Mingtao_Gao_HW2

February 2, 2020

[1]:
```python
# Python packages required for this assignment

#!/usr/bin/env python
import numpy as np
from statistics import mean
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

from sklearn.model_selection import train_test_split
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis,␣
↪QuadraticDiscriminantAnalysis
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix, classification_report,␣
↪accuracy_score, roc_auc_score, roc_curve
```

[2]:
```python
# Functions required for this assignment

def calculate_Y(x1, x2):
    return x1 + x1**2 + x2 + x2**2

def calculate_Y_2(x1, x2):
    return x1 + x2

def Bayes_classifier(Y_prob):
    return (Y_prob > 0.5)

def classifier_2(y):
    return (y > 0)
```

1 The Bayes Classifier

---

[3]:
```python
# Set up seed
seed = 666
np.random.seed(seed)
```

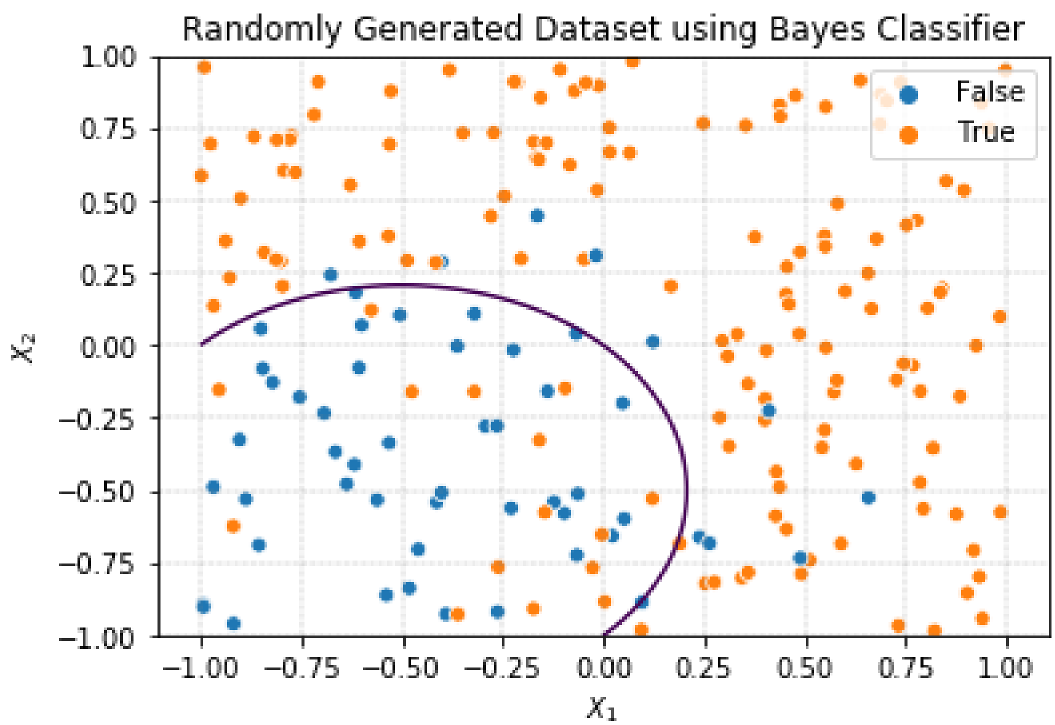
[4]:
```python
# Generate random dataset
X_1 = np.random.uniform(-1, 1, 200)
X_2 = np.random.uniform(-1, 1, 200)
Y = calculate_Y(X_1, X_2) + np.random.normal(0, 0.5, 200)
```

[5]:
```python
# From log-odds, calculate probability from Y
Y_prob = np.exp(Y) / (1 + np.exp(Y))
```

[6]:
```python
# Classification using Bayes classifier
cls = Bayes_classifier(Y_prob)
```

[7]:
```python
# Finding Bayes decision boundary
x_val = np.linspace(-1, 1, 100)
y_val = np.linspace(-1, 1, 100)
Z = []
for x in list(x_val):
    each = []
    for y in list(y_val):
        each.append(calculate_Y(x, y))
    Z.append(each)
Z = np.array(Z)
```

[8]:
```python
# Plot classification results and Bayes decision boundary
sns.scatterplot(X_1, X_2, hue=cls)
plt.contour(x_val, y_val, Z, [0])
plt.grid(color='grey', linestyle='-.', linewidth=0.3)
plt.title('Randomly Generated Dataset using Bayes Classifier')
plt.xlabel('$X_1$')
plt.ylabel('$X_2$')
plt.legend(loc="upper right")
plt.show()
```



The graph above demonstrates how we use Bayes Classifier to classify a randomly generated dataset by calculating and applying the conditional probability. The Bayes Decision Boundary is calculated when Y is equal to 0 as the boundary that divides two classes. This is a simple naive Bayes classifier, which can only be used for binary classification.

2 Exploring Simulated Differences between LDA and QDA

---

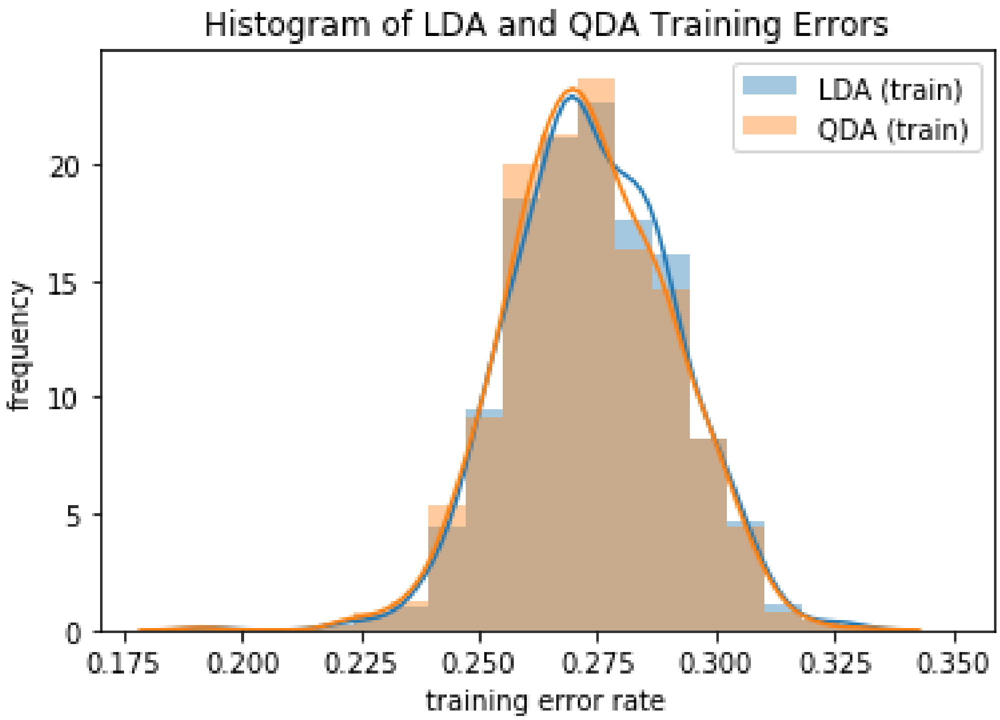
[9]:
```
# The main function used to generate answers for question 2 - 4
# This function randomly generate dataset to train and test LDA and QDA models
# Inputs:
#    N - the number of observations to generate each time
#    func - the function f(X), including linear and non-linear cases
# Outputs:
#    lda_err - a dictionary of LDA training errors and test errors data
#    qda_err - a dictionary of QDA training errors and test errors data

def compare_LDA_QDA(N, func):
    lda = LinearDiscriminantAnalysis()
    qda = QuadraticDiscriminantAnalysis()
    lda_err = {'train_errs': [], 'test_errs': []}
```

```
    qda_err = {'train_errs': [], 'test_errs': []}

    for i in range(1000):
        # Generate and split data into train and test
        X_1 = np.random.uniform(-1, 1, N)
        X_2 = np.random.uniform(-1, 1, N)
        X = np.stack((X_1, X_2), axis=-1)
        y = classifier_2(func(X_1, X_2) + np.random.normal(0, 1, N))
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
↪random_state=seed)

        # Train data using LDA model
        lda.fit(X_train, y_train)
        lda_err['train_errs'].append(1 - accuracy_score(y_train, lda.
↪predict(X_train)))
        lda_err['test_errs'].append(1 - accuracy_score(y_test, lda.
↪predict(X_test)))

        # Train data using QDA model
        qda.fit(X_train, y_train)
        qda_err['train_errs'].append(1 - accuracy_score(y_train, qda.
↪predict(X_train)))
        qda_err['test_errs'].append(1 - accuracy_score(y_test, qda.
↪predict(X_test)))
    return lda_err, qda_err
```

