HW2_Aabir_AK

February 2, 2020

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

sns.set()

np.random.seed(42)

%matplotlib inline
```

0.1 1. Bayes Classifier

(20 points) 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 = x0) where x0 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 x0. 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 = x0) > 0.5, and class two otherwise. Produce a graph illustrating this concept. Specifically, implement the following elements in your program:

a. Set your random number generator seed.

b. Simulate a dataset of N = 200 with X1, X2 where X1, X2 are random uniform variables between [−1, 1].

c. CalculateY =X1+X12+X2+X2+ ,where N(=0, 2 =0.25).

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

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.

[2]:
```
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split

X1, X2 = [np.random.uniform(-1, +1, 200) for i in range(2)]
```

[3]:
```
Y = X1 + X1**2 + X2 + X2**2 + np.random.normal(0, np.sqrt(0.25), 200)
```

Y = log odds of success = $log\left(\frac{Pr(success)}{Pr(failure)}\right)$

$$\therefore Pr(success) = \frac{exp(Y)}{1 + exp(Y)}$$

[4]:
```
pr_success = np.exp(Y)/(1+np.exp(Y))
print(pr_success.shape)
```

```
(200,)
```

[5]:
```
successes = [i for i, p in enumerate(pr_success) if p > 0.5]
failures = [i for i, p in enumerate(pr_success) if p < 0.5]


plt.scatter(X1[successes], X2[successes], c='b', label='success')
plt.scatter(X1[failures], X2[failures], c='r', label='failure')
plt.xlabel("X1")
plt.ylabel("X2")
plt.ylim(-1.1, 1.4)
plt.xlim(-1.1, 1.4)
plt.legend(loc='upper right')
plt.title('Classification of success and failure - training data')
plt.show()
```

[6]:
```
# I'm not splitting the data for this exercise as we are asked to calculate the␣
 ↪decision boundary using X1 and X2

Xs = np.array([X1, X2]).T.astype('float64')
Ys = 1*Y>0.5
print(Xs.shape, Y.shape, Xs.dtype, Y.dtype)

gnb = GaussianNB()
gnb.fit(X=Xs, y=Ys)
```

```
(200, 2) (200,) float64 float64
```

[6]:
```
GaussianNB(priors=None, var_smoothing=1e-09)
```

[7]:
```
gnb.predict([[-1, -1], [1, 1]])
```

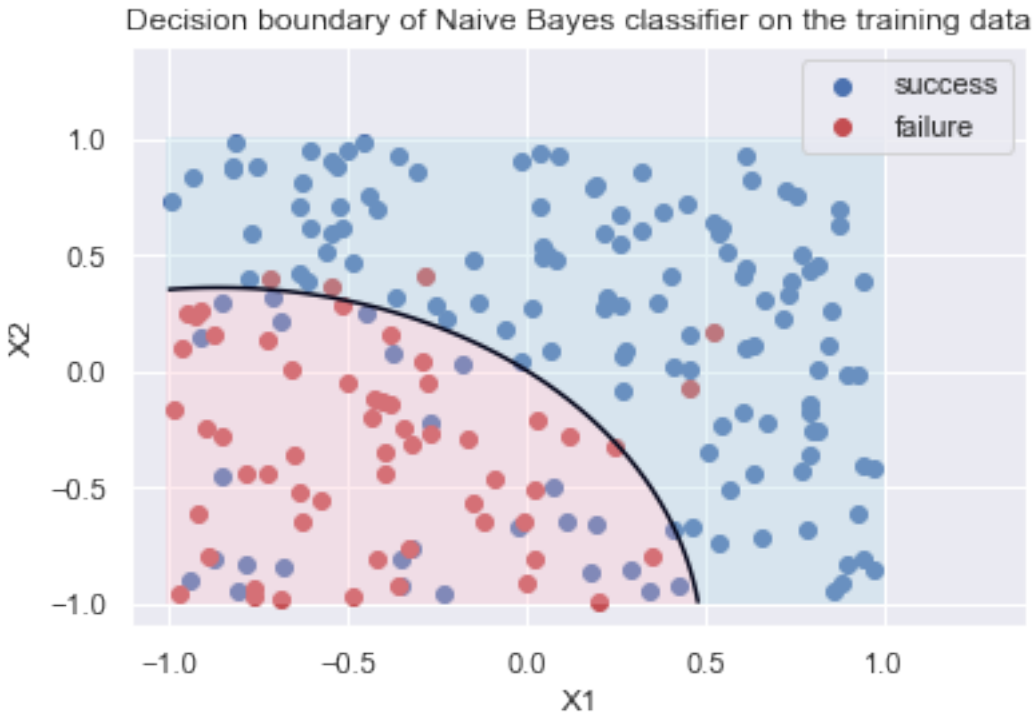
[7]:
```
array([False,  True])
```

[8]:
```
# generate gridded data to plot the decision boundary of our Naive Bayes␣
 ↪classifier

