HillClimbing

Fan Zhang'17

2015-02-05

HillClimbing is a R package created to reproduce results in the paper "Local Hillclimbing on an Economic Landscape" by David Kane in 1996. The package has a class named 'HillClimbing"" and six different methods: generateObject( ), profitFunction( ), findNeighborhood( ), neighborhoodProfit( ), climbingStrategy( ) and paperResult( ).

Paper background

The paper introduces an interesting idea of economic hillclimbing for different firms. Considering a firm faced with single-period profit maximization problem, the paper proposes a model with following basic components:

1. The set of inputs:

$x = x_1 x_2 \cdots x_n \mathrm{w} ith x_i \in \{0, 1, 2, \cdots\}$

2. Budget constraint:

$\sum_{i=1}^{n} x_i = B \in \{0, 1, 2, \cdots\}$

3. Profit function:

$\pi(X) = c_1 x_1 + c_2 x_2 + \cdots + c_n x_n + c_{n+1} x_1^2 + c_{n+2} x_2^2 + \cdots + c_{2n} x_n^2 + c_{2n+1} x_1 x_2 + c_{2n+2} x_1 x_3 + \cdots + c_{n\frac{n+3}{2}} x_{n-1} x_n$

The paper pictures profit functions as economic landscapes and introduces the idea of a neighborhood.The neighborhood of a given allocation X is the set of points such that all the individual allocations are identical except for two; and for those two, one input receives one more dollar and the other input receives one less.

After calculating all possible outcomes of neighborhood allocations, the company has three choices regarding where to move the company: to choose the maximum in the neighborhood set and move the company to the corresponding budget allocation (SA),to choose the median in the neighborhood set higher than original profit (MA), or to choose the minimum in the neighborhood set higher than original level (LA).

By computing mean normalized profits for different firm strategies and different connections, the paper points out the superiority of the Least Ascent (LA) strategy. The paper suggests that it might be advisable for companies to be patient and choose the LA strategy to achieve their ultimate profit maximization.

Package

Definition of slots in HillClimbing class

1. approach: three simple strategies for hillclimbing firms, users should put "SA", "MA" or "LA" in this slot
2. budget: the amount of firm's total budget
3. input: the number of total inputs
4. connection: the number of connections per input

5. allocation: a vector which contains a set of inputs subject to the budget constraint, a randomized allocation for the firm
6. linearConstraint: a vector which gives upper and lower constraints for linear coefficients
7. squareConstraint: a vector which gives upper and lower constraints for square coefficients
8. crossproductConstraint: a vector which gives constraints for cross product coefficients
9. coef1: a vector which stores all linear coefficients of the profit function
10. coef2: a matrix which stores all quadratic coefficients (both square and cross product coefficients) of the profit function

Package usage

We first install and load the package.

```
library(HillClimbing)
```

To reproduce results in the paper, we need to give the package an object in the following format. We can change the value of different slots in the object.

```
  object <- new("HillClimbing", approach="SA",
                    budget=50, input=20, connection=2,
                    linearConstraint=c(-1,1),
                    squareConstraint=c(-1,1),
                    crossproductConstraint=c(-1,1)
                    )
```

We start by using the method generateObject( ) and we can get an object with all slots filled.

```
object <- generateObject(object)
```

Then we calculate the profit of the starting allocation with method profitFunction( ).

```
profitFunction(object)
```

We can find the neighborhood (all neighbors) of the starting allocation with method findNeighborhood( ).

```
findNeighborhood(object)
```

We then get all neighborhood profits with method neighborhoodProfit( ).

```
neighborhoodProfit(object)
```

We use the method climbingStrategy( ) to find the profit maximum in the local landscape.

```
climbingStrategy(object)
```

Finally, to find our mean normalized profits over many landscapes, we run the method paperResult( ). We may change the number of times we use the climbing strategy (the number of landscapes) by changing t in the paperResult() function.

```
paperResult(object)
```

However, if we are not interested in these details and want a quick result, we can also run method paperResult( ) directly after giving the object.

```
  object <- new("HillClimbing", approach="SA",
                    budget=50, input=20, connection=2,
                    linearConstraint=c(-1,1),
                    squareConstraint=c(-1,1),
                    crossproductConstraint=c(-1,1)
                    )
  paperResult(object)
```

Methods

This R package uses six different methods to reproduce results in the paper. Each method is explained separately below.