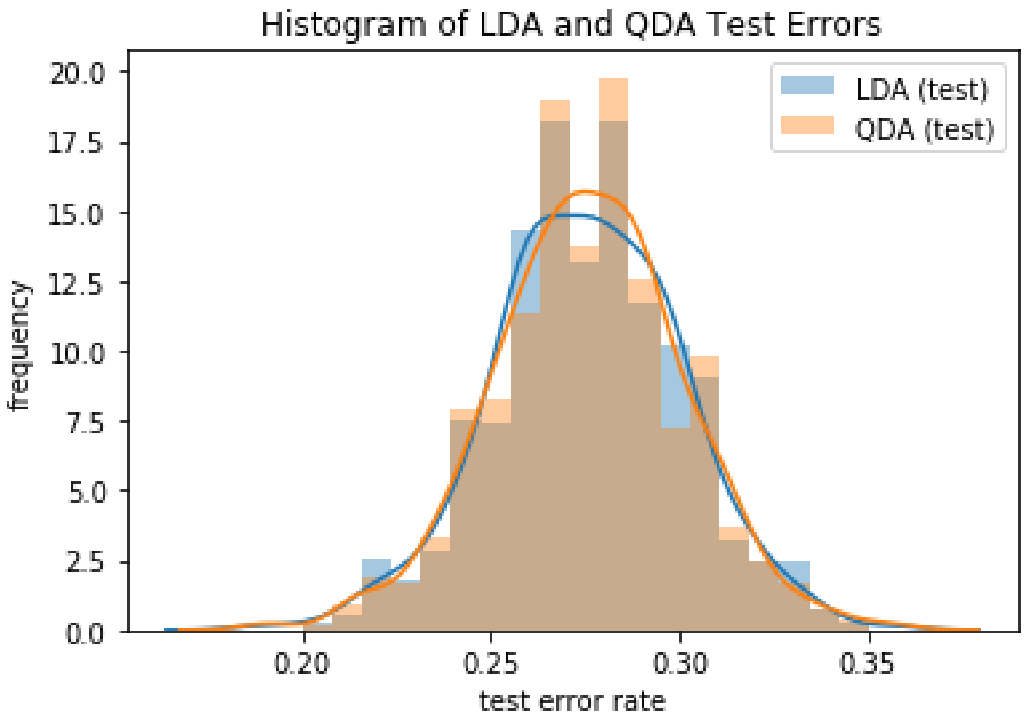
2.1 Linear Bayes Decision Boundary

[10]:
```
# Generate 1000 training and test errors of both model
linear_lda_err, linear_qda_err = compare_LDA_QDA(1000, calculate_Y_2)
```

[11]:
```
# Draw histogram of LDA and QDA training error rates for comparison
bins = np.linspace(0.2, 0.35, 20)
sns.distplot(linear_lda_err['train_errs'], bins, label='LDA (train)')
sns.distplot(linear_qda_err['train_errs'], bins, label='QDA (train)')
plt.legend(loc='upper right')
plt.title('Histogram of LDA and QDA Training Errors')
plt.xlabel('training error rate')
plt.ylabel('frequency')
plt.show()
```



[12]:
```
# Draw histogram of LDA and QDA test error rates for comparison
sns.distplot(linear_lda_err['test_errs'], bins, label='LDA (test)')
sns.distplot(linear_qda_err['test_errs'], bins, label='QDA (test)')
plt.legend(loc='upper right')
plt.title('Histogram of LDA and QDA Test Errors')
plt.xlabel('test error rate')
plt.ylabel('frequency')
plt.show()
```



[13]:
```
pd.DataFrame([[mean(linear_lda_err['train_errs']),␣
 ↪mean(linear_qda_err['train_errs'])],
              [mean(linear_lda_err['test_errs']),␣
 ↪mean(linear_qda_err['test_errs'])]],
             index=['Average Train Error Rate','Average Test Error Rate'],
             columns=['LDA', 'QDA'])
```

[13]:

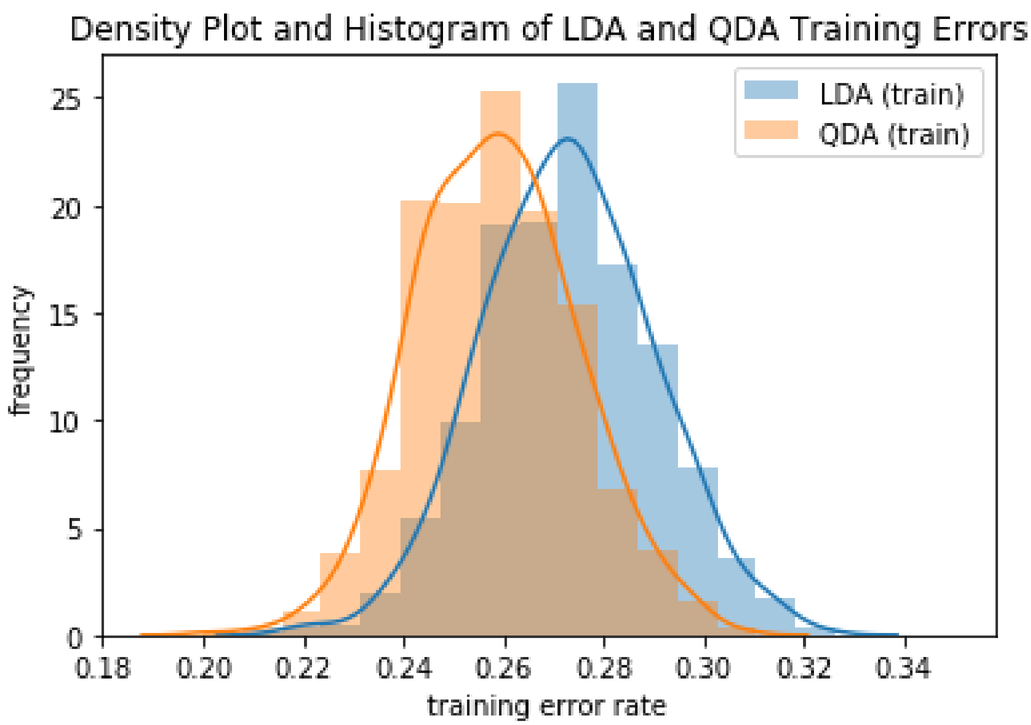
	LDA	QDA
Average Train Error Rate	0.274151	0.273231
Average Test Error Rate	0.275650	0.275843

Since linear discriminant function produces a linear decision boundary and quadratic discriminant function produces a quadratic decision boundary, when the decision boundary is linear, in this case $X_1 + X_2$, we expect LDA to perform better, as its average training error rate is lower than that of QDA. Besides, based on the bias-variance trade-off, QDA has more chance to overfit a training set. However, since LDA in general is less flexible than QDA, we would expect a higher training error with LDA.
In conclusion, with linear decision boundary, QDA performs better on training set and LDA performs better on testing set.

