Toward Formal Data Set Verification for Building Effective Machine Learning Models

Jorge López, Maxime Labonne and Claude Poletti
Airbus Defence and Space, Issy-Les-Moulineaux, France

Keywords: Machine Learning, Data Set Collection, Formal Verification, Trusted Artificial Intelligence.

Abstract: In order to properly train a machine learning model, data must be *properly* collected. To guarantee a proper data collection, verifying that the collected data set holds certain properties is a possible solution. For example, guaranteeing that that data set contains samples across the whole input space, or that the data set is balanced w.r.t. different classes. We present a formal approach for verifying a set of arbitrarily stated properties over a data set. The proposed approach relies on the transformation of the data set into a first order logic formula, which can be later verified w.r.t. the different properties also stated in the same logic. A prototype tool, which uses the z3 solver, has been developed; the prototype can take as an input a set of properties stated in a formal language and formally verify a given data set w.r.t. to the given set of properties. Preliminary experimental results show the feasibility and performance of the proposed approach, and furthermore the flexibility for expressing properties of interest.

1 INTRODUCTION

In the past few decades, Machine Learning (ML) has gained a lot of attention, partially due to the creation of software libraries (e.g., (Pedregosa et al., 2011)) that ease the usage of complex algorithms. In this context, the volume of stored data has dramatically increased over the last few years. However, an often overlooked task is the data extraction and collection to create proper data sets to train efficient machine learning models.

When retrieving information for the data set collection, there are key points to take into consideration. The reason is that ML models generalize their output based on the training (seen) data. However, a problem that is commonly encountered is that a model is expected to generalize well unseen regions of the input space while such regions do not behave in accordance to the provided training data. Another problem that often occurs is that there is a class in the data set which is underrepresented (e.g., for an anomaly detection data set, 99% of the examples are normal events). In general, many data biases can occur in a collected data set. A simple strategy while collecting data sets is to collect a large number of entries, conjecturing that important data are likely to be found if more data are available. However, this strategy yields incorrect results, and moreover, large data sets can cause ML models to be trained for longer than necessary; this in turn can make certain algorithms which may yield accurate results unusable for such cases. Additionally, with the proliferation of machine generated data sets, for example via Generative Adversarial Networks, assuring that the generated data set holds some properties of interest is of utmost importance.

In order to guide the collection of a proper data set to effectively train a ML model, verifying that a partially collected data set holds certain properties of interest is a possible solution. This verification can be done with the use for formal methods, such as for example Satisfiability Modulo Theories (SMT) (Barrett and Tinelli, 2018). With a formal proof that the data set holds certain properties, it is feasible to create a formal specification of a data set. Whenever this specification is violated (certain properties do not hold), identifying the properties that do not hold may help to diagnose the missing information. This paper is devoted to the formal verification of machine learning data sets through the use of SMT (for preliminary concepts on ML and SMT, see Section 2). The approach is based on the encoding of the data set into a Many-Sorted First Order Logic (MSFOL) formula which is later verified together with the desired set of properties (see Section 3).

A tool for the verification of data sets has been developed. The tool relies on the use of the widely-

López, J., Labonne, M. and Poletti, C.
Toward Formal Data Set Verification for Building Effective Machine Learning Models.
DOI: 10.5220/0010676500003064
In *Proceedings of the 13th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management (IC3K 2021) - Volume 1: KDIR*, pages 249-256
ISBN: 978-989-758-533-3; ISSN: 2184-3228

known z3 (De Moura and Bjørner, 2008) solver. Preliminary experimental results show that in spite of the high computational complexity of SMT procedures, for the verification of data sets, these properties can be verified in a reasonable amount of time (see Section 4).

It is important to note that verifying certain properties over a data set is a task which is consistently considered as necessary, and a norm for many practitioners. However, in the literature very few researchers focus on automatic validation of data sets (see for example (Carvallo. et al., 2017)). Furthermore, to the best of our knowledge, there is no work which aims at providing means for the verification of arbitrarily stated properties, and moreover, in a formal manner. In this light, this paper aims at exploring this direction.

2 PRELIMINARIES

2.1 Machine Learning and Structured Data Sets

We consider that a *structured machine learning data set* contains *examples* alongside with their *expected outputs*. Given the inputs and expected outputs, the final goal of a supervised ML algorithm is to learn how to map a training example to its expected output. For an unsupervised ML algorithm the goal is to learn patterns from the data; thus, the expected output does not exist. In our work, we consider that the expected outputs are always present, and thus, a data set for unsupervised machine learning (where there are no expected outputs) has the same expected output for all training examples. Further, we consider only structured data sets.

