entropy

MDPI

Article

Mixture-Based Probabilistic Graphical Models for the Label Ranking Problem †

Enrique G. Rodrigo [1,2], Juan C. Alfaro [1,2,*], Juan A. Aledo [2,3] and José A. Gámez [1,2]

1 Departamento de Sistemas Informáticos, Universidad de Castilla-La Mancha, 02071 Albacete, Spain; mail@enriquegrodrigo.com (E.G.R.); Jose.Gamez@uclm.es (J.A.G.)
2 Laboratorio de Sistemas Inteligentes y Minería de Datos, Instituto de Investigación en Informática de Albacete, 02071 Albacete, Spain; JuanAngel.Aledo@uclm.es
3 Departamento de Matemáticas, Universidad de Castilla-La Mancha, 02071 Albacete, Spain
* Correspondence: JuanCarlos.Alfaro@uclm.es
† This is an extended version in proceedings of the 15th European Conference on Symbolic and Quantitative Approaches with Uncertainty, Belgrade, Serbia, 18–20 September 2019.

Abstract: The goal of the *Label Ranking (LR) problem* is to learn *preference models* that predict the preferred ranking of class labels for a given unlabeled instance. Different well-known machine learning algorithms have been adapted to deal with the LR problem. In particular, fine-tuned instance-based algorithms (e.g., k-nearest neighbors) and model-based algorithms (e.g., decision trees) have performed remarkably well in tackling the LR problem. *Probabilistic Graphical Models* (*PGMs*, e.g., *Bayesian networks*) have not been considered to deal with this problem because of the difficulty of modeling permutations in that framework. In this paper, we propose a *Hidden Naive Bayes classifier* (*HNB*) to cope with the LR problem. By introducing a hidden variable, we can design a hybrid Bayesian network in which several types of distributions can be combined: multinomial for discrete variables, Gaussian for numerical variables, and *Mallows* for permutations. We consider two kinds of probabilistic models: one based on a *Naive Bayes* graphical structure (where only univariate probability distributions are estimated for each state of the hidden variable) and another where we allow interactions among the predictive attributes (using a multivariate Gaussian distribution for the parameter estimation). The experimental evaluation shows that our proposals are competitive with the start-of-the-art algorithms in both accuracy and in CPU time requirements.

Keywords: mixture models; EM algorithm; Naive Bayes; probabilistic graphical models; label ranking; preference learning; machine learning

Citation: Rodrigo, E.G.; Alfaro, J.C.; Aledo, J.A.; Gámez, J.A. Mixture-Based Probabilistic Graphical Models for the Label Ranking Problem. *Entropy* **2021**, *23*, 420. https://doi.org/10.3390/e23040420

Academic Editors: Rafael Rumí and Antonio Salmerón

Received: 9 March 2021
Accepted: 27 March 2021
Published: 31 March 2021

1. Introduction

Preferences are comparative judgments about a set of alternatives or choices. The *Label Ranking (LR) problem* [1–3] is a well-known non-standard supervised classification problem [4,5], whose goal is to learn *preference models* that predict the preferred ranking over a set of class labels for a given unlabeled instance. Practical applications of the LR problem are found in cases where an order of preference (or ranking) for the class labels is required given an input instance. Particular examples can be ranking a set of genes from their expression level, ranking the set of relevant topics for a given document, ranking a set of available machine learning algorithms for a given dataset and prediction task, etc. [6,7].

Formally, we consider a problem domain defined over n *predictive variables* (also known as *attributes*), $X_1, \ldots, X_n$, and a *class variable* C with m labels, $dom(C) = \{c_1, \ldots, c_m\}$. We are interested in predicting the ranking π of the labels for an unlabeled instance $e_t = (x_{1,t}, \ldots, x_{n,t}) \in dom(X_1) \times \cdots \times dom(X_n)$ given a dataset $\mathbf{D} = \{(x_{1,j}, \ldots, x_{n,j}, \pi_j)\}_{j=1}^{N}$ with N labeled instances. Therefore, the LR problem consists in learning a LR-Classifier $\mathcal{C}$ from $\mathbf{D}$ which generalizes well on unseen data.

In other words, the goal of the LR problem is to induce a model able to predict a permutation of the class labels by taking advantage of all the available information during the learning process. Different approaches have been proposed to tackle this problem:

- *Transformation methods*. They transform the ranking-based prediction problem into a set of single-class classifiers, whose outcomes must be later aggregated in order to obtain a ranking. Various approaches have been considered, such as *labelwise* [8] and *pairwise* approaches [9,10], *chain classifiers* [11], etc.
- *Adaptation methods*. They adapt well-known machine learning algorithms to cope with the new class structure. Cheng et al. in [2] introduced a *model-based algorithm* that induces a decision tree (*Label Ranking Trees (LRT)*) and a *model-free algorithm* which uses *k*-nearest neighbors (*Instance-Based Label Ranking (IBLR)*). Other techniques, like association rules [12] or neural networks [13], have also been adapted.
- *Ensemble methods*. Recently, different tree-based aggregation approaches, such as *Random Forests*, *Bagging predictors*, and *Boosting methods*, have been successfully applied to the LR problem [14–17].

In this paper, we propose a new model-based LR-classifier which belongs to the adaptation methods family. Our motivation is twofold:

- Although fine-tuned instance-based algorithms have exhibited remarkable performance (especially when the model is trained with *complete rankings*), they may demand a great amount of computational resources (memory and time) during model selection and inference when the size of the dataset grows.
- Although *Probabilistic Graphical Models* (*PGMs*; e.g., *Bayesian networks*) [18,19] constitute a standard approach in machine learning, they have not been used in this problem because of the difficulty in coping with permutations in this framework [2,20]. In this work, we successfully introduce the use of PGMs to deal with the LR problem, obtaining results which are competitive with the state-of-the-art IBLR and LRT algorithms.

The proposed probabilistic LR-classifier relies on the use of a *hybrid Bayesian network* [21] where different probability distributions are used to conveniently model variables of a different nature: Multinomial for discrete variables, Gaussian for numerical variables, and *Mallows* for permutations [22]. The Mallows probability distribution is usually considered to model a set of permutations and, in fact, is the core of the decision tree algorithm (LRT) proposed in [2].

To overcome the constraints regarding the topology of the network when dealing with different types of variables, in the preliminary version of this study, we proposed a mixture-based structure where the root is a *hidden discrete variable*. In [23], we based our proposal on a *Naive Bayes* graphical structure, where only univariate probability distributions are estimated for each state of the hidden variable. Learning and inference schemes were also designed in [23], based on the use of well-known *Expectation-Maximization* (EM) algorithm for parameter estimation and a combination of probabilistic inference with the *Kemeny Ranking Problem* (*KRP*) [24], respectively. Nonetheless, the proposed methods performed somewhat unevenly when dealing with the different datasets. With this more comprehensive paper, we successfully overcome the main weaknesses of our former proposal. Specifically, the main contributions of this study are as follows:

- After identifying early stopping as the main problem in our previous learning algorithm (Method A), we propose a new learning scheme (see Method B in Section 3) to search for the number of components in the mixture.
- For our Hidden Naive Bayes model, we explore discretization as an alternative to modeling numerical variables as Gaussian distributions.
- We extend the complexity of the naive Bayes-based structure model in order to allow interactions among the predictive attributes. In this new model, only numerical predictive attributes are allowed, and interactions are managed by using a multivariate Gaussian distribution.

- We perform an exhaustive experimental analysis over the standard benchmark for the label ranking problem.

The rest of the paper is structured as follows. In Section 2, we review some basic notions needed to deal with rank data. In Section 3, we formally describe the proposed *Hidden Naive Bayes* (*HNB*) as well as the algorithms to induce it from data and to carry out inference. Then, in Section 4, we extend our proposal to allow interactions between the (numerical) predictive attributes, by using a multivariate Gaussian mixture. In Section 5, we set out the empirical study conducted to evaluate the methods designed in this paper. In Section 6, we briefly comment on some limitations of the presented approach. Finally, in Section 7, we provide the conclusions and future research lines.

2. Background

In this section, we review the background to our proposal. In particular, we briefly describe some permutation-based notions, such as the *Kemeny Ranking Problem* [24] and the *Mallows probability distribution* [22]. We also revise the *Naive Bayes* model [18] and the two competing methods to tackle the LR problem used in this study: the *Label Ranking Trees* and the *Instance-Based Label Ranking* algorithms [2].

2.1. Kemeny Ranking Problem

Let $\mathbb{S}_m$ be the set of permutations defined over m elements $\{1, \ldots, m\}$. The *Kemeny Ranking Problem* (*KRP*) [24] consists in obtaining the *consensus permutation* (*mode*) $\pi_0 \in \mathbb{S}_m$ that best represents a sample with N permutations $\Pi = \{\pi_1, \ldots, \pi_N\}$, $\pi_i \in \mathbb{S}_m$.

Formally, the KRP looks for the consensus permutation $\pi_0 \in \mathbb{S}_m$ that minimizes

$$\pi_0 = \underset{\pi_i \in \mathbb{S}_m}{argmin} \sum_{i=1}^{N} D(\pi_0, \pi_i),$$

where $D(\pi, \tau)$, $\pi, \tau \in \mathbb{S}_m$ is a distance measure between two permutations π and τ. Normally, the *Kendall distance* [25] is used, which counts the number of pairwise disagreements between the two permutations, and the (greedy) *Borda count* algorithm [26] is employed to solve the KRP, because of its trade-off between efficiency and accuracy. The Borda count algorithm basically assigns m points to the item ranked first, $m-1$ to the second one, and so on. Once all the input rankings have been computed, the items are sorted according to the number of accumulated points.

When not all rankings are equally important, a weight can be associated with each one to reflect its relevance. Then, a generalized version of the Borda method called *weighted Borda count* is used, which basically balances the points received by a permutation taking its weight into account.

2.2. Kendall Rank Correlation Coefficient

In our learning process (see Section 3.3), the *Kendall rank correlation coefficient* τ_K is used as goodness score [27]. Given the class variable C with $dom(C) = \{c_1, \ldots, c_m\}$ and permutations π_1, π_2 of the values in $dom(C)$, the τ_K Kendall rank correlation coefficient is given by

$$\tau_K(\pi_1, \pi_2) = \frac{\sum_{i=1}^{m} \sum_{j=1}^{m} \beta_1^{ij} \cdot \beta_2^{ij}}{m \cdot (m-1)}$$

where

$$\beta_k^{ij} = \begin{cases} 1, & \text{if } c_i \succ_{\pi_k} c_j \\ -1, & \text{if } c_j \succ_{\pi_k} c_i \\ 0, & \text{if } i = j \end{cases}$$

for $k = 1, 2$. Here, $c_i \succ_{\pi_k} c_j$ means that c_i is ranked before c_j in π_k.

The τ_K Kendall rank correlation coefficient lies in the range $[-1, 1]$. In particular, $\tau_K(\pi_1, \pi_2) = 1$ means a total positive correlation between π_1 and π_2 ($\pi_1 = \pi_2$), whereas $\tau_K(\pi_1, \pi_2) = -1$ indicates a total negative correlation (actually this occurs when π_1 is the inverse of π_2). Values of τ_K close to 0 mean a poor correlation between the permutations.