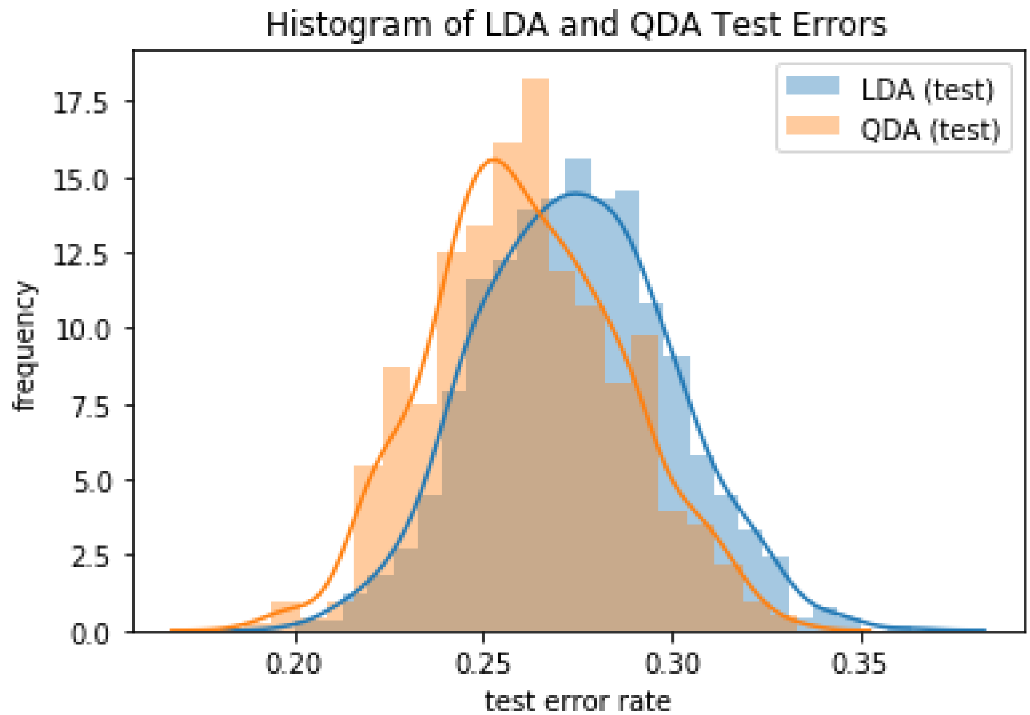
2.2 Non-linear Bayes Decision Boundary

[14]:
```
# Generate 1000 training and test errors of both model
nlinear_lda_err, nlinear_qda_err = compare_LDA_QDA(1000, calculate_Y)
```

[15]:
```
# Draw histogram of LDA and QDA training error rates for comparison
sns.distplot(nlinear_lda_err['train_errs'], bins, label='LDA (train)')
sns.distplot(nlinear_qda_err['train_errs'], bins, label='QDA (train)')
plt.legend(loc='upper right')
plt.title('Density Plot and Histogram of LDA and QDA Training Errors')
plt.xlabel('training error rate')
plt.ylabel('frequency')
plt.show()
```



[16]:
```
# Draw histogram of LDA and QDA test error rates for comparison
sns.distplot(nlinear_lda_err['test_errs'], label='LDA (test)')
sns.distplot(nlinear_qda_err['test_errs'], label='QDA (test)')
plt.legend(loc='upper right')
plt.title('Histogram of LDA and QDA Test Errors')
plt.xlabel('test error rate')
plt.ylabel('frequency')
plt.show()
```



[17]:
```
pd.DataFrame([[mean(nlinear_lda_err['train_errs']),␣
↪mean(nlinear_qda_err['train_errs'])],
              [mean(nlinear_lda_err['test_errs']),␣
↪mean(nlinear_qda_err['test_errs'])]],
             index=['Average Train Error Rate', 'Average Test Error Rate'],
             columns={'LDA', 'QDA'})
```

[17]:

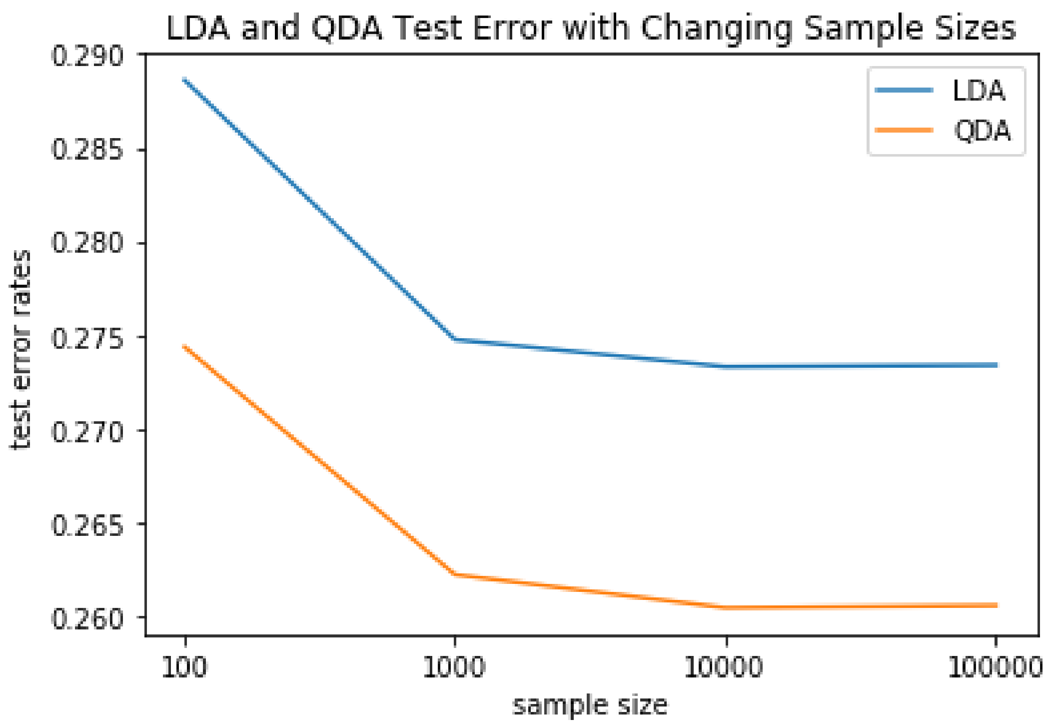
	QDA	LDA
Average Train Error Rate	0.272941	0.258601
Average Test Error Rate	0.274360	0.261320

When the Bayes decision boundary is non-linear, in this case $X_1 + X_1^2 + X_2 + X_2^2$, the QDA is expected to perform better than LDA, with a lower testing error rate. Based on two histograms generated, we can see the QDA error rate distribution is slightly left to LDA's. Since QDA is more flexible than LDA, we also expect a lower training error rate from QDA.
In conclusion, with non-linear decision boundary, QDA performs better on both training and testing sets.

2.3 Non-linear Bayes Decision Boundary with Changing N

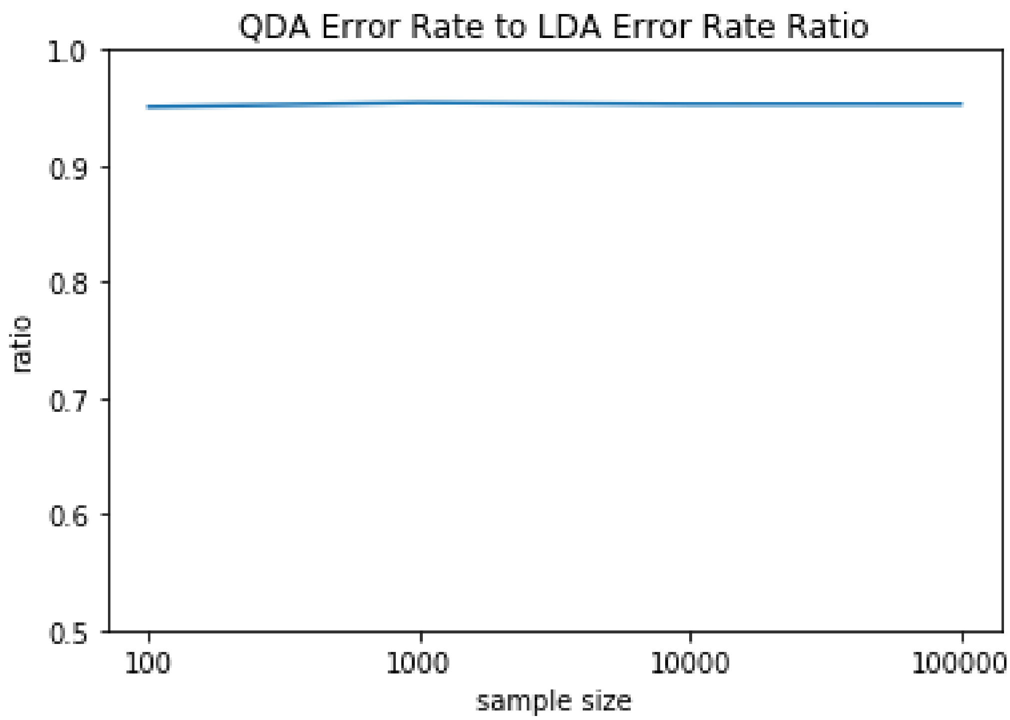
[18]:
```
# Generate different sizes of observations and record average test error rates
N_size = [100, 1000, 10000, 100000]
LDA = []
QDA = []
for N in N_size:
    lda_err, qda_err = compare_LDA_QDA(N, calculate_Y)
    LDA.append(mean(lda_err['test_errs']))
    QDA.append(mean(qda_err['test_errs']))
```

