Using BoolTraineR to reconstruct asynchronous Boolean models

Chee Yee Lim

2015-11-29

Contents

1 Brief introduction

`BoolTraineR` is a model learning algorithm for reconstructing and training asynchronous Boolean models using single-cell expression data. Refer to the paper for more details on the concepts behind the algorithm. This vignette serves as a tutorial to demonstrate example workflows that can be adapted to individual cases experienced by users.

Running `BoolTraineR` is straightforward. However, note that depending on the (1) size of single-cell expression data and (2) complexity of Boolean model, `BoolTraineR` may take a long time to complete the computation. In such cases, it is advisable to use the built-in parallel processing capability of `BoolTraineR`. This can be easily achieved by using `doParallel` package, as illustrated in the example.

Note that the examples presented in this vignette are different from the results presented in our paper. The examples presented here have been simplified to speed up the processing time.

2 Installation

`BoolTraineR` can be installed from CRAN.

```
install.packages('BoolTraineR')
```

Or from Github for the latest version.

```
install_github("cheeyeelim/booltrainer")
```

Also install `doParallel` package if you intend to use parallel processing.

3 Input data format

Depending on the analysis, only 3 types of data will ever be needed. The format of the data required is discussed below.

1. Expression data. A matrix with genes on the columns, and cells on the row.

The expression data should be preprocessed as in any standard sequencing data processing pipelines, which includes quality control filtering and normalisation.

Use `initialise_raw_data` to convert expression data into a suitable format for model inference.

```
data(wilson_raw_data)
round(wilson_raw_data[1:5,1:5], 4)

edata = initialise_raw_data(wilson_raw_data)
```

	bptf	cbfa2t3h	csf1r	dnmt3a	eif2b1
lmpp_002	1.0261	2.3944	2.6847	1.6636	2.0203
lmpp_003	2.6496	1.7800	1.6821	1.5941	2.7736
lmpp_004	10.3080	0.5889	4.2653	-0.5565	0.0026
lmpp_007	0.5419	1.8631	10.8468	0.1757	1.0873
lmpp_008	0.9209	2.6637	2.8549	2.1965	2.3663

2. Initial Boolean model. A data frame with two columns, targets and update functions.

Note that if an update function contains both activation and inhibition genes, they must be expressed with a separate clause containing only activation genes, and a separate clause containing only inhibition genes. (See the update functions of Gata1 and Gata2 for examples)

Use `initialise_model` to convert the input Boolean model into a BoolModel object.

```
data(krum_bmodel)
head(krum_bmodel)

bmodel = initialise_model(krum_bmodel)
```

targets	factors
gata2	gata2 & ! ((gata1 & fog1) | sfpi1)
gata1	(gata1 | gata2 | fli1) & ! sfpi1
fog1	gata1
eklf	gata1 & ! fli1
fli1	gata1 & ! eklf
scl	gata1 & ! sfpi1

3. Initial state.

A single row data frame with genes as the columns. The expression state of each gene must be in binarised form, i.e. 0s and 1s.

Note that all the genes that are present in the initial Boolean model must also be present here.

```
data(krum_istate)
head(krum_istate)
```

	cjun	cebpa	fli1	gata1	gata2	eklf	sfpi1	gfi1	scl	egrnab	fog1
initial_state	0	1	0	0	1	0	1	0	0	0	0

4 Output format

BoolTraineR supports several output formats for Boolean models, as shown below.

- `outgraph_model` - Outputs a Boolean model in a tab-delimited file with each line being an edge (i.e. gene interaction). This function also outputs a node attribute file, which can be used to distinguish gene and AND nodes in a graph plotting software. This format is readable by both Cytoscape and Gephi.
- `outgenysis_model` - Outputs a Boolean model in a space-delimited file with each line being an edge (i.e. gene interaction). This format is readable by genYsis (used for steady state analysis).
- `writeBM` - Outputs a Boolean model in a comma-delimited file similar in format to the input file format (i.e. two columns: genes and update functions).

BoolTraineR can also output a state transition graph.

- `outstate_graph` - Outputs a state space of a Boolean model simulated with an initial state. This format is readable by both Cytoscape and Gephi.

