hw2

February 2, 2020

In [4]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [161]:
```
# classification methods
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis
from sklearn.neighbors import KNeighborsClassifier
import sklearn.metrics as metrics
```

1 Question 1

In [31]:
```
# Set your random number generator seed.
np.random.seed(20)
```

In [32]:
```
# Simulate a dataset
x1=np.random.uniform(-1,1,200)
x2=np.random.uniform(-1,1,200)
```

In [33]:
```
# Calculate Y
y=x1+x1**2+x2+x2**2+np.random.normal(0, 0.5, 200)
```

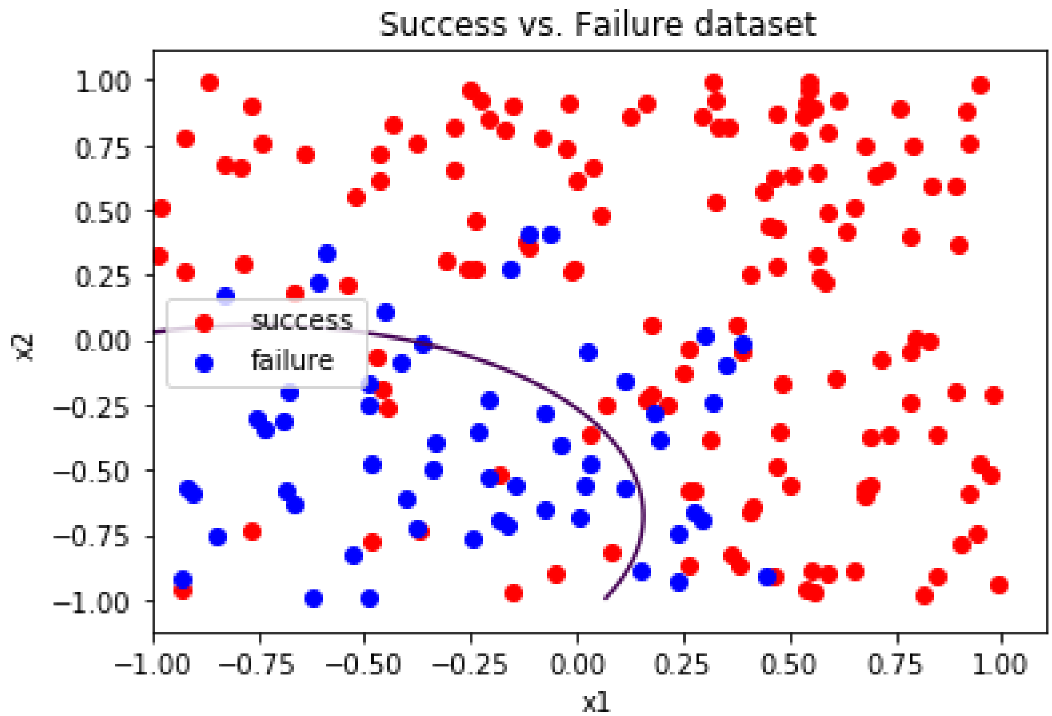
In [34]:
```
# Calculate the probability of success
prob=np.exp(y)/(1+np.exp(y))
```

In [35]:
```
# Plot each of the data points on a graph
def draw_plot(x1,x2,y):
    success=prob>0.5
    fail=prob<=0.5
    plt.scatter(x1[success], x2[success], color='red')
    plt.scatter(x1[fail], x2[fail], color='blue')
    plt.xlabel('x1')
    plt.ylabel('x2')
    plt.title("Success vs. Failure dataset")
```

```
    plt.legend(['success','failure'])
draw_plot(x1,x2,y)
```

In [36]:
```
# Bayes decision boundary
X=np.stack([x1,x2], axis=1)
X_df=pd.DataFrame(X)
nb=GaussianNB()
nb.fit(X_df, prob>0.5)
xx,yy=np.meshgrid(np.linspace(-1,1,100),np.linspace(-1,1,100))
z=nb.predict_proba(np.c_[xx.ravel(),yy.ravel()])
z=z[:,1].reshape((100,100))

draw_plot(x1,x2,y)
plt.contour(xx,yy,z,[0.5])
```

