Pricing Analytics - Project 2

Shen, Ruiling; Tao, Chenxi; Liang, Jiawen; Tran, Khanh; Ding, Xiaodan

2/10/2020

Contents

Set-up

```
# Load necessary packages
library("dummies")
library("AER")
library("plotly")
library('RColorBrewer')
library("rgl")
library("data.table")
library("mlogit")
library("gmnl")
library("reshape")
library("cluster")
library("fpc")
library("factoextra")
library("gridExtra")
```

```
# Clear the global environment
rm(list = ls())

# Set the working directory
dir = "E:/Studying/Simon/Classes/MKT440 - Pricing Analytics/Project 2"
setwd(dir)

# Load data
data = fread("kiwi_bubbles_P2.csv", stringsAsFactors = F)
demo = fread("demo_P2.csv", stringsAsFactors = F)

# Drop observations with outliers
data = data[!(data$price.KB==99),]
data = data[!(data$price.KR==99),]
data = data[!(data$price.MB==99),]

# Variable used in the project
unit.cost = 0.50
market.size = 1000
```

1 Logit model without segmentation

1.1 Calculating elasticities

We firstly utilize the available choice data to estimate the model.

```
# pass the data to the algorithm
mlogitdata = mlogit.data(data,id="id",varying=4:7,choice="choice",shape="wide")
# run MLE.
mle = gmnl(choice ~ price, data = mlogitdata)
# store the coefficients
coef = mle$coefficients
# estimate the model
summary(mle)
```

```
##
## Model estimated on: Thu Feb 13 1:49:35 PM 2020
##
## Call:
## gmnl(formula = choice ~ price, data = mlogitdata, method = "nr")
##
## Frequencies of categories:
##
##       0      KB      KR      MB
## 0.41564 0.18035 0.20039 0.20362
##
## The estimation took: 0h:0m:0s
##
## Coefficients:
##                Estimate Std. Error z-value  Pr(>|z|)
## KB:(intercept)  4.25316    0.32821  12.959 < 2.2e-16 ***
## KR:(intercept)  4.36240    0.32945  13.241 < 2.2e-16 ***
## MB:(intercept)  4.20440    0.31331  13.419 < 2.2e-16 ***
## price          -3.73793    0.23671 -15.791 < 2.2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Optimization of log-likelihood by Newton-Raphson maximisation
## Log Likelihood: -1909
## Number of observations: 1547
## Number of iterations: 4
## Exit of MLE: gradient close to zero
```

Then we create two functions to calculate choice probability and price elasticity respectively for future reuse.

```
## Function to calculate choice probability
# Arguments:
#  prices: a list of price of each product in the market
#  coefs: a list of estimated coefficients of the model
#  productIdx: the index of the product (1=KB; 2=KR; 3=MB)
demand=function(prices, coef, productIdx){
  # get beta0 and beta1
  beta1 = coef[length(coef)]
```

```
  beta0 = coef[-length(coef)]
  # temporary variable
  temp = exp(beta0 + beta1*prices)
  # Calculate choice probability
  prob = as.numeric(temp[productIdx]/(1+sum(temp)))
  return(prob)
}

## Function to calculate price elasticity
elasticity.Func=function(prices, coef, productIdx, type="own"){
  beta1 = coef[length(coef)]
  prob = demand(prices, coef, productIdx)
  if (type=="own") {
    # own-price elasticity
    elasticity = beta1*prices[productIdx]*(1-prob)
  } else {
    # cross-price elasticity
    elasticity = -beta1*prices[productIdx]*prob
  }
}
```

Now we will calculate own- and cross-price elasticity for all product combinations at their average prices.

```
# set the price of products at the average prices observed in the data
avg.price.KB = mean(data$price.KB)
avg.price.KR = mean(data$price.KR)
avg.price.MB = mean(data$price.MB)
avg.prices = c(avg.price.KB, avg.price.KR, avg.price.MB)

# create a matrix to save elasticities
elasticity.3 = matrix(nrow=3, ncol=3)
rownames(elasticity.3) = c("KB", "KR", "MB")
colnames(elasticity.3) = c("KB", "KR", "MB")

# fill the matrix
for (i in 1:3) {
  for (j in 1:3) {
    if (i == j) {
      elasticity.3[i, j] = elasticity.Func(avg.prices, coef, i, type="own")
    } else {
      elasticity.3[i, j] = elasticity.Func(avg.prices, coef, j, type="cross")
    }
  }
}

# export the result
data.frame(elasticity.3)
```

```
##            KB        KR         MB
## KB -4.2578474  1.019923  0.9601564
## KR  0.9054743 -4.131270  0.9601564
## MB  0.9054743  1.019923 -4.0695469
```

We've found **three** patterns: 1. The baseline popularity of each product is quite similar; 2. When each product changes price, the percent change of itself is similar; 3. When one product changes price, the percent changes in choice probability of the other two products are the same.

```
product.share = data.table(KB=sum(data$choice=="KB"),KR=sum(data$choice=="KR"),MB=sum(data$choice=="MB"))
rownames(product.share) = "size"
product.share
```

```
##     KB  KR  MB
## 1: 279 310 315
```

Firstly, we find that the data used is more like a quota sample, and all products are selected roughly at the same rate (KB 279 times, KR 310 times and MB 315 times).