1. generateObject( )

This method is used to generate all slots of the object. It gives the object a random starting allocation and randomizes coefficients of the profit function.

We generate a starting allocation subject to the budget constraint. We randomize the allocation of each dollar of the budget.

```
n <- object@input
object@allocation <- rep(0,n)
    for (i in 1:object@budget){
      random <- sample(1:n,1,replace=T)
      object@allocation[random] <- object@allocation[random]+1
    }
```

We put the randomized allocation into the slot "allocation". Then we randomize coefficients of the profit function and put these coefficients into slots "coef1" and "coef2".

```
    coef1 <- object@coef1
    coef2 <- object@coef2
    coef2 <- matrix(0,nrow=n,ncol=n)
    # randomize linear coefficients
    coef1 <- runif(n,object@linearConstraint[1], object@linearConstraint[2])
    for(i in 1:n){
      for (j in i:n){
        # randomize square coefficients
        if(i==j){
          coef2[i,j] <- runif(1,object@squareConstraint[1],object@squareConstraint[2])
        }
        # randomize cross product coefficients
        else {
          coef2[i,j] <- runif(1,object@crossproductConstraint[1],
                              object@crossproductConstraint[2])/2
```

```
        coef2[j,i] <- runif(1,object@crossproductConstraint[1],
                            object@crossproductConstraint[2])/2
        }
    }
  }
  return(object)
```

2. profitFunction( )

This method is used to calculate the normalized profit level for the given allocation. We calculate the profit by adding linear terms and quadratic terms(square and cross product terms) of the profit function. We then normalize the profit value and return it as a vector.

```
  allocation <- object@allocation
  coef1 <- object@coef1
  coef2 <- object@coef2
  profit <- coef1%*%allocation+allocation%*%(coef2%*%allocation)
  profit.norm <- as.vector((100*profit/(object@budget+object@budget^2)))
  return(profit.norm)
```

3. findNeighborhood( )

This method is used to find all neighbors of the given allocation in the economic landscape. We put all neighbors into columns of a matrix and return the neighborhood matrix.

We first define a n by n adjacency matrix to represent connection status between inputs (1 represents connected and 0 represents unconnected). In this way, if the number of connections is c, we can randomly connect an input wth c other inputs.

```
  n <- object@input
  connections <- matrix(0,nrow=n,ncol=n)
  for (i in 1:n){
    c <- 0
    while (c < object@connection){
      j <- sample(1:n,1,replace=T)
      while((i==j)|(connections[i,j]==1)){
        j <- sample(1:n,1,replace=T)
      }
      connections[i,j]=1
      c <- c+1
    }
  }
```

We start with the initial randomized allocation. If there exists a connection between Xi and Xj, we then transfer one dollar from input Xi to input Xj. Since we don't know the number of neighbors, we define 1000 columns at first and delete columns of zeros later. We return a neighborhood matrix with each neighbor in a separate column.

```
  neighbors <- matrix(0,nrow = n, ncol = 1000)
  k <- 1
  for (i in 1:n){
    for (j in 1:n){
```

```
      newNeighbor <- object@allocation
      if (connections[i,j] == 1 && newNeighbor[i]>0){
        newNeighbor[i] <- newNeighbor[i]-1
        newNeighbor[j] <- newNeighbor[j]+1
        neighbors[, k] <- newNeighbor
        k <- k + 1
      }
    }
  }
  # find out columns of zeros
  for(m in 1:1000){
    if(sum(neighbors[, m]) == 0) {
      break
    }
  }
  neighbors <- neighbors[,1:(m-1)]
  return(neighbors)
```

4. neighborhoodProfit( )

This method is used to calculate normalized profits for all neighbors found in the landscape. We use the same method to calculate the profit of each neighbor and store all normalized neighborhood profits in a vector.

```
  neighbors <- findNeighborhood(object)
  neighbors.iniprofit <- rep(0,ncol(neighbors))
  for (k in 1:ncol(neighbors)){
    coef1 <- object@coef1
    coef2 <- object@coef2
    neighbors.iniprofit[k] <- (coef1)%*%neighbors[,k]+neighbors[,k]%*%(coef2%*%neighbors[,k])
    neighbors.profit <- 100*neighbors.iniprofit/(object@budget + object@budget^2)
    }
  return(neighbors.profit)
```