Out[36]:
```
<matplotlib.contour.QuadContourSet at 0x1a218303d0>
```



2 Question 2-1

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

From the coding result, we could see that if the actual decision boundary is linear, then we should expect LDA to perform better on the test set. However, since QDA is more flexible than LDA, it could have better performance on training set especially when it overfits the data.

```
In [74]: def simulation(n, is_linear, seed):
             np.random.seed(seed)

             # Simulate a dataset of 1000 observations
             x1=np.random.uniform(-1,1,n)
             x2=np.random.uniform(-1,1,n)
             if is_linear==1:
                 y=x1+x2+np.random.normal(0, 1, n)
             else:
                 y=x1+x1**2+x2+x2**2+np.random.normal(0, 1, n)
             y_binary=y>0

             # Randomly split your dataset into 70/30% training/test sets.
             X=np.stack([x1,x2], axis=1)
             X_train, X_test, y_train, y_test=train_test_split(X,y_binary, test_size=0.3)
```

```
    # Use the training dataset to estimate LDA and QDA models.
    lda=LinearDiscriminantAnalysis()
    lda.fit(X_train, y_train)
    qda=QuadraticDiscriminantAnalysis()
    qda.fit(X_train, y_train)

    # Calculate each models training and test error rate.
    train_err_lda=1-lda.score(X_train, y_train)
    test_err_lda=1-lda.score(X_test, y_test)
    train_err_qda=1-qda.score(X_train, y_train)
    test_err_qda=1-qda.score(X_test, y_test)

    return train_err_lda,test_err_lda,train_err_qda,test_err_qda
```

In [75]:
```
# do question a. one time
train_err_lda,test_err_lda,train_err_qda,test_err_qda=simulation(1000, 1, 20)
err_dict={"Train Error": [train_err_lda, train_err_qda],"Test Error": [test_err_lda, t
pd.DataFrame.from_dict(err_dict, orient="index", columns=['LDA','QDA'])
```

Out[75]:

	LDA	QDA
Train Error	0.271429	0.274286
Test Error	0.310000	0.313333

In [78]:
```
# Summarize 1000 simulations
all_train_err_lda=[]
all_test_err_lda=[]
all_train_err_qda=[]
all_test_err_qda=[]

for i in range(1000):
    train_err_lda,test_err_lda,train_err_qda,test_err_qda=simulation(1000,1,i)
    all_train_err_lda.append(train_err_lda)
    all_test_err_lda.append(test_err_lda)
    all_train_err_qda.append(train_err_qda)
    all_test_err_qda.append(test_err_qda)
```

In [81]:
```
def avg(ls):
    return sum(ls)/len(ls)
```

In [82]:
```
# tabular form for the mean of 1000 simulation
err_dict={"Train Error": [avg(all_train_err_lda), avg(all_train_err_qda)],"Test Error'
pd.DataFrame.from_dict(err_dict, orient="index", columns=['LDA','QDA'])
```

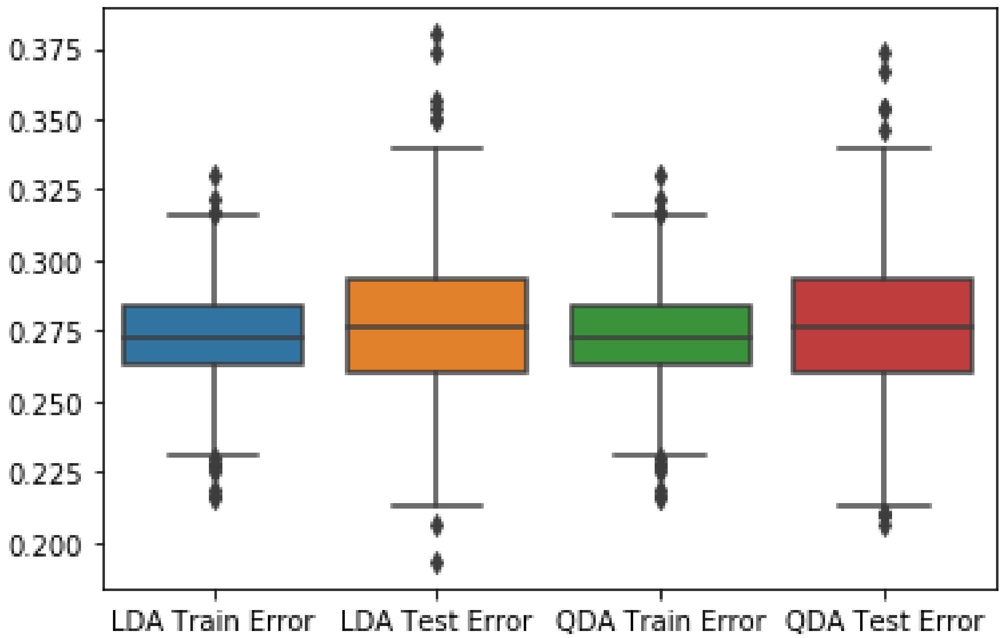
Out[82]:

