a) $\lambda = \infty$, g will be perfectly smooth, it will be a stright line. $m=0$, so $\hat{g} = \min \left(\sum_{i=1}^{n} (y_i - g(x_i))^2 + \lambda \cdot \int [g(x)]^2 \cdot dx \right.$

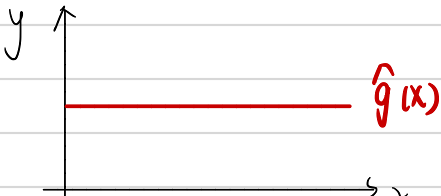
$\hat{g}$ can be 0 since it's min

b) $\lambda = \infty$, $m=1 \Rightarrow \hat{g} = \min \left(\sum_{i=1}^{n} (y_i - g(x_i))^2 + \lambda \cdot \int [g'(x)]^2 \cdot dx \right)$

$\Rightarrow \lambda \int [g'(x)]^2 \cdot dx = 0$, so $g'(x)$ need be 0, $\Rightarrow g(x)$ is constant

Since $g(x_i)$ is a constant, and we need to minimize $\sum_{i=1}^{n} (y_i - g(x_i))^2$

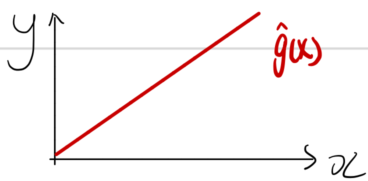
So it's better $g(x) = \bar{x}$.



c) $\lambda = \infty$, $m=2 \Rightarrow \hat{g} = \min \left(\sum_{i=1}^{n} (y_i - g(x_i))^2 + \lambda \cdot \int [g''(x)]^2 \cdot dx \right)$

to minimize $\hat{g}$, we need $\lambda \int [g''(x)]^2 \cdot dx = 0$, so $g''(x)$ need be 0,

$g'(x)$ should be a constant, $g(x)$ is a linear model: $y = ax + b$

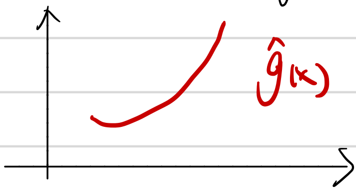


d) $\lambda = \infty$, $m=3$ $\Rightarrow \hat{g} = \min \left(\sum_{i=1}^{n} (y_i - g(x_i))^2 + \lambda \cdot \int [g'''(x)]^2 \cdot dx \right)$

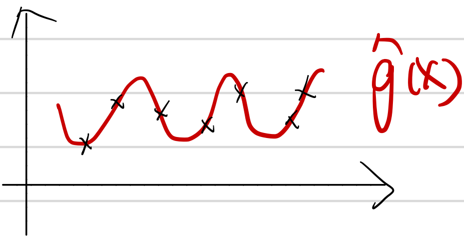
we need $\lambda \int [g'''(x)]^2 \cdot dx = 0$ so $g''(x)$ need be 0,

so $g'(x)$ should be a constant $\Rightarrow$ $g'(x)$: $y = ax+b$

$\Rightarrow g(x) = ax^2 + bx + c$



e) $\lambda = 0$, So $\lambda \cdot \int [g^{(m)}(x)]^2 \cdot dx)$ have no effect, we only care about $\hat{g} = \min \left(\sum_{i=1}^{n} (y_i - g(x_i))^2 \right)$, $\hat{g}(x)$ will be very jumpy and will exactly interpolate the training observations.



a) $\hat{g}_2$ have smaller training RSS

As $\lambda \to \infty$, and we need to minimize $\hat{g}_1$ and $\hat{g}_2$, In other words we need to minimize $\int (g^{(3)}(x))^2 \cdot dx$ and $\int (g^{(4)}(x)) \cdot dx$, since $\hat{g}_2$ have higher order polynomial, it will fit better than $\hat{g}_1$. so $\hat{g}_2$'s RSS smaller

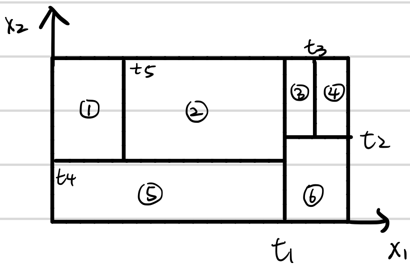
b) We can not say which one have smaller testing RSS. because we dont know the distribution of population data.

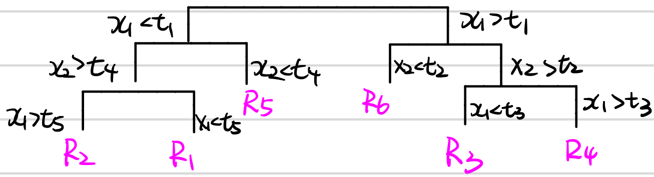
Usually, $\hat{g}_2$ have higher order polynomial, it may overfit the data, so $\hat{g}_1$'s RSS smaller

c) $\lambda = 0$, So $\lambda \cdot \int [g^{(m)}(x)]^2 \cdot dx)$ have no effect, we only care about $\hat{g} = \min \left(\sum_{i=1}^{n} (y_i - g(x_i))^2 \right)$, $\hat{g}_1$ and $\hat{g}_2$ have same training RSS and Test RSS.

