Handling missing values in apriori algorithm

Elad Meged, Sharon Yankovich

March 19, 2022

Abstract

Association rule learning is a data mining method for discovering interesting relations between variables in a dataset. In practical scenarios, many real world databases have missing values. Since Agrawal et al. suggestion of a fast algorithm to mine association rules, there have been multiple attemps at dealing with such missing values. In pattern mining, the significance of the analysis depends heavily on the accuracy of the database, compared to predictive models where missing values can be learned. Moreover, ignoring missing data might introduce bias into the models being evaludated and lead to inaccurate results. In this paper, we propose a combination of machine learning techniques to handle rule mining in the presence of missing values. We'll define performance metrics to measure the goodness of the gathered rules and show an uplift of up to 20% in those metrics using our solution compared to the apriori algorithm.

Problem Description

In assocation learning, we are dealing with a set of items $\mathcal{I}$. A *transaction* is a $\emptyset \neq T \subseteq \mathcal{I}$, and a databse D is a set of transactions. Given D, we call a rule of the form $A \Rightarrow B$ an *association rule* if (a) $A, B \subset \mathcal{I}$, (b) $A, B \neq \emptyset$, and (c) $A \cap B = \emptyset$. Our goal is to find such rules, and test how good they are (with "good" defined later).

Example. Given $\mathcal{I} = \{\text{Beer}, \text{Eggs}, \text{Water}\}$, a rule can be, for example: $\{\underbrace{\text{Beer}, \text{Eggs}}_{A}\} \implies \{\underbrace{\text{Water}}_{B}\}$

The aforementioned algorithm, called the apriori algorithm, is the basis with which we compare. This algorithm employs an iterative, level-wise search, where k-itemsets are used to explore $(k+1)$-itemsets. This algorithm has be thoroughly tested and it is known to perform well under certain circumstances. One such assumption is that the dataset we're working with is full, meaning it contains no missing values.

In our dataset, a *missing value* hides to value of an attribute. It might be missing for various reasons, such as the value being too large, or it being lost, and generaly it is a reasonable assumption that *some* data will be lost. Using the literture of the subject, we'll acknowledge three types of missing values:

(1) Missing completely at random (MCAR) - The missing value has no dependency on other variables.
(2) Missing at random (MAR) - The missing value depends on other variables.
(3) Missing not at random (MNAR) - The missing value depends on other missing values.

In our work, we assume the missing values are of type MAR, meaning we can deduce the remaining spots

in a complex manner using the data available to us. In this case, we could not use the current rules to infer the missing values variables, because they are dependent on each other. Table 1 presents a simplified scenario of missing values, where some values in the table are presented with a ? sign. Our goal will be, given a dataset with missing values, to recover those missing values, and then use the apriori algorithm on the recovered dataset. We claim the recovery of the lost values can generally increase the performace of the apriori algorithm, improving the association-learning based data mining used in the DS pipeline.

Table 1: Missing values in dataset

	x_1	x_2	x_3
y_1	0.1	0.158	8
y_2	0.12	1.3	?
y_3	?	2.4	11

Our estimation method is the *Jaccard index.* We define it as follows:

Definition 1. Given two finite sets A, B, we define their Jaccard index $J(A, B)$ to be $J(A, B) = \frac{|A \cap B|}{|A \cup B|}$.

We use this definition as a metric of how good a deduction of rules is. For a fixed set C, we define the function $S_C(U) = J(C, U)$ to be our *Jaccard score.* Notice that for all sets U we have $0 \leq S_C(U) \leq 1$. We say that the score of A is better than the score of B given C if $S_C(A) > S_C(B)$ and that is it worse than B if $S_C(A) < S_C(B)$.

Solution overview

Data handling

We focues on three datasets: Iris, Hamberman and Wine. For each dataset, we denote by $\mathcal{A}$ its set of attributes. Next, we denote by $\mathcal{D}$ the set of rows of the dataset.

Definition 2. Given $\mathcal{A}, \mathcal{D}$, the *set of transactions* is the set $\mathcal{A} \times \mathcal{D}$.

For each of the listed datasets, we calculate its transactions during the *data loading phase.* In particular, this phase generates for each dataset a python dictionary (or a set, really) of its data, attributes, and transactions. It is worth noting that these datasets are full, meaning they contain no missing values. Our assumption is that (simillar to how a communication channel is defined in an information-theoretic setting) there's a distribution for data loss. We denote by p_L a probability with which each cell $c \in d$ (where $d \in D$) is lost. We call this the *dropout phase.*

Example. For example, the transition $\boxed{1.2 \mid 5 \mid 2} \underset{p_L}{\Longrightarrow} \boxed{1.2 \mid 5 \mid ?}$ is a simulation of a dropout phase.