Secondly, we assume that demand is homogeneous without any segmentation of consumers, which means that consumers have similar preference toward each product.

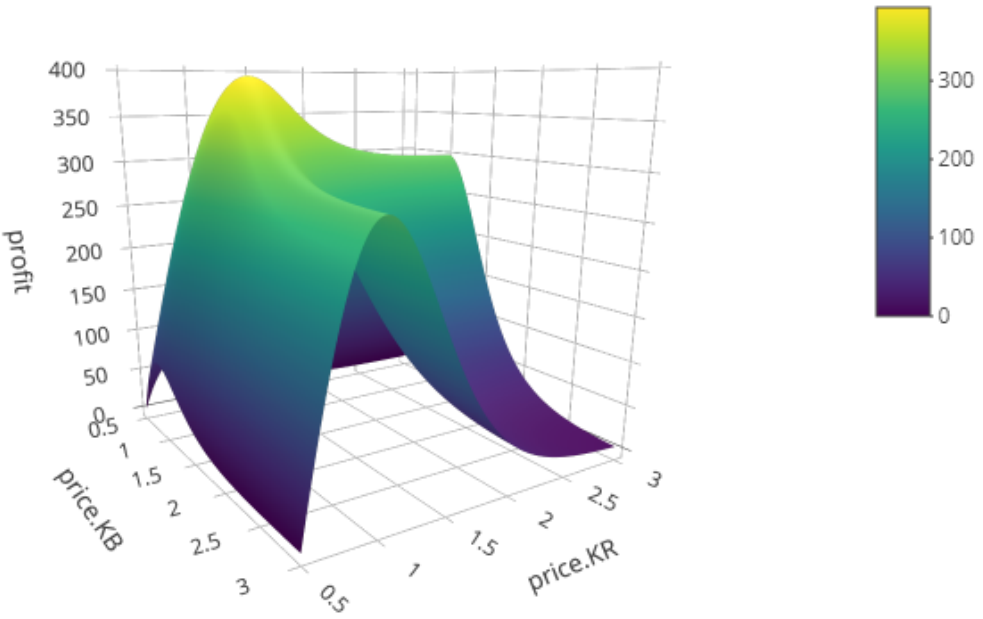
These basically are the reasons why our model has the patterns. However, we think it's **NOT reasonable**, because it conflicts with the reality and ignores different consumer preferences, which might leads to biased prediction and be detrimental to business decisions.

1.2 Product-line pricing

```
## Functions to calculate profit
noseg.profit.2D.Func = function (priceSpace.2D, rival.price, coef) {
  profitmat=matrix(0L, nrow(priceSpace.2D), 1)

  for (i in 1:nrow(priceSpace.2D)){
    # specify prices
    price.KB = priceSpace.2D[i, 1]
    price.KR = priceSpace.2D[i, 2]
    prices = c(price.KB, price.KR, rival.price)

    # calculate profit
    profitmat[i] = market.size*(demand(prices, coef, 1)*(prices[1]-unit.cost) +
                                demand(prices, coef, 2)*(prices[2]-unit.cost))
  }
  return(profitmat)
}

# Price space to search for optimal price
priceSpace.1D = seq(0.5, 3, 0.01)
priceSpace.2D = expand.grid(priceSpace.1D, priceSpace.1D)
price.MB = 1.43
profitmat = noseg.profit.2D.Func(priceSpace.2D, price.MB, coef)

# Plot (Uncomment to plot profits)
#profitMatrix = cbind(priceSpace.2D, profitmat)
#profitMatrix = as.matrix(cast(profitMatrix, Var1 ~ Var2, value="profitmat"))

#p <- plot_ly(x=priceSpace.1D, y=priceSpace.1D, z=profitMatrix) %>%
#  add_surface() %>%
```

```
#  layout(scene = list(xaxis = list(title = 'price.KB'),
#                      yaxis = list(title = 'price.KR'),
#                      zaxis = list(title = 'profit')))
#p

# Show max profit
max.profit.3 =
  list(max_profit=max(profitmat),
       optimal_price=as.numeric(priceSpace.2D[which.max(profitmat),]))
max.profit.3
```

```
## $max_profit
## [1] 393.4082
##
## $optimal_price
## [1] 1.16 1.16
```



