Topic A2:

Optimal pricing with estimated demand

Takeaki Sunada[1]

[1]Simon Business School
University of Rochester

Optimal pricing with estimated demand

- We have mostly focused on how to obtain a good causal estimate of the price coefficient, β_1.
- Once we have the right estimate, of course we next calculate the profit-maximizing price (that's the whole purpose of the exercise).
- We cover three ways to calculate the optimal price.
  - Brute-force numerical search
  - "optim" function - a numerical solver
  - Analytical calculation

Pricing with regression models

- Suppose your regression line takes the following form.

$$\log(Q) = 12 - 8.7 \log(P).$$

- Assume the unit cost is one. What is the optimal price?

Pricing with regression models

- Firm's profit is given by:

$$\begin{aligned} & P \times Q - 1 \times Q \\ = & (P-1) \times \exp(12 - 8.7\log(P)). \end{aligned}$$

- Don't forget that the estimated function is log demand!
- We want a P that maximizes this function.

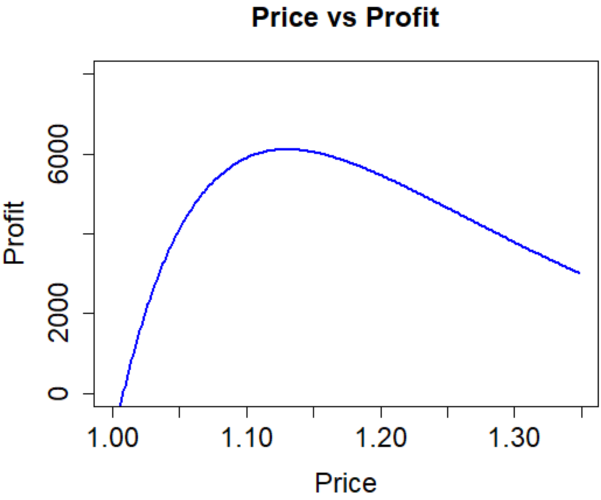
A brute-force approach

- With this sort of simple unidimensional problem, we can simply evaluate the profit at many different values of P and pick the maximum.
- Let's call it a brute-force approach for obvious reasons.

```
#Approach 1: brute force
#Take a lot of candidate prices
pricespace=seq(1,1.8,0.001)

#Evaluate demand at each point
logdemand=12-8.7*log(pricespace)

#Evaluate profit at each point
profit=(pricespace-1)*exp(logdemand)

#Find the maximum
pricespace[profit==max(profit)]
```

A brute-force approach

A brute-force approach



- The optimal price is around 1.13.

Limitation of brute-force approach

- While the brute-force approach is simple and easy, it lacks scalability: it quickly becomes infeasible as the problem becomes complicated.
- Suppose that we set prices for *two* products, whose demand is given as follows.

$$\log(Q_1) = 12 - 8.7\log(P_1) + 0.1\log(P_2).$$
$$\log(Q_2) = 10 - 7.9\log(P_2) + 0.6\log(P_1).$$

- Suppose the cost is 1 for both products. The profit is then:

$$(P_1 - 1) \times \exp(12 - 8.7\log(P_1) + 0.1\log(P_2))$$
$$+ (P_2 - 1) \times \exp(10 - 7.9\log(P_2) + 0.6\log(P_1)).$$

- We need to find an optimal P_1 and P_2. Imagine solving this problem with a brute-force approach.

Limitation of brute-force approach

```
#Brute force with two dimensional prices
#Create two-dimensional price space
aux=seq(1,1.8,0.001)
pricespace=expand.grid(aux,aux)

#Evaluate demand at each point
logdemand1=12-8.7*log(pricespace[,1])+0.1*log(pricespace[,2])
logdemand2=10-7.9*log(pricespace[,2])+0.6*log(pricespace[,1])

#Calculate profit
profit=(pricespace[,1]-1)*exp(logdemand1)+(pricespace[,2]-1)*exp(logdemand2)

#Find profit-maximizing price
pricespace[profit==max(profit),]
```

- Coding-wise, it still appears feasible. The only complication is that the candidate price to evaluate the profit with is now (P_1, P_2), which "expand.grid" function creates.

Limitation of brute-force approach

- The real problem is its computational burden.
- Consider setting an optimal price for 5 products (P_1, P_2, ..., P_5). For each dimension, we have 30 candidate price points (say 1 dollar to 4 dollars, with a 10-cent increment).
- Then the number of all combination of candidate price points (at which we evaluate profit) is $30^5 = 24$ million.
- As the number of products and/or the number of candidate prices per product increases, required computation will quickly blow up. This phenomenon is called "curse of dimensionality", which makes any brute-force approach very much useless in many real-world situations.

Use of numerical solver

- Heavy computational burden arises because we evaluate profit *at all possible candidate price combination*, including obviously suboptimal ones.
- Instead, what if we can iteratively "search" for an optimal price?
  - Start from some initial guess of price, say $(P_1^0, P_2^0, ..., P_5^0)$.
  - Take another guess $(P_1^1, P_2^1, ..., P_5^1)$ around the initial guess, evaluate the profit there and compare it with the profit at the initial guess.
  - Keep the price guess that provides the higher profit.
  - Repeat this sequence long enough, until the profit no longer increases.
- Then we only evaluate profit around the current guess, potentially saving time.
- R is equipped with a function that does such iterative numerical computation, called "optim".