Next, we perform the apriori algorithm two times, on the initial data, and on the data after the dropout phase. The apriori algorithm makes use of two notions, support and confidence.

Definition 3. Given a transaction set D and an association rule $r = A \Rightarrow B$ the support of rule r is defined by:

$$s(r) = support(r) = \Pr[A \cup B] = \frac{\sum_{T \in D}[I_{A \cup B \in T}]}{|D|}$$

also,

Definition 4. Given a transaction set D and an association rule $r = A \Rightarrow B$ the confidence of rule r is defined by:

$$c(r) = confidence(r) = \Pr[B \mid A] = \frac{s(A \cup B)}{s(A)}$$

Now, in the calculation of the apriori algorithm, we use two values: θ_s and θ_c. These are *minimal support* and *minimal confidence* respecrively. Our algorithm uses $\theta_s = 0.01, \theta_c = 0.8$ as these are generally good values that tend to give good results (as seen in class). We denote by F the result of the run of the apriori algorithm on the transactions of the full dataset (before dropout), using θ_s and θ_c. F is essentially a set of rules produced by the algorithm. We also denote by M the same thing, only now using the set of transactions of the dataset after the dropout phase.

These are the basic building blocks we use in our actual architecture to perform the missing value restoration and analysis of the results. Next, we'll describe the model of the ML pipeline we use to generate the missing values.

ML Architecture

We use five models to test against each other:

XGBoost Regressor

XGBoost is an efficient implementation of gradient boosting that can be used for regression predictive modeling. It was initially developed by Tianqi Chen and was described by Chen and Carlos Guestrin in their 2016 paper titled "XGBoost: A Scalable Tree Boosting System."

XGBoost provides a parallel tree boosting (also known as GBDT, GBM) that solve many data science problems in a fast and accurate way. The XGBoost model often achieves a higher accuracy than a decision tree, but it sacrifices the inherent interpretability of decision trees. For example, following the path that a decision tree takes to make its decision is trivial and self-explained, but following the paths of hundreds or thousands of trees is much harder.

SGD Regressor

Stochastic gradient descent (SGD) is an iterative method for optimizing an objective function with suitable smoothness properties (e.g. differentiable or subdifferentiable). It can be regarded as a stochastic approximation of gradient descent optimization, since it replaces the actual gradient (calculated from the entire data set) by an estimate thereof (calculated from a randomly selected subset of the data). Especially in high-dimensional optimization problems this reduces the computational burden, achieving faster iterations in trade for a lower convergence rate.

K Nearest Neighbors Regressor

In statistics, the k-nearest neighbors algorithm (k-NN) is a non-parametric supervised learning method first developed by Evelyn Fix and Joseph Hodges in 1951, and later expanded by Thomas Cover. It is used for classification and regression. In both cases, the input consists of the k closest training examples in a data set. The output depends on whether k-NN is used for classification or regression

In k-NN regression, the output is the property value for the object. This value is the average of the values of k nearest neighbors.

Linear Regression

In statistics, linear regression is a linear approach for modelling the relationship between a scalar response and one or more explanatory variables (also known as dependent and independent variables). The case of one explanatory variable is called simple linear regression; for more than one, the process is called multiple linear regression. This term is distinct from multivariate linear regression, where multiple correlated dependent variables are predicted, rather than a single scalar variable.

In linear regression, the relationships are modeled using linear predictor functions whose unknown model parameters are estimated from the data. Such models are called linear models.

Multi Layer Peceptron Regressor

A multilayer perceptron (MLP) is a class of feedforward artificial neural network (ANN). The term MLP is used ambiguously, sometimes loosely to mean any feedforward ANN, sometimes strictly to refer to networks composed of multiple layers of perceptrons (with threshold activation).

An MLP consists of at least three layers of nodes: an input layer, a hidden layer and an output layer. Except for the input nodes, each node is a neuron that uses a nonlinear activation function. MLP utilizes a supervised learning technique called backpropagation for training. Its multiple layers and non-linear activation distinguish MLP from a linear perceptron. It can distinguish data that is not linearly separable.

Estimation phase

For each model, we issue an estimation phase using the incomplete dataset (i.e., the dataset with missing values following the dropout phase). Our mission is to estimate the missing values and correct the dataset in order to restore the full dataset. We divide the data in the dataset into two groups: complete rows and incomplete rows.