[19]:
```
# Plot test error rates for LDA and QDA with changing N
plt.plot(LDA, label='LDA')
plt.plot(QDA, label='QDA')
plt.xticks(np.arange(4), N_size)
plt.title('LDA and QDA Test Error with Changing Sample Sizes')
plt.xlabel('sample size')
plt.ylabel('test error rates')
plt.legend()
plt.show()
```



[20]:
```
# Plot QDA error rate to LDA error rate
plt.plot((np.array(QDA)/np.array(LDA)))
plt.xticks(np.arange(4), N_size)
```

```
plt.ylim(0.5, 1)
plt.title('QDA Error Rate to LDA Error Rate Ratio')
plt.xlabel('sample size')
plt.ylabel('ratio')
plt.show()
```



As we discussed above, with non-linear Bayes decision boundary, we expect a lower QDA test error rate in general. As sample size N increases, from the above graph, we can see a decrease in test error rates for both LDA and QDA, because a large sample size can reduce variances and thus we observe a huge reduction in both test error rates. As N increases from 10000 to 100000, we observe less reduction and error rates remain unchanged.
In conclusion, as N increases, the test error rate of QDA relative to LDA decreases first and stays unchanged.

3 Modeling Voter Turn-out

---

[21]:
```
# Function used in this question to evaluate a model's performance on test set
# Inputs:
#    model - the trained model object
#    name - string, the name of the model
```

```
#    error_rate - a dictionary that stores the error rate of each model
# Outputs:
#   Test error rate and Receiver Operating Characteristic(ROC) curve with Area␣
 ↪Under Curve(AUC) calculated

def evaluate_model(model, name, performance):
    # Predict test set with model
    y_pred = model.predict(X_test)
    y_pred_prob = model.predict_proba(X_test)[:, 1]

    # Calculate and print the test error rate of the model
    test_err = round(1 - accuracy_score(y_test, y_pred), 4)
    performance[name] = [test_err]

    # Calculate AUC value
    logit_roc_auc = round(roc_auc_score(y_test, y_pred), 4)
    performance[name] += [logit_roc_auc]

    # Draw ROC curve
    fpr, tpr, thresholds = roc_curve(y_test, y_pred_prob)
    plt.plot(fpr, tpr, label='{} (AUC = {})'.format(name, logit_roc_auc))
    plt.plot([0, 1], [0, 1],'r--')
    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 for {} Model'.format(name))
    plt.legend(loc="lower right")
    plt.show()
```

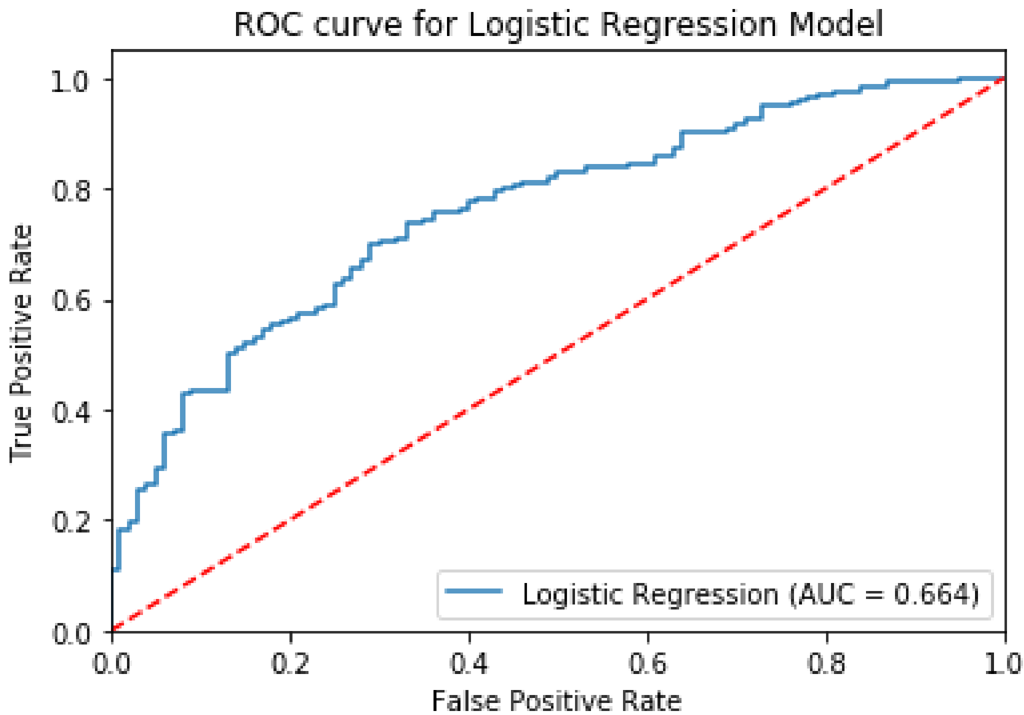
[22]:
```
# Generate and process dataframe
df = pd.read_csv('mental_health.csv')
df.dropna(inplace=True)
```

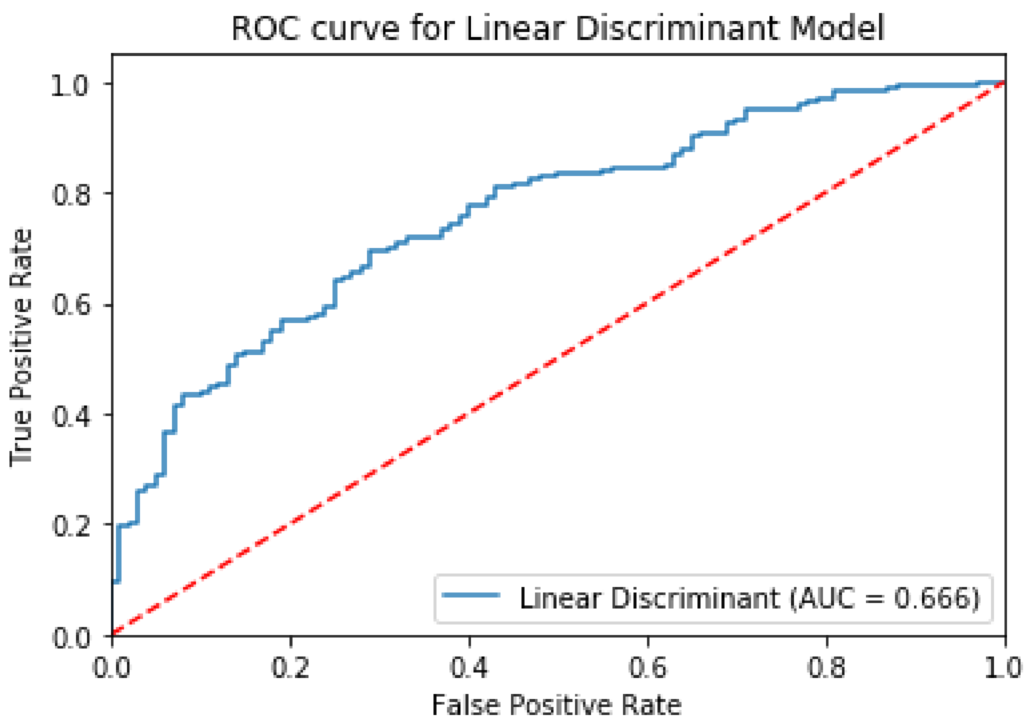
[23]:
```
# Split the data into a training and test set
cols = ['mhealth_sum', 'age', 'educ', 'black', 'female', 'married', 'inc10']
X = df[cols]
y = df['vote96']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
 ↪random_state=seed)
```