2.3. Mallows Probability Distribution

The *Mallows probability distribution* (also known as the *Mallows model*) [22] is an exponential probability distribution over permutations based on distances. The Mallows model, $\mathcal{M}(\pi_0, \theta)$, is parametrized by two parameters: the *central permutation* (*mode*) $\pi_0 \in \mathbb{S}_m$ and the *spread parameter* (*dispersion*) $\theta \in [0, +\infty)$. Given a distance D in $\mathbb{S}_m$, the probability assigned to a permutation $\pi \in \mathbb{S}_m$ by the Mallows distribution $\mathcal{M}(\pi_0, \theta)$ is

$$P(\pi; \pi_0, \theta) = \frac{e^{-\theta \cdot D(\pi, \pi_0)}}{\Psi(\theta)}$$

where $\Psi(\theta)$ is a normalization constant. The spread parameter θ quantifies the concentration of the distribution around π_0. For $\theta = 0$, a uniform distribution is obtained, while for $\theta = +\infty$ the model assigns a probability of 1 to π_0 and of 0 to the rest of the permutations. Both π_0 and θ can be estimated accurately in polynomial time [28]. For consensus permutation (π_0), the Borda count is usually employed. For the spread (θ), there is no closed form, so numerical algorithms (e.g., Newton–Raphson) are normally used.

In this study, we take the Kendall distance as D, which is the usual choice in the literature [2,29].

2.4. Naive Bayes

Naive Bayes (*NB*) models are well-known *probabilistic classifiers* based on the strong independence hypothesis that, given the class variable, every pair of features is considered conditionally independent [30]. This assumption allows an efficient factorization of the join probability distribution (see Equation (1)) as well as efficient learning and inference procedures. Figure 1 shows the graphical structure of an NB model.

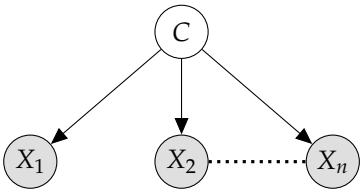

Figure 1. Naive Bayes model structure.

Like most probabilistic classifiers, NB models follow the *maximum a posteriori* (*MAP*) principle, that is, they return the most probable class label given the input instance as evidence. Formally, given an input instance $e_t = (x_{1,t}, \ldots, x_{n,t}) \in dom(X_1) \times \cdots \times dom(X_n)$ and being C the class variable with $dom(C) = \{c_1, \ldots, c_m\}$, a *Naive Bayes Classifier* $\mathcal{C}$ returns

$$\mathcal{C}(e_t) = \underset{c \in dom(C)}{argmax} P(c \,|e_t) = \underset{c \in dom(C)}{argmax} P(e_t, c) = \underset{c \in dom(C)}{argmax} \prod_{i=1}^{n} P(x_{i,t}|c) \cdot P(c) \tag{1}$$

according to *Bayes' theorem* and the conditional independence hypothesis, respectively. The above conditional distributions may be multinomial for discrete attributes and Gaussian for continuous attributes.

2.5. Instance-Based Label Ranking

The *Instance-Based Label Ranking* (*IBLR*) algorithm [2] is based on the nearest neighbors estimation principle. It takes, as input, an instance e_t to be classified, a training dataset $\mathbf{D}$ with N labeled instances and the number of nearest neighbors $k \in \mathbb{N}^+$, $k \leq N$, to

be considered. Using an appropriate distance, the IBLR algorithm then compares the input instance with all the N training ones, obtains the k nearest neighbors from $\mathbf{D}$, $\mathbf{R} = \{(x_{1,j}, \ldots, x_{n,j}, \pi_j)\}_{j=1}^{k}$, and takes the rankings associated with these instances, $\mathbf{R}_\Pi = \{\pi_j\}_{j=1}^{k}$. Then, the IBLR algorithm applies the Borda count algorithm to the permutations in $\mathbf{R}_\Pi$ and the obtained consensus permutation π_0 is returned as output.

The main advantage of instance-based learning is its local behavior, which allows it to locally estimate a different target function for each new instance to be classified instead of estimating a single target function for the entire instance space. On the other hand, its main disadvantage is its high computational cost in the inference stage, as it must compare the input instance against all the instances in the training dataset.

2.6. Label Ranking Trees

Decision trees are usually constructed by recursively partitioning the dataset. The *Label Ranking Trees* (*LRT*) algorithm [2] receives, at each call, a set of instances $\mathbf{R} = \{(x_{1,j}, \ldots, x_{k,j}, \pi_j)\}_{j=1}^{s}$ with $1 \leq k \leq n$ and $2 \leq s \leq N$, and must decide whether to stop the recursive call by creating a leaf node, or go on with the branching process by splitting the received training dataset $\mathbf{R}$ into several subsets according to the value of an attribute X_i.

The stopping and splitting criteria used in LRT are as follows:

- *Stopping criterion*. If we consider $\mathbf{R}_\Pi = \{\pi_j\}_{j=1}^{s}$ as the rankings associated with the instances in $\mathbf{R}$, the LRT algorithm stops the splitting process and creates a leaf node if either of the following two conditions hold:
  - All the rankings are consistent. For all the pairs of class labels $c_u, c_v \in dom(C)$, they maintain the same preference relation ($c_u \succ c_v$ or $c_v \succ c_u$) through all the rankings in $\mathbf{R}_\Pi$ which rank both c_u and c_v.
  - $s < 2m$. This condition is introduced as a pre-pruning operation to prevent overfitting.

  The leaf created is labeled with the consensus ranking π_0 obtained by applying the Borda count algorithm over the rankings in $\mathbf{R}_\Pi$.
- *Splitting criterion*. The LRT algorithm uses the spread parameter θ of the Mallows model (see Section 2.3) to measure the scattering of the rankings associated with a partition with respect to the consensus one. Formally, given an attribute X_i with domain $dom(X_i) = \{x_1, \ldots, x_{r_i}\}$, the uncertainty associated with a partition $\{R_1, R_2, \ldots, R_{r_i}\}$ of $\mathbf{R}$ is inversely proportional to

$$f(X_i) = f(R_1, \ldots, R_{r_i}\}) = \sum_{j=1}^{r_i} \frac{|R_j| \cdot \theta_j}{|\mathbf{R}|}, \tag{2}$$

  where θ_j is the spread parameter estimated from the rankings of the instances in R_j, which can be computed by means of standard numerical optimization methods [2,29]. The LRT algorithm proceeds in a standard way, that is, sorting the values of the attributes X_i in $\mathbf{R}$ and analyzing all the possible thresholds λ. Thus, it deals with the resulting two-state discrete attribute X_i^λ with domain $dom(X_i^\lambda) = \{X_i \leq \lambda, X_i > \lambda\}$, and selects the threshold λ of the attribute X_i that maximizes (2).

Then, an instance is classified by following the path from the root to the corresponding leaf, selecting, at each decision node, the branch corresponding to the value of the attribute in the instance to be classified. Thus, once a leaf node is reached, the permutation assigned to the leaf node is returned.

3. Hidden Naive Bayes for Label Ranking

In this section, we propose an NB-based model to deal with the LR problem. We start by defining the proposed PGM structure and then describe the parameter estimation process and two different methods for training the model.

3.1. Model Definition

To overcome the constraints regarding the topology of the network when dealing with different types of variables, the model proposed here combines an NB structure with a hidden (latent) variable.

This idea is not new, and has been used, for instance, for unsupervised clustering [21,31], to improve the performance (accuracy) of the base classifier [32], relax some of the independence statements increasing the classifier modeling capability [33–35], or obtain models for efficient probabilistic inference [36].

In this paper, the introduction of the hidden variable stems from the need to model the join probability distribution involving variables of a different nature: discrete, continuous, and permutation-based.

In the proposed NB model graphical structure, the root element of the model is a discrete hidden variable, which we will denote as H, with $dom(H) = \{h_1, \ldots, h_{r_H}\}, r_H$ being the total number of mixture models. The rest of the variables are observed variables. We consider two types of observed variables:

- The *feature variables*, observed both in the training and in the test phase. We consider two kinds: discrete variables, denoted as X_j, $j = 1, \ldots, n_J$ and $dom(X_j) = \{x_{j_1}, \ldots, x_{j_{r_j}}\}$, and continuous variables, denoted as Y_k, $k = 1, \ldots, n_K$.
- The *ranking variable*, denoted as π, which takes values in $\mathbb{S}_m$, this being the set of permutations defined over the class labels $\{c_1, \ldots, c_m\}$. This variable is used only during the training stage and is the target to be inferred.

Figure 2 shows a plate-based representation of the proposed model with the different types of variables described above ($n = n_J + n_K$). The model assumes that each of these variable types follows a different conditional distribution given the root variable:

- Discrete variables follow a Multinomial distribution,

$$P(X_j|H) \rightsquigarrow P(X_j|H = h_z) \sim Mult(\{p(x_{j_i}^{h_z})\}_{i=1}^{r_j}), \;\; z = 1, \ldots, r_H$$

- Continuous variables follow a Normal or univariate Gaussian distribution,

$$P(Y_k|H) \rightsquigarrow P(Y_k|H = h_z) \sim \mathcal{N}(\mu_k^{h_z}, \sigma_k^{h_z}), \;\; z = 1, \ldots, r_H$$

- The ranking variable follows a Mallows distribution,

$$P(\pi|H) \rightsquigarrow P(\pi|H = h_z) \sim \mathcal{M}(\pi_0^{h_z}, \theta^{h_z}), \;\; z = 1, \ldots, r_H$$

- The hidden variable follows a Multinomial distribution,

$$P(H) \sim Mult(\{p(h_z)\}_{z=1}^{r_H})$$

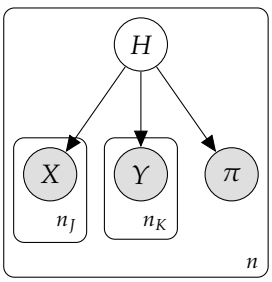

Figure 2. The proposed HNB model.

The parameters for each of the conditional distributions need to be estimated to perform inference using the model.

3.2. Parameter Estimation

As is common in most machine learning papers, we assume *i.i.d.* data. Furthermore, we also assume *complete* data, i.e., without missing values, both in the predictive and in the ranking variables. If there are missing values in the training data, they must be imputed previously to learn the model. The ranking variable can be imputed as described in [2]. Thus, we only deal with a hidden variable, H, and base our approach on the use of the *Expectation-Maximization* (*EM*) algorithm to estimate jointly the parameters of both the observed and hidden variables.

