Evaluating Classification Feasibility Using Functional Dependencies

Marie Le Guilly[1], Jean-Marc Petit[1], and Vasile-Marian Scuturici[1]

Univ Lyon, INSA Lyon, LIRIS, UMR 5205 CNRS, Villeurbanne, France

Abstract. With the vast amount of available tools and libraries for data science, it has never been easier to make use of classification algorithms: a few lines of code are enough to apply dozens of algorithms on any dataset. It is therefore "super easy" for data scientists to produce machine learning (ML) models in a very limited time. On the counterpart, domain experts may have the impression that such ML models are just a black box, almost magic, that would work on any dataset without really understanding why. For this reason, related to interpretability of machine learning, there is an urgent need to reconcile domain experts with ML models and to identify at the right level of abstraction, techniques to get them implied in the ML model construction.
In this paper, we address this notion of trusting ML models by using data dependencies. We argue that functional dependencies characterize the existence of a function that a classification algorithm seeks to define. From this simple yet crucial remark, we have made several contributions. First, we show how functional dependencies can give a tight upper bound for the classification's accuracy, leading to impressive experimental results on UCI datasets with state-of-the art ML methods. Second, we point out how to generate very difficult synthetic datasets for classification, showing evidence about the fact that for some datasets, it does not make any sense to use ML methods. Third, we propose a practical and scalable solution to assess the existence of a function before applying ML techniques, allowing to take into account real life data and to keep domain experts in the loop.

Keywords: Functional Dependencies · Classification · Feasibility

1 Introduction

As the number of machine learning libraries keeps increasing, with more and more tools and technologies, it is getting easier to apply classification models on any dataset, with a very limited number of steps. Indeed, for a data scientist, it only takes a few lines of code to train and test a model in her data, and to get first results in a few minutes. If this is very practical, it can also be dangerous: it is possible to try and classify on data for which it might not make any sense, or for which much more cleaning and preprocessing is necessary. Of course, in such cases, the scores used to evaluate the model will likely be pretty low, indicating

to the data scientist that there is room for improvement, regarding either the data or the algorithm's parameters.

Another problem of classification algorithm is its black box aspect: when applied on a dataset, the domain expert will not necessarily trust them, as it is not always possible to explain simply why it works or the prediction it produces. For this reason's models interpretability is an extremely important topic [8, 16]. Some models are interpretable by nature, such as decision tree, and there are many ongoing work to propose models that can be easily explicable [46, 48]. Having domain experts understanding why the model is adapted to their data, and then trust it, is crucial to build new artificial intelligence applications.

In this paper, we address this notion of trusting ML models by using data dependencies. We argue that *functional dependencies characterize the existence of a function that a classification algorithm seeks to define.* Rather than focusing on the model itself, we propose to explain the limitation of the main input of a classification problem, i.e. the classification dataset, to show why the accuracy of any classifier is limited by the available data, while keeping domain experts in the loop. Indeed, the existence of a function is much easier to understand than the function itself and turns out to be a very convenient concept for data scientists to get closer to domain experts.

On the one hand, functional dependencies are a key concept, at the foundation of the theory for relational database design (see [2]), data cleaning [5] or for query optimization, to mention a few. Supervised classification, on the other hand, is a traditional problem in machine learning: it seeks to find a model that can predict the class of a sample described by its value over some given attributes. Many algorithms have been developed to build such models (see [21] for an overview), from simple decision trees to the trendy deep neural networks. These algorithms have gained in popularity this last few years, especially thanks to the volume of data that is now available to data scientists. Classification problems arise in very different areas, and models are being produced for an even wider range of applications, from cancer prediction to targeted advertising.

Functional dependencies on one side, and supervised classification on the other, therefore appear as two problems that do not seem to have much in common. For instance, they do not look at data in the same way: for functional dependencies, the values themselves are not important, only their comparabilities matter, while values are crucial for learning algorithms. Nevertheless, these approaches turn out to be complementary, as shown in the following general observation:

> Given a dataset r over $\{A_1, \ldots, A_n, C\}$ where C C values represent the class to be predicted, classification algorithms seek to find out a function to predict an output (C value) based on a given input ($A_1, \ldots, A_n$ values) whereas the satisfaction of the functional dependency $A_1, \ldots, A_n \rightarrow C$ in r expresses nothing else than the existence of that function.

Between these two notions, one is clearly easier than the other. This observation raises a simple question: *Does it make sense to look for a function when the data themselves show that no function exists ?*

Indeed, there exist many machine learning libraries and tools that allow to build dozens of models, and therefore look for a function... that does not even exist.

As a result, in a classification setting, studying the existence of a functional dependency between the attributes and the class of the dataset could produce meaningful information regarding the possible performances of the classifier. No matter how complicated the function to find out is, looking at the data dependencies allow to simply validate its existence. It can also identify relevant counterexamples, i.e. tuples in the dataset that will cause a classifier to fail when predicting an output for it, because several outputs are possible for the exact same input. More precisely, we propose to qualify the existence of the functional dependency using well-known metrics, such that an upper bound for the classifier accuracy can be given. This can then be used for data analysts to better explain to domain experts what is going on with their data, before building a model on top of it. Using the score, they can make an informed decision, by being aware of some limitations of the model they are trying to build. Moreover, it can assist them in deciding whether or not they should jump into the learning phase, or if they should spend more time to do more data cleaning, add other attributes, etc.

Example 1. Let's take a small dataset from table 1 as an example. This is the dataset about passengers of the famous Titanic, with their ticket class (first or second), their age range (child or adult), their gender, and whether or not they survived. The purpose of this problem is to predict if a passenger has survived or not. Such an analysis can then be used to determine if some passengers were more likely to survive than the other.

In this dataset, the available attributes are not enough to determine the class. For example, tuples t_2 and t_5 both concern male children in second class, however one survived while the other did not. Similarly, the two adult males in first class from tuples t_7 and t_8 had two different outcomes. Whatever the classifier, it will irremediably misclassify at least one of them.

id	Ticket	Age	Gender	Survived
t_1	1st	Child	Female	no
t_2	2nd	Child	Male	yes
t_3	1st	Adult	Male	no
t_4	2nd	Adult	Female	yes
t_5	2nd	Child	Male	no
t_6	2nd	Child	Male	yes
t_7	1st	Adult	Male	no
t_8	1st	Adult	Male	yes
t_9	2nd	Child	Male	yes
t_{10}	1st	Child	Female	yes

Table 1: Toy dataset: Titanic relation

This very simple example shows how the satisfaction of the functional dependency between the attributes (Ticket, Age, Gender) and the class (Survived) in a classification dataset highlights the limits a classifier reaches on a dataset: according to the measure G_3 [23] (see section 3), the accuracy of classifiers on this dataset can not be more than 70%. In addition, the counterexamples do not only highlight why the classification performance will not be above a certain value, but also what are the tuples that cause problems.

To summarize, this paper is based on a dramatically simple but powerful observation, making a clear relationship between supervised classification and functional dependencies, especially on the interest of first studying the existence of a function before applying machine learning techniques. Thus, we propose the following contributions:

1. We give a tight upper bound of classifier's accuracy based on the G_3 measure [24] for functional dependencies. An algorithm is given to compute the upper bound, and experimentations are provided on datasets from the UCI repository. This is a practical solution to give understandability to the learning process, by quantifying whether or not it makes sense to use machine learning techniques on the considered dataset.
2. An algorithm to generate difficult synthetic classification datasets with a predictable upper bound for accuracy, whatever the classification algorithms. As far as we know, this is the first contribution to generate synthetic datasets so that their classification accuracy can be as hard as desired.
3. A practical and scalable solution to deal with real life datasets, and assess the existence of a function. The solution relies on a dataset reduction technique and crisp functional dependencies, delivering complex and meaningful counterexamples, crucial to keep domain experts in the loop. Experiments conducted on real-life datasets point out the scalability of our technique.

This work contributes to bridge the gap between data dependencies and machine learning, a timely and active research trend, see for example [35, 1, 38].

