4721 hw2

Yuning Ding yd2617

February 2022

1 Linear regression

Problem 1.1

The library function I use is `sklearn.linear_model.LinearRegression`.

feature	slope	intercept
lcavol	0.713	1.516
lweight	1.23	-2.006
age	0.037	0.079
lbph	0.217	2.437
svi	1.602	2.094
lcp	0.422	2.543
gleason	0.583	-1.475
pgg45	0.018	1.967

Table 1: slopes and intercepts of affine function of each feature

Problem 1.2

Here I use `matplot.pyplot.subplots` and `sns.scatterplot`.

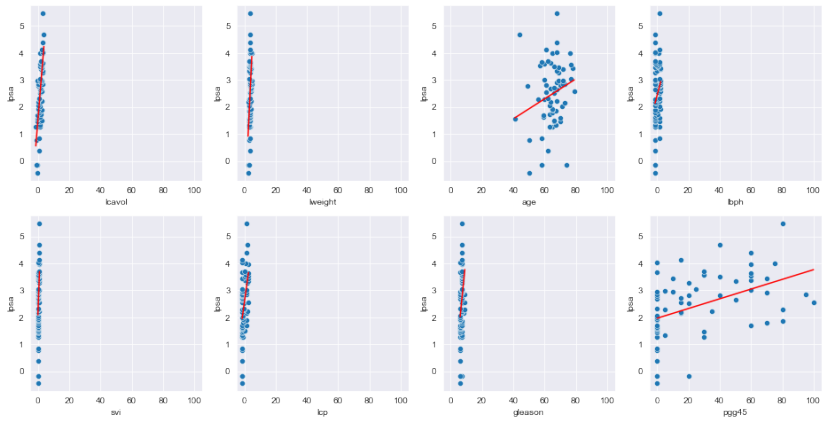


Figure 1: different features data with affine functions

Problem 1.3

I use `sklearn.preprocessing.StandardScaler` to stardardize the data.

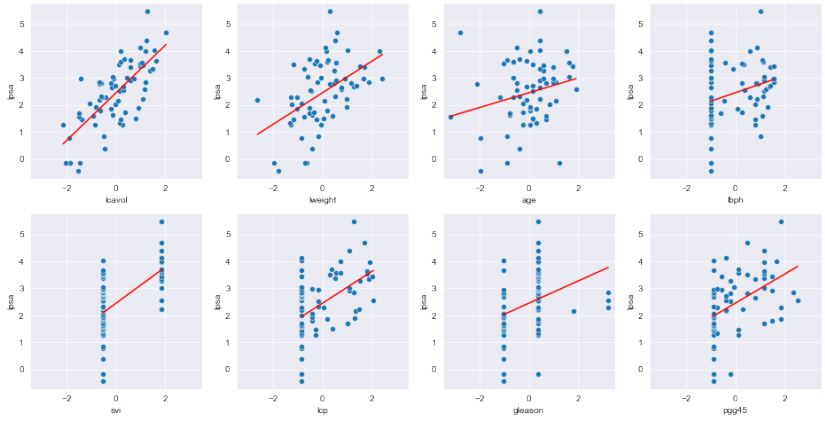


Figure 2: different features normalized data with affine functions

Problem 1.4

The coefficients are 0.577, 0.614 -0.019, 0.145, 0.737, -0.206, -0.03, 0.009. And the intercept is 0.429.

After normalization, the coefficients are 0.711, 0.29, -0.141, 0.21, 0.307 -0.287, -0.021, 0.275. The intercept is 2.452.

Problem 1.5

Assume we have n samples $\mathbf{x}^i = (x_1^i, x_2^i)$, $y^i = \frac{3}{2}x_1^i - \frac{3}{4}x_2^i$, in which $i = 1, 2, \cdots, n$. When $n \to \infty$, we can compute the MSE.

(a)

Let the linear function be $\hat{y} = w_1 x_1$. Then we have

$$\text{MSE} = \lim_{n\to\infty} \frac{1}{n} \sum_{i=1}^{n} (y^i - w_1 x_1^i)^2. \tag{1}$$

We now want to minimize the MSE. It must satisfy

$$\frac{\partial \text{MSE}}{\partial w_1} = 0. \tag{2}$$

After derivation, we have

$$\begin{aligned} w_1 &= \lim_{n\to\infty} \frac{\frac{1}{n}(\frac{3}{2}\sum_{i=1}^{n}(x_1^i)^2 - \frac{3}{4}\sum_{i=1}^{n}(x_1^i x_2^i))}{\frac{1}{n}\sum_{i=1}^{n}(x_1^i)^2} \\ &= \mathbb{E}[\frac{\frac{1}{n}(\frac{3}{2}\sum_{i=1}^{n}(x_1^i)^2 - \frac{3}{4}\sum_{i=1}^{n}(x_1^i x_2^i))}{\frac{1}{n}\sum_{i=1}^{n}(x_1^i)^2}]. \end{aligned} \tag{3}$$

Since $\sum_{i=1}^{n}(x_1^i)^2 \sim \chi_n^2$ and $\mathbb{E}[x_1 x_2] = \frac{2}{3}$, we have

$$w_1 = \frac{\frac{3}{2} - \frac{3}{4} \times \frac{2}{3}}{1} = 1. \tag{4}$$

Therefore, the linear function should be $y = x_1$.

(b)

Let the linear function be $\tilde{y} = w_2 x_2$. Similar to (a), we have

$$w_1 = \mathbb{E}[\frac{\frac{1}{n}(\frac{3}{2}\sum_{i=1}^{n}(x_1^i x_2^i) - \frac{3}{4}\sum_{i=1}^{n}(x_2^i)^2)}{\frac{1}{n}\sum_{i=1}^{n}(x_2^i)^2}], \tag{5}$$

and

$$w_2 = \frac{\frac{3}{2} \times \frac{2}{3} - \frac{3}{4}}{1} = \frac{1}{4}. \tag{6}$$

Therefore, the linear function should be $y = \frac{1}{4}x_2$.

(c)

Let the linear function be $\bar{y} = u_1 x_1 + u_2 x_2$. We now want to minimize the MSE. It must satisfy

$$\begin{aligned} \frac{\partial \text{MSE}}{\partial u_1} &= 0 \\ \frac{\partial \text{MSE}}{\partial u_2} &= 0. \end{aligned} \tag{7}$$

It can be derived as

$$\begin{aligned} \frac{\partial \text{MSE}}{\partial u_1} &= \frac{3}{2}\sum_{i=1}^{n}(x_i)^2 - u_1\sum_{i=1}^{n}(x_1^i)^2 - \frac{3}{4}\sum_{i=1}^{n}(x_1^i x_2^i) - u_2\sum_{i=1}^{n}(x_1^i x_2^i) = 0 \\ \frac{\partial \text{MSE}}{\partial u_2} &= \frac{3}{2}\sum_{i=1}^{n}(x_1^i x_2^i) - u_1\sum_{i=1}^{n}(x_1^i x_2^i) - \frac{3}{4}\sum_{i=1}^{n}(x_2^i)^2 - u_2\sum_{i=1}^{n}(x_2^i)^2 = 0, \end{aligned} \tag{8}$$

in which $n \to \infty$.

And we have

$$\begin{aligned} u_1 &= \frac{3}{2} \\ u_2 &= -\frac{3}{4}. \end{aligned} \tag{9}$$

Hence, the linear function would be $y = \frac{3}{2}x_1 - \frac{3}{4}x_2$.

Problem 1.6

I use `sklearn.metrics.mean_squared_error` to compute the mse.

feature	train mse	test mse
lcavol	0.665	0.48
lweight	1.099	1.013
age	1.363	1.128
lbph	1.338	1.177
svi	0.991	0.688
lcp	1.093	0.545
gleason	1.269	0.851
pgg45	1.149	0.954

Table 2: train mse and test mse of affine functions in problem 1.1

The train mse in problem 1.4 is 0.439, and the test mse is 0.521.

