LINFO2364 - Project 3 - Group 48 Bayesian networks for predicting missing values in tabular data

Bourez Nicolas
UCLouvain
Ecole polytechnique de Louvain-la-Neuve
Louvain-la-Neuve, Belgium
28462000

Demaret Dimitri
UCLouvain
Ecole polytechnique de Louvain-la-Neuve
Louvain-la-Neuve, Belgium
38502000

I. Introduction

In this report, we explore the use of Bayesian networks to address the challenge of missing value imputation in tabular datasets. We present a detailed examination of the underlying methods involved in Bayesian inference, parameter learning, and structure learning, demonstrating their application through a series of results. Our goal is to assess the efficiency and accuracy of Bayesian networks in predicting missing values across different datasets.

II. Methods

A. Bayesian Network Inference

The main task of the project is to find the most appropriate value for missing ones in a given tabular dataset. To find this value, we implemented a function `mostProbableValue(distribution)` which, given a dictionary of probabilities for a the different value a variable might take, finds the maximum and returns the value which corresponds to this maximum probability.

1) Imputation for 1 missing value: Four one value and given a Bayesian Network, the imputation is quite straightforward: we simply have to find the value of the variable that has the biggest probability, given evidence. In our case, this evidence concerns all the other variables. From this big evidence, we simply have to take the values of the parents and take the probability present in the cpt table.

2) Imputation for 2 missing values: For two values, the case is a little bit more complex. We have to find 2 values with the highest product of partial probabilities. This can be done by iterating on the different values of the two variables, with a double `for loop`. For more than two variables, we would have to add other `for loops`. We also have to deal with the case that one of the two variables is the parent of the other. In this case, we just made an imputation on the parent variable, and then input for the second variable knowing the value of the first one. We return the combination of value with the highest probability.

B. Parameter Learning

We created a function `estimate_parameters` that, given a Bayesian network and a training file under the CSV format, will fit the cpt tables of all the variables.

To do so, we created a loop over each variable of our Bayesian network. Each time, we store its parents (or an empty list if there is no parent for the variable).

Then, our function employs `defaultdict` and `Counter` from the `collections` module to manage nested data and count variable occurrences efficiently. Specifically, `defaultdict` is used to create a dictionary where each key is a tuple of parent values, and each value is a `Counter` object. This setup allows for dynamic key creation and initialization with a `Counter` that contains all possible outcomes of the variable initialized to zero. This is particularly useful for handling cases where certain outcomes do not appear in the dataset but must still be accounted for in probability calculations.

The `Counter` class is utilized to count the occurrences of each variable value under different parent conditions. This allows us to compute conditional probabilities in Bayesian networks, where the frequency of each outcome must be known. The use of `Counter` simplifies the process of updating counts and facilitates the computation of probabilities.

We also incorporated Laplace smoothing (with the same parameters as in the directive PDF) to handle zero-frequency cases effectively.

C. Structure Learning

To build a structure learning algorithm for a Bayesian Network, we have been inspired by the greedy algorithm presented in the slides of the course, and the already existing package `pgmpy` [1]. Our algorithm is directly implemented in the Bayesian Network class and takes three parameters :

- **A number of iterations** `max_iterations`. This parameter represents the number of operations our algorithm will perform. We will present these operations later on. We can see this parameter as a bound on the number of edges present in the network.
- **A tolerance of changing** `tol`. If the network is not sufficiently changing according to a certain score after a given operation, the algorithm stops.
- **A score metric** `score`. We implemented three different scoring functions to evaluate the structure of our network, namely the AIC, the BIC, and the K2 score.

1) The operations: At each iteration, our algorithm is capable of *add, remove* or *reverse* an edge to the network. To select the best operation, we compute the score between a given variable with its old parents and the score between this variable and its new parents. To compute the new parents, we just add the other variable at the extremity of the edge to the parents of the first variable. If the score with the new parents is higher than for the old parents, this means we are decreasing the global likelihood of the network by doing this operation.

We follow the same kind of procedure (but changing some details) for the 2 others operations, and keep the highest *changing* score from the three operations. We then apply the chosen operation to our building network.

To avoid some *cycling* operations, i.e. add and then remove the same edge periodically, we stored in a list all the operations we don't want to do anymore.