5 Useful functions in BoolTraineR

Besides training Boolean models, BoolTraineR can be used for simulating a Boolean model asynchronously and calculate the score of a Boolean model with respect to a data.

- `model_train` - Core function in `BoolTraineR` that performs Boolean model inference.
- `simulate_model` - Simulate a Boolean model asynchronously using an initial state, and return its state space.
- `calc_mscore` - Calculate a distance score for a Boolean model with respect to an expression data.
- `model_dist` - Calculate the number of genes in the update functions that differ between two Boolean models.
- `model_setdiff` - Show the genes in the update functions that differ between two Boolean models.

6 Example workflows

Three example workflows will be discussed in this vignette: (1) Inferring model without an initial model, (2) Inferring model with an initial model, (3) Extending model with more genes. The two workflows are largely similar, which only differ in the data preparation step.

6.1 Inferring model without an initial model

This workflow is intended for use on inferring a Boolean model without an initial model.

When no initial model is used, BoolTraineR will reconstruct gene interactions from a list of user-specified genes. If the number of genes in the expression data is low (e.g. in qPCR), it is also possible to use all the genes in the expression data.

6.1.1 Full workflow

Full workflow is included here for easy referencing. Each step is discussed in further details below.

```
set.seed(0)  #use to ensure reproducibility. remove in actual use.

# (1) Setup paths and environment.
library(BoolTraineR)

# If intending to use parallel processing, uncomment the following lines.
# library(doParallel) num_core = 4 #specify the number of cores to be used.
# doParallel::registerDoParallel(cores=num_core)

# (2) Load data.
data(wilson_raw_data)  #load a data frame of expression data.
edata = wilson_raw_data

# (3) Filter cell types.
cell_ind = grepl("cmp", rownames(edata)) | grepl("gmp", rownames(edata)) | grepl("mep",
    rownames(edata))  #select cells to be included.
edata = edata[cell_ind, ]

# (4) Filter genes.
gene_ind = c("fli1", "gata1", "gata2", "gfi1", "scl", "sfpi1")  #select genes to be included.
edata = edata[, gene_ind]

# (5) Inferring Boolean model.
final_model = model_train(edata, max_varperrule = 4, verbose = T)

# (6) Visualise the Boolean model generated.
plotBM(final_model)
```

6.1.2 Initial setup

The first step is to load the `BoolTraineR` package. If you are intending to use parallel processing, you will also need to load the `doParallel` package. Then specify how many cores you intend to use using `registerDoParallel` from the `doParallel` package.

```
set.seed(0) #use to ensure reproducibility. remove in actual use.

#(1) Setup paths and environment.
library(BoolTraineR)

#If intending to use parallel processing, uncomment the following lines.
#library(doParallel)
```

```
#num_core = 4 #specify the number of cores to be used.
#doParallel::registerDoParallel(cores=num_core)
```

6.1.3 Data preparation

Only the expression data is needed for inferring a Boolean model without an initial model.

To load the data into R, use `read.table` or `read.csv`. In this example, we are using the example data included with the package, so we are accessing it by using `data`.

```
#(2) Load data.
data(wilson_raw_data) #load a data frame of expression data.
edata = wilson_raw_data
```

Once data is loaded, filter the cell types or genes to be included in the analysis if needed. It is advisable to reduce the number of genes to be included if the computation takes too long to complete.

```
# (3) Filter cell types.
cell_ind = grepl("cmp", rownames(edata)) | grepl("gmp", rownames(edata)) | grepl("mep",
    rownames(edata))  #select cells to be included.
edata = edata[cell_ind, ]

# (4) Filter genes.
gene_ind = c("fli1", "gata1", "gata2", "gfi1", "scl", "sfpi1")  #select genes to be included.
edata = edata[, gene_ind]
```