The EM algorithm [37] consists of two steps: the *expectation* step (*E* step), where the values for the hidden variable are estimated, and the *maximization* step (*M* step), where the parameters for the conditional distributions are obtained. Below, we describe these steps:

- **E step**: Under the assumption that the parameters of the model $\{\mu_k^{h_z}, \sigma_k^{h_z}, p(x_{j_i}^{h_z}), \pi_0^{h_z}, \theta^{h_z}, p(h_z)\}$ $k = 1, \ldots, n_K$, $j = 1, \ldots, n_J$, $i = 1, \ldots, r_j$, $z = 1, \ldots, r_H$, are known, the probability of an example $e_t = (x_{1,t}, \ldots, x_{n_J,t}, y_{1,t}, \ldots, y_{n_K,t}, \pi_t)$ being in a mixture h_z is

$$\begin{aligned} P(h_z|x_{1,t}, \ldots, x_{n_J,t}, y_{1,t}, \ldots, y_{n_K,t}, \pi_t) &= \frac{1}{C} \cdot P(h_z) \cdot P(x_{1,t}, \ldots, x_{n_J,t}, y_{1,t}, \ldots, y_{n_K,t}, \pi_t|h_z) \\ &= \frac{1}{C} \cdot P(h_z) \cdot P(\pi_t|h_z) \cdot \prod_{j=1}^{n_J} P(x_{j,t}|h_z) \cdot \prod_{k=1}^{n_K} P(y_{k,t}|h_z) \\ &= \frac{1}{C} p(h_z) \cdot \frac{e^{-\theta^{h_z} \cdot D(\pi_t, \pi_0^{h_z})}}{\Psi(\theta^{h_z})} \cdot \prod_{j=1}^{n_J} p(x_{j,t}^{h_z}) \cdot \prod_{k=1}^{n_K} \frac{1}{\sigma_k^{h_z}\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{y_{k,t}-\mu_k^{h_z}}{\sigma}\right)^2} \end{aligned} \tag{3}$$

  Here, $C = \sum_{z=1}^{r_H} P(h_z) \cdot P(x_{1,t}, \ldots, x_{n_J,t}, y_{1,t}, \ldots, y_{n_K,t}, \pi_t|h_z)$ is the normalization constant.
- **M step**: Under the assumption that the probabilities of belonging to each mixture for all examples are known, the parameters of the model can be estimated as follows:
  - Multinomial parameters for the discrete variables. Each multinomial parameter $p(x_{j_i}^{h_z})$ is estimated by using MLE, where the count for each instance is weighted by the probability of $H = h_z$ given the instance.
  - Gaussian distribution parameters for the continuous variables. The parameters $\mu_k^{h_z}$ and $\sigma_k^{h_z}$ of the Gaussian distribution are estimated through MLE for each $H = hz$, weighting each instance by the probability of it being in the mixture.
  - Mallows distribution parameters for the ranking variable. For each component of the mixture h_z of H, a Mallows distribution $\mathcal{M}(\pi_0^{h_z}, \theta^{h_z})$ must be estimated (see Section 2.3). In particular, $\pi_0^{h_z}$ is computed by applying a weighted version of the Borda count algorithm (points assigned to items are weighted by the probability of $H = h_z$ given the instance), and θ^{h_z} is calculated by using the Newton–Raphson numerical optimization method.
  - The mixture model probabilities $P(H)$ are computed according to the weights $P(h_z|e_t)$ for each mixture h_z of H (see Equation (3)) by means of MLE.

Stopping condition: The EM algorithm can easily accommodate different types of stop conditions, most of them based on checking the convergence of some score (logarithm likelihood, accuracy, etc.).

3.3. Learning Process

In addition to the graphical structure and the parameter estimation already described, we also need to determine some kind of *structural* learning in order to find the inner structure of H, that is, its cardinality or number of states.

Basically, we follow a greedy technique by initializing r_H to a certain number and then running consecutive executions of the EM algorithm with an increasing number of mixtures.

There are several points to discuss regarding the learning process: the initial value for r_H, the value used to increment r_H between two consecutive iterations, the way the components of the mixture are initialized, and how the goodness of the model is evaluated and the final value of r_H is selected. Below, we describe the two proposed schemes.

3.3.1. Method A: HNBE-LR

First, we describe the scheme proposed in the preliminary (conference) version of this study [23], based on the learning process used in [36] and which basically wraps the EM method for parameter estimation. In Algorithm 1, we show our adaptation from the NB estimation algorithm [31] to the LR problem. The main characteristics of this method are as follows:

- It is a wrapper method. Thus, the data received is divided into training Tr and validation Tv datasets, using the *Kendall coefficient* τ_K to assess the models and parameterizations explored during the search.
- The search for the number of components of the mixture is carried out greedily. We start with an initial number of components z_0 and a new component is added at each iteration. However, low probability components are pruned both during the search and during the EM-based parameter estimation. The search stops when the obtained model does not improve the best one in a given number of consecutive iterations. The improvement is assessed by evaluating the current parameterized model over the training dataset Tr.
- Every time a new component is added to the mixture, the model parameters corresponding to this component are initialized from a set of instances of Tr obtained by using sampling with replacement.
- For each number of components in the mixture r_H tried, an EM is run for parameter estimation. After each iteration (E and M steps), the model is evaluated using the Kendall coefficient τ_K^{Tr} for the training data Tr. A threshold on the difference between this value and the previous one is used to check convergence.
- When the number of components changes (either because of pruning low probability ones or because of the addition of a new one), the component weights are properly rescaled.

3.3.2. Method B: HNB-LR

The results obtained in [23] shed light on certain drawbacks. The main one is that the algorithm reaches the stopping condition too soon, which results in a small number of components for the mixture. As the authors in [36] noted, in contrast to clustering (e.g., AutoClass [31]), a high number of mixture components is required to obtain an accurate approximation of the joint probability estimation.

Bearing this in mind, and also that the number of different values tried for r_H must be small for reasons of efficiency, we propose an alternative scheme that we call *Method B*. As in Method A, the main idea is to wrap the EM algorithm by a search procedure to look for the number of components to be included in the mixture. In order to do that, we introduce important design modifications. Algorithms 2 and 3 show the scheme of this approach. Below, we comment on their main characteristics and differences with respect to Method A.

- In Method B, low probability components are not pruned, and so the EM algorithm is carried out in the search process (see Algorithm 2). Furthermore, the convergence of the EM algorithm is checked by using the log-likelihood (LL) of the data (Tr) given the model, that is, no wrapper evaluation is carried out to compute τ_K inside EM.
- The search process works in a wrapper style. Thus, we divide the data received into a training Tr and validation Tv datasets, and use τ_K to assess the models explored during the search.

Algorithm 1: Method A: HNBE-LR

Data: T: Training dataset; z_0: Initial #of components; β: Maximum #of iterations allowed without improvement; α: Minimum required improvement for each EM iteration; p: #of cycles to prune low-weight components of the model.
Result: The HNBE-LR model fully specified.

Create a holdout partition $\{Tr, Tv\}$ from T; $Tr = \{e_t\}_{t=1}^{|Tr|}$;
$r_H \leftarrow z_0$;
Initialize the model with r_H mixture components and compute all the model parameters from a sample with replacement of Tr;
$It \leftarrow 0$;
$\tau_K^{Tr} \leftarrow -1$;
$\tau_K^{Tv} \leftarrow -1$;
while $(It < \beta)$ **do**
    `// EM - Parameter estimation`
    improving $\leftarrow$ *true*;
    $\tau_K^c \leftarrow -1$;
    while *improving* **do**
        `// E step`
        Update $P(h_z|e_t)$, $z = 1, \ldots, r_H$ and $t = 1, \ldots, |Tr|$ using Equation (3);
        `// M step`
        Compute model parameters as described in Section 3.2 (M-step);
        Every p cycles, prune low-weight components of the model and update r_H;
        $\tau_K \leftarrow$ Evaluate the model over Tr dataset;
        if $(\tau_K - \tau_K^c) < \alpha$ **then**
            improving $\leftarrow$ False;
        else
            $\tau_K^c \leftarrow \tau_K$;
        end
    end
    Prune low-weight components of the model and update r_H;
    $\tau_K \leftarrow$ Evaluate the model over Tv dataset;
    if $(\tau_K > \tau_K^{Tv})$ **then**
        best $\leftarrow$ model;
        $\tau_K^{Tv} \leftarrow \tau_K$;
    end
    $\tau_K^{Tr} \leftarrow$ Evaluate the model over Tr dataset;
    if $(\tau_K > \tau_K^{Tr})$ **then**
        $\tau_K^{Tr} \leftarrow \tau_K$;
        $It \leftarrow 0$;
    else
        $It \leftarrow It + 1$;
    end
    `// Adding a new mixture component`
    Re-scale mixture component weights by a factor $\frac{r_H}{r_H+1}$;
    Add a new mixture component to the model with weight $\frac{1}{r_H+1}$ and compute its parameters from a sample with replacement of Tr;
end
return *best*;

- The search for the number of components is carried out greedily, but we now split it into two phases. The first is a forward search, where we evaluate the model with $r_H = 2^1, 2^2, 2^4, \ldots, 2^{10}$. We then select the best value r'_H according to τ_K^{Tv} and run a binary search between $\frac{r'_H}{2}$ and r'_H. Finally, the best value r^*_H found in the binary

search (see Section 5.2) according to τ_K^{Tv} is returned as the number of components for our model.
The intuition behind this greedy search is to be efficient (at most, 20 values are tested) and to quickly try large values for r_H, as we identified this point as a shortcoming of Method A.

- Each time a new value for r_H is tried, the process starts from scratch, that is, all the components are initialized simultaneously, instead of being added to the model as in Algorithm 1. To initialize the component parameters (probabilities and weights), k-means clustering processes [38] with $k = r_H$ are run, and the better one according to the minimal sum of distances between points and clusters centroids is selected. The instances associated with each cluster are used to initialize the corresponding mixture component.

Algorithm 2: Method: EM

Data: $Tr = \{e_t\}_{t=1}^{|Tr|}$: Training data set; r_H: #of mixture components; β: Maximum #of iterations; α: Minimum required improvement for each EM iteration; γ: #of k-means clustering algorithm restarts.
Result: The HNB-LR model fully specified.

```
Initialize the model with r_H mixture components by using the best of γ restarts of the k-means clustering algorithm with k = r_H;
LL_c ← −∞;
for i ← 0 to β do
    // E step
    Update P(h_z|e_t), z = 1,...,r_H and t = 1,...,|Tr| using Equation (3);
    // M step
    Compute model parameters as described in Section 3.2 (M-step);
    Compute LL for Tr given the current model;
    if (LL − LL_c < α) then
        // Early stopping because of convergence
        break;
    end
end
return model;
```

3.4. Inference Process

In the inference process, the method needs to predict the ranking associated with a given instance e_t. In our proposal, the probability of a ranking π_s given e_t can be obtained by marginalizing out variables until we obtain an expression for the posterior probability

$$P(\pi_s|e_t) \propto \sum_{z=1}^{r_H} \left(P(h_z) \cdot P(\pi_s|h_z) \cdot \prod_{j=1}^{n_J} P(x_{j,t}|h_z) \cdot \prod_{k=1}^{n_K} P(y_{k,t}|h_z) \right)$$

The outcome can then be obtained by using the MAP principle, that is, choosing the ranking π^* which maximizes the score

$$\pi^* = \underset{\pi_s \in \mathbb{S}_m}{argmax}\ P(\pi_s|e_t).$$

However, due to the possible high cardinality of $\mathbb{S}_m$, we propose an approximate method:

1. Compute the probability a posteriori of each component of the mixture given the instance e_t:

$$P(h_z|e_t) \propto \prod_{j=1}^{n_J} P(x_{j,t}|h_z) \cdot \prod_{k=1}^{n_K} P(y_{k,t}|h_z) \quad (4)$$

Algorithm 3: Method B: HNB-LR

Data: T: Training dataset; β: Maximum #of EM iterations; α: Minimum required improvement for each EM iteration; γ: #of k-means clustering algorithm restarts.
Result: The HNB-LR model fully specified.

Create a holdout partition $\{Tr, Tv\}$ from T; $Tr = \{e_t\}_{t=1}^{|Tr|}$;
$\tau_K^{Tv} \leftarrow -1$;
$r_H \leftarrow 0$;
// Forward search
for $i \leftarrow 1$ **to** 10 **do**
  model $\leftarrow$ EM(Tr, 2^i, α, γ);
  $\tau_K \leftarrow$ Evaluate the model over Tv dataset;
  if $(\tau_K > \tau_K^{Tv})$ **then**
    $r^H \leftarrow 2^i$;
  end
end
// Binary search
$r_H \leftarrow$ Get the best number of components by using a binary search in range $[\frac{r_H}{2}, r_H]$ with EM to learn the model and τ_K over Tv to evaluate;
best $\leftarrow$ EM(T, r_H, α, γ);
return best;

2. Solve a generalized aggregation problem by using the weighted Borda count over the set of weighted rankings $\{(\pi_0^{h_1}, w_1), \ldots, (\pi_0^{h_{r_H}}, w_{r_H})\}$, that is, the consensus rankings associated with the components of the mixture, and taking as weight w_z, the probability a posteriori computed for the mixture component $P(h_z|e_t)$.

