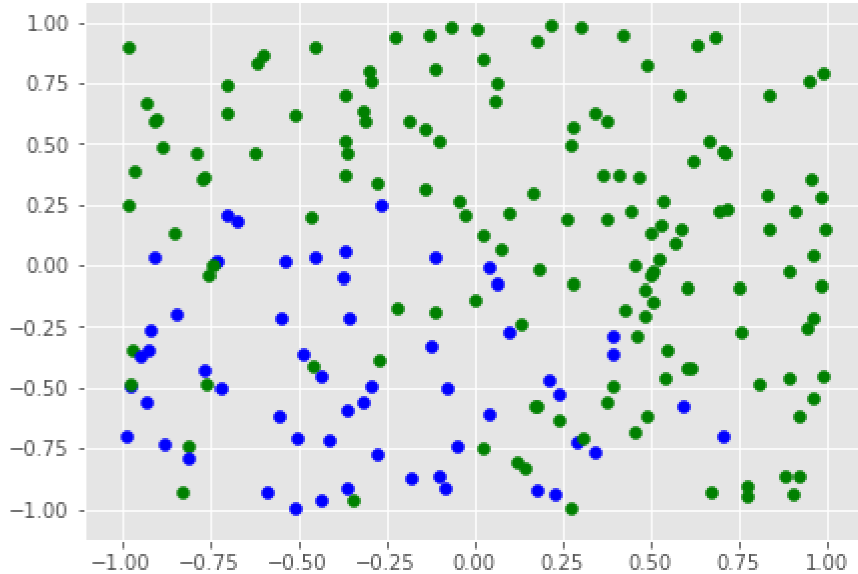
In [104]:
```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

import sklearn
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
```

The Bayes Classifier

Q1

In [217]:
```
import random
import math
from matplotlib.colors import ListedColormap

# Set your random number generator seed.
random.seed(1227)
n = 200

# Simulate a dataset of N = 200
def count_prob(x1, x2):
    err = 0.5 * np.random.randn()
    y = x1 + x1 ** 2 + x2 + x2 ** 2 + err
    p = math.exp(y) / (1 + math.exp(y))
    return p

rv1, rv2, p = [], [], []
for sample_size in range(n):
    x1 = random.uniform(-1, 1)
    x2 = random.uniform(-1, 1)
    rv1.append(x1)
    rv2.append(x2)
    p.append(count_prob(x1, x2))

fig = plt.figure(figsize=(7, 5))
ax = fig.add_subplot(111)
for index in range(n):
    if p[index] < 0.5:
        ax.scatter(rv1[index], rv2[index], color='blue', label='failure')
    else:
        ax.scatter(rv1[index], rv2[index], color='green', label='success')


# Bayes decision boundary
X1,X2 = np.meshgrid(rv1,rv2)
ls = np.c_[X1.ravel(), X2.ravel()].T
Z = np.array(list(map(count_prob, ls[0,:], ls[1,:])))
Z = Z.reshape(X1.shape)

# something went wrong with the countour function...
# ax.contour(X1, X2, Z,[0.5], colors='k')
plt.show()
```



