Classification Models

Yinjiang Xiong

The Bayes Classifier

1.For classification problems, the test error rate is minimized by a simple classifier that assigns each observation to the most likely class given its predictor values:

$$\Pr(Y = j | X = x_0)$$

where x_0 is the test observation and each possible class is represented by J. This is a conditional probability that $Y = j$, given the observed predictor vector x_0. This classifier is known as the Bayes classifier. If the response variable is binary (i.e. two classes), the Bayes classifier corresponds to predicting class one if $\Pr(Y = 1|X = x_0) > 0.5$, and class two otherwise.

a. Set your random number generator seed.

In [25]:
```
import random
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sb
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis as QDA
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score, roc_curve, auc
from statistics import mean
from prettytable import PrettyTable
np.random.seed(123)
```

b. Simulate a dataset of $N = 200$ with X_1, X_2 where X_1, X_2 are random uniform variables between $[-1, 1]$

In [10]:
```
x1 = np.random.uniform(-1,1,200)
x2 = np.random.uniform(-1,1,200)
```

c. Calculate $Y = X_1 + X_1^2 + X_2 + X_2^2 + \epsilon$, where $\epsilon \sim N(\mu = 0, \sigma^2 = 0.25)$.

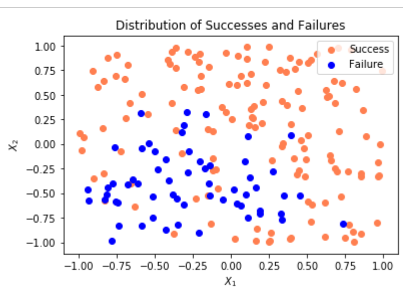
In [11]:
```
# generate error term
error = np.random.normal(0,0.5,200)
y = x1 + x1 * x1 + x2 + x2 * x2 + error
```

d. Y is defined in terms of the log-odds of success on the domain $[-\infty, +\infty]$. Calculate the probability of success bounded between $[0, 1]$.

In [12]:
```
prob = np.exp(y) / (1 + np.exp(y))
```

e. Plot each of the data points on a graph and use color to indicate if the observation was a success or a failure.

In [13]:
```
success = y >= 0
plt.scatter(x1[success],x2[success], color='coral')
plt.scatter(x1[~success],x2[~success], color='blue')
plt.xlabel('$X_1$')
plt.ylabel('$X_2$')
plt.legend(['Success','Failure'], loc=1)
plt.title('Distribution of Successes and Failures');
```



f. Overlay the plot with Bayes decision boundary, calculated using X_1, X_2

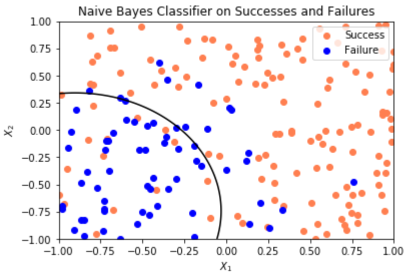
In [14]:
```
clf  = GaussianNB()
X = np.stack(((x1), (x2)), axis=1)
clf.fit(X, success)
```

Out[14]:
```
GaussianNB(priors=None, var_smoothing=1e-09)
```

In [136]:
```
xlim = (-1, 1)
ylim = (-1, 1)
xx, yy = np.meshgrid(np.linspace(xlim[0], xlim[1], 100),
                     np.linspace(ylim[0], ylim[1], 100))
Z = clf.predict_proba(np.c_[xx.ravel(), yy.ravel()])
Z = Z[:, 1].reshape(xx.shape)
```

g. Give your plot a meaningful title and axis labels.

In [135]:
```
fig = plt.figure()
ax = fig.add_subplot(111)
ax.scatter(x1[success],x2[success], color='coral')
ax.scatter(x1[~success],x2[~success], color='blue')
ax.contour(xx, yy, Z, [0.5], colors='k')
ax.set_xlim(xlim)
ax.set_ylim(ylim)
plt.xlabel('$X_1$')
plt.ylabel('$X_2$')
plt.legend(['Success','Failure'], loc=1)
plt.title('Naive Bayes Classifier on Successes and Failures')

plt.show()
```



Exploring Simulated Differences between LDA and QDA

2.If the Bayes decision boundary is linear, do we expect LDA or QDA to perform better on the training set? On the test set?