Formally, the inputs are called *features* or *parameters*. A *feature vector*, denoted as $\mathbf{X}$, is an n-tuple of the different inputs, $x_1, x_2, \ldots, x_n$. The expected output for a given feature vector is called a *label*, denoted simply as y, and the possible set of outputs is respectively denoted as Y. The set of examples, called a *training data set*, consists of pairs of a feature vector and a label; each pair is called a *training example*, denoted as $(\mathbf{X}, y)$. For convenience, we represent the data set as a matrix $D_{m\times n}$ and a vector O_m where D contains the feature vectors and O contains the expected outputs for a data set of cardinality m. The vector representing the i-th row (training vector) is denoted as D_i, and its associated expected output as O_i. Likewise, the j-th feature (column vector) is denoted as D^T_j (D^T denotes the transpose of the matrix D). Finally, the j-th parameter of the i-th training example is denoted by the matrix element $d_{i,j}$.

2.2 Satisfiability Modulo Theories (SMT)

SMT is a decision problem, such that for a given first order logic formula ϕ, searches if ϕ is satisfiable w.r.t. a set of background theories. For example, w.r.t. integer linear arithmetic, the following formula is satisfiable: $\Phi = (x \in \mathbb{Z}) \wedge (y \in \mathbb{Z}) \wedge (x < y) \wedge (x < 0) \wedge (y > 0) \wedge (x + y > 0)$; the formula can be satisfied for instance by the interpretation $x = -1, y = 2$. The importance of restricting an interpretation of certain function and predicate symbols in a first-order logic formula (according to a background theory $\mathcal{T}$), is that specialized decision procedures have been proposed; thus, making the problem of checking the satisfiability of such formulas decidable.

It is important to note that many of the applications that use SMT involve different data types (Barrett and Tinelli, 2018). Therefore, SMT usually works with a *sorted* (typed) version of first order logic (Manzano, 1993). Essentially, in SMT there exists a finite set of sort symbols (types) S and an infinite set of variables X for the (sorted) formulas, where each variable has a unique associated sort in S. This is an oversimplification of a many-sorted first order logic (MSFOL). As MSFOL is useful to express our formulas of interest, in the next subsection we provide a formal definition of its syntax (Finkbeiner and Zarba, 2006; Barrett and Tinelli, 2018; Barrett et al., 2009).

2.2.1 Many-sorted First-order Logic Syntax

A *signature* is a tuple $\Sigma = (S, C, F, P)$, where S is a non-empty and finite set of sorts, C is a countable set of constant symbols whose sorts belong to S, F and P are countable sets of function and predicate symbols correspondingly whose arities are constructed using sorts that belong to S. Predicates and functions have an associated arity in the form $\sigma_1 \times \sigma_2 \times \ldots \times \sigma_n \to \sigma$, where $n \geq 1$ and $\sigma_1, \sigma_2, \ldots, \sigma_n, \sigma \in S$.

A Σ-*term* of sort σ is either: (i) each variable x of sort (type) σ, where $\sigma \in S$; (ii) each constant c of sort (type) σ, where $\sigma \in S$; and (iii) $f \in F$ with arity $\sigma_1 \times \sigma_2 \times \ldots \times \sigma_n \to \sigma$, is a term of sort σ, thus, for $f(t_1, \ldots, t_n)$, t_i (for $i \in \{1, \ldots, n\}$) is a Σ-term of sort σ_i.

A Σ-*atom* (Σ-atomic formula) is an expression in the form $s = t$ or $p(t_1, t_2, \ldots, t_n)$, where $=$ denotes the equality symbol, s and t are Σ-terms of the same sort, $t_1, t_2, \ldots, t_n$ are Σ-terms of sort $\sigma_1, \sigma_2, \ldots, \sigma_n \in S$, respectively, and p is a predicate of arity

$\sigma_1 \times \sigma_2 \times \ldots \times \sigma_n$.

A Σ-*formula* is either: (i) a Σ-atom; (ii) if ϕ is a Σ-formula, $\neg\phi$ is a Σ-formula, where $\neg$ denotes negation; (iii) if both ϕ, ψ are Σ-formulas, then, $\phi \wedge \psi$ and $\phi \vee \psi$ are Σ-formulas (likewise, the short notations $\phi \rightarrow \psi$ and $\phi \leftrightarrow \psi$ for $\neg\phi \vee \psi$ and $(\phi \wedge \psi) \vee (\neg\phi \wedge \neg\psi)$); finally, (iv) if ϕ is a Σ-formula and x is a variable of sort σ, then, $\exists x \in \sigma\ \phi$ ($x \in \sigma$ is used to indicate that x has the sort σ) is a Σ-formula (likewise, the short notation $\forall x \in \sigma\ \phi$ for $\neg\exists x \in \sigma\ \neg\phi$), where $\exists$ denotes the existential quantifier and $\forall$ denotes the universal quantifier, as usual.

