DynTxRegime

"An R Package for Dynamic Treatment Regimes"
Part 2
Multiple Decision Points

Shannon Holloway

DPHS, Duke University

March 20, 2025

Toy Dataset

https://github.com/sth1402/ST790/ Download st790Data.MD.csv

Load Data

```
df <- read.csv(file = "path/st790Data.MD.csv", header = TRUE)
```

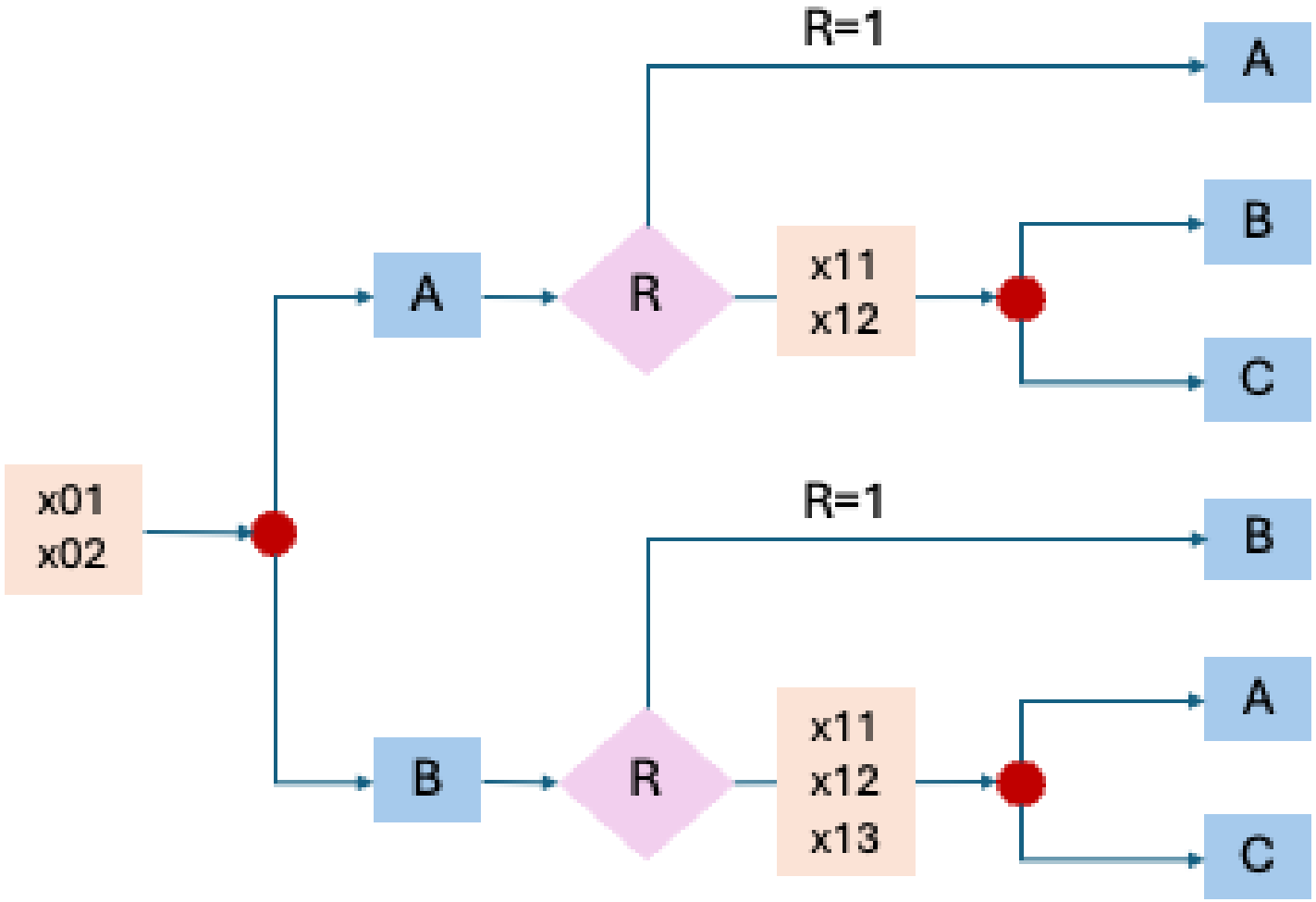
Summary Statistics

```
summary(object = df)
```

```
      x01              x02             A1
 Min.   : 6.604   Min.   :40.00   Length:1000
 1st Qu.: 9.327   1st Qu.:50.75   Class :character
 Median : 9.960   Median :61.00   Mode  :character
 Mean   : 9.973   Mean   :60.47
 3rd Qu.:10.616   3rd Qu.:70.00
 Max.   :13.196   Max.   :80.00

       r1               x11                x12
 Min.   :0.000   Min.   :-3.9880   Min.   :-14.0136
 1st Qu.:0.000   1st Qu.: 0.9574   1st Qu.: -3.9415
 Median :0.000   Median : 2.9217   Median : -0.5643
 Mean   :0.212   Mean   : 2.9964   Mean   : -0.7864
 3rd Qu.:0.000   3rd Qu.: 5.0652   3rd Qu.:  2.4046
 Max.   :1.000   Max.   :10.1618   Max.   : 10.9525
                 NA's   :212       NA's   :212
      x13              A2                  y
 Min.   :2.879   Length:1000         Min.   :-64.0147
 1st Qu.:4.417   Class :character    1st Qu.:-13.1031
 Median :5.027   Mode  :character    Median :  0.8212
 Mean   :5.053                       Mean   :  0.8835
 3rd Qu.:5.678                       3rd Qu.: 14.6074
 Max.   :8.041                       Max.   : 72.3621
 NA's   :623
```

Our data were generated under the following design:



We must incorporate the subset structure:

- Participants have different feasible treatment sets.

We can (and will) incorporate

- additional covariates available for a subset of the data.

We will work under "Scenario 1"

- the feasible treatment set and the data "match"

Feasible Treatment Sets

Feasible treatments are communicated through input `fSet`

A bit of a misnomer – input can define any subset structure of the analysis

- limit available treatments,
- use different models for regressions, and/or
- use different decision function models

There are two options for defining the input: simple vs. efficient

We'll focus only on the efficient version.

`fSet` is a user-defined function with formal argument

- `data` `function(data) { ... }` or
- individual covariate names `function(x11, x12) { ... }`

The function must return a list containing two elements:

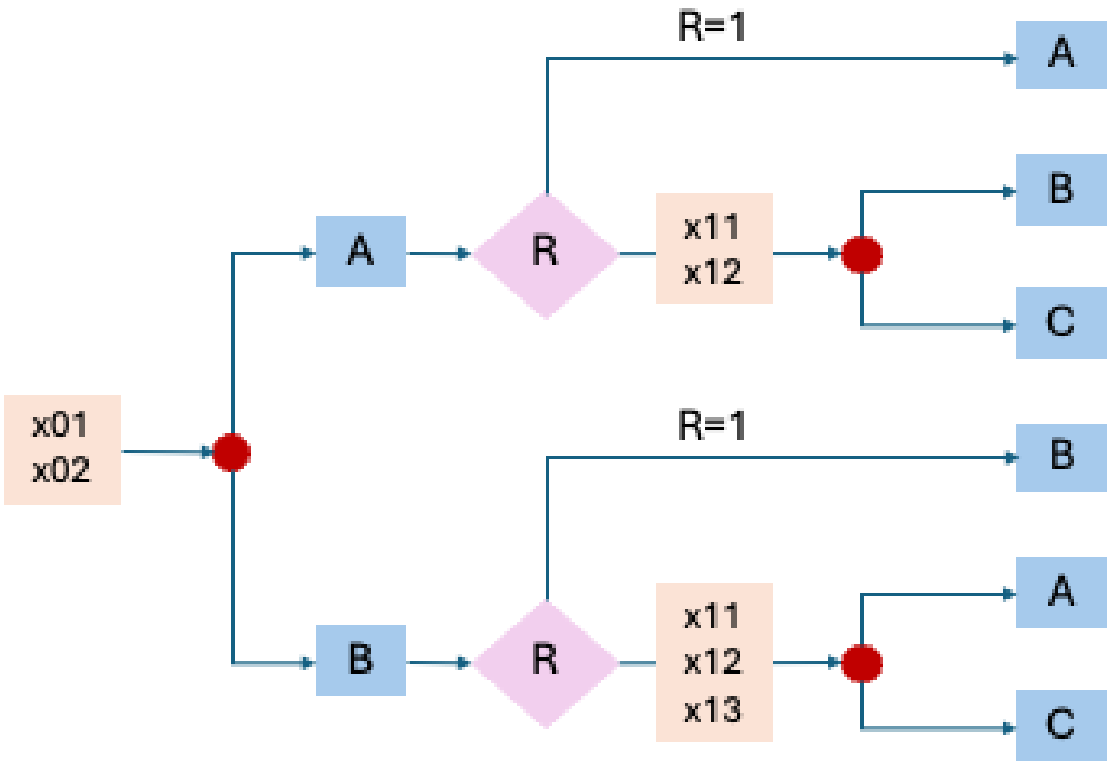
`subsets` - specification of all subsets;

`txOpts` - subset to which each participant belongs

Each function specifies the subset structure for one decision point.

