Assessing Heuristic Machine Learning Explanations with Model Counting *

N. Narodytska[1], A. Shrotri[2], K. Meel[3], A. Ignatiev[4,5], and J. Marques-Silva[4]

[1] VMware Research, CA, USA nnarodytska@vmware.com
[2] Rice University, Houston, USA as128@rice.edu
[3] National University of Singapore, Singapore meel@comp.nus.edu.sg
[4] Faculty of Science, University of Lisbon, Portugal
{aignatiev,jpms}@ciencias.ulisboa.pt
[5] ISDCT SB RAS, Irkutsk, Russia

Abstract. Machine Learning (ML) models are widely used in decision making procedures in finance, medicine, education, etc. In these areas, ML outcomes can directly affect humans, e.g. by deciding whether a person should get a loan or be released from prison. Therefore, we cannot blindly rely on black box ML models and need to explain the decisions made by them. This motivated the development of a variety of ML-explainer systems, including LIME and its successor ANCHOR. Due to the heuristic nature of explanations produced by existing tools, it is necessary to validate them. We propose a SAT-based method to assess the quality of explanations produced by ANCHOR. We encode a trained ML model and an explanation for a given prediction as a propositional formula. Then, by using a state-of-the-art approximate model counter, we estimate the quality of the provided explanation as the number of solutions supporting it.

1 Introduction

The advances in Machine Learning (ML) in recent years explain, to a large extent, the impact and societal significance commonly attributed to Artificial Intelligence (AI). As an indirect consequence, the fast growing range of ML applications includes settings where safety is paramount (e.g. self-driving vehicles), but also settings were humans are directly affected (e.g. finance, medicine, education, and judiciary). Work on improving confidence in ML models includes both solutions for verifying properties of these models, but also approaches for explaining predictions, namely in situations where the operation of the ML model is not readily interpretable by a human decision maker. The general field of eXplainable AI (XAI) targets both the development of naturally interpretable ML models (e.g. decision trees or sets), but also the computation of explanations in settings where the ML model is viewed as a black-box, e.g. neural networks, ensembles of classifiers, among others.

The best-known XAI approaches are heuristic-based, and offer so-called *local* explanations, in the sense that the space of feature values is *not* analyzed exhaustively.

* This work was supported by FCT grants ABSOLV (LISBOA-01-0145-FEDER-028986), FaultLocker (PTDC/CCI-COM/29300/2017), SAFETY (SFRH/BPD/120315/2016), SAMPLE (CEECIND/04549/2017), National Research Foundation Singapore under its AI Singapore Programme AISG-RP-2018-005 and NUS ODPRT Grant R-252-000-685-133.

Among a number of recently proposed approaches for computing local explanations, LIME [23] and its successor ANCHOR [24] represent two successful instantiations. However, and since both LIME and ANCHOR are heuristic-based, a natural question is: *how accurate in practice are heuristic-based explanations?* This paper focuses on ANCHOR [24] (since it improves upon LIME [23]) and proposes a novel approach for assessing the quality of heuristic approaches for computing (local) explanations. For each computed explanation, ANCHOR reports a measure of quality, namely the estimated precision of the explanation, i.e. the percentage of instances where the explanation applies and the prediction matches. Starting from an encoding of the ML model, the explanation computed by ANCHOR, and the target prediction, this paper proposes to use (approximate) model counting for assessing the actual precision of the computed explanation. Concretely, the paper considers Binarized Neural Networks (BNNs) [15], exploits a recently proposed propositional encoding for BNNs [22], and assesses the quality of ANCHOR on well-known datasets. As we demonstrate, the quality of ANCHOR's explanations can vary wildly, indicating that there are datasets where the explanations of ANCHOR are fairly accurate, but also that there are datasets where the explanations of ANCHOR can be rather inaccurate. The somewhat unexpected conclusions of our experimental evaluation offer further evidence to the need of formal techniques in XAI.

