Task 1

Let $f(x_1, x_2) = \beta_0 + \beta_1 x_1 + \beta_2 x_2$ and let $y = f(x_1, x_2) + e$.

Simulate data from this model and calculate (1) and (2) for different values of x_1. Plot the resulting partial dependence relationship.

In [1]:
```
import numpy as np
from sklearn.tree import DecisionTreeRegressor
import statsmodels.api as sm
import matplotlib.pyplot as plt
import pandas as pd
import time
```

```
/anaconda3/lib/python3.6/site-packages/statsmodels/compat/pandas.py:
56: FutureWarning: The pandas.core.datetools module is deprecated an
d will be removed in a future version. Please use the pandas.tseries
module instead.
  from pandas.core import datetools
```

In [2]:
```
np.random.seed(1234567)
X = np.random.uniform(0,10,[1000,2])
x1 = X[:,0]
x2 = X[:,1]
np.random.seed(3456789)
e = np.random.normal(0,4,1000)
```

In [3]:
```
beta_0, beta_1, beta_2, epsilon = 3, 5, 10, 10e-5
```

derivative w.r.t. $f(x_1, x_2) = \beta_0 + \beta_1 x_1 + \beta_2 x_2$

In [4]:
```
def function(x1, x2):
    return beta_0 + beta_1*x1 + beta_2*x2
```

In [5]:
```
def partial_derivative_fcn(x10, x20):
    return (function(x10 + epsilon, x20) - function(x10, x20)) / epsil
```

In [6]:
```
partial_derivative_x1 = np.empty([1000,1000])
for i in range(1000):
    for j in range(1000):
        partial_derivative_x1[j,i] = partial_derivative_fcn(x1[j], x2[
```

In [7]:
```
Temp = pd.DataFrame(partial_derivative_x1)
Temp.to_excel("/Users/zonglinli/Documents/MicroEconometrics/Assignment
del Temp
```

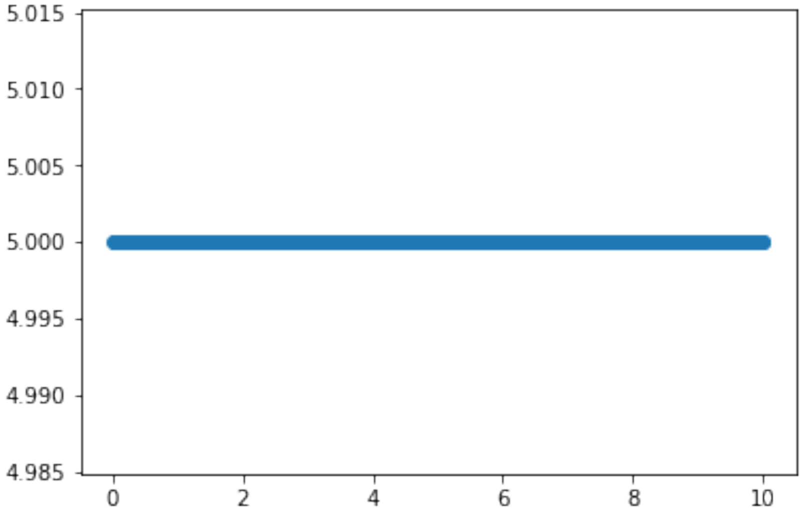
In [ ]:

In [8]:
```
total_derivative_x1 = np.sum(partial_derivative_x1, axis = 1)/1000
```

In [9]:
```
Temp = pd.DataFrame(total_derivative_x1)
Temp.to_excel("/Users/zonglinli/Documents/MicroEconometrics/Assignment5
del Temp
```

In [10]:
```
plt.scatter(x1, total_derivative_x1)
```

Out[10]: <matplotlib.collections.PathCollection at 0x1c0fe1d828>



In [ ]:

derivative w.r.t. $y = f(x_1, x_2) + e$

In [11]:
```
def Y(x1, x2):
    return function(x1, x2) + np.random.normal(0,0.05)
```

In [12]:
```
x10 = 300
x20 = 4
partial_derivative_x10 = (Y(x10 + epsilon, x20) - Y(x10, x20)) / epsil
partial_derivative_x10
```

Out[12]: -190.44189921487487

In [13]:
```
def partial_derivative_Y(x10, x20):
    return (Y(x10 + epsilon, x20) - Y(x10, x20)) / epsilon
```

In [14]:
```
partial_derivative_x1 = np.empty([1000,1000])
for i in range(1000):
    for j in range(1000):
        partial_derivative_x1[j,i] = partial_derivative_Y(x1[j], x2[i]
```

In [15]:
```
Temp = pd.DataFrame(partial_derivative_x1)
Temp.to_excel("/Users/zonglinli/Documents/MicroEconometrics/Assignment!
del Temp
```

