Getting Started with Catnet Package

Nikolay Balov

March 5, 2010

Catnet package provides tools for learning Bayesian networks from data with focus on model selection ([2]). A Bayesian network is defined by a graphical structure in form of directed acyclic graph and a probability model given as a set of conditional distributions, one for each node in the network. The package deals with networks which nodes are categorical random variables and, consequently, discrete probability models. In contrast to other learning algorithms, *catnet*'s search functions do not output one learned network but a set of networks with increasing complexity that fit the data according to the MLE criterion. These optimal networks explain and represent the relations between the node-variables. The final network selection is left to the user.

To demonstrate some of the features of *catnet* package we provide a network learning example applied on a well known in the literature network, the ALARM network. ALARM is a medical diagnostic alarm message system for patients monitoring developed by Beinlich, et.all., [1]. The belief propagation network behind ALARM has 37 nodes and 46 directed edges. It is potted in Figure 1. The node variables are described in the Appendix. We have available a sample of size 2000 and our goal is to reconstruct the network from that sample.

In order to obtain reproducible results we set a stored in advance seed value. Even with that measure however, the user may get different results. Processing time depends on the hardware and the values shown below (in seconds) are those obtained during the building of the package.

```
> library(catnet)
> data(alarmseed)
> saved.rng <- RNGkind()
> RNGkind(kind = "Mersenne-Twister")
```

We start with loading the data and the ground-truth network. Calling a `catnet` object by name, as below, shows a short object description that includes number of nodes, maximum number of parents per node, maximum number of categories per node, likelihood (if available) and network complexity.

```
> data(alarm)
> data(alarmnet)
> alarmnet

A catNetwork object with  37  nodes,  4  parents,  4  categories,
 Likelihood =  0 , Complexity =  509 .
```

We can also calculate the likelihood of the data with respect to the ground-truth network.

```
> cnLoglik(alarmnet, alarm)

[1] -21387.81
```

Since *catnet* implements a Maximum Likelihood estimation, the 'true' likelihood, which is about -21387, is the value we want to get near to later in the learning process.

Now we proceed with the network learning, first assuming the node order is known and then without using any prior knowledge of their order. In the latter case we use stochastic search in the space of orders. Note that this space includes all possible permutations of the 37 nodes and it is huge.

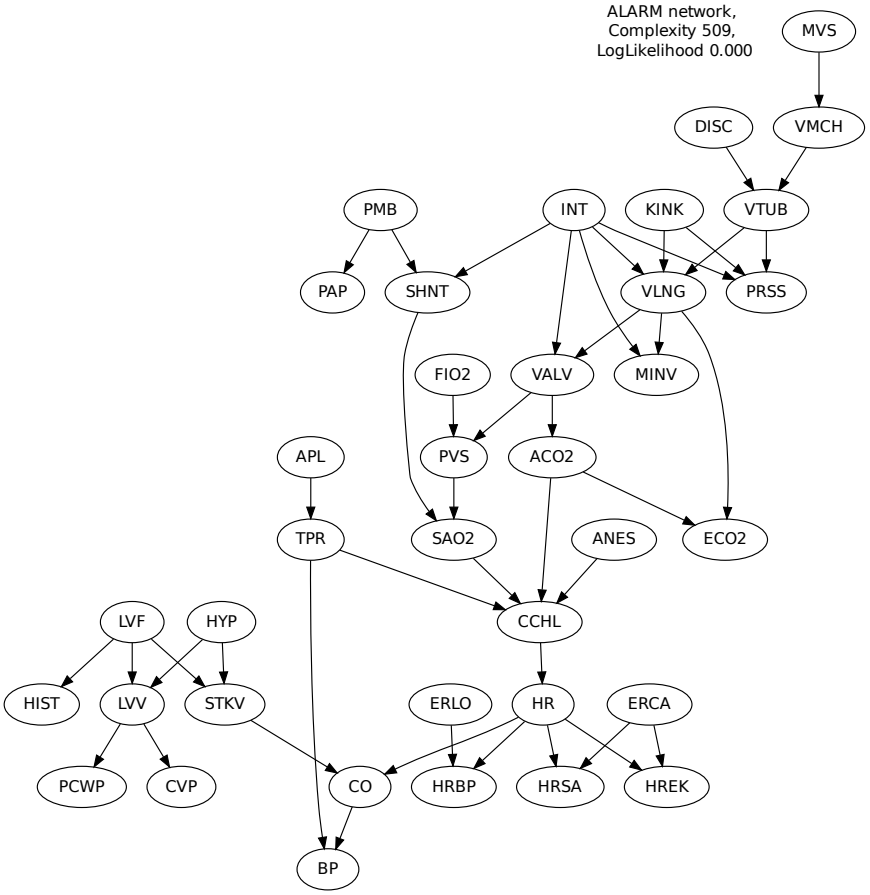

Figure 1: The original ALARM network.

1 Reconstruction using the true node order

We use `cnEvaluate` function that performs search for the `alarmnet`'s node order, the true order, finds a list of networks with increasing complexity, from the minimal possible (37) to the complexity of original network (509), and finally, compares each of these networks to `alarmnet`. To greatly speed-up the processing `cnEvaluate` function is called with parameter `maxParentSet` set to 3, although the original network has one node with four parents.

A diagnostic plot shows the likelihood versus complexity, the achieved true positive directed and undirected edges, as well as the Hamming distance, which is the sum of the false positive and false negative directed edges.

```
> eval <- cnEvaluate(object = alarmnet, data = alarm, perturbations = NULL,
+     maxParentSet = 3)
> eval

 Number of nodes    = 37,
 Sample size        = 2000,
 Number of networks = 442
 Processing time    = 6.187

> cnPlot(eval)
```

The reconstructed network (`bstnet`) with complexity 509 almost perfectly matches the original ALARM network with only two edges missed (FN=2).

```
> bstnet <- cnFind(eval, cnComplexity(alarmnet))
> bstnet

A catNetwork object with  37  nodes,  3  parents,  4  categories,
 Likelihood =  -21220.25 , Complexity =  509 .

> cnCompare(alarmnet, bstnet)

 Edges:
                  TP = 44,
                  FP = 16,
                  FN = 2,

 Hamming:
             (FP+FN) = 18,
                 exp = 175,

 Skeleton:
                  TP = 44,
                  FP = 16,
                  FN = 2,

 Order:
                  FP = 0,
                  FN = 0,

 Markov blanket:
                  FP = 10,
                  FN = 5
```

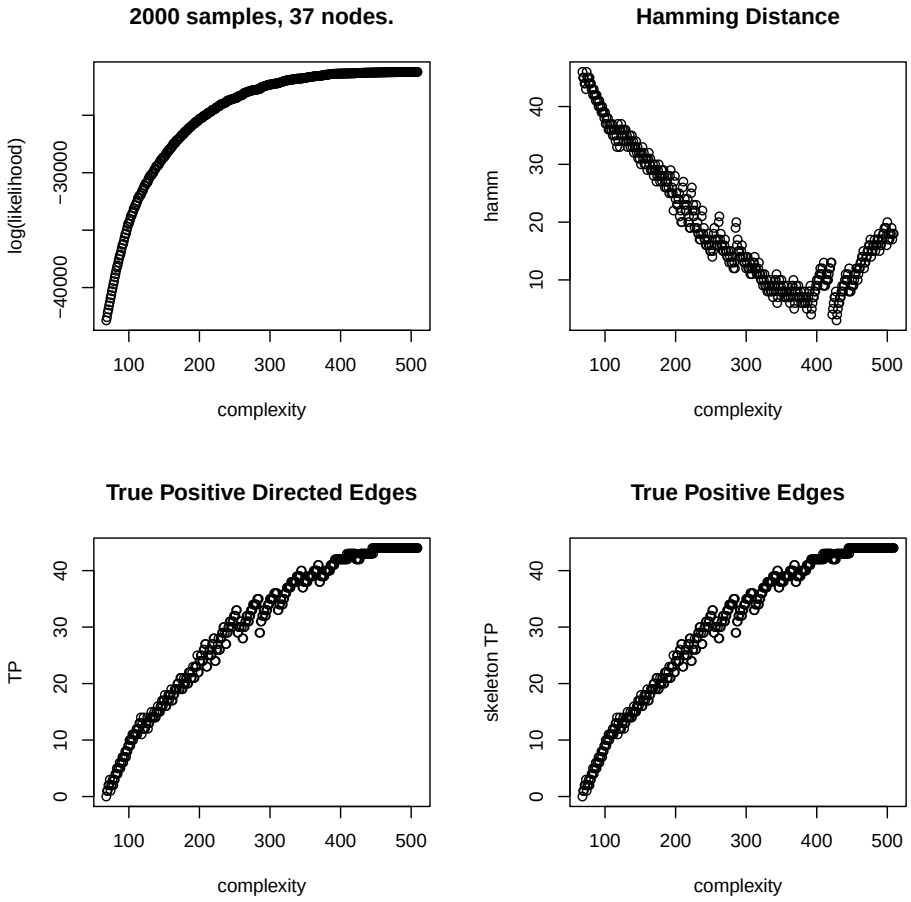

Figure 2: Evaluation results. Estimated networks according to the true node order have complexities from 37 to 509. Reported are their likelihood, the number of true positive directed and undirected edges, and the hamming distance in comparison to the ground-truth network.

Note also that the likelihood with respect to `bstnet`, about -21220, is actually larger than the likelihood with respect to the ground-truth network, which was about -21387.

2 Learning without assumptions on the node order

In more realistic setting, we do not have any prior knowledge of how the nodes are ordered in the network explaining the data. Given the sheer vastness of the space of orders, exhaustive search is prohibitive and only stochastic approach seems possible. *Catnet* provides `cnSearchSA` and `cnSearchSAcluster` functions that implement Metropolis algorithm with Simulated Annealing (SA). The user may consult with the manual pages for more information on the functions' parameters. By varying some key parameters such as `orderShuffles` and `stopDiff`, and constructing a chain of searches such that each new search starts where the previous has ended, a great variety of different searching scenaria can be realized.

To speed-up the processing, `cnSearchSA` is called with parameter `maxParentSet` set to 2, for allowing 3 or 4 parents per node will make the computation many times slower. This, of course, limits the learning performance, but it is overall a good compromise.

For the purpose of this demonstration we will perform a two-stage search in the space of orders. First, we begin with a greedy search by processing 100 randomly selected orders and picking up the one with the best BIC score. Of course, 100 is a rather small number and in practice much larger one is recommended. To force the function `cnSearchSA` to use only random orders we set `orderShuffles` parameter to 0. This is equivalent to defining the neighborhood of each order to be the whole space of orders. This first stage of the search plays the role of warming-up for the next 'true' SA search.

```
> set.seed(alarmseed)
> sanets <- cnSearchSA(data = alarm, perturbations = NULL, maxParentSet = 2,
+     maxComplexity = 600, parentsPool = NULL, fixedParentsPool = NULL,
+     selectMode = "BIC", tempStart = 10, tempCoolFact = 1, tempCheckOrders = 100,
+     maxIter = 100, orderShuffles = 0, stopDiff = 1e-10, priorSearch = NULL)
> sanets

 Number of nodes    = 37,
 Sample size        = 2000,
 Number of networks = 533
 Processing time    = 241.470

> sanet1 <- cnFindBIC(sanets, nrow(alarm))
> cnCompare(alarmnet, sanet1)

 Edges:
                  TP = 29,
                  FP = 15,
                  FN = 17,

 Hamming:
             (FP+FN) = 32,
                 exp = 193,

 Skeleton:
                  TP = 37,
                  FP = 7,
                  FN = 9,

 Order:
                  FP = 9,
                  FN = 9,

 Markov blanket:
                  FP = 8,
                  FN = 17

> sanet1@meta <- "sanet1, SA learning, stage I"
> sanet1

A catNetwork object with  37  nodes,  2  parents,  4  categories,
 Likelihood =  -21589.15 , Complexity =  427 .
```

As it is seen from the results, this first stage is rather successful, with `sanet1` recovering 29 true positive directed edges and 37 undirected (from the skeleton). In the second step, we perform another SA search but this time working with smaller neighborhoods (`orderShuffles` is set to 4). We also start from the order of the best previously found network, `sanet1`, thus hoping to improve its performance.

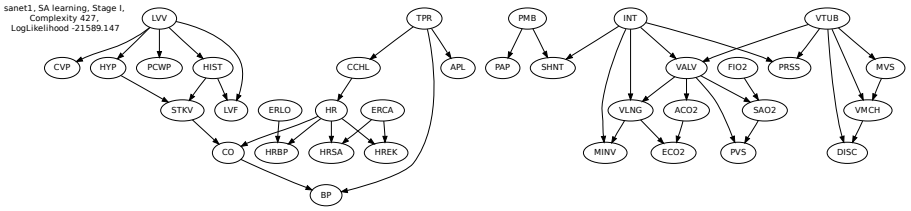

Figure 3: The network with the best BIC score learned in the first stage of SA.

```
> set.seed(alarmseed)
> sanets <- cnSearchSA(data = alarm, perturbations = NULL, maxParentSet = 2,
+     maxComplexity = 600, parentsPool = NULL, fixedParentsPool = NULL,
+     selectMode = "BIC", tempStart = 10, tempCoolFact = 0.95,
+     tempCheckOrders = 16, maxIter = 256, orderShuffles = 4, stopDiff = 1e-10,
+     priorSearch = sanets)
> sanets

 Number of nodes    = 37,
 Sample size        = 2000,
 Number of networks = 533
 Processing time    = 342.440

> sanet2 <- cnFindBIC(sanets, nrow(alarm))
> cnCompare(alarmnet, sanet2)

 Edges:
                 TP = 30,
                 FP = 16,
                 FN = 16,

 Hamming:
             (FP+FN) = 32,
                 exp = 194,

 Skeleton:
                 TP = 39,
                 FP = 7,
                 FN = 7,

 Order:
                 FP = 10,
                 FN = 10,

 Markov blanket:
                 FP = 8,
                 FN = 15

> sanet2@meta <- "sanet2, SA learning, stage II"
> sanet2
```

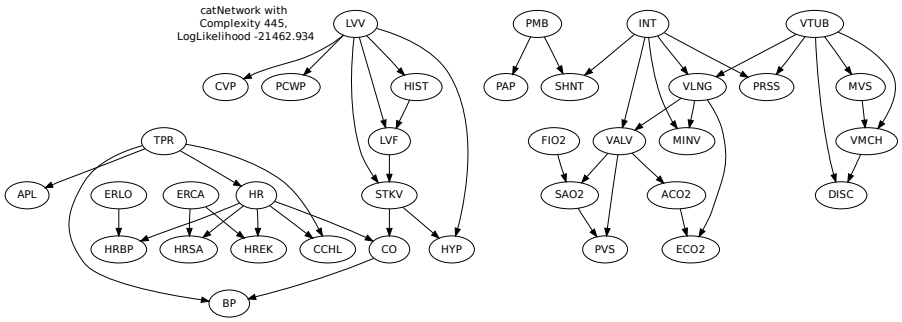

Figure 4: The network with the best BIC score learned in the second stage of SA.

```
A catNetwork object with  37  nodes,  2  parents,  4  categories,
 Likelihood =  -21462.93 , Complexity =  445 .
```

And indeed, this second search step brings some performance improvement- in comparison to `alarmnet`, `sanet2` has 30 and 39 true positive directed and undirected edges, respectfully. The best found network, according to BIC, is depicted in Figure 4.

Finally, we demonstrate the parallel search abilities of *catnet*. By calling `cnSearchSAcluster` function with parameter `clustrNodes` set to 4, we create 4 parallel processing units each performing a SA search. At the end, the best out of the 4 sets of networks is selected according to its BIC score. The final network (again selected according BIC) is depicted in Figure 5. This parallel processing approach is suitable on machines with quad-core processors.

```
> set.seed(alarmseed)
> sacnets <- cnSearchSAcluster(data = alarm, perturbations = NULL,
+     maxParentSet = 2, maxComplexity = 600, parentsPool = NULL,
+     fixedParentsPool = NULL, tempStart = 100, tempCoolFact = 0.9,
+     tempCheckOrders = 16, maxIter = 256, orderShuffles = -4,
+     stopDiff = 1e-08, priorSearch = NULL, clusterNodes = 4)
> sacnets

 Number of nodes    = 37,
 Sample size        = 2000,
 Number of networks = 532
 Processing time    = 197.936

> sanet3 <- cnFindBIC(sacnets, nrow(alarm))
> cnCompare(alarmnet, sanet3)

 Edges:
                  TP = 26,
                  FP = 21,
                  FN = 20,

 Hamming:
             (FP+FN) = 41,
                 exp = 248,
```

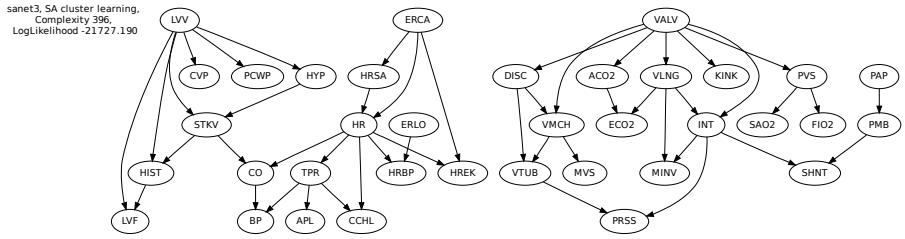

Figure 5: The best network according to BIC learned by SA in cluster.

```
 Skeleton:
                  TP = 38,
                  FP = 9,
                  FN = 8,

 Order:
                  FP = 23,
                  FN = 23,

 Markov blanket:
                  FP = 7,
                  FN = 17
```

```
> sanet3@meta <- "sanet3, SA cluster learning"
> sanet3

A catNetwork object with  37  nodes,  2  parents,  4  categories,
 Likelihood =  -21727.19 , Complexity =  396 .
```

At this point we end the session.

```
> RNGkind(kind = saved.rng[1])
```

References

[1] Beinlich, I., Suermondth, G., Chavez, R., Cooper, G., The ALARM monitoring system. 1989, In Proc. 2-nd Euro. Conf. on AI and Medicine.

[2] Salzman, P., Almudevar, A., Using Complexity for the Estimation of Bayesian Networks. 2006, Statistical Applications in Genetics and Molecular Biology, Vol. 5, Issue 1.

3 Appendix

The ALARM network's node variables are described in the following list.

1. CVP (central venous pressure): a three-level factor with levels LOW, NORMAL and HIGH.
2. PCWP (pulmonary capillary wedge pressure): a three-level factor with levels LOW, NORMAL and HIGH.

3. HIST (history): a two-level factor with levels TRUE and FALSE.
4. TPR (total peripheral resistance): a three-level factor with levels LOW, NORMAL and HIGH.
5. BP (blood pressure): a three-level factor with levels LOW, NORMAL and HIGH.
6. CO (cardiac output): a three-level factor with levels LOW, NORMAL and HIGH.
7. HRBP (heart rate / blood pressure): a three-level factor with levels LOW, NORMAL and HIGH.
8. HREK (heart rate measured by an EKG monitor): a three-level factor with levels LOW, NORMAL and HIGH.
9. HRSA (heart rate / oxygen saturation): a three-level factor with levels LOW, NORMAL and HIGH.
10. PAP (pulmonary artery pressure): a three-level factor with levels LOW, NORMAL and HIGH.
11. SAO2 (arterial oxygen saturation): a three-level factor with levels LOW, NORMAL and HIGH.
12. FIO2 (fraction of inspired oxygen): a two-level factor with levels LOW and NORMAL.
13. PRSS (breathing pressure): a four-level factor with levels ZERO, LOW, NORMAL and HIGH.
14. ECO2 (expelled CO2): a four-level factor with levels ZERO, LOW, NORMAL and HIGH.
15. MINV (minimum volume): a four-level factor with levels ZERO, LOW, NORMAL and HIGH.
16. MVS (minimum volume set): a three-level factor with levels LOW, NORMAL and HIGH.
17. HYP (hypovolemia): a two-level factor with levels TRUE and FALSE.
18. LVF (left ventricular failure): a two-level factor with levels TRUE and FALSE.
19. APL (anaphylaxis): a two-level factor with levels TRUE and FALSE.
20. ANES (insufficient anesthesia/analgesia): a two-level factor with levels TRUE and FALSE.
21. PMB (pulmonary embolus): a two-level factor with levels TRUE and FALSE.
22. INT (intubation): a three-level factor with levels NORMAL, ESOPHAGEAL and ONE SIDED.
23. KINK (kinked tube): a two-level factor with levels TRUE and FALSE.
24. DISC (disconnection): a two-level factor with levels TRUE and FALSE.
25. LVV (left ventricular end-diastolic volume): a three-level factor with levels LOW, NORMAL and HIGH.
26. STKV (stroke volume): a three-level factor with levels LOW, NORMAL and HIGH.
27. CCHL (catecholamine): a two-level factor with levels NORMAL and HIGH.
28. ERLO (error low output): a two-level factor with levels TRUE and FALSE.
29. HR (heart rate): a three-level factor with levels LOW, NORMAL and HIGH.
30. ERCA (electrocauter): a two-level factor with levels TRUE and FALSE.
31. SHNT (shunt): a two-level factor with levels NORMAL and HIGH.

32. PVS (pulmonary venous oxygen saturation): a three-level factor with levels LOW, NORMAL and HIGH.

33. ACO2 (arterial CO2): a three-level factor with levels LOW, NORMAL and HIGH.

34. VALV (pulmonary alveoli ventilation): a four-level factor with levels ZERO, LOW, NORMAL and HIGH.

35. VLNG (lung ventilation): a four-level factor with levels ZERO, LOW, NORMAL and HIGH.

36. VTUB (ventilation tube): a four-level factor with levels ZERO, LOW, NORMAL and HIGH.

37. VMCH (ventilation machine): a four-level factor with levels ZERO, LOW, NORMAL and HIGH.