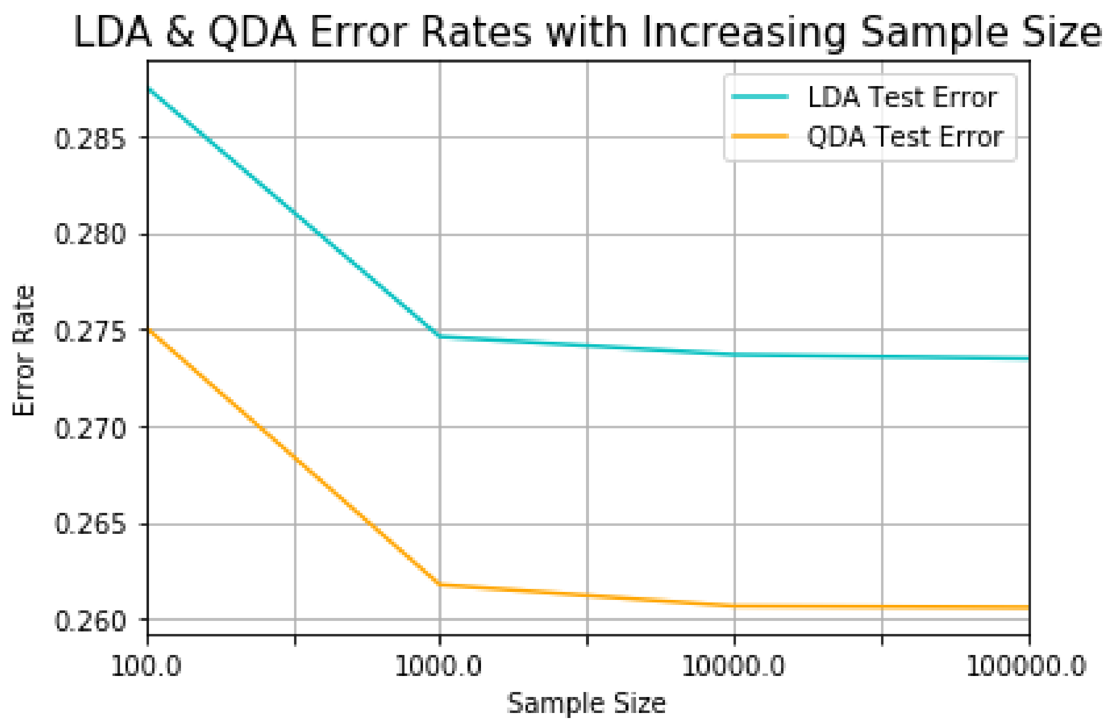
4 Question 4

[7]:
```
lda_results = {}
qda_results = {}
trials = [1e02, 1e03, 1e04, 1e05] # sample sizes for simulating non-linear data

# simulation
for size in trials:
    lda_errors =  simulation(size=int(size), rep=1000, model='lda',␣
↪linearity=False)
    lda_results[str(size)] = lda_errors[1]
    qda_errors =  simulation(size=int(size), rep=1000, model='qda',␣
↪linearity=False)
    qda_results[str(size)] = qda_errors[1]
```

[8]:
```
# creating table for visualization
df_lda = pd.DataFrame(lda_results, index=['LDA Test Error']).T
df_qda = pd.DataFrame(qda_results, index=['QDA Test Error']).T
df_Q4 = df_lda.merge(df_qda, left_index=True, right_index=True)

# Error Comparison of LDA and QDA with Increasing Sample Sizes (Non-linear Case)
df_Q4
```

[8]:
```
          LDA Test Error  QDA Test Error
100.0           0.287533        0.275067
1000.0          0.274597        0.261757
10000.0         0.273664        0.260651
100000.0        0.273460        0.260584
```

[9]:
```
# in graphical form
df_Q4.plot(color=['c','orange'], grid=True)
plt.title('LDA & QDA Error Rates with Increasing Sample Size', fontsize=15)
plt.xlabel('Sample Size')
plt.ylabel('Error Rate');
```



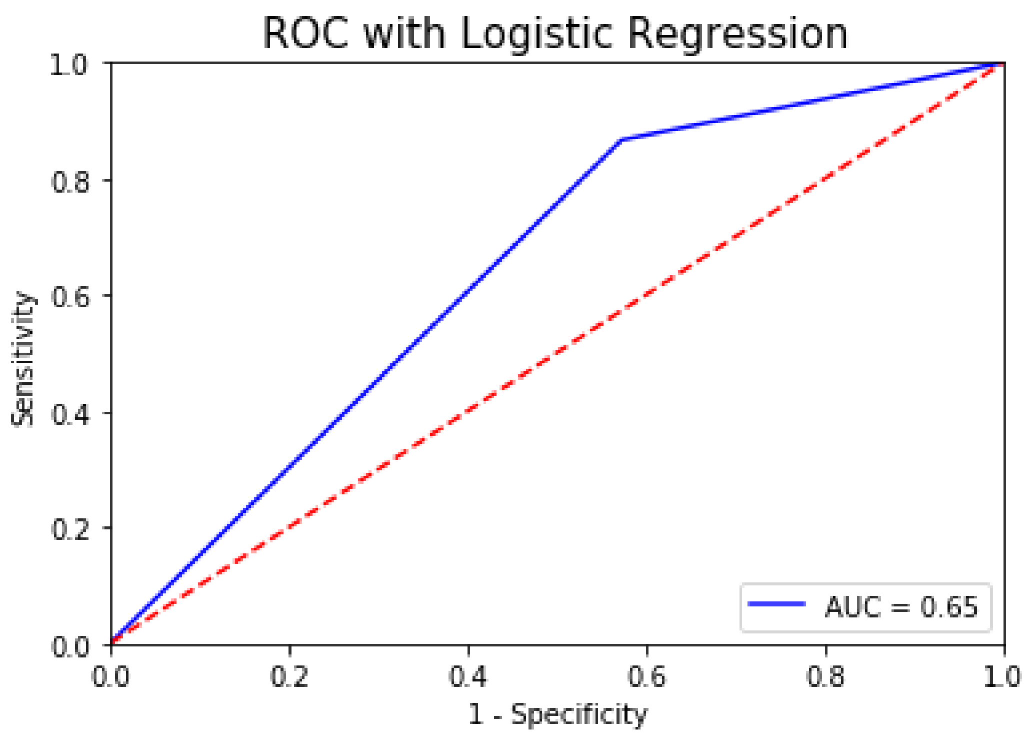
Both LDA and QDA have their test error decreasing when sample size increases, but LDA did not necessarily get better than QDA. The distance in test error between LDA and QDA remains unchanged. This confirms that the larger the sample size, the more generalizable the model to new data (less overfitting) which leads to a lower test error. The test error rate plateaus after n = 10000, where the sample size is large enough that the size does not affect model fitting anymore. However, the QDA model, no matter how large the sample size is, always performs better than LDA model with a lower test error rate. Given we retrieved our data from a quadratic random variable, QDA in theory would have greater predictive power than LDA. Our result confirmed this. LDA does not work well under non-linearity, when covariance matrix is not universal across all classes.