Answer: QDA is expected to perform better on the training set for its higher flexibility, and LDA is expected to perform better on the test set because the decision boundary is linear

a. Repeat the following process 1000 times.

i. Simulate a dataset of 1000 observations with $X_1, X_2 \sim \text{Uniform}(-1, +1)$. Y is a binary response variable defined by a Bayes decision boundary of $f(X) = X_1 + X_2$, where values 0 or greater are coded TRUE and values less than 0 or coded FALSE. Whereas your simulated Y is a function of $X_1 + X_2 + \epsilon$ where $\epsilon \sim N(0, 1)$. That is, your simulated Y is a function of the Bayes decision boundary plus some irreducible error.

In [15]:
```
x1 = np.random.uniform(-1,1,1000)
x2 = np.random.uniform(-1,1,1000)
error = error = np.random.normal(0,1,1000)
Y = x1 + x2 + error
```

ii. Randomly split your dataset into 70/30% training/test sets.

In [16]:
```
X = np.stack(((x1), (x2)), axis=1)
y = Y >= 0
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)
# check split data
X_train.shape
```

Out[16]:
```
(700, 2)
```

iii. Use the training dataset to estimate LDA and QDA models.

In [17]:
```
clf_l = LDA()
clf_q = QDA()
clf_l.fit(X_train, y_train)
clf_q.fit(X_train, y_train)
```

Out[17]:
```
QuadraticDiscriminantAnalysis(priors=None, reg_param=0.0,
                              store_covariance=False, tol=0.0001)
```

iv. Calculate each model's training and test error rate.

In [302]:
```
print('LDA training error rate = ', 1 - clf_l.score(X_train, y_train))
print('QDA training error rate = ', 1 - clf_q.score(X_train, y_train))
print('LDA test error rate = ', 1 - clf_l.score(X_test, y_test))
print('QDA test error rate = ', 1 - clf_q.score(X_test, y_test))
```

```
LDA training error rate =  0.23857142857142855
QDA training error rate =  0.23142857142857143
LDA test error rate =  0.31999999999999995
QDA test error rate =  0.32666666666666666
```

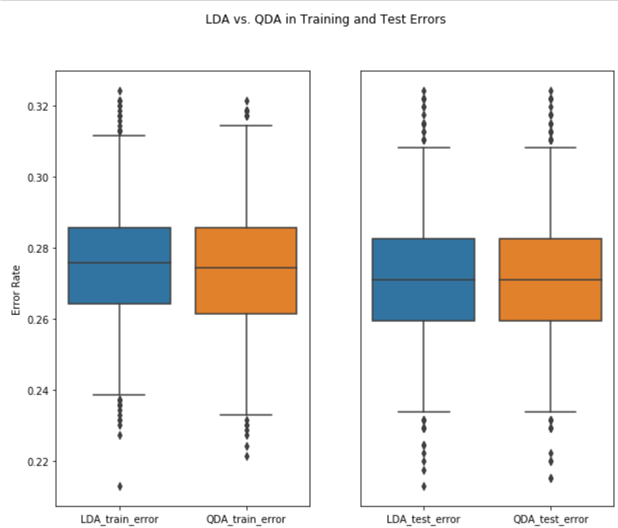
b. Summarize all the simulations' error rates and report the results in tabular and graphical form. Use this evidence to support your answer.

In [304]:
```
# create empty lists
LDA_train_error = []
QDA_train_error = []
LDA_test_error = []
QDA_test_error = []
# loop for 1000 times
for n in range(1000):
    x1 = np.random.uniform(-1,1,1000)
    x2 = np.random.uniform(-1,1,1000)
    error = error = np.random.normal(0,1,1000)
    Y = x1 + x2 + error
    X = np.stack(((x1), (x2)), axis=1)
    y = Y >= 0
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)
    clf_l.fit(X_train, y_train)
    clf_q.fit(X_train, y_train)
    LDA_train_error.append(1 - clf_l.score(X_train, y_train))
    QDA_train_error.append(1 - clf_q.score(X_train, y_train))
    LDA_test_error.append(1 - clf_l.score(X_test, y_test))
    QDA_test_error.append(1 - clf_q.score(X_test, y_test))
```

