Bivariate Data Grid Models for Supervised Learning

Note technique

Référence : **Autre référence :** **Version :** **Date d'édition :**	NSM/R&D/TECH/EASY/TSI/4/MB 1.0 13/02/2008	*Vérifié par : Fabrice Clérot* *Affiliation : TECH/EASY* Le : 13/02/2008
Auteurs :	Boullé Marc TECH/EASY	*Approuvé par : Patrice Soyer* *Affiliation : TECH/EASY* Le : 13/02/2008

Résumé :
In the domain of data preparation for supervised learning, filter methods for variable ranking are time efficient. However, their intrinsic univariate limitation prevents them from detecting redundancies or constructive interactions between variables. This paper introduces a new method to automatically, rapidly and reliably evaluate the predictive information of a pair of variables. It is based on a partitioning of each input variable, in intervals in the numerical case and in groups of values in the categorical case. The resulting input data grid allows to evaluate the correlation between the two input variables and the output variable. The best joint partitioning is searched owing to a Bayesian model selection approach. Intensive experiments demonstrate the benefits of the approach, especially the significant improvement of accuracy for classification tasks.
This paper is divided into two chapters. Chapter 1 introduces the method, the bivariate evaluation criterion and reports the results of extensive experiments on artificial and real datasets. Chapter 2 focuses on the optimization algorithms with a detailed analysis of their computational complexity.
Mots clés : Data Mining, Data Analysis, Data Preparation, Discretization, Value Grouping, Bayesianism, Model Selection, Naive Bayes
Thème : 7800 - Intelligence artificielle - IA

Chapter 1

Optimal Bivariate Evaluation for Supervised Learning using Data Grid Models

Optimal Bivariate Evaluation for Supervised Learning using Data Grid Models

Marc Boullé

France Télécom R&D Lannion,
marc.boulle@orange-ftgroup.com

Abstract. In the domain of data preparation for supervised learning, filter methods for variable ranking are time efficient. However, their intrinsic univariate limitation prevents them from detecting redundancies or constructive interactions between variables. This paper introduces a new method[1] to automatically, rapidly and reliably evaluate the predictive information of a pair of variables. It is based on a partitioning of each input variable, in intervals in the numerical case and in groups of values in the categorical case. The resulting input data grid allows to evaluate the correlation between the two input variables and the output variable. The best joint partitioning is searched owing to a Bayesian model selection approach. Intensive experiments demonstrate the benefits of the approach, especially the significant improvement of accuracy for classification tasks.

1 Introduction

In a data mining project, the data preparation phase aims at constructing a data table for the modeling phase [Pyl99,CCK+00]. The data preparation is both time consuming and critical for the quality of the mining results. It mainly consists in a search of an efficient data representation, based on variable selection. The purpose of variable selection is three-fold: improve the classifier accuracy, reduce the training and deployment time, and ease the comprehensibility of the classifier [GE03,GGHD06]. Two main approaches, filter and wrapper [KJ97], have been studied in the literature. Filter methods evaluate the correlation between the input variables and the output variable, independently of any modeling technique. Wrapper methods search the best subset of variables for a given classification technique, used as a black box. Wrapper methods, which are very time consuming, are restricted to the modeling phase of data mining, as a post-optimization of a classifier. Filter methods are better suited to the data preparation phase, since they do not rely on modeling assumptions. In this paper, we focus on the filter approach.

[1] French patent N° 06 01499

1.1 Univariate filter methods

Univariate filter methods, also called ranking methods, evaluate each input variable individually. They allow to identify informative variables among a very large set of candidate variables. The input variables are ranked according to the method criterion and a subset of the variables can be selected once a threshold is chosen. The simplest way to determine this threshold is to keep as many variables as the modeling technique (often constrained by scalability issues) can handle. Another classical approach is to train the model with several subsets of increasing size. The best subset is chosen according to a tradeoff between the performance of the classifier and the size of the subset.

The most commonly used ranking methods are based on statistical tests [Sap90], such as the chi-square test for categorical input variables, or Student or Fisher-Snedecor tests for numerical input variables. These statistical tests are easy to apply, but they suffer from serious limitations. They are restricted to a Boolean discrimination between dependent and independent variables, which does not provide an accurate ranking of the input variables. They are also subject to strong constraints (minimum expected frequency in each cell of the contingency table for categorical variable, Gaussian distribution for numerical variables). Many alternative measures of associations between two variables have been studied in the context of decision trees [Kas80,BFOS84,Qui93,ZR00]. These criteria are based on a partition of the values of the input variable to evaluate the dependence between the input parts and the output values. Supervised discretization methods split the numerical domain into a set of intervals and supervised value grouping methods partition the input values into groups. Fine grained partitions allow an accurate discrimination of the output values, whereas coarse grained partitions tend to be more reliable. When the size of the partition is a free parameter, the trade-off between information and reliability is an issue. In the MODL (Minimum Optimized Description Length) approach, supervised discretization [Bou06a] (or value grouping [Bou05]) is considered as a non-parametric model of dependence between the input and output variables. The best partition is found using a Bayesian model selection approach, which provides a measure of association that is both accurate and reliable.

1.2 Multivariate filter methods

Filter methods suffer from their univariate limitation, being unable to reveal interactions between input variables. For example, redundant variables, which bring the same predictive information, cannot be detected. On the opposite, input variables might be uninformative in univariate evaluations and strongly informative in a joint evaluation. These two cases are illustrated in Figure 1, using multiple scatterplots where each point is drawn in a different shape according to its output value. The left diagram shows the case of two redundant variables. The right diagram corresponds to an XOR pattern: each input variable taken alone is a random mixture of the output values, whereas the two variables taken jointly allow a perfect discrimination of the output values.

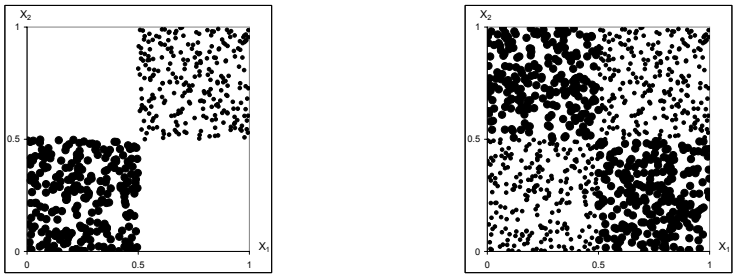


Fig. 1. Multiple scatterplots for two input variables X_1 and X_2, and two output values (small and large circles). The left diagram shows the case of two redundant variables and the right diagram the case of two jointly informative variables

In the case of two numerical input variables, multiple scatterplots are a popular visualization technique to detect interactions between the input and output variables. Scatterplot matrices [CLNL87] extend this technique to sets of input variables and allow to show all pairwise interactions between the variables. These methods are widely used in exploratory data analysis, but they do not provide an evaluation of the joint information contained in the variable pairs. Furthermore, these methods do not apply in the case of large numbers of variables: 100 input variables lead to $4950 = 100 * 99/2$ scatterplots, which cannot be managed by the data analyst.

An automatic evaluation of variable interactions is needed due to the increasing number of variables in datasets. The concept of mutual information between two attributes has been extended to the multivariate case [McG54,Han80] to quantify k-way interactions between variables. The approach is based on comparing the joint information of k variables and that of any subset of at most $(k-1)$ variables. The problem with this approach comes from the evaluation of the joint information, which is the same as evaluating the joint probability distribution function (PDF). In the case of categorical variables, the joint PDF is usually evaluated using the empirical distribution of the data, by counting the number of data instances for each combination of the values. In problems with many variables or with many values per variable, the joint PDF becomes sparse and its empirical evaluation unreliable. The main approach to avoid sparseness of the data is to assume partial independence between variables, which enables to estimate the joint PDF using factorization. The popular naive Bayes classifier [LIT92] relies on the assumption that input variables are independent given the output values. This assumption has been relaxed in semi-naive Bayesian classifiers [Kon91] or in Bayesian network classifiers [FGG97]. Experiments show that these methods improve the naive Bayes accuracy. However, these methods need to analyze dependencies between at least three variables (two input variables and one output variable) and are thus still subject to sparseness of the data. Furthermore, they apply only to categorical data: numerical variables are discretized using a supervised discretization method (MDLPC: Minimum Description Length Principal Cut [FI92] is used in [WBW05] for example), which

may destroy the dependencies between input variables. Another approach to avoid sparseness is to use latent variables. For example in [MC99,VR03], a clustering of the instances is performed for each output value and the discovered clusters are used as latent variables.

1.3 Our contribution

In this paper, we extend the MODL approach to the bivariate case for all pairs of input variables, numerical, categorical or mixed types. Each input variable is partitioned, in intervals in the numerical case and in groups of values in the categorical case. This joint partitioning defines a distribution of the instances in a bi-dimensional input data grid. The correlation between the cells of this data grid and the output values allows to quantify the joint predictive information. The tradeoff between information and reliability is established using a Bayesian model selection approach. This provides an evaluation criterion for any joint partitioning of the input variables. Several optimization heuristics, including pre-optimization and post-optimization are proposed to search the best possible joint-partitioning in a super-linear computation time.

Our method combines several interesting properties. It is able to manage both numerical and categorical input variables. It focuses on the conditional PDF only, is non parametric and non asymptotic. It is regularized to tackle the sparseness issue and optimally strike the balance between informative and reliable models. The models are efficiently optimized in super-linear computation time. Finally, it also provides a filter criterion for the selection of pairs of variables and it builds easily understandable models.

The paper is organized as follows. Section 2 summarizes the MODL method in the univariate case. Section 3 introduces the extension of the approach to the bivariate case and presents the resulting evaluation criterion. Section 4 summarizes the optimization algorithms, which are detailed in Chapter 2. Section 5 evaluates the effectiveness of the method on artificial datasets, where the joint PDF is known in advance. Section 6 demonstrates the benefits of the approach on real datasets, both for the data preparation and data modeling phases of data mining. Finally, section 7 gives a summary.

1.4 Related work

Multivariate discretization or similar techniques have already been proposed in various contexts. For example, the joint partitioning of the lines and rows of contingency table has been studied in the general case [NG05] for data exploration, or in the case of decision trees for the joint partitioning of one input variable and the output variable [ZRES05]. Multivariate discretization has also been developed in the case of association rule mining [Bay01], learning the structure of Bayesian network [SJ04] or for decision rule induction [KK99].

One main difference with these approaches is that our method models the conditional PDF only, not the distribution of the input data. Other major differ-

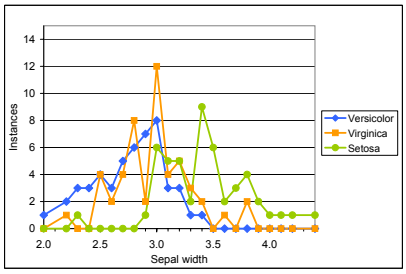
ences are our Bayesian approach for the evaluation criterion and our optimization algorithm with super-linear computation time.

2 The MODL univariate supervised evaluation methods

This section summarizes the MODL approach for supervised discretization [Bou06a] and value grouping [Bou05].

2.1 The MODL discretization method

The objective of supervised discretization is to induce a list of intervals which splits the numerical domain of a continuous input variable, while keeping the information relative to the output variable. A compromise must be found between information quality (homogeneous intervals with regard to the output variable) and statistical quality (sufficient sample size in every interval to ensure generalization). For example, we present on the left of Figure 2 the number of instances of each class of the Iris dataset [BM96] w.r.t. the sepal width variable. The problem is to find the partition of the numerical domain in intervals which gives us optimal information about the distribution of the data between the three output values.



	]-∞, 2.95[	[2.95, 3.35[	[3.35, +∞[
Versicolor	34	15	1
Virginica	21	24	5
Setosa	2	18	30
Total	57	57	36

Fig. 2. MODL discretization of the Sepal Width variable for the classification of the Iris dataset in 3 classes

In the MODL approach [Bou06a], the discretization is turned into a model selection problem. First, a space of discretization models is defined. The parameters of a specific discretization are the number of intervals, the bounds of the intervals and the output frequencies in each interval. Then, a prior distribution is proposed on this model space. This prior exploits the hierarchy of the parameters: the number of intervals is first chosen, then the bounds of the intervals and finally the output frequencies. The choice is uniform at each stage of the hierarchy. Finally, we assume that the multinomial distributions of the output values in each interval are independent from each other. A Bayesian approach is applied to select the best discretization model, which is found by maximizing

the probability $p(Model|Data)$ of the model given the data. Using the Bayes rule and since the probability $p(Data)$ is constant under varying the model, this is equivalent to maximizing $p(Model)p(Data|Model)$.

Let N be the number of instances, J the number of output values, I the number of intervals for the input domain. $N_{i.}$ denotes the number of instances in the interval i, and N_{ij} the number of instances of output value j in the interval i. In the context of supervised classification, the number of instances N and the number of classes J are supposed to be known. A discretization model is then defined by the parameter set $\left\{I, \{N_{i.}\}_{1\leq i\leq I}, \{N_{ij}\}_{1\leq i\leq I, 1\leq j\leq J}\right\}$.

It is noteworthy that the data partition obtained by applying such a discretization model is invariant by any monotonous variable transformation since it only depends on the variable ranks.

Owing to the definition of the model space and its prior distribution, the Bayes formula is applicable to exactly calculate the prior probabilities of the models and the probability of the data given a model. Taking the negative log of the probabilities, this provides the evaluation criterion given in formula 1.

$$\log N + \log \binom{N+I-1}{I-1} + \sum_{i=1}^{I} \log \binom{N_{i.}+J-1}{J-1} + \sum_{i=1}^{I} \log \frac{N_{i.}!}{N_{i1}!N_{i2}!\ldots N_{iJ}!} \quad (1)$$

The first term of the criterion stands for the choice of the number of intervals and the second term for the choice of the bounds of the intervals. The third term corresponds to the choice of the output distribution in each interval and the last term represents the conditional likelihood of the data given the model. Therefore "complex" models with large numbers of intervals are penalized.

Once the optimality of the evaluation criterion is established, the problem is to design a search algorithm in order to find a discretization model that minimizes the criterion. In [Bou06a], a standard greedy bottom-up heuristic is used to find a good discretization. In order to further improve the quality of the solution, the MODL algorithm performs post-optimizations based on hill-climbing search in the neighborhood of a discretization. The neighbors of a discretization are defined with combinations of interval splits and interval merges. Overall, the time complexity of the algorithm is $O(JN \log N)$.

The MODL discretization method for classification provides the most probable discretization given the data sample. Extensive comparative experiments report high quality performance. In the Iris example, the three intervals of the MODL discretization are shown on the left of Figure 2. The contingency table on the right gives us comprehensible rules such as "for a sepal width less than 2.95, the probability of occurrence of the Versicolor class is $34/57 = 0.60$".

2.2 The MODL value grouping method

Categorical variables are analyzed in a way similar to that of numerical variables, owing to a partitioning model of the input values. In the numerical case, the input

values are constrained to be adjacent and the only considered partitions are the partitions in intervals. In the categorical case, there are no such constraints between the values and any partition in groups of values is possible. For instance, Figure 3 illustrates the grouping of the values of the Cap Color variable of the Mushroom dataset [BM96]. The initial input values provide a fine grained estimation of the class conditional probabilities. The problem is to improve the reliability of this estimation owing to a reduced number of groups of values, while keeping the groups as much informative as possible. Producing a good grouping is harder with large numbers of input values since the risk of overfitting the data increases. In the extreme situation where the number of values is the same as the number of instances, overfitting is obviously so important that efficient grouping methods should produce one single group, leading to the elimination of the variable.

Value	edible	poisonous	Frequency
BROWN	55.2%	44.8%	1610
GRAY	61.2%	38.8%	1458
RED	40.2%	59.8%	1066
YELLOW	38.4%	61.6%	743
WHITE	69.9%	30.1%	711
BUFF	30.3%	69.7%	122
PINK	39.6%	60.4%	101
CINNAMON	71.0%	29.0%	31
GREEN	100.0%	0.0%	13
PURPLE	100.0%	0.0%	10

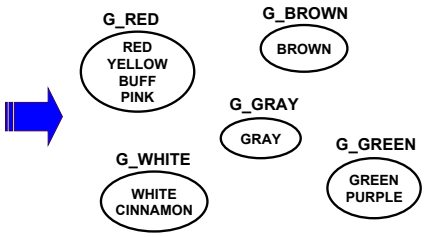


Fig. 3. MODL grouping of the values of the Cap Color variable for the classification of the Mushroom dataset in 2 classes

Let N be the number of instances, V the number of input values, J the number of output values and I the number of groups. $N_{i.}$ denotes the number of instances in the group i, and N_{ij} the number of instances of output value j in the group i. The Bayesian model selection approach is applied like in the discretization case and provides the evaluation criterion given in formula 2. This formula has a similar structure as that of formula 1. The two first terms correspond to the prior distribution of the partitions of the input values, in groups of values in formula 2 and in intervals in formula 1. The two last terms are the same in both formula.

$$\log V + \log B(V,I) + \sum_{i=1}^{I} \log \binom{N_{i.} + J - 1}{J - 1} + \sum_{i=1}^{I} \log \frac{N_{i.}!}{N_{i1}!N_{i2}!\ldots N_{iJ}!} \qquad (2)$$

$B(V,I)$ is the number of divisions of the V values into I groups (with eventually empty groups). When $I = V$, $B(V,I)$ is the Bell number. In the general case, $B(V,I)$ can be written as a sum of Stirling numbers of the second kind.