Section 2 introduces the necessary preliminary notions on functional dependencies and supervised classification. Section 3 explains the thought process that goes from the existence of a function using functional dependencies to finding the function with a classification algorithm. Section 4 exposes the generation of difficult datasets and the corresponding tests. Section 5 explains how to use counterexamples in practical settings, with our practical and scalable solution for real life datasets. Finally section 6 describes the related work, before concluding in section 7.

2 Preliminaries

We first recall basic notations and definitions that will be used throughout the paper. It is assumed that the reader is familiar with databases notations (see [28]).

Let U be a set of attributes. A relation schema R is a name associated with attributes of U, i.e. $R \subseteq U$.

Let D be a set of constants, $A \in U$ and R a relation schema. The domain of A is denoted by $dom(A) \subseteq D$. The definition of a tuple t over R t over R is a function from R to D. A relation r over R is a set of tuples over R. In the sequel, we will use interchangeably the term relation or dataset. If $X \subseteq U$, and if t is a tuple over U, then we denote the restriction of t to X by $t[X]$. If r is a relation over U, then $r[X] = \{t[X], t \in R\}$. The active domain of A in r, denoted by $adom(A, r)$, is the set of values taken by A in r.

2.1 Functional dependencies

We now define the syntax and the semantics of a Functional Dependency (FD).

Definition 1. *Let R be a relation schema, and $X, Y \subseteq R$. A FD on R is an expression of the form $R : X \to Y$ (or simply $X \to Y$ when R is clear from context)*

Definition 2. *Let r be a relation over R and $X \to Y$ a functional dependency on R. $X \to Y$ is satisfied in r, denoted by $r \models X \to Y$, if and only if for all $t_1, t_2 \in r$, if $t_1[X] = t_2[X]$ then $t_1[Y] = t_2[Y]$.*

The satisfaction can be verified using this well-known property:

Property 1. Let r be a relation over $\mathcal{R}$ and $X \to Y$ an FD over R. Then: $r \models X \to Y \Leftrightarrow |\pi_{XY}(r)| = |\pi_X(r)|$

Understanding what tuples prevent a given functional dependency to be satisfied relies on the notion of counterexamples, defined as follows:

Definition 3. *Let r be a relation over R and $X \to Y$ a FD f on R. The set of counterexamples of f over r is denoted by $CE(X \to Y)$ and defined as follows:*

$$CE(X \to Y, r) = \{(t_1, t_2) | t_1, t_2 \in r, \text{ for all } A \in X, t_1[A] = t_2[A] \text{ and there exists } B \in Y, t_1[B] \neq t_2[B]\}$$

The error of the functional dependency in a relation has been addressed in in [23], in which three measures are presented, given a functional dependency $X \to Y$ and a relation r. Other measures, based on information theory, are presented in [15], but are out of the scope of this paper.

The first measure, G_1, gives the proportion of counterexamples in the relation:

$$G_1(X \to Y, r) = \frac{|\{(u, v) | u, v \in r, u[X] = v[X], u[Y] \neq v[Y]\}|}{|r|^2}$$

Using definition 3, this can be rewritten as:

$$G_1(X \to Y, r) = \frac{|CE(X \to Y, r)|}{|r|^2}$$

Following this first measure, it is also possible to determine the proportion of tuples involved in counterexamples. This measure G_2 is given as follows:

$$G_2(X \to Y, r) = \frac{|\{u | u \in r, \exists v \in r : u[X] = v[X], u[Y] \neq v[Y]\}|}{|r|}$$

These two metrics are designed to evaluate the importance of counterexamples in the relation. Similarly, measure G_3 computes the size of the set of tuples in r to obtain a maximal new relation s satisfying $X \to Y$. Contrary to [23] that present this measure as an error, we propose it as follows:

$$G_3(X \to Y, r) = \frac{max(\{|s| | s \subseteq r, s \models X \to Y\})}{|r|}$$

Example 2. Using table 1, considering the functional dependency $fd = Ticket, Age, Gender \to Survived$, we have:

$CE(fd, Titanic) =$
$\{(t_1, t_{10}), (t_2, t_5), (t_3, t_8), (t_5, t_6), (t_5, t_9), (t_7, t_8)$
$(t_{10}, t_1), (t_5, t_2), (t_8, t_3), (t_6, t_5), (t_9, t_5), (t_8, t_7)\}$

As a result:

- $G_1(fd, Titanic) = \frac{12}{100} = 12\%$
- $G_2(fd, Titanic) = \frac{9}{10} = 90\%$
- $G_3(fd, Titanic) = 70\%$

2.2 Supervised classification in machine learning

Traditionally, a supervised classification problem (see [30]) consists in a set of N training examples, of the form
$\{(x_1, y_1), ..., (x_N,\ y_N)\}$ where x_i is the feature vector of the i-th example and y_i its label. The number of labels (also known as class), k, is limited and usually much smaller than the number of examples ($k = 2$ in binary classification problems). Given the training examples, classification is the task of learning a target function g that maps each example x_i to one of the k classes.

The function g, known as a classifier is an element of some space of possible functions G, usually called the *hypothesis space*. The objective of a learning algorithm is to output the classifier with the lowest possible error rate, which is the portion of misclassified examples according to their ground truth label.

It is sometimes convenient to represent g using a scoring function $f : X \times Y \to \mathbb{R}$ such that g is defined as returning the y value that gives the highest score:

$$g(x) = \arg\max_y f(x, y)$$

This optimal function can take different forms, depending on the learning algorithm used to define it: polynomial, exponential, sometimes not even expressible using simple formulas (black boxes). This function g is often referred to as the *model* of the classification task.

In the rest of the paper, we will consider a relation r over $\{A_1, \ldots, A_n, C\}$ such that for every tuple $t_i \in r$, $t_i[A_1 \ldots A_n] = x_i$ and $t_i[C] = y_i$.

Evaluating the performances of a classification model is a crucial part in a learning process. It allows to evaluate its quality and how well the model generalizes. It is also useful to compare the performances of different learning algorithms over a given dataset, to choose the most appropriate one given the problem at hand. In this paper, we will focus mainly on *accuracy*, a simple but efficient measure. We refer to [39] for a detailed overview of how to deal with the evaluation of classification models.

Accuracy is a widely used metrics, as it is both informative and easy to compute. Given a classification dataset of N samples, accuracy is the proportion of samples that are correctly classified by the model. This score lies between 0 and 1, and ideally should get as close as possible to 1. Given a model M over a relation r, the accuracy is defined as follows:

$$accuracy(M, r) = \frac{\#\, of\, correct\, predictions}{|r|}$$

3 From machine learning to functional dependencies

3.1 Existence versus determination of a function

At first sight, supervised classification and functional dependencies do not appear to be two related concepts: they do not apply to the same problems. Both share the well-known concept of *function*, recalled below:

Definition 4. *A function $f : X \to Y$ is a mapping of each element x of a set X (the domain of the function), to a* unique *element y of another set Y (the codomain of the function).*

According to the core definition of a classification problem in section 2, a classifier is itself a function: for any input vector, it predicts a unique output value. As a result, classifiers rely on the assumption that there exists a function from the attributes to the class in the dataset.

Functional dependencies also rely heavily on this notion of unique output. Indeed, a relation r satisfies a FD $X \to Y$ if and only if all tuples that are equal on X are associated with the same unique value on Y. As a result, $r \models X \to Y$ if and only if there exists a function f from X to Y, on the active domain of r. It should be noted that a functional dependency is a statement of the existence of a function in every possible relation r over R. A classification task can't be defined at the schema level, and requires a dataset to be provided as an input.

If we combine these results, it appears that a classifier determines a function over a relation, whereas a satisfied functional dependency guaranties the existence of a partial function from the active domain of X to the active domain of Y. Therefore, it follows:

Property 2. $r \models A_1, \ldots, A_n \rightarrow C \iff$ there exists a function f from $adom(A_1, r) \times ... \times adom(A_n, r)$ to $adom(C, r)$

This problem is clearly easier than the associated classification problem: determining the function itself requires more investigation than proving its existence. However, it is interesting to notice that in most classification problems, the existence of the function is assumed but not formally verified. This is surprising, as it could be useful at several steps of the learning process. First, FDs satisfaction in the training set can be used for data cleaning. Indeed, the tuples that are counterexamples of the functional dependency can be used to identify inconsistencies in the dataset, that can have various explanations. Seeing those counterexamples could then assist an analyst during the data cleaning part, to see where those inconsistencies come from: are they normal, or do they correspond to measurement errors that should be corrected ? If a few counterexamples can be expected, when their proportion is too high, the classification model might not be trusted. This observation allows the data scientist and the domain expert to stay connected, and to demystify part of the black box aspect of machine learning, by explaining concrete limitations of the model.