In [305]:
```
# tabular plot
t = PrettyTable(['','LDA_train_error', 'QDA_train_error', 'LDA_test_error', 'QDA_test_error'])
t.add_row(['Mean', mean(LDA_train_error), mean(QDA_train_error), mean(LDA_test_error), mean(QDA_test_error)])
t.add_row(['Max', max(LDA_train_error), max(QDA_train_error), max(LDA_test_error), max(QDA_test_error)])
t.add_row(['Min', min(LDA_train_error), min(QDA_train_error), min(LDA_test_error), min(QDA_test_error)])
print(t)
```

```
+------+---------------------+---------------------+---------------------+---------------------+
|      |   LDA_train_error   |   QDA_train_error   |    LDA_test_error   |    QDA_test_error   |
+------+---------------------+---------------------+---------------------+---------------------+
| Mean |  0.2749142857142857 |  0.2739614285714286 | 0.27712333333333333 |  0.2777866666666667 |
| Max  |  0.3242857142857143 |  0.3214285714285714 |  0.3533333333333334 |  0.3533333333333334 |
| Min  | 0.21285714285714286 | 0.22142857142857142 | 0.19333333333333336 | 0.19666666666666666 |
+------+---------------------+---------------------+---------------------+---------------------+
```

In [306]:
```
# convert the data into dataframes
train_error = list(zip(LDA_train_error,QDA_train_error))
test_error = list(zip(LDA_test_error,QDA_test_error))
df_train = pd.DataFrame(train_error, columns=['LDA_train_error', 'QDA_train_error'])
df_test = pd.DataFrame(test_error, columns=['LDA_test_error', 'QDA_test_error'])
# graphical plot
fig= plt.figure(figsize=(10,8))
fig.suptitle('LDA vs. QDA in Training and Test Errors')
ax1 = plt.subplot(121)
sb.boxplot(data=df_train, whis=1.2)
plt.ylabel('Error Rate')
ax2 = plt.subplot(122)
sb.boxplot(data=df_test, whis=1.2)
ax2.set_yticks(());
```



In summary, QDA has a slightly lower error rate in training set and a slightly higher error rate in test set than LDA, as predicted.

3.If the Bayes decision boundary is non-linear, do we expect LDA or QDA to perform better on the training set? On the test set?

Answer: In this case, QDA should produce less errors in both training and test sets, since the Bayes decision boundary is not linear.

a. Repeat the following process 1000 times.

i. Simulate a dataset of 1000 observations with $X_1, X_2 \sim \text{Uniform}(-1, +1)$. Y is a binary response variable defined by a Bayes decision boundary of $f(X) = X_1 + X_1^2 + X_2 + X_2^2$, where values 0 or greater are coded TRUE and values less than 0 or coded FALSE. Whereas your simulated Y is a function of $X_1 + X_1^2 + X_2 + X_2^2 + \epsilon$ where $\epsilon \sim N(0, 1)$. That is, your simulated Y is a function of the Bayes decision boundary plus some irreducible error.

In [289]:
```
x1 = np.random.uniform(-1,1,1000)
x2 = np.random.uniform(-1,1,1000)
error = error = np.random.normal(0,1,1000)
Y = x1 + x1 * x1 + x2 + x2 * x2 + error
```

ii. Randomly split your dataset into 70/30% training/test sets.

In [290]:
```
X = np.stack(((x1), (x2)), axis=1)
y = Y >= 0
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)
# check split data
X_train.shape
```

Out[290]:
```
(700, 2)
```

iii. Use the training dataset to estimate LDA and QDA models.

In [291]:
```
clf_l = LDA()
clf_q = QDA()
clf_l.fit(X_train, y_train)
clf_q.fit(X_train, y_train)
```

Out[291]:
```
QuadraticDiscriminantAnalysis(priors=None, reg_param=0.0,
                              store_covariance=False, tol=0.0001)
```

iv. Calculate each model's training and test error rate.