Exploring Simulated Differences between LDA and QDA

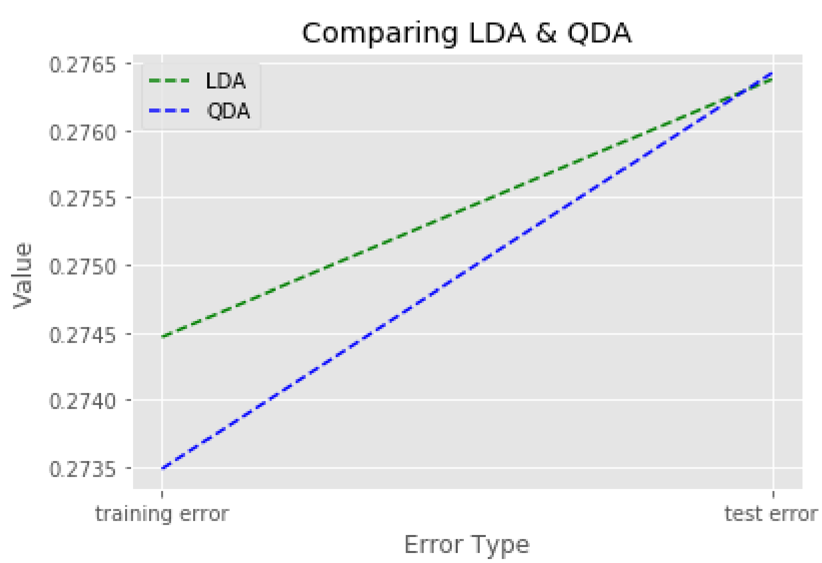
Q2

In [29]:
```
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 [30]:
```
def LDA_fit(df):
    train, test = train_test_split(df, test_size=0.3, train_size=0.7, rando
    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, rando
    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 [33]:
```
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_err],
p2=plt.plot(['training error','test error'], [qda_train_err, qda_test_err],

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

```
               LDA   QDA
training error 0.2745 0.2735
  test error   0.2764 0.2764
```



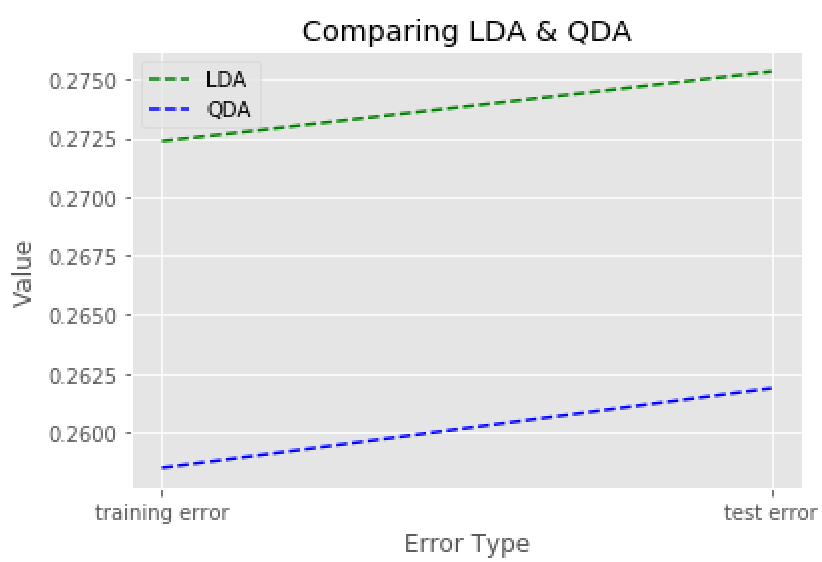
As depicted above, LDA has lower test error and QDA has lower training error. This may be expained by the fact that QDA is more flexible and may risk overfitting. Since we focus more on minimizing test error, LDA has better performance.

Q3

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_err],
p2=plt.plot(['training error','test error'], [qda_train_err, qda_test_err],

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
```

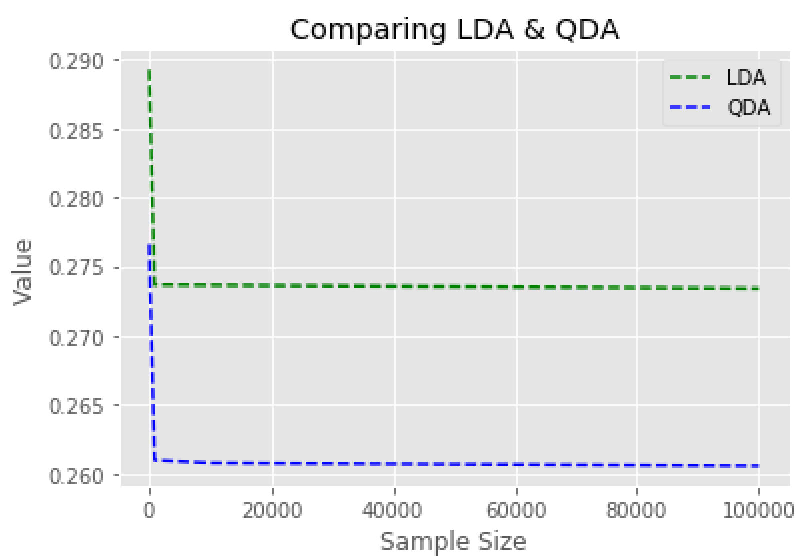


As depicted above, given a non-linear Bayes decision boundary, QDA outperforms LDA on both training error and test error.