5. climbingStrategy( )

This method is used to calculate the normalized profit maximum on a local economic landscape. As the paper discusses, there are three strategies for hillclimbing firms: Steepest Ascent(SA), Median Ascent(MA) and Least Ascent(LA). This method returns the maximum profit of the local landscape.

```
  neighbors <- findNeighborhood(object)
  profit.norm <- profitFunction(object)
  select <- NULL
  current.profit <- profit.norm
  neighbors.profit <- neighborhoodProfit(object)
  landscape.max <- 0
  # loop ends when there is no neighborhood profit higher than the current profit level
  while(length(neighbors.profit[which(neighbors.profit>profit.norm)])!=0) {
    # select the subset of neighborhood profits higher than the current profit level
    select <- neighbors.profit[which(neighbors.profit>profit.norm)]
    # the Steepest Ascent strategy
    if(object@approach =="SA"){
```

```
      k <- which(neighbors.profit==max(select))
      current.profit <- neighbors.profit[k]
    }
    # the Median Ascent strategy
    else if(object@approach == "MA"){
      medianIndex <- which.min(abs(select-median(select)))
      k <- which(neighbors.profit==select[medianIndex])
      current.profit <- neighbors.profit[k]
    }
    # the Least Ascent strategy
    else if(object@approach == "LA"){
      k <- which(neighbors.profit == min(select))
      current.profit <- neighbors.profit[k]
    }
    # move from the starting allocation to its neighbor given different approaches
    object@allocation <- neighbors[,k]
  }
  landscape.max <- current.profit
  return(landscape.max)
```

6. paperResult( )

This method is used to run the climbingStrategy( ) method for t times and find the mean normalized profit over t different landscapes. We may change the number of t as we hillclimb over different number of landscapes.

```
  # t is the number of times we want to use the according strategy
  t <- 1000
  Result <- NULL
  for (i in 1:t){
    Result[i] <- climbingStrategy(object)
    result <- mean(Result)
  }
  return(result)
```

Results and Conclusion

1. Table 1

Mean normalized profits over 1000 landscapes for different firm strategies and connections per input.

For this table, budget=50, input=20, linearConstraint=c(-1,1), squareConstraint=c(-1,1), crossproductConstraint=c(-1,1).

Connections per input	1	2	3	4	5
Steepest Ascent	14.97216	25.36077	30.16342	38.22828	43.48658
Median Ascent	15.8626	24.77427	31.27791	36.38747	44.07121
Least Ascent	23.79984	29.4174	35.43877	39.96177	46.17567

2. Table 2

Mean normalized profits over 1, 000 landscapes for different firm strategies and connections per input.

For this table, budget=50, input=20, linearConstraint=c (0,1), squareConstraint=c (0,1), crossproductConstraint=c (-1,0).

Connections per input	1	2	3	4	5
Steepest Ascent	-2.88448	7.247748	11.08525	16.66479	20.28711
Median Ascent	-2.89668	6.092761	14.3331	15.76148	22.6064
Least Ascent	-2.81460	7.204734	15.23751	16.53196	19.41239

3. Table 3

Mean normalized profits over 1, 000 landscapes for different firm strategies and connections per input.

For this table, budget=50, input=20, linearConstraint=c (0,0), squareConstraint=c (-19,0), crossproductConstraint=c (0,2). | Connections per input | 1 | 2 | 3 | 4 | 5 | |———-|:———-:|:——:|:——:|:——:|:——:|:———:|:——:| | Steepest Ascent |-8.423737 | 0.266066 | 10.24508 | 12.9103 | 14.98578 | | Median Ascent | | | Least Ascent ||

4. Conclusion

The results produced from the R package successfully test and support the idea in the paper. We notice that in almost all cases, Least Ascent does at least as well as Median Ascent and Steepest Ascent. In some cases, LA reaches greater profit levels than MA and SA. This result is consistent with the paper.

References

David Kane, “Local Hillclimbing on an Economic Landscape,” 1996