Definition 5. A complete row in a dataset is a row that contains no missing values. Consequently, an incomplete row contains at least one missing value.

Now, we can define new datasets: The complete dataset, and the incomplete dataset. The complete dataset is a dataset composed only of the complete rows of the original dataset. Likewise, the incomplete dataset contains only incomplete rows. Let C be the complete dataset and I be the incomplete one and M be a ML model. We define the following algorithm for estimation:

Estimation algorithm(C, I, M)**:**
1. let $I' = I$
2. for each column i in the I do:
2.1. if it has missing values:
2.1.1. denote by `x_traing` C without the ith column
2.1.2. denote by `y_train` the ith column in C
2.1.3. train the model M on (`x_train, y_train`)
2.1.4. let `x_test` be I without its ith column
2.1.5. predict the missing values of `x_test`

2.1.6. for each filled missing cell of the ith column of `x_test` do:
2.1.6.1. cluster it with `find_nearest`
2.1.6.2. add estimated cell to I'
3. return I'

Notice we make a use a of an algorithm called *find nearest*. This algorithm is used for clustering of the result. The data we're dealing with is mostly continious, so we've used regression techniques to estimate the value of the missing values. However, in the apriori algorithm we need a finite set of items, so the approach we've used is to cluster the regression results using the L^1 norm.

The algorithm takes an array `y_train` of the training set we've defined earlier, and a prediction value pr. We cluster accordin to:

$$\texttt{find\_nearest(y\_train,}pr\texttt{)} = \underset{y \in \texttt{y\_train}}{\operatorname{argmin}} \left| y - pr \right|$$

Finally, we let E denote the dataset returned by the estimation phase. Combining everything we've seen, we calculate both $J(C, I)$ and $J(C, E)$ and compare them.

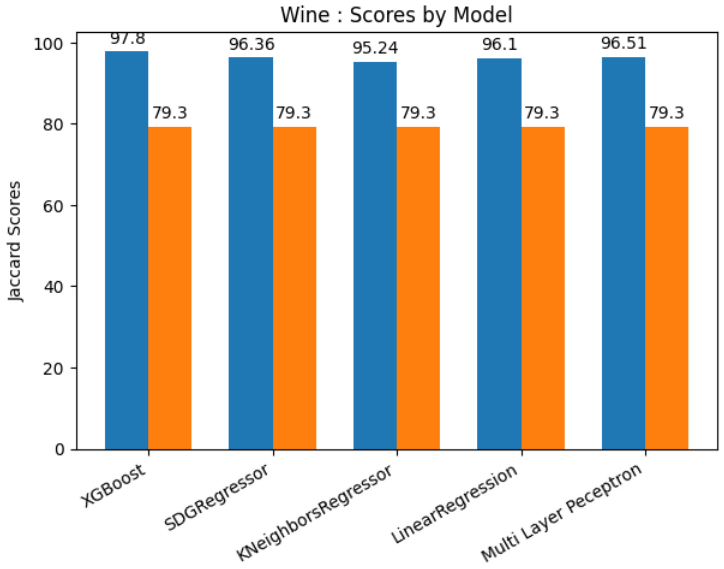
Experimental evaluation - Results and Analysis

As described in Measurement section, we have measured the dateset improvement using Jaccard Score of the apriori rules. For each model, we have calculated $J(C, I)$ and the $J(C, E)$ to measure and compare the model evaluation. In the following charts, we view the performance of our model compared to the naive use of the apriori algorithm using the methods above.

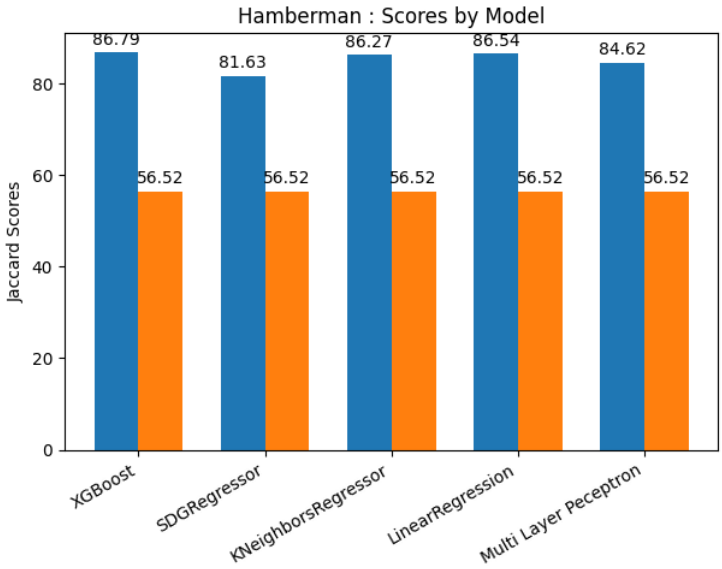
The blue bars represent the estimated values Jaccard score, while the orange bars represent the missing values Jaccard score. We compare them to see the performance upflift we've gotten from our missing value evaluation.