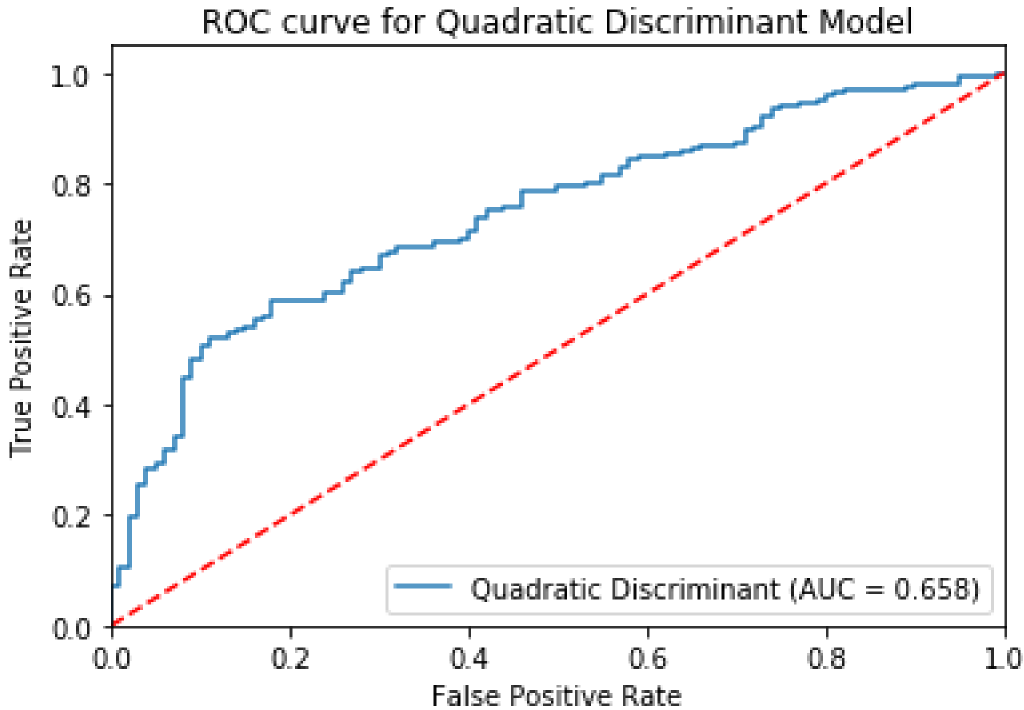
[24]:
```
# Train the dataset with different models
# Logistic regression model
logreg = LogisticRegression()
logreg.fit(X_train, y_train)

# Linear discriminant model
```

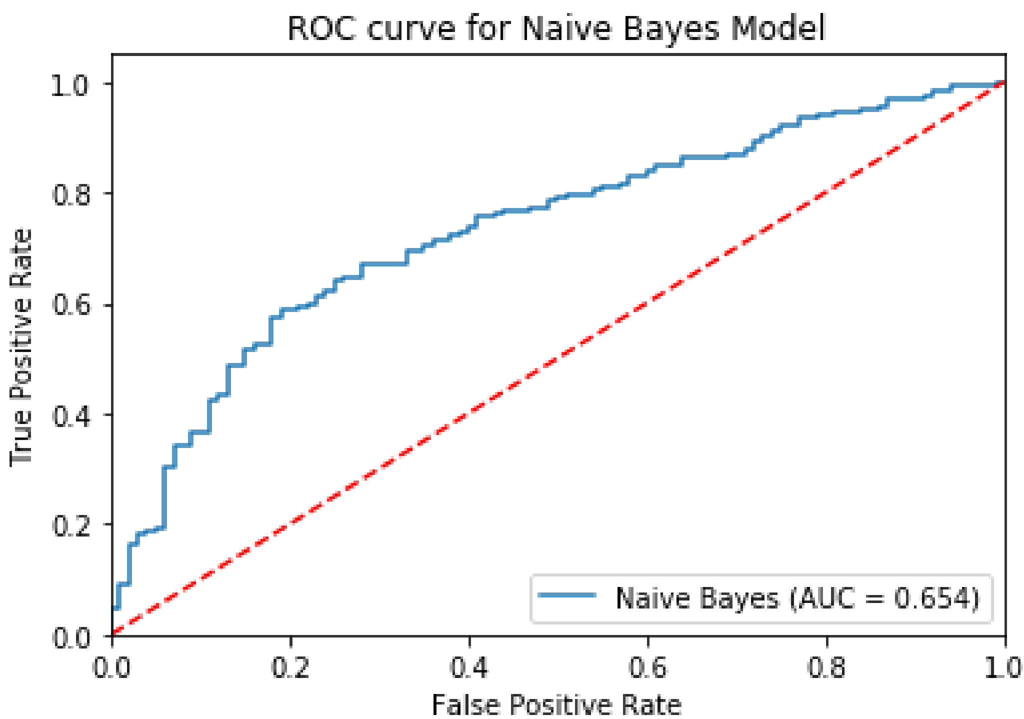
```
lda = LinearDiscriminantAnalysis()
lda.fit(X_train, y_train)

# Quadratic discriminant model
qda = QuadraticDiscriminantAnalysis()
qda.fit(X_train, y_train)

# Gaussian Naive Bayes model
gnb = GaussianNB()
gnb.fit(X_train, y_train)

# K-nearest neighbors model
knns = []
k_range = range(1, 11)
for k in k_range:
    knn = KNeighborsClassifier(n_neighbors=k, metric='euclidean')
    knns.append(knn.fit(X_train, y_train))
```

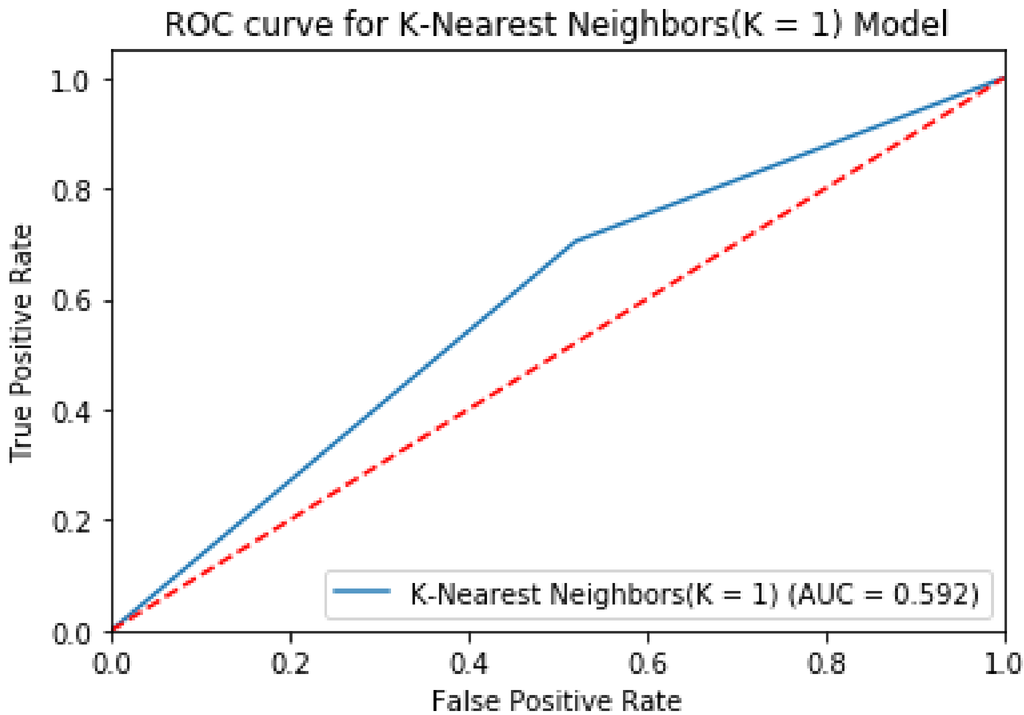
[25]:
```
# Evaluate each model
performance = {}

# Logistic regression model
evaluate_model(logreg, 'Logistic Regression', performance)

# Linear discriminant model
evaluate_model(lda, 'Linear Discriminant', performance)

# Quadratic discriminant model
evaluate_model(qda, 'Quadratic Discriminant', performance)

# Gaussian Naive Bayes model
evaluate_model(gnb, 'Naive Bayes', performance)

# K-nearest neighbors model
for k in range(len(knns)):
    evaluate_model(knns[k], 'K-Nearest Neighbors(K = {})'.format(k + 1),␣
 ↪performance)
```