x1s = np.linspace(-1, 1, 100)
xx, yy = np.meshgrid(x1s, x1s)
```

```
Z = gnb.predict_proba(np.c_[xx.ravel(), yy.ravel()])
Z = Z[:,1].reshape(100, 100)
print(Z.shape, xx.shape, yy.shape)

plt.scatter(X1[successes], X2[successes], c='b', label='success')
plt.scatter(X1[failures], X2[failures], c='r', label='failure')
plt.xlabel("X1")
plt.ylabel("X2")
plt.contour(xx, yy, Z, [0.5])
plt.contourf(xx, yy, Z, [0, 0.5, 1], alpha=0.3, colors=['pink', 'lightblue'])
plt.ylim(-1.1, 1.4)
plt.xlim(-1.1, 1.4)
plt.title('Decision boundary of Naive Bayes classifier on the training data')
plt.legend(loc='upper right')
```

```
(100, 100) (100, 100) (100, 100)
```

[8]: <matplotlib.legend.Legend at 0x1a2090af60>



0.2 2. Exploring Simulated Differences between LDA and QDA

Note: Unless otherwise specified, assume the number of observations N = 1000.

2. (20 points) If the Bayes decision boundary is linear, do we expect LDA or QDA to perform

better on the training set? On the test set?

a. Repeat the following process 1000 times.

b. Simulate a dataset of 1000 observations with X1 , X2 Uniform(−1, +1). Y is a binary

response variable defined by a Bayes decision boundary of f(X) = X1 + X2, where values 0 or greater are coded TRUE and values less than 0 or coded FALSE. Whereas your simulated Y is a function of X1 + X2 + where N (0, 1). That is, your simulated Y is a function of the Bayes decision boundary plus some irreducible error.

ii. Randomly split your dataset into 70/30% training/test sets. iii. Use the training dataset to estimate LDA and QDA models. iv. Calculate each model's training and test error rate.

b. Summarize all the simulations' error rates and report the results in tabular and graphical form. Use this evidence to support your answer.3. (20 points) 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?

c. Repeat the following process 1000 times.

d. Simulate a dataset of 1000 observations with X1 , X2 Uniform(−1, +1). Y is a binary

response variable defined by a Bayes decision boundary of f(X) = X1 + X12 + X2 + X2, where values 0 or greater are coded TRUE and values less than 0 or coded FALSE. Whereas your simulated Y is a function of X1 +X12 +X2 +X2 + where N(0,1). That is, your simulated Y is a function of the Bayes decision boundary plus some irreducible error.

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

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

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

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

4. (20 points) 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?

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).

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.

[67]:
```
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis,␣
 ↪LinearDiscriminantAnalysis
from tabulate import tabulate

