Wang_Miaohan_HW2

February 2, 2020

[1]:
```
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]:
```
# 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()
```

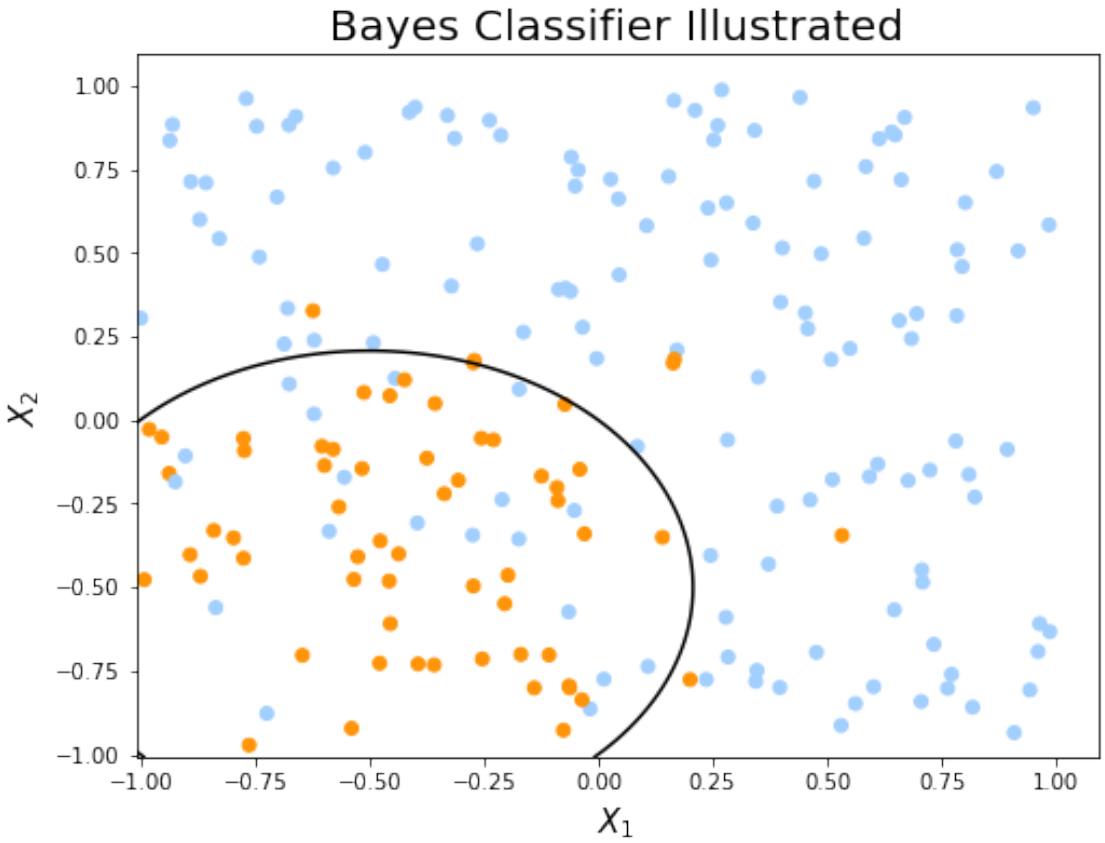
```
        elif model == 'qda':
            mod = QuadraticDiscriminantAnalysis()

        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

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'])
df2
```

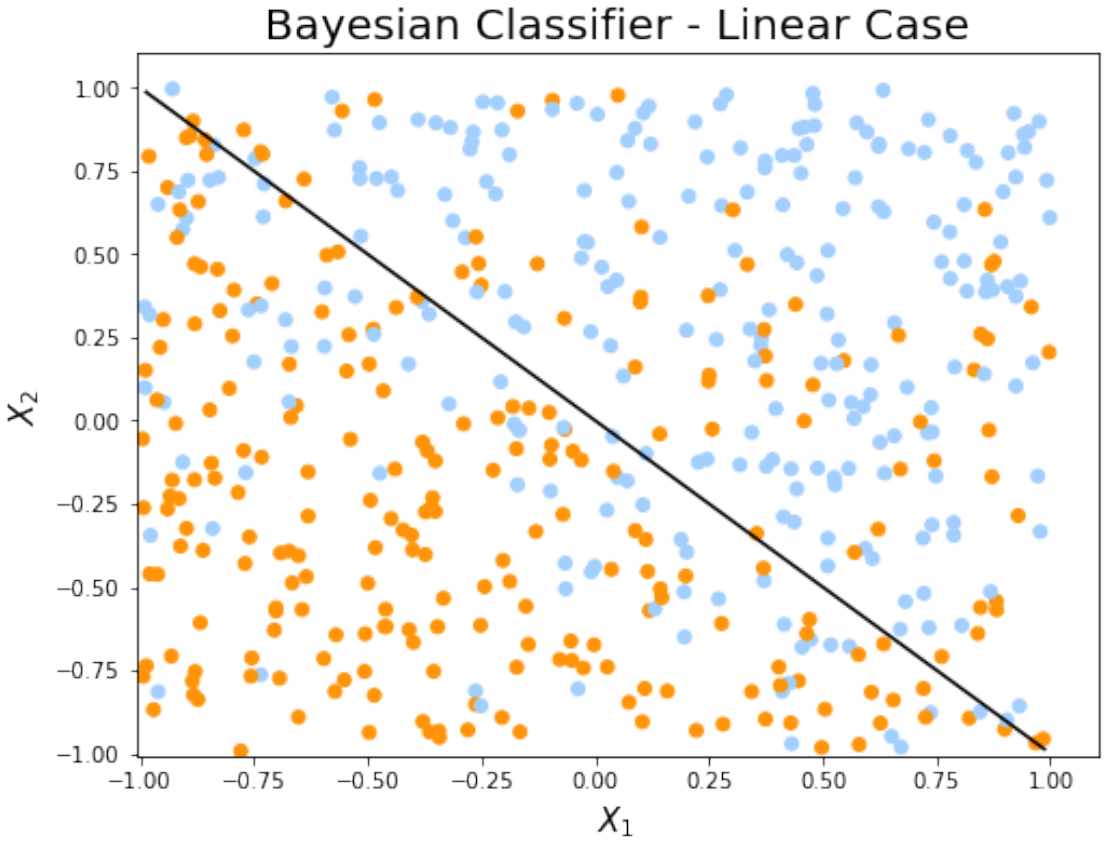
[5]:

	LDA	QDA
Train Error	0.273826	0.272657
Test Error	0.277313	0.276843

We observe that with $f(X) = X_1 + X_2 + \epsilon$, the error rates between LDA and QDA modeling are very close. This is because the covariance of X_1 and X_2 are very similar in both classes across the boundary. (As illustrated in the following graph, the ratio of orange dots to blue dots inside the boundary is very similar to the ratio of blue dots to orange dots outside the bounary.) The data

given by $f(X)$ here is linear, which fits the assumption of LDA that all classes shares the same covariance matrix. Thus, LDA and QDA are not differentiated in performance. However, this is not the case when $f(X)$ becomes non-linear.

[6]:
```
X_1 = np.random.uniform(-1, 1, 500)
X_2 = np.random.uniform(-1, 1, 500)
plt.figure(figsize=(8,6))
plt.title('Bayesian Classifier - Linear Case', fontsize=20)
plt.scatter(X_1, X_2, c=np.where(X_1 + X_2 + np.random.normal(0,1,500) > 0,␣
↪'#a2cffe', '#ff9408'))
plt.xlabel('$X_1$', fontsize=15)
plt.ylabel('$X_2$', fontsize=15)

