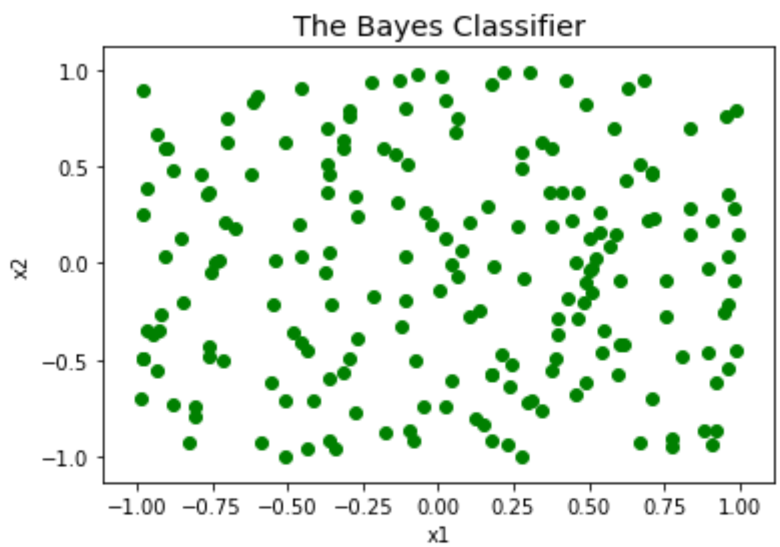
In [98]:
```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

import sklearn
from sklearn.model_selection import train_test_split
```

The Bayes Classifier

1 Produce a graph illustrating this concept. :

e. Plot each of the data points on a graph and use color to indicate if the observation was a success or a failure. f. Overlay the plot with Bayes decision boundary, calculated using X1,X2. g. Give your plot a meaningful title and axis labels. h. The colored background grid is optional.

In [62]:
```
import random
import math

n = 200
dataset = []
random.seed(1227)
for i in range(200):
    x1 = random.uniform(-1, 1)
    x2 = random.uniform(-1, 1)
    err = 0.5 * np.random.randn()
    y = x1 + x1 ** 2 + x2 + x2 ** 2 + err
    p = math.exp(y) / (1 + math.exp(y))
    if p < 0.5:
        r = 0
    else:
        r = 1
    dataset.append([x1, x2, p])


for point in dataset:
    if p <= 0.5:
        plt.scatter(point[0], point[1], color='blue', label='failure')
    else:
        plt.scatter(point[0], point[1], color='green', label='sueccess')


plt.style.use('ggplot')
plt.xlabel('x1')
plt.ylabel('x2');
plt.title('The Bayes Classifier')
plt.show();
```



Exploring Simulated Differences between LDA and QDA

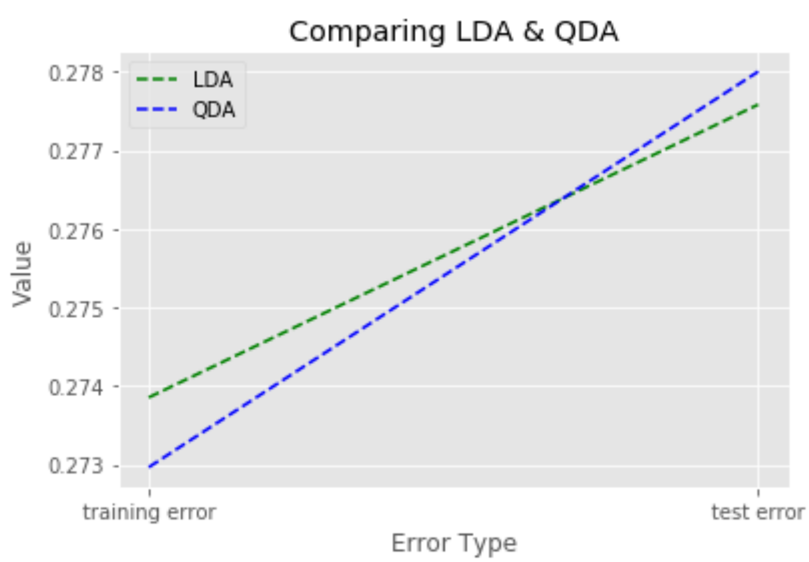
2

In [116]:
```
def simulate_linear_dataset(seed):
    random.seed(seed)
    data = []
    for i in range(1000):
        x1 = random.uniform(-1, 1)
        x2 = random.uniform(-1, 1)
        err = np.random.randn()
        if (x1 + x2 + err) >= 0:
            y = 1
        else:
            y = 0
        data.append([x1, x2, y])

    return np.array(data)
```