def get_XY(n):
    n = int(n)
    X1, X2 = (np.random.uniform(-1, 1, n) for i in range(2))
    f = X1+X2
    fbin = f>=0
```

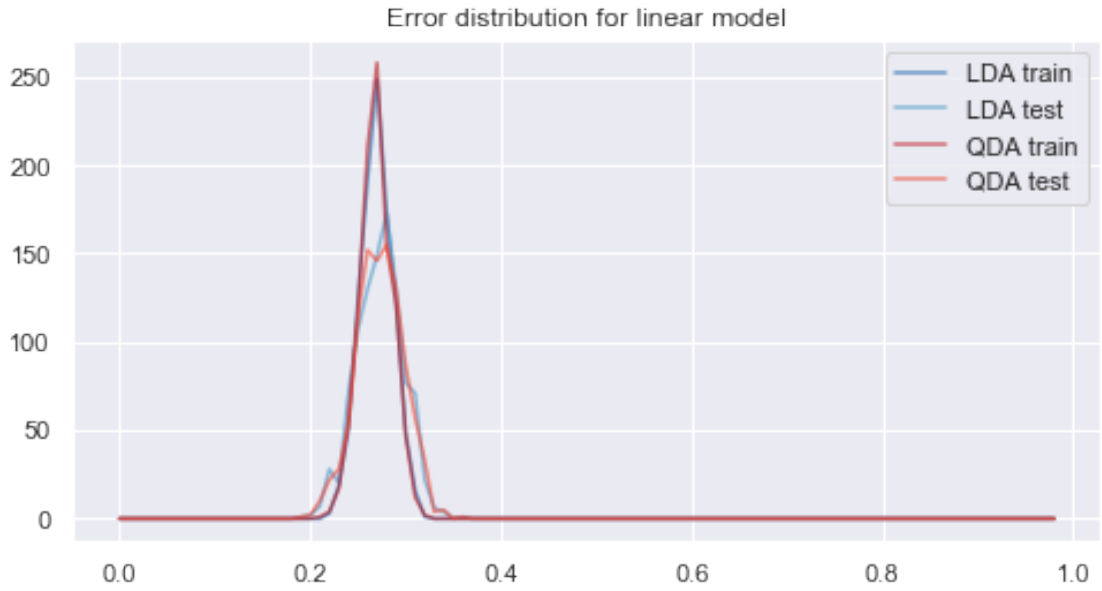
```
    Y = X1+X2+np.random.randn(n)
    Y = Y>=0
    Xs = np.column_stack((X1, X2))
    return Xs, Y

def get_XY2(n):
    n = int(n)
    X1, X2 = [np.random.uniform(-1, 1, n) for i in range(2)]
    f = X1+X2
    fbin = f>=0
    Y = X1+X2+X1**2+X2**2+np.random.randn(n)
    Y = Y>=0
    Xs = np.column_stack((X1, X2))
    return Xs, Y

def get_summary_stats(x):
    l = [np.min(x), *np.percentile(x, [25, 50]), np.mean(x), np.percentile(x,␣
 ↪75), np.max(x), np.std(x)]
    return l

def tabulate_errors(lda_tr, lda_te, qda_tr, qda_te):
    print(tabulate([['LDA train', *get_summary_stats(lda_tr)],
                ['LDA test', *get_summary_stats(lda_te)],
                ['QDA train', *get_summary_stats(qda_tr)],
                ['QDA test', *get_summary_stats(qda_te)]],
                headers=['Model/Data', 'Min', 'p25', 'Median', 'Mean', 'p75',␣
 ↪'Max', 'Std dev']))

def plot_err_dist(lda_tr, lda_te, qda_tr, qda_te, title=''):
    fig = plt.figure(figsize=(8, 4))
    labels = iter(['LDA train', 'LDA test', 'QDA train', 'QDA test'])
    cols = iter([plt.cm.Blues(0.8), plt.cm.Blues(0.6), plt.cm.Reds(0.8), plt.cm.
 ↪Reds(0.6)])
    for errors in [lda_tr, lda_te, qda_tr, qda_te]:
        hist, edg = np.histogram(errors, bins=np.arange(0, 1, 0.01))
        plt.plot(edg[:-1], hist, label=next(labels), c=next(cols), alpha=0.6)
    plt.title(f'Error distribution {title}')
    plt.legend()
    plt.show()
```

[46]:
```
qda_train_err = []
qda_test_err = []
lda_train_err = []
lda_test_err = []

for r in range(1000):
    Xs, Y = get_XY(1000)
```

```
    X_train, X_test, y_train, y_test = train_test_split(Xs, Y, train_size=0.7,␣
↪test_size=0.3)
    qda = QuadraticDiscriminantAnalysis()
    qda.fit(X_train, y_train)
    qda_train_err.append(1-qda.score(X_train, y_train))
    qda_test_err.append(1-qda.score(X_test, y_test))
    lda = LinearDiscriminantAnalysis()
    lda.fit(X_train, y_train)
    lda_train_err.append(1-lda.score(X_train, y_train))
    lda_test_err.append(1-lda.score(X_test, y_test))
```

[47]:
```
tabulate_errors(lda_train_err, lda_test_err, qda_train_err, qda_test_err)

plot_err_dist(lda_train_err, lda_test_err, qda_train_err, qda_test_err, 'for␣
↪linear model')
```