	LDA	QDA
Train Error	0.273524	0.272579
Test Error	0.276933	0.277523

In [88]:
```
# graphic form
all_errors= pd.DataFrame(
```

```
    {'LDA Train Error': all_train_err_lda,
     'LDA Test Error': all_test_err_lda,
     'QDA Train Error': all_train_err_lda,
     'QDA Test Error': all_test_err_qda
    })

sns.boxplot(data=all_errors)
```

Out[88]: <matplotlib.axes._subplots.AxesSubplot at 0x1a23566990>



3 Question 2-2

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?

From the coding result, we could find that QDA generally has better performance in both the training and test set if the Bayes decision boundary is non-linear.

In [89]:
```
# do question a. one time
train_err_lda,test_err_lda,train_err_qda,test_err_qda=simulation(1000, 0, 20)
err_dict={"Train Error": [train_err_lda, train_err_qda],"Test Error": [test_err_lda, t
pd.DataFrame.from_dict(err_dict, orient="index", columns=['LDA','QDA'])
```

Out[89]:

	LDA	QDA
Train Error	0.255714	0.250000
Test Error	0.263333	0.246667

In [92]:
```
# Summarize 1000 simulations
all_train_err_lda=[]
all_test_err_lda=[]
all_train_err_qda=[]
all_test_err_qda=[]

for i in range(1000):
    train_err_lda,test_err_lda,train_err_qda,test_err_qda=simulation(1000,0,i)
    all_train_err_lda.append(train_err_lda)
    all_test_err_lda.append(test_err_lda)
    all_train_err_qda.append(train_err_qda)
    all_test_err_qda.append(test_err_qda)
```

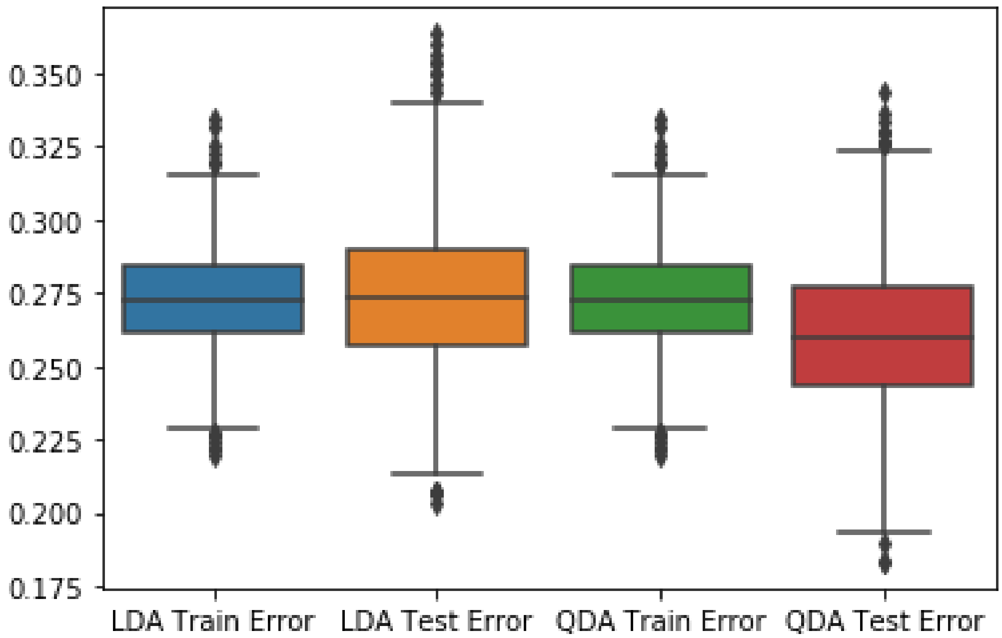
In [93]:
```
# tabular form for the mean of 1000 simulation
err_dict={"Train Error": [avg(all_train_err_lda), avg(all_train_err_qda)],"Test Error'
pd.DataFrame.from_dict(err_dict, orient="index", columns=['LDA','QDA'])
```

Out[93]:

	LDA	QDA
Train Error	0.272643	0.259181
Test Error	0.274383	0.261157

In [94]:
```
# graphic form
all_errors= pd.DataFrame(
    {'LDA Train Error': all_train_err_lda,
     'LDA Test Error': all_test_err_lda,
     'QDA Train Error': all_train_err_lda,
     'QDA Test Error': all_test_err_qda
    })

sns.boxplot(data=all_errors)
```

Out[94]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1a2356ca10>
```



4 Question 2-3

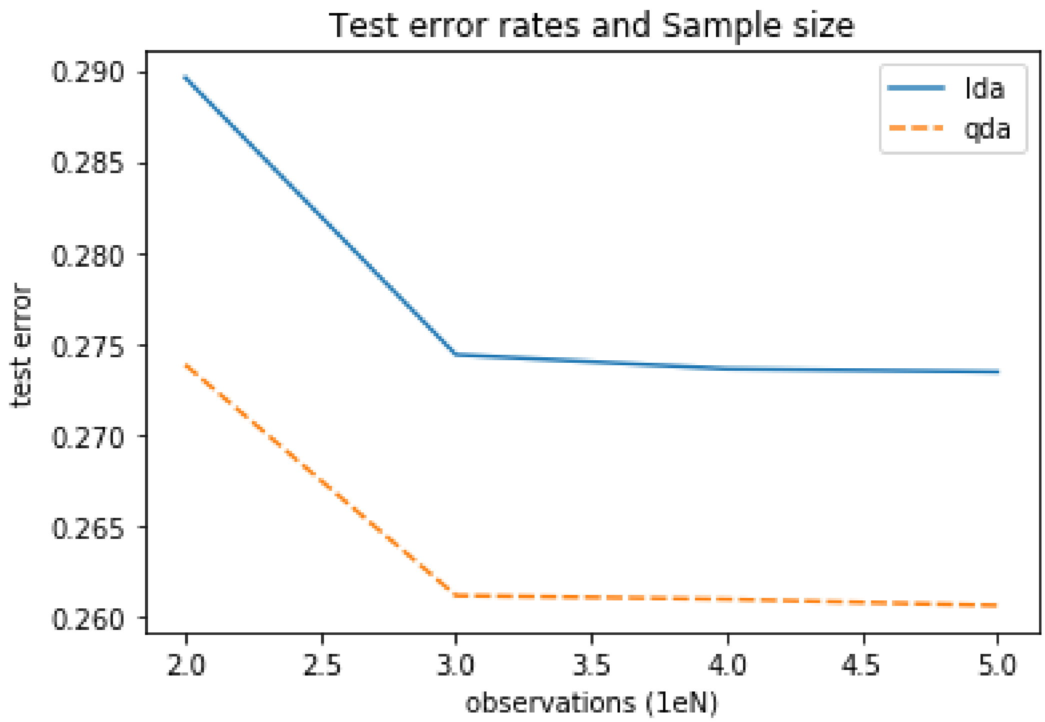
In general, as sample size n increases, do we expect the test error rate of QDA relative to LDA to improve, decline, or be unchanged? Why?

From the graph, we could see that as sample size n increases, we woyld expect QDA to have more performance improvement than LDA. This is because larger sample size and non-linear classficiation both make QDA a more suitable model to fit the data.

```
In [180]: n_ls = [1e02, 1e03, 1e04, 1e05]
          all_test_error={}

          for n in n_ls:
              all_test_error[int(np.log10(n))]={}
              all_test_err_lda=[]
              all_test_err_qda=[]

              for i in range(1000):
                  train_err_lda,test_err_lda,train_err_qda,test_err_qda=simulation(int(n),0,i)
                  all_test_err_lda.append(test_err_lda)
                  all_test_err_qda.append(test_err_qda)

              all_test_error[int(np.log10(n))]['lda']=avg(all_test_err_lda)
              all_test_error[int(np.log10(n))]['qda']=avg(all_test_err_qda)

In [181]: all_test_err_df=pd.DataFrame.from_dict(all_test_error, orient="index")
```

```
In [182]: ax=sns.lineplot(data=all_test_err_df)
          ax.set_xlabel("observations (1eN)")
          ax.set_ylabel("test error")
          ax.set_title("Test error rates and Sample size")
```

```
Out[182]: Text(0.5, 1.0, 'Test error rates and Sample size')
```



5 Question 3

```
In [149]: # data preparation
          health_df=pd.read_csv("mental_health.csv")
          health_df.dropna(inplace=True)
```