To summarize, our first proposition is, given a classification dataset, to verify the satisfaction of the functional dependency $attributes \rightarrow class$. Contrary to many pattern mining problems related to functional dependencies' enumeration, that pose combinatorial complexity, this problem is dramatically simpler, as we only consider one dependency, $attributes \rightarrow class$, with $attributes = A_1 \ldots A_n$.

3.2 Upper bound for accuracy

In this setting, measure G_3 appears to be of crucial importance for the classification problem, as it represent the proportion of tuples in the dataset, that satisfy the considered functional dependency. In this subset of the original data, there is therefore no counterexample. This means that in the subset s defined for G_3, there exists a function between the left and right hand side of the dependency. Theoretically, it is therefore possible for a classifier to reach a perfect score if it identifies the correct underlying function, independently of its capabilities to generalize from it. On the opposite, the counterexamples to $A_1, \ldots, A_n \rightarrow class$ are blocking point for any classification algorithms, as they introduce pairs of tuples for such that the classifier will misclassify at least one of them. As a consequence, we propose the following result:

Proposition 1. *Let $A_1, \ldots, A_n \rightarrow class$ be a FD over $\mathcal{R}$, r a relation on $\mathcal{R}$, and M a classifier from $A_1, \ldots, A_n$ to C. Then:*

$$accuracy(M, r) \leq G_3(A_1, \ldots, A_n \rightarrow C, r)$$

Proof. Let s be a maximum subset of r such that $s \models A_1, \ldots, A_n \rightarrow C$.

For all $(t_i, t_j) \in s$, if $t_i[attributes] = t_j[attributes]$ then $t_i[class] = t_j[class]$.

Let $t_h \in r \setminus s$. Then there exists $t_i \in s$, such that $(t_i, t_j) \in CE(attributes \rightarrow class, r)$, otherwise s is not maximal. If only the tuples from $r \setminus s$ are misclassified, and all the tuples from s correctly classified, $accuracy(M, r) = \frac{|s|}{|r|} = G_3(attributes \rightarrow class, r)$.

If some tuples are misclassified due to the algorithm itself, this can only lower the accuracy, and thus the result follows.

This upper-bound result is simple but powerful, as it can be applied to any classification dataset, and offer guaranties on the feasibility of classification over it. Moreover, G_3 is closely related to Bayes error [43], allowing to revisit this error through the prism of functional dependencies. The statistical learning theory give theoretical bounds for the capacity of learning of a given type of classifier [44], but these bounds are extremely difficult to compute. Our usage of G3 give an estimation of a high bound which can eventually be attended by a classifier on a given dataset. In addition, we make simpler assumptions by only considering the available data and comparing the given tuples, and do not consider a probability distribution. What we propose is an upper bound, which is only based on the tuples used to evaluate the accuracy, which means that this can be influenced by how the available data is split between training and testing sets for example.

3.3 Validation on classification datasets

We propose to compute this upper bound on various classification datasets, to see how state-of-the-art algorithms perform in terms of accuracy, with respect to this upper bound.

***G_3* computation** Contrary to G_1 and G_2 that can be quite easily computed by looking at each pair of tuples, G_3 requires to identify the tuples to be removed from the dataset so that is satisfies the dependency. In addition, the minimum possible number of tuples should be removed. In the general case, this is a NP-complete problem, as it is equivalent to the minimal vertex cover problem for graphs [42].

However, when the data is discrete and when using crisp functional dependencies, the comparison of values is transitive, and it is therefore possible to compute G_3. Following the definition of G_3, in order for s to be maximal, it should keep as many tuples as possible, while removing all the counterexamples of the given functional dependency. As a result, this can be done by grouping all the tuples that share the same left hand side, and then selecting among them the ones that share the same right hand side, and that are the majority. This allows to remove all the counterexamples, while removing the minimum number of counterexamples. The size of s is therefore the sum, for each different left hand side, of the size of the majority right hand side. Therefore, G_3 can be computed with the following proposition:

Proposition 2. *Let r be a relation over $\mathcal{R}$. Then:*

$$G_3(X \to Y, r) = \frac{\sum_{x_i} \max_{y_i} |\pi_{XY}(\sigma_{X=x_i \wedge Y=y_i}(r))|}{|r|}$$

where $x_i \in \pi_X(r)$ *and* $y_i \in \pi_Y(\sigma_{X=x_i}(r))$.

Note that $X = x_i$ is a simplification for $A_1 = v_1 \wedge \ldots \wedge A_n = v_n$ for $X = \langle A_1..A_n \rangle$ and $x_i = \langle v_1..v_n \rangle$.

To compute this measure, we propose algorithm 1. It relies on a specific data structure, presented on figure 1, with tuples from table 1 as an example. It is a hash map, with the values over the attributes as key, and another hash map as value. For the second map, the key is the class, and the value the number of times this class appears (for these given attributes). The construction of this map is explained from line 3 to line 14 of algorithm 1: for each row in the dataset, the corresponding values in the map are filled or created when necessary. Once this data structure is complete, the algorithm looks at each key in the map: it will then retrieve the number of occurrences for the class that has the highest value in the second map. All these maximum values are added to one another, as they correspond to the maximum number of tuples that can be kept, among the ones that share the same attributes value, in order to satisfy the functional dependency. This process is explained though line 15 to 19 in algorithm 1, with a complexity in $\mathcal{O}(\log |r|)$.

Algorithm 1: G_3 computation algorithm

```
1  procedure ComputeG3 (r);
   Input  : r the classification dataset,
            A_1 ... A_n → C a functional dependency
   Output: G_3(A_1 ... A_n → C, r)
2  map = {}
3  for row ∈ r do
4      if row[A_1..A_n] ∈ map then
5          if row[class] ∈ map[row[A_1..A_n]] then
6              map[row[A_1..A_n]][row[class]] += 1
7          else
8              map[row[A_1..A_n]][row[class]] = 1
9      else
10         map[row[A_1..A_n]] = {}
11         map[row[A_1..A_n]][row[class]] = 1
12 maxsum = 0
13 foreach key ∈ map do
14     maxfrequent = max(map[key])
15     maxsum += maxfrequent
16 return maxsum / |r|
```

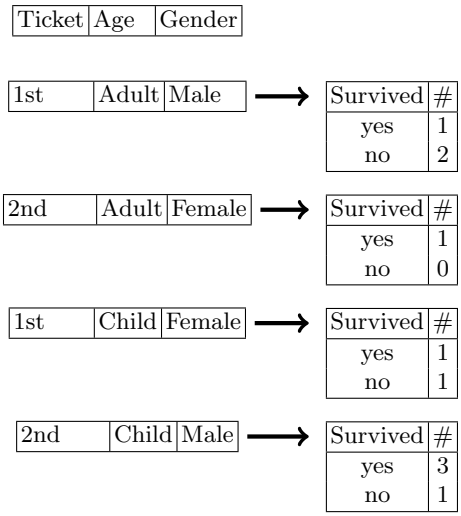


Fig. 1: Data structure for G_3 computation

Example 3. Using figure 1, measure G_3 can be computed as follows for the Titanic dataset:

$G_3(Ticket, Age, Gender \rightarrow Survived, Titanic) = \frac{2+1+1+3}{10} = \frac{7}{10} = 70\%$

Experimentations We decided to perform experimentations, in order to show experimentally the upper bound given by G_3 on the accuracy of classifiers. To do so, we ran experiments on well-known datasets, to compare the state-of-the-art accuracy results for these datasets with their G_3 measure. The datasets and the accuracy measure come from [19], a thorough study on the accuracy of 179 classification algorithms over 121 datasets: the measures we use are therefore the ones given by this study using these algorithms.