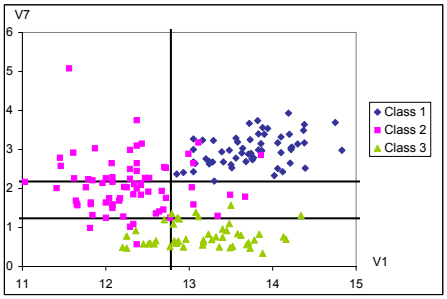
In [Bou05], a standard greedy bottom-up heuristic is proposed to find a good grouping of the input values. Several pre-optimization and post-optimization steps are incorporated, in order to both ensure an algorithmic time complexity of $O(JN\log(N))$ and to obtain accurate value groupings.

3 Extension to supervised bivariate evaluation

In this section, we extend the MODL methods to the supervised evaluation of pairs of input variables. We first introduce the approach using an illustrative example and then present the bivariate evaluation criterion in the case of two numerical variables. We generalize the criterion to any case of pairs of input variables and finally introduce the compression gain, a normalized version of the criteria.

3.1 Interest of the joint partitioning of two input variables

Figure 4 draws the multiple scatter plot (per class value) of the input variables V1 and V7 of the Wine dataset [BM96]. This diagram allows to visualize the conditional probability of the output values given the pair of input variables. The V1 variable taken alone cannot separate Class 1 from Class 3 for input values greater than 13. Similarly, the V7 variable is a mixture of Class 1 and Class 2 for input values greater than 2. Taken jointly, the two input variables allow a better separation of the class values.



]2.18, +∞[	(0, 23, 0)	(59, 0, 4)
]1.235, 2.18]	(0, 35, 0)	(0, 5, 6)
]-∞, 1.235]	(0, 4, 11)	(0, 0, 31)
V7xV1	]-∞, 12.78]	]12.78, +∞[

Fig. 4. Multiple scatterplot (per class value) of the input variables V1 and V7 of the Wine dataset. The optimal MODL supervised bivariate partition of the input variables is drawn on the multiple scatterplot, and the triplet of class frequencies per data grid cell is reported in the right table

Extending the univariate case, we partition the dataset on the cross-product of the input variables to quantify the relationship between the input and output variables. Each input variable is partitioned into a set of *parts* (intervals in the numerical case and groups of values in the categorical case). The cross-product of the univariate input partitions defines a *data grid*, which partitions the instances

into a set of *data cells*. Each data cell is defined by a pair of parts. The connection between the input variables and the output variable is evaluated owing to the distribution of the output values in each cell of the data grid. It is noteworthy that the considered partitions can be factorized on the input variables.

For instance in Figure 4, the V1 variable is discretized into 2 intervals (one bound 12.78) and the V7 variable into 3 intervals (two bounds 1.235 and 2.18). The instances of the dataset are distributed in the resulting bidimensional data grid. In each cell of the grid, the distribution of the output values can be estimated by counting. For example, the right table in Figure 4 shows that the cell defined by the intervals $]12.78, +\infty[$ on V1 and $]2.18, +\infty[$ on V7 contains 63 instances. These 63 instances are distributed on 59 instances for Class 1 and 4 instances for Class 3.

Coarse grained data grids tend to be reliable, whereas fine grained data grids allow a better separation of the output values. In our example, the MODL optimal data grid is drawn on the multiple scatter plot on the left of Figure 4.

3.2 Evaluation criterion for pairs of numerical variables

We extend the MODL approach to find the best tradeoff between information and reliability. We introduce in Definition 1 a family of bivariate partitioning models and select the best model owing to a Bayesian model selection approach. We first focus on numerical input variables, before generalizing to any pair of input variables, categorical or mixed types.

Definition 1. *A data grid model is a bivariate partitioning model defined by a partition of each input variable in a set of intervals and by a multinomial distribution of the output values in each cell of the data grid resulting from the cross-product of the univariate partitions.*

Notations.

- Y: output variable,
- X_1, X_2: input variables,
- N: number of instances,
- J: number of output values,
- I_1, I_2: number of intervals for each input variable,
- $N_{i_1..}$: number of instances in the interval i_1 of variable X_1,
- $N_{.i_2.}$: number of instances in the interval i_2 of variable X_2,
- $N_{i_1 i_2.}$: number of instances in the input data cell (i_1, i_2),
- $N_{i_1 i_2 j}$: number of instances of output value j in the input data cell (i_1, i_2).

A data grid model describes the distribution of the output values given the input values. It is completely defined by the numbers of intervals I_1 and I_2, the bounds of the intervals $\{N_{i_1..}\}$ and $\{N_{.i_2.}\}$ and the distribution of the output values $\{N_{i_1 i_2 j}\}$ in each cell (i_1, i_2) of the data grid . It is noteworthy that the numbers of instances per cell $\{N_{i_1 i_2.}\}$ do not belong to the parameters of the data grid models: they are derived from the definition of the two univariate partitions and from the dataset.

Any input information is used to define the family of models introduced in Definition 1. The bounds of the univariate partition come from the input values and the frequencies of the input data cells come from the dataset. In that sense, the data grid models are data dependent. What is described in the model is the connection between the input variables and the output variable.
We now introduce in Definition 2 a prior distribution on the parameters of the data grid models. This prior exploits the hierarchy of the parameters and is uniform at each stage of this hierarchy.

Definition 2. *The hierarchical prior of the data grid models is defined as follows:*

- *the numbers of input intervals are independent from each other, and uniformly distributed between* 1 *and* N*,*
- *for each input variable and for a given number of intervals, every partition in intervals is equiprobable,*
- *for each cell of the data grid, every distribution of the output values is equiprobable,*
- *the distributions of the output values in each cell are independent from each other.*

We apply the Bayesian model selection approach and obtain the evaluation criterion of a data grid model M in formula 3.

$$\begin{aligned} c(M) &= \log N + \log \binom{N + I_1 - 1}{I_1 - 1} + \log N + \log \binom{N + I_2 - 1}{I_2 - 1} \\ &+ \sum_{i_1=1}^{I_1} \sum_{i_2=1}^{I_2} \log \binom{N_{i_1 i_2 .} + J - 1}{J - 1} + \sum_{i_1=1}^{I_1} \sum_{i_2=1}^{I_2} \log \frac{N_{i_1 i_2 .}!}{N_{i_1 i_2 1}! N_{i_1 i_2 2}! \ldots N_{i_1 i_2 J}!} \end{aligned} \tag{3}$$

As in the case of univariate discretization (formula 1), the two first terms correspond to the prior probability of the parameters (number of intervals and choice of the bounds) of the discretization of the input variable X_1. Similarly, the two following terms correspond to the prior probability of the discretization of the input variable X_2. The binomial term in the first double sum represents the choice of the multinomial distribution of the output values in each cell. The multinomial term in the last double sum represents the conditional likelihood of the output values given the data grid model.

3.3 Evaluation criterion for any pair of variable

In the case of two categorical variables X_1 and X_2 with V_1 and V_2 input values, we apply the same approach. The X_1 variable is partitioned into I_1 groups of values (instead of intervals in the numerical case) and the X_2 variable into I_2 groups. The distribution of the output values is described in each cell of the data grid resulting from the joint partitioning of the input variables. Compared to the numerical case, the only change is the prior distribution of each univariate

partition. The impact in formula 3 is to replace the terms related to the prior distribution of the partition into intervals (two first terms of the univariate discretization of formula 1) by the corresponding value grouping terms (two first terms of the univariate value grouping of formula 2). The resulting evaluation criterion of a data grid model M for two categorical input variables is given in formula 4.

$$\begin{aligned} c(M) &= \log V_1 + \log B(V_1, I_1) + \log V_2 + \log B(V_2, I_2) \\ &+ \sum_{i_1=1}^{I_1} \sum_{i_2=1}^{I_2} \log \binom{N_{i_1 i_2 .} + J - 1}{J - 1} + \sum_{i_1=1}^{I_1} \sum_{i_2=1}^{I_2} \log \frac{N_{i_1 i_2 .}!}{N_{i_1 i_2 1}! N_{i_1 i_2 2}! \ldots N_{i_1 i_2 J}!} \end{aligned} \tag{4}$$

In the mixed case of one categorical variable X_1 with V_1 values and one numerical variable X_2, the first variable is grouped and the second one is discretized. The resulting evaluation criterion of a data grid model M for mixed type input variables is given in formula 5.

$$\begin{aligned} c(M) &= \log V_1 + \log B(V_1, I_1) + \log N + \log \binom{N + I_2 - 1}{I_2 - 1} \\ &+ \sum_{i_1=1}^{I_1} \sum_{i_2=1}^{I_2} \log \binom{N_{i_1 i_2 .} + J - 1}{J - 1} + \sum_{i_1=1}^{I_1} \sum_{i_2=1}^{I_2} \log \frac{N_{i_1 i_2 .}!}{N_{i_1 i_2 1}! N_{i_1 i_2 2}! \ldots N_{i_1 i_2 J}!} \end{aligned} \tag{5}$$

3.4 Compression gain

The evaluation criterion $c(M)$ given in formulas 3, 4, 5 is related to the probability that a data grid model M explains the output variable. The criterion $c(M)$ can also be interpreted as the ability of a data grid model to encode the output values given the input values, since negative log of probabilities are no other than coding lengths [Sha48]. Let $M_\emptyset$ be the null model with only one part for each univariate partition and one cell in the data grid. $c(M_\emptyset)$ represents the coding length of the output values when no input information is used, and is asymptotically equal to N times the Shannon's entropy of the output variable. More complex data grid models may better compress the output values, since the entropy of the output values is defined locally to each input cell. Fine grained cells allow to identify input regions where the output entropy is low (unbalanced mixture of the output values), but too complex data grid models with many cells are penalized with an increasing coding length of the model parameters.

Given these probabilistic and compression interpretations, we propose to use the evaluation criterion $c(M)$ to build a relevance criterion for each pair of input variables. The variable pairs can be sorted by decreasing probability of explaining the output variable. In order to provide a normalized indicator, we consider the following transformation of $c(M)$:

$$g(M) = 1 - \frac{c(M)}{c(M_\emptyset)}. \tag{6}$$

The compression gain $g(M)$ holds its values between 0 and 1 for models which are better than the null model ($g(M)$ is negative otherwise). It has value 0 for the null model and is maximal when the best possible explanation of the output values conditionally to the pair of input variables is achieved.

4 Optimization algorithms

The space of data grid models is so large that straightforward algorithms almost surely fail to obtain good solutions within a practicable computational time. Given that the MODL criterion is optimal, the design of sophisticated optimization algorithms is both necessary and meaningful. Such algorithms are described in Chapter 2. They finely exploit the sparseness of the data grids and the additivity of the MODL criterion, and allow a deep search in the space of data grid models with $O(N)$ memory complexity and a $O(N \log N)$ time complexity.

In this section, we give an overview of the data grid optimization algorithms which are fully detailed in Chapter 2.

Let us first focus on the case of two numerical input variables. The optimization of a data grid is a combinatorial problem. For each input variable X_1 and X_2, there are 2^N possible univariate discretizations, which represents $\left(2^N\right)^2$ possible bivariate discretizations. An exhaustive search through the whole space of models is unrealistic. We describe in algorithm 1 a greedy bottom up merge heuristic (GBUM) to optimize the data grids. The method starts with the maximum data grid M_{Max}, which corresponds to the finest possible univariate discretizations, with single value intervals. It evaluates all the merges between adjacent intervals, and performs the best merge if the evaluation criterion decreases after the merge. The process is reiterated until no further merge decreases the criterion.

Algorithm 1 Greedy Bottom Up Merge heuristic (GBUM)

Require: M {Initial data grid solution}
Ensure: $M^*, c(M^*) \leq c(M)$ {Final solution with improved cost}
1: $M^* \leftarrow M$
2: **while** improved solution **do**
3: **for all** Merge m between two parts of variable X_1 or X_2 **do**
4: $M' \leftarrow M^* + m$ {Evaluate merge m on data grid M^*}
5: **if** $c(M') < c(M^*)$ **then**
6: $M^* \leftarrow M'$
7: **end if**
8: **end for**
9: **end while**

Each evaluation of a data grid requires $O(N^2)$ time, since the initial data grid model M_{Max} contains N^2 cells. Each step of the algorithm relies on $O(N)$ evaluations of interval merges, and there are at most $O(N)$ steps, since the data

grid becomes equal to the null model $M_\emptyset$ once all the possible merges have been performed. Overall, the time complexity of the algorithm is $O(N^4)$ using a straightforward implementation of the algorithm. However, the method can be optimized in $O(N \log N)$ time, as demonstrated in Chapter 2. The optimized algorithm mainly exploits the sparseness of the data and the additivity of the evaluation criterion. Although a data grid may contain $O(N^2)$ cells, at most N cells are non empty. Thus, each evaluation of a data grid can be performed in $O(N)$ owing to a specific algorithmic data structure. The additivity of the evaluation criterion means that the criterion can be decomposed on the hierarchy of the components of the data grid: variables, parts and cells. Using this additivity property, all the merges between adjacent parts can be evaluated in $O(N)$ time. Furthermore, when the best merge is performed, the only impacted merges that need to be reevaluated for the next optimization step are the merges that share instances with the best merge. Since the data grid is sparse, the number of reevaluations of data grids is small on average. Sophisticated algorithmic data structures and algorithms are necessary to exploit these optimization principles and guarantee a time complexity of $O(N \log N)$.

The optimized version of the greedy heuristic is time efficient, but it may fall into a local optimum. First, the greedy heuristic may stop too soon and produce too many parts for each input variable. Second, the boundaries of the intervals may be sub-optimal since the merge decisions of the greedy heuristic are never rejected. The post-optimization algorithms described in [Bou06a] in the case of univariate discretization are applied alternatively to each input variable, for a frozen partition of the other input variable.

While post-optimizations may help to refine a good solution, the main heuristic may be unable to obtain such an initial good solution. This problem is tackled using the VNS meta-heuristic [HM01], which mainly benefits from multiple runs of the algorithms with different random initial solutions.

In the case of categorical variables, the combinatorial problem is still worse for large numbers of values V. The number of possible partitions of the values is equal to the Bell number $B(V) = \frac{1}{e} \sum_{k=1}^{\infty} \frac{k^V}{k!}$ which is far greater than the $O(N^2)$ possible discretizations. Furthermore, the number of possible merges between adjacent parts is $O(V^2)$ for categorical variables instead of $O(N)$ for numerical variables. Specific pre-processing and post-processing heuristics are necessary to efficiently handle the categorical input variables. Mainly, the number of groups of values is bounded by $O(\sqrt{N})$ in the algorithms, and the initial and final groupings are locally improved by exchange of values between groups. This allows to keep an $O(N)$ memory complexity and bound the time complexity by $O(N\sqrt{N} \log N)$ for categorical variables.

5 Evaluation on artificial datasets

In this section, the bivariate analysis method is evaluated using the optimization algorithms described in Chapter 2. The evaluation is performed on artificial datasets, where the true data distribution is known. Three patterns are consid-

ered: noise, chessboard and Gaussian mixture. An empirical study of the time complexity of the optimization algorithms is also reported.

5.1 Noise pattern

The purpose of the noise pattern experiment is to evaluate the noise resistance of the method, under variation of the sample size. The noise pattern dataset consists of an output variable independent from the input variables. The expected data grid contains one single cell, meaning that the class conditional information is independent from the input variables.

The output variable is equidistributed on two values. In the case of numerical input variables, the input values are uniformly distributed on the [0, 1] numerical domain. In the case of categorical input variables, the input values are equidistributed on V input values. Six families of noise datasets are considered: one for numerical input variables and five for categorical input variables with 2, 8, 32, 128 and 512 values. The experiment is performed on a large range of sample sizes ranging from 2 to 100,000 instances, in a geometric progression. Small sample sizes allow to study non-asymptotic behavior whereas large sample sizes focus on scalability issues.

The evaluated criterion is the number of cells in the data grid. In order to obtain reliable results, the experiment is performed on 100 randomly generated train datasets for each sample size. Figure 5 presents the mean cell number, for each dataset family and each sample size.

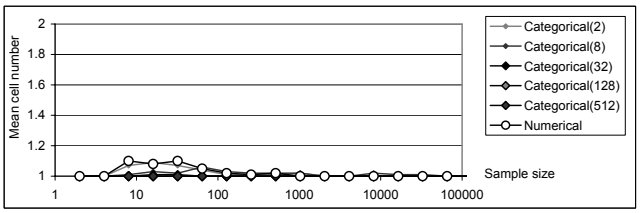


Fig. 5. Mean cell number for the noise pattern datasets, for sample size ranging from 2 to 100,000

Below sample size 4, there are not enough instances to constitute any pattern. Between sample size 10 and 100, a pattern is detected in about 10% of the cases for numerical input variables and for categorical input variables having less than two values. For larger sample sizes, or larger numbers of categorical input values, the noise pattern is almost always correctly detected.

5.2 Chessboard pattern

The purpose of the chessboard pattern experiment is to evaluate the ability of the method to identify complex bivariate patterns, that cannot be detected

using univariate analysis. A chessboard pattern of size 2 corresponds to an XOR pattern, as pictured in the right scatterplot of Figure 1. We generalize this XOR pattern using chessboards having S columns and S rows. Each input value is given an index between 1 and S, and the output value results from the parity of the sum of the indexes of the two input values.

In case of numerical input variables, the input values are uniformly distributed on the [0, S] numerical domain and rounded to the closest ceiling integer to obtain the input indexes. In case of categorical variables, S input values are considered, which directly provide the input indexes. Five sizes of chessboard patterns are considered (2x2, 8x8, 32x32, 128x128, 512x512) both for numerical or categorical input variables. The evaluation protocol is the same as for the noise pattern.