```
In [150]: # Split the data into a training and test set (70/30).
          vote96=health_df['vote96']
          x_df=health_df.drop(columns=['vote96'])
          x_train, x_test, y_train, y_test=train_test_split(x_df,vote96, test_size=0.3)
```

```
In [151]: clfs={}
```

```
In [152]: # Logistic regression model
          log=LogisticRegression()
          log.fit(x_train, y_train)
          clfs['log']=log
```

```
/Users/ziwenchen/opt/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/logistic.py:432
  FutureWarning)
```

```
In [153]: # Linear discriminant model
          lda=LinearDiscriminantAnalysis()
          lda.fit(x_train, y_train)
          clfs['lda']=lda
```

```
In [154]: # Quadratic discriminant model
          qda=QuadraticDiscriminantAnalysis()
          qda.fit(x_train, y_train)
          clfs['qda']=qda
```

```
In [155]: # Naive Bayes
          nb=GaussianNB()
          nb.fit(x_train, y_train)
          clfs['nb']=nb
```

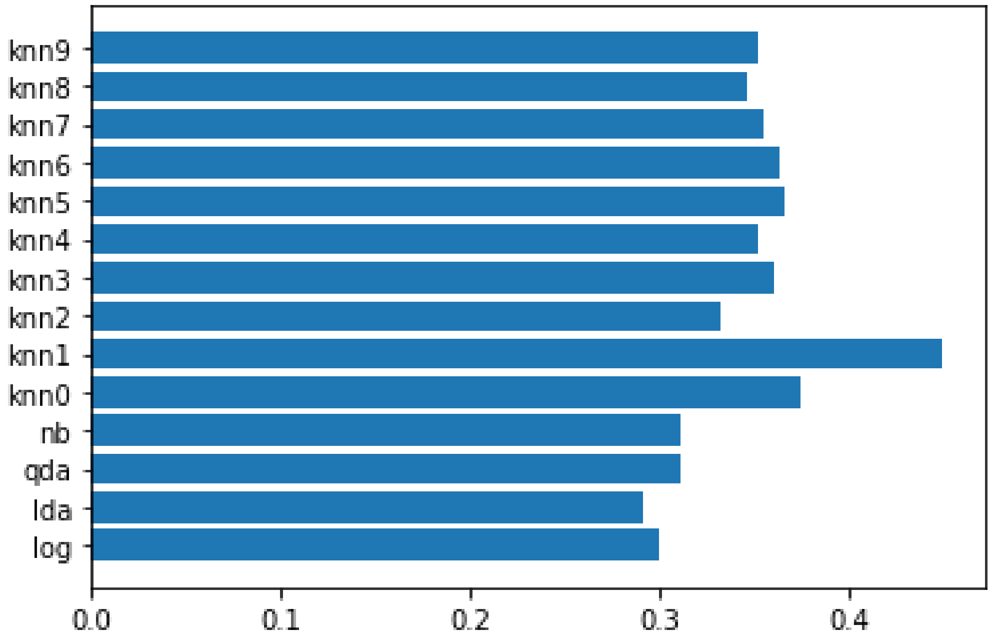
```
In [156]: # K-nearest neighbors
          for k in range(10):
              clfs['knn'+str(k)]=KNeighborsClassifier(k+1)
              clfs['knn'+str(k)].fit(x_train, y_train)
```

```
In [175]: list(err_rate_dict.values())
```

```
Out[175]: [0.30000000000000004,
           0.2914285714285715,
           0.3114285714285714,
           0.3114285714285714,
           0.37428571428571433,
           0.4485714285714286,
           0.3314285714285714,
           0.36,
           0.3514285714285714,
           0.36571428571428577,
           0.3628571428571429,
           0.3542857142857143,
           0.34571428571428575,
           0.3514285714285714]
```

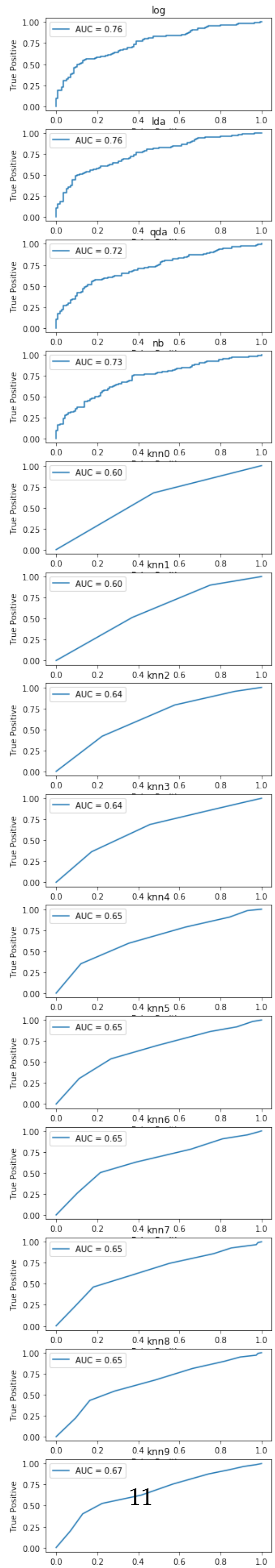
```
In [176]: # Error rate for the models
          err_rate_dict={}