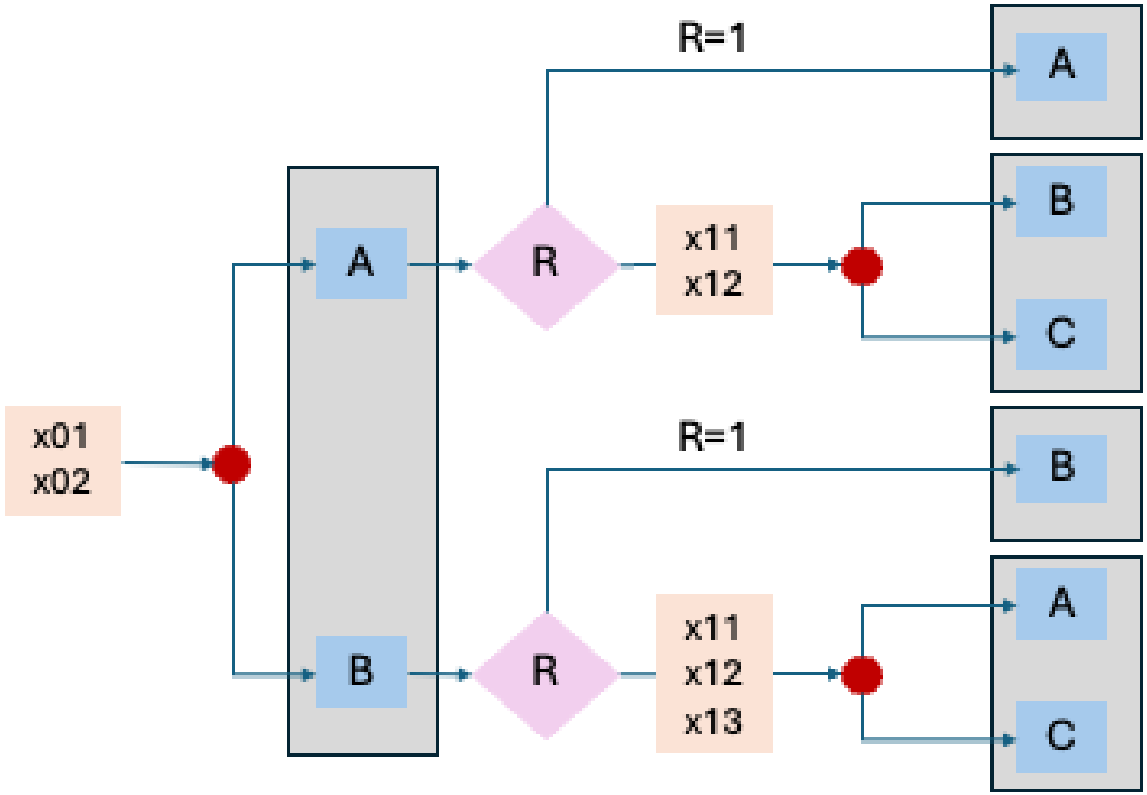
- Multiple decision points $\rightarrow$ multiple functions

Subsets



How many subsets?

Subsets



5
1 in decision point 1
4 in decision point 2

The first stage is straightforward. All participant have the same

- feasible treatment options
- measured covariates

All we need to do is

- pick a name for the subset, say "fs";
- specify the available treatment options {"A", "B"}
- assign all participants to "fs"

For example,

```
fset1 <- function(data) {
  list("subset" = list(list("fs", c("A", "B"))),
       "txOpts" = rep("fs", nrow(data)))
}
```

`subset` is a list of subsets

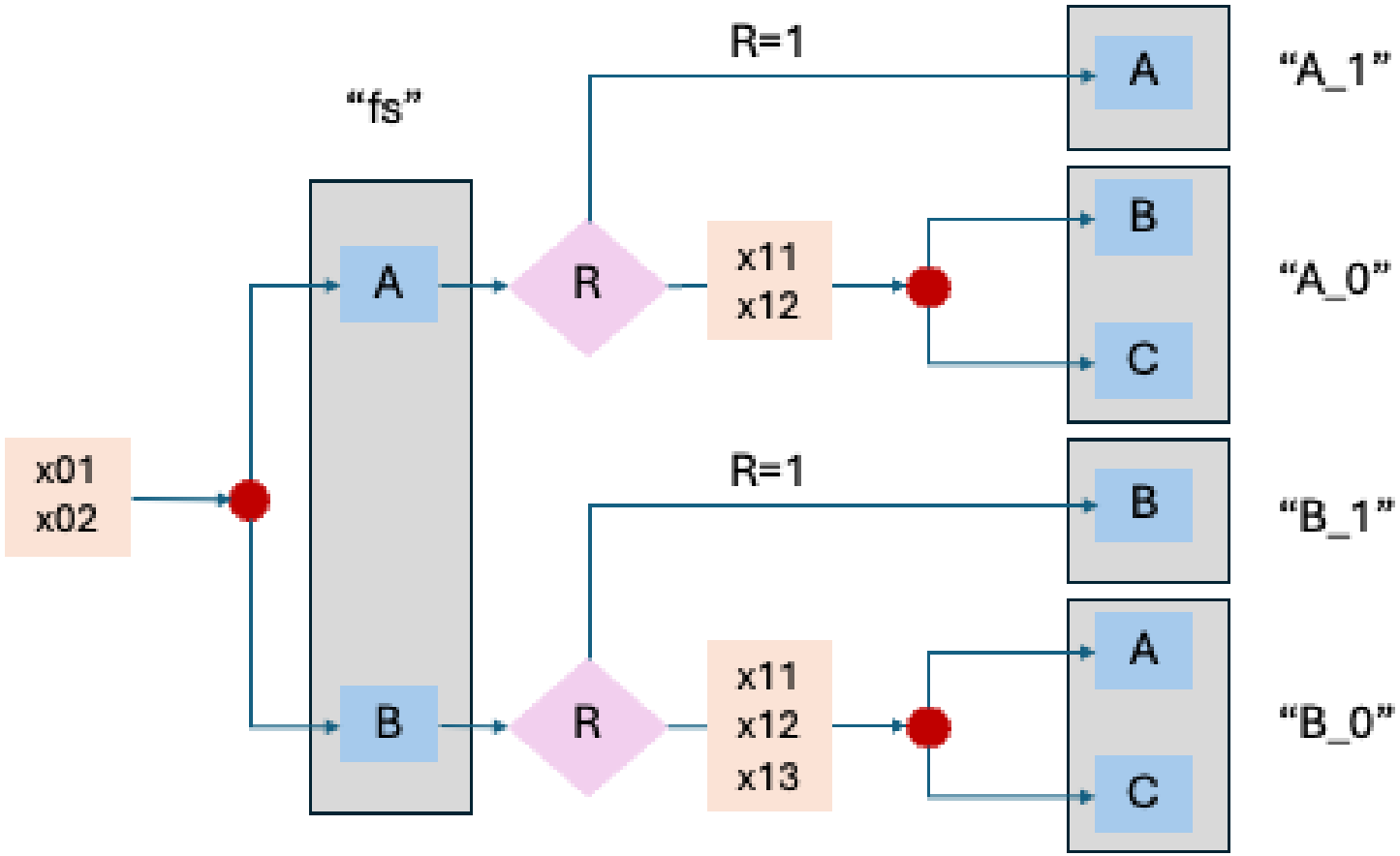
- "a subset" is a list containing a name and treatment options

`txOpts` is a character vector

The second stage is more complex

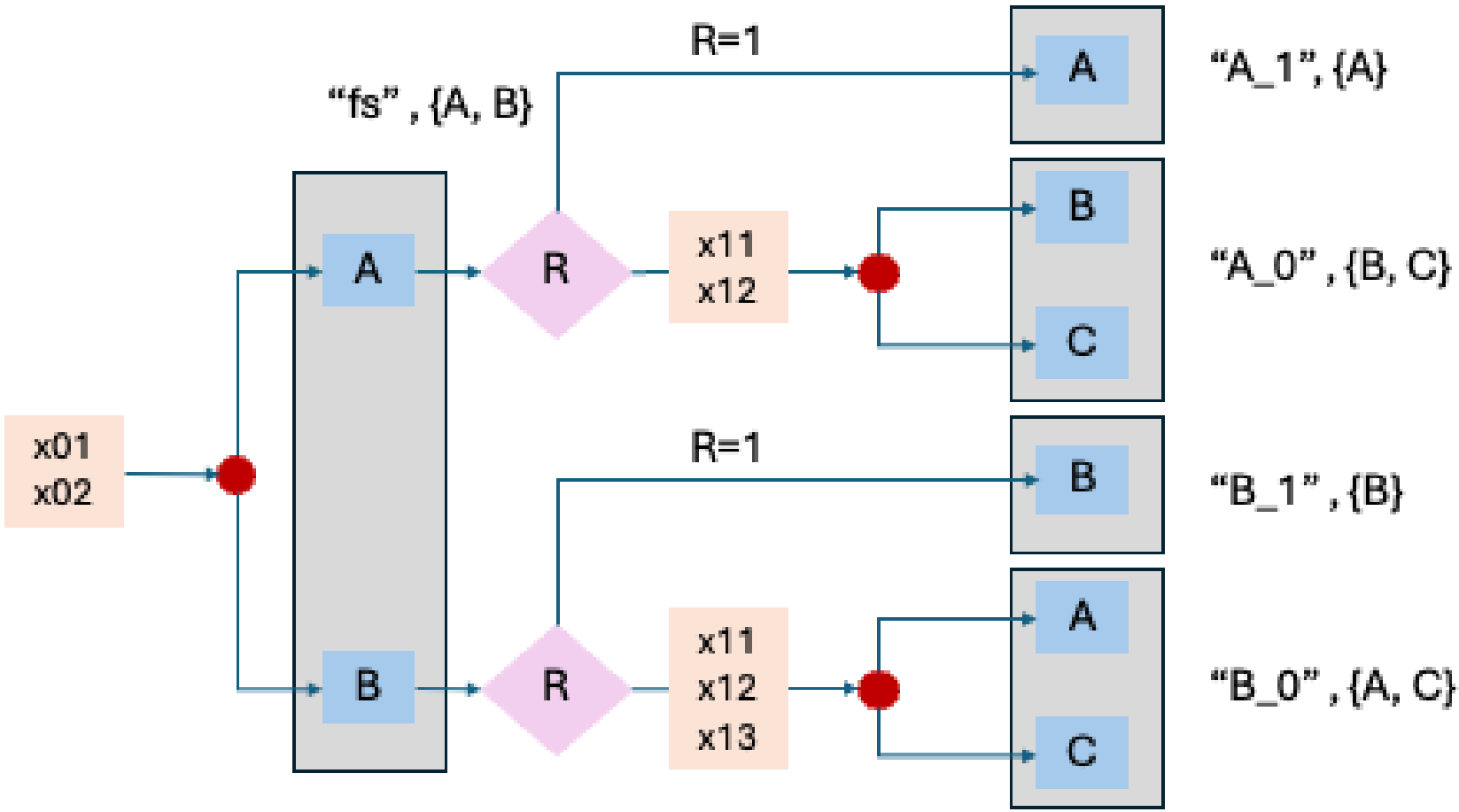
Step 1: Name each subset.

The nickname can be anything you want – just be consistent throughout the analysis.