2 Model averaging

2.1 Problem 2.1

Since A and B are independent and identically distributed random variables, we have

$$\begin{aligned}\mathbb{E}[(A-B)^2] &= \mathbb{E}[A^2] - 2\mathbb{E}[AB] + \mathbb{E}[B^2] \\ &= \sigma^2 + \mathbb{E}[A]^2 - 2\mathbb{E}[AB] + \sigma^2 + \mathbb{E}[B]^2 \\ &= 2\sigma^2 + (\mathbb{E}[A] - \mathbb{E}[B])^2 \\ &= 2\sigma^2\end{aligned} \tag{10}$$

2.2 Problem 2.2

We define a random variable T whose distribution is uniformly over $\{1, 2, \cdots, M\}$, and then random variable $f_T(\mathbf{x})$ would be $f_i(\mathbf{x})$, in which $i = \{1, 2, \cdots, M\}$, with probability of $\frac{1}{M}$.

We have its expectation

$$\mathbb{E}(f_T(\mathbf{x})) = f_{\text{avg}}(\mathbf{x}) = \frac{1}{M}\sum_{t=1}^{M} f_t(\mathbf{x}). \tag{11}$$

For variance, we define f'_T and f_T are independent and identically distributed random variables. Then we use the relationship identified in Problem 2.1

$$\begin{aligned}\frac{1}{M^2}\sum_{s=1}^{M}\sum_{t=1}^{M}(f_s(\mathbf{x}) - f_t(\mathbf{x}))^2 &= \mathbb{E}[(f_T(\mathbf{x}) - f'_T(\mathbf{x}))^2] \\ &= 2\text{Var}(f_T(\mathbf{x})) + (\mathbb{E}[f_T(\mathbf{x})] - \mathbb{E}[f'_T(\mathbf{x})])^2 \\ &= 2\text{Var}(f_T(\mathbf{x}))\end{aligned} \tag{12}$$

Therefore

$$\text{Var}(f_T(\mathbf{x})) = \frac{1}{2M^2}\sum_{s=1}^{M}\sum_{t=1}^{M}(f_s(\mathbf{x}) - f_t(\mathbf{x}))^2 \tag{13}$$

Hence, Equation (2) could be rewritten as

$$(\mathbb{E}[f_T(\mathbf{x})] - y)^2 = \mathbb{E}[(f_T(\mathbf{x}) - y)^2] - Var(f_T(\mathbf{x})). \tag{14}$$

$$\mathbb{E}[(f_T(\mathbf{x}) - y)^2] = (\mathbb{E}[f_T(\mathbf{x})] - y)^2 + Var(f_T(\mathbf{x})). \tag{15}$$

3 Bagging regression and classification trees

3.1 Problem 3.1

The mean squared error is 0.371.

3.2 Problem 3.2

The mean squared error is 0.379.

3.3 Problem 3.3

In the figure, we can see that the slope of the large regression tree is smaller than zero, while the slope of the ensemble of 100 (small) regression trees is greater than zero.

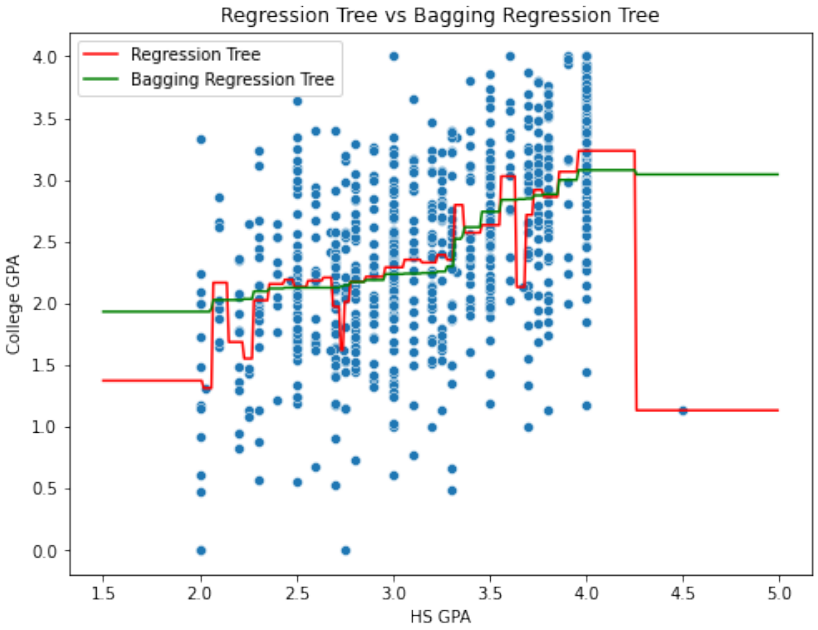


Figure 3: different models in dartmouth data

3.4 Problem 3.4

For a single decision tree, the training error rate is 0.00028 and the test error rate is 0.096.

For the ensemble of classification tree, the training error rate is 0.00417 and the test error rate is 0.065.

3.5 Problem 3.5

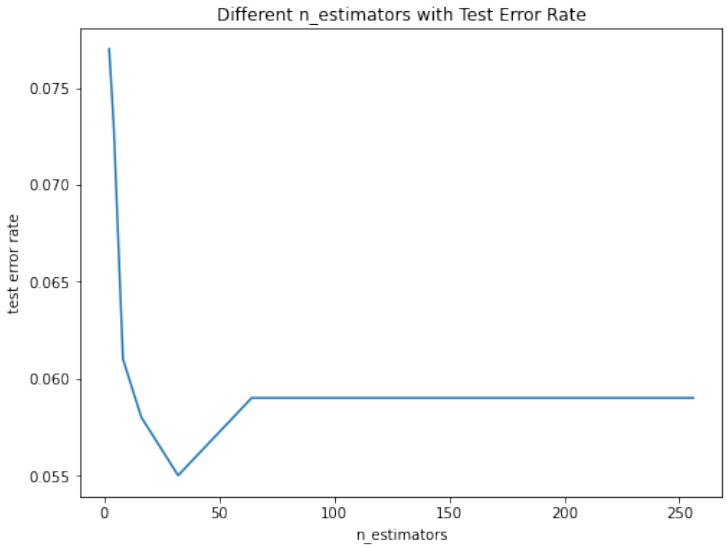


Figure 4: different models in dartmouth data

4 Optimal classifiers

4.1 Problem 4.1

The $f^\star$ is

$$f^\star = \begin{cases} 1, & 0.2 < x < 0.8 \\ 0, & others \end{cases} \tag{16}$$

Since X follows the uniformly distribution, we have

$$\Pr(x \leq 0.2) = 0.2, \quad \Pr(0.2 < x < 0.8) = 0.6, \quad \Pr(x \geq 0.8) = 0.2 \tag{17}$$

Therefore, the value of $\mathrm{err}(f^\star)$ is

$$\begin{aligned} \mathrm{err}(f^\star) &= \Pr(f(X) \neq Y) \\ &= 0.2 \times 0.2 + 0.4 \times 0.6 + 0.3 \times 0.2 \\ &= 0.34 \end{aligned} \tag{18}$$

4.2 Problem 4.2

$$\mathrm{err}(f_{T_1}) = 0.46, \quad \mathrm{err}(f_{T_2}) = 0.5, \quad \mathrm{err}(f_{T_3}) = 0.5. \tag{19}$$

4.3 Problem 4.3

I draw a error rate with different stumps plot and find that this decision stump $T^\star$ has the smallest error rate.