C = A + B
prob_C = np.exp(C) / (1 + np.exp(C))
plt.contour(A, B, prob_C, levels=[0.5], colors='black');
```



3 Question 3

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:

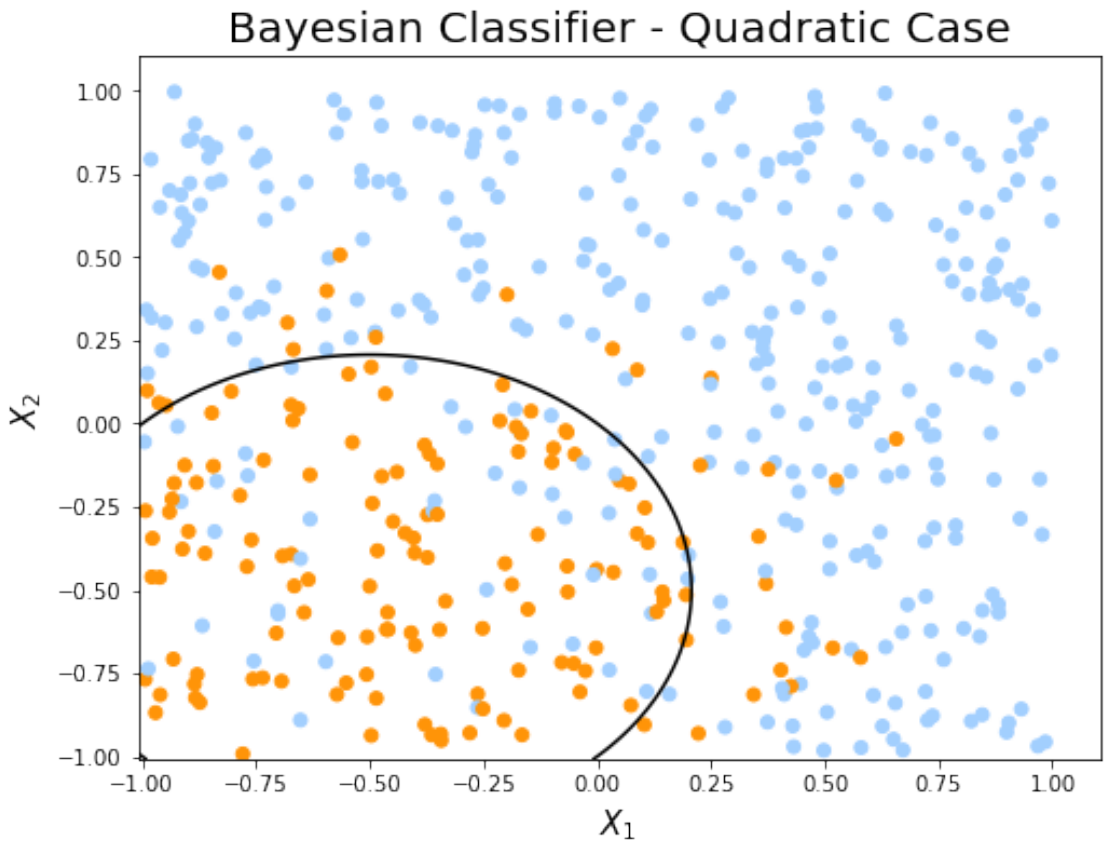
[7]:
```
LDA_err_Q3 = simulation(size=1000, rep=1000, model='lda', linearity=False)
QDA_err_Q3 = simulation(size=1000, rep=1000, model='qda', linearity=False)
df3 = pd.DataFrame({'LDA': LDA_err_Q3, 'QDA': QDA_err_Q3}, index=['Train␣
 ↪Error', 'Test Error'])
df3
```

[7]:

	LDA	QDA
Train Error	0.273326	0.259293
Test Error	0.274633	0.261803

When we switch to $f(X) = X_1 + X_1^2 + X_2 + X_2^2 + \epsilon$, the performances of LDA and QDA are differentiated. QDA has both a lower train error rate and a lower test error rate. This is because the covariance in and out of the decision boundary is not the same. As illustrated below, the spread of orange dots is much more sparse outside the boundary comparing to that of blue dots inside the boundary. When the two classes don't share the same covariance matrix, the assumption of LDA is violated. QDA therefore adapts to such a non-linear case better, giving a better predictive performance.

[8]:
```
plt.figure(figsize=(8,6))
plt.title('Bayesian Classifier - Quadratic Case', fontsize = 20)
plt.scatter(X_1, X_2, c=np.where(X_1 + X_1**2 + X_2 + X_2**2 + np.random.
 ↪normal(0, 0.5, 500) > 0, '#a2cffe', '#ff9408'))
plt.xlabel('$X_1$', fontsize=15)
plt.ylabel('$X_2$', fontsize=15)