5 Question 5

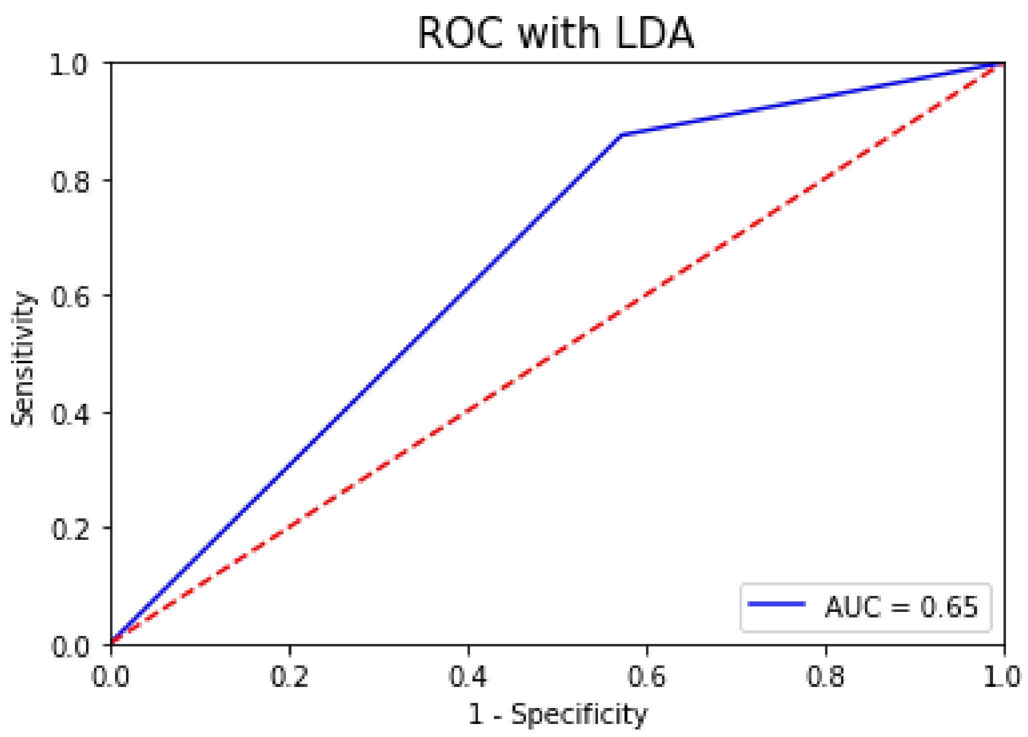
[10]:
```
# reading and preparing the data
mental_health_df = pd.read_csv('mental_health.csv')
mental_health_df.dropna(inplace=True)
outcome_data = mental_health_df['vote96']
factor_data = mental_health_df.drop(['vote96'], axis=1)

split_dataset = train_test_split(factor_data, outcome_data, test_size=0.3,␣
 ↪random_state=1020)
```

After preparing the data, we fit different models including Logistic Regression, LDA, QDA, Naives Bayes (Gaussian) and K1~10 Nearest Neighbors:

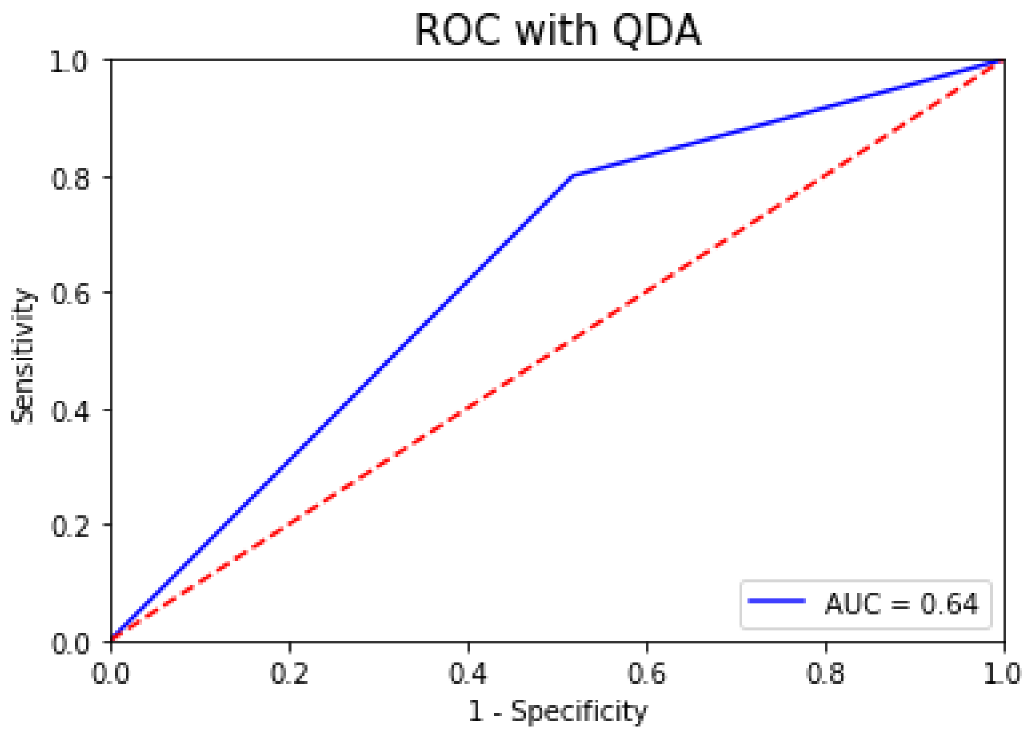
[11]:
```
# collecting error rates and AUC scores
mod_errors = {}

# Logistic Regression
log_err = fit_model(split_dataset=split_dataset, model='log')
mod_errors['Logistic Regression'] = log_err
```



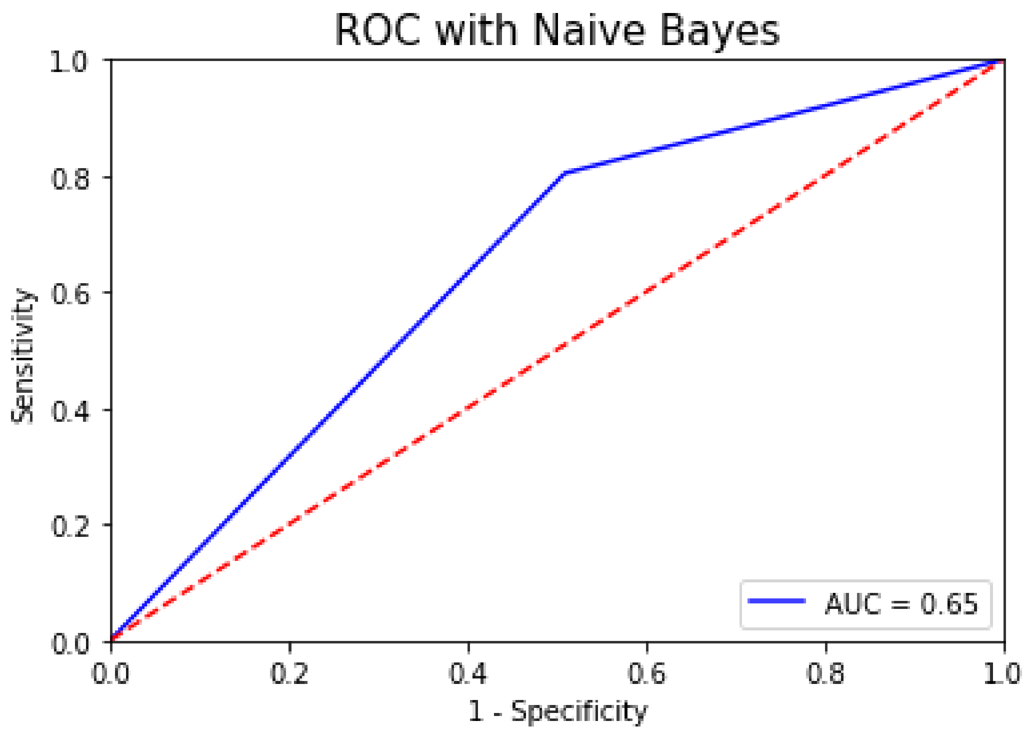
[12]:
```
# Linear Discriminant Analysis
lda_err = fit_model(split_dataset=split_dataset, model='lda')
mod_errors['LDA'] = lda_err
```



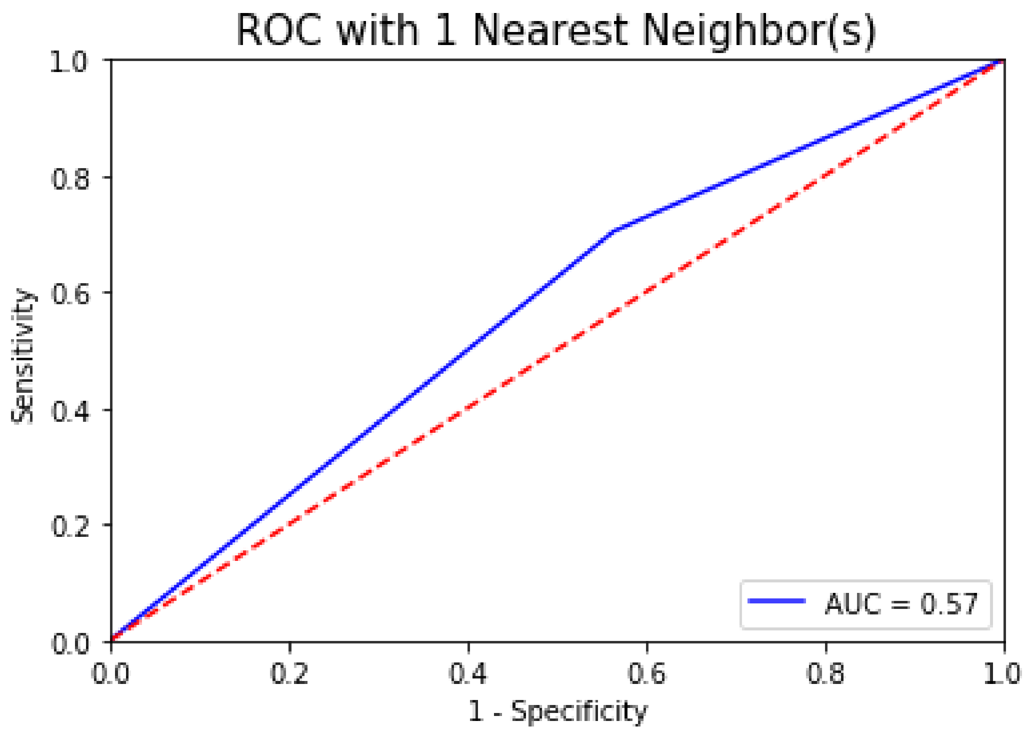
[13]:
```
# Quadratic Discriminant Analysis
qda_err = fit_model(split_dataset=split_dataset, model='qda')
mod_errors['QDA'] = qda_err
```