Figure 6 presents the mean cell number for each dataset in the numerical family and for each sample size. The expected data grid contains S^2 cells, resulting from two univariate discretizations of S intervals. The results exhibit the same behavior for all sizes of numerical chessboards. Below a given threshold, the number of instances is not sufficient to detect the pattern, and beyond this threshold, the expected number of cells is rapidly and accurately detected. About 16 instances are necessary to detect the 2x2 pattern, 250 instances for the 8x8 pattern, 2000 instances for the 32x32 pattern, 16000 for the 128x128 pattern and 150000 for the 512x512 pattern. Interestingly, large sample sizes allow to detect very complex patterns with a remarkably small average number of instances per data grid cell. About 4 instances per cell are necessary to detect the 2x2 pattern, and only 0.5 instance per cell on average for the 512x512 pattern.

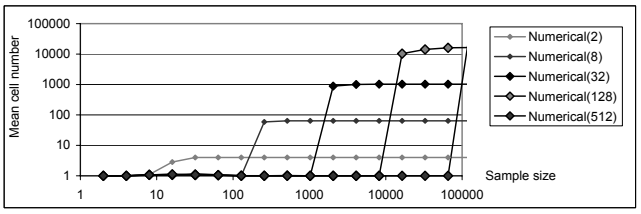


Fig. 6. Mean cell number for the numerical chessboard pattern datasets, for sample size ranging from 2 to 100,000

Figure 7 presents the mean cell number for each dataset in the categorical family and for each sample size. The expected data grid reduces to an XOR pattern with 4 cells, resulting from two univariate groupings of the S values into two groups (according the the parity of the value indexes). Compared to the numerical case, the univariate partitions are more complex to describe, but the output distribution can be described with only 4 cells (instead of S^2). The results exhibit the same kind of behavior as in the numerical case, but the pattern

detection thresholds are much smaller. Only 500 instances are sufficient to detect 128x128 pattern and 4000 for the 512x512 pattern, which is remarkably small.

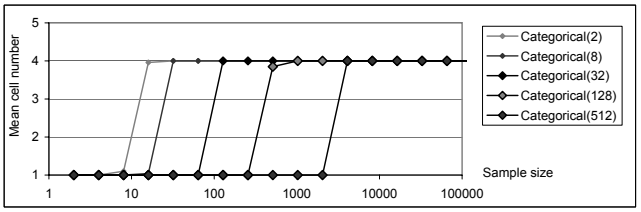


Fig. 7. Mean cell number for the categorical chessboard pattern datasets, for sample size ranging from 2 to 100,000

5.3 Gaussian mixture pattern

The purpose of the Gaussian mixture pattern is to evaluate the limits of the method, when applied to a dataset which is far from its learning bias. Although data grid models are non parametric and able to approximate any distribution provided that there are enough instances, they are biased in favor of constant conditional probabilities in each cell and of partition boundaries which are parallel to the axis. The true distribution in a Gaussian mixture pattern does obviously not fit this bias.

The pattern in the experiment contains two equidistributed output values. For each output value, the input values are distributed according to a bidimensional Gaussian vector with independent variables, as shown in Figure 8.

The evaluation protocol is similar to that of the noise pattern. We also evaluate the root mean square error (RMSE) in order to compare the true conditional probability p, known from the Gaussian mixture parameters, to the estimated conditional probability q, computed from the trained data grid. In each non empty cell (i_1, i_2) of the trained data grid, the conditional probability q of the output value j is estimated with $\frac{N_{i_1 i_2 j}+\epsilon}{N_{i_1 i_2 .}+J\epsilon}$, where $\epsilon = \frac{1}{N}$ is used to avoid zero values. For empty cells, the conditional probability is taken as that of the whole dataset according to $\frac{N_{..j}+\epsilon}{N+J\epsilon}$. A test dataset D_{Test} containing 100,000 instances is generated once for all the experiments, to evaluate the RMSE, according to formula 7.

$$RMSE = \sqrt{\sum_{(x,y)\in D_{Test}} \left(p(y|x) - q(y|x)\right)^2} \tag{7}$$

Figure 9 presents the mean cell number and Figure 10 the average RMSE value of the trained data grid models for each sample size. We also report the

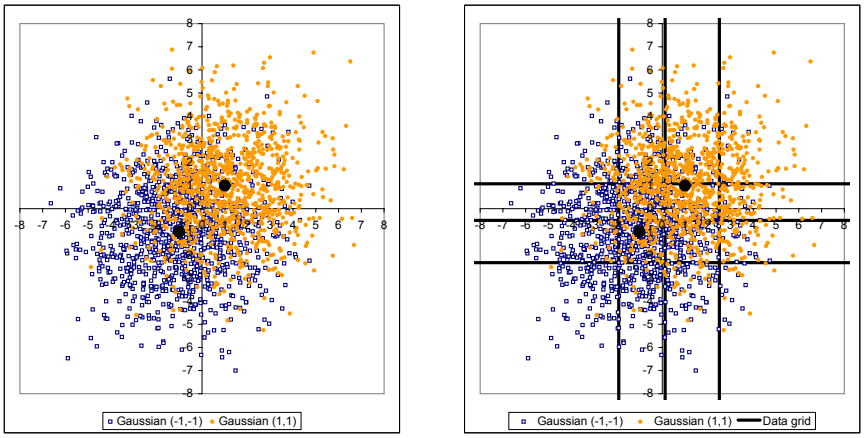


Fig. 8. Gaussian mixture for two Gaussian distributions centered on $(-1, -1)$ and $(+1, +1)$ with standard deviation of 2. The scatterplot displays 2000 instances, and the optimal data grid which contains 16 cells is presented on the right

RMSE results obtained by a basic parametric model, which assumes that there is exactly one Gaussian vector with independent variables for each output value. This parametric model estimates the means and variances of the Gaussian vectors (overall eight parameters) from the empirical data, which corresponds to a maximum likelihood estimate.

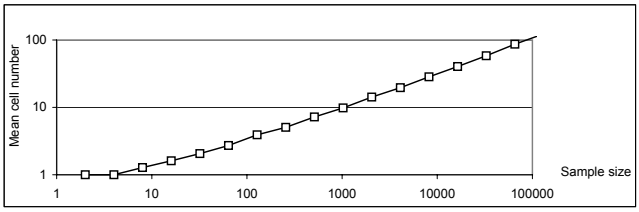


Fig. 9. Mean cell number for the Gaussian mixture pattern dataset, for sample size ranging from 2 to 100,000

The results allow to "quantify" the non asymptotic behavior of the method. Although data grids are non parametric models of conditional density estimation, only 8 instances are sufficient to detect that the data comes more probably from a pattern than from noise. The number of cells steadily grows with the sample size to better approximate the Gaussian mixture pattern. The quality of the approximation, estimated by the RMSE, is of course better with the parametric

method, which needs to estimate only 8 parameters instead of more than 100 for data grid models and sample size 100,000.

Let us now focus on the trade-off between the precision (as many cells as possible) and the reliability of the approximation (as many instances per cell as possible). In the case of this Gaussian pattern, the average number of cells is about $\sqrt{N/8}$ and the average number of instances per cell is about $\sqrt{8N}$, resulting in a $RMSE \approx N^{-1/4}$ (compared to a $RMSE \approx N^{-1/2}$ for the straightforward parametric model). Although these results are only experimental, they provide a quantitative insight on the behavior of the data grid models.

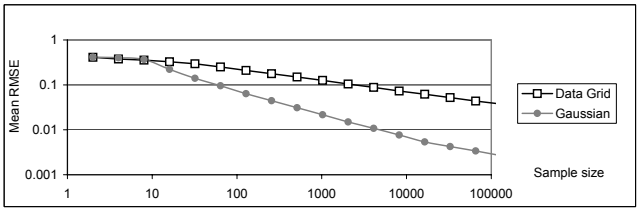


Fig. 10. Mean RMSE value for the Gaussian mixture pattern dataset, for sample size ranging from 2 to 100,000. The results are reported both for the data grid non parametric estimator and the Gaussian mixture parametric estimator

Overall, the method is both resilient to noise and able to detect complex fine grained patterns. It is able to approximate any conditional data distribution as close as requested, provided that there are enough instances in the train dataset.

5.4 Evaluation of the optimization algorithms

The objective of this section is to evaluate the computational efficiency of the data grid optimization heuristics and to investigate the main components of the algorithms introduced in Chapter 2: greedy bottom-up merge heuristic, metaheuristic and post-optimization.

The evaluation is performed on a PC with Intel P4 2.5 Ghz processor and 1 Go RAM. Figure 11 reports the average computation time w.r.t the sample size for four types of numerical patterns extracted from the experiments on artificial datasets. The results confirm that the algorithmic complexity is $O(N \log N)$, as shown by the corresponding slope drawn on the figure.

Figure 12 reports the same kind of results in the case of categorical patterns. According to the algorithmic study in Chapter 2, the computation time should be comprised between $O(N \log N)$ for small numbers of input categorical values and $O(N\sqrt{N} \log N)$ for numbers of values beyond $\sqrt{N}$. This is confirmed by Figure 12, which also shows that the computation time depends both on the number of input values and on the number of instances.

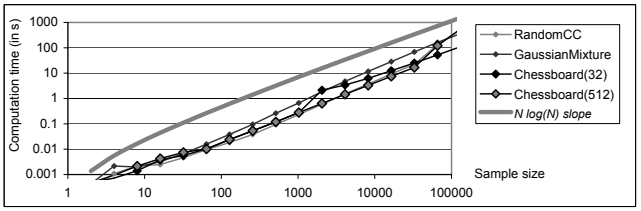


Fig. 11. Computation time for numerical bivariate patterns

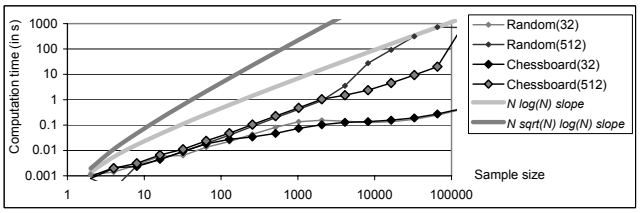


Fig. 12. Computation time for categorical bivariate patterns

We finally focus on the contribution of each algorithmic component described in Chapter 2 to the quality of the optimized data grids. We perform a comparative experiment using the numerical XOR pattern (cf. numerical chessboard pattern of size 2 x 2). The first heuristic evaluated is the greedy bottom-up merge heuristic (GBUM), which is at the heart of the optimization algorithms. The second one is the VNS meta-heuristic which runs the GBUM algorithm several times starting from random data grids of varying size. The third one is the same meta-heuristic, equipped with a post-optimization(VNS+PostOpt) after each run of the GBUM heuristic. This last heuristic is the complete data grid optimization algorithm used in all the experiments of this paper. The evaluation protocol is the same as that of the chessboard pattern experiments. Figure 13 reports the number of cells detected by each algorithm, w.r.t. the sample size.

Surprisingly, the GBUM heuristic needs a very large number of instances to discover the XOR pattern. This can be explained by the sparseness of the initial data grid, which contains N^2 cells, but at most N non empty cells. The GBUM algorithm performs about $2N$ merges between adjacent intervals, but only half of them involve non empty cells. This means that half of the merges are chosen "'randomly"', without being guided by the data. These random merges are likely to destroy the pattern in the early steps of the heuristic, so that the last merges are no longer able to detect the pattern.

The VNS meta-heuristic greatly improves the efficiency of the algorithm, and needs about 100 times less instances than the GBUM heuristic to detect the XOR pattern. The key point is that it starts from initial bivariate discretizations with fewer intervals, so that the initial data grid contains enough instances per

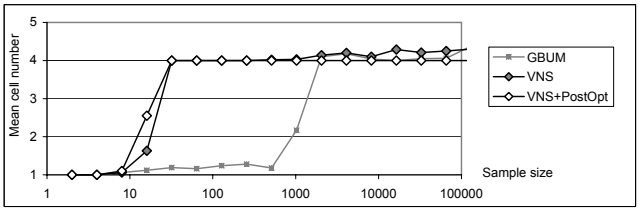


Fig. 13. Comparison of the performance of three data grid optimization algorithms for the detection of an XOR numerical pattern

cell to guide the merge process. The drawback is that the boundaries of the patterns are only approximated, since the initial random data grid are likely to miss the correct boundaries. This results in data grids of size 2 x 3, 3 x 2 or 3 x 3 for the XOR pattern. Although the meta-heuristic is partly able to improve this, finding the correct boundaries appears to be too difficult with the VNS randomization strategy. This is illustrated in Figure 13 by an average cell number beyond 4 for the VNS heuristic, especially for large sample sizes. The post-optimization heuristic performs an efficient local search around good initial solutions, and results in the correct detection of the XOR pattern.

The XOR pattern was chosen to be very simple for illustrative reasons. With more complex patterns, the differences of quality between each heuristic are far more important.

6 Evaluation on real datasets

This section evaluates the impact of the MODL bivariate method on supervised classification. The benefits for data preparation are first presented on one example dataset, prior to an intensive evaluation on many datasets.

6.1 Benefits for data preparation

This section evaluates the impact of the bivariate evaluation method on data preparation, especially concerning variable selection and data visualization. The Adult dataset [BM96] is used as an illustrative example. This dataset comes from the US census bureau and contains about 50,000 instances described with 15 input variables (8 categorical and 7 numerical). The objective of the classification task is to predict the output label '> 50K' (rich) or '<= 50K' (poor). The train dataset used in this experiment contains about 70% of the instances.

The univariate analysis is performed using the MODL discretization and value grouping methods. The results are reported in Table 1, where the input variables are sorted by decreasing compression gain g(M). The univariate analysis reveals that two variables, Label and Fnlwgt, are not informative. The two most informative variables are RelationShip and MaritalStatus,

with $g(M) \approx 20\%$. The CapitalGain Variable comes third ($g(M) \approx 13\%$), followed by a group of three variables, Education, Age and EducationNum, with $g(M) \approx 11\%$. This variable ranking provides a first insight on the informative variables, but new questions arise concerning the interactions between these variables. The interaction can be constructive, if there is more information in a pair of variable taken jointly than in the sum of the two univariate informations (like in the XOR pattern). It can be additive if the two variables bring independent information. Redundancy can also be detected if the pair of variables does not bring more information than the most informative variable of the pair. The bi-

Table 1. Univariate ranking of the input variables of the Adult Dataset, sorted by decreasing compression gain g(M)

Rank	g(M)	Variable	Type	Partition size
1	20.8%	Relationship	Categorical	4
2	19.7%	MaritalStatus	Categorical	4
3	13.5%	CapitalGain	Numerical	20
4	11.4%	Education	Categorical	7
5	11.4%	Age	Numerical	7
6	11.2%	EducationNum	Numerical	6
7	9.0%	Occupation	Categorical	7
8	7.1%	HoursPerWeek	Numerical	6
9	5.3%	CapitalLoss	Numerical	15
10	4.6%	Sex	Categorical	2
11	2.2%	Workclass	Categorical	4
12	1.0%	Race	Categorical	2
13	0.7%	NativeCountry	Categorical	3
14	0.0%	Label	Numerical	1
15	0.0%	Fnlwgt	Numerical	1

variate analysis is then performed using the method presented in the paper. The results are reported in Table 2 for the first twenty pairs of variables (among 105 pairs). The two most informative variables RelationShip and MaritalStatus look redundant, since they bring similar information in all the pairs of variables where they are involved. This is confirmed by an inspection of the MaritalStatus x Relationship pair, ranked 20^{th}, whose compression level is only slightly better than that of the Relationship variable taken alone. The redundancy is perfectly detected for the Education x EducationNum pair, since the corresponding data grid, ranked 67^{th}, reduces to a 7 x 1 grid identical to the univariate partition of variable Education. On the opposite, the interaction between Education and MaritalStatus is approximatively additive, since the compression gain of the pair (30.3%) is not far from the sum of the compression gains of each variable

Table 2. Bivariate ranking of the twenty first pairs of input variables of the Adult Dataset, sorted by decreasing compression gain g(M)

Rank	g(M)	Variable 1	Variable 2	Data Grid size
1	31.8%	CapitalGain	Relationship	12 x 4
2	31.1%	CapitalGain	MaritalStatus	15 x 3
3	30.3%	Education	Relationship	7 x 3
4	30.3%	Education	MaritalStatus	7 x 3
5	30.1%	EducationNum	Relationship	6 x 3
6	30.1%	EducationNum	MaritalStatus	7 x 3
7	27.4%	Occupation	Relationship	6 x 4
8	26.7%	MaritalStatus	Occupation	3 x 6
9	24.4%	CapitalLoss	Relationship	9 x 4
10	24.0%	Age	Relationship	5 x 3
11	23.9%	HoursPerWeek	Relationship	5 x 4
12	23.6%	Age	MaritalStatus	5 x 2
13	23.5%	CapitalLoss	MaritalStatus	9 x 3
14	23.5%	HoursPerWeek	MaritalStatus	5 x 3
15	22.6%	Age	CapitalGain	6 x 11
16	21.9%	Relationship	Workclass	4 x 4
17	21.7%	CapitalGain	Education	11 x 6
18	21.6%	CapitalGain	EducationNum	11 x 6
19	21.3%	NativeCountry	Relationship	2 x 4
20	21.1%	MaritalStatus	Relationship	3 x 3

(32.2%). Let us now visualize in Figure 14 the interaction between one numerical variable, EducationNum, and one categorical variable, MaritalStatus. The EducationNum variable is discretized in 7 intervals, with an increasing proportion of rich persons. The MaritalStatus variable is partitioned into three groups of values that look consistent: {Never-Married}, {Divorced, Separated, Widowed} and {Married-civ-spouse, Married-AF-spouse}. The two variables taken jointly bring a meaningful and easily understandable information. The proportion of rich people always increases with the number of education years, but the corresponding curve has not the same shape and is not at the same level according to the marital status. The data grid model can also easily be transformed into an understandable set of rules, with one rule for each cell. For example, the rightmost upper cell in Figure 14 can be described with the following rule: for married people with at least 15 years of education, the proportion of rich is beyond 80%.