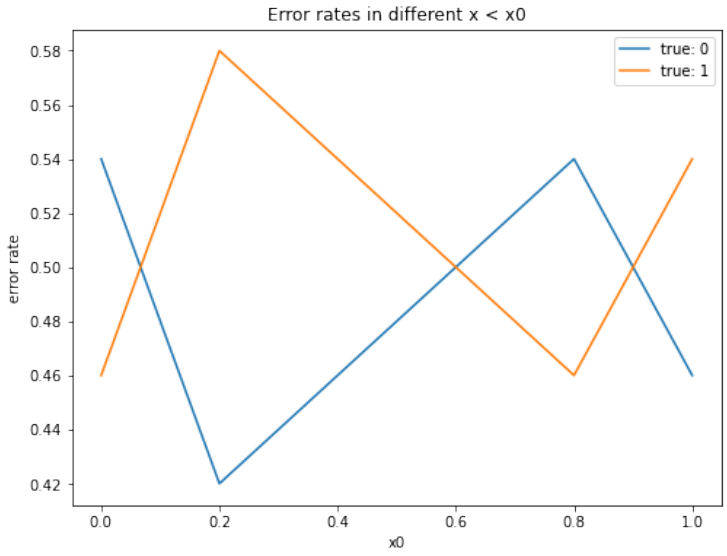


Figure 5: different error rate with different x_0

We can also calculate the decision stump T^* with the smallest error rate.
Case1: $x \leq x_0$ return 1.

$$eff = \begin{cases} 0.54 - 0.6x_0, & x_0 \leq 0.2 \\ 0.38 + 0.2x_0, & 0.2 < x_0 \leq 0.8 \\ 0.86 - 0.4x_0, & x_0 > 0.8 \end{cases} \tag{20}$$

Case2: $x \leq x_0$ return 0.

$$eff = \begin{cases} 0.46 + 0.6x_0, & x_0 \leq 0.2 \\ 0.62 - 0.2x_0, & 0.2 < x_0 \leq 0.8 \\ 0.14 + 0.4x_0, & x_0 > 0.8 \end{cases} \tag{21}$$

Hence T^* is

$$T^\star = \text{“if } x \leq 0.2\text{, then return 0, else return 1”}. \tag{22}$$

The error rate is 0.42.

4.4 Problem 4.4

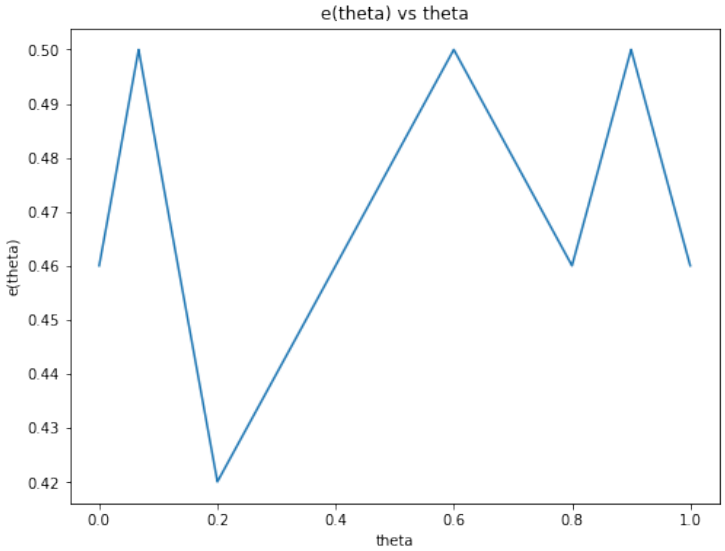


Figure 6: different e(θ) with different θ

4.5 Problem 4.5

The left graph is the threshold

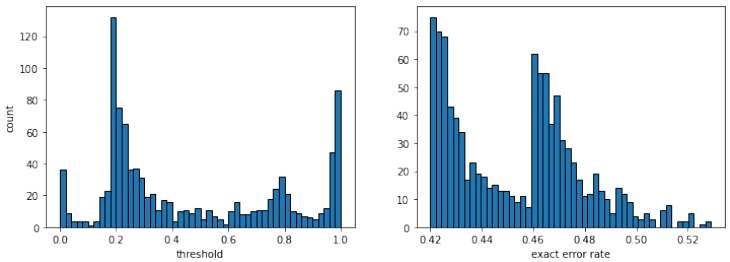


Figure 7: Threshold $\hat{\theta}$, error rate err($f_{\hat{T}}$) histograms

problem1

In [1]:
```
from csv import reader
import numpy as np
```

In [2]:
```
with open('prostate-train.csv','r') as f:
    data = [datum for datum in reader(f, delimiter = ",")]

data_array = np.asarray(data)
headers = data_array[0] # first row is headers
training_data = np.asarray(data_array[1:], dtype=np.float64)
```

In [3]:
```
with open('prostate-test.csv','r') as f:
    data = [datum for datum in reader(f, delimiter = ",")]

test_data = np.asarray(np.asarray(data)[1:], dtype=np.float64)
```

In [4]:
```
print(headers)
```

```
['lcavol' 'lweight' 'age' 'lbph' 'svi' 'lcp' 'gleason' 'pgg45' 'lpsa']
```

In [5]:
```
print(training_data[:10])
```

```
[[-0.5798185   2.769459   50.         -1.38629436   0.         -1.38629436
   6.          0.         -0.4307829 ]
 [-0.99425227  3.319626   58.         -1.38629436   0.         -1.38629436
   6.          0.         -0.1625189 ]
 [-0.51082562  2.691243   74.         -1.38629436   0.         -1.38629436
   7.         20.         -0.1625189 ]
 [-1.2039728   3.282789   58.         -1.38629436   0.         -1.38629436
   6.          0.         -0.1625189 ]
 [ 0.75141609  3.432373   62.         -1.38629436   0.         -1.38629436
   6.          0.          0.3715636 ]
 [-1.04982212  3.228826   50.         -1.38629436   0.         -1.38629436
   6.          0.          0.7654678 ]
 [ 0.69314718  3.539509   58.          1.53686722   0.         -1.38629436
   6.          0.          0.8544153 ]
 [ 0.25464222  3.604138   65.         -1.38629436   0.         -1.38629436
   6.          0.          1.2669476 ]
 [-1.34707365  3.598681   63.          1.2669476    0.         -1.38629436
   6.          0.          1.2669476 ]
 [ 1.61342993  3.022861   63.         -1.38629436   0.         -0.597837
   7.         30.          1.2669476 ]]
```

import packages

In [6]:
```
# import necessary packages
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error
import seaborn as sns
import matplotlib.pyplot as plt
```

```
sns.set_style('darkgrid')
%matplotlib inline
```

In [7]:
```
# use pandas to modify the training data
df = pd.DataFrame(training_data)
df.columns = headers

X_train = df.iloc[:,:8]
y_train = df.iloc[:,8]
```

In [8]:
```
X_test = test_data[:,:8]
y_test = test_data[:,8]
```

problem 1.4

In [9]:
```
# linear regression
lr = LinearRegression()
lr.fit(X_train, y_train)
print(f'coefficient: {np.round(lr.coef_, 3)}')
print(f'intercept: {np.round(lr.intercept_, 3)}')
```

```
coefficient: [ 0.577  0.614 -0.019  0.145  0.737 -0.206 -0.03   0.009]
intercept: 0.429
```

problem 1.6

In [10]:
```
y_train_pred = lr.predict(X_train)
y_test_pred = lr.predict(X_test)
print(f'train mse: {round(mean_squared_error(y_train, y_train_pred),3)}')
print(f'test mse: {round(mean_squared_error(y_test, y_test_pred),3)}')
```

```
train mse: 0.439
test mse: 0.521
```

problem 1.1 & 1.6

In [11]:
```
# linear regression
lr = LinearRegression()
coefs = []
intercepts = []
train_mse = []
test_mse = []
for i in range(8):
    lr.fit(np.array(df.iloc[:,i]).reshape(-1, 1), y_train)
    y_train_pred = lr.predict(np.array(df.iloc[:,i]).reshape(-1, 1))
    y_test_pred = lr.predict(X_test[:,i].reshape(-1, 1))
    coefs.append(np.round(lr.coef_, 3)[0])
    intercepts.append(np.round(lr.intercept_, 3))
    train_mse.append(round(mean_squared_error(y_train, y_train_pred),3))
    test_mse.append(round(mean_squared_error(y_test, y_test_pred),3))

print(f'slope: {coefs}')
print(f'intercept: {intercepts}')
print(f'train mse: {train_mse}')
print(f'test mse: {test_mse}')
```