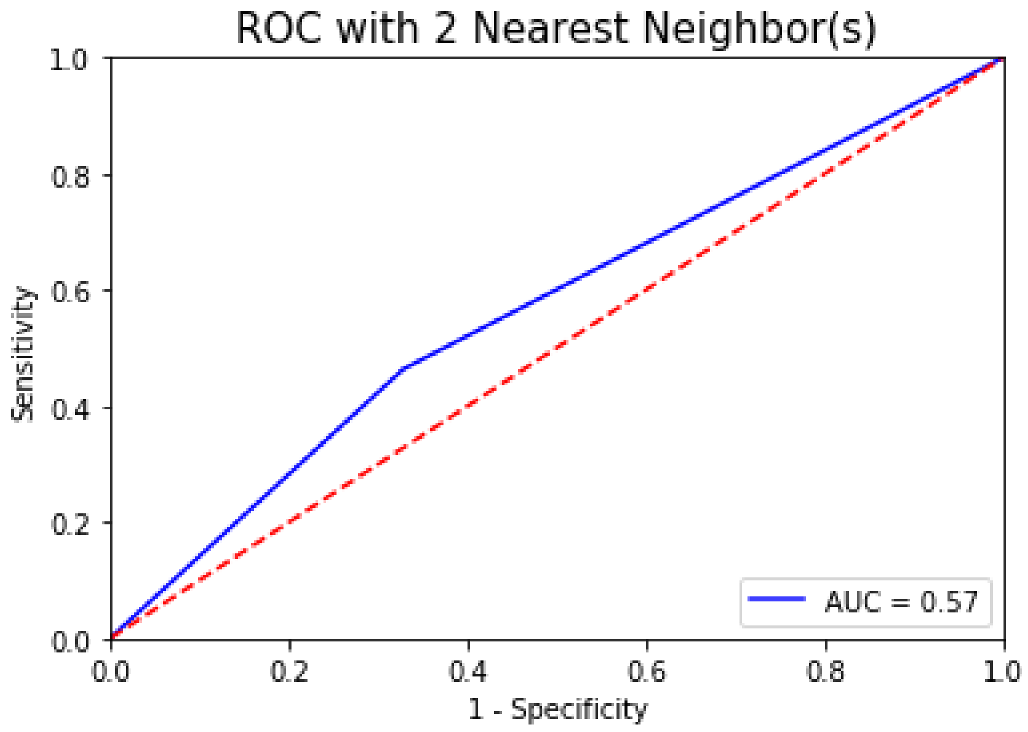
[14]:
```
# Naive Bayes
nb_err = fit_model(split_dataset=split_dataset, model='nb')
mod_errors['Naive Bayes'] = nb_err
```



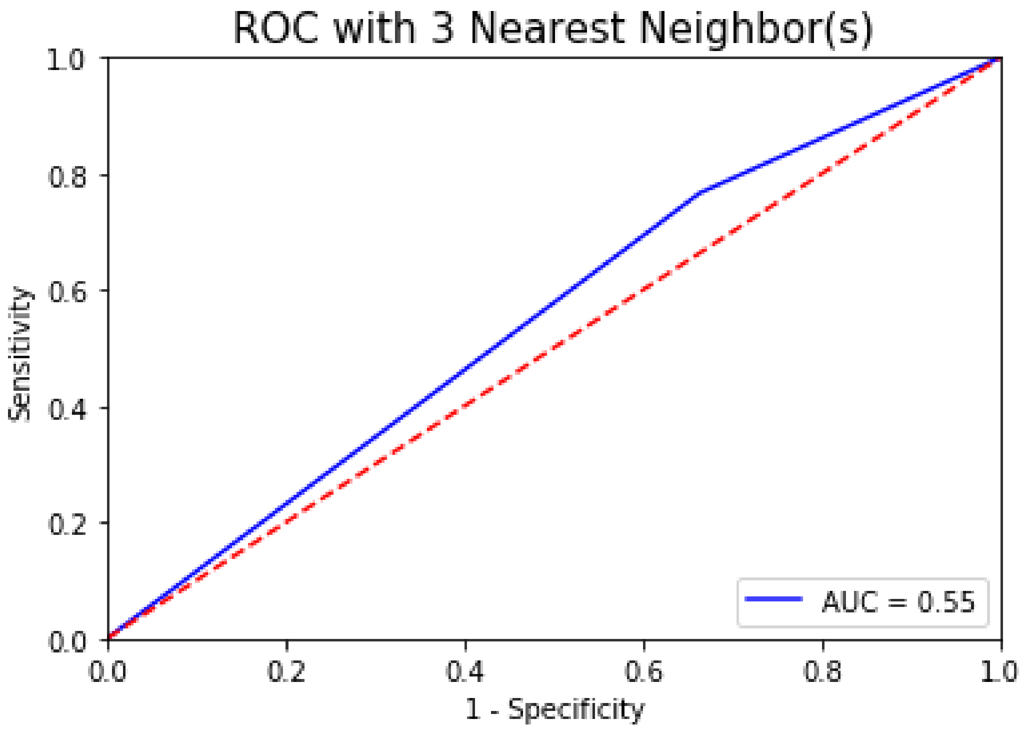
```
[15]: # 1 Nearest Neighbor
      k1_err = fit_model(split_dataset=split_dataset, model='knn', k=1)
      mod_errors['1 Nearest Neighbor'] = k1_err
```