In [292]:
```
print('LDA training error rate = ', 1 - clf_l.score(X_train, y_train))
print('QDA training error rate = ', 1 - clf_q.score(X_train, y_train))
print('LDA test error rate = ', 1 - clf_l.score(X_test, y_test))
print('QDA test error rate = ', 1 - clf_q.score(X_test, y_test))
```

```
LDA training error rate =  0.27857142857142886
QDA training error rate =  0.25857142857142856
LDA test error rate =  0.2566666666666667
QDA test error rate =  0.2533333333333333
```

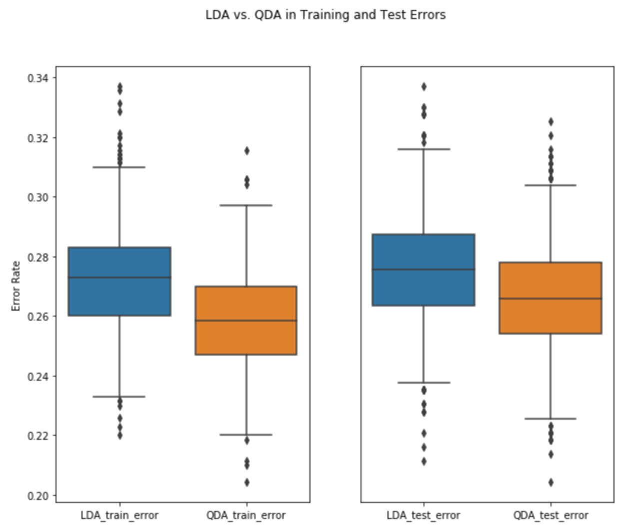
b. Summarize all the simulations' error rates and report the results in tabular and graphical form. Use this evidence to support your answer.

In [296]:
```
# create empty lists
LDA_train_error = []
QDA_train_error = []
LDA_test_error = []
QDA_test_error = []
# loop for 1000 times
for n in range(1000):
    x1 = np.random.uniform(-1,1,1000)
    x2 = np.random.uniform(-1,1,1000)
    error = np.random.normal(0,1,1000)
    Y = x1 + x1 * x1 + x2 + x2 * x2 + error
    X = np.stack(((x1), (x2)), axis=1)
    y = Y >= 0
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)
    clf_l.fit(X_train, y_train)
    clf_q.fit(X_train, y_train)
    LDA_train_error.append(1 - clf_l.score(X_train, y_train))
    QDA_train_error.append(1 - clf_q.score(X_train, y_train))
    LDA_test_error.append(1 - clf_l.score(X_test, y_test))
    QDA_test_error.append(1 - clf_q.score(X_test, y_test))
```

In [297]:
```
t = PrettyTable(['','LDA_train_error', 'QDA_train_error', 'LDA_test_error', 'QDA_test_error'])
t.add_row(['Mean', mean(LDA_train_error), mean(QDA_train_error), mean(LDA_test_error), mean(QDA_test_error)])
t.add_row(['Max', max(LDA_train_error), max(QDA_train_error), max(LDA_test_error), max(QDA_test_error)])
t.add_row(['Min', min(LDA_train_error), min(QDA_train_error), min(LDA_test_error), min(QDA_test_error)])
print(t)
```

```
+------+---------------------+--------------------+---------------------+---------------------+
|      |   LDA_train_error   |  QDA_train_error   |    LDA_test_error   |    QDA_test_error   |
+------+---------------------+--------------------+---------------------+---------------------+
| Mean | 0.27209285714285714 |      0.25878       |       0.27439       |  0.2608933333333333 |
| Max  |  0.3371428571428572 | 0.3157142857142857 |         0.36        |  0.3433333333333334 |
| Min  | 0.21999999999999997 | 0.2042857142857143 | 0.18333333333333335 | 0.17333333333333334 |
+------+---------------------+--------------------+---------------------+---------------------+
```

In [298]:
```
# convert the data into dataframes
train_error = list(zip(LDA_train_error,QDA_train_error))
test_error = list(zip(LDA_test_error,QDA_test_error))
df_train = pd.DataFrame(train_error, columns=['LDA_train_error', 'QDA_train_error'])
df_test = pd.DataFrame(test_error, columns=['LDA_test_error', 'QDA_test_error'])
# graphical plot
fig= plt.figure(figsize=(10,8))
fig.suptitle('LDA vs. QDA in Training and Test Errors')
ax1 = plt.subplot(121)
sb.boxplot(data=df_train, whis=1.2)
plt.ylabel('Error Rate')
ax2 = plt.subplot(122)
sb.boxplot(data=df_test, whis=1.2)
ax2.set_yticks(());
```



In this case, we can see QDA consistently produces less errors in both training and test sets.