In [92]:
```
def LDA_fit(df):
    train, test = train_test_split(df, test_size=0.3, train_size=0.7, ra
ndom_state=1227)
    X_train = train[:, 0:2]
    y_train = train[:, 2]
    X_test = test[:, 0:2]
    y_test = test[:, 2]

    clf = LinearDiscriminantAnalysis()
    clf.fit(X_train, y_train)
    train_err = 1 - clf.score(X_train, y_train)
    test_err = 1 - clf.score(X_test, y_test)

    return train_err, test_err

def QDA_fit(df):
    train, test = train_test_split(df, test_size=0.3, train_size=0.7, ra
ndom_state=1227)
    X_train = train[:, 0:2]
    y_train = train[:, 2]
    X_test = test[:, 0:2]
    y_test = test[:, 2]

    clf = QuadraticDiscriminantAnalysis()
    clf.fit(X_train, y_train)
    train_err = 1 - clf.score(X_train, y_train)
    test_err = 1 - clf.score(X_test, y_test)

    return train_err, test_err
```

In [109]:
```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis

seed = 1227
lda_train_err = 0; lda_test_err = 0; qda_train_err = 0; qda_test_err = 0
for i in range(1000):
    data = simulate_linear_dataset(seed)
    seed += 1
    l1, l2 = LDA_fit(data)
    q1, q2 = QDA_fit(data)
    lda_train_err += l1 / 1000
    lda_test_err += l2 / 1000
    qda_train_err += q1 / 1000
    qda_test_err += q2 / 1000

print('              ', 'LDA  ', 'QDA')
print('training error', "%.4f" %lda_train_err, "%.4f" %qda_train_err)
print('  test error  ', "%.4f" %lda_test_err, "%.4f" %qda_test_err)

p1=plt.plot(['training error','test error'], [lda_train_err, lda_test_er
r], 'g--', label='LDA')
p2=plt.plot(['training error','test error'], [qda_train_err, qda_test_er
r], 'b--', label='QDA')

plt.xlabel('Error Type')
plt.ylabel('Value');
plt.title('Comparing LDA & QDA')
plt.legend()
plt.show();
```

```
               LDA   QDA
training error 0.2739 0.2730
  test error   0.2776 0.2780
```



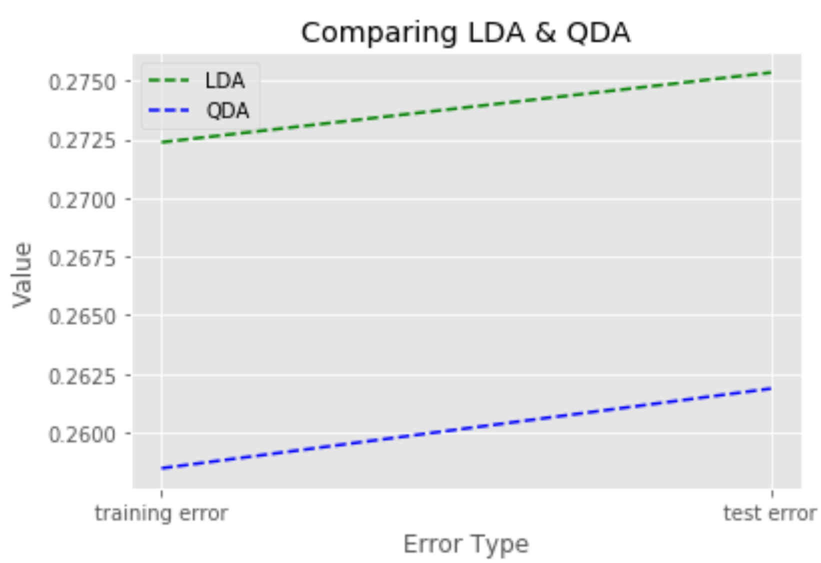
As depicted above, LDA has lower test error and QDA has lower training error. This may be expained by the fact that QDA is more flexible and may risk overfitting. Since we focus more on minimizing test error, LDA has better performance.

3

In [117]:
```
def simulate_nonlinear_dataset(seed, n):
    random.seed(seed)
    data = []
    for i in range(n):
        x1 = random.uniform(-1, 1)
        x2 = random.uniform(-1, 1)
        err = np.random.randn()
        if (x1 + x1**2 + x2 + x2**2 + err) >= 0:
            y = 1
        else:
            y = 0
        data.append([x1, x2, y])

    return np.array(data)
```