2 Preliminaries

Boolean Satisfiability. We assume notation and definitions standard in the area of Boolean Satisfiability (SAT), i.e. the decision problem for propositional logic [5]. Formulas are represented in Conjunctive Normal Form (CNF) and defined over a set of variables $Y = \{y_1, \ldots, y_n\}$. A CNF formula $\mathcal{F}$ is a conjunction of clauses, a clause is a disjunction of literals, and a literal l_i is a variable y_i or its complement $\neg y_i$. A truth assignment is a map from variables to $\{\text{FALSE}, \text{TRUE}\}$. Given a truth assignment, a clause is satisfied iff at least one of its literals is assigned value TRUE. A formula is *satisfied* iff all of its clauses are satisfied. If there is an assignment μ that satisfies formula $\mathcal{F}$, then $\mathcal{F}$ is said to be *satifiable*, and assignment μ is called a *model* of formula $\mathcal{F}$.

CNF encodings of *cardinality constraints*, i.e. constraints of the form $\sum_{i=0}^{n} l_i \circ k$ where $\circ \in \{<, \leq, =, \neq, \geq, >\}$, have been studied extensively, and will be assumed throughout [5]. In this work we employ *reified* cardinality constraints: $y \Leftrightarrow \sum_{i=0}^{n} l_i \geq k$. We use the full sequential counters encoding [31] to model this constraint in SAT. However, other encodings can be used.

Model counting. Given a CNF formula, the problem of model counting is to calculate the number satisfying assignments or models. For a formula $\mathcal{F}$, we denote its count by $\#\mathcal{F}$. This problem is complete for the complexity class $\#\mathcal{P}$ [36], which contains the entire polynomial hierarchy [35]. Despite the high complexity, a number of tools for exact model counting have been developed [27,34,21,17], which were shown to work well on certain benchmarks arising in practice. However, for many applications, obtaining exact counts is not necessary and a good approximation often suffices, especially when it leads to better scalability. These requirements have led to the emergence of *approximate* counting approaches [13,7,32] which employ universal hash functions [6] along with specialized SAT solvers [33] for balancing accuracy and scalability. In our experiments, we use a state-of-the-art tool called ApproxMC3 [32] which takes as input a CNF formula $\mathcal{F}$ along with a tolerance value ε and confidence δ, and outputs an approximate count C such that $\Pr[\frac{1}{(1+\varepsilon)}\#\mathcal{F} \leq C \leq (1+\varepsilon)\#\mathcal{F}] \geq 1-\delta$, where the probability is defined over the random choice of hash functions. A key advantage of ApproxMC3 is that it supports *projected* model counting, i.e. for a formula $\mathcal{F}$ over variable set $Y = Y_1 \cup Y_2$, ApproxMC3 can approximately count the number of assignments to the variables in Y_1, called *sampling set*, such that the formula $\exists Y_2(\mathcal{F})$ evaluates to true. Since we use parsimonious encodings of cardinality constraints in the current work, projection is not strictly required from a correctness perspective. Nevertheless, it greatly speeds up computation as our encoding scheme provides us access to independent support [1], which is specified as sampling set. Furthermore, usage of projection may be necessary for applying our approach to other encodings and more complex models.

Classification problems. We define the supervised classification problem. We are given a dataset of training samples (i.e. a training set) $\mathcal{E} = \{e_1, \ldots, e_M\}$ over a set of categorical features $\mathcal{F} = \{f_1, \ldots, f_K\}$. Each sample e_i is a pair $\{(v_1^i, \ldots, v_K^i), q^i\}$, where $(v_1^i, \ldots, v_K^i)$ are values of the corresponding features, $v_i^j \in \mathbb{Z}$, and q^i determines the class of the sample e_i, $q^i \in Q$, where Q is a set of possible classes. Note that we assume that all features are categorical, so v_i^js take only discrete values.

Solving a classification problem consists of building a classifier function that maps a sample to its class label, $\mathcal{G} : \{0,1\}^K \to Q$. Given a training set, a classifier is learned during the training phase so that it captures the relationship between samples and class labels. After the training phase is complete, the classifier is *fixed*. W.l.o.g. we work with a binary classification problem, so that $Q = \{0,1\}$. However, our approach also works for the classification with multiple class labels case, without additional modifications.

Binarized Neural Networks. A binarized neural network (BNN) is a feedforward network where weights and activations are binary and take values $\{-1,1\}$ [15]. A BNN is built from blocks of layers. Each block performs a number of linear and non-linear transformations such as batch normalization, hyperbolic tangent, and binarization. While internal layers of a block can produce real-valued intermediate outputs, the output of the block is a binary vector. The output of the last layer is a real-valued vector of size $|Q|$ that is used to determine the winner.