1. Draw an example (of your own invention) of a partition of two-dimensional feature space that could result from recursive binary splitting. Your example should contain at least six regions. Draw a decision tree corresponding to this partition. Be sure to label all aspects of your figures, including the regions $R_1, R_2, \ldots$, the cutpoints $t_1, t_2, \ldots$, and so forth.

Hint: Your result should look something like Figures 8.1 and 8.2.





5. Suppose we produce ten bootstrapped samples from a data set containing red and green classes. We then apply a classification tree to each bootstrapped sample and, for a specific value of X, produce 10 estimates of $P(\text{Class is Red}|X)$:

$$0.1, 0.15, 0.2, 0.2, 0.55, 0.6, 0.6, 0.65, 0.7, \text{ and } 0.75.$$

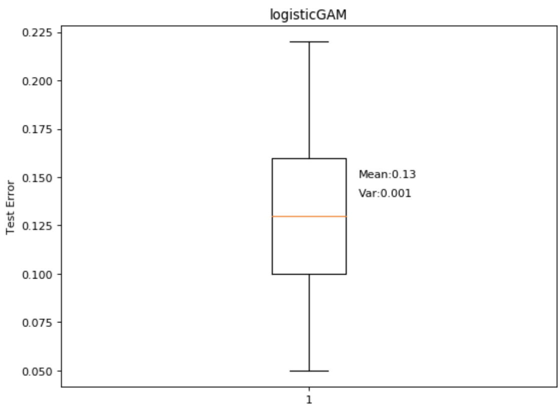
Way 1: 0.1, 0.15, 0.2, 0.2 respesent reject : class is red,
(median) 0.55, 0.6, 0.6, 0.65, 0.7 and 0.75 respent accept : class is red.
Since 4 is reject and 6 accept, we accept the : class is red.

Way 2:
(mean) $AVG = \frac{1}{10}(0.1+0.15+0.2+0.2+0.55+0.6+0.65+0.7+0.75)$

If $Avg < 0.5$, we reject : class is red, otherwise we acept it.

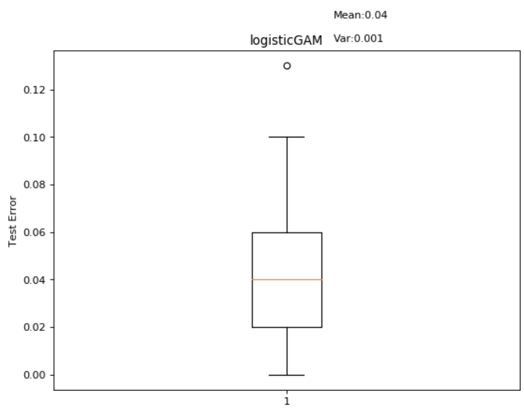
Q3:

Part a)



The mean of test error is 0.13 and the variance of test error is 0.001.

Part b)



the runing time of $p=30$ is slower than $p=10$, and the mean is 0.04 and variance is 0.001.

Xuecheng_Zhang_HW3

March 2, 2020

[ ]:
```python
import  numpy  as np
import  matplotlib.pyplot  as plt
from pygam import LogisticGAM ## LogisticGAM is for classification.



def  generate_data(n, p):
    #n1 is the data size for Y=1
    n_1 = np.random.binomial(n,0.5)
    #n0 is the daa size for Y=0
    n_0 = n-n_1
    #set the x1 and x0
    x_1=np.zeros((n_1,p))
    x_0=np.zeros((n_0,p))
    #case 1
    for i in range (p):
        if (i+1)%2==0:
            x_1[:,i]=np.random.standard_t(1,n_1)+2
        else:
            x_1[:,i]=np.random.exponential(1,n_1)
    #case 2
    for i in range (p):
        if (i+1)%2==0:
            x_0[:,i]=np.random.standard_t(1,n_0)
        else:
            x_0[:,i]=np.random.exponential(1/3,n_0)
    #merge x1,x0
    X=np.vstack((x_1,x_0))
    Y=np.repeat([1,0],(n_1,n_0))
    return X,Y
    #define an empty list for error term
def simulation(No_T,n,p,box_plot=True):
    err=[]
    for i in range (No_T):
    #generate the test data
        X_train,Y_train=generate_data(n,p)
        X_test,Y_test= generate_data(n,p)
```

```
        logit_gam = LogisticGAM()
        logit_gam.gridsearch(X_train,Y_train)

        #calculate test error
        test_err=sum(logit_gam.predict(X_test)!=Y_test)/n
        err.append(test_err)
    if box_plot:
        plt.figure(num=None,figsize=(8,6),dpi=80)
        plt.boxplot(err)
        plt.text(1.1,0.15,"Mean:{:.2f}".format(np.mean(err)))
        plt.text(1.1,0.14,"Var:{:.3f}".format(np.var(err)))
        plt.title("logisticGAM")
        plt.ylabel("Test Error")
        plt.show()
simulation(100,100,10)
simulation(100,100,30)
```