Q4

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

Q5

In [96]:
```
from sklearn.metrics import roc_auc_score
import pandas as pd

df = pd.read_csv('mental_health.csv')
df.dropna(inplace=True)
df.drop('married', axis=1, inplace=True)
df.head()
```

Out[96]:

	vote96	mhealth_sum	age	educ	black	female	inc10
0	1.0	0.0	60.0	12.0	0	0	4.8149
2	1.0	1.0	36.0	12.0	0	0	8.8273
3	0.0	7.0	21.0	13.0	0	0	1.7387
7	0.0	6.0	29.0	13.0	0	0	10.6998
11	1.0	1.0	41.0	15.0	1	1	8.8273

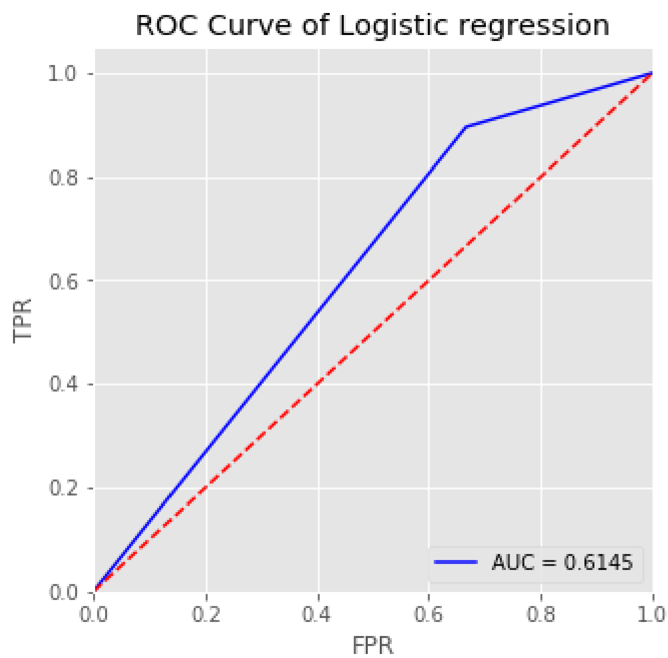
In [97]:
```
# Split the data into a training and test set (70/30)
train, test = train_test_split(df, test_size=0.3, train_size=0.7, random_st
X_train, y_train = train.drop('vote96', axis=1), train['vote96']
X_test, y_test = test.drop('vote96', axis=1), test['vote96']

model_data = {'name': [],
              'train error rate': [],
              'test error rate': [],
              'ROC_auc':[]}
```

In [84]:
```
from sklearn.metrics import roc_curve, auc

def plot_roc(y_test, y_scores, name):
    fpr, tpr, thresholds = roc_curve(y_test, y_scores)
    roc_auc = auc(fpr,tpr)

    plt.figure()
    plt.figure(figsize=(5,5))
    plt.plot(fpr, tpr, 'b', label='AUC = %0.4f' % roc_auc)
    plt.plot([0, 1], [0, 1], color='r', linestyle='--')
    plt.title('ROC Curve of '+name)
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.legend(loc='lower right')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.ylabel('TPR')
    plt.xlabel('FPR')
    plt.show();

    return roc_auc

def model_run(clf, name):
    model_data['name'].append(name)
    train_err = 1 - clf.score(X_train, y_train)
    model_data['train error rate'].append(train_err)
    test_err = 1 - clf.score(X_test, y_test)
    model_data['test error rate'].append(test_err)
    auc = plot_roc(y_test, clf.predict(X_test), name)
    model_data['ROC_auc'].append(auc)
```

In [98]:
```
# logistic regression
from sklearn.linear_model import LogisticRegression

clf_lr = LogisticRegression().fit(X_train, y_train)
model_run(clf_lr, 'Logistic regression')
```