In [16]:
```
a = np.array([[1,2],[3,4]])
np.sum(a, axis = 1)
```

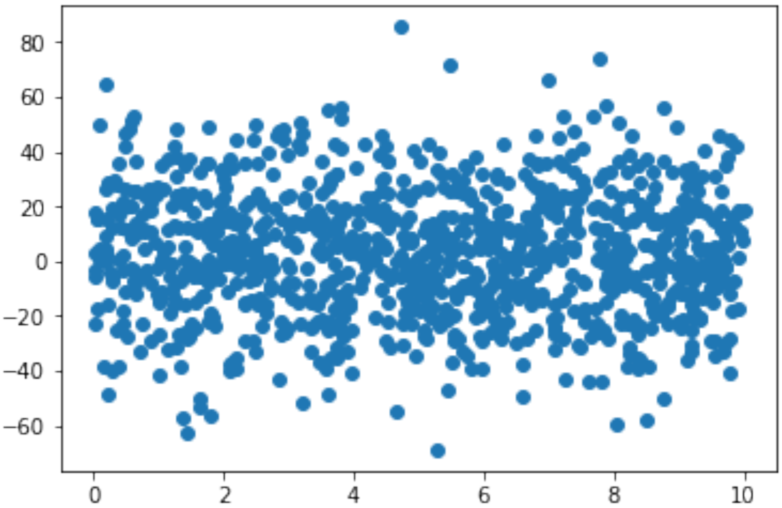
Out[16]:
```
array([3, 7])
```

In [17]:
```
total_derivative_x1 = np.sum(partial_derivative_x1, axis = 1)/1000
```

In [18]:
```
Temp = pd.DataFrame(total_derivative_x1)
Temp.to_excel("/Users/zonglinli/Documents/MicroEconometrics/Assignment!
del Temp
```

In [19]:
```
plt.scatter(x1, total_derivative_x1)
```

Out[19]:
```
<matplotlib.collections.PathCollection at 0x1c196b5c88>
```



In [ ]:
```
```

derivative w.r.t. $f\hat{}(x_1, x_2) = \hat{\beta}_0 + \hat{\beta}_1 x_1 + \hat{\beta}_2 x_2$

In [20]:
```
y = function(x1, x2) + e
```

In [21]:
```
model2 = sm.OLS(y, X)
results = model2.fit()
print(results.summary())
```

```
0.998
Method:                 Least Squares   F-statistic:
2.056e+05
Date:                Sun, 22 Dec 2019   Prob (F-statistic):
0.00
Time:                        11:06:21   Log-Likelihood:
-2843.2
No. Observations:                1000   AIC:
5690.
Df Residuals:                     998   BIC:
5700.
Df Model:                           2
Covariance Type:            nonrobust
====================================================================
==========
                 coef    std err          t      P>|t|      [0.025
0.975]
--------------------------------------------------------------------
----------
```

In [22]:
```
x10 = 5
x20 = 4
partial_derivative_x10 = (model2.predict(x10 + epsilon, x20) - model2.
partial_derivative_x10
```

Out[22]: 3.9999999999906777

In [23]:
```
def partial_derivative_Yhat(x10, x20):
    return (model2.predict(x10 + epsilon, x20) - model2.predict(x10, x
```

In [24]:
```
partial_derivative_x1 = np.empty([1000,1000])
for i in range(1000):
    for j in range(1000):
        partial_derivative_x1[j,i] = partial_derivative_Yhat(x1[j], x2
```

In [25]:
```
Temp = pd.DataFrame(partial_derivative_x1)
Temp.to_excel("/Users/zonglinli/Documents/MicroEconometrics/Assignment5
del Temp
```

In [ ]:

In [26]:
```
total_derivative_x1 = np.sum(partial_derivative_x1, axis = 1) / 1000
```

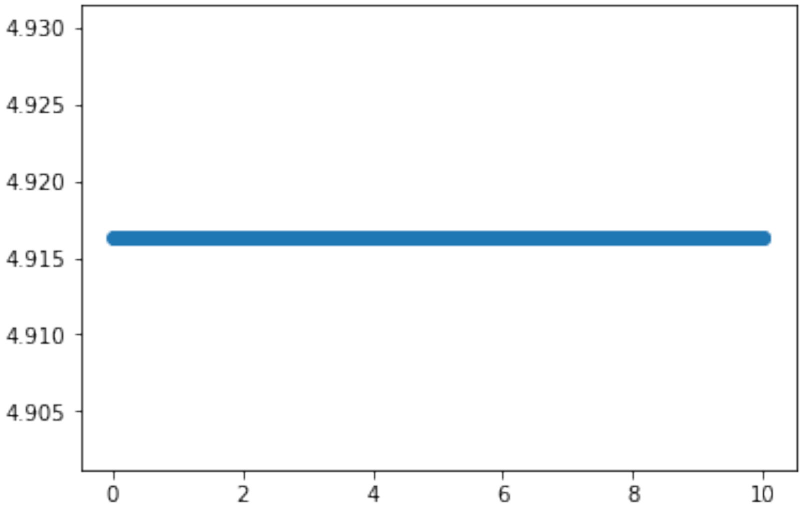
In [27]:
```
total_derivative_x1[2]
```

Out[27]: 4.916321338580669

In [28]:
```
Temp = pd.DataFrame(total_derivative_x1)
Temp.to_excel("/Users/zonglinli/Documents/MicroEconometrics/Assignment!
del Temp
```

In [29]:
```
plt.scatter(x1, total_derivative_x1)
```

Out[29]: <matplotlib.collections.PathCollection at 0x1c1c078048>



In [ ]:

In [ ]:

Task 2-1

Find (or simulate) any data set. Fit a decision tree and calculate (1) and (2) for different values of an input variable.