The results are presented in table 2. As expected, for all the datasets, the maximum accuracy measured in [19] is always below our measured G_3 value: the difference between the two values is indicated in the last column of the table, showing some significant differences for some datasets. For some datasets, the G_3 measure is 100%, which is reassuring, as it means that their exists a function between the attributes and the class, and that it does actually make sense to perform classification. It is also interesting that despite the existence of a function, the average accuracy is still very low sometimes, such as for the Contract dataset. Finally, the Titanic and the Led display are very interesting datasets, as their G_3 measure is pretty low compared to the other ones. Therefore it is worth questioning the interest of performing classification on these datasets. For the Titanic dataset, the maximum accuracy is strictly equal to G_3, meaning their exists a *perfect* classifier on this dataset, that only failed on tuples from counterexamples.

These results are very interesting, as they show how this paper's result could be used on any given datasets. It indicates both to data scientist wether or not it is worth performing classification, or if there are simply too many counterexamples for the results to be interesting. In addition, depending on the classification's results, it is interesting to see if G_3 is reached or not. Based on these measures, it could also be used to compare various classification methods, in order to see how they influence the accuracy, and if there are methods that are more often that other closer to the upper bound defined by $G3$.

Dataset	# tuples	# classes	Average Accuracy (%)	Max Accuracy G_3 (%)	$G_3 - max$
Titanic	2201	3	76.6	79.1	79.1	0.0
Breast Cancer	286	9	71.2	76.2	97.8	21.6
Abalone	4177	5	60.1	67,4	100	32.6
Adult	48842	13	81.8	86,2	99.9	13.7
Bank	4521	16	88.4	90,5	100	9.5
Car	1728	6	86	99,2	100	0.8
Contrac	1473	9	49.6	57,2	95,5	38.3
Ecoli	336	7	77.6	90,9	100	9.1
Iris	150	4	89.4	99,3	100	0.7
Led-display	1000	7	60.3	74,8	76	1.2
Lenses	24	4	74	95,8	100	4.2
wine-quality-red	1599	11	55.6	69	100	31
yeast	1484	8	52.5	63,7	100	36.3
zoo	101	16	86.5	99.0	100	1.0

Table 2: Comparison of accuracy and G_3 measure over classification datasets

4 Generating difficult datasets for classification

In this section, we are interested in generating datasets for classification as hard as possible, using the notion of counterexamples.

4.1 Generation

The idea to generate datasets with a very low G_3 measure, such that any classifier will not be able to perform efficiently on it, as shown in the previous section. Generating such datasets can then be used to test new classifier, or to improve existing one so that they get as close as possible to the theoretical maximum accuracy (i.e. they only misclassify counterexamples).

To generate such datasets, it is necessary to create counterexamples, and to play with their number to increase G_3 error and lower the maximum accuracy. To get very difficult classification datasets, we propose to start with an initial classification relation that does not have any identical tuples, and to duplicate them, by associating each new tuple to a different class at each duplication. This

will naturally introduce counterexamples, and their number will increase with the number of duplication.

id	A_1	A_2	...	A_{n-1}	A_n	Class	Scaling Factor
1	13	9	...	21	16	1	
2	58	13	...	18	5	2	
⋮	⋮	⋮	⋮	⋮	⋮	⋮	
...	...	...	...	...	...	k	1
...	...	...	...	...	...	1	
⋮	⋮	⋮	⋮	⋮	⋮	⋮	
N	35	9	...	21	11	4	
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
$(j-1)\times N+1$	13	9	...	21	16	$(t_{((j-1)\times N+1)-N}[Class]+1)\%k$	
...	...	...	...	...	...	...	
i	$t_{i\%N}[A_1]$	$t_{i\%N}[A_2]$	...	$t_{i\%N}[A_{n-1}]$	$t_{i\%N}[A_n]$	$(t_{i-N}[Class]+1)\%k$	j
...	...	...	...	...	...	...	
$j\times N$	35	9	...	21	11	$(t_{(j\times N)-N}[Class]+1)\%k$	
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
$(sf-1)\times N+1$	13	9	...	21	16	$(t_{((sf-1)\times N+1)-N}[Class]+1)\%k$	
...	...	...	...	...	...	...	sf
$sf\times N$	35	9	...	21	11	$(t_{(sf\times N)-N}[Class]+1)\%k$	

Table 3: Generation of a difficult dataset

The generation strategy is illustrated in table 3. It requires the size N of the initial relation, the number n of attributes of the relation, the number k of classes, and the scaling factor sf (total number of relations after the duplications) used to produce counterexamples. The strategy works as follows: let r be a relation over $R = A_1 \dots A_n$. Then:

1. Insert N unique tuples t_i for $i \in 1..N$. For each tuple $t_i, i \in 1..N$, add a value for the *class* attribute as follows: $t_i[Class] = i\%k$. This corresponds to the rows 1 to N in table 3.
2. The duplication process is repeated $sf - 1$ times as follows:
   Let j be the current duplication, $2 \leq j \leq sf$. The initial relation is duplicated, generating N new tuples, numbered $t_{(j-1)\times N+1}$ to $t_{j\times N}$. For each duplicated tuple $t_i, (j-1)\times N+1 \leq i \leq j\times N$, the values do not change over attributes A_1 to A_n, i.e. $t_i[A_1...A_n] = t_{i\%N}[A_1...A_n]$. However, the class value is shifted by one with respect to the previous duplicate, i.e. $t_i[Class] = (t_{i-N}[Class]+1)\%k$

Algorithm 2 give the details of the generation process. Let us mention a few important point not detailed here.

Algorithm 2: Difficult dataset generation algorithm

```
1  procedure GenerateDifficult (n, N, k, sf);
   Input  : n the number of attributes, N the number of tuples before
            duplication, k the number of classes, and sf the scaling factor
   Output: d a difficult dataset for classification
2  class = 1
3  relation = [N][n+1]
4  for i ∈ 1..N do
5      for j ∈ 1..n do
6          relation[i][j] = random([0:ln(N)]
7      relation[i][n+1] = class
8      if class == k then
9          class = 1
10     class + = 1
11 if A_1..A_n is not a key ;                    // Check that all rows are unique
12 then
13 dataset = relation
14 copy = relation
15 for i ∈ 2..sf do
16     copy = Duplicate(copy, k);
17     dataset = dataset ⋃ copy;
18 return dataset
19 Function Duplicate(r, k):
20     duplicate = []
21     for row ∈ r do
22         new[A_1...A_n] = row[A_1...A_n]
23         new[class] = (row[class] + 1)%k
24         duplicate += new
25     return duplicate
```

– First, the domain of attributes is to be defined. In table 3, we use integers for the sake of clarity, but any other type of attribute would work exactly the same. However the data types will have an impact on the classifiers, as for example non-numerical values would require some pre-processing to be used with most classifiers. This also underlines how classification is impacted par features' domains while FDs are not.
– The only limitation on the attribute domain is to have enough values to generate unique rows, at least $\frac{ln(N)}{ln(n)}$ values[1]. Using a really high number of different values will only increase the difficulty for a classifier, as there will

[1] Given $|dom|$ different values, there exists $n^{|dom|}$ different vectors of size n. Therefore it is necessary that $N < n^{|dom|}$ and thus $|dom| > \frac{ln(N)}{ln(n)}$

be very little redundancy between the values. This is a parameter than can be used to tune the difficulty of the classification dataset. In Algorithm 2 and the experiments, we used $\ln(N)$ different values (see line 6 in algorithm 2.