6.1.4 Run model training

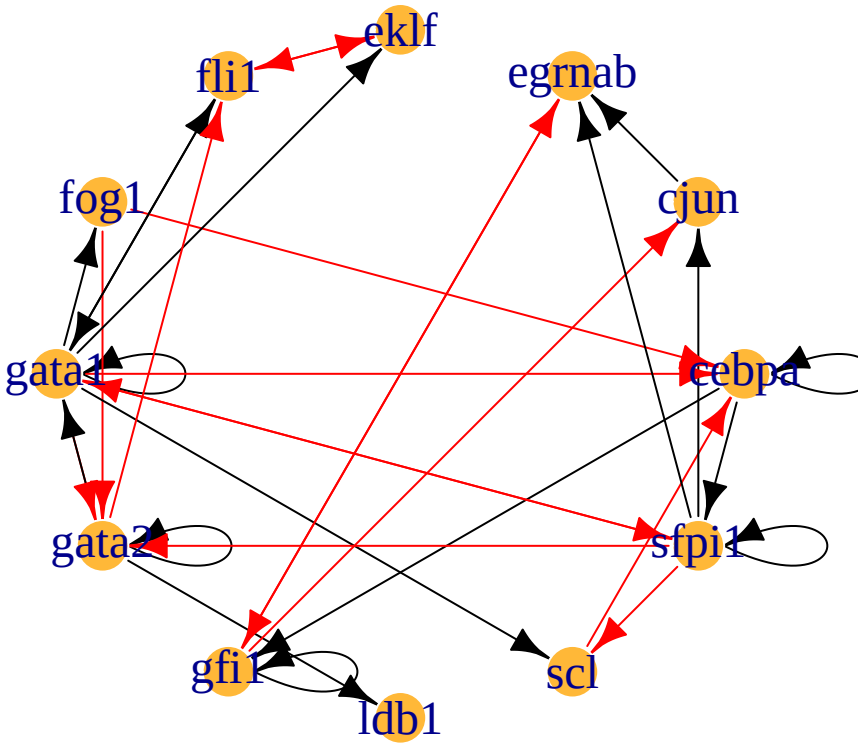
To reconstruct a Boolean model from an expression data, run `model_train`.

In this example, `model_train` takes a few seconds to be completed on a single core. If this steps take a very long time to complete, do consider using the parallel processing option as described above.

You will receive a BoolModel object at the end of the model training process. The BoolModel object can be visualise quickly using `plotBM`, which is based on `igraph` package. For easier manipulation, output the Boolean model using `outgraph_model` and display it with Cytoscape or Gephi.

```
#(5) Inferring Boolean model.
final_model = model_train(edata, max_varperrule=4, verbose=T)

#(6) Visualise the Boolean model generated.
plotBM(final_model)
```



6.2 Inferring model with an initial model

This workflow is intended for use on inferring a Boolean model with an initial model.

When an initial model is used, note that only genes that are both present in the initial model and expression data will be used for reconstructing gene interactions. Any genes in the initial model that do not have corresponding expression values in the data will keep their original gene interactions as specified in the initial model without any modifications.

6.2.1 Full workflow

Full workflow is included here for easy referencing. Each step is discussed in further details below.

```
set.seed(0)  #use to ensure reproducibility. remove in actual use.

# (1) Setup paths and environment.
```

```
library(BoolTraineR)

# If intending to use parallel processing, uncomment the following lines.
# library(doParallel) num_core = 4 #specify the number of cores to be used.
# doParallel::registerDoParallel(cores=num_core)

# (2) Load data.
data(krum_bmodel)  #load a data frame of Boolean model.
data(krum_istate)  #load a data frame of initial state.
data(wilson_raw_data)  #load a data frame of expression data.

bmodel = initialise_model(krum_bmodel)
istate = krum_istate
edata = wilson_raw_data

# (3) Filter cell types.
cell_ind = grepl("cmp", rownames(edata)) | grepl("gmp", rownames(edata)) | grepl("mep",
    rownames(edata))  #select cells to be included.
edata = edata[cell_ind, ]

# (4) Inferring Boolean model.
final_model = model_train(edata, bmodel, istate, max_varperrule = 4, verbose = T)

# (5) Visualise the Boolean model generated.
plotBM(final_model)
```

6.2.2 Initial setup

The first step is to load the `BoolTraineR` package. If you are intending to use parallel processing, you will also need to load the `doParallel` package. Then specify how many cores you intend to use using `registerDoParallel` from the `doParallel` package.

```
set.seed(0) #use to ensure reproducibility. remove in actual use.

#(1) Setup paths and environment.
library(BoolTraineR)

#If intending to use parallel processing, uncomment the following lines.
#library(doParallel)
#num_core = 4 #specify the number of cores to be used.
#doParallel::registerDoParallel(cores=num_core)
```

6.2.3 Data preparation

3 pieces of data are needed to infer a Boolean model with an initial model: an expression data, an initial Boolean model and an initial state.