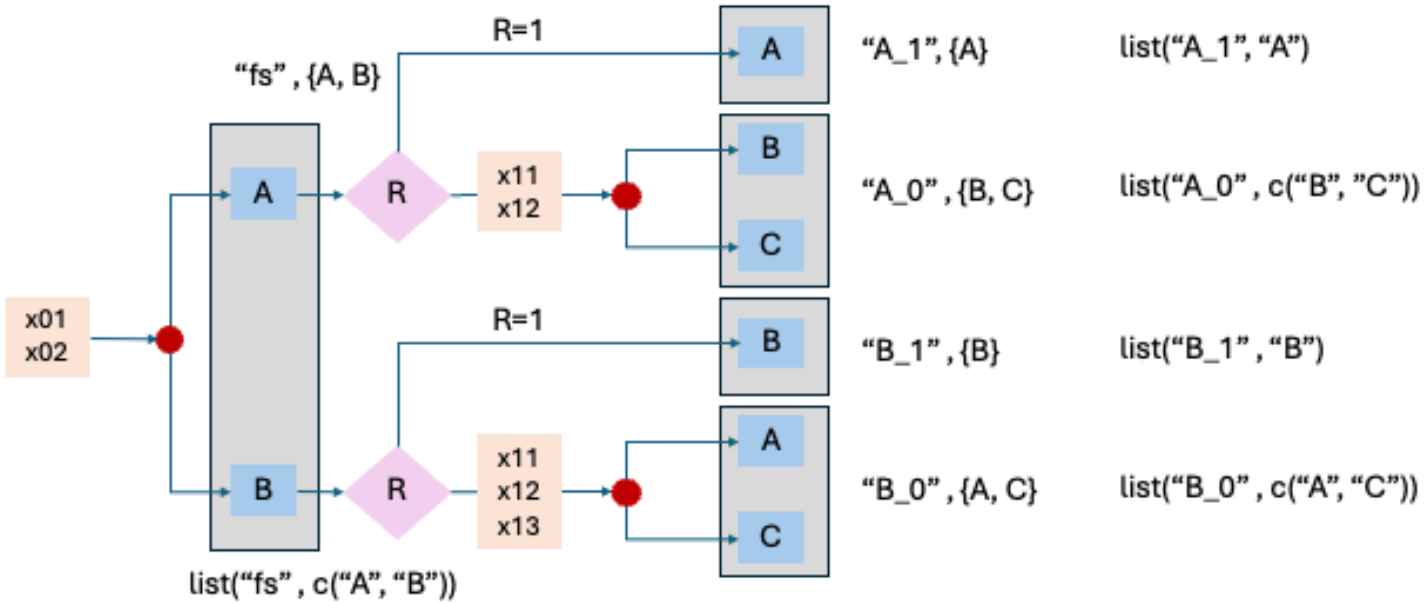
The second stage is more complex

Step 2: Specify available treatment options.



The second stage is more complex

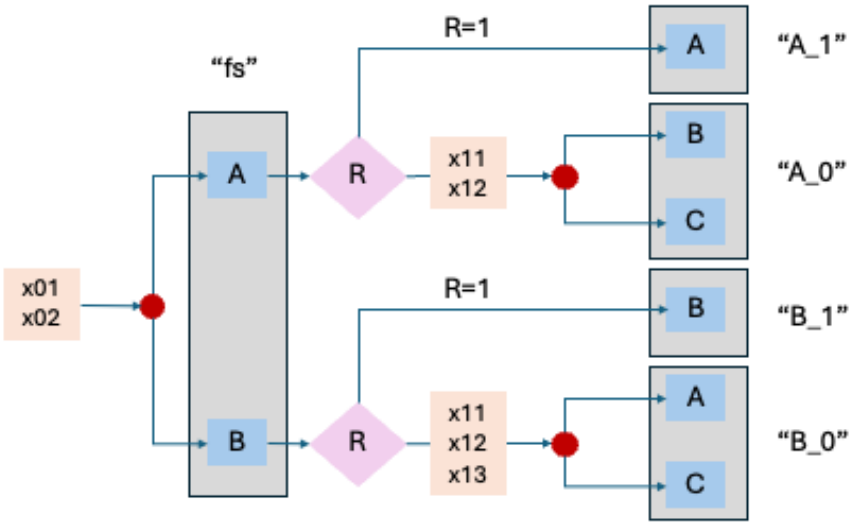
For each subset, define a list containing the name and treatments




```
subsets2 <- list(list("A_1", c("A")),
                 list("A_0", c("B", "C")),
                 list("B_1", c("B")),
                 list("B_0", c("A", "C")))
```

The second stage is more complex

`txOpts` assign participants to a subset




```
txOpts2 <- paste(A1, r1, sep = "_")
```

```
fSet2 <- function(A1, r1) {
  list("subsets" = list(list("A_1", c("A")),
                        list("A_0", c("B", "C")),
                        list("B_1", c("B")),
                        list("B_0", c("A", "C"))),
       "txOpts" = paste(A1, r1, sep = "_"))
}
```

```
fSet1 <- function(data) {
  list("subsets" = list(list("fs", c("A","B"))),
       "txOpts" = rep("fs", times = nrow(x = data)))
}

fSet2 <- function(A1, r1) {
  list("subsets" = list(list("A_1", c("A")),
                        list("A_0", c("B", "C")),
                        list("B_1", c("B")),
                        list("B_0", c("A", "C"))),
       "txOpts" = paste(A1, r1, sep = "_"))
}
```

It's always a good idea to test them first!

```
result <- fSet1(data = df)

print(result$subsets)
```

```
[[1]]
[[1]][[1]]
[1] "fs"

[[1]][[2]]
[1] "A" "B"
```

```
print(table(result$txOpts))
```

```
  fs
1000
```

It's always a good idea to test them first!

```
result <- fSet2(A1 = df$A1, r1 = df$r1)

print(result$subsets)
```

```
[[1]]
[[1]][[1]]
[1] "A_1"

[[1]][[2]]
[1] "A"


[[2]]
[[2]][[1]]
[1] "A_0"

[[2]][[2]]
[1] "B" "C"


[[3]]
[[3]][[1]]
[1] "B_1"

[[3]][[2]]
[1] "B"


[[4]]
[[4]][[1]]
[1] "B_0"

[[4]][[2]]
[1] "A" "C"
```

It's always a good idea to test them first!

```
result <- fSet2(A1 = df$A1, r1 = df$r1)

print(table(result$txOpts))
```

```
A_0 A_1 B_0 B_1
411 108 377 104
```

compared with

```
table(df$A1, df$r1)
```

```
     0   1
  A 411 108
  B 377 104
```

Q-Learning

An optimal Ψ-specific regime d^{opt} can be represented in terms of Q-functions

$$Q_K(h_K, a_K) = Q_K(\overline{x}_K, \overline{a}_K) = E(Y|\overline{X} = \overline{x}, \overline{A} = \overline{a})$$

and for $k = K-1, \ldots, 1$

$$Q_k(h_k, a_k) = Q_k(\overline{x}_k, \overline{a}_k) = E\{V_{k+1}(\overline{x}_k, X_{k+1}, \overline{a}_k)|\overline{X}_k = \overline{x}_k, \overline{A}_k = \overline{a}_k\},$$

where for $k = 1, \ldots, K$,

$$V_k(h_k) = V_k(\overline{x}_k, \overline{a}_{k-1}) = \max_{a_k \in \Psi_k(h_k)} Q_k(h_k, a_k).$$

Estimation of d^{opt} via Q-learning is accomplished by positing models for the Q-functions $Q_k(h_k, a_k)$,

$$Q_k(h_k, a_k; \beta_k) = Q_k(\overline{x}_k, \overline{a}_k; \beta_k), \quad k = K, K-1, \ldots, 1.$$

Estimators $\widehat{\beta}_k$ for β_k are obtained in a backward iterative fashion for $k = K, K-1, \ldots, 1$ by solving suitable M-estimating equations.

The estimated rules for $k = 1, \ldots, K$ are

$$\widehat{d}^{opt}_{Q,k}(h_k) = d^{opt}_k(h_k; \widehat{\beta}_k) = \underset{a_k \in \Psi_k(h_k)}{\text{argmax}} \; Q_k(h_k, a_k; \widehat{\beta}_k),$$

and the pseudo outcomes are

$$\tilde{V}_{ki} = \max_{a_k \in \Psi_k(h_k)} Q_k(h_k, a_k; \widehat{\beta}_k).$$

An estimated optimal Ψ-specific regime is then given by

$$\widehat{d}^{opt}_Q = \{\widehat{d}^{opt}_{Q,1}(h_1), \ldots, \widehat{d}^{opt}_{Q,K}(h_K)\}$$

and an estimator for the value $\mathcal{V}(d^{opt})$ is given by

$$\widehat{\mathcal{V}}_Q(d^{opt}) = n^{-1} \sum_{i=1}^{n} \tilde{V}_{1i} = n^{-1} \sum_{i=1}^{n} \max_{a_1 \in \Psi_1(H_{1i})} Q_1(H_{1i}, a_1; \widehat{\beta}_1).$$

Accomplished through repeated calls to qLearn()

```
str(object = qLearn)
```

```
function (..., moMain, moCont, data, response, txName,
    fSet = NULL, iter = 0L, verbose = TRUE)
```

Argument	Class	Description
...		Ignored. Included to require named input.
moMain	modelObj	for main effects terms (mu)
moCont	modelObj	for terms interacting with treatment (C)
data	data.frame	covariates and treatment history
response	vector	outcome of interest or value object from previous step
txName	character	treatment variable name
fSet	function	feasible set definition
iter	integer	if >0, iterative methods used
verbose	logical	if FALSE, screen prints suppressed

Accomplished through repeated calls to qLearn()

```
str(object = qLearn)
```

```
function (..., moMain, moCont, data, response, txName,
    fSet = NULL, iter = 0L, verbose = TRUE)
```

Argument	Class	Description
...		Ignored. Included to require named input.
moMain	modelObj	for main effects terms (mu)
moCont	modelObj	for terms interacting with treatment (C)
data	data.frame	covariates and treatment history
response	vector	outcome of interest or value object from previous step
txName	character	treatment variable name
fSet	function	feasible set definition
iter	integer	if >0, iterative methods used
verbose	logical	if FALSE, screen prints suppressed

Accomplished through repeated calls to qLearn()

```
str(object = qLearn)
```

```
function (..., moMain, moCont, data, response, txName,
    fSet = NULL, iter = 0L, verbose = TRUE)
```

The first STEP of the Q-learning algorithm is the analysis of STAGE 2

- Postulate a model for $Q_2(\mathrm{H}_2, \mathrm{a}_2)$. (Recall, $\sim \mu + \mathrm{A}_2\mathrm{C}$)
- Last time, we used `buildModelObj()`.
- But, under Scenario 1, we have to provide models for two subsets, each with different covariates.