In [111]:
```
seed = 5872
lda_train_err = 0; lda_test_err = 0; qda_train_err = 0; qda_test_err = 0
for i in range(1000):
    data = simulate_nonlinear_dataset(seed)
    seed += 1
    l1, l2 = LDA_fit(data)
    q1, q2 = QDA_fit(data)
    lda_train_err += l1 / 1000
    lda_test_err += l2 / 1000
    qda_train_err += q1 / 1000
    qda_test_err += q2 / 1000

print('              ', 'LDA  ', 'QDA')
print('training error', "%.4f" %lda_train_err, "%.4f" %qda_train_err)
print('  test error  ', "%.4f" %lda_test_err, "%.4f" %qda_test_err)

p1=plt.plot(['training error','test error'], [lda_train_err, lda_test_er
r], 'g--', label='LDA')
p2=plt.plot(['training error','test error'], [qda_train_err, qda_test_er
r], 'b--', label='QDA')

plt.xlabel('Error Type')
plt.ylabel('Value');
plt.title('Comparing LDA & QDA')
plt.legend()
plt.show();
```

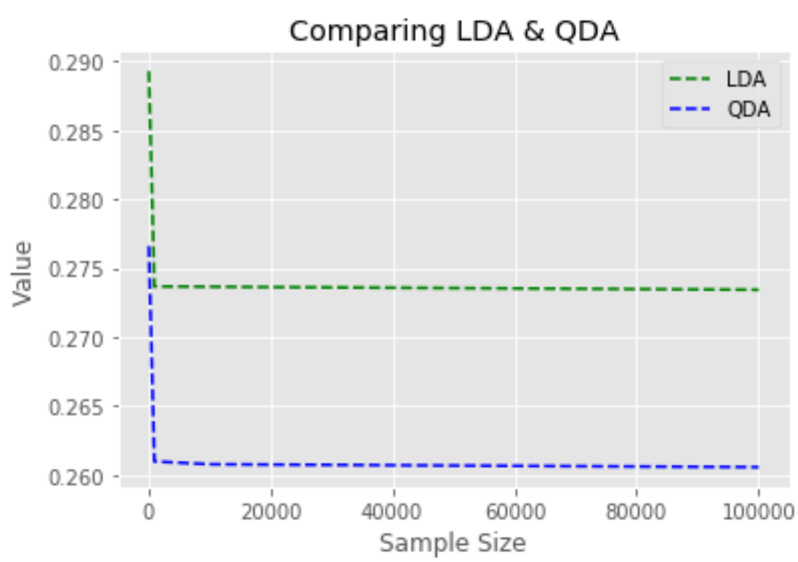
```
               LDA    QDA
training error 0.2724 0.2585
  test error   0.2753 0.2619
```



As depicted above, given a non-linear Bayes decision boundary, QDA outperforms LDA on both training error and test error.

4

In [119]:
```
seed = 2456
sample_size = [1e02, 1e03, 1e04, 1e05]

lda_test_err = []
qda_test_err = []
for n in sample_size:
    l_err = 0
    q_err = 0
    for i in range(1000):
        data = simulate_nonlinear_dataset(seed, int(n))
        seed += 1
        l1, l2 = LDA_fit(data)
        q1, q2 = QDA_fit(data)
        l_err += l2 / 1000
        q_err += q2 / 1000
    lda_test_err.append(l_err)
    qda_test_err.append(q_err)


p1=plt.plot(sample_size, lda_test_err, 'g--', label='LDA')
p2=plt.plot(sample_size, qda_test_err, 'b--', label='QDA')

plt.xlabel('Sample Size')
plt.ylabel('Value');
plt.title('Comparing LDA & QDA')
plt.legend()
plt.show();
```



In [125]:
```
for i in range(4):
    print(lda_test_err[i] - qda_test_err[i])
```

```
0.0127000000000001
0.012703333333333067
0.012888000000000177
0.012878999999999918
```

In general, as sample size n increases, the test error rate of both LDA and QDA declines, but the difference between the two essantially remains the same with a slight increase. The possible explanation is that, as the sample size increases, the training set increases correspondingly and the original risk of overfitting for QDA due to its high flexibility is reduced. On the other hand, its flexibility leads to better fitting than LDA which is more constrict.

Modeling voter turnout

1. (20 points) Building several classifiers and comparing output. a. 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,...,10 (that is, 10 separate models varying K) and Euclidean distance metrics c. Using the test set, calculate the following model performance metrics: i. Error rate ii. ROC curve(s) / Area under the curve (AUC) d. Which model performs the best? Be sure to define what you mean by “best” and identify supporting evidence to support your conclusion(s).

