DEA Assignment

Gaurav Kudeshia

2023-11-05

Assignment Overview:

Utilized DEA to assess energy consumption and performance metrics in natural DEA and constant returns to scale (CRS) scenarios across diverse data-center demands. Analyzed representative sets of high and low demands for both small and large data-centers. Leveraged DEA outcomes to recommend optimal energy policies, equipping data-center managers with insights to identify inefficiencies and enact necessary improvements. This approach empowers decision-makers to enhance energy efficiency, optimize resource allocation, and elevate overall data-center performance.

CONCLUSION

• The ‘matrix’ variable table presents three-dimensional reference points for each energy policy, based on input factors (such as “D.C size” and “Number of Shutdowns”) and output metrics (“Computing Time(h)”, “MWh Consumed”, and “Queue time(ms)”). These reference points are represented along the x, y, and z axes, outlining the frontiers or boundary conditions for the policies.

• Upon conducting DEA analysis, the efficiencies of the 18 energy policies were evaluated, resulting in efficiency scores of [1.0000 1.0000 0.9991 0.4818 1.0000 0.4872 1.0000 0.9826 0.9578 1.0000 0.9806 0.4754 1.0000 0.9944 1.0000 0.9970 0.5290 0.4783]. Among these, seven policies demonstrated 100% efficiency, establishing them as the frontiers, while the remaining 11 policies require enhancement in their performance.

• To quantify the improvements needed, the least efficient policy, “Gamma Omega High,” requires a mere 0.09% enhancement, whereas the most inefficient, “Margin Omega Low,” necessitates a substantial 52.46% improvement. These values were determined using the lambda function under Constant Return to Scale (CRS) conditions, encapsulating a broad analytical scope.

• This analysis implies that, for instance, “Gamma Omega High” could achieve efficiency by adopting strategies from top-performing policies like “Margin Mesos High,” “Margin Omega High,” and “Margin Mono. High.” By learning from these best practices, energy policies can bridge the efficiency gap, ultimately enhancing their accuracy and operational effectiveness. This approach enables organizations to make data-driven decisions, optimize their energy policies, and align their practices with the most successful models in the field.

SUMMARY

• Start by installing the “Benchmarking” and “IpSolveAPI” libraries, enabling the usage of functions like “dea” for Constant Returns to Scale calculations.

• Create two matrices, ‘x’ and ‘y’, incorporating data on data center size, shutdowns, computing time, MWh consumption, and queue time.

• Combine ‘x’ and ‘y’ using the ‘cbind’ function, merging information about data center operations and energy usage.

• Assign meaningful row names to the ‘matrix’ variable for precise data point identification, enhancing clarity in the analysis.

• Printed the ‘matrix’ variable to visualize the comprehensive dataset, encompassing essential metrics such as D.C size, Number of Shutdowns, Computing Time(h), and MWh Consumed.

• Apply Data Envelopment Analysis (DEA) on ‘x’ and ‘y’ utilizing the ‘dea’ function, specifying the “RTS” parameter as “crs” for Constant Returns to Scale assessment.

• Display the efficiency scores derived from the DEA analysis, providing insights into the effectiveness of different energy policies.

• Utilize the ‘peers’ function to identify peers for each data point, aiding in understanding relative efficiency scores and benchmarking against peers.

• Extract and store efficiency scores for each data point in the ‘C_Weightage_find’ variable using the ‘lambda’ function, enabling further analysis and comparison of efficiency levels within the dataset.

Loaded datasets from the necessary libraries

```
library(Benchmarking)
```

```
## Loading required package: lpSolveAPI
```

```
## Loading required package: ucminf
```

```
## Loading required package: quadprog
```

```
library(lpSolveAPI)
```

