Recommendation for product configuration with Bayesian networks and hard constraints

Hélène Fargier
Pierre-François Gimenez
Jérôme Mengin

IRIT-CNRS
University of Toulouse

AIGM – 14 December 2017

Context: product configuration for e-commerce (BR4CP project)

Configuration of complex, highly customizable products (combinatorial domains)

$\rightarrow$ cars, computers, travels, kitchens...

$\rightarrow$ number of possibilities exponential in the number of configuration variables

$\rightarrow$ all products aren't feasible (like a convertible car with a sunroof)

The constraints are hard : some products are infeasible

They come from :

- technical limitation (no sunroof on a convertible car)
- commercial consideration (no leather wheel on a lower-end car)
- stock variability (out-of-stock item)
- etc.

Help to choose a product: interactive configuration process

The user chooses a variable to assign and chooses a consistent value among the values proposed by the configurator

At each step, there is a partial, ongoing configuration

Recommendation = recommend, given a **partial configuration** u, a **value** for a **variable** Next

A good recommendation is:

- accurate
  $\rightarrow$ the user is willing to accept
- quick
  $\rightarrow$ on-line application

In our context:

- We have a sales history, no other information
  $\rightarrow$ no information about the user
- The user chooses the variables one by one
  $\rightarrow$ order of the variables is unknown
- There are constraints on allowed configurations
  $\rightarrow$ the issue of computing consistent values has been handled by others
- The sales history products may or may not satisfy the constraints

Recommendation in interactive configuration

Two categories of tools:

- *k*-nearest neighbours (*Coster et al.*, 2002)[1]
- Bayesian network

Goal: experiment and compare these methods

[1] Enhancing web-based configuration with recommendations and cluster-based help

Outline

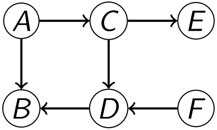
1. Context and issue
2. Algorithms
   1. based on Bayesian networks
   2. based on k-nearest neighbours
3. Experiments
4. Conclusion

Bayesian networks represent a probability distribution on the configurations by means of direct acyclic graph (DAG) and probability tables

- Each node is a variable
- An edge between A and B means that the probability of A depends on the value of B (and vice-versa)

Almost every probability distribution can be encoded into an Bayesian network

Computing a marginal $p(a \mid b)$ ("inference") is NP-hard



Probability $p(o)$ that a product o will be bought

Our recommendation is based on:

$$\underset{x \in \underline{\mathrm{Next}}}{\operatorname{argmax}}\, p(\mathrm{Next} = x \mid \mathrm{Assigned} = u)$$

Next is the variable the user chose, u the partial ongoing configuration

We assume that sales history is a representative sample of future user choices

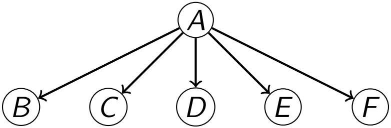
Two phases:

- Learning from Bayesian network the sales history off-line
  $\rightarrow$ constraints aren't taken into account during the learning
- The inference is done on-line
  $\rightarrow$ the learning isn't critical, the inference is

Naive Bayesian network: special case of Bayesian network with strong assumptions of independence

+ inference is quick

— roughly approximates the real probability distribution



3 algorithms based on k-nearest neighbours

Instead of using the whole sample, they use previous sales similar to the current one

The algorithms process differently these neighbours

Among the k-nearest neighbours of the current configuration (using the Hamming distance)

Weighted Majority Voter: each neighbours votes with a weight proportional to its similarity with the current configuration

Naive Bayes voter: uses the neighbours to learn a naive Bayesian network. In contrary to the "classical" naive Bayes, it cannot be learnt off-line

Most popular choice: computes the most probable configuration completion and recommend the value of Next in it

Most popular choice doesn't order the values of Next
$\rightarrow$ problem if the recommended value isn't allowed

Experimental protocol: 10 folds cross-validation
$\rightarrow$ history sales split into a training set and a test set

- Training set: Bayesian networks learning / neighbours searching
- Test set: for each item we simulate a configuration.
  For each recommendation for Next, we compare the recommended value with the value really chosen
  $\rightarrow$ Only one possible value: no evaluation
  $\rightarrow$ Recommanded = chosen: success, else: failure

We measure the success rate and the recommendation time w.r.t. the number of assigned variables

We have a method ("Oracle") to compute the lowest possible error rate.