3 Explanations for Machine Learning Models

There is a large body of work on generating explanations for ML models [26,20,19,25,37,29,2,3]. The main motivation for this line of research is the practical need to interpret, analyze and understand decisions of ML models, as these decisions can affect humans. For example, ML models can be used to decide whether a person should get a loan or be released from prison [16,28]. There are multiple ways to attack the explainability problem depending on our assumptions about the model. One approach is to treat the model as a white box. For example, we can analyze an ML model and extract a (nearly) equivalent interpretable model, e.g. one can distill a neural network into a decision tree model [14,38]. However, this approach has several drawbacks, e.g. a converter needs to be developed for each pair of model classes, the extraction of an equivalent model can be computationally hard, etc. There are also heuristic gradient-based white-box methods [30], but they are mostly designed for computer vision tasks.

An alternative approach is to treat the ML model as a black box. Methods that make no assumptions about the underlying ML model are known as model-agnostic explanation generators. Since working with black-box models is challenging, these methods are restricted to finding explanations that hold *locally* in a subspace of the input space. Prominent examples include ML-explainer LIME and its successor ANCHOR [23,24].

In this work, we consider ANCHOR, proposed by Ribeiro *et al.* [24], which is the only system that generates explanations, but also provides quality metrics for them. However, these metrics are obtained purely heuristically and are not guaranteed to be accurate. We take this approach a step further, proposing a rigorous method with guaranteed error bounds. Since this problem is intractable in the general case, we focus on types of ML models that allow a succinct representation as a Boolean formula. For such models, we reformulate the computation of quality metrics as a Boolean logic problem, specifically the problem of determining the number of solutions of a Boolean formula.

3.1 ANCHOR's Heuristic Explanations

We start by describing the concepts behind ANCHOR. We use notations and definitions from [24]. ANCHOR introduces the notion of an anchor explanation, which is *"a rule [over inputs] that sufficiently anchors the prediction locally so that changes to the rest of inputs does not affect the prediction"*. Let $e = ((v_1, \ldots, v_K), e_q)$ be a sample from the dataset $\mathcal{E}$ and let us denote vector $(v_1, \ldots, v_K)$ of input feature values by e_v.

Example 1. Consider a sample e from the *adult* dataset describing characteristics of a person [16]. A sample contains twelve features such as age, sex, race, occupation, marriage status, etc. We consider a binary classifier that predicts whether the income of a person is above or below \$50K. Suppose we have a sample:

$$e = ((\text{Age is } (37, 48], \text{Private, Husband, Sales, White, Male, Married}), > \$50\text{K}).$$

Hence, $e_v = (\text{Age is } (37, 48], \text{Private, Husband, Sales, White, Male, Married})$ is the vector of features (some features are undefined) and the class label e_q is $> \$50$K. □

Let A be a set of predicates over input features e_v such that e_v satisfies all predicates in A, denoted $A(e_v) = 1$. An anchor is defined as follows.

Definition 1. *A set of unary predicates A is an anchor for prediction e_q of model $\mathcal{G}$ if*

$$\mathbb{E}_{\mathcal{D}(e'_v|A)}[\mathcal{G}(e'_v) = e_q] \geq \tau, A(e_v) = 1, \tag{1}$$

where $\mathcal{D}(e'_v \mid A)$ denotes samples that satisfy A and τ is a parameter close to 1. In other words, A is an anchor if all samples that match A are most likely classified as e_q.

Example 2. Continuing with our example, the ANCHOR algorithm starts from a sample e and a trained ML model. Assume that it produces an anchor $A = (\text{White, Male})$. The interpretation of this anchor is that if we consider a population of white males then the ML model mostly likely predicts the class $> \$50$K. □

To estimate the quality of the explanation, the following precision metric of anchor was introduced:

$$prec(A) = \mathbb{E}_{\mathcal{D}(e'_v|A)}[\mathcal{G}(e'_v) = e_q], \tag{2}$$

In other words, the precision metric measures the fraction of samples that are classified as e_q among those that match A. As pointed out in [24], for an arbitrary dataset and a black-box model, *it is intractable to compute this precision directly*, prompting a probabilistic definition of precision $P(prec(A) \geq \tau) \geq 1-\delta$, where τ and δ are parameters of the algorithm. The problem of finding a high-precision anchor was formulated as a pure-exploration multi-armed bandit problem and solved using a dedicated algorithm. In our example, the precision of A is 0.99%.