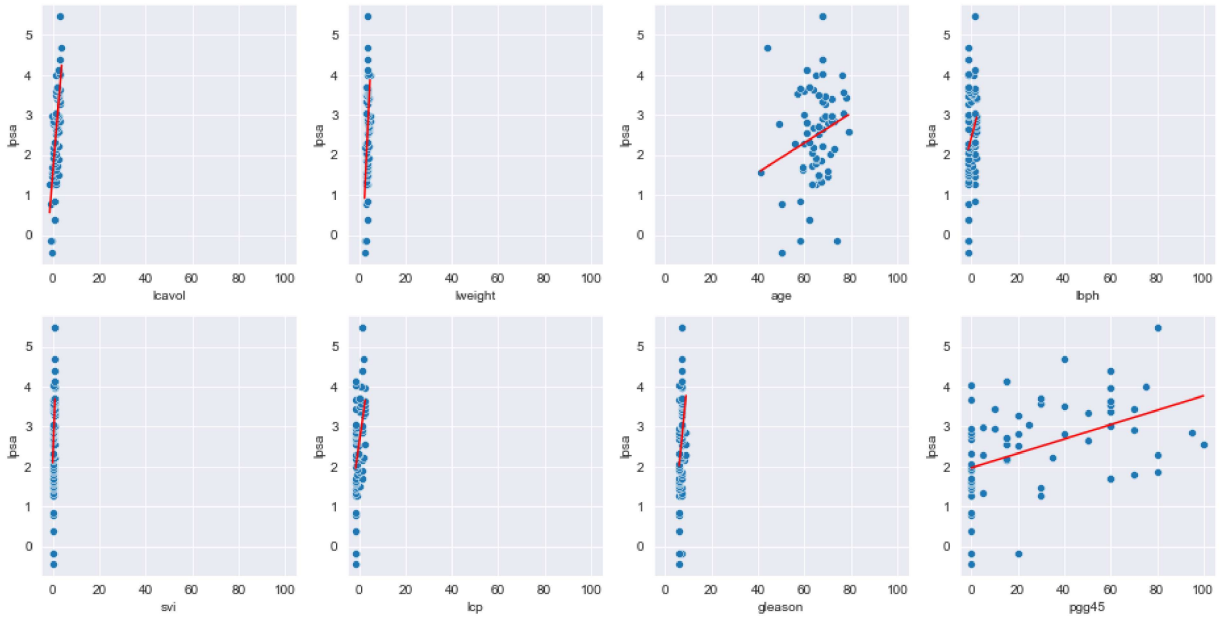
```
slope: [0.713, 1.23, 0.037, 0.217, 1.602, 0.422, 0.583, 0.018]
```

```
intercept: [1.516, -2.006, 0.079, 2.437, 2.094, 2.543, -1.475, 1.967]
train mse: [0.665, 1.099, 1.363, 1.338, 0.991, 1.093, 1.269, 1.149]
test mse: [0.48, 1.013, 1.128, 1.177, 0.688, 0.545, 0.851, 0.954]
```

problem 1.2

In [12]:

```
fig,ax = plt.subplots(2,4,figsize=(16,8))
for i in range(8):
    x_predict = [df.iloc[:,i].min(),df.iloc[:,i].max()]
    y_hat = np.array(x_predict) * coefs[i] + intercepts[i]

    plt_i = i//4
    plt_j = i%4
    ax[plt_i][plt_j] = sns.scatterplot(x=df.iloc[:,i],y=df.lpsa, ax=ax[plt_i][plt_j])
    ax[plt_i][plt_j].plot(x_predict,y_hat, color = 'red');
    ax[plt_i][plt_j].set_xlim([-5,105])
```



problem 1.3

In [13]:

```
# normalize training data
scaler = StandardScaler()
scaler.fit(X_train)
df.iloc[:,:8] = scaler.transform(X_train)
X_train = df.iloc[:,:8]
```

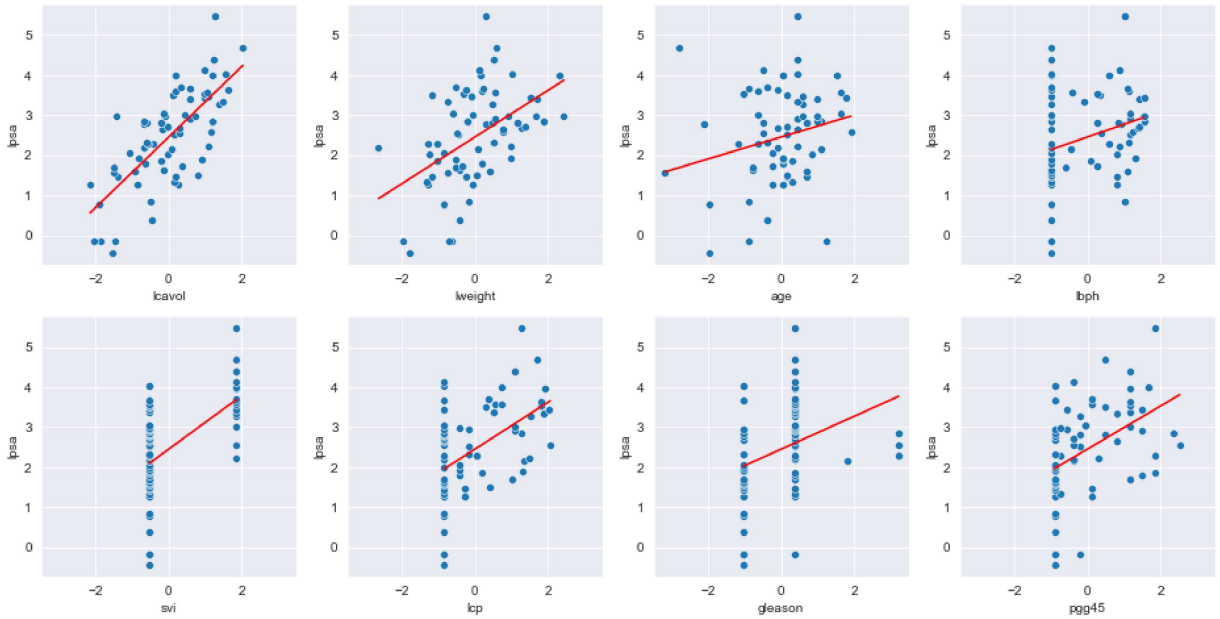
In [14]:

```
# linear regression
lr = LinearRegression()
coefs = []
intercepts = []
for i in range(8):
    lr.fit(np.array(df.iloc[:,i]).reshape(-1, 1), y_train)
    coefs.append(np.round(lr.coef_, 3)[0])
    intercepts.append(np.round(lr.intercept_, 3))
print(f'coefficient: {coefs}')
print(f'intercept: {intercepts}')
```

```
coefficient: [0.879, 0.582, 0.273, 0.315, 0.668, 0.586, 0.41, 0.537]
intercept: [2.452, 2.452, 2.452, 2.452, 2.452, 2.452, 2.452, 2.452]
```