Finally, let us focus in Figure 15 on two numerical variables, Age and EducationNum. This pair is ranked 31^{th} with a compression gain of 20.0%. The two input variables are discretized in 7 and 5 intervals (instead of 7 and 6 in the univariate case). The 3D histogram resulting from the data grid model al-

lows a to visualize the interaction between the two input variables in a clearly understandable way.

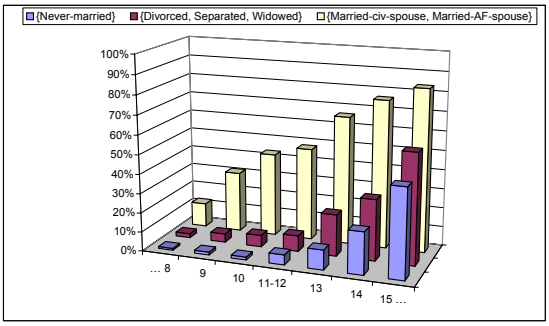


Fig. 14. 3D histogram for the Adult dataset, with the two input variables EducationNum and MaritalStatus on the X and Y axis and the percentage of rich people per cell on the Z axis

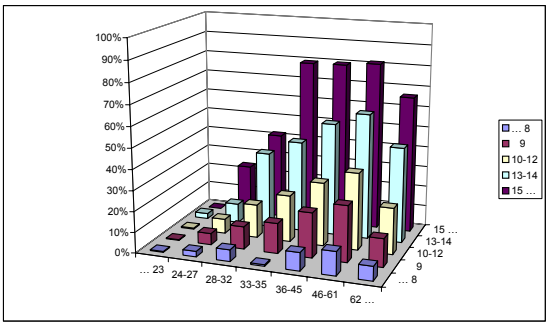


Fig. 15. 3D histogram for the Adult dataset, with the two input variables Age and EducationNum on the X and Y axis and the percentage of rich people per cell on the Z axis

Overall, the bivariate evaluation method is very helpful in the data preparation step of data mining, with ranking of pairs of variables, detection of constructive interactions or of redundancies in the representation space, and easily understandable visualizations of the joint conditional information carried out by each pair of variables.

6.2 Benefits for modeling

In this section, we focus on the predictive accuracy of the data grid models.

In order to evaluate the intrinsic performance of the MODL bivariate method, we introduce a new type of classifier called best bivariate (B2). This classifier first searches the best pair of variables, which maximizes the probability that its partitioning model explains the target variable. In order to classify a test instance, the input cell related to the instance is retrieved from the learned data grid and the majority target value of this cell is used for prediction. In case where this cell was empty in the learned data grid, the majority class value on the whole train dataset is used for prediction. For sanity check, we also evaluate the best univariate classifier (B1), which proceeds in the same way on the basis of the MODL univariate analysis, and we present the results of the majority classifier (M) which serves as a ground level reference.

In order to evaluate the impact of the method on multivariate classifiers, we evaluate the naive Bayes classifier [LIT92], on the basis of the univariate preprocessing (NB1) and bivariate preprocessing (NB2). We also exploit the enhancements of this classifier described in [Bou06b] [2] , which incorporates both variable selection and model averaging and results in a naive Bayes classifier with weighted variables. This enhanced selective naive Bayes classifier (SNB) is applied using the univariate preprocessing (SNB1) and bivariate preprocessing(SNB2). The bivariate preprocessing is basically exploited in the experiments, since each bivariate partitioning is simply managed as a constructed variable which expands the data representation space.

To summarize, the evaluated classifiers are:

- M: majority classifier,
- B1: best univariate classifier,
- B2: best bivariate classifier (based on the best preprocessed pair of variables),
- NB1: naive Bayes classifier,
- NB2: naive Bayes classifier (based on bivariate preprocessing),
- SNB1: selective naive Bayes classifier,
- SNB2: selective naive Bayes classifier (based on bivariate preprocessing).

The experiments are performed on 30 datasets from the UCI repository [BM96] described in Table 3. They represent a large variety of domains, instance numbers, variable numbers, types of variables (numerical or categorical) and numbers of target values. The test accuracy is evaluated using a stratified ten fold cross-validation. In order to determine whether the performances are significantly different between the SNB2 method and the alternative methods, the t-statistics of the difference of the results is computed, at the 5% confidence level.

The results are summarized in Table 4 with the mean of the test accuracy on all the datasets. The number of significant differences for the NB2 classifier

[2] Tool available as a shareware at http://www.francetelecom.com/en/group/rd/offer/software/technologies/middlewares/khiops.html.

Table 3. UCI Datasets

N°	Name	Instances	Numerical variables	Categorical variables	Classes	Majority accuracy
1	Abalone	4177	7	1	28	16.5
2	Adult	48842	7	8	2	76.1
3	Australian	690	6	8	2	55.5
4	Breast	699	10	0	2	65.5
5	Crx	690	6	9	2	55.5
6	German	1000	24	0	2	70.0
7	Glass	214	9	0	6	35.5
8	Heart	270	10	3	2	55.6
9	Hepatitis	155	6	13	2	79.4
10	HorseColic	368	7	20	2	63.0
11	Hypothyroid	3163	7	18	2	95.2
12	Ionosphere	351	34	0	2	64.1
13	Iris	150	4	0	3	33.3
14	LED	1000	7	0	10	11.4
15	LED17	10000	24	0	10	10.7
16	Letter	20000	16	0	26	04.1
17	Mushroom	8416	0	22	2	53.3
18	PenDigits	7494	16	0	10	10.4
19	Pima	768	8	0	2	65.1
20	Satimage	6435	36	0	6	23.8
21	Segmentation	2310	19	0	7	14.3
22	SickEuthyroid	3163	7	18	2	90.7
23	Sonar	208	60	0	2	53.4
24	Spam	4307	57	0	2	64.7
25	Thyroid	7200	21	0	3	92.6
26	TicTacToe	958	0	9	2	65.3
27	Vehicle	846	18	0	4	25.8
28	Waveform	5000	21	0	3	33.9
29	Wine	178	13	0	3	39.9
30	Yeast	1484	8	1	10	31.2

is also reported, as well as the average rank of each method. It is noteworthy that the classifier based on one single variable (B1) is as accurate as the best multivariate classifier evaluated in the benchmark (SNB2) in about one quarter of the datasets (no significant differences in 7 datasets out of 30). The classifier that selects only two variables (B2) obtains the best performance in about one third of datasets (10 datasets out of 30).

Table 4. Mean of the test accuracy, number of significant differences for the NB2 classifier and average rank of each classifier on 30 UCI datasets

	SNB2	NB2	SNB1	NB1	B2	B1	M
Mean	83.9%	81.9%	82.4%	81.4%	73.4%	67.6%	48.5%
Win/Draw/Loss		15/15/0	12/18/0	14/16/0	20/10/0	23/7/0	
Average rank	1.8	3.3	2.3	3.4	4.4	5.4	

In order to analyse the results with deeper details, Figure 16 presents the accuracy per dataset for the best univariate, best bivariate and naive Bayes classifiers, relatively to the accuracy of the majority classifier. The best bivariate classifier is always more accurate than the best univariate classifier, which confirms the capacity of the bivariate evaluation method to efficiently select a predictive pair of variables. However, the best bivariate classifier is significantly dominated by the naive Bayes classifier, which exploits the whole set of variables.

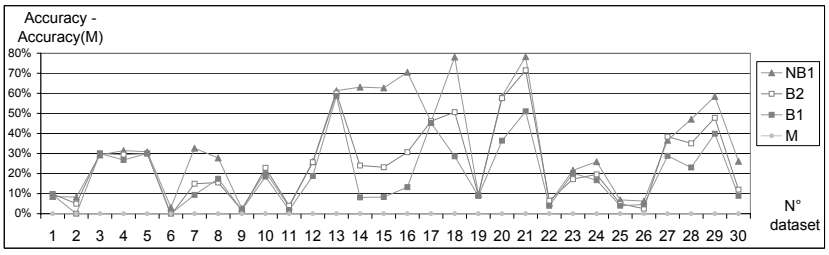


Fig. 16. Difference of accuracy between each evaluated classifier and the majority classifier (M), used as baseline. The evaluated classifiers are the best univariate (B1), best bivariate (B2) and naive Bayes (NB1) classifiers

Figure 17 focuses on the naive Bayes multivariate classifier, and studies the impact of exploiting or not the pairs of variables (NB2 and NB1) and of that of variable selection (SNB2 and SNB1). Using the pairs of variables enlarges

the representation space, which potentially allows to detect new predictive information. On the other hand, redundancies in the univariate representation are multiplied in the bivariate representation, which is detrimental to the naive Bayes assumption. Figure 17 shows that the two effects are observed on the datasets of the experiments, with significant loss of accuracy for datasets 1, 2, 6, 9, 22, 26, and strong gain of accuracy for datasets 16, 18, 20, 23, 27. The variable selection method [Bou06b] used in the SNB1 classifier confirms its beneficial impact on test accuracy, systematic but slight, compared to the NB1 classifier. When efficient variable selection is used together with the pairs of variables (SNB2), the gain in accuracy becomes both important, with an average improvement of 2.5% (15% for the Letter dataset), and highly significant, with 14 significant wins and 0 loss.

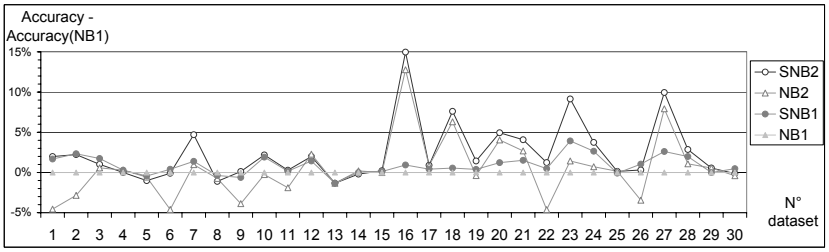


Fig. 17. Difference of accuracy between each evaluated classifier and the naive Bayes classifier (NB1), used as baseline. The evaluated classifiers are the naive Bayes classifier exploiting all pairs of variables (NB2) and the selective naive Bayes classifiers based on univariate preprocessing (SNB1) or bivariate preprocessing (SNB2)

7 Conclusion

The bivariate evaluation method introduced in this paper is based on a partitioning model of each input variables, in intervals for numerical variables and in groups of values for categorical variables. The cross-product of the univariate partitions, called a data grid, allows to quantify the conditional information relative to the output variable. The data grid models are evaluated using a Bayesian approach, and the best joint partitioning is searched in the model space owing to efficient heuristics.

Our method is non parametric both in the statistical and algorithmic sense : it does not rely on any statistical hypothesis for the data distribution (like Gaussianity for instance) and, as the criterion is regularized, there is no parameter to tune before optimizing it. This strong point enables to consider large datasets.

The data grid models are non asymptotic universal approximators of the class conditional density for pairs of input variables. Experiments on artificial

datasets show that the method is both very resilient to noise and able to detect complex fine grained patterns, even with few instances. This requires sophisticated algorithms, such as those described in Chapter 2. The experiments confirm that these algorithms run in $O(N \log N)$ time and that less complex algorithms fail to be as efficient.

The benefit of data grid models for data exploration is evaluated as a case study on a real dataset. The results demonstrate the ability of the method to detect constructive interactions or, on the opposite, redundancies between the input variables, and highlight the vizualisation and data understanding capacities of the data grids.

The impact of bivariate preprocessing on classification accuracy is evaluated through extensive experiments on 30 UCI datasets. The results show that the bivariate evaluation method is able to select strongly predictive pairs of variables. However, the average impact on classification accuracy is not conclusive for the naive Bayes classifier when all the pairs of variables are exploited. The problem is that the potential benefit of additional predictive information in the pairs of variables is balanced by the detrimental effect of increased redundancies in the presentation space. When the naive Bayes classifier is equipped with an efficient variable selection method, the benefit of bivariate preprocessing becomes both systematic and important: the classification accuracy always increases, with significant differences in half of the cases.

References

[Bay01] S.D. Bay. Multivariate discretization for set mining. *Machine Learning*, 3(4):491–512, 2001.

[BFOS84] L. Breiman, J.H. Friedman, R.A. Olshen, and C.J. Stone. *Classification and Regression Trees*. California: Wadsworth International, 1984.

[BM96] C.L. Blake and C.J. Merz. UCI repository of machine learning databases, 1996. http://www.ics.uci.edu/mlearn/MLRepository.html.

[Bou05] M. Boullé. A Bayes optimal approach for partitioning the values of categorical attributes. *Journal of Machine Learning Research*, 6:1431–1452, 2005.

[Bou06a] M. Boullé. MODL: a Bayes optimal discretization method for continuous attributes. *Machine Learning*, 65(1):131–165, 2006.

[Bou06b] M. Boullé. Regularization and averaging of the selective naive Bayes classifier. In *International Joint Conference on Neural Networks*, pages 2989–2997, 2006.

[CCK+00] P. Chapman, J. Clinton, R. Kerber, T. Khabaza, T. Reinartz, C. Shearer, and R. Wirth. *CRISP-DM 1.0 : step-by-step data mining guide*, 2000.

[CLNL87] D.B. Carr, R.J. Littlefield, W.L. Nicholson, and J.S. Littlefield. Scatterplot matrix techniques for large n. *Journal of the American Statistical Association*, 82:424–436, 1987.

[FGG97] N. Friedman, D. Geiger, and M. Goldsmidt. Bayesian network classifiers. *Machine Learning*, 29:131–163, 1997.

[FI92] U. Fayyad and K. Irani. On the handling of continuous-valued attributes in decision tree generation. *Machine Learning*, 8:87–102, 1992.

[GE03] I. Guyon and A. Elisseeff. An introduction to variable and feature selection. *Journal of Machine Learning Research*, 3:1157–1182, 2003.

[GGHD06] I. Guyon, S. Gunn, A. Ben Hur, and G. Dror. Design and analysis of the nips2003 challenge. In *Feature Extraction: Foundations And Applications*, chapter 9, pages 237–263. Springer, 2006.

[Han80] T.S. Han. Multiple mutual informations an multiple interactions in frequency data. *Information and Control*, 46(1):26–45, July 1980.

[HM01] P. Hansen and N. Mladenovic. Variable neighborhood search: principles and applications. *European Journal of Operational Research*, 130:449–467, 2001.

[Kas80] G.V. Kass. An exploratory technique for investigating large quantities of categorical data. *Applied Statistics*, 29(2):119–127, 1980.

[KJ97] R. Kohavi and G. John. Wrappers for feature selection. *Artificial Intelligence*, 97(1-2):273–324, 1997.

[KK99] W. Kwedlo and M. Kretowski. An evolutionary algorithm using multivariate discretization for decision rule induction. In *Principles of Data Mining and Knowledge Discovery*, pages 392–397, 1999.

[Kon91] I. Kononenko. Semi-naive Bayesian classifier. In Y. Kodrato, editor, *Sixth European Working Session on Learning (EWSL91)*, volume 482 of *LNAI*, pages 206–219. Springer, 1991.

[LIT92] P. Langley, W. Iba, and K. Thompson. An analysis of Bayesian classifiers. In *10th national conference on Artificial Intelligence*, pages 223–228. AAAI Press, 1992.

[MC99] S. Monti and G.F. Cooper. A latent variable model for multivariate discretization. In *The Seventh International Workshop on Artificial Intelligence and Statistics*, pages 249–254, 1999.

[McG54] W.J. McGill. Multivariate information transmission. *IEEE Trans. Information Theory*, 4(4):93–111, 1954.

[NG05] M. Nadif and G. Govaert. Block clustering of contingency table and mixture model. In *Intelligent Data Analysis*, pages 249–259, 2005.

[Pyl99] D. Pyle. *Data preparation for data mining*. Morgan Kaufmann Publishers, Inc. San Francisco, USA, 1999.

[Qui93] J.R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1993.

[Sap90] G. Saporta. *Probabilités analyse des données et statistique*. Technip, 1990.

[Sha48] C.E. Shannon. A mathematical theory of communication. Technical report, Bell systems technical journal, 1948.

[SJ04] H. Steck and T. Jaakkola. Predictive discretization during model selection. *Pattern Recognition*, LNCS 3175:1–8, 2004.

[VR03] R. Vilalta and I. Rish. A decomposition of classes via clustering to explain and improve naive Bayes. In *Proceedings of the 14th European Conference on Machine Learning*, pages 444–455, 2003.

[WBW05] G.I. Webb, J.R. Boughton, and Z. Wang. Not so naive bayes: Aggregating one-dependence estimators. *Machine Learning*, 58(1):5–24, 2005.

[ZR00] D.A. Zighed and R. Rakotomalala. *Graphes d'induction*. Hermes, France, 2000.

[ZRES05] D.A. Zighed, G. Ritschard, W. Erray, and V.M. Scuturici. Decision trees with optimal joint partitioning. *International Journal of Intelligent System*, 20(7):693–718, 2005.