4.In general, as sample size n increases, do we expect the test error rate of QDA relative to LDA to improve, decline, or be unchanged? Why?

Answer: As n increases, QDA should perform better at test error because larger data reduce overfitting, and QDA performs better on bias.

a. Use the non-linear Bayes decision boundary approach from part (2) and vary n across your simulations (e.g., simulate 1000 times for n = c(1e02, 1e03, 1e04, 1e05)

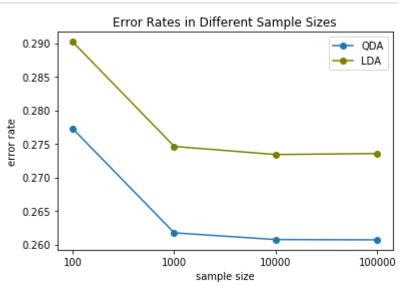
In [314]:
```python
clf_l = LDA()
clf_q = QDA()
LDA_test_error_100 = []
QDA_test_error_100 = []
LDA_test_error_1000 = []
QDA_test_error_1000 = []
LDA_test_error_10000 = []
QDA_test_error_10000 = []
LDA_test_error_100000 = []
QDA_test_error_100000 = []
for n in range(1000):
    x1 = np.random.uniform(-1,1,100)
    x2 = np.random.uniform(-1,1,100)
    error = np.random.normal(0,1,100)
    Y = x1 + x1 * x1 + x2 + x2 * x2 + error
    X = np.stack(((x1), (x2)), axis=1)
    y = Y >= 0
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)
    clf_l.fit(X_train, y_train)
    clf_q.fit(X_train, y_train)
    LDA_test_error_100.append(1 - clf_l.score(X_test, y_test))
    QDA_test_error_100.append(1 - clf_q.score(X_test, y_test))


    x1 = np.random.uniform(-1,1,1000)
    x2 = np.random.uniform(-1,1,1000)
    error = error = np.random.normal(0,1,1000)
    Y = x1 + x1 * x1 + x2 + x2 * x2 + error
    X = np.stack(((x1), (x2)), axis=1)
    y = Y >= 0
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)
    clf_l.fit(X_train, y_train)
    clf_q.fit(X_train, y_train)
    LDA_test_error_1000.append(1 - clf_l.score(X_test, y_test))
    QDA_test_error_1000.append(1 - clf_q.score(X_test, y_test))


    x1 = np.random.uniform(-1,1,10000)
    x2 = np.random.uniform(-1,1,10000)
    error = np.random.normal(0,1,10000)
    Y = x1 + x1 * x1 + x2 + x2 * x2 + error
    X = np.stack(((x1), (x2)), axis=1)
    y = Y >= 0
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)
    clf_l.fit(X_train, y_train)
    clf_q.fit(X_train, y_train)
    LDA_test_error_10000.append(1 - clf_l.score(X_test, y_test))
    QDA_test_error_10000.append(1 - clf_q.score(X_test, y_test))


    x1 = np.random.uniform(-1,1,100000)
    x2 = np.random.uniform(-1,1,100000)
    error = np.random.normal(0,1,100000)
    Y = x1 + x1 * x1 + x2 + x2 * x2 + error
    X = np.stack(((x1), (x2)), axis=1)
    y = Y >= 0
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)
    clf_l.fit(X_train, y_train)
    clf_q.fit(X_train, y_train)
    LDA_test_error_100000.append(1 - clf_l.score(X_test, y_test))
    QDA_test_error_100000.append(1 - clf_q.score(X_test, y_test))
```