4. Gaussian Mixture-Based Semi-Naive Bayes for Label Ranking

In this section, we go one step further by allowing interactions between the predictive variables. However, in order to maintain the complexity of the learning process under control, we decided to use a model in which, apart from identifying the number of mixture components, no structural learning is needed. Our proposal falls in the so-called *Semi-Naive Bayes* approach [30,39], and we restrict our study to continuous predictive variables. This constraint is quite natural in the LR problem, as all the benchmark datasets contain only continuous variables. In the future, we plan to adapt our method to also allow discrete predictive attributes, which, in general, means learning constrained graphical structures by limiting the number of dependencies allowed [30] or even dealing with *hybrid Bayesian networks* [40]. Learning PGMs with hidden variables is not an easy task, but there are several approaches in the literature, most based on the use of the *Structural EM* (*SEM*) algorithm [41].

4.1. Model Definition

Once we limit our model to contain only continuous predictive attributes $Y_1, \ldots, Y_{n_K}$, π, and H, and also avoid structural learning apart from n_H, we have to deal with representing the interactions between the variables. We maintain the interaction between π and the predictive variables to be channeled through the hidden variable H. Thus, explicit interactions are only allowed between the continuous attributes. As no structural learning of these relations is desired, we decided to model them by using a *multivariate Gaussian distribution*, $\mathcal{MN}(\vec{\mu}, \Sigma)$. This has the advantage of having to estimate only ${n_K}^2 + n_K$ parameters, as n_K are the values in the vector of means $\vec{\mu}$ and ${n_K}^2$ in the covariance matrix.

In the literature, a *Gaussian Mixture Model* (*GMM*) [42] is a parametric probability density function represented as a weighted sum of Gaussian component densities, where each component density is a multivariate Gaussian function. Therefore, we take advantage

of the widely used GMM to plug them into our PGM to deal with the LR problem. In the literature, we can find several variants of the GMM, which differ in the way the covariance matrix is constrained or not constrained. In particular, the standard or *full* GMM estimates a covariance matrix for each component with no additional constraint. On the other hand, in the *diag* variant, such a covariance matrix is constrained to be diagonal, which is equivalent to the NB assumption. The third option is to use a *tied* covariance matrix, which means estimating an unconstrained covariance matrix, but using it for all the components.

Figure 3 shows the graphical representation of the proposed *semi-naive Bayes* (*SNB*) model, where the large node including all the continuous attributes emphasizes the idea of modeling them jointly. The difference regarding the HNB presented in Section 3 is that the continuous variables now follow a multivariate Gaussian distribution given the root variable

$$P(Y|H) \rightsquigarrow P(Y|H = h_z) \sim \mathcal{MN}(\vec{\mu}^{h_z}, \Sigma^{h_z}), \;\; z = 1, \ldots, r_H$$

where Y is the set of continuous variables $\{Y_1, \ldots, Y_{n_K}\}$, $\vec{\mu}$ is the vector of means for $Y_1, \ldots, Y_{n_K}$, and Σ is the $n_K \times n_K$ covariance matrix.

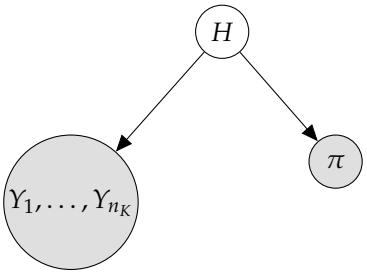

Figure 3. The proposed GMSNB model.

4.2. Parameter Estimation

As in the case of the proposed HNB algorithm, we use the EM algorithm to estimate the model parameters. Next, we point out the differences between this method and the univariate case (see Section 3.2).

- **E step**: Under the assumption that the parameters of the model $\{\vec{\mu}^{h_z}, \Sigma^{h_z}, \pi_0^{h_z}, \theta^{h_z}, p(h_z)\}, z = 1, \ldots, r_H$, are known, the probability of an example $e_t = (y_{1,t}, \ldots, y_{n_K,t}, \pi_t) = (\vec{y}_t, \pi_t)$ being in a mixture h_z is

$$\begin{aligned} P(h_z|\vec{y}_t, \pi_t) &= \frac{1}{C} \cdot P(h_z) \cdot P(\vec{y}_t, \pi_t|h_z) \\ &= \frac{1}{C} \cdot P(h_z) \cdot P(\pi_t|h_z) \cdot P(\vec{y}_t|h_z) \\ &= \frac{1}{C} \cdot P(h_z) \cdot \mathcal{M}(\pi_t : \pi_0^{h_z}, \theta^{h_z}) \cdot \mathcal{MN}(\vec{y}_t : \vec{\mu}^{h_z}, \Sigma^{h_z}) \end{aligned} \tag{5}$$

  where $C = \sum_{z=1}^{r_H} P(h_z)P(\vec{y}_t, \pi_t|h_z)$ is the normalization constant and $\mathcal{MN}(\vec{y}_t : \vec{\mu}^{h_z}, \Sigma^{h_z})$ stands for the probability density function of the multinormal distribution with parameters $\vec{\mu}^{h_z}$ and Σ^{h_z} given by

$$\mathcal{MN}(\vec{y} : \vec{\mu}^{h_z}, \Sigma^{h_z}) = \frac{1}{\sqrt{(2\pi)^{n_K}|\Sigma|^{h_z}}} e^{-\frac{1}{2}(\vec{y}-\vec{\mu}^{h_z})^T(\Sigma^{h_z})^{-1}(\vec{y}-\vec{\mu}^{h_z})}$$

  Here, $\vec{y}$ is a configuration of values for variables $(Y_1, \ldots, Y_{n_K})$, $|\Sigma|$ is the determinant of Σ, and -1 and T denote the inverse and transpose matrix operators, respectively.
- **M step**: Under the assumption that the probabilities of belonging to each mixture for all the examples are known, the parameters of the model can be estimated as follows:

– Continuous variables. Empirical means and covariance matrices are calculated in the standard way, using each instance being weighted by $w_t^{h_z} = P(h_z|\vec{y}_t, \pi_t)$ according to the expressions

$$N^{h_z} = \sum_{t=1}^{N} w_t^{h_z}$$

$$\vec{\mu}^{h_z} = \frac{1}{N^{h_z}} \sum_{t=1}^{N} w_t^{h_z} \cdot \vec{y}_t$$

$$\Sigma^{h_z} = \frac{1}{N^{h_z}} \sum_{t=1}^{N} w_t^{h_z} \cdot (\vec{y}_t - \vec{\mu}^{h_z}) \times (\vec{y}_t - \vec{\mu}^{h_z})^T$$

Here, $\times$ stands for the usual matrix product.
In the *tied* case, where all the components share the same covariance matrix, Σ, it is estimated as [43] (p. 71):

$$\Sigma = \frac{1}{N} \sum_{z=1}^{r_H} \sum_{t=1}^{N} w_t^{h_z} \cdot (\vec{y}_t - \vec{\mu}^{h_z}) \times (\vec{y}_t - \vec{\mu}^{h_z})^T$$

4.3. Learning Process

Method B, described in Section 3.3, is used to estimate the number of components for the mixture H. To do so, Algorithm 2 is modified as follows:

- In the E step, Equation (5) is used instead of Equation (3).
- In the M step, the expressions in Section 4.2 are used instead of the respective ones in Section 3.2.

4.4. Inference Process

The same inference process is used as in the HNB model (see Section 3.4). The only difference is that we now compute the posterior probability of each component of the mixture given the instance e_t by using the multivariate probability density function instead of Equation (4).

$$P(h_z|e_t) = P(h_z|\vec{y}_t) \propto \frac{1}{\sqrt{(2\pi)^{n_K}|\Sigma^{h_z}|}} e^{-\frac{1}{2}(\vec{y}-\vec{\mu}^{h_z})^T(\Sigma^{h_z})^{-1}(\vec{y}-\vec{\mu}^{h_z})} \quad (6)$$

5. Experimental Evaluation

In this section, we assess the mixture-based algorithms proposed to solve the LR problem. Below, we detail the datasets used, the algorithms tested, the methodology adopted, and the results obtained.

5.1. Datasets

Table 1 shows the main characteristics of the 21 datasets widely used as benchmark for the LR problem. The first 16 datasets were turned from multi-class (Type A) and regression (Type B) problems into the LR problem [2], while the last 5 datasets (Type R) correspond to real-world biological problems [10]. The columns #rankings and max #rankings represent the actual number of different rankings in the dataset and the maximum number of different rankings according to the number of classes (#classes), respectively. In the 21 datasets considered, all the predictive attributes (features) are continuous variables. A more detailed description of the datasets is provided at: https://scikit-lr.readthedocs.io/en/latest/user_guide/datasets.html#datasets (accessed on 29 March 2021).

5.2. Algorithms

In this study, we considered the following algorithms:

- The IBLR algorithm introduced in [2] (see Section 2.5). To identify the nearest neighbors, the Euclidean distance was used. To compute the prediction, the permutations associated with the k-nearest neighbors were weighted according to the neighbor's (inverse) distance to the input instance. Although the IBLR algorithm belongs to the *lazy* paradigm of machine learning, we carried out model learning to select the number k of nearest neighbors. We applied the following process using a fivefold cross-validation method (5-cv) over the training dataset to assess the goodness of each candidate value:
  1. We started with $k = 5$ and doubled it while the score was improving. From this process, we obtained k_l and k_r, that is, the number of nearest neighbors leading to the best score (the penultimate value tested) and the one stopping the iterative process (the last value tested), respectively.
  2. We applied a binary search in the range $[k_l, k_r]$. In this process, we took $k_m = \lfloor \frac{k_l + k_r}{2} \rfloor$, and if the score improved for k_m with respect to k_l, we then repeated this recursive process using the range $[k_m, k_r]$. Otherwise, the range $[k_l, k_m]$ was used.

  We kept the number of nearest neighbors that led to the best score.
- The LRT algorithm introduced in [2] (see Section 2.6).
- The HNBE-LR algorithm introduced in [23] (see Section 3.3.1). Note that all the attributes of the datasets are continuous variables. Thus, the parameters of the model were estimated by means of Gaussian distribution parameters (HNBE-LR-G). The hyperparameter values were $z_0 = 5$, $\beta = 1$, $\alpha = 0.001$, $p = 5$. The holdout for the training and validation datasets was 75%/25%.
- The HNB-LR algorithm (see Section 3.3.2). As in the previous case, we estimated the conditional probability distributions with Gaussian univariate distributions (HNB-LR-G). Furthermore, we binned the continuous variables using equal-frequency (HNB-LR-F), equal-width (HNB-LR-W), and entropy-based [44] (HNB-LR-E) discretization techniques, estimating the parameters of the model with Multinomial distribution parameters. The number of bins was set to 5 for equal-width and equal-frequency cases. The holdout for the training and validation datasets was 80%/20%. The γ value was fixed to 10.
- The GMSNB-LR algorithm (see Section 4), with a different covariance matrix for each component, *full* approach (GMSNB-LR-F), and sharing the same covariance matrix between all the components of the mixture, *tied* approach (GMSNB-LR-T). The holdout for the training and validation datasets was 80%/20%. The γ value was fixed to 10.

Table 1. Description of the datasets.