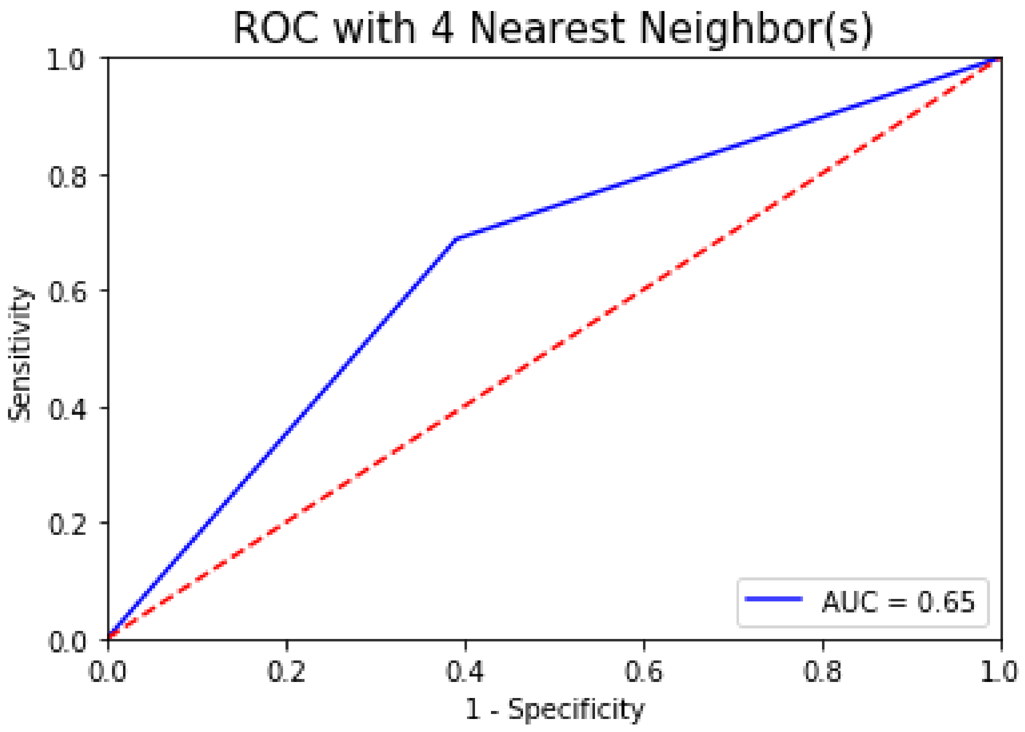
```
[16]: # 2 Nearest Neighbors
      k2_err = fit_model(split_dataset=split_dataset, model='knn', k=2)
      mod_errors['2 Nearest Neighbors'] = k2_err
```



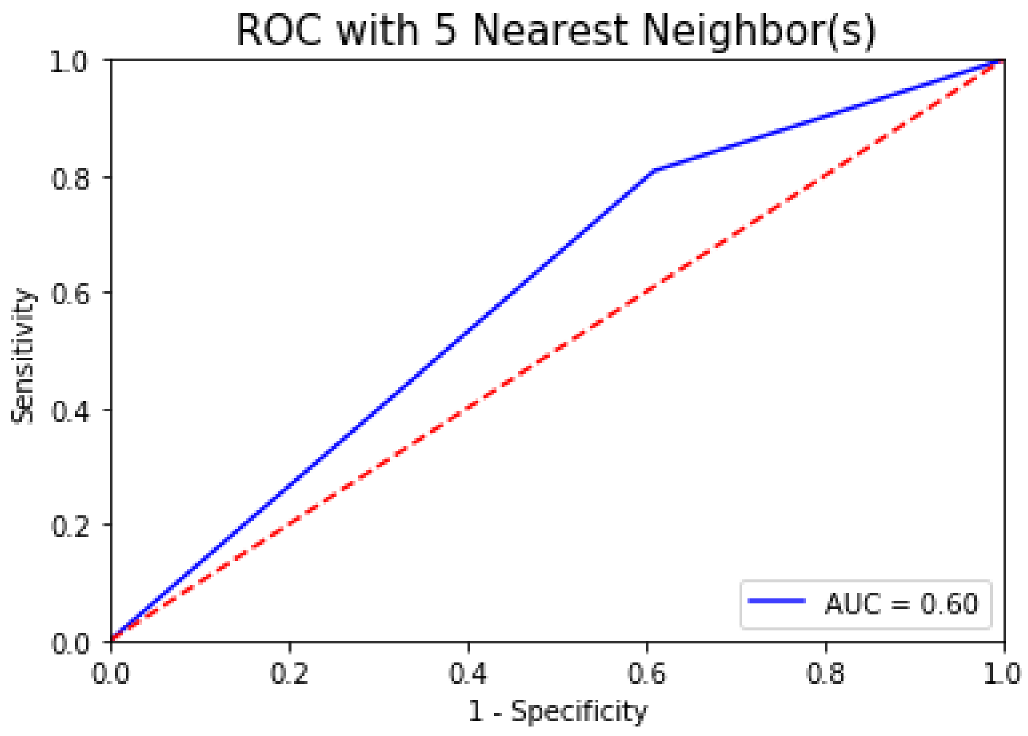
[17]:
```
# 3 Nearest Neighbors
k3_err = fit_model(split_dataset=split_dataset, model='knn', k=3)
mod_errors['3 Nearest Neighbors'] = k1_err
```



