Appendix II:

Technical Report Describing the Datasets of the Challenge

Datasets of the Causation and Prediction Challenge

The Causality Workbench Team

Isabelle Guyon
Clopinet, California
isabelle@clopinet.com
Constantin Aliferis
New-York University, New-York.
constantin.aliferis@nyumc.org
Greg Cooper
University of Pittsburgh, Pennsylvania
gfc@pitt.edu
André Elisseeff
IBM Research, Zürich
ael@zurich.ibm.com
Jean-Philippe Pellet
IBM Research and ETH, Zürich
jep@zurich.ibm.com
Peter Spirtes
Carnegie Mellon University, Pennsylvania
ps7z@andrew.cmu.edu
Alexander Statnikov
New-York University, New-York.
statnikov@gmail.com

Introduction

We prepared four datasets for the first challenge on causality we organized for the World Congress on Computational Intelligence, WCCI 2008. The focus of this challenge, entitled "**Causation and Prediction**", was on the **evaluation of causal modeling techniques**, aiming at predicting **the effect of "interventions"** performed by an external agent. Examples of that problem are found in the medical domain to predict the effect of a drug prior to administering it, or in econometrics to predict the effect of a new policy prior to issuing it. We concentrated on a given target variable to be predicted (*e.g.*, health status of a patient) from a number of candidate predictive variables or "features" (*e.g.*, risk factors in the medical domain). We limited ourselves to **binary target variables** (two-class classification problems), but **the input variables are either binary or continuous**. For each task, a **training set drawn from a "natural" distribution** is given and three test sets: **one test set from the same distribution as the training set** and **two test sets obtained after an external agent manipulated certain variables** (i.e. set them to arbitrary values, not drawn from the natural distribution). The target variable itself is never manipulated and it is assumed that the external agent interventions do not alter the mechanisms by which one variable is determined by the value of others. The participants were asked to provide predictions of the target variable on test data and the list of variables (features) used to make predictions. The challenge platform remains open for post-challenge submissions (see http://clopinet.com/causality). The datasets were also used for the task **LOCANET**, which was part of the second causality challenge we organized for the Neural Information Processing Systems conference (NIPS 2008). The goal of LOCANET was to uncover the LOcal CAusal NETwork around the target. **This report was not available to the participants of the challenges.**

Table 1: Datasets of the causation and prediction challenge. The datasets of the challenge are indicated in boldface. Two other toy datasets (LUCAS and LUCAP) were used for illustration purpose.

Dataset	Description	Var. type	Var. #	Train	Test
LUCAS	Toy example (Bayes net)	binary	11	10000	20000
LUCAP	Toy example (Bayes net + probes)	binary	143	10000	20000
REGED	Genomics re-simulated	numeric	999	500	20000
SIDO	Pharmacology (real data + probes)	binary	4932	12678	10000
CINA	Marketing (real data + probes)	mixed	132	16033	10000
MARTI	Same as REGED + correlated noise	numeric	1024	500	20000

Brief description of the data

- **REGED** is a dataset generated by a simulator of gene expression data, which was trained on real DNA microarray data. The target variable is lung cancer subtype. Hence, the task is to discover genes, which trigger disease or are a consequence of disease. The manipulations simulate the effect of agents such as drugs and/or RNA silencing. For REGED1, the list of manipulated variable is provided, but not for REGED2. REGED has 999 features, of which the Markov blanket contains 2 direct causes, 13 direct effects and 6 spouses in REGED0, but only 2 direct causes, 6 direct effects and 4 spouses in REGED1, and 2 direct causes in REGED2.
- **SIDO** consists of real data, from a drug discovery problem. The variables represent molecular descriptors of pharmaceutical compounds, whose activity on the HIV virus must be determined (the target variable). Knowing which molecule feature is a cause of activity would be of great help to chemical engineers to design new compounds. To test the efficacy of causal discovery algorithm, artificial "distractor" variables (called "probes") were added, which are "non-causes" of the target. All the probes are manipulated in the test sets SIDO1 and SIDO2. The probes must be filtered out to get a good causal discovery score and good prediction performance on test data. SIDO has 4932 features, of which 1644 real features and 3288 probes.
- **CINA** is also a real dataset. The problem is to predict the revenue level of people from census data (marital status, years of study, gender, etc.). As a causal discovery problem, the task is to find causes, which might influence revenue. Similarly as for SIDO, artificial variables (probes) were added. CINA has 132 features, of which 44 real features and 88 probes; all probes are manipulated in CINA1 and 2.
- **MARTI** is a noisy version of REGED. Correlated noise was added to simulate measurement artifacts and introduce spurious relationships between variables. This dataset illustrates that without proper calibration/normalization of data, causal discovery algorithms may yield wrong causal structures. MARTI has 1024 features and the same causal graph as REGED. However, 25 calibrant variables were added to help taking out the noise.

Overall method:

Preparing the data included the following steps:

- Adding artificial variables (probes) in real datasets.
- Preprocessing data to obtain features in the same numerical range (0 to 999 for continuous data and 0/1 for binary data).
- Randomizing the order of the patterns and the features to homogenize the data.
- Splitting the data into training and test set.

The classification results were evaluated with the Area under the ROC Curve (AUC) on test data. The target values on test examples were never revealed. The web site remains available to assess performances of new algorithms. No validation set was used to provide on-line feed-back to the participants during the challenge. Rather, the participants submitted results on test data and, during the development period, obtained a coarse information on the web site about their ranking (in which quartile their submission ranked).

Although the participants were strictly evaluated on prediction performance of the target variable, other metrics were computed to assess the correlation between correct causal structure discovery and correct target value predictions. To assess causal structure discovery, an index measuring the similarity of the feature set to the Markov boundary of the post-manipulation distribution was calculated (see the website of the challenge for details). In a follow up challenge (NIPS 2008), we asked the participants to return the depth 3 local structure, and we assessed its correctness with an edit distance to the true graph (see http://www.causality.inf.ethz.ch/data/LOCANET.html).

Real and artificial data:

We use two types of data:

- **Re-simulated data:** We train a causal model with real data. The model is then used to generate artificial training and test data for the challenge. Truth values of causal relationships are known for the data generating model and used for scoring causal discovery results. REGED is an example of re-simulated dataset.
- **Real data with probe variables:** We use a dataset of real samples. Some of the variables may be causally related to the target and some may be predictive but non-causal. The nature of the causal relationships of the variables to the target is unknown (although domain knowledge may allow us to validate the discoveries to some extent). We have added to the set of real variables a number of distractor variables called “probes”, which are generated by an artificial stochastic process, including explicit functions of some of the real variables, other artificial variables, and/or the target. All probes are non-causes of the target, some are completely unrelated to the target. The identity of the probes in concealed. The fact that truth values of causal relationships are known only for the probes affects the evaluation of causal discovery, which is less reliable than for artificial data.

We give in appendix details about the method we used to generate random probes.

Evaluation:

For the first causality challenge "Causation and Prediction" organized for WCCI 2008, the participants were asked to return *prediction scores* or discriminant values v for the target variable on test examples, and a list of features used for computing the prediction scores, sorted in order of decreasing predictive power, or unsorted. The classification decision is made by setting a threshold μ on the discriminant value v: predict the positive class if $v > \mu$ and the negative class otherwise. The participants could optionally provide results for nested subsets of features, varying the subset size by powers of 2 (1, 2, 4, 8, etc.). Two scores were used:

- **Tscore:** The participants were ranked according to the area under the ROC curve (AUC) computed for test examples (referred to as Tscore), which is the area under the curve plotting sensitivity *vs.* (1¡ specificity) when the threshold μ is varied (or equivalently the area under the curve sensitivity *vs.* specificity). We call "sensitivity" the error rate of the positive class and "specificity" the error rate of the negative class. The AUC is a standard metric in classification. If results were provided for nested subsets of features, the best Tscore was retained. There are several ways of estimating error bars for the AUC. We use a simple heuristic, which gives us approximate error bars, and is fast and easy to implement: we find on the AUC curve the point corresponding to the largest balanced accuracy BAC = 0.5 (sensitivity + specificity). We then estimate the standard deviation of the BAC as: σ = (1/2) sqrt($p_+(1-p_+)/m_+ + p_-(1-p_-)/m_-$), where m_+ is the number of examples of the positive class, m_- is the number of examples of the negative class, and p_+ and p_- are the probabilities of error on examples of the positive and negative class, approximated by their empirical estimates, the sensitivity and the specificity.
- **Fscore:** We also computed other statistics, which were not used to rank participants, but used in the analysis of the results. Those included the number of features used by the participants called "Fnum", and a statistic assessing the quality of causal discovery in the feature set selected called "Fscore". As with the Tscore, we provided quartile feed-back on Fnum and Fscore during the competition. For the Fscore, we used the AUC for the problem of separating features belonging to the Markov blanket of the test set distribution *vs.* other features. Details are provided on the web site of the challenge. As it turns out, this statistic correlates poorly with the Tscore. After experimenting with various scores, we found better alternatives.
- **New Fscore**. We ended up using as the new Fscore the Fmeasure for REGED and MARTI and the precision for SIDO and CINA, after experimenting with various alternative measures inspired by information retrieval. We use the following definitions: precision = $tp/(tp + fp)$, recall = $tp/(tp + fn)$ (also called sensitivity), and Fmeasure = 2 precision recall / (precision + recall). Our explorations indicate that precision, recall, and Fmeasure correlate well with Tscore for artificially generated datasets (REGED and MARTI). The *Fmeasure*, which captures the tradeoff between precision and recall, is a good measure of feature set quality for these datasets. However, recall correlates poorly with Tscore for SIDO and CINA,

which are datasets of real variables with added artificial “probes”. In these cases, we approximate the recall by the fraction of real variables recalled (present in the selected feature set), which can be very different from the true recall that is the fraction of relevant variables. For instance, if many real variables are irrelevant, a good causal discovery algorithm might eliminate them, thus obtaining a poor estimated recall. Hence, we can only use *precision* as of feature set quality for those datasets.

For the LOCANET task of the second causality challenge (NIPS 2008 Pot-luck Challenge), we assessed performance by comparing the local causal network (of depth 3) to the actual local causal network, using an edit distance. A confusion matrix C_{ij} was computed, recording the number of relatives confused for another type of relative, among the 14 types of relatives in depth 3 networks. A cost matrix A_{ij}, was then applied to account for the distance between relatives (computed with an edit distance as the number of substitutions, insertion, or deletion to go from one string to the other, using the string description described above). The score of the solution was computed as:

$S = \text{sum}_{ij} A_{ij} C_{ij}$

Cost matrix (A_{ij}):

Depth	Desired		1	1	2	2	2	2	3	3	3	3	3	3	3	3	X
Obtained	Relationship		P	C	Sp	GC	Si	GP	GGP	uud	N	PS	SC	IL	CP	GGC	Other
			u	d	du	dd	ud	uu	uuu	uud	udd	udu	ddu	duu	dud	ddd	
1	Parents	u	0	1	1	2	1	1	2	2	2	2	2	2	2	3	4
1	Children	d	1	0	1	1	1	2	3	2	2	2	2	2	2	2	4
2	Spouses	du	1	1	0	1	2	1	2	2	2	1	1	1	1	2	4
2	Gchildren	dd	2	1	1	0	1	2	3	2	1	2	1	2	1	1	4
2	Siblings	ud	1	1	2	1	0	1	2	1	1	1	2	2	1	2	4
2	Gparents	uu	1	2	1	2	1	0	1	1	2	1	2	1	2	3	4
3	Ggparents	uuu	2	3	2	3	2	1	0	1	2	1	2	1	2	3	4
3	Uncles/Aunts	uud	2	2	2	2	1	1	1	0	1	2	3	2	1	2	4
3	Nieces/Nephews	udd	2	2	2	1	1	2	2	1	0	1	2	3	2	1	4
3	Parents of siblings	udu	2	2	1	2	1	1	1	2	1	0	1	2	2	2	4
3	Spouses of children	ddu	2	2	1	1	2	2	2	3	2	1	0	1	2	1	4
3	Parents in law	duu	2	2	1	2	2	1	1	2	3	2	1	0	1	2	4
3	Children of spouses	dud	2	2	1	1	1	2	2	1	2	2	2	1	0	1	4
3	Ggchildren	ddd	3	2	2	1	2	3	3	2	1	2	1	2	1	0	4
X	Other		4	4	4	4	4	4	4	4	4	4	4	4	4	4	0

For artificially generated data (REGED and MARTI), the ground truth for the target local neighborhood was determined by the generative model. For real data with artificial "probe" variables (SIDO and CINA), we do not have ground truth for the relationships of the real variables to the target. The score was computed on the basis of the artificial variables only.

Data formats:
All the data sets are in the same format and include 4 files in text format:
dataname.param: Parameters and statistics about the data
dataname_train.data: Training set (a sparse or a regular matrix, patterns in lines, features in columns).
dataname_test.data: Test set.
dataname_train.targets: Labels (truth values of the classes) for training examples.
The matrix data formats used are a space delimited file with a new-line character at the end of each line.
The results on each dataset should be formatted in 3 ASCII files:
dataname_train.predict: a discriminant value for training set output (a discriminant value is a score, which is large for examples of the positive class and small for examples of the negative class).
dataname_test.predict: a discriminant value for test set output.
dataname_feat.slist: a sorted list of features used.
or
dataname_feat.ulist: an unsorted list of features used.
Single predictions for each training or test examples could be provided, or multiple predictions corresponding to nested tested subsets of features could be given in the form of a data table (see the website of the challenge for details).

Dataset A: REGED

1) Topic

REGED stands for **RE**simulated **G**ene **E**xpression **D**ataset. The goal of **REGED** is to find genes, which could be responsible of lung cancer subtype. The data are "re-simulated", i.e. generated by a model derived from real human lung-cancer microarray gene expression data. From the causal discovery point of view, it is important to separate genes whose activity cause lung cancer from those whose activity is a consequence of the disease.
We propose three tasks, REGED0, REGED1, and REGED2. All three datasets includes 999 features, the same 500 training examples, and different test sets of 20000 examples. The target variable is binary; it separates adenocarcinoma samples from squamous samples. The three tasks differ in the test data distribution, which results from various types of manipulations:
REGED0: No manipulation (distribution identical to the training data).
REGED1: The following variables are manipulated:
20, 27, 36, 70, 82, 83, 85, 91, 118, 125, 139, 143, 160, 169, 176, 185, 191, 204, 219, 224, 229, 239, 243, 251, 252, 269, 281, 282, 295, 297, 301, 319, 320, 321, 342, 350, 357, 359, 361, 378, 387, 407, 409, 412, 429, 430, 469, 472, 499, 501, 507, 512, 540, 545, 552, 561, 566, 572, 580, 586, 593, 618, 622, 637, 651, 663, 674, 681, 683, 686, 690, 702, 727, 754, 762, 764, 773, 786, 805, 815, 835, 861, 872, 873, 877, 880, 889, 904, 935, 936, 939, 942, 949, 962, 977, 985, 989, 991, 992, 994.
REGED2: Many variables are manipulated, including all the consequences of the target. When a manipulation is performed, the values of the manipulated variables are clamped to given values by an "external agent". All other variable values are obtained after the system stabilizes when it is let to evolve according to its own dynamics.

2) Sources

a. Original owners

Alexander Statnikov and Constantin F. Aliferis

References:

- Aliferis, C.F. and Statnikov, A. (2007) High-Fidelity Resimulations from High-Throughput Data. *Technical Report DSL 07-03, Discovery Systems Laboratory, Vanderbilt University*.

b. Donor of database

This version of the database was prepared for the WCCI2008 by the Causality Workbench team.

c. Date prepared: Summer 2007.

d. Date released for the challenge: December 2007.

3) Past usage

Used for the two first challenges organized by the Causality Workbench Team: (1) the Causation and Prediction Challenge (WCCI 2008), (2) the NIPS 2008 Pot-Luck challenge, as part of the LOCANET task (see http://clopinet.com/causality).

4) Experimental design

1. Creation of a resimulated gene expression dataset that is modeled closely after the real microarray gene expression data

The ability to produce realistic simulated data is a critical component of evaluating discovery algorithms in a systematic manner. In machine learning, a standard practice is to use expert-derived networks (a prototypical example being the ALARM network [1]). Such networks do not correspond to biological systems, they are too small, and the distributions are highly discrete. In bioinformatics, researchers have simulated data from ad-hoc unvalidated generating models, or from validated but very small models (e.g., http://bioinformatics.upmc.edu/GE2/, http://www.phil.cmu.edu/projects/tetrad/, [2]). In order to obtain large, realistic networks and data capturing the characteristics of human gene expression data, we applied a High-Fidelity Data Resimulation technique [3] that generates synthetic data from a causal process that is induced from the real data and guarantees that the synthetic data is non-distinguishable from the real data.

The High-Fidelity Data Resimulation method was applied to 1,000 randomly selected variables (999 oligonucleotide probes and one phenotype variable) from the 12,600 probes in the Affymetrix U95A array lung cancer gene expression data of [4]. Once a network (*REGED0*) was obtained by HITON-Bach algorithm, a set of 30,000 samples was generated from this network. The area under the ROC curve (AUC) for discriminating the real from the synthetic data (i.e., joint real and synthetic distributions of the 1,000 variables) indicated minor discrepancies between the two distributions.

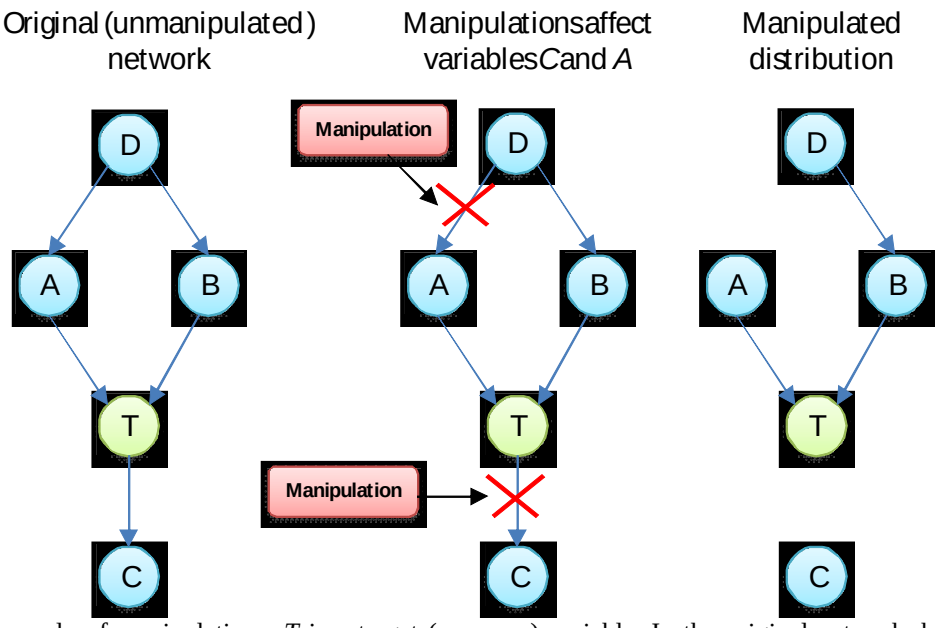


Figure 1. Example of manipulations. *T* is a target (response) variable. In the original network, both *A* (direct cause of *T*) and *C* (direct effect of *T*) are predictive of *T*. However, once *C* and *A* are manipulated, only *A* remains predictive of *T* in the manipulated network.

2. Creation of additional re-simulated gene expression datasets with manipulations

The datasets with manipulations are generated from the original network (*REGED0*) by disconnecting some variables from their direct causes (see Figure 1). After the manipulations were performed in the network graph, the network was re-parameterized accordingly and data (30,000 samples) was generated from it as described in the previous section. We produced two datasets with manipulations:

- *REGED1*: We manipulated 1 direct cause of the target, 5 its mostly predictive direct effects, and 94 randomly selected variables that are scattered throughout the network but do not belong to the local neighborhood of target. In total, 100 variables were manipulated.
- *REGED2*: We manipulated 1 direct cause of the target, all (13) its direct effects, and 86 randomly selected variables that are scattered throughout the network but do not belong to the local neighborhood of target. In total, 100 variables were manipulated.

For *REGED1*, the Challenge provides a list of variables that were manipulated but does not disclose what these variables are (e.g., direct causes, direct consequences, or neither). Since participants have access only to unmanipulated training data (e.g., *REGED0*), the optimal prediction for the target in *REGED1* can be achieved by using a Markov blanket of the target inferred in *REGED0* and excluding all manipulated direct effects and their direct causes.

For *REGED2*, the Challenge mentions that many variables including all direct effects of the target were manipulated but does not provide indices of these variables in the dataset (unlike *REGED1*). Since participants have access only to unmanipulated training data, the optimal solution for *REGED2* is to use only direct causes of the target variable.

3. Structure details:

- Local neighborhood of *T* contains 2 direct causes and 13 direct effects; there are also 6 spouses of *T*.
- Out of 999 non-target variables in the network, 789 are connected to *T* by an undirected path and 210 are not.
- Local structure in the natural distribtution (all numbers below refer to column indices in the 2D-array data.)
    - Direct causes: {322, 931}
    - Direct effects: {84, 252, 345, 410, 426, 454, 572, 594, 595, 740, 818, 826, 940}
    - Local neighborhood: {84, 252, 322, 345, 410, 426, 454, 572, 594, 595, 740, 818, 826, 931, 940}

Note: the variable numbers are offset by 1 compared to the column number c in the data tables. The target is numbered 1. All other variables are numbered c+1.

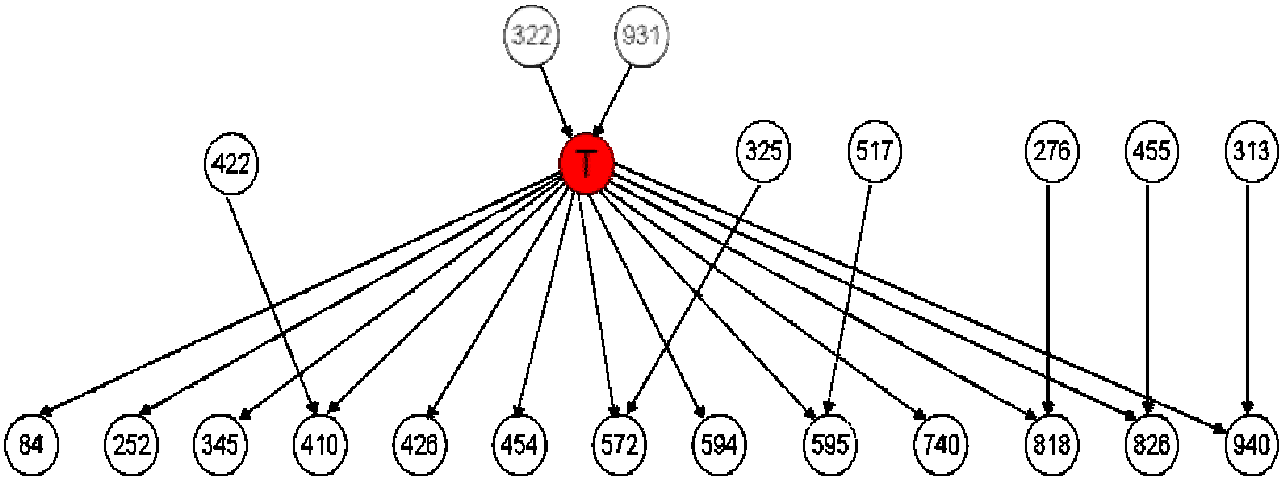


Figure 2: Local neighborhood of *T* in *REGED* network.

Reference List

1. Beinlich I, Suermondt HJ, Chavez R, Cooper G: **The ALARM monitoring system: A case study with two probabilistic inference techniques for belief networks**. *Proceedings of the Second European Conference on Artificial Intelligence in Medicine* 1989.
2. Basso K, Margolin AA, Stolovitzky G, Klein U, la-Favera R, Califano A: **Reverse engineering of regulatory networks in human B cells**. *Nat Genet* 2005, **37:**382-390.

3. Aliferis CF, Statnikov A: **High-Fidelity Resimulation from High-Throughput Data**. *Technical Report DSL 07-03* 2007.
4. Bhattacharjee A, Richards WG, Staunton J, Li C, Monti S, Vasa P, Ladd C, Beheshti J, Bueno R, Gillette M, Loda M, Weber G, Mark EJ, Lander ES, Wong W, Johnson BE, Golub TR, Sugarbaker DJ, Meyerson M: **Classification of human lung carcinomas by mRNA expression profiling reveals distinct adenocarcinoma subclasses**. *Proc Natl Acad Sci U S A* 2001, **98:**13790-13795.

5) Number of examples and class distribution

REGED0	Positive ex.	Negative ex.	Total	Check sum
Training set	59	441	500	167114863.00
Test set	1853	18147	20000	6678340769.00
All	1912	18588	20500	6845455632.00

REGED1	Positive ex.	Negative ex.	Total	Check sum
Training set	59	441	500	167114863.00
Test set	1833	18167	20000	6657027579.00
All	1892	18608	20500	6824142442.00

REGED2	Positive ex.	Negative ex.	Total	Check sum
Training set	59	441	500	167114863.00
Test set	1781	18219	20000	6658529997.00
All	1840	18660	20500	6825644860.00

Note: the training set is the same for all three datasets.

6) Type of input variables and variable statistics

Artificial variables	Random probes	Total
999	0	999

All variables are **integer** quantized on 1000 levels. There are **no missing values**.

7) Results of baseline methods

Several quality assurance tests were performed with datasets in the Challenge. The following two goals were pursued in these tests:

1. Ensure sufficient strength of predictive signal.
2. Assess effectiveness and impact of manipulations on predictivity of the target in manipulated datasets.

We analyzed classification performance of different feature subsets. Namely, we analyzed difference in predictivity in manipulated data when using Markov blanket from manipulated network and Markov blanket from unmanipulated network. Also, we compared to using no variable selection combined with a powerful regularized classifier (e.g., SVMs). In addition, we executed several state-of-the-art causal discovery algorithms to obtain baseline results for the Challenge datasets:

a. HITON-PC (semi-interleaved, without symmetry correction)
b. HITON-PC (semi-interleaved, without symmetry correction) with FDR prefiltering of variables

The reference submission uploaded to the website of the challenge. (reference_hpc) is for HITON-PC (max-k=3, alpha=0.05) & Linear SVMs (C=0.001) using original data for both algorithms. The test AUCs are:
REGED0 0.9998
REGED1 0.9800
REGED2 0.8447
The best AUC predictions of challenge participants are:
REGED0 1.000
REGED1 0.989
REGED2 0.839
The best score of participants on the LOCANET task is: 0.22.
For reference, we also trained a linear SVM classifier using various feature subsets derived from the truth values of the causal relationships:

Feature subsets

		1	2	3	4	5
REGED0	1	Parents (N=2)	Children (N=13)	Parents + Children (N=15)	Markov Blanket (N=21)	All (N=999)
REGED1	2	Parents (N=2)	Children (N=13)	Parents + Children (N=15)	Markov Blanket (N=21)	All (N=999)
REGED2	3	Parents (N=2)	Children (N=13)	Parents + Children (N=15)	Markov Blanket (N=21)	All (N=999)
REGED1	4	Parents (N=2)	unmanipulated Children (N=8)	Parents + unmanipulated Children (N=10)	Markov Blanket - manipulated Children and their Parents (N=14)	All (N=999)
REGED2	5	Parents (N=2)	unmanipulated Children (N=0)	Parents + unmanipulated Children (N=2)	Markov Blanket - manipulated Children and their Parents (N=2)	All (N=999)

(a)

AUC results (subsets of features of the previous table)

		1	2	3	4	5
REGED0	1	0.9411	0.9994	0.9998	0.9998	0.9946
REGED1	2	0.9265	0.9154	0.9817	0.9841	0.9344
REGED2	3	0.9343	0.4952	0.8426	0.8177	0.7254
REGED1	4	0.9265	0.9788	0.9941	0.9956	0.9344
REGED2	5	0.9343	0.5	0.9343	0.9343	0.7254

(b)

Table A1: Results obtained with subsets of features derived from the true causal relationships. (a) Subsets chosen. (b) AUC obtained with a linear SVM.

Dataset B: SIDO

1) Topic

SIDO stands for **SI**mple **D**rug **O**peration mechanisms. The SIDO dataset contains descriptors of molecules, which have been tested against the AIDS HIV virus. The target values indicate the molecular activity (+1 active, -1 inactive). The causal discovery task is to uncover causes of molecular activity among the molecule descriptors. This would help chemists in the design of new compounds, retaining activity, but having perhaps other desirable properties (less toxic, easier to administer). The molecular descriptors were generated programmatically from the three dimensional description of the molecule, with several programs used by pharmaceutical companies for QSAR studies (Quantitative Structure-Activity Relationship). For example, a descriptor may be the number of carbon molecules, the presence of an aliphatic cycle, the length of the longest saturated chain, etc. The dataset includes 4932 variables (other than the target), which are either molecule descriptors (all potential causes of the target) or "probes" (artificially generated variables, which are not causes of the target). The training set and the unmanipulated test set are similarly distributed. They are constructed such that some of the "probes" are effects (consequences) of the target and/or of other real variables, and some are unrelated to the target or other real variables. Hence, both in the training set and the unmanipulated test set, all the probes are non-causes of the target, yet some of them may be predictive of the target. In the manipulated test set, all the "probes" are "manipulated" in every sample by an "external agent" (*i.e.* set to given values, not affected by the dynamics of the system) and can therefore not be relied upon to predict the target. The identity of the probes was concealed during the challenge. The probes were used to assess the effectiveness of the algorithms to dismiss non-causes of the target for making predictions in manipulated test data.

2) Sources

a. Original owners

The data was made available by the National Cancer Institute (NCI), via the DTP AIDS Antiviral Screen program at: http://dtp.nci.nih.gov/docs/aids/aids_data.html. The DTP AIDS Antiviral Screen has checked tens of thousands of compounds for evidence of anti-HIV activity. Available are screening results and chemical structural data on compounds that are not covered by a confidentiality agreement.

b. Donor of database

This version of the database was prepared for the WCCI2008 by the Causality Workbench team.

c. Date prepared: Fall 2007.

d. Date released for the challenge: December 2007.

3) Past usage

Another version of this dataset was used under the name HIVA for past challenges (the WCCI06 challenge on performance predictions, the NIPS08 model selection game and the IJCNN 07 “agnostic learning vs. prior knowledge” challenge. See http://clopinet.com/challenges/ for details). SIDO uses the same original data, but

differently formatted and split. SIDO was used for the two first challenges organized by the Causality Workbench Team: (1) the Causation and Prediction Challenge (WCCI 2008), (2) the NIPS 2008 Pot-Luck challenge, as part of the LOCANET task (see http://clopinet.com/causality).

4) Experimental design

We describe first the raw data and the preprocessing we made (similar for HIVA and SIDO). We then describe the process used to add artificial variables (probes).

1. Preprocessing

The screening results of the May 2004 release containing the screening results for 43,850 compounds were used. The results of the screening tests are evaluated and placed in one of three categories:

- **CA** - Confirmed active
- **CM** - Confirmed moderately active
- **CI** - Confirmed inactive

We converted this into a two-class classification problem: Inactive (CI) vs. Active (CA or CM.)

Chemical structural data for 42,390 compounds was obtained from the web page. It was converted to structural features by three different methods, yielding three feature sets that were concatenated. We matched the compounds in the structural description files and those in the compound activity file, using the NSC id number. We ended up with 42678 examples.

First feature set (the same as the one used for HIVA):

The program ChemTK version 4.1.1, Sage Informatics LLC was used to generate features. (Appemdix C).

Reference:

Miller, D.W. A Chemical Class-Based Approach to Predictive Model Generation. J. Chem. Inf. Comput. Sci. 2003, 43, 568-578
http://www.ncbi.nlm.nih.gov/pubmed/12653523

The **1617** features of the original HIVA dataset were included in SIDO; they are all binary:

- unbranched_fragments: 750 features
- pharmacophores: 495 features
- branched_fragments: 219 features
- internal_fingerprints: 132 features
- ring_systems: 21 features

Only binary features having a total number of ones larger than 100 (>400 for unbranched fragments) and at least 2% of ones in the positive class were retained. In all cases, the default program settings were used to generate keys (except for the pharmacophores for which “max number of pharmacophore points” was set to 4 instead of 3; the pharmacophore keys for Hacc, Hdon, ExtRing, ExtArom, ExtAliph were generated, as well as those for Hacc, Hdon, Neg, Pos.) The keys were then converted to attributes.

We briefly describe the attributes/features:
Branched fragments: each fragment is constructed through an "assembly" of shortest-path unbranched fragments, where each of the latter is required to be bounded by two atoms belonging to one or more pre-defined "terminal-atom".
Unbranched fragments: unique non-branching fragments contained in the set of input molecules.
Ring systems: A ring system is defined as any number of single or fused rings connected by an unbroken chain of atoms. The simplest example would be either a single ring (e.g., benzene) or a single fused system (e.g., naphthalene).
Pharmacophores: ChemTK uses a type of pharmacophore that measures distance via bond connectivity rather than a typical three-dimensional distance. For instance, to describe a hydrogen-bond acceptor and hydrogen-bond donor separated by five connecting bonds, the corresponding key string would be "HAcc.HDon.5". The pharmacophores were generated from the following features:
Neg. Explicit negative charge.
Pos. Explicit positive charge.
HAcc. Hydrogen-bond acceptor.
HDon. Hydrogen-bond donor.
ExtRing. Ring atom having a neighbor atom external to the ring.
ExtArom. Aromatic ring atom having a neighbor atom external to the ring.
ExtAliph. Aliphatic ring atom having a neighbor atom external to the ring.
Internal fingerprints: small, fixed catalog of pre-defined queries roughly similar to the MACCS key set developed by MDL.

Second feature set (new in SIDO)
Hans Bitter (Roche Palo Alto) kindly provided us with features computed with the Chemical Computing Group software (Appendix D). We retained **13** binary features:

FFType_ang
opr_leadlike
Q_VSA_FPOS
FFType_oop
lip_druglike
FFType_bond
FFType_atom
C2
reactive
C1
FFType_all
Q_VSA_FHYD
FFType_tor

He also provided us with features computed with the program ISIS We retained **11** binary ISIS features:

MACCS(145)
MACCS(--6)
MACCS(125)
MACCS(120)
MACCS(--5)
MACCS(--4)
MACCS(162)
MACCS(-49)
MACCS(-10)
MACCS(136)
MACCS(166)

MACCS stands for Molecuar ACCess System. The MACCS keys are a set of questions about a chemical structure. Here are some of the questions:

- Are there fewer than 3 oxygens?
- Is there a S-S bond?
- Is there a ring of size 4?
- Is at least one F, Cl, Br, or I present?

Third feature set (new in SIDO)
Georg Wichard (`Institute of Molecular Pharmacology, Germany`) kindly provided us with several feature sets.
Those include:

- "Ghose-Crippen" Descriptors [**Prediction of Hydrophobic (Lipophilic) Properties of Small Organic Molecules Using Fragmental Methods: An Analysis of ALOGP and CLOGP Methods Arup K. Ghose,* Vellarkad N. Viswanadhan, and John J. Wendoloski**, *J. Phys. Chem. A* **1998,** *102,* 3762-3772. We retained only **one** binary feature ALogP_Count[98] (I attached to C^3_{sp3}).
- FMP features (we retained **2** binary features: ES_Count_ssNH2 and ES_Count_ssPH)

2. Adding artificial variables (probes)
The motivation for adding artificial variables is that the truth values of the causal relationships between the real variables are not known. Compared to purely artificially generated data, using real variables allows us to work on realistic data distributions. The added artificial variables allow us to assess the performances of causal discovery algorithms.

The target variable is a real variable. Consequently, no artificial variable may be a cause of the target (direct or indirect). The artificial variables are constructed as functions (plus noise) of subsets of real variables (which may include the target) and other artificial variables. Some artificial variables are generated randomly (hence have no dependency with the real variables).

The method used for generating random probes is described in Appendix A. The specific parameters used for SIDO are found below:

```
    n=size(X,2); % Number of true variables
    nc=round(n/2); % Number of confounder probes
    ne=round(n); % Number of effect probes
    np=nc; % Number of truly random probes
    tpnc = 3; % Number of parent true variables for confounders
    ppnc = 2; % Number of parent, which are noise, for confounders
    tpne = 2; % Number of parent confounder variables for effects
    ppne = 2; % Number of parent, which are noise, for effects
    nlval=2; % non-linearity level 2
    noise=0.05; % random noise level (fraction of output range)
    top_num=50; % number of top ranking causes kept
    noise_Y=0.1;
    num_manip=0;
    [X, parents]=add_probes(X, Y, np, nc, ne, tpnc, ppnc, tpne, ppne,
nlval, noise, num_manip, top_num, noise_Y);
```

Note: the probes are first created unmanipulated for SIDO0 and then manipulated according to the methods described in Appendix A.

We summarize the statistics of the probes added:

```
== Total number of variables: 4932 ==
## Real variables (1644):
## Probes:
== Random (822):
  205 spouses of true var:
  205 spouses of target:
  412 independent of target: Warning, some probes assigned to be
spouses are unused
== Confounders (822): Warning, wierd set
  3.49+- 0.98 true variable parents,  2.50+- 0.50 parents unrelated to
target
== Effects (1644): Warning, wierd set
  2.51+- 0.50 confounder parents,  2.50+ 0.50 parents unrelated to
target
```

5) Number of examples and class distribution

SIDO0	Positive ex.	Negative ex.	Total	Check sum
Training set	452	12226	12678	6155080
Test set	351	9649	10000	4863127
All	803	21875	22678	11018207

SIDO1	Positive ex.	Negative ex.	Total	Check sum
Training set	452	12226	12678	6155080
Test set	335	9665	10000	4879177
All	787	21891	22678	11034257

SIDO2	Positive ex.	Negative ex.	Total	Check sum
Training set	452	12226	12678	6155080
Test set	365	9635	10000	4865938
All	817	21861	22678	11021018

Note: the training set is the same for all three datasets.

6) Type of input variables and variable statistics

Real variables	Random probes	Total
1644	3288	4932

All variables are **binary**. There are **no missing values**.

7) Results of baseline methods

Prior to releasing the data, we performed experiments with various causal discovery algorithms to assess the dataset and adjust the probe generation to a proper level of difficulty. We show below the experiments with the last version of the probe generation algorithm, which we ended up using. The final dataset that was released contains fewer variables because we removed at the last minute a few non-binary variables, which were accidentally left out and slightly reduced the number of probe. The systematic test were not re-run. However, several baseline results were uploaded to the web site of the challenge.

Select features from **natural distribution with probes**
Estimate classification performance in **manipulated distribution with probes**

Variable subset	Classification AUC	# of selected variables	# of selected real features	# of selected probes	# of selected probes that are children of the target
PC1	0.5771	259	75	184	119
PC1, real features only	0.6994	75	75	0	0
PC1, probes only	0.4553	184	0	184	119
PC2	0.6144	367	116	251	157
PC2, real features only	0.6296	116	116	0	0
PC2, probes only	0.4458	251	0	251	157
PC3	0.5989	25	12	13	8
PC3, real features only	0.4264	12	12	0	0
PC3, probes only	0.4166	13	0	13	8
PC4	0.5998	41	16	25	19
PC4, real features only	0.4621	16	16	0	0
PC4, probes only	0.4208	25	0	25	19
PC5	0.5421	10	4	6	3
PC5, real features only	0.5000	4	4	0	0

PC5, probes only	0.4135	6	0	6	3
PC6	0.6068	25	10	15	15
PC6, real features only	0.4074	10	10	0	0
PC6, probes only	0.4453	15	0	15	15
PC-FDR1	0.5867	230	69	161	96
PC-FDR1, real features only	0.6709	69	69	0	0
PC-FDR1, probes only	0.4480	161	0	161	96
PC-FDR2	0.5995	338	108	230	141
PC-FDR2, real features only	0.6107	108	108	0	0
PC-FDR2, probes only	0.4552	230	0	230	141
PC-FDR3	0.6038	28	12	16	11
PC-FDR3, real features only	0.5570	12	12	0	0
PC-FDR3, probes only	0.4238	16	0	16	11
PC-FDR4	0.6027	47	19	28	22
PC-FDR4, real features only	0.6448	19	19	0	0
PC-FDR4, probes only	0.4258	28	0	28	22
PC-FDR5	0.6240	12	9	3	2
PC-FDR5, real features only	0.5742	9	9	0	0
PC-FDR5, probes only	0.5000	3	0	3	2
PC-FDR6	0.6079	20	12	8	6
PC-FDR6, real features only	0.5407	12	12	0	0
PC-FDR6, probes only	0.4284	8	0	8	6

PC1: HITON-PC with alpha = 0.01, max-k =1
PC2: HITON-PC with alpha = 0.05, max-k =1
PC3: HITON-PC with alpha = 0.01, max-k =2
PC4: HITON-PC with alpha = 0.05, max-k =2
PC5: HITON-PC with alpha = 0.01, max-k =3
PC6: HITON-PC with alpha = 0.05, max-k =3
Same numbering scheme applies to PC-FDR methods. Benjamnin FDR prefiltering was applied prior to running HITON-PC.
We used G2 test for all versions of HITON-PC.

The reference submission uploaded to the website of the challenge. (reference_hpc) is for HITON-PC (max-k=1, alpha=0.01) & Linear SVM (C=1), using original (binary) data for both algorithms. HITON-PC uses first 5000 samples in the training data:
SIDO0 0.9377
SIDO1 0.6825
SIDO2 0.6174

The best AUC results of the challenge participants are:
SIDO0 0.944
SIDO1 0.753
SIDO2 0.668

The best score on the LOCANET task is: 3.31

Dataset C: CINA

1) Topic

CINA (**C**ensus **I**s **N**ot **A**dult) is derived from census data (the UCI machine-learning repository Adult database). The data consists of census records for a number of individuals. The causal discovery task is to uncover the socio-economic factors affecting high income (the target value indicates whether the income exceeds 50K). The 14 original attributes (features) including age, workclass, education, education, marital status, occupation, native country, etc. have been coded to eliminate categorical variables. Distractor features (artificially generated variables, which are not causes of the target) were added. In training data, some of these distractors are effects (consequences) of the target and/or of other real variables. Some are unrelated to the target or other real variables. Hence, some of the distractors may be correlated to the target in training data, although they do not cause it. The unmanipulated test data are distributed like the training data. Hence both causes and consequences of the target my be predictive in the unmanipulated test data. In contrast, in the manipulated test data, all the distractors are "manipulated" by an "external agent" (*i.e.* set to given value, not affected by the dynamics of the system) and are therefore they cannot be relied upon to predict the target.

2) Sources

a. Original owners

This data was extracted from the census bureau database found at
http://www.census.gov/ftp/pub/DES/www/welcome.html
Donor: Ronny Kohavi and Barry Becker,
Data Mining and Visualization
Silicon Graphics.
e-mail: ronnyk@sgi.com for questions.

The information below is exerpted from the UCI machine learning repository:

Extraction was done by Barry Becker from the 1994 Census database. The prediction task is to determine whether a person makes over 50K a year. The attributes are:
age: continuous.
workclass: Private, Self-emp-not-inc, Self-emp-inc, Federal-gov, Local-gov, State-gov, Without-pay, Never-worked.
fnlwgt: continuous.
education: Bachelors, Some-college, 11th, HS-grad, Prof-school, Assoc-acdm, Assoc-voc, 9th, 7th-8th, 12th, Masters, 1st-4th, 10th, Doctorate, 5th-6th, Preschool.
education-num: continuous.
marital-status: Married-civ-spouse, Divorced, Never-married, Separated, Widowed, Married-spouse-absent, Married-AF-spouse.
occupation: Tech-support, Craft-repair, Other-service, Sales, Exec-managerial, Prof-specialty, Handlers-cleaners, Machine-op-inspct, Adm-clerical, Farming-fishing, Transport-moving, Priv-house-serv, Protective-serv, Armed-Forces.
relationship: Wife, Own-child, Husband, Not-in-family, Other-relative, Unmarried.
race: White, Asian-Pac-Islander, Amer-Indian-Eskimo, Other, Black.
sex: Female, Male.
capital-gain: continuous.
capital-loss: continuous.
hours-per-week: continuous.
native-country: United-States, Cambodia, England, Puerto-Rico, Canada, Germany, Outlying-US(Guam-USVI-etc), India, Japan, Greece, South, China, Cuba, Iran, Honduras, Philippines, Italy, Poland, Jamaica, Vietnam, Mexico, Portugal, Ireland, France, Dominican-Republic, Laos, Ecuador, Taiwan, Haiti, Columbia, Hungary, Guatemala, Nicaragua, Scotland, Thailand, Yugoslavia, El-Salvador, Trinadad&Tobago, Peru, Hong, Holand-Netherlands.
income: >50K, <=50K.

```
Split into train-test using MLC++ GenCVFiles (2/3, 1/3 random).
 48842 instances, mix of continuous and discrete    (train=32561, test=16281)
 45222 if instances with unknown values are removed (train=30162, test=15060)
 Duplicate or conflicting instances : 6
 Class probabilities for adult.all file
 Probability for the label '>50K'  : 23.93% / 24.78% (without unknowns)
 Probability for the label '<=50K' : 76.07% / 75.22% (without unknowns)

 Description of fnlwgt (final weight)
 The weights on the CPS files are controlled to independent estimates of the
 civilian noninstitutional population of the US.  These are prepared monthly
 for us by Population Division here at the Census Bureau.  We use 3 sets of
 controls. People with similar demographic characteristics should have
 similar weights.
```

a. Donor of database

A first version of the database was prepared for the WCCI 2006 performance prediction challenge by Isabelle Guyon, 955 Creston Road, Berkeley, CA 94708, USA (isabelle@clopinet.com) under the name ADA. CINA resembles ADA, except that only binary variables were retained and the data were reshuffled.
The present version of CINA was prepared for the causation and prediction challenge by the Causality Workbench Team.

b. Date released for the challenge: January 2008.

3) Past usage

```
 First cited in:
 @inproceedings{kohavi-nbtree,
    author={Ron Kohavi},
    title={Scaling Up the Accuracy of Naive-Bayes Classifiers: a
           Decision-Tree Hybrid},
    booktitle={Proceedings of the Second International Conference on
               Knowledge Discovery and Data Mining},
    year = 1996}
 Error Accuracy reported as follows, after removal of unknowns from
    train/test sets):
    C4.5       : 84.46+-0.30
    Naive-Bayes: 83.88+-0.30
    NBTree     : 85.90+-0.28
 The following algorithms were later run with the following error rates,
    all after removal of unknowns and using the original train/test split.
    All these numbers are straight runs using MLC++ with default values.

    Algorithm              Error
 -- ----------------       -----
 1  C4.5                   15.54
 2  C4.5-auto              14.46
 3  C4.5 rules             14.94
 4  Voted ID3 (0.6)        15.64
 5  Voted ID3 (0.8)        16.47
 6  T2                     16.84
 7  1R                     19.54
 8  NBTree                 14.10
 9  CN2                    16.00
 10 HOODG                  14.82
 11 FSS Naive Bayes        14.05
 12 IDTM (Decision table)  14.46
 13 Naive-Bayes            16.12
 14 Nearest-neighbor (1)   21.42
 15 Nearest-neighbor (3)   20.35
 16 OC1                    15.04
 17 Pebls                  Crashed.  Unknown why (bounds WERE increased)
```

Note: The performances reported are error rates, not BER. We tried to reproduce these performances and obtained 15.62% error with a linear ridge regression classifier. The performances slightly degraded when we tried to group features (15.67% when we replace the country code by a binary US/nonUS value and 16.40% with further reduction to 33 features.)
Used under the name ADA for the WCCI 2006 Performance Prediction Challenge, the NIPS 2006 Model Selection game and the IJCNN 2007 Agnostic Learning vs. Prior Knowledge challenge. See http://clopinet.com/challenges.
Used for the two first challenges organized by the Causality Workbench Team: (1) the Causation and Prediction Challenge (WCCI 2008), (2) the NIPS 2008 Pot-Luck challenge, as part of the LOCANET task (see http://clopinet.com/causality).

4) Experimental design

The following steps were performed to create the original ADA dataset:

- Convert the features to 14 numeric values $a \in 1 \ldots n$.
- Convert the numeric values to binary codes (a vector of n zeros with value one at the a^{th} position. This results in 88 features. The missing values get an all zero vector.
- Downsize the number of features to 48 by replacing the country code by a binary US/nonUS feature.
- Randomize the feature and pattern order.
- Remove the entries with missing values for workclass.

Table C.1. Features of the ADA datasets.

Feature name	**min**	**max**	**numval**	**comments**
Age	0.19	1	continuous	No missing value.
workclass_Private	0	1	2	2799 missing values (corresponding entries removed.)
workclass_Self_emp_not_inc	0	1	2	
workclass_Self_emp_inc	0	1	2	
workclass_Federal_gov	0	1	2	
workclass_Local_gov	0	1	2	
workclass_State_gov	0	1	2	
workclass_Without_pay	0	1	2	
workclass_Never_worked	0	1	2	
Fnlwgt	0.008	1	continuous	No missing value.
EducationNum	0.06	1	16	Number corresponding to 16 discrete levels of education
maritalStatus_Married_civ_spouse	0	1	2	No missing value.
maritalStatus_Divorced	0	1	2	
maritalStatus_Never_married	0	1	2	
maritalStatus_Separated	0	1	2	
maritalStatus_Widowed	0	1	2	
maritalStatus_Married_spouse_absent	0	1	2	
maritalStatus_Married_AF_spouse	0	1	2	
occupation_Tech_support	0	1	2	

occupation_Craft_repair	0	1	2	2809 missing values (corresponding entries removed.)
occupation_Other_service	0	1	2	
occupation_Sales	0	1	2	
occupation_Exec_managerial	0	1	2	
occupation_Prof_specialty	0	1	2	
occupation_Handlers_cleaners	0	1	2	
occupation_Machine_op_inspct	0	1	2	
occupation_Adm_clerical	0	1	2	
occupation_Farming_fishing	0	1	2	
occupation_Transport_moving	0	1	2	
occupation_Priv_house_serv	0	1	2	
occupation_Protective_serv	0	1	2	
occupation_Armed_Forces	0	1	2	
relationship_Wife	0	1	2	No missing value.
relationship_Own_child	0	1	2	
relationship_Husband	0	1	2	
relationship_Not_in_family	0	1	2	
relationship_Other_relative	0	1	2	
relationship_Unmarried	0	1	2	
race_White	0	1	2	No missing value.
race_Asian_Pac_Islander	0	1	2	
race_Amer_Indian_Eskimo	0	1	2	
race_Other	0	1	2	
race_Black	0	1	2	
Sex	0	1	2	0=female, 1=male. No missing value.
CapitalGain	0	1	continuous	No missing value.
CapitalLoss	0	1	continuous	No missing value.
HoursPerWeek	0.01	1	continuous	No missing value.
NativeCountry	0	1	2	0=US, 1=non-US. 857 missing values replaced by 1.

Four features were removed because they were found constant in training data.
We generated the probes according to the method described in Appendix A. Specifically, we used for CINA the following parameters:

```
n=size(X,2); % Number of true variables
nc=round(n/2); % Number of confounder probes
ne=round(n); % Number of effect probes
np=nc; % Number of truly random probes
tpnc = 3; % Number of parent true variables for confounders
ppnc = 2; % Number of parent, which are noise, for confounders
tpne = 2; % Number of parent confounder variables for effects
ppne = 2; % Number of parent, which are noise, for effects
nlval=2; % non-linearity level 2
noise=0.05; % random noise level (fraction of output range)
```

```
    top_num=round(n/2); % number of top ranking causes kept
    noise_Y=0.1;
    X=full(X);
    num_manip=0;
    [X, parents]=add_probes(X, Y, np, nc, ne, tpnc, ppnc, tpne, ppne,
nlval, noise, num_manip, top_num, noise_Y);
```

Note: the probes are first created unmanipulated for CINA0 and then manipulated according to the methods described in Appendix A.

The statistics on the generated probes are found below:

```
== Total number of variables: 132 ==
## Real variables (44): 1 ... 44
## Probes:
== Random (22):
  5 spouses of true var:
  5 spouses of target:
  12 independent of target: 121 ... 132
== Confounders (22): 50 ... 71
  3.14+- 0.83 true variable parents,  2.32+- 0.48 parents unrelated to
target
== Effects (44): 77 ... 120
  2.48+- 0.51 confounder parents,  2.55+ 0.50 parents unrelated to
target
```

5) Number of examples and class distribution

CINA0	Positive ex.	Negative ex.	Total	Check sum
Training set	3939	12094	16033	142172387.00
Test set	2425	7575	10000	88827113.00
All	6364	19669	26033	230999500.00

CINA1	Positive ex.	Negative ex.	Total	Check sum
Training set	3939	12094	16033	142172387.00
Test set	2540	7460	10000	88535493.00
All	6479	19554	26033	230707880.00

CINA2	Positive ex.	Negative ex.	Total	Check sum
Training set	3939	12094	16033	142172387.00
Test set	2518	7482	10000	88617057.00
All	6457	19576	26033	230789444.00

Note: the training set is the same for all three datasets.

6) Type of input variables and variable statistics

Real variables	Random probes	Total
44	88	132

The variables are **mixed** (continous and binary). There are **no missing values**.

7) Results of baseline methods on CINA

Cheating ranking of features

First, we created a ranking of features, using the knowledge of the causa relationships. All real variables are tentatively assumed to be parents of the target. The features belonging belonging to the MB of the post-manipulation distribution are ranked first (then sorted by Pearson correlation coefficient), all other features are ranked last (also sorted by Pearson correlation coefficient). We trained classifiers on increasing numbers of features using this ranking. The classifier used is a **linear ridge regression classifier**. DAUC and DBER are AUC and BER (balanced error rate) on training date. TAUC and TBER are performance on test data.

CINA0

Num. Var.	DAUC	DERR	TAUC	TERR
1	0.8891	0.0023	0.8889	0.0029
2	0.9045	0.0028	0.9054	0.0035
4	0.9260	0.0027	0.9260	0.0034
8	0.9300	0.0027	0.9314	0.0034
16	0.9478	0.0027	0.9497	0.0034
32	0.9623	0.0028	0.9632	0.0036
64	0.9661	0.0027	0.9663	0.0034
128	0.9674	0.0027	0.9670	0.0035
132	0.9674	0.0027	0.9670	0.0035

CINA1

Num. Var.	DAUC	DERR	TAUC	TERR
1	0.7556	0.0030	0.7600	0.0037
2	0.7574	0.0030	0.7603	0.0037
4	0.7730	0.0033	0.7726	0.0041
8	0.8691	0.0035	0.8684	0.0044
16	0.8845	0.0034	0.8838	0.0043
32	0.8907	0.0034	0.8900	0.004
64	0.9675	0.0027	0.7937	0.0052
128	0.9674	0.0027	0.7883	0.005
132	0.9674	0.0027	0.7873	0.0049

CINA2

Num. Var.	DAUC	DERR	TAUC	TERR
1	0.7556	0.0030	0.7605	0.0038
2	0.7574	0.0030	0.7605	0.0038
4	0.8558	0.0039	0.8573	0.0049
8	0.8691	0.0035	0.8723	0.0045
16	0.8834	0.0033	0.8848	0.0043
32	0.8909	0.0033	0.8910	0.0042
64	0.9675	0.0027	0.5492	0.0041
128	0.9674	0.0027	0.5483	0.0044
132	0.9674	0.0027	0.5481	0.0043

We then performed experiments with various causal discovery algorithms to select features. A linear SVM with C = 1 is used in these experiments.

Select features from **natural distribution w/o probes**

Estimate classification performance in **natural distribution w/o probes (CINA0)**

Experiment id	Variable subset	Classification AUC	# of selected variables	# of selected real features	# of selected probes	# of selected probes that are children of the target
1	**PC** (HITON-PC)	0.8982	23	23	0	0
2	**PC** (HITON-PC-FDR)	0.8982	23	23	0	0
3	**Parents** (MMHC)	0.8502	6	6	0	0
4	**Children** (MMHC)	0.7514	8	8	0	0
5	**All**	0.8999	44	44	0	0

Select features from **natural distribution with probes**

Estimate classification performance in **natural distribution with probes (CINA0)**

Experiment id	Variable subset	Classification AUC	# of selected variables	# of selected real features	# of selected probes	# of selected probes that are children of the target
6	**PC** (HITON-PC)	0.9721	37	20	17	16
7	**PC** (HITON-PC), real features only	0.8967	20	20	0	0
8	**PC** (HITON-PC), probes only	0.9201	17	0	17	16
9	**PC** (HITON-PC-FDR)	0.9721	37	20	17	16
10	**PC** (HITON-PC-FDR), real features only	0.8967	20	20	0	0
11	**PC** (HITON-PC-FDR), probes only	0.9201	17	0	17	16
12	**Parents** (MMHC)	0.8479	4	4	0	0
13	**Parents** (MMHC), real features only	0.8479	4	4	0	0
14	**Parents** (MMHC), probes only	0.5000	0	0	0	0
15	**Children** (MMHC)	0.9480	15	9	6	5
16	**Children** (MMHC), real features only	0.7743	9	9	0	0
17	**Children** (MMHC), probes only	0.8967	6	0	6	5
18	**All**	0.9728	132	44	88	44
19	**All**, real features only	0.8988	44	44	0	0
20	**All**, probes only	0.9447	88	0	88	44

Select features from **natural distribution with probes**

Estimate classification performance in **manipulated distribution with probes (CINA1)**

Experiment id	Variable subset	Classification AUC	# of selected variables	# of selected real features	# of selected probes	# of selected probes that are children of the target
21	**PC** (HITON-PC)	0.8496	37	20	17	16
22	**PC** (HITON-PC), real features only	0.8982	20	20	0	0
23	**PC** (HITON-PC), probes only	0.5114	17	0	17	16
24	**PC** (HITON-PC-FDR)	0.8496	37	20	17	16
25	**PC** (HITON-PC-FDR), real features only	0.8982	20	20	0	0
26	**PC** (HITON-PC-FDR), probes only	0.5114	17	0	17	16
27	**Parents** (MMHC)	0.8508	4	4	0	0
28	**Parents** (MMHC), real features only	0.8508	4	4	0	0
29	**Parents** (MMHC), probes only	0.5000	0	0	0	0
30	**Children** (MMHC)	0.6558	15	9	6	5
31	**Children** (MMHC), real features only	0.7693	9	9	0	0
32	**Children** (MMHC), probes only	0.5114	6	0	6	5
33	**All**	0.8482	132	44	88	44
34	**All**, real features only	0.8999	44	44	0	0
35	**All**, probes only	0.4987	88	0	88	44

Select features from **natural distribution with probes**

Estimate classification performance in **manipulated distributionl with probes (CINA2)**

Experiment id	Variable subset	Classification AUC	# of selected variables	# of selected real features	# of selected probes	# of selected probes that are children of the target
21	**PC** (HITON-PC)	0.6643	37	20	17	16
22	**PC** (HITON-PC), real features only	0.8985	20	20	0	0
23	**PC** (HITON-PC), probes only	0.3445	17	0	17	16
24	**PC** (HITON-PC-FDR)	0.6643	37	20	17	16
25	**PC** (HITON-PC-FDR), real features only	0.8985	20	20	0	0
26	**PC** (HITON-PC-FDR), probes only	0.3445	17	0	17	16
27	**Parents** (MMHC)	0.8559	4	4	0	0
28	**Parents** (MMHC), real features only	0.8559	4	4	0	0

29	**Parents** (MMHC), probes only	0.5000	0	0	0	0
30	**Children** (MMHC)	0.4603	15	9	6	5
31	**Children** (MMHC), real features only	0.7631	9	9	0	0
32	**Children** (MMHC), probes only	0.3146	6	0	6	5
33	**All**	0.6584	132	44	88	44
34	**All**, real features only	0.9005	44	44	0	0
35	**All**, probes only	0.3257	88	0	88	44

The reference submission uploaded to the website of the challenge. (reference_hpc) is for HITON-PC (max-k=2, alpha=0.01) & Linear SVM (C=1) using original data for SVM and discrete data for HITON-PC. The test AUC results are:
CINA0 0.9721
CINA1 0.8496
CINA2 0.6643

The AUC best results of the challenge participants on CINA are:
CINA0 0.976
CINA1 0.869
CINA2 0.816
The best score of the LOCANET task is 1.70.

Dataset D: MARTI

1) Topic

MARTI stands for **M**easurement **ARTI**fact. MARTI was obtained from the same data generative process as REGED, a source of simulated genomic data, but a noise model was added to simulate the imperfections of the measurement device.

2) Sources

a. Original owners

This is a modified version of REGED (Alexander Statnikov and Constantin F. Aliferis, 2007) created by Isabelle Guyon.

a. Donor of database

This version of the database was prepared for the WCCI2008 by the Causality Workbench team.

b. Date prepared: Fall 2007.

c. Date released for the challenge: January 2008.

3) Past usage

Used for the two first challenges organized by the Causality Workbench Team: (1) the Causation and Prediction Challenge (WCCI 2008), (2) the NIPS 2008 Pot-Luck challenge, as part of the LOCANET task (see http://clopinet.com/causality).

4) Experimental design

The goal of MARTI, like REGED, is to find genes, which could be responsible of lung cancer. The target variable is binary; it separates malignant samples (adenocarcinoma) from control samples (squamous). The feature values representing measurements of gene expression levels are assumed to have been recorded from a two-dimensional microarray 32x32. The training set was perturbed by a zero-mean correlated noise model (neighboring values in one array are generally similarly affected, but the noise pattern is different in every training example).

The test sets have no added noise. This situation simulates a case where we would be using different instruments at "training time" and "test time", e.g. we would use DNA microarrays to collect training data and PCR for testing. We avoided adding noise to the test set because it would be too difficult to filter it without visualizing the test data or computing statistics on the test data, which we forbid. So the scenario is that the second instrument (used at test time) is more accurate. In practice, the measurements would also probably be more expensive, so part of the goals of training would be to reduce the size of the feature set (we are not making this a requirement in this first challenge).

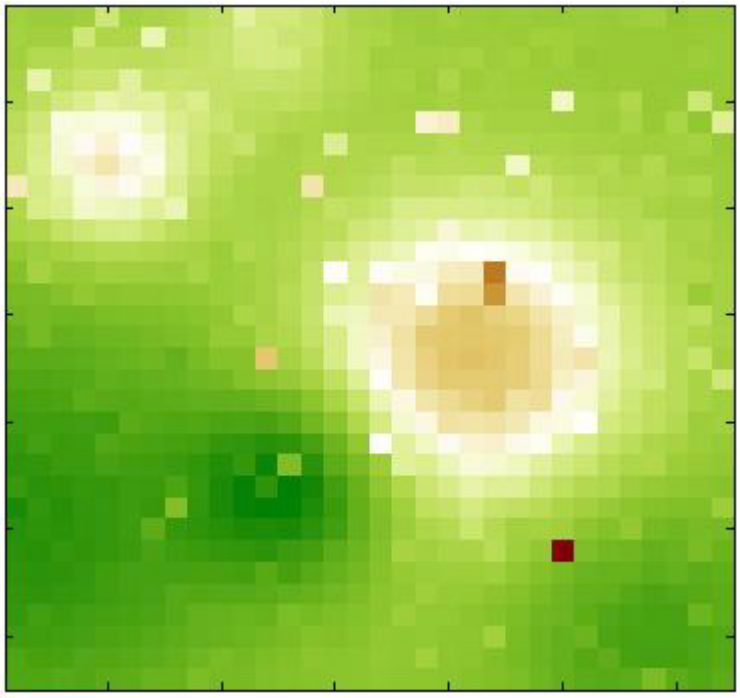


Figure 7: Example of MARTI simulated micro-array

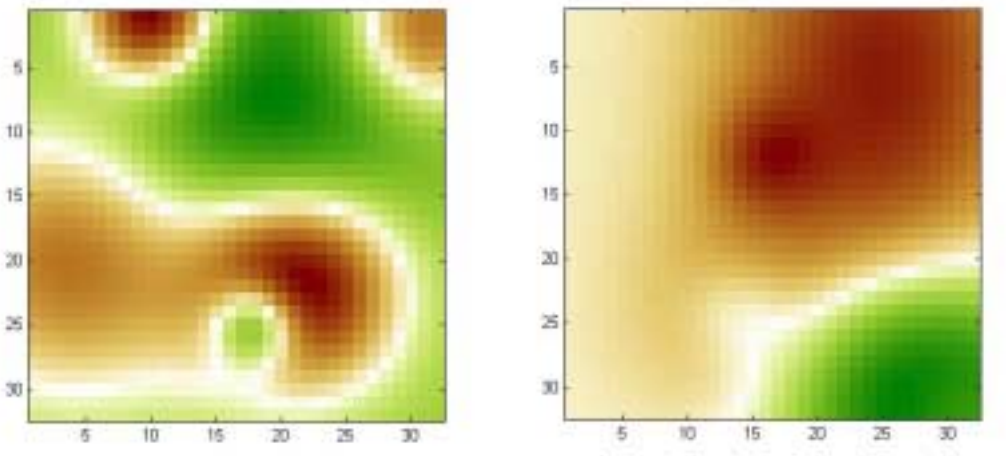
Technical details:
- The features/variables are randomly arranged in a 2d array 32x32. Variables 1:32 form the first column, 33:64 the second, etc.
- To obtain 1024 features, the 999 features of REGED are complemented by 25 "calibrant features", which have value zero plus a small amount of Gaussian noise. The calibrants are spread regularly accross the array and have variable indices 34 44 54 64 199 209 219 354 364 374 384 519 529 539 674 684 694 704 839 849 859 994 1004 1014 1024.
- Like for REGED, we proposed 3 tasks MARTI0, MARTI1, and MARTI2, all having the same training set of 500 examples (from the "unmanipulated distribution"), and different test sets of 20000 examples.
- Like for REGED, the three tasks differ in the test data distribution, which results from various types of manipulations:

MARTI0: No manipulation (distribution identical to the training data).
MARTI1: The following variables are manipulated:
5, 19, 27, 35, 37, 42, 49, 67, 70, 71, 102, 137, 144, 145, 153, 158, 185, 188, 194, 221, 225, 229, 232, 235, 244, 268, 273, 284, 294, 295, 305, 310, 331, 356, 368, 379, 385, 396, 398, 404, 411, 412, 413, 417, 425, 430, 455, 479, 481, 482, 491, 492, 509, 510, 550, 553, 555, 603, 609, 627, 642, 646, 654, 679, 682, 706, 736, 744, 755, 761, 763, 771, 807, 809, 812, 821, 853, 869, 870, 872, 888, 894, 895, 906, 914, 918, 926, 931, 932, 941, 963, 973, 978, 979, 986, 988, 990, 1001, 1010, 1017.
MARTI2: Many variables are manipulated, including all the consequences of the target.

Filtering the noise and/or taking into account the geomety of the array should be necessary to obtain good results.

In MARTI, what does it mean that "the noise pattern is different in every training example"?

Using our noise model, we drew a noise pattern for every example and added it to that example. When the features are arranged in a 2d 32x32 array (as explained in the documentation), the noise pattern has a smooth structure (neighboring coefficients have similar values). This is kind of background with low frequency. A different noise template is added to each example, but all noise templates are all drawn from the same noise model. If you visualize the training examples after rearranging them as a 32x32 array, you will see this right away. For each feature, the expected value of the noise is zero. But the noise of two neighboring features is correlated. We show below examples of noise patterns (positive values in red and negative values in green).



In MARTI, what does it mean that 25 "calibrant features" have value zero plus a small amount of Gaussian noise? The averages for every calibrant feature is far from zero.

We have 2 kinds of noise. The calibrants are 0+-[small Gaussian noise]. Then, on top of that, in training data only, we add the correlated noise model. After we add the correlated noise, because of the small sample size and the large variance, the calibrant values are no longer close to zero (even on average) in training data. However, the median is close zero on average for almost all calibrants, relatively to the signal amplitude: abs(mean(median(X(:,calib))/std(abs(X(:))))~e-005.
In training data, we get: mean(abs(mean(X)))~e+004 but and mean((mean(X(:,calib))))~5e+003. In test data, because we did not add noise, the calibrant values are close to zero, relatively speaking: mean(abs(mean(X)))~5e+003 but mean(abs(mean(X(:,calib))))~1. The calibrants can be used to preprocess the training data by subtracting a bias value after the low frequency noise is removed, so that the calibrant values are zero after preprocessing the training data.

REGED and MARTI do not look like regular microarray data. What kind of normalization did you do?

REGED was obtained by fitting a model to real microrray data. REGED features were shifted and rescaled individually then rounded to integer spanning the range 0:999.

MARTI was obtained from data generated by the same model as REGED without rescaling features individually. For MARTI, a particular type of correlated noise was added. The data were then scaled and quantized globally so the features span -999999:999999.
We chose to make the noise model simple but of high amplitude to make it easy to filter out the noise but hard to ignore it. If you think of the spots on a microarray as an image (MARTI patterns are 32x32 "images"), the noise in MARTI corresponds to patches of more or less intense values, added on top of the original image, representing some kind of slowly varying background. Nowadays, microarray technology has progressed to a point that such heavy backgrounds are not common and occasional contaminated arrays would not pass quality control; furthermore microarray reading software calibrate and normalize data so you would not see data that "bad". But for new instruments under development, such levels of noise are not uncommon.
MARTI illustrates the fact that if you do not take out correlated noise, the result of causal discovery may be severely impaired. Even though the amplitude of the noise is large, the noise is easy to filter out, using the fact that neighboring spots are affected similarly, and using the spots having constant values before noise is added (calibrants). After noise filtering, the residual noise may still impair causal discovery, so it its your challenge to see what can be done to avoid drawing wrong conclusions in the presence of correlated noise.

5) Number of examples and class distribution

MARTI0	Positive ex.	Negative ex.	Total	Check sum
Training set	59	441	500	4824538021.00
Test set	1852	18148	20000	72836533619.00
All	1911	18589	20500	77661071640.00

MARTI1	Positive ex.	Negative ex.	Total	Check sum
Training set	59	441	500	4824538021.00
Test set	1765	18235	20000	72845954638.00
All	1824	18676	20500	77670492659.00

MARTI2	Positive ex.	Negative ex.	Total	Check sum
Training set	59	441	500	4824538021.00
Test set	1662	18338	20000	72914605414.00
All	1721	18779	20500	77739143435.00

Note: the training set is the same for all three datasets.

6) Type of input variables and variable statistics

Artificial variables	Random probes	Total
1024	0	1024

All variables are **integer** quantized on 1000 levels. There are **no missing values**.

7) Results of baseline methods

Below are results for MARTI datasets. All feature sets mentioned in the table below are the true ones (i.e., obtained from the data generating network).
The first and three last columns are obtained by training on the raw training set as provided. We also include for comparison the results on REGED and tests of the unmanipulated data (version 0) when traning with other versions of the training data:

- MARTI00: training data without noise added (should give results similar to REGED).
- MARTI01: like the MARTI0 training set, but after a crude filtering was performed.

Linear SVM with C = 0.001

	REGED0	MARTI00	MARTI01	MARTI0	MARTI1	MARTI2
Parents	0.941	0.936	0.842	0.883	0.854	0.853
Children	0.999	0.999	0.990	0.481	0.436	0.500
PC	1.000	1.000	0.994	0.749	0.872	0.853
MB	1.000	1.000	0.994	0.894	0.460	0.853
All	0.995	0.995	0.982	0.886	0.779	0.731
All \ MB	0.882	0.875	0.799	0.766	0.727	0.707
Calibrators	N/A	0.496	0.499	0.512	0.505	0.512
All \ Calibrators	N/A	0.995	0.982	0.887	0.773	0.724
MB in natural distr.	See results for MB above				0.793	0.746

Linear SVM with C = 1

	REGED0	MARTI00	MARTI01	MARTI0	MARTI1	MARTI2
Parents	0.952	0.947	0.861	0.883	0.855	0.853
Children	0.999	0.999	0.994	0.943	0.702	0.500
PC	1.000	1.000	0.995	0.941	0.866	0.853
MB	1.000	1.000	0.996	0.951	0.756	0.853
All	0.996	0.997	0.985	0.948	0.808	0.730
All \ MB	0.864	0.846	0.782	0.767	0.727	0.692
Calibrators	N/A	0.489	0.499	0.505	0.485	0.497
All \ Calibrators	N/A	0.997	0.985	0.948	0.808	0.730
MB in natural distr.	See results for MB above				0.865	0.614

Polynomial SVM optimized by cross-validation

	REGED0	MARTI00	MARTI01	MARTI0	MARTI1	MARTI2
Parents	0.948	0.943	0.867	0.883	0.617	0.853
Children	0.999	0.999	0.990	0.842	0.712	0.500
PC	1.000	1.000	0.997	0.981	0.845	0.853
MB	1.000	1.000	0.995	0.975	0.744	0.853
All	0.995	0.997	0.983	0.974	0.843	0.727
All \ MB	0.882	0.875	0.782	0.762	0.728	0.677
Calibrators	N/A	0.493	0.492	0.517	0.505	0.496
All \ Calibrators	N/A	0.997	0.985	0.974	0.843	0.728
MB in natural distr.	See results for MB above				0.865	0.614

We give in Appendix E the Matlab code to obtain MARTI01 from MARTI0 training data. Other preprocessing have been provided by the participants:
http://clopinet.com/isabelle/Projects/WCCI2008/Analysis.html#MARTIprepro

The reference submission uploaded to the website of the challenge. (reference_hpc) is for HITON-PC (max-k=1, alpha=0.01) & Linear SVM (C=0.001), using original for both algorithms. The test AUC results are:
MARTI0 0.9830
MARTI1 0.8595
MARTI2 0.7652

The best AUC performance of challenge particiants are:
MARTI0 1.000
MARTI1 0.947
MARTI2 0.798

The best score on the LOCANET task obtained by participants is: 0.21

Appendix A: Generation of random probes

We describe a method aimed as assessing the fraction of non-causes in a subset of causes of a target variable selected by a causal discovery algorithm. The method consists in generating variables whose distribution resembles the real variables, but are either unrelated to the target, or related to it in a non causal way (consequences or confounders). Those variables, called "probes" by analogy to the probe method in feature selection, are intermixed with the real variables and a causal discovery algorithm is run. The fraction of probes in the variables selected as causes of the target may be used to determine the false positive rate and false discovery rate.

Notations

We call X the data matrix with p lines (patterns) and n columns (features/variables). We call R the matrix of random probes of dimension (p, r), which are unrelated to the target. We call C the matrix of confounders and consequences of dimension (p, c).

Generating variables unrelated to the target (R matrix)

Variables unrelated to the target are generated by taking blocks of variables in the original data matrix and permuting the order of the rows randomly. The resulting variables should be uncorrelated to the target, except for coincidental correlations due to the small number of samples. If the blocks are of size one, the variables generated are uncorrelated with one another. Otherwise, there are block correlations between them. We show the Matlab code in Appendix A1.

Generating consequences and confounders (C and E matrices)

To generate confounders, subsets of real variables (X matrix) and of the probes unrelated to the target (R matrix) are used as an input to a non-linear function. The same method is applied to generate consequences of the target by adding the target in the set of inputs. The code generating such probes in shown in Appendix A3.

As a non-linear function, we use a 2 layer neural network. The inputs are expected to lie approximately between –1 and 1 (or between 0 and 1). For all neurons, we use weights drawn randomly from $N(0,1)/\text{fanin}$. The hidden units use the tanh(ax) as squashing function. The slope a determines the amount of non-linearity added by the hidden layer because the second layer is connected both to the hidden units and directly to the inputs. Random noise drawn from $N(0,\varepsilon)$ is added to the output (ε is proportional to the output range). Finally, the distribution of the output values is mapped to the distribution of one of the real variables (this adds additional non-linearity). We show the Matlab code of the non-linear function in Appendix A2. In Figure A1, we show an example of function obtained for a univariate input between 0 and 1.

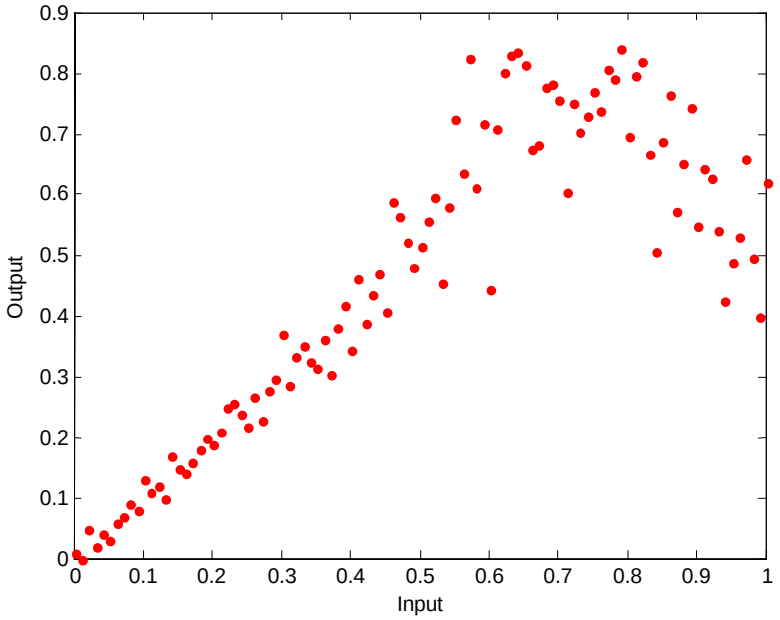


Figure A1: Example of artificial non-linear function. The example was obtained using the code of Appendix A2, with x=[0:.01:1]; y=rand_func(x', sin(x'), 0.05, 1, 2, 1);

Manipulations

Only probes are manipulated in test data in such a way that they become all independent of the target. This is achieved by permuting the values of each probe in test data.

Example architecture

In Figure A2, we show the architecture of the fake variable (probe) network. In Appendix A4, we reproduce the code of this probe network architecture:

- We do not know the causal relationships of the true variables to the target.
- We first draw `probe_num` random probes independent from the target (R).
- We reserve ½ of R, which we do not use as input to the probe network and thus remain fully independent of the target.
- We use ¼ of R as spouses of causes of the target to create `confounder_num` confounders (C).
- We use ¼ of R as spouses of the target to create `consequence_num` consequences (E).
- Random subsets of the confounders are also used to influence the consequences.
- The average fanin (number of variables influencing a probe) is monitored by the parameter “sparsity”, which we chose to be 0.01 time the number of real variables. We use a parameter `slope_squash=2` to monitor the amount of non-linearity.

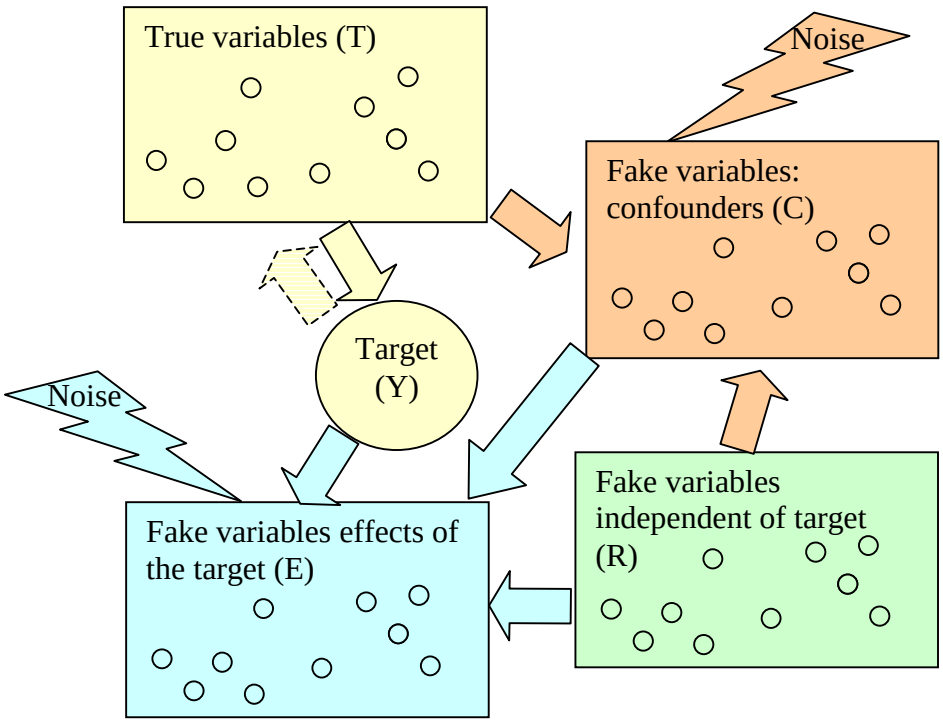


Figure A2: Architecture of the probe network. In yellow we indicate the dependencies of the real variables (some of which may be causes or consequences of the target). In green, we indicate the random probes drawn first, which are independent of the target. In orange, we indicate how the confounders are generated having as patents subsets of the green and yellow variables. In cyan, we indicate consequences are created using as parent the target and subsets of green and orange variables. This architecture exhibits the following conditional independencies: $T \perp R$, $Y \perp R$, $Y \perp C|T$, $E \perp T|(Y,C)$.

Testing

To test our simulator, we compute several correlation coefficients *R* for subsets of variables in unmanipulated and manupulated data. We call T_i, R_i, C_i, and E_i a column of the T, R, C, and E matrices, respectively, and Y the target vector.
We compute statistics for the absolute value, including:

- *R* (T_i, R_j) – expected to be close to 0
- *R* (T_i, R_j | $C(T_i, R_j)$), $C(T_i, R_j)$ effect of T_i and R_j – expected to be **non-zero**
- *R* (Y, R_j) – expected to be close to 0
- *R* (Y, R_j | $E(Y, R_j)$), $E(Y, R_j)$ effect of Yand R_j – expected to be **non-zero**
- *R* (Y, C_j) – expected to be **non-zero**
- *R* (Y, C_j | T) – expected to be close to 0, except in manipulated data
- *R* (T_i, C_j) – expected to be **non-zero** for some pairs, except in manipulated data
- *R* (Y, E_j) – expected to be **non-zero**, except in manipulated data
- *R* (T_i, E_j | Y, C) – expected to be close to 0

To compute the conditional correlation coefficient of A_i and B_j given C, where C is a matrix of column vectors, we proceed as follows:

- standardize A_i, B_j and the columns of C
- project A_i, and B_j on the null space of C
- compute the correlation of the projections

It can be shown that if C is a single column then

$$R(A_i, B_j \mid C) = [R(A_i, B_j) - R(A_i, C)\,R(B_j, C)] / sqrt[(1 - R(A_i, C))^2 (1 - R(B_j, C))^2]$$

The verification code is reported in Appendix A5.

Here is a simple Matlab example that runs the code:

```
p=1000; % Number of samples
x=randn(1000,1);
y=sign(x);
x=x(:,ones(10,1))+randn(1000,10)/5; % replicate the same variable and
add noise
fnum=size(x, 2);
fprintf('Created dataset with %d features, with average correl to
target=%5.4f+-%5.4f\n', fnum, mean(condcor(x,y)), std(condcor(x,y)));
rp=20; % Completely random, not related to target
ca=10; % Confounders
ef=10; % Effects of target
nl=2;  % Level of non-linearity
mn=p/2;% Number of manipulated variables
% Add probes to the data matrix
[xx, parents]=add_probes(x, y, rp,ca,ef,[],[],[],[],nl,[],mn);

fprintf('\nAdded %d random probes:\n', (rp+ca+ef));
fprintf('%d not related to the target,\n', rp);
fprintf('%d confounders\n', ca);
fprintf('%d effects\n', ef);
fprintf('\n+++++++++++++++++++++++++++++++++\n');
fprintf('+++ Testing unmanipulated data +++\n');
fprintf('+++++++++++++++++++++++++++++++++\n\n');
test_net(xx(1:500,:), y(1:500), parents, fnum);
fprintf('\n+++++++++++++++++++++++++++++++\n');
fprintf('+++ Testing manipulated data +++\n');
fprintf('+++++++++++++++++++++++++++++++\n');
show_net_again=0;
test_net(xx(501:1000,:), y(501:1000), parents, fnum, show_net_again);
```

```
Created dataset with 10 features, with average correl to
target=0.7829+-0.0037
*** Creating 20 random probes by blocks of 1
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
*** Creating 10 confounders,
    with on average 1 parents drawn randomly from the 10 true variables
    and 1 random probe parents from a pool of 5 probes
Distillating
Keeping only 10/10 true variables most correlated to Y
Normalizing
```

```
Adding variables,
      average number of true parents=1
      average number of probe parents=1
10% 20% 30% 40% 50% 60% 70% 80% 90% 100%
*** Creating 10 effects of the target,
    with on average 1 parents drawn randomly from the 10 confounders
    and 1 random probe parents from a pool of 5 probes
Normalizing
Adding variables,
      average number of true parents=1
      average number of probe parents=1
10% 20% 30% 40% 50% 60% 70% 80% 90% 100%
Added 40 random probes:
20 not related to the target,
10 confounders
10 effects

++++++++++++++++++++++++++++++++
+++ Testing unmanipulated data +++
++++++++++++++++++++++++++++++++

== Total number of variables: 50 ==
== Real variables (10): 1 ... 10
== Probes (20):
  4 spouses of true var:
(11 -> 16  18  20 )
(13 -> 17  19  22  23  24 )
(14 -> 16  21  24  25 )
(15 -> 17  19  22 )
  5 spouses of target:
(26 -> 35  38  40 )
(27 -> 34  37 )
(28 -> 31  33  39 )
(29 -> 36 )
(30 -> 31  32  36  37  38  39 )
  11 independent of target: Warning, some probes assigned to be spouses
are unused
== Confounders (10): 16 ... 25
(16 <- 10  6  14  11 )
(17 <- 4  13  15 )
(18 <- 10  4  11 )
(19 <- 10  13  15 )
(20 <- 10  11 )
(21 <- 5  4  14 )
(22 <- 1  15  13 )
(23 <- 2  13 )
(24 <- 8  13  14 )
(25 <- 7  2  14 )
== Effects (10): 31 ... 40
(31 <- 0  18  21  28  30 )
(32 <- 0  21  30 )
(33 <- 0  20  19  28 )
(34 <- 0  25  27 )
(35 <- 0  18  26 )
(36 <- 0  22  21  29  30 )
(37 <- 0  20  27  30 )
(38 <- 0  25  30  26 )
```

```
(39 <- 0  21  24  30  28 )
(40 <- 0  20  22  26 )

** Those should NOT be close to zero (ever) **
**> R (Y, Tj) -- Target dependent on true variables
    -- Top 1%:        <abs(R)>=0.7966+-0.0000
    -- Top 10%:       <abs(R)>=0.7966+-0.0000
    -- All:           <abs(R)>=0.7857+-0.0069

** Those should be close to zero **
==> R (Ti, Rj) -- Probes independent of the true variables
    -- Top 1%:        <abs(R)>=0.1129+-0.0003
    -- Top 10%:       <abs(R)>=0.0851+-0.0190
    -- All:           <abs(R)>=0.0310+-0.0242
==> R (Y, Rj) -- Probes independent of the target
    -- Top 1%:        <abs(R)>=0.1028+-0.0000
    -- Top 10%:       <abs(R)>=0.1006+-0.0030
    -- All:           <abs(R)>=0.0311+-0.0267
==> R (Y, Cj | T) -- Confounders independent of target given the true
variables (max of 50 confounders sampled)
    -- Top 1%:        <abs(R)>=0.0932+-0.0000
    -- Top 10%:       <abs(R)>=0.0932+-0.0000
    -- All:           <abs(R)>=0.0370+-0.0301
==> R (Ti, Ej | Y, C) -- True variables (top most corr w. Y)
independent of effects given the parents of the effects (max of 50
effects sampled)
    -- Top 1%:        <abs(R)>=0.1185+-0.0000
    -- Top 10%:       <abs(R)>=0.0862+-0.0143
    -- All:           <abs(R)>=0.0325+-0.0269
    -- For comparison, unconditioned dependency of same true var (top
most corr w. Y) and effects (same effects sampled)
    -- Top 1%:        <abs(R)>=0.7752+-0.0000
    -- Top 10%:       <abs(R)>=0.7461+-0.0118
    -- All:           <abs(R)>=0.5212+-0.2056
    -- For comparison, unconditioned dependency of same true var (top
most corr w. Y) and effects (all effects)
    -- Top 1%:        <abs(R)>=0.7752+-0.0000
    -- Top 10%:       <abs(R)>=0.7461+-0.0118
    -- All:           <abs(R)>=0.5212+-0.2056
    -- For comparison, unconditioned dependency of same true var (top
most corr w. Y) and the target (all effects)
    -- Top 1%:        <abs(R)>=0.7966+-0.0000
    -- Top 10%:       <abs(R)>=0.7966+-0.0000
    -- All:           <abs(R)>=0.7857+-0.0069

** Those should be close to zero ONLY in manipulated test data **
==> R (Ti, Cj) -- Confounders dependent on their parents (true
variables)
    -- Parents, which are true variables
    -- Top 1%:        <abs(R)>=0.8926+-0.0000
    -- Top 10%:       <abs(R)>=0.8926+-0.0000
    -- All:           <abs(R)>=0.4737+-0.3109
    -- Parents, which are probes
    -- Top 1%:        <abs(R)>=0.9526+-0.0000
    -- Top 10%:       <abs(R)>=0.9436+-0.0128
    -- All:           <abs(R)>=0.5310+-0.3201
==> R (Ti, Rj | C(Ti, Rj)), C(Ti, Rj) effect of Ti and Rj
```

```
    -- Dependency of true var and probes induced by confounders (max of
50 confounders sampled)
    -- Top 1%:        <abs(R)>=0.7284+-0.0000
    -- Top 10%:       <abs(R)>=0.7275+-0.0013
    -- All:           <abs(R)>=0.3552+-0.2284
    -- For comparison, unconditioned dependency of same true var and
probes (same confounders sampled)
    -- Top 1%:        <abs(R)>=0.0518+-0.0000
    -- Top 10%:       <abs(R)>=0.0515+-0.0004
    -- All:           <abs(R)>=0.0339+-0.0102
    -- For comparison, unconditioned dependency of same true var and
probes (all samples)
    -- Top 1%:        <abs(R)>=0.0518+-0.0000
    -- Top 10%:       <abs(R)>=0.0515+-0.0004
    -- All:           <abs(R)>=0.0339+-0.0102
**> R (Y, Cj) -- Target dependent on counfounders
    -- Top 1%:        <abs(R)>=0.7135+-0.0000
    -- Top 10%:       <abs(R)>=0.7135+-0.0000
    -- All:           <abs(R)>=0.3563+-0.2551
==> R (Ci, Rj | E), E are effects of the target, Ci, and Rj are parents
of these effects
    -- Target spouses become dependent given their children (max of 50
effects sampled)
    -- Top 1%:        <abs(R)>=0.7728+-0.0000
    -- Top 10%:       <abs(R)>=0.6309+-0.2006
    -- All:           <abs(R)>=0.1670+-0.2078
    -- For comparison, the same without conditioning on the
effects(same effects)
    -- Top 1%:        <abs(R)>=0.0878+-0.0000
    -- Top 10%:       <abs(R)>=0.0819+-0.0083
    -- All:           <abs(R)>=0.0313+-0.0284
    -- For comparison, the same without conditioning on the effects(all
effects)
    -- Top 1%:        <abs(R)>=0.0878+-0.0000
    -- Top 10%:       <abs(R)>=0.0819+-0.0083
    -- All:           <abs(R)>=0.0313+-0.0284
    -- For comparison, effects and their probe parents (same effects)
    -- Top 1%:        <abs(R)>=0.8433+-0.0000
    -- Top 10%:       <abs(R)>=0.7681+-0.1063
    -- All:           <abs(R)>=0.3204+-0.2218
    -- For comparison, effects and their confounder parents (same
effects)
    -- Top 1%:        <abs(R)>=0.8768+-0.0000
    -- Top 10%:       <abs(R)>=0.8255+-0.0725
    -- All:           <abs(R)>=0.3394+-0.2670
==> R (Y, Rj | E(Y, Rj)), E(Y, Rj) effect of Y and Rj
    -- Target spouses and target become dependent given their children
(max of 50 souses sampled)
    -- Top 1%:        <abs(R)>=0.4694+-0.0000
    -- Top 10%:       <abs(R)>=0.4694+-0.0000
    -- All:           <abs(R)>=0.2432+-0.1422
    -- For comparison, correlation target spouses and target, without
conditioning (same spouses)
    -- Top 1%:        <abs(R)>=0.0985+-0.0000
    -- Top 10%:       <abs(R)>=0.0985+-0.0000
    -- All:           <abs(R)>=0.0420+-0.0342
```

```
    -- For comparison, correlation target spouses and target, without
conditioning (all spouses)
    -- Top 1%:        <abs(R)>=0.0985+-0.0000
    -- Top 10%:       <abs(R)>=0.0985+-0.0000
    -- All:           <abs(R)>=0.0420+-0.0342
**> R (Y, Ej) -- Effects of the target correlated to the target
    -- Top 1%:        <abs(R)>=0.8092+-0.0000
    -- Top 10%:       <abs(R)>=0.8092+-0.0000
    -- All:           <abs(R)>=0.6203+-0.2725
==> R (Ei, Cj) and R (Ei, Rj) -- Effects of the target correlated to
their other parents
    -- Parents, which are confounders
    -- Top 1%:        <abs(R)>=0.8768+-0.0000
    -- Top 10%:       <abs(R)>=0.8255+-0.0725
    -- All:           <abs(R)>=0.3394+-0.2670
    -- Parents, which are probes
    -- Top 1%:        <abs(R)>=0.8433+-0.0000
    -- Top 10%:       <abs(R)>=0.7681+-0.1063
    -- All:           <abs(R)>=0.3204+-0.2218

++++++++++++++++++++++++++++++++
+++ Testing manipulated data +++
++++++++++++++++++++++++++++++++
== Total number of variables: 50 ==
== Real variables (10): 1 ... 10
== Probes (20):
  4 spouses of true var:
  5 spouses of target:
  11 independent of target: Warning, some probes assigned to be spouses
are unused
== Confounders (10): 16 ... 25
  1.40+- 0.52 true variable parents,  1.50+- 0.53 parents unrelated to
target
== Effects (10): 31 ... 40
  1.50+- 0.53 confounder parents,  1.50+ 0.53 parents unrelated to
target

** Those should NOT be close to zero (ever) **
**> R (Y, Tj) -- Target dependent on true variables
    -- Top 1%:        <abs(R)>=0.7924+-0.0000
    -- Top 10%:       <abs(R)>=0.7924+-0.0000
    -- All:           <abs(R)>=0.7865+-0.0050

** Those should be close to zero **
==> R (Ti, Rj) -- Probes independent of the true variables
    -- Top 1%:        <abs(R)>=0.1304+-0.0015
    -- Top 10%:       <abs(R)>=0.0895+-0.0323
    -- All:           <abs(R)>=0.0274+-0.0265
==> R (Y, Rj) -- Probes independent of the target
    -- Top 1%:        <abs(R)>=0.0727+-0.0000
    -- Top 10%:       <abs(R)>=0.0715+-0.0018
    -- All:           <abs(R)>=0.0353+-0.0230
==> R (Y, Cj | T) -- Confounders independent of target given the true
variables (max of 50 confounders sampled)
    -- Top 1%:        <abs(R)>=0.0899+-0.0000
    -- Top 10%:       <abs(R)>=0.0899+-0.0000
    -- All:           <abs(R)>=0.0330+-0.0306
```

```
==> R (Ti, Ej | Y, C) -- True variables (top most corr w. Y)
independent of effects given the parents of the effects (max of 50
effects sampled)
    -- Top 1%:        <abs(R)>=0.1231+-0.0000
    -- Top 10%:       <abs(R)>=0.0990+-0.0103
    -- All:           <abs(R)>=0.0520+-0.0269
    -- For comparison, unconditioned dependency of same true var (top
most corr w. Y) and effects (same effects sampled)
    -- Top 1%:        <abs(R)>=0.1098+-0.0000
    -- Top 10%:       <abs(R)>=0.0971+-0.0053
    -- All:           <abs(R)>=0.0405+-0.0306
    -- For comparison, unconditioned dependency of same true var (top
most corr w. Y) and effects (all effects)
    -- Top 1%:        <abs(R)>=0.1098+-0.0000
    -- Top 10%:       <abs(R)>=0.0971+-0.0053
    -- All:           <abs(R)>=0.0405+-0.0306
    -- For comparison, unconditioned dependency of same true var (top
most corr w. Y) and the target (all effects)
    -- Top 1%:        <abs(R)>=0.7924+-0.0000
    -- Top 10%:       <abs(R)>=0.7924+-0.0000
    -- All:           <abs(R)>=0.7865+-0.0050

** Those should be close to zero ONLY in manipulated test data **
==> R (Ti, Cj) -- Confounders dependent on their parents (true
variables)
    -- Parents, which are true variables
    -- Top 1%:        <abs(R)>=0.0864+-0.0000
    -- Top 10%:       <abs(R)>=0.0864+-0.0000
    -- All:           <abs(R)>=0.0311+-0.0202
    -- Parents, which are probes
    -- Top 1%:        <abs(R)>=0.1102+-0.0000
    -- Top 10%:       <abs(R)>=0.0796+-0.0433
    -- All:           <abs(R)>=0.0347+-0.0252
==> R (Ti, Rj | C(Ti, Rj)), C(Ti, Rj) effect of Ti and Rj
    -- Dependency of true var and probes induced by confounders (max of
50 confounders sampled)
    -- Top 1%:        <abs(R)>=0.0425+-0.0000
    -- Top 10%:       <abs(R)>=0.0425+-0.0001
    -- All:           <abs(R)>=0.0264+-0.0128
    -- For comparison, unconditioned dependency of same true var and
probes (same confounders sampled)
    -- Top 1%:        <abs(R)>=0.0437+-0.0000
    -- Top 10%:       <abs(R)>=0.0424+-0.0019
    -- All:           <abs(R)>=0.0270+-0.0125
    -- For comparison, unconditioned dependency of same true var and
probes (all samples)
    -- Top 1%:        <abs(R)>=0.0437+-0.0000
    -- Top 10%:       <abs(R)>=0.0424+-0.0019
    -- All:           <abs(R)>=0.0270+-0.0125
**> R (Y, Cj) -- Target dependent on counfounders
    -- Top 1%:        <abs(R)>=0.0803+-0.0000
    -- Top 10%:       <abs(R)>=0.0803+-0.0000
    -- All:           <abs(R)>=0.0368+-0.0188
==> R (Ci, Rj | E), E are effects of the target, Ci, and Rj are parents
of these effects
    -- Target spouses become dependent given their children (max of 50
effects sampled)
```

```
    -- Top 1%:        <abs(R)>=0.0934+-0.0000
    -- Top 10%:       <abs(R)>=0.0871+-0.0090
    -- All:           <abs(R)>=0.0328+-0.0279
    -- For comparison, the same without conditioning on the
effects(same effects)
    -- Top 1%:        <abs(R)>=0.0929+-0.0000
    -- Top 10%:       <abs(R)>=0.0870+-0.0083
    -- All:           <abs(R)>=0.0327+-0.0282
    -- For comparison, the same without conditioning on the effects(all
effects)
    -- Top 1%:        <abs(R)>=0.0929+-0.0000
    -- Top 10%:       <abs(R)>=0.0870+-0.0083
    -- All:           <abs(R)>=0.0327+-0.0282
    -- For comparison, effects and their probe parents (same effects)
    -- Top 1%:        <abs(R)>=0.0674+-0.0000
    -- Top 10%:       <abs(R)>=0.0647+-0.0038
    -- All:           <abs(R)>=0.0355+-0.0240
    -- For comparison, effects and their confounder parents (same
effects)
    -- Top 1%:        <abs(R)>=0.0720+-0.0000
    -- Top 10%:       <abs(R)>=0.0699+-0.0029
    -- All:           <abs(R)>=0.0331+-0.0248
==> R (Y, Rj | E(Y, Rj)), E(Y, Rj) effect of Y and Rj
    -- Target spouses and target become dependent given their children
(max of 50 souses sampled)
    -- Top 1%:        <abs(R)>=0.0727+-0.0000
    -- Top 10%:       <abs(R)>=0.0727+-0.0000
    -- All:           <abs(R)>=0.0332+-0.0279
    -- For comparison, correlation target spouses and target, without
conditioning (same spouses)
    -- Top 1%:        <abs(R)>=0.0727+-0.0000
    -- Top 10%:       <abs(R)>=0.0727+-0.0000
    -- All:           <abs(R)>=0.0348+-0.0291
    -- For comparison, correlation target spouses and target, without
conditioning (all spouses)
    -- Top 1%:        <abs(R)>=0.0727+-0.0000
    -- Top 10%:       <abs(R)>=0.0727+-0.0000
    -- All:           <abs(R)>=0.0348+-0.0291
**> R (Y, Ej) -- Effects of the target correlated to the target
    -- Top 1%:        <abs(R)>=0.0886+-0.0000
    -- Top 10%:       <abs(R)>=0.0886+-0.0000
    -- All:           <abs(R)>=0.0364+-0.0272
==> R (Ei, Cj) and R (Ei, Rj) -- Effects of the target correlated to
their other parents
    -- Parents, which are confounders
    -- Top 1%:        <abs(R)>=0.0720+-0.0000
    -- Top 10%:       <abs(R)>=0.0699+-0.0029
    -- All:           <abs(R)>=0.0331+-0.0248
    -- Parents, which are probes
    -- Top 1%:        <abs(R)>=0.0674+-0.0000
    -- Top 10%:       <abs(R)>=0.0647+-0.0038
    -- All:           <abs(R)>=0.0355+-0.0240
```

Appendix A1: Generation of variables unrelated to the target

```
function Xp=create_random_probes(X, probe_num, block_size)
%Xp=create_random_probes(X, probe_num, block_size)
% Create a matrix Xp containing probes.
% This is done by permuting blocks of the original matrix.
% X             -- Data matrix p x n
% probe_num     -- dim(Xp, 2)=N
% block_size    -- number of features permuted in block. If
block_size=1,
%                  the all probes correspond to variables individually
permuted.
% Returns:
% Xp            -- matrix of probes of dim p x N

% Isabelle Guyon -- isabelle@clopinet.com -- October 2007

[p, n]=size(X);
N=probe_num;
if issparse(X)
    Xp=sparse(p, N);
else
    Xp=zeros(p, N);
end

beg0=1;
fin0=min(block_size, n);
beg1=beg0;
fin1=fin0;
while 1
    fprintf('%d ', fin1);
    % define a new block of data from X
    rng0=beg0:fin0;
    % define where to put it in Xp
    rng1=beg1:fin1;
    % Create a random permutation
    ip=randperm(p);
    % Assign values
    Xp(:,rng1)=X(ip,rng0);
    % next bounds in X
    beg0=fin0+1;
    if beg0>n, % restart at the beginning
        beg0=1;
        fin0=min(block_size, n);
    else
        fin0=min(fin0+block_size, n);
    end
    % next bounds in Xp
    beg1=fin1+1;
    if beg1>N, break; end
    fin1_new=fin1+length(beg0:fin0);
    if fin1_new>N
        fin0=fin0-(fin1_new-N);
```

```
        fin1=N;
    else
        fin1=fin1_new;
    end
end
fprintf('\n');
```

Appendix A2: Non-linear function used to generate confounders and consequences

```
function y=rand_func(x, r, noise, h, slope_squash, debug)
%y=rand_func(x, r, noise, h, slope_squash, debug)
% Take an x vector as an input and generates a random function from it
% We use a neural network of 2 layers with as many hidden units as
inputs
% and a direct connection from input to output.
% x: input vector (also works for data matrices, patterns in lines)
% Expects inputs roughly between 0 and 1 or between -1 and 1.
% r: variable of which we want to mimic the distribution
% noise: noise level of output_range*N(0, noise) added
% slope_squash: slope at origin of the squashing function (tanh)
% h: number of hidden units
% debug: debug flag

% Isabelle Guyon -- isabelle@clopinet.com -- October 2007

if nargin<2, r=[]; end
if nargin<3, noise=0.05; end
if nargin<4, h=[]; end
if nargin<5, slope_squash=1; end
if nargin<6, debug=0; end

[p, n]=size(x);
[pr, nr]=size(r);

% number of hidden units
if isempty(h), h=n; end
if debug, h=10; end

% First layer parameters
w1=randn(h,n)./n;
b1=randn(h,1)./n;
b1=b1(:,ones(1,p));
% Second layer parameters
w2=randn(1,n+h)./(n+h);
b2=randn./(n+h);

% Network computations
v=x*w1'+b1';
y1=tanh(slope_squash*v);
y1=[y1, x];
v=y1*w2'+b2';
if debug
    y=tanh(slope_squash*v); % Try also squashing
    [sv, vi]=sort(v);
    h1=figure; subplot(3,1,1); plot(sv, '.'); title('Blue: Raw v');
```

```
    [sy, si]=sort(y);
    h2=figure; subplot(3,1,1); plot(sy, '.'); title('Blue: Raw y');
    yorig=y;
    vorig=v;
end

% Add random noise
d=max(v)-min(v);
v=v+noise*d*randn(size(v));
if debug
    y=y+noise*randn(size(y));
    % Note: the effect of squashing and adding noise to y
    % at this level makes the noise not hoeoscedastic
    figure(h1); hold on; subplot(3,1,1); plot(v(si), 'r.');
title('Blue: Raw v distribution; Red: plus noise');
    figure(h2); hold on; subplot(3,1,1); plot(y(si), 'r.');
title('Blue: Raw y distribution; Red: plus noise');
    if ~isempty(r)
        % Mimic the distribution of r for y
        [ys, is]=sort(y);
        rs=sort(r);
        y(is)=rs;
    end
end

if ~isempty(r)
    % Mimic the distribution of r directly for v
    [vs, is]=sort(v);
    rs=sort(r);
    v(is)=rs;
end

if debug
    figure(h1); hold on; subplot(3,1,2); plot(v(vi), 'g.');
title('Green=NL mapping');
    figure(h2); hold on; subplot(3,1,2); plot(y(si), 'g.');
title('Green=NL mapping');
    figure(h1); hold on; subplot(3,1,3); plot(vorig, v, 'r.');
xlabel('v before'); ylabel('v after');
    figure(h2); hold on; subplot(3,1,3); plot(yorig, y, 'r.');
xlabel('y before'); ylabel('y after');
end

% In the end we find better to use the direct non-linear mapping of v
to r
y=v;

if debug
    figure; plot(vorig, y, 'r.'); xlabel('Net output'); ylabel('Output
mapped to matched desired distribution');
    if size(x, 2)==1
        figure; plot(x, y, 'r.'); xlabel('Input'); ylabel('Output');
    end
end
```

Appendix A3: Generation of variables related to others (may include the target)

```
function [Xc, parents]=create_confounders(X, Xp, Y, confounder_num,
true_parent_num, probe_parent_num, non_linearity_level, noise, top_num)
%[Xc, parents]=create_confounders(X, Xp, Y, confounder_num,
true_parent_num, probe_parent_num, non_linearity_level, noise, top_num)
% Create a matrix Xc containing confounders that are consequences of
real variables and probes.
% This is done by defining a sparse architecture and then applying
functions to the inputs to generate new inputs.
% The output is then made to resemble the distribution of a real
variable.
% X             -- Data matrix p x n containing real variables
% Xp             -- Data matrix p x m containing probes
% Y             -- Target values (p) (not provided if we want to
exclude
%                   the target)
% confounder_num -- dim(Xc, 2)= N
%                  the all probes correspond to variables individually
permuted.
% true_parent_num, probe_parent_num -- number of parents of the created
variables
% non_linearity_level -- Slope of thr tanh in the hidden layer (the
larger
%                       the more non-linear. Use 1 for almost linear.
% noise -- Random noise level (as a fraction of the variable output
range).
% top_num      -- number of examples most correlated to Y used in X as
%                  input to confounders. Y is then not used as input.
% Returns:
% Xc            -- Matrix of probes of dim p x N
% parents       -- A cell array with lists of parents among input
variables
%                  The variables are numbers 1..n in X and n+1...n+m in
Xp
% If the target is given, we add it to all input variable sets (this
%                   creates consequences).

% Isabelle Guyon -- isabelle@clopinet.com -- October 2007

if nargin<8, noise=0.05; end
do_not_add_Y=0;
if nargin>=9,
    do_not_add_Y=1;
else
    top_num=size(X,2);
end

[p, n]=size(X);
[p, m]=size(Xp);
N=confounder_num;

if ~isempty(Y) & do_not_add_Y
    % Distillate the data
```

```
    fprintf('Distillating\n');
    idx_feat=balcor_select(X, Y);
    idx_good=idx_feat(1:top_num);
    Y=[];
    fprintf('Keeping only %d/%d true variables most correlated to Y\n',
length(idx_good), n);
else
    idx_good=[1:n]';
end
ng=length(idx_good);

if issparse(X)
    Xc=sparse(p, N);
else
    Xc=zeros(p, N);
end

% Divide variables by their maximum, to bring them between 0 and 1
Xs=[X Xp];
xmax=max(Xs);
fprintf('Normalizing\n');
for k=1:size(Xs,2)
    if xmax(k)~=0
        Xs(:,k)=Xs(:,k)./xmax(k);
    end
end

% Average number of variables to be selected
parents={};
fprintf('Adding variables,\n\taverage number of true parents=%d',
true_parent_num);
fprintf('\n\taverage number of probe parents=%d\n', probe_parent_num);
percent_done=0;
old_percent_done=0;
for k=1:N
    percent_done=floor(k/N*100);
    if ~mod(percent_done,10) & percent_done~=old_percent_done,
        fprintf('%d%% ', percent_done);
    end
    old_percent_done=percent_done;
    % Select a random subset of real variables and of probes
    fanin_real=max(1, min(ceil(true_parent_num*(1+(rand-0.5))), ng));
    rp_real=idx_good(randperm(ng)); % We prefer the variables
correlated to the target
    fanin_probe=max(1, min(ceil(probe_parent_num*(1+(rand-0.5))), m));
    rp_probe=randperm(m)';
    parents{k}=[rp_real(1:fanin_real); n+rp_probe(1:fanin_probe)];
    % Select a real variable at random
    rp=randperm(n);
    r=X(:,rp(1)); % This one should not be standardized
    % Input the variables to the non-linear function
    h=fanin_real+fanin_probe; % number of hidden units
    Xc(:,k)=rand_func([Xs(:,parents{k}), Y] , r, noise, h,
non_linearity_level);
    % Add Y as parent
    if ~isempty(Y)
        parents{k}=[0; parents{k}];
```

```
    end
end
function [idx_feat, cor_val]=balcor_select(X, Y, feat_num)
%[idx_feat, cor_val]=balcor_select(X, Y, feat_num)
% feature selection with correlation coefficient
% which balances the 2 classes by subsampling the second one.

% Isabelle Guyon -- isabelle@clopinet.com -- October 2007

pidx=find(Y==1);
nidx=find(Y==-1);
% take a random subset of negative class elements of the same size as
% the number of positive
rp=randperm(length(nidx));
nidx=nidx(rp(1:min(length(pidx), length(nidx))));
RR=condcor(X([pidx; nidx],:), Y([pidx; nidx]));
[cor_val, idx_feat]=sort(-abs(RR));

if nargin>2
    idx_feat=idx_feat(1:feat_num);
    cor_val=-cor_val(1:feat_num);
else
    cor_val=-cor_val;
end
```

Appendix A4: Generation of all probes

```
function [X, parents]=add_probes(X, Y, probe_num, confounder_num,
consequence_num, conf_true_parent_num, conf_probe_parent_num,
cons_true_parent_num, cons_probe_parent_num, non_linearity_level,
noise, num_manipulated, top_num, noise_Y)
%[X, parents]=add_probes(X, Y, probe_num, confounder_num,
consequence_num, conf_true_parent_num, conf_probe_parent_num,
cons_true_parent_num, cons_probe_parent_num, non_linearity_level,
noise, num_manipulated, top_num, noise_Y)
% Create a matrix X containing probes, urelated to the target or
% confounders and consequeces. The architecture is built in.
% X                 -- Data matrix p x n
% Y                 -- Target values (p) (not provided if we want to
exclude
%                    the target)
% probe_num         -- Number of probes not consequences of real
variables or
%                    the target
% confounder_num    -- Number of confounders
% consequence_num   -- Number of consequences
% conf_true_parent_num, conf_probe_parent_num -- number of parents of
% confounders
% cons_true_parent_num, cons_probe_parent_num -- number of parents of
% consequences
% non_linearity_level -- Slope of the tanh in the hidden layer (the
larger
%                        the more non-linear. Use 1 for almost linear.
% noise             -- Random noise level (as a fraction of the
variable output range).
```

```
% num_manipulated   -- for the num_manipulated last entries the probes
values
%                   are randomized, making all probes independent of
the
%                   target.
% top_num           -- number of examples most correlated to Y used in
X as
%                   input to confounders
% tone_Y_down       -- Multipicative factor to tone Y down as a cause
of its
%                   effects
%
% Returns:
% Xnew          -- Matrix of probes of dim p x N
% parents       -- A cell array with lists of parents of all the
variables
%                  Variables are numbered 1 to n. The target is 0.

% Isabelle Guyon -- isabelle@clopinet.com -- October 2007

[p, n]=size(X);

if nargin<6 | isempty(conf_true_parent_num), conf_true_parent_num=1;
end
if nargin<7 | isempty(conf_probe_parent_num), conf_probe_parent_num=1;
end
if nargin<8 | isempty(cons_true_parent_num), cons_true_parent_num=1;
end
if nargin<9 | isempty(cons_probe_parent_num), cons_probe_parent_num=1;
end
if nargin<10 | isempty(non_linearity_level), non_linearity_level=1; end
if nargin<11 | isempty(noise), noise=0.05; end
if nargin<12 | isempty(num_manipulated), num_manipulated=0; end
if nargin<13 | isempty(top_num), top_num=size(X,2); end
if nargin<14 | isempty(noise_Y), noise_Y=0; end

% Repartition of the probes marginally independent of the target
spouse_cause=floor(probe_num/4);
spouse_target=floor(probe_num/4);
true_random=probe_num-spouse_cause-spouse_target;

% Creation of probes marginally independent of the target
block_size=max(1,round(n/10));
fprintf('*** Creating %d random probes by blocks of %d\n', probe_num,
block_size);
Xp=create_random_probes(X, probe_num, block_size);

% Split probes into spouses and purely random
% First shuffle them
rp=randperm(size(Xp, 2));
Xp=Xp(:, rp);
% Then split into 3 parts
Xp1=Xp(:,1:spouse_cause);
Xp2=Xp(:,spouse_cause+1:spouse_cause+spouse_target);
Xp3=Xp(:,spouse_cause+spouse_target+1:probe_num);
clear Xp;
```

```
% Create confounders, not consequences of the target, using true
variables
% and a subset of the previously drawn probes
fprintf('*** Creating %d confounders, \n    with on average %d parents
drawn randomly from the %d true variables \n    and %d random probe
parents from a pool of %d probes\n', ...
    confounder_num, conf_true_parent_num, size(X,2),
conf_probe_parent_num, size(Xp1, 2) );
[Xc, p1_parents]=create_confounders(X, Xp1, Y, confounder_num,
conf_true_parent_num, conf_probe_parent_num, non_linearity_level,
noise, top_num);

% Create effects of the target and a subset of the previously drawn
probes
fprintf('\n*** Creating %d effects of the target, \n    with on average
%d parents drawn randomly from the %d confounders \n    and %d random
probe parents from a pool of %d probes\n', ...
    consequence_num, cons_true_parent_num, size(Xc,2),
cons_probe_parent_num, size(Xp2, 2));
if noise_Y>0,
    %flip noise_Y*p examples
    rp=randperm(p);
    rp=rp(1:round(noise_Y*p));
    Y(rp)=-Y(rp);
end
[Xe, p2_parents]=create_confounders(Xc, Xp2, Y , consequence_num,
cons_true_parent_num, cons_probe_parent_num, non_linearity_level,
noise);

% Add everything together
X=[X, Xp1, Xc, Xp2, Xe, Xp3];
total_var=n+probe_num+confounder_num+consequence_num;
parents=cell(total_var,1);
% Confounder parents: X (n variables) and Xp1 (spouse _cause variables)
% have no parents: start at n+spouse_cause+1
parents(n+spouse_cause+1:n+spouse_cause+confounder_num)=p1_parents;
% Effect parents: Xp2 have no parents, start at
% n+spouse_cause+confounder_num+spouse_target.
% Offset the indices of p2_parents by n+spouse_cause
for k=1: length(p2_parents)
    p2_parents{k}=p2_parents{k}+n+spouse_cause;
    p2_parents{k}(1)=0; % The target value is not offset
end
parents(n+spouse_cause+confounder_num+spouse_target+1:total_var-
true_random)=p2_parents;

if num_manipulated>0
    probe_idx=(n+1):size(X,2);
    manip_idx=(p-num_manipulated+1):p;
    for k=1:length(probe_idx)
        rp=randperm(num_manipulated);
        X(manip_idx,probe_idx(k))=X(manip_idx(rp),probe_idx(k));
    end
end
```

Appendix A5: Verification code

```
function test_net(X, Y, parents, true_num, debug, lean)
%test_net(X, Y, parents, true_num, , lean)
% Test the independencies in the net
% X -- data matrix (p samples x n variables)
% Y -- target vector (dim p)
% parents -- cell aray of lists of parents of the variables
% true_num -- number of true variables (first in the X matrix)
% debug -- flag to show or not the network
% lean -- flag to remove the calculation of the pvalues

[p, n]=size(X);
if nargin<4, true_num=[]; end
if nargin<5, debug=0; end % display net
if nargin<6, lean=1; end

% Maximum number of conditional correlation coeff computed (for
% computational reasons)
maxval=50;

[parents, children, no_parent_idx, effect_idx, confounder_idx,
true_var_idx, rp_idx, spouse_target_idx, spouse_true_idx,
other_rp_idx]= ...
    draw_net(parents, true_num, debug);

% Reduce the test matrix by selecting a balanced number of examples
pidx=find(Y==1);
nidx=find(Y==-1);
rp=randperm(length(nidx));
nidx=nidx(rp(1:min(length(pidx), length(nidx)))); % The positive class
is usually more depleted
X=X([pidx;nidx],:);
Y=Y([pidx;nidx]);

% Find the true variables
T=X(:,true_var_idx);
% Find the random probes indep Y
R=X(:,rp_idx);
% Find the consequences of the target
E=X(:,effect_idx);
% The rest are the counfounders
C=X(:,confounder_idx);

% Find candidate causes (top maxval variables most correlated to the
target)
fprintf('\n** Those should NOT be close to zero (ever) **\n');
[RR, PP]=condcor(Y, T, [], lean);
fprintf('**> R (Y, Tj) -- Target dependent on true variables\n');
show_R(RR,PP);
[SR, IR]=sort(-abs(RR));
top_cause_idx=IR(1:min(maxval, length(IR)));

% Compute the other dependencies
warning off
% Independence of random probes with other variable
fprintf('\n** Those should be close to zero **\n');
```

```
[RR, PP]=condcor(T, R, [], lean);
fprintf('==> R (Ti, Rj) -- Probes independent of the true
variables\n');
show_R(RR,PP);

[RR, PP]=condcor(Y, R, [], lean);
fprintf('==> R (Y, Rj) -- Probes independent of the target\n');
show_R(RR,PP);

if ~isempty(C)
    % Confounders and target shielded by true variables
    fprintf('==> R (Y, Cj | T) -- Confounders independent of target
given the true variables (max of %d confounders sampled)\n', maxval);
    RR=[]; PP=[];
    for k=1:min(maxval, length(confounder_idx)) % Too long if we
calculate for all
        p_idx=parents{confounder_idx(k)};
        p_idx=intersect(p_idx, true_var_idx);
        [rr,pp]=condcor(Y, C(:,k), X(:,p_idx), lean);
        RR=[RR rr];
        PP=[PP pp];
    end
    show_R(RR,PP);

    % Effects and true variables shielded by parents
    fprintf('==> R (Ti, Ej | Y, C) -- True variables (top most corr w.
Y) independent of effects given the parents of the effects (max of %d
effects sampled)\n', maxval);
    RR=[]; PP=[];
    ms=min(length(effect_idx), maxval);
    for k=1:ms
        p_idx=parents{effect_idx(k)};
        p_idx=intersect(p_idx, confounder_idx);
        [rr,pp]=condcor(T(:, top_cause_idx), E(:,k), [Y X(:,p_idx)],
lean);
        RR=[RR rr];
        PP=[PP pp];
    end
    show_R(RR,PP);
    fprintf('    -- For comparison, unconditioned dependency of same
true var (top most corr w. Y) and effects (same effects sampled)\n');
    [RR,PP]=condcor(T(:, top_cause_idx), E(:, 1:ms), [], lean);
    show_R(RR,PP);
    fprintf('    -- For comparison, unconditioned dependency of same
true var (top most corr w. Y) and effects (all effects)\n');
    [RR, PP]=condcor(T(:, top_cause_idx), E, [], lean);
    show_R(RR,PP);
    fprintf('    -- For comparison, unconditioned dependency of same
true var (top most corr w. Y) and the target (all effects)\n');
    [RR, PP]=condcor(T(:, top_cause_idx), Y, [], lean);
    show_R(RR,PP);
end

if ~isempty(C)
    fprintf('\n** Those should be close to zero ONLY in manipulated
test data **\n');
    % Confounders
```

```
    % C and parents
    fprintf('==> R (Ti, Cj) -- Confounders dependent on their parents
(true variables)\n');
    fprintf('    -- Parents, which are true variables\n');
    RR=[]; PP=[];
    for k=1:length(confounder_idx)
        p_idx=parents{confounder_idx(k)};
        p_idx=intersect(p_idx, true_var_idx);
        [rr,pp]=condcor(X(:,p_idx), C(:,k), [], lean);
        RR=[RR; rr];
        PP=[PP; pp];
    end
    show_R(RR,PP);
    fprintf('    -- Parents, which are probes\n');
    RR=[]; PP=[];
    for k=1:length(confounder_idx)
        p_idx=parents{confounder_idx(k)};
        p_idx=intersect(p_idx, rp_idx);
        [rr,pp]=condcor(X(:,p_idx), C(:,k), [], lean);
        RR=[RR; rr];
        PP=[PP; pp];
    end
    show_R(RR,PP);

    % Induced by C
    fprintf('==> R (Ti, Rj | C(Ti, Rj)), C(Ti, Rj) effect of Ti and
Rj\n    -- Dependency of true var and probes induced by confounders
(max of %d confounders sampled)\n', maxval);
    RR=[]; PP=[];
    for k=1:min(length(confounder_idx), maxval)
        p_idx=parents{confounder_idx(k)};
        tpar_idx=intersect(p_idx, true_var_idx);
        rpar_idx=intersect(p_idx, rp_idx);
        [rr, pp]=condcor(X(:,tpar_idx), X(:,rpar_idx), C(:,k), lean);
        RR=[RR; rr(:)];
        PP=[PP; pp(:)];
    end
    show_R(RR,PP);

    fprintf('    -- For comparison, unconditioned dependency of same
true var and probes (same confounders sampled)\n');
    RR=[]; PP=[];
    for k=1:min(length(confounder_idx), maxval)
        p_idx=parents{confounder_idx(k)};
        tpar_idx=intersect(p_idx, true_var_idx);
        rpar_idx=intersect(p_idx, rp_idx);
        [rr, pp]=condcor(X(:,tpar_idx), X(:,rpar_idx), [], lean);
        RR=[RR; rr(:)];
        PP=[PP; pp(:)];
    end
    show_R(RR,PP);

    fprintf('    -- For comparison, unconditioned dependency of same
true var and probes (all samples)\n');
    RR=[]; PP=[];
    for k=1:length(confounder_idx)
        p_idx=parents{confounder_idx(k)};
```

```
        tpar_idx=intersect(p_idx, true_var_idx);
        rpar_idx=intersect(p_idx, rp_idx);
        [rr, pp]=condcor(X(:,tpar_idx), X(:,rpar_idx), [], lean);
        RR=[RR; rr(:)];
        PP=[PP; pp(:)];
    end
    show_R(RR,PP);

    % Y and C
    fprintf('**> R (Y, Cj) -- Target dependent on counfounders\n');
    [RR, PP]=condcor(Y, C, [], lean);
    show_R(RR,PP);
end

if ~isempty(E)
    % Parents of the effects
    fprintf('==> R (Ci, Rj | E), E are effects of the target, Ci, and
Rj are parents of these effects\n    -- Target spouses become dependent
given their children (max of %d effects sampled)\n', maxval);
    RR=[]; PP=[];
    for k=1:min(length(effect_idx), maxval)
        p_idx=parents{effect_idx(k)};
        rpar_idx=intersect(p_idx, spouse_target_idx);
        tpar_idx=intersect(p_idx, confounder_idx);
        [rr, pp]=condcor(X(:,tpar_idx) , X(:,rpar_idx), E(:,k), lean);
        RR=[RR; rr(:)];
        PP=[PP; pp(:)];
    end
    show_R(RR,PP);
    fprintf('    -- For comparison, the same without conditioning on
the effects(same effects)\n');
    RR=[]; PP=[];
    for k=1:min(length(effect_idx), maxval)
        p_idx=parents{effect_idx(k)};
        rpar_idx=intersect(p_idx, spouse_target_idx);
        tpar_idx=intersect(p_idx, confounder_idx);
        [rr, pp]=condcor(X(:,tpar_idx), X(:,rpar_idx), [], lean);
        RR=[RR; rr(:)];
        PP=[PP; pp(:)];
    end
    show_R(RR,PP);
    fprintf('    -- For comparison, the same without conditioning on
the effects(all effects)\n');
    RR=[]; PP=[];
    for k=1:length(effect_idx)
        p_idx=parents{effect_idx(k)};
        rpar_idx=intersect(p_idx, spouse_target_idx);
        tpar_idx=setdiff(setdiff(p_idx, rpar_idx), [0]);
        [rr, pp]=condcor(X(:,tpar_idx), X(:,rpar_idx), [], lean);
        RR=[RR; rr(:)];
        PP=[PP; pp(:)];
    end
    show_R(RR,PP);
    fprintf('    -- For comparison, effects and their probe parents
(same effects)\n');
    RR=[]; PP=[];
    for k=1:min(length(effect_idx), maxval)
```

```
        p_idx=parents{effect_idx(k)};
        rpar_idx=intersect(p_idx, spouse_target_idx);
        tpar_idx=intersect(p_idx, confounder_idx);
        [rr, pp]=condcor(E(:,k), X(:,rpar_idx), [], lean);
        RR=[RR; rr(:)];
        PP=[PP; pp(:)];
    end
    show_R(RR,PP);
    fprintf('    -- For comparison, effects and their confounder
parents (same effects)\n');
    RR=[]; PP=[];
    for k=1:min(length(effect_idx), maxval)
        p_idx=parents{effect_idx(k)};
        rpar_idx=intersect(p_idx, spouse_target_idx);
        tpar_idx=intersect(p_idx, confounder_idx);
        [rr, pp]=condcor(E(:,k), X(:,tpar_idx), [], lean);
        RR=[RR; rr(:)];
        PP=[PP; pp(:)];
    end
    show_R(RR,PP);


    % Target spouses
    fprintf('==> R (Y, Rj | E(Y, Rj)), E(Y, Rj) effect of Y and Rj\n
-- Target spouses and target become dependent given their children (max
of %d souses sampled)\n', maxval);
    RR=[]; PP=[];
    ms=min(length(spouse_target_idx), maxval);
    for k=1:ms
        c_idx=children{spouse_target_idx(k)};
        [rr, pp]=condcor(Y, X(:,spouse_target_idx(k)), X(:,c_idx),
lean);
        RR=[RR rr];
        PP=[PP pp];
    end
    show_R(RR,PP);
    fprintf('    -- For comparison, correlation target spouses and
target, without conditioning (same spouses)\n');
    [RR, PP]=condcor(Y, X(:,spouse_target_idx(1:ms)), [], lean);
    show_R(RR,PP);
    fprintf('    -- For comparison, correlation target spouses and
target, without conditioning (all spouses)\n');
    [RR, PP]=condcor(Y, X(:,spouse_target_idx), [], lean);
    show_R(RR,PP);

    % Effects and Y
    fprintf('**> R (Y, Ej) -- Effects of the target correlated to the
target\n');
    [RR, PP]=condcor(Y, E, [], lean);
    show_R(RR,PP);
    % Effects and the other parents
    fprintf('==> R (Ei, Cj) and R (Ei, Rj) -- Effects of the target
correlated to their other parents\n');
    fprintf('    -- Parents, which are confounders\n');
    RR=[]; PP=[];
    for k=1:length(effect_idx)
        p_idx=parents{effect_idx(k)};
```

```
        p_idx=intersect(p_idx, confounder_idx);
        [rr,pp]=condcor(E(:,k), X(:,p_idx), [], lean);
        RR=[RR rr];
        PP=[PP pp];
    end
    show_R(RR,PP);
    fprintf('    -- Parents, which are probes\n');
    RR=[]; PP=[];
    for k=1:length(effect_idx)
        p_idx=parents{effect_idx(k)};
        p_idx=intersect(p_idx, spouse_target_idx);
        [rr,pp]=condcor(E(:,k), X(:,p_idx), [], lean);
        RR=[RR rr];
        PP=[PP pp];
    end
    show_R(RR,PP);

end



function [parents, children, no_parent_idx, effect_idx, confounder_idx,
true_var_idx, rp_idx, spouse_target_idx, spouse_true_idx,
other_rp_idx]=draw_net(parents, true_num, debug)
%[parents, children, no_parent_idx, effect_idx, confounder_idx,
%true_var_idx, rp_idx, spouse_target_idx, spouse_true_idx,
other_rp_idx]=draw_net(parents, true_num, debug)
% Show the network.
% Inputs:
% parents   -- A cell array containing lists of variable parents
%               if true_num is given, it is assumed that the first few
variables are true variables
%               the variables are numbered 1, ..., i, ...n and
parents{i} is the list of parents
%               of variable i.
%               if true_num=[], parents{1} is the list of true
variables
%               and parent{i+1} are the parents of i.
% true_num  -- Number of true variables.
% debug     -- debug blag: if 1, show the whole structure.
% Returns:
% parents   -- parents of the variables numbered 1, ..., i, ...n:
parents{i}
%               is the list of parents of variable i.
% children  -- children{i} is the list of children of variable i.
% no_parent_idx -- indices of variables having no parents (includes
true
%                and random probes independent of the target.
% effect_idx -- indices of rpobes which are effects of the target.
% confounder_idx -- indices of probes which are consequences of treu
variables
% true_var_idx -- indices of true variables
% rp_idx     -- indices of random probes independent of the target
% spouse_target_idx -- indices of spouses of the target (probes)
% spouse_true_idx -- indices of spouses of true variables (probes)
% other_rp_idx -- indices of other random probes, indept of target
```

```
%
length(other_rp_idx)+length(spouse_target_idx)+length(spouse_true_idx)
% ==length(rp_idx)

% Isabelle Guyon -- isabelle@clopine.,com -- October 2007

if nargin<2 | isempty(true_num),
    true_var_idx=parents{1};
    parents=parents(2:length(parents));
else
    % Find the true variables
    true_var_idx=1:true_num;
end
if nargin<3, debug=1; end

% Invert the index
children=cell(size(parents));
no_parent_idx=[];
effect_idx=[];
confounder_idx=[];
for k=1:length(parents)
    par=parents{k};
    if isempty(par)
        no_parent_idx=[no_parent_idx k];
    else
        if par(1)==0
            effect_idx=[effect_idx k];
            par=par(2:length(par));
        else
            confounder_idx=[confounder_idx k];
        end
        for j=1:length(par)
            children{par(j)}=[children{par(j)} k];
        end
    end
end

% Find the random probes indep Y
rp_idx=setdiff(no_parent_idx, true_var_idx);

%Spouses:
spouse_true_idx=[];
spouse_target_idx=[];
for k=1:length(rp_idx)
    c_idx=children{rp_idx(k)};
    if ~isempty(c_idx)
        if ~isempty(intersect(c_idx, effect_idx))
            spouse_target_idx=[spouse_target_idx, rp_idx(k)];
        else
            spouse_true_idx=[spouse_true_idx, rp_idx(k)];
        end
    end
end
other_rp_idx=setdiff(setdiff(rp_idx, spouse_target_idx),
spouse_true_idx);
```

```
if
true_num+length(rp_idx)+length(confounder_idx)+length(effect_idx)~=leng
th(parents)
    error('Number of variables do no add up');
end
if
length(other_rp_idx)+length(spouse_target_idx)+length(spouse_true_idx)~
=length(rp_idx)
    error('Number of probes do no add up');
end

fprintf('== Total number of variables: %d ==\n', length(parents));
if ~isempty(true_num)
    fprintf('== Real variables (%d): 1 ... %d\n', true_num, true_num);
else
    fprintf('== Real variables (%d):\n', length(true_var_idx));
end
fprintf('== Probes (%d): \n', length(rp_idx));
fprintf('  %d spouses of true var: \n', length(spouse_true_idx));
if debug
    for k=1:length(spouse_true_idx)
        c_idx=children{spouse_true_idx(k)};
        fprintf('(%d ->',  spouse_true_idx(k));
        for j=1:length(c_idx)
            fprintf(' %d ', c_idx(j));
        end
        fprintf(')\n');
    end
end
fprintf('  %d spouses of target: \n', length(spouse_target_idx));
if debug
    for k=1:length(spouse_target_idx)
        c_idx=children{spouse_target_idx(k)};
        fprintf('(%d ->',  spouse_target_idx(k));
        for j=1:length(c_idx)
            fprintf(' %d ', c_idx(j));
        end
        fprintf(')\n');
    end
end
fprintf('  %d independent of target: ', length(other_rp_idx));
MM=max(other_rp_idx);
mm=min(other_rp_idx);
if(MM-mm+1== length(other_rp_idx))
    fprintf('%d ... %d\n', mm, MM);
else
    fprintf('Warning, some probes assigned to be spouses are
unused\n');
end
fprintf('== Confounders (%d): ', length(confounder_idx));
MM=max(confounder_idx);
mm=min(confounder_idx);
if(MM-mm+1== length(confounder_idx))
    fprintf('%d ... %d\n', mm, MM);
else
    if ~isempty(confounder_idx),
        fprintf('Warning, wierd set\n');
```

```
    end
end
if debug
    for k=1:length(confounder_idx)
        p_idx=parents{confounder_idx(k)};
        fprintf('(%d <-',  confounder_idx(k));
        for j=1:length(p_idx)
            fprintf(' %d ', p_idx(j));
        end
        fprintf(')\n');
    end
else
    nt=[];
    np=[];
    for k=1:length(confounder_idx)
        p_idx=parents{confounder_idx(k)};
        pt_idx=intersect(p_idx, true_var_idx);
        pr_idx=intersect(p_idx, rp_idx);
        nt=[nt length(pt_idx)];
        np=[np length(pr_idx)];
    end
    fprintf(' %5.2f+-%5.2f true variable parents, %5.2f+-%5.2f parents
unrelated to target\n', mean(nt), std(nt), mean(np), std(np));
end

fprintf('== Effects (%d): ', length(effect_idx));
MM=max(effect_idx);
mm=min(effect_idx);
if(MM-mm+1== length(effect_idx))
    fprintf('%d ... %d\n', mm, MM);
else
    if ~isempty(effect_idx), fprintf('Warning, wierd set\n'); end
end
if debug
    for k=1:length(effect_idx)
        p_idx=parents{effect_idx(k)};
        fprintf('(%d <-',  effect_idx(k));
        for j=1:length(p_idx)
            fprintf(' %d ', p_idx(j));
        end
        fprintf(')\n');
    end
else
    nc=[];
    np=[];
    for k=1:length(effect_idx)
        p_idx=parents{effect_idx(k)};
        pc_idx=intersect(p_idx, confounder_idx);
        pr_idx=intersect(p_idx, rp_idx);
        nc=[nc length(pc_idx)];
        np=[np length(pr_idx)];
    end
    fprintf(' %5.2f+-%5.2f confounder parents, %5.2f+%5.2f parents
unrelated to target\n', mean(nc), std(nc), mean(np), std(np));

end
```

```
function [r, pval]=condcor(x, y, C, lean)
%[r, pval]=condcor(x, y, C, lean)
% Computes the correlation between the column vectors x and y
% given the column vectors of matrix C.
% lean -- flag, if 1, do not compute pvalue

if nargin<3, C=[]; end
if nargout>1
    if lean
        pval_compute=0;
    else
        pval_compute=1;
    end
else
    pval_compute=0;
end
pval=[];

debug=0;

[p, n]=size(x);
[pp, m]=size(y);
if p~=pp, error('wrong dimensions'); end

v=1/sqrt(p);

% Center and normalize
x=v*standard(x);
y=v*standard(y);
if ~isempty(C)
    C=v*standard(C);
end

if debug & length(C)==length(x)
    r_verif= (x'*y - (x'*C) * (y'*C))/sqrt((1-(x'*C)^2) * (1-(y'*C)^2))
end

% Project on null space
if ~isempty(C)
    proj=C*pinv(C);
    x=x-proj*x;
    y=y-proj*y;
    % Center and normalize again
    x=v*standard(x);
    y=v*standard(y);
end

% Compute dot product
if pval_compute
    [R, P]=corrcoef([x, y]);
    r=R(1:n,n+1:n+m);
    pval=P(1:n,n+1:n+m);
else
    r=x'*y;
```

```
end

return

% verification:
% condcor(a, b, c) = (condcor(a, b)-condcor(a, c)*condcor(b,
c))/sqrt((1-condcor(a, c)^2)*(1-condcor(b, c)^2) )



function X=standard(X)
%X=standard(X)
% Standardize matrix of column vectors

[p, n]=size(X);
M=mean(X);
S=std(X,1);
X=(X-M(ones(p,1),:));
S(find(S==0))=1;
X=X./S(ones(p,1),:);
X=X./S(ones(p,1),:);
function show_R(RR, PP)
%show_R(RR, PP)
% Show statistics about vectors of correlation coefficients RR and
their
% pvalues PP.

% Isabelle Guyon -- isabelle@clopinet.com -- October 2007

RR=full(abs(RR(:)));
PP=full(PP(:));
[SR0, IR]=sort(-RR);

fprintf('    -- Top 1%%:\t');
mval=max(1, round(length(SR0)/100));
SR=-SR0(1:mval);
fprintf('    <abs(R)>=%5.4f+-%5.4f', mean(SR), std(SR));
if ~isempty(PP),
    SP=PP(IR(1:mval));
    fprintf(', <pval>=%5.4f+-%5.4f\n', mean(SP), std(SP));
else fprintf('\n');
end

fprintf('    -- Top 10%%:\t');
mval=max(1, round(length(SR0)/10));
SR=-SR0(1:mval);
fprintf('    <abs(R)>=%5.4f+-%5.4f', mean(SR), std(SR));
if ~isempty(PP),
    SP=PP(IR(1:mval));
    fprintf(', <pval>=%5.4f+-%5.4f\n', mean(SP), std(SP));
else fprintf('\n');
end

fprintf('    -- All:\t');
fprintf('\t    <abs(R)>=%5.4f+-%5.4f', mean(RR), std(RR));
if ~isempty(PP), fprintf(', <pval>=%5.4f+-%5.4f\n', mean(PP), std(PP));
else fprintf('\n'); end
```

Appendix B: Probe method for scoring causes & consequences

This appendix provides an algorithm to **compute the AUC for ROC curves** plotting hit rate vs. false alarm rate in the classification of **"relevant" vs. "irrelevant" variables**. "Relevancy" can take one of several meanings, including dependency to the target, causal relationships to the target, etc. The method is therefore applicable to **variable selection**, where relevant variables are those, which are predictive of a given outcome (e.g. a target variable), and irrelevant variables are not. It is also applicable to **causal discovery**, where a score can indicate causal proximity to the target, with the goal of separating e.g. causes from non-causes or direct causes from other variables.

The assumption we make is that **we do not know the truth values** of the variable classification (relevant vs. irrelevant) but **we know the "null distribution"** of irrelevant variables and we can draw as many artificial examples of such irrelevant variables as we want (we call them "probes"). It is assumed that an empirical variable ranking (from most relevant to least relevant) can be established using training data (samples of variable values) and an algorithm of our choice. For instance, such ranking may be established using a variable score, where a low score indicates that the variable is more likely to belong to one of the classes (e.g. the "relevant variables") and a high score that it belongs to the other (e.g. the "irrelevant ones"). Using the ranking method, we compute the AUC for sets of variables intermixed with "random probes", as an estimate of the AUC for the classification "relevant" vs. "irrelevant" variables.

The algorithm (Matlab implementation in Appendix B5)

The original data consists of a matrix of m lines (samples) and n_r columns (real variables). The n_r real variables include a n_+ positive examples ("relevant" variables) and a n_- negative ($n_r=n_++n_-$). It is not known which variables are relevant (truth values) nor how many of them are relevant, thus n_+ and n_- are not known.

1) A number n_p of artificial random variables called "probes" are drawn from an assumed "null distribution". In turn m samples are drawn from these probes and the resulting ($m \times n_p$) values are added to the original matrix to form an ($m \times (n_r+ n_p)$) matrix.
2) All real variables and probes are ranked with a given algorithm, in decreasing order of relevance (most relevant variables come first).
3) The sum of the ranks of the probes SPR is formed.
4) The area under the ROC curve for the data including probes is estimated as $PAUC= [SPR - n_p.(n_p+1)/2] / (n_p.n_r)$

In the asymptotic case of infinite number of real variables and probes, PAUC is linearly related to the AUC for the classification "relevant" vs. "irrelevant" variables:

$$PAUC = (n_+/n_r)\ AUC + 0.5\ (n_-/n_r).$$

This monotonic dependency allows us to use PAUC as a surrogate for the real AUC for algorithm comparison and model selection. In the finite sample case, we will use the following estimator of the PAUC standard deviation:

$$\sigma = 0.5\ sqrt[sen(1-sen)/ n_r + spe(1-spe)/ n_p]$$

where $spe = 1 – k/neg$, $sen = (r_k – k)/pos$, r_k is the rank of the k^{th} probe and k maximizes the average of sen and spe.

The algorithm is justified in what follows.

Calculation of the AUC and the Gini index

Assume we are given a ranked list of objects belonging to one of 2 classes, a positive and a negative class (for instance, causes and non-causes). We have "pos" examples of the positive class and "neg" examples of the negative class, and neg+pos=m=tot (the total number of examples).

We can compute, for each value of the rank:
fp: the number of false positive
tp: the number of true positive
fn: the number of false negatine
tn: the number of true negative.
We have tp+fn=pos and tn+fp=neg

We define:
fpr (false positive rate or false alarm rate)=fp/neg
fnr (false negative rate)=fn/pos
Hit rate=sensitivity=tp/pos=1-fnr
Specificity=tn/neg=1-fpr
sel (the fraction of selected up to the rank)=fp+tp

Figure B1 shows how these statistics relate to one another.
To avoid notation confusions, in what follows, if we are considering the real variables only, we use:
tot=n_r=number of real variables
pos=n_+=number of examples of the positive class
neg=n_-=number of the negative class
If we are adding probe variables, we use:
tot=n_r+n_p=number of variables including real and probes
pos=n_r=number of real variables
neg=n_p=number of probes

The ROC curve (Figure B2) plots the "hit rate" vs. the "false alarm rate" i.e. (1-fnr) vs. fpr. The AUC is the area under the ROC curve. Note that it is identical to the area under the curve plotting sensitivity (aka "hit rate") vs. specificity (1-fpr).

The lift curve (often used in marketing) plots "hit rate" vs. the fraction of selected "sel" (Figure B3). The Gini index is defined as the ratio M/O and it can be shown (Appendix B1) that **Gini = 2 AUC –1**.

This provides a means of computing efficiently the AUC using the area under the lift curve, because it is easy to compute the area under the lift curve. The area above the lift curve (AALC) can be upper and lower bounded by Lebesgue integrals using the sum of the ranks of the objects of the positive class (when those are sorted with the most relevant coming first) normalized by pos*tot (Figure B4):

[sum(ranks_of_pos)-pos]/(pos*tot)< AALC < sum(ranks_of_pos)/(pos*tot)
Hence the estimation of the AALC by the trapeze method :
AALC ~ [sum(ranks_of_pos)-pos/2]/(pos*tot)
Thus the area M is :
M = 0.5 - [sum(ranks_of_pos)-pos/2]/(pos*tot)
The area O is given by:
O = 0.5 - 0.5*pos/tot
Thus
Gini = M/O = {0.5 - [sum(ranks_of_pos)-pos/2]/(pos*tot)} / (0.5-0.5*pos/tot)
Gini = [pos*(tot+1)– 2 sum(ranks_of_pos)] / [pos*(tot-pos)]

Note that by symmetry with the negative class, we also have:

Gini = [neg*(tot+1) – 2 sum(ranks_of_neg)] / [neg*(tot-neg)]

where sum(ranks_of_neg) is the sum of the ranks of the negative class when the ranking is such that the most likely to be negative come first, i.e. the ranking is done in order of increasing probability of being « relevant ».

Note that we can sort one way or the other. For instance, if we sort in order of increasing probability of being « relevant » (object believe to be from the negative class come first) and compute the sum of the ranks of the positive class and call it Sp we can relate it to sum(ranks_of_pos), the quantity defined above when sorting in the other direction:
$\text{sum(ranks\_of\_pos)} = \text{sum}_{\text{\_pos}}(\text{tot} - j + 1) = \text{pos*tot} - \text{Sp} + \text{pos}$
thus
Gini = [pos*tot – 2 (pos*tot-Sp+pos) + pos] / [pos*(tot-pos)]
= [2 Sp – pos*(tot+1)] / (pos*neg)
and
AUC=(Gini+1)/2= 0.5 [2 Sp – pos*tot – pos + pos*tot – pos^2] / (pos*neg)

AUC = [Sp – pos*(pos+1)/2] / (pos*neg)

This last formula is the basis for the algorithm shown in Appendix B2.

We can also sort in decreasing order of relevance (most relevant objects believed to be from the positive class come first) and compute the sum of the ranks of the positive class and call it Sn. Similarly as before, we have :

Gini = [2 Sn – neg*(tot+1)] / [neg*(tot-neg)] (1)

Thus the alternative formula for the AUC:

AUC = [Sn – neg*(neg+1)/2] / (pos*neg) (2)

Relationship between ROC curve and negative lift curve

While in marketing the lift curve is the most useful way of visualizing the data, for our purpose, we rather focus on the negative class for the purpose of using the "probe" method. In Figure B7, we represent the non-normalized "negative lift curve", that is the number of false positive as a function of the total number of examples selected. The negative lift curve is obtained by normalizing the x axis by "tot" and the y axis by "neg".

If we change coordinates to the green axes, we obtain the non-normalized ROC curve. The ROC curve is obtained by normalizing the number of true positive by "pos" to get the sensitivity and the number of true negative by "neg" to get the specificity (Here we adopt as the definition of the ROC curve the plot "sensitivity vs. specificity, which has same AUC as hit rate vs. false alarm rate and is obtained by reversing the x axix of the ROC curve).

From this diagram, we easily see how the AUC relates to the area under the false positive rate A (negative lift) and the ideal negative lift A*. The AUC is the green shaded area, after normalizing by pos neg. hence:

AUC = (1-A-A*)(tot/pos).

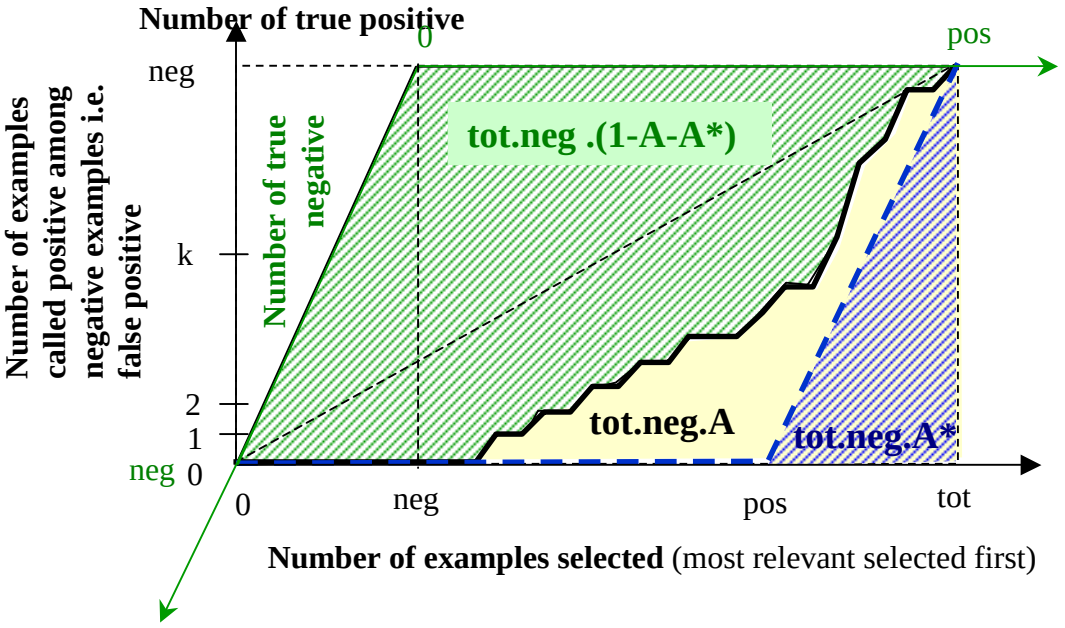


Figure B7: Relationship between ROC curve and lift curve.

From Figure B7, we also see that for a given point on the ROC curve, the sensitivity is given by:

specificity = 1 – k/neg (3)

sensitivity = (r_k – k)/pos (4)

where r_k is the rank of the k^{th} negative example in the example ordering, where most relevant come first.

We easily confirm Equation 2 with

AUC = (1/neg) $\Sigma_{k=1:neg}$ sensitivity (5)

AUC error bar

Many estimators of the AUC error bars have been proposed. Some are easy to compute but provide loose bounds, others are more accurate but very computationally expensive. For the purpose of the challenge, we propose to compromise and use an empirical formula easy to justify and which gives satisfactory results in numerical experiments.

We define the balanced accuracy (BAC) as:

$$BAC = 0.5(\text{sensitivity}+\text{specificity})$$

where, if we call tp the number of true positive and tn the number of true negative , we define sensitivity=tp/pos (accuracy of classification for positive examples) and specificity=tn/neg (accuracy of classification for negative examples). In the case where the score upon which the ranking is based is binary (e.g. hard classification decisions are used rather than a discriminant value), we have exactly AUC=BAC=1-BER, where BER is the balanced error rate defined as 1-BAC (see Appendix B3 for a proof).

The idea is to **approximate the AUC with the maximum BAC on the ROC curve** (Figure B8). Subsequently, we will use the BAC error bar to estimate the AUC error bar. From Equations (3) and (4) giving the sensitivity and specificity , we see that our approximation amounts to computing:

$$AUC \cong \max BAC = \max_k 0.5[(r_k - k)/pos + 1 - k/neg]$$

In what follows, we call k* the value of k maximizing BAC and sen and spe the corresponding values of the sensitivity and specificity.

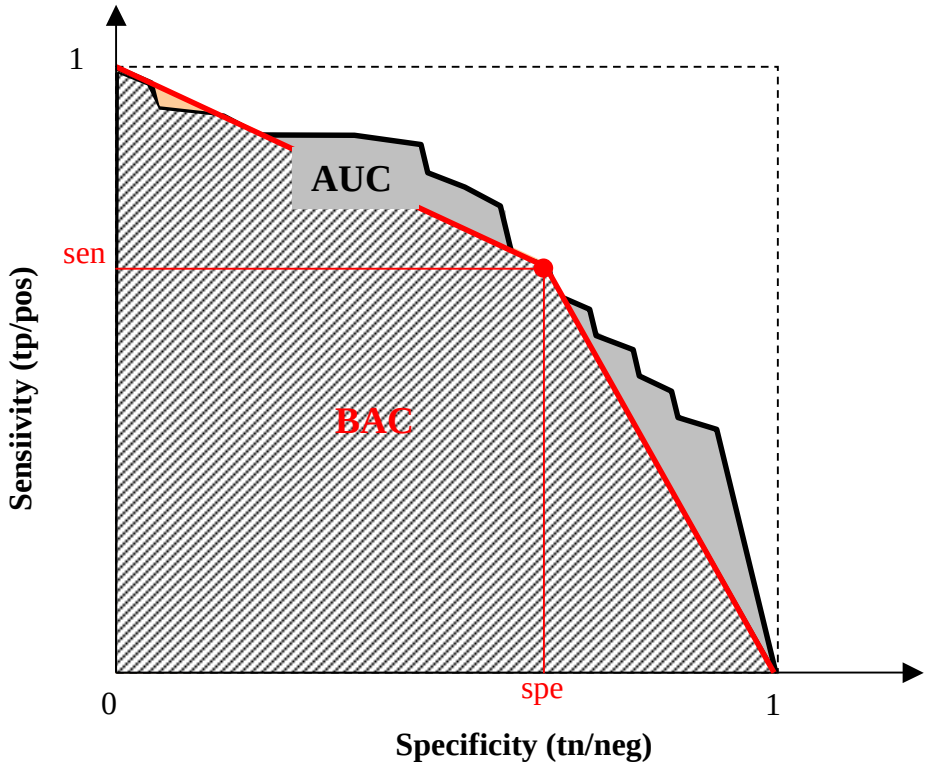


Figure B8: Approximation of the ROC curve. We replace the AUC by the largest BAC.

A BAC or a BER error bar estimator is obtained as follows. As is known, for i.i.d. errors corresponding to Bernouilli trials with a probability of error p, the standard deviation of the error rate E computed on a data set of size n is sqrt(p(1 - p)/n). This result can be adapted to the balanced error rate. Let us call pos the number of examples of the positive class, neg the number of examples of the negative class, p_+ the probability of error on examples of the positive class (one minus the expected value of the sensitivity), p_- the probability of error on examples of the negative class (one minus the expected value of the specificity), and E_+ and E- their corresponding empirical estimates. Both processes generating errors on the positive or negative class are Bernouilli processes. By definition, the balanced error rate is BER = (1/2)(E_++ E-), and its variance is var(BER)=(1/4)(var(E_+) + var(E-)). Therefore, the standard deviation of the BER (and that of the BAC) using n_+ and n_- examples is:

$\sigma = 0.5\ \mathrm{sqrt}(p_+(1-p_+)/pos + p_-(1-p_-)/neg)$

For sufficiently large data sets, we may substitute p_+ by E_+ and p_- by E- to compute σ.
Equivalently, since sensitivity=1- E_+ and specificity=1- E_+ we obtain the following estimator of the BAC standard deviation:

$\sigma = 0.5\ \mathrm{sqrt}(sen(1-sen)/pos + spe(1-spe)/neg)$ (6)

where we abbreviate sensitivity by sen end specificity by spe.

Application to the probe method

Assume that we are using the "probe" method and inject artificial probes, which are examples of the negative class for which we know the truth value (negative). The "real variables" may be either from the positive class or the negative class. Let us call:

n_r: the total number of real variables

n_p: the total number of probes

n_{sp}: the number of selected probes

It is common to plot the fraction of probes selected n_{sp}/n_p as a function of the number of variables selected (Figure B9).

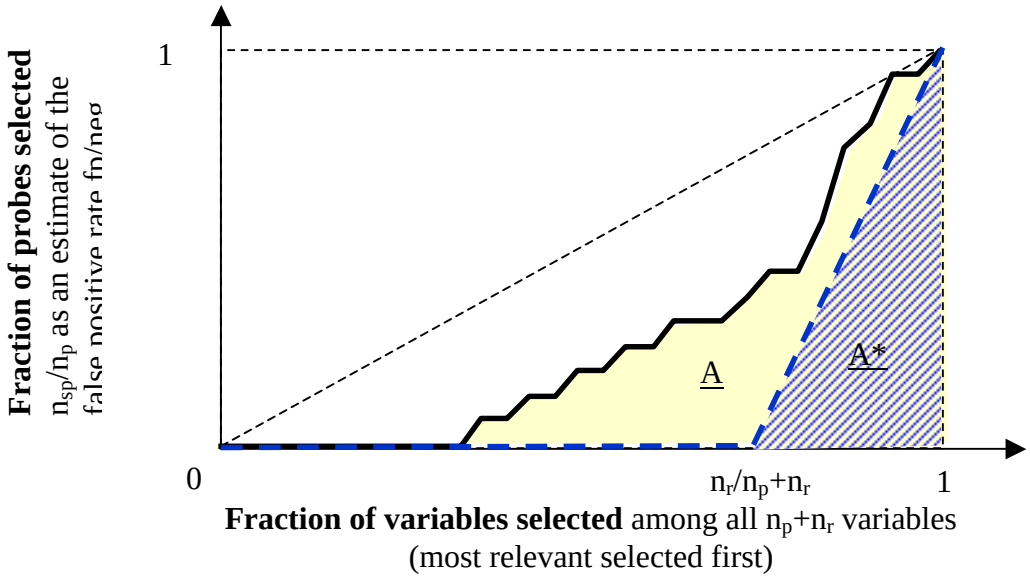


Figure B9: Area under the fraction of probe selected.

Doing that, we assume that we rank the variables in decreasing order of relevance (best first). We can define the sum of the rank of the probes SPR for that order.
The area A is given by
$A=1 - (SPR - n_p/2)/(n_p (n_p+n_r))$.
The area A* corresponding to the best achievable A (where all the probes show up last in the ranking) is given by
$A^*=0.5\ n_p/(n_p+n_r)=0.5(1- n_r/(n_p+n_r))$.
Using Equation (1), we get for the probes
$PGini = [2\ SPR - n_p.(n_p+n_r+1)] / (n_p.n_r)$,
therefore
$PGini=(1-2A)(n_p+n_r)/ n_r$
$PGini=(1-2A)/(1-2A^*)$
$PGini=(0.5-A)/(0.5-A^*)$
This last formula is equivalent to that of Figure B4.

Simply, the AUC for the probe method, which we call PAUC is obtained by computing the regular AUC for truth values +1 for all real variables and –1 for all probes (instead of +1 for the positive class variables and –1 for the negative class variables, in the absence of probes). Thus, the real variables become the positive class and the probes the negative class. For Equation (2), we get:
$PAUC= [SPR - n_p.(n_p+1)/2] / (n_p.n_r)$
We show in Appendix B4 that asymptotically (for an infinite number of examples and probes):
$PAUC= (n_+/n_r)\ AUC + 0.5\ n_-/n_r$
where AUC is the true AUC, which cannot be computed and n_+ and n_- are the unknown number for positive and negative examples for the real variables.

An error bar on PAUC is obtained in the finite sample case from Equation (6):
$\sigma = 0.5\ sqrt(sen(1-sen)/pos + spe(1-spe)/neg)$

with (from Equations (3) and (4))
$spe = 1 – k/neg$
$sen = (r_k – k)/pos$
for the value of k, which maximizes: BAC = 0.5(sen+spe).

The error BAR on PAUC may be use to determine the significance of the difference between two ranking methods yielding values of PAUC P_1 and P_2 and corresponding standard deviations σ_1 and σ_2**.** The difference will be called significant e.g. if
$abs(P_1-P_2) > 2\ sqrt(\sigma_1^2 + \sigma_2^2)$.

Numerical simulations:

We illustrate the result PAUC = (n_+/n_r) AUC + 0.5 (n_-/n_r) with some simple numerical simulation. The code is reported in Appendix B6. In this example, we have 2000 "real variables" and 2000 "probes". We vary the fraction of positive examples (n_+/n_r) and compute a noisy score for variables as

```
D=Y+0.5*randn(size(Y))+k*noise*randn(size(Y));
```

From D we compute the "real" AUC and PAUC. We plot PAUC as a function of AUC (Figure B10).

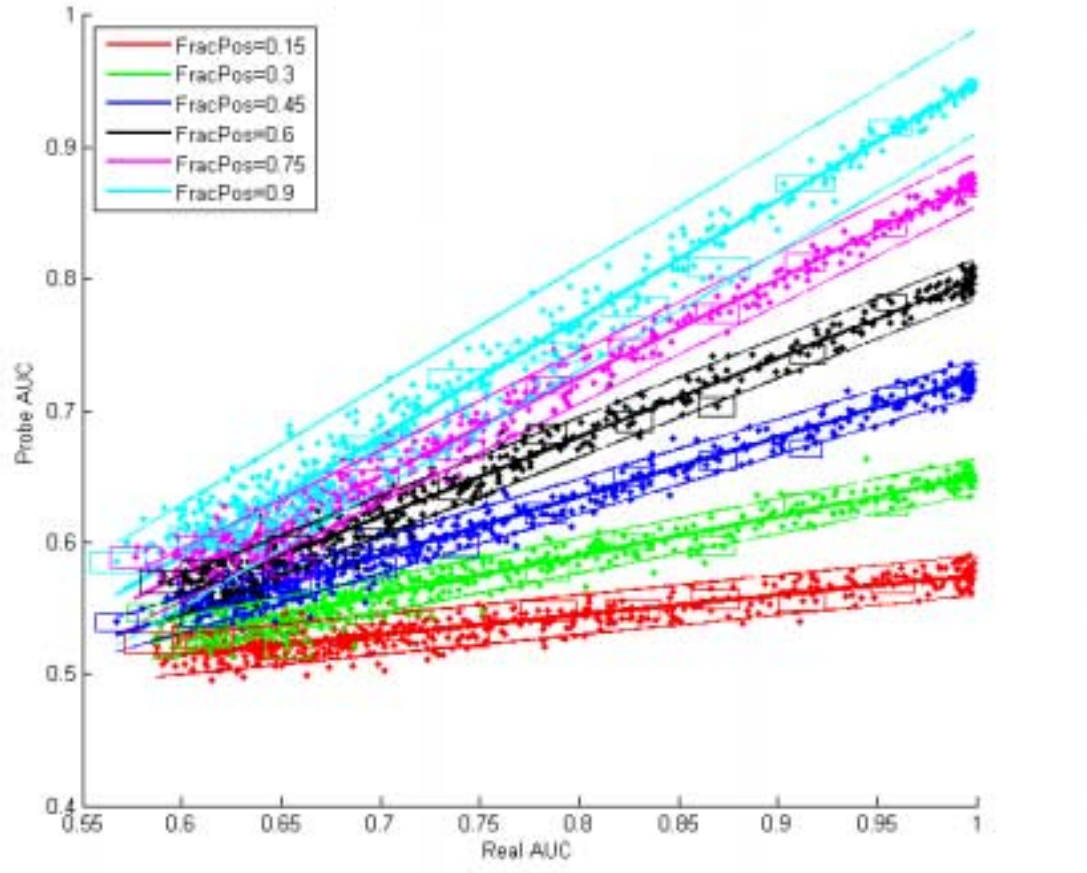


Figure B10: Relationship between the real AUC and that estimated by the probe method. The dots represent samples of the pairs {"real"AUC, PAUC} for various fractions of positive examples, when 2000 real variables and 2000 probes are used. The boxes show the 1 sigma error bars for AUC and PAUC. The thick lines are plots of $y=(n_+/n_r)$ AUC + 0.5 (n_-/n_r). The thin lines indicate the estimated error tube.

We verified that for all fractions of positive examples considered the regression coefficients match closely the theoretical values obtained in the final sample size limit:
1) frac_pos=0.15, frac_neg=0.85, w=0.148399, 2*b=0.851758
2) frac_pos=0.3, frac_neg=0.7, w=0.297821, 2*b=0.702991

3) frac_pos=0.45, frac_neg=0.55, w=0.445261, 2*b=0.556114
4) frac_pos=0.6, frac_neg=0.4, w=0.601203, 2*b=0.398861
5) frac_pos=0.75, frac_neg=0.25, w=0.737146, 2*b=0.271517
6) frac_pos=0.9, frac_neg=0.1, w=0.8821, 2*b=0.124806

In the finite sample case, we have errors both on the estimates of both AUC and PAUC. Note that in practice we would not be able to compute the "real" AUC. Still, to verify the validity of our error bar estimates, we can use it here. We compute:

- The average empirical sigma as the average distance of the points to the line $y=(n_+/n_r)\ AUC + 0.5\ (n_-/n_r)$.
- The average theoretical sigma as the sigma_th=mean(sqrt($\sigma_{AUC}^2 + \sigma_{PAUC}^2$)), computing σ_{AUC} and σ_{PAUC} with formula (6).

1) Average empirical sigma=0.0089, Average theoretical sigma=0.0149
2) Average empirical sigma=0.0091, Average theoretical sigma=0.0127
3) Average empirical sigma=0.0085, Average theoretical sigma=0.0121
4) Average empirical sigma=0.0092, Average theoretical sigma=0.0122
5) Average empirical sigma=0.0095, Average theoretical sigma=0.0132
6) Average empirical sigma=0.0126, Average theoretical sigma=0.0171

We see that our estimate is slightly pessimistic, but gives the right order of magnitude.
To visualize our error bar estimates, we drew boxes of sides $2\ \sigma_{AUC}$ x $2\ \sigma_{PAUC}$ around a few points. The box usually overlaps with the thick line. We also drew thin lines at sigma_th/cos(alpha), where alpha is the slope of the line. This allows us to draw an error bar taking into account both the error for estimating AUC and that for estimating PAUC.

Appendix B1:

Proof of Gini = 2 AUC –1

L = lift

Hitrate = tp/pos

Farate = fp/neg

Selected = sel/tot = (tp+fp)/tot = pos/tot.tp/pos + neg/tot.fp/neg = pos/tot Hitrate + neg/tot Farate

AUC = sum Hitrate d(Farate)

L = sum Hitrate d(Selected)

= sum Hitrate d(pos/tot Hitrate + neg/tot Farate)

= pos/tot sum Hitrate d Hitrate + neg/tot sum Hitrate d Farate

= ½ pos/tot + neg/tot AUC

2L-1 = -(1-pos/tot) + 2(1-pos/tot) AUC = (1-pos/tot) (2AUC-1)

Gini = (L-1/2)/(1-pos/tot)/2

= (2L-1)/(1-pos/tot) = 2AUC-1

Appendix B2

```
function area = auc(Output, Target)
%area = auc(Output, Target)
% This computation gives the same results as the AUC when there are no
% ties.
% Inputs:
%  Output -- Matrix of classifier discriminant values of dim (num
pattern, num tries)
%  Target -- Vector of corresponding +-1 target values.
%  Returns:
%  area -- Area under the ROC curve.
% We still need to work out the case of ties.
% From Hollander and Wolfe pp 107 & 117.

% Isabelle Guyon -- isabelle@clopinet.com -- June 2005

% Compute the Wilcoxon statistic
midx=find(Target<0);
nidx=find(Target>0);
m=length(midx);
n=length(nidx);
[u,i]=sort(Output);
S(i)=1:n+m;
W=sum(S(nidx));

% Compute the Mann-Withney statistic
U=W-n*(n+1)/2;

% Compute the AUC
area=U/(m*n);
```

Appendix B3

Demonstration that AUC=1-BER in the case of binary outputs.

Assume that the outputs are binary ±1 instead of being discriminant values. Then we have the following situation for the histogram of output values:

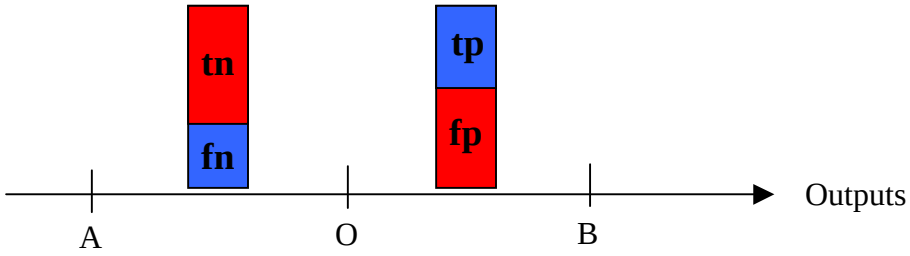


Figure B6a

With at mid point the sensitivity and specificity given by $Sen_0=tp/(tp+fn)$ and the specificity given by $Spe_0=tn/(tn+fp)$.

We have the following ROC curve:

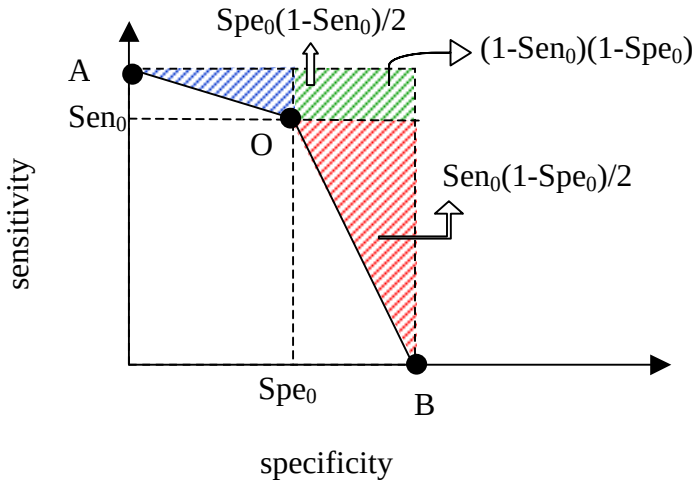


Figure B6b

Therefore,

$1\text{-AUC} = Spe_0(1-Sen_0)/2 + Sen_0(1-Spe_0)/2 + (1-Sen_0)(1-Spe_0)$

$= (1/2)(Spe_0 - Spe_0\, Sen_0 + Sen_0 - Spe_0\, Sen_0 + 2 - 2\, Spe_0 - 2\, Sen_0 + 2\, Spe_0\, Sen_0)$

$= 1- (Spe_0 + Sen_0)/2$

$= BER$ □

Since we have BER=1-BAC, we deduce that BAC=AUC.

Appendix B4

Proof of PAUC= (n_+/n_r) AUC + 0.5 n_-/n_r

From Equations (4) and (5), we have:

$AUC = (1/neg) \sum_{k=1:neg} (r_k - k)/pos$

which for the real variables gives:

$n_+ AUC = (1/n_-) \sum_{k=1:n-} (r_k - k)$

and for the combination of real variables and probes gives:

$n_r PAUC = (1/n_p) \sum_{k=1:np} (r_k^{(probe)} - k)$

We assume that the negative examples and the probes are drawn from the same distribution. Consider the case where real variables and probes are intermixed. On average, half of the negative example fall before the mean value of the rank of the probe and half after. Hence, if we call r_k the rank of a probe if there were no negative examples (hence only positive examples and probes), we have on average over all possible drawings of negative examples and probes:

$$(1/n_p) \sum_{k=1:np} (r_k^{(probe)}) = (1/n_p) \sum_{k=1:np} (r_k) = (1/n_p) + neg/2.$$

In the limit of infinite number of probes and negative examples we define the 2 following quantities:

$A_r = \lim_{n-\to\infty} (1/n_-) \sum_{k=1:n-} (r_k - k)$

$A_p = \lim_{np\to\infty} (1/n_p) \sum_{k=1:np} (r_k^{(probe)} - k) = \lim_{np\to\infty} (1/n_p) \sum_{k=1:np} (r_k - k) + neg/2$

Thus $A_p = A_r + neg/2$

In the limit of infinite number of probes and negative examples

$n_+ AUC = A_r$ and $n_r PAUC = A_p$

therefore

$n_r PAUC = A_p = A_r + neg/2 = n_+ AUC + neg/2$

and

$PAUC = (n_+/n_r) AUC + 0.5\, n_-/n_r$ €

We get a similar result with the BAC:

$BAC = 0.5(tp/n_+ + tn/n_-) = 0.5(sen+spe)$

$PBAC = 0.5((tp+fp)/n_r + (n_p - n_{sp})/n_p)$

Let us call SEN and SPE the expected value of the sensitivity and specificity.

We have:

$SPE = \lim_{n-\to\infty} tn/n_- = \lim_{np\to\infty} (n_p - n_{sp})/n_p$

$SEN = \lim_{n+\to\infty} tp/n_+$

$\lim_{np\to\infty} PBAC = 0.5\, ((tp+fp)/n_r + SPE$

$\lim_{np\to\infty} \lim_{n-\to\infty} PBAC = 0.5\, ((SEN\, n_+ + (n_- - tn))/n_r + SPE$

$\lim_{np\to\infty} \lim_{n-\to\infty} \lim_{n+\to\infty} PBAC = 0.5\, ((SEN\, n_+ + (1 - SPE)\, n_-)/n_r + SPE$

$= 0.5\, (n_+/n_r)\, (SEN+SPE) + 0.5\, n_-/n_r$

$= (n_+/n_r) \lim_{n-\to\infty} \lim_{n+\to\infty} BAC + 0.5\, n_-/n_r$

Appendix B5: The final algorithm

```
function [area, sigma] = auc2(Output, Target, pos_small)
%[area, sigma] = auc2(Output, Target, pos_small)
% This is the algorithm proposed for
% computing the AUC and the error bar.
% It is assumed that the outputs provide a score
% with the negative examples having the lowest score
% unless the flag pos_small = 1.

% Isabelle Guyon -- isabelle@clopinet.com -- November 2007

if nargin<3, pos_small=0; end
if ~pos_small, Output=-Output; end

negidx=find(Target<0);
posidx=find(Target>0);
neg=length(negidx);
pos=length(posidx);
[u,i]=sort(Output); % best come first
S(i)=1:(neg+pos);
SEN=(sort(S(negidx))-[1:neg])/pos;
SPE=1-(1:neg)/neg;
area=sum(SEN)/neg;

two_BAC=SEN+SPE;
[u,k]=max(two_BAC);
sen=SEN(k);
spe=SPE(k);
sigma= 0.5 * sqrt(sen*(1-sen)/ pos + spe*(1-spe)/ neg);
```

Appendix B6: Numerical simulations

```
% We verify the formula AUC = frac_pos PAUC + 0.5 frac_neg
% in the large sample size limit.

col='rgbkmc';
noise=0.01;
probe_num=2000;
real_num=2000;
N=probe_num+probe_num;
fp={};
repeat_num=500;
AReal=zeros(repeat_num, length(col));
AProbe=zeros(repeat_num, length(col));
EReal=zeros(repeat_num, length(col));
EProbe=zeros(repeat_num, length(col));
frac_pos=zeros(length(col),1);
frac_neg=zeros(length(col),1);
for j=1:length(col)
    frac_pos(j)=0.15*j;
    fp{j}=['FracPos=' num2str(frac_pos(j))];
    frac_neg(j)=1-frac_pos(j);
    pos_num=real_num*frac_pos(j);
    neg_num=real_num*frac_neg(j);
```

```
    Y=ones(N,1);
    probe_idx=1:probe_num;
    neg_idx=probe_num+1:probe_num+neg_num;
    pos_idx=probe_num+neg_num+1:N;
    real_idx=[neg_idx, pos_idx];
    Y(probe_idx)=-1;
    Y(neg_idx)=-1;

    Yprobe=ones(N,1);
    Yprobe(probe_idx)=-1;
    for k=1:repeat_num
        % Compute a fake discriminant value correlated with Y
        D=Y+0.5*randn(size(Y))+k*noise*randn(size(Y));
        Dreal=D(real_idx);
        Yreal=Y(real_idx);

        [Aprobe, Eprobe]=auc2(D, Yprobe);
        [Areal, Ereal]=auc2(Dreal, Yreal);
        AReal(k,j)=Areal;
        AProbe(k,j)=Aprobe;
        EReal(k,j)=Ereal;
        EProbe(k,j)=Eprobe;
    end
    % Linear fit
    w=[AReal(:,j), ones(size(AReal(:,j)))]\AProbe(:,j);
    Probe_hat=[AReal(:,j), ones(size(AReal(:,j)))]*w;
    fprintf('frac_pos=%g, frac_neg=%g, w=%g, 2*b=%g\n', frac_pos(j),
frac_neg(j), w(1), 2*w(2));
end

figure; hold on
for j=1:length(col)
    plot(AReal(:,j), frac_pos(j) * AReal(:,j) + 0.5 *
frac_neg(j),[col(j) '-'] );
end
legend(fp, 'Location', 'NorthWest');
for j=1:length(col)
    plot(AReal(:,j), AProbe(:,j), [col(j) '.']); xlabel('Real AUC');
ylabel('Probe AUC');
end
% Take at random a few points and draw the error box
Mini=min(min(AReal));
Maxi=1;
div=10;
vals=[Mini:(Maxi-Mini)/div:Maxi];
for j=1:length(col)
    for k=1:div+1
        [m,i]=min(abs(AReal(:,j)-vals(k)));
        xm=AReal(i,j)-EReal(i,j);
        xM=AReal(i,j)+EReal(i,j);
        ym=AProbe(i,j)-EProbe(i,j);
        yM=AProbe(i,j)+EProbe(i,j);
        plot([xm, xM, xM, xm, xm], [yM, yM, ym, ym, yM], [col(j) '-']);
    end
end
```

```
% Note: it is normal that empirical is larger then theoretical because
of
% the uncertainty on AReal
% Other method
Ediag=zeros(size(EProbe));
for j=1:length(col)
    % Compute the average distance to the line
    W1=frac_pos(j);
    W2=-1;
    N=sqrt(W1.^2+W2.^2);
    W1=W1./N;
    W2=W2./N;
    W0=0.5 * frac_neg(j)./N;
    sigma_emp=sqrt(mean((W1 * AReal(:,j) + W2 * AProbe(:,j) + W0).^2));
    sigma_th=mean(sqrt(EReal(:,j).^2+EProbe(:,j).^2));
    fprintf('Average empirical sigma=%5.4f, Average theoretical
sigma=%5.4f\n', sigma_emp, sigma_th);
    alpha=asin(frac_pos(j));
    Eboth(:,j)= sigma_th/cos(alpha); % Correct for the uncertainty of
real AUC
end

figure; hold on
for j=1:length(col)
    plot(AReal(:,j), frac_pos(j) * AReal(:,j) + 0.5 *
frac_neg(j),[col(j) '-'], 'LineWidth', 2 );
end
legend(fp, 'Location', 'NorthWest');
for j=1:length(col)
    plot(AReal(:,j), AProbe(:,j), [col(j) '.']); xlabel('Real AUC');
ylabel('Probe AUC');
end
% Draw a tube
for j=1:length(col)
    emed=Eboth(:,j);
    plot(AReal(:,j), emed+(frac_pos(j) * AReal(:,j) + 0.5 *
frac_neg(j)),[col(j) '--'] );
    plot(AReal(:,j), -emed+(frac_pos(j) * AReal(:,j) + 0.5 *
frac_neg(j)),[col(j) '--'] );
end
```

Performance Assessment

Cost matrix		Predictions: F(x)			
		Class -1	Class +1	Total	Class +1 / Total
Truth: y	Class -1	tn	fp	neg=tn+fp	False alarm = fp/neg
	Class +1	fn	tp	pos=fn+tp	Hit rate = tp/pos
	Total	rej=tn+fn	sel=fp+tp	m=tn+fp +fn+tp	Frac. selected = sel/m
	Class+1 /Total		Precision = tp/sel	False alarm rate = type I errate = 1-specificity Hit rate = 1-type II errate = sensitivity = recall = test power	

Compare F(x) = sign(f(x)) to the target y, and report:

- **Error rate = (fn + fp)/m**
- **{Hit rate , False alarm rate} or {Hit rate , Precision} or {Hit rate , Frac.selected}**
- **Balanced error rate (BER) = (fn/pos + fp/neg)/2 = 1 – (sensitivity+specificity)/2**
- **F measure = 2 precision.recall/(precision+recall)**

Vary the decision threshold F(x) = sign(f(x)+θ) and plot:

- **ROC curve: Hit rate *vs.* False alarm rate**
- **Lift curve: Hit rate *vs.* Fraction selected**
- **Precision/recall curve: Hit rate *vs.* Precision**

Figure B1

ROC Curve

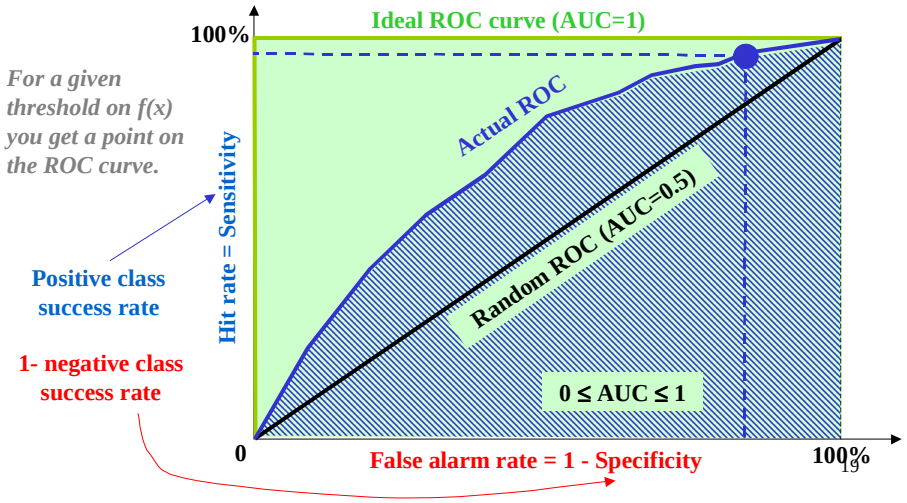


Figure B2

Lift Curve

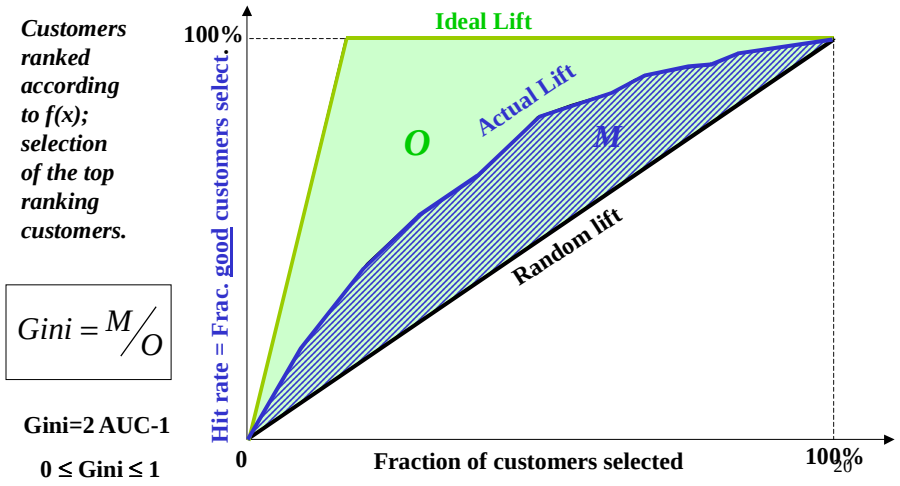


Figure B3

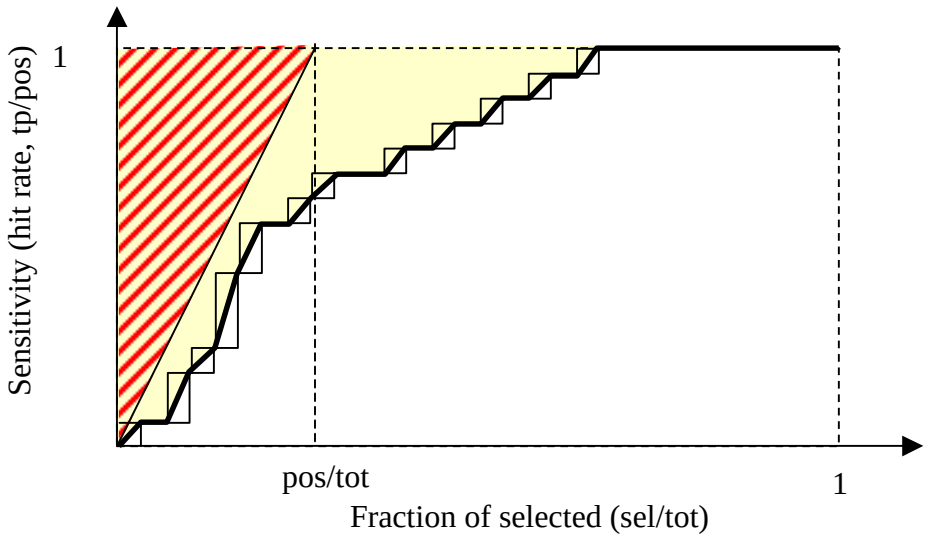


Figure B4: The yellow area represents the area above the lift curve. It is estimated by the trapeze method as [sum(rank_of_pos)-pos/2] /(pos*tot). The red shaded area is the area above the ideal lift, which is equal to 0.5 pos/tot.

False Discovery Rate

$$FDR = n_{fp}/n_{sc}$$

fp=false positive=features falsely found relevant
sc= selected candidate features

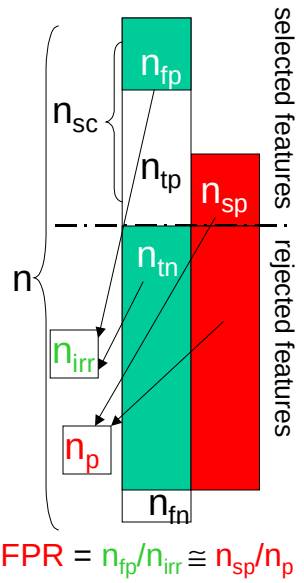
n_{fp} is unknown, but FPR can be calculated from pval or using the probe method.
Bound the FDR:

$FPR = n_{fp}/n_{irr} \geq n_{fp}/n$ (irr=irrelevant feat.)

$FDR = (n_{fp}/n)\ (n/n_{sc}) \leq FPR\ n/n_{sc}$

$$FDR \leq FPR\ n/n_{sc} \leq \alpha$$

We obtain $FPR \leq \alpha\ n_{sc}/n$, intermediate between $FPR \leq \alpha$ and $FPR \leq \alpha/n$.



$FPR = n_{fp}/n_{irr} \cong n_{sp}/n_p$

Figure B5

Appendix C: ChemTK QSAR descriptors used for SIDO

ChemTK generates a variety of descriptors or keys, which are used as features in the SIDO dataset. These include properties such as molecular weight, hydrogen-bond donor and acceptor counts, and rotatable bond counts, for a particular data set. Other keys include 2- and 3-point pharmacophore descriptors. We provide below *excerpts of the ChemTK manual to help decoding the key symbols.*

Smarts keys

ChemTK supports the Smarts query language developed by Daylight CIS, Inc. That company's website (www.daylight.com) provides an excellent tutorial for the Smarts language, so details of the syntax will not be provided here. Note that Boolean queries (e.g., "[c,n;!D3]") and recursive queries (e.g., "[$(C=O)]") are both supported.

In addition, users may specify named-property queries (see below) within a Smarts pattern using the angled bracket syntax (<>). For instance, the query "<HAcc>~C~<HAcc>" can be used to search for two hydrogen-bond acceptors connected via a single Carbon atom. The query names can correspond either to ChemTK defaults, or to user-defined queries as described below. Note that any named-property queries are assumed to describe single atoms; if multi-atom queries are used, only the information pertaining to the first atom will be retained.

Pharmacophore keys

ChemTK uses a type of pharmacophore that measures distance via bond connectivity rather than a typical three-dimensional distance. For instance, to describe a hydrogen-bond acceptor and hydrogen-bond donor separated by five connecting bonds, the corresponding key string would be "HAcc.HDon.5". More generally, a ChemTK pharmacophore contains two components: a list of features (hydrogen-bond acceptor, donor, etc.) and a list of pairwise distances measured in bond counts. Thus a 3-point pharmacophore has three features and three distances, while a 4-point pharmacophore has four features and six distances. In a key string used to represent a pharmacophore, all elements of the pharmacophore are separated by ".". Thus the following are examples of valid pharmacophore strings:

HAcc.HAcc.1
HDon.HDon.ExtArom.2.2.2
HAcc.HDon.Pos.ExtRing.2.5.2.3.1.2

The order in which the bond-based distances are listed in the above examples must correspond exactly to the order of the listed features, and should reflect the order of pairwise iteration. Hence in the third example, the six consecutive distances correspond to the following feature pairs: HAcc-HDon, HAcc-Pos, HAcc-ExtRing, HDon-Pos, HDon-ExtRing, Pos-ExtRing.

The following are included in the default feature list:

HAcc. Hydrogen-bond acceptor.
HDon. Hydrogen-bond donor.
Neg. Explicit negative charge.
Pos. Explicit positive charge.
ExtRing. Ring atom having a neighbor atom external to the ring.

ExtArom. Aromatic ring atom having a neighbor atom external to the ring.
ExtAliph. Aliphatic ring atom having a neighbor atom external to the ring.

Ring System keys

This option yields a key set comprised of all unique ring systems contained in the set of input molecules. A ring system is defined as any number of single or fused rings connected by an unbroken chain of atoms. The simplest example would be either a single ring (e.g., benzene) or a single fused system (e.g., naphthalene). A more complex case would be these same two example systems (benzene and naphthalene) connected together by a chain of carbon atoms. Any connecting chains must be devoid of any terminating branches (such as a carbonyl group), such that all atoms in the final ring-system structure will always be connected to at least two other atoms. The user can place lower and upper bounds on the number of individual rings allowed in a ring system.
Keys are distinguished using atom type and aromaticity only, and are represented using a notation similar to Daylight Smarts. Thus a benzene ring would be represented using the notation "c1ccccc1". Counts of rings in a system are based on the typical SSSR definition, whereby, for example, benzene has one ring, naphthalene two, and a basic steroid scaffold four.

Unbranched Fragment keys

This option yields a key set comprised of all unique non-branching fragments contained in the set of input molecules. The user must specify a maximum and minimum size (in atoms) for all fragments. Keys are distinguished using atom type and aromaticity only, and are represented using a notation similar to Daylight Smarts. Thus, using the aniline molecule as an example, the two-atom keys are "cc" and "cN"; the three-atom keys are "ccc" and "ccN"; and so forth.
ChemTK may generate a special type of unbranched fragments called **isotopic fragments**. Keys of this type are annotated with special symbols that describe the precise ring topology of the fragment. For example a simple non-annotated key such as "cc" describes two aromatic carbon atoms connected by a single or aromatic bond. In contrast, the annotated key "[c;i1][c;i2]" describes a similar fragment, but also specifies a requirement that the first atom belong to a single ring (arbitrarily labeled 1) and that the second atom belong to a single ring **different** from the first. While the first key could equally match a single benzene ring, the juncture atoms of a naphthalene system, or the bridge atoms in a bi-phenyl structure, the latter key can match only the third example, since only in that case do the two atoms belong to single and distinct rings. Note that given a particular set of bounds on fragment size, the number of isotopic fragments is ordinarily far greater than the number of standard unbranched fragments, and the time required for key generation is correspondingly greater as well.

Branched Fragment keys

This option yields a key set comprised of all unique branched fragments contained in the set of input molecules. This method is intended to provide keys having a richer, more complex description than those available through the Ring System and Unbranched Fragment approaches. A detailed definition of branched fragment will not be provided in this Reference Guide. Briefly, each fragment is constructed through an "assembly" of

shortest-path unbranched fragments, where each of the latter is required to be bounded by two atoms belonging to one or more pre-defined “terminal-atom” types selected by the user. The following options are available as terminal-atom types:
C. Non-aromatic Carbon atom.
c. Aromatic Carbon atom.
N. Non-aromatic Nitrogen atom.
n. Aromatic Nitrogen atom.
O. Non-aromatic Oxygen atom.
o. Aromatic Oxygen atom.
S. Non-aromatic Sulfur atom.
s. Aromatic Sulfur atom.
P. Non-aromatic Phosphorus atom.
HAcc. Hydrogen-bond acceptor.
HDon. Hydrogen-bond donor.
ExtRing. Ring atom having a neighbor atom external to the ring.
ExtArom. Aromatic ring atom having a neighbor atom external to the ring.
ExtAliph. Aliphatic ring atom having a neighbor atom external to the ring.
Some of these feature types are discussed in the section on **Pharmacophore** keys, as well as in the **Query Formats** section of this Reference Guide. In particular, see the discussion in the former section regarding **group features**, of which ExtRing, ExtArom, and ExtAliph are examples. The treatment of group features in the branched-fragment method is somewhat different from their treatment in the pharmacophore approach. Here, the requirement is that any branched fragment containing a group feature must contain no other feature that intersects the atoms described by the group feature. Thus, a branched-fragment key containing the ExtArom feature might validly describe a pyridine-containing molecule, but it is invalid for the features ExtArom and n (aromatic Nitrogen) to simultaneously describe the same pyridine ring: the ExtArom group feature excludes all other features from hitting the ring.
Branched-fragment keys are distinguished using atom type and aromaticity only, and are represented using a notation similar to Daylight Smarts. All feature atoms are designated using the angled brackets (“<>”) notation. Thus a key using the features O, N and ExtArom might have a representation similar to the following: “[<O>]CC[<N>]CC[<ExtArom>]1ccccc1”.

Named-property keys

Act. Molecule activity. Note that if a user does not specify an activity field when an SD file is first opened, a value of zero is stored.
ArRing. Aromatic ring atom. [**Group feature**].
ExtRing. Ring atom having a neighbor atom external to the ring. [**Group feature**].
ExtArom. Aromatic ring atom having a neighbor atom external to the ring. [**Group feature**].
ExtAliph. Aliphatic ring atom having a neighbor atom external to the ring. [**Group feature**].
HAcc. Hydrogen-bond acceptor atom.
HDon. Hydrogen-bond donor atom.
MolWeight. Molecular weight.

Neg. Explicit negative charge.
Pos. Explicit positive charge.
Ring. Ring atom. [**Group feature**].
RotBond. Rotatable bond.
Generally, these names are used in conjunction with the "NPQ" tag to form an **encoded query** (discussed in a following section). For instance, the query "NPQ MolWeight < 500" can be used to search a document for all molecules having a molecular weight less than 500. Named-property queries can also be used within pharmacophore, branched-fragment, and Smarts queries, as described in those sections.
The named-property queries labeled as **group features** in the above list all describe more than a single atom. Such queries have special significance when used in the definitions of pharmacophore and branched-fragment keys. See those topics in the **Generating Keys** section for a description of how group features are handled in each case. Note that by using recursive Smarts syntax, a user can create a query that includes more than one atom in the definition but is not a group feature. For instance, the query "[$(c1ccccc1)]" cannot define a group feature since it is actually a single-atom query. In contrast, the similar query "c1ccccc1" is a six-atom query and therefore can define a group feature.
Users also have the option of defining new named-property queries, using the **Define Named Queries** option on the **Search** menu. When a new query is defined, the query name is stored internally along with the default names, and can be used in an identical fashion. ChemTK remembers the name until the application is closed, after which time the query must be re-loaded by the user. To define a new query, it is first necessary to create a special type of query file containing the query name and its definition. The **Create Query File** option on the **Search** menu contains an option to save a query file having the appropriate syntax. The following is an example of such a query file (containing two queries):

```
NAMEQUERY ZincBinder
SLQ ONC=O
$$$$
NAMEQUERY KeyScaffold
SLQ n1ccccc1
$$$$
```

In the first line of each query entry, the NAMEQUERY tag indicates that a new named-property query is being defined. The subsequent name (e.g., "ZincBinder") is the name by which the new query will be referenced, just as "MolWeight" is used to reference the molecular-weight query. The second line defines the query itself, and like all query-file records is in the form of an encoded query. Thus "SLQ" indicates a Smarts pattern, and (for example) "ONC=O" is the pattern for a hydroxamic acid group. See the subsequent discussion of encoded queries, and the section on **File Formats**, for additional information on these topics.
Once an appropriate query file has been generated, it is loaded using the **Load** button on the **Define named queries** dialog. At this point the new queries should appear in the window along with the defaults. Note that a user may elect to override these default named-property queries. For example, the name "HAcc" could be used in the above NAMEQUERY entry, in which case the new definition would take precedence over the ChemTK default.

Encoded queries

One of the principal query formats supported by ChemTK is the **encoded query**. This internal format provides users considerable flexibility in creating queries based on substructure, pharmacophore signature, and molecular property, with the additional option for Boolean logic and range specification (e.g., a molecular weight range). ChemTK supports the use of encoded queries in document searches and in key generation, and these queries serve as the format for individual records within query files. Each of these topics is covered in a relevant section of this Reference Guide.

A single encoded query is constructed by writing one or more encoded-query **primitives**, separated by **logical operators**. The format for an encoded-query primitive is:

QUERY_CODE QUERY_STRING SEARCH_CODE RANGE

The individual elements, which may be separated by any type of whitespace (excluding newline), are discussed below.

QUERY_CODE. This is a three-letter code that specifies the particular type of query that is being requested (e.g., a Smarts query). The following list provides the most important of the supported codes. A few additional codes, including **RSQ**, **SGQ** and **NULL**, are used internally and are not described.

NPQ. Specifies a named-property query. This code indicates that the subsequent query string will be a special name recognized by ChemTK as a synonym for a molecular property. Recognized names include MolWeight (molecular weight), Act (activity), HAcc (hydrogen-bond acceptor), as well as any names defined by the user. See the earlier discussion of named-property queries in this section of the Reference Guide for the complete list of supported query names and for instructions on defining customized named-property queries.

SLQ. Specifies a query based on the Smarts syntax defined by Daylight CIS, Inc. See www.daylight.com for a detailed tutorial on the Smarts language. Note that the SLQ code also permits specification of named-property queries, using the angled bracket syntax (<>). For instance, the string "<HAcc>~C~<HDon>" specifies a hydrogen-bond donor and acceptor group, separated by a single Carbon atom. See the discussion of Smarts queries in the **Query Formats** section for more detail.

PHQ. Specifies a pharmacophore query. An example query is "HAcc.HDon.2", which specifies a hydrogen-bond donor and acceptor, separated by two bonds. See the earlier sections (e.g., **Generating Keys**) for a discussion of the required syntax.

NAQ. Specifies a numeric-attribute query. This code indicates that the subsequent query string will be the name of a numeric molecule attribute previously loaded by the user. For instance, if a user has loaded a numeric attribute named "cLogP," he can search against this property using a syntax such as "NAQ cLogP > 0.5 < 3.5". Note that molecules not having the requested attribute (e.g., missing values) are treated as having a value of 0.0 when query results are derived. For more information regarding molecule attributes, see the section of this Reference Guide entitled **Loading Molecule Activities/Attributes**.

QUERY_STRING. This is the actual query, written in a format appropriate for the preceding query code. Hence "MolWeight" would be an appropriate query string for the "NPQ" code, while "c1ccccc1" (benzene) would be an appropriate query string for the "SLQ" code.

SEARCH_CODE. (Optional). One of three single-letter codes to indicate whether to perform a **full** match ("F"), **single** match ("S"), or **unique** match ("U"). For instance, if a benzene query is used to search a naphthalene molecule, a full match will return 24 hits, a unique match two hits, and a single match one hit. Note that if no code is provided, the default is to perform a full match.

RANGE. (Optional). A range specification applied to the match result. This range must take the form of either one or two symbol/value pairs, where the symbol is either ">", "<", or "=". For example, "= 3" indicates a result equal to 3, and "> 1 < 4" indicates a result that is greater than 1 and less than 4. The match-result value depends on the type of match requested. For example, "NPQ MolWeight" returns an actual weight value, while "SLQ c1ccccc1" returns the number of benzene matches. This convention is identical to that of key-value calculations; see the relevant discussion in the **Generating Keys** section for more detail.

The following are examples of valid encoded-query primitives:

NPQ MolWeight < 500
NPQ HAcc > 2 < 11
SLQ c1ccccc1 U = 2
SLQ <HDon>~<HAcc> S
PHQ HAcc.HAcc.HAcc.2.2.2
50

The first query specifies molecules having a molecular weight less than 500. The second query specifies molecules having between 3 and 10 hydrogen-bond acceptor groups. The third query specifies molecules having two distinct benzene rings. Note that here the "U" specifies a unique match, without which a large number of redundant matches would be returned. The forth query specifies all molecules containing at least one example of a hydrogen-bond acceptor connected to a hydrogen-bond donor. The "S" ensures that at most a single match will be identified in each molecule. The fifth query specifies all molecules containing the particular pharmacophore.

Appendix D: Chemical Computing Group (CCG) QSAR descriptors

The CGC QSAR descriptors are partitioned into *classes*:

- **2D**. 2D descriptors only use the atoms and connection information of the molecule for the calculation. 3D coordinates and individual conformations are not considered.
- **i3D**. Internal 3D descriptors use 3D coordinate information about each molecule; however, they are invariant to rotations and translations of the conformation.
- **x3D**. External 3D descriptors also use 3D coordinate information but also require an absolute frame of reference (e.g., molecules docked into the same receptor).

Details on the CGC features are found at: http://www.chemcomp.com/.

Appendix E: Matlab code to filter MARTI data

```
function X=nreged_recover(Xold, data_dir, data_name)

cidx=load([data_dir '/' upper(data_name) '/' data_name '_feat.calib'
]);
cal0=load([data_dir '/' upper(data_name) '/' data_name '_feat.calval'
]);

view=0;
[p, n]=size(Xold);
X=zeros(p,n);
for k=1:p
    X(k,:)=nreged_filter(Xold(k,:), view, cidx, cal0);
end

function XF=nreged_filter(X, view, cidx, cal0)
%XF=nreged_filter(X)
% Filters a 2d pattern

if nargin<2, view=0; end
if nargin<3, cidx=[]; cal0=1; end

n=length(X);
t=sqrt(n);
val=sort(X);
% Compute background
background=median(val(1:50));
% Reshape as square
XP=reshape(X, t, t);
if view, cmat_display(XP); end

% add border
XB=zeros(size(XP)+6);
XB([1:t]+3, [1:t]+3)=XP;
XB([t-2:t]+6,[1:3])=(XP(t-1,1)+XP(t-2,2))/2 *ones(3);
XB([1:3],[1:3])=(XP(1,1)+XP(2,2))/2 *ones(3);
XB([1:3], [t-2:t]+6)=(XP(1, t-1)+XP(2, t-2))/2 *ones(3);
XB([t-2:t]+6,[t-2:t]+6)=(XP(t-1,t-1)+XP(t-2,t-2))/2 *ones(3);
```

```
XB([1:t]+3, 1:3)=(XP(:,[1 1 1])+XP(:,[2 2 2]))/2;
XB([1:t]+3, [t-2:t]+6)=(XP(:,[t-1, t-1, t-1])+XP(:,[t t t]))/2;
XB(1:3, [1:t]+3)=(XP([1 1 1],:)+XP([2 2 2],:))/2;
XB([t-2:t]+6, [1:t]+3)=(XP([t-1, t-1, t-1],:)+XP([t t t],:))/2;
if view, cmat_display(XB); end

% Remove the outliers
XBN=XB;
st=std(X);
tt=size(XB,1);
for i=1:tt
    for j=1:tt
        I=[i-1 i i+1];
        gidx=find(I>0 & I<tt);
        I=I(gidx);
        J=[j-1 j j+1];
        gidx=find(J>0 & J<tt);
        J=J(gidx);
        m=-XB(i,j);
        for ii=I
            for jj=J
               m=m+XB(ii,jj);
            end
        end
        m=m/9;
        if XB(i,j)>m+st | XB(i,j)<m-st
            XBN(i,j)=m;
        end
    end
end
if view, cmat_display(XBN); end

XB=XBN;

ker=[1 4 6 4 1]'*[1 4 6 4 1];
ker=ker./sum(sum(ker));
XBS=conv2(XB, ker, 'same');
XPS=XBS(4:t+3,4:t+3);
if view, cmat_display(XPS); end
if isempty(cidx)
    XC=XP-XPS+background;
else
    XC=XP-XPS;
end
if view, cmat_display(XC); end

XF=XC(:)';
if ~isempty(cidx)
    cal=mean(XF(cidx));
    XF=XF-cal+cal0;
end
```