Dataset	Type	#Instances	#Features	#Classes	#Rankings	Max #Rankings
authorship	A	841	70	4	17	4!
bodyfat	B	252	7	7	236	7!
calhousing	B	20,640	4	4	24	4!
cpu	B	8192	6	5	119	5!
elevators	B	16,599	9	9	131	9!
fried	B	40,769	9	5	120	5!
glass	A	214	9	6	30	6!
housing	B	506	6	6	112	6!
iris	A	150	4	3	5	3!
pendigits	A	10,992	16	10	2081	10!
segment	A	2310	18	7	135	7!
stock	B	950	5	5	51	5!
vehicle	A	846	18	4	18	4!
vowel	A	528	10	11	294	11!
wine	A	178	13	3	5	3!
wisconsin	B	194	16	16	194	16!
spo	R	2465	24	11	2361	11!
heat	R	2465	24	6	622	6!
dtt	R	2465	24	4	24	4!
cold	R	2465	24	4	24	4!
diau	R	2465	24	7	967	7!

5.3. Methodology

We adopted the following design decisions:

- We used five repetitions of a tenfold cross-validation method (5×10-cv) to assess the algorithms.
- We used the Kendall rank correlation coefficient τ_K as goodness score: the higher, the better (see Section 2.2).
- To properly analyze the results, we carried out the standard statistical analysis procedure for machine learning [45,46], using the `exreport` tool [47]. The procedure is divided into two steps:
  1. First, we carried out a *Friedman test* [48] with significance level $\alpha = 0.05$. If the obtained p-value $\leq \alpha$, then we rejected the null hypothesis H_0 and concluded that at least one algorithm is not equivalent to the rest.
  2. Second, once the previous step rejected H_0, we applied a post hoc test using the *Holm's procedure* [49] to discover the outstanding algorithms. This test compares all the algorithms against the control algorithm, that is, the one ranked first by the Friedman test.
- We executed the experiments on computers running the CentOS Linux 7 operating system with an Intel(R) Xeon(R) E5–2630 CPU running at 2.40 GHz, and with 16 GB of RAM memory.

5.4. Results

In this section, we present and analyze the results obtained. We focus on accuracy (τ_K score) and CPU time.

5.4.1. Accuracy

First, we analyze the results obtained by the HNB-LR algorithms. The τ_K accuracy results for this family of algorithms are shown in Table 2. The cells contain the average and standard deviation over the test sets of the cross validation method for the rank correlation coefficient τ_K between the real and predicted permutations. The boldfaced values correspond to the algorithm(s) achieving the best mean accuracy for each dataset.

Table 2. Mean accuracy for each HNB-LR algorithm.

Dataset	HNBE-LR-G	HNB-LR-G	HNB-LR-F	HNB-LR-W	HNB-LR-E
authorship	0.907 (±0.028)	**0.919 (±0.018)**	0.905 (±0.021)	0.905 (±0.019)	0.909 (±0.017)
bodyfat	0.078 (±0.074)	**0.128 (±0.063)**	0.115 (±0.066)	0.116 (±0.062)	0.117 (±0.068)
calhousing	0.171 (±0.018)	0.303 (±0.024)	0.278 (±0.009)	0.198 (±0.011)	**0.343 (±0.013)**
cpu	0.360 (±0.023)	0.435 (±0.013)	**0.461 (±0.013)**	0.334 (±0.016)	0.459 (±0.013)
elevators	0.646 (±0.025)	**0.728 (±0.014)**	0.695 (±0.011)	0.664 (±0.013)	0.688 (±0.016)
fried	0.489 (±0.105)	**0.895 (±0.034)**	0.367 (±0.308)	0.404 (±0.303)	0.812 (±0.014)
glass	0.788 (±0.067)	0.793 (±0.061)	**0.849 (±0.051)**	0.834 (±0.057)	0.846 (±0.048)
housing	0.400 (±0.116)	**0.734 (±0.034)**	0.667 (±0.045)	0.639 (±0.041)	0.711 (±0.044)
iris	**0.963 (±0.060)**	0.962 (±0.048)	0.845 (±0.082)	0.900 (±0.056)	0.866 (±0.086)
pendigits	0.721 (±0.026)	0.914 (±0.003)	0.916 (±0.002)	0.912 (±0.003)	**0.920 (±0.003)**
segment	0.656 (±0.096)	0.926 (±0.008)	0.909 (±0.009)	0.928 (±0.008)	**0.939 (±0.007)**
stock	0.791 (±0.040)	**0.910 (±0.016)**	0.852 (±0.022)	0.861 (±0.016)	0.885 (±0.016)
vehicle	0.744 (±0.054)	0.790 (±0.038)	0.806 (±0.037)	0.798 (±0.059)	**0.810 (±0.033)**
vowel	0.545 (±0.067)	0.817 (±0.046)	**0.865 (±0.027)**	0.863 (±0.028)	0.616 (±0.092)
wine	**0.935 (±0.044)**	0.935 (±0.054)	0.927 (±0.064)	0.915 (±0.076)	0.928 (±0.053)
wisconsin	0.295 (±0.070)	0.355 (±0.050)	**0.373 (±0.046)**	0.334 (±0.055)	0.324 (±0.093)
cold	0.071 (±0.039)	**0.080 (±0.035)**	0.080 (±0.037)	0.073 (±0.037)	0.076 (±0.030)
diau	0.215 (±0.023)	**0.219 (±0.022)**	0.215 (±0.025)	0.219 (±0.024)	0.217 (±0.023)
dtt	0.119 (±0.034)	**0.120 (±0.034)**	0.114 (±0.033)	0.107 (±0.030)	0.119 (±0.031)
heat	0.054 (±0.025)	**0.061 (±0.028)**	0.057 (±0.026)	0.055 (±0.027)	0.059 (±0.024)
spo	0.147 (±0.016)	0.146 (±0.015)	**0.148 (±0.016)**	0.146 (±0.015)	0.148 (±0.015)

We base our analysis on the statistical procedure described in Section 5.3:

1. The p-value obtained in the Friedman test was 3.613×10^{-5}). Therefore, the null hypothesis (H_0) was rejected, and at least one of the tested algorithms was different.
2. Table 3 shows the results for the post hoc test by taking HNB-LR-G, the algorithm ranked first by the Friedman test, as the control. The columns *rank* and p-value represent the ranking obtained by the Friedman test and the p-value adjusted by Holm's procedure, respectively. The columns *win*, *tie*, and *loss* contain the number of times that the control algorithm wins, ties, and loses with respect to the row-wise algorithm. The p-values for the non-rejected null hypothesis are boldfaced.

Table 3. Results of the post hoc test for the mean accuracy of HNB-LR algorithms.

Method	Rank	p-Value	Win	Tie	Loss
HNB-LR-G	2.05	-	-	-	-
HNB-LR-E	2.33	$\mathbf{5.5818} \times 10^{-1}$	14	0	7
HNB-LR-F	2.86	$\mathbf{1.9422} \times 10^{-1}$	14	0	7
HNB-LR-W	3.62	3.8394×10^{-3}	16	0	5
HNBE-LR-G	4.14	7.0206×10^{-5}	18	0	3

According to these results and the statistical analysis performed, we can conclude the following.

- The HNBE-LR-G algorithm is the worst method. The reason is obvious if we analyze Table 4, where we show the number of components (on average) selected for each algorithm. It is clear that this number is too small for HNBE-LR-G, which clearly suffers from premature early stopping. Furthermore, we must recall that this algorithm is the only one which prunes low weight components during its performance. As a consequence, HNBE-LR-G does not obtain a good probability estimation.
- The HNB-LR-G algorithm is ranked first and is statistically different to the HNB-LR-W and HNBE-G algorithms. In the case of HNB-LR-W, the reason is not the number of

selected components, but the equal-width discretization carried out, which produces poor binning in comparison, for example, with the supervised entropy-based method.

- Although the HNB-LR-G algorithm is ranked first, the post hoc test reveals no significant difference with respect to the HNB-LR-E and HNB-LR-F algorithms. This opens the door to future research on more complex Bayesian network structures.
- As stated in [36], a large number of components are required to obtain a good probability estimation and, in our problem, a good ranking prediction.

Once we have determined that the best HNB-LR algorithms are HNB-LR-G, HNB-LR-E, and HNB-LR-F, we introduce the two algorithms allowing interactions among the predictive attributes in the study, that is, those based on the use of the multivariate Gaussian distribution to jointly model the (numerical) attributes: GMSNB-LR-F and GMSNB-LR-T. The results are shown in the two leftmost columns of Table 5. To complete the comparison, we also show the results for the two state-of-the-art LR classifiers [2] described in Sections 2.5 and 2.6.

Again, we applied the statistical analysis procedure described in Section 5.3, including also the three HNB-LR algorithms selected from the previous study:

1. The p-value obtained for the Friedman test was 2.503×10^{-7}. Therefore, we rejected the null hypothesis (H_0), i.e., at least one algorithm is different to the rest.
2. As IBLR is ranked first by the Friedman test, we took it as the control and performed a post hoc test using Holm's procedure. Table 6 shows the results for the post hoc test.

Considering these results, we can conclude the following.

- The IBLR algorithm is ranked first, being statistically different to all the tested algorithms except GMSNB-LR-T. Note that IBLR is a fine-tuned algorithm, as can be observed from the number of neighbors selected for each dataset (see Table 4), which are far from standard values $(3, 5, \ldots)$.
- The GMSNB-LR-T algorithm has a remarkable performance, being non-significantly different to IBLR. This is a very important finding because, as recognized in the literature, the instance-based algorithm generally outperforms the model-based algorithms, being necessary to use ensembles of the LRT algorithm to compete with it [14].
- The GMSNB-LR-T algorithm also has in its favor that it is able to cope with all the tested datasets, while the experiments for IBLR cannot finish in a maximum of 168 h (one week) for `fried` dataset (notice the *empty* cell in Table 5). As can be observed, `fried` is the largest dataset in our experiments, which reveals the disadvantage of using IBLR for larger domains.
- The GMSNB-LR-T algorithm behaves better than GMSNB-LR-F, which is also ranked behind HNB-LR-G. Two explanations are plausible for this behavior: First, the amount of data considered is limited, so it can be scarce for the estimation of many covariance matrices when the number of components grows. Second, it is well known that increasing the number of components can be enough to model the correlations between the features. In fact, if we observe Table 4, we realize that the numbers of components selected for GMSNB-LR-T and HNB-LR-G are noticeably greater than that selected for GMSNB-LR-F.

Table 4. Mean number of components for each mixture-based algorithm and mean number of nearest neighbors for the IBLR algorithm.