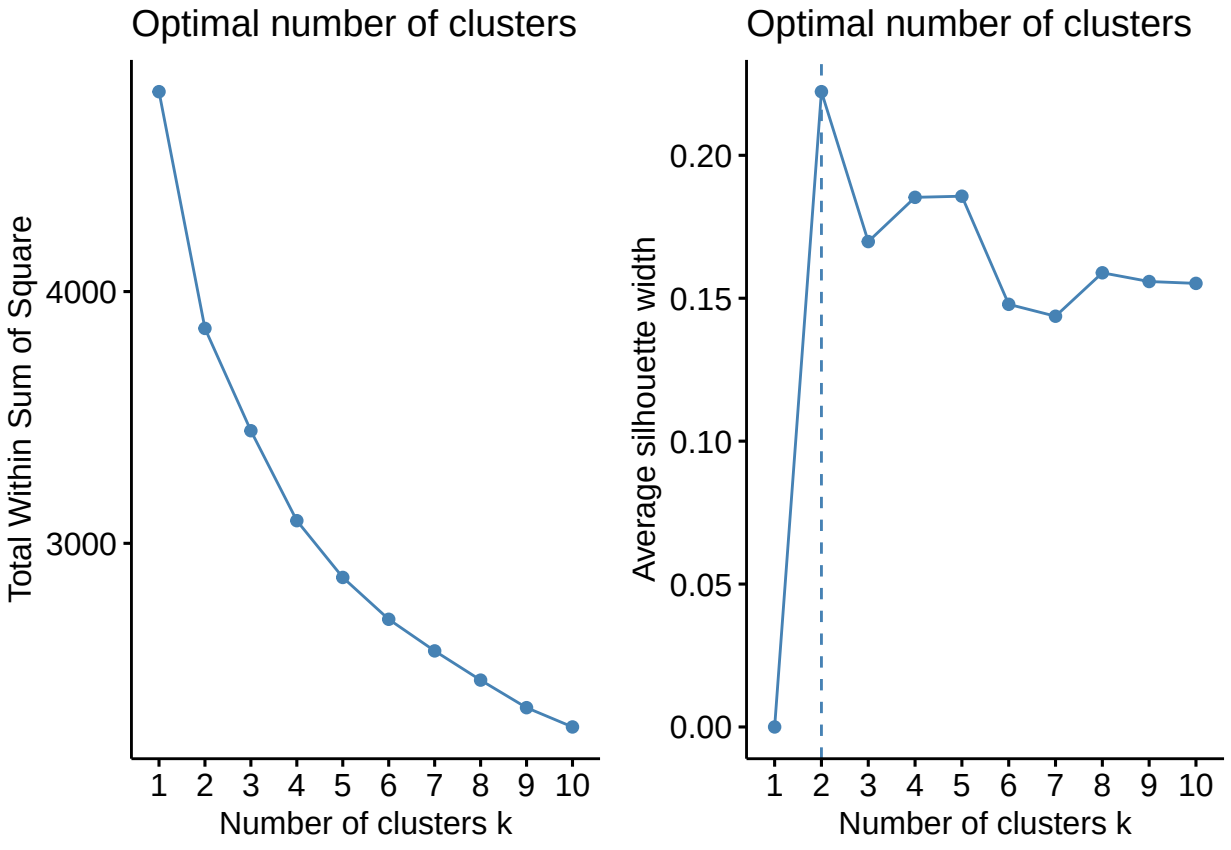
The optimal prices for both KB and KR are \$1.16, when the maximum profit is \$393.4082.

2 Logit model with segmentation

2.1 Clustering

In this section, we allow β_0^j and β_1 to differ across consumers. First, we use k-means clustering to group consumers into segments based on demographic information.

```r
# evaluate number of clusters to use on data with visualizations
clustTest = function(toClust,print=TRUE,scale=TRUE,maxClusts=15,seed=12345,nstart=20,iter.max=100){
  if(scale){ toClust = scale(toClust);}
  set.seed(seed);    # set random number seed before doing cluster analysis
  wss <- (nrow(toClust)-1)*sum(apply(toClust,2,var))
  for (i in 2:maxClusts) wss[i] <- sum(kmeans(toClust,centers=i,nstart=nstart,iter.max=iter.max)$within
  ##gpw essentially does the following plot using wss above.
  #plot(1:maxClusts, wss, type="b", xlab="Number of Clusters",ylab="Within groups sum of squares")
  gpw = fviz_nbclust(toClust,kmeans,method="wss",iter.max=iter.max,nstart=nstart,k.max=maxClusts) #alte
  pm1 = pamk(toClust,scaling=TRUE)
  ## pm1$nc indicates the optimal number of clusters based on
  ## lowest average silhoutte score (a measure of quality of clustering)
  #alternative way that presents it visually as well.
  gps = fviz_nbclust(toClust,kmeans,method="silhouette",iter.max=iter.max,nstart=nstart,k.max=maxClusts
  if(print){
    grid.arrange(gpw,gps, nrow = 1)
  }
  list(wss=wss,pm1=pm1$nc,gpw=gpw,gps=gps)
}

checkClust = clustTest(demo[,2:18],print=TRUE,scale=TRUE,maxClusts=10,
                       seed=12345,nstart=20,iter.max=100)
```



We plot *total within sum of square* over different numbers of clusters (from 2 to 10) to identify any "elbow". However, there is no "clear" elbow. In addition, the estimated coefficients among segments are not very different when the numbers of clusters are small. In order to see a significant difference among segments, we

increase the number of segments until several segments have very few people (3-4 people).