A1	r1	ψ_2	data
A	1	A_1	y
A	0	A_0	x11, x12, y
B	1	B_1	y
B	0	B_0	x11, x12, x13, y

```
str(object = buildModelObjSubset)
```

```
function (..., model, solver.method, solver.args = NULL,
    predict.method = NULL, predict.args = NULL, dp = 1L,
    subset = NA)
```

Recall from the **modelObj** package

```
str(object = buildModelObj)
```

```
function (model, solver.method = NULL, solver.args = NULL,
    predict.method = NULL, predict.args = NULL)
```

- `...` - require named inputs
- dp - decision point to which the model pertains
- subset - the nickname of the subset to which the model pertains (must match `fSet`)

Assume for subset 'A_0'

$$Q_2(\mathrm{H}_2, \mathrm{A}_2, \beta_2) = \beta_{20} + \beta_{21}\mathrm{x01} + \beta_{22}\mathrm{x02} + \beta_{23}\mathrm{x11} + \beta_{24}\mathrm{x12} + \mathrm{A2}(\beta_{25} + \beta_{26}\mathrm{x11} + \beta_{27}\mathrm{x12})$$

Use least squares with a single model

```
moMain_A_0 <- buildModelObjSubset(model = ~ x01 + x02 + x11 + x12,
                                  solver.method = 'lm',
                                  subset = 'A_0', dp = 2L)

moCont_A_0 <- buildModelObjSubset(model = ~ x11 + x12,
                                  solver.method = 'lm',
                                  subset = 'A_0', dp = 2L)
```

Note that **all** individuals in subset 'A_0' have A1 = A and r1 = 0

- these cannot be included in the model.

And for subset 'B_0'

$$Q_2(\mathrm{H}_2, \mathrm{A}_2, \beta_2) = \beta_{20} + \beta_{21}\mathrm{x01} + \beta_{22}\mathrm{x11} + \beta_{23}\mathrm{x13} + \mathrm{A2}(\beta_{24} + \beta_{25}\mathrm{x11} + \beta_{26}\mathrm{x13})$$

Use least squares with a single model

```
moMain_B_0 <- buildModelObjSubset(model = ~ x01 + x11 + x13,
                                  solver.method = 'lm',
                                  subset = 'B_0', dp = 2L)

moCont_B_0 <- buildModelObjSubset(model = ~ x11 + x13,
                                  solver.method = 'lm',
                                  subset = 'B_0', dp = 2L)
```

Again – **all** individuals in subset 'B_0' have A1 = B and r1 = 0

- these cannot be included in the model.

Subsets 'A_1' and 'B_1' are not modeled – "Scenario 1".

Accomplished through repeated calls to qLearn()

```
str(object = qLearn)
```

```
function (..., moMain, moCont, data, response, txName,
    fSet = NULL, iter = 0L, verbose = TRUE)
```

The first STEP of the Q-learning algorithm is the analysis of STAGE 2

Postulate a model for $Q_2(\mathrm{H}_2, \mathrm{a}_2)$. (Recall, $\sim \mu + \mathrm{A}_2\mathrm{C}$)

```
moMain_ss <- list(moMain_A_0, moMain_B_0)

moCont_ss <- list(moCont_A_0, moCont_B_0)
```

```
str(object = qLearn)
```

```
function (..., moMain, moCont, data, response, txName,
    fSet = NULL, iter = 0L, verbose = TRUE)
```

- `data` is a complete dataset, (df)
- `response` is the "vector" outcome of interest (df$y)
- `txName` is the treatment variable name, ("A2")

Our data is not complete!

```
summary(df$x13)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
  2.879   4.417   5.027   5.053   5.678   8.041     623
```

Can set NA values to 0.

```
df[is.na(df)] <- 0.0
```

```
str(object = qLearn)
```

```
function (..., moMain, moCont, data, response, txName,
    fSet = NULL, iter = 0L, verbose = TRUE)
```

---

fSet is the previously defined function

```
fSet2 <- function(A1, r1) {

  list("subsets" = list(list("A_1", c("A")),
                        list("A_0", c("B", "C")),
                        list("B_1", c("B")),
                        list("B_0", c("A", "C"))),
       "txOpts" = paste(A1, r1, sep = "_"))

}
```

```
moMain_A_0 <- buildModelObjSubset(model = ~ x01 + x02 + x11 + x12,
                                  solver.method = 'lm',
                                  subset = 'A_0', dp = 2L)

moCont_A_0 <- buildModelObjSubset(model = ~ x11 + x12,
                                  solver.method = 'lm',
                                  subset = 'A_0', dp = 2L)

moMain_B_0 <- buildModelObjSubset(model = ~ x01 + x11 + x13,
                                  solver.method = 'lm',
                                  subset = 'B_0', dp = 2L)

moCont_B_0 <- buildModelObjSubset(model = ~ x11 + x13,
                                  solver.method = 'lm',
                                  subset = 'B_0', dp = 2L)

moMain_ss <- list(moMain_A_0, moMain_B_0)

moCont_ss <- list(moCont_A_0, moCont_B_0)

fSet2 <- function(A1, r1) {

  list("subsets" = list(list("A_1", c("A")),
                        list("A_0", c("B", "C")),
                        list("B_1", c("B")),
                        list("B_0", c("A", "C"))),
       "txOpts" = paste(A1, r1, sep = "_"))

}

qObj_ss <- qLearn(moMain = moMain_ss, moCont = moCont_ss, iter = 0L,
                  data = df, response = df$y, txName = 'A2',
                  fSet = fSet2, verbose = TRUE)
```

```
qObj_ss <- qLearn(moMain = moMain_ss, moCont = moCont_ss, iter = 0L,
                  data = df, response = df$y, txName = 'A2',
                  fSet = fSet2, verbose = TRUE)
```

```
First step of the Q-Learning Algorithm.

Subsets of treatment identified as:
$A_0
[1] "B" "C"

$A_1
[1] "A"

$B_0
[1] "A" "C"

$B_1
[1] "B"

Number of patients in data for each subset:
A_0 A_1 B_0 B_1
411 108 377 104

Outcome regression.

subset(s) A_1, B_1 excluded from outcome regression

Fitting models for  A_0 using 411 patient records.
Regression analysis for Combined:

Call:
lm(formula = YinternalY ~ x01 + x02 + x11 + x12 + A2 + x11:A2 +
    x12:A2, data = data)

Coefficients:
(Intercept)          x01          x02          x11
 -13.820922     1.709259    -0.008765    -0.187254
        x12          A2C      x11:A2C      x12:A2C
   0.260286    -2.483716     0.736931     0.039200

Fitting models for  B_0 using 377 patient records.
Regression analysis for Combined:
```

Accomplished through repeated calls to qLearn()

```
str(object = qLearn)
```

```
function (..., moMain, moCont, data, response, txName,
    fSet = NULL, iter = 0L, verbose = TRUE)
```

The second STEP of the Q-learning algorithm is the analysis of STAGE 1

Postulate a model for $Q_1(\mathrm{H}_1, \mathrm{a}_1)$. Recall, $\sim \mu + \mathrm{A}_1\mathrm{C}$

$$Q_1(\mathrm{H}_1, a_1) = \beta_{10} + \beta_{11}\mathrm{x01} + \beta_{12}\mathrm{x02} + \mathrm{A1}(\beta_{13} + \beta_{14}\mathrm{x02})$$

We do not need subset modeling

```
moMain_fs <- buildModelObj(model = ~ x01 + x02,
                           solver.method = 'lm')

moCont_fs <- buildModelObj(model = ~ x02,
                           solver.method = 'lm')
```

```
str(object = qLearn)
```

```
function (..., moMain, moCont, data, response, txName,
    fSet = NULL, iter = 0L, verbose = TRUE)
```

- `data` is a complete dataset, (df)
- `response` is the value object returned by the second stage analysis (`qObj_ss`)
- `txName` is the treatment variable name, ("A1")
- `fSet` is not required

```
moMain_fs <- buildModelObj(model = ~ x01 + x02,
                           solver.method = 'lm')

moCont_fs <- buildModelObj(model = ~ x02,
                           solver.method = 'lm')

qObj_fs <- qLearn(moMain = moMain_fs, moCont = moCont_fs, iter = 0L,
                  data = df, response = qObj_ss, txName = 'A1',
                  fSet = NULL, verbose = TRUE)
```

```
qObj_fs <- qLearn(moMain = moMain_fs, moCont = moCont_fs, iter = 0L,
                  data = df, response = qObj_ss, txName = 'A1',
                  fSet = NULL, verbose = TRUE)
```

```
Step 2 of the Q-Learning Algorithm.

Outcome regression.
Combined outcome regression model: ~ x01+x02 + A1 + A1:(x02) .
Regression analysis for Combined:

Call:
lm(formula = YinternalY ~ x01 + x02 + A1 + x02:A1, data = data)

Coefficients:
(Intercept)          x01          x02          A1B
   -2.43410      0.65179     -0.01837      3.86749
    x02:A1B
   -0.09121


Recommended Treatments:
  A   B
931  69

Estimated value: 2.964196
```

Methods available:

Method	Description
`Call(name)`	retrieve the unevaluated call
`coef(object)`	retrieve parameter estimates
`DTRstep(object)`	print description of method
`estimator(x)`	retrieve estimated value
`fitObject(object)`	retrieve value object returned for regression steps
`optTx(object)`	retrieve recommended optimal treatments for training data
`optTx(object, newdata)`	estimate optimal treatments based on a prior analysis
`outcome(object)`	retrieve value object returned for outcome regression
`plot(x, suppress)`	plot regression results
`summary(object)`	retrieve regression summaries

Methods available: Model Diagnostics

Method	Description
`Call(name)`	retrieve the unevaluated call
`coef(object)`	retrieve parameter estimates
`DTRstep(object)`	print description of method
`estimator(x)`	retrieve estimated value
`fitObject(object)`	retrieve value object returned for regression steps
`optTx(object)`	retrieve recommended optimal treatments for training data
`optTx(object, newdata)`	estimate optimal treatments based on a prior analysis
`outcome(object)`	retrieve value object returned for outcome regression
`plot(x, suppress)`	plot regression results
`summary(object)`	retrieve regression summaries

```
coef(object = qObj_ss)
```

```
$outcome
$outcome$`Subset=A_0`
$outcome$`Subset=A_0`$Combined
  (Intercept)           x01           x02           x11
-13.820921663   1.709259320  -0.008764618  -0.187254365
          x12           A2C       x11:A2C       x12:A2C
  0.260286153  -2.483716256   0.736930523   0.039199961


$outcome$`Subset=B_0`
$outcome$`Subset=B_0`$Combined
 (Intercept)           x01           x11           x13
-17.18636701   1.37322165   1.29859595  -0.01385819
         A2C       x11:A2C       x13:A2C
  0.78390504  -1.97408271   0.69372810
```

```
coef(object = qObj_fs)
```

```
$outcome
$outcome$Combined
(Intercept)          x01          x02          A1B      x02:A1B
-2.43410280   0.65178872 -0.01837287   3.86748929 -0.09121476
```

```
fitObj <- fitObject(object = qObj_ss)
fitObj
```

```
$outcome
$outcome$`Subset=A_0`
$outcome$`Subset=A_0`$Combined

Call:
lm(formula = YinternalY ~ x01 + x02 + x11 + x12 + A2 + x11:A2 +
    x12:A2, data = data)

Coefficients:
(Intercept)          x01          x02          x11
 -13.820922     1.709259    -0.008765    -0.187254
        x12          A2C      x11:A2C      x12:A2C
   0.260286    -2.483716     0.736931     0.039200



$outcome$`Subset=B_0`
$outcome$`Subset=B_0`$Combined

Call:
lm(formula = YinternalY ~ x01 + x11 + x13 + A2 + x11:A2 + x13:A2,
    data = data)

Coefficients:
(Intercept)          x01          x11          x13
  -17.18637      1.37322      1.29860     -0.01386
         A2C      x11:A2C      x13:A2C
    0.78391     -1.97408      0.69373
```

```
is(object = fitObj$outcome$'Subset=A_0'$Combined)
```

```
[1] "lm"       "oldClass"
```