Dataset	HNBE-LR-G	HNB-LR-G	HNB-LR-F	HNB-LR-W	HNB-LR-E	GMSNB-LR-F	GMSNB-LR-T	IBLR
authorship	5.82 (±0.87)	28.66 (±24.98)	25.08 (±21.39)	22.18 (±21.95)	38.60 (±45.37)	3.26 (±0.44)	44.92 (±49.51)	6.52 (±1.62)
bodyfat	5.88 (±0.90)	36.74 (±27.42)	21.04 (±17.29)	24.92 (±23.04)	20.14 (±32.20)	20.66 (±14.57)	42.20 (±30.10)	28.68 (±13.49)
calhousing	7.12 (±1.67)	280.78 (±103.40)	357.72 (±84.48)	119.56 (±59.87)	461.96 (±81.83)	257.90 (±116.43)	453.58 (±88.94)	27.24 (±6.85)
cpu	6.60 (±1.84)	142.62 (±106.17)	153.10 (±83.49)	17.40 (±12.02)	230.06 (±116.99)	60.94 (±45.28)	372.62 (±129.13)	46.52 (±8.61)
elevators	5.44 (±0.67)	216.10 (±78.63)	95.30 (±42.99)	105.98 (±45.74)	134.12 (±49.31)	52.12 (±18.61)	219.50 (±75.32)	27.10 (±5.43)
fried	7.14 (±2.19)	500.82 (±36.01)	167.34 (±138.34)	171.58 (±131.69)	408.98 (±105.35)	257.62 (±21.08)	475.22 (±92.00)	
glass	5.34 (±0.56)	14.72 (±11.81)	74.56 (±37.55)	42.40 (±17.82)	26.14 (±10.63)	5.84 (±5.63)	48.32 (±13.45)	5.26 (±0.53)
housing	5.56 (±0.67)	64.08 (±19.45)	79.46 (±47.78)	68.52 (±47.02)	38.76 (±36.03)	45.94 (±25.61)	116.60 (±28.84)	39.88 (±6.94)
iris	5.78 (±0.71)	14.44 (±9.45)	21.46 (±14.90)	17.12 (±10.96)	16.36 (±12.78)	8.50 (±2.39)	21.62 (±16.59)	8.46 (±2.00)
pendigits	6.22 (±1.31)	503.20 (±29.99)	484.00 (±55.38)	451.62 (±87.11)	484.00 (±49.35)	126.16 (±6.82)	509.44 (±18.10)	6.18 (±0.56)
segment	5.56 (±0.61)	166.98 (±55.22)	341.02 (±138.26)	303.68 (±145.76)	383.14 (±139.14)	45.10 (±16.55)	371.08 (±109.12)	8.16 (±1.74)
stock	6.16 (±1.54)	123.42 (±48.48)	99.66 (±33.97)	56.60 (±31.58)	95.54 (±30.49)	45.96 (±13.12)	244.78 (±107.07)	5.66 (±1.15)
vehicle	6.92 (±1.63)	48.06 (±48.45)	239.58 (±169.52)	179.26 (±171.77)	111.72 (±114.48)	10.34 (±4.96)	191.38 (±154.49)	8.80 (±2.22)
vowel	6.04 (±1.19)	96.70 (±27.75)	239.58 (±32.12)	231.36 (±38.00)	132.70 (±60.35)	13.28 (±4.44)	251.86 (±13.70)	5.98 (±0.98)
wine	5.48 (±0.65)	11.76 (±13.61)	12.22 (±12.34)	16.30 (±17.10)	11.40 (±13.93)	3.68 (±1.25)	9.58 (±12.81)	7.16 (±2.48)
wisconsin	5.88 (±1.24)	23.10 (±14.23)	21.10 (±13.76)	49.90 (±41.21)	45.38 (±27.82)	4.60 (±1.85)	35.84 (±15.83)	14.24 (±4.75)
cold	6.28 (±1.40)	84.84 (±107.13)	72.00 (±99.91)	84.70 (±109.52)	98.42 (±148.12)	83.00 (±143.73)	229.26 (±134.65)	23.44 (±22.88)
diau	5.76 (±1.04)	13.24 (±20.64)	23.50 (±50.16)	15.92 (±23.00)	14.58 (±14.84)	18.40 (±15.89)	35.76 (±48.31)	132.04 (±40.12)
dtt	5.50 (±0.84)	104.22 (±154.80)	82.84 (±118.50)	120.76 (±148.76)	20.50 (±27.44)	36.52 (±46.53)	116.42 (±152.29)	101.80 (±27.78)
heat	6.24 (±1.15)	71.08 (±133.42)	74.44 (±116.93)	66.44 (±128.37)	77.72 (±127.87)	31.48 (±80.92)	79.68 (±135.62)	23.14 (±13.40)
spo	5.90 (±0.95)	10.16 (±9.16)	9.24 (±12.53)	11.26 (±13.44)	12.62 (±15.67)	18.28 (±31.55)	6.96 (±5.76)	354.66 (±314.51)

Table 5. Mean accuracy for the GMSNB-LR, IBLR, and LRT algorithms.

Dataset	GMSNB-LR-F	GMSNB-LR-T	IBLR	LRT
authorship	0.838 (±0.033)	0.925 (±0.019)	**0.935 (±0.014)**	0.883 (±0.024)
bodyfat	0.080 (±0.078)	0.151 (±0.074)	**0.230 (±0.055)**	0.151 (±0.066)
calhousing	0.295 (±0.020)	0.284 (±0.017)	**0.351 (±0.010)**	0.319 (±0.012)
cpu	0.418 (±0.016)	0.432 (±0.027)	**0.506 (±0.013)**	0.404 (±0.014)
elevators	0.774 (±0.017)	**0.780 (±0.009)**	0.730 (±0.006)	0.668 (±0.010)
fried	0.908 (±0.025)	**0.927 (±0.014)**		0.727 (±0.103)
glass	0.790 (±0.061)	0.879 (±0.059)	0.864 (±0.051)	**0.902 (±0.040)**
housing	0.695 (±0.044)	0.782 (±0.029)	0.716 (±0.031)	**0.811 (±0.029)**
iris	0.925 (±0.053)	0.962 (±0.035)	**0.963 (±0.042)**	0.953 (±0.044)
pendigits	0.891 (±0.003)	0.927 (±0.002)	**0.943 (±0.002)**	0.942 (±0.002)
segment	0.884 (±0.015)	0.947 (±0.008)	**0.961 (±0.005)**	0.955 (±0.006)
stock	0.894 (±0.015)	0.922 (±0.014)	**0.926 (±0.014)**	0.898 (±0.016)
vehicle	0.721 (±0.051)	0.802 (±0.042)	**0.860 (±0.026)**	0.818 (±0.040)
vowel	0.589 (±0.039)	**0.906 (±0.015)**	0.889 (±0.018)	0.753 (±0.033)
wine	0.937 (±0.054)	**0.944 (±0.042)**	0.941 (±0.048)	0.870 (±0.078)
wisconsin	0.244 (±0.085)	0.336 (±0.108)	**0.499 (±0.041)**	0.374 (±0.040)
cold	0.072 (±0.036)	**0.090 (±0.042)**	0.090 (±0.035)	0.048 (±0.031)
diau	0.222 (±0.025)	0.219 (±0.025)	**0.234 (±0.026)**	0.129 (±0.022)
dtt	0.124 (±0.030)	0.119 (±0.032)	**0.159 (±0.033)**	0.080 (±0.033)
heat	0.064 (±0.029)	0.046 (±0.025)	**0.070 (±0.022)**	0.039 (±0.023)
spo	0.148 (±0.016)	0.148 (±0.015)	**0.149 (±0.017)**	0.090 (±0.018)

Table 6. Results of the post hoc test for the mean accuracy of the algorithms.

Method	Rank	p-Value	Win	Tie	Loss
IBLR	1.76	-	-	-	-
GMSNB-LR-T	2.93	$\mathbf{8.012} \times 10^{-2}$	14	0	7
HNB-LR-G	3.98	1.791×10^{-3}	19	0	2
LRT	4.52	1.029×10^{-5}	18	0	3
HNB-LR-E	4.67	5.271×10^{-5}	20	0	1
GMSNB-LR-F	5.00	5.955×10^{-2}	19	0	2
HNB-LR-F	5.14	2.369×10^{-6}	20	0	1

5.4.2. Time

In this study, we consider model-based and instance-based machine learning algorithms, which clearly distribute the CPU time for the learning and inference steps differently. Although the CPU time for the whole process (learning from the training dataset and validating with test dataset) is generally reported, we separate the CPU time for the learning and inference steps because (i) a model is learnt once but queried many times and (ii) most real-world applications require online predictions but allow for offline fitting. Tables 7 and 8 show the average CPU time for the learning and inference steps. In light of these results, we can conclude the following.

- The HNBE-LR-G algorithm is the fastest method during the learning step because it suffers from premature early stopping, which gives rise to a fast but poor algorithm. On the other hand, the LRT algorithm is the fastest method during the inference step, which is a common situation for tree-based algorithms.
- The IBLR algorithm is faster than the HNB-LR and GMSNB-LR algorithms in the learning step. However, during inference, the IBLR algorithm computes the distance between the input instance and the instances in the training dataset, which clearly increases the CPU time required by the algorithm.
- The GMSNB-LR-F algorithm is generally faster than the GMSNB-LR-T algorithm, both in learning and inference. This is due to the number of components selected by the GMSNB-LR-F algorithm in comparison to the GMSNB-LR-T algorithm. In a similar way, the HNB-LR-G algorithm is faster than the HNB-LR-F, HNB-LR-W, and HNB-LR-E algorithms, as the the latter ones apply a discretization procedure prior to the learning and inference steps.

Table 7. Mean CPU time (in seconds) for the learning step of each algorithm.

Dataset	HNBE-LR-G	HNB-LR-G	HNB-LR-F	HNB-LR-W	HNB-LR-E	GMSNB-LR-F	GMSNB-LR-T	IBLR	LRT
authorship	1.705 ± 0.532	61.196 ± 9.992	89.829 ± 12.334	87.027 ± 12.214	89.588 ± 17.826	45.615 ± 12.528	63.305 ± 26.925	**0.750** ± 0.123	6.417 ± 0.359
bodyfat	**0.242** ± 0.091	5.225 ± 1.886	14.765 ± 5.212	19.063 ± 9.117	26.695 ± 14.450	4.178 ± 1.082	7.411 ± 2.700	0.408 ± 0.117	0.459 ± 0.021
calhousing	**24.985** ± 10.129	4460.303 ± 2556.481	934.879 ± 403.026	549.931 ± 330.861	924.278 ± 193.472	2477.003 ± 894.089	1640.548 ± 879.352	222.602 ± 18.316	617.978 ± 95.233
cpu	**10.822** ± 6.953	832.624 ± 453.921	434.776 ± 115.788	284.175 ± 155.825	334.790 ± 92.549	880.115 ± 271.609	1469.499 ± 588.102	53.066 ± 1.789	119.613 ± 7.050
elevators	**15.681** ± 4.200	2252.878 ± 909.937	3878.157 ± 1111.013	2912.576 ± 1199.323	1822.243 ± 564.493	3696.911 ± 817.021	1974.011 ± 467.630	154.594 ± 9.019	1685.996 ± 174.829
fried	**51.613** ± 29.227	6783.099 ± 4193.126	3050.954 ± 1459.152	8534.450 ± 3529.126	3255.716 ± 1247.376	3836.842 ± 625.103	7345.125 ± 4261.981		6049.934 ± 3015.594
glass	**0.139** ± 0.037	4.784 ± 1.416	21.871 ± 6.983	24.472 ± 6.890	21.390 ± 4.724	5.001 ± 0.877	12.855 ± 3.936	0.142 ± 0.003	0.439 ± 0.038
housing	**0.278** ± 0.070	33.743 ± 9.953	74.223 ± 23.715	77.063 ± 36.041	72.623 ± 24.394	12.118 ± 4.511	37.575 ± 20.451	1.011 ± 0.129	0.843 ± 0.035
iris	0.119 ± 0.031	5.103 ± 1.850	5.940 ± 2.908	8.612 ± 2.794	10.060 ± 4.408	2.891 ± 0.520	3.740 ± 1.200	0.146 ± 0.029	**0.024** ± 0.015
pendigits	**14.921** ± 6.446	4958.630 ± 533.195	7526.738 ± 1086.477	3363.739 ± 1379.314	2181.268 ± 503.646	1370.683 ± 169.772	2171.361 ± 377.718	37.522 ± 0.314	137.337 ± 3.893
segment	**1.893** ± 0.633	306.300 ± 71.676	614.536 ± 388.197	297.176 ± 96.372	633.483 ± 207.526	155.155 ± 31.039	680.656 ± 198.149	3.290 ± 0.712	43.915 ± 0.459
stock	1.153 ± 0.642	78.335 ± 18.392	206.016 ± 20.207	203.993 ± 29.806	215.489 ± 23.562	51.965 ± 10.650	110.541 ± 46.287	**0.678** ± 0.078	1.767 ± 0.044
vehicle	1.361 ± 0.645	46.526 ± 17.221	75.399 ± 19.064	131.554 ± 42.902	187.489 ± 51.417	28.025 ± 13.600	120.622 ± 65.717	**0.784** ± 0.162	2.614 ± 0.201
vowel	0.581 ± 0.247	39.580 ± 5.972	72.241 ± 8.261	76.060 ± 10.918	189.684 ± 98.049	22.197 ± 2.436	47.121 ± 6.398	**0.364** ± 0.004	5.231 ± 0.136
wine	**0.125** ± 0.039	1.565 ± 0.502	4.887 ± 1.482	5.235 ± 2.170	6.360 ± 2.088	1.264 ± 0.129	1.614 ± 0.468	0.132 ± 0.029	0.250 ± 0.095
wisconsin	**0.265** ± 0.155	17.549 ± 3.963	13.207 ± 2.893	20.881 ± 10.524	40.325 ± 9.262	6.196 ± 0.661	13.596 ± 4.252	0.287 ± 0.060	2.414 ± 0.081
cold	**3.400** ± 1.779	318.876 ± 133.326	373.824 ± 192.488	287.441 ± 121.705	212.068 ± 85.827	309.730 ± 195.366	342.582 ± 154.869	5.489 ± 2.379	558.146 ± 268.333
diau	**2.485** ± 1.233	234.570 ± 60.736	296.337 ± 85.944	175.379 ± 46.762	273.500 ± 88.171	266.421 ± 62.646	176.723 ± 60.193	14.260 ± 2.636	509.939 ± 173.209
dtt	**2.308** ± 0.900	320.371 ± 164.964	440.020 ± 233.137	317.019 ± 167.063	155.423 ± 32.987	246.605 ± 80.833	228.628 ± 119.854	11.159 ± 1.455	436.914 ± 52.761
heat	**3.444** ± 1.448	323.654 ± 161.205	359.967 ± 148.471	243.728 ± 119.877	293.548 ± 147.877	250.223 ± 92.645	226.749 ± 144.860	6.322 ± 1.336	149.847 ± 22.709
spo	**3.106** ± 1.114	345.584 ± 57.367	364.797 ± 97.080	238.538 ± 47.706	357.678 ± 121.603	338.189 ± 90.514	218.729 ± 34.441	33.453 ± 19.871	330.353 ± 61.667