To load the data into R, use `read.table` or `read.csv`. In this example, we are using the example data included with the package, so we are accessing it by using `data`.

`initialise_model` converts the data frame containing the Boolean model into a BoolModel object.

```
#(2) Load data.
data(krum_bmodel) #load a data frame of Boolean model.
data(krum_istate) #load a data frame of initial state.
data(wilson_raw_data) #load a data frame of expression data.

bmodel = initialise_model(krum_bmodel)
istate = krum_istate
edata = wilson_raw_data
```

Once data is loaded, filter the cell types or genes to be included in the analysis if needed. It is advisable to reduce the number of genes to be included if the computation takes too long to complete. In this example, genes are not filtered as all genes that are present in both expression data and Boolean model are used automatically.

```
# (3) Filter cell types.
cell_ind = grepl("cmp", rownames(edata)) | grepl("gmp", rownames(edata)) | grepl("mep",
    rownames(edata))  #select cells to be included.
edata = edata[cell_ind, ]
```

6.2.4 Run model training

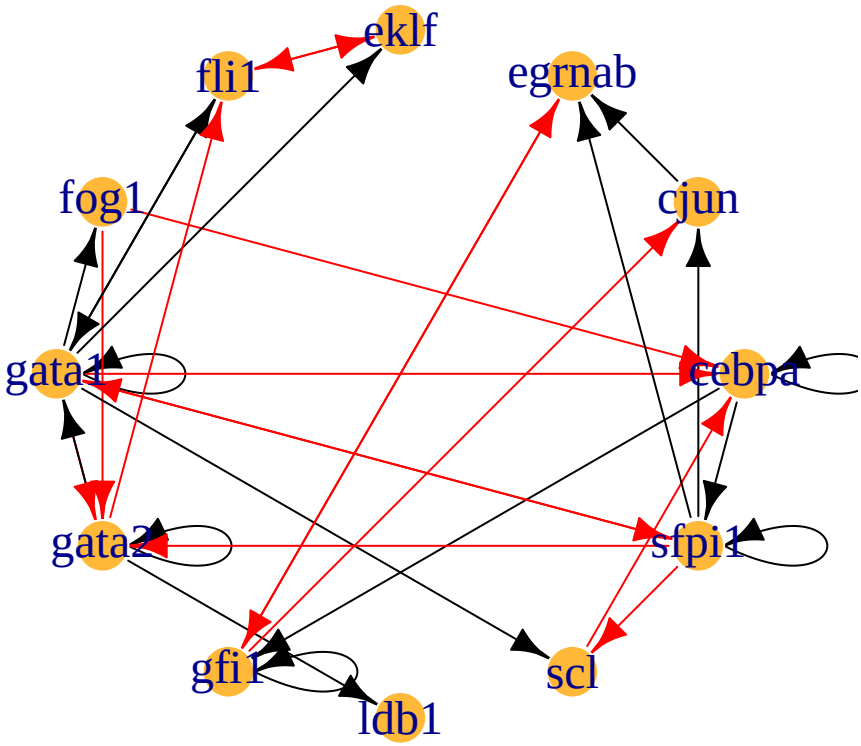
To reconstruct a Boolean model from an expression data, run `model_train`.

In this example, `model_train` takes a few seconds to be completed on a single core. If this steps take a very long time to complete, do consider using the parallel processing option as described above.

You will receive a BoolModel object at the end of the model training process. The BoolModel object can be visualise using `plotBM`, which is based on `igraph` package. For easier manipulation, output the Boolean model using `outgraph_model` and display it with Cytoscape or Gephi.

```
#(4) Inferring Boolean model.
final_model = model_train(edata, bmodel, istate, max_varperrule=4, verbose=T)

#(5) Visualise the Boolean model generated.
plotBM(final_model)
```



6.3 Extending model with more genes

This workflow is intended for use on extending an initial Boolean model with additional genes.