Methods available: Training Diagnostics

Methods available: Training Diagnostics

Method	Description
`Call(name)`	retrieve the unevaluated call
`coef(object)`	retrieve parameter estimates
`DTRstep(object)`	print description of method
`estimator(x)`	retrieve estimated value
`fitObject(object)`	retrieve value object returned for regression steps
`optTx(object)`	retrieve recommended optimal treatments for training data
`optTx(object, newdata)`	estimate optimal treatments based on a prior analysis
`outcome(object)`	retrieve value object returned for outcome regression
`plot(x, suppress)`	plot regression results
`summary(object)`	retrieve regression summaries

```
ot <- optTx(x = qObj_ss)

names(ot)
```

```
[1] "optimalTx"    "decisionFunc"
```

```
table(ot$optimalTx)
```

```
  A   B   C
337 340 323
```

```
head(cbind(ot$decisionFunc,ot$optimalTx))
```

```
           A          B           C
[1,] 2.67502         NA -5.56995495 1
[2,]      NA  0.5879284  2.17568821 3
[3,]      NA  4.2127188  1.36644014 2
[4,]      NA -0.9162128 -0.07949545 3
[5,]      NA  1.1769982  1.43376155 3
[6,]      NA         NA          NA 1
```

What do the NA values mean?

```
ot <- optTx(x = qObj_ss)

names(ot)
```

```
[1] "optimalTx"    "decisionFunc"
```

```
table(ot$optimalTx)
```

```
  A   B   C
337 340 323
```

```
head(cbind(ot$decisionFunc,ot$optimalTx))
```

```
           A          B           C
[1,] 2.67502         NA -5.56995495 1
[2,]      NA  0.5879284  2.17568821 3
[3,]      NA  4.2127188  1.36644014 2
[4,]      NA -0.9162128 -0.07949545 3
[5,]      NA  1.1769982  1.43376155 3
[6,]      NA         NA          NA 1
```

What do the NA values mean?

```
df[1:6,]
```

```
        x01 x02 A1 r1         x11       x12      x13 A2
1  8.792934  44  B  0  6.0408282 -5.307951 4.174856  C
2 10.277429  47  A  0  5.8617800 -6.332879 0.000000  B
3 11.084441  65  A  0 -0.4063732 -1.609532 0.000000  B
4  7.654302  64  A  0  4.2644090  4.537260 0.000000  B
5 10.429125  42  A  0  4.0659284 -6.526209 0.000000  C
6 10.506056  68  A  1  0.0000000  0.000000 0.000000  A
          Y
```

```
estimator(x = qObj_ss)
```

```
[1] 2.17416
```

This is **NOT** the estimator for the value $\mathcal{V}(d^{opt})$.

Recall:

$$\widehat{\mathcal{V}}_Q(d^{opt}) = n^{-1} \sum_{i=1}^{n} \tilde{V}_{1i} = n^{-1} \sum_{i=1}^{n} \max_{a_1 \in \Psi_1(H_{1i})} Q_1(H_{1i}, a_1; \widehat{\beta}_1).$$

```
estimator(x = qObj_ss)
```

```
[1] 2.17416
```

This is **NOT** the estimator for the value $\mathcal{V}(d^{opt})$.

Recall:

$$\widehat{\mathcal{V}}_Q(d^{opt}) = n^{-1} \sum_{i=1}^{n} \tilde{V}_{1i} = n^{-1} \sum_{i=1}^{n} \max_{a_1 \in \Psi_1(H_{1i})} Q_1(H_{1i}, a_1; \widehat{\beta}_1).$$

```
estimator(x = qObj_fs)
```

```
[1] 2.964196
```

```
ot <- optTx(x = qObj_fs)

names(ot)
```

```
[1] "optimalTx"    "decisionFunc"
```

```
table(ot$optimalTx)
```

```
  A   B
931  69
```

```
head(cbind(ot$decisionFunc, "optTX" = ot$optimalTx))
```

```
            A          B optTX
[1,] 2.488626  2.3426660     1
[2,] 3.401085  2.9814802     1
[3,] 3.596374  1.5349041     1
[4,] 1.379021 -0.5912341     1
```

Methods available: Predictions

Method	Description
`Call(name)`	retrieve the unevaluated call
`coef(object)`	retrieve parameter estimates
`DTRstep(object)`	print description of method
`estimator(x)`	retrieve estimated value
`fitObject(object)`	retrieve value object returned for regression steps
`optTx(object)`	retrieve recommended optimal treatments for training data
`optTx(object, newdata)`	estimate optimal treatments based on a prior analysis
`outcome(object)`	retrieve value object returned for outcome regression
`plot(x, suppress)`	plot regression results
`summary(object)`	retrieve regression summaries

```
newPatient <- data.frame(x01 = c(9.3, 10.6), x02 = c(51.8, 70))
optTx(x = qObj_fs, newdata = newPatient)
```

```
$optimalTx
[1] A A
Levels: A B

$decisionFunc
            A         B
[1,] 2.675818 1.8183822
[2,] 3.188757 0.6712126
```

```
newPatient <- cbind(newPatient,
                    "A1" = optTx(x = qObj_fs, newdata = newPatient)$optimalTx,
                    "r1" = c(1,0),
                    x11 = c(NA,2.9),
                    x12 = c(NA,-.6))
optTx(x = qObj_ss, newdata = newPatient)
```

```
$optimalTx
[1] A B
Levels: A B C

$decisionFunc
      A        B        C
[1,] NA       NA       NA
[2,] NA 2.984495 2.614357
```

Value Search

The augmented inverse probability weighted estimator for $\mathcal{V}(d_\eta)$ for fixed η is

$$\widehat{\mathcal{V}}_{AIPW}(d_\eta) = n^{-1}\sum_{i=1}^{n}\left[\frac{C_{d_\eta,i}Y_i}{\left\{\prod_{k=2}^{K}\pi_{d_\eta,k}(\overline{X}_{ki};\overline{\eta}_k,\widehat{\gamma}_k)\right\}\pi_{d_\eta,1}(X_{1i};\eta_1,\widehat{\gamma}_1)}\right.$$
$$\left.+\sum_{k=1}^{K}\left\{\frac{C_{\overline{d}_\eta,k-1,i}}{\overline{\pi}_{d_\eta,k-1}(\overline{X}_{k-1,i};\widehat{\overline{\gamma}}_{k-1})}-\frac{C_{\overline{d}_\eta,k,i}}{\overline{\pi}_{d_\eta,k}(\overline{X}_{ki};\widehat{\overline{\gamma}}_k)}\right\}\mathcal{Q}_{d_\eta,k}(\overline{X}_{ki};\widehat{\beta}_k)\right],$$

where $C_{d_\eta} = \mathrm{I}\{\overline{A} = \overline{d}_\eta(\overline{X})\}$;

$$\pi_{d_\eta,1}(X_1;\gamma_1) = \omega_1(X_1,1;\gamma_1)\mathrm{I}\{d_{\eta,1}(X_1)=1\} + \omega_1(X_1,0;\gamma_1)\mathrm{I}\{d_{\eta,1}(X_1)=0\},$$

$$\pi_{d_\eta,k}(\overline{X}_k;\gamma_k) = \omega_k\{\overline{X}_k,\overline{d}_{\eta,k-1}(\overline{X}_{k-1}),1;\gamma_k\}\ \mathrm{I}[d_{\eta,k}\{\overline{X}_k,\overline{d}_{\eta,k-1}(\overline{X}_{k-1})\}=1]$$
$$+\omega_k\{\overline{X}_k,\overline{d}_{\eta,k-1}(\overline{X}_{k-1}),0;\gamma_k\}\ \mathrm{I}[d_{\eta,k}\{\overline{X}_k,\overline{d}_{\eta,k-1}(\overline{X}_{k-1})\}=0].$$

Further $\omega_k(h_k,a_k;\gamma_k)$, $k=1,\ldots,K$ are models for the propensity scores $P(A_k=a_k|H_k=h_k)$, and $\widehat{\gamma}_k$ is a suitable estimator for γ_k, $k=1,\ldots,K$.