In [15]:

```
fig,ax = plt.subplots(2,4,figsize=(16,8))
for i in range(8):
    x_predict = [df.iloc[:,i].min(),df.iloc[:,i].max()]
    y_hat = np.array(x_predict) * coefs[i] + intercepts[i]

    plt_i = i//4
    plt_j = i%4
    ax[plt_i][plt_j] = sns.scatterplot(x=df.iloc[:,i],y=df.lpsa, ax=ax[plt_i][plt_j])
    ax[plt_i][plt_j].plot(x_predict,y_hat, color = 'red');
    ax[plt_i][plt_j].set_xlim([-3.5,3.5])
```



problem 1.4

In [16]:
```
# linear regression
lr = LinearRegression()
lr.fit(X_train, y_train)
print(f'coefficient: {np.round(lr.coef_, 3)}')
print(f'intercept: {np.round(lr.intercept_, 3)}')
```

```
coefficient: [ 0.711  0.29  -0.141  0.21  0.307 -0.287 -0.021  0.275]
intercept: 2.452
```

problem 1.6

In [17]:
```
y_train_pred = lr.predict(X_train)
y_test_pred = lr.predict(X_test)
print(f'train mse: {round(mean_squared_error(y_train, y_train_pred),3)}')
print(f'test mse: {round(mean_squared_error(y_test, y_test_pred),3)}')
```

```
train mse: 0.439
test mse: 44.544
```

In [18]:
```
y_test
```

Out[18]:
```
array([0.7654678, 1.047319 , 1.047319 , 1.3987169, 1.6582281, 1.7316555,
       1.7664417, 1.8164521, 2.008214 , 2.0215476, 2.0856721, 2.3075726,
       2.3749058, 2.5687881, 2.5915164, 2.5915164, 2.6844403, 2.6912431,
       2.7047113, 2.7880929, 2.8535925, 2.8820035, 2.8820035, 2.8875901,
       3.0563569, 3.0750055, 3.5130369, 3.5709402, 5.1431245, 5.5829322])
```

In [19]:
```
y_test_pred
```

Out[19]:
```
array([-4.66690213, -3.73999153, -5.37856576, -2.69198242,  1.72822411,
       -5.36111924, -4.89581756, -0.04888502, -4.87751849, -4.25010259,
       -2.39002081, -2.36001105, -2.80453772,  6.18025336, -1.39269605,
       -5.15340063, 14.24116887,  6.36317372, -1.29453832, -2.15176386,
        7.00536451, 12.19438006, -4.09467762,  1.73507964, -0.02330649,
       20.85989464,  9.84557253, 15.08964365,  0.06527586,  1.38138473])
```

In [20]:
```
y_train_pred
```

Out[20]:
```
array([0.75353858, 0.70040498, 0.44906559, 0.55687338, 1.700083  ,
       0.76462172, 2.23169054, 1.46213549, 0.95764547, 2.01839575,
       1.77468627, 1.85140133, 1.39943465, 2.28081126, 1.24066142,
       1.90227158, 1.97740214, 1.06025544, 2.56035833, 2.19369936,
       2.25005272, 1.94428706, 2.14202486, 1.97291915, 0.98846838,
       1.58796666, 0.95575097, 3.86245665, 1.68493079, 2.15305679,
       2.12067165, 2.45045739, 2.62328973, 4.09638   , 2.57075566,
       3.17981669, 3.06252822, 2.39221612, 2.30957586, 2.89233974,
       2.52992457, 2.93714431, 3.12648065, 3.12713435, 1.4759386 ,
       3.02721263, 3.19521036, 2.25589475, 3.33590171, 3.44930283,
       3.50837247, 3.59520175, 3.30376434, 2.11377357, 2.87284884,
       3.48312979, 2.62355283, 3.6956864 , 3.13189774, 2.87004781,
       4.09922459, 3.36357643, 3.43564731, 4.14249256, 3.77414281,
       4.44984337, 4.30839196])
```

In [21]:
```
np.array(y_train)
```

Out[21]:
```
array([-0.4307829, -0.1625189, -0.1625189, -0.1625189,  0.3715636,
        0.7654678,  0.8544153,  1.2669476,  1.2669476,  1.2669476,
        1.3480731,  1.446919 ,  1.4701758,  1.4929041,  1.5581446,
        1.5993876,  1.6389967,  1.6956156,  1.7137979,  1.8000583,
        1.8484548,  1.8946169,  1.9242487,  2.008214 ,  2.0476928,
        2.1575593,  2.1916535,  2.2137539,  2.2772673,  2.2975726,
        2.3272777,  2.5217206,  2.5533438,  2.5687881,  2.6567569,
        2.677591 ,  2.7180005,  2.7942279,  2.8063861,  2.8124102,
        2.8419982,  2.8535925,  2.9204698,  2.9626924,  2.9626924,
        2.9729753,  3.0130809,  3.0373539,  3.2752562,  3.3375474,
        3.3928291,  3.4355988,  3.4578927,  3.5160131,  3.5307626,
        3.5652984,  3.5876769,  3.6309855,  3.6800909,  3.7123518,
        3.9843437,  3.993603 ,  4.029806 ,  4.1295508,  4.3851468,
        4.6844434,  5.477509 ])
```

In [ ]:
```

```

In [1]:
```
from csv import reader
import numpy as np
```

In [2]:
```
with open('dartmouth-train.csv','r') as f:
    data = [datum for datum in reader(f, delimiter = ",")]

data_array = np.asarray(data)
headers = data_array[0] # first row is headers
training_data = np.asarray(data_array[1:], dtype=np.float64)
```

In [3]:
```
with open('dartmouth-test.csv','r') as f:
    data = [datum for datum in reader(f, delimiter = ",")]

test_data = np.asarray(np.asarray(data)[1:], dtype=np.float64)
```

In [4]:
```
print(headers)
```

```
['SAT Score' 'HS GPA' 'College GPA']
```

In [5]:
```
print(training_data[:10])
```

```
[[1080.    3.3   3.24]
 [1220.    2.8   2.83]
 [1210.    3.75  3.18]
 [1150.    2.6   1.7 ]
 [1030.    3.8   2.66]
 [1030.    2.8   2.85]
 [1080.    3.5   2.63]
 [1030.    2.8   3.29]
 [ 890.    2.75  1.44]
 [ 790.    3.25  1.54]]
```

import necessary packages

In [6]:
```
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_squared_error
from sklearn.ensemble import BaggingRegressor
import matplotlib.pyplot as plt
import seaborn as sns
```

problem 3.1

In [7]:
```
DTR = DecisionTreeRegressor(max_leaf_nodes=300,
                            random_state=42)
DTR.fit(X=training_data[:,1].reshape(-1,1), y=training_data[:,2])
y_predict = DTR.predict(X=test_data[:,1].reshape(-1,1))
print(round(mean_squared_error(test_data[:,2], y_predict),3))
```

```
0.371
```