2) The scoring functions: We used three different scoring functions to evaluate our structure according to our data, inspired by the paper [3]:

- **K2 score**. It is a well-known evaluation measure for learning Bayesian networks from data. It is derived by assuming uniform prior distributions on the values of an attribute for each possible instantiation of its parent attributes. This assumption introduces a tendency to select simpler network structures. The implementation of the score starts from the formula of the BD score [2] :

$$\text{BD}(B,\mathcal{D}) = \log(P(B)) + \sum_{i=1}^{n}\sum_{j=1}^{q_i}\left(\log\left(\frac{\Gamma(N'_{ij})}{\Gamma(N_{ij}+N'_{ij})}\right)\right.$$
$$\left.+\sum_{k=1}^{r_i}\log\left(\frac{\Gamma(N_{ijk}+N'_{ijk})}{\Gamma(N'_{ijk})}\right)\right)$$

  Setting the Dirichlet priors N'_{ijk} to 1 gives the K2 score.[1]
- **the Akaike Information Criterion (AIC)**.

$$\text{AIC}(B|\mathcal{D}) = \log(P(\mathcal{D})) - |B|$$

- **the Bayesian Information Criterion (BIC)**.

$$\text{BIC}(B|\mathcal{D}) = \log(P(\mathcal{D})) - \frac{1}{2}\log(N)|B|$$

[1]Considering the complexity of implementation of the formula, we had been highly inspired by the function implemented in the package `pgmpy`

Where:
- $\log(P(\mathcal{D}))$ is the log-likelihood of the data $\mathcal{D}$ given the Bayesian network B.
- $|B|$ is the number of parameters in the Bayesian network.
- N is the number of data points.

III. Experiments and Results

A. Dataset Description

As always, let us start by looking at the data we have at our disposal. To perform our results analysis, we used the 4 following datasets: `Alarms`,`Asian`, `Sachs`, and `Waters`.

Let's start with the repartition of those datasets into the number of variables and the number of rows. We can see that there is nearly the same amount of rows to be analyzed in each training data, close to 1600 (see Figure 1). For the number of variables, the `Alarms` and the `Waters` have much more variables (see Figure 2).

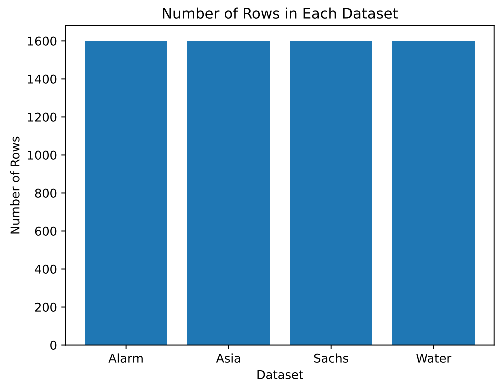

Fig. 1. Number of rows per training set

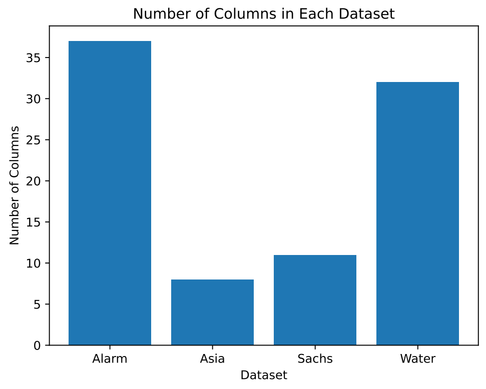

Fig. 2. Number of variables per set

One more interesting feature to analyze is the number of missing values that we will need to fill in. It can be seen in the following Figure 3.

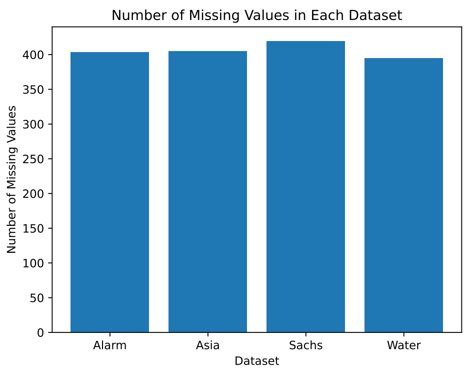

Fig. 3. Number of missing values

B. *Results Analysis*

To compute the accuracy of our results, we create a function `comparison_completed_files` which takes in argument the `test_file.csv` provided with the directives, the `test_missing.csv` and the file that we filled with our missing value imputation function. The accuracy is computed only by comparing the values that were missing initially (those who were already there are not taken into account or we would have had too high results).

Let us first look at the results obtained with the K2 score. For that, we plotted the results in function of the `max_iterations` parameter that we defined earlier. We can observe the results in Figure 4. We observe that we get accuracies over 75% for three datasets, but the `Sachs` (remember that it has a low number of variables) has an accuracy that does not reach 65%. We also notice that the other dataset with low number of variables, the `asian` dataset has an accuracy highly increasing while we increase the `max_iterations` parameter (we will see later for bigger values of this parameter). Let us now compare with the other scores.

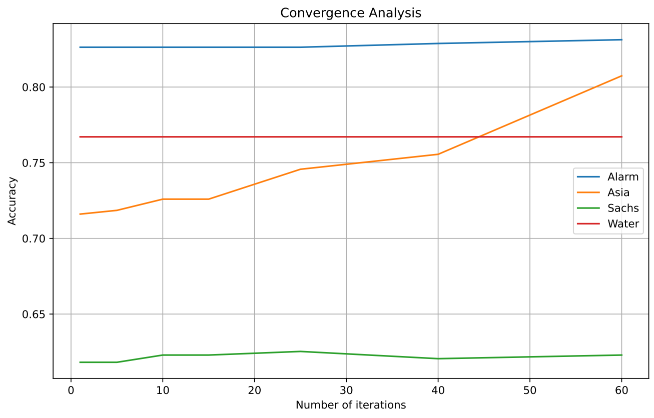

Fig. 4. Accuracy using K2 score

For the second score, we can see the results with the BIC score in Figure 5, which are quite similar to the one we obtained using K2 score.