We leave out the formal semantics of MSFOL formulas, their interpretations and satisfiability as we feel it can unnecessarily load the paper with unused formalism. However, we briefly discuss some aspects of MSFOL formula satisfiability. As previously mentioned, for some signatures, there exist decision procedures, which help to determine if a given formula is satisfiable. For example, consider the signature with a single sort $\mathbb{R}$, all rational number constants, functions $+, -, *$ and the predicate symbol $\leq$; SMT will interpret the constants, symbols and predicates as in the usual real arithmetic sense $\mathbb{R}$. The satisfiability of Σ-formulas for this theory (real arithmetic) is decidable, even for formulas with quantifiers (Barrett and Tinelli, 2018; Manna and Zarba, 2003), i.e., for some infinite domain theories, there exist procedures[1] to decide if a given quantified formula is satisfiable. Therefore, the satisfiability for formulas as: $\exists n \in \mathbb{R}\, \forall x \in \mathbb{R}\, x + n = x$ can be automatically determined (via a computer program implementing the decision procedure, i.e., an SMT solver). If a formula is satisfiable, there exists an interpretation (or model) for the formula, i.e., a set of concrete values for the variables, predicates and functions of the formula that makes this formula evaluate to TRUE.

3 DATA SET ENCODING AND FORMAL VERIFICATION

As previously mentioned (see Section 2), a ML data set is composed of a matrix $D_{m\times n}$ and a vector O_m, where m is the number of training examples, n the number of features, D contains the training examples, and O the expected outputs. However, note that in our definition of this matrix we never mentioned the type of each feature in the data set. In general, there is no theoretical limitation over the type of these features, nonetheless, for practical reasons, we consider that all features are real valued. The main reason is that otherwise additional information would be required for each of the features. Moreover, in practice, well-known libraries work with real-valued features. As usual, for those features which are not naturally real, an encoding must be found (for example, one hot encoding for categorical features, etc.). Thus, we consider that $d_{i,j}, o_i \in \mathbb{R}\ \forall i \in \{1,\ldots,m\}, j \in \{1,\ldots,n\}$. Additionally, we assume that O is always present in the data sets, independently if this data set is meant for supervised or unsupervised machine learning. If a data set is not *labeled*, then $\forall i,k \in \{1,\ldots,m\}\ o_i = o_k$.

Encoding a ML Dataset as a MSFOL Formula. Having a convenient formal description for a data set eases the encoding of this data set as a MSFOL formula. To encode the data as a formula, we make use of the theory of arrays[2]. We denote that an object a is of sort array with indices of type (sort) $\mathcal{T}1$ and holding objects of type $\mathcal{T}2$ as $a \in \mathbb{A}_{\mathcal{T}1,\mathcal{T}2}$. Indeed, a data set can be encoded using Algorithm 1; the algorithm creates a formula that is satisfiable by a model of arrays which represent the data set.

Algorithm 1: Data set encoding.

```
Input  : A data set $D_{M\times N}$ (with $N$ features and
         $M$ training examples), and its
         expected output vector $O_M$
Output: A MSFOL formula representation of
         the data set, $\phi$
Step 0: Set $\phi \leftarrow$ TRUE, set $labels \leftarrow$ARRAY(),
  and set $L \leftarrow 0$;
Step 1: Set
  $\phi \leftarrow \phi \wedge (m,n,l \in \mathbb{Z}) \wedge (m = M) \wedge (n = N)$;
Step 2: Set $\phi \leftarrow \phi \wedge (\mathcal{D} \in \mathbb{A}_{\mathbb{Z},\mathbb{A}_{\mathbb{Z},\mathbb{R}}}) \wedge (\mathcal{O} \in$
  $\mathbb{A}_{\mathbb{Z},\mathbb{R}}) \wedge (\mathcal{L} \in \mathbb{A}_{\mathbb{Z},\mathbb{R}})$;
Step 3: for $i \leftarrow 0; i < M; i \leftarrow i+1$ do
    Set $add \leftarrow$TRUE;
    for $j \leftarrow 0; i < N; j \leftarrow j+1$ do
        Set $\phi \leftarrow \phi \wedge (\mathcal{D}[i][j] = d_{i,j})$;
    Set $\phi \leftarrow \phi \wedge (\mathcal{O}[i] = o_i)$;
    for $k \leftarrow 0; k < L; k \leftarrow k+1$ do
        if $labels[k] = o_i$ then
            Set $add \leftarrow$FALSE;
    if $add$ then
        Set $labels[L] \leftarrow o_i$;
        Set $\phi \leftarrow \phi \wedge (\mathcal{L}[L] = o_i)$;
        Set $L \leftarrow L+1$;
Step 4: Set $\phi \leftarrow \phi \wedge (l = L)$ and return $\phi$
```