Chapter 2

Optimization Algorithms for Bivariate Evaluation of Data Grid Models

Optimization Algorithms for Bivariate Evaluation of Data Grid Models

Marc Boullé

France Télécom R&D Lannion,
marc.boulle@orange-ftgroup.com

Abstract. In the domain of data preparation for supervised learning, many ranking methods have been proposed to assess the predictive information of individual input variable. For example, supervised discretization methods partition the input variable into a set of intervals, which allows to evaluate the correlation between the input and output variables. Such partitions can be extended to the bivariate case, owing to a partition of two input variables, in intervals in the numerical case and in groups of values in the categorical case. The resulting input data grid allows a joint evaluation of the two input variables with respect to the output variable. In this paper, we introduce a family of additive evaluation criteria for data grid models and present new algorithms which efficiently search the model space.

1 Introduction

In Chapter 1, we have introduced bivariate partitioning models called data grids in order to quantify the predictive importance of pairs of input variables in supervised learning. These models can be evaluated owing to an analytic criterion, which is Bayes optimal but is difficult to optimize. The purpose of this paper is to describe efficient algorithms for the optimization of data grid models. Their evaluation on artificial and real data is reported in Chapter 1.

Many discretization methods have been proposed in the literature to evaluate the correlation between a numerical input variable and an output variable [Cat91,Hol93,DKS95,ZR00,LHTD02]. Some discretization methods such as ChiMerge [Ker91] or MDLPC [FI92] evaluate a bipartition of one interval into two sub-intervals and apply the method recursively. Since the criterion evaluates only a local decision for two adjacent intervals, the discretization of the whole numerical domain cannot be optimized globally. Other discretization methods such as BalancedGain [KBR84], Fusinter [ZRR98] or MODL [Bou06] exploit a global criterion which evaluates the complete discretization, so that looking for an optimal discretization makes sense.

For some classes of global criteria such as additive criteria, an optimal algorithm based on dynamic programming [FKS95,ER96] allows to find the optimal discretization in $O(N^3)$, where N is the number of instances. A more practical

time complexity is achievable using top-down or bottom-up heuristics. Top-down methods start from the complete numerical domain interval and recursively split it into smaller intervals. Bottom-up methods start from the set of single value intervals and iteratively merge neighboring intervals. In the case of global criteria, each evaluation of a discretization requires $O(N)$ time since the discretization contains $O(N)$ intervals. At each step of the heuristic, there are $O(N)$ splits or merges to evaluate, and the number of steps is $O(N)$. Overall, a straightforward implementation of the heuristic runs in $O(N^3)$ time. However, in case of additive criteria, computation time can be saved provided that intermediate evaluation results are kept into memory. Using a careful implementation such as that described in [Bou04], the algorithm runs in $O(N \log N)$ time.

While discretization methods look for a partition of a numerical variable into intervals, value grouping methods search a partition of categorical variable into groups of values. Value grouping methods exploit either local criteria such as CHAID [Kas80] or global criteria such as Gain Ratio [Qui93]. Even with additive evaluation criteria, no optimal algorithm is available in the literature, since any partition of the input values is possible for value grouping. The time efficient bottom-up discretization heuristic can be applied to value grouping, but its time complexity is $O(N^2 \log N)$ since $O(N^2)$ merge decisions have to be evaluated for very large numbers of input values. In [Bou05], the bottom-up standard heuristic is enhanced with pre-processing and post-processing steps, in order to reach a practical $O(N \log N)$ time complexity without sacrifying the quality of the solution.

The univariate partitioning of an input variable has been extended to the bivariate case in Chapter 1, using data grid models. Each input variable is partitioned, in intervals in the numerical case and in groups of values in the categorical case. The resulting input data grid allows a joint evaluation of the two input variables with respect to the output variable. The optimization of a data grid is a combinatorial problem. Let us first focus on numerical input variables. We describe in Algorithm 1 an adaptation of the greedy bottom up merge heuristic (GBUM) to optimize the data grids. The method starts with the maximum data grid M_{Max}, which corresponds to the finest possible univariate discretizations, based on single value intervals. It evaluates all the merges between adjacent intervals, and perform the best merge if the evaluation criterion decreases after the merge. The process is reiterated until no further merge can decrease the criterion.

Each evaluation of a data grid requires $O(N^2)$ time, since the initial data grid model M_{Max} contains N^2 cells. Each step of the algorithm relies on $O(N)$ evaluations of interval merges, and there are at most $O(N)$ steps, since the data grid reduces to the null model $M_\emptyset$ once all the possible merges have been performed. Overall, the time complexity of the algorithm is $O(N^4)$ using a straightforward implementation of the algorithm.

We introduce in section 2 a family of additive criteria for data grid models and describe in section 3 an algorithmic structure that exploits the sparseness of the data in the bidimensional space. We show in section 4 how to exploit this

Algorithm 1 Greedy Bottom Up Merge heuristic (GBUM)

Require: M {Initial data grid solution}
Ensure: $M^*, c(M^*) \leq c(M)$ {Final solution with improved cost}
1: $M^* \leftarrow M$
2: **while** improved solution **do**
3: **for all** Merge m between two parts of variable X_1 or X_2 **do**
4: $M' \leftarrow M^* + m$ {Evaluate merge m on data grid M^*}
5: **if** $c(M') < c(M^*)$ **then**
6: $M^* \leftarrow M'$
7: **end if**
8: **end for**
9: **end while**

algorithmic structure to implement the GBUM heuristic in $O(N \log N)$ time for additive criteria in the case of numerical input variables. We present in section 5 several post-optimization heuristics, that still run in $O(N \log N)$ time. While post-optimizations may help to refine a good solution, the main heuristic may be unable to obtain such an initial good solution. We propose to tackle this problem using a meta-heuristic described in section 6, which mainly benefits from multiple runs of the algorithms with different random initial solutions. We extend the algorithm to handle categorical input variables in section 7. Finally, the algorithms are summarized in section 8.

2 Additive data grid evaluation criterion

This section formally states the definitions and notations related to data grid models and introduces a family of additive evaluation criteria, first in the case of univariate models, then in the case of bivariate data grid models.

2.1 Data grid models

Let us first focus on the univariate case and formalize in Definition 1 the univariate partitioning models introduced for the discretization of numerical variables [Bou06] and for the grouping of values of categorical variables [Bou05].

Definition 1. *An univariate partitioning model is defined by a set of intervals (resp. groups of values) and by the distribution of the output values in each interval (resp. group of values).*

Notations.

- Y: output variable,
- X: input variables,
- N: number of instances,
- J: number of output values,

- V: number of values(in the categorical case),
- I: number of parts (intervals or groups of values),
- $N_{i.}$: number of instances in the input part i,
- N_{ij}: number of instances of output value j in the input part i.

We now present in Definition 2 the data grid models introduced in Chapter 1. They consist in a family of bivariate partitioning models, the purpose of which is to evaluate the joint information between a pair of input variables and an output categorical variable. The input variables can be of any type, numerical or categorical.

Definition 2. *A data grid model is a bivariate partitioning model defined by a partition of each input variable, in intervals in the numerical case or groups of values in the categorical case, and by the distribution of the output values in each cell of the data grid resulting from the cross-product of the univariate partitions.*

The components of a *data grid* model are the input *variables*, the *parts* (intervals or groups of values) in the univariate partitions, and the *cells* (cross-product of two parts). An illustrative instance of data grid model is given in Figure 1.

Notations.

- Y: output variable,
- X_1, X_2: input variables,
- N: number of instances,
- J: number of output values,
- V_1, V_2: number of values for each input variable (in the categorical case),
- I_1, I_2: number of parts for each input variable,
- $N_{i_1..}$: number of instances in the part i_1 of variable X_1,
- $N_{.i_2.}$: number of instances in the part i_2 of variable X_2,
- $N_{i_1 i_2.}$: number of instances in the input data cell (i_1, i_2),
- $N_{i_1 i_2 j}$: number of instances of output value j in the input data cell (i_1, i_2).

A data grid model describes the distribution of the output values given the input values. It is completely defined by the numbers of parts I_1 and I_2, the specification of the univariate partitions which results in part frequencies $\{N_{i_1..}\}$ and $\{.N_{i_2.}\}$ and the distribution of the output values $\{N_{i_1 i_2 j}\}$ in each cell (i_1, i_2) of the data grid . It is noteworthy that the cell frequencies $\{N_{i_1 i_2.}\}$ do not belong to the parameters of the data grid models: they are derived from the specification of the two univariate partitions and from the dataset.

2.2 Univariate case

In the univariate case (discretization or value grouping), the considered models consist in a partition of one input variable X in I parts. We introduce in definition 3 a family of additive criteria to evaluate such partitions.

Definition 3. *An evaluation criterion $c(M)$ of a univariate partition model M is additive if it can be decomposed as a sum of the following terms:*
- *a variable criterion $c^{(V)}(X, I)$, which relies only on features of the input variable X and on the number of parts I in the partition,*
- *a part criterion $c^{(P)}(P_i)$ for each part P_i of the univariate partition of the input variable X, which relies only on features of the part.*

An additive univariate partition evaluation criterion can be written like in formula 1. In the rest of the paper, $C(M)$ is referred to as the cost of M.

$$c(M) = c^{(V)}(X, I) + \sum_{i=1}^{I} c^{(P)}(P_i) \tag{1}$$

For example, the discretization criterion [Bou06] is additive with

$$c^{(V)}(X, I) = \log N + \log \binom{N + I - 1}{I - 1},$$
$$c^{(P)}(P_i) = \log \binom{N_{i\cdot} + J - 1}{J - 1} + \log \frac{N_{i\cdot}!}{N_{i1}! N_{i2}! \dots N_{iJ}!}.$$

The value grouping criterion [Bou05] is also additive with

$$c^{(V)}(X, I) = \log V + \log B(V, I),$$
$$c^{(P)}(P_i) = \log \binom{N_{i\cdot} + J - 1}{J - 1} + \log \frac{N_{i\cdot}!}{N_{i1}! N_{i2}! \dots N_{iJ}!}.$$

We now describe in property 1 the impact of a merge between two parts for additive evaluation criteria.

Property 1. Let M be a univariate partition model of variable X, P_{i_a} and P_{i_b} two parts of the partition and M' the partition resulting from the merge of the two parts. Then the variation of the evaluation criterion $\delta c(M', M) = c(M') - c(M)$ can be decomposed as a sum of one variation term for the variable criterion and one variation term for the part criterion.

More formally, we have

$$\delta c(M', M) = \delta c^{(V)}(X, I) + \delta c^{(P)}(P_{i_a}, P_{i_b}), \tag{2}$$

with $\delta c^{(V)}(X, I) = c^{(V)}(X, I - 1) - c^{(V)}(X, I)$ and
$\delta c^{(P)}(P_{i_a}, P_{i_b}) = c^{(P)}(P_{i_a} \cup P_{i_b}) - c^{(P)}(P_{i_a}) - c^{(P)}(P_{i_b})$.

Property 1 means that each merge has only a local impact on the criterion. In the case of discretization, all the merges can thus be evaluated in $\mathrm{O}(N)$ time since each interval is involved in at most two merges, each of which evaluated in $\mathrm{O}(1)$. This key property is at the ground of the optimized version of the univariate greedy heuristic [Bou06], which runs in $\mathrm{O}(N \log N)$ time (and $\mathrm{O}(N)$ memory complexity) instead of $\mathrm{O}(N^3)$ time with a straightforward implementation.

2.3 Bivariate case

We now extend in definition 4 the notion of additive criterion to data grid models, which rely on a univariate partition of the values for each input variable, and for each cell of the cross-product of the univariate partitions, on the distribution of the output values.

Definition 4. *An evaluation criterion $c(M)$ of a data grid model M is additive if it can be decomposed as a sum of the following terms:*

- *a grid criterion $c^{(G)}(G)$, which relies only on the number $G = I_1 I_2$ of cells in the data grid,*
- *a variable criterion $c^{(V)}(X_k, I_k)$, which relies only on features of the input variable X_k ($k \in \{1, 2\}$) and on the number of parts I_k of its partition,*
- *a part criterion $c^{(P)}\left(P_{i_k}^{(k)}\right)$ for each part $P_{i_k}^{(k)}$ of the univariate partition of the input variable X_k ($k \in \{1, 2\}$), which relies only on features of the part,*
- *a cell criterion $c^{(V)}(C_{i_1 i_2})$ for each cell $C_{i_1 i_2}$ of the data grid, which relies only on features of the cell, and which is null for empty cells.*

An additive data grid evaluation criterion can be written like in formula 3.

$$\begin{aligned} c(M) = {} & c^{(G)}(G) + \sum_{k \in \{1,2\}} c^{(V)}(X_k, I_k) \\ & + \sum_{k \in \{1,2\}} \sum_{i_k=1}^{I_k} c^{(P)}\left(P_{i_k}^{(k)}\right) + \sum_{i_1=1}^{I_1} \sum_{i_2=1}^{I_2} c^{(C)}(C_{i_1 i_2}) \end{aligned} \tag{3}$$

For example, in the case of two numerical input variables, the evaluation criterion introduced in Chapter 1 is additive with

$$\begin{aligned} c^{(G)}(G) &= 0, \\ c^{(V)}(X_1, I_1) &= \log N + \log \binom{N + I_1 - 1}{I_1 - 1}, \\ c^{(V)}(X_2, I_2) &= \log N + \log \binom{N + I_2 - 1}{I_2 - 1}, \\ c^{(P)}\left(P_{i_1}^{(1)}\right) &= 0, \\ c^{(P)}\left(P_{i_2}^{(2)}\right) &= 0, \\ c^{(C)}(C_{i_1 i_2}) &= \log \binom{N_{i_1 i_2} + J - 1}{J - 1} + \log \frac{N_{i_1 i_2 .}!}{N_{i_1 i_2 1}! N_{i_1 i_2 2}! \ldots N_{i_1 i_2 J}!}. \end{aligned}$$

Since the number of instances N and of output values J are supposed to be known, they may be used in every component of the additive criterion. On the opposite, the features of variables, parts and cells are local to their component: they mainly consist in the frequency and frequency per output value locally

to the component (for example, $N_{i_1 i_2 \cdot}$ and $N_{i_1 i_2 j}$ frequencies in the cells). The other data grid evaluation criteria introduced in Chapter 1 in the case of two categorical variables or mixed type variables are also additive.

We now describe in property 2 the impact of a merge between two parts of the same variable in a data grid model for additive evaluation criteria. For sake of simplicity, we consider the variable X_1, without loss of generality.

Property 2. Let M be a data grid model, $P^{(1)}_{i_{1_a}}$ and $P^{(1)}_{i_{1_b}}$ two parts of the univariate partition of variable X_1 and M' the data grid resulting from the merge of the two parts. Then the variation of the evaluation criterion $\delta c(M', M) = c(M') - c(M)$ can be decomposed as a sum of the variation of the criterion for each component of the data grid involved in the merge.

More formally, we have

$$\begin{aligned} \delta c\left(M', M\right) = {} & \delta c^{(G)}\left(G, I_1\right) + \delta c^{(V)}\left(X_1, I_1\right) \\ & + \delta c^{(P)}\left(P_{i_a}, P_{i_b}\right) + \sum_{i_2=1}^{I_2} \delta c^{(C)}\left(C_{i_{1_a} i_2}, C_{i_{1_b} i_2}\right), \end{aligned} \tag{4}$$

$$\begin{aligned} \text{with } \delta c^{(G)}\left(G, I_1\right) &= c^{(G)}\left(G\frac{I_1 - 1}{I_1}\right) - c^{(G)}\left(G\right), \\ \delta c^{(V)}\left(X_1, I_1\right) &= c^{(V)}\left(X_1, I_1 - 1\right) - c^{(V)}\left(X_1, I_1\right), \\ \delta c^{(P)}\left(P^{(1)}_{i_{1_a}}, P^{(1)}_{i_{1_b}}\right) &= c^{(P)}\left(P^{(1)}_{i_{1_a}} \cup P^{(1)}_{i_{1_b}}\right) - c^{(P)}\left(P^{(1)}_{i_{1_a}}\right) - c^{(P)}\left(P^{(1)}_{i_{1_b}}\right) \text{ and} \\ \delta c^{(C)}\left(C_{i_{1_a} i_2}, C_{i_{1_b} i_2}\right) &= c^{(C)}\left(C_{i_{1_a} i_2} \cup C_{i_{1_b} i_2}\right) - c^{(C)}\left(C_{i_{1_a} i_2}\right) - c^{(C)}\left(C_{i_{1_b} i_2}\right). \end{aligned}$$

Given Property 2, the evaluation of a merge can be performed in O(1) for the data grid, variable and part components of the criterion and in O(1) per non empty cell involved in the merge.

3 Algorithmic structure for data grid optimization

In this section, we present an algorithmic structure designed to efficiently optimize data grids. This structure exploits the sparseness of the data in the bivariate case. Figure 1 shows a data grid for the (V1, V7) variables of the Wine dataset [BM96], with about half of the cells being empty. In the extreme case, the maximum data grid M_{Max} have O(N) parts in each univariate discretization and O(N^2) cells. However, this maximum data grid contains at most N non-empty cells, since the number of such cells is below the number of instances in the dataset.

We define below and illustrate in Figure 2 the components of an algorithmic structure that allows an efficient storage of data grids.