```
Model/Data          Min       p25     Median       Mean       p75       Max
Variance of error
------------  --------  --------  --------  --------  --------  --------
--------------------
LDA train      0.221429  0.262857  0.274286  0.27384   0.285714  0.327143
0.0169038
LDA test       0.196667  0.26      0.276667  0.276677  0.293333  0.366667
0.0247173
QDA train      0.217143  0.262857  0.272857  0.273083  0.284286  0.325714
0.0167419
QDA test       0.193333  0.26      0.276667  0.276693  0.293333  0.363333
0.024713
```



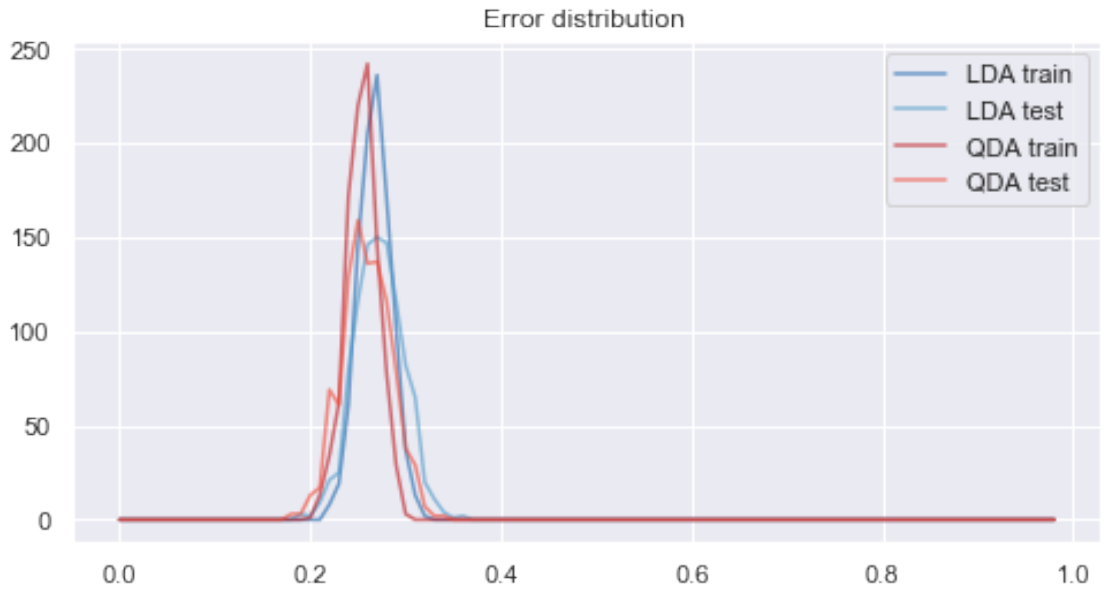
For the linear model, we see that LDA and QDA both perform about as well as each other. This is to be expected, as the quadratic fit of QDA would include the general case of linear behavior. One would also expect that as the ground truth is linear, QDA (which assumes quadratic dependence in addition to linear dependence) might overfit the model. However, we do not significantly see this - the error rates and distributions are almost consistent (within <1% of each other for each quartile of the test sets). One slight indication of overfitting could be that the QDA model's lowest training error is a whole 1.6% less than that of LDA. The small difference in overfitting could potentially be because the quadratic terms are trained to be almost negligible.

Ultimately, one would expect LDA to perform better on a linear decision boundary. This is because it excludes quadratic terms that may be cause noisiness or overfitting.

[52]:
```
qda2_train_err = []
qda2_test_err = []
lda2_train_err = []
lda2_test_err = []

