Proceedings of Machine Learning Research vol 72, 427-438, 2018 PGM 2018

Formal Verification of Bayesian Network Classifiers

Andy Shih ANDYSHIH@CS.UCLA.EDU
Arthur Choi AYCHOI@CS.UCLA.EDU
Adnan Darwiche DARWICHE@CS.UCLA.EDU
Computer Science Department, University of California, Los Angeles

Abstract

A new approach was recently proposed for *explaining* the decisions made by Bayesian network classifiers. This approach is based on first compiling a given classifier (i.e., its decision function) into a tractable representation called an Ordered Decision Diagram (ODD). Given an ODD representation of the decision function, we get the ability to provide reasons for why a classifier labels a given instance positively or negatively. We show in this paper that this approach also gives us the ability to *verify* the behavior of classifiers. We also provide case studies in explaining and verifying classifiers for some real-world domains, such as in medical diagnosis and educational assessment.

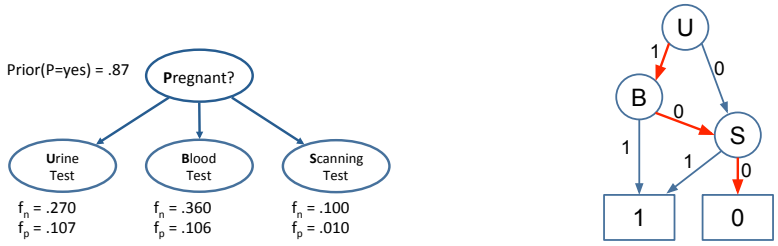
1. Introduction

Recent progress in artificial intelligence and the increased deployment of AI systems have led to highlighting the need for *explaining* the decisions made by such systems, particularly classifiers; see, e.g., (Ribeiro et al., 2016; Elenberg et al., 2017; Lundberg and Lee, 2017; Ribeiro et al., 2018).[1] For example, one may want to explain *why* a classifier decided to turn down a loan application, or rejected an applicant for an academic program, or recommended surgery for a patient. Answering such *why?* questions is particularly central to assigning blame and responsibility, which lies at the heart of legal systems and may be required in certain contexts.[2]

To address such challenges, we recently proposed an approach for explaining the decisions of Bayesian network classifiers (Shih et al., 2018a). In this paper, we further consider the *formal verification* of such classifiers; that is, mathematically analyze and validate the behavior of a given classifier. For example, we may want to verify that the classifier's behavior is consistent with the knowledge of a domain expert (a doctor may assert that if tests X and Y are observed positively, then the classifier should make a positive diagnosis, independent of the results of any other test). We may also want to verify whether a classifier respects certain properties, such as *monotonicity* (a student whose correct answers include those of another cannot be assessed less positively). Moreover, whenever we conclude that a classifier is *not* behaving as intended, we want to extract a simple example demonstrating that behavior (enabling the interactive design of a classifier, via debugging).

We pursue a *symbolic* approach to explaining and verifying Bayesian network classifiers based on the following insight. Consider a classifier that classifies instances as 1 or 0 based on its observed, discrete features. Any such classifier (Bayesian network or otherwise) specifies a symbolic function that maps features into a binary decision (1 or 0) (Chan and Darwiche, 2003). This is called the

1. It is now recognized that opacity, or lack of explainability is "one of the biggest obstacles to widespread adoption of artificial intelligence" (The Wall Street Journal, August 10, 2017).
2. See, for example, the EU general data protection regulation, which has a provision relating to explainability, `https://www.privacy-regulation.eu/en/r71.htm`.



(a) A naive Bayes classifier (b) An OBDD

Figure 1: A naive Bayes classifier and its corresponding decision function as an OBDD. The classifier is specified using the class prior (87%), threshold (90%), in addition to the false positive (f_p) and false negative (f_n) rates of features. The class variable and features are all binary. In Figure 1(b), an edge labeled 1/0 corresponds to a positive/negative test result and a sink labeled 1/0 corresponds to a yes/no prediction.

classifier's *decision function* and unambiguously characterizes its behavior. The approach is to represent this decision function using a symbolic and tractable representation, which is then used to symbolically reason about the classifier's behavior. Recent results showed how to compile the decision function of a Bayesian network classifier into an Ordered Decision Diagram (ODD) and use the ODD to explain the decisions made by the classifier (Shih et al., 2018a).[3] We pursue this approach further by showing how it can be used to *formally verify* the classifier's behavior as well. We illustrate our approach through case studies from real-world Bayesian networks developed in the domains of educational assessment and medical diagnosis.

Our approach follows a recent trend in analyzing machine learning models using symbolic approaches; see, e.g., (Katz et al., 2017; Leofante et al., 2018; Narodytska et al., 2018; Shih et al., 2018a). While machine learning and statistical methods are key for learning classifiers, it is evident that symbolic and logical approaches, which are independent of any of the models parameters, are key for analyzing and reasoning about them, as we shall demonstrate later.