          for clf in clfs.keys():
              err_rate=1-clfs[clf].score(x_test, y_test)
              err_rate_dict[clf]=err_rate

          clf_ls=list(clfs.keys())
          plt.barh(clf_ls, list(err_rate_dict.values()))
```

Out[176]: <BarContainer object of 14 artists>



In [158]:
```
def get_auc(model, x_test, y_test):
    fpr, tpr, thresholds = metrics.roc_curve(y, model.predict(x_test))
    return metrics.auc(fpr, tpr)
```

In [168]:
```
# ROC curve(s) / Area under the curve (AUC)
fig, ax=plt.subplots(len(clfs),1,figsize=(5,35))
i=0
auc_dict={}
for clf in clfs.keys():
    model=clfs[clf]
    fpr, tpr, thresholds = metrics.roc_curve(y_test, model.predict_proba(x_test)[:,1
    auc=metrics.auc(fpr, tpr)
    auc_dict[clf]=auc
    ax[i].plot(fpr, tpr, label='AUC = %0.2F'%auc)
    ax[i].set_xlabel("False Positive")
    ax[i].set_ylabel("True Positive")
    ax[i].set_title(clf)
    ax[i].legend()
    i+=1
```


log
AUC = 0.76
lda
AUC = 0.76
qda
AUC = 0.72
nb
AUC = 0.73
knn0
AUC = 0.60
knn1
AUC = 0.60
knn2
AUC = 0.64
knn3
AUC = 0.64
knn4
AUC = 0.65
knn5
AUC = 0.65
knn6
AUC = 0.65
knn7
AUC = 0.65
knn8
AUC = 0.65
knn9
AUC = 0.67
True Positive
False Positive
0.00 0.25 0.50 0.75 1.00
0.0 0.2 0.4 0.6 0.8 1.0

```
In [169]: # sorted auc
          for key, value in sorted(auc_dict.items(), key=lambda item: item[1]):
              print("%s: %s" % (key, value))
```

```
knn1: 0.5996905393457117
knn0: 0.6005378720895962
knn3: 0.6398283230179782
knn2: 0.6424624226348363
knn8: 0.651543619216033
knn6: 0.6528698791629826
knn4: 0.6536066902446211
knn5: 0.6541592985558503
knn7: 0.654325081049219
knn9: 0.6688218390804597
qda: 0.722332743884468
nb: 0.7263483642793989
log: 0.7574417919245506
lda: 0.758841732979664
```

```
In [177]: # sorted err rate
          for key, value in sorted(err_rate_dict.items(), key=lambda item: item[1]):
              print("%s: %s" % (key, value))
```

```
lda: 0.2914285714285715
log: 0.30000000000000004
qda: 0.3114285714285714
nb: 0.3114285714285714
knn2: 0.3314285714285714
knn8: 0.34571428571428575
knn4: 0.3514285714285714
knn9: 0.3514285714285714
knn7: 0.3542857142857143
knn3: 0.36
knn6: 0.3628571428571429
knn5: 0.36571428571428577
knn0: 0.37428571428571433
knn1: 0.4485714285714286
```

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

Based on the error rate and AUC/ROC of the models, we can see that LDA has the highest AUC and lowest error rate. Therefore, LDA should be the best model for this predicted task. Also, logistic regression also has quite similar performance to LDA. Therefore, logistic regression might be another suitable method for the task.

The definition of "best" is based both on error rates and ROC/AUC. Error rates represent model's acuracy on the test set, while AUC measures the tradeoff between true/false postive. The higher the AUC, the lower the error rate, the better the model.