for r in range(1000):
    Xs, Y = get_XY2(1000)
    X_train, X_test, y_train, y_test = train_test_split(Xs, Y, train_size=0.7,␣
 ↪test_size=0.3)
    qda = QuadraticDiscriminantAnalysis()
    qda.fit(X_train, y_train)
    qda2_train_err.append(1-qda.score(X_train, y_train))
    qda2_test_err.append(1-qda.score(X_test, y_test))
    lda = LinearDiscriminantAnalysis()
    lda.fit(X_train, y_train)
    lda2_train_err.append(1-lda.score(X_train, y_train))
    lda2_test_err.append(1-lda.score(X_test, y_test))
```

[53]:
```
tabulate_errors(lda2_train_err, lda2_test_err, qda2_train_err, qda2_test_err)

plot_err_dist(lda2_train_err, lda2_test_err, qda2_train_err, qda2_test_err)
```

```
Model/Data         Min       p25    Median      Mean       p75       Max
Variance of error
-----------  --------  --------  --------  --------  --------  --------
-------------------
LDA train     0.221429  0.26      0.272857  0.271944  0.282857  0.321429
0.0169891
LDA test      0.196667  0.256667  0.273333  0.27577   0.293333  0.366667
0.0256897
QDA train     0.208571  0.247143  0.258571  0.258857  0.27      0.302857
0.0163736
QDA test      0.186667  0.243333  0.26      0.26272   0.28      0.35
0.025197
```

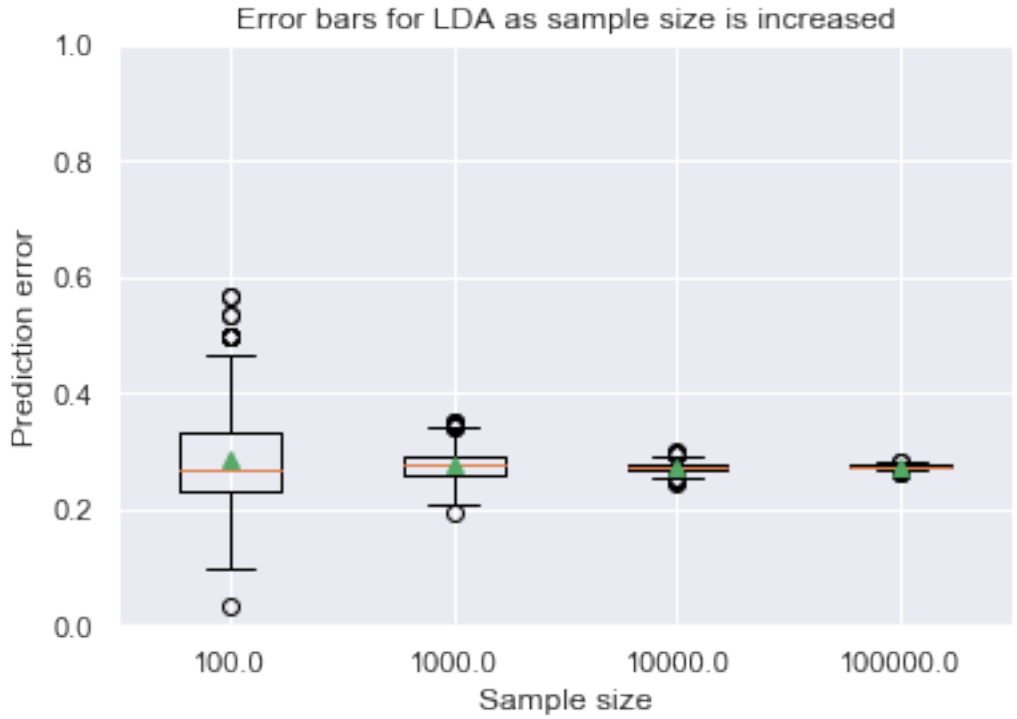


Fitting LDA and QDA models on a quadratic decision boundary exposes the limitations of a linear model on learning a non-linear relationship. The QDA model has a mean test error around 1.3% less than the LDA model, and it's best performance (minimum error) is also 1% better than that of LDA. The variance in the error is also about 0.05% less than that of LDA, which implies that QDA is not overfitting.

QDA is thus the better option for modeling a non-linear relationship.

[60]:
```
lda_tes, qda_tes = [], []
data_sizes = [1e02, 1e03, 1e04, 1e05]
for n in data_sizes:
    qda3_tr, qda3_te = [], []
    lda3_tr, lda3_te = [], []
    for r in range(1000):
        Xs, Y = get_XY2(n)
        X_train, X_test, y_train, y_test = train_test_split(Xs, Y, train_size=0.
↪7, test_size=0.3)
        qda = QuadraticDiscriminantAnalysis()
        qda.fit(X_train, y_train)
        #qda3_tr.append(1-qda.score(X_train, y_train))
        qda3_te.append(1-qda.score(X_test, y_test))
        lda = LinearDiscriminantAnalysis()
        lda.fit(X_train, y_train)
        #lda3_tr.append(1-lda.score(X_train, y_train))
        lda3_te.append(1-lda.score(X_test, y_test))
    lda_tes.append(lda3_te)
    qda_tes.append(qda3_te)
```

[65]:
```
plt.boxplot(lda_tes, vert=True, showmeans=True, labels=data_sizes)
plt.xlabel("Sample size")
plt.ylabel("Prediction error")
plt.title("Error bars for LDA as sample size is increased")
plt.ylim(0, 1)
plt.show()
```



[66]:
```
plt.boxplot(qda_tes, vert=True, showmeans=True, labels=data_sizes)
plt.xlabel("Sample size")
plt.ylabel("Prediction error")
plt.title("Error bars for QDA as sample size is increased")
plt.ylim(0, 1)
plt.show()
```

[87]:
```
fig, axs = plt.subplots(1, 4, figsize=(12, 6), sharey=True)
for i in range(4):
    axs[i].boxplot([lda_tes[i], qda_tes[i]], vert=True, showmeans=True,␣
 ↪labels=['LDA', 'QDA'])
    axs[i].set_ylabel("Prediction error")
    axs[i].set_title(f"n={int(data_sizes[i])}")
    axs[i].set_yticks(np.arange(0, 1, 0.1))
    axs[i].set_ylim(0, 1)
```

We would expect QDA to perform better than LDA on learning a non-linear relationship. As the amount of data increases, we would expect QDA to continue to perform better than LDA, though LDA may begin to perform marginally better and more data usually leads to higher accuracy. We would also expect the 'generality' of a quadratic model to lead to higher variance for QDA with low n, but much lower variance as it gets more data. The same would be expected of LDA.

As can be seen in the comparative summary above, QDA consistently performs better than LDA on non-linear decision boundaries. For both models, the average error, as well as the standard deviation of the test error go down as more data is fed to them. The standard deviation drops sharply for both models with n=10e5 - but QDA is still higher in accuracy by about 2%.