This paper is organized as follows. In Section 2, we cover technical preliminaries and discuss recent advances in compiling Bayesian network classifiers into ODDs. In Section 3, we highlight how to formally verify classifiers when their decision functions are represented as ODDs. In Section 4, we provide two case studies, illustrating the utility of ODDs for explaining and verifying classifiers. We finally conclude in Section 5.

2. Compiling Bayesian Network Classifiers

Consider Figure 1(a) which depicts a naive Bayes classifier for detecting pregnancy. Given results for the three tests, if the probability of pregnancy passes the given threshold (90%), we would then obtain a "yes" decision on pregnancy.

Figure 1(b) depicts the decision function of this classifier, in the form of an Ordered Binary Decision Diagram (OBDD). Given some test results, we make a corresponding decision on pregnancy

3. ODDs extends Ordered Binary Decision Diagrams (OBDDs) to use multi-valued variables (discrete features), while maintaining the tractability and properties of OBDDs (Bryant, 1986; Meinel and Theobald, 1998; Wegener, 2000).

by simply navigating the OBDD. We start at the root, which is labeled with the Urine (U) test. Depending on the outcome of this test, we follow the edge labeled 1 (positive), or the edge labeled 0 (negative). We repeat for the test labeled at the next node. Eventually, we reach a leaf node labeled "1" or "0," which provides the resulting classification.

The decisions rendered by this OBDD are guaranteed to match those obtained from the naive Bayes classifier. We have thus converted a probabilistic classifier into an equivalent classifier that is symbolic and tractable. This technique was employed recently to explain the decisions made by Bayesian network classifiers (Shih et al., 2018a). We show in this paper that it can also be used to formally verify the classifier's behavior, while making additional contributions to explanation. We need to first settle some technical preliminaries in the rest of this section.

For the rest of the paper, upper case letters (e.g., X) will be used to denote variables. For a Boolean variable X, we use x to denote its positive literal and $\bar{x}$ to denote its negative literal. Bold upper case letters (e.g., $\mathbf{X}$) will be used to denote sets of variables and bold lower case letters to denote their instantiations (e.g., $\mathbf{x}$).

2.1 Bayesian Network Classifiers

A *Bayesian network classifier* is a Bayesian network containing a special set of variables: a single *class* variable C and n *feature* variables $\mathbf{X} = \{X_1, \ldots, X_n\}$. The class C is usually a root in the network and the features $\mathbf{X}$ are usually leaves. In this paper, we assume that the class variable is binary, with two values c and $\bar{c}$ that correspond to positive and negative classes, respectively (i.e., "1" and "0" decisions). An instantiation of variables $\mathbf{X}$ is denoted $\mathbf{x}$ and called an *instance.* A Bayesian network classifier specifying probability distribution $Pr(.)$ will classify an instance $\mathbf{x}$ positively iff $Pr(c \mid \mathbf{x}) \geq T$, where T is called the classification *threshold.*

Definition 1 (Decision Function) *Suppose that we have a Bayesian network classifier with features $\mathbf{X}$, class variable C and a threshold T. Let $f(\mathbf{X})$ be a function that maps instances $\mathbf{x}$ into $\{0, 1\}$. We say that $f(\mathbf{X})$ is the classifier's decision function iff*

$$f(\mathbf{x}) = \begin{cases} 1 & \text{if } Pr(c \mid \mathbf{x}) \geq T \\ 0 & \text{otherwise.} \end{cases}$$

Instance $\mathbf{x}$ is positive if $f(\mathbf{x}) = 1$ and negative if $f(\mathbf{x}) = 0$.

The *naive Bayes classifier* is a special type of a Bayesian network classifier, where edges extend from the class to features (no other nodes or edges). Figure 1(a) depicts a naive Bayes classifier.

2.2 Ordered Decision Diagrams

An Ordered Binary Decision Diagram (OBDD) is a rooted, directed acyclic graph with two sinks called the 1-sink and 0-sink. An OBDD is a graphical representation of a Boolean function on variables $\mathbf{X} = X_1, \ldots, X_n$. Every node (except the sinks) in the OBDD is labeled with a variable X_i and has two labeled outgoing edges: the 1-edge and the 0-edge. The labeling of the OBDD nodes respects some global ordering of the variables $\mathbf{X}$: if there is an edge from a node labeled X_i to a node labeled X_j, then X_i must come before X_j in the global ordering. To evaluate the OBDD on an instance $\mathbf{x}$, start at the root node of the OBDD and let x_i be the label of the current node.

Repeatedly follow the x_i-edge of the current node, until a sink node is reached. Reaching the 1-sink means $\mathbf{x}$ is evaluated to 1 and reaching the 0-sink means $\mathbf{x}$ is evaluated to 0 by the OBDD.