In [315]:
```python
print('The ratio between QDA test error rate and LDA test error rate when n = 100 is ',mean(QDA_test_error_100) /
      mean(LDA_test_error_100))
print('The ratio between QDA test error rate and LDA test error rate when n = 1000 is ',mean(QDA_test_error_1000) /
      mean(LDA_test_error_1000))
print('The ratio between QDA test error rate and LDA test error rate when n = 10000 is ',mean(QDA_test_error_10000) /
      mean(LDA_test_error_10000))
print('The ratio between QDA test error rate and LDA test error rate when n = 100000 is ',mean(QDA_test_error_100000) /
      mean(LDA_test_error_100000))
```

```
The ratio between QDA test error rate and LDA test error rate when n = 100 is  0.9554330346887203
The ratio between QDA test error rate and LDA test error rate when n = 1000 is  0.953123103439905
The ratio between QDA test error rate and LDA test error rate when n = 10000 is  0.9536653223250121
The ratio between QDA test error rate and LDA test error rate when n = 100000 is  0.9529864998931388
```

b. Plot the test error rate for the LDA and QDA models as it changes over all of these values of n. Use this graph to support your answer.

In [320]:
```python
QDA_test_error_means = [mean(QDA_test_error_100), mean(QDA_test_error_1000), mean(QDA_test_error_10000),
                        mean(QDA_test_error_100000)]
LDA_test_error_means = [mean(LDA_test_error_100), mean(LDA_test_error_1000), mean(LDA_test_error_10000),
                        mean(LDA_test_error_100000)]
sample_size = ['100', '1000', '10000', '100000']
plt.plot(sample_size, QDA_test_error_means, marker='o', label='QDA')
plt.plot(sample_size, LDA_test_error_means, marker='o', color='olive', label='LDA')
plt.legend()
plt.xlabel('Sample Size')
plt.ylabel('Error Rate')
plt.title('Error Rates in Different Sample Sizes');
```



We can clearly see from the graph that QDA performs better than LDA in non-linear samples. From the ratio report, we can see that the error ratio between QDA and LDA generally decreases as n increases, meaning QDA performs better in larger sample size relative to LDA.

Modeling Voter Turnout

5.Building several classifiers and comparing output.

a. Split the data into a training and test set (70/30).

In [18]:
```
df = pd.read_csv('mental_health.csv')
# load and observe the data
df.head(5)
```

Out[18]:

	vote96	mhealth_sum	age	educ	black	female	married	inc10
0	1.0	0.0	60.0	12.0	0	0	0.0	4.8149
1	1.0	NaN	27.0	17.0	0	1	0.0	1.7387
2	1.0	1.0	36.0	12.0	0	0	1.0	8.8273
3	0.0	7.0	21.0	13.0	0	0	0.0	1.7387
4	0.0	NaN	35.0	16.0	0	1	0.0	4.8149

In [19]:
```
# we notice that there are some missing values
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2832 entries, 0 to 2831
Data columns (total 8 columns):
vote96         2613 non-null float64
mhealth_sum    1414 non-null float64
age            2828 non-null float64
educ           2820 non-null float64
black          2832 non-null int64
female         2832 non-null int64
married        2831 non-null float64
inc10          2503 non-null float64
dtypes: float64(6), int64(2)
memory usage: 177.1 KB
```

In [20]:
```
df_clean = df.dropna()
df_clean.info()
# we have a high attrition rate, but without additional information, dropping NAs is how I'm dealing with missing values
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1165 entries, 0 to 2830
Data columns (total 8 columns):
vote96         1165 non-null float64
mhealth_sum    1165 non-null float64
age            1165 non-null float64
educ           1165 non-null float64
black          1165 non-null int64
female         1165 non-null int64
married        1165 non-null float64
inc10          1165 non-null float64
dtypes: float64(6), int64(2)
memory usage: 81.9 KB
```

In [21]:
```
y = df_clean['vote96']
X = df_clean.drop('vote96', axis=1)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)
```

b. Using the training set and all important predictors, estimate the following models with vote96 as the response variable:
i. Logistic regression model
ii. Linear discriminant model
iii. Quadratic discriminant model
iv. Naive Bayes (you can use the default hyperparameter settings)
v. K-nearest neighbors with $K = 1, 2, \ldots, 10$ (that is, 10 separate models varying K) and Euclidean distance metrics