It is worth noting that for the smaller sample sizes, QDA has higher variance than LDA. This can be interpreted as the balance between having a more generalized model (quadratic rather than linear) and matching the training data. This tradeoff, as expected, becomes much less significant with larger amounts of training data. With more training examples QDA algorithm is now able to learn the exact nature of the relationship, and the high variance that came from its previous generalizability gives way to accurate predictions.

0.3 3. Modeling voter turnout

An important question in American politics is why do some people participate in the political process, while others do not? Participation has a direct impact on outcomes – if you fail to participate in politics, the government and political officials are less likely to respond to your concerns. Typical explanations focus on a resource model of participation – individuals with greater resources, such as time, money, and civic skills, are more likely to participate in politics. One area of importance is understanding voter turnout, or why people participate in elections. Using the resource model of participation as a guide, we can develop several expectations. First, women, who more frequently are the primary caregiver for children and earn a lower income, are less likely to participate in elections than men. Second, older Americans, who typically have more time and higher incomes available to participate in politics, should be more likely to participate in elections

than younger Americans. Finally, individuals with more years of education, who are generally more interested in politics and understand the value and benefits of participating in politics, are more likely to participate in elections than individuals with fewer years of education. While these explanations have been repeatedly tested by political scientists, an emerging theory assesses an individual's mental health and its effect on political participation. Depression increases feelings of hopelessness and political efficacy, so depressed individuals will have less desire to participate in politics. More importantly to our resource model of participation, individuals with depression suffer physical ailments such as a lack of energy, headaches, and muscle soreness which drain an individual's energy and requires time and money to receive treatment. For these reasons, we should expect that individuals with depression are less likely to participate in election than those without symptoms of depression.

The 1998 General Social Survey included several questions about the respondent's mental health. mental_health.csv reports several important variables from this survey.

- vote96 - 1 if the respondent voted in the 1996 presidential election, 0 otherwise
- mhealth_sum - index variable which assesses the respondent's mental health, ranging from 0 (an individual with no depressed mood) to 9 (an individual with the most severe depressed mood)
- age - age of the respondent
- educ - Number of years of formal education completed by the respondent
- black - 1 if the respondent is black, 0 otherwise
- female - 1 if the respondent is female, 0 if male
- married - 1 if the respondent is currently married, 0 otherwise • inc10 - Family income, in $10,000s

5. (20 points) Building several classifiers and comparing output.

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

b. Using the training set and all important predictors, estimate the following models with vote the response variable:

c. 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

vi. Using the test set, calculate the following model performance metrics: i. Error rate

vii. ROC curve(s) / Area under the curve (AUC)

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

[90]:
```
data = pd.read_csv('./mental_health.csv')
print(data.shape)
data.head()
```