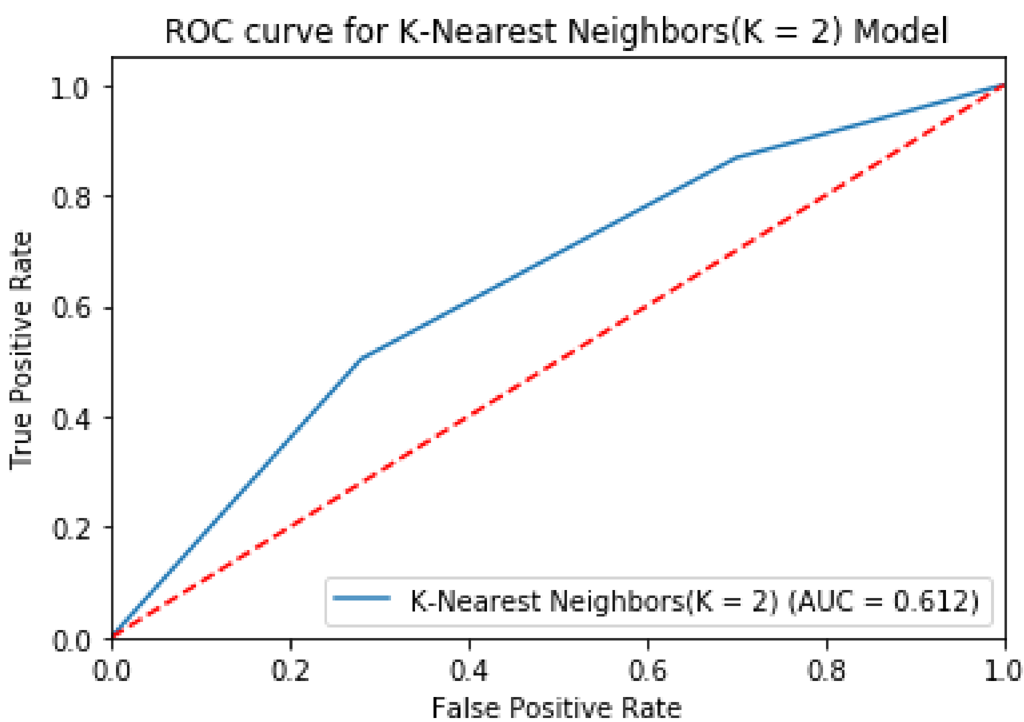

ROC curve for Logistic Regression Model
1.0
0.8
0.6
0.4
0.2
0.0
True Positive Rate
Logistic Regression (AUC = 0.664)
0.0
0.2
0.4
0.6
0.8
1.0
False Positive Rate

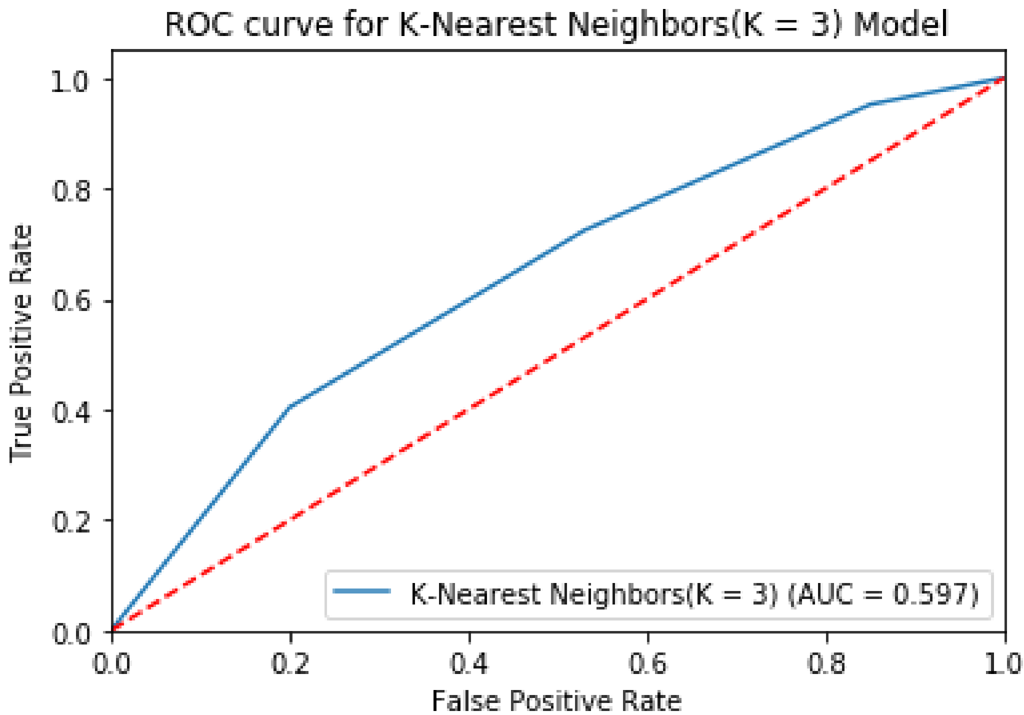

ROC curve for Linear Discriminant Model
1.0
0.8
0.6
0.4
0.2
0.0
True Positive Rate
Linear Discriminant (AUC = 0.666)
0.0
0.2
0.4
0.6
0.8
1.0
False Positive Rate

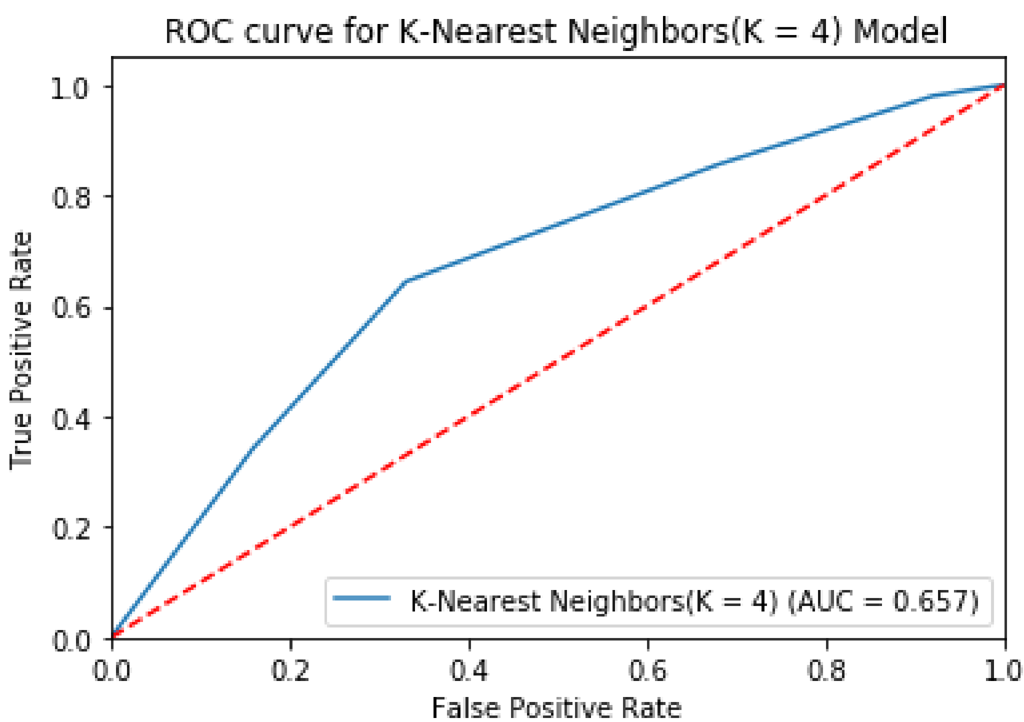

ROC curve for Quadratic Discriminant Model
1.0
0.8
0.6
0.4
0.2
0.0
True Positive Rate
Quadratic Discriminant (AUC = 0.658)
0.0
0.2
0.4
0.6
0.8
1.0
False Positive Rate