When an initial model is used, note that only genes that are both present in the initial model and expression data will be used for reconstructing gene interactions. Any genes in the initial model that do not have corresponding expression values in the data will keep their original gene interactions as specified in the initial model without any modifications.

6.3.1 Full workflow

Full workflow is included here for easy referencing. Each step is discussed in further details below.

Note that this example takes a few minutes to run on a single core. The use of parallel processing is recommended.

```r
set.seed(0)  #use to ensure reproducibility. remove in actual use.

# (1) Setup paths and environment.
library(BoolTraineR)

# If intending to use parallel processing, uncomment the following lines.
# library(doParallel) num_core = 4 #specify the number of cores to be used.
# doParallel::registerDoParallel(cores=num_core)

# (2) Load data.
data(krum_bmodel)  #load a data frame of Boolean model.
data(krum_istate)  #load a data frame of initial state.
data(wilson_raw_data)  #load a data frame of expression data.

bmodel = initialise_model(krum_bmodel)
istate = krum_istate
edata = wilson_raw_data

# (3) Filter cell types.
cell_ind = grepl("cmp", rownames(edata)) | grepl("gmp", rownames(edata)) | grepl("mep",
    rownames(edata))  #select cells to be included.
edata = edata[cell_ind, ]

# (4) Adding extra genes to the initial Boolean model. extra_genes =
# setdiff(colnames(wilson_raw_data), bmodel@target) #to view available genes
# to be added.
print(extra_genes)  #to view available genes to be added.
add_gene = "ldb1"  #genes to be added: ldb1
grown_bmodel = grow_bmodel(add_gene, bmodel)

# (5) Estimating initial state for the extra genes.
tmp_data = initialise_raw_data(wilson_raw_data)[[1]]  #preprocess data.
tmp_istate = mean(tmp_data[grepl("cmp", rownames(tmp_data)), add_gene])  #estimating initial state from
tmp_istate = matrix(round(tmp_istate), nrow = 1)
colnames(tmp_istate) = add_gene
grown_istate = cbind(istate, tmp_istate)
grown_istate = initialise_data(grown_istate)

# (6) Inferring Boolean model.
final_model = model_train(edata, grown_bmodel, grown_istate, max_varperrule = 4,
    verbose = T)

# (7) Visualise the Boolean model generated.
plotBM(final_model)
```

6.3.2 Initial setup

The first step is to load the `BoolTraineR` package. If you are intending to use parallel processing, you will also need to load the `doParallel` package. Then specify how many cores you intend to use using `registerDoParallel` from the `doParallel` package.

```
set.seed(0) #use to ensure reproducibility. remove in actual use.

#(1) Setup paths and environment.
library(BoolTraineR)

#If intending to use parallel processing, uncomment the following lines.
#library(doParallel)
#num_core = 4 #specify the number of cores to be used.
#doParallel::registerDoParallel(cores=num_core)
```

6.3.3 Data preparation

3 pieces of data are needed to infer a Boolean model with an initial model: an expression data, an initial Boolean model and an initial state.

To load the data into R, use `read.table` or `read.csv`. In this example, we are using the example data included with the package, so we are accessing it by using `data`.

`initialise_model` converts the data frame containing the Boolean model into a BoolModel object.

```
#(2) Load data.
data(krum_bmodel) #load a data frame of Boolean model.
data(krum_istate) #load a data frame of initial state.
data(wilson_raw_data) #load a data frame of expression data.

bmodel = initialise_model(krum_bmodel)
istate = krum_istate
edata = wilson_raw_data
```

Once data is loaded, filter the cell types or genes to be included in the analysis if needed. It is advisable to reduce the number of genes to be included if the computation takes too long to complete. In this example, genes are not filtered as all genes that are present in both expression data and Boolean model are used automatically.

```
# (3) Filter cell types.
cell_ind = grepl("cmp", rownames(edata)) | grepl("gmp", rownames(edata)) | grepl("mep",
    rownames(edata))  #select cells to be included.
edata = edata[cell_ind, ]
```