In [28]:
```
logreg = LogisticRegression()
lda_clf = LDA()
qda_clf = QDA()
nb_clf = GaussianNB()
knn_1 = KNeighborsClassifier(n_neighbors=1, metric='euclidean')
knn_2 = KNeighborsClassifier(n_neighbors=2, metric='euclidean')
knn_3 = KNeighborsClassifier(n_neighbors=3, metric='euclidean')
knn_4 = KNeighborsClassifier(n_neighbors=4, metric='euclidean')
knn_5 = KNeighborsClassifier(n_neighbors=5, metric='euclidean')
knn_6 = KNeighborsClassifier(n_neighbors=6, metric='euclidean')
knn_7 = KNeighborsClassifier(n_neighbors=7, metric='euclidean')
knn_8 = KNeighborsClassifier(n_neighbors=8, metric='euclidean')
knn_9 = KNeighborsClassifier(n_neighbors=9, metric='euclidean')
knn_10 = KNeighborsClassifier(n_neighbors=10, metric='euclidean')
logreg.fit(X_train, y_train)
lda_clf.fit(X_train, y_train)
qda_clf.fit(X_train, y_train)
nb_clf.fit(X_train, y_train)
knn_1.fit(X_train, y_train)
knn_2.fit(X_train, y_train)
knn_3.fit(X_train, y_train)
knn_4.fit(X_train, y_train)
knn_5.fit(X_train, y_train)
knn_6.fit(X_train, y_train)
knn_7.fit(X_train, y_train)
knn_8.fit(X_train, y_train)
knn_9.fit(X_train, y_train)
knn_10.fit(X_train, y_train)
```

Out[28]:
```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='euclidean',
                     metric_params=None, n_jobs=None, n_neighbors=10, p=2,
                     weights='uniform')
```

c. Using the test set, calculate the following model performance metrics: i. Error rate ii. ROC curve(s) / Area under the curve (AUC)

In [23]:
```
# logreg error rate & AUROC
print('Logistic regression has an error rate of', (1 - logreg.score(X_test, y_test)),
    'and an AUC of', roc_auc_score(y_test, logreg.predict(X_test)))
# lda error rate & AUROC
print('LDA has an error rate of', (1 - lda_clf.score(X_test, y_test)),
    'and an AUC of', roc_auc_score(y_test, lda_clf.predict(X_test)))
# qda error rate & AUROC
print('QDA has an error rate of', (1 - qda_clf.score(X_test, y_test)),
    'and an AUC of', roc_auc_score(y_test, qda_clf.predict(X_test)))
# naive bayes error rate & AUROC
print('Naive Bayes has an error rate of', (1 - nb_clf.score(X_test, y_test)),
    'and an AUC of', roc_auc_score(y_test, nb_clf.predict(X_test)))
# knn_1 error rate & AUROC
print('KNN(N=1) has an error rate of', (1 - knn_1.score(X_test, y_test)),
    'and an AUC of', roc_auc_score(y_test, knn_1.predict(X_test)))
# knn_2 error rate & AUROC
print('KNN(N=2) has an error rate of', (1 - knn_2.score(X_test, y_test)),
    'and an AUC of', roc_auc_score(y_test, knn_2.predict(X_test)))
# knn_3 error rate & AUROC
print('KNN(N=3) has an error rate of', (1 - knn_3.score(X_test, y_test)),
    'and an AUC of', roc_auc_score(y_test, knn_3.predict(X_test)))
# knn_4 error rate & AUROC
print('KNN(N=4) has an error rate of', (1 - knn_4.score(X_test, y_test)),
    'and an AUC of', roc_auc_score(y_test, knn_4.predict(X_test)))
# knn_5 error rate & AUROC
print('KNN(N=5) has an error rate of', (1 - knn_5.score(X_test, y_test)),
    'and an AUC of', roc_auc_score(y_test, knn_5.predict(X_test)))
# knn_6 error rate & AUROC
print('KNN(N=6) has an error rate of', (1 - knn_6.score(X_test, y_test)),
    'and an AUC of', roc_auc_score(y_test, knn_6.predict(X_test)))
# knn_7 error rate & AUROC
print('KNN(N=7) has an error rate of', (1 - knn_7.score(X_test, y_test)),
    'and an AUC of', roc_auc_score(y_test, knn_7.predict(X_test)))
# knn_8 error rate & AUROC
print('KNN(N=8) has an error rate of', (1 - knn_8.score(X_test, y_test)),
    'and an AUC of', roc_auc_score(y_test, knn_8.predict(X_test)))
# knn_9 error rate & AUROC
print('KNN(N=9) has an error rate of', (1 - knn_9.score(X_test, y_test)),
    'and an AUC of', roc_auc_score(y_test, knn_9.predict(X_test)))
# knn_10 error rate & AUROC
print('KNN(N=10) has an error rate of', (1 - knn_10.score(X_test, y_test)),
    'and an AUC of', roc_auc_score(y_test, knn_10.predict(X_test)))
```

```
Logistic regression has an error rate of 0.2657142857142857 and an AUC of 0.6186189044120447
LDA has an error rate of 0.27142857142857146 and an AUC of 0.6194944611519281
QDA has an error rate of 0.3028571428571428 and an AUC of 0.6418972933876432
Naive Bayes has an error rate of 0.29428571428571426 and an AUC of 0.6531462941109292
KNN(N=1) has an error rate of 0.37428571428571433 and an AUC of 0.5799802048041418
KNN(N=2) has an error rate of 0.4 and an AUC of 0.6241196847995737
KNN(N=3) has an error rate of 0.34571428571428575 and an AUC of 0.5906772241044578
KNN(N=4) has an error rate of 0.34857142857142853 and an AUC of 0.6388518786402223
KNN(N=5) has an error rate of 0.32571428571428573 and an AUC of 0.6102249800144657
KNN(N=6) has an error rate of 0.31999999999999995 and an AUC of 0.6420114964406716
KNN(N=7) has an error rate of 0.32285714285714284 and an AUC of 0.6022498001446572
KNN(N=8) has an error rate of 0.3314285714285714 and an AUC of 0.6111005367543492
KNN(N=9) has an error rate of 0.3057142857142857 and an AUC of 0.6146979329247402
KNN(N=10) has an error rate of 0.31999999999999995 and an AUC of 0.6168868247744489
```

