BNfinder users manual

Release 2.1

Norbert Dojer, Paweł Bednarz, Agnieszka Podsiadło and Bartek Wilczyński

September 25, 2015

Contents

1 Users manual

1.1 Installation

The BNFinder software uses setuptools, which is the standard library for packaging python software. The package itselff is available through the Python Package Index (http://pypi.python.org/pypi/BNfinder/) while more frequent releases as well as the source code are available from the project website in launchpad (http://launchpad.net/bnfinder). After downloading the archive containing the current version of BNF, you should extract it to a directory of choice. In unix-like systems you can do it by typing

```
tar -xzf bnfinder-xxx-xxx-.tgz
```

If you want to install the latest development version from launchpad, you can do it directly with bzr version management tool:

```
bzr branch lp:bnfinder
```

Once you have the sources extracted, the installation is performed by a single command

```
python setup.py install
```

in the source directory (**it may require the administrator privileges**).

This installs the BNfinder library to an apropriate location for your python interpreter, and the bnf, the bnf-cv and the bnc scripts which may be accessed from a command line.

In case you don't have administrator privileges, you can either use bnfinder without installation (by invoking commands within the downloaded directory) or by installing within user home directory:

```
python setup.py install --user
```

System requirements

BNFinder is written in pure python, so the only true requirement is the python2 interpreter (any reasonably recent version like 2.6.x or newer should do). Please note that BNFinder is not currently compatible with python3. If you are interested in porting BNFinder to python3 please let us know.

There are several packages one might need to install to have BNFinder working properly, most notably the packages scipy and fpconst are required for proper functioning of BNFinder

If you would like to generate ROC plots for classification, you will also need a working installation of the R language, the Rpy2 python package and ROCR package withing R.

1.2 Usage

bnf

The bnf script is the main part of BNfinder command line tools. Is is used for learning the bayesian network from data and can be executed by typing

```
bnf <options>
```

The following options are available:

-h, --help
: print out a brief summary of options and exit

-e, --expr <file>
load learning data from <file> (this option is obligatory)

-s, --score <name>
learn with <name> scoring criterion; possible names are BDE (default), BIC, MDL and MIT

-l, --limit <number>
limit the search space to networks with at most <number> parents for each vertex

-i, --suboptimal <number>
return at most <number> best scored parents sets for each vertex (default 1; negative values result in no limit)

-m, --min-empty <value>
for each vertex return only parents sets with relative probabilities greater than <value> (default 0)

-o, --min-optimal <value>
for each vertex return only parents sets with the ratio of their posterior probability to the optimal set's one greater than <value> (default 0)

-r, --fpr <value>
adjust g components of the scoring function to yield the expected proportion of false positives (i.e. FP edges unless -u is specified) equal to <value>; the procedure is switched off by default

-u, --fpr-nodes
interpret the parameter of -r as the expected proportion of regulons having false positive regulators

-x, --max-permutations <number>
do not perform more than <number> permutations in the predetermination of type I error rate

-d, --data-factor <number>
multiply (each item of) the dataset <number> times (default 1); this option may be used to change the proportion between d and g components of the scoring function (see the definition of the *splitting* assumption); it has no effect when the option -r is also used

-v, --verbose
print out communicates on standard output

-p, --prior-pseudocount <number>
set the pseudocounts of data items with specified values of a vertex and its parents set to $\frac{\texttt{<number>}}{|\mathcal{V}|^{|\mathbf{Pa}|+1}}$ (resulting in the total pseudocount equal to <number>) – this method follows Heckerman et al [3]; when the option is unspecified, all pseudocounts are set to 1, following Cooper and Herskovitz [1]; pseudocounts are used as hyperparameters of the Dirichlet priors of the BDE scoring criterion and also in the estimation of the conditional probability distributions (CPDs) of learned network

-n, --net <file>
write the learned network graph to <file> in the SIF format

-t, --txt <file>
write the learned suboptimal parents sets to <file>

-b, --bif <file>
write the learned optimal Bayesian network to <file> in the BIF format

-c, --cpd <file>
write the learned optimal Bayesian network to <file> as a Python dictionary

-f, --fraction <value>
set the minimal weight of a parent to be considered in a parent set

-a, --chi <value>
set the alpha value for the chi-square distribution used in MIT score (default=.9999)

-g, --sloops
allow self-loops in Dynamic Bayesian Networks (by default self-loops are disabled)

-k, --cpu <number>
use number of processes for multiprocessing - by default 0 is used (no multiprocessing)

bnf-cv

The bnf-cv script is a utility to perform cross-validation tests. The script can be executed by typing:

```
bnf-cv <options>
```