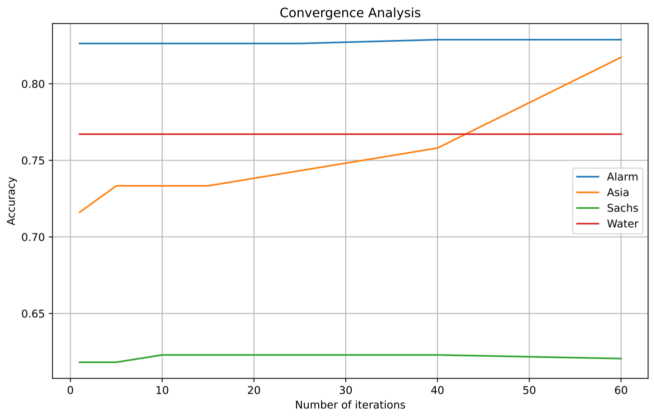

Fig. 5. Accuracy using BIC score

Finally, we can observe the results using AIC score in Figure 6. Once more they are very similar.

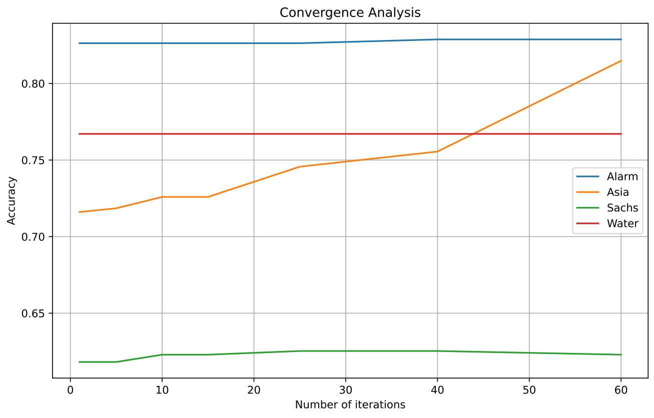

Fig. 6. Accuracy using AIC score

We asked ourselves what is going to happen with higher values of `max_iterations` so we computed the accuracy when setting this parameter 100 and 200. We can see the results plotted for AIC in Figure 7.

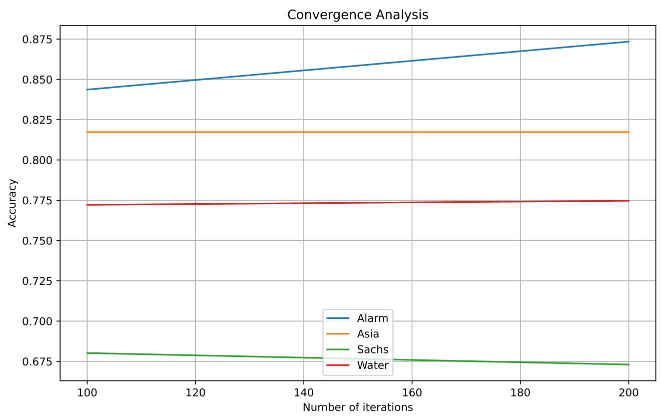

Fig. 7. Accuracy using AIC score for big number of iterations

The results seem to reach a ceiling for three of the four datasets, but we have the `Alarms` dataset which has an accuracy still in progress and even reaching an accuracy of 87.5% with 200 iterations. We can also see that the Sacha tends to overfit, so we need stop the algorithm at a certain number of iteration if we want to best accuracy as possible.

IV. Conclusion

The experiments conducted in this project show that Bayesian networks can be considered as a powerful tool for imputing missing values in tabular data. Our findings reveal that the performance of Bayesian networks is significantly influenced by the structure of the data and the specific methods used for parameter and structure learning. We tried various scoring functions—K2, AIC, even though it did not seem to change much the results. While the Bayesian approach demonstrates high accuracy in datasets with numerous variables, challenges remain in handling datasets with fewer variables or higher complexity in missing patterns.

V. References

References

[1] Ankur Ankan and Abinash Panda. pgmpy: Probabilistic graphical models using python. In *Proceedings of the Python in Science Conference*, SciPy. SciPy, 2015.

[2] Christian Borgelt and Rudolf Kruse. An empirical investigation of the k2 metric. In Salem Benferhat and Philippe Besnard, editors, *Symbolic and Quantitative Approaches to Reasoning with Uncertainty*, pages 240–251, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.

[3] Alexandra M. Carvalho and Tópicos Avançados. Scoring functions for learning bayesian networks. 2009.