[1]Often such procedures seek to "eliminate" the quantifiers and obtain an equivalent quantifier-free formula

[2]The theory of arrays considers basic read and write axioms (Stump et al., 2001)

3.1 Formal Verification of Data Sets

Indeed, a data set can be formally defined as an MSFOL formula ϕ_{ds} which holds the following properties: ϕ_{ds} is a conjunction of *five* main parts, that is, i) the assertion that an integer variable m is of the size of the number of training examples, a variable n is of the size of the features and a variable l is of the size of the distinct labels, ii) the assertion that $\mathcal{D}$ is a two-dimensional (integer indexed) real-valued array (of size $m \times n$) and $\mathcal{O}, \mathcal{L}$ are integer indexed real-valued arrays (of size m, and l, respectively) iii) $\mathcal{D}[i][j]$ contains the j-th feature value for the i-th training example; iv) $\mathcal{O}[i]$ contains the expected output for the i-th training example; and, v) $\mathcal{L}[i]$ contains the i-th (distinct) label.

We assume that we want to verify k properties over the data set, and furthermore, that these properties are expressed also in MSFOL. Indeed, MSFOL allows to express many properties of interest (in Section 3.2 we showcase its expressiveness). Therefore, we assume that we are given $\pi_1, \dots, \pi_k$ MSFOL formulas to verify. These properties involve the variables in ϕ_{ds}. Additionally, we assume that these formulas should all *hold* independently over the data set, and their conjunction is *satisfiable*. Thus, impose a restriction that $\pi_x \wedge \pi_y$ is satisfiable, for $x, y \in \{1, \dots, k\}$; we call this set of properties the *data set specification* σ. This means that two properties may not *contradict* each other. For example, it cannot be required that the data set has more than 30 training examples and at the same time that it must have at most 20 ($(\pi_1 \leftrightarrow (m > 30)) \wedge (\pi_2 \leftrightarrow (m \leq 20))$). Additionally, the conjunction of properties must be satisfiable means that there is an interpretation that makes this formula (the conjunction) evaluate to TRUE, i.e., there exists a data set which can satisfy this specification. Otherwise, the verification of any data set is useless as no data set can hold such set of properties.

The Formal Data Set Verification Problem. can be reduced to the following: given a data set formula ϕ_{ds} (created using Algorithm 1 from D and O) and a data set specification $\sigma = \bigwedge_{l=1}^{k} \pi_l$, is $\phi_{ds} \wedge \sigma$ satisfiable? If the conjunction of these formulas is satisfiable then, each of the properties must hold for the data set as the conjunction of all properties is satisfiable by itself; if the conjunction is satisfiable we say that the data set *holds* the properties $\pi_1, \dots, \pi_k$ or that the data set *conforms* to the specification σ. Perhaps this is quite an abstract view of the problem. For that reason, in the following subsection we provide concrete examples that should help the reader to better understand.

3.2 Example Data Set and Properties

First, let us consider a very small data set:

$$D = \begin{pmatrix} 0.051267 & 0.69956 \\ -0.092742 & 0.68494 \\ -0.21371 & 0.69225 \\ -0.375 & 0.50219 \\ -0.51325 & 0.46564 \\ -0.52477 & 0.2098 \\ -0.39804 & 0.034357 \\ -0.30588 & -0.19225 \\ 0.016705 & -0.40424 \\ 0.13191 & -0.51389 \end{pmatrix}, O = \begin{pmatrix} 1 \\ 0 \\ -1 \\ -1 \\ -1 \\ -1 \\ -1 \\ -1 \\ -1 \\ -1 \end{pmatrix}$$

After applying Algorithm 1 to D and O as shown before, the output (ϕ_{ds}) is:

$$\begin{aligned}
&(m,n,l \in \mathbb{Z}) \wedge (m = 10) \wedge (n = 2) \wedge \\
&(\mathcal{D} \in \mathbb{A}_{\mathbb{Z},\mathbb{A}_{\mathbb{Z},\mathbb{R}}}) \wedge (\mathcal{O} \in \mathbb{A}_{\mathbb{Z},\mathbb{R}}) \wedge (\mathcal{L} \in \mathbb{A}_{\mathbb{Z},\mathbb{R}}) \\
&\wedge (\mathcal{D}[0][0] = 0.051267) \wedge (\mathcal{D}[0][1] = 0.69956) \\
&\wedge (\mathcal{O}[0] = 1) \wedge (\mathcal{L}[0] = 1) \\
&\wedge (\mathcal{D}[1][0] = -0.092742) \wedge (\mathcal{D}[1][1] = 0.68494) \\
&\wedge (\mathcal{O}[1] = 0) \wedge (\mathcal{L}[1] = 0) \\
&\wedge (\mathcal{D}[2][0] = -0.21371) \wedge (\mathcal{D}[2][1] = 0.69225) \\
&\wedge (\mathcal{O}[2] = -1) \wedge (\mathcal{L}[2] = -1) \\
&\wedge (\mathcal{D}[3[0] = -0.375) \wedge (\mathcal{D}[3][1] = 0.50219) \\
&\wedge (\mathcal{O}[3] = -1) \\
&\wedge (\mathcal{D}[4][0] = -0.51325) \wedge (\mathcal{D}[4][1] = 0.46564) \\
&\wedge (\mathcal{O}[4] = -1) \\
&\wedge (\mathcal{D}[5][0] = -0.52477) \wedge (\mathcal{D}[5][1] = 0.2098) \\
&\wedge (\mathcal{O}[5] = -1) \\
&\wedge (\mathcal{D}[6][0] = -0.39804) \wedge (\mathcal{D}[6][1] = 0.034357) \\
&\wedge (\mathcal{O}[6] = -1) \\
&\wedge (\mathcal{D}[7][0] = -0.30588) \wedge (\mathcal{D}[7][1] = -0.19225) \\
&\wedge (\mathcal{O}[7] = -1) \\
&\wedge (\mathcal{D}[8][0] = 0.016705) \wedge (\mathcal{D}[8][1] = -0.40424) \\
&\wedge (\mathcal{O}[8] = -1) \\
&\wedge (\mathcal{D}[9][0] = 0.13191) \wedge (\mathcal{D}[9][1] = -0.51389) \\
&\wedge (\mathcal{O}[9] = -1) \wedge (l = 3)
\end{aligned}$$

Let us start by showcasing very simple properties and how their formal verification works. Suppose the specification consists of a single property: "the data set must contain at least 100 training examples," this property can be expressed in MSFOL simply as $\pi_{\#} \leftrightarrow (m \geq 100)$. Notice how $\phi_{ds} \wedge \pi_{\#}$ is not satisfiable as there does not exist an interpretation that makes it evaluate to TRUE; particularly, if m is greater than 99, then the clause (in ϕ_{ds}) $m = 10$ cannot evaluate to

TRUE and since this is a conjunction, $\phi_{ds} \wedge \pi_{\#}$ evaluates to FALSE. Similarly, if m is 10, then the $\pi_{\#}$ makes the conjunction evaluate to FALSE. Thus, we say that the data set does not hold the property $\pi_{\#}$.

Let us start examining more complex properties that can be formally verified over the data set. A slightly more complex property to verify is: "the data set must be min-max normalized," which can be expressed in MSFOL as $\pi_{\pm} \leftrightarrow \nexists(i,j \in \mathbb{Z})((i \geq 0) \wedge (i < n) \wedge (j \geq 0) \wedge (j < m) \wedge ((\mathcal{D}[i][j] < min) \vee (\mathcal{D}[i][j] > max)))$. Certainly min and max are defined constants (e.g., -1 and 1) an either these variables must be defined or the value must be replaced; for $min = -1$ and $max = 1$, ϕ_{ds} holds the property $\pi_{\pm}$ (as $\phi_{ds} \wedge \pi_{\pm}$ is satisfiable).

The previous properties are useful to showcase how easy is to translate desired properties into the formalism. However, verifying such properties is quite simple, and furthermore can be uninteresting as the data set can be normalized later on, for example. As previously stated, our motivation comes from the proper extraction and collection of the data set. We have discussed the case where training examples are provided for some regions of the input space and some other regions are overlooked. To verify that "the data set is sampled across the whole input space," the following property can be verified $\pi_{*} \leftrightarrow \nexists(p \in \mathcal{A}_{\mathbb{Z},\mathbb{R}})\forall(i \in \mathbb{Z})((i \geq 0) \wedge (i < m)) \implies (\sqrt{\sum_{j=0}^{m-1}(p[j] - \mathcal{D}[i][j])^2} > \delta)$; the property basically states that there does not exist a point such that it has a greater Eucledian distance that a chosen constant δ. As an example, for $\delta = 1$, our example data set does not hold the previous property π_{*} as there exists a point in the input space that has greater Eucliden distance, for example if $p[0] = 2$ and $p[1] = 2$. Note that the property never specifies the minimum or maximum values of the input space and thus, it is likely that no data set is sampled over an infinite domain. An easy solution is to add such constraints to π_{*}, i.e., $\nexists(l \in \mathbb{Z} \wedge (l \geq 0) \wedge (l < n)(\wedge(p[l] > max) \vee (p[l] < min)))$, for given max and min constants. We draw the reader's attention to the fact that a formal specification must be well-stated and this is an assumption of our work and generally in any formal verification strategy.