Indeed, for an arbitrary model computing the exact precision value is intractable. However, we argue that for a rich class of models we can compute the precision metric exactly or get a good approximation of this value.

3.2 Model-Counting Based Assessment of Explanations

In this section, we discuss how to evaluate the quality of explanations produced by ANCHOR for a subclass of ML models. Our main idea is to compute the precision metric directly using an logical representation of the ML model. Here we focus on ML models that admit an equivalent CNF encoding. For such models, we can formulate the problem of computing the precision metric as a model counting problem. To obtain such formulation, we need to encode the following three components as a CNF: (a) the ML model, (b) the anchor and (c) the set of valid inputs.

First, we define a subclass of ML models suitable for our approach. Consider an ML model $\mathcal{G}$ that maps an input vector x to an output vector o, $o = \mathcal{G}(x)$. The prediction of the model is given by ARGMAX(o).For simplicity, we consider a binary classification problem, so we have two classes, hence, o is a 2-dimensional vector. We require that there exists a CNF representation of the model, denoted BIN$\mathcal{G}(x, s)$, that simulates the model in the following sense: all models of BIN$\mathcal{G}(x, s)$ such that $s = 0$ are exactly the samples that are classified as class 0 by $\mathcal{G}$. Likewise, all models of BIN$\mathcal{G}(x, s)$ such that $s = 1$ are samples that are classified as class 1 by $\mathcal{G}$. For now, we assume that such models exist and consider a concrete classifier in the next session. Second, we consider a CNF encoding of an anchor A. We recall that A is a set of unary predicates over input categorical features, so it can be easily translated to CNF. We denote a CNF encoding of A as BINA. Third, we need a declarative representation of the space of samples. We require that a set of valid samples can be defined using a propositional formula that we denote by BIN$\mathcal{D}(x)$. The assumption that the input space can be represented as a logical formula is standard in the line of work on verification of neural networks [18,12]. However, it might not hold for some datasets, like images. An encoding of a valid instance space is an interesting research direction, which is outside the scope of this work. There are a number of natural cases that allow such representation. First, we can assume that all possible combinations of input features are valid. For instance, in Example 1, any combination of features is a plausible sample. Second, we may want to restrict the considered inputs, e.g. we would like to consider all inputs that are close to a given sample e w.r.t. a given distance measure. Finally, a user might have a set of preferences over samples that are expressible as a Boolean formula.

Now we put all three components together into the following formula

$$\mathcal{P}_A(x, s) = \text{BIN}\mathcal{G}(x, s) \wedge \text{BINA}(x) \wedge \text{BIN}\mathcal{D}(x). \tag{3}$$

Structure of kth internal block, $\textsc{Block}_k : \{-1,1\}^{n_k} \to \{-1,1\}^{n_{k+1}}$ on input $x_k \in \{-1,1\}^{n_k}$	
LIN	$y = A_k x_k + b_k$, where $A_k \in \{-1,1\}^{n_{k+1} \times n_k}$ and $b_k \in \mathbb{R}^{n_{k+1}}$
BN	$z_i = \alpha_{k_i} \left(\frac{y_i - \mu_{k_i}}{\sigma_{k_i}} \right) + \gamma_{k_i}$, where $y = (y_1, \ldots, y_{n_{k+1}})$, and $\alpha_{k_i}, \gamma_{k_i}, \mu_{k_i}, \sigma_{k_i} \in \mathbb{R}^{n_{k+1}}$
BIN	$x_{k+1} = \text{sign}(z)$ where $z = (z_1, \ldots, z_{n_{k+1}}) \in \mathbb{R}^{n_{k+1}}$ and $x_{k+1} \in \{-1,1\}^{n_{k+1}}$

Structure of output block, $\textsc{Output} : \{-1,1\}^{n_d} \to \{0,1\}$ on input $x_d \in \{-1,1\}^{n_d}$	
LIN	$w = A_d x_d + b_d$, where $A_d \in \{-1,1\}^{2 \times n_d}$ and $b_d \in \mathbb{R}^2$
BN	$o = \alpha_o \left(\frac{w - \mu_o}{\sigma_o} \right) + \gamma_o$, where $w = (w_1, w_2)$, and $\alpha_o, \gamma_o, \mu_o, \sigma_o \in \mathbb{R}^2$

Table 1: Structure of internal and output blocks which, stacked together, form a BNN.

