**Introduction**

This post is the second of a series of examples of the BooST (Boosting Smooth Trees) model. You can see an introduction to the model [here](https://insightr.wordpress.com/2018/08/14/boost-boosting-smooth-trees-a-new-machine-learning-model-for-partial-effect-estimation-in-nonlinear-regressions/) and the first example [here](https://insightr.wordpress.com/2018/08/20/boost-series-i-advantage-in-smooth-functions/). Our objective in this post is to use the derivatives of the BooST to obtain prices that maximize the profit for a given set of products. We will use a very simple setup that we know the true optimal prices to compare with the estimated prices. The tricky thing here is that the demand functions we defined are for substitute products. Therefore, if we increase the price of product A it will affect positively the demand for product B.

**Little Math for those who like it **

Suppose we have nproducts with linear demand function. As mentioned above, the demand of each product depends on it’s own price and the price of the other products. In matrix notation, we can write this relationship with the following expression:

\displaystyle q = \beta p + \alpha,

where q \in \mathbb{R}^nis the demand vector with the quantity sold, \betais a n \times nmatrix that has the coefficients of the demand curve, \alpha \in \mathbb{R}^nis the vector with the intercept of each demand curve and p \in \mathbb{R}^nis the vector of prices. Moreover, we will assume that the matrix \betais negative definite, which is simply an assumption that the higher the price, the lower the demand and that the products compete among themselves. If this assumption do not hold, we can come up with crazy situations, where the higher the price, the higher the demand. Therefore, we could charge an infinite price for the product and get extremely rich. Normally life is harder than that, right?

For the profit function, we can write it in the following form in matrix notation:

\displaystyle \pi(p,q) = \langle p - c, q \rangle,

where d,p,care the vectors of demand, prices and unit costs, respectively. \langle .,.\rangledenotes the inner product. The inner product is there just to sum the profit of all products. Using the first equation and the second we have:

\displaystyle \pi = \langle p - c, \beta p + \alpha \rangle,

Now, we can take the derivative of \piwith respect to the vector of prices p. The first order condition gives us:

\displaystyle p^\star = (\beta^\top + \beta)^{-1}(-\alpha + \beta^\top c),

where p^\staris the vector of optimal prices. Moreover, we know that this is a maximum point for the profit function, because \betais negative definite.

**Example**

In this example we are going to generate data for the demand function of three substitute products, for example, three different TVs. The data will be generated from the equations in the previous section (we included an error term in the demand equation to introduce some noise). The code below generates the data and presents the optimal analytical prices for each of the three TVs.

library(tidyverse)

library(reshape2)

library(BooST)

# Declare intercept

alpha1 = 2000

alpha2 = 2500

alpha3 = 6000

alpha = c(alpha1, alpha2, alpha3)

# Declare betas (effect in the own demand, must be negative)

b11 = - 3

b22 = - 3

b33 = - 5

# (effect in the demand of other products, must be positive but not too big)

b12 = 0.5 ; b13 = 0.1

b21 = 0.5 ; b23 = 0.5

b31 = 0.1; b32 = 0.5

# Matrix beta of the demand function

B = matrix(data = c(b11,b12,b13,b21,b22,b23,b31,b32,b33), byrow = T, ncol = 3)

# Declare costs

c1 = 200

c2 = 500

c3 = 1000

# Vector of costs

c = c(c1,c2,c3)

# Formula for optimal prices

op = as.vector(solve(t(B) + B)%\*%(-alpha + t(B)%\*%c))

op

## [1] 556.3723 854.3220 1169.5596

Now we are going to generate some data from the demand equations. For each equation we generated 300 observations with prices following a uniform distribution centered in the true optimal prices. Each equation was adjusted to have and R2 of approximately 67\%

set.seed(1)

N = 300

p1 = runif(N,op[1]-0.5\*op[1],op[1]+0.5\*op[1])

p2 = runif(N,op[2]-0.5\*op[2],op[2]+0.5\*op[2])

p3 = runif(N,op[3]-0.5\*op[3],op[3]+0.5\*op[3])