Plot the resulting partial dependence relationship.

In [30]:
```
X_train = np.empty([1000,2])
np.random.seed(3456789)
x1_train = X_train[:,0] = np.random.uniform(0.5,1.5,1000)
np.random.seed(4567890)
x2_train = X_train[:,1] = np.random.uniform(-1.5,-0.5,1000)

y_train = 0.5*x1*x1 + 0.5*x2*x2 - x1 + x2
```

Random Forest

In [31]:
```
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestRegressor

t0 = time.time()
gs = GridSearchCV(
    estimator=RandomForestRegressor(),
    param_grid={
        'max_depth': [10, None],
        'n_estimators': (100,125,150,175,200),
        'max_features': (1,2)}, cv=5, n_jobs=-1)
model = gs.fit(X_train,y_train)
print(time.time()-t0)
```

```
/anaconda3/lib/python3.6/site-packages/sklearn/ensemble/weight_boost
ing.py:29: DeprecationWarning: numpy.core.umath_tests is an internal
NumPy module and should not be imported. It will be removed in a fut
ure NumPy release.
  from numpy.core.umath_tests import inner1d

19.39178490638733
```

In [32]:
```
X_test = np.empty([500,2])
np.random.seed(6789345)
x1_test = X_test[:,0] = np.random.uniform(0.5,1.5,500)
np.random.seed(7890456)
x2_test = X_test[:,1] = np.random.uniform(-1.5,-0.5,500)
```

In [33]:
```
def partial_derivative_RF(X_epsilon, X):
    return (model.predict(X_epsilon) - model.predict(X)) / epsilon
```

In [34]:
```
partial_derivative_x1 = np.empty([500,500])
X_epsilon = np.empty([500,2])
X_epsilon[:,0] = x1_test + epsilon
X = np.empty([500,2])
X[:,0] = x1_test
for i in range(500):
    X_epsilon[:,1] = X[:,1] = x2_test[i]
    partial_derivative_x1[:,i] = partial_derivative_RF(X_epsilon, X)
```

In [35]:
```
Temp = pd.DataFrame(partial_derivative_x1)
Temp.to_excel("/Users/zonglinli/Documents/MicroEconometrics/Assignment!
del Temp
```

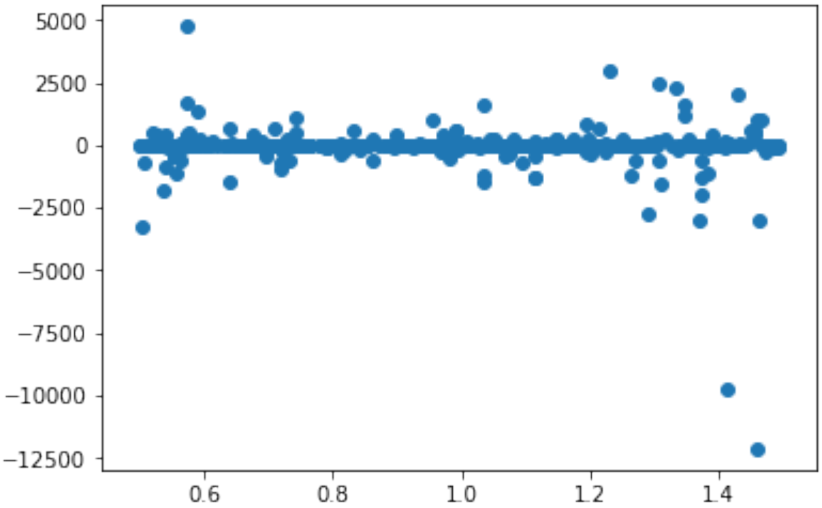
In [ ]:
```

```

In [36]:
```
total_derivative_x1 = np.sum(partial_derivative_x1, axis = 1) / 1000
```

In [37]:
```
Temp = pd.DataFrame(total_derivative_x1)
Temp.to_excel("/Users/zonglinli/Documents/MicroEconometrics/Assignment!
del Temp
```

In [38]:
```
plt.scatter(x1_test, total_derivative_x1)
```

Out[38]: <matplotlib.collections.PathCollection at 0x1c1a3ece48>



In [ ]:

Task2-2

As for the challenge task 2-2, we use the function $Y = 2X_1 - 3X_2 + 0.6X_1X_2$ as the population function.

```
rm(list=ls())
library(nnet)
library(rpart)
library(rpart.plot)
library(randomForest)
```

```
## randomForest 4.6-14
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```
library(gbm)
```

```
## Loaded gbm 2.1.5
```

```
library(caret)
```

```
## Loading required package: lattice
```

```
## Loading required package: ggplot2
```

```
## Registered S3 methods overwritten by 'ggplot2':
##   method         from
##   [.quosures     rlang
##   c.quosures     rlang
##   print.quosures rlang
```

```
##
## Attaching package: 'ggplot2'
```

```
## The following object is masked from 'package:randomForest':
##
##     margin
```

```
library(AER)
```

```
## Loading required package: car
```

```
## Loading required package: carData
```