6.3.4 Add extra genes to the initial Boolean model

Extra genes can be added to the initial model using `grow_bmodel`. The function will add extra genes into the initial model with empty update functions.

```
#(4) Adding extra genes to the initial Boolean model.
#extra_genes = setdiff(colnames(wilson_raw_data), bmodel@target) #to view available genes to be added.
add_gene = 'ldb1' #genes to be added: ldb1
grown_bmodel = grow_bmodel(add_gene, bmodel)
```

6.3.5 Estimate initial state for the extra genes

Initial state needs to be modify to include the initial expression of the extra genes. The initial state of the extra genes can be set manually, or it can be estimated from the data if the data contain multiple cell types with known relationships. In this example, CMPs are known to be at developmental upstream of erythro-myeloid differentiation, therefore initial state can be estimated by taking the average expression of the extra genes in CMPs.

```
#(5) Estimating initial state for the extra genes.
tmp_data = initialise_raw_data(wilson_raw_data)[[1]] #preprocess data.
tmp_istate = mean(tmp_data[grepl('cmp', rownames(tmp_data)), add_gene]) #estimating initial state from
tmp_istate = matrix(round(tmp_istate), nrow=1)
colnames(tmp_istate) = add_gene
grown_istate = cbind(istate, tmp_istate)
grown_istate = initialise_data(grown_istate)
```

6.3.6 Run model training

To reconstruct a Boolean model from an expression data, run `model_train`.

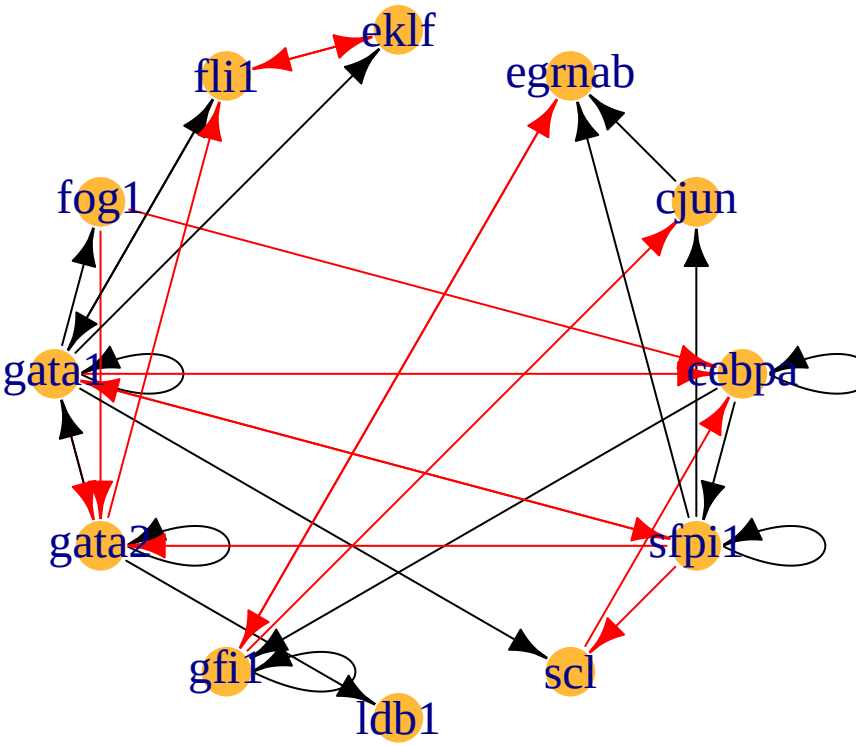
In this example, `model_train` takes a few minutes to be completed on a single core. If this steps take a very long time to complete, do consider using the parallel processing option as described above.

You will receive a BoolModel object at the end of the model training process. The BoolModel object can be visualise using `plotBM`, which is based on `igraph` package. For easier manipulation, output the Boolean model using `outgraph_model` and display it with Cytoscape or Gephi.

Note that this example takes a long time to run. The use of parallel processing is recommended.

```
#(6) Inferring Boolean model.
final_model = model_train(edata, grown_bmodel, grown_istate, max_varperrule=4, verbose=T)

#(7) Visualise the Boolean model generated.
plotBM(final_model)
```


eklf
fli1
egrnab
fog1
cjun
gata1
cebpa
gata2
sfpi1
gfi1
scl
ldb1