Finally, note that sometimes it is more convenient to state negated properties. For example, to verify that the data set is *balanced*, we can verify the following property: "there is no class which has less than $\frac{m}{(\beta * l)}$ samples," where l is the number of different outputs (labels) and β is a chosen constant. This property states that the data set must have equal amount of samples, up to a given constant. For example, if $\beta = 1$ the data set must be perfectly balanced, while if $\beta = 2$ only half of the samples (of a perfectly balanced data set) are required per class. It is important to state that unbalanced data sets represent a real problem for current machine learning algorithms, and moreover, it is often encountered in the domain. Accordingly, researchers actively try to tackle the problem (see for example (Lemaître et al., 2017)). Indeed, it can be not that intuitive how to state this property in first order logic. There are many particularities that must be considered; for example, the fact that there is no notion of loops in first order logic and we require to define a function to count the number of instances where a given label appears. To overcome this particular problem a recursive function can be defined. In order to keep the paper readable, we avoid this definitions and simply denote defined functions in mathematical bold-font. The interested reader can refer to the prototype implementation section (Section 4) and correspondingly to the tool's repository to check the full property implementation. We state the aforementioned property as: $\pi_{\equiv} \leftrightarrow \nexists i \in \mathbb{Z}((i \geq 0) \wedge (i < l) \wedge (\mathbf{S}(O, \mathcal{L}[i], m) < \frac{m}{\beta * l}))$, where $\mathbf{S}(A, v, s)$ is a function that returns the number of times the value v is found in an array A up to index s; that is, that is how many times the label is found in the label array.

We have exemplified different properties that can be formally verified in data sets. We do not focus on an extensive list of properties but, rather on providing means for formally verifying any property in a given data set. We could state much more properties, for example, there are no *contradicting training examples* in the data set, i.e., there does not exist two equal elements in D with different indices for which the corresponding elements in O differ. We limit this section with these examples. However, we note that as shown in the previous examples, the formalism is quite flexible for expressing real properties of interest.

Discussion – On Standard and Domain-specific Properties. We have showcased the flexibility of the proposed approach with somewhat standard properties to check. One can imagine more of these properties, for example, guaranteeing that there are no outlier training examples[3] in the data set can be logically expressed as finding points in the space with high variance. Nonetheless, it is interesting to point out that the approach is generic and domain-specific properties coming from expert knowledge can be also used to formulate properties. For example, consider a real state data set, where two categorical features, `isHouse` and `isApt` cannot be both TRUE at the same

[3]Training examples which have extreme values, far from the rest of data points.

time. Similarly, for other specific domains as computer networks, where network packets cannot have mutually exclusive headers (e.g., transmission control protocol and user datagram protocol). For the first case (real state), the property can be simply stated as $\nexists i \in \mathbb{Z}((i \geq 0) \wedge (i < m) \wedge (\mathcal{D}[i][J] = \mathcal{D}[i][K]))$, assuming J and K are the corresponding indices for the `isHouse` and `isApt` features. In general, as the properties to check can be added or removed arbitrarily, checking a particular set of those for a particular data set is possible (see the tool's description in the next section).

Discussion – On the Usability and Computational Complexity of the Proposed Approach. The proposed approach reduces the problem to the satisfiablity of many sorted first order logic. The problem is known to be NP-hard (for formulas without quantifiers), and depending on the the theories involved the problem becomes computationally harder (PSPACE-hard or even exponential or doubly exponential). This fact may lead the reader to believe that the approach is not feasible for real life data sets. This is true only if the particular instances belong to the worst case. In reality, millions problems which are considered untractable are easily solved as instances do not fall into the worst case. Additionally, as most of the showcased properties can be individually computed via adhoc procedures (with small programs), it may seem as if the approach is cumbersome. However, note that not all properties are easily computable. Take for instance the property of guaranteeing that there does not exist a point in the space which is more distant (than a given constant) to all data set points; brute force enumeration is not feasible. Furthermore, some of the properties to be verified may be computationally hard by themselves. As we showcase in the next section, our preliminary experimental results show that small data sets are verified in milliseconds. It is interesting to observe that many of the properties can be verified by batches in parallel. For example, the property of verifying that the data set is min/max normalized can be checked independently as there are no dependencies between the data; in general, any property that does not relate two training examples. However, this is out of the scope of this initial work. Finally, we also note that our approach is targeted to properly construct a data set. However, it is still useful for existing data sets which cannot be modified. Even if the collection is finished, knowing that the data set holds a given property or not is the first step toward fixing or contouring the problem.