C = A + B + A**2 + B**2
prob_C = np.exp(C) / (1 + np.exp(C))
plt.contour(A, B, prob_C, levels=[0.5], colors='black');
```



4 Question 4

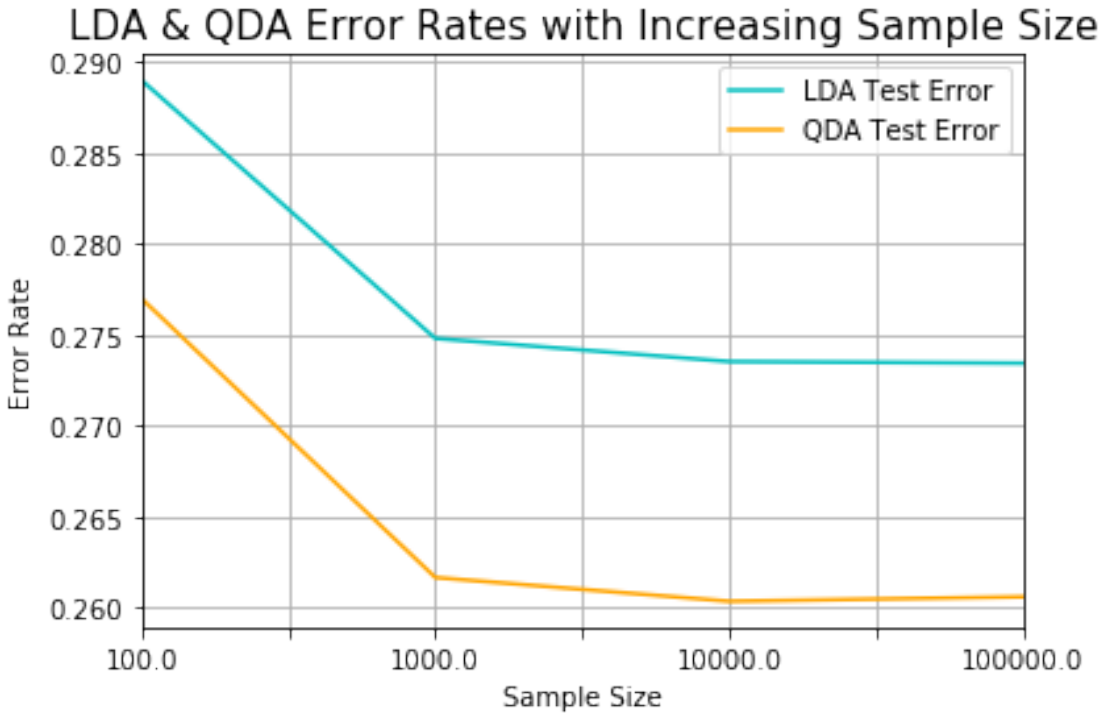
[9]:
```
lda_results = {}
qda_results = {}
trials = [1e02, 1e03, 1e04, 1e05] # sample sizes for simulating non-linear data␣
 ↪

# 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]
```

[10]:
```
# 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)
df_Q4
```

[10]:

	LDA Test Error	QDA Test Error
100.0	0.289000	0.277000
1000.0	0.274813	0.261673
10000.0	0.273541	0.260367
100000.0	0.273451	0.260611

[11]:
```
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. 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 substantially 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

[12]:
```
# 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']

# removing irrelevant noise parameters 'black' and 'married', not stated in␣
 ↪relevant theories
factor_data = mental_health_df.drop(['vote96', 'black', 'married'], axis=1)

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

[13]:
```
split_dataset[1]
```

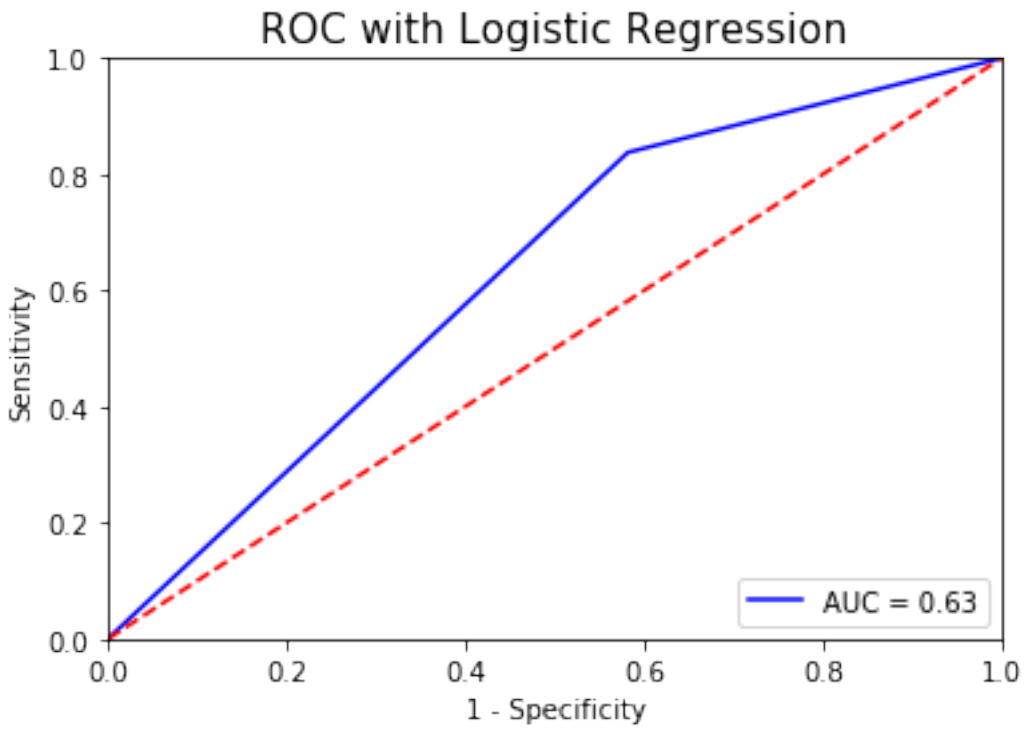
[13]:

	mhealth_sum	age	educ	female	inc10
564	0.0	36.0	15.0	0	5.8849
1913	7.0	45.0	6.0	1	1.7387
2147	1.0	42.0	16.0	0	10.6998
1094	3.0	39.0	11.0	0	4.0124
1653	0.0	52.0	12.0	1	7.2223
…	…	…	…	…	
2071	5.0	47.0	12.0	0	4.0124
2388	4.0	43.0	9.0	1	1.7387
74	2.0	25.0	14.0	1	1.4712
667	4.0	45.0	12.0	0	3.4774
1771	1.0	21.0	15.0	1	2.5412

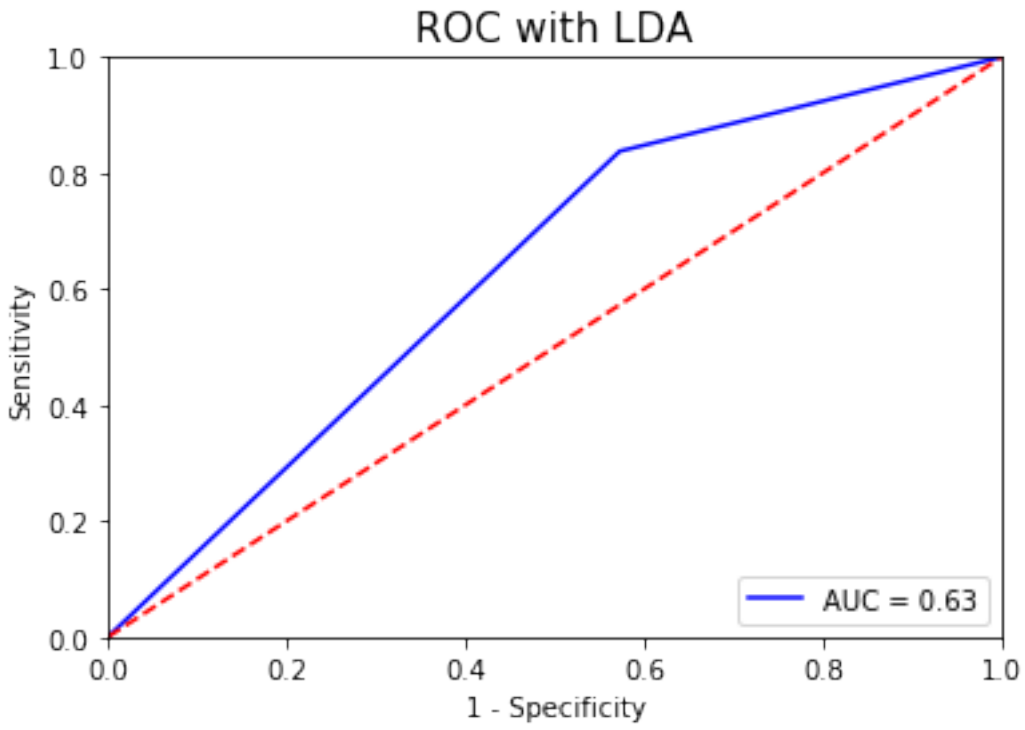
[350 rows x 5 columns]

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

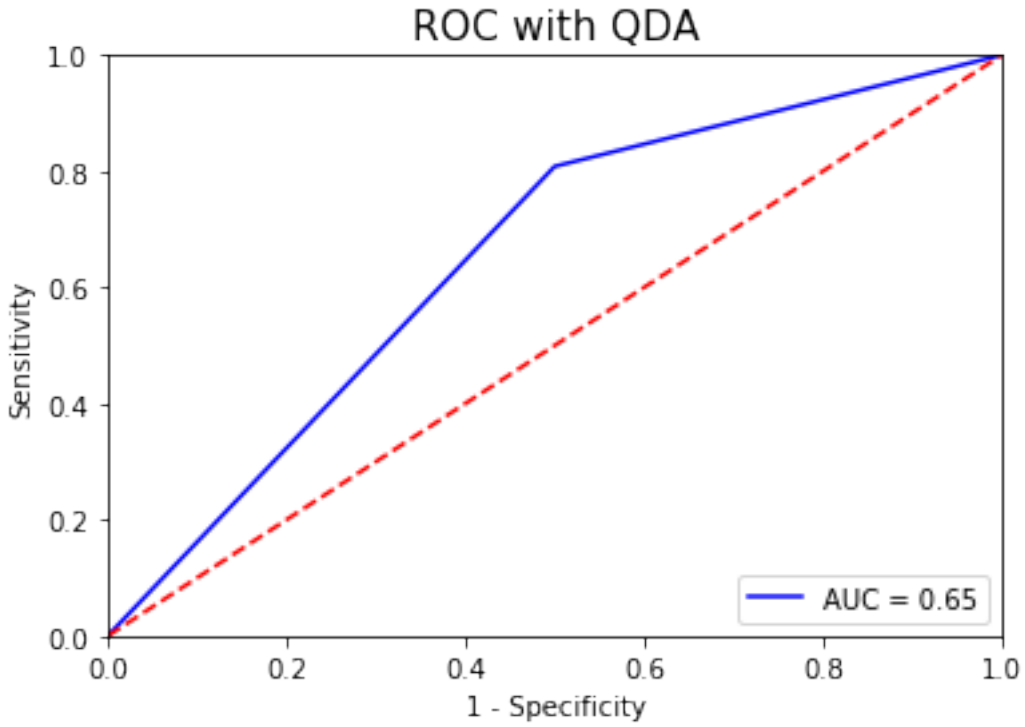
[14]:
```
mod_errors = {}