Experiments made on i5 processor at 3.4GHz, using one core

All algorithms written in Java

Bayesian networks

- learning algorithm: hill climbing (hc) (R package *bnlearn*)
- inference algorithm: junction tree (*Jayes* library)

Neighbourhood size : 20
$\rightarrow$ has no significant impact on precision

Datasets from Renault, genuine sales history

- dataset "*Renault-44*" : 44 variables and 14786 examples including 8252 examples consistent with the constraints
- dataset "*Renault-48*" : 48 variables and 27088 examples including 710 examples consistent with the constraints
- dataset "*Renault-87*" : 87 variables and 17715 examples including 8335 examples consistent with the constraints

Datasets contain examples that don't satisfy the constraints

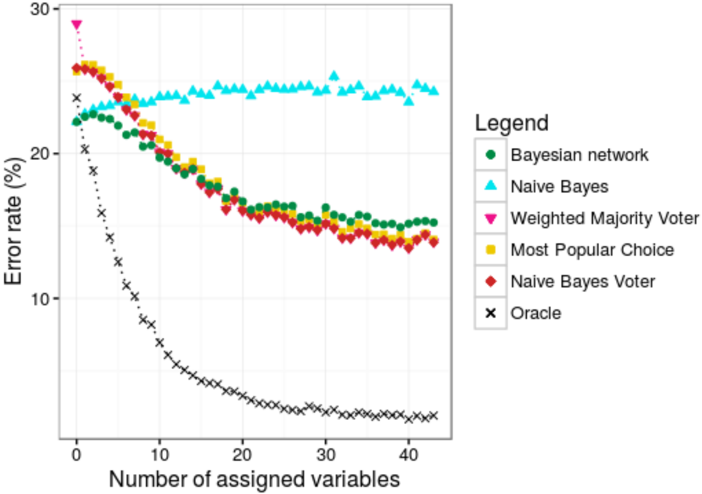
Should we learn these "invalid" examples or not ?

Results on *Renault-44* :

Precision	All examples	Consistent examples
Naive Bayes Voter	80.10	81.87
Weighted Maj. Voter	79.86	80.76
Most Pop. Choice	79.61	80.88
Bayesian network	80.86	81.72
Naive Bayesian net	76.29	78.08

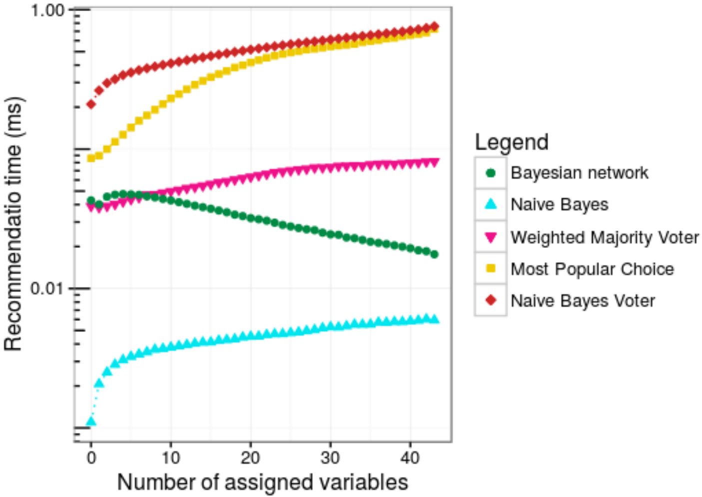
- Higher precision for Renault-44 and Renault-48
- Lower precision for Renault-87

Error rate w.r.t. the number of assigned variables



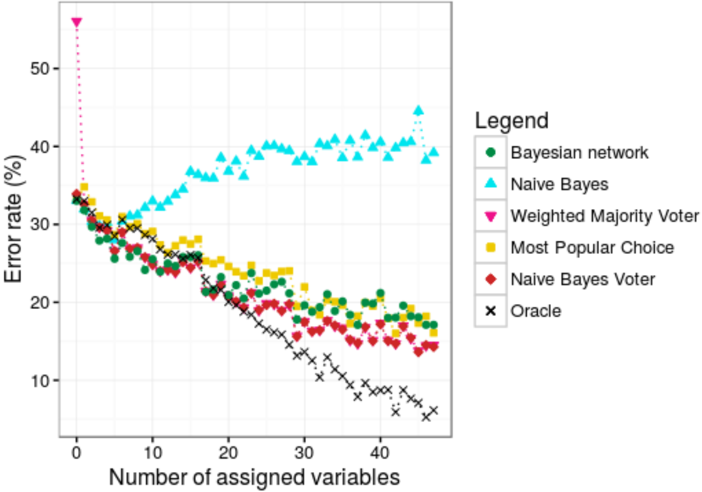
Experiment on *Renault-44* : 44 variables, 14786 examples including 8252 examples consistent with the constraints

Recommendation time w.r.t. the number of assigned variables



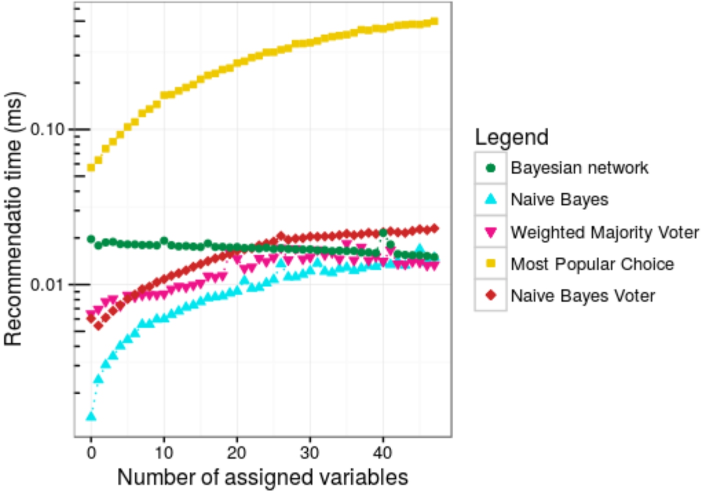
Experiment on *Renault-44* : 44 variables, 14786 examples including 8252 examples consistent with the constraints