Table 8. Mean CPU time (in seconds) for the inference step of each algorithm.

Dataset	HNBE-LR-G	HNB-LR-G	HNB-LR-F	HNB-LR-W	HNB-LR-E	GMSNB-LR-F	GMSNB-LR-T	IBLR	LRT
authorship	0.007 ± 0.001	0.012 ± 0.012	0.062 ± 0.041	0.057 ± 0.034	0.044 ± 0.047	0.008 ± 0.009	0.025 ± 0.026	0.022 ± 0.000	**0.002** ± 0.000
bodyfat	**0.001** ± 0.000	0.002 ± 0.000	0.012 ± 0.012	0.008 ± 0.010	0.007 ± 0.009	0.003 ± 0.002	0.005 ± 0.004	0.006 ± 0.001	**0.001** ± 0.000
calhousing	**0.053** ± 0.003	0.409 ± 0.217	0.682 ± 0.197	0.376 ± 0.177	0.617 ± 0.154	0.348 ± 0.162	0.354 ± 0.122	3.675 ± 0.033	0.116 ± 0.011
cpu	**0.024** ± 0.002	0.069 ± 0.038	0.219 ± 0.121	0.068 ± 0.025	0.263 ± 0.121	0.067 ± 0.031	0.260 ± 0.094	0.713 ± 0.020	0.033 ± 0.002
elevators	**0.059** ± 0.002	0.152 ± 0.049	0.321 ± 0.117	0.252 ± 0.098	0.128 ± 0.040	0.214 ± 0.055	0.345 ± 0.102	2.620 ± 0.027	0.243 ± 0.088
fried	**0.124** ± 0.013	0.779 ± 0.362	0.328 ± 0.249	1.016 ± 0.750	1.009 ± 0.493	0.437 ± 0.033	1.250 ± 0.798		0.264 ± 0.263
glass	**0.001** ± 0.000	0.002 ± 0.003	0.021 ± 0.013	0.015 ± 0.013	0.008 ± 0.008	0.003 ± 0.004	0.009 ± 0.008	0.004 ± 0.000	**0.001** ± 0.000
housing	0.002 ± 0.001	0.011 ± 0.011	0.026 ± 0.014	0.018 ± 0.011	0.014 ± 0.013	0.006 ± 0.005	0.024 ± 0.020	0.012 ± 0.001	**0.001** ± 0.000
iris	**0.001** ± 0.000	0.004 ± 0.007	0.005 ± 0.006	0.005 ± 0.006	0.003 ± 0.006	0.003 ± 0.004	0.004 ± 0.004	0.003 ± 0.000	**0.001** ± 0.000
pendigits	0.055 ± 0.004	0.533 ± 0.120	1.380 ± 0.274	0.648 ± 0.357	0.346 ± 0.049	0.166 ± 0.041	0.570 ± 0.111	1.266 ± 0.013	**0.025** ± 0.000
segment	0.011 ± 0.002	0.049 ± 0.019	0.093 ± 0.046	0.135 ± 0.080	0.191 ± 0.079	0.045 ± 0.016	0.190 ± 0.062	0.085 ± 0.002	**0.005** ± 0.000
stock	0.003 ± 0.000	0.015 ± 0.011	0.029 ± 0.013	0.021 ± 0.014	0.033 ± 0.013	0.022 ± 0.014	0.072 ± 0.030	0.020 ± 0.000	**0.002** ± 0.000
vehicle	0.004 ± 0.000	0.007 ± 0.008	0.065 ± 0.051	0.097 ± 0.086	0.069 ± 0.059	0.006 ± 0.007	0.064 ± 0.050	0.018 ± 0.000	**0.002** ± 0.000
vowel	0.003 ± 0.000	0.012 ± 0.011	0.084 ± 0.023	0.075 ± 0.028	0.046 ± 0.021	0.013 ± 0.013	0.041 ± 0.014	0.011 ± 0.000	**0.002** ± 0.000
wine	**0.001** ± 0.000	0.001 ± 0.001	0.007 ± 0.007	0.006 ± 0.007	0.007 ± 0.009	**0.001** ± 0.000	0.002 ± 0.001	0.003 ± 0.000	**0.001** ± 0.000
wisconsin	0.002 ± 0.000	0.007 ± 0.011	0.016 ± 0.012	0.027 ± 0.018	0.019 ± 0.014	0.004 ± 0.005	0.007 ± 0.006	0.005 ± 0.000	**0.001** ± 0.000
cold	**0.011** ± 0.001	0.033 ± 0.021	0.071 ± 0.073	0.096 ± 0.086	0.122 ± 0.165	0.048 ± 0.053	0.128 ± 0.095	0.099 ± 0.007	0.017 ± 0.005
diau	0.012 ± 0.001	0.023 ± 0.014	0.047 ± 0.041	0.042 ± 0.028	0.034 ± 0.018	0.031 ± 0.015	0.045 ± 0.034	0.148 ± 0.017	**0.011** ± 0.002
dtt	**0.010** ± 0.001	0.034 ± 0.027	0.083 ± 0.078	0.129 ± 0.136	0.039 ± 0.028	0.033 ± 0.024	0.076 ± 0.070	0.121 ± 0.008	0.011 ± 0.001
heat	0.012 ± 0.001	0.025 ± 0.016	0.054 ± 0.039	0.097 ± 0.141	0.102 ± 0.143	0.036 ± 0.035	0.065 ± 0.069	0.101 ± 0.006	**0.006** ± 0.001
spo	0.014 ± 0.001	0.022 ± 0.013	0.032 ± 0.022	0.043 ± 0.018	0.033 ± 0.012	0.036 ± 0.028	0.028 ± 0.014	0.310 ± 0.189	**0.006** ± 0.001

6. Limitations

As observed in the previous section, allowing interactions between the predictive variables represents a crucial issue with respect to the univariate case. Actually, our mixture-based model emerges as competitive with IBLR. However, there are still some weaknesses in our study/proposal:

- The CPU time data shown in Table 7 suggest that our method does not scale as would be desirable. In fact, it seems that the number of instances has a greater influence on the CPU time than the number of variables. However, more analysis is needed to clarify this point. Feature subset selection and stratification could lead to scalability improvements.
- Interactions between the predictive variables are limited to the numerical ones. Allowing interactions among the discrete variables and also mixed interactions should be studied. The literature on Bayesian network classifiers [30,39] and hybrid Bayesian networks may help in this task [50].
- Currently, the method only works with *complete rankings*. Nevertheless, in real-world applications it is usual to allow the agent to rank only certain labels. We think our techniques can be adapted to deal with incomplete rankings.

7. Conclusions

This study explores the use of mixture-based algorithms to solve the LR problem. The main problem is to model the target variable, as it takes values in the set of permutations defined over the class variable. We solve this shortcoming by introducing a hidden variable as root, so all the variables can be modeled by using conditional distributions. In particular, we base our approach on the Naive Bayes structure, with the hidden variable being the root of the model. We then go a step further by allowing interactions between the (numerical) predictive variables, thus designing a Semi-Naive Bayes model. Learning algorithms based on the well-known EM estimation principle are proposed for both cases. The inference is designed as a combination of probabilistic inference and rank aggregation.

From the experimental evaluation, we observe that the Naive Bayes approach is comparable in score to the decision trees for the LR problem, while the Semi-Naive Bayes approach (in particular, the one sharing a single covariance matrix among all the components) outperforms the Naive Bayes and decision tree-based algorithms, being also competitive with the state-of-the-art model-free algorithm based on the nearest neighbors method (IBLR). The good performance of this algorithm (GMSNB-LR-T) with respect to IBLR is reinforced by its better behavior at the inference stage, where IBLR needs a great amount of time when facing large datasets.

As future research, we propose two possible extensions to this work: First, we plan to extend our approach to cope with incomplete data in the ranking variable, that is, cases in which not all the labels are ranked in the instances of the training set. In the literature, this step is solved by *completing* those rankings before learning the model. However, we think this step can be introduced in the EM algorithm. In fact, model-based algorithms have shown better behavior than model-free ones when learning from incomplete rankings, so open the door to future promising research. Second, we plan to extend the mixture-based algorithms to the LR problems whose target ranking is a partial or bucket order, that is, a ranking in which some labels can be tied. This problem, which is generally termed the *Partial Label Ranking* (*PLR*) problem [51,52], introduces new challenges, such as the infeasibility of using the Mallows distribution to model the target variable.

Author Contributions: Conceptualization, E.G.R., J.C.A., J.A.A., and J.A.G.; Formal analysis, E.G.R., J.C.A., J.A.A., and J.A.G.; Funding acquisition, J.A.G.; Investigation, E.G.R., J.C.A., J.A.A., and J.A.G.; Methodology, E.G.R., J.C.A., J.A.A., and J.A.G.; Software, E.G.R., and J.C.A.; Supervision, J.A.A. and J.A.G.; Validation, E.G.R., J.C.A., J.A.A., and J.A.G.; Writing—original draft, E.G.R., J.C.A., J.A.A., and J.A.G.; Writing—review and editing, E.G.R., J.C.A., J.A.A., and J.A.G.. All authors have read and agreed to the published version of the manuscript.

