Code

- Show All Code
- Hide All Code
-
- Download Rmd

Product Line Optimization Using Conjoint Analysis

Asim Ansari

This is a toy example to illustrate how a firm can optimize a product line using conjoint resuls. Imagine that we conducted a conjoint analysis on a sample of customers and based on the estimated part-worths, we are able to calculate the overall utilities for each member of our sample, for any product profile. Also, suppose that the firm's R&D has identified a set of candidate products that could be produced, and the firm wants to choose a subset of three products that it wants to introduce. In this notebook, we will set up the optimization problem and show how R can be used to solve this problem. In particular, we will use Genetic algorithms to solve this optimization problem.

We begin by reading the data file that contains the overall utilities of 10 customers for 7 products. In reality, conjoint data are available for a large sample of subjects, but here, we restrict the analysis to a very small subset to explain how things are done.

The first product is a status-quo product, which can be understood as a product that is already in the market. The firm is interested in introducing a line of products to compete against the status-quo product. The other six products are concepts that the firm's R&D has developed, and the firm is interested in introducing a product line of 3 products.

```
utilsData <- read.csv("/home/maa48/Dropbox (CBS)/teaching/MarketingAnalytics/conjoint/productUtils.csv")
```

Given below are the profit margins for the six product concepts, excluding the status-quo product. This assumes that the firm has already set the prices for these product concepts, and the only decision is about which 3 to introduce as part of its product line.

```
margin<-c(8,7,8,6,9,7)
```

We convert the R dataframe into a matrix to facilitate further computation.

```
utils<-data.matrix(utilsData)
dim(utils)
```

```
[1] 10  7
```

We now write a function that computes the total profits for the firm, given a product line. The product line is given via the function argument named "offered". This is a vector of binary indicators, one for each of the 6 products, such that the indicator of a product takes a value of 1 if the firm includes it in the product line, and 0, otherwise. For example, if offered is given by c(1,0,1,1,0,0), then it means that the first, the third and the fourth product form the product line.

Given a product line that is available in the market, consumers are assumed to compare the offered products with the status-quo and purchase the product with the hightest utility. Thus each consumer buys a single product, either the status-quo, or one from the product line.

```
profit<-function(offered, utils, margin){
  
  offered1<-c(1, offered)
  
  uOffered <- t(apply(utils, 1, function(x)(x * offered1)))
  maxUtil <- apply(uOffered, 1, max)
  
  prodHighest<-matrix(0, nrow(utils), ncol(utils)-1)
  
  for(i in 1:nrow(utils)){
    for(j in 2:ncol(utils)){
      if(uOffered[i, j] == maxUtil[i]) {prodHighest[i,j-1]=1;break}
    }
  }
  profitVec<-apply(prodHighest, 1, function(x){x %*% margin})
  sum(profitVec)
}
```

We can evaluate the profit function for specific values of its arguments. We see from below that when the 1st, fifth and sixth product are offered to the market as a product line, the firm can get joint profits of 55, where as, if it introduces the first and the fifth, it makes a profit of 51.

```
profit(c(1,0, 0, 0, 1,1),utils, margin)
```

```
[1] 55
```

```
profit(c(1,0, 0, 0, 1,0),utils, margin)
```

```
[1] 51
```

The firm is interested in finding the optimal product line of numProd (e.g., 3) products that maximizes its profits for this market of customers. This is a constrained optimization problem where the decision variables are binary, and the constraint is such that the decision variables should sum to numProd. Given that we are dealing with binary variables, we use genetic algorithms to solve this problem, using the GA library in R. The constraint is incorporated via a penalty term, which penalizes potential solutions if the number of products offered varies from the desired number of products, i.e., numProd. We therefore use the following objective function.

```
obj<-function(offered, utils, margin, numProd){
  
  pr<-profit(offered, utils, margin)
  penalty<-10*max(margin)*abs(sum(offered)-numProd)
  
  pr-penalty
}
obj(c(1,1, 1, 0, 0,1),utils, margin, 4 )
```

```
[1] 65
```

We can see below that if we deviate from the desired number of products, the penalty kicks in and the objective function value is reduced significantly.

```
obj(c(1,1, 1, 0, 0,1),utils, margin, 3 )
```

```
[1] -25
```