An OBDD is defined over binary variables, but can be extended to discrete variables with arbitrary values. This is called an ODD: a node labeled with variable X_i has one outgoing edge for each value of variable X_i. Hence, an OBDD/ODD can be viewed as representing a function $f(\mathbf{X})$ that maps instances $\mathbf{x}$ into $\{0, 1\}$. Figure 1(b) depicted an OBDD. Note: in this paper, we use positive/1 and negative/0 interchangeably.

An OBDD is a *tractable* representation of a function $f(\mathbf{X})$ as it can be used to efficiently answer many queries about the function. For example, one can in linear time count the number of positive instances $\mathbf{x}$ (i.e., $f(\mathbf{x}) = 1$), called the *models* of f. One can also conjoin, disjoin and complement OBDDs efficiently. This tractability, which carries over to ODDs, was critical for efficiently generating explanations (Shih et al., 2018a). It will also be critical to our formal verification approach. For more on OBDDs, see Meinel and Theobald (1998); Wegener (2000).

2.3 Compiling Decision Functions

Chan and Darwiche (2003) proposed an algorithm for compiling a naive Bayes classifier into an ODD, while guaranteeing an upper bound on the time of compilation and the size of the resulting ODD. In particular, for a classifier with n features, the compiled ODD has a number of nodes that is bounded by $O(b^{\frac{n}{2}})$ and can be obtained in time $O(nb^{\frac{n}{2}})$. Here, b is the maximum number of values that a variable may have. The actual time and space complexity can be much less, depending on the classifier's parameters and variable order used for the ODD (as observed experimentally). In general, we do not expect a significantly better upper bound on the time complexity as the compilation process into ODDs is NP-hard (Shih et al., 2018a). More recently, algorithms were developed that can compile the decision function of a general Bayesian network classifier into an ODD (Shih et al., 2018a,b). We shall employ these algorithms in our case studies.

3. Verifying and Explaining Classifiers Using ODDs

Consider a decision function f over variables $\mathbf{X}$ and let $\mathbf{Y} \subseteq \mathbf{X}$. The *conditioning* of f on instantiation $\mathbf{y}$, written $f|\mathbf{y}$, is a *sub-function* over variables $\mathbf{X} \setminus \mathbf{Y}$ that results from setting variables $\mathbf{Y}$ to their values in $\mathbf{y}$. Furthermore, decision function f *entails* another decision function g, denoted by $f \models g$, iff every positive instance of f is also a positive instance of g (i.e., $f(\mathbf{x}) = 1$ implies $g(\mathbf{x}) = 1$). Lastly, we write $\bar{f}$ to denote the negation of f.

We will next propose a number of formal verification queries and show how to compute them efficiently. We will also contribute an additional explanation query, beyond (Shih et al., 2018a), and show how to compute it efficiently as well.

3.1 Verifying Monotonicity

We will first consider *monotonicity* queries while assuming binary features to simplify the treatment. Intuitively, a monotone classifier satisfies the following. A positive instance remains positive if we flip some of its features from 0 to 1. Moreover, a negative instance remains negative if we flip some of its features from 1 to 0. In certain domains, one expects or desires a Bayesian network classifier learned from data to be monotone (van der Gaag et al., 2004). For example, in the context

of educational assessment, we expect a student to be assessed positively if their correct answers are a superset of those of another student who has been assessed positively.

More formally, consider two instances $\mathbf{x}^\star$ and $\mathbf{x}$. We write $\mathbf{x}^\star \subseteq^1 \mathbf{x}$ to mean: the features set to 1 in $\mathbf{x}^\star$ is a subset of those set to 1 in $\mathbf{x}$. Monotone classifiers are then characterized by the following property of their decision functions, which is well-known in the literature on Boolean functions.

Definition 2 *A decision function $f(\mathbf{X})$ is monotone iff*

$$\mathbf{x}^\star \subseteq^1 \mathbf{x} \quad \textit{only if} \quad f(\mathbf{x}^\star) \leq f(\mathbf{x}).$$

One way to read the above formal definition is as follows. If the positive features in instance $\mathbf{x}$ contain those in instance $\mathbf{x}^\star$, then instance $\mathbf{x}$ must be positive if instance $\mathbf{x}^\star$ is positive.

In general, it is difficult to decide whether a Bayesian network classifier is monotone (van der Gaag et al., 2004). However, if the decision function of the classifier is represented as an ODD, then monotonicity can be decided in time quadratic in the ODD size (Horiyama and Ibaraki, 2002). A simpler but less efficient approach is based on the following observation: a decision function f is monotone iff $f|\bar{x} \models f|x$ for all variables X. Given an ODD f, we can perform conditioning ($f|\bar{x}$, $f|x$) and test entailment ($f|\bar{x} \models f|x$) in time polynomial in the size of input ODDs.

Finally, we consider unateness, a mild generalization of monotonicity, which can also be verified efficiently given a decision function represented by an ODD.