We finally choose 9 clusters to run the k-means algorithm, and have a total of 10 segments including consumers not belonging to any clusters.

The codes below group consumers into segments and show segments' shares.

```
# duplicate the choice data for segmentation
data.seg = data
# optimal number of clusters
n_clusters = 10
# clustering
demo_cluster = kmeans(x=demo[, 2:18], centers=n_clusters, nstart=1000)
# combine cluster identity into the raw data
cluster_id = data.frame(id=demo$id)
cluster_id$cluster = demo_cluster$cluster
data.seg = merge(data.seg, cluster_id, by="id", all.x = T)

# for those who don't fit in any cluster, group them into one additional cluster
n_clusters = n_clusters + 1
data.seg$cluster[is.na(data.seg$cluster)] = n_clusters

# calculate segment share
N = 359 # number of individuals
seg.share = c(table(demo_cluster$cluster),
              N - sum(table(demo_cluster$cluster))) / N
data.frame(seg.share)
```

```
##     seg.share
## 1  0.06128134
## 2  0.10863510
## 3  0.13370474
## 4  0.05292479
## 5  0.10584958
## 6  0.08077994
## 7  0.08077994
## 8  0.06685237
## 9  0.01114206
## 10 0.08635097
##    0.21169916
```

Now we estimate coefficients for each segment.

```
# store the coefficients in a data frames
coef.est = data.frame(segment = 1:n_clusters, intercept.KB = NA,
                      intercept.KR = NA, intercept.MB = NA,
                      price.coef = NA)

# write a for-loop.
for (seg in 1:n_clusters) {
  data.sub = subset(data.seg, cluster==seg)
  mlogitdata=mlogit.data(data.sub,id="id",varying=4:7,choice="choice",shape="wide")
  mle= gmnl(choice ~  price, data = mlogitdata)
  # store the outcome in the coef.est matrix.
```

```
  coef.est[seg, 2:5] = mle$coefficients
}

coef.est
```

```
##    segment intercept.KB intercept.KR intercept.MB price.coef
## 1        1    3.0878338     4.006679    3.1802458  -2.826519
## 2        2    2.3336806     3.112574    2.9252012  -2.896447
## 3        3    3.0017626     3.916981    2.7620620  -2.951180
## 4        4    6.6556359     6.515320    6.4858148  -5.043962
## 5        5    7.6063946     6.661934    7.4775392  -5.897474
## 6        6    4.7911334     4.456635    5.6901102  -4.606762
## 7        7    0.8341597     1.673929    0.4294365  -1.233209
## 8        8    8.2287472     6.680553    8.1223860  -6.788933
## 9        9    7.0950794     8.325625    7.4949559  -6.299830
## 10      10    3.9972969     3.958938    3.8837551  -3.715366
## 11      11    5.1174301     4.509340    4.5449628  -4.062526
```

2.2 Calculating elasticities

From the estimated coefficients of each segment, we will calculate aggregated choice probability and elasticity.

```
## Function to calculate aggregate choice
agg_choice=function(prices, coef.df=coef.est,
                    seg.share=seg.share, productIdx=1) {
  # number of segments
  num_segments = length(coef.df$segment)
  # number of coefs' columns
  num_columns = length(coef.df)

  # calculate aggregate choice
  agg_prob = 0
  for (i in 1:num_segments) {
    coef = as.numeric(coef.df[i, 2:num_columns])
    choice = demand(prices, coef, productIdx)
    agg_prob = agg_prob + seg.share[i]*choice
  }

  return(agg_prob)
}
```

We use both numerical approach and formula approach to calculate elasticities. Both approaches yield the same results.

Numerical approach:

```
## Function to numerically calculate elasticity
num.agg.Elasticity.Func = function(prices, coef.df, seg.share,
                                   productIdx_in, productIdx_out,
                                   delta_price=1e-3) {
    # old demand
    old_demand = agg_choice(prices, coef.df, seg.share, productIdx_out)
```

```
    # change price a small amount
    new.prices = prices
    new.prices[productIdx_in] = prices[productIdx_in] + delta_price

    # new demand after a small price change
    new_demand = agg_choice(new.prices, coef.df, seg.share, productIdx_out)

    # calculate elasticity
    delta_demand = new_demand - old_demand
    elasticity = (delta_demand / old_demand) /
                   (delta_price / prices[productIdx_in])

    return(elasticity)
}

# calculate elasticities and save them in a matrix
elasticity.4 = matrix(nrow=3, ncol=3)
rownames(elasticity.4) = c("KB", "KR", "MB")
colnames(elasticity.4) = c("KB", "KR", "MB")

for (i in 1:3) {
  for (j in 1:3) {
    elasticity.4[i, j] = num.agg.Elasticity.Func(avg.prices, coef.est,
                                                 seg.share,
                                                 j, i, 1e-10)
  }
}

elasticity.4
```

```
##            KB         KR         MB
## KB -4.5424147  0.8888050  1.1187790
## KR  0.8045427 -3.5474243  0.7999291
## MB  1.0968680  0.8663993 -4.4773646
```