Defining profit as a function

- Let's consider our two-price example to implement optim function.
- What optim does is to take our profit function as an input, and search for the optimal price that maximizes it.
- Hence, to use optim, our first step is to have R recognize that:

$$(P_1 - 1) \times \exp(12 - 8.7 \log(P_1) + 0.1 \log(P_2))$$
$$+ (P_2 - 1) \times \exp(10 - 7.9 \log(P_2) + 0.6 \log(P_1))$$

  represents our profit function, and that we want to maximize it with P_1 and P_2.

"function" command in R

```
#Define profit as a function of prices
#The input "price" is a 2x1 object. The first argument is P1, and the second is P2.
profit=function(price){
        profit=-1*((price[1]-1)*exp(12-8.7*log(price[1])+0.1*log(price[2]))
                   +(price[2]-1)*exp(10-7.9*log(price[2])+0.6*log(price[1])))
        return(profit)
}
```

- We use "function" command to do this.
- In the middle, we write the profit expression as a function of "price", which has two elements (the first element is P_1 and the second is P_2), *and multiply everything by -1*.
- We then sandwich the expression with a bracket, and label as "function (name of the input)".
- "return(profit)" at the end clarifies that the output of this function is the profit expression - not necessary for this simple function.

"function" command in R

- Written this way, R recognizes the profit as a function.
- If we plug some values of the price into the function, it returns the value of our profit ($\times -1$).

```
> profit(c(1,2))
[1] -92.21633
> profit(c(1,1.5))
[1] -447.5009
```

- This is a big difference from the "profit" in the brute-force code. There "profit" was a vector of numbers.

Find an optimal price with optim function

```
#Find the optimal price, starting from the initial guess of P1=1, P2=1.
optim(c(1,1),profit)
```

- Once we define profit as a function of prices, we run "optim" function.
- The syntax is very simple - we specify an initial guess (in this case, $P_1 = P_2 = 1$) in the first argument, and provide the function to maximize in the second argument.
- In fact, "optim" function *minimizes* our objective function. This is why we put -1 in our profit function (the P that maximizes the profit $=$ the P that minimizes $-1\times$ profit).

Find an optimal price with optim function

```
> optim(c(1,1),profit)
$par
[1] 1.131494 1.159512

$value
[1] -8590.001

$counts
function gradient
     233       NA

$convergence
[1] 10
```

- It returns the optimal prices and the profit at that price level ($\times -1$).
- Note that we reached the optimum after 233 evaluations - much less than the brute-force approach.

optim function

- In this course, we only consider up to two products. So brute force should work just fine. Feel free to choose whichever you prefer.
- However, in any realistic environments, you likely have more than 2 products: you would almost always use some sort of numerical algorithms (optim is one of them).

Side: writing an expression as a function

- We used a "function" feature to write our profit as a function.
- This feature is also useful in other situations - suppose that you have an expression (e.g., demand, profit, etc) that you want to use multiple times in the code.
- By keeping the expression as a function, whenever you need it you just call that function in your code. You only need to write the full expression only once.

Use of analytical solution

- If you studied intermediate microeconomics, you have probably learned how to derive the profit-maximizing price analytically.
- "Analytically" - this approach doesn't require any numerical evaluation. In principle, you can find the optimal price with a paper and a pencil (though I won't recommend it - humans make mistakes).

Use of analytical solution

- Suppose, for simplicity, we estimate demand without logs. i.e. the demand takes the following form.

$$Q = \beta_0 + \beta_1 P$$

- Again, assume that the unit cost is one. Then the profit is given as follows.

$$\text{profit} = (P - 1)(\beta_0 + \beta_1 P)$$

Use of analytical solution

- Because the objective function is quadratic in P, there's a unique maximum in P.
- Moreover, it is known that the maximum equals the value of P such that the first-order derivative of the profit function with respect to P equals zero ("first-order condition").
- By taking the first-order condition, we have the following equation:

$$\begin{aligned} \beta_0 + 2\beta_1 P - \beta_1 &= 0 \\ \rightarrow P &= \frac{\beta_1 - \beta_0}{2\beta_1}. \end{aligned}$$

- Once you estimate the demand, you know the value of β_0 and β_1, and hence you know the optimal P from the expression above.

Pros and cons of analytical solution

- Because you can calculate the optimal P by just plugging in numbers to a known formula, no numerical operations involved - saves time and resources.
- However, to apply this approach, your demand specification must be simple, so that the profit function has an analytical solution.
- An innate trade-off: to eliminate computational burden, we need to give up on flexible demand specifications that may fit the data better.
- In practice, industry-leading firms prefer the "complex model, giant computer" approach - they have resources to spend on computation.

Summary: finding the right price

- We covered three different approaches to derive the optimal price from the estimated demand.
- They face different kinds of advantages and disadvantages. You may use different approaches across different occasions.
- In practice (once you are in a firm), because the demand model gets so complicated (with multiple products, multiple segments of consumers, etc), that a numerical solver may be pretty much the only option.