Finally, $\mathcal{Q}_{d_\eta,k}(\overline{X}_{ki};\widehat{\beta}_k)$ are models for the conditional expectations $E\{Y^*(d_\eta)|\overline{X}^*_k(\overline{d}_{\eta,k-1})=\overline{x}_k\}$, $k=1,\ldots,K$. The IPW estimator is the special case of setting these to zero.

```
str(object = optimalSeq)
```

```
function (..., moPropen, moMain, moCont, data, response,
    txName, regimes, fSet = NULL, refit = FALSE, iter = 0L,
    verbose = TRUE)
```

Argument	Class	Description
`...`		Additional inputs for rgenoud()
`moPropen`	`modelObj`	for propensity regression
`moMain`	`modelObj`	for main effects terms (mu)
`moCont`	`modelObj`	for terms interacting with treatment (C)
`data`	`data.frame`	covariates and treatment history
`response`	`vector`	outcome of interest
`txName`	`character`	treatment variable name
`regimes`	`function`	definition of restricted class of regimes
`fSet`	`function`	feasible set definition
`refit`		deprecated
`iter`	`integer`	if >0, iterative methods used
`verbose`	`logical`	if FALSE, screen prints suppressed

```
str(object = optimalSeq)
```

```
function (..., moPropen, moMain, moCont, data, response,
    txName, regimes, fSet = NULL, refit = FALSE, iter = 0L,
    verbose = TRUE)
```

Postulate a model for $\pi_1(h_1, a_1)$ and use maximum likelihood.

$$\text{logit}(A1) \sim \gamma_1$$

Postulate a model for $\pi_2(h_2, a_2)$ for each subset and use maximum likelihood.

$$\text{logit}(A2) \sim \gamma_2$$

Need to use *buildModelObjSubset()* for **all** decision points

```
str(object = optimalSeq)
```

```
function (..., moPropen, moMain, moCont, data, response,
    txName, regimes, fSet = NULL, refit = FALSE, iter = 0L,
    verbose = TRUE)
```

```
moPropen_fs <- buildModelObjSubset(model = ~ 1,
                                   solver.method = 'glm',
                                   solver.args = list("family" = "binomial"),
                                   predict.args = list("type" = "response"),
                                   subset = 'fs', dp = 1L)

moPropen_A_0 <- buildModelObjSubset(model = ~ 1,
                                    solver.method = 'glm',
                                    solver.args = list("family" = "binomial"),
                                    predict.args = list("type" = "response"),
                                    subset = 'A_0', dp = 2L)

moPropen_B_0 <- buildModelObjSubset(model = ~ 1,
                                    solver.method = 'glm',
                                    solver.args = list("family" = "binomial"),
                                    predict.args = list("type" = "response"),
                                    subset = 'B_0', dp = 2L)
```

```
str(object = optimalSeq)
```

```
function (..., moPropen, moMain, moCont, data, response,
    txName, regimes, fSet = NULL, refit = FALSE, iter = 0L,
    verbose = TRUE)
```

Previously postulate models for $Q_2(h_2, a_2)$ for subset A_0 and for subset B_0 can be reused.

```
moMain_A_0 <- buildModelObjSubset(model = ~ x01 + x02 + x11 + x12,
                                  solver.method = 'lm',
                                  subset = 'A_0', dp = 2L)

moCont_A_0 <- buildModelObjSubset(model = ~ x11 + x12,
                                  solver.method = 'lm',
                                  subset = 'A_0', dp = 2L)

moMain_B_0 <- buildModelObjSubset(model = ~ x01 + x11 + x13,
                                  solver.method = 'lm',
                                  subset = 'B_0', dp = 2L)

moCont_B_0 <- buildModelObjSubset(model = ~ x11 + x13,
                                  solver.method = 'lm',
                                  subset = 'B_0', dp = 2L)
```

But the model for $Q_1(h_1, a_1)$ must be redefined using buildModelObjSubset().

$$Q_1(\mathrm{H}_1, a_1) = \beta_{10} + \beta_{11}\mathrm{x01} + \beta_{12}\mathrm{x02} + \mathrm{A1}(\beta_{13} + \beta_{14}\mathrm{x02})$$

```
moMain_fs <- buildModelObjSubset(model = ~ x01 + x02,
                                 solver.method = 'lm',
                                 subset = 'fs', dp = 1L)

moCont_fs <- buildModelObjSubset(model = ~ x02,
                                 solver.method = 'lm',
                                 subset = 'fs', dp = 1L)
```

```
str(object = optimalSeq)
```

```
function (..., moPropen, moMain, moCont, data, response,
    txName, regimes, fSet = NULL, refit = FALSE, iter = 0L,
    verbose = TRUE)
```

- `data` is a complete dataset, (df)
- `response` is the "vector" outcome of interest (df$y)
- `txName` is a vector of the treatment variable names, ("A1", "A2")
- `fSet` is a list of the fSet functions (fSet1, fSet2)

```
str(object = optimalSeq)
```

```
function (..., moPropen, moMain, moCont, data, response,
    txName, regimes, fSet = NULL, refit = FALSE, iter = 0L,
    verbose = TRUE)
```

Define the elements of the first stage restricted class of regimes as

$$d_1(h_1, \eta) = \begin{cases} A & \mathrm{I}(x_{01} < \eta_1 \ \& \ x_{02} < \eta_2) \\ B & \mathrm{I}(x_{01} \geq \eta_1 \ || \ x_{02} \geq \eta_2) \end{cases}$$

```
regimes1 <- function(eta1, eta2, data) {
  tst <- {data$x01 < eta1} & {data$x02 < eta2}
  d1 <- c("B", "A")[tst + 1L]
  d1
}
```

```
str(object = optimalSeq)
```

```
function (..., moPropen, moMain, moCont, data, response,
    txName, regimes, fSet = NULL, refit = FALSE, iter = 0L,
    verbose = TRUE)
```

Define the elements of the second stage restricted class of regimes as

$$d_1(h_1, \eta) = \begin{cases} A & \mathrm{I}(\mathrm{r1} = 1 \;\&\; \mathrm{A1} = A) \\ B & \mathrm{I}(\mathrm{r1} = 0 \;\&\; \mathrm{A1} = A)\, \mathrm{I}(\mathrm{x11} < \eta_1 \;\mid\; \mathrm{x12} < \eta_2) \\ C & \mathrm{I}(\mathrm{r1} = 0 \;\&\; \mathrm{A1} = A)\, \mathrm{I}(\mathrm{x11} \geq \eta_1 \;\mid\; \mathrm{x12} \geq \eta_2) \\ B & \mathrm{I}(\mathrm{r1} = 1 \;\&\; \mathrm{A1} = B) \\ A & \mathrm{I}(\mathrm{r1} = 0 \;\&\; \mathrm{A1} = B)\, \mathrm{I}(\mathrm{x11} < \eta_3 \;\mid\; \mathrm{x13} < \eta_4) \\ C & \mathrm{I}(\mathrm{r1} = 0 \;\&\; \mathrm{A1} = B)\, \mathrm{I}(\mathrm{x11} \geq \eta_3 \;\mid\; \mathrm{x13} \geq \eta_4) \end{cases}$$

```
regimes2 <- function(eta1, eta2, eta3, eta4, data) {
  d2 <- levels(data$A1)[data$A1]

  A_0 <- {data$r1 == 0L} & {data$A1 == "A"}
  B_0 <- {data$r1 == 0L} & {data$A1 == "B"}

  tstA <- {data$x11 < eta1} & {data$x12 < eta2}
  tstB <- {data$x11 < eta3} & {data$x13 < eta4}

  d2[A_0] <- c("C", "B")[tstA[A_0] + 1L]
  d2[B_0] <- c("C", "A")[tstB[B_0] + 1L]

  d2
}
```

```
str(object = optimalSeq)
```

```
function (..., moPropen, moMain, moCont, data, response,
    txName, regimes, fSet = NULL, refit = FALSE, iter = 0L,
    verbose = TRUE)
```

Uses genetic algorithm implemented by genoud()

Additional inputs for genoud() passed through ellipsis. Must include

- `starting.values` initial estimates for regime parameters
- `Domains` matrix defining search space
- `pop.size` population size

Remember to consider your covariates

```
summary(object = dfHold$x12)
```

```
    Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
-14.0136  -3.9415  -0.5643  -0.7864   2.4046  10.9525
    NA's
     212
```

```
summary(object = dfHold$x02)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  40.00   50.75   61.00   60.47   70.00   80.00
```

```
str(object = optimalSeq)
```

```
function (..., moPropen, moMain, moCont, data, response,
    txName, regimes, fSet = NULL, refit = FALSE, iter = 0L,
    verbose = TRUE)
```

Uses genetic algorithm implemented by genoud()

Additional inputs forgenoud() passed through ellipsis. Must include

- `starting.values` initial estimates for regime parameters
- `Domains` matrix defining search space
- `pop.size` population size

```
etas <- c("x01", "x02", "x11", "x12", "x11", "x13")
starting.values <- lapply(dfHold[etas], mean, na.rm = TRUE) |> unlist() |> unname()
Domains <- matrix(data = c(lapply(dfHold[etas], min, na.rm = TRUE) |> unlist() |> unname(),
                           lapply(dfHold[etas], max, na.rm = TRUE) |> unlist() |> unname()),
                  ncol = 2L)
pop.size <- 500 #TOO SMALL
```

```
vsObj <- optimalSeq(moPropen = list(moPropen_fs, moPropen_A_0, moPropen_B_0),
                    moMain = list(moMain_fs, moMain_A_0, moMain_B_0),
                    moCont = list(moCont_fs, moCont_A_0, moCont_B_0),
                    iter = 0L,
                    data = df, response = df$y, txName = c('A1','A2'),
                    regimes = list(regimes1, regimes2),
                    fSet = list(fSet1, fSet2),
                    Domains = Domains,
                    starting.values = starting.values,
                    pop.size = 500,
                    verbose = TRUE)
```

```
vsObj <- optimalSeq(moPropen = list(moPropen_fs, moPropen_A_0, moPropen_B_0),
                    moMain = list(moMain_fs, moMain_A_0, moMain_B_0),
                    moCont = list(moCont_fs, moCont_A_0, moCont_B_0),
                    iter = 0L,
                    data = df, response = df$y, txName = c('A1', 'A2'),
                    regimes = list(regimes1, regimes2),
                    fSet = list(fSet1, fSet2),
                    Domains = Domains,
                    starting.values = starting.values,
                    pop.size = 500,
                    verbose = TRUE)
```

```
Value Search - Coarsened Data Perspective 2 Decision Points
Decision point 1

Subsets of treatment identified as:
$fs
[1] "A" "B"

Number of patients in data for each subset:
  fs
1000
Decision point 2

Subsets of treatment identified as:
$A_0
[1] "B" "C"

$A_1
[1] "A"

$B_0
[1] "A" "C"

$B_1
[1] "B"

Number of patients in data for each subset:
A_0 A_1 B_0 B_1
411 108 377 104

Propensity for treatment regression.
```

Methods available:

method	description
Call(name)	retrieve the unevaluated call
coef(object)	retrieve parameter estimates
DTRstep(object)	print description of method
estimator(x)	retrieve estimated value
fitObject(object)	retrieve value returned by 'modelObj's
genetic(object)	retrieve value returned by genoud()
optTx(object)	retrieve recommended optimal treatments
optTx(object, newdata)	estimate optimal treatments based on a prior analysis
outcome(object)	retrieve value returned by outcome regression analysis
plot(x, suppress)	plot fit results
propen(object)	retrieve value returned by propensity regression analysis
regimeCoef(object)	retrieve estimated regime parameters
summary(object)	retrieve summary information