Error rate w.r.t. the number of assigned variables



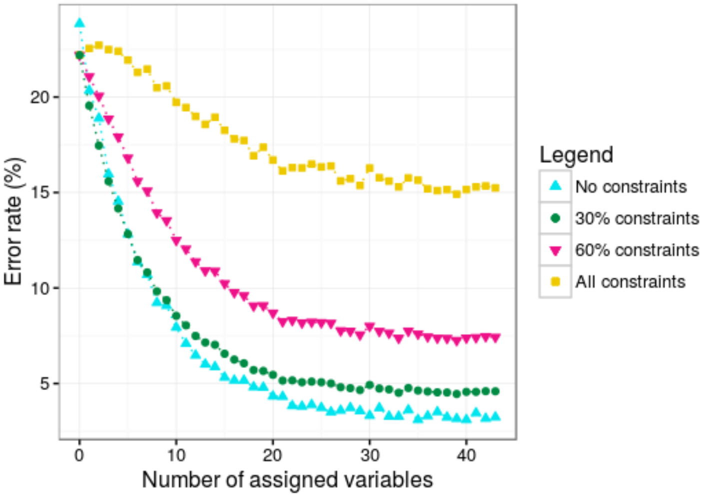
Experiment on *Renault-48* : 48 variables, 27088 examples including 710 examples consistent with the constraints

Recommendation time w.r.t. the number of assigned variables



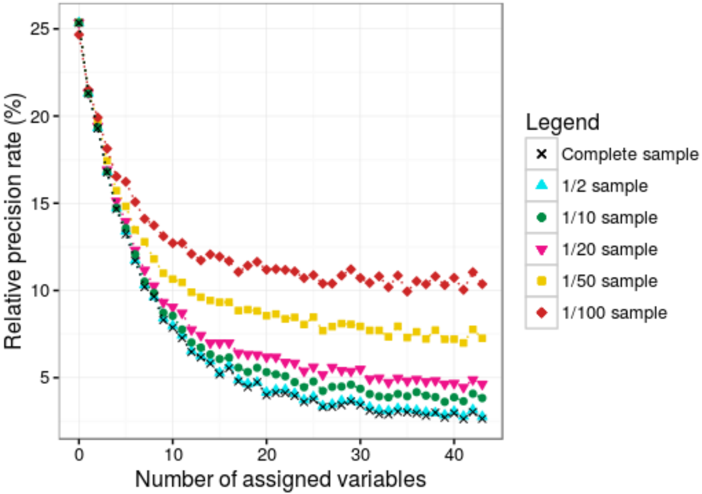
Experiment on *Renault-48* : 48 variables, 27088 examples including 710 examples consistent with the constraints

Error rate w.r.t. the amount of constraints



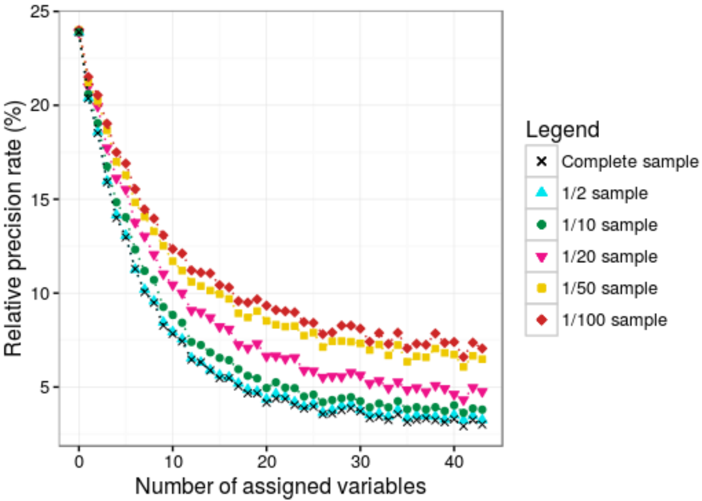
Experiment on *Renault-44* : 44 variables, 14786 examples including 8252 examples consistent with the constraints

Error rate w.r.t. the sample size (no const.) for Naive Bayes Voter



Experiment on *Renault-44* : 44 variables, 14786 examples

Error rate w.r.t. the sample size (no const.) for Bayesian network



Experiment on *Renault-44* : 44 variables, 14786 examples

Summary

- k-nearest neighbours and Bayesian networks are accurate and fast enough
- Naive Bayesian network is adapted when execution time is more critical than accuracy

- Bayesian networks are most robust to smaller sample size

- Constraints reduce the accuracy
- Learning only consistent examples : may be beneficial or harmful for the precision.