Then the precision of an anchor A for model $\mathcal{G}$ and prediction e_q defined by (2) can be written as

$$\text{M} = \frac{\#(\mathcal{P}_A(x, s) \wedge s = e_q)}{\#(\mathcal{P}_A(x, s))}. \tag{4}$$

In other words, M measures the fraction of models that are classified as e_q among those that match A and satisfy $\mathcal{D}$. Hence, this is exactly the fraction of models that defines the precision of the anchor A in (2). In practice, even for small ML models exact model counting is not feasible. Therefore, we use an efficient approximate model counting algorithm called ApproxMC3.

4 Encoding of Binarized Neural Networks

Let us discuss BNNs, which is our underlying machine learning model. A BNN can be described in terms of blocks of layers that map binary vectors to binary vectors. Hence, we define a *block* of binarized neural network (referred to as BLOCK) as a function that maps an input x to an output x', i.e. $\textsc{Block} : \{-1,1\}^n \to \{-1,1\}^m$. The last block has a different structure $\textsc{Output} : \{-1,1\}^n \to \mathbb{R}^2$. Each BLOCK takes an input vector x and applies three transformations: LIN, BN and BIN [6]. Table 1 shows transformations of internal and output blocks in detail. It was shown in [22], that BLOCK can be encoded as a system of reified cardinality constraints that can be translated into a Boolean formula efficiently [31]. We recall the main idea of the encoding.

Encoding of BLOCK. BLOCK applies three transformations: LIN, BN and BIN to an input vector x. The main insight here is that instead of applying these transformations sequentially, we can consider a composition of these functions. Hence, we relate input and output of a block as follows.

$$x' = \begin{cases} 1, & \text{if } \alpha \left(\frac{(Ax+b)-\mu}{\sigma} \right) + \gamma \geq 0 \\ -1, & \text{otherwise} \end{cases}$$

Next we note that we can re-group the inequality condition in such a way that the left side of the inequality must take the integer value and the right side is a real value. Namely, assuming $\alpha > 0$ ($\alpha < 0$ is similar), we rewrite inequality condition as $Ax \geq \frac{-\gamma\sigma}{\alpha} - b + \mu$. Note that Ax is an integer vector and $\frac{-\gamma\sigma}{\alpha} - b + \mu$ is a real-valued vector.

[6] In the training phase, there is an additional *hard tanh* layer after batch normalization but it is redundant in the inference phase.

Hence, we can perform rounding of the right hand side safely. This gives a relation that contains only integers, so it can be encoded into CNF efficiently.

Encoding of OUTPUT. For our purpose, we only need to know whether $o_1 \geq o_2$. We introduce a Boolean variable s that captures this relation.

$$s = \begin{cases} 0, & \text{if } \alpha_{o_1}\left(\frac{w_1-\mu_{o_1}}{\sigma_{o_1}}\right) + \gamma_{o_1} \geq \alpha_{o_2}\left(\frac{w_2-\mu_{o_2}}{\sigma_{o_2}}\right) + \gamma_{o_2} \\ 1, & \text{otherwise} \end{cases} \quad (5)$$

As $w = A_d x_d + b_d$, we can re-group the inequality condition the same way as for the BLOCK to obtain a reified constraint over integers. So, we have that $s = 0$ iff the prediction is the 0th class and $s = 1$ otherwise.

CNF Encoding. Based on the encoding of blocks BLOCK and OUTPUT, the network can be represented as a Boolean formula: $\text{BINBNN}(x, s) \equiv \bigwedge_{i=1}^{D} \text{BINBLOCK}_d(x^{d-1}, x^d) \wedge \text{BINO}(x_D, s)$, where $\text{BINBLOCK}_d(x^{d-1}, x^d)$ encodes the dth block BLOCK with an input x_d and an output x_{d+1}, $d \in [1, D]$, BINO is a Boolean encoding of the last set of layers. Note that $\text{BINBNN}(x, s)$ satisfies the requirements for ML model encoding that we stated in the previous section.

5 Experimental Results

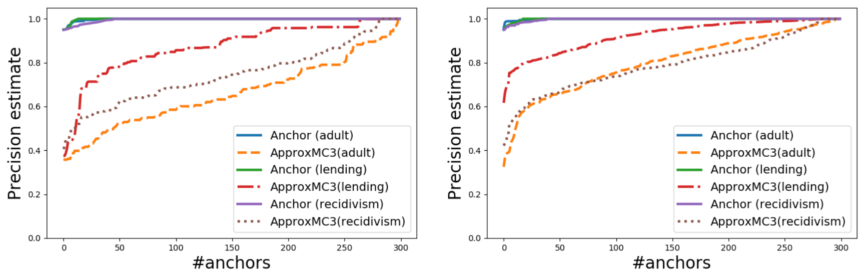
In this section we present results of our experimental evaluation. Our goal is to assess the quality of ANCHOR's precision metric.

Datasets. We consider three well-known publicly available datasets that were used in [24]. These datasets were processed the same way[7] as in [24]. The *adult* dataset [16] was collected by the Census bureau, where it was used to predict whether a given adult person earns more than \$50K a year depending on various attributes. The *lending* dataset was used to predict whether a loan on the Lending Club website will be granted. The *recidivism* dataset was used to predict recidivism for individuals released from North Carolina prisons in 1978 and 1980 [28].

ML model. We trained a BNN on each benchmark with three internal BLOCKs and an OUTPUT block. There are 25 neurons per layer on average. We use a standard onehot encoding to convert categorical features to Boolean inputs for BNN. We split the input data into training (80% of data) and test (20%) sets. The accuracy of BNN is 0.82 for the *adult*, 0.83 for *lending* and 0.63 for *recidivism*, which matches XGBoost [9] used in [24]. To generate anchors, we used the native implementation of ANCHOR. The sizes of CNF encodings are (a) $50K$ variables and $202K$ clauses for *adult*, (b) $21K$ variables and $80K$ clauses for *lending*, (c) $75K$ variables and $290K$ clauses for *recidivism*.

Quality assessment. We performed two sets of experiments depending on constraints on the sample space. First, we consider the case of an unrestricted set of samples. Second, we restrict samples to a local neighborhood of a sample we started with to generate an anchor. For the first case, the rejection sampling based algorithm by Dagum *et al.* [10] can be used for measuring accuracy up to desired tolerance and confidence, very efficiently. However, in general there can be additional constraints over the input, such as for encoding some notion of neighborhood of a sample, or for incorporating a set of user preferences. In such cases, rejection sampling based approaches fail. To

[7] https://github.com/marcotcr/anchor-experiments



(a) Unrestricted set of samples (b) Restricted set of samples

Fig. 1: The precision metric estimates for three datasets.

ensure wider applicability, we use the tool ApproxMC3, which can perform projected counting over the input variables of arbitrary constraints encoded as CNF formulas.

We invoke ApproxMC3 with the default tolerance and confidence ($\varepsilon = 0.8$ and $\delta = 0.2$), which has been the standard in earlier works [7,8,32]. Studies have reported that the error observed in practice ($\varepsilon_{obs} = 0.037$) is an order of magnitude better than the theoretical guarantee of 0.8 [32], which is similar to what we found in our preliminary experiments even when considering the quotient of two approximate counts as in (4). This obviates the need for stronger (ε, δ) as input, as the default settings are sufficiently accurate in practice for our purposes.

Unrestricted set of samples. Here we consider the case when we do not put any restrictions of a set of samples. We compute high-precision anchors for 300 randomly selected inputs of each test dataset with a default value of $\tau = 0.95$. On average, the precision metric reported by ANCHOR is high, over 0.99. For each anchor, we perform approximate model counting of solutions according to (4). Then, we compute the discrepancy between the precision metric returned by ANCHOR and the estimate computed by our method.

Figure 1a shows our results. Each cactus plot shows the precision that is returned by ANCHOR and is computed with ApproxMC3 for the corresponding dataset. ANCHOR's precision estimates are around 0.99 for the three datasets and so the corresponding lines in Figure 1 merge. Figure 1 shows that ANCHOR's estimates of the precision metric are good for the *lending* dataset. On average the discrepancy is 0.13 in this set. In contrast, the discrepancy was high in the *adult* dataset, 0.34 on average. For the *recidivism* dataset the mean discrepancy is 0.25. Overall, we can conclude that the metric produced by ANCHOR is more on the optimistic side, as we cannot confirm 0.99 precision.

Constrained set of samples. Second, we consider the case when we want to restrict the space of samples. One interesting case is to consider how good the anchor are among the samples that are close to the original sample e we started with. We define a neighborhood of e given an anchor A as all samples that match A and differ from e in at most 50% of the remaining features. We expect that ANCHOR performs better in the local neighborhood of e. Figure 1b shows our results. We obtain that on average the discrepancy is 0.08 for the *lending* dataset, 0.2 for the *adult* dataset and 0.21 for *recidivism*. So, we see a significant improvement for the *adult* dataset (the discrepancy dropped from 0.34 to 0.2) and minor improvements for the other two sets.

6 Conclusions and Future Work

This paper investigates the quality of heuristic (or local) explanations computed by a recent XAI tool, namely ANCHOR [24]. Although for some datasets the precisions claimed by Anchor can be confirmed, it is also the case that for several other datasets Anchor estimates precisions that are unrealistically high. There are a number of possible directions for future work. For example, it is interesting to consider powerful model compilation techniques [11] that can compute the exact number of solutions and, therefore, the precision metrics exactly. The main challenge here is to build an effective compiler from BNNs to BDDs [4]. Another direction is to investigate the application of the ideas in this paper to other XAI tools and consider other ML models that can be translated to SAT.

References

1. On computing minimal independent support and its applications to sampling and counting. Constraints **21**(1), 41–58 (2016)
2. Adebayo, J., Gilmer, J., Muelly, M., Goodfellow, I.J., Hardt, M., Kim, B.: Sanity checks for saliency maps. In: NeurIPS. pp. 9525–9536 (2018)
3. Alvarez-Melis, D., Jaakkola, T.S.: Towards robust interpretability with self-explaining neural networks. In: NeurIPS. pp. 7786–7795 (2018)
4. Andy Shih, A.D., Choi, A.: Verifying binarized neural networks by local automaton learning. In: VNN (2019)
5. Biere, A., Heule, M., van Maaren, H., Walsh, T. (eds.): Handbook of Satisfiability, Frontiers in Artificial Intelligence and Applications, vol. 185. IOS Press (2009)
6. Carter, J.L., Wegman, M.N.: Universal classes of hash functions. In: Proc. of STOC. pp. 106–112. ACM (1977)
7. Chakraborty, S., Meel, K.S., Vardi, M.Y.: A scalable approximate model counter. In: Proc. of CP. pp. 200–216 (2013)
8. Chakraborty, S., Meel, K.S., Vardi, M.Y.: Improving approximate counting for probabilistic inference: From linear to logarithmic sat solver calls. In: Proceedings of International Joint Conference on Artificial Intelligence (IJCAI) (7 2016)
9. Chen, T., Guestrin, C.: XGBoost: A scalable tree boosting system. In: KDD. pp. 785–794. ACM (2016)
10. Dagum, P., Karp, R., Luby, M., Ross, S.: An optimal algorithm for Monte Carlo estimation. SIAM Journal on Computing **29**(5), 1484–1496 (2000)
11. Darwiche, A., Marquis, P.: A knowledge compilation map. J. Artif. Intell. Res. **17**, 229–264 (2002). https://doi.org/10.1613/jair.989, https://doi.org/10.1613/jair.989
12. Dreossi, T., Ghosh, S., Sangiovanni-Vincentelli, A.L., Seshia, S.A.: A formalization of robustness for deep neural networks. CoRR **abs/1903.10033** (2019), http://arxiv.org/abs/1903.10033
13. Ermon, S., Gomes, C.P., Sabharwal, A., Selman, B.: Taming the curse of dimensionality: Discrete integration by hashing and optimization. In: Proc. of ICML. pp. 334–342 (2013)
14. Frosst, N., Hinton, G.E.: Distilling a neural network into a soft decision tree. In: Besold, T.R., Kutz, O. (eds.) Proceedings of the First International Workshop on Comprehensibility and Explanation in AI and ML 2017 co-located with 16th International Conference of the Italian Association for Artificial Intelligence (AI*IA 2017), Bari, Italy, November 16th and 17th, 2017. CEUR Workshop Proceedings, vol. 2071. CEUR-WS.org (2017)
15. Hubara, I., Courbariaux, M., Soudry, D., El-Yaniv, R., Bengio, Y.: Binarized neural networks. In: NIPS. pp. 4107–4115 (2016)

