Causal Explorer User's Manual
Public domain version 1.5 for Matlab (R2017b)

Department of Biomedical Informatics,
Vanderbilt University,
2209 Garland Avenue
Nashville, TN 37232, USA

References for citation:

- CF Aliferis, I Tsamardinos, A Statnikov. "*Causal Explorer: A Probabilistic Network Learning Toolkit for Biomedical Discovery*." The 2003 International Conference on Mathematics and Engineering Techniques in Medicine and Biological Sciences (METMBS '03), June 23-26, 2003.

- A Statnikov, I Tsamardinos, CF Aliferis. "An Algorithm for Generation of Large Bayesian Networks." Technical report DSL TR-03-01, May 28, 2003, Vanderbilt University, Nashville, TN, USA.

Authors:

- Constantin F. Aliferis (constantin.aliferis@vanderbilt.edu)
- Ioannis Tsamardinos (ioannis.tsamardinos@vanderbilt.edu)
- Alexander Statnikov (alexander.statnikov@vanderbilt.edu)

Contributors:

- Laura Brown (laura.e.brown@vanderbilt.edu)
    - SCA, MMHC, and Greedy Search codes;
    - modifications to MMPC, MMMB, and TPDA codes.
- Nafeh Fananapazir (nafeh.fananapazir@vanderbilt.edu)
    - original discretization code

Causal Explorer for Matlab R2017b is located in "*Matlab_R2017b*" folder.

Table of Contents

I. Bayesian network learning algorithms toolbox (BNLAT)

I.1. Description

The initial version of this toolbox is described in the following paper:

CF Aliferis, I Tsamardinos, A Statnikov. "Causal Explorer: A Probabilistic Network Learning Toolkit for Biomedical Discovery." The 2003 International Conference on Mathematics and Engineering Techniques in Medicine and Biological Sciences (METMBS '03), June 23-26, 2003.

Available in the file *Causal_Explorer.pdf*

I.2. Matlab Interface

function *varargout*=Causal_Explorer(*algorithm*, *varargin*)

I.3. Inputs and Outputs

algorithm = String with the name of algorithm. Can take one of the following values:

- GS - Grow/Shrink algorithm
- IAMB - Incremental Association-Based Markov Blanket (IAMB)
- IAMBnPC - IAMB with PC algorithm in the pruning phase
- interIAMB - IAMB with interleaved pruning phase
- interIAMBnPC - IAMB with PC algorithm in the interleaved pruning phase
- KS - Koller-Sahami algorithm
- PC - PC algorithm
- TPDA - Three Phase Dependency Analysis (also known as BN PowerConstructor) algorithm
- LCD2 - LCD2 (Local Causal Discovery) algorithm
- SCA - Sparse Candidate Algorithm (SCA)
- MMHC - Min Max Hill Climbing (MMHC) or Greedy Search algorithm
- MMPC - Min Max Parents and Children (MMPC)
- MMMB - Min Max Markov Blanket (MMMB)
- HITON_PC - HITON_PC algorithm
- HITON_MB - HITON_MB algorithm

varagin = inputs for the selected algorithm (see below)
varagout = outputs for the selected algorithm (see below)

A. GS, IAMB, interIAMB, interIAMBnPC, and IAMBnPC algorithms

Inputs *(varargin)*:

1^{st} input = data
The data used for training (matrix). Columns are variables, rows are observations. Note, that for certain statistics, such as Mutual Information and G^2, data has to be in a special format: variable *i* has to take values 0..domain_counts(*i*). For example, if domain_counts(2)=3, this means that 2^{nd} variable takes values {0,1,2}.

2^{nd} input = target_variable_index
Index of the target variable. Our goal will be to find Markov blanket of this variable.

3^{rd} input = domain_counts
Some statistical tests operating on discrete data (such as Mutual Information and G^2) require a vector with the size of the corresponding domain (for all variables). E.g. domain_counts = [2 2 3]. This specifies that the domain of the first and the second variable is {0, 1}, and the domain of the third is {0, 1, 2}. The array domain_counts should be empty (i.e. []) if Fisher's Z-test is used (since variables are continuous).

4^{th} input = statistic
Statistical test desired to use. It can be either 'mi' (Mutual Information for discrete data), 'g2' (G^2 test for discrete data), or 'z' (Fisher's z-test for continuous data).

5^{th} input = threshold
Threshold on statistic (either Mutual Information or p-value). For Fisher's z-test and G^2 test, it is common to use 0.05 threshold. However, there is no universal threshold for Mutual Information, and it should be determined by validation.