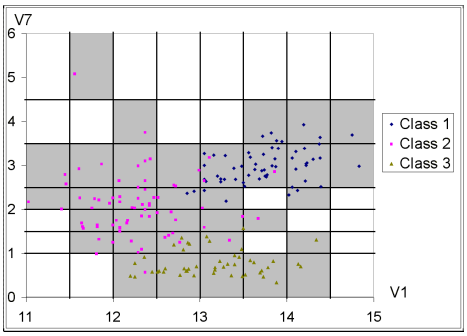


Fig. 1. Data grid for the input variables V1 and V7 of the Wine dataset. The V1 variable is discretized into 8 parts and the V7 variable into 7 parts. There are 34 non empty cells among the 56 cells of the data grid

- **Data Grid**: main data grid component, which collects the data grid features and contains a list a *Variable* components and a set of *Cell* components,
- **Variable**: sub-component of a *Data Grid*, which collects the variable features and contains a list of *Part* components,
- **Part**: sub-component of a *Variable*, which collects the part features and references a list of *Cell* components,
  - **Interval**: part in the case of a numerical variable,
  - **Value Group**: part in the case of a categorical variable,
- **Cell**: cell defined by its *Signature* (a pair of *Parts*).

In the rest of the paper, we will refer to the data grid concepts (variable, part, cell) using lower case characters and to the algorithmic components (Data Grid, Variable, Part, Cell) with a leading uppercase character.

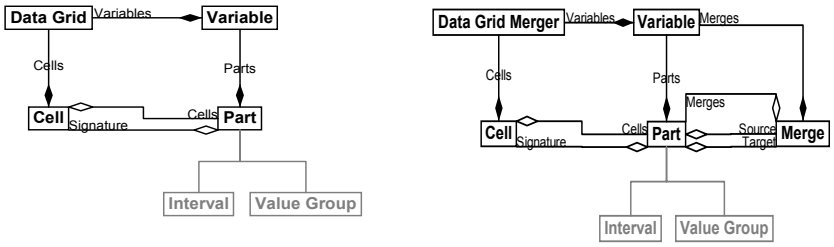


Fig. 2. Algorithmic structure Data Grid defined to store a data grid and its extension Data Grid Merger defined to store all the part merges considered by the bottom-up merge heuristic

For example, the data grid shown in Figure 1 contains 2 variables and 34 non-empty cells. The first variable contains 8 parts and the second one 7 parts. The first part of the V1 variable (interval $]-\infty, 11.5]$) references two cells, whose signatures are $(]-\infty, 11.5],]2, 2.5])$ and $(]-\infty, 11.5],]2.5, 3])$.

We now extend the Data Grid structure to the **Data Grid Merger** structure, which contains the list of all the possible part **Merges**. As shown in Figure 2, each variable contains the list of all the merges between two parts. Each merge references its *Source* part and its *Target* part, and each part maintains the list of all its related merges.

Finally, let us introduce two last terms which are useful to describe the optimization algorithms. Each merge corresponds to one input variable, called the *inner* variable. The other input variable is called the *outer* variable. Similarly, we relate to inner or outer parts (or to inner or outer merges), according to whether these components belong to the inner or outer variable.

4 Optimized implementation of the greedy bottom-up heuristic

In this section, we focus on numerical input variables. The case of categorical variables is discussed in section 7.

The purpose of this section is to demonstrate that the Greedy Bottom Up Merge (GBUM) heuristic (Algorithm 1) can run in $\mathrm{O}(N \log N)$ time, owing to the additivity of the criterion introduced in section 2 and to the Data Grid Merger algorithmic structure defined in section 3. We detail the main subroutines necessary to achieve this time complexity: initialization of the Data Grid structure in section 4.1, evaluation of all the possible merges and initialization of the Data Grid Merger structure in section 4.2, maintenance of this structure throughout the GBUM algorithm in sections 4.3 and 4.4. We summarize the algorithmic complexity results in section 4.5.

4.1 Initialization of the Data Grid structure

The Data Grid initialization subroutine is presented in Algorithm 2. Each instance in the dataset is stored in the Data Grid with at most one Cell and two Parts. Thus, the memory requirement of a Data Grid is $\mathrm{O}(N)$, like that of the dataset.

During the initialization, the Parts are first created, ranked by their input values. This requires a sort of the dataset for each variable, in $\mathrm{O}(N \log N)$ time. The cells can be efficiently initialized owing to a lookup table for the Parts of each Variable (based on a hash function of the input values) and a lookup table for the Cells of the Data Grid (based on a hash function of the cell signatures). These hash tables allow to create or retrieve each Cell in $O(1)$. Overall, the time complexity of the Data Grid initialization subroutine is $\mathrm{O}(N \log N)$.

Algorithm 2 Subroutine: initialization of a Data Grid Structure

Require: *Dataset* {Data in its tabular format (instances*variables)}
Ensure: *Data Grid* {Data in its Data Grid format}
Create an empty *Data Grid*
for all input variable $V \in \{X_1, X_2\}$ **do**
  Create an empty *Variable* V in the *Data Grid*
  Sort *Dataset* according to V
  Create initial *Parts* (*Interval* or *Value Group*) for each value of V
end for
for all instance in *Dataset* **do**
  Lookup the corresponding *Part* for each input *Variable*
  Compute the *Signature* of the corresponding *Cell*
  Lookup the *Cell* corresponding to this *Signature*
  if the *Cell* does not exist **then**
    Create the *Cell* in the *Data Grid*
  end if
  Update the *Cell* output value frequencies
end for

4.2 Initialization of the Data Grid Merger structure

The Data Grid Merger initialization subroutine is described in Algorithm 3. Since only adjacent intervals can be merged, there are at most $O(N)$ possible merges for each numerical input variable. Thus, the memory requirement of a Data Grid Merger is $O(N)$.

The main loop in the algorithm evaluates the local cost variation resulting from each merge, and takes benefit from the additivity of the evaluation criterion. The cost variation can be evaluated once at the variable level ($\delta c^{(V)}$) and at the data grid level ($\delta c^{(G)}$), since each merge results in the same new part number (see Property 2). On the opposite, the cost variation has to be evaluated locally to each merge at the part level ($\delta c^{(P)}$) for the two parts involved in the merge and at the cell level ($\delta c^{(C)}$) for the non-empty cells involved in merge. We show in Algorithm 4 that the merge evaluation subroutine requires a computation time linear in the number of cells involved in the merge. Since each cell participates to at most two merges (with the preceding and following interval), the overall computation time for all the merges is $O(N)$. Last, the merges are sorted according to their cost variation, in order to retrieve the most interesting merge. This sort requires $O(N \log N)$ time. Overall, the time complexity of the Data Grid Merger initialization subroutine is $O(N \log N)$.

We now comment the merge evaluation subroutine described in Algorithm 4. When the source and target parts are merged, the routine mainly evaluates the impact of the merge on the cells. Three situations may occur, as illustrated in Figure 3. In the first situation, the source cell collides with a target cell. This situation, identified in $O(1)$ per cell owing to the hash function of the cell signatures, has an impact on the cost variation related to the merge. In the two other situations, there is either no target cell corresponding to a source cell,

Algorithm 3 Subroutine: initialization of a Data Grid Merger Structure

Require: *Dataset* {Data in its tabular format (instances*variables)}
Ensure: *Data Grid Merger* {Data in its Data Grid Merger format}
Call subroutine 2 {Initialize the core Data Grid structure}
Initialize the cost of each component (*Data Grid, Variable, Part, Cell*)
for all *Variable* in the *Data Grid* **do**
    Compute the local variation of cost ($\delta c^{(V)}$) resulting from one part less in the variable partition
    for all *Part* **do**
        for all Possible merge between the *Part* and an adjacent *Part* **do**
            Create an empty *Merge*
            Link the *Merge* to each of its *Parts*
            Call subroutine 4 {Evaluation of the merge}
        end for
    end for
    Sort the *Merges* by decreasing cost variation and store them the *Variable* using a sortable list
end for

either no source cell corresponding to a target cell. This has no impact on the cost variation, since the involved cells are the same before and after the merge.

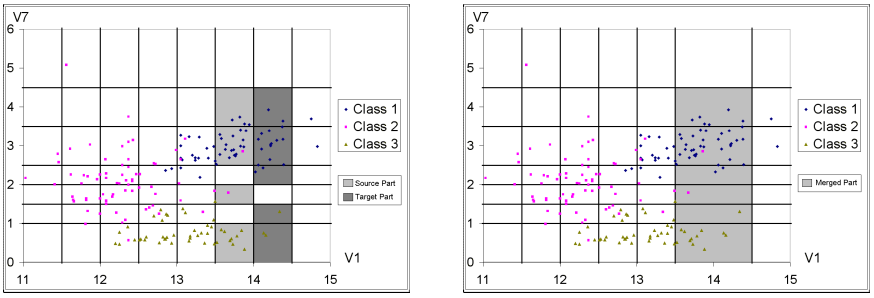


Fig. 3. Diagrams representing the merge between the source part $[13.5, 14.0[$ and the target part $[14.0, 14.5[$ of variable V_1 in the Wine dataset, before the merge (on the left) and after the merge (on the right). Three sources cells collide with three target cells, one source cell has no related target cell and two target cells have no related source cells

4.3 Completion of the merges: impacts on the Data Grid

Once the Data Grid Merger structure is initialized, the best merge per input variable can be retrieved in O(1) from the related sorted list of merges. Using the additivity of the evaluation criterion, we just have to add the cost variation related to the variable ($\delta c^{(V)}$) and to the data grid ($\delta c^{(G)}$) to retrieve the

Algorithm 4 Subroutine: evaluation of the local cost variation resulting from a Merge

Require: *Merge* $M^{(1)}$
{We assume that the inner variable of the merge is variable X_1 and that the number of cells is less in its source part than in its target part}
Ensure: Evaluation of $M^{(1)}$ $\{\delta c = \delta c^{(P)} + \sum \delta c^{(C)}\}$
$P_s^{(1)} \leftarrow$ source part of $M^{(1)}$
$P_t^{(1)} \leftarrow$ target part of $M^{(1)}$
$\delta c \leftarrow \delta c^{(P)}(P_s^{(1)}, P_t^{(1)})$
for all *Cell* $C \in P_s^{(1)}$ **do**
  $P^{(2)} \leftarrow$ outer part of C
  $s \leftarrow (P_t^{(1)}, P^{(2)})$ {compute the signature s of C' related to parts $(P_t^{(1)}, P^{(2)})$}
  if $s \in$ *Data Grid* **then**
    $C' \leftarrow$ *Cell* related to signature s {collision of cells during the merge}
    $\delta c \leftarrow \delta c + \delta c^{(C)}(C, C')$
  end if
end for

best merge among the two variable best merges. The next issue is to efficiently perform the merge and maintain the Data Grid structure.

Performing a merge is similar to evaluating a merge, except that all the components impacted by the merge in the data grid have to be updated. This is detailed in Algorithm 5.

During the completion of a merge, only the cells of the source part have an impact on the time complexity, since they have to be either merged with colliding target cells, either transfered to the target part. The target cells that do not collide with source cells are never considered in the merge completion subroutine. A close inspection of Algorithm 5 confirms that the time complexity of one merge completion is linear in the number of cells in the source part of the merge.

The time complexity of all the merge completions is then related to the total number of cells considered (cell merges or cell transfers) during the whole execution of the GBUM algorithm. Propositions 1 and 2 demonstrate that this process has an overall time complexity of $\mathrm{O}(N \log N)$.

Proposition 1. *The total number of cell merges during the whole execution of the GBUM algorithm is bounded by* N*.*

Proof. Since the initial number of non-empty cells is bounded by N, and since this number is decremented after each cell merge, the total number of cell merges is bounded by N. □

Proposition 2. *The total number of cell transfers or merges during the whole execution of the GBUM algorithm is bounded by* $O(N \log N)$*.*

Proof. Let us focus on one variable and on the part merges related to this variable. Let us first define precisely how we choose the source and target part of a

Algorithm 5 Subroutine: completion of a merge in the core Data Grid structure

Require: *Merge* $M^{(1)}$
{We assume that the inner variable of the merge is variable X_1 and that the number of cells is less in its source part than in its target part}
Ensure: *Data Grid* updated according to the merge
$P_s^{(1)} \leftarrow$ source part of $M^{(1)}$
$P_t^{(1)} \leftarrow$ target part of $M^{(1)}$
for all *Cell* $C \in P_s^{(1)}$ **do**
  $P^{(2)} \leftarrow$ outer part of C {the signature of C is $(P_s^{(1)}, P^{(2)})$}
  Unreference *Cell* C from its source *Part* $P_s^{(1)}$
  Unreference *Cell* C from the *Data Grid*
  $s \leftarrow (P_t^{(1)}, P^{(2)})$ {compute the signature s of C' related to parts $(P_t^{(1)}, P^{(2)})$}
  if $s \in$ *Data Grid* **then**
    {the source *Cell* C is merged in the target *Cell* C'}
    $C' \leftarrow$ *Cell* related to signature s
    Update the output value frequencies of C' with those of C
    Unreference the *Cell* C from its outer *Part* $P^{(2)}$
    Delete the orphan source *Cell* C
  else
    {the source *Cell* C is transfered to the target *Part* $P_t^{(1)}$}
    Replace the *Part* $P_s^{(1)}$ by *Part* $P_t^{(1)}$ in the *Cell* C
    Reference the *Cell* C in its new *Parts* $P_t^{(1)}$
    Reference the *Cell* C in the *Data Grid* with its new signature
  end if
end for
Unreference the source *Part* $P_s^{(1)}$ from the inner *Variable* X_1
Delete the orphan *Part* $P_s^{(1)}$

merge. The part of the merge having the smallest *cell* number is called the *cell-based* source part, and the one having the smallest *instance* number is called the *instance-based* source part. In Algorithm 5, source parts are chosen according to the cell-based rule. In this proof, we first study the algorithmic complexity for a variant of Algorithm 5 using the instance-based instead of the cell-based rule.

Let D be an instance and $\{M_k, 1 \leq k \leq K\}$ the list of the merges in which the instance D is considered, that is where D belongs to the source part. Let n_k be the number of instances in the source part of merge M_k. Since the number of instances in the target part is greater or equal than n_k, the number of instances in the new part resulting from the merge completion is at least twice n_k. The next merge M_{k+1} contains all the instances of this new part. We thus have $n_{k+1} \geq 2n_k$ and more generally $n_k \leq 2^{k-1}n_1$. Since $n_K \leq N$, the number K of merges that consider the instance D is bounded by $\mathrm{O}(\log N)$.

This is true for each instance and for each input variable. Thus, the total number of instances considered in the whole execution of the GBUM algorithm is bounded by $\mathrm{O}(N \log N)$, provided that the instance-based rule is used.

Let us now evaluate the algorithmic complexity using the initial Algorithm 5 which exploits the cell-based rule. The initial algorithm and its variant differ only on the way they choose the source part of each merge. Since the smallest number of cells between two parts is always lower than the smallest number of instances, the total number of cell operations (transfers or merges) is smaller using the cell-based rule of Algorithm 5. The claim follows. □

4.4 Completion of the merges: impacts on the Data Grid Merger

The last maintenance operation, summarized in Algorithm 6, is related to the impact of a merge completion on the other merges. To reach an efficient time complexity, the main issue is to restrict as much as possible the number of inner or outer merges that need to be reevaluated.

Let us analyze the detailed impacts of a merge completion and introduce the principles exploited to tackle the time complexity issue. Figure 4 illustrates the case of a merge in the Wine dataset. Once the merge is completed, at most two inner merges need to be reevaluated: these are the merges adjacent to the source part $P_s^{(1)}$ or to the target part $P_t^{(1)}$ of the merge under completion. On the other hand, potentially all the outer merges (related to variable V_7) have to be reevaluated. In fact, they need to be reevaluated only if they contain at least one cell involved in the merge under completion. More precisely, owing to the additivity of the criterion, the reevaluated outer merges are impacted by at most four cells, which they share with the merge under completion. This is formalized in proposition 3.

Proposition 3. *The number of cells that have an impact on the reevaluation of an outer merge is at most four.*

Proof. Let M be a data grid model with I_1 parts for variable X_1 and I_2 parts for variable X_2. Let $P_{i_{1_s}}^{(1)}$ and $P_{i_{1_t}}^{(1)}$ two parts of variable X_1 involved as the source

Algorithm 6 Subroutine: completion of a merge in the Data Grid Merger structure

Require: *Merge* $M^{(1)}$
{We assume that the inner variable of the merge is variable X_1 and that the number of cells is less in its source part than in its target part}
Ensure: *Data Grid Merger* updated according to the merge
1: **for all** inner merge M that need to be reevaluated **do**
2: Reevaluate M
3: Maintain the sortable list of merges of the inner variable
4: **end for**
5: **for all** outer merge M that need to be reevaluated **do**
6: Reevaluate M
7: Maintain the sortable list of merges of the outer variable
8: **end for**
9: Reevaluate the cost variation $\delta c^{(V)}$ for variable X_1
10: Reevaluate the cost variation $\delta c^{(G)}$ for the data grid

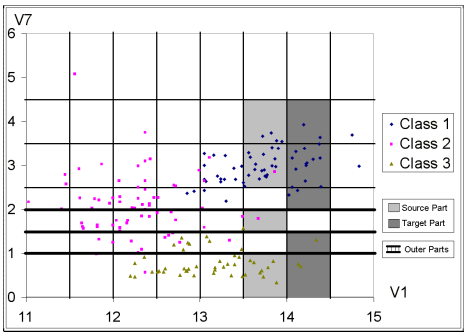


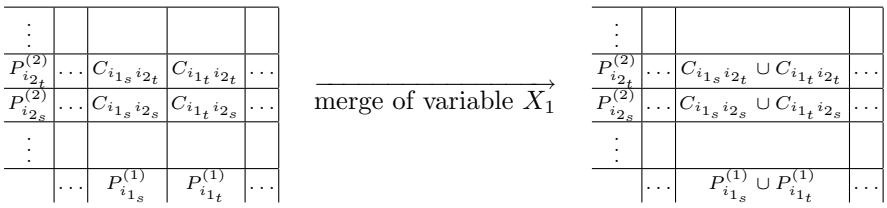
Fig. 4. Merge between the source part $P_s^{(1)} = [13.5, 14.0[$ and the target part $P_t^{(1)} = [14.0, 14.5[$ of the inner variable V_1 in the Wine dataset. One outer merge related to the outer variable V_7 is also represented, with its source part $P_s^{(2)} = [1.0, 1.5[$ and target part $P_t^{(2)} = [1.5, 2.0[$. Four cells (two of them are non-empty cells) shared by the merge under completion and the outer merge have an impact on the reevaluation of the outer merge

and target parts of a merge under completion. Let $P^{(2)}_{i_{2_s}}$ and $P^{(2)}_{i_{2_t}}$ the source and target parts of a merge of the outer variable X_2, M'_2 and M''_2 the data grid resulting from the merge of these outer parts before and after the merge between $P^{(1)}_{i_{1_s}}$ and $P^{(1)}_{i_{1_t}}$.