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



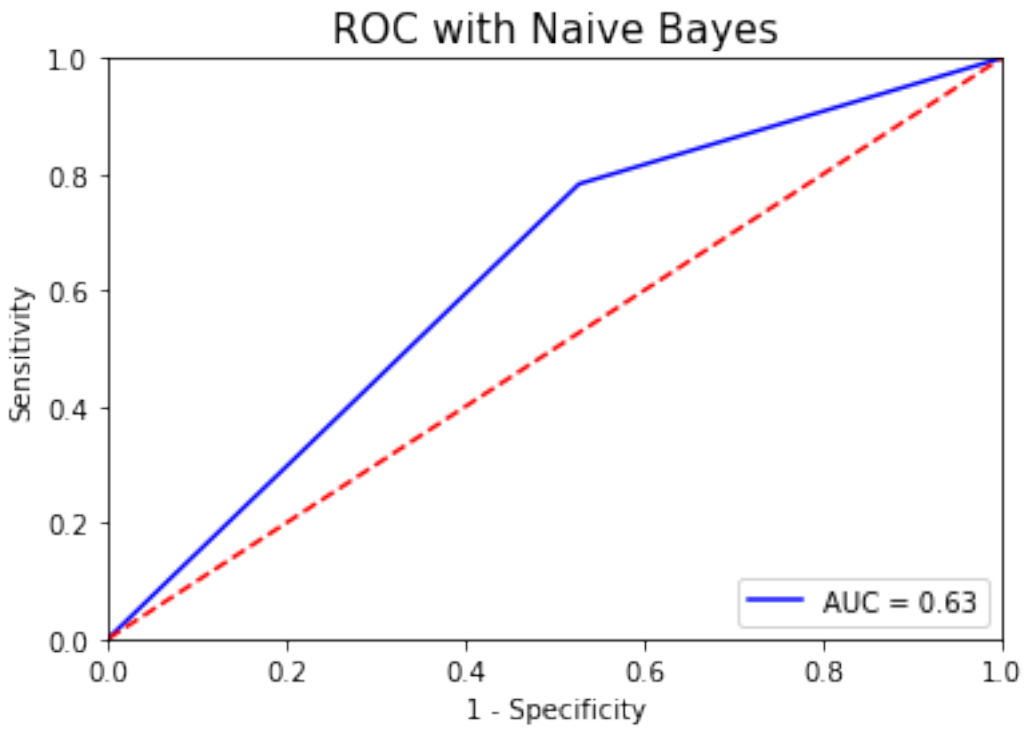
[15]:
```
lda_err = fit_model(split_dataset=split_dataset, model='lda')
mod_errors['LDA'] = lda_err
```