Outputs (*varargout*):

1^{st} output = A vector with the indexes of variables in the Markov Blanket.

B. KS algorithm

This algorithm works only with discrete data.

Inputs *(varargin)*:

> 1^{st} input = data
> The data used for training (matrix). Columns are variables, rows are observations. Data has to be in a special format: variable *i* has to take values 0..domain_counts(*i*). For example, if domain_counts(2)=3, this means that 2^{nd} variable takes values {0,1,2}.
>
> 2^{nd} input = target_variable_index
> Index of the target variable. Our goal will be to find Markov blanket of this variable.
>
> 3^{rd} input = domain_counts
> A vector with the size of the corresponding domain (for all variables). E.g. domain_counts = [2 2 3]. This specifies that the domain of the first and the second variable is {0, 1}, and the domain of the third is {0, 1, 2}.
>
> 4^{th} input = Number of features to remove.
>
> 5^{th} input = Size of the Markov Blanket estimator.

Outputs *(varargout)*:

> 1^{st} output = Removed features (zeros in this array are removed features).
>
> 2^{nd} output = Order of removed features.

C. TPDA algorithm

Inputs *(varargin)*:

1st input = data
The data used for training (matrix). Columns are variables, rows are observations. Note, that for certain statistics, such as Mutual Information and G^2, data has to be in a special format: variable *i* has to take values 0..domain_counts(*i*). For example, if domain_counts(2)=3, this means that 2nd variable takes values {0,1,2}.

2nd input = domain_counts
Some statistical tests operating on discrete data (such as Mutual Information and G^2) require a vector with the size of the corresponding domain (for all variables). E.g. domain_counts = [2 2 3]. This specifies that the domain of the first and the second variable is {0, 1}, and the domain of the third is {0, 1, 2}. The array domain_counts should be empty (i.e. []) if Fisher's Z-test is used (since variables are continuous).

3rd input = statistic
Statistical test desired to use. It can be either 'mi' (Mutual Information for discrete data), 'g2' (G^2 test for discrete data), or 'z' (Fisher's z-test for continuous data).

4th input = threshold
Threshold on statistic (either Mutual Information or p-value). For Fisher's z-test and G^2 test, it is common to use 0.05 threshold. However, there is no universal threshold for Mutual Information, and it should be determined by validation.

5th input = Flag indicating whether to use one conditioning superset in EdgeNeeded_H and Edge_Needed* subroutines. If this flag=0, C=S_x and C=S_y are considered. If this flag=1, C=S_x U S_y is considered. Please see TPDA reference paper for more information.

6th input = Flag indicating if the data is monotone faithful (=1) or not (=0).

Outputs *(varargout)*:

1st output = Adjacency matrix. The algorithm constructs undirected graph and attempt to direct all edges. If the element in the ith row and jth column of adjacency matrix is equal to 1, this means that variable i is a parent of variable j. However, if element in ith row and jth column is equal to 2 (and element in jth row and ith column is also equal to 2), this means that the algorithm failed to direct that edge (i.e. the resulting edge between variables i and j is undirected).

D. PC algorithm

Inputs *(varargin)*:

1st input = data
The data used for training (matrix). Columns are variables, rows are observations. Note, that for certain statistics, such as Mutual Information and G^2, data has to be in a special format: variable *i* has to take values 0..domain_counts(*i*). For example, if domain_counts(2)=3, this means that 2nd variable takes values {0,1,2}.

2nd input = domain_counts
Some statistical tests operating on discrete data (such as Mutual Information and G^2) require a vector with the size of the corresponding domain (for all variables). E.g. domain_counts = [2 2 3]. This specifies that the domain of the first and the second variable is {0, 1}, and the domain of the third is {0, 1, 2}. The array domain_counts should be empty (i.e. []) if Fisher's Z-test is used (since variables are continuous).

3rd input = statistic
Statistical test desired to use. It can be either 'mi' (Mutual Information for discrete data), 'g2' (G^2 test for discrete data), or 'z' (Fisher's z-test for continuous data).

4th input = threshold
Threshold on statistic (either Mutual Information or p-value). For Fisher's z-test and G^2 test, it is common to use 0.05 threshold. However, there is no universal threshold for Mutual Information, and it should be determined by validation.

5th input = k
Maximum cardinality on number of direct causes and effects of a node. Set this number to be equal to -1 if you do not want to impose this constraint.