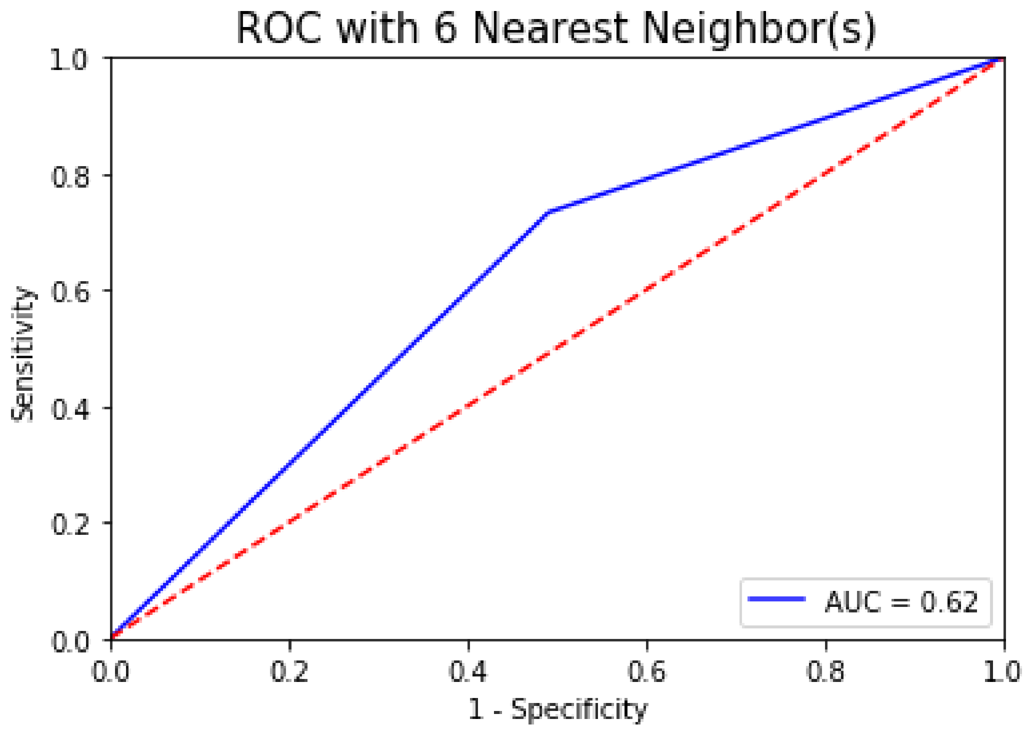
[18]:
```
# 4 Nearest Neighbors
k4_err = fit_model(split_dataset=split_dataset, model='knn', k=4)
mod_errors['4 Nearest Neighbors'] = k4_err
```



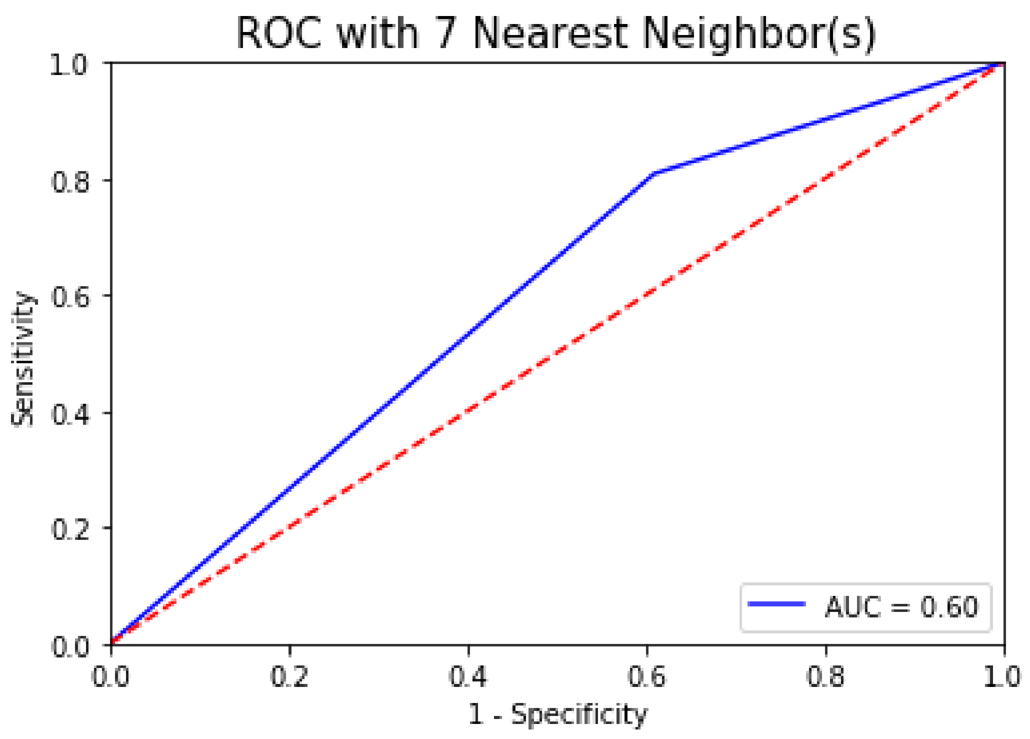
[19]:
```
# 5 Nearest Neighbors
k5_err = fit_model(split_dataset=split_dataset, model='knn', k=5)
mod_errors['5 Nearest Neighbors'] = k5_err
```