Formula approach:

```
## Function to calculate elasiticy using formula approach
agg.Elasticity.Func=function(prices, coef.df, seg.share,
                             productIdx_in, productIdx_out,
                             type="own"){
  # number of segments
  num_segments = length(coef.df$segment)
  # number of coefs' columns
  num_columns = length(coef.df)

  # calculate temporary variable
  sum_temp = 0
  for (i in 1:num_segments) {
    # get segment coefficient
    coef = as.numeric(coef.df[i, 2:num_columns])
    beta1 = coef[length(coef)]
```

```
    # calculate choice probability
    prob_in = demand(prices, coef, productIdx_in)
    prob_out = demand(prices, coef, productIdx_out)

    if (type == "own") {
      temp = -beta1*prob_out*(1-prob_out)
    } else {
      temp = beta1*prob_out*prob_in
    }

    sum_temp = sum_temp + seg.share[i]*temp
  }

  agg_prob_out = agg_choice(prices, coef.df, seg.share, productIdx_out)
  elasticity = -prices[productIdx_in]/agg_prob_out * sum_temp

  return(elasticity)
}

# calculate elasticities and save them in a matrix
elasticity.4 = matrix(nrow=3, ncol=3)
rownames(elasticity.4) = c("KB", "KR", "MB")
colnames(elasticity.4) = c("KB", "KR", "MB")

for (i in 1:3) {
  for (j in 1:3) {
    if (i == j) {
      # Calculate own-price elasticity
      elasticity.4[i, j] = agg.Elasticity.Func(avg.prices, coef.est,
                                               seg.share, j, i,
                                               type="own")
    } else {
      # Calculate cross-price elasticity
      elasticity.4[i, j] = agg.Elasticity.Func(avg.prices, coef.est,
                                               seg.share, j, i,
                                               type="cross")
    }
  }
}

elasticity.4
```

```
##            KB         KR         MB
## KB -4.5424126  0.8888025  1.1187788
## KR  0.8045405 -3.5474257  0.7999274
## MB  1.0968657  0.8663975 -4.4773662
```

We can see some differences in elasticity from when we didn't do segmentation. Own-price elasticity of KB and MB is larger while that of KR is smaller. Cross-price elasticity between KB and KR is a lot smaller, while that of KB and MG is larger. By doing segmentation, we can clearly see the substitution effect among products.

Which products are closer substitutes?

```
avg.elasticity = c(KB_KR=mean(c(elasticity.4[1, 2], elasticity.4[2, 1])),
                   KB_MB=mean(c(elasticity.4[1, 3], elasticity.4[3, 1])),
                   KR_MB=mean(c(elasticity.4[2, 3], elasticity.4[3, 2])))
avg.elasticity
```

```
##     KB_KR     KB_MB     KR_MB
## 0.8466715 1.1078222 0.8331624
```

After segmentation, the average cross elasticity between KB and MB is the largest, which means that they are closer substitutes. The cross elasticity between MB and KR is the smallest, so they are not as close substitutes. Therefore, launching KB has a strong impact on our competitor, and the cannibalization effect of launching KB on KR is not a big issue since the cross price elasticity between KB and KR is smaller than 1.

How does the underlying customer segmentation explain the substitution pattern you see in the elasticity?

Below are the coefficients for our 10 segments

```
coef.est
```

```
##    segment intercept.KB intercept.KR intercept.MB price.coef
## 1        1    3.0878338     4.006679    3.1802458  -2.826519
## 2        2    2.3336806     3.112574    2.9252012  -2.896447
## 3        3    3.0017626     3.916981    2.7620620  -2.951180
## 4        4    6.6556359     6.515320    6.4858148  -5.043962
## 5        5    7.6063946     6.661934    7.4775392  -5.897474
## 6        6    4.7911334     4.456635    5.6901102  -4.606762
## 7        7    0.8341597     1.673929    0.4294365  -1.233209
## 8        8    8.2287472     6.680553    8.1223860  -6.788933
## 9        9    7.0950794     8.325625    7.4949559  -6.299830
## 10      10    3.9972969     3.958938    3.8837551  -3.715366
## 11      11    5.1174301     4.509340    4.5449628  -4.062526
```

From the estimated coefficients, in almost all segments (7/10 segments), β_0^{KB} and β_0^{MB} are closer than with β_0^{KR}, showing that the popularity of KB and MB are closer than with KR. This result agrees with our analysis of cross-price elasticity: KB and MB are closer substitutes.

Where should KB be positioned?

```
which(apply(coef.est[,2:4], 1, which.max) == 1)
```

```
## [1]  4  5  8 10 11
```

The intercept of KB is the largest among 3 products in segment **1, 4, 5, 10**. This means Kiwi Bubble is the most popular product in these segments. Thus KB should be positioned in these segments.