Definition 3 *A decision function $f(\mathbf{X})$ is unate iff for all X*

$$f|\bar{x} \models f|x \quad \textit{or} \quad f|x \models f|\bar{x}.$$

Any monotone decision function is also unate. Intuitively, in a unate function, for each variable X there exists a polarity x or $\bar{x}$ where flipping variable X to that polarity will cause a positive instance to remain positive. In other words, it behaves like a monotone function when we interpret the appropriate polarity of each variable as if it were positive. As with monotone functions, we can efficiently test if a given decision function is unate given an ODD representation of that function. Moreover, if a decision function is monotone or unate, then certain explanations become more efficient to compute, such as prime-implicant explanations, as shown by Shih et al. (2018a).

3.2 Finding Irrelevant Features

Some features may be more pertinent to a class variable than others. At the extreme, there may exist a subset of features that render the remaining features irrelevant. This particularly comes up with Bayesian networks that have multiple class variables, as only a subset of features may turn out to be relevant to a particular class variable. In this case, we would like to detect and then drop these irrelevant features, to obtain a simpler and more computationally efficient classifier.

Definition 4 *A decision function $f(\mathbf{X})$ essentially depends on variable X iff $f|x \neq f|\bar{x}$.*

If the decision function does not essentially depend on variable X, we say that variable X is irrelevant to the decision. In an ODD, conditioning takes time that is linear in the size of the ODD. Moreover, equivalence testing can be done in constant time (assuming an appropriate implementation). Hence, determining if a variable is irrelevant can be done efficiently.[4]

4. In fact, if an ODD is reduced (all redundant nodes are removed/merged), then it suffices to scan the ODD to see if any ODD node is labeled by variable X.

3.3 Verifying Classification Robustness

Consider a classifier and one of its positive instances $\mathbf{x}$. In certain domains, we would like to know whether such a classification is robust to perturbations in the input; see, e.g., Narodytska et al. (2018) which uses SAT solvers to test the robustness of neural networks. As an example, in the domain of educational assessment, we may want to be aware whether a passing student was only a few test questions away from failing a test, or whether they would have still passed it, even if they had gotten many more questions wrong. As another example, in the context of image classification and self-driving cars, we may want to be aware if an image is only a few pixels away from being classified as containing a stop sign versus not.

Given the decision function f of a classifier, we define the robustness of an instance $\mathbf{x}$ as the minimum number of features that we need to flip (from positive to negative, or negative to positive), before we flip the instance $\mathbf{x}$ from positive to negative (or negative to positive).

Definition 5 *Given a non-trivial*[5] *decision function f and an instance $\mathbf{x}$, we define the robustness of the classification of $\mathbf{x}$ as:*

$$\mathsf{robustness}_f(\mathbf{x}) = \min_{\mathbf{x}': f(\mathbf{x}') \neq f(\mathbf{x})} d(\mathbf{x}', \mathbf{x})$$

where $d(\mathbf{x}', \mathbf{x})$ denotes the Hamming distance between $\mathbf{x}'$ and $\mathbf{x}$, i.e., the number of variables on which $\mathbf{x}$ and $\mathbf{x}'$ differ.

The following theorem implies an efficient algorithm for computing robustness.

Theorem 6 *Consider a decision function $f(Y, \mathbf{X})$. The robustness of a positive instance $y, \mathbf{x}$ satisfies:*

$$\mathsf{robustness}_f(y, \mathbf{x}) = \min\{\mathsf{robustness}_{f|y}(\mathbf{x}), 1 + \mathsf{robustness}_{f|\bar{y}}(\mathbf{x})\}$$

where $\mathsf{robustness}_f(\mathbf{x}) = 0$ if $f = \bot$ (i.e., false) and $\mathsf{robustness}_f(\mathbf{x}) = \infty$ if $f = \top$ (i.e., true).

Due to this theorem, it takes time linear in the size of an ODD to compute the robustness of a given instance (by caching intermediate results, each node of an ODD is evaluated at most once). Robustness can be viewed as an explanation query as it pertains to a particular instance and decision, in contrast to validating an instance-independent property of the classifier.

3.4 Verifying If-Then Rules and Decision Independence

Our next verification queries require the definitions of *partial instance* and an *implicant.*

Definition 7 *Given a decision function $f(\mathbf{X})$, let $\mathbf{Z} \subseteq \mathbf{X}$ and $\mathbf{Y} = \mathbf{X} \setminus \mathbf{Z}$. An instantiation $\mathbf{z}$ will be called a partial instance of function f. Moreover, the partial instance $\mathbf{z}$ is an implicant of function f iff $f(\mathbf{zy}) = 1$ for all instantiations $\mathbf{y}$.*

Consider a medical domain where we have learned a classifier from patient data, with a corresponding decision function f. One way of verifying the behavior of such a classifier is to compare it with the "if-then" rules of a domain expert, such as a doctor. For example, a doctor may pre-define