problem 3.2

In [8]:
```

```

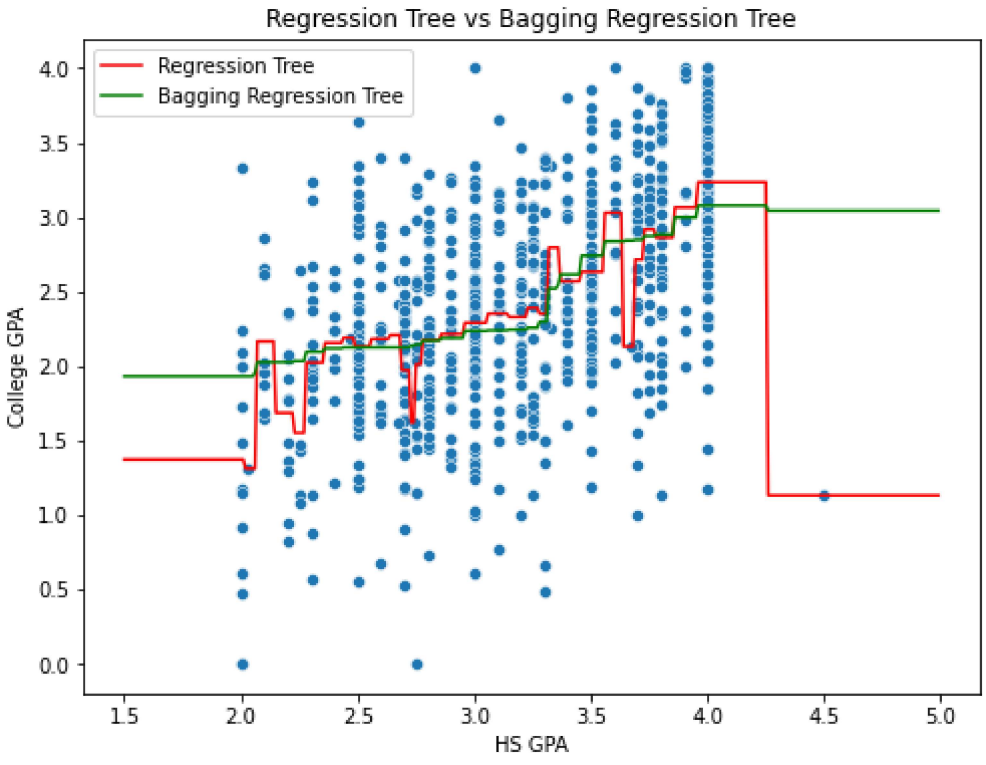
```
dst = DecisionTreeRegressor(max_leaf_nodes=3)
BR = BaggingRegressor(base_estimator=dst,
                      n_estimators=100,
                      random_state=42)
BR.fit(X=training_data[:,1].reshape(-1,1), y=training_data[:,2])
y_predict = BR.predict(X=test_data[:,1].reshape(-1,1))
print(round(mean_squared_error(test_data[:,2], y_predict),3))
```

0.379

problem 3.3

In [10]:

```
fig,ax = plt.subplots(1,1,figsize=(8,6))
sns.scatterplot(x=training_data[:,1], y=training_data[:,2]);
x_predict = np.arange(1.5, 5, 0.01)[:, np.newaxis]
y_predict_DTR = DTR.predict(x_predict.reshape(-1,1))
y_predict_BR = BR.predict(x_predict.reshape(-1,1))

line1, = ax.plot(x_predict, y_predict_DTR, color='red');
line2, = ax.plot(x_predict, y_predict_BR, color='green');
line1.set_label('Regression Tree')
line2.set_label('Bagging Regression Tree')
ax.set_xlabel('HS GPA')
ax.set_ylabel('College GPA')
ax.set_title('Regression Tree vs Bagging Regression Tree');
ax.legend();
```



In [ ]:

spambase

In [1]:
```
import pickle
with open('spam.pickle', 'rb') as f:
    spam = pickle.load(f)
```

In [2]:
```
spam['train_data'].shape
```

Out[2]:
```
(3601, 57)
```

In [3]:
```
spam['train_labels'].shape
```

Out[3]:
```
(3601,)
```

In [4]:
```
spam['test_data'].shape
```

Out[4]:
```
(1000, 57)
```

In [5]:
```
spam['test_labels'].shape
```

Out[5]:
```
(1000,)
```

In [6]:
```
spam['feature_names']
```

Out[6]:
```
['word_freq_make',
 'word_freq_address',
 'word_freq_all',
 'word_freq_3d',
 'word_freq_our',
 'word_freq_over',
 'word_freq_remove',
 'word_freq_internet',
 'word_freq_order',
 'word_freq_mail',
 'word_freq_receive',
 'word_freq_will',
 'word_freq_people',
 'word_freq_report',
 'word_freq_addresses',
 'word_freq_free',
 'word_freq_business',
 'word_freq_email',
 'word_freq_you',
 'word_freq_credit',
 'word_freq_your',
 'word_freq_font',
 'word_freq_000',
 'word_freq_money',
 'word_freq_hp',
 'word_freq_hpl',
 'word_freq_george',
 'word_freq_650',
 'word_freq_lab',
```

```
'word_freq_labs',
'word_freq_telnet',
'word_freq_857',
'word_freq_data',
'word_freq_415',
'word_freq_85',
'word_freq_technology',
'word_freq_1999',
'word_freq_parts',
'word_freq_pm',
'word_freq_direct',
'word_freq_cs',
'word_freq_meeting',
'word_freq_original',
'word_freq_project',
'word_freq_re',
'word_freq_edu',
'word_freq_table',
'word_freq_conference',
'char_freq_;',
'char_freq_(',
'char_freq_[',
'char_freq_!',
'char_freq_$',
'char_freq_#',
'capital_run_length_average',
'capital_run_length_longest',
'capital_run_length_total']
```

In [7]:
```
spam['class_names']
```

Out[7]:
```
['not spam', 'spam']
```

import necessary packages

In [8]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import BaggingClassifier
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
```

problem 3.4

In [9]:
```
DTC = DecisionTreeClassifier(random_state=42)
DTC.fit(X=spam['train_data'], y=spam['train_labels'])
print(round(1-DTC.score(X=spam['train_data'], y=spam['train_labels']),5))
print(DTC.score(X=spam['test_data'], y=spam['test_labels']))
```

```
0.00028
0.904
```

In [10]:
```
from sklearn.metrics import classification_report
print('Performance on training data')
print(classification_report(spam['train_labels'],
    DTC.predict(spam['train_data']),
    target_names=spam['class_names']))

print('Performance on test data')
```

```
print(classification_report(spam['test_labels'],
    DTC.predict(spam['test_data']),
    target_names=spam['class_names']))
```

```
Performance on training data
              precision    recall  f1-score   support

    not spam       1.00      1.00      1.00      2186
        spam       1.00      1.00      1.00      1415

    accuracy                           1.00      3601
   macro avg       1.00      1.00      1.00      3601
weighted avg       1.00      1.00      1.00      3601

Performance on test data
              precision    recall  f1-score   support

    not spam       0.94      0.90      0.92       602
        spam       0.85      0.91      0.88       398

    accuracy                           0.90      1000
   macro avg       0.90      0.91      0.90      1000
weighted avg       0.91      0.90      0.90      1000
```

In [11]:

```
dtc = DecisionTreeClassifier(random_state=42)
BC = BaggingClassifier(base_estimator=dtc,
                       n_estimators=10,
                       random_state=42)
BC.fit(X=spam['train_data'], y=spam['train_labels'])
print(round(1-BC.score(X=spam['train_data'], y=spam['train_labels']),5))
print(round(1-BC.score(X=spam['test_data'], y=spam['test_labels']),5))
```

```
0.00417
0.065
```

In [12]:

```
from sklearn.metrics import classification_report
print('Performance on training data')
print(classification_report(spam['train_labels'],
    BC.predict(spam['train_data']),
    target_names=spam['class_names']))

print('Performance on test data')

print(classification_report(spam['test_labels'],
    BC.predict(spam['test_data']),
    target_names=spam['class_names']))
```

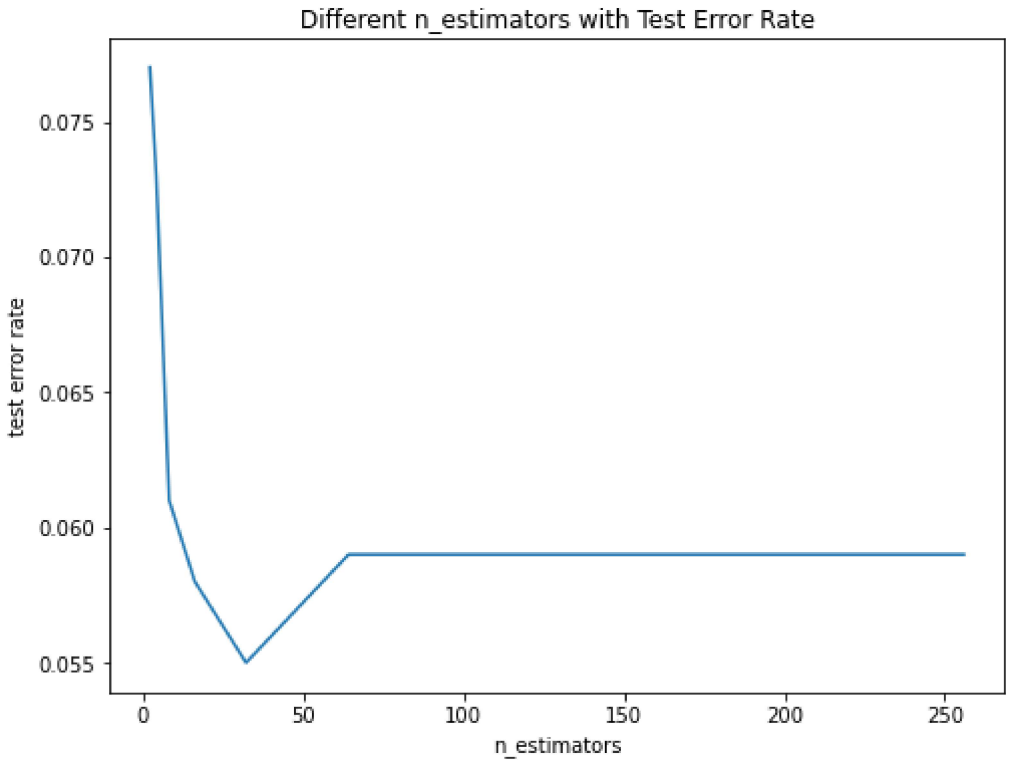
```
Performance on training data
              precision    recall  f1-score   support