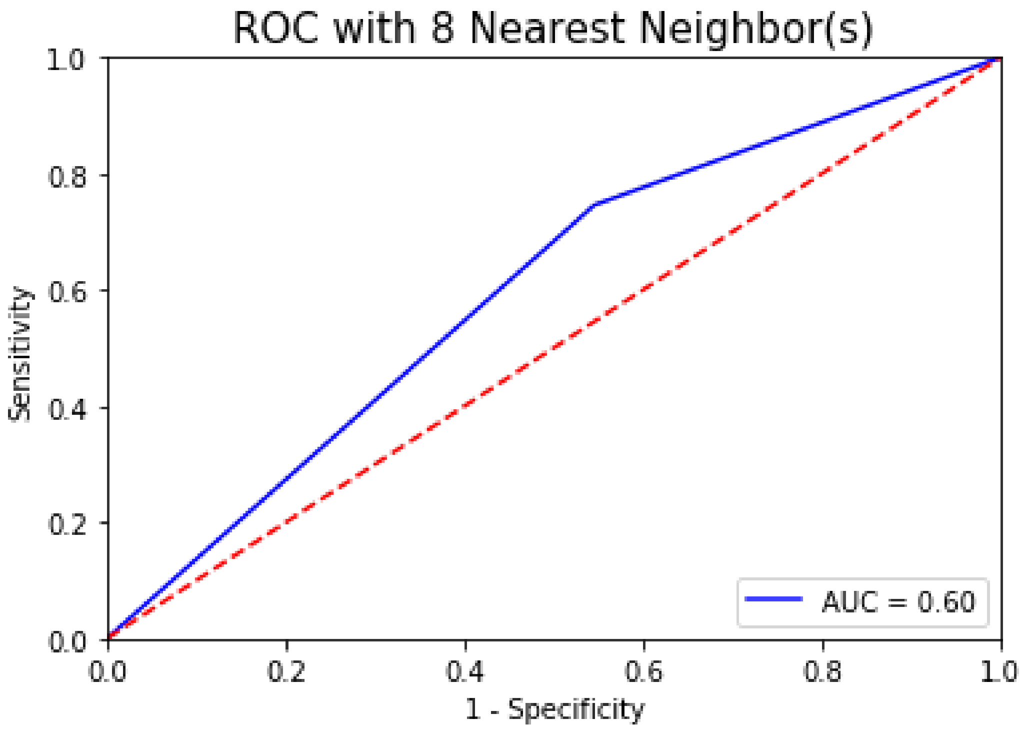
[20]:
```
# 6 Nearest Neighbors
k6_err = fit_model(split_dataset=split_dataset, model='knn', k=6)
mod_errors['6 Nearest Neighbors'] = k6_err
```



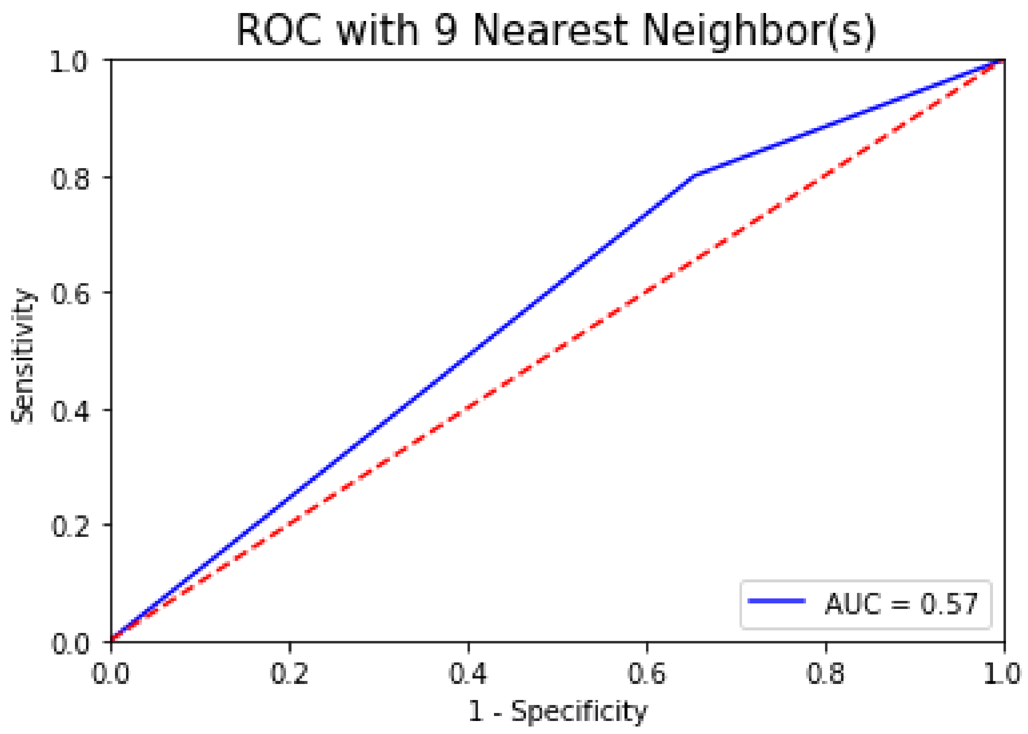
[21]:
```
# 7 Nearest Neighbors
k7_err = fit_model(split_dataset=split_dataset, model='knn', k=7)
mod_errors['7 Nearest Neighbors'] = k7_err
```