Outputs *(varargout)*:

1st output = Adjacency matrix. The algorithm constructs undirected graph and attempt to direct all edges. If the element in the ith row and jth column of adjacency matrix is equal to 1, this means that variable i is a parent of variable j. However, if element in ith row and jth column is equal to 2 (and element in jth row and ith column is also equal to 2), this means that the algorithm failed to direct that edge (i.e. the resulting edge between variables i and j is undirected).

E. LCD2 algorithm

Inputs *(varargin)*:

1st input = data
The data used for training (matrix). Columns are variables, rows are observations. Note, that for certain statistics, such as Mutual Information and G^2, data has to be in a special format: variable *i* has to take values 0..domain_counts(*i*). For example, if domain_counts(2)=3, this means that 2nd variable takes values {0,1,2}.

2nd input = target_variable_index
Index of the target variable. Our goal will be to find Markov blanket of this variable.

3rd input = domain_counts
Some statistical tests operating on discrete data (such as Mutual Information and G^2) require a vector with the size of the corresponding domain (for all variables). E.g. domain_counts = [2 2 3]. This specifies that the domain of the first and the second variable is {0, 1}, and the domain of the third is {0, 1, 2}. The array domain_counts should be empty (i.e. []) if Fisher's Z-test is used (since variables are continuous).

4th input = statistic
Statistical test desired to use. It can be either 'mi' (Mutual Information for discrete data), 'g2' (G^2 test for discrete data), or 'z' (Fisher's z-test for continuous data).

5th input = threshold
Threshold on statistic (either Mutual Information or p-value). For Fisher's z-test and G^2 test, it is common to use 0.05 threshold. However, there is no universal threshold for Mutual Information, and it should be determined by validation.

Outputs *(varargout)*:

1st output = Causal Relationships matrix. It is a 2-column matrix. The element in the first column is the index of the "cause" variable. The element in the second column is the index of the "effect" variable (corresponding to the "cause" variable in the first column).

F. SCA algorithm

This algorithm works only with discrete data.

Inputs *(varargin)*:

1st input = data
The data used for training (matrix). Columns are variables, rows are observations. Data has to be in a special format: variable *i* has to take values 0..domain_counts(*i*). For example, if domain_counts(2)=3, this means that 2nd variable takes values {0,1,2}.

2nd input = domain_counts
A vector with the size of the corresponding domain (for all variables). E.g. domain_counts = [2 2 3]. This specifies that the domain of the first and the second variable is {0, 1}, and the domain of the third is {0, 1, 2}.

3rd input = k
Number of candidates on first iteration (i.e., max fan-in). Default value can be either 5 or 10.

4th input = dw
Dirichlet Weight (default value is 10).

5th input = prior_type
Type of the priors. Default is 'BDeu'. Another option is uniform priors - 'unif'.

6th input = statistic
Statistical/Bayesian test desired to use. It can be either 'mi' (Mutual Information), or 'ms' (a Bayesian scoring metric).

Outputs *(varargout)*:

1st output = DAG
Best DAG found. DAG is represented via adjacency matrix. If the element in the ith row and jth column of adjacency matrix is equal to 1, this means that variable i is a parent of variable j.

2nd output = n_stats
Number of times a scoring function was called.

3rd output = score
Score of the best DAG

G. MMHC and Greedy Search algorithms

This algorithm works only with discrete data.

Inputs *(varargin)*:

1st input = data
The data used for training (matrix). Columns are variables, rows are observations. Data has to be in a special format: variable *i* has to take values 0..domain_counts(*i*). For example, if domain_counts(2)=3, this means that 2nd variable takes values {0,1,2}.

2nd input = domain_counts
A vector with the size of the corresponding domain (for all variables). E.g. domain_counts = [2 2 3]. This specifies that the domain of the first and the second variable is {0, 1}, and the domain of the third is {0, 1, 2}.

3rd input = cs_method
Method for choosing candidate parents, 'MMHC' or 'GreedySearch' (default = MMHC).

4th input = mmpc_options
Structure of options for MMPC, the candidate selection procedure used in MMHC,

- *options.threshold* = threshold on statistical test (default = 0.05)
- *options.epc* - elements per cell, MMPC attempts to find the maximum size of the conditioning set that is acceptable, where the available sample is more than epc for each cell of the conditional probability table (default = 5).
- *options.maxK* = The maximum size of the conditioning set allowed. This value is determined by MMPC using epc and the amount of sample, be a maximum value can also be set. Thus, MMPC's maximum conditioning set is min(maxK, k) where k is the max conditioning set calculated by MMPC (default = 10).
- *options.use_card_lim* = Flag is 1 if limited cardinality, and 0 otherwise (default = 0).
- *options.max_card* = Max number of variables to be added in the first phase of MMPC if the cardinality is limited (default = 0).