2.3 Optimal price

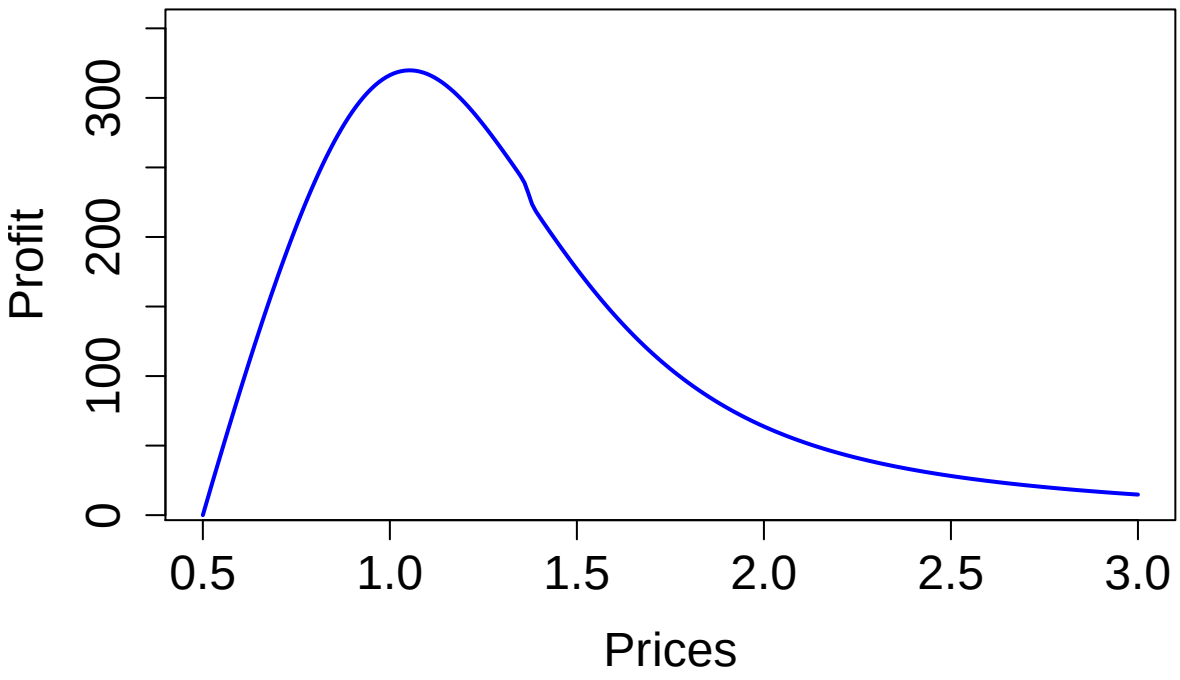
What is the optimal price of KR if we do not launch KB?

```r
# subset data with no KB
data.noKB = data.seg[data.seg$choice!="KB"]
data.noKB = data.noKB[, -c("price.KB")]

# store results in a data frame
coef.est.noKB = data.frame(segment = 1:n_clusters,
                      intercept.KR = NA, intercept.MB = NA,
                      price.coef = NA)

# write a for-loop to calculate segment coefficients
for (seg in 1:n_clusters) {
  data.sub = subset(data.noKB, cluster==seg)
  mlogitdata=mlogit.data(data.sub,id="id",varying=4:6,
                         choice="choice",shape="wide")
  mle= gmnl(choice ~  price, data = mlogitdata)
  # store the outcome in the coef.est matrix.
  coef.est.noKB[seg, 2:4] = mle$coefficients
}

# segment coefficients
coef.est.noKB
```

```
##    segment intercept.KR intercept.MB  price.coef
## 1        1     5.279829    4.3086700   -3.697935
## 2        2     3.343260    3.1358652   -3.062659
## 3        3     5.151201    3.9382864   -3.833536
## 4        4     6.330840    6.2638125   -4.864189
## 5        5     7.610561    8.2757855   -6.534792
## 6        6     5.188460    6.4282212   -5.135500
## 7        7     2.049520    0.7537445   -1.493157
## 8        8     6.288256    7.7420125   -6.477838
## 9        9   278.722170  265.3997493 -203.326224
## 10      10     5.057727    4.8681058   -4.505897
## 11      11     6.627696    6.5770453   -5.615066
```

```r
## Function to find profit
agg.profit.Func = function(prices, coef.df, seg.share, productIdx) {
  # calculate aggregate probability
  agg_prob = agg_choice(prices, coef.df, seg.share, productIdx)

  # calculate profit
  profit = market.size*agg_prob*(prices[productIdx] - unit.cost)

  return(profit)
}

## Function to find maximum profit
max.profit.1D.Func = function (priceSpace, rival.price,
                               coef.df, seg.share,
                               productIdx, plot=FALSE) {

  # number of products we are analyzing.
  num_products = length(names(coef.df)) - 2
```

```
  # specify 'prices' vector
  if (productIdx == num_products) {
    if (length(rival.price) == 2) {
      # if productIdx is MB, rival.price is price.KB and price.KR
      prices = cbind(rival.price[1], rival.price[2], priceSpace)
    } else {
      # if there is no KB
      prices = cbind(rival.price, priceSpace)
    }
  } else {
    prices = cbind(priceSpace, rival.price)
  }

  # calculate profit
  profit = apply(prices, 1,
                 function (x) agg.profit.Func(x, coef.df,
                                              seg.share,
                                              productIdx))

  # plot profit
  if (plot == TRUE) {
    plot(priceSpace,profit,type='l',xlab='Prices',
       ylab='Profit',ylim=c(10,350),col="blue",lwd=2,
       cex=2,cex.lab=1.5, cex.axis=1.5, cex.main=1.5, cex.sub=1.5)
  }

  # return max profit and price maximizing profit
  res = list(max_profit=max(profit),
             optimal_price=priceSpace[profit==max(profit)])

  return(res)
}

# specify price space
price.MB = 1.43
priceSpace.KR = seq(0.5,3,0.01)
res.noKB = max.profit.1D.Func(priceSpace.KR, price.MB,
                              coef.est.noKB, seg.share,
                              productIdx=1, plot=TRUE)
```




```
res.noKB
```

```
## $max_profit
## [1] 319.8152
##
## $optimal_price
## [1] 1.05
```