We now use the GA package to perform the optimization.

```
library("GA")
gaOpt<-ga("binary", fitness=obj, utils=utils, margin=margin, numProd=3, nBits=length(margin), maxiter=100)
```

```

GA | iter = 1
Mean = -21.14 | Best =  77.00

GA | iter = 2
Mean =  4.38 | Best = 77.00

GA | iter = 3
Mean = 15.52 | Best = 77.00
```

```
GA | iter = 4
Mean = 31.64 | Best = 77.00

GA | iter = 5
Mean = 38.96 | Best = 77.00

GA | iter = 6
Mean = 51.8 | Best = 77.0

GA | iter = 7
Mean = 62.92 | Best = 77.00

GA | iter = 8
Mean = 65.56 | Best = 77.00

GA | iter = 9
Mean = 68.86 | Best = 77.00

GA | iter = 10
Mean = 69.06 | Best = 77.00

GA | iter = 11
Mean = 71.42 | Best = 77.00

GA | iter = 12
Mean = 67.44 | Best = 77.00

GA | iter = 13
Mean = 65.46 | Best = 77.00

GA | iter = 14
Mean = 72.58 | Best = 77.00

GA | iter = 15
Mean = 73.08 | Best = 77.00

GA | iter = 16
Mean = 59.16 | Best = 77.00

GA | iter = 17
Mean = 62.84 | Best = 77.00

GA | iter = 18
Mean = 69.46 | Best = 77.00

GA | iter = 19
Mean = 65.22 | Best = 77.00

GA | iter = 20
Mean = 68.28 | Best = 77.00

GA | iter = 21
Mean = 63.24 | Best = 77.00

GA | iter = 22
Mean = 71.2 | Best = 77.0

GA | iter = 23
Mean = 67.78 | Best = 77.00

GA | iter = 24
Mean = 72.8 | Best = 77.0

GA | iter = 25
Mean = 62.58 | Best = 77.00

GA | iter = 26
Mean = 63.48 | Best = 77.00

GA | iter = 27
Mean = 62.82 | Best = 77.00

GA | iter = 28
Mean = 62.7 | Best = 77.0

GA | iter = 29
Mean = 69.48 | Best = 77.00

GA | iter = 30
Mean = 70.62 | Best = 77.00

GA | iter = 31
Mean = 70.54 | Best = 77.00

GA | iter = 32
Mean = 69.38 | Best = 77.00

GA | iter = 33
Mean = 66.76 | Best = 77.00

GA | iter = 34
Mean = 70.98 | Best = 77.00

GA | iter = 35
Mean = 69.2 | Best = 77.0

GA | iter = 36
Mean = 70.48 | Best = 77.00

GA | iter = 37
Mean = 73 | Best = 77

GA | iter = 38
Mean = 59.72 | Best = 77.00

GA | iter = 39
Mean = 61.34 | Best = 77.00

GA | iter = 40
Mean = 65.22 | Best = 77.00

GA | iter = 41
Mean = 68.42 | Best = 77.00

GA | iter = 42
Mean = 60.64 | Best = 77.00

GA | iter = 43
Mean = 70.88 | Best = 77.00

GA | iter = 44
Mean = 63.36 | Best = 77.00

GA | iter = 45
Mean = 61.48 | Best = 77.00

GA | iter = 46
```

```
Mean = 64.84 | Best = 77.00

GA | iter = 47
Mean = 66.14 | Best = 77.00

GA | iter = 48
Mean = 67.24 | Best = 77.00

GA | iter = 49
Mean = 65.46 | Best = 77.00

GA | iter = 50
Mean = 64.96 | Best = 77.00

GA | iter = 51
Mean = 63.94 | Best = 77.00

GA | iter = 52
Mean = 60.34 | Best = 77.00

GA | iter = 53
Mean = 66.94 | Best = 77.00

GA | iter = 54
Mean = 68.8 | Best = 77.0

GA | iter = 55
Mean = 68.82 | Best = 77.00

GA | iter = 56
Mean = 68.82 | Best = 77.00

GA | iter = 57
Mean = 62.56 | Best = 77.00

GA | iter = 58
Mean = 68.6 | Best = 77.0

GA | iter = 59
Mean = 58.18 | Best = 77.00

GA | iter = 60
Mean = 59.16 | Best = 77.00

GA | iter = 61
Mean = 63.02 | Best = 77.00

GA | iter = 62
Mean = 58.9 | Best = 77.0

GA | iter = 63
Mean = 59.7 | Best = 77.0

GA | iter = 64
Mean = 69.12 | Best = 77.00

GA | iter = 65
Mean = 66.56 | Best = 77.00

GA | iter = 66
Mean = 56.56 | Best = 77.00

GA | iter = 67
Mean = 59.66 | Best = 77.00

GA | iter = 68
Mean = 64.7 | Best = 77.0

GA | iter = 69
Mean = 67.06 | Best = 77.00

GA | iter = 70
Mean = 71.2 | Best = 77.0

GA | iter = 71
Mean = 73.22 | Best = 77.00

GA | iter = 72
Mean = 68.76 | Best = 77.00

GA | iter = 73
Mean = 68.6 | Best = 77.0

GA | iter = 74
Mean = 60.9 | Best = 77.0

GA | iter = 75
Mean = 64.48 | Best = 77.00

GA | iter = 76
Mean = 61.18 | Best = 77.00

GA | iter = 77
Mean = 68.58 | Best = 77.00

GA | iter = 78
Mean = 74.52 | Best = 77.00

GA | iter = 79
Mean = 64.48 | Best = 77.00

GA | iter = 80
Mean = 69.18 | Best = 77.00

GA | iter = 81
Mean = 66.06 | Best = 77.00

GA | iter = 82
Mean = 67.24 | Best = 77.00

GA | iter = 83
Mean = 60 | Best = 77

GA | iter = 84
Mean = 71.08 | Best = 77.00

GA | iter = 85
Mean = 74.82 | Best = 77.00

GA | iter = 86
Mean = 75.04 | Best = 77.00

GA | iter = 87
Mean = 64.54 | Best = 77.00

GA | iter = 88
Mean = 65.14 | Best = 77.00
```

```
GA | iter = 89
Mean = 64.7 | Best = 77.0

GA | iter = 90
Mean = 69.32 | Best = 77.00

GA | iter = 91
Mean = 64.98 | Best = 77.00

GA | iter = 92
Mean = 66.36 | Best = 77.00

GA | iter = 93
Mean = 66.94 | Best = 77.00

GA | iter = 94
Mean = 66.52 | Best = 77.00

GA | iter = 95
Mean = 65.54 | Best = 77.00

GA | iter = 96
Mean = 64.34 | Best = 77.00

GA | iter = 97
Mean = 69.1 | Best = 77.0

GA | iter = 98
Mean = 58.1 | Best = 77.0

GA | iter = 99
Mean = 65.22 | Best = 77.00

GA | iter = 100
Mean = 70.38 | Best = 77.00
```