– In addition, whenever $sf > k$, the dataset contains duplicates that share the same class values, as all values for the class have already been used for duplicates. This introduces redundancy in the data, but does not remove any counterexample. Given the parameters of algorithm 2, it is possible to compute the exact value of G_3 for the produced dataset:

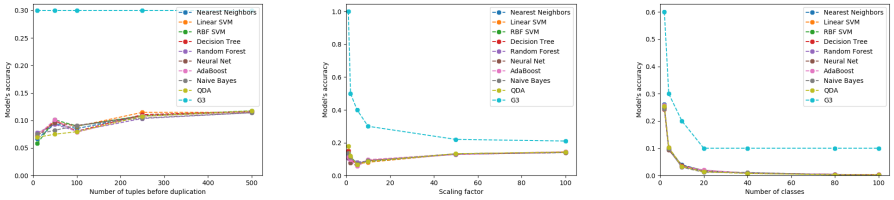
Proposition 3. *Let r_{hard} be a relation generated using algorithm 2. Then:*

$$G_3(A_1..A_n \rightarrow Class, r_{hard}) = \frac{1 + \frac{sf - sf\%k}{k}}{sf}$$

Proof. Let $i \in [1..sf]$ denote the i-th duplication of the initial relation. While $i < k$, it only introduces counterexamples. Therefore, for each duplicated tuple, there are i different classes, for each of the N original tuples. As a consequence, if $sf < k$, $G_3 = \frac{N}{N*sf} = \frac{1}{sf}$.

For $i \geq k$, there is redundancy for each duplicate: the duplicated tuples agree with the ones already produced. Therefore, there are as many agreeing tuples as the number of times $i\%k = 1$. The size of the set of agreeing tuples therefore depends of how many times an initial tuple is associated with the same initial class during the sf duplications, which is exactly the quotient of the euclidean division of sf by k, i.e $\frac{N + N * \frac{sf - sf\%k}{k}}{sf * N}$. And the result follows.

4.2 Experimentations

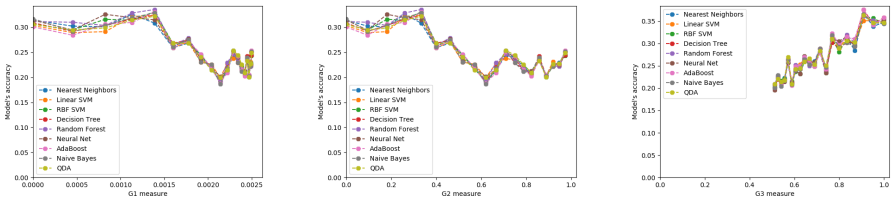


(a) Evolution of classifiers accuracy against the number of tuples the dataset, with respect to G_3 ($k = 5$, $sf = 10$)

(b) Evolution of classifiers accuracy against the scaling factor of the dataset, with respect to G_3 ($N = 100$, $k = 5$)

(c) Evolution of classifiers accuracy against the number of classes in the dataset, with respect to G_3 ($N = 100$, $sf = 10$)

Fig. 2: Classifiers accuracy given the parameters for generating difficult classification datasets, compared to G_3



(a) Measure G_1 (b) Measure G_2 (c) Measure G_3

Fig. 3: Evolution of classifiers accuracy given the FD error measure of the dataset

We implemented the generation algorithm, in order to measure the influence of its different parameters. To do so, we generated datasets with different parameters, and used ten different classifiers from the scikit-learn library [32] to estimate their accuracy over the datasets. In order to propose a general comparison over several algorithms, the algorithms are based on the default parameters from the library. All results presented below are averaged over 10 different instances randomly generated using the algorithm. The computing time being below one second, they are not discussed in this paper.

First, the influence of the number of tuples in the original relation (before duplication), was tested. The results are shown on figure 2a. It is worth noticing that the accuracy is in any case really low as expected: the maximum accuracy reached is 12%. However, as the initial number of tuples increases, so does the accuracy. Indeed, adding more tuples introduces some redundancy among the values for each attribute, allowing the classifier to find some sort of generalization for some cases. However, the number of counterexamples is way too high to reach good classification measure, and $G3$, which is constant as the number of tuples does not influence it, is also low: it can be seen that the model is far from reaching it.

Then, the influence of the scaling factor sf was tested, and results are presented on figure 2b. As it influences G_3, this measure slowly decreases with the scaling factor, as more and more counterexamples are introduced. For this parameter, the accuracy first drops, before slowly increasing with the scaling ratio. This increase starts as soon as $sf > k$, as explained previously, because there is then some redundancy allowing the classifier to make correct predictions for some tuples.

In addition, the influence of the number of classes is exposed on figure 2c. Once again, G_3 decreases with the number of classes, as their are more counterexamples. As expected, the accuracy only drops with the number of classes, as the classifier has then fewer examples for each class, and therefore fewer possibilities to find patterns and generalize.

Finally, the accuracy of the classifiers was evaluated with respect to the error measure of the functional dependency $A_1...A_n \rightarrow class$. The results are shown on figure 3. For measures G_1 and G_2 the accuracy drops as the error increases, as the

number of counterexamples also increases. On the opposite, when G_3 increases, so does the accuracy, as it means that the set of tuples that would satisfy the functional dependency is getting bigger, allowing the classifier to reach a higher accuracy.

5 Application to real life datasets

Using functional dependency to evaluate the feasibility of classification datasets can be very useful for industrial applications. Indeed, many companies now understand the interest of their data, and are more and more interested in applying machine learning methods to gain value out of their data. Being able to quickly tell the company what can be hoped for as classification results, given their data, is a very useful strategy to start the discussion, and to see what can be done to improve if necessary. However, real-life datasets are known to be often dirty, with null or imprecise values, incoherence and uncertainties. In this setting, we discuss the interest of applying the previous results right before ML techniques in practical scenarios, to let the domain experts in the loop of the construction of their ML model.

5.1 Interest and limits of G_3 measure

Measures like G_3 can be seen as an indicator of whether or not the classification should be done, in comparison with the accuracy required by domain experts. Moreover, expressing at a declarative level such a constraint to domain experts offers more guarantee about their understanding of the whole process.

The main problem concerns the inability of crisp functional dependencies to deal with real-life data. Fortunately, many propositions have been made to extend functional dependencies, for example [34, 7, 9, 12]. Indeed, when performing the comparison of tuples on each attribute, the strict equality might not be the best suited for comparing real-life values. When dealing for example with continuous values, it is very less likely for two values to be equal, and therefore all possible functional dependencies are likely to be satisfied in a given dataset. This is especially true for physical measurements, for which the precision of the measure is important to take into account: two values might not be exactly equal, but by considering the interval of measurement uncertainty, actually overlap.

Example 4. On figure 5a, the table presents data from a meteorological problem: given the temperature, pressure and humidity of a place, will it be raining the next hour ? All the attributes are measured using instruments that have a measurement uncertainty, well-known from the meteorologists:

- Temperature measurement uncertainty is $\pm 0.5°C$
- Pressure measurement uncertainty is $\pm 1hPa$
- Humidity measurement uncertainty is $\pm 2\%$

Considering this, every possible crisp FD is likely to be satisfied and all of them are thus useless.

We also note that very close values between two tuples but with a different class might confuse the classifier and prevent its generalization. In these situations, feedbacks of domain experts on the dataset is important to understand what it means for two values to be equal or similar on a given attribute.

Main issue: To sum up, we have to take into account some form of similarities between data values, without requiring domain experts to spend times on these time consuming tasks. This will be addressed in Section 5.3.

5.2 Interest of counterexamples

The counterexamples are also very important to understand the score, to see the data that causes conflicts, so that a domain expert can explain their presence, and eventually remove them to improve the classification results. The counterexamples are a powerful notion to avoid the domain expert from being overwhelmed by the data, as she then only have a small but meaningful subset of tuples to study. The counterexamples are therefore a perfect starting point for a discussion between data scientist and domain expert: while the first gain knowledge on data they are not expert on, the others can point out important information more easily. The counterexamples are a way for domain experts to read a concrete information that can have an impact on their day-to-day work.

With respect to counterexamples, not all of them are of equal interest. To help domain experts in exploring them, and especially if there are many of them, it is important to think about how to present them, so that the experts are not overflowed. To do so, several strategies are possible, for example ranking the counterexamples, by computing how much they differ on the class value, which is especially easy for numerical value, by computing the difference between the two values. For other types, a score can be manually defined. In addition, we propose to visualize all of the counterexamples, so that domain experts can see the global picture in one glance. For instance, a graph representation can be built as follows: Let $G = (V, E)$ a graph where V is the set of tuples and $(t_1, t_2) \in E$ if t_1, t_2 are implied in one counterexample. The degree of a node is a first criterion to sort out the tuples to focus on. This modelisation show how the computation of G_3 and the removal of counterexamples can be seen as a graph problem, equivalent to the minimal vertex cover [42].