5. A trivial decision function maps all instances to 1, denoted $\top$, or maps all instances to 0, denoted $\bot$.

network	class	# features	width	ODD size	compilation time (s)
adaptive-testing	HV1	20	8	1,164	40
adaptive-testing	HV2	20	8	1,924	31
adaptive-testing	MT	20	8	1,437	21
mammography	RegOutcome	15	4	156	7
math-skills	S6	46	4	3,693,629	61,088

Table 1: Compilations of Bayesian network classifiers.

certain types of behaviors that we would expect a reliable classifier would respect: "if tests A, B, and C are positive, then the patient should always be diagnosed positively." Such a constraint can be verified by testing whether the corresponding partial instance $\mathbf{z} = a, b, c$ is an implicant of the decision function f. By verifying that a classifier and a domain expert agree on many such constraints, one can improve the trust that one has in a classifier learned from data.

This query can be computed efficiently since a partial instance $\mathbf{z}$ is an implicant of f iff $f|\mathbf{z} = \top$. Given the set of tractable operations on ODDs, we can test whether $\mathbf{z}$ is an implicant of ODD f in time polynomial in the size of the ODD: we simply set each variable in $\mathbf{z}$ to its specified value, and test whether the resulting ODD is true.

Consider now a partial instance $\mathbf{z}$ which is either an implicant of f or an implicant of $\bar{f}$. In such a partial instance, no setting of the remaining features will change the decision of the classifier (regardless of what the decision is). That is, the *decision* is now independent of the remaining features, given the partial instance $\mathbf{z}$ so no further observations are necessary. This *decision* independence is weaker than *probabilistic* independence—additional observations may change the posterior of a class, but the decision may not, for a given threshold.

In some domains, we may also be interested in the *shortest* partial instances that yield decision independence. This is related to computing the *shortest* prime implicants (Marques-Silva, 1997; Coudert and Madre, 1993; Coudert et al., 1993; Minato, 1993; Shih et al., 2018a). Consider the domain of adaptive testing, where we are trying to assess the knowledge of a student who is taking a test. Typically, we select the questions to ask the student in an adaptive way, to minimize the number of questions asked while also minimizing our uncertainty about the student's knowledge. When this adaptive process is guided by a classifier, the above query will tell us the minimum number of test questions that, if answered in a certain way, will render the assessment independent of the remaining questions.

4. Explanation and Verification: Case Studies

In this section, we illustrate our ability to explain and verify a Bayesian network classifier using real-world networks from two domains: educational assessment and medical diagnosis. We also provide evidence on the scalability of techniques utilized, particularly the compilation into ODDs. We first consider a network used for adaptive testing in (Vomlel, 2002a,b). Here, each input instance $\mathbf{x}$ represents a student whose math skills we want to assess. Next, we consider a network used for diagnosing breast cancer (Gimenez et al., 2014). Here, each input instance represents the mammography report of a female patient that we want to diagnose.

We compile these networks into ODDs, using the compilation algorithm reported in Shih et al. (2018b), which generalizes the algorithm of Shih et al. (2018a) for compiling latent tree Bayesian

networks to more general Bayesian network classifiers. Our experiments were performed on a system using a single Intel(R) Xeon(R) CPU E5-2670 processor with access to 200 GB of memory.

Table 1 provides some statistics on the Bayesian network classifiers that we considered. For a given network, we picked class variables from the root variables, and considered every leaf variable as a feature. We evaluate network `adaptive-testing` in our case study on educational assessment, and we evaluate network `mammography` in our case study on medical diagnosis. We also report the compilation results of a much larger network for educational assessment, `math-skills` (Plajner and Vomlel, 2015). We use the reported classification threshold of 0.02 for the `mammography` network (Gimenez et al., 2014) and a classification threshold of $\frac{1}{2}$ for the `adaptive-testing` and `math-skills` networks, which had no reported classification threshold. In Table 1, column class reports the class node, column # features reports the number of features, column width reports the network's (tree)-width, as approximated using the minfill heuristic.

Columns ODD size and compilation time report the size of the compiled ODD and the time taken to compile it. The variable ordering is decided by the ordering heuristic provided in the compilation algorithm of Shih et al. (2018b), which roughly tries to minimize the size of the resulting ODD. We observe that networks `adaptive-testing` and `mammography` yield relatively succinct ODD representations for the number of features considered; in contrast, a complete decision tree would have a corresponding size of $O(2^n)$ for n binary features. We also successfully compiled the network `math-skills`, which has significantly more features and is outside the scope of brute-force enumeration.

4.1 Educational Assessment: Adaptive Testing

We consider first an educational assessment network that evaluates a student's ability to solve mathematical problems involving fractions (Vomlel, 2002a,b). This network was used for adaptive testing, where a student is dynamically evaluated, and test questions are selected adaptively for each student. This network has three root nodes: node HV1 judges addition and subtraction operations, node HV2 judges advanced addition and subtraction problems, and node MT judges multiplication of fractions. In total, there are 21 nodes representing skills and misconceptions. In addition, 20 leaf nodes (features) represent student responses to test questions, whose outcomes depend on different skills and misconceptions.