Methods available: Model Diagnostics

method	description
Call(name)	retrieve the unevaluated call
coef(object)	retrieve parameter estimates
DTRstep(object)	print description of method
estimator(x)	retrieve estimated value
fitObject(object)	retrieve value returned by 'modelObj's
genetic(object)	retrieve value returned by genoud()
optTx(object)	retrieve recommended optimal treatments
optTx(object, newdata)	estimate optimal treatments based on a prior analysis
outcome(object)	retrieve value returned by outcome regression analysis
plot(x, suppress)	plot fit results
propen(object)	retrieve value returned by propensity regression analysis
regimeCoef(object)	retrieve estimated regime parameters
summary(object)	retrieve summary information

```
coef(object = vsObj)
```

```
$propensity
$propensity$`dp=1`
$propensity$`dp=1`$`Subset=fs`
(Intercept)
-0.07603661


$propensity$`dp=2`
$propensity$`dp=2`$`Subset=A_0`
(Intercept)
-0.08277229

$propensity$`dp=2`$`Subset=B_0`
(Intercept)
 0.03713955



$outcome
$outcome$`dp=1`
$outcome$`dp=1`$`Subset=fs`
$outcome$`dp=1`$`Subset=fs`$Combined
(Intercept)         x01         x02         A1B     x02:A1B
-2.43410280  0.65178872 -0.01837287  3.86748929 -0.09121476
```

```
fitObj <- fitObject(object = vsObj)
print(x = fitObj)
```

```
$propensity
$propensity$`dp=1`
$propensity$`dp=1`$`Subset=fs`

Call:  glm(formula = YinternalY ~ 1, family = "binomial", data = data)

Coefficients:
(Intercept)
   -0.07604

Degrees of Freedom: 999 Total (i.e. Null);  999 Residual
Null Deviance:      1385
Residual Deviance: 1385     AIC: 1387


$propensity$`dp=2`
$propensity$`dp=2`$`Subset=A_0`

Call:  glm(formula = YinternalY ~ 1, family = "binomial", data = data)

Coefficients:
(Intercept)
   -0.08277

Degrees of Freedom: 410 Total (i.e. Null);  410 Residual
Null Deviance:      569.1
Residual Deviance: 569.1    AIC: 571.1

$propensity$`dp=2`$`Subset=B_0`

Call:  glm(formula = YinternalY ~ 1, family = "binomial", data = data)

Coefficients:
(Intercept)
    0.03714

Degrees of Freedom: 376 Total (i.e. Null);  376 Residual
Null Deviance:      522.5
Residual Deviance: 522.5    AIC: 524.5
```

```
genetic(object = vsObj)
```

```
$value
[1] 5.322525

$par
[1] 12.227718 78.383856  3.379561  2.720807 -2.543702
[6]  5.861444

$gradients
[1] NA NA NA NA NA NA

$generations
[1] 20

$peakgeneration
[1] 9

$popsize
[1] 500

$operators
[1] 65 62 62 62 62 62 62 62  0
```

Methods available: Training Diagnostics

method	description
Call(name)	retrieve the unevaluated call
coef(object)	retrieve parameter estimates
DTRstep(object)	print description of method
estimator(x)	retrieve estimated value
fitObject(object)	retrieve value returned by 'modelObj's
genetic(object)	retrieve value returned by genoud()
optTx(object)	retrieve recommended optimal treatments
optTx(object, newdata)	estimate optimal treatments based on a prior analysis
outcome(object)	retrieve value returned by outcome regression analysis
plot(x, suppress)	plot fit results
propen(object)	retrieve value returned by propensity regression analysis
regimeCoef(object)	retrieve estimated regime parameters
summary(object)	retrieve summary information

```
ot <- optTx(x = vsObj)
names(ot)
```

```
[1] "dp=1" "dp=2"
```

```
table(ot$"dp=1"$optimalTx)
```

```
  A   B
940  60
```

```
table(ot$"dp=2"$optimalTx)
```

```
  A   B   C
203 294 503
```

```
ot$"dp=1"$decisionFunc
```

```
[1] NA
```

```
regimeCoef(object = vsObj)
```

```
$`dp=1`
    eta1     eta2
12.22772 78.38386

$`dp=2`
     eta1      eta2      eta3      eta4
 3.379561  2.720807  2.543702  5.861444
```

Methods available: Predictions

method	description
Call(name)	retrieve the unevaluated call
coef(object)	retrieve parameter estimates
DTRstep(object)	print description of method
estimator(x)	retrieve estimated value
fitObject(object)	retrieve value returned by 'modelObj's
genetic(object)	retrieve value returned by genoud()
optTx(object)	retrieve recommended optimal treatments
optTx(object, newdata)	estimate optimal treatments based on a prior analysis
outcome(object)	retrieve value returned by outcome regression analysis
plot(x, suppress)	plot fit results
propen(object)	retrieve value returned by propensity regression analysis
regimeCoef(object)	retrieve estimated regime parameters
summary(object)	retrieve summary information

```
newPatient <- data.frame(x01 = c(9.3, 10.6), x02 = c(51.8, 70))
optTx(x = vsObj, newdata = newPatient, dp = 1L)
```

```
$optimalTx
[1] A A
Levels: A B

$decisionFunc
[1] NA
```

```
newPatient <- cbind(newPatient,
                    "A1" = optTx(x = vsObj, newdata = newPatient, dp = 1L)$optimalTx
                    "r1" = c(1,0),
                    x11 = c(NA,2.9),
                    x12 = c(NA,-.6),
                    x13 = c(NA, NA))
optTx(x = vsObj, newdata = newPatient, dp = 2L)
```

```
$optimalTx
[1] A B
Levels: A B C

$decisionFunc
[1] NA
```

Backward Outcome Weighted Learning

Backward outcome weighted learning (BOWL) can be viewed as the application of the OWL method within the framework of the backward iterative algorithm.

Let

$$\widehat{\mathcal{V}}_{IPW}^{(K)}(d_{\eta,K}) = n^{-1} \sum_{i=1}^{n} \frac{\mathrm{I}\left\{A_{Ki} = d_{\eta,K}(H_{Ki})\right\} Y_i}{\omega_K(H_{Ki}, A_{Ki}; \widehat{\gamma}_K)},$$

where $\omega_k(h_k, a_k; \gamma_k)$ is a model for $\omega_k(h_k, a_k) = P(A_k = a_k | H_k = h_k)$ and $\widehat{\gamma}_k$ is a suitable estimator of γ_k.

```
str(object = bowl)
```

```
function (..., moPropen, data, reward, txName, regime,
    response, BOWLObj = NULL, lambdas = 2, cvFolds = 0L,
    kernel = "linear", kparam = NULL, fSet = NULL,
    surrogate = "hinge", verbose = 2L)
```

argument	class	description
...		Ignored. Included to require named input.
moPropen	"modelObj"	propensity regression
data	"data.frame"	covariates and treatment histories
reward	"vector"	outcome of interest
txName	"character"	treatment variable name
regime	"formula"	covariates of the decision function
response	"numeric"	outcome of interest
BOWLObj	"BOWL" or NULL	NULL or the value object of the prior call to bowl()
lambdas	"numeric"	one or more tuning parameters
cvFolds	"integer"	number of cross-validation steps
kernel	"character"	one of {linear, poly, radial}
kparam	"numeric" or NULL	kernel parameter
fSet	"function" or NULL	feasible treatment set definition
surrogate	"character"	one of {'logit', 'exp', 'hinge', 'sqhinge', 'huber'}
verbose	"logical" or "numeric"	level of screen printing.

Accomplished through repeated call to bowl()

```
str(object = bowl)
```

```
function (..., moPropen, data, reward, txName, regime,
    response, BOWLObj = NULL, lambdas = 2, cvFolds = 0L,
    kernel = "linear", kparam = NULL, fSet = NULL,
    surrogate = "hinge", verbose = 2L)
```

The first STEP of the BOWL algorithm is the analysis of STAGE 2

Postulate a model for $\pi_2(h_2, a_2)$ for each subset and use maximum likelihood.

$$\text{logit}(A2) \sim \gamma_2$$

Use a list of the subset modeling objects.

```
moPropen_ss <- list(moPropen_A_0, moPropen_B_0)
```

```
str(object = bowl)
```

```
function (..., moPropen, data, reward, txName, regime,
    response, BOWLObj = NULL, lambdas = 2, cvFolds = 0L,
    kernel = "linear", kparam = NULL, fSet = NULL,
    surrogate = "hinge", verbose = 2L)
```

- `data` is a complete dataset, (df)
- `response`/`reward` is the "vector" outcome of interest (df$y)
- `txName` is the treatment variable name, ("A2")
- `BOWLObj` is NULL or the value object returned by a previous call to bowl() (NULL)
- `surrogate` is the surrogate function to be used for the 0-1 loss function ("logit", "exp", "hinge", "sqhinge", "huber")

```
str(object = bowl)
```

```
function (..., moPropen, data, reward, txName, regime,
    response, BOWLObj = NULL, lambdas = 2, cvFolds = 0L,
    kernel = "linear", kparam = NULL, fSet = NULL,
    surrogate = "hinge", verbose = 2L)
```

- `regime` is a formula (or list of formula) specifying the form of the decision function

$$f_{\prime rdp\prime}(X;\eta_{\prime rdp\prime}) = \sum_{i=1}^{n} \eta_{\prime rdp\prime i} k(X, X_i) + \eta_{\prime rdp\prime 0}$$

We will use linear kernels and thus

$$f_{\prime rdp\prime}(X;\eta_{\prime rdp\prime}) = \sum_{i=1}^{n} \eta_{\prime rdp\prime i} X_i + \eta_{\prime rdp\prime 0}$$

with $X_i = \{\text{x11}, \text{x12}\}$ for subset A_0 and $X_i = \{\text{x11}, \text{x12}, \text{x13}\}$ for subset B_0

```
regime <- list("A_0" = ~x11+x12, "B_0" = ~x11+x12+x13)
```

```
str(object = bowl)
```

```
function (..., moPropen, data, reward, txName, regime,
    response, BOWLObj = NULL, lambdas = 2, cvFolds = 0L,
    kernel = "linear", kparam = NULL, fSet = NULL,
    surrogate = "hinge", verbose = 2L)
```