5th input = dw
Dirichlet weight (default = 10)

6th input = prior_type
Type of priors, 'unif' or 'BDeu'. (default = 'BDeu')

Outputs *(varargout)*:

1st output = DAG
Best DAG found. DAG is represented via adjacency matrix. If the element in the ith row and jth column of adjacency matrix is equal to 1, this means that variable i is a parent of variable j.

2nd output = dag_score
Score of the best DAG found.

3rd output = num_stats
Number of statistics called in MMPC if used as a candidate parent selection method.

4th output = cp_time
Time to complete the candidate selection method.

5th output = cps
A matrix of the candidate parents selected. The dimensions of cps are [number of variables x number of variables]. If cps(i,j)=1, then i is considered for a parent of j (similarly, if cps(i,j)=0 then i is not considered to be a parent of j). Note, for plain Greedy Hill Climbing Search then this matrix is all 1s.

H. MMPC and MMMB algorithms

This algorithm works only with discrete data.

Inputs *(varargin)*:

1st input = data
The data used for training (matrix). Columns are variables, rows are observations. Data has to be in a special format: variable *i* has to take values 0..domain_counts(*i*). For example, if domain_counts(2)=3, this means that 2nd variable takes values {0,1,2}.

2nd input = target_variable_index
Index of the target variable. Our goal will be to find Markov blanket of this variable.

3rd input = domain_counts
A vector with the size of the corresponding domain (for all variables). E.g. domain_counts = [2 2 3]. This specifies that the domain of the first and the second variable is {0, 1}, and the domain of the third is {0, 1, 2}.

4th input = method
Type of MMPC algorithm (for choosing parents and children): 'MMPC' for regular MMPC, and 'PMMPC' for polynomial MMPC.

5th input = mmpc_options
Structure of options for MMPC, the candidate selection procedure used in MMHC,

- *options.threshold* = threshold on statistical test (default = 0.05)
- *options.epc* - elements per cell, MMPC attempts to find the maximum size of the conditioning set that is acceptable, where the available sample is more than epc for each cell of the conditional probability table (default = 5).
- *options.maxK* = The maximum size of the conditioning set allowed. This value is determined by MMPC using epc and the amount of sample, be a maximum value can also be set. Thus, MMPC's maximum conditioning set is min(maxK, k) where k is the max conditioning set calculated by MMPC (default = 10).
- *options.use_card_lim* = Flag is 1 if limited cardinality, and 0 otherwise (default = 0).
- *options.max_card* = Max number of variables to be added in the first phase of MMPC if the cardinality is limited (default = 0).

Outputs *(varargout)*:

1st output = A vector with the indexes of variables in the set of parents and children (for MMPC) and Markov blanket (for MMMB).

J. HITON_PC and HITON_MB algorithms

Inputs *(varargin)*:

1st input = data
The data used for training (matrix). Columns are variables, rows are observations. Note, that for certain statistics, such as Mutual Information and G^2, data has to be in a special format: variable *i* has to take values 0..domain_counts(*i*). For example, if domain_counts(2)=3, this means that 2nd variable takes values {0,1,2}.

2nd input = target_variable_index
Index of the target variable. Our goal will be to find Markov blanket of this variable.

3rd input = domain_counts
Some statistical tests operating on discrete data (such as Mutual Information and G^2) require a vector with the size of the corresponding domain (for all variables). E.g. domain_counts = [2 2 3]. This specifies that the domain of the first and the second variable is {0, 1}, and the domain of the third is {0, 1, 2}. The array domain_counts should be empty (i.e. []) if Fisher's Z-test is used (since variables are continuous).

4th input = statistic
Statistical test desired to use. It can be either 'g2' (G^2 test for discrete data) or 'z' (Fisher's z-test for continuous data). HITON_PC and HITON_MB do not currently support mutual information.

5th input = threshold
Threshold on statistic (either Mutual Information or p-value). For Fisher's z-test and G^2 test, it is common to use 0.05 threshold. However, there is no universal threshold for Mutual Information, and it should be determined by validation.

6th input = The maximum size of the conditioning set allowed.

Outputs *(varargout)*:

1st output = A vector with the indexes of variables in the set of parents and children (for HITON_PC) and Markov blanket (for HITON_MB).