We first examine the classifier using root node HV2 as the class variable, using a threshold of $\frac{1}{2}$. We start by verifying whether the network is monotonic or not. Each observed feature represents the student's answer to a given math problem (true/1 or false/0); hence, we would expect the network to be unate. That is, providing the correct answer (which may be 1 or 0) should not flip a positive assessment to a negative one. Using the test that we described in Section 3.1, we find that the network is not unate. We can pinpoint a specific counterexample that demonstrates the non-monotonicity of the classifier.[6]

Consider the following ordering π of features (which was used to compile the ODD), and the corresponding four instances α, α', β and β'.

$$\pi = [X_{11}, X_{10}, X_{13}, X_{16}, X_{15}, X_{17}, X_{14}, X_9, X_8, X_{18}, X_{19}, X_{20}, X_{12}, X_4, X_3, X_2, X_1, X_5, X_7, X_6]$$

$$\alpha = [0,0,0,\mathbf{0},0,0,0,1,1,1,1,1,0,0,1,0,0,1,1,1] \quad \beta = [0,0,0,\mathbf{0},1,1,0,1,1,0,1,1,1,0,0,0,0,1,1,1]$$

$$\alpha' = [0,0,0,\mathbf{1},0,0,0,1,1,1,1,1,0,0,1,0,0,1,1,1] \quad \beta' = [0,0,0,\mathbf{1},1,1,0,1,1,0,1,1,1,0,0,0,0,1,1,1]$$

6. First, we find a variable X where $f|\bar{x} \models f|x$ does not hold. We can then enumerate models of $f|\bar{x} \cdot \bar{f}|x$ which corresponds to positive instances where flipping X from negative to positive causes the instance to flip to negative.

In this case, a positive instance α becomes a negative instance α' by flipping feature X_{16} from negative (0) to positive (1); hence the classifier is not monotone. A negative instance β becomes a positive instance β' by flipping the same feature X_{16} from negative (0) to positive (1); hence the classifier is also not unate.[7] This suggests that the following scenario is possible. A student completes the math test, and while getting some questions wrong, they are predicted by the network to have a satisfactory level of mathematical aptitude. The student then corrects one of the incorrect responses, but is now predicted to have an unsatisfactory level of mathematical aptitude. If monotonicity or unateness is indeed a desired property of the classifier, then we can try to "debug" the classifier, starting with the example instances that we identified above as witnesses to the classifier's non-monotonicity.

Next, we test for irrelevant variables, as described in Section 3.2. By inspecting the compiled OBDD, we find that none of the features of the classifier are irrelevant. Hence, each question on the math test can be considered essential to every class variable: for each question, there exists a partial instance whose predicted label can be flipped by flipping the response of that question.

We next consider the robustness, as described in Section 3.3, of the students κ and γ with the following feature vectors:

$$\kappa = [0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1]$$
$$\gamma = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]$$

Student κ answered 11 questions as true (1), and 9 questions as false (0), and is assessed by the classifier positively. This particular instance has a robustness of 1: the student is one response away from being assessed negatively. In particular, if this student changes their response to question X_9, the classifier's prediction would flip. Student γ answered all 20 questions as false (0), and is assessed by the classifier negatively. This instance has a high robustness value of 5—the student cannot change the classifier's decision even by correcting up to 4 responses on the math test.

Finally, we consider the implicants of a classifier's decision function, as described in Section 3.4. In particular, the following partial instance $\kappa^\star$ is a minimal implicant of the decision function that is compatible with student κ we considered earlier:

$$\kappa^\star = [-, -, -, 0, 0, 0, -, 1, -, 1, 1, 1, 0, -, 1, -, 1, -, -, -]$$

where "–" denotes an irrelevant value. If we consider vector κ to be the answers that a student would provide to all test questions, then vector $\kappa^\star$ is a minimal subset of questions that we could ask, that would render the decision independent of the remaining features. Hence, there is no need to ask further test questions, as our assessment would not change. Further, there is no smaller selection of test questions for this student, that would allow us to stop asking questions earlier, since this implicant is minimal (also called a "prime implicant").

4.2 Medical Diagnosis: Breast Cancer

We next consider a Bayesian network classifier for detecting whether a female patient has breast cancer based on her mammography report (Gimenez et al., 2014). The classifier has 12 features and one root node RegOutcome representing the diagnosis of breast cancer, which has two states: Benign and Malignant. For simplicity, we represent each multi-valued feature with k states using $\lceil \log_2(k) \rceil$ binary variables. This transformation results in a classifier with 15 binary features. The

7. The classifier is not unate when using a threshold of $\frac{1}{2}$. Using a different threshold may result in a unate or even monotonic classifier.

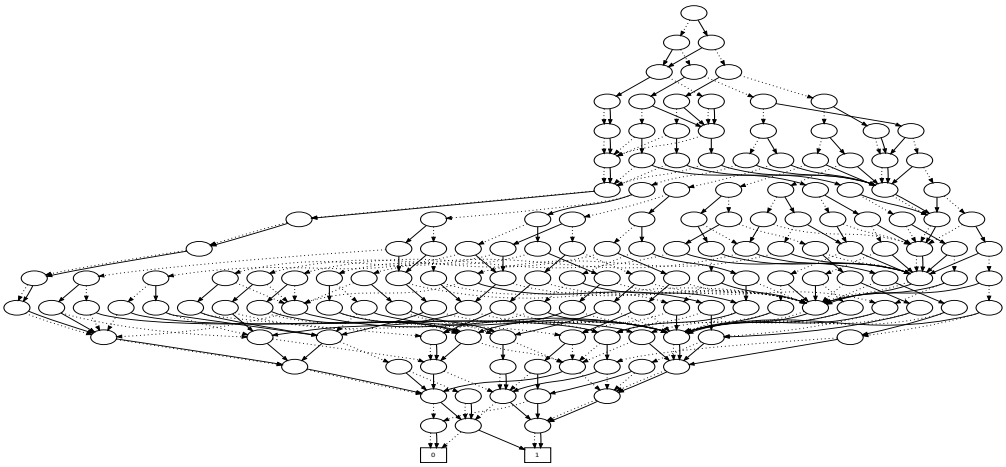


Figure 2: Mammography OBDD. Nodes at each level of the OBDD are labeled with the corresponding variable in the used variable ordering (i.e., nodes at the first level are labeled with AsymmetricDensity and nodes at the last level are labeled with FamilyHistory1).

threshold of this classifier is 0.02, which is the threshold for suspicious abnormality in the BI-RADS assessment scale (Gimenez et al., 2014). We used the following variable ordering to compile the classifier into an ODD, which is illustrated in Figure 2.

$\pi =$[AsymmetricDensity, MassSize, LymphNode, AxillaryAdenopathy, SkinLesion, NippleRetraction, MassShape1, MassShape0, PersonalHistory, TrabecularThickening, SkinRetraction, BreastDensity1, BreastDensity0, FamilyHistory0, FamilyHistory1]

Note that this ODD is relatively small (156 nodes) compared to the number of features (15); in contrast, the corresponding decision tree would have $O(2^{15})$ nodes.

Next, we verify whether the classifier is monotonic or not, as described in Section 3.1. Consider the following instances (corresponding to patients):

$$\alpha = [0,0,0,0,0,0,0,0,\mathbf{0},1,1,0,0,1,0] \quad \beta = [0,0,0,0,0,0,0,0,\mathbf{0},0,0,1,0,0,0]$$
$$\alpha' = [0,0,0,0,0,0,0,0,\mathbf{1},1,1,0,0,1,0] \quad \beta' = [0,0,0,0,0,0,0,0,\mathbf{1},0,0,1,0,0,0]$$

In this case, a positive instance α becomes a negative instance α' by flipping feature PersonalHistory from negative (0) to positive (1); hence the classifier is not monotone. A negative instance β becomes a positive instance β' by flipping the same feature PersonalHistory from negative (0) to positive (1); hence the classifier is also not unate. This classifier suggests that it is possible for two patients to share the same mammography report except for their personal history, and the patient with no past history of breast cancer will be predicted as having malignant findings compared to the patient with past history of breast cancer. Even though the network is not monotonic, there are variables that individually behave monotonically. Consider the variable MassSize, for example: flipping it from large to small will never flip the classification decision from Benign to Malignant.

Next, we test for the presence of irrelevant variables, as in Section 3.2, of which we found none. This can be confirmed by inspection of Figure 2, where we can find every variable appearing in a

(non-vacuous) node in the OBDD. Hence, we cannot prune any feature appearing as a field in the mammography report, as some partial instance will depend on that feature.

As in Section 3.3, we now examine the robustness of the following instance κ.

$$\kappa = [0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]$$

This instance represents a mammography report with all features observed as 0 except for MassSize and LymphNode (observed as 1). The classifier predicts this instance as Benign. Furthermore, the robustness of this instance is 2: we must flip at least two features in this instance in order to have it be classified as Malignant. The following instance κ' is one such example of a Malignant instance that is two flips away from κ.

$$\kappa' = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]$$

By analyzing the original network, we find that observing MassSize and LymphNode to be 1 is very suggestive of a Benign prediction. Finally, consider the following partial instance:

$$\kappa^\star = [-, 0, 0, -, -, -, 0, 0, -, 0, -, 0, 0, -, -]$$

which is compatible with patient κ' that is classified as Malignant. This $\kappa^\star$ is an implicant of the classifier's decision function, as discussed in Section 3.4. This implicant sets 7 of the features to 0. Interestingly, MassSize and LymphNode are two of the variables in the implicant: observing them and five other variables to 0 is strong enough to guarantee a classification of Malignant, independent of the remaining features. This is consistent with the robustness query in that the variables MassSize and LymphNode are strong predictors of the classification decision.

