causal-learn

Release 0.1

CLeaR

Nov 23, 2021

CONTENTS

causal-learn is a Python translation and extension of the Tetrad java code. It offers the implementations of up-to-date causal discovery methods as well as *simple* and *intuitive* APIs.

Note: This project is under active development. For source code, please kindly refer to our GitHub Repository.

CHAPTER

ONE

CONTENTS

1.1 Getting started

1.1.1 Installation

Requirements

- python 3
- numpy
- networkx
- pandas
- scipy
- scikit-learn
- statsmodels
- pydot

(For visualization)

- matplotlib
- graphviz
- pygraphviz (might not support the most recent Mac)

Install via PyPI

To use causal-learn, we could install it using pip:

```
(.venv) $ pip install causal-learn
```

Install from source

For development version, please kindly refer to our GitHub Repository.

1.1.2 Running examples

For search methods in causal discovery, there are various running examples in the 'tests' directory in our GitHub Repository, such as TestPC.py and TestGES.py.

For the implemented modules, such as (conditional) independent test methods, we provide unit tests for the convenience of developing your own methods.

1.1.3 Quick benchmarking

To help users get a quick sense of the running time of the algorithms of interest, we conducted benchmarking for several methods. We consider datasets with number of variables from {10, 25, 50, 100} and average degree from {2, 3, 4, 5}. The random graphs are Erdős-Rényi graphs. The average degree is the average number of edges connected to a node. We simply calculate it by dividing the sum of degrees by the total number of nodes in the graph. The sample size is 1000. All algorithms were run on a single cluster node with 8 CPUs (Intel Xeon E5-2470) and 16 GB Memory. We denote running time as '>D' if it is more than one day. All results are average over 10 runs with different random seeds.

Running time (in seconds) for PC (with Fisher-z test)

Ave. Degree \ # of Vars	10	25	50	100
2	0.4	4.5	17.6	58.7
3	0.7	5.9	35.2	79.9
4	1.2	12.4	62.3	122.7
5	2	18.4	94.5	217.8

Running time (in seconds) for FCI (with Fisher-z test)

Ave. Degree \ # of Vars	10	25
2	1.9	18968.5
3	4.55	>D
4	7.65	>D
5	7.24	>D

Running time (in seconds) for GES (with BIC score)

Ave. Degree \ # of Vars	10	25	50
2	5.3	170.4	2384.8
3	7.1	296.1	5534.5
4	9.1	2392.6	9060.9
5	13.5	1368.7	16323.5

1.1.4 Contributors

Team Leaders: Kun Zhang, Joseph Ramsey, Mingming Gong, Ruichu Cai, Shohei Shimizu, Peter Spirtes, Clark Glymour

Coordinators: Biwei Huang, Yujia Zheng, Wei Chen

Developers:

- Wei Chen, Biwei Huang, Yuequn Liu, Zhiyi Huang, Feng Xie: *PC*, *FCI*, *GES*, *GIN*, and *graph operations*.
- Mingming Gong, Erdun Gao: *PNL*, *ANM*, *Granger causality*, and *KCI*.
- Shohei Shimizu, Takashi Nicholas Maeda, Takashi Ikeuchi: *LiNGAM-based methods*.
- Madelyn Glymour: several helpers.
- Ruibo Tu: *Missing-value/test-wise deletion PC*.
- Wai-Yin Lam: *PC*.
- Biwei Huang: *CD-NOD*.
- Ignavier Ng, Yujia Zheng: *Exact search*.
- Joseph Ramsey, Wei Chen, Zhiyi Huang: *Evaluations*.

1.2 Search methods

In this section, we would like to introduce search methods for causal discovery in causal-learn.

Contents:

1.2.1 Constrained-based causal discovery methods

In this section, we would like to introduce constrained-based causal discovery methods, including PC[1], FCI[2], and CD-NOD[3].

Contents:

PC

Algorithm Introduction

Perform Peter-Clark (PC[1]) algorithm for causal discovery. We also allowed data sets with missing values, for which testwise-deletion PC is included (choosing 'MV-Fisher_Z" for the test name).

If you would like to use missing-value PC[2], please set 'mvpc' as True.

[1] Spirtes, P., Glymour, C. N., Scheines, R., & Heckerman, D. (2000). Causation, prediction, and search. MIT press.

[2] Spirtes, P., Meek, C., & Richardson, T. (1995, August). Causal inference in the presence of latent variables and selection bias. In Proceedings of the Eleventh conference on Uncertainty in artificial intelligence (pp. 499-506).

[3] Huang, B., Zhang, K., Zhang, J., Ramsey, J. D., Sanchez-Romero, R., Glymour, C., & Schölkopf, B. (2020). Causal Discovery from Heterogeneous/Nonstationary Data. J. Mach. Learn. Res., 21(89), 1-53.

[1] Spirtes, P., Glymour, C. N., Scheines, R., & Heckerman, D. (2000). Causation, prediction, and search. MIT press.

[2] Tu, R., Zhang, C., Ackermann, P., Mohan, K., Kjellström, H., & Zhang, K. (2019, April). Causal discovery in the presence of missing data. In The 22nd International Conference on Artificial Intelligence and Statistics (pp. 1762-1770). PMLR.