In [150]:
```
from sklearn.metrics import roc_auc_score
import pandas as pd
df = pd.read_csv('mental_health.csv')
df = df.dropna()
df.head()
```

Out[150]:

	vote96	mhealth_sum	age	educ	black	female	married	inc10
0	1.0	0.0	60.0	12.0	0	0	0.0	4.8149
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
7	0.0	6.0	29.0	13.0	0	0	0.0	10.6998
11	1.0	1.0	41.0	15.0	1	1	1.0	8.8273

In [154]:
```
# Split the data into a training and test set (70/30)
train, test = train_test_split(df, test_size=0.3, train_size=0.7, random
_state=1227)
X_train, y_train = train.drop('vote96', axis=1), train['vote96']
X_test, y_test = test.drop('vote96', axis=1), test['vote96']

mode = {'Model': ['Logistic Regression', 'LDA', 'QDA', 'Naive Bayes', 'k
1','k2','k3','k4','k5','k6','k7','k8','k9','k10'],
        'error rate':[],
        'AUC':[]}
```

In [155]:
```
def plot_roc(labels, predict_prob):
    false_positive_rate,true_positive_rate,thresholds=roc_curve(labels,
predict_prob)
    roc_auc=auc(false_positive_rate, true_positive_rate)
    plt.title('ROC')
    plt.plot(false_positive_rate, true_positive_rate,'b',label='AUC = %
0.4f'% roc_auc)
    plt.legend(loc='lower right')
    plt.plot([0,1],[0,1],'r--')
    plt.ylabel('TPR')
    plt.xlabel('FPR')
```

In [159]:

```
# logistic regression
from sklearn.datasets import load_iris
from sklearn.linear_model import LogisticRegression
X_train, y_train = load_iris(return_X_y=True)
clf_lr = LogisticRegression(random_state=0).fit(X_train, y_train)
test_err_lr = 1 - clf_lr.score(X_test, y_test)
mode['error rate'].append(test_err_lr)
plot_roc(y_test, clf_lr.predict_proba(X_test))
```

```
/opt/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/logisti
c.py:432: FutureWarning: Default solver will be changed to 'lbfgs' in
0.22. Specify a solver to silence this warning.
  FutureWarning)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/logisti
c.py:469: FutureWarning: Default multi_class will be changed to 'auto'
in 0.22. Specify the multi_class option to silence this warning.
  "this warning.", FutureWarning)

-----------------------------------------------------------------------
----
ValueError                                Traceback (most recent call l
ast)
<ipython-input-159-4908cea77fd5> in <module>
      4 X_train, y_train = load_iris(return_X_y=True)
      5 clf_lr = LogisticRegression(random_state=0).fit(X_train, y_trai
n)
----> 6 test_err_lr = 1 - clf_lr.score(X_test, y_test)
      7 mode['error rate'].append(test_err_lr)
      8 plot_roc(y_test, clf_lr.predict_proba(X_test))

/opt/anaconda3/lib/python3.7/site-packages/sklearn/base.py in score(sel
f, X, y, sample_weight)
    355         """
    356         from .metrics import accuracy_score
--> 357         return accuracy_score(y, self.predict(X), sample_weight
=sample_weight)
    358
    359

/opt/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/base.py
in predict(self, X)
    287             Predicted class label per sample.
    288         """
--> 289         scores = self.decision_function(X)
    290         if len(scores.shape) == 1:
    291             indices = (scores > 0).astype(np.int)

/opt/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/base.py
in decision_function(self, X)
    268         if X.shape[1] != n_features:
    269             raise ValueError("X has %d features per sample; exp
ecting %d"
--> 270                              % (X.shape[1], n_features))
    271
    272         scores = safe_sparse_dot(X, self.coef_.T,

ValueError: X has 7 features per sample; expecting 4
```

In [160]:
```
# LDA
clf_lda = LinearDiscriminantAnalysis()
clf_lda.fit(X_train, y_train)
test_err = 1 - clf.score(X_test, y_test)
mode['error rate'].append(test_err_lr)
plot_roc(y_test, clf_lda.predict_proba(X_test))
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-160-dec6c04c29a1> in <module>
      2 clf_lda = LinearDiscriminantAnalysis()
      3 clf_lda.fit(X_train, y_train)
----> 4 test_err = 1 - clf.score(X_test, y_test)
      5 mode['error rate'].append(test_err_lr)
      6 plot_roc(y_test, clf_lda.predict_proba(X_test))

/opt/anaconda3/lib/python3.7/site-packages/sklearn/base.py in score(self, X, y, sample_weight)
    355         """
    356         from .metrics import accuracy_score
--> 357         return accuracy_score(y, self.predict(X), sample_weight=sample_weight)
    358 
    359 

/opt/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/base.py in predict(self, X)
    287             Predicted class label per sample.
    288         """
--> 289         scores = self.decision_function(X)
    290         if len(scores.shape) == 1:
    291             indices = (scores > 0).astype(np.int)

/opt/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/base.py in decision_function(self, X)
    268         if X.shape[1] != n_features:
    269             raise ValueError("X has %d features per sample; expecting %d"
--> 270                              % (X.shape[1], n_features))
    271 
    272         scores = safe_sparse_dot(X, self.coef_.T,

ValueError: X has 7 features per sample; expecting 2
```