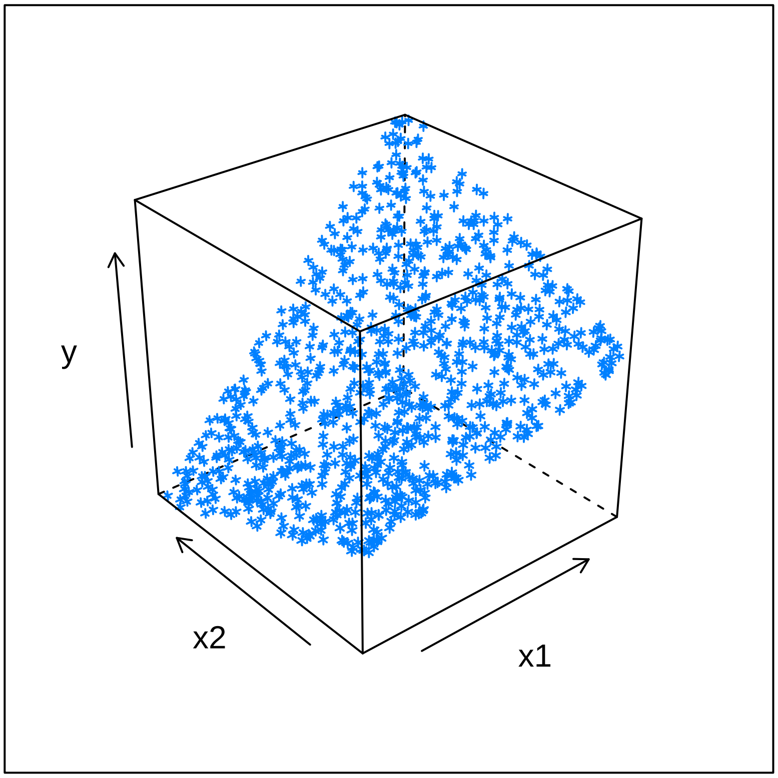
```
## Loading required package: lmtest
```

```
## Loading required package: zoo
```

```
##
## Attaching package: 'zoo'
```

```
## The following objects are masked from 'package:base':
##
##     as.Date, as.Date.numeric

## Loading required package: sandwich

## Loading required package: survival

##
## Attaching package: 'survival'

## The following object is masked from 'package:caret':
##
##     cluster
```

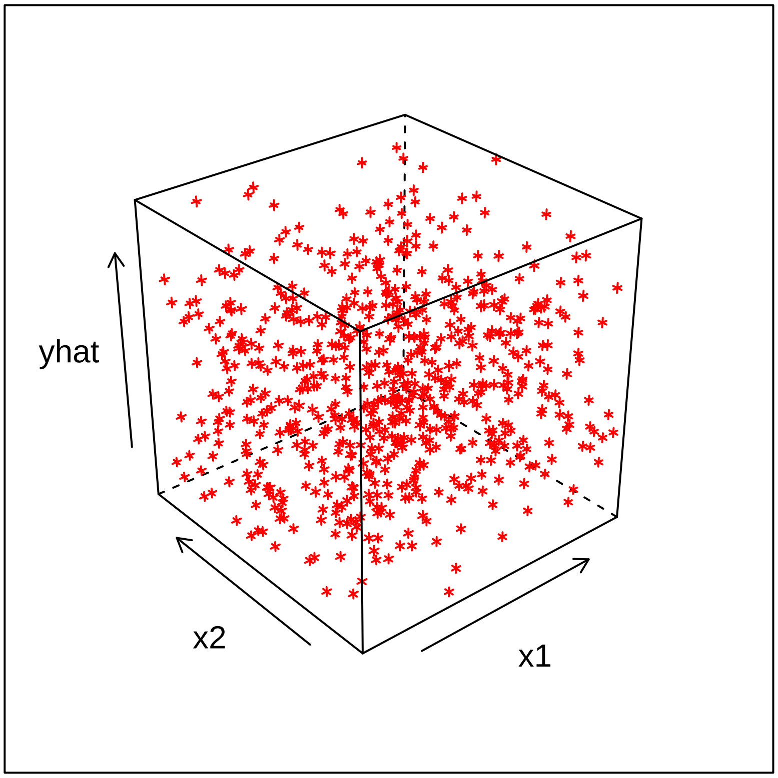
```r
set.seed(61)
x1t=runif(1000,0,12)
x2t=runif(1000,0,12)
yt=2*x1t-3*x2t+.6*x1t*x2t
#########
# create training and test set
n=length(x1t)
data = data.frame(x1=x1t,x2=x2t,y=yt)
train = sample(n,n*0.3)##30% as the training data
data_train = data[train,]
data_test = data[-train,]
ytrue = data_test[,"y"]
attach(data)
cloud(y~x1+x2)#draw the picture
```




```
#################
# Random Forest #
#################
set.seed(66)
fit = randomForest(y~.,data=data_train,mtry=1,n.tree=1000)

# test err
yhat = predict(fit,data_test)
mean((yhat-ytrue)^2)#R square
```

```
## [1] 10.23578
```

```
cloud(yhat~x1+x2,col="red")
```