Usage

```
from causallearn.search.ConstraintBased.PC import pc
G = pc(data, alpha, indep_test, stable, uc_rule, uc_priority, mvpc, correction_name,␣
↪background_knowledge)

# visualization using pydot
cg.draw_pydot_graph()

# visualization using networkx
# cg.to_nx_graph()
# cg.draw_nx_graph(skel=False)
```

Parameters

data: numpy.ndarray, shape (n_samples, n_features). Data, where n_samples is the number of samples and n_features is the number of features.

alpha: desired significance level (float) in (0, 1).

indep_test: Independence test method function.

- "*fisherz*": Fisher's Z conditional independence test.
- "*chisq*": Chi-squared conditional independence test.
- "*gsq*": G-squared conditional independence test.
- "*kci*": kernel-based conditional independence test. (As a kernel method, its complexity is cubic in the sample size, so it might be slow if the same size is not small.)
- "*mv_fisherz*": Missing-value Fisher's Z conditional independence test.

stable: run stabilized skeleton discovery if True (default = True).

uc_rule: how unshielded colliders are oriented.

- 0: run uc_sepset.
- 1: run maxP. Orient an unshielded triple X-Y-Z as a collider with an aditional CI test.
- 2: run definiteMaxP. Orient only the definite colliders in the skeleton and keep track of all the definite non-colliders as well.

uc_priority: rule of resolving conflicts between unshielded colliders.

- -1: whatever is default in uc_rule.
- 0: overwrite.
- 1: orient bi-directed.
- 2: prioritize existing colliders.
- 3: prioritize stronger colliders.
- 4: prioritize stronger* colliders.

mvpc: use missing-value PC or not. Default: False.

correction_name. Missing value correction if using missing-value PC. Default: 'MV_Crtn_Fisher_Z'

background_knowledge: class BackgroundKnowledge. Add prior edges according to assigned causal connections. For detailed usage, please kindly refer to its usage example.

Returns

cg : a CausalGraph object. Nodes in the graph correspond to the column indices in the data.

FCI

Algorithm Introduction

Causal Discovery with Fast Causal Inference (FCI[1]).

Usage

```
from causallearn.search.ConstraintBased.FCI import fci
G = fci(data, indep_test, alpha, verbose=True)
```

Parameters

data: numpy.ndarray, shape (n_samples, n_features). Data, where n_samples is the number of samples and n_features is the number of features.

alpha: Significance level of individual partial correlation tests.

indep_test: Independence test method function.

- "*fisherz*": Fisher's Z conditional independence test.
- "*chisq*": Chi-squared conditional independence test.
- "*gsq*": G-squared conditional independence test.
- "*kci*": kernel-based conditional independence test. (As a kernel method, its complexity is cubic in the sample size, so it might be slow if the same size is not small.)
- "*mv_fisherz*": Missing-value Fisher's Z conditional independence test.

verbose: 0 - no output, 1 - detailed output.

Returns

G : a GeneralGraph object. Nodes in the graph correspond to the column indices in the data. For visualization, please refer to the running example.

[1] Spirtes, P., Meek, C., & Richardson, T. (1995, August). Causal inference in the presence of latent variables and selection bias. In Proceedings of the Eleventh conference on Uncertainty in artificial intelligence (pp. 499-506).

CD-NOD

Algorithm Introduction

Perform Peter-Clark algorithm for causal discovery on the augmented data set that captures the unobserved changing factors (CD-NOD,[1]).

Usage

```
from causallearn.search.ConstraintBased.CDNOD import cdnod
G = cdnod(data, c_indx, alpha, indep_test, stable, uc_rule, uc_priority, mvpc,␣
↪correction_name)
G.to_nx_graph()
G.draw_nx_graph(skel=False)
```

Parameters

data: numpy.ndarray, shape (n_samples, n_features). Data, where n_samples is the number of samples and n_features is the number of features.

c_indx: time index or domain index that captures the unobserved changing factors.

alpha: desired significance level (float) in (0, 1).

indep_test: Independence test method function.

- "*fisherz*": Fisher's Z conditional independence test.
- "*chisq*": Chi-squared conditional independence test.
- "*gsq*": G-squared conditional independence test.
- "*kci*": kernel-based conditional independence test. (As a kernel method, its complexity is cubic in the sample size, so it might be slow if the same size is not small.)
- "*mv_fisherz*": Missing-value Fisher's Z conditional independence test.

stable: run stabilized skeleton discovery if True (default = True).

uc_rule: how unshielded colliders are oriented.

- 0: run uc_sepset.
- 1: run maxP. Orient an unshielded triple X-Y-Z as a collider with an aditional CI test.
- 2: run definiteMaxP. Orient only the definite colliders in the skeleton and keep track of all the definite non-colliders as well.

uc_priority: rule of resolving conflicts between unshielded colliders.

- -1: whatever is default in uc_rule.
- 0: overwrite.
- 1: orient bi-directed.
- 2: prioritize existing colliders.

[1] Huang, B., Zhang, K., Zhang, J., Ramsey, J. D., Sanchez-Romero, R., Glymour, C., & Schölkopf, B. (2020). Causal Discovery from Heterogeneous/Nonstationary Data. J. Mach. Learn. Res., 21(89), 1-53.

- 3: prioritize stronger colliders.
- 4: prioritize stronger* colliders.

mvpc: use missing-value PC or not. Default (and suggested for CDNOD): False.

correction_name. Missing value correction if using missing-value PC. Default: 'MV_Crtn_Fisher_Z'

Returns

cg : a CausalGraph object. Nodes in the graph correspond to the column indices in the data.

1.2.2 Score-based causal discovery methods

In this section, we would like to introduce Score-based causal discovery methods, including GES and Exact search methods. For GES, we implemented it with BIC score[1] and generalized score[2]. For Exact search, we implemented DP[3] and A*[4].

Contents:

GES with the BIC score or generalized score

Algorithm Introduction

Greedy Equivalence Search (GES) algorithm with BIC score[1] and generalized score[2].

Usage

```
from causallearn.search.ScoreBased.GES import ges
Record = ges(X, score_func, maxP, parameters)
```

Parameters

X: numpy.ndarray, shape (n_samples, n_features). Data, where n_samples is the number of samples and n_features is the number of features.

score_func: The score function you would like to use, including (see score_functions.).

- "*local_score_BIC*": BIC score[3].
- "*local_score_BDeu*": BDeu score[4].

[1] Chickering, D. M. (2002). Optimal structure identification with greedy search. Journal of machine learning research, 3(Nov), 507-554.
[2] Huang, B., Zhang, K., Lin, Y., Schölkopf, B., & Glymour, C. (2018, July). Generalized score functions for causal discovery. In Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (pp. 1551-1560).
[3] Silander, T., & Myllymäki, P. (2006, July). A simple approach for finding the globally optimal Bayesian network structure. In Proceedings of the Twenty-Second Conference on Uncertainty in Artificial Intelligence (pp. 445-452).
[4] Yuan, C., & Malone, B. (2013). Learning optimal Bayesian networks: A shortest path perspective. Journal of Artificial Intelligence Research, 48, 23-65.
[1] Chickering, D. M. (2002). Optimal structure identification with greedy search. Journal of machine learning research, 3(Nov), 507-554.
[2] Huang, B., Zhang, K., Lin, Y., Schölkopf, B., & Glymour, C. (2018, July). Generalized score functions for causal discovery. In Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (pp. 1551-1560).
[3] Schwarz, G. (1978). Estimating the dimension of a model. The annals of statistics, 461-464.
[4] Buntine, W. (1991). Theory refinement on Bayesian networks. In Uncertainty proceedings 1991 (pp. 52-60). Morgan Kaufmann.

- "*local_score_CV_general*": Generalized score with cross validation for data with single-dimensional variates[?].
- "*local_score_marginal_general*": Generalized score with marginal likelihood for data with single-dimensional variates[?].
- "*local_score_CV_multi*": Generalized score with cross validation for data with multi-dimensional variables[?].
- "*local_score_marginal_multi*": Generalized score with marginal likelihood for data with multi-dimensional variates[?].

maxP: Allowed maximum number of parents when searching the graph.

parameters: when using CV likelihood,

- parameters['kfold']: k-fold cross validation.
- parameters['lambda']: regularization parameter.
- parameters['dlabel']: for variables with multi-dimensions, indicate which dimensions belong to the i-th variable.

Returns

- **Record['G']**: learned causal graph.
- **Record['update1']**: each update (Insert operator) in the forward step.
- **Record['update2']**: each update (Delete operator) in the backward step.
- **Record['G_step1']**: learned graph at each step in the forward step.
- **Record['G_step2']**: learned graph at each step in the backward step.
- **Record['score']**: the score of the learned graph.

Exact Search

Algorithm Introduction

Search for the optimal graph using Dynamic Programming (DP[1]) or A* search[2].

Usage

```
from causallearn.search.ScoreBased.ExactSearch import bic_exact_search
dag_est, search_stats = bic_exact_search(X, super_graph, search_method,
                use_path_extension, use_k_cycle_heuristic,
                k, verbose, include_graph, max_parents)
```

[1] Silander, T., & Myllymäki, P. (2006, July). A simple approach for finding the globally optimal Bayesian network structure. In Proceedings of the Twenty-Second Conference on Uncertainty in Artificial Intelligence (pp. 445-452).

[2] Yuan, C., & Malone, B. (2013). Learning optimal Bayesian networks: A shortest path perspective. Journal of Artificial Intelligence Research, 48, 23-65.

Parameters

X: numpy.ndarray, shape=(n, d). The data to fit the structure too, where each row is a sample and each column corresponds to the associated variable.

super_graph: numpy.ndarray, shape=(d, d). Super-structure to restrict search space (binary matrix). If None, no super-structure is used. Default is None.

search_method: str. Method of exact search (['astar', 'dp']). Default is astar.

use_path_extension: bool. Whether to use optimal path extension for order graph. Note that this trick will not affect the correctness of search procedure. Default is True.

use_k_cycle_heuristic: bool. Whether to use k-cycle conflict heuristic for astar. Default is False.

k: int. Parameter used by k-cycle conflict heuristic for astar. Default is 3.

verbose: bool. Whether to log messages related to search procedure.

max_parents: int. The maximum number of parents a node can have. If used, this means using the k-learn procedure. Can drastically speed up algorithms. If None, no max on parents. Default is None.

Returns

dag_est: numpy.ndarray, shape=(d, d). Estimated DAG.

search_stats: dict. Some statistics related to the seach procedure.

1.2.3 Causal discovery methods based on constrained functional causal models

In this section, we would like to introduce causal discovery methods based on constrained functional causal models. Now we have LiNGAM-based methods (ICA-based LiNGAM[1], DirectLiNGAM[2], VAR-LiNGAM[3], RCD[4], and CAM-UV[5]), post-nonlinear (PNL[6]) causal models, and additive noise models (ANM[7]).

Contents:

[1] Shimizu, S., Hoyer, P. O., Hyvärinen, A., Kerminen, A., & Jordan, M. (2006). A linear non-Gaussian acyclic model for causal discovery. Journal of Machine Learning Research, 7(10).

[2] Shimizu, S., Inazumi, T., Sogawa, Y., Hyvärinen, A., Kawahara, Y., Washio, T., … & Bollen, K. (2011). DirectLiNGAM: A direct method for learning a linear non-Gaussian structural equation model. The Journal of Machine Learning Research, 12, 1225-1248.

[3] Hyvärinen, A., Zhang, K., Shimizu, S., & Hoyer, P. O. (2010). Estimation of a structural vector autoregression model using non-gaussianity. Journal of Machine Learning Research, 11(5).

[4] Maeda, T. N., & Shimizu, S. (2020, June). RCD: Repetitive causal discovery of linear non-Gaussian acyclic models with latent confounders. In International Conference on Artificial Intelligence and Statistics (pp. 735-745). PMLR.

[5] Maeda, T. N., & Shimizu, S. (2021). Causal Additive Models with Unobserved Variables. UAI.

[6] Zhang, K., & Hyvärinen, A. (2009, June). On the Identifiability of the Post-Nonlinear Causal Model. In 25th Conference on Uncertainty in Artificial Intelligence (UAI 2009) (pp. 647-655). AUAI Press.

[7] Hoyer, P. O., Janzing, D., Mooij, J. M., Peters, J., & Schölkopf, B. (2008, December). Nonlinear causal discovery with additive noise models. In NIPS (Vol. 21, pp. 689-696).

LiNGAM-based Methods

Estimation of Linear, Non-Gaussian Acyclic Model from observed data. It assumes non-Gaussianity of the noise terms in the causal model.

causal-learn has the official implementations for a set of LiNGAM-based methods (e.g., ICA-based LiNGAM[1], DirectLiNGAM[2], VAR-LiNGAM[3], RCD[4], and CAM-UV[5]). And we are actively updating the list.

ICA-based LiNGAM

```
from causallearn.search.FCMBased import lingam
model = lingam.ICALiNGAM(random_state, max_iter)
model.fit(X)

print(model.causal_order_)
print(model.adjacency_matrix_)
```

Parameters

random_state: int, optional (default=None). The seed used by the random number generator.

max_iter: int, optional (default=1000). The maximum number of iterations of FastICA.

X: array-like, shape (n_samples, n_features). Training data, where n_samples is the number of samples and n_features is the number of features.

Returns

model.causal_order_: array-like, shape (n_features). The causal order of fitted model, where n_features is the number of features.

model.adjacency_matrix_: array-like, shape (n_features, n_features). The adjacency matrix B of fitted model, where n_features is the number of features.

[1] Shimizu, S., Hoyer, P. O., Hyvärinen, A., Kerminen, A., & Jordan, M. (2006). A linear non-Gaussian acyclic model for causal discovery. Journal of Machine Learning Research, 7(10).

[2] Shimizu, S., Inazumi, T., Sogawa, Y., Hyvärinen, A., Kawahara, Y., Washio, T., … & Bollen, K. (2011). DirectLiNGAM: A direct method for learning a linear non-Gaussian structural equation model. The Journal of Machine Learning Research, 12, 1225-1248.

[3] Hyvärinen, A., Zhang, K., Shimizu, S., & Hoyer, P. O. (2010). Estimation of a structural vector autoregression model using non-gaussianity. Journal of Machine Learning Research, 11(5).

[4] Maeda, T. N., & Shimizu, S. (2020, June). RCD: Repetitive causal discovery of linear non-Gaussian acyclic models with latent confounders. In International Conference on Artificial Intelligence and Statistics (pp. 735-745). PMLR.

[5] Maeda, T. N., & Shimizu, S. (2021). Causal Additive Models with Unobserved Variables. UAI.

DirectLiNGAM

```
from causallearn.search.FCMBased import lingam
model = lingam.DirectLiNGAM(random_state, prior_knowledge, apply_prior_knowledge_softly,␣
↪measure)
model.fit(X)

print(model.causal_order_)
print(model.adjacency_matrix_)
```

Parameters

random_state: int, optional (default=None). The seed used by the random number generator.

prior_knowledge: array-like, shape (n_features, n_features), optional (default=None). Prior knowledge used for causal discovery, where `n_features` is the number of features. The elements of prior knowledge matrix are defined as follows:

- 0: x_i does not have a directed path to x_j
- 1: x_i has a directed path to x_j
- -1: No prior knowledge is available to know if either of the two cases above (0 or 1) is true.

apply_prior_knowledge_softly: boolean, optional (default=False). If True, apply prior knowledge softly.

measure: {'pwling', 'kernel'}, optional (default='pwling'). Measure to evaluate independence: 'pwling' or 'kernel'.

X: array-like, shape (n_samples, n_features). Training data, where n_samples is the number of samples and n_features is the number of features.

Returns

model.causal_order_: array-like, shape (n_features). The causal order of fitted model, where n_features is the number of features.

model.adjacency_matrix_: array-like, shape (n_features, n_features). The adjacency matrix B of fitted model, where n_features is the number of features.

VAR-LiNGAM

```
from causallearn.search.FCMBased import lingam
model = lingam.VARLiNGAM(lags, criterion, prune, ar_coefs, lingam_model, random_state)
model.fit(X)

print(model.causal_order_)
print(model.adjacency_matrices_[0])
print(model.adjacency_matrices_[1])
print(model.residuals_)
```

Parameters

lags: int, optional (default=1). Number of lags.

criterion: {'aic', 'fpe', 'hqic', 'bic', None}, optional (default='bic'). Criterion to decide the best lags within 'lags'. Searching the best lags is disabled if 'criterion' is None.

prune: boolean, optional (default=False). Whether to prune the adjacency matrix or not.

ar_coefs: array-like, optional (default=None). Coefficients of AR model. Estimating AR model is skipped if specified 'ar_coefs'. Shape must be ('lags', n_features, n_features).

lingam_model: lingam object inherits 'lingam._BaseLiNGAM', optional (default=None). LiNGAM model for causal discovery. If None, DirectLiNGAM algorithm is selected.

random_state: int, optional (default=None). 'random_state' is the seed used by the random number generator.

X: array-like, shape (n_samples, n_features). Training data, where n_samples is the number of samples and n_features is the number of features.

Returns

model.causal_order_: array-like, shape (n_features). The causal order of fitted model, where n_features is the number of features.

model.adjacency_matrices_: array-like, shape (lags, n_features, n_features). The adjacency matrix of fitted model, where n_features is the number of features.

model.residuals_: array-like, shape (n_samples). Residuals of regression, where n_samples is the number of samples.

RCD

```
from causallearn.search.FCMBased import lingam
model = lingam.RCD(max_explanatory_num, cor_alpha, ind_alpha, shapiro_alpha, MLHSICR, bw_
↪method)
model.fit(X)

print(model.adjacency_matrix_)
print(model.ancestors_list_)
```

Parameters

max_explanatory_num: int, optional (default=2). Maximum number of explanatory variables.

cor_alpha: float, optional (default=0.01). Alpha level for pearson correlation.

ind_alpha: float, optional (default=0.01). Alpha level for HSIC.

shapiro_alpha: float, optional (default=0.01). Alpha level for Shapiro-Wilk test.

MLHSICR: bool, optional (default=False). If True, use MLHSICR for multiple regression, if False, use OLS for multiple regression.

bw_method: str, optional (default='mdbs'). The method used to calculate the bandwidth of the HSIC.

- 'mdbs': Median distance between samples.

- 'scott': Scott's Rule of Thumb.
- 'silverman': Silverman's Rule of Thumb.

X: array-like, shape (n_samples, n_features). Training data, where n_samples is the number of samples and n_features is the number of features.

Returns

model.adjacency_matrix_: array-like, shape (n_features, n_features). The adjacency matrix B of fitted model, where n_features is the number of features.

model.ancestors_list_: array-like, shape (n_features). The list of causal ancestors sets, where n_features is the number of features.

CAM-UV

```
from causallearn.search.FCMBased.lingam import CAMUV
P, U = CAMUV.execute(data, alpha, num_explanatory_vals)

for i, result in enumerate(P):
    if not len(result) == 0:
        print("child: " + str(i) + ",  parents: " + str(result))

for result in U:
    print(result)
```

Parameters

data: array-like, shape (n_samples, n_features). Training data, where n_samples is the number of samples and n_features is the number of features.

alpha: the alpha level for independence testing.

num_explanatory_vals: the maximum number of variables to infer causal relationships. This is equivalent to d in the paper.

Returns

P: P[i] contains the indices of the parents of Xi.

U: The indices of variable pairs having UCPs or UBPs.

Post-nonlinear causal models

Algorithm Introduction

Causal discovery based on the post-nonlinear (PNL[1]) causal models. If you would like to apply the method to more than two variables, we suggest you first apply the PC algorithm and then use pair-wise analysis in this implementation to find the causal directions that cannot be determined by PC.

Usage

```
from causallearn.search.FCMBased.PNL.PNL import PNL
pnl = PNL()
p_value_foward, p_value_backward = pnl.cause_or_effect(data_x, data_y)
```

Parameters

data_x: input data (n, 1), n is the sample size.

data_y: output data (n, 1), n is the sample size.

Returns

pval_forward: p value in the x->y direction.

pval_backward: p value in the y->x direction.

Additive noise models

Algorithm Introduction

Causal discovery based on the additive noise models (ANM[1]). If you would like to apply the method to more than two variables, we suggest you first apply the PC algorithm and then use pair-wise analysis in this implementation to find the causal directions that cannot be determined by PC.

Usage

```
from causallearn.search.FCMBased.ANM.ANM import ANM
anm = ANM()
p_value_foward, p_value_backward = anm.cause_or_effect(data_x, data_y)
```

[1] Zhang, K., & Hyvärinen, A. (2009, June). On the Identifiability of the Post-Nonlinear Causal Model. In 25th Conference on Uncertainty in Artificial Intelligence (UAI 2009) (pp. 647-655). AUAI Press.

[1] Hoyer, P. O., Janzing, D., Mooij, J. M., Peters, J., & Schölkopf, B. (2008, December). Nonlinear causal discovery with additive noise models. In NIPS (Vol. 21, pp. 689-696).

Parameters

data_x: input data (n, 1).

data_y: output data (n, 1).

Returns

pval_forward: p value in the x->y direction.

pval_backward: p value in the y->x direction.

1.2.4 Hidden causal representation learning

In this section, we would like to introduce methods in hidden causal representation learning, such as generalized independent noise (GIN[1]) condition-based method.

Contents:

Generalized Independence Noise (GIN) condition-based method

Algorithm Introduction

Learning the structure of Linear, Non-Gaussian LAtent variable Model (LiNLAM) based the GIN[1] condition.

Usage

```
from causallearn.search.FCMBased.GIN.GIN import GIN
G, K = GIN(data)
```

Parameters

data: numpy.ndarray, shape (n_samples, n_features). Data, where n_samples is the number of samples and n_features is the number of features.

Returns

G: GeneralGraph. Causal graph.

K: list. Causal Order.

[1] Xie, F., Cai, R., Huang, B., Glymour, C., Hao, Z., & Zhang, K. (2020, January). Generalized Independent Noise Condition for Estimating Latent Variable Causal Graphs. In NeurIPS.

[1] Xie, F., Cai, R., Huang, B., Glymour, C., Hao, Z., & Zhang, K. (2020, January). Generalized Independent Noise Condition for Estimating Latent Variable Causal Graphs. In NeurIPS.

1.2.5 Granger causality

In this section, we would like to introduce methods in Granger Causality. Now we have linear Granger Causality.

Contents:

Linear granger causality

Algorithm Introduction

Implementation of granger causality[1], including 1) regression+hypothesis test and 2) lasso regression.

Usage

```
from causallearn.search.Granger.Granger import Granger
G = Granger()
p_value_matrix = G.granger_test_2d(data)
coeff = G.granger_lasso(data)
```

Parameters

data: numpy.ndarray, shape (n_samples, n_features). Data, where n_samples is the number of samples and n_features is the number of features.

Returns

p_value_matrix: p values for x1->x2 and x2->x1 (for 'granger_test_2d', which is the granger causality test for two-dimensional time series).

coeff: coefficient matrix (for 'granger_lasso', which is the granger causality test for multi-dimensional time series).

1.3 (Conditional) independence tests

In this section, we would like to introduce (conditional) independence tests in causal-learn. Currently we have Fisher-z test[1], Missing-value Fisher-z test, Chi-Square test, Kernel-based conditional independence (KCI) test and independence test[2], and G-Square test[3].

Contents:

[1] Granger, C. W. (1969). Investigating causal relations by econometric models and cross-spectral methods. Econometrica: journal of the Econometric Society, 424-438.

[1] Fisher, R. A. (1921). On the'probable error'of a coefficient of correlation deduced from a small sample. Metron, 1, 1-32.

[2] Zhang, K., Peters, J., Janzing, D., & Schölkopf, B. (2011, July). Kernel-based Conditional Independence Test and Application in Causal Discovery. In 27th Conference on Uncertainty in Artificial Intelligence (UAI 2011) (pp. 804-813). AUAI Press.

[3] Tsamardinos, I., Brown, L. E., & Aliferis, C. F. (2006). The max-min hill-climbing Bayesian network structure learning algorithm. Machine learning, 65(1), 31-78.

1.3.1 Fisher-z test

Perform an independence test using Fisher-z's test[1]. This test is optimal for linear-Gaussian data.

Usage

```
from causallearn.utils.cit import fisherz
p = fisherz(data, X, Y, condition_set, correlation_matrix)
```

Parameters

data: numpy.ndarray, shape (n_samples, n_features). Data, where n_samples is the number of samples and n_features is the number of features.

X, Y and condition_set: column indices of data.

correlation_matrix: correlation matrix; None means without the parameter of correlation matrix.

Returns

p: the p-value of the test.

1.3.2 Missing-value Fisher-z test

Perform a testwise-deletion Fisher-z independence test to data sets with missing values. With testwise-deletion, the test makes use of all data points that do not have missing values for the variables involved in the test.

Usage

```
from causallearn.utils.cit import mv_fisherz
p = mv_fisherz(mvdata, X, Y, condition_set)
```

Parameters

mvdata: numpy.ndarray, shape (n_samples, n_features). Data with missing value, where n_samples is the number of samples and n_features is the number of features.

X, Y and condition_set: column indices of data.

[1] Fisher, R. A. (1921). On the'probable error'of a coefficient of correlation deduced from a small sample. Metron, 1, 1-32.

Returns

p: the p-value of the test.

1.3.3 Chi-Square test

Perform an independence test on discrete variables using Chi-Square test.

Usage

```
from causallearn.utils.cit import chisq
p = chisq(data, X, Y, conditioning_set)
```

Parameters

data: numpy.ndarray, shape (n_samples, n_features). Data, where n_samples is the number of samples and n_features is the number of features.

X, Y and condition_set: column indices of data.

G_sq: True means using G-Square test; False means using Chi-Square test.

Returns

p: the p-value of the test.

1.3.4 Kernel-based conditional independence (KCI) test and independence test

Kernel-based conditional independence (KCI) test and independence test[1]. To test if x and y are conditionally or unconditionally independent on Z. For unconditional independence tests, Z is set to the empty set.

Usage

```
from causallearn.utils.cit import kci
p = kci(data, X, Y, condition_set, kernelX, kernelY, kernelZ, est_width, polyd, kwidthx,␣
↪kwidthy, kwidthz)
```

[1] Zhang, K., Peters, J., Janzing, D., & Schölkopf, B. (2011, July). Kernel-based Conditional Independence Test and Application in Causal Discovery. In 27th Conference on Uncertainty in Artificial Intelligence (UAI 2011) (pp. 804-813). AUAI Press.

Parameters

data: numpy.ndarray, shape (n_samples, n_features). Data, where n_samples is the number of samples and n_features is the number of features.

X, Y, and condition_set: column indices of data. condition_set could be None.

KernelX/Y/Z (condition_set): ['GaussianKernel', 'LinearKernel', 'PolynomialKernel']. (For 'PolynomialKernel', the default degree is 2. Currently, users can change it by setting the 'degree' of 'class PolynomialKernel()'.

est_width: set kernel width for Gaussian kernels.

- 'empirical': set kernel width using empirical rules (default).
- 'median': set kernel width using the median trick.

polyd: polynomial kernel degrees (default=2).

kwidthx: kernel width for data x (standard deviation sigma).

kwidthy: kernel width for data y (standard deviation sigma).

kwidthz: kernel width for data z (standard deviation sigma).

Returns

p: the p value.

1.3.5 G-Square test

Perform an independence test using G-Square test[1]. This test is based on the log likelihood ratio test.

Usage

```
from causallearn.utils.cit import gsq
p = gsq(data, X, Y, conditioning_set)
```

Parameters

data: numpy.ndarray, shape (n_samples, n_features). Data, where n_samples is the number of samples and n_features is the number of features.

X, Y and condition_set: column indices of data.

G_sq: True means using G-Square test; False means using Chi-Square test.

[1] Tsamardinos, I., Brown, L. E., & Aliferis, C. F. (2006). The max-min hill-climbing Bayesian network structure learning algorithm. Machine learning, 65(1), 31-78.

Returns

p: the p-value of the test

1.4 Score functions

In this section, we would like to introduce score functions in causal-learn. Currently we have BIC score[1], BDeu score[2], generalized score with cross validation or marginal likelihood[3].

Contents:

1.4.1 BIC score

Calculate the local score with Bayesian Information Criterion (BIC[1]) for the linear Gaussian case.

Usage

```
from causallearn.score.LocalScoreFunction import local_score_bic
score = local_score_bic(Data, i, PAi, parameters)
```

Parameters

Data: (sample, features).

i: current index.

PAi: parent indexes.

parameters: None.

Returns

score: Local BIC score.

1.4.2 BDeu score

Calculate the local score with BDeu[1] for the discrete case.

[1] Schwarz, G. (1978). Estimating the dimension of a model. The annals of statistics, 461-464.
[2] Buntine, W. (1991). Theory refinement on Bayesian networks. In Uncertainty proceedings 1991 (pp. 52-60). Morgan Kaufmann.
[3] Huang, B., Zhang, K., Lin, Y., Schölkopf, B., & Glymour, C. (2018, July). Generalized score functions for causal discovery. In Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (pp. 1551-1560).
[1] Schwarz, G. (1978). Estimating the dimension of a model. The annals of statistics, 461-464.
[1] Buntine, W. (1991). Theory refinement on Bayesian networks. In Uncertainty proceedings 1991 (pp. 52-60). Morgan Kaufmann.

Usage

```
from causallearn.score.LocalScoreFunction import local_score_bdeu
score = local_score_bdeu(Data, i, PAi, parameters)
```

Parameters

Data: (sample, features).

i: current index.

PAi: parent indexes.

parameters:

- sample_prior: sample prior.
- structure_prior: structure prior.
- r_i_map: number of states of the finite random variable 'X_{i}'.

Returns

score: Local BDeu score.

1.4.3 Generalized score with cross validation

Generalized score with cross validation for single-dimensional variables

Calculate the local score using negative k-fold cross-validated log likelihood as the score, based on a regression model in RKHS[1].

Usage

```
from causallearn.score.LocalScoreFunction import local_score_cv_general
score = local_score_cv_general(Data, Xi, PAi, parameters)
```

Parameters

Data: (sample, features).

Xi: current index.

PAi: parent indexes.

parameters:

- kfold: the fold number in cross validation.
- lambda: regularization parameter.