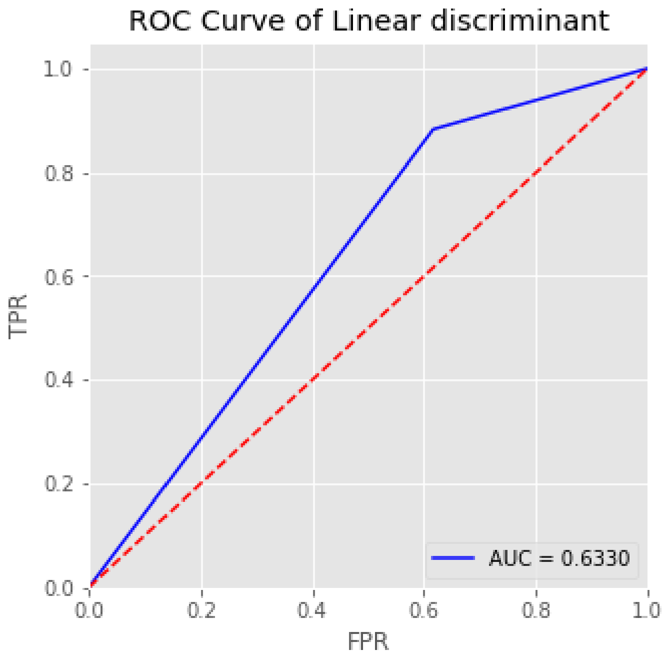
```
/opt/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/logistic.
py:432: FutureWarning: Default solver will be changed to 'lbfgs' in 0.22.
Specify a solver to silence this warning.
  FutureWarning)
```

```
<Figure size 432x288 with 0 Axes>
```



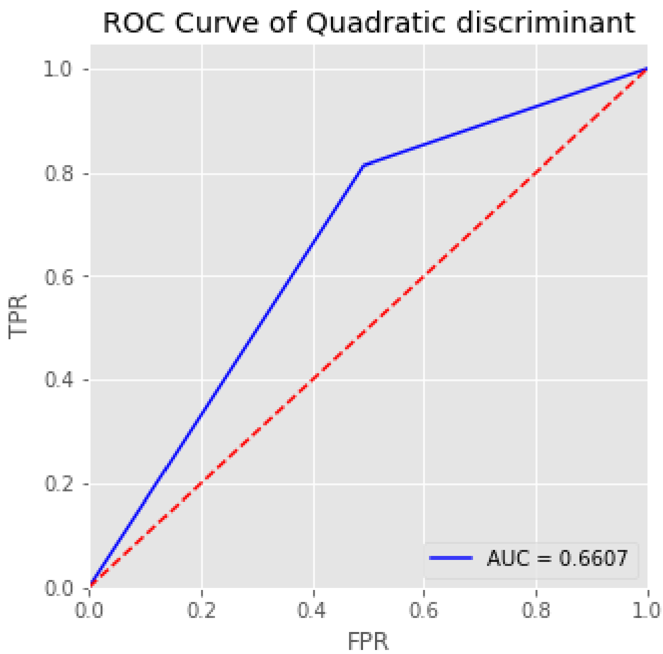
In [99]:
```
# LDA
clf_lda = LinearDiscriminantAnalysis().fit(X_train, y_train)
model_run(clf_lda, 'Linear discriminant')
```

```
<Figure size 432x288 with 0 Axes>
```



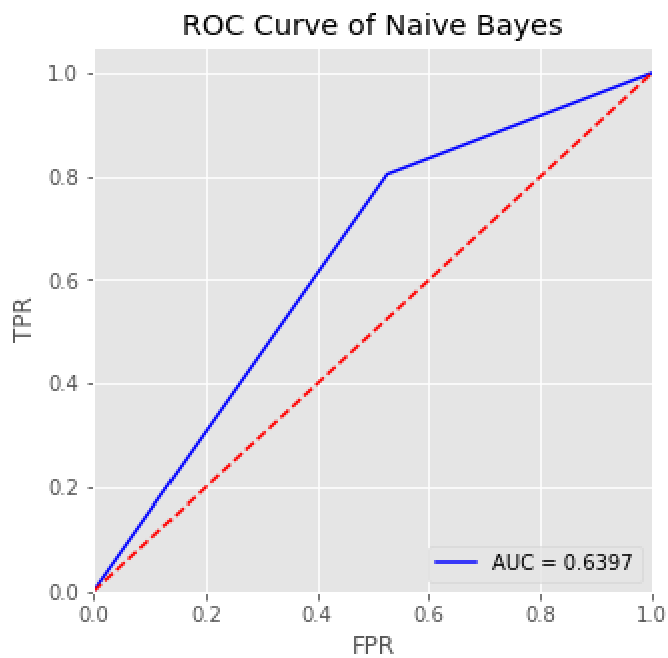
In [100]:
```
# QDA
clf_qda = QuadraticDiscriminantAnalysis().fit(X_train, y_train)
model_run(clf_qda, 'Quadratic discriminant')
```

```
<Figure size 432x288 with 0 Axes>
```