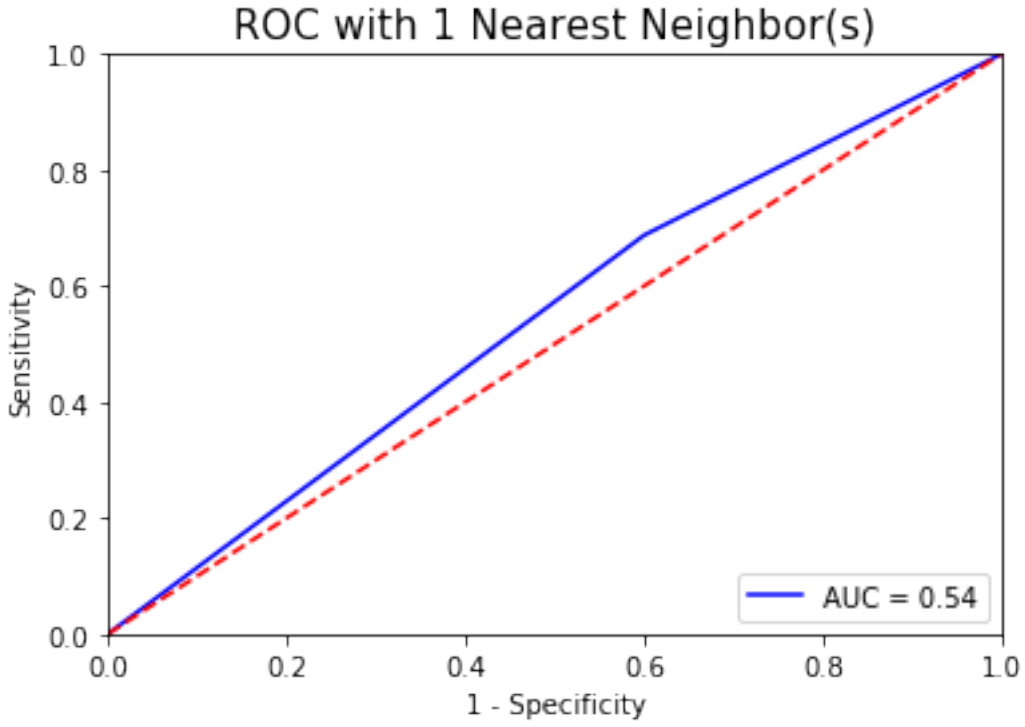
[16]:
```
qda_err = fit_model(split_dataset=split_dataset, model='qda')
mod_errors['QDA'] = qda_err
```



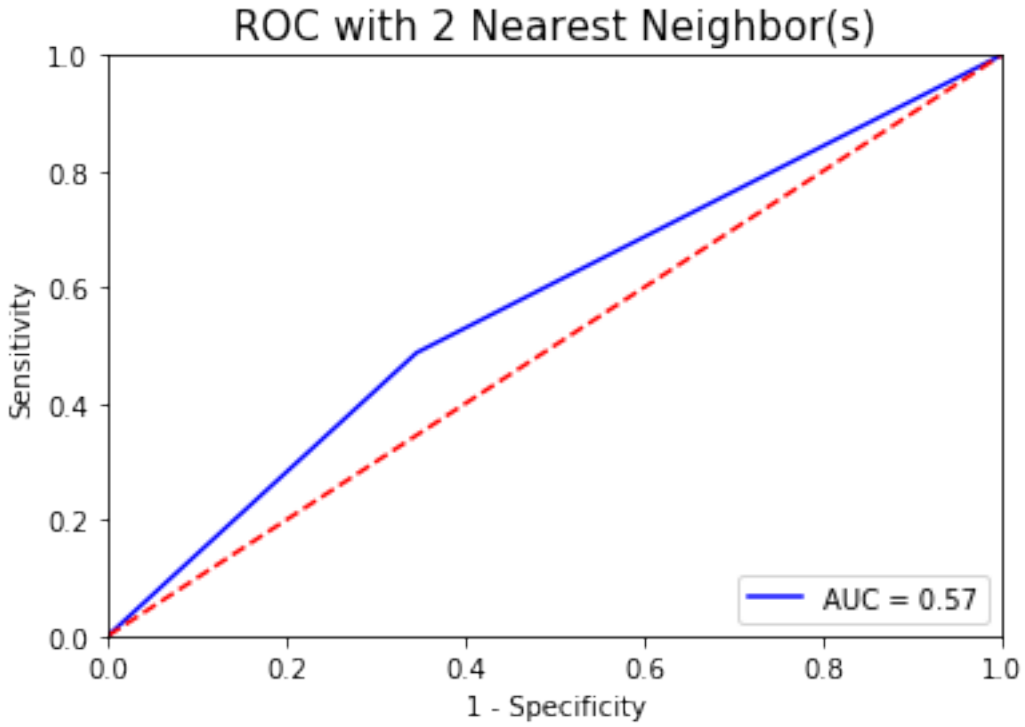
[17]:
```
nb_err = fit_model(split_dataset=split_dataset, model='nb')
mod_errors['Naive Bayes'] = nb_err
```



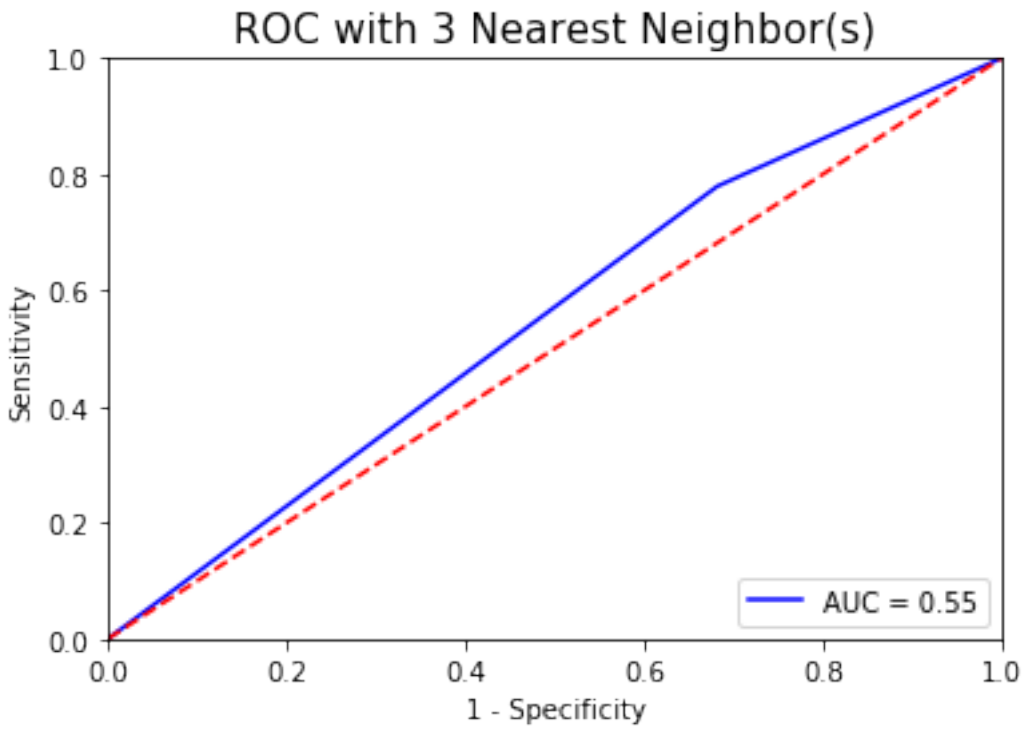
[18]:
```
k1_err = fit_model(split_dataset=split_dataset, model='knn', k=1)
mod_errors['1 Nearest Neighbor'] = k1_err
```