When we do not launch KB and MB's price is \$1.43, the optimal price for KR is **\$1.06**. At this price we yield the maximum profit of **\$325.19**.

What is the optimal price of KB and KR if we launch KB?

```
## Function to calculate profit
agg.profit.2D.Func = function (priceSpace.2D, rival.price,
                               coef.df=coef.est, seg.share) {

  profitmat=matrix(0L, nrow(priceSpace.2D), 1)

  for (i in 1:nrow(priceSpace.2D)){
    # specify prices
    price.KB = priceSpace.2D[i, 1]
    price.KR = priceSpace.2D[i, 2]
    prices = c(price.KB, price.KR, rival.price)

    # calculate profit
```

```
    profitmat[i] = agg.profit.Func(prices, coef.df, seg.share, 1) +
      agg.profit.Func(prices, coef.df, seg.share, 2)
  }

  return(profitmat)
}

## Function to find max profit
max.profit.2D.Func = function(priceSpace.2D, rival.price,
                              coef.df=coef.est, seg.share) {

  # calculate profitmat
  profitmat = agg.profit.2D.Func(priceSpace.2D, rival.price,
                                 coef.df, seg.share)

  # find optimal price and maximum profit
  optimal.prices = as.numeric(priceSpace.2D[which.max(profitmat),])
  res = list(max_profit=max(profitmat), optimal_price=optimal.prices)
  return(res)
}

# choose space of prices to search for the optimal price over
aux = seq(0.5, 1.5, 0.01)
priceSpace.2D = expand.grid(aux,aux)

# search for maximum price
res = max.profit.2D.Func(priceSpace.2D, price.MB, coef.df=coef.est, seg.share)
res
```

```
## $max_profit
## [1] 397.9532
##
## $optimal_price
## [1] 1.14 1.20
```

When we launch KB and MB's price is \$1.43, the optimal price for KB and KR is **\$1.15** and **\$1.19** respectively. At this price we yield the maximum profit of **\$395.37**.

How does the profit of Kiwi and Mango change as Kiwi launches KB?

Our profit changes from \$325.19 to \$395.37. Rival profit doesn't change much when we launch KB. The reason for that could be KB attracts the portion of consumers that did not buy any products before.

```
# when not launch KB
before.profit.MB = agg.profit.Func(prices=c(1.06, 1.43),
                                   coef.df=coef.est.noKB,
                                   seg.share, productIdx=2)
# when launch KB
after.profit.MB = agg.profit.Func(prices=c(1.16, 1.16, 1.43),
                                  coef.df=coef.est,
                                  seg.share, productIdx=3)
# change in MB's profit
data.frame(before=before.profit.MB,after=after.profit.MB,
           difference=after.profit.MB - before.profit.MB)
```

```
##    before    after difference
## 1 87.0182 85.57827  -1.439927
```

Does the model justify the launch of KB? If we do not launch kiwi bubble, 3 segments prefer Mango Bubble, and other segments prefer Kiwi Regular. After we launch Kiwi Bubble, because Kiwi Bubble and Mango Bubble are closer substitutes, only 1 segment would still be attracted by Mango Bubble, and we get customers in the other 2 segments to buy our newly launched Kiwi Bubble. Therefore, the market share of our products will increase when we launch Kiwi Bubble, resulting in a higher profit.