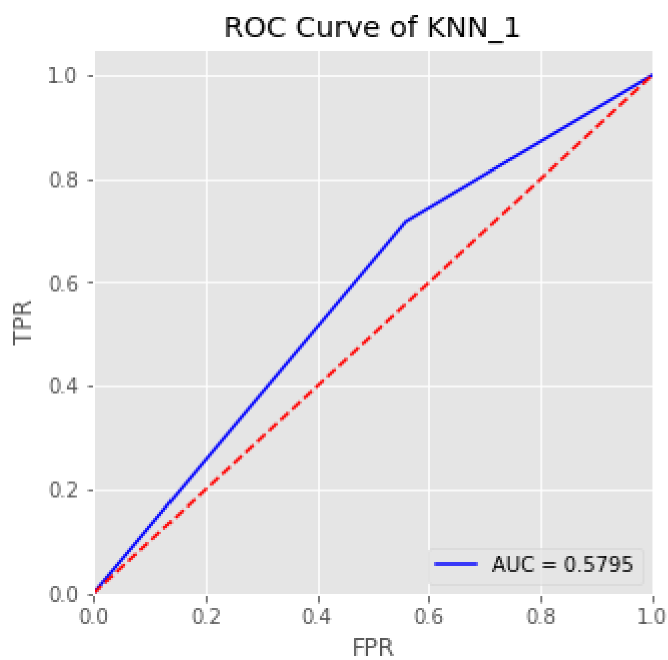
In [101]:
```
# Naive Bayes
clf_nb = GaussianNB().fit(X_train, y_train)
model_run(clf_nb, 'Naive Bayes')
```

```
<Figure size 432x288 with 0 Axes>
```

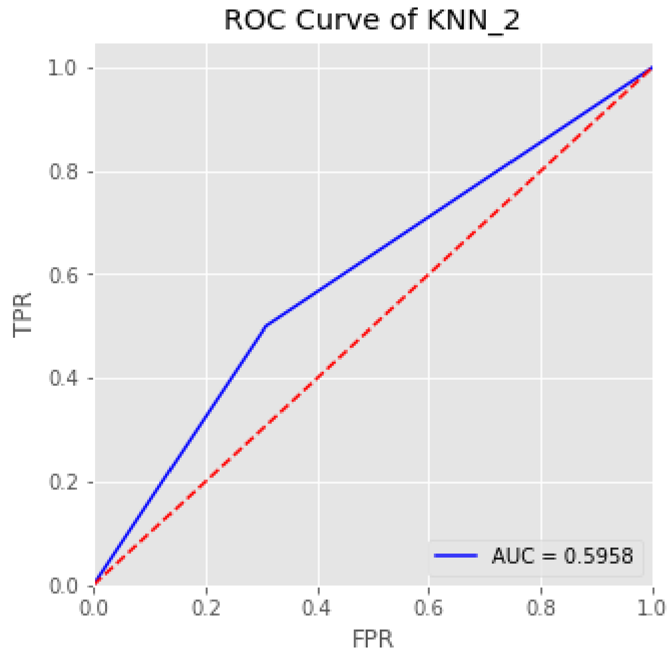


In [102]:
```
from sklearn.neighbors import KNeighborsClassifier

for k in range(1, 11):
    knn = KNeighborsClassifier(n_neighbors=k, metric='euclidean')
    knn.fit(X_train, y_train)
    model_run(knn, 'KNN_'+str(k))
```