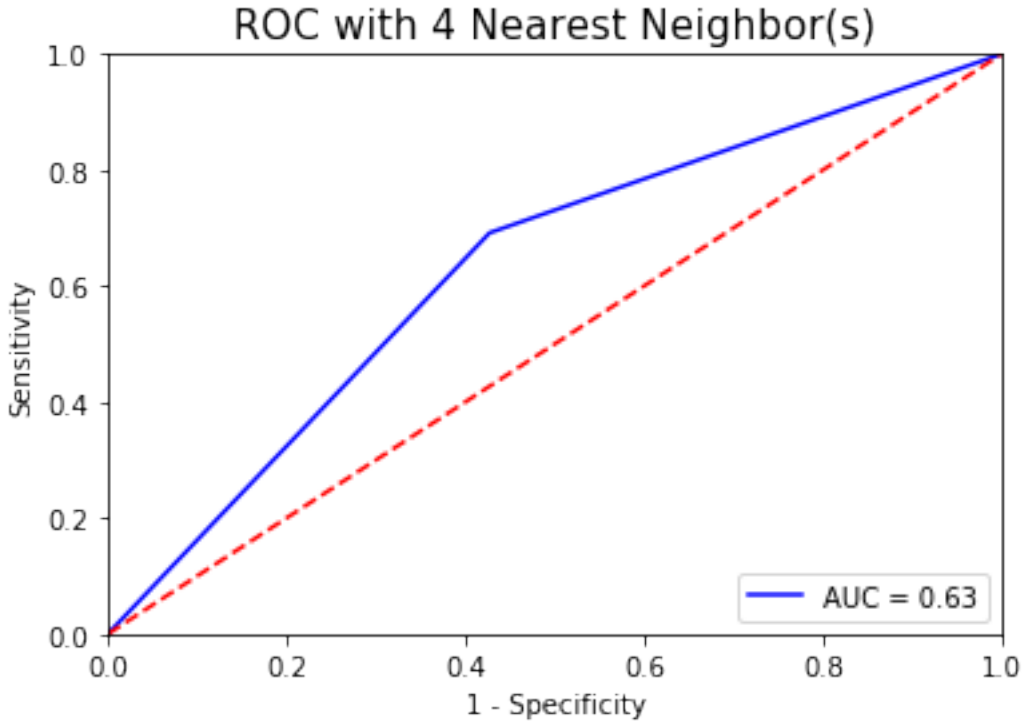
[19]:
```
k2_err = fit_model(split_dataset=split_dataset, model='knn', k=2)
mod_errors['2 Nearest Neighbors'] = k2_err
```



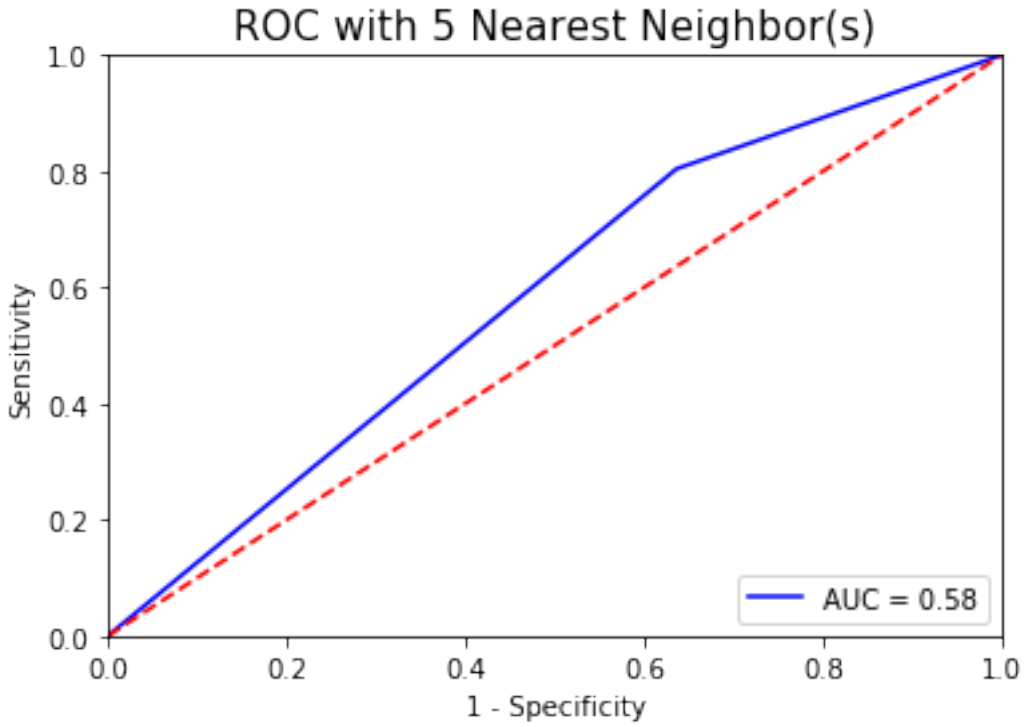
[20]:
```
k3_err = fit_model(split_dataset=split_dataset, model='knn', k=3)
mod_errors['3 Nearest Neighbors'] = k1_err
```



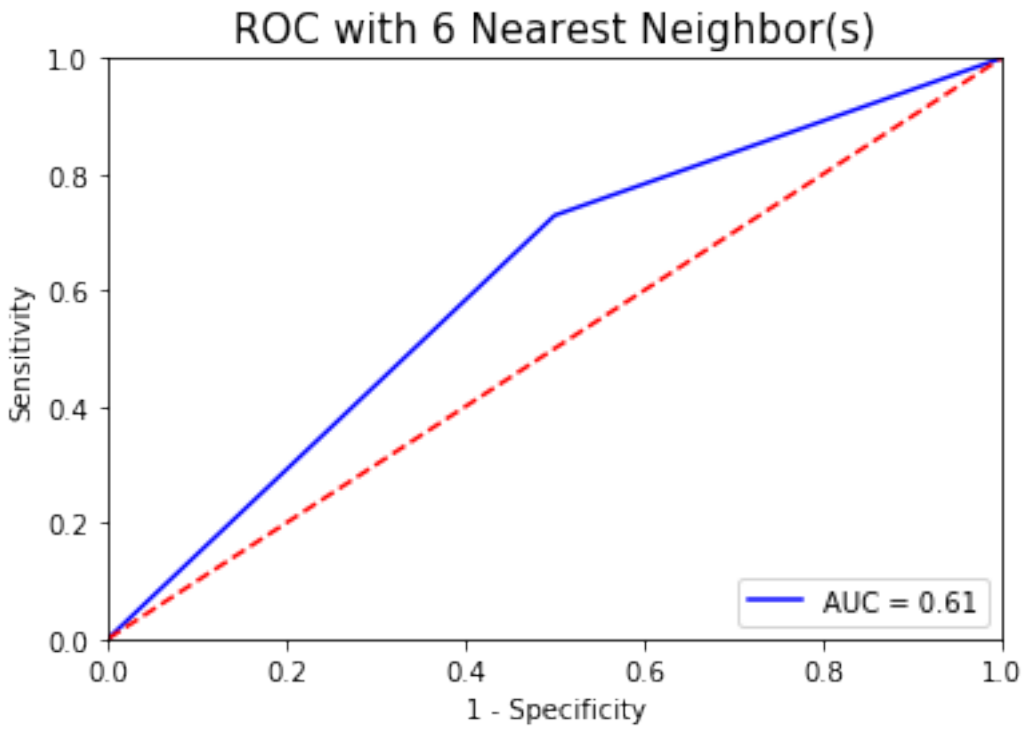
[21]:
```
k4_err = fit_model(split_dataset=split_dataset, model='knn', k=4)
mod_errors['4 Nearest Neighbors'] = k4_err
```