---

fSet is the previously defined function

```
fSet2 <- function(A1, r1) {

  list("subsets" = list(list("A_1", c("A")),
                        list("A_0", c("B", "C")),
                        list("B_1", c("B")),
                        list("B_0", c("A", "C"))),
       "txOpts" = paste(A1, r1, sep = "_"))

}
```

```
moPropen_A_0 <- buildModelObjSubset(model = ~ 1,
                                    solver.method = 'glm',
                                    solver.args = list("family" = "binomial"),
                                    predict.args = list("type" = "response"),
                                    subset = 'A_0', dp = 2L)
moPropen_B_0 <- buildModelObjSubset(model = ~ 1,
                                    solver.method = 'glm',
                                    solver.args = list("family" = "binomial"),
                                    predict.args = list("type" = "response"),
                                    subset = 'B_0', dp = 2L)

moPropen_ss <- list(moPropen_A_0, moPropen_B_0)

regime <- list("A_0" = ~ x11 + x12, "B_0" = ~x11 + x12 + x13)

fSet2 <- function(A1, r1) {

  list("subsets" = list(list("A_1", c("A")),
                        list("A_0", c("B", "C")),
                        list("B_1", c("B")),
                        list("B_0", c("A", "C"))),
       "txOpts" = paste(A1, r1, sep = "_"))

}

bObj_ss <- bowl(moPropen = moPropen_ss,
                data = df, response = df$y, txName = 'A2',
                regime = regime, BOWLObj = NULL,
                kernel = list("A_0"='linear',"B_0"='linear'), kparam = NULL,
                fSet = fSet2, surrogate = 'sqhinge', verbose = TRUE)
```

```
bObj_ss <- bowl(moPropen = moPropen_ss,
                data = df, response = df$y, txName = 'A2',
                regime = regime, BOWLObj = NULL,
                kernel = list("A_0"='linear',"B_0"='linear'), kparam = NULL,
                fSet = fSet2, surrogate = 'sqhinge', verbose = TRUE)
```

```
BOWL optimization step 1

Subsets of treatment identified as:
$A_0
[1] "B" "C"

$A_1
[1] "A"

$B_0
[1] "A" "C"

$B_1
[1] "B"

Number of patients in data for each subset:
A_0 A_1 B_0 B_1
411 108 377 104

Propensity for treatment regression.

subset(s) A_1, B_1 excluded from propensity regression

Fitting models for  A_0 using 411 patient records.
Regression analysis for moPropen:

Call:  glm(formula = YinternalY ~ 1, family = "binomial", data = data)

Coefficients:
(Intercept)
   -0.08277

Degrees of Freedom: 410 Total (i.e. Null);  410 Residual
Null Deviance:      569.1
Residual Deviance: 569.1    AIC: 571.1
Fitting models for  B_0 using 377 patient records.
```

Accomplished through repeated calls to bowl()

```
str(object = bowl)
```

```
function (..., moPropen, data, reward, txName, regime,
    response, BOWLObj = NULL, lambdas = 2, cvFolds = 0L,
    kernel = "linear", kparam = NULL, fSet = NULL,
    surrogate = "hinge", verbose = 2L)
```

The second STEP of the BOWL algorithm is the analysis of STAGE 1

Postulate a model for $\pi_1(h_1, a_1)$ and use maximum likelihood.

$$\text{logit}(A1) \sim \gamma_1$$

```
moPropen_fs <- buildModelObj(model = ~ 1,
                             solver.method = 'glm',
                             solver.args = list("family" = "binomial"),
                             predict.args = list("type" = "response"))
```

```
str(object = bowl)
```

```
function (..., moPropen, data, reward, txName, regime,
    response, BOWLObj = NULL, lambdas = 2, cvFolds = 0L,
    kernel = "linear", kparam = NULL, fSet = NULL,
    surrogate = "hinge", verbose = 2L)
```

- **data** is a complete dataset, (df)
- **response**/**reward** is the stage reward (0)
- **BOWLObj** is NULL or the value object returned by the second stage analysis (`bObj_ss`)
- **txName** is the treatment variable name, ("A1")
- **fSet** is not required
- **surrogate** is the surrogate function to be used for the 0-1 loss function ('logit', 'exp', 'hinge', 'sqhinge', 'huber')

```
str(object = bowl)
```

```
function (..., moPropen, data, reward, txName, regime,
    response, BOWLObj = NULL, lambdas = 2, cvFolds = 0L,
    kernel = "linear", kparam = NULL, fSet = NULL,
    surrogate = "hinge", verbose = 2L)
```

- **regime** is a formula (or list of formulas) defining the decision function

We will again use a linear kernel and thus

$$f_{\prime rdp\prime}(X;\eta_{\prime rdp\prime}) = \sum_{i=1}^{n} \eta_{\prime rdp\prime i} X_i + \eta_{\prime rdp\prime 0}$$

with $X_i = \{\text{x01}, \text{x02}\}$.

```
regime <- ~x01+x02
```

```
moPropen_fs <- buildModelObj(model = ~ 1,
                             solver.method = 'glm',
                             solver.args = list("family" = "binomial"),
                             predict.args = list("type" = "response"))

regime <- ~ x01 + x02

bObj_fs <- bowl(moPropen = moPropen_fs,
                data = df, response = rep(0, nrow(df)), txName = 'A1',
                regime = regime, BOWLObj = bObj_ss,
                kernel = 'linear', kparam = NULL,
                fSet = NULL, surrogate = 'sqhinge', verbose = TRUE)
```

```
bObj_fs <- bowl(moPropen = moPropen_fs,
                data = df, response = rep(0, nrow(df)), txName = 'A1',
                regime = regime, BOWLObj = bObj_ss,
                kernel = 'linear', kparam = NULL,
                fSet = NULL, surrogate = 'sqhinge', verbose = TRUE)
```

```
BOWL optimization step 2

Propensity for treatment regression.
Regression analysis for moPropen:

Call:  glm(formula = YinternalY ~ 1, family = "binomial", data = data)

Coefficients:
(Intercept)
   -0.07604

Degrees of Freedom: 999 Total (i.e. Null);  999 Residual
Null Deviance:      1385
Residual Deviance: 1385     AIC: 1387

Outcome regression.
No outcome regression performed.

Final optimization step.
Optimization Results

Kernel
kernel = linear
kernel model = ~x01 + x02 - 1
lambda=  2
Surrogate: SqHingeSurrogate
$par
[1]  0.16414926 -0.01779646 -0.00105415

$value
[1] 54.10977

$counts
function gradient
      29        5
```

Methods available:

method	description
Call(name)	retrieve the unevaluated call
coef(object)	retrieve parameter estimates
DTRstep(object)	print description of method
estimator(x)	retrieve estimated value
fitObject(object)	retrieve value returned by 'modelObj's
optTx(object)	retrieve recommended optimal treatments
optTx(object, newdata)	estimate optimal treatments based on a prior analysis
outcome(object)	retrieve value returned by regression analysis
plot(x, suppress)	plot fit results
summary(object)	retrieve summary information

Methods available: Model Diagnostics

method	description
Call(name)	retrieve the unevaluated call
coef(object)	retrieve parameter estimates
DTRstep(object)	print description of method
estimator(x)	retrieve estimated value
fitObject(object)	retrieve value returned by 'modelObj's
optTx(object)	retrieve recommended optimal treatments
optTx(object, newdata)	estimate optimal treatments based on a prior analysis
outcome(object)	retrieve value returned by regression analysis
plot(x, suppress)	plot fit results
summary(object)	retrieve summary information

```
coef(object = bObj_ss)
```

```
$propensity
$propensity$`Subset=A_0`
(Intercept)
-0.08277229

$propensity$`Subset=B_0`
(Intercept)
 0.03713955
```

```
coef(object = bObj_fs)
```

```
$propensity
(Intercept)
-0.07603661
```

```
fitObj <- fitObject(object = bObj_ss)
fitObj
```

```
$propensity
$propensity$`Subset=A_0`

Call:  glm(formula = YinternalY ~ 1, family = "binomial", data = data)

Coefficients:
(Intercept)
   -0.08277

Degrees of Freedom: 410 Total (i.e. Null);  410 Residual
Null Deviance:      569.1
Residual Deviance: 569.1    AIC: 571.1

$propensity$`Subset=B_0`

Call:  glm(formula = YinternalY ~ 1, family = "binomial", data = data)

Coefficients:
(Intercept)
    0.03714

Degrees of Freedom: 376 Total (i.e. Null);  376 Residual
Null Deviance:      522.5
Residual Deviance: 522.5    AIC: 524.5
```

```
#is(object = fitObj$outcome$'Subset=A_0'$Combined)
```

Methods available: Training Diagnostics

method	description
Call(name)	retrieve the unevaluated call
coef(object)	retrieve parameter estimates
DTRstep(object)	print description of method
estimator(x)	retrieve estimated value
fitObject(object)	retrieve value returned by 'modelObj's
optTx(object)	retrieve recommended optimal treatments
optTx(object, newdata)	estimate optimal treatments based on a prior analysis
outcome(object)	retrieve value returned by regression analysis
plot(x, suppress)	plot fit results
summary(object)	retrieve summary information

```
ot <- optTx(x = bObj_ss)
names(ot)
```

```
[1] "optimalTx"    "decisionFunc"
```

```
table(ot$optimalTx)
```

```
  A   B   C
326 334 340
```

```
head(cbind(ot$decisionFunc, "optTx" = ot$optimalTx))
```

```
                optTx
[1,] -0.18465414     1
[2,]  0.05500416     3
[3,] -0.08997375     2
[4,]  0.03377397     3
[5,]  0.01095293     3
[6,]          NA     1
```

```
estimator(x = bObj_ss)
```

```
[1] 2.660591
```

```
ot <- optTx(x = bObj_fs)
table(ot$optimalTx)
```

Methods available: Predictions

method	description
Call(name)	retrieve the unevaluated call
coef(object)	retrieve parameter estimates
DTRstep(object)	print description of method
estimator(x)	retrieve estimated value
fitObject(object)	retrieve value returned by 'modelObj's
optTx(object)	retrieve recommended optimal treatments
optTx(object, newdata)	estimate optimal treatments based on a prior analysis
outcome(object)	retrieve value returned by regression analysis
plot(x, suppress)	plot fit results
summary(object)	retrieve summary information

```
newPatient <- data.frame(x01 = c(9.3, 10.6), x02 = c(51.8, 70))
optTx(x = bObj_fs, newdata = newPatient)
```

```
$optimalTx
[1] A A
Levels: A B

$decisionFunc
[1] -0.05596278 -0.09828371
```

```
newPatient <- cbind(newPatient,
                    "A1" = optTx(x = bObj_fs, newdata = newPatient)$optimalTx,
                    "r1" = c(1,0),
                    "x11" = c(NA,2.9),
                    "x12" = c(NA,-.6),
                    "x13" = c(NA,8))
optTx(x = bObj_ss, newdata = newPatient)
```

```
$optimalTx
[1] A B
Levels: A B C

$decisionFunc
[1]           NA -0.007807652
```

Conclusion

There are more multiple decision point methods available.

- optimalClass() can be used iteratively like qLearn() and bowl() (Value Search Classification)
- iqLearn() is a two-decision point method (IQ-Learning)
- earl() is a two-decision point method (Efficient Augmentation and Relaxation Learning)
- rwl() is a two-decision point method (Residual Weighted Learning)

If you have **ANY** questions or encounter any problems, please do not hesitate to contact me

Shannon Holloway sthollow@ncsu.edu