```
(2832, 8)
```

[90]:

	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

[91]:
```
data = data.dropna()
print(data.shape)
```

```
(1165, 8)
```

[97]:
```
Y = data['vote96']
X = data[['mhealth_sum','age','educ','black','female','married','inc10']]
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, train_size=0.7,␣
 ↪test_size=0.3)

print(X_train.shape, Y_train.shape)
print(X_test.shape, Y_test.shape)
```

```
(815, 7) (815,)
(350, 7) (350,)
```

[99]:
```
# 1. logistic regression
from sklearn.linear_model import LogisticRegression

log_reg = LogisticRegression()
log_reg.fit(X_train,Y_train)
log_reg_error = 1 - log_reg.score(X_test, Y_test)
```

[101]:
```
# 2. LDA
lda = LinearDiscriminantAnalysis()
lda.fit(X_train, Y_train)
lda_error = 1 - lda.score(X_test, Y_test)
```

[103]:
```
# 3. QDA
qda = QuadraticDiscriminantAnalysis()
qda.fit(X_train, Y_train)
qda_error = 1 - qda.score(X_test, Y_test)
```

[104]:
```
# 4. Gaussian Naive Bayes
gnb = GaussianNB()
gnb.fit(X_train, Y_train)
gnb_error = 1 - gnb.score(X_test, Y_test)
```

[115]:
```
# 5. KNN

from sklearn.neighbors import KNeighborsClassifier

def evaluate_KNN(X_train, Y_train, X_test, Y_test, n_neighbors):
    KNN = KNeighborsClassifier(n_neighbors=n_neighbors, metric='minkowski', p=2)
    KNN.fit(X_train, Y_train)
    KNN_error = 1-KNN.score(X_test, Y_test)
    return KNN, KNN_error