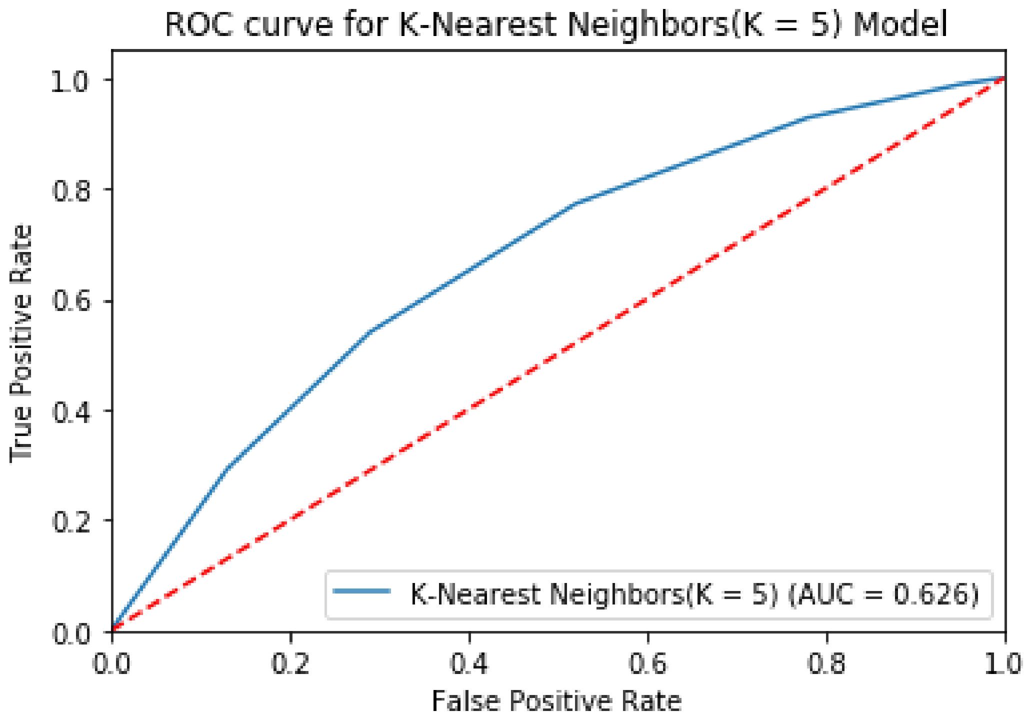

ROC curve for Naive Bayes Model
1.0
0.8
0.6
0.4
0.2
0.0
True Positive Rate
Naive Bayes (AUC = 0.654)
0.0
0.2
0.4
0.6
0.8
1.0
False Positive Rate

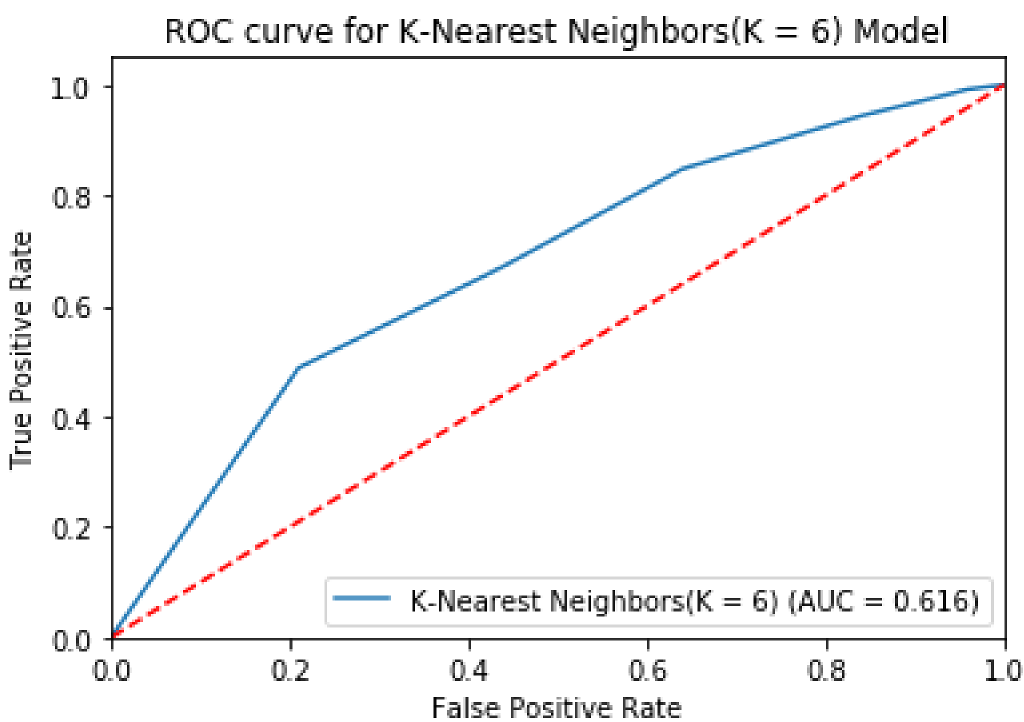

ROC curve for K-Nearest Neighbors(K = 1) Model
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
K-Nearest Neighbors(K = 1) (AUC = 0.592)

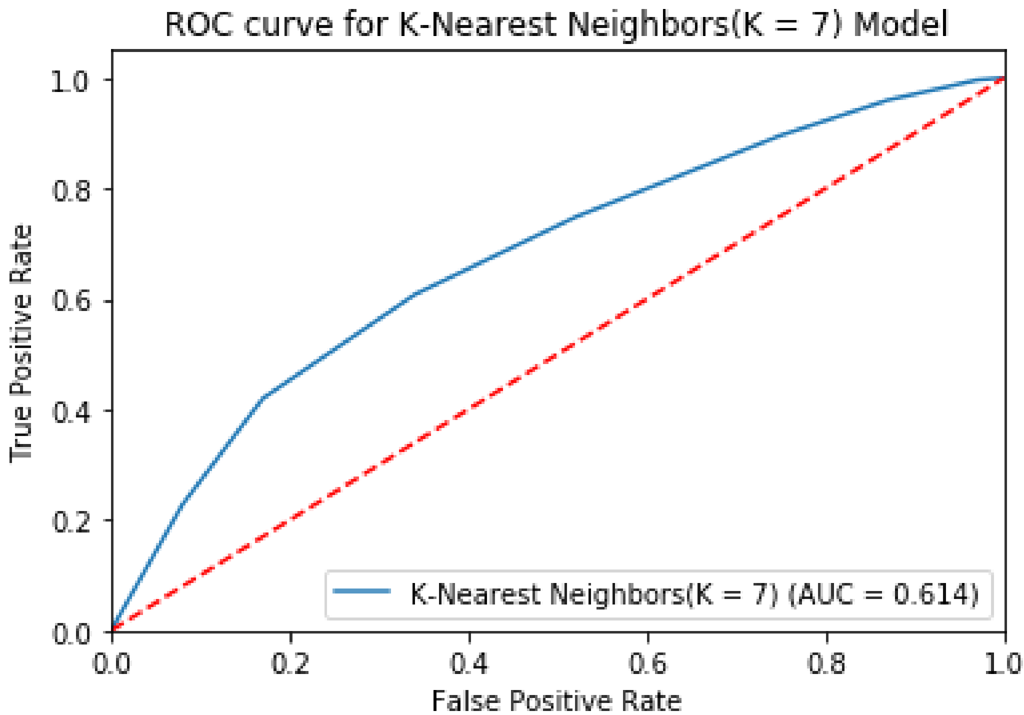

ROC curve for K-Nearest Neighbors(K = 2) Model
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
K-Nearest Neighbors(K = 2) (AUC = 0.612)

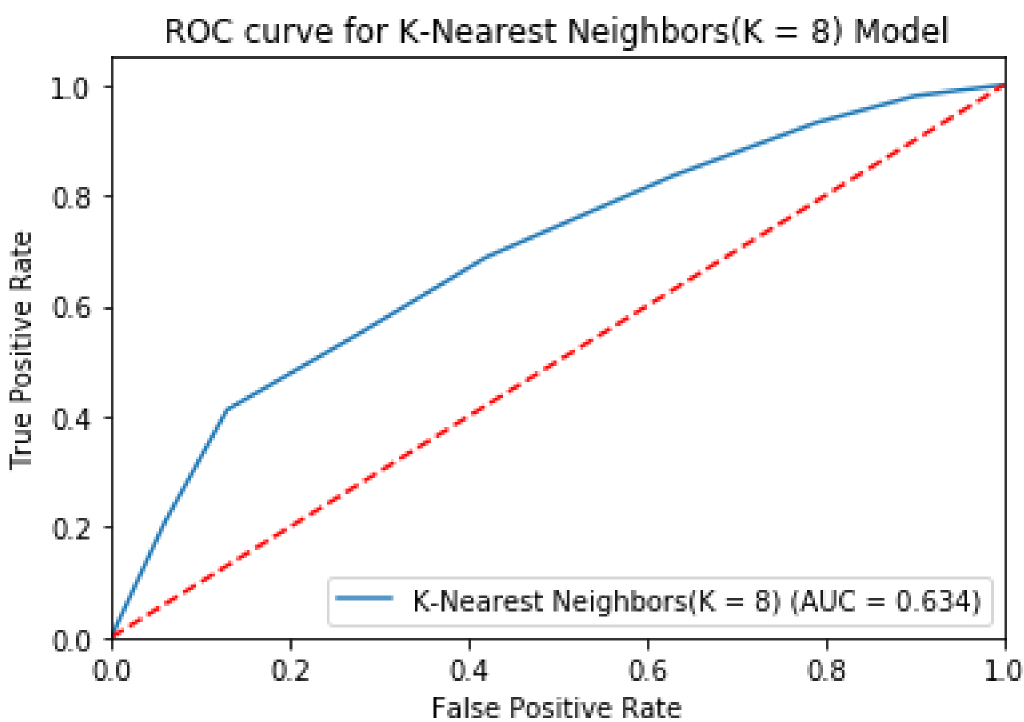

ROC curve for K-Nearest Neighbors(K = 3) Model
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
K-Nearest Neighbors(K = 3) (AUC = 0.597)