    not spam       0.99      1.00      1.00      2186
        spam       1.00      0.99      0.99      1415

    accuracy                           1.00      3601
   macro avg       1.00      0.99      1.00      3601
weighted avg       1.00      1.00      1.00      3601

Performance on test data
              precision    recall  f1-score   support

    not spam       0.96      0.93      0.95       602
```

spam	0.90	0.94	0.92	398
accuracy			0.94	1000
macro avg	0.93	0.94	0.93	1000
weighted avg	0.94	0.94	0.94	1000

problem 3.5

In [13]:
```
params = [2, 4, 8, 16, 32, 64, 128, 256]
test_error_rate = []
for param in params:
    BC = BaggingClassifier(n_estimators=param, random_state=42)
    BC.fit(X=spam['train_data'], y=spam['train_labels'])
    test_error_rate.append(1-BC.score(X=spam['test_data'], y=spam['test_labels']))
```

In [14]:
```
fig,ax = plt.subplots(1,1,figsize=(8,6))
ax.plot(params, test_error_rate);
ax.set_xlabel('n_estimators')
ax.set_ylabel('test error rate')
ax.set_title('Different n_estimators with Test Error Rate');
```



In [ ]:

In [1]:
```
import matplotlib.pyplot as plt
import random
from scipy.stats import bernoulli
```

problem 4.2

In [2]:
```
pryx = [0.2, 0.6, 0.3]
x = [0, 0.2, 0.8, 1]

def ErrorRate(stump):
    '''
    :type stump: dict
    {decision_value: val, true: TrueResult}
    :rtype int
    '''
    error_rate = 0
    for i in range(1,4):
        if stump['decision_value'] > x[i]:
            if stump['true'] == 1:
                error_rate += (x[i] - x[i-1]) * (1 - pryx[i-1])
            else:
                error_rate += (x[i] - x[i-1]) * pryx[i-1]
        elif x[i-1] < stump['decision_value'] <= x[i]:
            if stump['true'] == 1:
                error_rate += (stump['decision_value'] - x[i-1]) * (1 - pryx[i-1])
                error_rate += (x[i] - stump['decision_value']) * pryx[i-1]
            else:
                error_rate += (stump['decision_value'] - x[i-1]) * pryx[i-1]
                error_rate += (x[i] - stump['decision_value']) * (1 - pryx[i-1])
        else:
            if stump['true'] == 1:
                error_rate += (x[i] - x[i-1]) * pryx[i-1]
            else:
                error_rate += (x[i] - x[i-1]) * (1 - pryx[i-1])
    return error_rate
```

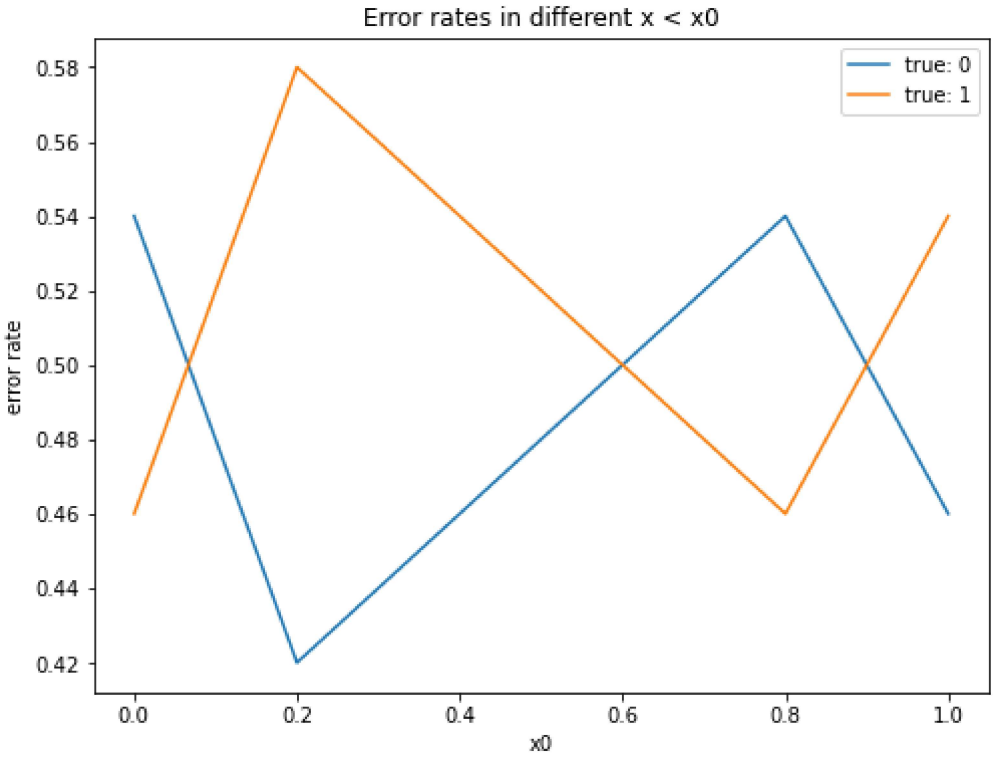
In [3]:
```
stumps = []
stump = {}
stump['decision_value'] = 0.4
stump['true'] = 0
stumps.append(stump)
stump = {}
stump['decision_value'] = 0.6
stump['true'] = 0
stumps.append(stump)
stump = {}
stump['decision_value'] = 0.6
stump['true'] = 1
stumps.append(stump)

