Derivation of Constraints from Machine Learning Models and Applications to Security and Privacy

Moreno Falaschi, Catuscia Palamidessi, Marco Romanelli

Derivation of Constraints from Machine Learning Models and Applications to Security and Privacy

Moreno Falaschi
Università di Siena, Italy
https://www3.diism.unisi.it/~falaschi/

Catuscia Palamidessi
Inria, France
LIX, Ecole Polytechnique, Institut Polytechnique de Paris, France
https://www.lix.polytechnique.fr/~catuscia/

Marco Romanelli
Inria, France
LIX, Ecole Polytechnique, Institut Polytechnique de Paris, France
Università di Siena, Italy
http://www.lix.polytechnique.fr/Labo/Marco.Romanelli/

Abstract

This paper shows how we can combine the power of machine learning with the flexibility of constraints. More specifically, we show how machine learning models can be represented by first-order logic theories, and how to derive these theories. The advantage of this representation is that it can be augmented with additional formulae, representing constraints of some kind on the data domain. For instance, new knowledge, or potential attackers, or fairness desiderata. We consider various kinds of learning algorithms (neural networks, k-nearest-neighbours, decision trees, support vector machines) and for each of them we show how to infer the FOL formulae. Then we focus on one particular application domain, namely the field of security and privacy. The idea is to represent the potentialities and goals of the attacker as a set of constraints, then use a constraint solver (more precisely, a solver modulo theories) to verify the satisfiability. If a solution exists, then it means that an attack is possible, otherwise, the system is safe. We show various examples from different areas of security and privacy; specifically, we consider a side-channel attack on a password checker, a malware attack on smart health systems, and a model-inversion attack on a neural network.

2012 ACM Subject Classification Computing methodologies → Learning paradigms; Computing methodologies → Symbolic and algebraic manipulation; Security and privacy → Formal security models; Security and privacy → Privacy-preserving protocols; Security and privacy → Information flow control

Keywords and phrases Constraints, machine learning, privacy, security

Digital Object Identifier 10.4230/OASIcs.Gabbrielli.2020.11

Funding This work was supported by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme. Grant agreement № 835294.

Acknowledgements We thank the anonymous reviewers for their detailed suggestions and comments that helped us to improve the presentation of our paper.

1 Introduction

Machine learning (ML) is pervasive in nowadays society: systems based on this technology run in hospitals to help diagnose diseases, in cars to help avoid car accidents, in banks to evaluate loans and manage investments, at insurance agencies to evaluate coverage suitability and costs for clients.

Machine learning refers to the automated pattern detection from data and it is is strictly linked to the idea of "generalized learning" as opposed to "memorization learning". The latter is based on storing information in the shape of data and running a comparison between new input data and the memorized one in order to make a decision. In contrast, ML aims at finding complex patterns that not only fit the available data but also generalize to new input, going beyond the mere comparison. Therefore, ML is particularly useful for tasks in which decision rules are particularly difficult to be hard coded and when adaptivity is necessary, like for instance speech and hand writing recognition.

In this paper, we propose to combine the power of machine learning with the flexibility of constraint systems, which is one of the most successful methodologies for solving hard discrete optimization problems. Nowadays, constraint programming is a mature technology, there are very good commercial and open-source solvers, and the range of applications is quite wide. In order to exploit a coordinated combination of the best solvers, portfolios of collaborative solvers have been designed [4].

Recently, much research on constraint solving has been devoted to the satisfiability modulo theories (SMT) problem. SMT solvers can deal with the satisfaction problem of logical formulae formalized in first-order logic (FOL), often with equality, in combination with some background theories. Examples of background theories can be the theory of real numbers, integers, some data structures, etc. SMT solvers are useful for many applications, such as verification of programs and software testing based on symbolic execution. Several recent efficient SMT solvers are currently available, see for instance [5, 16].

As stated above, the goal of this paper is to combine ML and constraint systems. More specifically, we show how machine learning models can be represented by first-order logic theories, and how to derive these theories. We argue that this representing ML by FOL can have numerous applications:

First of all, the representation can help to understand the ML model (*explainability*), which, especially in the case of deep neural networks, could be quite mysterious. Providing an explanations of the decision taken by the system is important from the point of view of the users, especially since these decisions are often critical for the concerned people (diagnosing the right disease, obtaining a loan, etc.).

Second, the representation of ML models in terms of constraint provides an *automatic* way of creating new knowledge, since the learning process in ML is totally automatized and simply consists in applying an algorithm to the available data (*training data*). Since this new knowledge is represented in terms of constraints, it can be processed, queried, checked for satisfiability or implications, etc..

Third, the representation can be augmented with additional formulae, representing constraints of some kind on the data domain. For instance, new knowledge, or potential attackers, or fairness desiderata.

In this paper, we focus on this latter application domain. In particular, we show how our proposed approach can be used to detect potential security and privacy breaches on the ML system, or to prove that they do not exist and the system is therefore secure. The idea is to represent the system as a FOL theory, and the specific user and the attacker as a set of constraints. A SMT solver can then be applied. The existence of a solution then reveals the possibility of an attack, and its nature. The non satisfiability means that the system is secure[1].

[1] It could happen that the SMT solver is not able to find a solution, but cannot prove the non-satisfiability either. For the more complicated theories, indeed, it could happen that the SMT is not decidable, which

1.1 Related work

The relation between constraint rules and automatic learning is a widely investigated topic in the fields of artificial intelligence and machine learning. In particular, large bodies of work have explored the possibility of extending the ability of learning from data to learn constraints (see, among others, [19, 24, 34, 35])[2]. In other words, while in many applications a trained model is considered as a black-box that creates a descriptive representation of the problem by summarizing the knowledge coming from data, in this case, the focus is on the symbolic interpretation of the model by extracting rules from the model itself (cfr. [9, 30, 43]).

According to the popular framework of *learning from constraints*, it is possible to reinterpret the learning theory based on supervised learning. For instance, the empirical risk minimization approach (cfr. 2.1) can be seen as learning by constraining the error between prediction and supervision to be minimal. Through first order logic (FOL) it is possible to model families of logic constraints which, via the so called task functions, can be mapped onto the *real valued* constraints which are typical of machine learning. The idea was first introduced in [28] and then it has been adapted to kernel base machine learning models in [19].

Building on these basic notions, the works in [25] and [31] propose a solution to learn constraints from observable samples and solve search optimization problems for which the constraints either are not given or need to be estimated. The solution is based and builds on the idea of using data to select constraints from a finite domain, a problem that was already addressed in [6]. In [34] and [35], the authors tackle the problem from an inductive logic programming (ILP) standpoint, building on the framework first introduced in [28]. On the one hand, they propose, once again, to learn constraints within a finite domain which can be modeled by a certain language $\mathcal{L}_c$. On the other hand, by using ILP, they work in a specific machine learning in which first-order logic is used to represent the data as well as the learned hypotheses which, in turn, can be expressed via $\mathcal{L}_c$. Therefore, the solution to the problem can be reduced to finding a particular hypothesis $\in \mathcal{L}_c$ such that it holds for all the positive samples and for none of the negative samples. It is important to notice that the fact that the hypothesis is to be selected as one possible choice within a set of hypothesis recalls the typical machine learning requirement of probably approximately correct (PAC) learning [44].

The common denominator of the works referenced so far is that the only constraints that are involved in the learning are somehow known a priori, and the learning involves taking a decision on which constraints represent the hypotheses learnt from the samples.

In [13], a different framework, which represents a step in a new direction, is introduced. As in the previous work (cfr. [25, 31, 34, 35]), the authors aim at learning constraints from a learnable set. However, they propose to use information theoretic principles (maximizing the information transfer from the concept space (hypotheses) to the rule space (constraints)), and to model the constraints as neural networks. Doing so, they realize that this process leads to the unsupervised development of new constraints that they analyze from the standpoint of FOL.

[27] investigates how to optimise the ML process, and has some similarities with our

means that in some cases neither the satisfiability nor the unsatisfiability can be proved. In this case, however, the SMT solver should return a warning, and in this case the diagnosis of the security of the system is not conclusive, but at least we know it. Namely, the approach is correct, even though it may not be complete.

[2] A meaningful distinction must be highlighted between this and the idea of using constraints to drive the learning of a model, for instance by including constraints in the optimized loss functions in a way that recalls the minimization (maximization) of an objective functions according to constraints [36].

approach as they present some coding of Artificial Neural Network and Decision Trees in Local Search, Constraint Programming, Mixed Integer Non-Linear Programming (only ANNs) and SAT Modulo Theory (only DTs).

1.2 Structure of the paper

In Section 2 we present some preliminary technical definitions on Machine Learning, Constraint systems, and SMT solvers. Then, in Section 3 we consider various machine learning algorithms and we show how the resulting model can be represented as a FOL theory. In Section 4 we show how to apply our methodology to some examples from the field of security. In Section 5 we consider one application to a model-inversion attack. Finally, Section 6 draws some conclusions and discusses some future work.

2 Preliminaries

2.1 Machine learning

We give here a brief introduction to the learning process and the derivation of the model. We will focus on the supervised learning scenario in the context of classification problems, which cover all examples considered in this paper. We describe the basic elements common to all learning algorithms, and to this purpose we introduce a generic learner model based on a well established statistic framework. The details specific to the various algorithms will be described in the next section.

A learning problem is defined by:

- a domain $\mathcal{X}$ of objects, represented as a vector of *features* (aka *attributes*) that we would like to classify;
- a set of *labels* (aka *classes*) $\mathcal{Y}$;
- a set of training data, i.e., a sequence $\mathcal{S} = ((\vec{x}_1; y_1) \dots (\vec{x}_m; y_m))$ of pairs in $\mathcal{X} \times \mathcal{Y}$;
- a correct labelling function $f : \mathcal{X} \to \mathcal{Y}$, such that, for all i, $y_i = f(\vec{x}_i)$;
- a distribution $\mathcal{D}$ of type $\mathcal{X} \times \mathcal{Y}$, according to which the samples are generated;
- the *prediction rule* or *hypothesis* $h : \mathcal{X} \to \mathcal{Y}$, that can be used to predict the label of new domain points;
- a measure of success that quantifies the predictor's error.

Ideally, the goal of the learning process is to select an h that minimizes the *risk*, defined as:

$$L_{\mathcal{D},f}(h) \stackrel{\text{def}}{=} \mathbb{P}_{\vec{x}\sim\mathcal{D}}\left[h(\vec{x}) \neq f(\vec{x})\right], \tag{1}$$

which represents the *expected probability* ($\mathbb{P}$) of a mismatch between h and f, measured with respect to the distribution $\mathcal{D}$.

In practice however we cannot compute analytically the h that minimizes (1), because we do not have a mathematical description of $\mathcal{D}$. What we can do, instead, is to use the training set $\mathcal{S}$, that, being generated from $\mathcal{D}$, represents an approximation of it. Then h is selected so to minimize the *empirical risk* over m samples, which is defined as:

$$L_{\mathcal{S}}(h) \stackrel{\text{def}}{=} \frac{|\{i \in [m] : h(\vec{x}_i) \neq y_i\}|}{m}. \tag{2}$$

This principle is called *empirical risk minimization* (ERM). The way this minimization is achieved depends on the specific algorithm, and the function h that is derived is called *model*.

For an extended discussion of the topic as well as a more complete overview of the learning problem we refer to [41, 44]. For further information about ML and the most

popular algorithms and applications we refer to [21], while for a more theoretical and statistical background on the learning problem we refer to [15, 22].

2.2 Constraints and SMT solvers

In this paper we consider constraint systems as first order logic formulae. We note that there are alternative approaches in the literature, for instance using Scott's information systems [40, 39].

Following [23], we define a constraint system as a 4-tuple $(\Sigma, \mathcal{D}, \mathcal{L}, \mathcal{T})$, where Σ is a signature, $\mathcal{D}$ is a Σ-structure, $\mathcal{L}$ is a set of Σ-formulae, and $\mathcal{T}$ is a first order Σ theory.

The idea is that Σ defines the syntax of the functions and predicates with their arities, $\mathcal{D}$ is the (Σ-)structure on which the computation is performed, and which allows to give a semantic interpretation to the functions and predicates defined in Σ, $\mathcal{L}$ are the constraints which can be syntactically expressed, and $\mathcal{T}$ is an axiomatization of some properties of $\mathcal{D}$.

The pair $(\mathcal{D}, \mathcal{L})$ is called a *constraint domain*.

We make the following assumptions:

- the terms and constraints in $\mathcal{L}$ are defined in a first-order language.
- the binary predicate symbol $=$ is always in Σ and is interpreted as the identity in $\mathcal{D}$.
- there are two constants $true$ and $false$ in $\mathcal{L}$ which are respectively identically true and identically false in $\mathcal{D}$.
- the set of constraints $\mathcal{L}$ is closed under variable renaming, conjunction, and existential quantification.

Examples of constraint domains [23] include, for instance, sets of linear equations and/or inequations over real numbers, the domain of word equations on strings, the finite domains over integers, where linear equations/inequations are built over variables which can assume values on intervals of integers, boolean constraints, constraints over finite trees (namely logic programming syntactic equations on data terms) etc.

We assume that our domains support a test of *consistency* or satisfiability. So we assume that it is possible in any moment to perform the following check:

$$(\mathcal{D}, \mathcal{L}) \vdash c \tag{3}$$

meaning that there exists a solution for the conjunction of constraints c, or that the variables in c can be instantiated in such a way to be solvable.

We also assume that we can perform an *entailment* (or *implication*) test of one constraint c' by another one c:

$$(\mathcal{D}, \mathcal{L}) \vdash c \Rightarrow c'. \tag{4}$$

Sometimes we use the equivalent notation $c \vdash c'$.

A conjunction of constraints can be simplified by using several formal techniques, maintaining the same set of solutions for the constraints to be solved.

Not for all constraint domains are available solvers for all possible cases, as the problem might be undecidable in the general case. In this case the solution of some constraints can be delayed until (and if) they become easier and (possibly) solvable.

Several refined techniques for solving constraints have been defined. We mention just a few of them:

- constraint propagation, exploiting local consistency conditions of subsets of constraints. This normally allows to reduce the search space of solutions.

- constraint simplification, trying to replace constraints by equivalent ones easier to solve.
- backtracking search, which allows to solve incrementally a subset of the constraints, backtracking on some assignments as soon as the set of constraints clearly gets not solvable.
- local search, which tries to modify a value of a variable at each step, choosing assignments close to the previous one in the search space.

For a more extensive overview please consult [38].

2.2.1 SMT solvers

SMT solvers are decision procedures for satisfiability of fragments of first-order logic with equality, where variables range over SMT data types, such as Booleans, integers, and reals. Satisfiability Modulo Theories (SMT) problem for a theory T [29] expressed as a set of closed first order formulae, can be intuitively defined as follows: given a formula F (it can be a propositional formula, or a ground formula in first order logic, or a formula in first order logic), determine whether F is T-satisfiable, i.e., whether there exists a model of T that is also a model of F. A formula F is T-satisfiable or T-*consistent* if $F \wedge T$ is satisfiable in the first-order sense. If the theory T is empty the problem reduces to satisfiability of a set of propositional/first order logic formulae. SAT solvers are available for this subproblem. A lot of research has been devoted to develop efficient SMT solvers [29]. There exist eager or lazy approaches. In eager approaches the input formula is transformed in a satisfiability equivalent one, usually in conjunctive normal form and then SAT solvers are applied. In a lazy approach the atoms of T are considered as propositions by the SAT solver. If the SAT solver returns a propositional model M of F, then this assignment (seen as a conjunction of literals) is checked by a T-solver. If M is found T-consistent then it is a T-model of F. Otherwise the process restarts. Incremental techniques, theory propagation and simplification can be also applied. Some examples of efficient SMT solvers are CVC4 [5], Yices 2 [16] and Z3 [14]. It is also possible to use efficient open source constraint programming solvers containing constraint solvers integrated with SAT modules, such as Chuffed [11] and OR-tools [32].

3 FOL theories from machine learning models

In this section we consider various machine learning algorithms and we show how the resulting model can be represented as a FOL theory.

3.1 Decision trees

Decision Trees (DTs) are models of supervised learning rather simple to understand and to interpret, also thanks to their graphical representation as mathematical trees [33]. They are predictive models, i.e., they allow to derive the value of a target variable from the value of the features of a given sample, and they are of two main kinds: *classification trees*, if the target variable is categorical, or *regression tree* if the predicted outcome is a real number. Here we consider the first case.

In general, DTs are constructed via an algorithmic approach that tries to identify the best ways to split the data set according to various criteria. To find the optimal tree, however, is a NP problem, so usually a greedy approach is used, which, at each step, identifies a feature and a test on that feature. In general, we choose the feature that gives the best *information gain* at that point of the process, which should help to keep the weighted tree balanced, at

least locally. The feature is then associated to an intermediate node of the tree[3] and the possible outcomes of the test, that determine a partition on the data, are associated to the subtrees children of that node, and label the corresponding edges. The process goes on until we reach a unique classification for the target variable of the remaining data, which is then associated to a leaf. Figure 1a shows an example of decision tree where each node x_i is submitted to a binary test, i.e., whether it satisfies or not a certain property P_i. More in general however, the test could have more than two outcomes; the important thing is that they are mutually exclusive and cover the whole range of possibilities.

The representation of the tree in terms of constraints is defined as follows. The constraint system $(\Sigma, \mathcal{D}, \mathcal{L}, \mathcal{T})$ is such that:

- Each sort of Σ represents a feature, plus there is a sort for the class. The values of the features and of the class are represented in Σ by symbols of the corresponding sort. Furthermore Σ contains the propositional symbols corresponding to the properties decorating the edges of the tree.
- For each feature x, Σ contains a distinct variable X of the sort corresponding to x. Furthermore, Σ contains an additional variable L of sort class.
- $\mathcal{D}$ is a structure containing the domains of the features values, the domain of the class values, and the properties that decorate the edges. Because of the way the tree is constructed, each property is satisfied by at least one feature value.

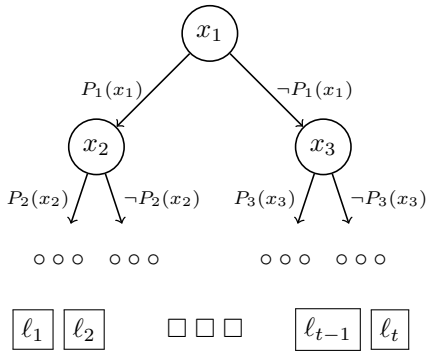
In order to define the theory $\mathcal{T}$ we introduce the following notation: Given a path γ from the root to a leaf, we denote by γ_i the proposition associated to the i-th edge in the path, and let x_i be the node just before that edge. Let ℓ be the label decorating the leaf of that path. Then, for each path γ, we assume that $\mathcal{T}$ contains the following formula.

$$(\bigwedge_i \gamma_i(X_i)) \Rightarrow L = \ell \tag{5}$$

For example, the set of formulae constructed from the binary tree of Figure 1a is shown in Figure 1b. If the properties decorating the edges are equalities or negations of equality, then we don't need to add anything else in $\mathcal{T}$. Otherwise, we need to define the meaning of the propositions. For finite domains this can be done by adding to $\mathcal{T}$ all the statements of the form $P(v)$, where P is a proposition and v a value symbol, whenever the corresponding property holds for that value. Otherwise, we should enrich the theory with formulas defining P. For instance if P is an ordering $\leq$ on numbers, we should incorporate in the constraint system the axioms defining this relation.

The set of constraints $\mathcal{L}$ depends on the potential attack and on the victim. In general the victim is the object that we want to classify, hence it is specified by a tuple $(v_1, v_2, \ldots, v_n)$ representing the value of each feature. The constraints relative to the victim are therefore equalities of the form $X_i = v_i$. As for the attacker, it is characterized by its prior knowledge and its capabilities, usually consisting of the properties on the features that the attacker knows or can infer from observing the system, Hence, typically the attacker can be represented by FOL contraints constructed on the $\gamma_i(X_i)$'s atomic formulas. In Sections 4 and 5 we will see examples of attacks and how to derive the corresponding constraints.

[3] A generalization of this method consists in associating to a node a set of features, and in this case the test represents a relation among the features. Here however we consider only the simpler case of a single feature per node.



(a) A binary decision tree.

$$P_1(X_1) \wedge P_2(X_2) \wedge \ldots \wedge P_l(X_l) \implies L = \ell_1$$

$$P_1(X_1) \wedge P_2(X_2) \wedge \ldots \wedge \neg P_l(X_l) \implies L = \ell_2$$

$$\vdots$$

$$\neg P_1(X_1) \wedge \neg P_3(X_3) \wedge \ldots \wedge P_r(X_r) \implies L = \ell_{t-1}$$

$$\neg P_1(X_1) \wedge \neg P_3(X_3) \wedge \ldots \wedge \neg P_r(X_r) \implies L = \ell_t$$

(b) The formulae derived from the decision tree.

Figure 1

3.2 Support vector machines

Support-vector machines are able to support both classification and regression problems. Their foundation relies on the theory of Vapnik and Chervonekis [8].

In this learning method, each object is represented as a point in the n-dimensional space, where n is the number of features. The idea behind the (linear) support vector machine is to construct a hyperplane or a set of hyperplanes of dimension $n-1$, which partition the space in such a way that, ideally, each subspace contains only elements of the same class. In practice however a perfect partition is usually not possible, so the presence of elements of different classes is tolerated, we just try to minimize their number[4].

A hyperplane is described by a formula like

$$\vec{w} \cdot \vec{x} - a = 0 \tag{6}$$

where $\vec{w}$ is a vector of numerical coefficients, and a is a numerical constant. Both $\vec{w}$ and a are determined by applying the minimization explained above.

For instance, in a binary classification problem, we would determine $\vec{w}$ and a so that as many elements as possible of class ℓ_1 are below the hyperplane defined by (6) ($\vec{w} \cdot \vec{x} - a < 0$), and as many elements as possible of class ℓ_2 are above it ($\vec{w} \cdot \vec{x} - a > 0$). An example is shown in Figure 2: no linear hyperplane can completely separate the classes ℓ_1 and ℓ_2, but the hyperplane h_2 is optimal.

To derive a constraint system from such model we proceed in a way similar to the one described in Subsection 3.1. In this case, however, Σ must contain also the symbols $+, \times$ (numerical addition and multiplication), and $<$ (strict ordering), and $\mathcal{T}$ must contain also the axioms defining these operations and the ordering relation. The specific axioms representing the classification are then:

$$\vec{w} \cdot \vec{X} < a \Rightarrow L = \ell_1 \qquad \vec{w} \cdot \vec{X} > a \Rightarrow L = \ell_2 \tag{7}$$

[4] One method to improve the result is to consider more features, which corresponds to increasing the number of dimensions. Another possibility is to consider also non-linear separation surfaces. A generalizazion of support vector machines in this sense has been proposed as well.

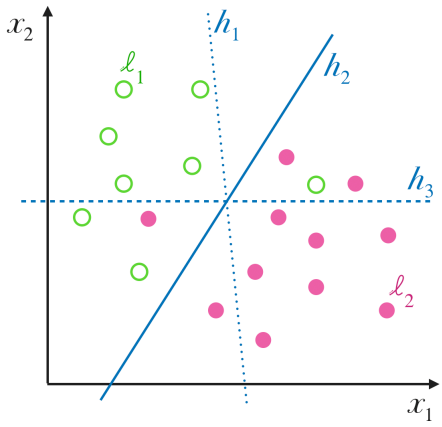


Figure 2 Separating the classes ℓ_1 and ℓ_2 by hyperplanes. Among the three hyperplanes h_1, h_2 and h_3, the one that produces the best discrimination is h_2. The resulting partition is not perfect but it is optimal, because no linear hyperplane can completely separate the classes ℓ_1 and ℓ_2.

In case of more than 2 labels, each of these clauses will contain a conjunction of premises of the form $\vec{w} \cdot \vec{X} < a$ and $\vec{w} \cdot \vec{X} > a$, representing discriminations operated by various hyperplanes. In some cases, which might be interesting for certain applications, failing to recognize a sample as belonging to a certain class, for instance ℓ_1, might be less severe than failing to recognize a sample as belonging to ℓ_2. A typical scenario is that of preliminary tests meant to quickly tell sick patients (ℓ_2) and healthy ones (ℓ_1) apart in order to assign the first ones to further tests. In this case, classifying an healthy person as sick is usually considered less severe than misclassifying a sick patient who needs to be thoroughly checked through further tests. Therefore, among two (or more) hyperplanes providing the same overall misclassification rate (i.e. number of wrongly classified samples over the total amount of samples), the one(s) with the lowest misclassification rate on the samples of class ℓ_2 is to be preferred.

3.3 Nearest neighbors

The k-nearest neighbors algorithm (k-NN), where k is a numerical parameter, is a learning method proposed by Thomas Cover that supports both classification and regression [2]. In both cases we assume that the space of the features is equipped with a notion of distance d. The basic idea is the following: every time we need to classify a new sample, we find the k samples in the training set whose features are closest to those of the new one (nearest neighbors). Once the k nearest neighbors are selected, a majority vote over their class labels is performed to decide which class should be assigned to the new sample. See Figure 3 for an example.

To derive a constraint system from such model we proceed in a way similar to the one described in Subsection 3.1. In this case, however, Σ must contain also the symbol d (distance), and $\leq$ (ordering), and $\mathcal{T}$ must contain also the axioms defining the notion of distance and the ordering relation. Let us consider for simplicity the case $k = 1$. The specific axioms representing the classification are as follows: for any tuple $(\vec{v}, \ell)$ in the training set:

$$\forall \vec{w}.\ d(\vec{X}, \vec{v}) \leq d(\vec{X}, \vec{w}) \Rightarrow L = \ell. \tag{8}$$

Note that the training set is finite, hence the quantification on $\vec{w}$ is done over a finite domain,

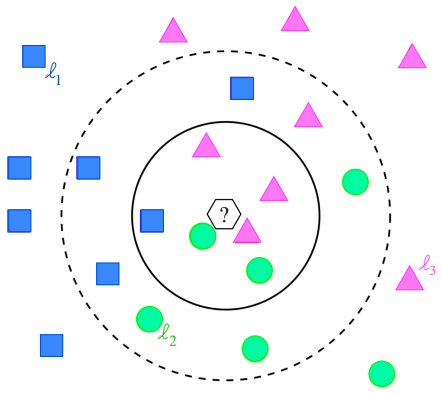


Figure 3 Classifying a new sample '?' using k-NN. If $k = 1$ the new object will be classified as ℓ_2. If $k = 6$ it will be classified as ℓ_3. If $k = 13$ it will be classified as ℓ_2 again.

which means that we can eliminate it by reducing to a conjunction of formulae. Note also that we need to take a bit of care to avoid conflicts in the classification where there are two or more samples that are at the same minimal distance from the new object. In this case, we must establish some priority among samples, and add to $\mathcal{T}$ only the clause that sets the labeling along with the sample of highest priority.

The above idea can be extended to the case of a generic k. For any set of k tuples $\{(\vec{v}_1, \ell_1), \ldots, (\vec{v}_k, \ell_k)\}$ in the training set, let ℓ be the most frequent among the labels $\ell_1, \ldots, \ell_k$ (again, in case of equality we must establish some kind of priority). The axioms are:

$$\forall \vec{w} \neq \vec{v}_1, \ldots \vec{v}_k.\ (d(\vec{X}, \vec{v}_1) \leq d(\vec{X}, \vec{w}) \wedge \ldots \wedge d(\vec{X}, \vec{v}_k) \leq d(\vec{X}, \vec{w})) \Rightarrow L = \ell$$

where the quantification $\forall \vec{w} \neq \vec{v}_1, \ldots \vec{v}_k$ is to be replaced by a conjunction of all the formulae in which $\vec{w}$ is different from $\vec{v}_1, \ldots \vec{v}_k$.

3.4 Neural networks

Artificial neural networks (ANNs) are computing systems inspired by the biological brains [7, 21, 22]. An ANN consists of a collection of connected units called nodes, simulating the behavior of neurons. The connections are called edges. Each node, like the synapses in a biological brain, can receive, process, and transmit a signal to the nodes connected to it. The "signal" is a real number, and the output of each node is computed by some function of its inputs. Nodes and edges typically have a weight that increases or decreases the strength of the signal at a connection, and is computed during the learning phase. Typically, neurons are aggregated into layers, and signals travel from the first layer (the input layer), to the last layer (the output layer).

The relation between neural networks and constraints has been the object of investigation of several works already, especially in the context of the *learning from constraints* paradigm, in which the idea is to use additional knowledge, represented as constraints, to guide and refine the learning process. Two of these works, [13] and [12], provide also an interesting point of view on the problem of *learning explainability*, which focuses on the interpretation of the decision making process of the neural black-box models.

The key idea for this approach is to realize a mapping from the real valued functions represented by neural networks models to the space of constraints. Let us consider an

input space $\mathcal{X}$ of dimension n, i.e., the space of the samples fed to a net. A learning environment for a multi-task (aka multi-label) classification problem can be defined as a vector $f = [f_1, \ldots, f_t]$, where $f : \mathcal{X} \subset \mathbb{R}^n \to \mathcal{Y} \subset \mathbb{R}^t$ and t is the number of classes. The environment of the constraints can be defined as $\phi = [\phi_1, \ldots, \phi_c]$, where $\phi : \mathcal{Y} \subset \mathbb{R}^t \to \mathcal{Z}^c$, and c is the number of constraints. Typically each constraint ϕ_j is embedded into a non-negative penalty function $\widehat{\phi}_j$ for $j = 1, \ldots, c$, so that the optimal learning environment, f^* can be defined as

$$f^* = \operatorname*{argmin}_f \sum_{j=1}^{c} \sum_{x_k \in X_{\phi_j}} \widehat{\phi}_j(f(x_k)) + \gamma_f, \tag{9}$$

where γ is a regularization term associated to f and X_{ϕ_j} is the sample space associated to the j-th constraint. For example, f^* could be defined in terms of the classical error minimization, which corresponds to imposing $\widehat{\phi}_j(f(x)) = (f(x) - y(x))^2$, where $y(x)$ represents the supervision. However, the role of ϕ_j in the considered learning framework is more general, since ϕ_j could represent different types of knowledge expressed by means of FOL formulae. In [19] the authors discuss how to use directly constraints as FOL, so to transfer logic knowledge to neural networks models.

The authors of [13] refine the above paradigm and their system can actually automatically learn new constraints ψ, initially modeled in terms of numerical functions of type $\mathbb{R}^n \to [0, 1]$. Then, they show how to convert them into logical formulae, so to obtain a symbolic representation of this new knowledge. The main idea consists in approximating the input of each neuron with the vertices of the Boolean hyper-cube, while the neuron output is approximated with a Boolean value.

The work [12] also refines the above paradigm, and focuses on how to interpret the outcome of the network in term of symbolic constraints, but the approach is quite different from the one in [13]. Specifically, their proposal consists in introducing another neural network that operates in the output space of the classifier, and whose purpose is to build the formulae that represent the explanation of the classifier. The two networks are trained jointly in the learning process, thus implicitly introducing a latent dependency between the development of the explanation mechanism and the development of the classifier.

Both these approaches can be used for our purposes. We refer to the corresponding papers [12] and [13] for the details.

4 Applications to Security

In this section we consider some examples of applications to security. The idea is to model the attacker's capabilities, its prior knowledge (aka *side knowledge*), and its goals in terms of constraints. Then, consider the union of these constraints and the theory coming from the machine learning model and use the SMT module to check the existence of a solution. In the negative case we can conclude that the system is safe. Otherwise, a threat exists, and the solution produced by the SMT provides the description of the potential attack and the level of vulnerability.

We will consider two examples from two different areas of computer security: information flow and malware.

4.1 Information flow

Secure information flow is concerned with the inference of secrets from information made publicly available by the system, or anyway, information that the attacker is able to obtain

by observing its behavior. Typically, these are physical observations made during a run, such as the execution time, the level of energy consumption, etc., and the corresponding kind of security breach is called *side channel attack*. The leakage of information in these situations is due to the correlation between the secret and the observable, and, by definition, it cannot be prevented using the typical security defenses, such as encryption or access control: it usually requires a re-thinking of the system architecture. Given the situation, a verification of the system in terms of constraints can be useful not only to show that the system is safe, but also, in case of the existence of a breach, to indicate its cause and help redesigning the system so to eliminate it.

The complete lack of any information leakage is called *non-interference*, and has been an active area of research since the seminal paper of Goguen and Meseguer [20]. However, in more recent times, it is recognized that for practical systems the non-interference property is usually impossible to achieve, because some correlation between the output and the secret is inherent to the specification of the system and therefore cannot be totally eliminated. For example, a password checker always reveals a little bit of information about the secret password: in case of success we know that the password is the string we have entered, and in case of failure we know that the password must be different from the string we have entered. Therefore, a more significant analysis is not to detect the existence of a leak, but, rather, its magnitude. These considerations have given rise to the "modern" approach to secrecy called *quantitative information flow*, which focuses on measuring the amount of leakage and the threat that it implies. The metrics used for the measurements are usually based on probabilistic aspects (typically, the probability of a successful attack) or on the complexity of the attack (typically, the number of times the attack needs to be repeated to ensure success) [3].

The constraint-based approach we propose can be used for verifying non-interference by proving that the capabilities of the attacker do not allow to derive any information about the value of the secret. This can be done by showing that there is no correlation between the properties on the features accessible to the attacker and the value of the secret (an implication from properties to values has no solution). More interestingly, our approach can also be used to *measure* the leakage of information, by computing the number of possible values for the secret from the point of view of the attacker. This number can then be interpreted as the probability of a successful attack (in one-try scenario) or as the complexity of the attack (in a repeated-try scenario). We show here an example of the latter in the form of a timing attack to a password checker[5]. A timing attack is a particular kind of side-channel attack based on the information that can be derived from the execution time.

We consider a password of n bits, and we assume that the checker takes in input a string of n bits $b_1 b_2 \dots b_n$ from the user (who could be an attacker trying to crack the password), and checks it, bit by bit, against the stored password $p_1 p_2, \dots p_n$. We assume that the system stops as soon as it finds a mismatch. Namely, at each run the system performs k steps, with $1 \leq k \leq n$, where k is the position of the first bit that does not match the corresponding bit in the password. If all bits match, then $k = n$ and the system output *success*, otherwise *fail*. Namely:

$$k \stackrel{\text{def}}{=} \begin{cases} \underset{i}{\text{argmin}}\,(b_i \neq p_i) & \text{if } \exists i.\, b_i \neq p_i \\ n & \text{otherwise} \end{cases} \tag{10}$$

[5] This is a toy example, but it illustrates well a typical class of realistic and practical timing attacks, namely those against encryption keys. We have chosen a simpler example so to avoid introducing cryptographic notions, which are orthogonal to the issues considered here.

It is well known that such behavior produces a security breach. In fact, if an attacker is able to observe the exact execution time, it can infer correctly a password prefix: for $1 \leq k \leq n$, a k-steps run with output *fail* means that the first k bits of the password are $b_1 b_2 \dots b_{k-1}(b_k \oplus 1)$, where $\oplus$ is the sum modulo 2. (A n-steps run with output *success* means, of course, that the password is the same as the string.) As a consequence, the complexity of the attacks reduces from exponential to linear (in n), in the sense that after at most n re-iterations of the attack, the adversary can infer the entire password.

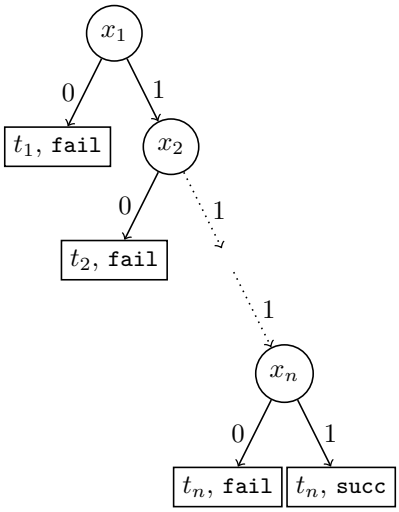
One technique to mitigate the security breach is the so-called *bucketing* [26], which consists in letting the system run for a longer time, so to avoid revealing the exact step where the first mismatch occurs. More precisely, the bucketing technique partitions a system's possible execution times into intervals, called buckets, of variable length. Given the number of steps k as defined above, the system waits until the upper bound of the bucket containing k, then it ends the computation and returns the result. For instance, if all buckets have size h then the system stops after m time units, where m is the smallest number which is greater than or equal to k, and is a multiple of h. Often, however, the buckets are of different sizes, to further confuse the adversaries and/or to optimize the trade-off between execution time and security.

Suppose that we have a password checker produced by an untrusted third party, and we want to check how secure it is. We can interact with the system, but we don't know its code. Namely, we are in a *black box* scenario. In [10, 37] it was shown how to estimate the leakage of information in a black box situation using machine learning. In short, by interacting with the system the analyst collects a set of examples, representing pairs secret-observables, and uses them to train a classifier that, given an observable, tries to infer the corresponding secret. The expected error of the classifier gives a quantitative estimation of the threat.

We present here an approach alternative to [10, 37], based on the idea of representing the machine learning model, and the attacker, by constraints. The simplest model for our purposes, in the case of this example, is the decision tree. Assume that the features x_1, x_2, $\dots$, x_n, express the result of the comparison of the string bits against those of the password, i.e., $x_i = 0$ if $b_i \neq p_i$, and $x_i = 1$ otherwise. Then, the model is represented by the tree in Figure 4a, where (not necessarily distinct) values t_1, t_2, $\dots$, t_n indicate the time units (labels). The corresponding set of constraints is given in Figure 4b.

We now consider how to represent an attacker. We assume that the adversary can give in input a string and observe the number of time units before the system stops, and repeat the attack until it infers the whole password. In a given iteration step during the attack, the relevant background knowledge consists of the password bits p_1, p_2, $\dots$, p_m that it has already discovered. Obviously, the best strategy for the adversary consists in entering a string of the form $p_1 p_2 \dots p_m b_{m+1} \dots b_n$. Hence, this iteration step of the attack can be represented by the disjunctive formula in Figure 5.

The union of the constraints in Figure 4b and Figure 5 has many solutions, that can be produced by the SMT module, and that can be partitioned according to the value assigned to the variable *Time*. Let S_t be the set of solutions in which the variable *Time* is assigned the value t. Clearly, the largest such set is for $Time = t_{m+1}$. In this case, in fact, the attacker has no information about the remaining password bits, except that at least one among $b_{m+1}, b_{m+2}, \dots b_{m+h}$ must be wrong, where h is the size of the bucket that contains the $m+1$ bit. Equivalently, one of the $X_{m+1}, X_{m+2}, \dots X_{m+h}$ must be 0. The number of such configurations is $2^h - 1$, and can be determined automatically by counting the number



(a) The decision tree for the password example.

$$X_1 = 0 \implies Time = t_1 \wedge Result = \texttt{fail}$$

$$X_1 = 1 \wedge X_2 = 0 \implies Time = t_2 \wedge Result = \texttt{fail}$$

$$\vdots$$

$$X_1 = 1 \wedge X_2 = 1 \wedge \ldots \wedge X_n = 0 \implies Time = t_n \wedge Result = \texttt{fail}$$

$$X_1 = 1 \wedge X_2 = 1 \wedge \ldots \wedge X_n = 1 \implies Time = t_n \wedge Result = \texttt{succ}$$

(b) The set of constraints derived from the decision tree for the password example.

Figure 4

$$(X_1 = 1 \wedge X_2 = 1 \wedge \ldots \wedge X_m = 1) \wedge (Time = t_1 \vee Time = t_2 \vee \ldots \vee Time = t_n) \wedge (Result = \texttt{fail} \vee Result = \texttt{succ})$$

Figure 5 The attack in the password example

of solutions for these variables[6]. Assuming that the substring of length h have the same probability $1/2^h$ to be part of the correct password, and that the attacker chooses uniformly the next one to try, the expected number of attempts that the attacker needs to perform to get to the next bucket is:

$$\sum_{i=1}^{2^h-1} \frac{1}{2^h} i \quad = \quad \frac{1}{2^h} \frac{(2^h-1)\,2^h}{2} \quad = \quad \frac{(2^h-1)}{2}.$$

By repeating this process, we can determine the average-time complexity of the attack (the expected total number of attempts before discovering the entire password), which is given by $(2^{h_1}+2^{h_2}+\ldots+2^{h_s}-s)/2$, where $h_1, h_2, \ldots, h_s$ are the size of the buckets. We remark that s and $h_1, h_2, \ldots, h_s$ are not known, and that this result is computed automatically from the constraints derived from the machine learning model. Furthermore, for more complicated timing attacks, such as attacks to encryption key, the relation between the stopping time, the size of the buckets and the bits of the key is not necessarily known, or may be very complicated to compute. The method illustrated above gives a quick and easy way to determine the complexity of the attack also in these cases.

[6] This can be done for example in CLP(FD) [23] by using the predicate **fd_size**, which returns the number of elements in the current domain of a variable.

$$Pulse > 160 - 0.5\,Age \quad \Rightarrow \quad Hearth\_alarm = \texttt{on}$$

$$\left(\begin{array}{c}((Sex = \texttt{female} \wedge Weight > .85(Height - 100)) \\ \vee \\ (Sex = \texttt{male} \wedge Weight > .90(Height - 100))) \\ \wedge \\ Blood\_pressure > 130\end{array}\right) \quad \Rightarrow \quad Diabetes\_alarm = \texttt{on}$$

Figure 6 Representation of a SHS in form of FOL constraints

4.2 Malware

Machine learning models are often used in critical applications which involve decisions of significant personal, societal, or economical impact. Examples include network intrusion detection, spam and phishing detection, healthcare systems, production planning, etc. Malware attacks to such decisional models typically consist in altering the values of some of the features so to induce a misclassification, and therefore a wrong decision. In this section we consider an example in the healthcare domain, and we show how our methodology can help to detect a potential attack, or prove its impossibility.

In the last decade there has been a major evolution towards automatization in healthcare, thanks to advances of research in the Internet of Things that has allowed to connect body sensors and other implantable medicals devices to networks of computing and big data resources. Smart healthcare systems (SHS's) continuously collect data from the medical sensors connected to the human body and process them for making decisions accordingly. This trend will increase in the future due also to the development of personalised medicine. Typically, machine learning is used to classify possible health problems from the symptoms and prescribe the necessary treatment. Unfortunately, these devices are exposed to potential attacks, especially due to the possible presence of malware that can compromise the readings of the sensors, like in the case of MEDJACK [42].

We assume that the SHS uses a classifier that takes in input, as features, the physical characteristics of the patient (such as weight, height, age, etc.), and the readings from the body sensors, to decide whether there is some potential health threat, and consequently raise the corresponding alarm. In this example we assume that the classifier is implemented as a Support Vector Machine. Figure 6 shows some typical constraints that could be derived from the classifier, to signal critical situations that could bring to a diabetic attack or to a hearth attack. The first formula represents the monitoring of the pulse in relation to age, during physical exercise. In the second formula the relations between height and weight are based on the Broca equations for women and men, respectively, and indicate the condition of being overweight.

In this scenario, a particular patient can be represented by the values of his or her attributes. For instance:

$$\mathrm{P}_1 \qquad Sex = \texttt{female} \wedge Weight = 80 \wedge Height = 165 \wedge Age = 40 \tag{11}$$

$$\mathrm{P}_2 \qquad Sex = \texttt{male} \wedge Weight = 75 \wedge Height = 180 \wedge Age = 40 \tag{12}$$

An attacker can be represented by its goals and its capability in tampering with the readings of the sensors. For instance, an attacker that aims at thwarting the diabetes alarm and is

able to alter the reading of the sensor *Blood_pressure* up to ± 20, can be represented by the constraint:

$$R - 20 \leq \mathit{Blood\_pressure} \leq R + 20 \wedge \mathit{Diabetes\_alarm} = \texttt{off} \tag{13}$$

where R is the true reading of the sensor.

Consider the set of constraints C consisting of set of formulae in Figure 6 together with the one representing the patient ((11) or (12)), and the one representing the attacker (13). By applying the SMT to C we can verify the existence of a potential attack. For instance, consider the patient P_1. For any $r \leq 150$ C is satisfiable by the following assignment (where we use $\doteq$ to represent the association between a value and a variable):

$$\begin{array}{l} \mathit{Sex} \doteq \texttt{female}\ ,\ \mathit{Weight} \doteq 80\ ,\ \mathit{Height} \doteq 165\ ,\ \mathit{Age} \doteq 40\ , \\ R \doteq r\ ,\ \mathit{Blood\_pressure} \doteq r - 20\ ,\ \mathit{Diabetes\_alarm} \doteq \texttt{off} \end{array} \tag{14}$$

which for $130 < r \leq 150$ represents a security breach because in such situation the alarm should be switched on.

Apart from reinforcing the security of the sensor, a possible countermeasure against this attack would be to specialize the SHS for patient P_1 so that the attacker could not succeed to prevent the activation of the alarm whenever $r > 130$. In order to ensure the non-satisfiability of (13) we derive that the threshold for *Blood_pressure* must be at most 110. Namely, we need to replace the constraint on *Blood_pressure* in Figure 6 with $\mathit{Blood\_pressure} > 110$. As for patient P_2, the attack on the diabetes alarm is not possible, thanks to the fact that, due to his low weight, he is not a patient-at-risk. In both cases, the lack of an attack possibility can be automatically verified by proving, via the STM, that the set of constraints has no solution.

5 Applications to Privacy

5.1 Model inversion

Model-inversion (MI) attacks aim at deriving sensitive features of a target individual by taking advantage of their correlation with the output revealed by the machine learning model. The first work that pointed out the existence of such privacy threat was [18]. In that paper, the authors considered a linear regression model for personalized medicine (recommendation of the dosage of a drug called *warfarin*), and they showed that an attacker that has access to some of the non-sensitive attributes of the victim (age, race, height, and weight) and to the outcome of the model (the dosage), may infer the victim's private genomic attributes, especially if he or she has participated in the dataset used for training. (The study focused on two genes, VKORC1 and CYP2C9, that are associated with the mechanism with which the body metabolizes the drug.) Successive works (for instance, [17, 1]) have extended MI attacks to other settings, e.g., recovering an image of a person from a face recognition model given just their name, and other target models, e.g., logistic regression and neural networks.

Usually MI attacks are formalized in probabilistic terms, namely the attacker is supposed to know the distributions of the data, and its strategy consists in determining which possible value for the sensitive feature achieves the maximum likelihood, given its knowledge of the non-sensitive values and the result of the model. However, for the sake of simplicity, here we consider a non-probabilistic model of attacker.

Suppose that the model represents the classification $h : \mathcal{X} \to \mathcal{Y}$, and that each $x \in \mathcal{X}$ is a tuple of n features (attributes) $\langle x_1, x_2, \ldots, x_n \rangle$. Suppose that x_n is the sensitive attribute,

and that the attacker is interested in discovering whether the value of x_n belongs to a target domain A. Suppose that the attacker knows the value $a_1, a_2, \ldots, a_{n-1}$ of the first $n-1$ attributes, and the classification outcome b. Then, the attack is effective if all the values a_n, such that $h(a_1, a_2, \ldots, a_{n-1}, a_n) = b$, belong to A. Otherwise, the attacker does not have enough evidence.

One way to model such attack by constraints is by representing its opposite, namely the possible existence of an alternative value for x_n which still satisfies the relation. This gives rise to the formula:

$$X_1 = a_1 \wedge X_2 = a_2 \wedge \ldots \wedge X_{n-1} = a_{n-1} \wedge Y = b \wedge \bigwedge_{a \in A} X_n \neq a \tag{15}$$

The outcome of the SMT solver in this case is to be interpreted as follows: no solutions implies that there is an attack, and viceversa one or more solution implies that the attacker has not enough evidence.

6 Conclusion

In this work we have discussed how to combine constraint solving and ML in a novel way. We have shown that several algorithms for ML, such as decision trees, support vector machines, k-nearest neighbours, can be transformed in a corresponding set of formulas in FOL to be solved by means of constraint solvers and SMT solvers. There are several advantages of this approach. One is to exploit the existing (and future) efficient solvers. Another one is to be able to combine the formulae in FOL with additional formulae expressing some critical facts on which we want to test the system. We have presented some examples of attempts of security and privacy breaches, such as an attempt to discover a password using information flow, an example of malware attack for smart health systems, and one model-inversion attack to a neural network for personalised medicine. We have then represented the attacker by an appropriate set of constraints, which we have combined together with the constraints of the original system. Finally, a check of satisfiability by a (SMT) solver of the full set of constraints allows us to derive whether the attack is possible, and how to prevent it.

As future work we intend to make some complexity analysis for the formulas that we derive with our methodology and perform experiments to evaluate the effectiveness of our approach. For this we need to encode the constraints generated with an appropriate constraint solver as mentioned in Section 2.2.

References

1 Ulrich Aïvodji, Sébastien Gambs, and Timon Ther. GAMIN: an adversarial approach to black-box model inversion. *CoRR*, abs/1909.11835, 2019. URL: `http://arxiv.org/abs/1909.11835`, `arXiv:1909.11835`.

2 N. S. Altman. An introduction to kernel and nearest-neighbor nonparametric regression. *The American Statistician*, 46(3):175–185, 1992. URL: `http://www.jstor.org/stable/2685209`.

3 Mário S. Alvim, Konstantinos Chatzikokolakis, Annabelle McIver, Carroll Morgan, Catuscia Palamidessi, and Geoffrey Smith. *The Science of Quantitative Information Flow*. Springer International Publishing, 2020. `doi:10.1007/978-3-319-96131-6`.

4 Roberto Amadini, Maurizio Gabbrielli, and Jacopo Mauro. Portfolio approaches for constraint optimization problems. *Ann. Math. Artif. Intell.*, 76(1-2):229–246, 2016. `doi:10.1007/s10472-015-9459-5`.

5 Clark W. Barrett, Christopher L. Conway, Morgan Deters, Liana Hadarean, Dejan Jovanovic, Tim King, Andrew Reynolds, and Cesare Tinelli. CVC4. In Ganesh Gopalakrishnan and

Shaz Qadeer, editors, *Computer Aided Verification - 23rd International Conference, CAV 2011, Snowbird, UT, USA, July 14-20, 2011. Proceedings*, volume 6806 of *Lecture Notes in Computer Science*, pages 171–177. Springer, 2011. doi:10.1007/978-3-642-22110-1_14.

6 Nicolas Beldiceanu and Helmut Simonis. A model seeker: Extracting global constraint models from positive examples. In Michela Milano, editor, *Principles and Practice of Constraint Programming - 18th International Conference, CP 2012, Québec City, QC, Canada, October 8-12, 2012. Proceedings*, volume 7514 of *Lecture Notes in Computer Science*, pages 141–157. Springer, 2012. doi:10.1007/978-3-642-33558-7_13.

7 Christopher M. Bishop. *Pattern recognition and machine learning, 5th Edition*. Information science and statistics. Springer, 2007. URL: https://www.worldcat.org/oclc/71008143.

8 Bernhard E. Boser, Isabelle Guyon, and Vladimir Vapnik. A training algorithm for optimal margin classifiers. In David Haussler, editor, *Proceedings of the Fifth Annual ACM Conference on Computational Learning Theory, COLT 1992, Pittsburgh, PA, USA, July 27-29, 1992*, pages 144–152. ACM, 1992. doi:10.1145/130385.130401.

9 Will Bridewell, Pat Langley, Ljupco Todorovski, and Saso Dzeroski. Inductive process modeling. *Mach. Learn.*, 71(1):1–32, 2008. doi:10.1007/s10994-007-5042-6.

10 Giovanni Cherubin, Konstantinos Chatzikokolakis, and Catuscia Palamidessi. F-BLEAU: fast black-box leakage estimation. In *2019 IEEE Symposium on Security and Privacy, SP 2019, San Francisco, CA, USA, May 19-23, 2019*, pages 835–852. IEEE, 2019. doi:10.1109/SP.2019.00073.

11 Geoffrey Chu, Peter J. Stuckey, Andreas Schutt, Thorsten Ehlers, Graeme Gange, and Kathryn Francis. Chuffed, a lazy clause generation solver. URL: https://github.com/chuffed/chuffed.

12 Gabriele Ciravegna, Francesco Giannini, Marco Gori, Marco Maggini, and Stefano Melacci. Human-driven FOL explanations of deep learning. In Christian Bessiere, editor, *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI 2020*, pages 2234–2240. ijcai.org, 2020. doi:10.24963/ijcai.2020/309.

13 Gabriele Ciravegna, Francesco Giannini, Stefano Melacci, Marco Maggini, and Marco Gori. A constraint-based approach to learning and explanation. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020*, pages 3658–3665. AAAI Press, 2020. URL: https://aaai.org/ojs/index.php/AAAI/article/view/5774.

14 Leonardo de Moura and Nikolaj Bjørner. Z3: An efficient SMT solver. In *Tools and Algorithms for the Construction and Analysis (TACAS)*, volume 4963 of *Lecture Notes in Computer Science*, pages 337–340, Berlin, 2008. Springer-Verlag. doi:10.1007/978-3-540-78800-3_24.

15 Luc Devroye, László Györfi, and Gábor Lugosi. *A Probabilistic Theory of Pattern Recognition*, volume 31 of *Stochastic Modelling and Applied Probability*. Springer, 1996. doi:10.1007/978-1-4612-0711-5.

16 Bruno Dutertre. Yices 2.2. In Armin Biere and Roderick Bloem, editors, *Computer Aided Verification - 26th International Conference, CAV 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 18-22, 2014. Proceedings*, volume 8559 of *Lecture Notes in Computer Science*, pages 737–744. Springer, 2014. doi:10.1007/978-3-319-08867-9_49.

17 Matt Fredrikson, Somesh Jha, and Thomas Ristenpart. Model inversion attacks that exploit confidence information and basic countermeasures. In Indrajit Ray, Ninghui Li, and Christopher Kruegel, editors, *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, October 12-16, 2015*, pages 1322–1333. ACM, 2015. doi:10.1145/2810103.2813677.

18 Matthew Fredrikson, Eric Lantz, Somesh Jha, Simon M. Lin, David Page, and Thomas Ristenpart. Privacy in pharmacogenetics: An end-to-end case study of personalized warfarin dosing. In Kevin Fu and Jaeyeon Jung, editors, *Proceedings of the 23rd USENIX Security Symposium, San Diego, CA, USA, August 20-22, 2014*, pages 17–32. USENIX Association, 2014. URL: https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/fredrikson_matthew.

19 Giorgio Gnecco, Marco Gori, Stefano Melacci, and Marcello Sanguineti. Foundations of support constraint machines. *Neural Comput.*, 27(2):388–480, 2015. doi:10.1162/NECO_a_00686.

20 J. A. Goguen and J. Meseguer. Security policies and security models. In *Proceedings of the 1982 Symposium on Security and Privacy (SSP '82)*, pages 11–20, Los Alamitos, Ca., USA, April 1990. IEEE Computer Society Press. doi:10.1109/SP.1982.10014.

21 Ian J. Goodfellow, Yoshua Bengio, and Aaron C. Courville. *Deep Learning*. Adaptive computation and machine learning. MIT Press, 2016. URL: http://www.deeplearningbook.org/.

22 Trevor Hastie, Robert Tibshirani, and Jerome H. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction, 2nd Edition*. Springer Series in Statistics. Springer, 2009. doi:10.1007/978-0-387-84858-7.

23 Joxan Jaffar and Michael J. Maher. Constraint logic programming: A survey. *J. Log. Program.*, 19/20:503–581, 1994. doi:10.1016/0743-1066(94)90033-7.

24 Samuel Kolb, Stefano Teso, Andrea Passerini, and Luc De Raedt. Learning SMT(LRA) constraints using SMT solvers. In Jérôme Lang, editor, *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden*, pages 2333–2340. ijcai.org, 2018. doi:10.24963/ijcai.2018/323.

25 Samuel M. Kolb. Learning constraints and optimization criteria. In Parisa Kordjamshidi, editor, *Declarative Learning Based Programming, Papers from the 2016 AAAI Workshop, Phoenix, Arizona, USA, February 13, 2016*, volume WS-16-07 of *AAAI Workshops*. AAAI Press, 2016. URL: http://www.aaai.org/ocs/index.php/WS/AAAIW16/paper/view/12651.

26 Boris Köpf and Markus Dürmuth. A provably secure and efficient countermeasure against timing attacks. In *Proceedings of the 22nd IEEE Computer Security Foundations Symposium, CSF 2009, Port Jefferson, New York, USA, July 8-10, 2009*, pages 324–335. IEEE Computer Society, 2009. doi:10.1109/CSF.2009.21.

27 Michele Lombardi, Michela Milano, and Andrea Bartolini. Empirical decision model learning. *Artif. Intell*, 244:343–367, 2017. doi:10.1016/j.artint.2016.01.005.

28 Stephen Muggleton. Inductive logic programming. *New Gener. Comput.*, 8(4):295–318, 1991. doi:10.1007/BF03037089.

29 Robert Nieuwenhuis, Albert Oliveras, and Cesare Tinelli. Solving SAT and SAT modulo theories: From an abstract davis–putnam–logemann–loveland procedure to dpll(T). *J. ACM*, 53(6):937–977, 2006. doi:10.1145/1217856.1217859.

30 Chunki Park, Will Bridewell, and Pat Langley. Integrated systems for inducing spatio-temporal process models. In Maria Fox and David Poole, editors, *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2010, Atlanta, Georgia, USA, July 11-15, 2010*. AAAI Press, 2010. URL: http://www.aaai.org/ocs/index.php/AAAI/AAAI10/paper/view/1742.

31 Tomasz P. Pawlak and Krzysztof Krawiec. Automatic synthesis of constraints from examples using mixed integer linear programming. *Eur. J. Oper. Res.*, 261(3):1141–1157, 2017. doi:10.1016/j.ejor.2017.02.034.

32 Laurent Perron and Vincent Furnon. Or-tools. URL: https://developers.google.com/optimization/.

33 J. Ross Quinlan. Induction of decision trees. *Mach. Learn.*, 1(1):81–106, 1986. doi:10.1023/A:1022643204877.

34 Luc De Raedt, Anton Dries, Tias Guns, and Christian Bessiere. Learning constraint satisfaction problems: An ILP perspective. In Christian Bessiere, Luc De Raedt, Lars Kotthoff, Siegfried Nijssen, Barry O'Sullivan, and Dino Pedreschi, editors, *Data Mining and Constraint Programming - Foundations of a Cross-Disciplinary Approach*, volume 10101 of *Lecture Notes in Computer Science*, pages 96–112. Springer, 2016. doi:10.1007/978-3-319-50137-6_5.

35 Luc De Raedt, Andrea Passerini, and Stefano Teso. Learning constraints from examples. In Sheila A. McIlraith and Kilian Q. Weinberger, editors, *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial*

Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018, pages 7965–7970. AAAI Press, 2018. URL: https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/17229.

36 Marco Romanelli, Konstantinos Chatzikokolakis, and Catuscia Palamidessi. Optimal obfuscation mechanisms via machine learning. In *33rd IEEE Computer Security Foundations Symposium, CSF 2020, Boston, MA, USA, June 22-26, 2020*, pages 153–168. IEEE, 2020. doi:10.1109/CSF49147.2020.00019.

37 Marco Romanelli, Konstantinos Chatzikokolakis, Catuscia Palamidessi, and Pablo Piantanida. Estimating g-leakage via machine learning. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security (CCS '20)*. ACM, 2020. To appear. Preprint in CoRR abs/2005.04399. URL: https://arxiv.org/abs/2005.04399, arXiv:2005.04399.

38 Francesca Rossi, Peter van Beek, and Toby Walsh, editors. *Handbook of Constraint Programming*, volume 2 of *Foundations of Artificial Intelligence*. Elsevier, 2006. URL: http://www.sciencedirect.com/science/bookseries/15746526/2.

39 Vijay Saraswat. *Concurrent Constraint Programming*. MIT Press, 1993. doi:https://doi.org/10.7551/mitpress/2086.001.0001.

40 Vijay A. Saraswat, Martin Rinard, and Prakash Panangaden. The semantic foundations of concurrent constraint programming. In *Conference Record of the Eighteenth Annual ACM Symposium on Principles of Programming Languages*, pages 333–352. ACM Press, 1991. doi:10.1145/99583.99627.

41 Shai Shalev-Shwartz and Shai Ben-David. *Understanding Machine Learning - From Theory to Algorithms*. Cambridge University Press, 2014. URL: http://www.cambridge.org/de/academic/subjects/computer-science/pattern-recognition-and-machine-learning/understanding-machine-learning-theory-algorithms.

42 Darlene Storm. MEDJACK: Hackers hijacking medical devices to create backdoors in hospital networks. Computerworld, June 2015. URL: https://www.computerworld.com/article/2932371/medjack-hackers-hijacking-medical-devices-to-create-backdoors-in-hospital-networks.html.

43 Geoffrey G. Towell and Jude W. Shavlik. Extracting refined rules from knowledge-based neural networks. *Mach. Learn.*, 13:71–101, 1993. doi:10.1007/BF00993103.

44 Leslie G. Valiant. A theory of the learnable. In Richard A. DeMillo, editor, *Proceedings of the 16th Annual ACM Symposium on Theory of Computing, April 30 - May 2, 1984, Washington, DC, USA*, pages 436–445. ACM, 1984. doi:10.1145/800057.808710.