#

q1 = alpha1+b11\*p1+p2\*b12+p3\*b13

q2 = alpha2+b21\*p1+p2\*b22+p3\*b23

q3 = alpha3+b31\*p1+p2\*b32+p3\*b33

set.seed(1)

q1 = q1 + rnorm(N,sd = sd(q1)/2)

q2 = q2 + rnorm(N,sd = sd(q2)/2)

q3 = q3 + rnorm(N,sd = sd(q3)/2)

The plot below gives us an idea of the three demand functions. Note that we plotted each demand against only their respective price. However, each demand function also depends on the prices of the other TVs.

ind = as.factor(c(rep(1,300),rep(2,300),rep(3,300)))

dfplot = data.frame(product = ind, q = c(q1,q2,q3), p = c(p1,p2,p3))

ggplot(data = dfplot) +

geom\_point(aes(x = p, y = q, color = product))



Next, we use the generated data to run the BooST for each demand. Note that all models must have the three prices to capture the substitution effect.

# Run BooST #

df = data.frame(q1,q2,q3,p1,p2,p3)

y1 = df$q1

y2 = df$q2

y3 = df$q3

x = as.matrix(df[,-c(1:3)])

set.seed(1)

BooST1 = BooST(x,y1, v = 0.1, M = 50)

BooST2 = BooST(x,y2, v = 0.1, M = 50)

BooST3 = BooST(x,y3, v = 0.1, M = 50)

Before estimating the optimal prices we can explore the resulting demand functions. Recall that the demand of product 1 also depends on the prices of products 2 and 3 and cannot be visualized in two dimensions. To make the example more visual we fixed the prices of products 2 and 3 in their median and estimated the fitted demand curve for product one (black line in the plot below).

dfplot1 = as.data.frame(x)

dfplot1$p2 = median(dfplot1$p2)

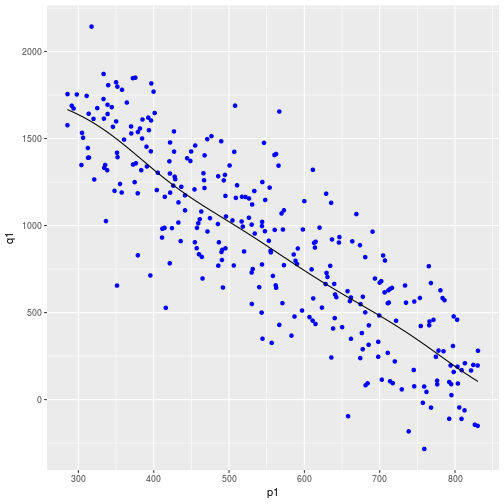
dfplot1$p3 = median(dfplot1$p3)

dfplot1$fitted = predict(BooST1,dfplot1)

dfplot1$q1 = q1

ggplot(data = dfplot1) + geom\_point(aes(x = p1, y = q1),color = "blue") +

geom\_line(aes(x=p1,y=fitted))

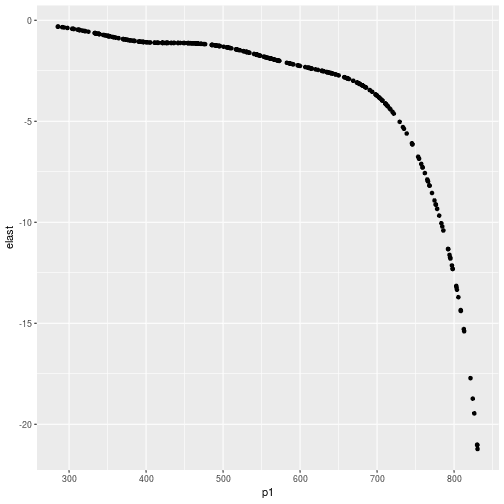


We can follow the same idea from the previous plot to obtain the price elasticity for product 1.