3 Understanding strategic reponses

When we do not launch Kiwi Bubble:

```
## Function to find equilibrium
response.noKB.Func = function(original_prices,
                              coef.df, seg.share,
                              num_iters=5) {

  # price space to search for optimal price
  priceSpace.1D = seq(0.5, 1.5, 0.01)

  # save results in a data frame
  response.Df = data.frame(iter=1:num_iters,
                           price.KR=NA, price.MB=NA,
                           our.profit=NA, rival.profit=NA)
  # original prices
  prices = original_prices

  for (i in 1:num_iters) {
    ## find our optimal prices
    our.res = max.profit.1D.Func(priceSpace.1D, rival.price=prices[2],
                                 coef.df, seg.share, productIdx=1)
    # update our prices
    prices[1] = our.res[[2]]

    ## rival's response
    rival.res = max.profit.1D.Func(priceSpace.1D, rival.price=prices[1],
                                   coef.df, seg.share, productIdx=2)
    # update rival's price
    prices[2] = rival.res[[2]]

    ## our profit after rival's response
    # our.profit = agg.profit.Func(prices, coef.df, seg.share, 1)
    our.profit = our.res[[1]]

    ## save responses
    response.Df[i, 2:5] = c(price.KR = prices[1],
                            price.MB = prices[2],
                            our.profit = our.profit,
                            rival.profit = rival.res[[1]])
  }
  return(response.Df)
}
```

```
# original prices
original_prices = c(price.KR=1, price.MB=1.43)
response.Df = response.noKB.Func(original_prices, coef.est.noKB,
                           seg.share, num_iters=5)
response.Df
```

```
##   iter price.KR price.MB our.profit rival.profit
## 1    1     1.05     0.96   319.8152     215.6460
## 2    2     0.95     0.88   203.6189     181.3013
## 3    3     0.92     0.91   177.3748     170.9030
## 4    4     0.94     0.87   187.1958     177.4754
## 5    5     0.92     0.91   174.0242     170.9030
```

When we choose not to launch kiwi bubble, if our competitor strategically responds to our pricing strategy, we will walk into a price war, in the sense that our price will never converge to nash equilibrium prices. In the analysis, we find out that if our competitor sets the Mango Bubble (MB) price at 0.87 we will set the price at 0.92 for Kiwi Regular (KR); however, when we set the KR price at 0.92, they will set the MB price at 0.91 accordingly. But when the MB price is at 0.91, our optimal strategy is to set the KR price at 0.94, yet the optimal MB price regarding is 0.87. So the price war falls into an endless loop, leaving neither us nor our competitor reaching an optimal profit.

When we launch Kiwi Bubble:

```
## Function to find equilibrium
response.Func = function(original_prices,
                         coef.df, seg.share, num_iters=5) {

  # price space to search for optimal price
  priceSpace.1D = seq(0.8, 1.1, 0.01)
  priceSpace.2D = expand.grid(priceSpace.1D, priceSpace.1D)

  # original prices
  prices = original_prices

  # save results in a data frame
  response.Df = data.frame(iter=1:num_iters, price.KB=NA,
                           price.KR=NA, price.MB=NA,
                           our.profit=NA, rival.profit=NA)
  for (i in 1:num_iters) {
    ## find our optimal prices
    our.res = max.profit.2D.Func(priceSpace.2D, rival.price=prices[3],
                                 coef.df=coef.est, seg.share)
    # update our prices
    prices[1:2] = our.res[[2]]

    ## rival's response
    rival.res = max.profit.1D.Func(priceSpace.1D, rival.price=prices[1:2],
                                   coef.est, seg.share,
                                   productIdx=3)
    # update rival's price
    prices[3] = rival.res[[2]]

    ## our profit after rival's response
```

```
    our.profit = agg.profit.Func(prices, coef.df, seg.share, 1) +
      agg.profit.Func(prices, coef.df, seg.share, 2)

    ## save responses
    response.Df[i, 2:6] = c(price.KB = prices[1],
                            price.KR = prices[2],
                            price.MB = prices[3],
                            our.profit = our.profit,
                            rival.profit = rival.res[[1]])
  }

  return(response.Df)
}

# original prices
original_prices = c(price.KB=1, price.KR=1, price.MB=1.43)
response.Df = response.Func(original_prices, coef.est,
                            seg.share, num_iters=5)
response.Df
```

```
##   iter price.KB price.KR price.MB our.profit rival.profit
## 1    1     1.10     1.10     0.93   260.9151     165.9545
## 2    2     0.99     1.09     0.91   259.3435     145.5865
## 3    3     0.99     1.08     0.91   259.3677     144.6630
## 4    4     0.99     1.08     0.91   259.3677     144.6630
## 5    5     0.99     1.08     0.91   259.3677     144.6630
```

However, if we choose to launch the Kiwi Bubble (KB). With my competitor's strategic response in our analysis, we will reach a nash equilibrium with KB price: 1.01, KR price: 1.07, and MB price: 0.91. In this nash equilibrium, we can reach an optimal profit of 260.82 and my competitor can reach the optimal profit of 144.49.

Difference in the strategic advantage of launching Kiwi Bubble

At question 4, we assume that our competitor would not strategically react to our price change; therefore, in that case, our strategic advantage of launching Kiwi bubble is to capture more market share and ultimately increase our profit. Nevertheless, considering the strategic response of my competitor, our strategic advantage of launching Kiwi bubble shall be to avoid the price war, where neither me nor my competitor can reach a nash equilibrium, and to reach the nash equilibrium price and reach an optimal profit under that circumstance.