Using The Lobico Function

Bo Li
Wail Ba-Alawi
Christopher Eeles
Benjamin Haibe-Kains

09/07/2019

Introduction

R-LOBICO is an R package that aims to provide implementations of several logical modelling algorithms. In the field of cancer biomarker discovery, where models suffer from high dimensionality of the molecular data and the low number of samples, single predictor models are in general not accurate enough, reflecting the importance of acknowledging the interaction between biological features. On the other hand, some machine learning approaches produce complex multi-predictor models that are more accurate but hard to interpret and not amenable to the generation of hypotheses. Logical models satisfy the urgent need for approaches that build small, interpretable, yet accurate models which capture the interplay between biological features. LOBICO, namely Logic Optimization for Binary Input to Continuous Output, is one computational approach that was applied on a large pharmacogenomic dataset to find robust candidate biomarkers. The package applies LOBICO model to large and complex datasets, formulates the logic mapping as an integer linear programming problem (ILP), and uses the advanced ILP solvers (IBM ILOG CPLEX) to find the optimal mapping.

Installing the Package

Please note that installation of this file depends on the package Rcpp to compile the C code. Additionally, to use the package and its function you must have a working installation of IBM ILOG CPLEX 12.9.

Data Loading

Load the real data example. In this example we load the responses of applying bibw2992 drug in pharmacogenomic datasets.

```
load("../data/bibw2992.RData")
MutationMatrix <- bibw2992
Samples <- MutationMatrix$Cell.lines
IC50s <- MutationMatrix$BIBW2992
MutationMatrix <- MutationMatrix[, -2:-1]
Features <- colnames(MutationMatrix)
rownames(MutationMatrix) <- Samples
```

Configuring Parameters

Configure the parameters to formulate the logical model

```
## Create binary input, output, and weight vector
#binary input
X <- MutationMatrix

N <- nrow(X)
P <- ncol(X)

#binarization threshold th
```

```
th <- 0.063218
Y <- as.double(IC50s < th)
W <- abs(IC50s - th)

#class weights
FPW <- 1
FPN <- 1

#normalize weights
W[Y == 1] <- FPW * W[Y == 1] / sum(W[Y == 1])
W[Y != 1] <- -(FPN * W[Y != 1] / sum(W[Y != 1]))

## Logic model complexity
K <- 2
M <- 1
```

CPLEX Options

Notes about configuring the options for CPLEX and the use cases for each configuration. The `param` and `solve` parameters are excluded from the lobico function in this release, but support for these features will be added in future releases.

```
param <- rbind(list('tilim', 60000, 'MaxTime'), #Maximum time for IP )in seconds)
               list('mipltolerances.integrality.Cur', 1e-5, 'Integrality'), #Integrality constraint; de
               list('epgap', 1e-4, 'RelGap'), #Optimality gap tolerance; default = 1e-4 (0.01% of optim
               list('threads.Cur', 8, 'Threads'), #Number of threads to use (default = 0, automatic) (s
               list('parallel.Cur', -1, 'ParallelMode'), #Parallel optimization mode, -1 = opportunisti
               list('solnpoolgap', 1e-1, 'Pool_relgap'), #Relative gap for suboptimal solutions in the
               list('mip.pool.intensity.Cur', 1, 'Pool_intensity'), #Pool intensity; default 1 = mild:
               list('mip.pool.replace.Cur', 2, 'Pool_replace'), #Pool replacement strategy; default 2 =
               list('mip.pool.capacity.Cur', 11, 'Pool_capacity'), #Pool capacity; default 11 = best +
               list('mip.limits.populate.Cur', 11, 'Pool_size'), #Number of solutions generated; defaul
               list('preind', 0, 'Presolver'),
               list('aggind', 1, 'Aggregator'))

param <- lapply(1:ncol(param), function(x){

  if (x==2) {
    return(as.numeric(unlist(param[, x])))
  }
  return(unlist(param[, x]))

})
param <- data.frame("V1" = param[[1]], "V2" = param[[2]], "V3" = param[[3]])
```

CPLEX Solver

Build the logical model and launch the CPLEX solver.

```
## Cplex solver
sol <- lobico(X = X,
              Y = W,
              K = K,
              M = M,
```

```
              solve = 1,
              param = param)
```

```
## Construction optimizing function...
## Constructing constaints...
```

Validating Results

Return and validate the results.

```
## Check solution
print('********************')
```

```
## [1] "********************"
```

```
solMat <- .getsolution(sol, K, M, P)
print(solMat)
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13]
## [1,]    0    0    0    0    0    0    0    0    0     0     1     1     0
##      [,14] [,15] [,16] [,17] [,18] [,19] [,20] [,21] [,22] [,23] [,24]
## [1,]     0     0     0     0     0     0     0     0     0     0     0
##      [,25] [,26] [,27] [,28] [,29] [,30] [,31] [,32] [,33] [,34] [,35]
## [1,]     0     0     0     0     0     0     0     0     0     0     0
##      [,36] [,37] [,38] [,39] [,40] [,41] [,42] [,43] [,44] [,45] [,46]
## [1,]     0     0     0     0     0     0     0     0     0     0     0
##      [,47] [,48] [,49] [,50] [,51] [,52] [,53] [,54] [,55] [,56] [,57]
## [1,]     0     0     0     0     0     0     0     0     0     0     0
##      [,58] [,59] [,60]
## [1,]     0     0     0
```

```
str <- .showformula(solMat, K, M, Features)
print('Inferred logical model')
```

```
## [1] "Inferred logical model"
```

```
print(str)
```

```
## [1] "EGFR  | ERBB2 "
```