Funding: This study has been partially funded by the Spanish Government, FEDER funds and the JCCM through the projects PID2019–106758GB–C33/AEI/10.13039/501100011033, TIN2016–77902–C3–1–P, and SBPLY/17/180501/000493. Juan C. Alfaro has also been funded by the FPU scholarship FPU18/00181 by MCIU.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The running examples of the paper together with other basic models are available at: https://github.com/alfaro96/scikit-lr (accessed on 30 March 2021) .

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Vembu, S.; Gärtner, T. Label Ranking Algorithms: A Survey. In *Preference Learning*; Springer: Berlin/Heidelberg, Germany, 2010; pp. 45–64.
2. Cheng, W.; Hühn, J.; Hüllermeier, E. Decision tree and instance-based learning for label ranking. In Proceedings of the 26th Annual International Conference on Machine Learning, Montreal, QC, Canada, 14–18 June 2009; pp. 161–168.
3. Dery, L. Multi-label Ranking: Mining Multi-label and Label Ranking Data. *arXiv* **2021**, arXiv:2101.00583.
4. Hernández, J.; Inza, I.; Lozano, J.A. Weak supervision and other non-standard classification problems: A taxonomy. *Pattern Recognit. Lett.* **2016**, *69*, 49–55. [CrossRef]
5. Charte, D.; Charte, F.; García, S.; Herrera, F. A snapshot on nonstandard supervised learning problems: taxonomy, relationships, problem transformations and algorithm adaptations. *Prog. Artif. Intell.* **2019**, *8*, 1–14. [CrossRef]
6. Werbin-Ofir, H.; Dery, L.; Shmueli, E. Beyond majority: Label ranking ensembles based on voting rules. *Expert Syst. Appl.* **2019**, *136*, 50–61. [CrossRef]
7. Esmeli, R.; Bader-El-Den, M.; Abdullahi, H. Session Similarity Based Approach for Alleviating Cold-start Session Problem in e-Commerce for Top-N Recommendations. In Proceedings of the 12th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management, Setubal, Portugal, 2–4 November 2020; pp. 179–186.
8. Cheng, W.; Henzgen, S.; Hüllermeier, E. Labelwise versus Pairwise Decomposition in Label Ranking. In Proceedings of the Workshop on Lernen, Wissen & Adaptivität, Bamberg, Germany, 7–9 October 2013; pp. 129–136.
9. Gurrieri, M.; Fortemps, P.; Siebert, X. Alternative Decomposition Techniques for Label Ranking. In Proceedings of the 15th International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems, Montpellier, France, 15–19 July 2014; pp. 464–474.
10. Hüllermeier, E.and Fürnkranz, J.; Cheng, W.; Brinker, K. Label ranking by learning pairwise preferences. *Artif. Intell.* **2008**, *172*, 1897–1916. [CrossRef]
11. Har-Peled, S.; Roth, D.; Zimak, D. Constraint Classification for Multiclass Classification and Ranking. In Proceedings of the 2002 Neural Information Processing Systems Conference, Vancouver, BC, Cabada, 9–14 December 2002; pp. 785–792.
12. de Sá, C.R.; Soares, C.; Jorge, A.M.; Azevedo, P.; Costa, J. Mining Association Rules for Label Ranking. In *Advances in Knowledge Discovery and Data Mining*; Springer: Berlin/Heidelberg, Germany, 2011; pp. 432–443.
13. Ribeiro, G.; Duivesteijn, W.; Soares, C.; Knobbe, A.J. Multilayer Perceptron for Label Ranking. In Proceedings of the 22nd international conference on Artificial Neural Networks and Machine Learning, Lausanne, Switzerland, 11–14 September 2012; pp. 25–32.
14. Aledo, J.; Gámez, J.; Molina, D. Tackling the supervised label ranking problem by bagging weak learners. *Inf. Fusion* **2017**, *35*, 38–50. [CrossRef]
15. de Sá, C.R.; Soares, C.; Knobbe, A.; Cortez, P. Label Ranking Forests. *Expert Syst.* **2017**, *34*, e12166. [CrossRef]
16. Zhou, Y.; Qiu, G. Random forest for label ranking. *Expert Syst. Appl.* **2018**, *112*, 99–109. [CrossRef]
17. Dery, L.; Shmueli, E. BoostLR: A Boosting-Based Learning Ensemble for Label Ranking Tasks. *IEEE Access* **2020**, *8*, 176023–176032. [CrossRef]
18. Koller, D.; Friedman, N. *Probabilistic Graphical Models: Principles and Techniques—Adaptive Computation and Machine Learning*; MIT Press: Cambridge, MA, USA, 2009.
19. Jensen, F.V.; Nielsen, T.D. *Bayesian Networks and Decision Graphs*; Springer: Berlin/Heidelberg, Germany, 2007.
20. Cheng, W.; Dembczynski, K.; Hüllermeier, E. Label Ranking Methods based on the Plackett-Luce Model. In Proceedings of the 27th Annual International Conference on Machine Learning, Haifa, Israel, 21–24 June 2010; pp. 215–222.
21. Fernández, A.; Gámez, J.A.; Rumí, R.; Salmerón, A. Data clustering using hidden variables in hybrid Bayesian networks. *Prog. Artif. Intell.* **2014**, *2*, 141–152. [CrossRef]
22. Mallows, C.L. Non-Null Ranking Models. *Biometrika* **1957**, *44*, 114–130. [CrossRef]
23. Alfaro, J.C.; González, E.; Aledo, J.A.; Gámez, J.A. A Probabilistic Graphical Model-Based Approach for the Label Ranking Problem. In Proceedings of the 15th European Conference on Symbolic and Quantitative Approaches with Uncertainty, Belgrade, Serbia, 18–20 September 2019; pp. 351–362.
24. Kemeny, J.; Snell, J. *Mathematical Models in the Social Sciences*; MIT Press: Cambridge, MA, USA, 1972.

25. Kendall, M.G. *Rank Correlation Methods*; Griffin: Hong Kong, China, 1948.
26. Borda, J. *Memoire Sur Les Elections au Scrutin*; Histoire de l'Academie Royal des Sciences: Paris, France, 1770.
27. Kendall, M.G. A New Measure of Rank Correlation. *Biometrika* **1938**, *30*, 81–93. [CrossRef]
28. Irurozk, E.; Calvo, B.; Lozano, J.A. PerMallows: An R Package for Mallows and Generalized Mallows Models. *J. Stat. Softw.* **2016**, *71*, 1–30.
29. Ali, A.; Meilă, M. Experiments with Kemeny ranking: What works when? *Math. Soc. Sci.* **2012**, *64*, 28–40. [CrossRef]
30. Bielza, C.; Larrañaga, P. Discrete Bayesian Network Classifiers: A Survey. *ACM Comput. Surv.* **2014**, *47*. [CrossRef]
31. Stutz, J.; Cheeseman, P. Autoclass—A Bayesian Approach to Classification. In *Maximum Entropy and Bayesian Methods*; Skilling, J., Sibisi, S., Eds.; Springer: Berlin/Heidelberg, Germany, 1996; pp. 117–126.
32. Stewart, B. Improving performance of naive bayes classifier by including hidden variables. In Proceedings of the 11th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems: Methodology and Tools in Knowledge-Based Systems, Castellon, Spain, 1–4 June 1998; pp. 272–280.
33. Flores, M.J.; Gámez, J.A.; Martínez, A.M.; Puerta, J.M. HODE: Hidden One-Dependence Estimator. In Proceedings of the 15th European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty, Belgrade, Serbia, 18–20 September 2009; pp. 481–492.
34. Langseth, H.; Nielsen, T.D. Classification using Hierarchical Naïve Bayes models. *Mach. Learn.* **2006**, *63*, 135–159. [CrossRef]
35. Jiang, L.; Zhang, H.; Zhihua, C. A Novel Bayes Model: Hidden Naive Bayes. *IEEE Trans. Knowl. Data Eng.* **2009**, *21*, 1361–1371. [CrossRef]
36. Lowd, D.; Domingos, P. Naive Bayes models for probability estimation. In Proceedings of the 22nd International Conference on Machine Learning, Bonn, Germany, 7–11 August 2005; pp. 529–536.
37. Dempster, A.P.; Laird, N.M.; Rubin, D.B. Maximum Likelihood from Incomplete Data Via the EM Algorithm. *J. R. Stat. Soc. Ser. B Methodol.* **1997**, *39*, 1–22.
38. Wu, X.; Kumar, V. *The Top Ten Algorithms in Data Mining*; Chapman and Hall: London, UK, 2009.
39. Flores, M.; Gámez, J.A.; Martínez, A. Supervised Classification with Bayesian Networks: A Review on Models and Applications. In *Intelligent Data Analysis for Real-Life Applications: Theory and Practice*; IGI Global: Hershey, PA, USA, 2012; pp. 72–102.
40. Salmerón, A.; Rumí, R.; Langseth, H.; Nielsen, T.D.; Madsen, A.L. A Review of Inference Algorithms for Hybrid Bayesian Networks. *J. Artif. Intell. Res.* **2018**, *62*, 799–828. [CrossRef]
41. Friedman, N. The Bayesian Structural EM Algorithm. In Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence, Madison, WI, USA, 24–26 July 1998; pp. 129–138.
42. Reynolds, D., Gaussian Mixture Models. In *Encyclopedia of Biometrics*; Springer: Berlin/Heidelberg, Germany, 2009; pp. 659–663.
43. McLachlan, G.; Krishnan, T. *The EM algorithm and Extensions*, 2nd ed.; Wiley: Hoboken, NJ, USA, 2008.
44. Fayyad, U.M.; Irani, K.B. Multi-interval discretization of continuous-valued attributes for classification learning. *Artif. Intell.* **1993**, *13*, 1022–1027.
45. Demšar, J. Statistical Comparisons of Classifiers over Multiple Data Sets. *J. Mach. Learn. Res.* **2006**, *7*, 1–30.
46. García, S.; Herrera, F. An Extension on "Statistical Comparisons of Classifiers over Multiple Data Sets" for all Pairwise Comparisons. *J. Mach. Learn. Res.* **2008**, *9*, 2677–2694.
47. Arias, J.; Cózar, J. ExReport: Fast, Reliable and Elegant Reproducible Research. CRAN. 2016. Available online: https://cran.r-project.org/web/packages/exreport (accessed on 21 March 2021).
48. Friedman, M. A comparison of alternative tests of significance for the problem of m rankings. *Ann. Math. Stat.* **1940**, *11*, 86–92. [CrossRef]
49. Holm, S. A Simple Sequentially Rejective Multiple Test Procedure. *Scand. J. Stat.* **1979**, *6*, 65–70.
50. Pérez-Bernabé, I.; Maldonado, A.D.; Salmerón, A.; Nielsen, T.D. MoTBFs: An R Package for Learning Hybrid Bayesian Networks Using Mixtures of Truncated Basis Functions. *R J.* **2020**, *12*, 321. [CrossRef]
51. Alfaro, J.C.; Aledo, J.A.; Gámez, J.A. Averaging-Based Ensemble Methods for the Partial Label Ranking Problem. In Proceedings of the 15th International Conference on Hybrid Artificial Intelligence Systems, Gijón, Spain, 11–13 November 2020; pp. 410–423.
52. Alfaro, J.C.; Aledo, J.A.; Gámez, J.A. Learning decision trees for the partial label ranking problem. *Int. J. Intell. Syst.* **2021**, *36*, 890–918. [CrossRef]