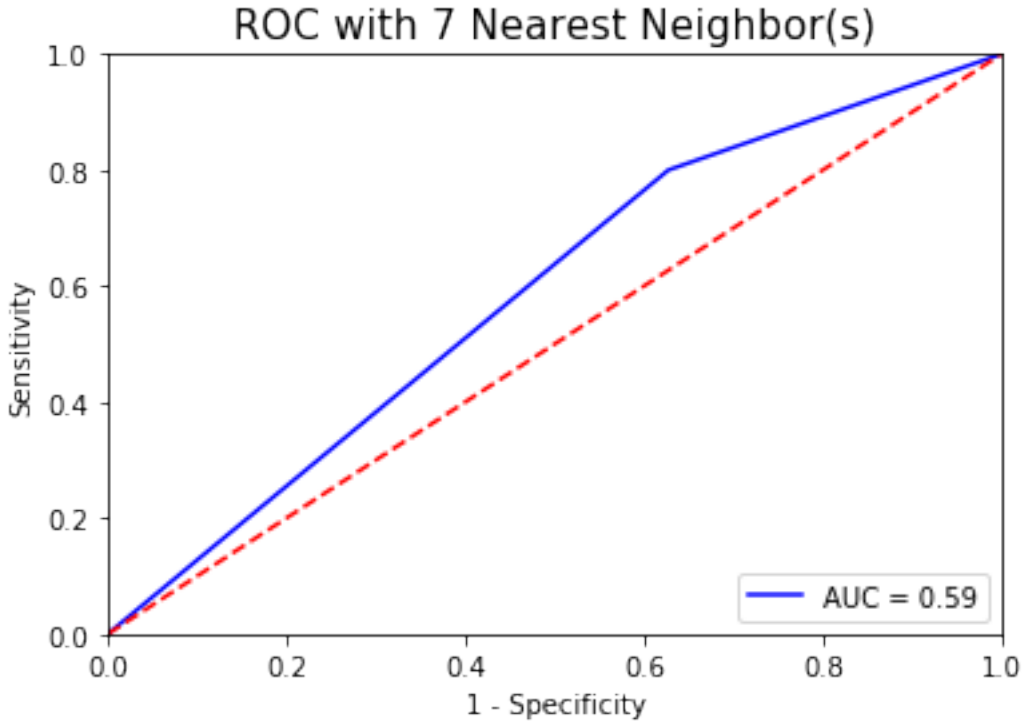
[22]:
```
k5_err = fit_model(split_dataset=split_dataset, model='knn', k=5)
mod_errors['5 Nearest Neighbors'] = k5_err
```



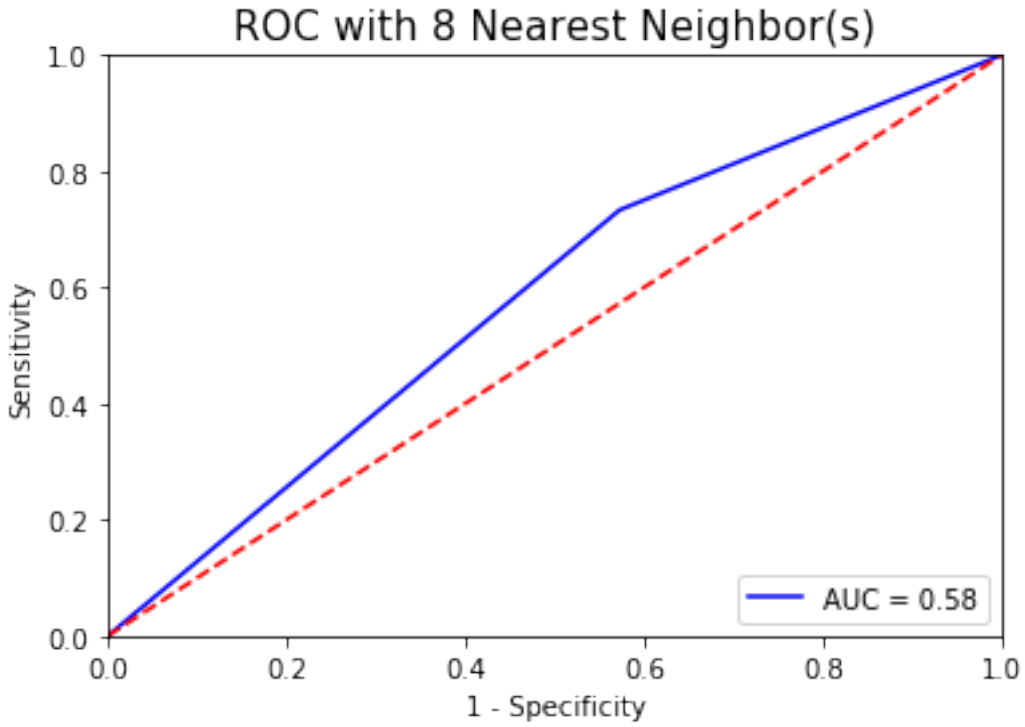
[23]:
```
k6_err = fit_model(split_dataset=split_dataset, model='knn', k=6)
mod_errors['6 Nearest Neighbors'] = k6_err
```