Generated matrices with identical row counts.

```
x <- matrix(c(1000, 1000, 1000, 1000, 1000, 1000, 5000, 5000, 5000, 5000, 5000, 50
00, 10000, 10000, 10000, 10000, 10000, 10000,37166,13361,14252,36404,19671,32407,6
981,9877,33589,8578,11863,15452,9680,11388,18150,18409,29707,40772), ncol = 2)
y <- matrix(c(104.42, 104.26, 104.17, 49.25, 49.63, 49.34, 99.96, 99.96, 100.03, 1
00.26, 100.26, 46.7, 101.56, 101.56, 101.63, 101.63, 45.83, 46.09, 49.01, 49.65, 4
9.6, 23.92, 24.65, 24.19, 237.09, 235.92, 234.9, 239.13, 236.95, 115.82, 481.36, 4
79.36, 486.11, 484.69, 228.31, 233.5, 90.1, 1093, 0.1, 78.3, 1188.7, 1.1, 126.2, 1
29.8, 1122.6, 0.7, 1, 0.5, 325.2, 327.9, 2.6, 2.5, 1107.6, 3.8), ncol = 3)
```

Assigned specific attributes to columns as needed and printed columns stored in ‘x’ and ‘y’

```
colnames(y) <- c("Computing Time(h)", "MWh Consumed", "Queue time(ms)")
colnames(x) <- c("D.C size", "Number of Shutdowns")

print(x)
```

```
##       D.C size Number of Shutdowns
##  [1,]     1000               37166
##  [2,]     1000               13361
##  [3,]     1000               14252
##  [4,]     1000               36404
##  [5,]     1000               19671
##  [6,]     1000               32407
##  [7,]     5000                6981
##  [8,]     5000                9877
##  [9,]     5000               33589
## [10,]     5000                8578
## [11,]     5000               11863
## [12,]     5000               15452
## [13,]    10000                9680
## [14,]    10000               11388
## [15,]    10000               18150
## [16,]    10000               18409
## [17,]    10000               29707
## [18,]    10000               40772
```

```
print(y)
```

```
##       Computing Time(h) MWh Consumed Queue time(ms)
##  [1,]            104.42        49.01           90.1
##  [2,]            104.26        49.65         1093.0
##  [3,]            104.17        49.60            0.1
##  [4,]             49.25        23.92           78.3
##  [5,]             49.63        24.65         1188.7
##  [6,]             49.34        24.19            1.1
##  [7,]             99.96       237.09          126.2
##  [8,]             99.96       235.92          129.8
##  [9,]            100.03       234.90         1122.6
## [10,]            100.26       239.13            0.7
## [11,]            100.26       236.95            1.0
## [12,]             46.70       115.82            0.5
## [13,]            101.56       481.36          325.2
## [14,]            101.56       479.36          327.9
## [15,]            101.63       486.11            2.6
## [16,]            101.63       484.69            2.5
## [17,]             45.83       228.31         1107.6
## [18,]             46.09       233.50            3.8
```

Ensured matching row counts for matrices ‘x’ and ‘y,’ followed by employing the cbind function and then counted the dimentions of ‘matrix’ table and printed ‘matrix’

```
matrix <- cbind(x, y)
row.names(matrix) <- c("Always Monolithic High", "Margin Mesos High", "Gamma Omega
High", "Always Mono. Low", "ExponentialMesos Low", "Load Omega Low", "Margin Mono.
High", "Gamma Mono. High", "Random Mesos High", "Margin Omega High", "ExponentialO
mega High", "Margin Omega Low", "Margin Mono. High", "Gamma Mono. High", "Margin O
mega High", "Gamma Omega High", "Gamma Mesos Low", "Random Omega Low")

dim(matrix)
```

```
## [1] 18  5
```

```
print(matrix)
```

```
##                         D.C size Number of Shutdowns Computing Time(h)
## Always Monolithic High     1000               37166            104.42
## Margin Mesos High          1000               13361            104.26
## Gamma Omega High           1000               14252            104.17
## Always Mono. Low           1000               36404             49.25
## ExponentialMesos Low       1000               19671             49.63
## Load Omega Low             1000               32407             49.34
## Margin Mono. High          5000                6981             99.96
## Gamma Mono. High           5000                9877             99.96
## Random Mesos High          5000               33589            100.03
## Margin Omega High          5000                8578            100.26
## ExponentialOmega High      5000               11863            100.26
## Margin Omega Low           5000               15452             46.70
## Margin Mono. High         10000                9680            101.56
## Gamma Mono. High          10000               11388            101.56
## Margin Omega High         10000               18150            101.63
## Gamma Omega High          10000               18409            101.63
## Gamma Mesos Low           10000               29707             45.83
## Random Omega Low          10000               40772             46.09
##                         MWh Consumed Queue time(ms)
## Always Monolithic High         49.01           90.1
## Margin Mesos High              49.65         1093.0
## Gamma Omega High               49.60            0.1
## Always Mono. Low               23.92           78.3
## ExponentialMesos Low           24.65         1188.7
## Load Omega Low                 24.19            1.1
## Margin Mono. High             237.09          126.2
## Gamma Mono. High              235.92          129.8
## Random Mesos High             234.90         1122.6
## Margin Omega High             239.13            0.7
## ExponentialOmega High         236.95            1.0
## Margin Omega Low              115.82            0.5
## Margin Mono. High             481.36          325.2
## Gamma Mono. High              479.36          327.9
## Margin Omega High             486.11            2.6
## Gamma Omega High              484.69            2.5
## Gamma Mesos Low               228.31         1107.6
## Random Omega Low              233.50            3.8
```