Try to use the ***monte carlo integration*** to see the "effect" when $X_1 = 5$:

$$\left.\frac{df\left(x_1, x_2\right)}{dx_1}\right|_{x_1=5} \approx \sum_{i=1}^{M} \left.\frac{\partial f\left(x_1, x_{2i}\right)}{\partial x_1}\right|_{x_1=5} = \sum_{i=1}^{M} \frac{f\left(5+\epsilon, x_{2i}\right) - f\left(5, x_{2i}\right)}{\epsilon}$$

```
e<-0.0001
xp1=rep(5,1000)
xp2=seq(0.012,12,0.012)
datap=cbind(xp1,xp2)
yp=predict(fit,datap)

xp11=rep(5+e,1000)
xp21=seq(0.012,12,0.012)
datap1=cbind(xp11,xp21)
yp1=predict(fit,datap1)
try=cbind(yp1,yp)
head(try)#Almost the same for the predict value and the original value
```

```
##          yp1         yp
## 1   6.917325   6.917325
## 2  41.841305  41.841305
## 3   6.593089   6.593089
## 4 -18.810732 -18.810732
```

```
## 5 -18.887095 -18.887095
## 6  -1.084416  -1.084416
```

```
sum(yp1-yp)/e
```

```
## [1] 0
```

Now, try to draw the picture of "effect" of X_1

```
z<-c()
m<-c()
for(i in 1:120)#Try to sole the effect for different X1
{
e<-0.1
xp1=rep(i/10,1000)
xp2=seq(0.012,12,0.012)
datap=cbind(xp1,xp2)
yp=predict(fit,datap)

xp11=rep(i/10+e,1000)
xp21=seq(0.012,12,0.012)
datap1=cbind(xp11,xp21)
yp1=predict(fit,datap1)



z[i]=sum(yp1-yp)/e
m[i]=i/10
print(i)
print(z[i])
}
```

```
## [1] 1
## [1] 0
## [1] 2
## [1] 0
## [1] 3
## [1] 0
## [1] 4
## [1] 0
## [1] 5
## [1] 0
## [1] 6
```

```
## [1] 0
## [1] 7
## [1] 0
## [1] 8
## [1] 0
## [1] 9
## [1] 0
## [1] 10
## [1] 0
## [1] 11
## [1] 0
## [1] 12
## [1] 0
## [1] 13
## [1] 0
## [1] 14
## [1] 0
## [1] 15
## [1] 0
## [1] 16
## [1] 0
## [1] 17
## [1] 0
## [1] 18
## [1] 0
## [1] 19
## [1] 0
## [1] 20
## [1] 0
## [1] 21
## [1] 0
## [1] 22
## [1] 0
## [1] 23
## [1] 0
## [1] 24
## [1] 0
## [1] 25
## [1] 0
```

```
## [1] 26
## [1] 0
## [1] 27
## [1] 0
## [1] 28
## [1] 0
## [1] 29
## [1] 0
## [1] 30
## [1] 0
## [1] 31
## [1] 0
## [1] 32
## [1] 0
## [1] 33
## [1] 0
## [1] 34
## [1] 0
## [1] 35
## [1] 0
## [1] 36
## [1] 0
## [1] 37
## [1] 0
## [1] 38
## [1] 0
## [1] 39
## [1] 0
## [1] 40
## [1] 0
## [1] 41
## [1] 0
## [1] 42
## [1] 0
## [1] 43
## [1] 0
## [1] 44
## [1] 0
## [1] 45
```

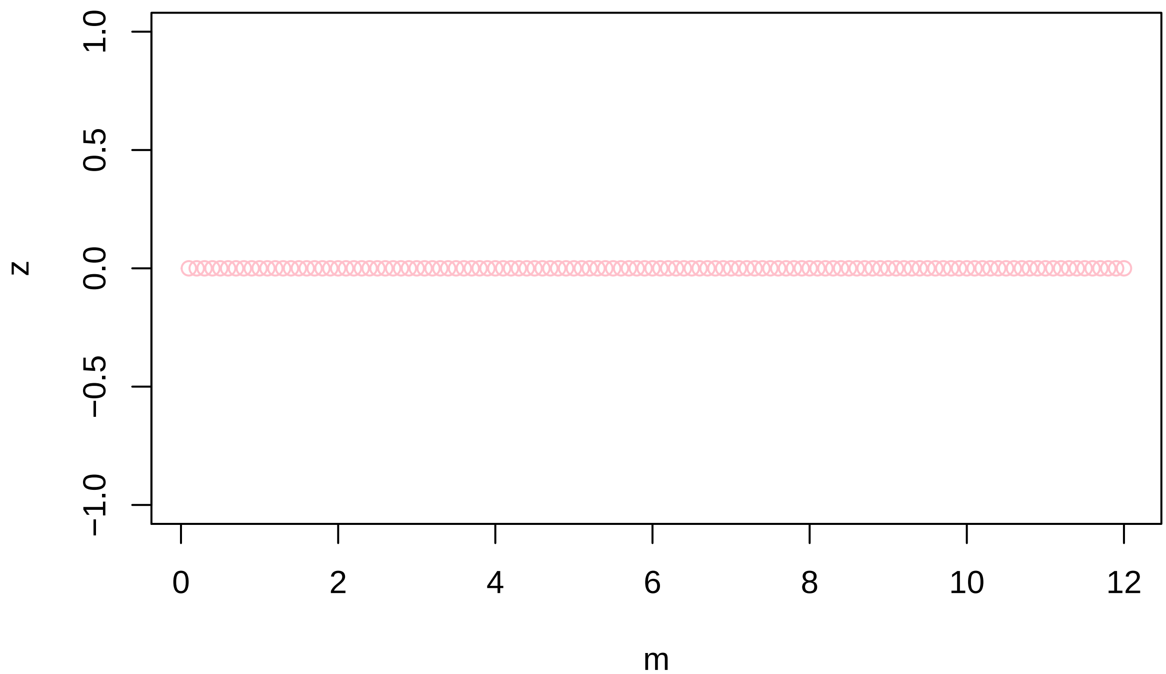
```
## [1] 0
## [1] 46
## [1] 0
## [1] 47
## [1] 0
## [1] 48
## [1] 0
## [1] 49
## [1] 0
## [1] 50
## [1] 0
## [1] 51
## [1] 0
## [1] 52
## [1] 0
## [1] 53
## [1] 0
## [1] 54
## [1] 0
## [1] 55
## [1] 0
## [1] 56
## [1] 0
## [1] 57
## [1] 0
## [1] 58
## [1] 0
## [1] 59
## [1] 0
## [1] 60
## [1] 0
## [1] 61
## [1] 0
## [1] 62
## [1] 0
## [1] 63
## [1] 0
## [1] 64
## [1] 0
```

```
## [1] 65
## [1] 0
## [1] 66
## [1] 0
## [1] 67
## [1] 0
## [1] 68
## [1] 0
## [1] 69
## [1] 0
## [1] 70
## [1] 0
## [1] 71
## [1] 0
## [1] 72
## [1] 0
## [1] 73
## [1] 0
## [1] 74
## [1] 0
## [1] 75
## [1] 0
## [1] 76
## [1] 0
## [1] 77
## [1] 0
## [1] 78
## [1] 0
## [1] 79
## [1] 0
## [1] 80
## [1] 0
## [1] 81
## [1] 0
## [1] 82
## [1] 0
## [1] 83
## [1] 0
## [1] 84
```

```
## [1] 0
## [1] 85
## [1] 0
## [1] 86
## [1] 0
## [1] 87
## [1] 0
## [1] 88
## [1] 0
## [1] 89
## [1] 0
## [1] 90
## [1] 0
## [1] 91
## [1] 0
## [1] 92
## [1] 0
## [1] 93
## [1] 0
## [1] 94
## [1] 0
## [1] 95
## [1] 0
## [1] 96
## [1] 0
## [1] 97
## [1] 0
## [1] 98
## [1] 0
## [1] 99
## [1] 0
## [1] 100
## [1] 0
## [1] 101
## [1] 0
## [1] 102
## [1] 0
## [1] 103
## [1] 0
```

```
## [1] 104
## [1] 0
## [1] 105
## [1] 0
## [1] 106
## [1] 0
## [1] 107
## [1] 0
## [1] 108
## [1] 0
## [1] 109
## [1] 0
## [1] 110
## [1] 0
## [1] 111
## [1] 0
## [1] 112
## [1] 0
## [1] 113
## [1] 0
## [1] 114
## [1] 0
## [1] 115
## [1] 0
## [1] 116
## [1] 0
## [1] 117
## [1] 0
## [1] 118
## [1] 0
## [1] 119
## [1] 0
## [1] 120
## [1] 0
```

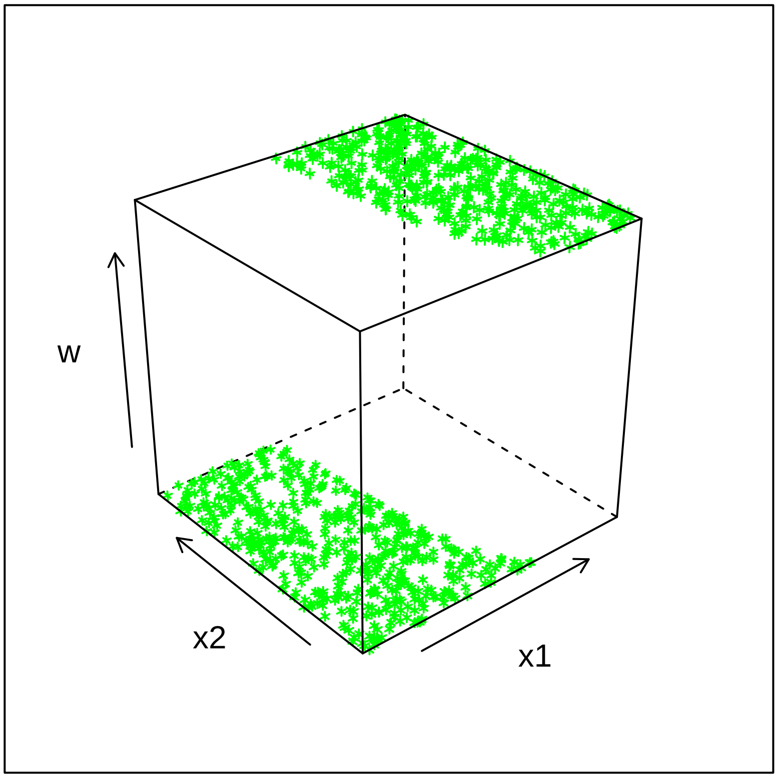
```r
plot(m,z,col="pink")
```



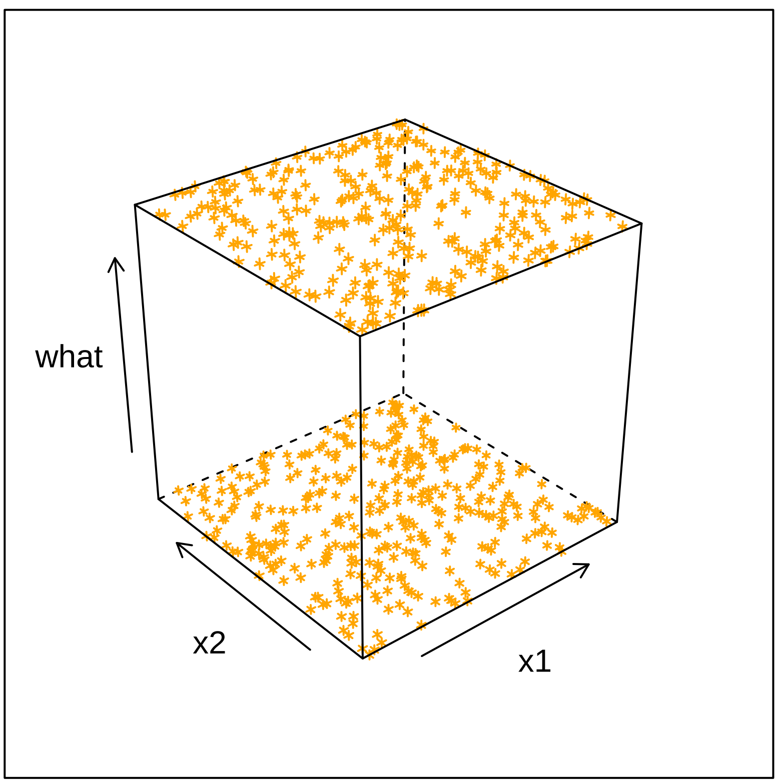
Seems that because it is still the (piecewise) constant model, it is not good at predict the "effect".

Now I am trying the binary and boosting case.

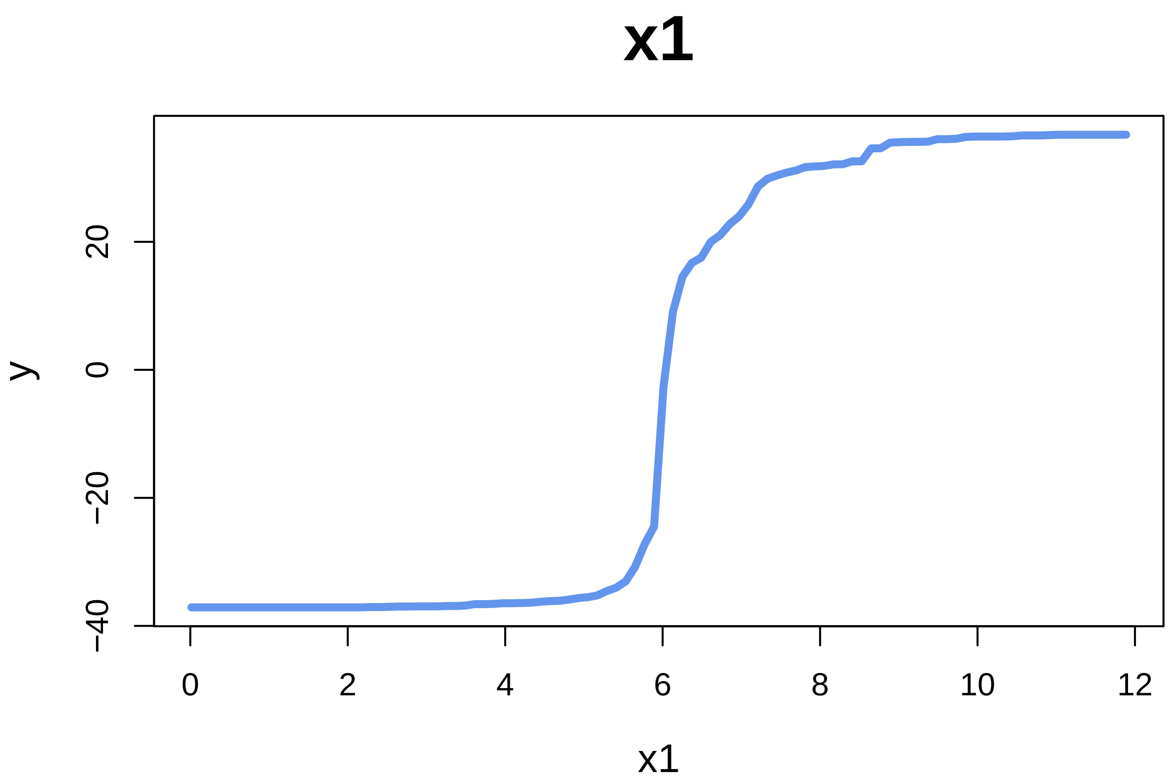
```
#change it into the binary function
set.seed(100)
w=as.numeric(y>mean(y))
cloud(w~x1+x2,col="green")
```