dfplot1$d1 = estimate\_derivatives(BooST1,dfplot1[,1:3],1)

dfplot1$elast = dfplot1$d1\*(dfplot1$p1/dfplot1$fitted)

ggplot(data = dfplot1) + geom\_point(aes(x = p1,y = elast))



Since our objective is to estimate the optimal prices, the next step is to create a gradient function that estimates the derivatives from each BooST and use them to obtain the gradient of the profit.

gr = function(x,BooST1,BooST2,BooST3){

X = matrix(c(x[1],x[2],x[3]),nrow = 1)

q1p1 = estimate\_derivatives(BooST1,X,1)

q1p2 = estimate\_derivatives(BooST2,X,1)

q1p3 = estimate\_derivatives(BooST3,X,1)

q2p1 = estimate\_derivatives(BooST1,X,2)

q2p2 = estimate\_derivatives(BooST2,X,2)

q2p3 = estimate\_derivatives(BooST3,X,2)

q3p1 = estimate\_derivatives(BooST1,X,3)

q3p2 = estimate\_derivatives(BooST2,X,3)

q3p3 = estimate\_derivatives(BooST3,X,3)

q1 = predict(BooST1, X)

q2 = predict(BooST2, X)

q3 = predict(BooST3, X)

g1 = q1+(x[1]-c1)\*q1p1+(x[2]-c2)\*q1p2+(x[3]-c3)\*q1p3

g2 = q2+(x[1]-c1)\*q2p1+(x[2]-c2)\*q2p2+(x[3]-c3)\*q2p3

g3 = q3+(x[1]-c1)\*q3p1+(x[2]-c2)\*q3p2+(x[3]-c3)\*q3p3

g = c(g1,g2,g3)

return(g)

}

All we have to do now is to move on the gradient until the prices converge. The code below does 10 iterations in a very naive algorithm to obtain the optimal prices. The starting values are far from the true optimal prices but they must be in the support of observed prices.

p\_est = matrix(NA,10,3)

p\_est[1,] = c(300,1200,1700)

for(i in 2:10){

p\_est[i,] = p\_est[i-1,]+0.1\*gr(p\_est[i-1,],BooST1,BooST2,BooST3)

}

p\_est = as.data.frame(p\_est)

colnames(p\_est) = c("p1","p2","p3")

p\_est$iter = 1:10

Finally, the results. The dashed lines show the true optimal prices and the continuous lines show the algorithm moving towards the optimal values as we increase the iterations.

dfm = melt(p\_est,id.vars = "iter")

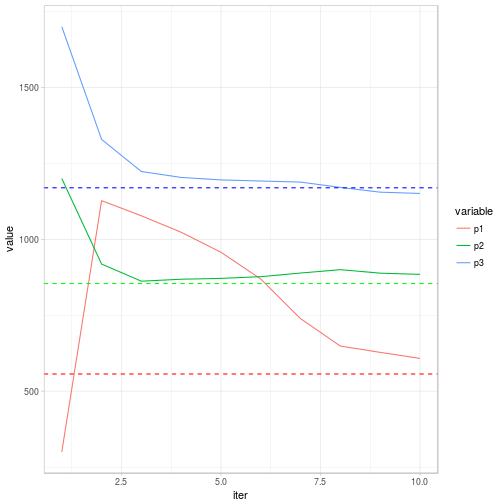
ggplot(data = dfm) + geom\_line(aes(x = iter, y = value, color = variable)) +

theme\_light() +

geom\_hline(yintercept = op[1],col = "red",linetype = 2) +

geom\_hline(yintercept = op[2],col = "green",linetype = 2) +

geom\_hline(yintercept = op[3],col = "blue",linetype = 2)



**Some observations**

* You could also use the gradient function we defined in the R function optim and find similar results. It is always good to use the analytical gradient in the optim, especially if we are optimizing on many variables.
* We estimated three BooSTs, one for each product. However, it is possible to obtain the same results with only one BooST if we estimate the model straight on the profits without looking at the demand. The downside is that we will recover the individual demand functions separated from the profit.