Per-Dataset Results

(a) Iris dataset performance by model



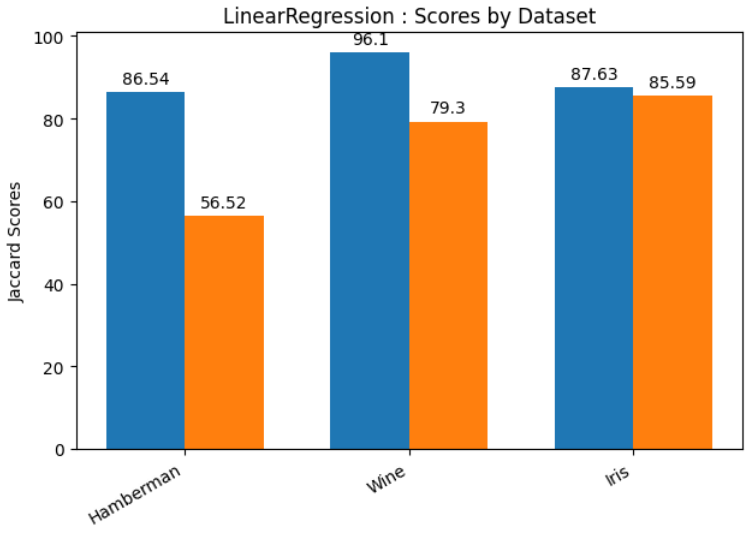
(b) Wine dataset performance by model



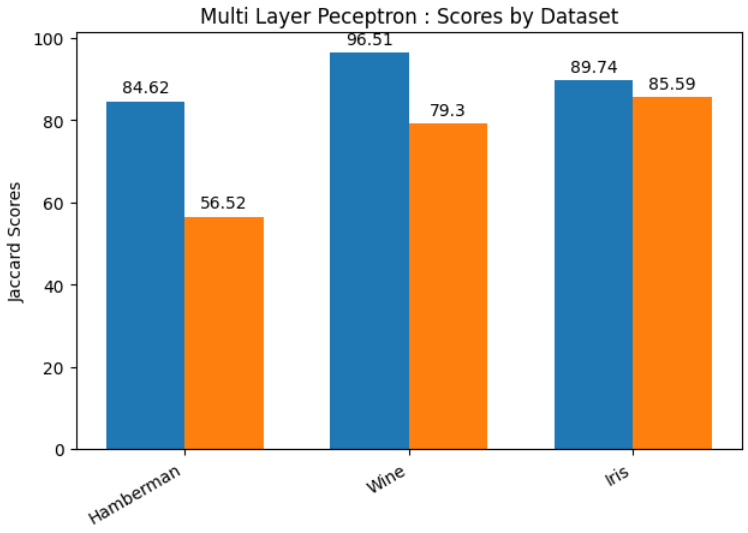
(c) Hamberman dataset performance by model