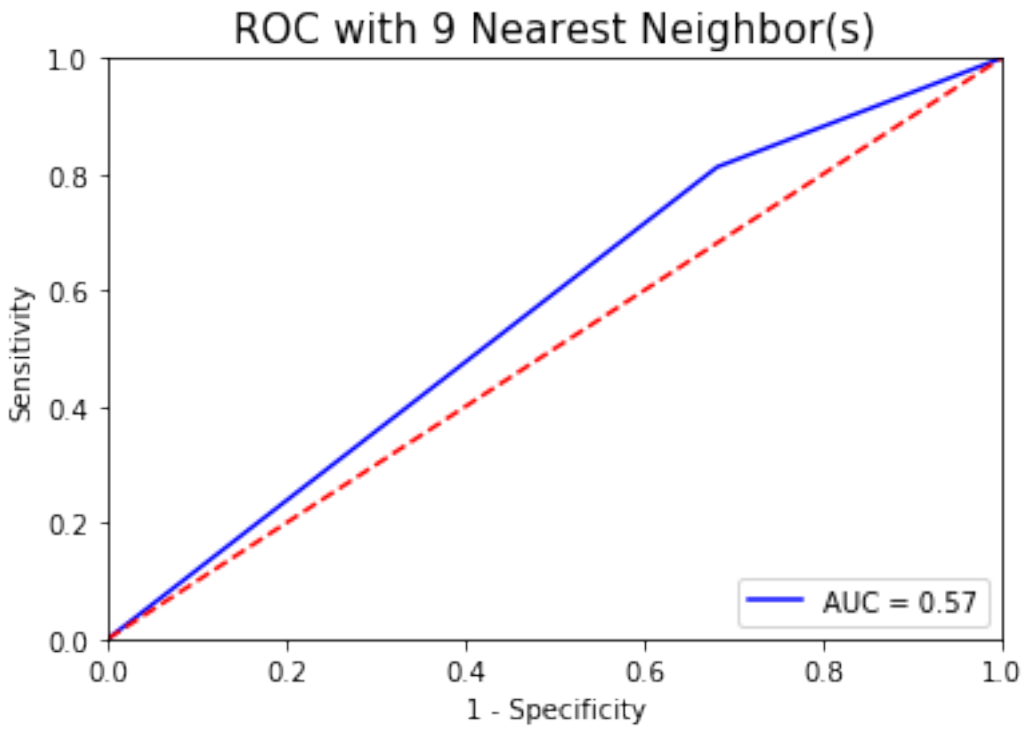
[24]:
```
k7_err = fit_model(split_dataset=split_dataset, model='knn', k=7)
mod_errors['7 Nearest Neighbors'] = k7_err
```



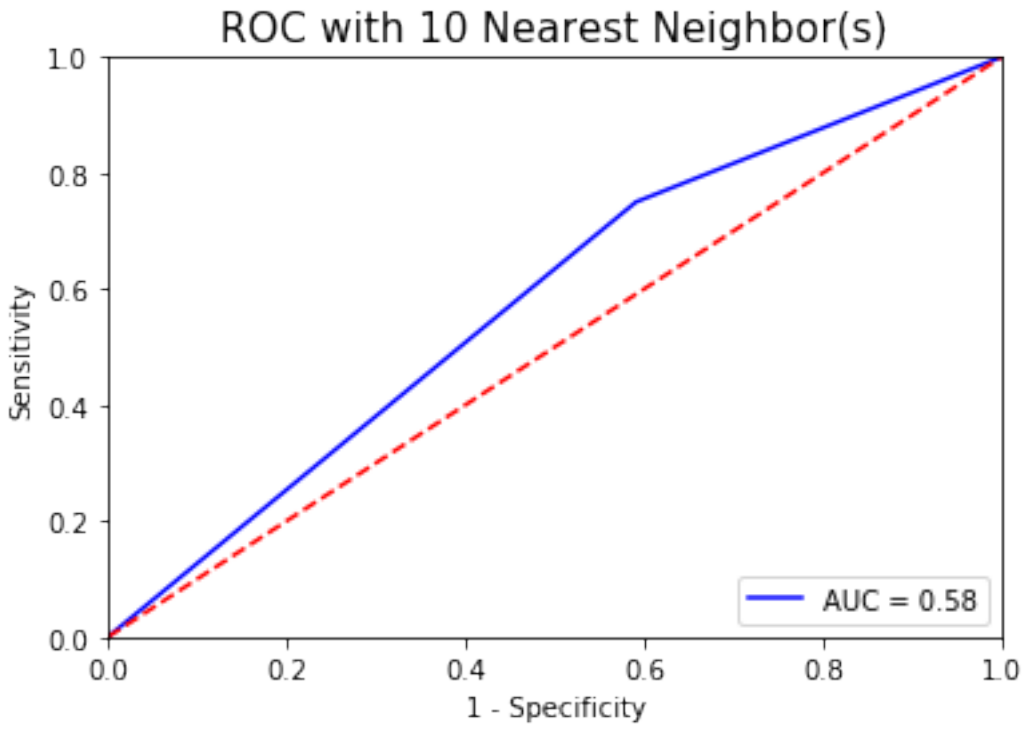
[25]:
```
k8_err = fit_model(split_dataset=split_dataset, model='knn', k=8)
mod_errors['8 Nearest Neighbors'] = k8_err
```



[26]:
```
k9_err = fit_model(split_dataset=split_dataset, model='knn', k=9)
mod_errors['9 Nearest Neighbors'] = k9_err
```



[27]:
```
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:

[28]:
```
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
```

[28]:

	Train Err	Test Err	AUC-ROC
QDA	0.274847	0.288571	0.654167
LDA	0.273620	0.291429	0.632386
Logistic Regression	0.276074	0.294286	0.627841
Naive Bayes	0.274847	0.314286	0.628030
5 Nearest Neighbors	0.218405	0.334286	0.583902
7 Nearest Neighbors	0.235583	0.334286	0.586364
6 Nearest Neighbors	0.236810	0.342857	0.614583
9 Nearest Neighbors	0.258896	0.342857	0.565341
4 Nearest Neighbors	0.226994	0.345714	0.632197
10 Nearest Neighbors	0.261350	0.357143	0.579545
8 Nearest Neighbors	0.238037	0.362857	0.580303
1 Nearest Neighbor	0.001227	0.402857	0.543750
3 Nearest Neighbors	0.001227	0.402857	0.543750
2 Nearest Neighbors	0.168098	0.460000	0.571023

[29]:
```
errors_df.sort_values(by='AUC-ROC', axis=0, ascending=False) # sorted by AUC-ROC
```

[29]:

	Train Err	Test Err	AUC-ROC
QDA	0.274847	0.288571	0.654167
LDA	0.273620	0.291429	0.632386
4 Nearest Neighbors	0.226994	0.345714	0.632197
Naive Bayes	0.274847	0.314286	0.628030
Logistic Regression	0.276074	0.294286	0.627841
6 Nearest Neighbors	0.236810	0.342857	0.614583
7 Nearest Neighbors	0.235583	0.334286	0.586364
5 Nearest Neighbors	0.218405	0.334286	0.583902
8 Nearest Neighbors	0.238037	0.362857	0.580303
10 Nearest Neighbors	0.261350	0.357143	0.579545
2 Nearest Neighbors	0.168098	0.460000	0.571023
9 Nearest Neighbors	0.258896	0.342857	0.565341
1 Nearest Neighbor	0.001227	0.402857	0.543750
3 Nearest Neighbors	0.001227	0.402857	0.543750

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, QDA possesses both of these criteria. Therefore, in my opinion, QDA is the best-performing

model in this scenario.

[ ]: