ABDUCTION BASED EXPLANATIONS FOR ML MODELS

CHRISTIAN D'ERRICO

Faculty of Engineering and Computer Science, University of Bologna, Italy
e-mail address: christian.derrico@studio.unibo

Abstract. Abductive Reasoning is a form of logical inference: it starts with an observation or set of observations and then seeks the simplest and most likely conclusion.
Abduction has already been studied in the field of XAI, the understandable Artificial Intelligence, as an approach to explain the machine learning prediction of samples from a data set by generating subset-minimal or cardinality-minimal explanations with respect to features.
This work is born as a reproduction of the study *Abduction-Based Explanations for Machine Learning Models*, curated by Alexey Ignatiev, Nina Narodytska and Joao Marques-Silva, and it's focused on building such an abductive model agnostic strategy based on two algorithms in order to obtain explanations for neural networks (NNs).
The experimental results, on well-known datasets, validate the scalability of the proposed approach as well as the quality of the computed solutions.

INTRODUCTION

Model interpretability is a long-standing problem in machine learning that has become quite acute with the accelerating pace of widespread adoption of complex predictive algorithms. The growing range of applications of Machine Learning (ML) in a multitude of settings motivates the need of computing small explanations for predictions made.
Explainable AI (XAI), or Interpretable AI, or Explainable Machine Learning (XML), is artificial intelligence (AI) in which humans can understand the decisions or predictions made by the AI. It contrasts with the "black box" concept where even its designers cannot explain why an AI arrived at a specific decision.

Preprint submitted to
Intelligent Systems Engineering

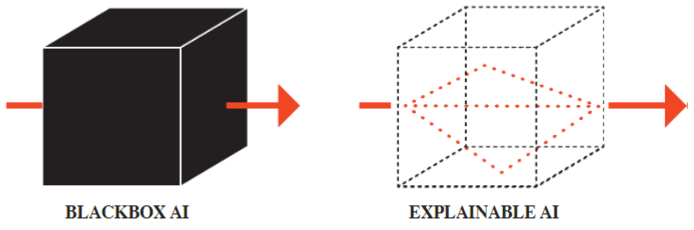


FIGURE 1. From Blackbox AI to Explainable AI

By refining the mental models of users of AI-powered systems and dismantling their misconceptions, XAI promises to help them perform more effectively.
One of the reasons for such a high interest in understanding the processes behind these algorithms is also the increase in public sensitivity to privacy through the introduction of various international laws such as the EU GDPR which impose high levels of private protection and at the same time transparency in the processing of information, something that AI's algorithms are currently struggling to achieve.
All this translates into a growing need for transparency in AI algorithms: it is crucial for an efficient distribution of intelligent systems not only to ensure that the models work as they were designed but also to instill confidence in the users, so that they can make decisions without fear of making wrong choices.
The need for transparency in the context of AI has led to the growth of the so-called eXplain-able AI (XAI), which represents a set of techniques that allow understanding and presenting a transparent vision of machine learning models and AI algorithms in general.

Abductive Reasoning. Abduction is defined as "a syllogism in which the major premise is evident but the minor premise and therefore the conclusion only probable." Basically, it involves forming a conclusion from the information that is known. A familiar example of abduction is a detective's identification of a criminal by piecing together evidences at a crime scene. In an everyday scenario, someone may be puzzled by a half-eaten sandwich on the kitchen counter. Abduction will lead to the best explanation. The reasoning might be that the teenage son made the sandwich and then saw that he was late for work. In a rush, he put the sandwich on the counter and left.
So, Abductive Reasoning is a form of logical inference: it starts with an observation or set of observations and then seeks the simplest and most likely conclusion from the observations. This process, unlike deductive reasoning, yields a plausible conclusion but does not positively verify it. Abductive conclusions are thus qualified as having a remnant of uncertainty or doubt, which is expressed in retreat terms such as "best available" or "most likely".

Abductive Reasoning and XAI. As said above, the importance of computing explanations is further underscored by a number of recent works.
For logic-based models, e.g. decision trees and sets, explanations can be obtained directly from the model, and related research work has mostly focused on minimizing the size of representations. Nevertheless, important ML models, that include neural networks (NNs),

support vector machines (SVMs), bayesian network classifiers (BNCs), among others, do not naturally provide explanations to predictions made.
Most work on computing explanations for such models is based on heuristic approaches, with no guarantees of quality, and these can perform poorly in practical settings.
In this context the abductive approach takes over: abduction has already been well studied in the field of computational logic, and logic programming in particular, for a few decades by now.
Abduction in logic programs, for example, offers a formalism to declaratively express problems in a variety of areas and it has many applications, e.g. in decision-making, diagnosis, planning, belief revision, and hypothetical reasoning.
So, in the XAI scenario, the idea is building a model agnostic method for computing explanations of ML models with formal guarantees (e.g. cardinality-minimal or subset-minimal explanations), a strategy that can be applied to any model, after the training, treating it as a black box.
This type of method does not require a look into the inner workings of the model: if it can be expressed in a suitable formalism then it can be explained.
In the context of the put-in-act experiments, this method is applied to neural network (NNs) models, especially multilayer perceptron (MLP), a fully connected class of feedforward neural network (NN).

1. THEORICAL-BACKGROUND

In logic, an abduction starts with an observation or a set of observations and then find the best explanation to them.

Definition 1.1. An abduction problem is a 3-tuple $P = (M, C, \mathcal{E})$ where M is a theory (a set of logical formulas), C is a set of hypothesis (also a set of logical formulas), and $\mathcal{E}$ is (a set of) observations.

Definition 1.2. An explanation (a solution) to an abduction problem $(M, C, \mathcal{E})$ is a subset $C_0 \subseteq C$ such that $M, C_0 \models \mathcal{E}$, where $\models$ is a classical entailment.

Definition 1.3. An explanation C_0 is called subset-minimal if no proper subset $C_0' \subset C_0$ satisfies $M, C_0' \models \mathcal{E}$. More strictly, an explanation C_0 is called a cardinality-minimal explanation if for another explanation C_0' of the 3-tuple $(M, C, \mathcal{E})$, one has $|C_0| \leq |C_0'|$; that is, no explanation with strictly smaller size than C_0 in terms of cardinality.

Let's now suppose that a trained classifier predicts an input data $v \in R_n$ to be of class j, where $j \in \{1, 2, \ldots, m\}$ possible classes, that is $y_j \geq y_k$ for any $k \in$ {1,2,. . . , m}. Let us denote M be the encoding of our trained model as explained.
The encoding can be seen as a set of constraints that rule what kind of prediction the encoded model M can make based on the input.
Let $\mathcal{E}$ be the observation that $\bigwedge_{k=1}^{k} (y_j \geq y_k)$, that is the classifier gives prediction class j to a data v. Let C be the set of equational formulas $\{x_i = v_i, i = 1, ..., n\}$.
An explanation to the abduction problem $(M, C, \mathcal{E})$ is then a set of features of v which are sufficient for M to predict v to be of class j; no matter what values the remaining features take.

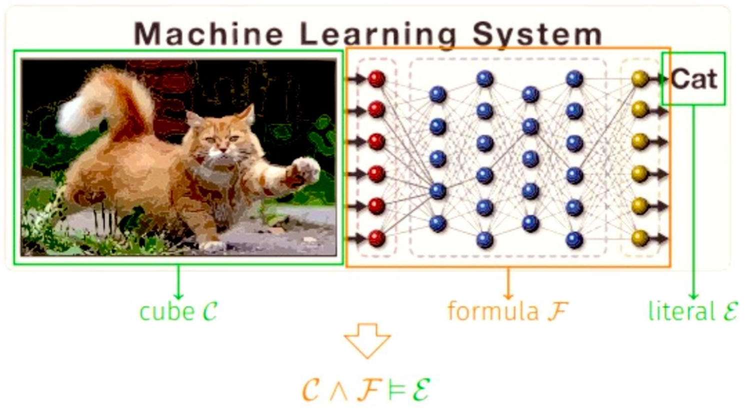


FIGURE 2. Here every item described above

2. ENCODING

For the purpose of this study, an (artificial) neural network with ReLU as activation function is represented, in the language of some constraint reasoning system (in this case, Satisfiability Modulo Theories and Mixed Integer Linear Programming), using a set of constraints precisely.
Let the NN be made by $K+1$ layers, numbered from 0 to K. Layer 0 is fictitious and corresponds to the input of the NN, while the last layer, K, corresponds to its outputs. Each layer $k \in \{0, 1, ..., K\}$ is made by n_k units (i.e., nodes in networks, or neurons), numbered from 1 to n_k, and the j-th unit of layer k are denoted by UNIT(j,k).
A layer consists of a linear transformation and a nonlinear transformation.
For each layer $k \geq 1$, UNIT(j,k) computes its output vector x_k through the formula: $x^k = \sigma(W^{k-1}x^{k-1} + b^{k-1})$, where $\sigma(.)$ is non-linear function and W^{k-1} (resp. b^{k-1}) is a given matrix of weights (resp., vector of biases).
As in many applications, it's assumed that σ is a rectified linear unit, i.e., the equations governing the NN are $y = ReLU(W^{k-1}x^{k-1} + b^{k-1}), k = 1, ..., K$ where, for a real vector y, $ReLU(y) := max\{0, y\}$ (componentwise).
The weight/bias matrices (W, b) can contain negative entries, while all the output vectors x_k are nonnegative, with the possible exception of the vector x_0 that represents the input of the DNN as a whole.
The presence of categorical and binary features is taken into consideration and they are managed as it should be: bounds are fixed and only integer values are allowed.
To get a valid 0-1 MILP model for a given NN, one needs to study the basic scalar equation $y = ReLU(w^T x + b)$.
To this end, one can write the linear conditions $w^T x + b = y - s, x \geq 0, s \geq 0$.
To impose that at least one of the two terms x and s must be zero, the choice is to introduce a binary activation variable z and to impose the logical implications:

$$\left.\begin{aligned} z = 1 \rightarrow x \leq 0 \\ z = 0 \rightarrow s \leq 0 \\ z \in \{0,1\} \end{aligned}\right\}$$

To see why the encoding is correct two cases are considered. First, $z_i = 1$: as mentioned above, y_i must be zero in this case. Indeed, if $z = 1$ then $y_i \leq 0$ holds. Together with $y_i \geq 0$, it's consequential that $y_i = 0$. Similarly, $z_i = 0$ should ensure that $y_i = \sum_{j=1}^{n} a_{i,j}x_j + b_i$: if $z = 0$ then $s_i \leq 0$ forcing $s_i = 0$.

Using a binary activation variable z_{jk} for each UNIT(j,k) then leads to the following 0-1 MILP formulation of the NN:

$$\left.\begin{array}{r} \sum_{i=1}^{n_{k-1}} w_{ij}^{k-1}x_i^{k-1}+b_j^{k-1}=x_j^k-s_j^k \\ x_j^k,s_j^k\geq 0 \\ z_j^k\in\{0,1\} \\ z_j^k=1\rightarrow x_j^k\leq 0 \\ z_j^k=0\rightarrow s_j^k\leq 0\end{array}\right\} k=1,\ldots,K,\ j=1,\ldots,n_k$$

FIGURE 3. Complete formulation

Here a CPLEX example of encoding:

```
Minimize
 obj1: 0 Z_0_layer_1 + 0 Z_1_layer_1 + 0 Z_2_layer_1 + 0 Z_3_layer_1
       + 0 Z_4_layer_1 + 0 Z_5_layer_1 + 0 Z_6_layer_1 + 0 Z_7_layer_1
       + 0 Z_8_layer_1 + 0 Z_9_layer_1 + 0 Z_10_layer_1 + 0 Z_11_layer_1
       + 0 Z_12_layer_1 + 0 Z_13_layer_1 + 0 Z_14_layer_1
Subject To
 constraint_0#0:   - 0.706829011440277 X_1_layer_0_type_B_min_1_max_2
                   + 0.450135856866837 X_2_layer_0_type_B_min_0_max_2
                   + 0.285952478647232 X_3_layer_0_type_B_min_1_max_2
                   + 0.294078528881073 X_4_layer_0_type_B_min_0_max_2
                   + 0.0534391514956951 X_5_layer_0_type_B_min_0_max_2
                   + 0.15550534427166 X_6_layer_0_type_B_min_0_max_2
                   + 0.341544687747955 X_7_layer_0_type_B_min_0_max_2
                   - 0.162461936473846 X_8_layer_0_type_B_min_0_max_2
                   + 0.0357605963945389 X_9_layer_0_type_B_min_0_max_2
                   + 0.239943385124207 X_10_layer_0_type_B_min_0_max_2
                   - 0.194777756929398 X_11_layer_0_type_B_min_0_max_2
                   + 0.43754169344902 X_12_layer_0_type_B_min_0_max_2
                   - 0.368912279605865 X_18_layer_0_type_B_min_1_max_2
                   - 0.515840411186218 X_0_layer_0_type_C
                   - 0.255599051713943 X_13_layer_0_type_C
                   + 0.164519041776657 X_14_layer_0_type_C
                   - 0.286041438579559 X_15_layer_0_type_C
                   + 0.0362951308488846 X_16_layer_0_type_C
                   - 0.42959001660347 X_17_layer_0_type_C - X_0_layer_1_type_C
                   + S_0_layer_1_type_C  = -0.397678345441818
[...]
constraint_0#15:  - 0.450046539306641 X_0_layer_1_type_C
                 - 0.066827118396759 X_1_layer_1_type_C
                 - 0.206780970096588 X_2_layer_1_type_C
                 - 0.255253821611404 X_3_layer_1_type_C
                 - 0.766376256942749 X_4_layer_1_type_C
                 - 0.0191777236759663 X_5_layer_1_type_C
                 + 0.47623997926712 X_6_layer_1_type_C
                 - 0.114404246211052 X_7_layer_1_type_C
                 + 0.161891058087349 X_8_layer_1_type_C
                 + 0.254643321037292 X_9_layer_1_type_C
                 - 0.461219012737274 X_10_layer_1_type_C
                 + 0.00602369243279099 X_11_layer_1_type_C
                 + 0.0902262553572655 X_12_layer_1_type_C
                 + 0.0656921416521072 X_13_layer_1_type_C
                 + 0.358971178531647 X_14_layer_1_type_C - X_0_layer_2_type_C
                  = 0.155237451195717

ic_0_layer_1#0:   Z_0_layer_1 = 1 -> X_0_layer_1_type_C =< 0
ic_0_layer_1#1:   Z_0_layer_1 = 0 -> S_0_layer_1_type_C =< 0
ic_1_layer_1#2:   Z_1_layer_1 = 1 -> X_1_layer_1_type_C =< 0
ic_1_layer_1#3:   Z_1_layer_1 = 0 -> S_1_layer_1_type_C =< 0
ic_2_layer_1#4:   Z_2_layer_1 = 1 -> X_2_layer_1_type_C =< 0
ic_2_layer_1#5:   Z_2_layer_1 = 0 -> S_2_layer_1_type_C =< 0

Bounds
0 <= X_1_layer_0_type_B_min_1_max_2 <= 2
0 <= X_2_layer_0_type_B_min_0_max_2 <= 2
0 <= X_3_layer_0_type_B_min_1_max_2 <= 2
[...]
0 <= Z_0_layer_1 <= 1
0 <= Z_1_layer_1 <= 1
0 <= Z_2_layer_1 <= 1
0 <= Z_3_layer_1 <= 1
0 <= Z_4_layer_1 <= 1
[...]
0 <= Z_12_layer_1 <= 1
0 <= Z_13_layer_1 <= 1
0 <= Z_14_layer_1 <= 1
    X_0_layer_2_type_C Free
    X_1_layer_2_type_C Free

Binaries
 Z_0_layer_1  Z_1_layer_1  Z_2_layer_1  Z_3_layer_1  Z_4_layer_1  Z_5_layer_1
 Z_6_layer_1  Z_7_layer_1  Z_8_layer_1  Z_9_layer_1  Z_10_layer_1
 Z_11_layer_1  Z_12_layer_1  Z_13_layer_1  Z_14_layer_1

Generals
 X_1_layer_0_type_B_min_1_max_2  X_2_layer_0_type_B_min_0_max_2
 X_3_layer_0_type_B_min_1_max_2  X_4_layer_0_type_B_min_0_max_2
 X_5_layer_0_type_B_min_0_max_2  X_6_layer_0_type_B_min_0_max_2
 X_7_layer_0_type_B_min_0_max_2  X_8_layer_0_type_B_min_0_max_2
 X_9_layer_0_type_B_min_0_max_2  X_10_layer_0_type_B_min_0_max_2
 X_11_layer_0_type_B_min_0_max_2  X_12_layer_0_type_B_min_0_max_2
 X_18_layer_0_type_B_min_1_max_2

End
```

This is the SMT counter-part of the encoding:

```
Assertion:  (((X_1_layer_0_type_B_min_0_max_1 = 0.0) | (X_1_layer_0_type_B_min_0_max_1 = 1.0))
& ((X_1_layer_0_type_B_min_0_max_1 = 0.0) -> (! (X_1_layer_0_type_B_min_0_max_1 = 1.0))))
Assertion:  (((X_5_layer_0_type_B_min_0_max_1 = 0.0) | (X_5_layer_0_type_B_min_0_max_1 = 1.0))
& ((X_5_layer_0_type_B_min_0_max_1 = 0.0) -> (! (X_5_layer_0_type_B_min_0_max_1 = 1.0))))
Assertion:  (((X_8_layer_0_type_B_min_0_max_1 = 0.0) | (X_8_layer_0_type_B_min_0_max_1 = 1.0))
& ((X_8_layer_0_type_B_min_0_max_1 = 0.0) -> (! (X_8_layer_0_type_B_min_0_max_1 = 1.0))))
Assertion:  (((X_2_layer_0_type_B_min_0_max_3 = 0.0) | (X_2_layer_0_type_B_min_0_max_3 = 1.0)
| (X_2_layer_0_type_B_min_0_max_3 = 2.0) | (X_2_layer_0_type_B_min_0_max_3 = 3.0))
& (((X_2_layer_0_type_B_min_0_max_3 = 0.0) -> (! ((X_2_layer_0_type_B_min_0_max_3 = 1.0)
                                              | (X_2_layer_0_type_B_min_0_max_3 = 2.0)
                                              | (X_2_layer_0_type_B_min_0_max_3 = 3.0))))
& ((X_2_layer_0_type_B_min_0_max_3 = 1.0) -> (! ((X_2_layer_0_type_B_min_0_max_3 = 2.0)
                                              | (X_2_layer_0_type_B_min_0_max_3 = 3.0))))
& ((X_2_layer_0_type_B_min_0_max_3 = 2.0) -> (! (X_2_layer_0_type_B_min_0_max_3 = 3.0)))))
[...]

Assertion:  (((9383879/33554432 X_0_layer_0_type_C) + (-3458953/8388608 X_1_layer_0_type_B_min_0_max_1)
+ (11239877/16777216 X_2_layer_0_type_B_min_0_max_3) + (3182059/8388608 X_3_layer_0_type_C)
+ (8643373/16777216 X_4_layer_0_type_C) + (10425105/16777216 X_5_layer_0_type_B_min_0_max_1)
+ (-7193445/536870912 X_6_layer_0_type_B_min_0_max_2) + (-13520057/268435456 X_7_layer_0_type_C)
+ (2968391/8388608 X_8_layer_0_type_B_min_0_max_1) + (15637885/134217728 X_9_layer_0_type_C)
+ (5784433/16777216 X_10_layer_0_type_B_min_0_max_2) + (-12140939/134217728 X_11_layer_0_type_B_min_0_max_3)
+ (-1433981/16777216 X_12_layer_0_type_B_min_0_max_2) + -5150427/67108864)
= (X_0_layer_1_type_C - S_0_layer_1_type_C))
Assertion:  ((Z_0_layer_1 = 1) -> (X_0_layer_1_type_C <= 0.0))
Assertion:  ((Z_0_layer_1 = 0) -> (S_0_layer_1_type_C <= 0.0))
Assertion:  (((Z_0_layer_1 = 0) | (Z_0_layer_1 = 1)) & ((Z_0_layer_1 = 0) -> (! (Z_0_layer_1 = 1))))
Assertion:  (0.0 <= X_0_layer_1_type_C)
Assertion:  (0.0 <= S_0_layer_1_type_C)

Assertion:  (((-11729451/67108864 X_0_layer_0_type_C) + (14454449/67108864 X_1_layer_0_type_B_min_0_max_1)
+ (2885389/4194304 X_2_layer_0_type_B_min_0_max_3) + (-9454285/67108864 X_3_layer_0_type_C)
+ (11575835/67108864 X_4_layer_0_type_C) + (9323441/33554432 X_5_layer_0_type_B_min_0_max_1)
+ (120553/262144 X_6_layer_0_type_B_min_0_max_2) + (-10992883/16777216 X_7_layer_0_type_C)
+ (5599009/16777216 X_8_layer_0_type_B_min_0_max_1) + (8558781/33554432 X_9_layer_0_type_C)
+ (8953077/67108864 X_10_layer_0_type_B_min_0_max_2) + (600123/2097152 X_11_layer_0_type_B_min_0_max_3)
+ (10695813/16777216 = (X_1_layer_1_type_C - S_1_layer_1_type_C))

Assertion:  ((Z_1_layer_1 = 1) -> (X_1_layer_1_type_C <= 0.0))
Assertion:  ((Z_1_layer_1 = 0) -> (S_1_layer_1_type_C <= 0.0))
Assertion:  (((Z_1_layer_1 = 0) | (Z_1_layer_1 = 1)) & ((Z_1_layer_1 = 0) -> (! (Z_1_layer_1 = 1))))
Assertion:  (0.0 <= X_1_layer_1_type_C)
Assertion:  (0.0 <= S_1_layer_1_type_C)

[...]

Assertion:  (((-7775301/16777216 X_0_layer_1_type_C) + (-1965937/8388608 X_1_layer_1_type_C)
+ (7333589/8388608 X_2_layer_1_type_C) + (15228763/16777216 X_3_layer_1_type_C)
+ (10787765/33554432 X_4_layer_1_type_C) + (295309/1048576 X_5_layer_1_type_C)
+ (1945961/8388608 X_6_layer_1_type_C) + (-10023989/67108864 X_7_layer_1_type_C)
+ (-5804371/8388608 X_8_layer_1_type_C) + (-13469317/33554432 X_9_layer_1_type_C)
+ (-11344805/33554432 X_10_layer_1_type_C) + (-4395235/8388608 X_11_layer_1_type_C)
+ (9015333/268435456 X_12_layer_1_type_C) + (6888163/16777216 X_13_layer_1_type_C)
+ (5029483/8388608 X_14_layer_1_type_C) + -16473301/268435456) = X_0_layer_2_type_C)

Assertion:  (((6930481/16777216 X_0_layer_1_type_C) + (10158901/16777216 X_1_layer_1_type_C)
+ (-15681653/33554432 X_2_layer_1_type_C) + (-16431095/67108864 X_3_layer_1_type_C)
+ (-11145717/16777216 X_4_layer_1_type_C) + (-8396069/8388608 X_5_layer_1_type_C)
+ (-3103117/4194304 X_6_layer_1_type_C) + (13629259/16777216 X_7_layer_1_type_C)
+ (4378113/8388608 X_8_layer_1_type_C) + (10630393/33554432 X_9_layer_1_type_C)
+ (8032053/16777216 X_10_layer_1_type_C) + (5191467/16777216 X_11_layer_1_type_C)
+ (2680895/4194304 X_12_layer_1_type_C) + (-4320555/16777216 X_13_layer_1_type_C)
+ (-10623759/16777216 X_14_layer_1_type_C) + 1029581/16777216) = X_1_layer_2_type_C)

Assertion:  (X_0_layer_2_type_C < X_1_layer_2_type_C)
```

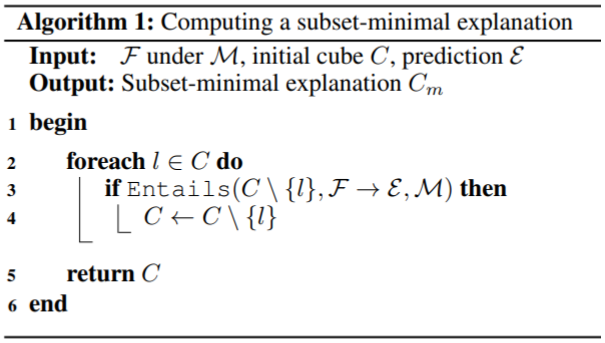
---

3. EXPLAINING PHASE

Provided that a model can be encoded as seen, and that an oracle (a solver) that can answer entailment queries is accessable (it's fundamental for the explanation procedure), it's possible to define the quality of an explanation with the number of specified features associated with a prediction.
As a result, one of the main goals is to compute cardinality-minimal explanations. Another, in practice more relevant due to performance challenges of the first goal, is to compute subset-minimal explanations.

3.1. **Algorithm 1: Computing a subset-minimal explanation.**

The main idea of Algorithm 1 consists of dropping the features from the explanation when the remaining ones are still sufficient to force the neural network make the same prediction.
The algorithm will always produce a subset-minimal explanation and is order-sensitive in a sense that the outcome depends on the order of features listed in C.
Given the cube C encoding a data sample for the prediction $\mathcal{E}$, the procedure returns its minimal subset C_m s.t. $F^1 \wedge C_m \models \mathcal{E}$, under M.
Iteratively, literals l of the input cube C are removed, with a check whether the remaining subcube implies formula $F \rightarrow \mathcal{E}$, i.e. $(C \setminus l) \models (F \rightarrow \mathcal{E})$.
To check the entailment is suffices to check if $F \wedge C_m \wedge \neg\mathcal{E}$ is false.
The algorithm traverses all literals of the cube and, thus, makes $|C|$ calls to the T-oracle.



```
Algorithm 1: Computing a subset-minimal explanation
  Input:  $\mathcal{F}$ under $\mathcal{M}$, initial cube $C$, prediction $\mathcal{E}$
  Output: Subset-minimal explanation $C_m$
begin
    foreach $l \in C$ do
        if Entails($C \setminus \{l\}, \mathcal{F} \rightarrow \mathcal{E}, \mathcal{M}$) then
            $C \leftarrow C \setminus \{l\}$
    return $C$
end
```

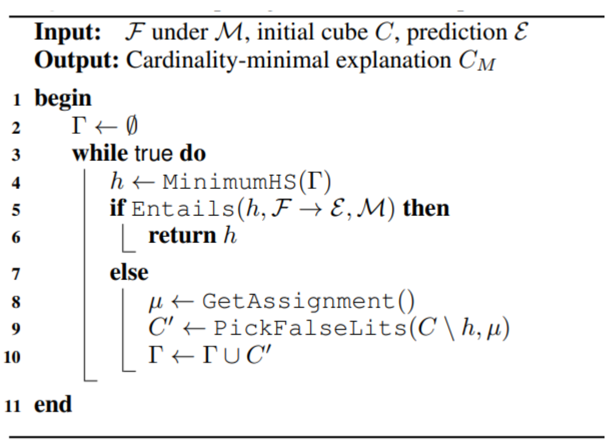
FIGURE 4. First procedure

3.2. **Algorithm 2: Computing a subset-minimal explanation.**

This algorithm is an iterative process, which deals with a set Γ of sets to hit.
Given a collection of sets Γ, a hitting-set h is a set satisfying $h \cap S \neq \emptyset$ for every $S \in \Gamma$: the intersection between h and S is non-empty.
A hitting-set h of S is minimal if none of its proper subset is a hitting set for S.
Initially, set Γ is empty. At each iteration, a new smallest size hitting set h for Γ is computed.

[1]F is logical representation of the ML model

Hitting set h is then treated as a cube, which is tested on whether it implies formula $F \to \mathcal{E}$ under model M.
As it has been discussed above, this can be tested by calling a T-oracle on formula $h \wedge F \wedge \neg\mathcal{E}$. If the T-oracle returns false, the algorithm reports hitting set h as smallest size explanation for the prediction and stops. Otherwise an assignment μ for the free variables of formula $h \wedge F \wedge \neg\mathcal{E}$ is extracted.
Assignment μ is then used to determine a subset C' of literals of cube $C \setminus h$ that were falsified by the previous call to the T-oracle. Finally, set Γ is updated to include C' and the process continues.



```
Input:  $\mathcal{F}$ under $\mathcal{M}$, initial cube $C$, prediction $\mathcal{E}$
Output: Cardinality-minimal explanation $C_M$
begin
    $\Gamma \leftarrow \emptyset$
    while true do
        $h \leftarrow$ MinimumHS($\Gamma$)
        if Entails($h, \mathcal{F} \to \mathcal{E}, \mathcal{M}$) then
            return $h$
        else
            $\mu \leftarrow$ GetAssignment()
            $C' \leftarrow$ PickFalseLits($C \setminus h, \mu$)
            $\Gamma \leftarrow \Gamma \cup C'$
end
```

FIGURE 5. Second procedure

4. EXPERIMENTAL RESULTS

This section evaluates the results of the conducted experiments and the quality of the computed explanations (in terms of the minimality guarantees). The benchmarks considered include the well-known text-based datasets from the UCI Machine Learning Repository and Penn Machine Learning Benchmarks, as well as the widely used MNIST digits database.

4.1. **Setup and prototype implementation.** The code ran on a desktop computer having an AMD Ryzen 7 2700 Eight-Core Processor 3.20 GHz with 8GByte of memory on board. The prototype implementation of the proposed approach follows Algorithm 1 and Algorithm 2 for computing subset- and cardinality-minimal explanations, respectively. Everything is written in Python (from model training and encoding to explanations procedures) and targets both SMT and MILP solvers.
SMT solvers are accessed through the PySMT framework[2], which provides a unified interface to SMT solvers like CVC4, MathSAT5, Yices2, and Z3.
Yices2 has been the most efficient SMT solver, in computational terms, and has been fundamental in order to obtain the major number of explanations in the established time limit. CPLEX 12.8.0 (IBM ILOG 2018) is used as a MILP oracle accessed via its official Python API[3].
The implementation of minimum hitting set enumeration in Algorithm 2 is based on an award-winning maximum satisfiability solver RC2 written on top of the PySAT toolkit.

4.2. **Quality of explanations.** For the experiments a selection of datasets from UCI Machine Learning Repository and Penn Machine Learning Benchmarks is used. The datasets have 9–32 features and contain 164–691 data samples.
The experiment is organized as follows:

(1) Each dataset gets a simple preprocessing, such as filling missing value and standard normalization. For the MNIST dataset, every value is reduced to be 1 (white pixel) or 0 (black pixel).
(2) For each dataset, a neural network model with the same architecture is provided.
Each neural network considered has one hidden layer with $i \in \{10, 15, 20\}$ neurons. The activation function is ReLu and SoftMax for the last layer (this type of function doesn't impact on the algorithms for explanations). The accuracy of the trained NN classifiers is at least 92% on all the considered datasets.
(3) The explanation procedure (computing either a subset- or a cardinality-minimal explanation) is ran for each sample: NN model is encoded as MILP and SMT problems, as described above.
A dataset is considered explained if it's possible to explain every sample within an interval of 1800 seconds from the beginning of the experiment. Otherwise, the process is interrupted (but not automatically and abruptly), the explanations for some of the samples are not extracted successfully and the table with results doesn't report the measures.

[2]https://pysmt.readthedocs.io/en/latest/

[3]https://www.ibm.com/docs/en/icos/20.1.0?topic=cplex-setting-up-python-api

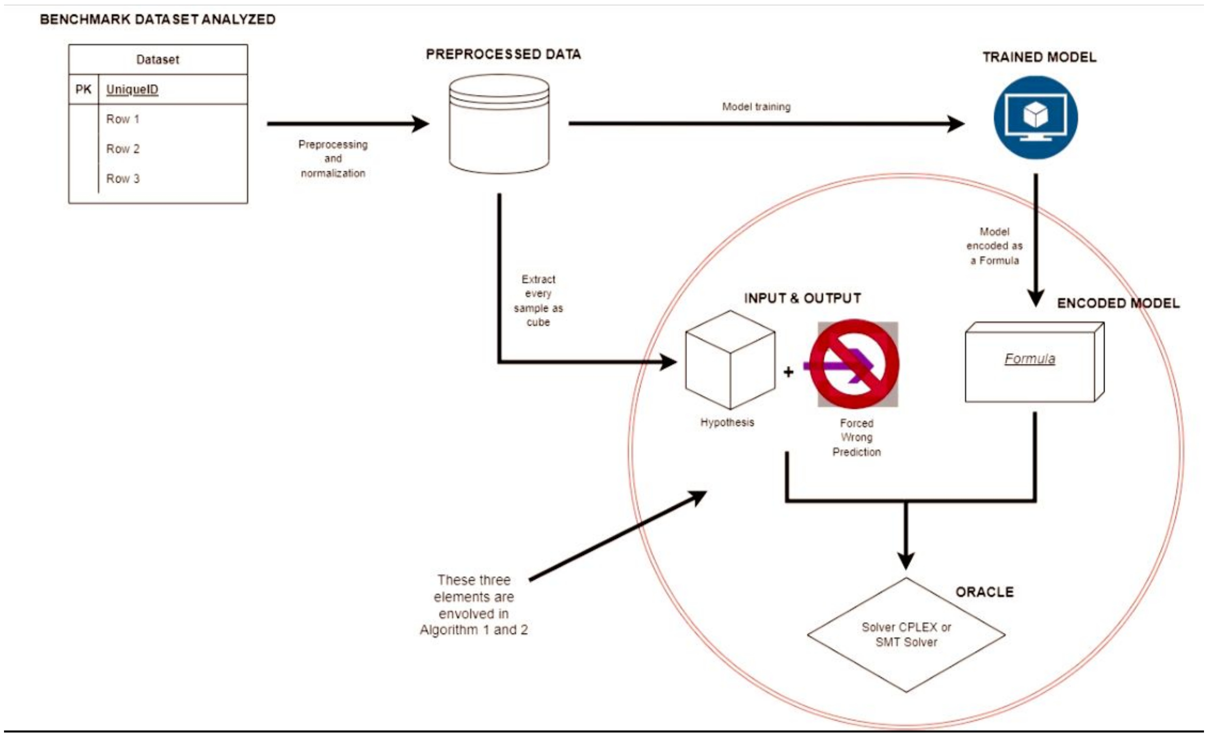
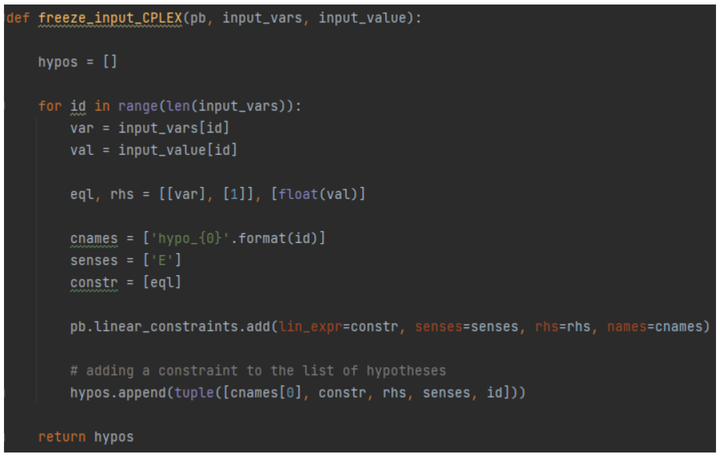

BENCHMARK DATASET ANALYZED
Dataset
PK
UniqueID
Row 1
Row 2
Row 3
Preprocessing and normalization
PREPROCESSED DATA
Model training
TRAINED MODEL
Model encoded as a Formula
Extract every sample as cube
INPUT & OUTPUT
ENCODED MODEL
Formula
Hypothesis
Forced Wrong Prediction
These three elements are envolved in Algorithm 1 and 2
ORACLE
Solver CPLEX or SMT Solver

Figure 6. Experiment phases

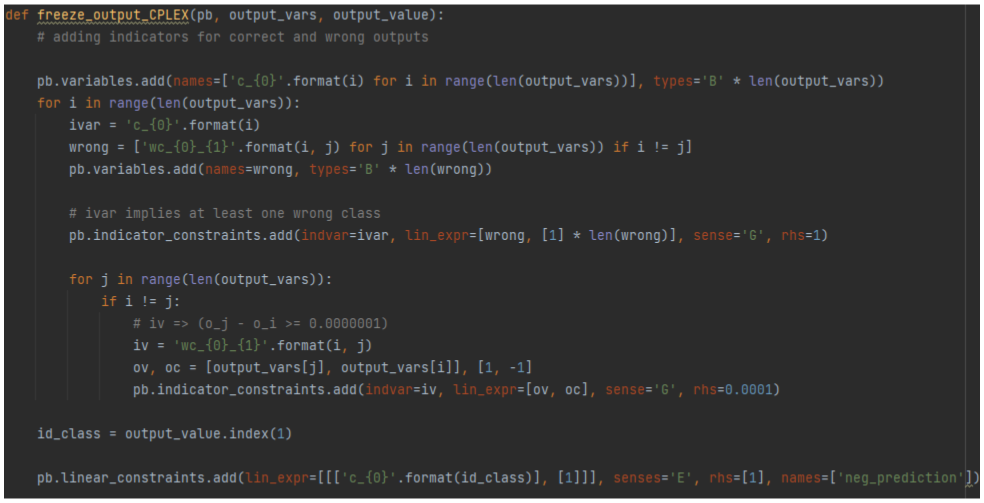
Every feature of every sample (cube under explanation phase) is added, accordingly to the algorithms, to the formulation as a set of clauses:



```python
def freeze_input_CPLEX(pb, input_vars, input_value):

    hypos = []

    for id in range(len(input_vars)):
        var = input_vars[id]
        val = input_value[id]

        eql, rhs = [[var], [1]], [float(val)]

        cnames = ['hypo_{0}'.format(id)]
        senses = ['E']
        constr = [eql]

        pb.linear_constraints.add(lin_expr=constr, senses=senses, rhs=rhs, names=cnames)

        # adding a constraint to the list of hypotheses
        hypos.append(tuple([cnames[0], constr, rhs, senses, id]))

    return hypos
```

Figure 7. Added Hypothesis to the problem formulation of the NN model

The output as it's predicted by the NN is negated, according to the fact that this formula-proving method works refutationally: this is translated into additional clauses for the encoded problem.



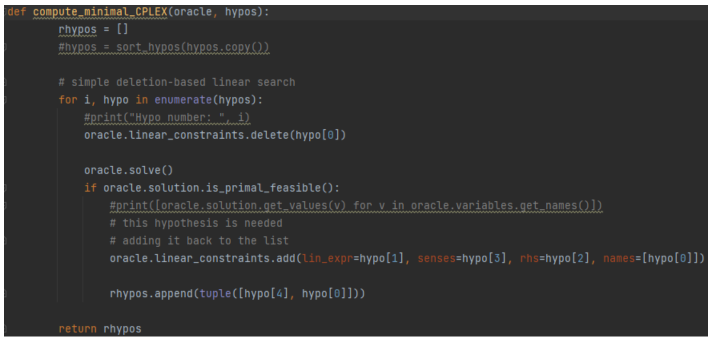
```python
def freeze_output_CPLEX(pb, output_vars, output_value):
    # adding indicators for correct and wrong outputs

    pb.variables.add(names=['c_{0}'.format(i) for i in range(len(output_vars))], types='B' * len(output_vars))
    for i in range(len(output_vars)):
        ivar = 'c_{0}'.format(i)
        wrong = ['wc_{0}_{1}'.format(i, j) for j in range(len(output_vars)) if i != j]
        pb.variables.add(names=wrong, types='B' * len(wrong))

        # ivar implies at least one wrong class
        pb.indicator_constraints.add(indvar=ivar, lin_expr=[wrong, [1] * len(wrong)], sense='G', rhs=1)

        for j in range(len(output_vars)):
            if i != j:
                # iv => (o_j - o_i >= 0.0000001)
                iv = 'wc_{0}_{1}'.format(i, j)
                ov, oc = [output_vars[j], output_vars[i]], [1, -1]
                pb.indicator_constraints.add(indvar=iv, lin_expr=[ov, oc], sense='G', rhs=0.0001)

    id_class = output_value.index(1)

    pb.linear_constraints.add(lin_expr=[[['c_{0}'.format(id_class)], [1]]], senses='E', rhs=[1], names=['neg_prediction'])
```

Figure 8. Forced wong output

This is the problem formulation:

```
[...]
hypo_0#17:         X_0_layer_0_type_C  = 30
hypo_1#18:         X_1_layer_0_type_C  = 21
hypo_2#19:         X_2_layer_0_type_C  = 27
hypo_3#20:         X_3_layer_0_type_C  = 69
hypo_4#21:         X_4_layer_0_type_C  = 5
hypo_5#22:         X_5_layer_0_type_C  = 44
hypo_6#23:         X_6_layer_0_type_B  = 1
hypo_7#24:         X_7_layer_0_type_B  = 0
hypo_8#25:         X_8_layer_0_type_B  = 0
hypo_9#26:         X_9_layer_0_type_B  = 1
hypo_10#27:        X_10_layer_0_type_B  = 0

[...]

neg_prediction#59: c_1  = 1

[...]

i63#30:           c_0 = 1 -> wc_0_1 => 1
 i64#31:           wc_0_1 = 1 -> X_1_layer_2_type_C - X_0_layer_2_type_C
                   => 0.0001
 i65#32:           c_1 = 1 -> wc_1_0 => 1
 i66#33:           wc_1_0 = 1 -> X_0_layer_2_type_C - X_1_layer_2_type_C
                   => 0.0001
```

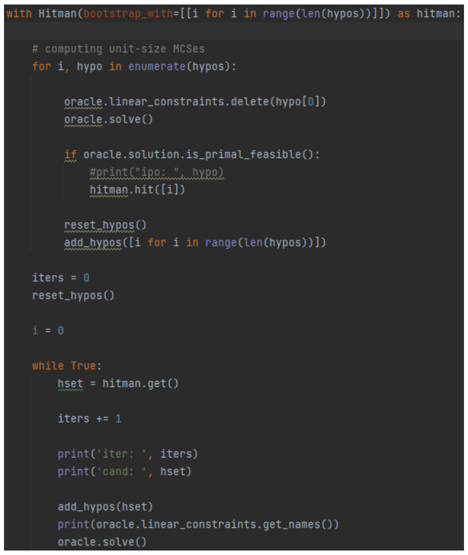
So, concretely, first algorithm consists of deleting iteratively all hypothesis and seeing if there's a solution: if it's found, this means that deleted hypo is fundamental for the classification, because the elimination provokes a misclassification.



```
def compute_minimal_CPLEX(oracle, hypos):
        rhypos = []
        #hypos = sort_hypos(hypos.copy())

        # simple deletion-based linear search
        for i, hypo in enumerate(hypos):
            #print("Hypo number: ", i)
            oracle.linear_constraints.delete(hypo[0])

            oracle.solve()
            if oracle.solution.is_primal_feasible():
                #print([oracle.solution.get_values(v) for v in oracle.variables.get_names()])
                # this hypothesis is needed
                # adding it back to the list
                oracle.linear_constraints.add(lin_expr=hypo[1], senses=hypo[3], rhs=hypo[2], names=[hypo[0]])

                rhypos.append(tuple([hypo[4], hypo[0]]))

        return rhypos
```

FIGURE 9. Here first algorithm implemented for CPLEX solver

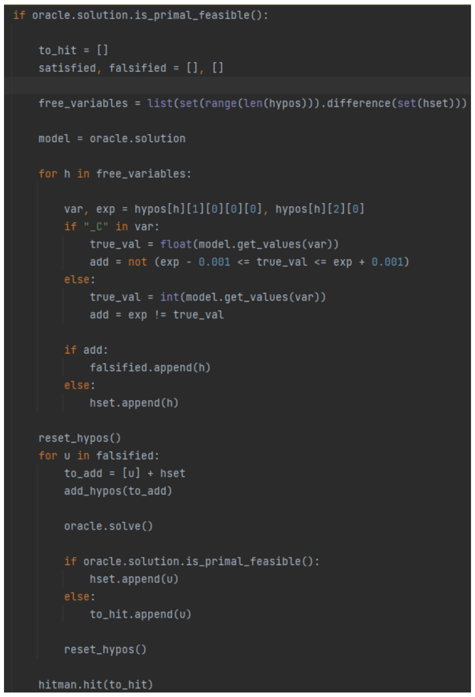
Second algorithm, instead, is first of all based on HitMan[4] object, a cardinality-/subset-minimal hitting set enumerator which is useful for finding everytime a possible hitting set candidate to be smallest explanation.



```
with Hitman(bootstrap_with=[[i for i in range(len(hypos))]]) as hitman:

    # computing unit-size MCSes
    for i, hypo in enumerate(hypos):

        oracle.linear_constraints.delete(hypo[0])
        oracle.solve()

        if oracle.solution.is_primal_feasible():
            #print("ipo: ", hypo)
            hitman.hit([i])

        reset_hypos()
        add_hypos([i for i in range(len(hypos))])

    iters = 0
    reset_hypos()

    i = 0

    while True:
        hset = hitman.get()

        iters += 1

        print('iter: ', iters)
        print('cand: ', hset)

        add_hypos(hset)
        print(oracle.linear_constraints.get_names())
        oracle.solve()
```

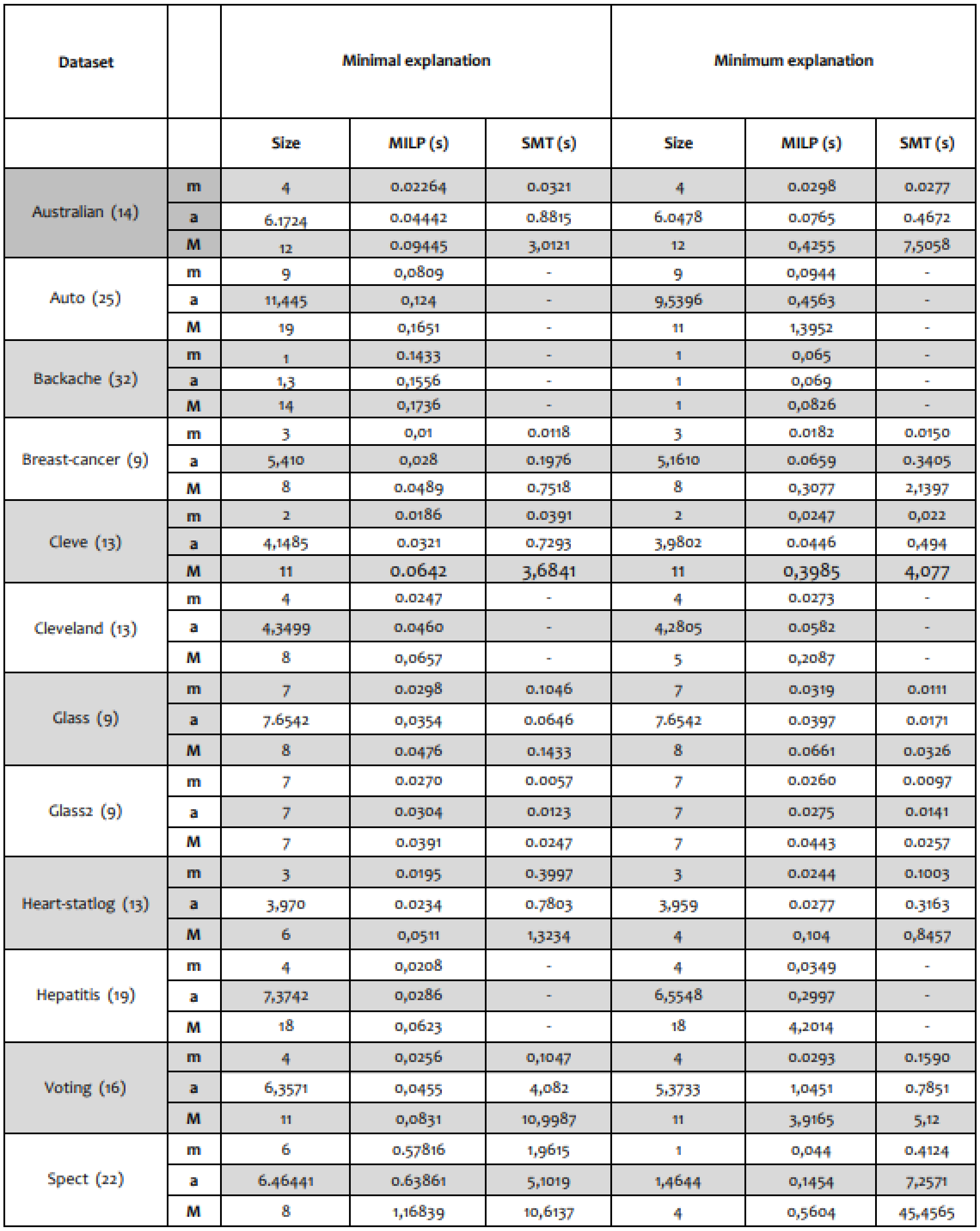
Hitting set is added as set of clauses to the problem and if there's no solution, desired explanation is found.
Otherwise, it's necessary to get an assignment from the free variables: during solution exploration, all free variables (not included in the candidate hitting set), whose computed value is different from the original one, are selected.
Every free variable is combined to the previous hitting set, now expanded with all not free variables, but initially not considered: again, if there's no solution, the examined free variable is added to a new set which will be added to the collection of sets to be hit, otherwise "dropped", but fixed, such as the other not considered variables.

[4]https://pysathq.github.io/docs/html/api/examples/hitman.html



```python
if oracle.solution.is_primal_feasible():

    to_hit = []
    satisfied, falsified = [], []

    free_variables = list(set(range(len(hypos))).difference(set(hset)))

    model = oracle.solution

    for h in free_variables:

        var, exp = hypos[h][1][0][0][0], hypos[h][2][0]
        if "_C" in var:
            true_val = float(model.get_values(var))
            add = not (exp - 0.001 <= true_val <= exp + 0.001)
        else:
            true_val = int(model.get_values(var))
            add = exp != true_val

        if add:
            falsified.append(h)
        else:
            hset.append(h)

    reset_hypos()
    for u in falsified:
        to_add = [u] + hset
        add_hypos(to_add)

        oracle.solve()

        if oracle.solution.is_primal_feasible():
            hset.append(u)
        else:
            to_hit.append(u)

        reset_hypos()

    hitman.hit(to_hit)
```

4.3. **Results.** Using a MILP oracle is preferred as it consistently outperforms its SMT rival. In general, the MILP-based solution is able to compute a subset-minimal explanation of a sample in a fraction of a second. Also it can be noted that subset- and cardinality-minimal explanation size varies a lot depending on the dataset.
On the one hand, it may be (*Backache*, for example) enough to keep just 1 feature to explain the outcome, which is ≈ 3,125% of the data samples. On average, the relative size of subset minimal explanations varies from 33,25% to 71,08% with the mean value being 39.57%. This is deemed to provide a reasonable reduction of sample size, which may help a human to interpret the outcomes of a machine learning classifier.
Compared to subset-minimal explanations, computing smallest size explanations is significantly more expensive, so more seconds are necessary, but allows to obtain explanation with a smaller size, accordingly to the definition of this kind of explanation.
The following table shows the results obtained running the experiment with twelve UCI and PennML datasets and two above quoted solvers under corresponding encoding, with registered time measures (in particular, minimum, average and maximum time of computations). Among the values, it's important to observe how for Congressional Voting Records dataset (named "*voting*") almost the same results of the Ignatiev's team are obtained, further validating their considerations about the experiments.



Dataset		Minimal explanation			Minimum explanation		
		Size	MILP (s)	SMT (s)	Size	MILP (s)	SMT (s)
Australian (14)	m	4	0.02264	0.0321	4	0.0298	0.0277
	a	6.1724	0.04442	0.8815	6.0478	0.0765	0.4672
	M	12	0.09445	3,0121	12	0,4255	7,5058
Auto (25)	m	9	0,0809	-	9	0,0944	-
	a	11,445	0,124	-	9,5396	0,4563	-
	M	19	0,1651	-	11	1,3952	-
Backache (32)	m	1	0.1433	-	1	0,065	-
	a	1,3	0,1556	-	1	0,069	-
	M	14	0,1736	-	1	0,0826	-
Breast-cancer (9)	m	3	0,01	0.0118	3	0.0182	0.0150
	a	5,410	0,028	0.1976	5,1610	0.0659	0.3405
	M	8	0.0489	0.7518	8	0,3077	2,1397
Cleve (13)	m	2	0.0186	0.0391	2	0,0247	0,022
	a	4,1485	0.0321	0.7293	3,9802	0.0446	0,494
	M	11	0.0642	3,6841	11	0,3985	4,077
Cleveland (13)	m	4	0.0247	-	4	0.0273	-
	a	4,3499	0.0460	-	4,2805	0.0582	-
	M	8	0,0657	-	5	0,2087	-
Glass (9)	m	7	0.0298	0.1046	7	0.0319	0.0111
	a	7.6542	0,0354	0.0646	7.6542	0.0397	0.0171
	M	8	0.0476	0.1433	8	0.0661	0.0326
Glass2 (9)	m	7	0.0270	0.0057	7	0.0260	0.0097
	a	7	0.0304	0.0123	7	0.0275	0.0141
	M	7	0.0391	0.0247	7	0.0443	0.0257
Heart-statlog (13)	m	3	0.0195	0.3997	3	0.0244	0.1003
	a	3,970	0.0234	0.7803	3,959	0.0277	0.3163
	M	6	0,0511	1,3234	4	0,104	0,8457
Hepatitis (19)	m	4	0,0208	-	4	0,0349	-
	a	7,3742	0,0286	-	6,5548	0,2997	-
	M	18	0,0623	-	18	4,2014	-
Voting (16)	m	4	0,0256	0,1047	4	0.0293	0.1590
	a	6,3571	0,0455	4,082	5,3733	1,0451	0.7851
	M	11	0,0831	10,9987	11	3,9165	5,12
Spect (22)	m	6	0.57816	1,9615	1	0,044	0.4124
	a	6.46441	0.63861	5,1019	1,4644	0,1454	7,2571
	M	8	1,16839	10,6137	4	0,5604	45,4565

4.4. **MNIST digits.** Last examined database for the study is MNIST database (Modified National Institute of Standards and Technology database), a large database of handwritten digits that is commonly used for training various image processing systems.
Accordingly to the work realized by the Ignatiev's team, MNIST is used to train two different neural network, with 15 and 20 neurons, to distinguish two digits (first time with 1 and 3, and the 3 and 8). Each MNIST data sample describes a picture of size 28 $\times$ 28, and so the total number of features is 784.
This number makes the experiment more challenging, because samples of Mnist have certainly bigger dimensions than samples of the other datasets: this fact brings to use only MILP oracle (because of the slowness of SMT Oracle), to not fix time limits as previously done and to not compute cardinality minimal explanations, too expansive in resource and time terms. The average size of subset-minimal explanations varies from 38.26% to 67.95% of the total number of pixels (i.e. 784), with the mean value being 52.65%.

4.4.1. *Here a legenda to interpret the following images:*

- Images are greyscale, but they are reduced to black and white tones, so these colors represent background and central figure respectively.
- Aqua pixels are those ones, belonging to the area of the digit inside the image (white pixels), resulted useful to the Neural model in order to obtain the given prediction
- Red pixels are considered by the Neural Model but they originally belong to the background (black pixels)

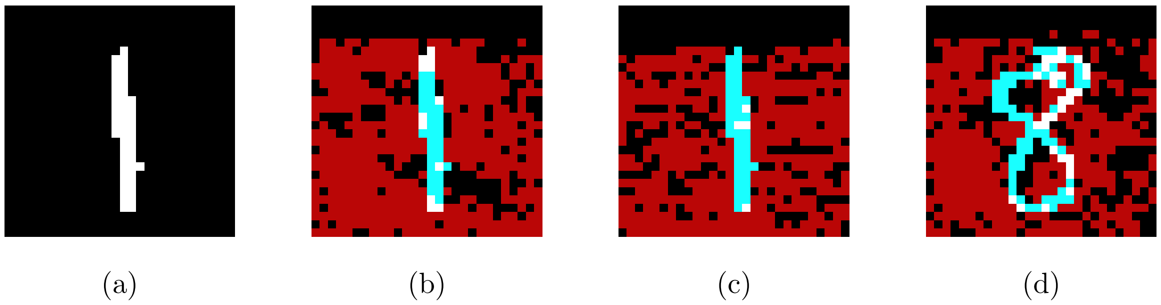

(a) (b) (c) (d)

Figure 10. Possible minimal explanation for digit "one"

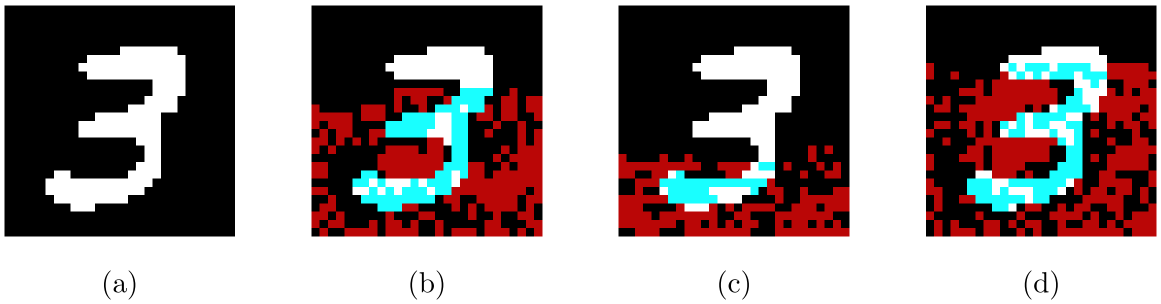
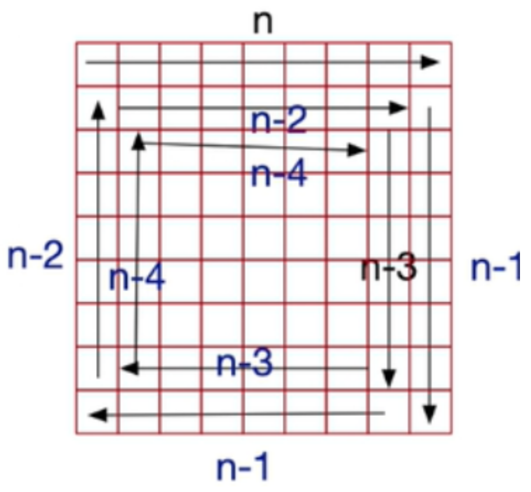


FIGURE 11. Possible minimal explanation for digit "three"

Figure 10 (a) and figure 11 (b) show two possible ways to write 1 and 3 digits, belonging to the findable writings in Mnist.
Pictures with red and magenta pixels represent possible subset minimal explanations occur for the experiment.
In particular, figures (b) show two possible explanations obtained according a simple implementation of the Algorithm 1: hypothesis are removed according the base sorting of the features, one by one starting from the top left corner and ending at the bottom right corner.
As observed by Ignatiev's team, this procedure ends up keeping the bottom part of the image as it's needed to forbid misclassifications while the top part of the image being removed. This is logically correct, but it can not satisfy an user who wants to understand why an image is classified as it is classified.
For this purpose, an alternative way of sorting hypothesis is proposed: according to the idea of removing pixels that are far from the image center, because the most important information on the image is typically located close to its center, hypothesis are sorted in a clockwise way (in ascending order):



Applying this strategy lets to better know how trained Neural Model makes his decision for the prediction: especially for the digit “3” (it’s evident with figure 11 (c)), the classifier distinguishes it from “1” because of the terminal part of the handwriting; what really is important for the decision is the appendix of the digit and its proximal pixels zone.
It’s also important to observe how explanations vary according to couple of numbers in exam: again, this is evident with digit “3”, one time compared to 1 and then to 8.
Figure 11(b) describes the first couple classification, and it has been just said that the terminal part is the discriminatory piece of the image; this is not valid for couple 8-3: model has to be more clinical in building the digit’s profiles, and what makes the difference is the left part of the 8, which is the “missing part” that can transform a 3 into a 8.

5. Conclusions

This work totally covers the Alexey Ignatiev's team experience with the study "*Abduction-Based Explanations for Machine Learning Models*": this has been possible with not few difficulties, because of the confusing and sometimes poor available documentation about solvers and their uses and especially for the brevity and the conciseness of the paper that explains the original work.
Anyway, this study belongs to a not conventional field of the IT, so every difficulty was predictable, but despite of these, at the end of the path, it revealed itself as a formative experience, which enters as an important part of the heritage that this study career brings with it.
According to these reasons, I can be satisfied of what I achieved and happy to reached a goal that initially seemed hard to reach.

References

[BS16] P.; Fischetti M.; Lodi A.; Monaci M.; Nogales-Gomez A.; Belotti, P.; Bonami and D. Salvagnin. On handling indicator constraints in mixed integer programming. In *Computational Optimization and Applications*, pages 545—-566, 2016.

[Dut14] B. Dutertre. Yices 2.2. In *Computer-Aided Verification*, pages 737—-744, 2014.

[EG95] T. Eiter and G Gottlob. The complexity of logicbased abduction. In *ACM*, 1995.

[FJ18] M. Fischetti and J Jo. Deep neural networks and mixed integer linear optimization. In *Constraints*, pages 296—-309, 2018.

[GM15] M. Gario and A Micheli. Pysmt: a solver-agnostic library for fast prototyping of smt-based algorithms. In *SMT Workshop*, 2015.

[HW17] M.; Wang S.; Huang, X.; Kwiatkowska and M Wu. Safety verification of deep neural networks. In *CAV*, pages 3—29, 2017.

[IBM18] IBM. Ibm ilog: Cplex optimizer 12.8.0. 2018.

[IMS16] F.; Narodytska N.; Ignatiev, A.; Pereira and J. Marques-Silva. Propositional abduction with implicit hitting sets. In *ECAI*, pages 1327—1335, 2016.

[IMS18a] A.; Ignatiev, A.; Morgado and J. Marques-Silva. Pysat: A python toolkit for prototyping with sat oracles. In *SAT*, pages 428—-437, 2018.

[IMS18b] F.; Narodytska N.; Ignatiev, A.; Pereira and J. Marques-Silva. A sat-based approach to learn explainable decision sets. In *IJCAR*, pages 627—-645, 2018.

[IMS19] N.; Ignatiev, A.; Narodytska and J Marques-Silva. On relating explanations and adversarial examples. In *NeurIPS*, pages 15857—-15867, 2019.

[IMS22] Y.; Stuckey J. P.; Ignatiev, A.; Izza and J. Marques-Silva. Using maxsat for efficient explanations of tree ensembles. 2022.

[KK17] C. W.; Dill D. L.; Julian K.; Katz, G.; Barrett and M. J Kochenderfer. Reluplex: An efficient smt solver for verifying deep neural networks. In *CAV*, pages 97—-117, 2017.

[LT18] N.; Pulina L.; Leofante, F.; Narodytska and A. Tacchella. Automated verification of neural networks: Advances, challenges and perspectives. In *CoRR abs*, 2018.

[Mar00] P Marquis. Consequence finding algorithms. In *Handbook of Defeasible Reasoning and Uncertainty Management Systems*, pages 41—-145, 2000.

[Pen21] PennML. 2021.

[Reg16] EU Data Protection Regulation. Regulation (eu) 2016/679 of the european parliament and of the council. 2016.

[Rep21] UCI Machine Learning Repository. 2021.

[RG18] S.; Ribeiro, M. T.; Singh and C Guestrin. Anchors: High-precision model-agnostic explanations. In *AAAI*, 2018.

[RK18] X.; Ruan, W.; Huang and M. Kwiatkowska. Reachability analysis of deep neural networks with provable guarantees. In *IJCAI*, pages 2651—-2659, 2018.