4 TOOL DEVELOPMENT AND EXPERIMENTAL RESULTS

In order to assess the feasibility and efficiency of the proposed approach, a prototype tool has been developed in Julia (Bezanson et al., 2017). Generally, speaking, the tool takes as an input: a Comma Separated Values (CSV) file as a data set, assuming that the last column of each row must be the expected output for the training example (remainder of the columns); a directory, where the properties to be checked are stored, one per file in the `SMT-LIB` language.

SMT-LIB. is a language that many SMT solvers can take as an input and its syntax is quite intuitive. For example, for expressing the property $\nexists(i, j \in \mathbb{Z})((i \geq 0) \wedge (i < n) \wedge (j \geq 0) \wedge (j < m) \wedge ((\mathcal{D}[i][j] < min) \vee (\mathcal{D}[i][j] > max)))$ can be simply done in SMT-LIB as shown in Listing 1.

Listing 1: $\pi_{\pm}$ in SMT-LIB.

```
(assert
 (not
  (exists ((i Int) (j Int))
   (and
    (>= i 0)
    (< i n)
    (>= j 0)
    (< j m)
    (or
     (< (select (select D i) j) min )
     (> (select (select D i) j) max )
    )
   )
  )
 )
)
```

The tool works as described in Algorithm 2. Note that, *SMT* is an SMT procedure call to determine if the given formula is satisfiable. In our tool, we use the z3 (De Moura and Bjørner, 2008) solver (which takes as an input the SMT-LIB format). The interested reader can check the properties stated in SMT and more information about our tool in the tool's repository (López, 2021).

4.1 Preliminary Experimental Results

All experiments were executed with commodity hardware with the intention to showcase the performance of the proposed approach. The experiments were performed with an Ubuntu 20.04LTS with 4 Intel(R) Core(TM) i5-6300U CPU @ 2.40GHz, and 8GB of RAM.

Algorithm 2: Data Set Verification.

```
Input  : A CSV data set file $f$ (with $n \geq 1$
         features, and $m \geq 1$ training
         examples), and a directory $d$
         containing property files
Output: Verdicts for each property $\pi \in d$
Step 0: Read $f$ and store it into the arrays $D$
  and $O$, and set $m$ and $n$, correspondingly;
Step 1: Use Algorithm 1 to obtain $\phi_{ds}$ from
  $D, O, m$, and $n$;
Step 2: foreach file $p \in d$ do
    Read the contents of $p$ into the formula $\pi$;
    if $SMT(\phi_{ds} \wedge \pi)$ is satisfiable then
        $display(\pi$ holds in the data set $f)$
    else
        $display(\pi$ does not hold for the data
          set $f)$
```

In order to evaluate the feasibility of our proposed solution, the properties $\pi_{\#}, \pi_{\pm}, \pi_{*}$ and $\pi_{\equiv}$ have been encoded in SMT-LIB, and a data set was incrementally tested. We present the results of both the performance and satisfiability of properties w.r.t. the data sets in Figures 1, 2, respectively. As can be seen, the performance of the proposed approach is acceptable; as any formal verification approach, the decision procedures are often exponential in the worst case. For formally guaranteeing that the data set holds certain properties of interest, this procedure can be executed once, in which case the running time is not much of a constraint. Our preliminary experimental evaluation shows that properties are solved fast (milliseconds per hundreds of training examples), specially simple properties (e.g., $\pi_{\#}$).

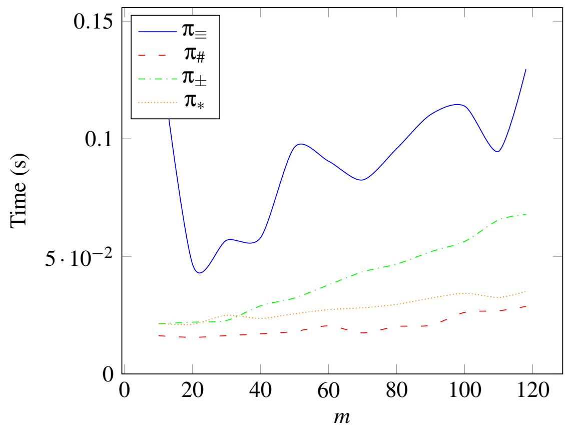

Figure 1: Performance of formal data set verification.