We can see from below that the optimal objective function value is 77, and the optimal solution involves launching 3 products, namely 1, 2 and 5.

```
summary(gaOpt)
```

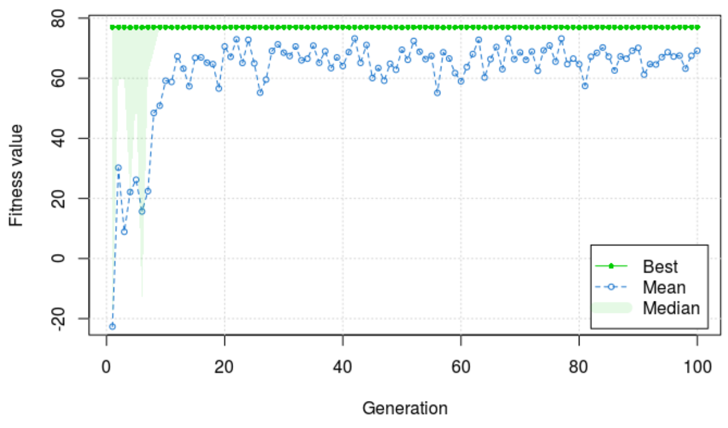
```
+-----------------------------------+
|         Genetic Algorithm         |
+-----------------------------------+

GA settings:
Type                  =  binary
Population size       =  50
Number of generations =  100
Elitism               =  2
Crossover probability =  0.8
Mutation probability  =  0.1

GA results:
Iterations             = 100
Fitness function value = 77
Solution =
     x1 x2 x3 x4 x5 x6
[1,]  1  1  0  0  1  0
```

```
sol <- gaOpt@solution
sol
```

```
     x1 x2 x3 x4 x5 x6
[1,]  1  1  0  0  1  0
```

The following shows how the Genetic algorithm stochastically improves the objective function value over the iterations.

```
plot(gaOpt)
```



The following table shows the utilities that the 10 subjects derive from the status quo product and the products within the product line.

```
t(apply(utils, 1, function(x)(x * c(1,sol[1,]))))
```

```
      Status.Quo Product1 Product2 Product3 Product4 Product5 Product6
 [1,]         55       47       56        0        0       39        0
 [2,]         62       55       72        0        0       71        0
 [3,]         71       63       70        0        0       79        0
 [4,]         47       55       43        0        0       50        0
 [5,]         90       95       48        0        0       71        0
 [6,]         70       62       81        0        0       61        0
 [7,]         63       58       71        0        0       42        0
 [8,]         59       47       62        0        0       79        0
 [9,]         81       83       80        0        0       71        0
[10,]         77       66       78        0        0       32        0
```

We can compute the optimal profit using the profit function that we specified above. Note that the profit function is distinct from the objective function, as the latter also contains the penalty term.

```
profit(sol[1,], utils, margin)
```

```
[1] 77
```

References

Belloni A, Robert Freund, Matthew Selove, and Duncan Simester (2008), “Optimizing Product Line Designs: Efficient Methods and Comparisons,” Management Science, 54(9), 1544-1552.

Green P. E., Krieger A. M. Models and heuristics for product line selection. Marketing Science. (1985) 4(1):1–19.