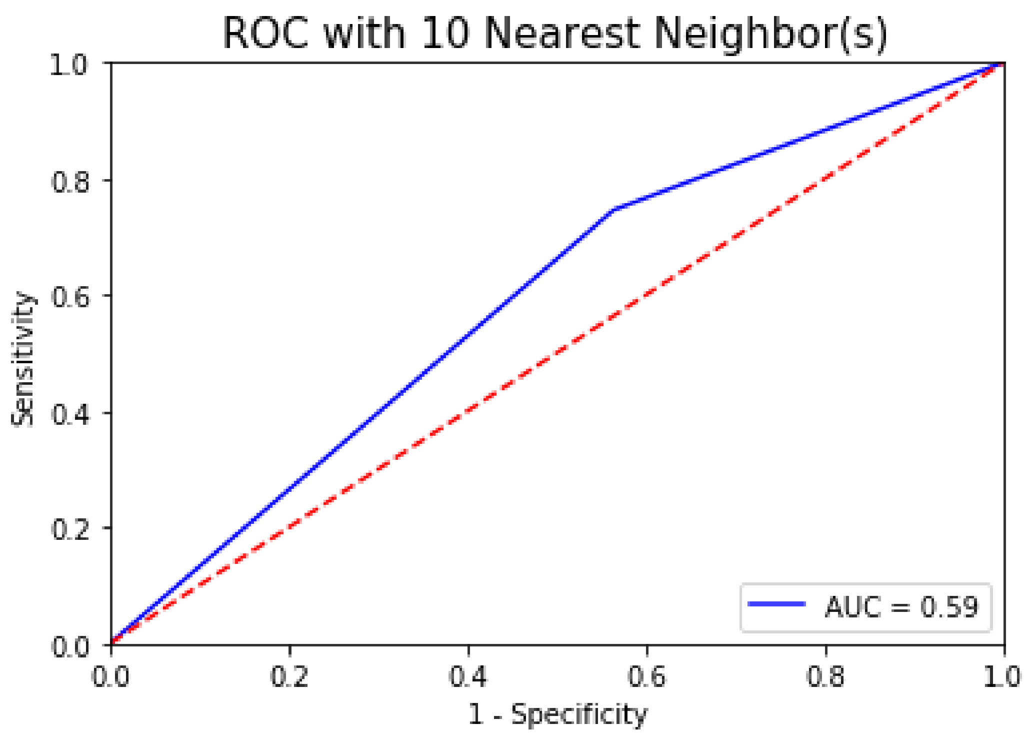
```
[22]: # 8 Nearest Neighbors
      k8_err = fit_model(split_dataset=split_dataset, model='knn', k=8)
      mod_errors['8 Nearest Neighbors'] = k8_err
```



[23]:
```
# 9 Nearest Neighbors
k9_err = fit_model(split_dataset=split_dataset, model='knn', k=9)
mod_errors['9 Nearest Neighbors'] = k9_err
```



[24]:
```
# 10 Nearest Neighbors
k10_err = fit_model(split_dataset=split_dataset, model='knn', k=10)
mod_errors['10 Nearest Neighbors'] = k10_err
```



Then we compare the models by their error rates and AUC:

```
[25]: errors_df = pd.DataFrame(mod_errors, index=['Train Err', 'Test Err',␣
       ↪'AUC-ROC']).T
      errors_df.sort_values(by='Test Err', axis=0) # sorted by test error rate
```

```
[25]:                       Train Err  Test Err   AUC-ROC
      LDA                    0.278528  0.265714  0.651136
      Logistic Regression    0.268712  0.271429  0.646970
      Naive Bayes            0.300613  0.294286  0.647538
      QDA                    0.269939  0.300000  0.640909
      5 Nearest Neighbors    0.218405  0.322857  0.599621
      7 Nearest Neighbors    0.239264  0.322857  0.599621
      4 Nearest Neighbors    0.235583  0.337143  0.648295
      6 Nearest Neighbors    0.241718  0.337143  0.621212
      9 Nearest Neighbors    0.250307  0.342857  0.572727
      8 Nearest Neighbors    0.235583  0.345714  0.600189
      10 Nearest Neighbors   0.256442  0.351429  0.591098
      1 Nearest Neighbor     0.001227  0.380000  0.570265
      3 Nearest Neighbors    0.001227  0.380000  0.570265
      2 Nearest Neighbors    0.168098  0.471429  0.567614
```

```
[26]: errors_df.sort_values(by='AUC-ROC', axis=0, ascending=False) # sorted by AUC-ROC
```

[26]:
```
                      Train Err  Test Err   AUC-ROC
LDA                    0.278528  0.265714  0.651136
4 Nearest Neighbors    0.235583  0.337143  0.648295
Naive Bayes            0.300613  0.294286  0.647538
Logistic Regression    0.268712  0.271429  0.646970
QDA                    0.269939  0.300000  0.640909
6 Nearest Neighbors    0.241718  0.337143  0.621212
8 Nearest Neighbors    0.235583  0.345714  0.600189
5 Nearest Neighbors    0.218405  0.322857  0.599621
7 Nearest Neighbors    0.239264  0.322857  0.599621
10 Nearest Neighbors   0.256442  0.351429  0.591098
9 Nearest Neighbors    0.250307  0.342857  0.572727
1 Nearest Neighbor     0.001227  0.380000  0.570265
3 Nearest Neighbors    0.001227  0.380000  0.570265
2 Nearest Neighbors    0.168098  0.471429  0.567614
```

When choosing a classifier model, we first want it to have low test error rate, so that it has good generalizability whenever it meets new data. Second, we want the classifier to predict actual 0's as 0 and actual 1's as 1, in another word, to produce as little false positives as possible. Hence, we are looking for the model with the lowest test error rate possible and largest AUC score possible. In our case, LDA possesses both the lowest test error rate and the highest AUC score. Therefore, in my opinion, LDA is the best-performing model in this scenario.