Example 5. Figure 4 presents such a graph for the Titanic dataset. The degree of t_5 (resp. t_8) is 3 (resp. 2). Clearly t_5 and t_8 cause more counterexamples than the others. By changing the class value for these two tuples, the number of counterexamples drops to 1 (instead of 6).

Such visualizations can be useful for data cleaning, as the most conflictual tuples could then be easily identified and removed, to easily decrease significantly the number of counterexamples. Another solution would be to correct the counterexamples, with a minimal number of correction, by assigning them a class that does not contradict the other tuples. These vizualisation is also useful to see that finding an optimal solution is not an easy problem. Indeed, the objective is,

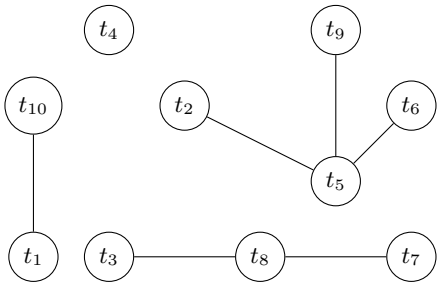


Fig. 4: Couterexamples interaction graph for the Titanic dataset

from the graph, to remove as few vertices as possible, so that there are no more edges. This is known as the minimal vertex problem, which is NP-complete.

Main issue: To sum up, when it comes to counterexamples enumeration, the main bottleneck lies on the comparison of each pair of tuples, which is quadratic in the number of tuples. This problem appears in different contexts such as deduplication, and several techniques have been proposed to perform such comparison in a reasonable amount of time, such as blocking [4], or parallelization using specific distribution strategies as in [13].

5.3 Similarity-aware dataset reduction for scaling counterexamples enumeration

In order to overcome the problems mentioned in sections 5.1 and 5.2, we propose a hybrid approach, that can both scale the counterexamples retrieval, while also allowing to consider similarities between values of a given attribute. Our approach is based on single attribute clustering and dataset reduction. It is a form of discretization, which is a classic approach in machine learning. In this setting, we use it to reduce the size of the dataset on which to evaluate the FD, and to therefore reduce the number of necessary comparisons between pairs of tuples.

The idea is to first group similar values together, so that they are all assigned to a unique same value: this allows to define similarity. Second, as the number of different values is likely to be much smaller, it is possible to "reduce" the dataset by only keeping unique rows. The detailed process is illustrated on figure 5, and works as follows:

- First, each attribute of the original data is clustered, to group similar values. The clustering algorithm can be adapted to the need, we propose to use k-means [29], using the silhouette coefficient [36] to automatically determine the best value for k. Of course, domain knowledge can also be used if it is available. It should be noted that this can be adapted and fine-tuned to each application, and that specific similarity measures can be defined in this step if necessary. Clustering could also be extended to deal with more than one attribute (not detailed here).

id	Temp.	Pres.	Hum.	Rain
t_1	27.2	1004.5	98.7	yes
t_2	26.5	1018.4	42.5	no
t_3	15.7	1008.6	78.9	yes
t_4	16.1	1016.9	76.7	no
t_5	25.9	1017.5	43.8	yes
t_6	28.1	1021.7	41.7	no
t_7	4.1	1007.2	74.3	yes
t_8	15.9	1022.3	79.1	no
t_9	27.3	1019.8	39.5	no
t_{10}	3.8	1006.7	71.4	yes

(a) Original data

id	Temp.	Pres.	Hum.	Rain
t_1	1	1	4	yes
t_2	1	2	1	no
t_3	3	1	2	yes
t_4	2	2	2	no
t_5	1	2	1	yes
t_6	1	2	1	no
t_7	3	1	2	yes
t_8	2	2	2	no
t_9	1	2	1	no
t_{10}	3	1	2	yes

(b) Clustered data

id	Temp.	Pres.	Hum.	Rain	#
t'_1	1	1	4	yes	1
t'_2	1	2	1	no	3
t'_3	2	2	2	no	2
t'_4	1	2	1	yes	1
t'_5	3	1	2	yes	3

(c) Reduced data

Fig. 5: Data reduction process

- Once an attribute is clustered, each of its value is replaced by the number of cluster it belongs to. As functional dependencies do not care about the order between values, this does not impact the validity of the dependency.
- Once the data is clustered, as the number of different values for each attribute is equal to the number of clusters, there might be identical rows in the dataset. It is therefore possible to dramatically reduce the size of the original data, by only keeping unique rows, and adding an additional attribute to memorize how many times this row appears in the clustered data.

To perform this process, we propose algorithm 3: the clustering process is described from line 3 to 9, and the data reduction is performed on line 12.

Example 6. In the reduced dataset given in table 6c, let us consider the counterexample (t'_2, t'_4) of the functional dependency $Temp, Pres,\ Hum \rightarrow Rain$. Using table 6b, it concerns in fact 4 tuples: t_2, t_6, t_9 and t_5 (cf previous example). It is then possible to go back to the original data in table 6a to see what does that mean on the real values. For example, values $\langle 25.9, 1017.5, 43.8\rangle$ of t_5 and values $\langle 28.1, 1021.7, 41.7\rangle$ of t_6 are considered similar and then, both tuples form a counterxample.

5.4 Experimentations

We evaluate both the data reduction technique and the scalability of counterexamples enumeration. We first looked at how the data is reduced. We took the 4 biggest datasets from table 2, and computed their reduction ratio, when applying our algorithm. In this situation, we used the silhouette coefficient [36] to determine the most appropriate number of clusters for each attribute. The results are presented in table 4. It shows that the ratio differs from one dataset to another, with some very significant drops for some datasets such as Titanic, indicating their must be many redundant values. We then evaluated how well

Algorithm 3: Scaling algorithm

```
procedure Reduce (r) over attributes A_1..A_n;
  Input  : r the classification dataset
  Output: A clustered and reduced dataset of integers
d = []
for A ∈ {A_1..A_n} do
    if A is continuous then
        k = max_silhouette(r[A])
        clusters = cluster(r[A], k)
        d[A] = clusters
    else
        d[A] = r[A]
d[class] = r[class]
r_reduced =Select A_1 ... A_n, C, count(*) as '#' From d Group By A_1 ... A_n, C
return r_reduced
```

the data reduction technique improves the counterexamples retrieval time, to see how the approach would scale on large datasets. We evaluated it on astrophysical data from the Large Synoptic Survey Telescope[2] containing 500 000 tuples over 25 attributes. For different sizes of datasets, we compute:

- The reducing ratio, i.e how much the initial dataset is reduced:

$$ratio = \frac{|r| - |r_{reduced}|}{|r|}$$

  Where $r_{reduced}$ is the reduced dataset.
- The computation time for counterexamples retrieval on the original data.
- The computation time for counterexamples retrieval on the reduced data.

The results of these experimentations are presented on figure 6. The first observation is that on the original data, the computing time is quickly too long even on relatively small instances. On the opposite, on the reduced data, it increases very slowly with the number of tuples, allowing for a reasonable time for retrieving the counterexamples. This is tightly linked with the reducing ratio, that increases significantly with the number of tuples: the more tuples there are, the more the original data is reduced with respect to its original size.

On datasets on which our approach would not be sufficient to scale, it is always possible to apply blocking [4] or parallelization [13], but this is out of the scope of this paper.

[2] https://www.lsst.org/

Dataset	# tuples before	#tuples after	reduction ratio
Titanic	2201	24	98.9%
Abalone	4177	242	94.2%
Adult	48842	10176	79.2%
Bank	4521	4031	10.8%

Table 4: Reduction ratio for some datasets from table 2

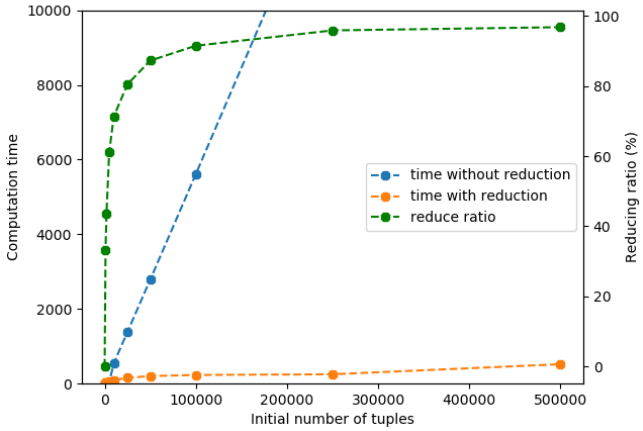


Fig. 6: Validation of G_3 computing time, on both original and reduced data, in parallel with the reducing ratio

6 Related work

This related work section has been organized according to three axes: data dependencies for machine learning, extension of functional dependencies and data dependencies for data cleaning.