From the variation of the data grid cost given in formula 4, we get

$$\begin{aligned}\delta c\left(M'_2, M\right) = {} & \delta c^{(G)}\left(I_1 I_2, I_2\right) + \delta c^{(V)}\left(X_2, I_2\right) \\ & + \delta c^{(P)}\left(P^{(2)}_{i_{2_s}}, P^{(2)}_{i_{2_t}}\right) + \sum_{i_1=1}^{I_1} \delta c^{(C)}\left(C_{i_1 i_{2_s}}, C_{i_1 i_{2_t}}\right).\end{aligned} \tag{5}$$

After the merge completion, the new cost variation $\delta c\left(M''_2, M\right)$ involves the same cells belonging to the outer parts $P^{(2)}_{i_{2_s}}$ and $P^{(2)}_{i_{2_t}}$, except those belonging to the parts $P^{(1)}_{i_{1_s}}$ and $P^{(1)}_{i_{1_t}}$. This is illustrated below.



The new cost variation for the outer merge can thus be computed from the former cost variation.

$$\begin{aligned}\delta c\left(M''_2, M\right) = {} & \delta c\left(M'_2, M\right) \\ & + \delta c^{(G)}\left((I_1 - 1) I_2, I_2\right) - \delta c^{(G)}\left(I_1 I_2, I_2\right) \\ & + \delta c^{(C)}\left(C_{i_{1_s} i_{2_s}} \cup C_{i_{1_t} i_{2_s}}, C_{i_{1_s} i_{2_t}} \cup C_{i_{1_t} i_{2_t}}\right) \\ & - \delta c^{(C)}\left(C_{i_{1_s} i_{2_s}}, C_{i_{1_s} i_{2_t}}\right) - \delta c^{(C)}\left(C_{i_{1_t} i_{2_s}}, C_{i_{1_t} i_{2_t}}\right).\end{aligned} \tag{6}$$

The claim follows. □

Sixteen combinations of the four impacted cells may occur, according to whether they are empty or not. Let us call them *cell patterns* and analyze their impact on the cost variation of the outer merge (see formula 6).

The *empty* pattern is $\left|\begin{smallmatrix}\circ & \circ \\ \circ & \circ\end{smallmatrix}\right|$.

The *singleton* pattern ($\left|\begin{smallmatrix}\circ & \circ \\ \circ & \bullet\end{smallmatrix}\right|$, $\left|\begin{smallmatrix}\circ & \circ \\ \bullet & \circ\end{smallmatrix}\right|$, $\left|\begin{smallmatrix}\circ & \bullet \\ \circ & \circ\end{smallmatrix}\right|$, $\left|\begin{smallmatrix}\bullet & \circ \\ \circ & \circ\end{smallmatrix}\right|$) involves one single cell in the outer merge. The cost variation of the outer merge remains unchanged.

The *outer collision* pattern ($\left|\begin{smallmatrix}\bullet & \circ \\ \bullet & \circ\end{smallmatrix}\right|$, $\left|\begin{smallmatrix}\circ & \bullet \\ \circ & \bullet\end{smallmatrix}\right|$) involves exactly two cells that collide for the outer variable. The cost variation of the outer merge remains unchanged since the collision between the two cells is the same before and after the inner merge.

The *inner collision* pattern ($\left|\begin{smallmatrix}\circ&\circ\\\bullet&\bullet\end{smallmatrix}\right|$, $\left|\begin{smallmatrix}\bullet&\bullet\\\circ&\circ\end{smallmatrix}\right|$, $\left|\begin{smallmatrix}\circ&\bullet\\\bullet&\bullet\end{smallmatrix}\right|$, $\left|\begin{smallmatrix}\bullet&\circ\\\bullet&\bullet\end{smallmatrix}\right|$, $\left|\begin{smallmatrix}\bullet&\bullet\\\circ&\bullet\end{smallmatrix}\right|$, $\left|\begin{smallmatrix}\bullet&\bullet\\\bullet&\circ\end{smallmatrix}\right|$, $\left|\begin{smallmatrix}\bullet&\bullet\\\bullet&\bullet\end{smallmatrix}\right|$) involves at least two cells that collide in the inner merge. The cost variation of the outer merge needs to be reevaluated, since some cells are merged during the part merge under completion.

The *diagonal* pattern ($\left|\begin{smallmatrix}\circ&\bullet\\\bullet&\circ\end{smallmatrix}\right|$, $\left|\begin{smallmatrix}\bullet&\circ\\\circ&\bullet\end{smallmatrix}\right|$) involves one outer cell merge after the current inner merge completion, whereas no cell needed to be merged before the merge completion. The cost variation of the outer merge must then be reevaluated.

All in all, a close look at all the cell patterns allow to derive the interesting property 3.

Property 3. An outer merge does not need to be reevaluated when the cell pattern is like $\left|\begin{smallmatrix}\circ&*\\\circ&*\end{smallmatrix}\right|$, that is when it contains only empty cells in the source part of the merge under completion.

We now present in lemmas 1, 2, 3, 4 and 5 intermediate results related to the time complexity of the Data Grid Merger maintenance operations. We conclude in proposition 4 that the time complexity of all these maintenance operations is $\mathrm{O}(N \log N)$.

Lemma 1. *The total number of inner merges reevaluated during the whole execution of the GBUM algorithm is bounded by $O(N)$.*

Proof. Since at most two inner merges need to be reevaluated after each merge completion, and since the number of merge completions is at most N for each input variable, the claim follows. □

Lemma 2. *The total number of cells considered in the inner merges during the whole execution of the GBUM algorithm is bounded by $O(N \log N)$.*

Proof. The principle of the proof is similar to that of Proposition 2.

For each merge, at most two inner merges (*left* and *right*) must be reevaluated. Let us focus on one variable, on the part merges related to this variable and on the reevaluation of the left inner merges.

In Algorithm 6, source parts are chosen according to the cell-based rule (see Proposition 2 for a definition of the cell-based and instance based rules). In this proof, we first study the algorithmic complexity for a variant of Algorithm 6 using the instance-based instead of the cell-based rule.

Let D be an instance and $\{M_k, 1 \leq k \leq K\}$ the list of the reevaluated left inner merges in which the instance D is considered. Let n_k be the number of instances in the source part of merge M_k. Since the number of instances in the target part is greater or equal than n_k, the number of instances in the new part resulting from the reevaluated merge is at least twice n_k. The next reevaluated merge M_{k+1} contains all the instances of this new part. We thus have $n_{k+1} \geq 2n_k$ and more generally $n_k \leq 2^{k-1} n_1$. Since $n_K \leq N$, the number K of reevaluated left inner merges which consider the instance D is bounded by $\mathrm{O}(\log N)$.

This is true for each instance, for each input variable and for each type (*left* and *right*) of impacted inner merge. Thus, the total number of instances considered in the whole maintenance operations of the Data Grid Merger structure is bounded by $O(N \log N)$, provided that the instance-based rule is used.

Let us now evaluate the algorithmic complexity using the initial Algorithm 6 which exploits the cell-based rule. The initial algorithm and its variant differ only on the way they choose the source part of each merge. Since the smallest number of cells between two parts is always lower than the smallest number of instances, the total number of cell operations (transfers or merges) is smaller using the cell-based rule of Algorithm 6. The claim follows.

□

Lemma 3. *The total number of outer merges reevaluated during the whole execution of the GBUM algorithm is bounded by $O(N \log N)$.*

Proof. Exploiting property 3, an outer merge is reevaluated only if it contains a cell considered in the current inner merge. From proposition 2, the total number of cells considered in the whole merge process is bounded by $O(N \log N)$. The claim follows. □

Lemma 4. *The total number of outer merges the reevaluation of which has a non null impact on the data grid cost is bounded by $O(N)$.*

Proof. Let C be a non-empty cell let us focus on the cell pattern $\left|\begin{smallmatrix} * & * \\ \bullet & * \end{smallmatrix}\right|$, where C is in the lower left corner of the cell pattern. When a merge occurs, a new pattern is associated to the cell C. Let us focus on the chain of the cell patterns related to C during to whole merge process.

This chain can be described by 16 possible states (16 cell patterns) and $256 = 16^2$ possible transitions between the states. A close look at the cell patterns shows that the only outer merges which impact the reevaluation are related to the inner collision pattern and to the diagonal pattern. Concerning the inner collision states, the only possible transitions involve at least one cell merge. Concerning the diagonal state, transitions are possible only to inner collision states or to outer collision states, which contain either two cells in the same row or two cells in the same column. A new diagonal state can be encountered in the chain if and only if a cell merge happens.

The total number of states having an impact on the reevaluation is then bounded by the total number of cell merges in the merge process, that is by $O(N)$ according to proposition 1. This is true for each position of the cell in the cell pattern (lower left, lower right, upper left or upper right corner). The claim follows. □

Lemma 5. *The total number of cells considered in the outer merges during the whole execution of the GBUM algorithm is bounded by $O(N \log N)$.*

Proof. Lemma 3 states that the total number of reevaluated outer merges is bounded by $O(N \log N)$ and proposition 3 states that each reevaluation involves at most four cells. The claim follows. □

Proposition 4. *The overall time complexity of the maintenance operations of the Data Grid Merger structure during the whole execution of the GBUM algorithm is bounded by* $O(N \log N)$.

Proof. The maintenance operations for one merge are summarized in Algorithm 6. From lemmas 2 and 5, the cell cost variation $\delta c^{(C)}$ needs to be computed $O(N \log N)$ times. From lemmas 1 and 3, the part cost variation $\delta c^{(P)}$ needs to be computed $O(N \log N)$ times. From lemma 1 and 4, the number of parts merges which have an impact on the cost reevaluation is $\mathrm{O}(N)$. These merges need to be sorted again in the sorted list of part merges related to each input variable. Each sort maintenance can be performed in $\mathrm{O}(\log N)$, using a maintainable sorted list such as an AVL tree [AVL62]. The total time complexity for the maintenance of the sorted list of merges is then $\mathrm{O}(N \log N)$. The cost variation for the data grid $\delta c^{(G)}$ and for the input variables $\delta c^{(V)}$ needs to be reevaluated after each merge, that is at most $\mathrm{O}(N)$ times. Overall, the time complexity of all the maintenance operations of the Data Grid Merger is thus $\mathrm{O}(N \log N)$. □

4.5 Overall algorithmic complexity

We have shown in section 4.1 that the Data Grid structure can be initialized from the dataset in $\mathrm{O}(N \log N)$ time, and in section 4.2 that all the possible part merges can be evaluated and stored in the Data Grid Merger structure in $\mathrm{O}(N \log N)$ time. These structures need to be maintained during the greedy merge process. We have demonstrated in section 4.3 that all the maintenance operations run in $\mathrm{O}(N \log N)$ time for the Data Grid structure, and in section 4.4 that they run in $\mathrm{O}(N \log N)$ time for the Data Grid Merger structure. The memory complexity is $\mathrm{O}(N)$ both for the Data Grid structure and the Data Grid Merger structure.

Whereas a straightforward implementation of the GBUM algorithm requires an $\mathrm{O}(N^4)$ time complexity and an $\mathrm{O}(N^2)$ memory complexity, a carefully optimized implementation runs in $\mathrm{O}(N \log N)$ time with a $\mathrm{O}(N)$ memory requirement. This optimized implementation exploits both the additivity of the evaluation criterion and the sparseness of the data grids.

5 Post-optimization

The greedy heuristic is time efficient, but it may fall into a local optimum. First, the greedy heuristic may stop too soon and produce too many parts for each input variable. Second, the boundaries of the intervals may be sub-optimal since the merge decisions of the greedy heuristic are never rejected. We propose to reuse the post-optimization algorithms described in [Bou06] in the case of univariate discretization.

In a first stage called *exhaustive merge*, the greedy heuristic merge steps are performed without stopping condition until the data grid consists of one single cell. The best encountered data grid is then memorized. This stage allows escaping local minima with several successive merges and needs $\mathrm{O}(N \log N)$ time.

In a second stage called *greedy post-optimization*, a hill-climbing search is performed in the neighborhood of the best data grid. This search alternates the optimization on each input variable. For a given input variable, the univariate partition is frozen, and the other input variable is optimized using the univariate discretization post-optimization algorithm introduced in [Bou06]. This second stage converges very quickly in practice and requires only a few steps.

We summarize the post-optimization of data grids in Algorithm 7.

Algorithm 7 Post-optimization of a Data Grid

Require: M {Initial data grid solution}
Ensure: $M^*; c(M^*) \leq c(M)$ {Final solution with improved cost}
1: $M^* \leftarrow$ call *exhaustive merge* (M)
2: **while** improved **do**
3: {Univariate post-optimization of variable X_1}
4: freeze the univariate partition of variable X_2
5: $M^* \leftarrow$ call *univariate post-optimization* (M^*) for variable X_1
6: {Univariate post-optimization of variable X_2}
7: freeze the univariate partition of variable X_1
8: $M^* \leftarrow$ call *univariate post-optimization* (M^*) for variable X_2
9: **end while**

The univariate post-optimization exploits a neighborhood of a discretization consisting of combinations of interval splits and interval merges:

- a new interval can be added with one split,
- an interval boundary can be moved with one merge combined with one split,
- an interval can be removed with two merges combined with one split.

Owing to proposition 5, the univariate post-optimization algorithms can be reused directly.

Proposition 5. *Let $c(M)$ be an additive evaluation criterion of a data grid model M. Let $c_1(M_1)$ be the univariate evaluation criterion of the univariate partition model M_1 of variable X_1 when the partition of variable X_2 is frozen. Then $c_1(M_1)$ is an additive univariate evaluation criterion.*

Proof. From equation 3, we get

$$\begin{aligned} c(M) &= c^{(G)}(G) + c^{(V)}(X_1, I_1) + c^{(V)}(X_2, I_2) \\ &+ \sum_{i_1=1}^{I_1} c^{(P)}\left(P_{i_1}^{(1)}\right) + \sum_{i_2=1}^{I_2} c^{(P)}\left(P_{i_2}^{(2)}\right) + \sum_{i_1=1}^{I_1}\sum_{i_2=1}^{I_2} c^{(C)}(C_{i_1 i_2}) \end{aligned} \tag{7}$$

Let us freeze the partition of variable X_2. Thus, the number of parts I_2 and the parts $P_{i_2}^{(2)}$ become constants for the problem of optimizing the univariate

partition of X_1. The criterion $c_1(M_1)$ can be formulated as

$$c_1(M_1) = c_1^{(V)}(X_1, I_1) + \sum_{i_1=1}^{I_1} c_1^{(P)}\left(P_{i_1}^{(1)}\right) \tag{8}$$

with

$$c_1^{(V)}(X_1, I_1) = c^{(G)}(X_1X_2) + c^{(V)}(X_1, I_1) + c^{(V)}(X_2, I_2) + \sum_{i_2=1}^{I_2} c^{(P)}\left(P_{i_2}^{(2)}\right) \tag{9}$$

$$c_1^{(P)}\left(P_{i_1}^{(1)}\right) = c^{(P)}\left(P_{i_1}^{(1)}\right) + \sum_{i_2=1}^{I_2} c^{(C)}(C_{i_1i_2}). \tag{10}$$

The claim follows. □

In the univariate case, each evaluation of a part requires O(1) time, and the overall time complexity of the post-optimization algorithm is $O(N \log N)$ as shown in [Bou06]. In the bivariate case, both the variable criterion $c_1^{(V)}$ and the part criterion $c_1^{(P)}$ are more complex to evaluate. However, the sum term in the variable criterion is constant for a given partition of X_2 and can be evaluated once. The part criterion involves a sum of term for the cells of the current X_1 part related to each (frozen) part of X_2. Since the criterion is null on the empty cells, this sum can be evaluated only for non-empty cells. Thus, the overall evaluation of all the neighbor models of the current best bivariate model require $O(N)$ cell evaluations. Similarly to the main GBUM algorithm, the additivity of the criterion and the sparseness of the data grid can be exploited to perform the greedy post-optimization algorithm in $O(N \log N)$ time.