Applied convexity, free disposability, and constant returns to scale principles, specifying input and output parameters. Utilized the dea function to calculate efficiency.

```
C <- dea(x,y, RTS = "crs")
print(C)
```

```
##  [1] 1.0000 1.0000 0.9991 0.4818 1.0000 0.4872 1.0000 0.9826 0.9578 1.0000
## [11] 0.9806 0.4754 1.0000 0.9944 1.0000 0.9970 0.5290 0.4783
```

Identified peer idols using the “peers” function.

```
P <- peers(C)
print(P)
```

```
##       peer1 peer2 peer3
##  [1,]     1    NA    NA
##  [2,]     2    NA    NA
##  [3,]     1     2    NA
##  [4,]     2    NA    NA
##  [5,]     5    NA    NA
##  [6,]     2    NA    NA
##  [7,]     7    NA    NA
##  [8,]     2    10    13
##  [9,]     2    15    NA
## [10,]    10    NA    NA
## [11,]     2    13    15
## [12,]     2    15    NA
## [13,]    13    NA    NA
## [14,]     2    13    15
## [15,]    15    NA    NA
## [16,]     2    15    NA
## [17,]     2    13    NA
## [18,]     2    15    NA
```

Lambda is employed for understanding the weightage of specific elements in the analysis.

```
C_Weightage_find <- lambda(C)
C_Weightage_find
```

```
##                L1         L2 L5 L7       L10       L13        L15
##  [1,] 1.000000000 0.00000000  0  0 0.0000000 0.0000000 0.00000000
##  [2,] 0.000000000 1.00000000  0  0 0.0000000 0.0000000 0.00000000
##  [3,] 0.009970484 0.98915099  0  0 0.0000000 0.0000000 0.00000000
##  [4,] 0.000000000 0.48177241  0  0 0.0000000 0.0000000 0.00000000
##  [5,] 0.000000000 0.00000000  1  0 0.0000000 0.0000000 0.00000000
##  [6,] 0.000000000 0.48721047  0  0 0.0000000 0.0000000 0.00000000
##  [7,] 0.000000000 0.00000000  0  1 0.0000000 0.0000000 0.00000000
##  [8,] 0.000000000 0.22098286  0  0 0.5914729 0.1734861 0.00000000
##  [9,] 0.000000000 2.03346741  0  0 0.0000000 0.0000000 0.27553094
## [10,] 0.000000000 0.00000000  0  0 1.0000000 0.0000000 0.00000000
## [11,] 0.000000000 0.53626578  0  0 0.0000000 0.4082527 0.02840485
## [12,] 0.000000000 0.26256674  0  0 0.0000000 0.0000000 0.21144095
## [13,] 0.000000000 0.00000000  0  0 0.0000000 1.0000000 0.00000000
## [14,] 0.000000000 0.04516562  0  0 0.0000000 0.8554257 0.13443418
## [15,] 0.000000000 0.00000000  0  0 0.0000000 0.0000000 1.00000000
## [16,] 0.000000000 0.02236541  0  0 0.0000000 0.0000000 0.99479451
## [17,] 0.000000000 0.89985422  0  0 0.0000000 0.3814863 0.00000000
## [18,] 0.000000000 0.93720988  0  0 0.0000000 0.0000000 0.38461980
```