knn_list = [evaluate_KNN(X_train, Y_train, X_test, Y_test, i) for i in range(1,␣
 ↪11)]
KNNs = [k[0] for k in knn_list]
KNN_errors = [k[1] for k in knn_list]
```

[149]:
```
print(tabulate([['Logistic Regression', log_reg_error], ['LDA', lda_error],
                ['QDA', qda_error],
                ['Naive Bayes', gnb_error], *[[f'KNN(n={i+1})', KNN_errors[i]]␣
 ↪for i in range(10)]],
                headers = ['Model', 'Error']))
```

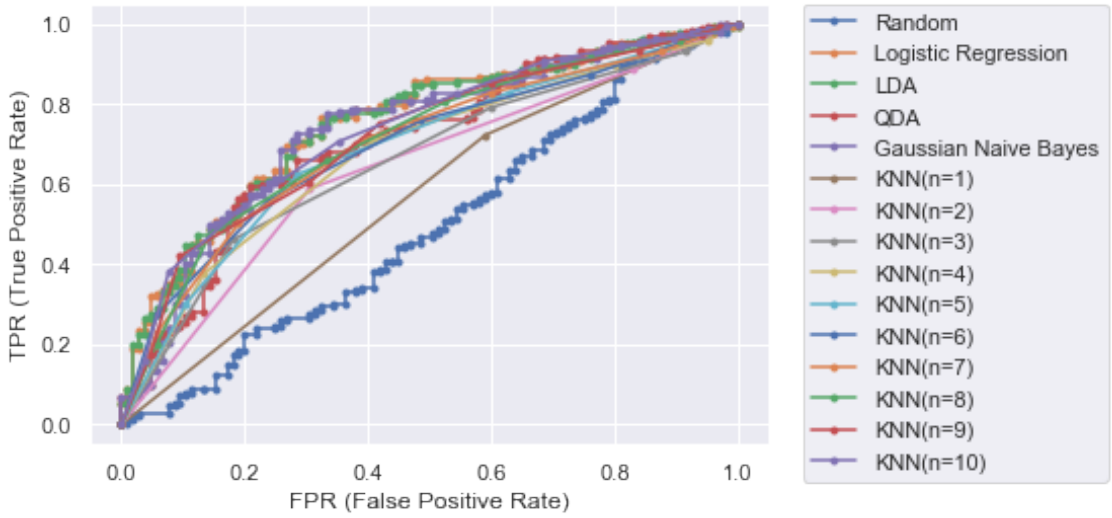
```
Model                   Error
--------------------  --------
Logistic Regression  0.285714
LDA                  0.282857
QDA                  0.294286
Naive Bayes          0.288571
KNN(n=1)             0.371429
KNN(n=2)             0.38
KNN(n=3)             0.325714
KNN(n=4)             0.325714
KNN(n=5)             0.308571
KNN(n=6)             0.314286
KNN(n=7)             0.3
KNN(n=8)             0.294286
KNN(n=9)             0.282857
KNN(n=10)            0.285714
```

[147]:
```
from sklearn.metrics import roc_auc_score, roc_curve

def plot_roc_curve(model, X_test, modelname):
    #print(modelname)
    if modelname.lower()=='random':
        Y_pred = np.random.uniform(0, 1, (X_test.shape[0],))
    else:
        Y_pred = model.predict_proba(X_test)[:, 1]
    auc_score = roc_auc_score(Y_test, Y_pred)
```

```
    model_fpr, model_tpr, model_thresholds = roc_curve(Y_test, Y_pred)
    plt.plot(model_fpr, model_tpr, marker='.', label=modelname)
    plt.xlabel('FPR (False Positive Rate)')
    plt.ylabel('TPR (True Positive Rate)')
    plt.legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.)
    return [modelname, '%.3f' % (auc_score)]

models = ['random', log_reg, lda, qda, gnb, *KNNs]
labels = ['Random', 'Logistic Regression', 'LDA', 'QDA', "Gaussian Naive Bayes",
          ␣
 ↪'KNN(n=1)','KNN(n=2)','KNN(n=3)','KNN(n=4)','KNN(n=5)','KNN(n=6)','KNN(n=7)','KNN(n=8)','KN
```

[148]:
```
count = 0
op = []
for i, model in enumerate(models):
    txt = plot_roc_curve(model, X_test, labels[i])
    op.append(txt)

print(tabulate(op, headers=['Model', 'AUC of ROC']))

plt.show()
```

```
Model                    AUC of ROC
-----------------------  ------------
Random                          0.496
Logistic Regression             0.754
LDA                             0.753
QDA                             0.71
Gaussian Naive Bayes            0.737
KNN(n=1)                        0.566
KNN(n=2)                        0.637
KNN(n=3)                        0.658
KNN(n=4)                        0.677
KNN(n=5)                        0.687
KNN(n=6)                        0.695
KNN(n=7)                        0.7
KNN(n=8)                        0.714
KNN(n=9)                        0.718
KNN(n=10)                       0.734
```



LDA and KNN(n=9) were the two classifiers with the lowest error rate (0.283). Interestingly, KNN(n=10) has a higher error than KNN(n=10), probably due to overfitting.

LDA and Logistic regression, have the highest area-under-curve values of 0.754 and 0.753 on the ROC plot. The AUC for the ROC plot is a good check for overfitting, as it accounts for false positives, and not just true positives.

Combining the best performers on these two metrics, we can identify LDA as the best performing classifier in this exercise.