It is interesting to observe the satisfiability of the properties. It is normal that when adding more training examples the data set may get balanced or unbalanced ($\pi_{\equiv}$); it is also normal that all data sets which have less than 100 training examples fail the property $\pi_{\#}$. One can conclude that the example data set is also well min/max normalized as $\pi_{\pm}$ is always satisfiable. Finally, note that even if the language allows it and solver can read the property π_{*}, the property is very complicated as it is quantified over an array; the solver cannot process such complex formulation and so the property always returns an unknown status. We envision different strategies to overcome this problem. For example, instead of formulating the problem as it is, to pre-process the dimension of the training vector, and ask the formula quantified over n reals ($\exists p_1, \ldots, p_n \in \mathbb{R}\psi$). This should effectively reduce the complexity of the formula, however, this may require a Domain Specific Language (DSL) for stating properties of interest. However, note that this approach is out of the scope of this initial study.

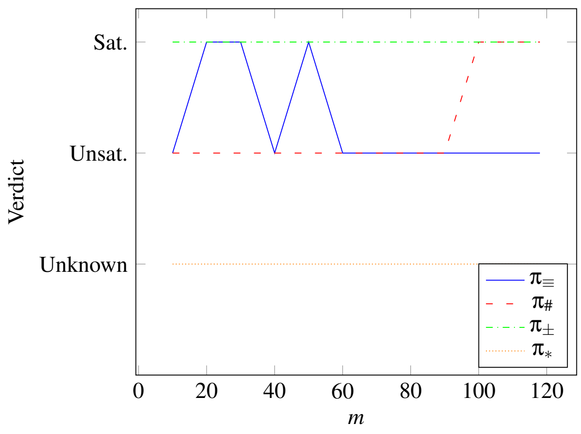

Figure 2: Satisifiability of properties (w.r.t the data set conjunction).

5 CONCLUSION AND FUTURE WORK

In this paper, we have proposed a formal data set verification approach. Such formal verification can be used for guaranteeing that the data extraction is adequate for properly training a machine learning model. We have showcased different formal properties to be verified over the data sets, and experimentally proven that the approach is feasible, and furthermore flexible w.r.t. the semantic capabilities of the proposed formalism.

As for future work we plan to test the performance of the approach on large scale data sets. Considering performance enhancement by automatically recognizing properties that can be verified by batches in parallel is another interesting direction. Also, we intend to further investigate DSLs for property specification (as discussed in Section 4). Additionally, as each of the training examples gets translated into a part of a formula, it is interesting to try to remove some training examples when a property is not satisfiable in order to obtain a satisfiable one; this would allow to automatically repair data sets w.r.t. a set of properties. Nevertheless, different elements must be taken into consideration, for example, the fact that the model found by the solver may include other training examples, which are *fictitious*. Finally, an interesting direction is to consider the formal verification of unstructured data for machine learning.

REFERENCES

Barrett, C., Sebastiani, R., Seshia, S., and Tinelli, C. (2009). *Satisfiability modulo theories*, volume 185 of *Frontiers in Artificial Intelligence and Applications*, pages 825–885. IOS Press, 1 edition.

Barrett, C. and Tinelli, C. (2018). Satisfiability modulo theories. In *Handbook of Model Checking*, pages 305–343. Springer.

Bezanson, J., Edelman, A., Karpinski, S., and Shah, V. B. (2017). Julia: A fresh approach to numerical computing. *SIAM review*, 59(1):65–98.

Carvallo., P., R. Cavalli., A., and Kushik., N. (2017). Automatic derivation and validation of a cloud dataset for insider threat detection. In *Proceedings of the 12th International Conference on Software Technologies - ICSOFT,*, pages 480–487. INSTICC, SciTePress.

De Moura, L. and Bjørner, N. (2008). Z3: An efficient smt solver. In *International conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 337–340. Springer.

Finkbeiner, B. and Zarba, C. G. (2006). Many-sorted logic. https://www.react.uni-saarland.de/teaching/decision-procedures-verification-06/ch01.pdf. Last Accessed: 2020-05-12.

Lemaître, G., Nogueira, F., and Aridas, C. K. (2017). Imbalanced-learn: A python toolbox to tackle the curse of imbalanced datasets in machine learning. *Journal of Machine Learning Research*, 18(17):1–5.

López, J. (2021). Dsverif – a formal data set verification tool. https://github.com/jorgelopezcoronado/DSVerif.

Manna, Z. and Zarba, C. G. (2003). Combining decision procedures. In *Formal Methods at the Crossroads. From Panacea to Foundational Support*, pages 381–422. Springer.

Manzano, M. (1993). Introduction to many-sorted logic. In Meinke, K. and Tucker, J. V., editors, *Many-sorted Logic and Its Applications*, pages 3–86. John Wiley & Sons, Inc., New York, NY, USA.

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., and Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

Stump, A., Barrett, C. W., Dill, D. L., and Levitt, J. (2001). A decision procedure for an extensional theory of arrays. In *Proceedings 16th Annual IEEE Symposium on Logic in Computer Science*, pages 29–37. IEEE.