d. Which model performs the best? Be sure to define what you mean by “best” and identify supporting evidence to support your conclusion(s).

Answer: Judging based on error rate alone, LDA and logistic regression perform the best. AUC indicates that QDA, KNN(N=6) and Naive Bayes perform the best. Overall, the decision of "best" model requires domain knowledge, and the possible candidates are LDA, QDA, logistic regression and Naive Bayes.

In [27]:
```
# plot the roc curve
def get_roc_auc(model, label):
    #calcualte auc score
    model_auc = roc_auc_score(y_test, model.predict(X_test))
    # summarize scores
    print(label+ ': AUC = %.3f' % (model_auc))
    # calculate roc curves
    model_fpr, model_tpr, _ = roc_curve(y_test, model.predict(X_test))
    # plot the roc curve for the model
    plt.plot(model_fpr, model_tpr, marker='.', label=label)
    # axis labels
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    # show the legend
    plt.legend()

def plot_random_classifier():
    plt.figure(figsize=(10,8))
    # generate a random prediction line
    random_probs = [0 for _ in range(len(y_test))]
    # calculate scores
    random_auc = roc_auc_score(y_test, random_probs)
    random_fpr, random_tpr, _ = roc_curve(y_test, random_probs)
    plt.plot(random_fpr, random_tpr, linestyle='--', label='Random Classifier')

plot_random_classifier()

models = [logreg, lda_clf, qda_clf, nb_clf, knn_1, knn_2, knn_3, knn_4 ,knn_5, knn_6, knn_7, knn_8, knn_9, knn_10]
labels = ['Logistic Regression', 'LDA', 'QDA', "Naive Bayes" ,
          'knn1','knn2','knn3','knn4','knn5','knn6','knn7','knn8','knn9','knn10']

count = 0
for model in models:
    get_roc_auc(model, labels[count])
    count+=1
```

```
Logistic Regression: AUC = 0.619
LDA: AUC = 0.619
QDA: AUC = 0.642
Naive Bayes: AUC = 0.653
knn1: AUC = 0.580
knn2: AUC = 0.624
knn3: AUC = 0.591
knn4: AUC = 0.639
knn5: AUC = 0.610
knn6: AUC = 0.642
knn7: AUC = 0.602
knn8: AUC = 0.611
knn9: AUC = 0.615
knn10: AUC = 0.617
```