[1] Huang, B., Zhang, K., Lin, Y., Schölkopf, B., & Glymour, C. (2018, July). Generalized score functions for causal discovery. In Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (pp. 1551-1560).

Returns

score: Local score.

Generalized score with cross validation for multi-dimensional variables

Calculate the local score using negative k-fold cross-validated log likelihood as the score, based on a regression model in RKHS for data with multi-dimensional variables[Page 24, 1].

Usage

```
from causallearn.score.LocalScoreFunction import local_score_cv_multi
score = local_score_cv_multi(Data, Xi, PAi, parameters)
```

Parameters

Data: (sample, features).

Xi: current index.

PAi: parent indexes.

parameters:

- kfold: the fold number in cross validation.
- lambda: regularization parameter.
- dlabel: indicate the data dimensions that belong to each variable. It is only used when the variables have multivariate dimensions.

Returns

score: Local score.

1.4.4 Generalized score with marginal likelihood

Generalized score with marginal likelihood for single dimensional variables

Calculate the local score by negative marginal likelihood, based on a regression model in RKHS[1].

[1] Huang, B., Zhang, K., Lin, Y., Schölkopf, B., & Glymour, C. (2018, July). Generalized score functions for causal discovery. In Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (pp. 1551-1560).

Usage

```
from causallearn.score.LocalScoreFunction import local_score_marginal_general
score = local_score_marginal_general(Data, Xi, PAi, parameters)
```

Parameters

Data: (sample, features).

Xi: current index.

PAi: parent indexes.

parameters: None.

Returns

score: Local score.

Generalized score with marginal likelihood for multi-dimensional variables

Calculate the local score by negative marginal likelihood, based on a regression model in RKHS for data with multi-dimensional variables[Page 25, 1].

Usage

```
from causallearn.score.LocalScoreFunction import local_score_marginal_multi
score = local_score_marginal_multi(Data, Xi, PAi, parameters)
```

Parameters

Data: (sample, features).

Xi: current index.

PAi: parent indexes.

parameters:

- dlabel: indicate the data dimensions that belong to each variable. It is only used when the variables have multivariate dimensions.

Returns

score: Local score.

1.5 Utilities

In this section, we would like to introduce utilities in causal-learn, such as graph operations and evaluations.

Contents:

1.5.1 Graph operations

In this section, we would like to introduce graph operations in causal-learn.

Contents:

DAG2CPDAG

Covert a DAG to its corresponding CPDAG.

Usage

```
from causallearn.utils.DAG2CPDAG import dag2cpdag
CPDAG = dag2cpdag(G)
```

Parameters

G: Direct Acyclic Graph.

Returns

CPDAG: Completed Partially Direct Acyclic Graph.

DAG2PAG

Covert a DAG to its corresponding PAG.

Usage

```
from causallearn.utils.DAG2PAG import dag2pag
PAG = dag2pag(dag, islatent)
```

Parameters

dag: Direct Acyclic Graph.

islatent: the indexes of latent variables. [] means there is no latent variable.

Returns

PAG: Partial Ancestral Graph.

PDAG2DAG

Covert a PDAG to its corresponding DAG.

Usage

```
from causallearn.utils.PDAG2DAG import pdag2dag
Gd = pdag2dag(G)
```

Parameters

G: Partially Direct Acyclic Graph.

Returns

Gd: Direct Acyclic Graph.

TXT2GeneralGraph

Convert text file of Tetrad results into GeneralGraph class for causal-learn.

Usage

```
from causallearn.utils.TXT2GeneralGraph import txt2generalgraph
G = txt2generalgraph(filename)
```

Parameters

filename: Text file of Tetrad results.

Returns

G: GeneralGraph Class for causal-learn.

1.5.2 Evaluations

Usage

```
from causallearn.graph.ArrowConfusion import ArrowConfusion
from causallearn.graph.AdjacencyConfusion import AdjacencyConfusion
from causallearn.graph.SHD import SHD

# For arrows
arrow = ArrowConfusion(truth_cpdag, est)

arrowsTp = arrow.get_arrows_tp()
arrowsFp = arrow.get_arrows_fp()
arrowsFn = arrow.get_arrows_fn()
arrowsTn = arrow.get_arrows_tn()

arrowPrec = arrow.get_arrows_precision()
arrowRec = arrow.get_arrows_recall()

# For adjacency matrices
adj = AdjacencyConfusion(truth_cpdag, est)

adjTp = adj.get_adj_tp()
adjFp = adj.get_adj_fp()
adjFn = adj.get_adj_fn()
adjTn = adj.get_adj_tn()

adjPrec = adj.get_adj_precision()
adjRec = adj.get_adj_recall()

# Structural Hamming Distance
shd = SHD(truth_cpdag, est).get_shd()
```

Parameters

X: Data with T*D dimensions.

truth_cpdag: Graph class.

est: Graph class.

Returns

arrowsTp/Fp/Fn/Tn: True postive/false positive/false negative/true negative arrows.

arrowPrec: Precision for arrows.

arrowRec: Recall for arrows.

adjTp/Fp/Fn/Tn: True postive/false positive/false negative/true negative edges.

adjPrec: Precision for the adjacency matrix.

adjRec: Recall for the adjacency matrix.

shd: Structural Hamming Distance.