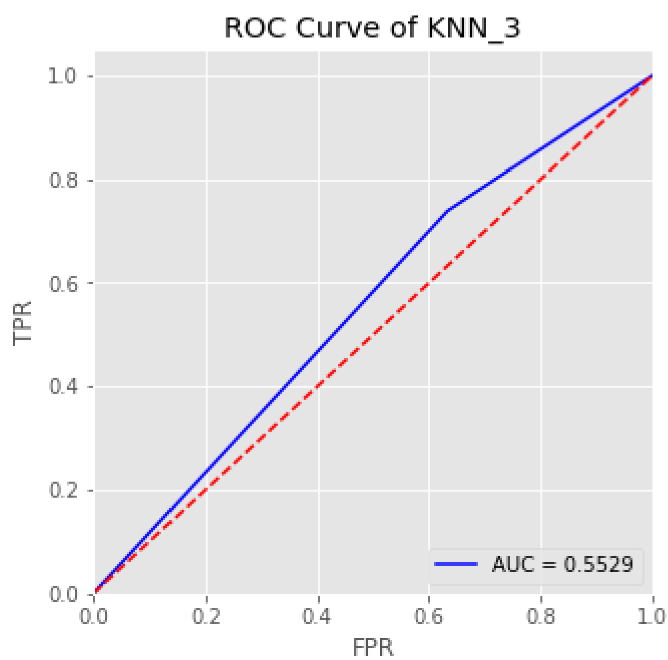
```
<Figure size 432x288 with 0 Axes>
```



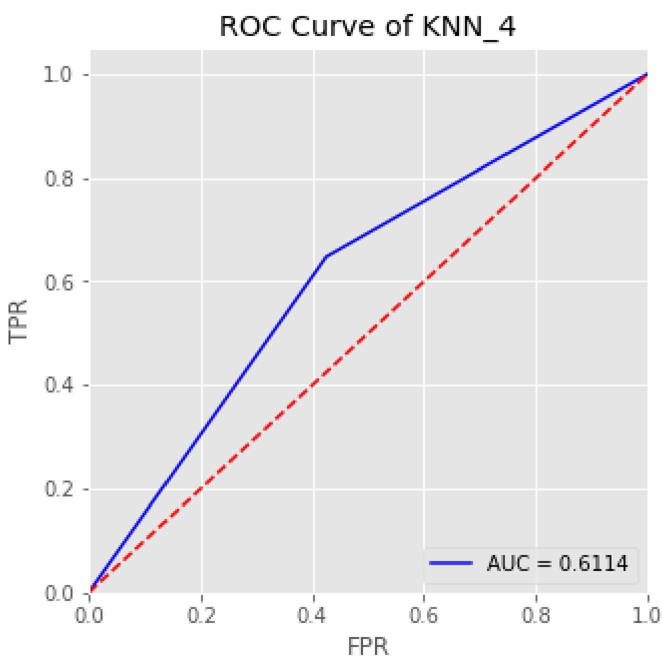

```
<Figure size 432x288 with 0 Axes>
```