Note on application of HITON_PC and HITON_MB to continuous data:

Unlike other algorithms which require all variables of the dataset to be continuous, HITON_PC and HITON_MB require that the target variable is discrete and takes consecutive integer values starting from 0 (i.e. 0, 1, 2, …)

I.4. Examples

A. Examples for discrete data

In order to work with examples below, please load ALARM dataset into variable "data" and domain counts into variable "domain_counts":

load ../Data/alarm_h;
load ../Data/alarm_h_dc;

GS algorithm:

mb=Causal_Explorer('GS', data, 3, domain_counts, 'mi', 0.02)

mb=Causal_Explorer('GS', data, 5, domain_counts, 'g2', 0.05)

IAMB algorithm:

mb=Causal_Explorer('IAMB', data, 4, domain_counts, 'mi', 0.04)

mb=Causal_Explorer('IAMB', data, 2, domain_counts, 'g2', 0.05)

interIAMB algorithm:

mb=Causal_Explorer('interIAMB', data, 2, domain_counts, 'mi', 0.01)

mb=Causal_Explorer('interIAMB', data, 2, domain_counts, 'g2', 0.05)

interIAMBnPC algorithm:

mb=Causal_Explorer('interIAMBnPC', data, 4, domain_counts, 'mi', 0.05)

mb=Causal_Explorer('interIAMBnPC', data, 2, domain_counts, 'g2', 0.05)

IAMBnPC algorithm:

mb=Causal_Explorer('IAMBnPC', data, 3, domain_counts, 'mi', 0.01)

mb=Causal_Explorer('IAMBnPC', data, 2, domain_counts, 'g2', 0.05)

KS algorithm:

[features, order]=Causal_Explorer('KS', data, 3, domain_counts, 34, 2)

TPDA algorithm:

A=Causal_Explorer('TPDA', data, domain_counts, 'mi', 0.01, 0, 1)

A=Causal_Explorer('TPDA', data, domain_counts, 'mi', 0.01, 0, 0)

A=Causal_Explorer('TPDA', data, domain_counts, 'mi', 0.01, 1, 0)

A=Causal_Explorer('TPDA', data, domain_counts, 'mi', 0.01, 1, 1)

A=Causal_Explorer('TPDA', data, domain_counts, 'g2', 0.01, 0, 1)

A=Causal_Explorer('TPDA', data, domain_counts, 'g2', 0.01, 0, 0)

A=Causal_Explorer('TPDA', data, domain_counts, 'g2', 0.01, 1, 0)

A=Causal_Explorer('TPDA', data, domain_counts, 'g2', 0.01, 1, 1)

PC algorithm:

A=Causal_Explorer('PC', data, domain_counts, 'mi', 0.01, 2)

A=Causal_Explorer('PC', data, domain_counts, 'mi', 0.01, -1)

A=Causal_Explorer('PC', data, domain_counts, 'g2', 0.05, 2)

A=Causal_Explorer('PC', data, domain_counts, 'g2', 0.05, -1)

LCD2 algorithm:

CR=Causal_Explorer('LCD2', data, 1, domain_counts, 'mi', 0.01)

CR=Causal_Explorer('LCD2', data, 1, domain_counts, 'g2', 0.05)

SCA algorithm:

[A,stats,bs]=Causal_Explorer('SCA', data, domain_counts, 5, 10, 'BDeu', 'ms')

[A,stats,bs]=Causal_Explorer('SCA', data, domain_counts, 5, 10, 'BDeu', 'mi')

[A,stats,bs]=Causal_Explorer('SCA', data, domain_counts, 5, 10, 'unif', 'ms')

[A,stats,bs]=Causal_Explorer('SCA', data, domain_counts, 5, 10, 'unif', 'mi')

MMHC algorithm:

```
[A,score,stats,cp_time,cps] = Causal_Explorer('MMHC',data,domain_counts,'MMHC',[],10,'BDeu')
```

```
options.threshold = 0.05;
options.epc = 10;
options.maxK = 10;
options.use_card_lim = 0;
options.max_card = 0;
[A,score,stats,time,cps] = Causal_Explorer('MMHC',data,domain_counts,'MMHC',options,10,'BDeu');
```

Greedy Search algorithm:

```
[A,score,stats,time,cps] = Causal_Explorer('MMHC',data,domain_counts,'GreedySearch',[],10,'BDeu');
```