5. Conclusion

We explored the formal verification and explanation of Bayesian network classifiers. Using recently developed algorithms for compiling Bayesian network classifiers into Ordered Decision Diagrams (ODDs), we obtain an unambiguous and tractable representation of a classifier's decision function, which we exploit to verify and explain the behavior of the classifier. In particular, we proposed a few verification queries and how to compute them in polynomial time. We also proposed a new explanation query relating to decision robustness and provided a corresponding polynomial-time algorithm as well. We demonstrated the utility of these methods by generating insights into real-world networks from the domains of educational assessment and medical diagnosis.

Acknowledgments

This work has been partially supported by NSF grant #IIS-1514253, ONR grant #N00014-18-1-2561 and DARPA XAI grant #N66001-17-2-4032.

References

R. E. Bryant. Graph-based algorithms for Boolean function manipulation. *IEEE Transactions on Computers*, C-35:677–691, 1986.

H. Chan and A. Darwiche. Reasoning about Bayesian network classifiers. In *UAI*, pages 107–115, 2003.

O. Coudert and J. C. Madre. Fault tree analysis: 10^{20} prime implicants and beyond. In *Proc. of the Annual Reliability and Maintainability Symposium*, pages 240–245, 1993.

O. Coudert, J. C. Madre, H. Fraisse, and H. Touati. Implicit prime cover computation: An overview. In *Proceedings of the 4th SASIMI Workshop*, 1993.

E. R. Elenberg, A. G. Dimakis, M. Feldman, and A. Karbasi. Streaming weak submodularity: Interpreting neural networks on the fly. In *NIPS*, pages 4047–4057, 2017.

F. J. Gimenez, Y. Wu, E. Burnside, and D. L. Rubin. A Novel Method to Assess Incompleteness of Mammography Reports. In *AMIA*, pages 1758–1767, 2014.

T. Horiyama and T. Ibaraki. Ordered binary decision diagrams as knowledge-bases. *Artificial Intelligence (AIJ)*, 136(2):189–213, 2002.

G. Katz, C. W. Barrett, D. L. Dill, K. Julian, and M. J. Kochenderfer. Reluplex: An efficient SMT solver for verifying deep neural networks. In *Computer Aided Verification CAV*, pages 97–117, 2017.

F. Leofante, N. Narodytska, L. Pulina, and A. Tacchella. Automated verification of neural networks: Advances, challenges and perspectives. *CoRR*, abs/1805.09938, 2018. URL `http://arxiv.org/abs/1805.09938`.

S. M. Lundberg and S. Lee. A unified approach to interpreting model predictions. In *NIPS*, pages 4768–4777, 2017.

J. Marques-Silva. On computing minimum size prime implicants. In *International Workshop on Logic Synthesis*, 1997.

C. Meinel and T. Theobald. *Algorithms and Data Structures in VLSI Design: OBDD — Foundations and Applications*. Springer, 1998.

S. Minato. Fast generation of prime-irredundant covers from binary decision diagrams. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, 76(6):967–973, 1993.

N. Narodytska, S. P. Kasiviswanathan, L. Ryzhyk, M. Sagiv, and T. Walsh. Verifying properties of binarized deep neural networks. In *AAAI*, 2018.

M. Plajner and J. Vomlel. Bayesian network models for adaptive testing. In *Proceedings of the Twelfth UAI Bayesian Modeling Applications Workshop*, pages 24–33, 2015.

M. T. Ribeiro, S. Singh, and C. Guestrin. "Why should I trust you?": Explaining the predictions of any classifier. In *KDD*, pages 1135–1144, 2016.

M. T. Ribeiro, S. Singh, and C. Guestrin. Anchors: High-precision model-agnostic explanations. In *AAAI*, pages 1527–1535, 2018.

A. Shih, A. Choi, and A. Darwiche. A symbolic approach to explaining Bayesian network classifiers. In *IJCAI*, 2018a.

A. Shih, A. Choi, and A. Darwiche. Compiling Bayesian network classifiers into decision graphs, 2018b. Under review.

L. C. van der Gaag, H. L. Bodlaender, and A. J. Feelders. Monotonicity in Bayesian networks. In *UAI*, pages 569–576, 2004.

J. Vomlel. Exploiting functional dependence in Bayesian network inference. In *UAI*, pages 528–535, 2002a.

J. Vomlel. Bayesian networks in educational testing. In *PGM*, 2002b.

I. Wegener. *Branching Programs and Binary Decision Diagrams*. SIAM, 2000.