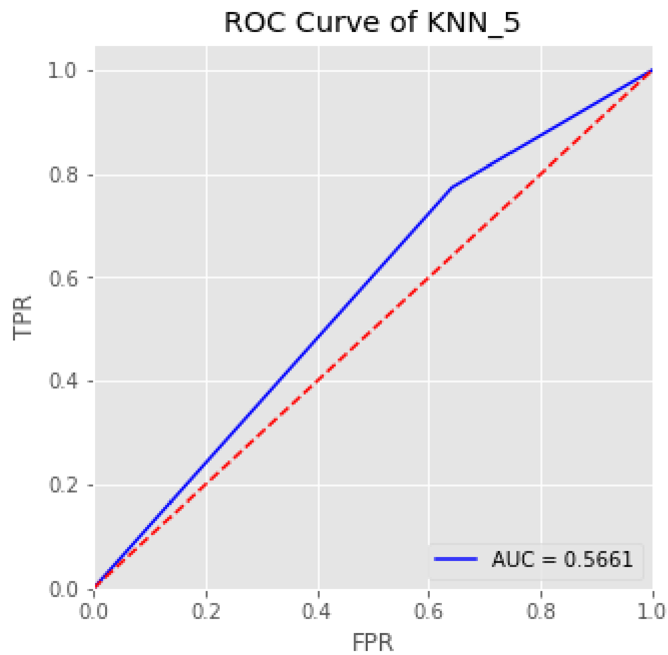
<Figure size 432x288 with 0 Axes>



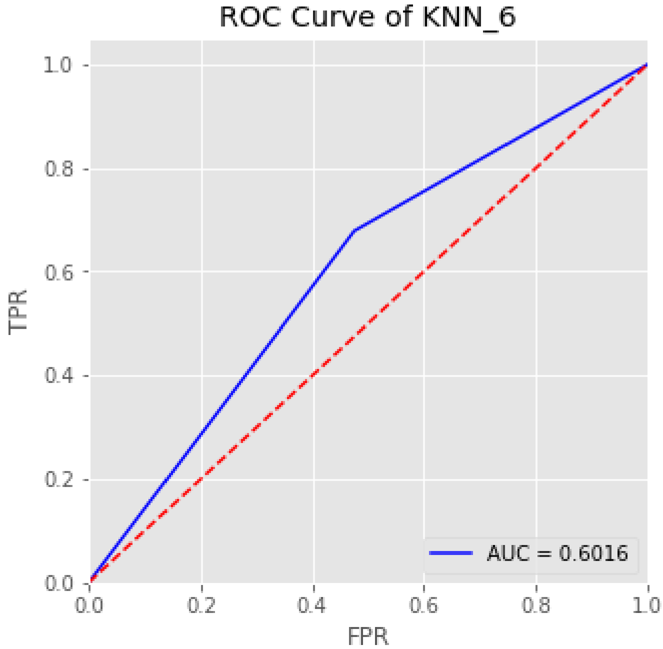
```
<Figure size 432x288 with 0 Axes>
```



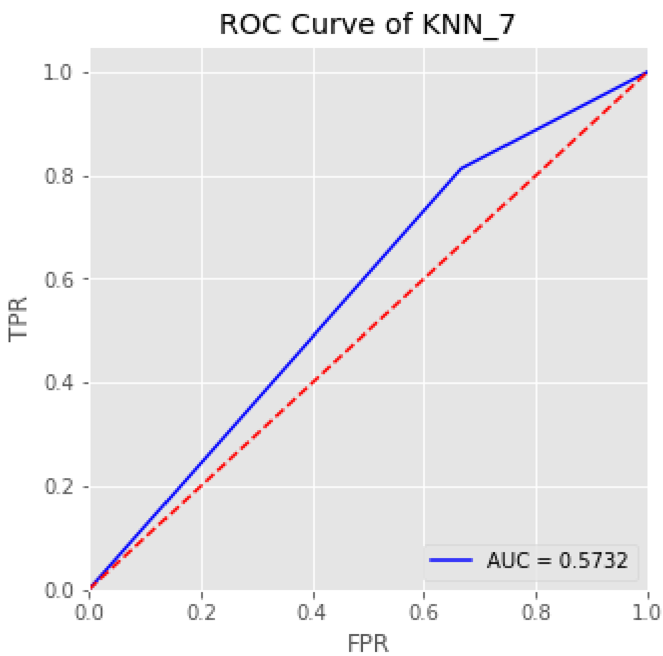
```
<Figure size 432x288 with 0 Axes>
```



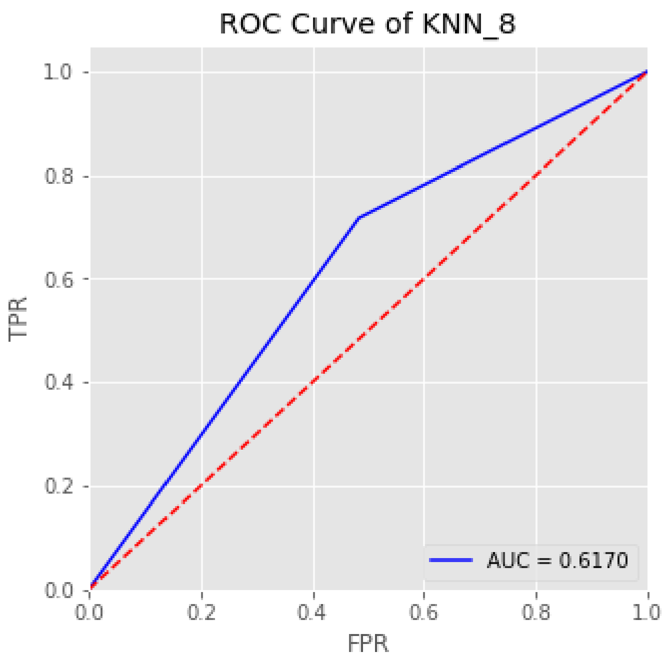

```
<Figure size 432x288 with 0 Axes>
```