16. Kohavi, R.: Scaling up the accuracy of naive-bayes classifiers: A decision-tree hybrid. In: KDD. pp. 202–207 (1996)
17. Lagniez, J.M., Marquis, P.: An improved decision-dnnf compiler. In: IJCAI. pp. 667–673 (2017)
18. Leofante, F., Narodytska, N., Pulina, L., Tacchella, A.: Automated verification of neural networks: Advances, challenges and perspectives. CoRR **abs/1805.09938** (2018), http://arxiv.org/abs/1805.09938
19. Li, O., Liu, H., Chen, C., Rudin, C.: Deep learning for case-based reasoning through prototypes: A neural network that explains its predictions. In: AAAI. pp. 3530–3537 (2018)
20. Montavon, G., Samek, W., Müller, K.: Methods for interpreting and understanding deep neural networks. Digital Signal Processing **73**, 1–15 (2018)
21. Muise, C., McIlraith, S.A., Beck, J.C., Hsu, E.: DSHARP: Fast d-DNNF Compilation with sharpSAT. In: Canadian Conference on Artificial Intelligence (2012)
22. Narodytska, N., Kasiviswanathan, S.P., Ryzhyk, L., Sagiv, M., Walsh, T.: Verifying properties of binarized deep neural networks. In: AAAI. pp. 6615–6624 (2018)
23. Ribeiro, M.T., Singh, S., Guestrin, C.: "Why should I trust you?": Explaining the predictions of any classifier. In: KDD. pp. 1135–1144 (2016)
24. Ribeiro, M.T., Singh, S., Guestrin, C.: Anchors: High-precision model-agnostic explanations. In: AAAI (2018)
25. Ross, A.S., Doshi-Velez, F.: Improving the adversarial robustness and interpretability of deep neural networks by regularizing their input gradients. In: AAAI. pp. 1660–1669 (2018)
26. Ross, A.S., Hughes, M.C., Doshi-Velez, F.: Right for the right reasons: Training differentiable models by constraining their explanations. In: IJCAI. pp. 2662–2670 (2017)
27. Sang, T., Beame, P., Kautz, H.: Performing bayesian inference by weighted model counting. In: Prof. of AAAI. pp. 475–481 (2005)
28. Schmidt, P., Witte, A.D.: Predicting recidivism in north carolina, 1978 and 1980. Inter-University Consortium for Political and Social Research (1988), https://www.ncjrs.gov/App/Publications/abstract.aspx?ID=115306
29. Shih, A., Choi, A., Darwiche, A.: A symbolic approach to explaining bayesian network classifiers. In: IJCAI. pp. 5103–5111 (2018)
30. Simonyan, K., Vedaldi, A., Zisserman, A.: Deep inside convolutional networks: Visualising image classification models and saliency maps. CoRR **abs/1312.6034** (2013), http://arxiv.org/abs/1312.6034
31. Sinz, C.: Towards an optimal CNF encoding of Boolean cardinality constraints. In: CP. pp. 827–831 (2005)
32. Soos, M., Meel, K.S.: Bird: Engineering an efficient cnf-xor sat solver and its applications to approximate model counting. In: Proceedings of AAAI Conference on Artificial Intelligence (AAAI) (1 2019)
33. Soos, M., Nohl, K., Castelluccia, C.: Extending SAT solvers to cryptographic problems. In: Theory and Applications of Satisfiability Testing - SAT 2009, 12th International Conference, SAT 2009, Swansea, UK, June 30 - July 3, 2009. Proceedings. pp. 244–257 (2009)
34. Thurley, M.: SharpSAT: counting models with advanced component caching and implicit BCP. In: Proc. of SAT. pp. 424–429 (2006)
35. Toda, S.: Pp is as hard as the polynomial-time hierarchy. SIAM Journal on Computing **20**(5), 865–877 (1991)
36. Valiant, L.: The complexity of enumeration and reliability problems. SIAM Journal on Computing **8**(3), 410–421 (1979)
37. Wu, M., Hughes, M.C., Parbhoo, S., Zazzi, M., Roth, V., Doshi-Velez, F.: Beyond sparsity: Tree regularization of deep models for interpretability. In: AAAI. pp. 1670–1678 (2018)
38. Zhang, Q., Yang, Y., Wu, Y.N., Zhu, S.: Interpreting CNNs via decision trees. CoRR **abs/1802.00121** (2018), http://arxiv.org/abs/1802.00121