6.1 Data dependencies for machine learning

Many research papers make use of principles from the database community to tackle machine learning problems, and vice versa (see for example [6, 3, 1]). Recently, relational learning has been proposed [25, 20], an alternative machine learning approach based on declarative relational representations paired with probabilistic models.

Functional dependencies are used in [27] to build decision trees, leading to more accurate classifiers with a more compact structure. In [1], the authors propose to perform in-database learning, and use functional dependencies to tackle optimization problems. More generally, there is a raising interest for integrated

key database theory concepts into machine learning, such as in [41] that builds least squares regression models over training datasets defined by arbitrary join queries on database table. It is also worth mentioning [50] where an entire machine learning library has been adapted so that it is compatible with a storage of data in a DBMS. There is also [45] which is a SQL extension for data mining. Similar approach was used in [31] with an implementation of the K-means algorithm using SQL queries.

In [26], the authors show that if there is a functional dependency between features, it is likely to affect the classifer negatively. Similarly in [40], functional dependency are used to build a graph of dependency among the classification attributes, that is used to cluster the attributes, and therefore reduce total number of attributes in the dataset, which is a form of feature selection.
In comparison to these papers, to the best of our knowledge, we notice that the main observation made in this paper – a function has to exist between the features and the class to be predicted before using ML methods to determine that function – despite its simplicity and its common sense, appears to be new, not explicitly stated in previous works. We argue that it has many consequences for trusting trendy ML techniques.

6.2 Extension of functional dependencies

Although many other types of data dependencies exist, such as inclusion [10] and multivalued dependencies [17], functional dependencies proved to be the most appropriate for the given problem, as they capture the notion of function between two sets of attributes.

Many extensions of functional dependencies exist, among which we quote [22, 12, 7, 34, 11]. Approximate functional dependencies [22] allow to define dependencies that almost hold in a relation, so that not all tuples have to be looked at, as long as enough support the dependency. Relaxed functional dependencies [9] and RQL [12] are general frameworks to extend functional dependencies. Fuzzy functional dependencies [7, 34] introduce a *fuzzy* resemblance measure to compare two values on the same domain. Similarly, [11] defines similarity over complex attributes over a multimedia database.
In comparison, our clustering technique on every attribute appears as a trick allowing to both 1/ take into account complex similarities, that can be fully automated if the domain expert does not exactly know how to define the similarity and 2/ reduce the size to the database to enumerate efficiently counterexamples.

6.3 Data dependencies for data cleaning

Data cleaning is a crucial part in most of data science application, as data scientist actually spend around 80% of their time on cleaning the data [49]. As a consequence, many research has be done on addressing this problem [33]. For instance, [47] proposes a semantic data profiler that can compute samples that satisfy the same constraints than a given dataset. As the limited expressiveness of functional dependencies did not always adapt well to the need of data cleaning

on real datasets, specific dependencies have been proposed to identify inconsistencies in a dataset, and eventually repair it. Conditional dependencies [5] are functional dependencies that hold only on a subset of the dataset. Matching dependencies [18] for data repairing uses matching rules to relax the equality on functional dependencies and assign values for data repairing. In Holoclean [35], dependencies are used to clean automatically a dataset. In [37], a formal framework is proposed to bridge the gap between database theory and learnability theory, and is applied to three applications: data cleaning, probabilistic query answering, and learning. It can even be used to clean dataset in order to provide *fairness* [38]. [14] introduces denial constraints, allowing to declaratively specify logical formulae to exclude counterexamples. This work acknowledges the importance of counterexamples for data cleaning, in collaboration with domain experts.
There is a tight relationship between denial constraints and our counterexamples. Indeed, counterexamples of functional dependencies are no more than a special case of denial constraints, i.e. $\nexists t_1, t_2$ such that $t_1[X] = t_2[X]$ and $t_1[Y] \neq t_2[Y]$. Due to our data reduction techniques, our counterexamples are clearly much more general and complex than those of crisp FDs. Interestingly, we do not rely on expert users to specify logical statements for defining denial constraints, and thus counterexamples. Our proposition is fully automatic, and the counterexamples we provide at the end are complex and do not require any user input. The price to be paid is that we cannot express the induced denial constraints.

7 Conclusion

In this paper, we have proposed to estimate the feasibility of a classification over a given dataset. To do so, we have used functional dependencies, a well-known concept in the database community, and checked whether the dataset satisfies a dependency between the features and the class to predict. When it does not, which is not unusual for real life datasets, measures exist to estimate the proportion of counterexamples to the dependency, and therefore estimate whether or not it is reasonable to perform classification on the dataset. We argue that this is a way to bring more interpretability to classification, by explaining to domain experts the limitations of the model, that come from the dataset itself. This could help them to better accept the produced model, by demystifying their performances.

In addition, we have provided a tight upper-bound for the accuracy any classifier could reach on the dataset. We have proposed an algorithm and showed consistent results on well-known classification datasets. In addition, we have devised an algorithm to generate difficult datasets for classification. Finally, we have designed a hybrid approach based on single attribute clustering and on a data reduction technique, allowing to both deal with dirty data and scale the enumeration of counterexamples. Experimental results have been conducted with very good compression ratio and scalability.

The results obtained in this paper provide guarantees on the classification dataset. They can be used to decide whether or not it is worth trying to fit a model, or if more time should be spent on the dataset itself. This is very useful for companies starting with new classification problems on data that might not be yet ready for classification. In addition, the counterexamples of functional dependencies are very useful to engage the discussion with domain experts, to understand the dataset into details, and therefore give them a better understanding of what can be done classification-wise.

Acknowledgment The authors would like to thank the anonymous reviewers for their valuable remarks that helped improve and clarify this manuscript.

References

1. Abo Khamis, M., Ngo, H.Q., Nguyen, X., Olteanu, D., Schleich, M.: In-database learning with sparse tensors. In: Proceedings of the 35th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems. pp. 325–340. ACM (2018)
2. Armstrong, W.W.: Dependency structures of database relationship. Information processing pp. 580–583 (1974)
3. Berlin, J., Motro, A.: Database schema matching using machine learning with feature selection. In: International Conference on Advanced Information Systems Engineering. pp. 452–466. Springer (2002)
4. Bilenko, M., Kamath, B., Mooney, R.J.: Adaptive blocking: Learning to scale up record linkage. In: Sixth International Conference on Data Mining (ICDM'06). pp. 87–96. IEEE (2006)
5. Bohannon, P., Fan, W., Geerts, F., Jia, X., Kementsietsidis, A.: Conditional functional dependencies for data cleaning. In: 2007 IEEE 23rd international conference on data engineering. pp. 746–755. IEEE (2007)
6. Bonifati, A., Ciucanu, R., Staworko, S.: Interactive inference of join queries (2014)
7. Bosc, P., Dubois, D., Prade, H.: Fuzzy functional dependencies and redundancy elimination. Journal of the American Society for Information Science **49**(3), 217–235 (1998)
8. Bratko, I.: Machine learning: Between accuracy and interpretability. In: Learning, networks and statistics, pp. 163–177. Springer (1997)
9. Caruccio, L., Deufemia, V., Polese, G.: Relaxed functional dependencies—a survey of approaches. IEEE Transactions on Knowledge and Data Engineering **28**(1), 147–165 (2015)
10. Casanova, M.A., Fagin, R., Papadimitriou, C.H.: Inclusion dependencies and their interaction with functional dependencies. Journal of Computer and System Sciences **28**(1), 29–59 (1984)
11. Chang, S.K., Deufemia, V., Polese, G., Vacca, M.: A normalization framework for multimedia databases. IEEE Transactions on Knowledge and Data Engineering **19**(12), 1666–1679 (2007)
12. Chardin, B., Coquery, E., Pailloux, M., Petit, J.: RQL: A query language for rule discovery in databases. Theor. Comput. Sci. **658**, 357–374 (2017). https://doi.org/10.1016/j.tcs.2016.11.004, https://doi.org/10.1016/j.tcs.2016.11.004
13. Chu, X., Ilyas, I.F., Koutris, P.: Distributed data deduplication. Proceedings of the VLDB Endowment **9**(11), 864–875 (2016)