The post-optimization holds two straightforward notable properties. The first one is that post-optimizing can only improve the results of the GBUM algorithm since its comes after. The second one is that in case of input variables whose optimal joint partitioning consists of one single cell, the optimum data grid is necessarily found owing to the exhaustive merge stage of the post-optimization.

6 Meta-heuristic

The GBUM algorithm allows to evaluate $O(N^2)$ data grid models among $O(2^{2N})$ potential data grids in $O(N \log N)$ time. About $2N$ merges are performed along the algorithm but at most N merges involve a cell collision. This means that about half of the merges have no impact on the mixture of the output values in the cells, so that these merge decisions are blind with respect to the distribution of the data and may destroy interesting patterns. The post-optimization heuristic may bring a significant improvement, but it remains sticked to a close neighborhood of the best encountered solution.

Since the GBUM algorithm is time efficient, it is then natural to apply it repeatedly in order to better explore the search space. This is done according to

Algorithm 8 VNS meta-heuristic for data grid optimization

Require: M {Initial data grid solution}
Require: $MaxLevel$ {Optimization level}
Ensure: $M^*, c(M^* \leq c(M)$ {Final solution with improved cost}
1: $L \leftarrow 1$
2: **while** $L \leq MaxLevel$ **do**
3: {Generate a random solution in the neighborhood of M^*}
4: $M'' \leftarrow$ random solution with $(N/\log N)L/MaxLevel$ intervals per input variable
5: $M' \leftarrow M^* \cup M''$
6: {Optimize and evaluate the new solution}
7: $M' \leftarrow$ call Greedy Bottom-Up Merge(M')
8: $M' \leftarrow$ call Post-Optimization(M')
9: **if** $c(M') < c(M^*)$ **then**
10: $M^* \leftarrow M'$
11: $L \leftarrow 1$
12: **else**
13: $L \leftarrow L + 1$
14: **end if**
15: **end while**

the Variable Neighborhood Search (VNS) [HM01], which consists in applying the primary heuristic (GBUM algorithm and post-optimization) to a neighbor of the solution. If the new solution is not better, a bigger neighborhood is considered. Otherwise, the algorithm restarts with the new best solution and a minimal size neighborhood. The process is controlled by the maximum length of the series of growing neighborhoods to explore.

This meta-heuristic is described in Algorithm 8. According to the level of the neighborhood size l, a new solution M' is generated close to the current best solution. A random discretization for each input variable is obtained with the choice of random interval bounds without replacement. For $L = MaxLevel$, we bound the size of the random discretization by $N/\log N$.

The VNS meta-heuristic only requires the number of sizes of neighborhood as a parameter. This can easily be turned into an anytime optimization algorithm, by calling iteratively the VNS algorithm with parameters of increasing size and stopping the optimization only when the allocated time is elapsed. In Chapter 1, all the experiments are performed by calling the VNS algorithm three times with successive parameters equal to 1, 2 and 4.

In order to improve the initial solution, we choose to first optimize the univariate partition of each variable (discretization or value grouping) and to build the initial solution from a cross-product of the univariate partitions. Although this cannot help in case a strictly bivariate patterns (such as XOR for example), this might be helpful otherwise.

7 Adaptation to categorical variables

In marketing applications for example, variables such as Country, State, ZipCode, FirstName, ProductID usually hold many different values. Preprocessing these variables is critical to produce efficient classifiers.

In this section, we focus on categorical variables. We first analyze the balance between intensity of optimization and computation time, propose a practical trade-off, and study the impact of this trade-off on the optimization algorithms.

7.1 Optimization efficiency versus computation time

In the categorical case, $\mathrm{O}(V^2)$ part merges have to be evaluated at each step of the GBUM algorithm, instead of $\mathrm{O}(N)$ in the numerical case (intervals are constrained to be adjacent, not group of values). When $V \ll N$, this has little impact on the algorithmic time complexity. For large enough V, the optimization algorithms have to be adapted to keep a practical time complexity without sacrifying the quality of the solution.

For $V \geq \sqrt{N}$, the overall time complexity of the optimization algorithms may exceed $\mathrm{O}(N \log N)$ and even reach $\mathrm{O}(N^2 \log N)$ when $V = N$. However, the number of partitions is based on the Bell number in the case of value grouping instead of the binomial coefficient in the case of discretization. For regularized criteria such as those introduced in Chapter 1, the regularization term for the partition grows very quickly with the size of the partition, which penalizes large partitions. Furthermore, since no part can be isolated from the others (no adjacency constraints like for discretizations), the size I of the optimal partition is likely to be small w.r.t the number of instances.

In the following, we assume that $I \leq I_{Max} = \sqrt{N}$ and we restrict all the optimization algorithms to work with constrained variables, according to Definition 5. All the Propositions in section 7 relate implicitly to the case of constrained categorical variables.

Definition 5. *A categorical variable is constrained if the size of the partition of its values is bounded by* $I_{Max} = \sqrt{N}$.

According to Proposition 6, the time complexity of the GBUM algorithm is lower bounded by $\mathrm{O}(N\sqrt{N} \log N)$. However, we demonstrate in section 7.2 that this lower bound is also an upper bound of the time complexity of the algorithm.

Proposition 6. *The time complexity of the GBUM algorithm cannot be better than* $O(N\sqrt{N} \log N)$.

Proof. The principle of this proof is based on a counterexample, the evaluation of which in the GBUM algorithm requires at least $\mathrm{O}(N\sqrt{N} \log N)$ operations.

Let us consider a data grid with two categorical variables, each of which having $\sqrt{N}$ parts. The number of cells is N. Let us assume that all these cells are non-empty cells, which is possible provided that each cell contains exactly one instance.

When a merge is completed on one variable, both the source and target parts of the merge contain $\sqrt{N}$ cells, which all collide. Thus all the N outer merges need to reevaluated and all of them have a potentially non null impact on the merge cost. Thus, these N merges need to be sorted again in the sorted list of the outer variable, with an overall time complexity of $\mathrm{O}(N \log N)$.

The configuration of the merged part is the same as that of its source and target part, since it always contains $\sqrt{N}$ non-empty cells. Such merges may be performed up to $\sqrt{N}$ times if they are related to the same variable. The claim follows. □

7.2 Impact on the greedy bottom-up heuristic

In this section, we study the algorithmic complexity of the GBUM algorithm in the case of constrained categorical variables. We show in Proposition 7 that the memory complexity of the algorithm is still $\mathrm{O}(N)$ and in Propositions 8, 9 and 10 that its time complexity is $\mathrm{O}(N\sqrt{N}\log N)$.

It is noteworthy that the only algorithmic component with $\mathrm{O}(N\sqrt{N}\log N)$ time complexity is the maintenance of the sorted list of merges per variable, as illustrated in the counterexample given in Proposition 6. All the other components of the GBUM algorithm run in $\mathrm{O}(N\sqrt{N})$ time.

Proposition 7. *The memory complexity of the GBUM algorithm is $O(N)$.*

Proof. The memory complexity of the algorithm is that of the Data Grid and Data Grid Merger structures. The Data Grid structure contains two Variables, $\mathrm{O}(N)$ Parts and $\mathrm{O}(N)$ non-empty cells. The Data Grid Merger structure contains $\mathrm{O}(N)$ Merges, since the number of possible merges if $\mathrm{O}(N)$ both for numerical variables and for constrained categorical variables (${I_{Max}}^2 \leq N$). □

Proposition 8. *The time complexity of the initialization of the Data Grid and Data Grid Merger structures is $O(N \log N)$.*

Proof. The initialization of the Data Grid structure is performed according to Algorithm 2 which has a $\mathrm{O}(N \log N)$ time complexity.

Concerning the initialization of the Data Grid Merger structure, we have to evaluate the number of operations at the part level and cell level to initialize all the merges. Since each part and each non-empty cell is involved in $\mathrm{O}(I_{Max})$ merges, and since there are at most $\mathrm{O}(I_{Max})$ parts and $\mathrm{O}(N)$ non-empty cells, the claim follows. □

Proposition 9. *The time complexity of the maintenance operations of the Data Grid structure during the whole execution of the GBUM algorithm is $O(N\sqrt{N})$.*

Proof. Each merge involves at most two parts and $\mathrm{O}(N)$ cells. Since the number of merges necessary to go from the initial data grid to the terminal data grid is bounded by $\mathrm{O}(I_{Max})$, the claim follows. □

Proposition 10. *The time complexity of the maintenance operations of the Data Grid Merger structure during the whole execution of the GBUM algorithm is* $O(N\sqrt{N}\log N)$.

Proof. The time complexity of the maintenance of the Data Grid Merger structure is related to the number of merges that need to be reevaluated and their impact at the part and cell level. The case of numerical variables has been examined in section 4.4. In this proof, we focus on constrained categorical variables.

After each merge completion, $\mathrm{O}(\sqrt{N})$ inner merges and $\mathrm{O}(N)$ outer merges have to be reevaluated. The total number of reevaluated merges is bounded by $\mathrm{O}(N\sqrt{N})$. The variables need to maintain their sorted lists of merges, which requires $\mathrm{O}(\log N)$ operations per merge. Overall, the maintenance operations have a $\mathrm{O}(N\sqrt{N}\log N)$ time complexity at the merge level and part level (each merges involves two parts).

At the cell level, let us first focus on the total number of cells considered in the inner merges. For a given merge M, let P_0 be the merged part and $P_i, 1 \leq i \leq I$ all the other inner parts of the data grid. Let $N^{(C)}(P_i)$ be the number of cells in part i. Let us evaluate the total number of cells $N^{(C)}(M)$ considered in all the reevaluated inner merges. According to Algorithm 6, the cells are considered only when the cell number is less in the source part than in the target part of the reevaluated merge. We have

$$
\begin{aligned}
N^{(C)}(M) &\leq \sum_{i=1}^{I} \min(N^{(C)}(P_i), N^{(C)}(P_0)),\\
N^{(C)}(M) &\leq \sum_{i=1}^{I} N^{(C)}(P_i)),\\
N^{(C)}(M) &\leq N.
\end{aligned}
$$

Since at most N cells are considered after each merge completion and since there are at most $\mathrm{O}(\sqrt{N})$ merges in the GBUM algorithms, the total number of cells considered in the inner merges is bounded by $\mathrm{O}(N\sqrt{N})$.

Let us finally focus on the total number of cells considered in the outer merges. From Proposition 3, at most four cells need to be considered in the reevaluation of each outer merge. For each inner merge completion, $\mathrm{O}(N)$ outer merges between the $\mathrm{O}(\sqrt{N})$ outer parts need to be reevaluated. Since the GBUM algorithm involves at most $\mathrm{O}(\sqrt{N})$ inner merges, the total number of cells considered in the outer merges is bounded by $\mathrm{O}(N\sqrt{N})$.

The claim follows. □

7.3 Impact on the post-optimization and meta-heuristic

In the post-optimization algorithm, we keep the *exhaustive merge* algorithm, and use an univariate post-optimization algorithm for value grouping derived from [Bou05]. This heuristic consists in evaluating every move of a categorical value

from one group to another and performing the moves as soon as they improve the evaluation criterion. This heuristic is applied on a randomized sort of the categorical values and iterated as long as the criterion is improved. This post-optimization requires $O(VI)$ ($\leq O(N\sqrt{N})$) computation time per iteration and converges very quickly in case of optimized data grid, so that we choose not to bound the number of iterations.

In the meta-heuristic algorithm, we generate random solutions containing at most I_{Max} groups of values for the univariate partitions of categorical variables. Unfortunately, these random partitions are likely to be poor initial solutions for the GBUM heuristic, since each part is a random mixture of values. In order to improve these initial solutions, we pre-optimize them by moving the values across the groups, as described in the post-optimization heuristic. In this pre-optimization, we decide to bound the number of iterations (by two in practice) to both get sufficiently "pure" parts in the initial data grid and control the pre-optimization computation time.

7.4 Synthesis

Overall in the case of categorical variables, a compromise between the time complexity and the quality of the solution is necessary as soon as $V \geq \sqrt{N}$. We choose to constrain the partition of categorical variables to contain at most $O(\sqrt{N})$ parts and showed that even in this case, the computational complexity of the GBUM algorithm can reach $O(N\sqrt{N}\log N)$.

However, we demonstrated that the time complexity of the optimization heuristics is no more than $O(N\sqrt{N}\log N)$ in case of categorical variables with numerous values (beyond $\sqrt{N}$) and that their memory complexity is still $O(N)$. Compared to the numerical case, the same algorithms are used: greedy bottom-up heuristic, post-optimization and meta-heuristic. Another algorithmic component, pre-optimization, needs to be employed in order to "clean" the randomized data grids and feed the meta-heuristic with good initial solutions.

Overall, the time complexity of the optimization heuristics is $O(N\log N)$ in the case of two numerical variables and $O(N\sqrt{N}\log N)$ when one or two of the input variables are categorical.

8 Summary

Data grid models have been introduced in Chapter 1 to evaluate the correlation between a pair of input variables and an output variable. They rely on the partition of each input variable into a set of parts (intervals or groups of values). The cross-product of the partitions forms a data grid of cells, each of which allows to locally describe the distribution of the output values.

In this paper, we have introduced the concept of additive evaluation criteria for data grids, which can be decomposed hierarchically as a sum of terms at the data grid, variable, part and cell level.

We have studied the standard greedy top-down merge heuristic, whose straightforward implementation runs in $O(N^4)$ time. After introducing specific algorithmic structures to store a data grid model as well as all the possible merges between parts, we have shown that the time complexity of the heuristic can be reduced to $O(N \log N)$. The optimized heuristic mainly takes benefit of the additivity of the evaluation criterion and of the sparseness of the data grids which contain at most $O(N)$ non-empty cells for $O(N^2)$ cells.

In order to tackle the greediness of the heuristic, we have introduced post-optimization heuristics which exploit local neighborhoods around initial solutions, and a meta-heuristic to globally explore the search space according to neighborhoods of varying size.

Finally, the case of categorical input variables required specific adaptations to strike a balance between the time complexity and the quality of the optimization.

Overall, the data grid optimization algorithms run in $O(N \log N)$ for numerical input variables and in $O(N\sqrt{N} \log N)$ when categorical variables with numerous values (beyond $\sqrt{N}$) are involved. The memory requirement is always $O(N)$. Intensive experiments are reported in Chapter 1 to evaluate the optimization algorithms presented in this paper .

References

[AVL62] G. Adelson-Velskii and E.M. Landis. An algorithm for the organization of information. *Doklady Akademii Nauk SSSR, 146 263-266, 1962 (Russian)*, 3:1259–1263, 1962. English translation by Myron J. Ricci in Soviet Math. Doklady.

[BM96] C.L. Blake and C.J. Merz. UCI repository of machine learning databases, 1996. http://www.ics.uci.edu/mlearn/MLRepository.html.

[Bou04] M. Boullé. Khiops: a statistical discretization method of continuous attributes. *Machine Learning*, 55(1):53–69, 2004.

[Bou05] M. Boullé. A Bayes optimal approach for partitioning the values of categorical attributes. *Journal of Machine Learning Research*, 6:1431–1452, 2005.

[Bou06] M. Boullé. MODL: a Bayes optimal discretization method for continuous attributes. *Machine Learning*, 65(1):131–165, 2006.

[Cat91] J. Catlett. On changing continuous attributes into ordered discrete attributes. In *Proceedings of the European Working Session on Learning*, pages 87–102. Springer, 1991.

[DKS95] J. Dougherty, R. Kohavi, and M. Sahami. Supervised and unsupervised discretization of continuous features. In *Proceedings of the 12th International Conference on Machine Learning*, pages 194–202. Morgan Kaufmann, San Francisco, CA, 1995.

[ER96] T. Elomaa and J. Rousu. Finding optimal multi-splits for numerical attributes in decision tree learning. Technical Report NC-TR-96-041, Royal Holloway, University of London, 1996. NeuroCOLT.

[FI92] U. Fayyad and K. Irani. On the handling of continuous-valued attributes in decision tree generation. *Machine Learning*, 8:87–102, 1992.

[FKS95] T. Fulton, S. Kasif, and S. Salzberg. Efficient algorithms for finding multi-way splits for decision trees. In *Proceedings of the Twelfth International Conference on Machine Learning*, pages 244–251. Morgan Kaufmann, 1995.

[HM01] P. Hansen and N. Mladenovic. Variable neighborhood search: principles and applications. *European Journal of Operational Research*, 130:449–467, 2001.

[Hol93] R.C. Holte. Very simple classification rules perform well on most commonly used datasets. *Machine Learning*, 11:63–90, 1993.

[Kas80] G.V. Kass. An exploratory technique for investigating large quantities of categorical data. *Applied Statistics*, 29(2):119–127, 1980.

[KBR84] I. Kononenko, I. Bratko, and E. Roskar. Experiments in automatic learning of medical diagnostic rules. Technical report, Ljubljana: Joseph Stefan Institute, Faculty of Electrical Engineering and Computer Science, 1984.

[Ker91] R. Kerber. Chimerge discretization of numeric attributes. In *Proceedings of the 10th International Conference on Artificial Intelligence*, pages 123–128. AAAI Press, 1991.

[LHTD02] H. Liu, F. Hussain, C.L. Tan, and M. Dash. Discretization: An enabling technique. *Data Mining and Knowledge Discovery*, 4(6):393–423, 2002.

[Qui93] J.R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1993.

[ZR00] D.A. Zighed and R. Rakotomalala. *Graphes d'induction*. Hermes, France, 2000.

[ZRR98] D.A. Zighed, S. Rabaseda, and R. Rakotomalala. Fusinter: a method for discretization of continuous attributes for supervised learning. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 6(33):307–326, 1998.