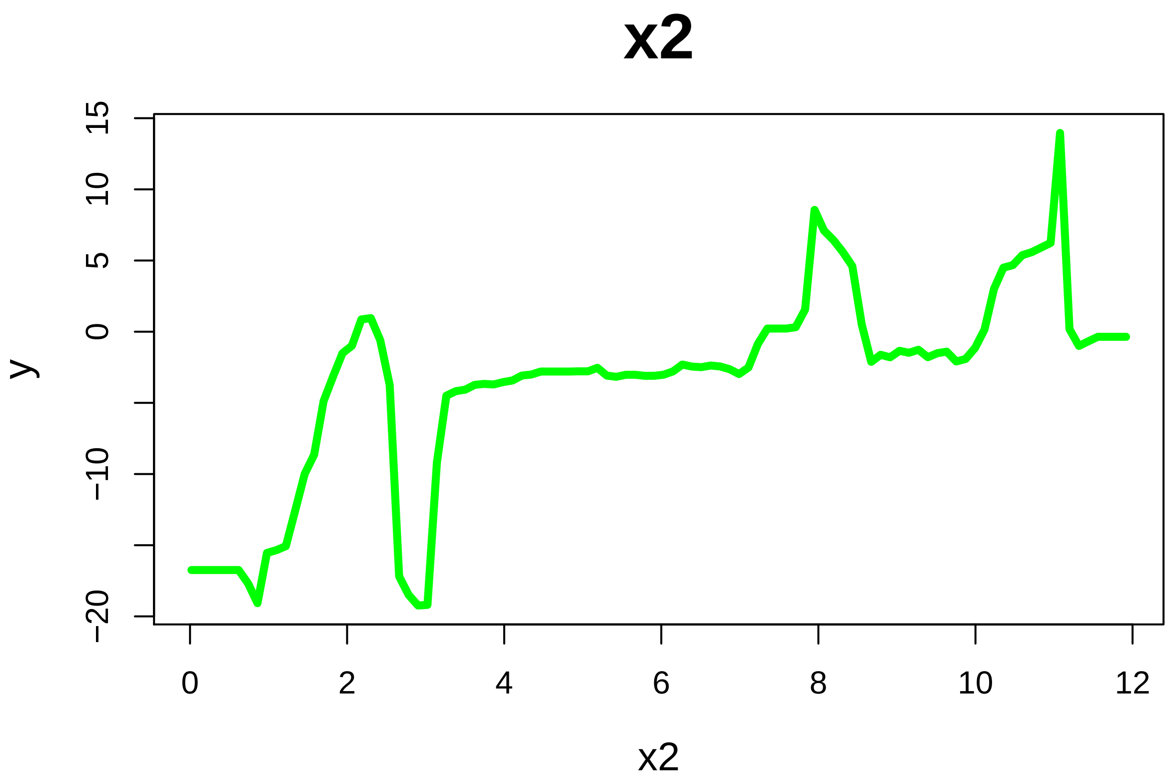

```
what=as.numeric(yhat>mean(yhat))
cloud(what~x1+x2,col="orange")
```



```
##use boosting to draw the "effect"
data_boost = transform(data_train,y=as.numeric(y>mean(y)))
ntree = 500
fit = gbm(y~.,data_boost, distribution="adaboost",
          n.trees=ntree,interaction.depth=10,shrinkage=0.1)
h1 = plot(fit,i="x1",return.grid=T)



h2 = plot(fit,i="x2",return.grid=T)
plot(h1$x1,h1$y,type="l",col="cornflowerblue",lwd=4,
     main="x1",
     #xlim=c(-3,2),
     cex.main=2, cex.lab=1.25,
     xlab="x1",ylab="y")
```




```
plot(h2$x2,h2$y,type="l",col="green",lwd=4,
     main="x2",
     #xlim=c(-9,100),
     cex.main=2, cex.lab=1.25,
     xlab="x2",ylab="y")
```



References

[1] Random Forest with GridSearchCV in Python and Decision Trees. https://mlfromscratch.com/random-forest-gridsearchcv-python/#/

[2] Jiaming MAO's lecture.

https://github.com/jiamingmao/data-analysis/blob/master/Lectures/Decision_Trees_and_Ensemble_Methods.pdf