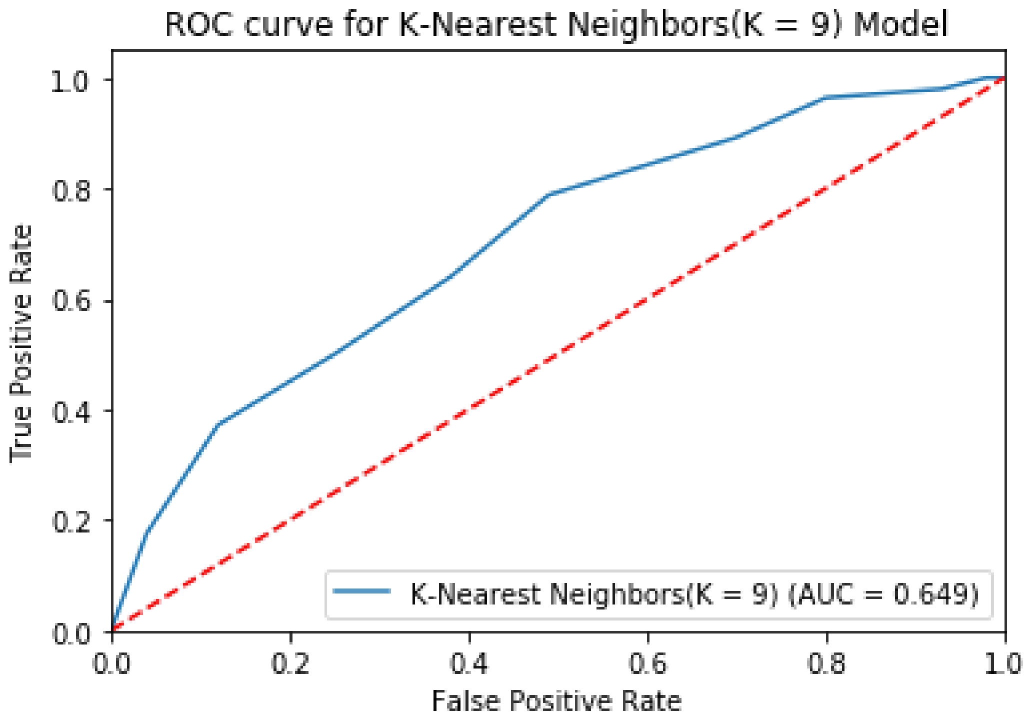

ROC curve for K-Nearest Neighbors(K = 4) Model
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
K-Nearest Neighbors(K = 4) (AUC = 0.657)

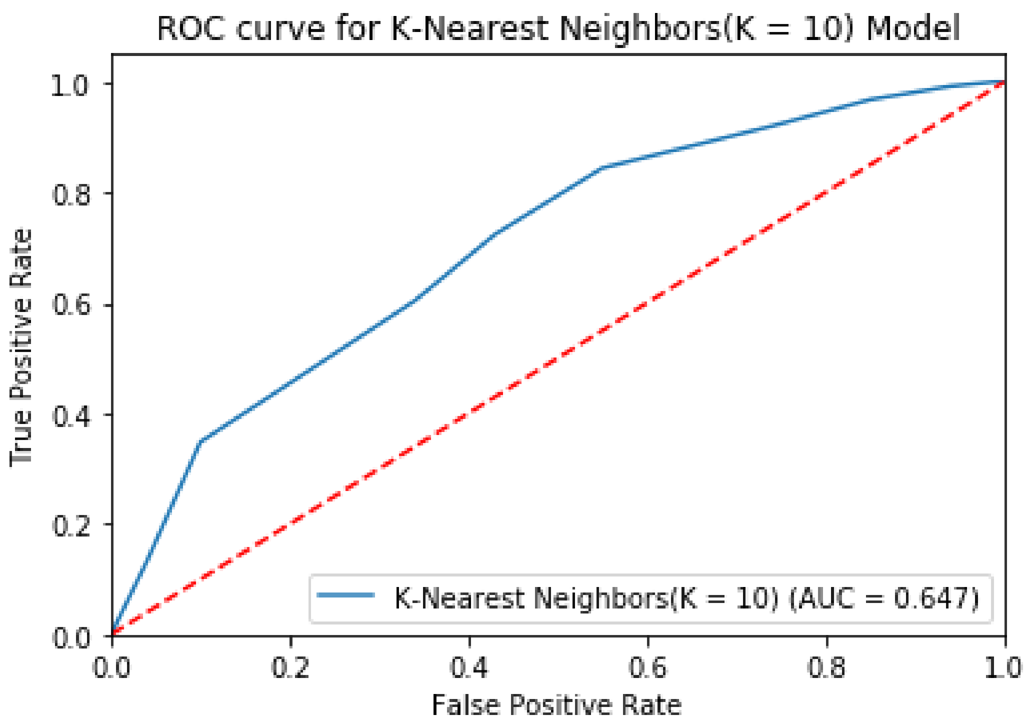

ROC curve for K-Nearest Neighbors(K = 5) Model
True Positive Rate
False Positive Rate
K-Nearest Neighbors(K = 5) (AUC = 0.626)
0.0
0.2
0.4
0.6
0.8
1.0


ROC curve for K-Nearest Neighbors(K = 6) Model
True Positive Rate
False Positive Rate
K-Nearest Neighbors(K = 6) (AUC = 0.616)
0.0
0.2
0.4
0.6
0.8
1.0


ROC curve for K-Nearest Neighbors(K = 7) Model
True Positive Rate
False Positive Rate
K-Nearest Neighbors(K = 7) (AUC = 0.614)
0.0
0.2
0.4
0.6
0.8
1.0


ROC curve for K-Nearest Neighbors(K = 8) Model
True Positive Rate
False Positive Rate
K-Nearest Neighbors(K = 8) (AUC = 0.634)
0.0
0.2
0.4
0.6
0.8
1.0


ROC curve for K-Nearest Neighbors(K = 9) Model
1.0
0.8
0.6
0.4
0.2
0.0
True Positive Rate
K-Nearest Neighbors(K = 9) (AUC = 0.649)
0.0
0.2
0.4
0.6
0.8
1.0
False Positive Rate


ROC curve for K-Nearest Neighbors(K = 10) Model
1.0
0.8
0.6
0.4
0.2
0.0
True Positive Rate
K-Nearest Neighbors(K = 10) (AUC = 0.647)
0.0
0.2
0.4
0.6
0.8
1.0
False Positive Rate

[26]:
```
# Generate a table with each model's error rate and AUC value
pd.DataFrame(performance.values(), index=performance.keys(), columns=['Test␣
↪Error Rate', 'AUC'])
```

[26]:

	Test Error Rate	AUC
Logistic Regression	0.2657	0.664
Linear Discriminant	0.2629	0.666
Quadratic Discriminant	0.3000	0.658
Naive Bayes	0.2971	0.654
K-Nearest Neighbors(K = 1)	0.3600	0.592
K-Nearest Neighbors(K = 2)	0.4343	0.612
K-Nearest Neighbors(K = 3)	0.3486	0.597
K-Nearest Neighbors(K = 4)	0.3486	0.657
K-Nearest Neighbors(K = 5)	0.3114	0.626
K-Nearest Neighbors(K = 6)	0.3600	0.616
K-Nearest Neighbors(K = 7)	0.3286	0.614
K-Nearest Neighbors(K = 8)	0.3429	0.634
K-Nearest Neighbors(K = 9)	0.2914	0.649
K-Nearest Neighbors(K = 10)	0.3200	0.647

According the above ROC curve graphs and the table of each model's test error rate and AUC value, we found that LDA yields the lowest test error rate and highest AUC value.

Thus, based on test error rates and AUC values, Linear Discriminant Model performs the best