for stump in stumps:
    print(round(ErrorRate(stump),2))
```

```
0.46
0.5
0.5
```

problem 4.3

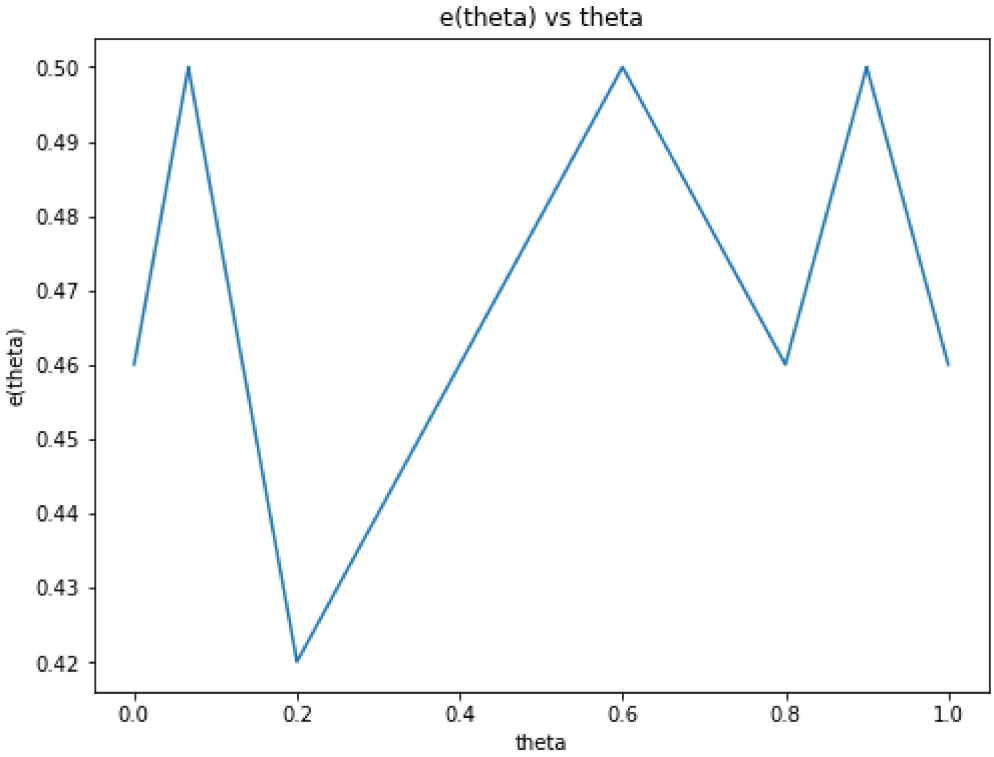
In [4]:
```
stumps = []
error1 = []
error2 = []
error3 = []
x_e = []
for i in range(0, 10001):
    stump1 = {}
    stump2 = {}
    stump1['decision_value'] = i * 0.0001
    stump1['true'] = 0
    stump2['decision_value'] = i * 0.0001
    stump2['true'] = 1
    x_e.append(i * 0.0001)
    error1.append(ErrorRate(stump1))
    error2.append(ErrorRate(stump2))
    error3.append(min(ErrorRate(stump1), ErrorRate(stump2)))
```

In [5]:
```
fig, ax = plt.subplots(1, 1, figsize=(8,6))
line1, = ax.plot(x_e, error1);
line2, = ax.plot(x_e, error2);
line1.set_label('true: 0')
line2.set_label('true: 1')
ax.legend();
ax.set_xlabel('x0');
ax.set_ylabel('error rate')
ax.set_title('Error rates in different x < x0');
```



problem 4.4

In [6]:
```
fig, ax = plt.subplots(1, 1, figsize=(8,6))
line1, = ax.plot(x_e, error3);
ax.set_xlabel('theta');
ax.set_ylabel('e(theta)')
ax.set_title('e(theta) vs theta');
```



problem 4.5

In [7]:

```
def generate_data(seed):
    training_data = [0 for row in range(100)]
    label = [0 for row in range(100)]
    random.seed(seed)

    for i in range(100):
        xi = random.random()
        training_data[i] = xi
        if xi <= 0.2:
            label[i] = bernoulli.rvs(p=0.2, size=1)[0]
        elif xi <= 0.8:
            label[i] = bernoulli.rvs(p=0.6, size=1)[0]
        else:
            label[i] = bernoulli.rvs(p=0.3, size=1)[0]

    return training_data, label
```

In [8]:

```
def predict(stump, data):
    y_predict = [0 for row in range(len(data))]
    for i in range(len(data)):
        if data[i] <= stump['decision_value']:
            if stump['true'] == 1:
                y_predict[i] = 1
            else:
                y_predict[i] = 0
        else:
            if stump['true'] == 1:
                y_predict[i] = 0
            else:
                y_predict[i] = 1
    return y_predict
```

In [9]:

```
def err(y_predict, y_real):
```

```
    error = 0
    for i in range(len(y_predict)):
        if y_predict[i] != y_real[i]:
            error += 1
    error_rate = error / len(y_predict)
    return error_rate
```

In [10]:

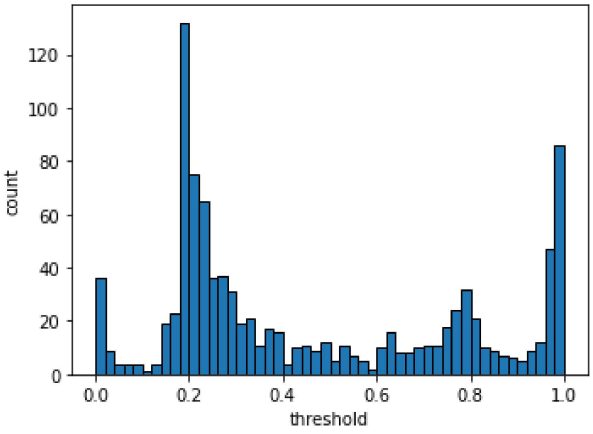
```
def train(training_data, label):
    best_stump = {}
    best_stump['error_rate'] = 1
    best_stump['decision_value'] = 0
    best_stump['exact_error_rate'] = 1
    best_stump['true'] = 0
    for i in range(len(training_data)):
        stump0 = {}
        stump0['decision_value'] = training_data[i]
        stump0['true'] = 0
        error_rate0 = err(predict(stump0, training_data), label)

        stump1 = {}
        stump1['decision_value'] = training_data[i]
        stump1['true'] = 1
        error_rate1 = err(predict(stump1, training_data), label)
        if error_rate0 >= error_rate1 and best_stump['error_rate'] > error_rate1:
            best_stump['error_rate'] = error_rate1
            best_stump['decision_value'] = stump1['decision_value']
            best_stump['true'] = 1
            best_stump['exact_error_rate'] = ErrorRate(best_stump)
        elif error_rate1 >= error_rate0 and best_stump['error_rate'] > error_rate0:
            best_stump['error_rate'] = error_rate0
            best_stump['decision_value'] = stump0['decision_value']
            best_stump['true'] = 0
            best_stump['exact_error_rate'] = ErrorRate(best_stump)
    return best_stump
```

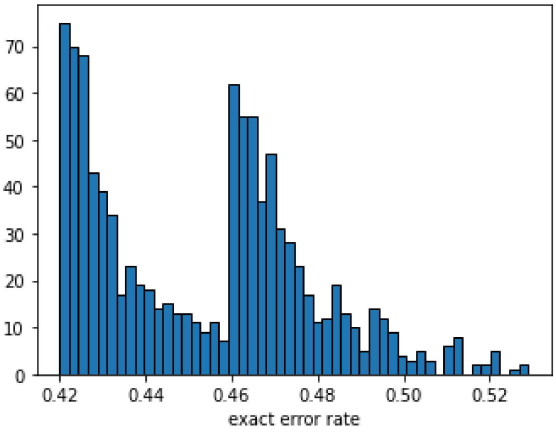
In [11]:

```
thresholds = [0 for row in range(1000)]
err_rates = [0 for row in range(1000)]
exact_err_rates = [0 for row in range(1000)]
for i in range(1000):
    training_data, label = generate_data(i)
    best_stump = train(training_data, label)
    thresholds[i] = best_stump['decision_value']
    exact_err_rates[i] = best_stump['exact_error_rate']
    err_rates[i] = best_stump['error_rate']
```

In [12]:

```
x_e = [i for i in range(1000)]
fig, ax = plt.subplots(1, 2, figsize=(12,4))
ax[0].hist(thresholds, align='mid', edgecolor='black', bins=50);
ax[0].set_xlabel('threshold');
ax[0].set_ylabel('count');
ax[1].hist(exact_err_rates, align='mid', edgecolor='black', bins=50);
ax[1].set_xlabel('exact error rate');
```


count
threshold
0
20
40
60
80
100
120
0.0
0.2
0.4
0.6
0.8
1.0


exact error rate
0
10
20
30
40
50
60
70
0.42
0.44
0.46
0.48
0.50
0.52

In [ ]: