Bayesian Networks

K2 algorithm and the `bnstruct` library

Paolo Lapo Cerni
Lorenzo Vigorelli
Arman Singh Bains

29 August 2024

UNIVERSITÀ DEGLI STUDI DI PADOVA

Introduction

Objective 1

Implement the K2 algorithm to learn the topology of a Bayesian network

Objective 2

Test the K2 algorithm to a range of datasets

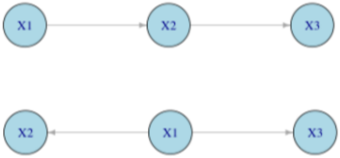
Presentation outline

This presentation will be subdivided in:

- Bayesian network overview
- K2 description and implementation
- K2 benchmarks

Motivation

Given propositions X_1, X_2, X_3, we want to relate them logically:



Bayesian Network

Definition

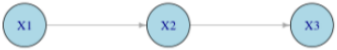
A directed graph is defined as a pair $G = (V, E)$ where V is a set of nodes and E is a set of directed edges, where each directed edge is an ordered pair of distinct vertices $(i, j) \in E$, i.e. a graph where each edge is directed from one vertex to another

A Bayesian network is a probabilistic model interpretable as a directed acyclic graph (DAG):

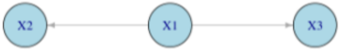
- nodes represent variables
- edges represent conditional dependencies (unconnected nodes thus represent conditionally independent variables)

A simple example

If I were to describe a system were $P(X_1, X_2, X_3) = P(X_3|X_2)P(X_2|X_1)P(X_1)$ the network would be



If I were to describe a system were $P(X_1, X_2, X_3) = P(X_3|X_1)P(X_2|X_1)P(X_1)$ the network would be



In general you would write $P(X_1, ..., X_n) = \prod_{j=1}^{n} P(X_j|\pi(X_j))$

Key assumptions

- If database variables Z are discrete

$$P(B_S, D) = \int_{B_P} P(D|B_S, B_P) f(B_P|B_S) P(B_S) dB_P$$

- Cases occur independently
- No variables have missing values
- The density function is uniform

Scoring function

We want to modify the maximization of $P(B_S, D)$ to use a greedy-search method:

$$f(i, \pi_i) = \prod_{j=1}^{q_i} \frac{(r_i - 1)!}{(N_{ij} + r_i - 1)!} \prod_{k=1}^{r_i} \alpha_{ijk}!$$

Scoring function

$$\text{f(i, } \pi_i) = \prod_{j=1}^{q_i} \frac{(r_i - 1)!}{(N_{ij} + r_i - 1)!} \prod_{k=1}^{r_i} \alpha_{ijk}!$$

This is the **scoring function** used in the K2 algorithm. The intuition is that $f(i, \pi_i)$ is the probability of the dataset given that the parents of x_i are π_i. Let's break it down in its components:

π_i**:** set of parents of node x_i

q_i**:** number of all possible instantiations of the parents of x_i in the data.

Scoring function

$$f(i,\ \pi_i) = \prod_{j=1}^{q_i} \frac{(r_i - 1)!}{(N_{ij} + r_i - 1)!} \prod_{k=1}^{r_i} \alpha_{ijk}!$$

r_i**:** number of all the possible values of the attribute x_i.

```r
r_i <- data |> distinct(data[[x_i]]) |> nrow()
```

Scoring function

$$\text{f(i, } \pi_i) = \prod_{j=1}^{q_i} \frac{(r_i - 1)!}{(N_{ij} + r_i - 1)!} \prod_{k=1}^{r_i} \alpha_{ijk}!$$

α_{ijk}**:** number of cases in the data in which x_i is equal to its k^{th} value, and the parents in π_i are equal to their j^{th} instantiation.

```r
alpha <- data |> group_by(data[c(x_i, parents)]) |> count()
```

Scoring function

$$f(\mathrm{i},\ \pi_i) = \prod_{j=1}^{q_i} \frac{(r_i - 1)!}{(N_{ij} + r_i - 1)!} \prod_{k=1}^{r_i} \alpha_{ijk}!$$

N_{ij}: number of cases in the data in which the parents in π_i are equal to their j^{th} instantiation. Thus, $N_{ij} = \sum_k \alpha_{ijk}$

```
N <- alpha |> group_by(alpha[parents]) |>
  summarise(N = sum(n), .groups = "drop") |> select(N)
```

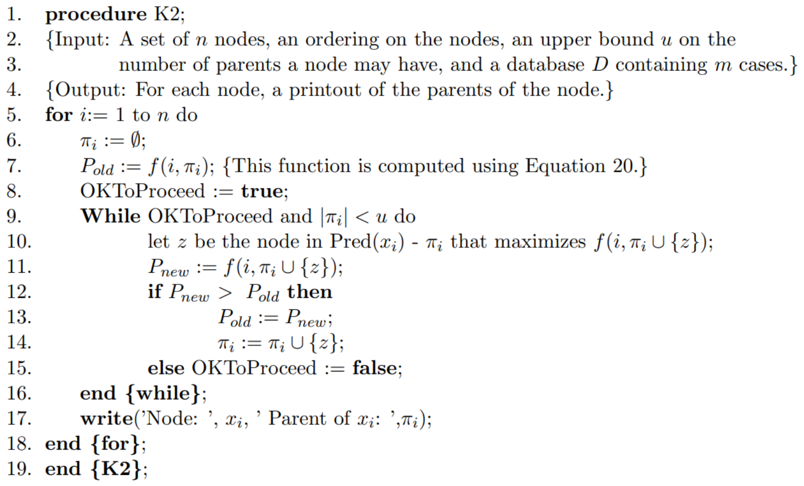
Scoring function

$$\mathrm{f}(\mathrm{i},\ \pi_i) = \prod_{j=1}^{q_i} \frac{(r_i - 1)!}{(N_{ij} + r_i - 1)!} \prod_{k=1}^{r_i} \alpha_{ijk}!$$

In practice, we used the **logarithm of the scoring function**:

- It reduces the computational time
- It prevents round-off errors

K2 algorithm



```
procedure K2;
{Input: A set of $n$ nodes, an ordering on the nodes, an upper bound $u$ on the
        number of parents a node may have, and a database $D$ containing $m$ cases.}
{Output: For each node, a printout of the parents of the node.}
for $i$:= 1 to $n$ do
    $\pi_i := \emptyset$;
    $P_{old} := f(i, \pi_i)$; {This function is computed using Equation 20.}
    OKToProceed := true;
    While OKToProceed and $|\pi_i| < u$ do
        let $z$ be the node in Pred($x_i$) - $\pi_i$ that maximizes $f(i, \pi_i \cup \{z\})$;
        $P_{new} := f(i, \pi_i \cup \{z\})$;
        if $P_{new} > P_{old}$ then
            $P_{old} := P_{new}$;
            $\pi_i := \pi_i \cup \{z\}$;
        else OKToProceed := false;
    end {while};
    write('Node: ', $x_i$, ' Parent of $x_i$: ',$\pi_i$);
end {for};
end {K2};
```

where Pred(x_i) is the set nodes that have already been processed, i.e. the set `0:(i-1)`

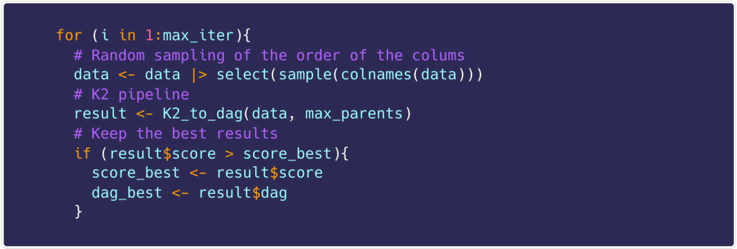
Notes on K2 algorithm

Here's some consideration about the algorithm:

- This is a **greedy algorithm**. Thus, it's not assured to reach the global optimum.
- The pipeline requires an additional parameter, the **maximum number of parents**. This is meant to bound the complexity.
- This algorithm **depends on the order of the columns**.

Notes on K2 algorithm

This algorithm **depends on the order of the columns**: you can optimize the score over different trials.

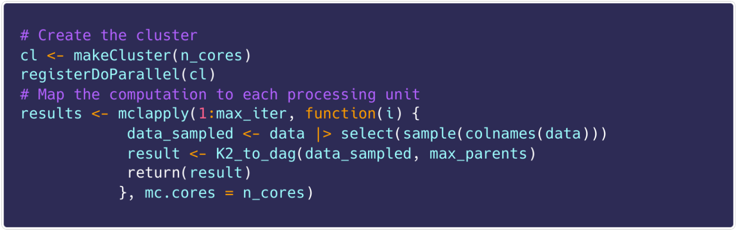


```
for (i in 1:max_iter){
  # Random sampling of the order of the colums
  data <- data |> select(sample(colnames(data)))
  # K2 pipeline
  result <- K2_to_dag(data, max_parents)
  # Keep the best results
  if (result$score > score_best){
    score_best <- result$score
    dag_best <- result$dag
  }
```

Figure: Sequential computation

Notes on K2 algorithm

This algorithm **depends on the order of the columns**: you can optimize the score over different trials.

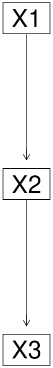


```r
# Create the cluster
cl <- makeCluster(n_cores)
registerDoParallel(cl)
# Map the computation to each processing unit
results <- mclapply(1:max_iter, function(i) {
            data_sampled <- data |> select(sample(colnames(data)))
            result <- K2_to_dag(data_sampled, max_parents)
            return(result)
           }, mc.cores = n_cores)
```

Figure: Parallel computation

Datasets

We used 4 **different datasets** to test and compare the performance.



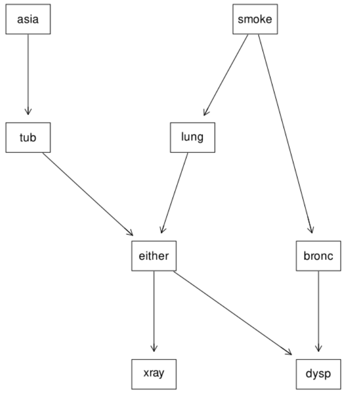
Ruiz Dataset

- Tiny dataset suitable for testing.
- Analytical computations available in the reference.

Ruiz, C. (2005). Illustration of the K2 algorithm for learning Bayes net structures. Department of Computer Science, WPI.

Datasets

We used 4 **different datasets** to test and compare the performance.



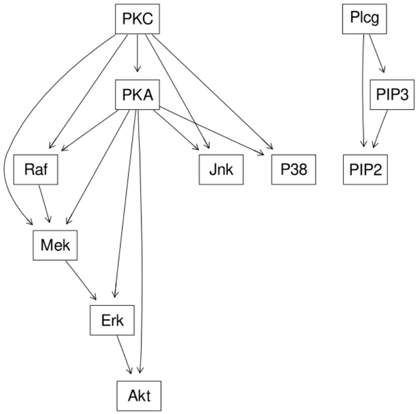
Asia Dataset

- Small network: 8 nodes, 8 edges.
- 10k complete records: no missing data, no latent feature

S. Lauritzen, D. Spiegelhalter. Local Computation with Probabilities on Graphical Structures and their Application to Expert Systems (with discussion). Journal of the Royal Statistical Society: Series B (Statistical Methodology), 50(2):157-224, 1988.

Datasets

We used 4 **different datasets** to test and compare the performance.



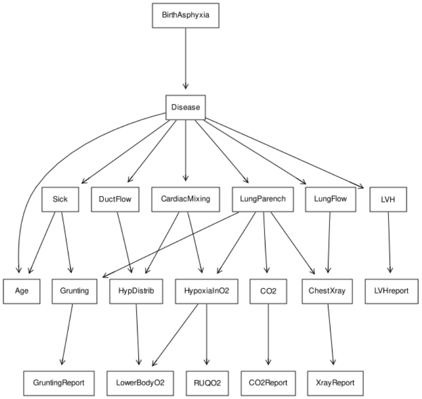
Sachs Dataset

- Biological features of proteins and phospholipids in human cells
- Small network: 11 nodes, 17 edges.
- The ground truth graph has two connected components

K. Sachs, O. Perez, D. Pe'er, D. A. Lauffenburger and G. P. Nolan. Causal Protein-Signaling Networks Derived from Multiparameter Single-Cell Data. Science, 308:523-529, 2005.

Datasets

We used 4 **different datasets** to test and compare the performance.



Child Dataset

- Medical diagnostic dataset used for predicting diseases based on symptoms.
- Medium network: 20 nodes, 25 edges.
- Both raw and imputed data are present.

D. J. Spiegelhalter, R. G. Cowell (1992). Learning in probabilistic expert systems. In Bayesian Statistics 4 (J. M. Bernardo, J. 0. Berger, A. P. Dawid and A. F. M. Smith, eds.) 447-466. Clarendon Press, Oxford.

Convergence analysis: Max Parents

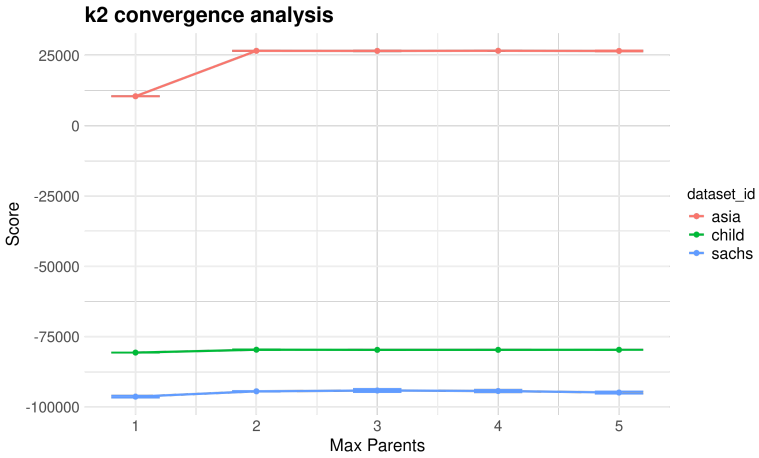


Figure: **Final score** as a function of the max number of parents

Convergence analysis: Cost

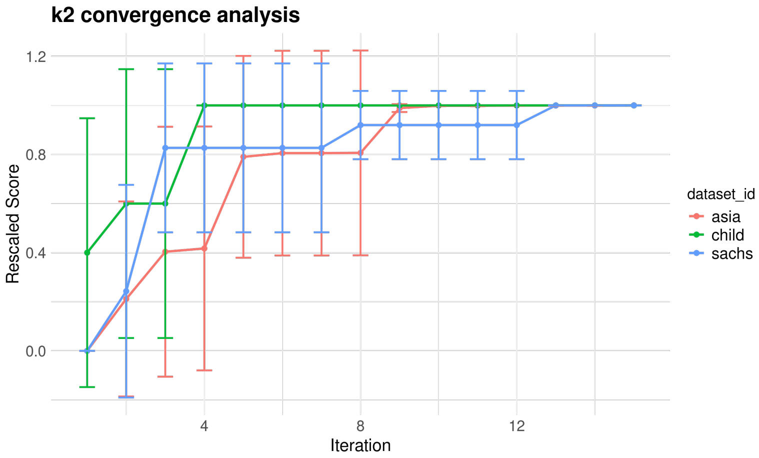
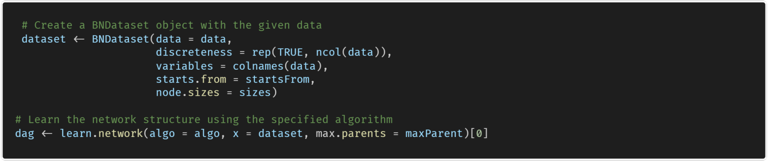


Figure: **Score** during different iterations of the training iterations

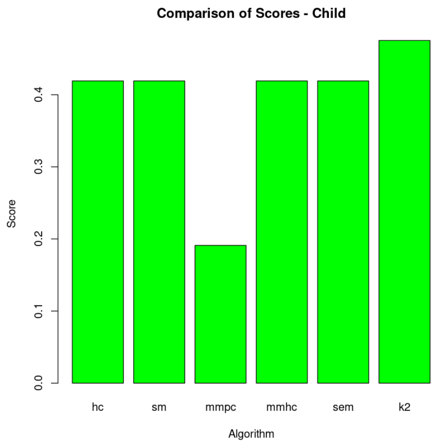
Overview of `bnstruct`

- Focused high-dimensional
- Specializes in handling mixed data types (continuous and discrete) and missing values. Performs imputation.
- Supports various algorithms such as hill-climbing (`hc`), constraint-based methods (`mmpc`), and hybrid algorithms (`mmhc`).
- Scoring methods available include Bayesian Information Criterion (BIC), Akaike Information Criterion (AIC), and Bayesian Dirichlet equivalent (BDe).



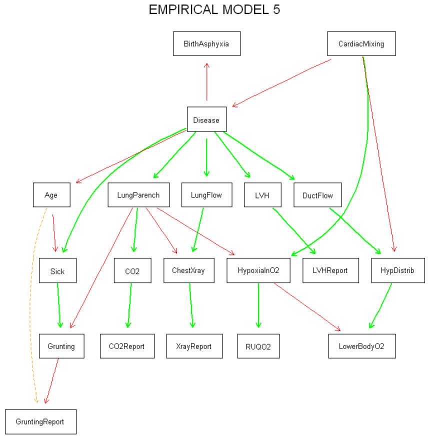
```
 # Create a BNDataset object with the given data
 dataset ← BNDataset(data = data,
                     discreteness = rep(TRUE, ncol(data)),
                     variables = colnames(data),
                     starts.from = startsFrom,
                     node.sizes = sizes)

# Learn the network structure using the specified algorithm
dag ← learn.network(algo = algo, x = dataset, max.parents = maxParent)[0]
```

Comparison of Algorithms - Child Dataset



- Applied different algorithms (`hc`, `sm`, `mmpc`, `mmhc`, `sem`) using `bnlearn` and `bnstruct` on the Child dataset.
- Evaluated the performance using Structural Hamming Distance (SHD) between the theoretical and learned networks.

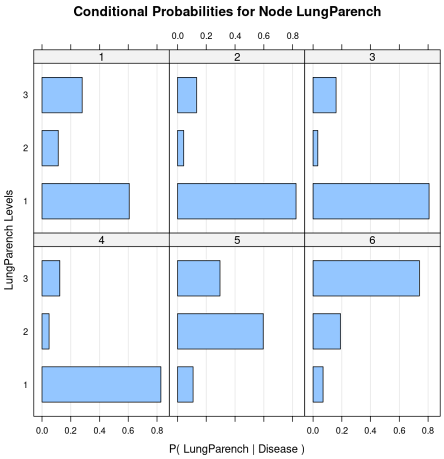
Dag mmpc - Child Dataset



Meaning of Colors and Lines

- **Green:** *True Positives (TP)*.
- **Red:** *False Positives (FP)*.
- **Dashed Orange:** *False Negatives (FN)*.

Fitting Overview



- **Objective:** Fit Bayesian networks to the Child dataset using different structures and examine the posterior distributions.
- **Method:** Used `bn.fit` with the `bayes` method and `iss = 10`.

Other possible studies

Other topics that could be studied are:

- time complexity of the algorithms
- how the percentage of the dataset used for the training influence the score and the shd from the theoretical DAG