MMPC algorithm:

```
[pc, stats] = Causal_Explorer('MMPC', data, 3, domain_counts, 'MMPC', [])

[pc, stats] = Causal_Explorer('MMPC', data, 4, domain_counts, 'PMMPC', [])

options.threshold = 0.05;
options.epc = 5;
options.maxK = 5;
options.use_card_lim = 0;
options.max_card = 0;
[pc, stats] = Causal_Explorer('MMPC', data, 3, domain_counts, 'MMPC', options)

options.threshold = 0.05;
options.epc = 7;
options.maxK = 3;
options.use_card_lim = 0;
options.max_card = 0;
[pc, stats] = Causal_Explorer('MMPC', data, 4, domain_counts, 'PMMPC', options)
```

MMMB algorithm:

```
[mb, pc, pc_pc] = Causal_Explorer('MMMB', data, 5, domain_counts, 'MMPC', [])

[mb, pc, pc_pc] = Causal_Explorer('MMMB', data, 5, domain_counts, 'PMMPC', [])

options.threshold = 0.05;
options.epc = 5;
options.maxK = 5;
options.use_card_lim = 0;
options.max_card = 0;
[mb, pc, pc_pc] = Causal_Explorer('MMMB', data, 5, domain_counts, 'MMPC', options)
```

options.threshold = 0.05;
options.epc = 5;
options.maxK = 5;
options.use_card_lim = 0;
options.max_card = 0;
[mb, pc, pc_pc] = Causal_Explorer('MMMB', data, 5, domain_counts, 'PMMPC', options)

HITON_PC algorithm:

pc=Causal_Explorer('HITON_PC', data, 4, domain_counts, 'g2', 0.05, 3)

HITON_MB algorithm:

mb=Causal_Explorer('HITON_MB', data, 4, domain_counts, 'g2', 0.05, 3)

B. Examples for continuous data

In order to work with examples below, please load a continuous dataset:

load ../Data/random_data_1;

GS algorithm:

mb=Causal_Explorer('GS', data, 3, [], 'z', 0.05)

IAMB algorithm:

mb=Causal_Explorer('IAMB', data, 3, [], 'z', 0.05)

interIAMB algorithm:

mb=Causal_Explorer('interIAMB', data, 3, [], 'z', 0.05)

IAMBnPC algorithm:

mb=Causal_Explorer('IAMBnPC', data, 3, [], 'z', 0.05)

interIAMBnPC algorithm:

mb=Causal_Explorer('interIAMBnPC', data, 4, [], 'z', 0.05)

TPDA algorithm:

A=Causal_Explorer('TPDA', data, [], 'z', 0.05, 1, 1)

PC algorithm:

A=Causal_Explorer('PC', data, [], 'z', 0.05, 1, 1)

LCD2 algorithm:

CR=Causal_Explorer('LCD2', data, 11, [], 'z', 0.05)

In order to work with examples below, please load a continuous dataset with discrete target and a target variable index:

load ../Data/random_data_2;

HITON_PC algorithm:

pc=Causal_Explorer('HITON_PC', data, target_variable_index, [], 'z', 0.05, 3)

HITON_MB algorithm:

mb=Causal_Explorer('HITON_MB', data, target_variable_index, [], 'z', 0.05, 3)

II. Bayesian network tiling tool (BNTT)

II.1. Description

Please see the following technical report:

A Statnikov, I Tsamardinos, CF Aliferis. "An Algorithm for Generation of Large Bayesian Networks." Technical report DSL TR-03-01, May 28, 2003, Vanderbilt University, Nashville, TN, USA

Available in the file *BN_Tiling.pdf*

II.2. Matlab Interface

nodes=bn_tiling(*network_filename*, *dataset_filename, num_variables, k*)

II.3. Inputs and Outputs

Inputs:

> 1st input = network_filename
> Cell array of names of network files in HUGIN format, where network_filename{i} is a name of network file in HUGIN format. The network filename should include a path and should be specified with the extension.
>
> 2nd input = dataset_filename
> Cell array of names of dataset files for networks specified in cell array "network_filename" (in the same order), where dataset_filename{i} is a name of dataset in Matlab format (.mat) corresponding to network network_filename{i}. These datasets will be used for estimation of joint probabilities. The dataset file should contain an array "data" (where rows are observations/samples and columns are variables of the original network). Each column (variable) in "data" should take discrete values {0,1,...,domain counts of this variable} (where "domain counts" is the number of unique values this variable can take). It is recommended to specify a dataset with a large number of observations (typically, 10,000 is enough). In order to generate this file, one can simulate cases of the network network_filename{i} using HUGIN (without missing values), and use utility in the file *data_converter_hugin.dll* to convert generated data file into Matlab array "data" as described above. In the later versions of Bayesian Network Tiling Tool there will be no need to specify datasets, since an inference algorithm will be implemented in the software.