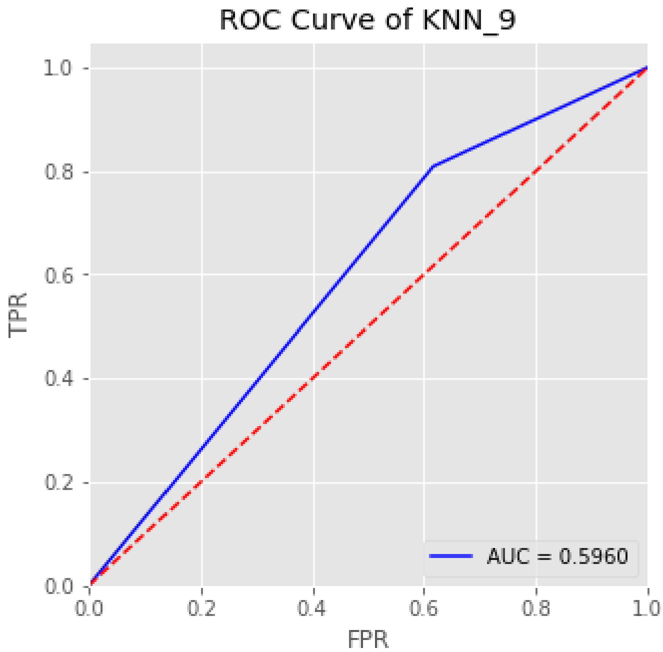

```
<Figure size 432x288 with 0 Axes>
```



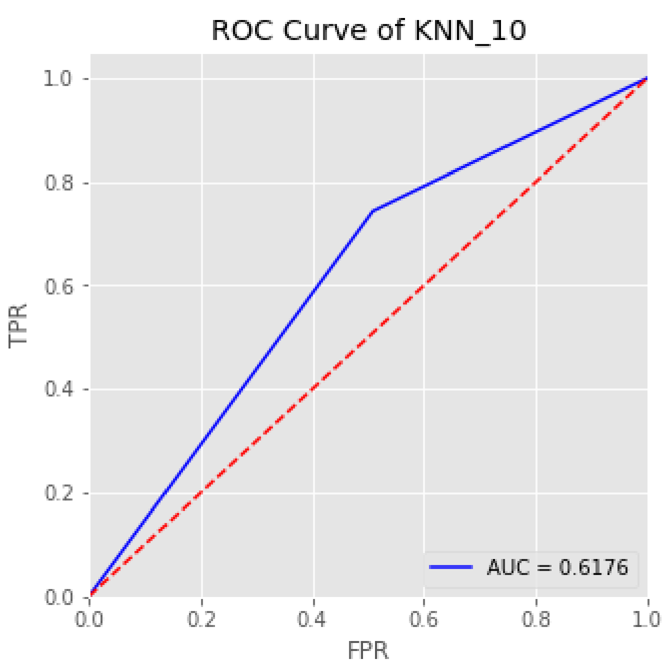
<Figure size 432x288 with 0 Axes>



<Figure size 432x288 with 0 Axes>



<Figure size 432x288 with 0 Axes>



In [103]:
```
model_df = pd.DataFrame(model_data)
model_df.sort_values(by='ROC_auc', axis=0, ascending=False)
```

Out[103]:

	name	train error rate	test error rate	ROC_auc
2	Quadratic discriminant	0.278528	0.291429	0.660688
3	Naive Bayes	0.289571	0.308571	0.639674
1	Linear discriminant	0.276074	0.288571	0.632971
13	KNN_10	0.256442	0.342857	0.617572
11	KNN_8	0.255215	0.351429	0.617029
0	Logistic regression	0.274847	0.297143	0.614493
7	KNN_4	0.229448	0.377143	0.611413
9	KNN_6	0.236810	0.374286	0.601630
12	KNN_9	0.241718	0.337143	0.596014
5	KNN_2	0.182822	0.434286	0.595833
4	KNN_1	0.000000	0.377143	0.579529
10	KNN_7	0.238037	0.351429	0.573188
8	KNN_5	0.240491	0.368571	0.566123
6	KNN_3	0.180368	0.388571	0.552899

For classification models, AUC is a better measure of classifier performance than accuracy because it does not bias on size of test or evaluation data. Consequently, the above data is sorted by auc, and QDA performs the best.