The following options are available:

-h, --help
print out a brief summary of options and exit

-e, --expr <file>
load learning data from `<file>` (this option is obligatory)

-k, --cross-val-folds <number>
set the number of cross-validation folds to `k`; defaults to 10; if equal to 1, do not perform cross-validation at all

-s, --score <name>
learn with `<name>` scoring criterion; possible names are `BDE` (default), `BIC`, `MDL` and `MIT`

-l, --limit <number>
limit the search space to networks with at most `<number>` parents for each vertex

-i, --suboptimal <number>
return at most `<number>` best scored parents sets for each vertex (default 1; negative values result in no limit)

-m, --min-empty <value>
for each vertex return only parents sets with relative probabilities greater than `<value>` (default 0)

-o, --min-optimal <value>
for each vertex return only parents sets with the ratio of their posterior probability to the optimal set's one greater than `<value>` (default 0)

-r, --roc <file>
write ROC curve to the given `file`; it will contain a curve for each cross-validation fold and one additional curve averaging remaining ones

-d, --data-factor <number>
multiply (each item of) the dataset `<number>` times (default 1); this option may be used to change the proportion between d and g components of the scoring function (see the definition of the *splitting* assumption);

-v, --verbose
print out communicates on standard output

-p, --prior-pseudocount <number>
set the pseudocounts of data items with specified values of a vertex and its parents set to $\frac{\texttt{<number>}}{|\mathcal{V}|^{|\mathbf{Pa}|+1}}$ (resulting in the total pseudocount equal to `<number>`) – this method follows Heckerman et al [3]; when the option

is unspecified, all pseudocounts are set to 1, following Cooper and Herskovitz [1]; pseudocounts are used as hyperparameters of the Dirichlet priors of the BDE scoring criterion and also in the estimation of the conditional probability distributions (CPDs) of learned network

-n, --net <file>
write the learned network graph in the SIF format to `<file>0`, `<file>1`, ..., `<file>(k-1)` for each cross-validation fold respectively; `k` is the number of crossvalidation folds

-t, --txt <file>
write the learned suboptimal parents sets to to `<file>0`, `<file>1`, ..., `<file>(k-1)` for each cross-validation fold respectively; `k` is the number of crossvalidation folds

-b, --bif <file>
write the learned optimal Bayesian network in the BIF format to `<file>0`, `<file>1`, ..., `<file>(k-1)` for each cross-validation fold respectively; `k` is the number of crossvalidation folds

-c, --cpd <file>
write the learned optimal Bayesian network as a Python dictionary to `<file>0`, `<file>1`, ..., `<file>(k-1)` for each cross-validation fold respectively; `k` is the number of crossvalidation folds

-f, --fraction <value>
set the minimal weight of a parent to be considered in a parent set

-a, --chi <value>
set the alpha value for the chi-square distribution used in MIT score (default=.9999)

-x, --xaxis <name>
specify x axis of the roc curve (`tpr` is default); see Rocr manual for posiible options

-y, --yaxis
specify x axis of the roc curve (`fpr` is default); see Rocr manual for posiible options

-A, --diag
set to 0 to remove diagonal line from roc plots

bnc

The bnc script allows you to perform a classification task, using a network encoded in a file in cpd format. The script can be executed by typing:

```
bnc <options>
```

The following options are available:

-h, --help
print out a brief summary of options and exit

-c, --cpd <file>
load `<file>` with the conditional probability distribution gotten from BNfinder (this option is obligatory)

-d, --data <file>
load `<file>` with values of all regulator variables (this option is obligatory)

-m, --ml
output only the most likely value; switched off for default

-p, --prob <number>
output the probability of given class `number` (for all nodes)

-o, --out <file>
write the result of the classification to `file` (by default writes to `stdout`)

When neither `-m` nor `-p` option is given, writes out for every regulatee and experiment the python dictionary with probabilities for all classes.

1.3 Input format for structure learning

The learning data must be passed to BNfinder in a text file splitted into 3 parts: preamble, experiment specification and experiment data. The preamble allows user to specify some features of data and/or network, while the next two parts contain the learning data, essentially formatted as a table with space- or tab-separated values.

Preamble

The preamble allows specifying experiment peturbations, structural constraints, vertex value types, vertex CPD types and edge weights. Each line in the preamble has the following form:

```
#<command> <arguments>
```

Experiments with perturbed values of some vertices carry no information regarding their regulatory mechanism. Thus including these experiments data in learning parents of their perturbed vertices biases the result (see [2] for a detailed treatment). The following command handles perturbations:

#perturbed <experiment/serie> <vertex list>
omit data from experiment (serie of experiments in the case of dynamic networks) `<experiment/serie>` when learning parents of vertices from `<vertex list>`

One possible way of specifying structural constraints with BNfinder is to list potential parents of particular vertices. An easier method is available for constraints of the cascade form, where the vertex set is splitted into a sequence of groups and each parent of a vertex must belong to one of previous groups (a simple but extremely useful example is a cascade with 2 groups: regulators and regulatees). There are 2 commands specifying structural constraints:

#parents <vertex> <vertex list>
restrict the set of potential parents of `<vertex>` to `<vertex list>`.

#regulators <vertex list>
restrict the set of potential parents of all vertices except specified with `#parents` command or with previous or present `#regulators` command to vertices included in `<vertex list>` of previous or present `#regulators` command.

Note that structural constraints forcing network's acyclicity are necessery for learning a static Bayesian network with BNfinder.

Vertex value types may be specified with the following commands:

#discrete <vertex> <value list>
let `<value list>` be possible values of `<vertex>`

#continuous <vertex list>
let float numbers be possible values of all vertices in `<vertex list>`

#default <value list>
let <value list> be possible values of all vertices except specified with #discrete or #continuous command (when <value list> is FLOAT, float numbers are possible values)

Values in <value list> may be integers or words (strings without whitespaces). When some vertices are left unspecified, BNfinder tries to recognize their possible value sets. However it may miss, in particular when some float numbers are written in integer format or when some possible values are not represented in the dataset (note that the size of the set of possible values affects the score).

The space of possible CPDs of some vertices given their parents may be restricted to *noisy-and* or *noisy-or* distributions. In this case, the sets of possible values of these vertices and their potential parents must be either $\{0, 1\}$ or float numbers. Moreover, BNfinder should be executed with the MDL scoring criterion. The following commands specify vertices with *noisy* CPDs:

#and <vertex list>
restrict the space of possible CPDs of vertices from <vertex list> to *noisy-and* distributions

#or <vertex list>
restrict the space of possible CPDs of vertices from <vertex list> to *noisy-or* distributions

The following commands set prior weights on network edges:

#prioredge <vertex> <weight> <vertex list>
set the prior weights of all edges originating from vertices from <vertex list> and aiming at <vertex> to <weight>

#priorvert <weight> <vertex list>
set the prior weights of all edges originating from vertices from <vertex list> (except specified in <prioredge> command) to <weight>

Weights must be positive float numbers. Edges with greater weights are penalized harder. The default weight is 1.

Experiment specification

The experiment specification has the following form:

```
<name> <experiment list>
```

where <name> is a word starting with a symbol other then #. It is used in some output formats as the name of the learned network (until <name> = conditions, in such case the input file name is used). The form of experiment names depends on the data type and, consequently, on the type of learned network:

- When the dataset consists of results of independent experiments and a static Bayesian network is to be learned, experiment names are words without the symbol ':'.
- When the dataset consists of results of time series experiments and a dynamic Bayesian network is to be learned, experiment names have the form <serie>:<condition>. Each serie must be ordered according to the condition times and cannot be interrupted by experiments from other series.

Experiment data

Each line of the experiment data part has the following form:

```
<vertex> <value list>
```

where <vertex> is a word and values are listed in the order corresponding to <experiment list>.

1.4 Input format for classification task

Format for classification task's input consists of two parts:

Experiment specification

This part follows exactly the same rules as when specifying the input for learning the bayesian network structure.

Experiment data

Each line of the experiment data part has the following form:

```
<vertex> <value list>
```

where `<vertex>` is a word and values are listed in the order corresponding to `<experiment list>`. In this part we specify only values for regulators. The same names for vertices as while learning the network structure should be given here.

1.5 Output formats

Suboptimal parents sets

Suboptimal parents sets are written to a file in a simple text format splitted into sections representing the sets of the parents of each vertex. Each section contains a leading line with the vertex name followed by lines representing its consecutive suboptimal parents sets. Each of these lines has the form:

```
<relative probability> <vertex list>
```

were `<relative probability>` is the ratio of the set's posterior probability to the posterior probability of the empty parents set and `<vertex list>` contains the elements of the set. Lines are ordered decreasingly according to `<relative probability>`.

To show it by example, the section:

```
C
 2.333333  B
 1.000000
 0.592593  B A
```

reports 3 most probable parents sets of the vertex C: $\{B\}, \emptyset, \{B, A\}$. Moreover, it states that $\{B\}$ is 2.333333 times more probable than the empty set and the corresponding ratio for $\{B, A\}$ equals 0.592593.

SIF format

The SIF (Simple Interaction File), usually contained in files with `.sif` extension is the simplest of the supported network formats and carries only information on its topology. In this format, each line represents the fact of a single interaction. In our case such interaction represents the fact that one variable depends on some other variable. Each line contains three values:

- Parent variable identifier,
- Interaction label; its type depends on the argument of the `-i` option (specifying the number of reported suboptimal parents sets)

  - if =1 (default), `+/-` is reported when positive or negative correlation between variables is found
  - otherwise, the edge label represents the posterior probability of the interaction represented by the edge (under the assumption that the true parents set is among the suboptimal ones)

- Child variable identifier.

To show it by example, the file:

```
A + B
B - C
```

describes a network of the following shape:

$$A \xrightarrow{+} B \xrightarrow{-} C.$$

Running BNFinder with the same input file and $\neq 1$ argument of `-i` option could result in the file of the form:

```
B 0.2254 A
C 0.1560 A
A 0.8563 B
B 0.0358 B
A 0.0247 C
B 0.9463 C
```

showing that the posterior probabilities of the interactions $A \to B$ and $B \to C$ are significantly higher than the other ones.

The main advantage of this format is that it can be read by the Cytoscape (`http://cytoscape.org`) software allowing for quick visualization. It is also trivial to use such data in one's own software.

BIF format

Bayesian Interchange Format (BIF) is a simple text format dedicated to Bayesian networks. It is supported in some BN applications (e.g. JavaBayes, Bayes Networks Editor) and may be easily converted with available tools to other popular formats (including XML formats and BNT format of K. Murphy's Bayes Net Toolbox). BNfinder writes learned networks in BIF version 0.15.

Python dictionary

A network saved in `<file>` as a dictionary may be loaded to your Python environment by

```
eval(open(<file>).read())
```

The dictionary consists of items corresponding to all network's vertices. Each item has the following form:

```
<vertex name> : <vertex dictionary>
```

Vertex dictionaries have the following items:

'vals' : <value list>

'pars' : <parent list>

'type' : <CPD type>
(only for vertices with *noisy* CPDs, possible values of `<CPD type>` are `'and'` and `'or'`)

'cpds' : <CPD dictionary>

The form of the vertex CPD dictionary depends on the vertex type. In the case of noisy CPD, the dictionary items have the following form:

<vertex name> : <probability>
which means (in the case of noisy-and/-or distribution) that the considered vertex is assigned value 1/0 with <probability> given all its parents but <vertex name> equal 1/0

In the case of general CPD, the dictionary has items of the following form:

<value vector> : <distribution dictionary>
where <value vector> is a tuple of parents' values and the distribution of the considered vertex given <parent list> = <value vector> is defined in <distribution dictionary> in the following way:

<value> : <probability>
means that the vertex is assigned <value> with <probability>

None : <probability>
means that the vertex is assigned with <probability> each of its possible values unspecified in a separate item

None : <probability>
means that given <parent list> equal to a value vector unspecified in a separate item the vertex is assigned each of its possible values with <probability>

Standard output

When BNfinder is executed from a command line with the option -v, it prints out communicates related to its current action: loading data, learning regulators of consecutive vertices and writing output files. Moreover, after finishing computations for a vertex its predicted best parents sets and their scores are reported and after finishing computations for all vertices BNfinder reports the score and structure of the optimal network.

Classification results

The result of bnc command is written to a file with the following format. First line contains only names of experiments:

```
classes <name list>
```

Next lines contain information depending on the options chosen. When bnc is executed with -m option, lines with most likely values for regulatees are printed to a file. All of them have the following form:

```
<vertex name> <ML value list>
```

When bnc is executed with -p option, for every regulatee one line with probability of being in the specified class is written to the output:

```
<vertex name> <probability list>
```

References

[1] Gregory F. Cooper and Edward Herskovits. A Bayesian method for the induction of probabilistic networks from data. *Machine Learning*, 9:309–347, 1992.

[2] Norbert Dojer, Anna Gambin, Bartek Wilczyński, and Jerzy Tiuryn. Applying dynamic Bayesian networks to perturbed gene expression data. *BMC Bioinformatics*, 7:249, 2006.

[3] David Heckerman, Dan Geiger, and David Maxwell Chickering. Learning Bayesian networks: The combination of knowledge and statistical data. *Machine Learning*, 20(3):197–243, Sep. 1995.