3rd input = num_variables
Desired maximum number of variables in the output network.

4th input = k
Connectivity parameter (see technical report for details).

Outputs:

1st output = nodes
Resulting network in a cell array. Each cell of this array is a data-structure corresponding to a variable. The number of variables is approximately "num_variables" (since we include only full tiles). If two or more original networks are specified in "network_filename", the resulting network will contain the same number of variables from different original networks. For example, nodes{i} corresponding to the ith variable in the resulting tiled network contains:

- *nodes{i}.name* = a name of the original variable (parsed from HUGIN network file)
- *nodes{i}.parents* = pointers to the parents (variable indices in the new network)
- *nodes{i}.cpt* = conditional probability table (cpt) as an n-dimensional array, where n is number of parents + 1.
  The semantics are: P(A=a | B=b C=c D=d ..) = cpt(1, 1, 1, 1, ..) when a, b, c, and d are the first values in order in the domains of the variables.

II.3. Examples

A. Generate a tiled ALARM network consisting of approximately 2,000 variables with connectivity parameter = 2

```
network_filename={'../Data/alarm_h.net'};
dataset_filename={'../Data/alarm_h.mat'};
nodes=bn_tiling(network_filename, dataset_filename, 2000, 2);
```

B. Generate a tiled ALARM and HAILFINDER network consisting of approximately 1,000 variables with connectivity parameter = 3

```
network_filename={'../Data/alarm_h.net', '../Data/hailfinder_h.net'};
dataset_filename={'../Data/alarm_h.mat', '../Data/hailfinder_h.mat'};
nodes=bn_tiling(network_filename, dataset_filename, 1000, 3);
```

III. Data converter from HUGIN format to BNTT

III.1. Matlab Interface

data=data_converter_hugin(*data_file, network_file*)

III.2. Inputs and Outputs

Inputs:

1st input = data_file
Filename with simulated data in HUGIN format

2nd input = network_file
Filename with original network in HUGIN format

Outputs:

1st output = data
Data array in Matlab format. The rows are observations/samples and columns are variables of the original network. Each column (variable) in "data" takes discrete values {0,1,...,domain counts of this variable} (where "domain counts" is the number of unique values this variable can take).

III.3. Example

data=data_converter_hugin('../Data/alarm_h.dat', '../Data/alarm_h.net');

IV. Utility to simulate data from a Bayesian network and generate an adjacency matrix

IV.1. Matlab Interface

[*data, graph*]=simulate_data*(nodes, num_cases)*

IV.2. Inputs and Outputs

Inputs:

1st input = nodes
Network in a cell array. Each cell of this array is a data-structure corresponding to a variable. For example, nodes{i} corresponds to the ith variable of the network and contains:
- *nodes{i}.name* = a name of the variable
- *nodes{i}.parents* = pointers to the parents (variable indices in the network)
- *nodes{i}.cpt* = conditional probability table (cpt) as an n-dimensional array, where n is number of parents + 1.
  The semantics are: P(A=a | B=b C=c D=d ..) = cpt(1, 1, 1, 1, ..) when a, b, c, and d are the first values in order in the domains of the variables.

Please see example below for more details on this data structure.

2nd input = num_cases
Number of cases to be simulated.

Outputs:

1st output = data
Data array in Matlab format. The rows are observations/samples and columns are variables of the original network. Each column (variable) in "data" takes discrete values {0,1,...,domain counts of this variable} (where "domain counts" is the number of unique values this variable can take).

2nd output = graph
Adjacency matrix. If the element in the ith row and jth column of adjacency matrix is equal to 1, this means that variable i is a parent of variable j.

IV.3. Example

load ../Data/nodes.mat
[data, graph]=simulate_data(nodes, 1000);

V. Supervised discretization of continuous data

V.1. Description

The discretization routine performs the following steps:

- Data is normalized so that each variable has mean 0 and standard deviation 1.

- After normalization, association of each variable with the target is computed using either Wilcoxon rank sum test (for binary target) or Kruskal-Wallis ANOVA (for multicategory target) with 0.05 alpha level.

- If a variable is not significantly associated with the target, it is discretized as follows:
  - *0* for values less then -1 standard deviation
  - *1* for values between -1 and 1 standard deviation
  - *2* for values greater than 1 standard deviation

- If a variable is significantly associated with the target, it is discretized using sliding threshold (into binary) or using sliding window (into ternary). The discretization threshold(s) is determined by a Chi-squared test.

The discretization routine uses information only from the training samples.