14. Chu, X., Ilyas, I.F., Papotti, P.: Discovering denial constraints. Proceedings of the VLDB Endowment **6**(13), 1498–1509 (2013)
15. Dalkilic, M.M., Roberston, E.L.: Information dependencies. In: Proceedings of the nineteenth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems. pp. 245–253. ACM (2000)
16. Doshi-Velez, F., Kim, B.: Towards a rigorous science of interpretable machine learning. arXiv preprint arXiv:1702.08608 (2017)
17. Fagin, R.: Multivalued dependencies and a new normal form for relational databases. ACM Transactions on Database Systems (TODS) **2**(3), 262–278 (1977)
18. Fan, W.: Dependencies revisited for improving data quality. In: Proceedings of the twenty-seventh ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems. pp. 159–170. ACM (2008)
19. Fernández-Delgado, M., Cernadas, E., Barro, S., Amorim, D.: Do we need hundreds of classifiers to solve real world classification problems? The Journal of Machine Learning Research **15**(1), 3133–3181 (2014)
20. Getoor, L.: The power of relational learning (invited talk). In: 22nd International Conference on Database Theory, ICDT 2019, March 26-28, 2019, Lisbon, Portugal. pp. 2:1–2:1 (2019). https://doi.org/10.4230/LIPIcs.ICDT.2019.2, https://doi.org/10.4230/LIPIcs.ICDT.2019.2
21. Han, J.: Data Mining: Concepts and Techniques. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (2005)
22. Huhtala, Y., Kärkkäinen, J., Porkka, P., Toivonen, H.: Tane: An efficient algorithm for discovering functional and approximate dependencies. The computer journal **42**(2), 100–111 (1999)
23. Kivinen, J., Mannila, H.: Approximate inference of functional dependencies from relations. Theoretical Computer Science **149**(1), 129–149 (1995)
24. Kivinen, J., Mannila, H.: Approximate inference of functional dependencies from relations. Theor. Comput. Sci. **149**(1), 129–149 (Sep 1995). https://doi.org/10.1016/0304-3975(95)00028-U, http://dx.doi.org/10.1016/0304-3975(95)00028-U
25. Koller, D., Friedman, N., Džeroski, S., Sutton, C., McCallum, A., Pfeffer, A., Abbeel, P., Wong, M.F., Heckerman, D., Meek, C., et al.: Introduction to statistical relational learning. MIT press (2007)
26. Kwon, O., Sim, J.M.: Effects of data set features on the performances of classification algorithms. Expert Systems with Applications **40**(5), 1847–1857 (Apr 2013). https://doi.org/10.1016/j.eswa.2012.09.017
27. Lam, K.W., Lee, V.C.: Building decision trees using functional dependencies. In: International Conference on Information Technology: Coding and Computing, 2004. Proceedings. ITCC 2004. vol. 2, pp. 470–473. IEEE (2004)
28. Levene, M., Loizou, G.: A guided tour of relational databases and beyond. Springer Science & Business Media (2012)
29. Lloyd, S.: Least squares quantization in pcm. IEEE transactions on information theory **28**(2), 129–137 (1982)
30. Mohri, M., Rostamizadeh, A., Talwalkar, A.: Foundations of machine learning. MIT press (2018)
31. Ordonez, C.: Integrating k-means clustering with a relational dbms using sql. IEEE Trans. on Knowl. and Data Eng. **18**(2), 188–201 (Feb 2006). https://doi.org/10.1109/TKDE.2006.31, http://dx.doi.org/10.1109/TKDE.2006.31

32. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E.: Scikit-learn: Machine learning in Python. Journal of Machine Learning Research **12**, 2825–2830 (2011)
33. Rahm, E., Do, H.H.: Data cleaning: Problems and current approaches. IEEE Data Eng. Bull. **23**(4), 3–13 (2000)
34. Raju, K., Majumdar, A.K.: Fuzzy functional dependencies and lossless join decomposition of fuzzy relational database systems. ACM Transactions on Database Systems (TODS) **13**(2), 129–166 (1988)
35. Rekatsinas, T., Chu, X., Ilyas, I.F., Ré, C.: Holoclean: Holistic data repairs with probabilistic inference. Proceedings of the VLDB Endowment **10**(11), 1190–1201 (2017)
36. Rousseeuw, P.J.: Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. Journal of computational and applied mathematics **20**, 53–65 (1987)
37. Sa, C.D., Ilyas, I.F., Kimelfeld, B., Ré, C., Rekatsinas, T.: A formal framework for probabilistic unclean databases. In: 22nd International Conference on Database Theory, ICDT 2019, March 26-28, 2019, Lisbon, Portugal. pp. 6:1–6:18 (2019). https://doi.org/10.4230/LIPIcs.ICDT.2019.6, https://doi.org/10.4230/LIPIcs.ICDT.2019.6
38. Salimi, B., Rodriguez, L., Howe, B., Suciu, D.: Interventional fairness: Causal database repair for algorithmic fairness. In: Proceedings of the 2019 International Conference on Management of Data. pp. 793–810. ACM (2019)
39. Santafe, G., Inza, I., Lozano, J.A.: Dealing with the evaluation of supervised classification algorithms. Artificial Intelligence Review **44**(4), 467–508 (2015)
40. Santanu, P., Jaya, S., Das, A.K., et al.: Feature selection by attribute clustering of infected rice plant images. International journal of Machine intelligence **3**(2), 74–88 (2011)
41. Schleich, M., Olteanu, D., Ciucanu, R.: Learning linear regression models over factorized joins. In: Proceedings of the 2016 International Conference on Management of Data. pp. 3–18. ACM (2016)
42. Song, S., Chen, L.: Differential dependencies: Reasoning and discovery. ACM Transactions on Database Systems (TODS) **36**(3), 1–41 (2011)
43. Tumer, K., Ghosh, J.: Estimating the bayes error rate through classifier combining. In: Proceedings of 13th International Conference on Pattern Recognition. vol. 2, pp. 695–699. IEEE (1996)
44. Vapnik, V., Levin, E., Cun, Y.L.: Measuring the vc-dimension of a learning machine. Neural computation **6**(5), 851–876 (1994)
45. Wang, H., Zaniolo, C., Luo, C.R.: Atlas: A small but complete sql extension for data mining and data streams. In: Proceedings of the 29th international conference on Very large data bases-Volume 29. pp. 1113–1116. VLDB Endowment (2003)
46. Wang, T., Rudin, C., Velez-Doshi, F., Liu, Y., Klampfl, E., MacNeille, P.: Bayesian rule sets for interpretable classification. In: 2016 IEEE 16th International Conference on Data Mining (ICDM). pp. 1269–1274. IEEE (2016)
47. Wei, Z., Link, S.: Dataprof: semantic profiling for iterative data cleansing and business rule acquisition. In: Proceedings of the 2018 International Conference on Management of Data. pp. 1793–1796. ACM (2018)
48. Zeng, J., Ustun, B., Rudin, C.: Interpretable classification models for recidivism prediction. Journal of the Royal Statistical Society: Series A (Statistics in Society) **180**(3), 689–722 (2017)

49. Zhang, S., Zhang, C., Yang, Q.: Data preparation for data mining. Applied artificial intelligence **17**(5-6), 375–381 (2003)
50. Zou, B., Ma, X., Kemme, B., Newton, G., Precup, D.: Data mining using relational database management systems. In: Pacific-asia conference on knowledge discovery and data mining. pp. 657–667. Springer (2006)