Figure 1: Performance by dataset

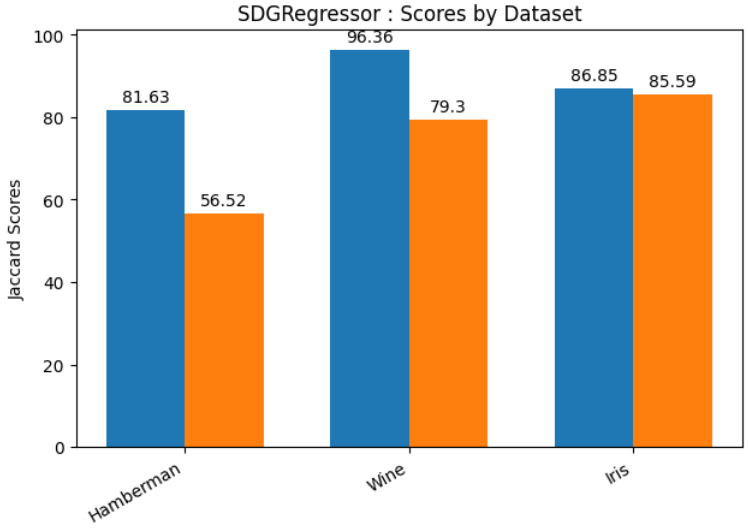
Per-model Results



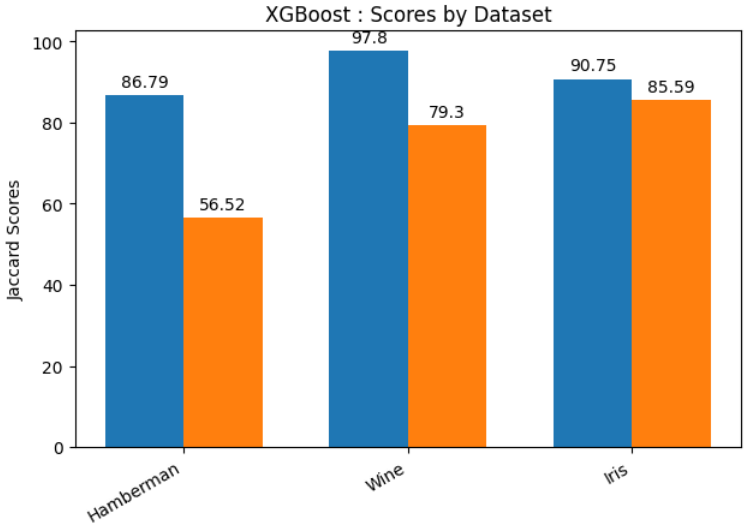
(a) LinearRegression performance



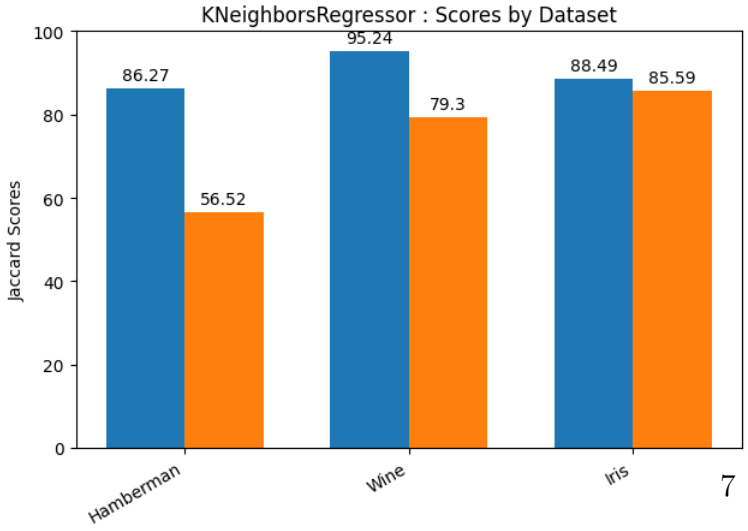
(b) Multilayer Perceptron performance



(c) SDGRegressor performance



(d) XGBoost performance



(e) KNRegressor performance

Figure 2: Performance by model

Related work

There are three main related works we've read and used as literature for our project. First, there's "Treatment of Missing Values" (Arnaud Ragel and Bruno Cremilleux, 1998). This paper deals with the issue of missing values in the context of the apriori algorithm. In this paper, too, the authors provide a method of handling the missing values. However, their approach is purely statistical and there's no learning involved. Our approach uses much more modern tools in order to attack the same problem.

The two following works, "A hybrid method for imputation of missing values using optimized fuzzy c-means with support vector regression and a genetic algorithm" (Ibrahim Berkan Aydilek and Ahmet Arslan, 2013) and https://www.analyticsvidhya.com/blog/2021/05/dealing-with-missing-values-in-python-a-complete-guide, do use modern techniques to fill missing values.

However, these works only concern themselves with filling those missing values. Our work, in contrast, tries to better the association rule mining on the dataset. Thus, we define a measurement of the goodness of the apriori algorithm outpus, and compare the performance pre and post our treatment of missing values. Our goal, ultimately, is to get better performance out of the apriori algorithm - not to restore a dataset.

Conclusion

In our work, we tried to bridge the gap between the apriori algorithm, and more modern ML advances. We saw the problem of missing values as a problem in the DS pipeline as a whole and tried to address it. As seen in the results, the performance using our method is generally higher than using the apriori as is. This makes us believe that the possiblity of using recent techniques can elevate and solve many pitfalls found in classic statistical/mining methods, and that we've only scratched the surface when it comes to applications of methods like ours for mining, data cleaning, data recovery, and data automation tasks.