For a general idea, refer to Mitchell's book "Machine Learning", 1997, pages 72-73.

V.2. Matlab Interface

data_d = discretization(*data, target_variable_index, training_samples_index, variables_index*)

V.3. Inputs and Outputs

Inputs:

1st input = data
Continuous input dataset in the form of a matrix. Rows should correspond to observations/samples and columns should correspond to variables. The column corresponding to target (i.e. response variable) should take consecutive integer values starting from 1 (i.e. 0, 1, 2,…).

2nd input = target_variable_index
Index of the target column (i.e. response variable). This column should take consecutive integer values starting from 1 (i.e. 0, 1, 2,…).

3rd input = training_samples_index
Vector with indices of rows (samples) that will be used for training.

4th input = variables_index
Vector with indices of columns (variables) that will be discretized.

Outputs:

1st output = data_d
Discrete dataset in the form of a matrix.

V.4. Example

In order to work with example below, please load a continuous dataset with discrete target and a target variable index:

load ../Data/random_data_2;

Define training samples and variables to be discretized:

training_samples_index=1:700; % use first 700 samples for training;
variables_index=setdiff(1:size(data,2), target_variable_index); % all variables but the target

Run discretization:

data_d = discretization(data, target_variable_index, training_samples_index, variables_index);

VI. References

TPDA algorithm:

- Cheng, J., Greiner, R., Kelly, J., Bell, DA and Liu, W., Learning Bayesian Networks from Data: an Information-Theory Based Approach, The Artificial Intelligence Journal, Volume 137, Pages 43-90, 2002

GS algorithm:

- D. Margaritis and S. Thrun. Bayesian Network Induction via Local Neighborhoods. Advances in Neural Information Processing Systems 12 (NIPS), Denver, Colorado. December 1999.

KS algorithm:

- Koller, D., and Sahami, M. (1996). Toward Optimal Feature Selection. In: Machine Learning: Proceedings of the Thirteenth International Conference. Morgan Kaufmann.

PC algorithm:

- Peter Spirtes, Clark Glymour, and Richard Scheines. Causation, Prediction and Search (second edition). The MIT Press, 2000.

LCD2 algorithm:

- Subramani Mani, Gregory F. Cooper. A Study in Causal Discovery from Population-Based Infant Birth and Death Records. Proceedings of the AMIA Annual Fall Symposium, 1999, p315--319. Hanley and Belfus Publishers, Philadelphia, PA.

IAMB, interIAMB, interIAMBnPC, IAMBnPC, pchIAMB algorithms:

- Tsamardinos I., C.F. Aliferis, A. Statnikov. Algorithms for Large Scale Markov Blanket Discovery," in Proceedings of the 16th International FLAIRS Conference (FLAIRS 2003), 2003

SCA algorithm:

- Nir Friedman, Iftach Nachman and Dana Pe'er. Learning Bayesian Network Structure from Massive Datasets: The ``Sparse Candidate'' Algorithm. Fifteenth Conference on Uncertainty in Artificial Intelligence, 1999.

MMHC algorithm:

- L.E. Brown, I. Tsamardinos, C.F. Aliferis. "A Novel Algorithm for Scalable and Accurate Bayesian Network Learning" In Proceedings of the 11th World Congress on Medical Informatics (MEDINFO), September 7-11, 2004, San Francisco, California, USA

MMPC and MMMB algorithms:

- I. Tsamardinos, C.F. Aliferis, A. Statnikov. "Time and Sample Efficient Discovery of Markov Blankets and Direct Causal Relations" In Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, August 24-27, 2003, Washington, DC, USA, ACM Press, pages 673-678

- L.E. Brown, I. Tsamardinos, C.F. Aliferis. "A Comparison of Novel and State-of-the-Art Polynomial Bayesian Network Learning Algorithms" In Proceedings of the Twentieth National Conference on Artificial Intelligence (AAAI), 2005

Greedy Search algorithm:

- L.E. Brown, I. Tsamardinos, C.F. Aliferis. "A Comparison of Novel and State-of-the-Art Polynomial Bayesian Network Learning Algorithms" In Proceedings of the Twentieth National Conference on Artificial Intelligence (AAAI), 2005

HITON_PC and HITON_MB algorithms:

- C. F. Aliferis, I. Tsamardinos, A. Statnikov. "HITON, A Novel Markov Blanket Algorithm for Optimal Variable Selection" In Proceedings of the 2003 American Medical Informatics Association (AMIA) Annual Symposium, November 8-12, 2003, Washington, DC, USA, pages 21-25.