Programs as Black-Box Explanations

Sameer Singh
Department of Computer Science
University of California, Irvine CA
sameer@uci.edu

Marco Tulio Ribeiro **Carlos Guestrin**
Department of Computer Science
University of Washington, Seattle WA
{marcotcr, guestrin}@uw.edu

With increasing complexity of machine learning systems being used[1], there is a crucial need for providing insights into what these models are doing. *Model-agnostic* approaches [18], such as Baehrens et al. [1] and Ribeiro et al. [17], have shown that insights into complex, black-box models do not have to come at a cost of accuracy, and that accurate *local* explanations can successfully be provided for a number of complex classifiers (such as random forests and deep neural networks) and domains (text and images) for which interpretable models have not performed competitively. However, we still need to identify which interpretable representation would be suitable to convey the local behavior of the model in an accurate and succinct manner, and existing model-agnostic approaches have focused only on (sparse) linear models. Work in interpretable machine learning, on the other hand, has proposed many more other representations when designing their models, ranging from additive models, to decision rules, trees, sets, and lists, amongst others [8, 10].

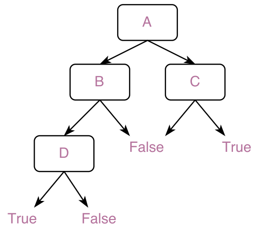
There are a number of open questions when selecting which of these representations to use for model-agnostic explanations. It is clear that no single one of these representations, by itself, provides the necessary tradeoff between expressivity and interpretability. Further, there have not been adequate studies into understanding this tradeoff (with Huysmans et al. [7] being an exception), and it is likely that different representations are appropriate for different kinds of users and domains. It is thus clear that picking any single such intrepretable representation as the choice of model-agnostic representation is not ideal.

In this position paper, we propose using *programs* to explain the local behavior of black-box systems. There are a number of advantages that such explanations will provide over using any single existing representation. First, programming languages are designed to capture complex behavior using a high-level syntax that is both succinct and intuitive, and there is a growing group of users that are already trained in reading and writing them. Second, programs can represent *any* Turing-complete behavior; any of the existing interpretable representation used in literature can be written as a program, but further, programs can also represent arbitrary combinations of multiple of these representations. It is also possible to trade off the expressivity and the comprehensibility of the program, for example simple programs for new programmers (at the cost of being an approximation of the complex system) or detailed, longer program for more accurate explanation of the behavior. Finally, we can potentially apply research in program/software analysis to evaluate various aspects of complex systems, such as automatically characterizing the complexity, security, privacy, and so on. By providing programs as model-agnostic explanations, we are essentially proposing an approach to *decompile* the local behavior of any black-box, complex systems.

In the following sections, we first demonstrate that program snippets provide a unified yet comprehensible syntax for the commonly used interpretable representations such as decision trees and linear models. We then formalize the problem of inducing programs as local explanations of black-box models, and describe a simulated-annealing based prototype implementation. Finally, we provide examples of generated programs for two datasets using multiple classifiers, demonstrating the expressivity and comprehensibility of using programs to approximate complex, black-box behavior.

[1] For example, a neural network with a thousand layers was introduced by He et al. [6].

30th Conference on Neural Information Processing Systems (NIPS 2016), Barcelona, Spain.



(a) Example decision tree

```
if A:
    if B: return D
    else: return False
else: return not C
```

(b) Program for decision tree

```
return 10*A - 9*B +2
```

```
return A or not B
```

(c) Equivalent programs for a linear model

Figure 1: Example decision tree (a) and its corresponding program (b) in our syntax, showing that the complexity is on a similar level. We also show two versions of the linear model, the latter of which is more compact and easier to understand. These programs were manually written.

1 Other Interpretable Representations as Programs

In this section we provide some simple examples of interpretable models currently used in the literature, and describe how their programmatic equivalent would look like. As it will become apparent, most of these existing interpretable models retain their readability when written as programs.

We use a fairly simple but expressive language for our programs that consists of boolean constants (True, False) and operators (and, or, not), absence/presence of the features of the input instance (Smoker), real valued constants (0.5) and algebraic operators (+,-,*), real-valued features (Age), and if-then-else conditions. This language is fairly expressive, but due to the lack of looping, recursion, and variables, is still not a complete programming language. We will use the Python syntax to render our programs, slightly abused to conserve space.

One of the most commonly used interpretable representations is that of decision trees [3]. From the simple decision tree in Figure 1a, along with the program for it in Figure 1b, it is clear that the program is a fairly intuitive representation. Along with decision trees, sparse linear models have also been used in a number of applications as interpretable representations of machine learning [20]. In Figure 1c, we shows two programs: first that exactly captures the behavior over the relevant features, while the second demonstrates a simpler program that has the same behavior if the features are binary and the prediction is true if the linear model evaluates to a positive score.

```
if RespIllness and Smoker and Age>=50: LungCancer
elif RiskDepression: Depression
elif BMI>=0.2 and Age>=60: Diabetes
elif Headaches and Dizziness: Depression
elif DocVisits>=0.3: Diabetes
elif DispTiredness: Depression
else: Diabetes
```

(a) Decision list

```
if RespIllness and Smoker and Age>=50: LungCancer
if RiskLungCancer and BP>=0.3: LungCancer
if RiskDepression and PastDepression: Depression
if BMI>=0.3 and Insurance=None and BP>=0.2: Depression
if Smoker and BMI>=0.2 and Age>=60: Diabetes
if RiskDiabetes and BMI>=0.4 and ProbInfections>=0.2: Diabetes
if DocVisits>=0.4 and ChildObesity: Diabetes
```

(b) Decision Set

Figure 2: Example of programs for a decision set and a decision list, originally appearing in Lakkaraju et al. [10], demonstrating that they remain easy to read as programs. These were written manually.

Recently, decision lists [21, 11] and decision sets [10] have been introduced as more comprehensible representations than decision trees, while being much more powerful that linear models. Since these are often presented using pseudo code, the program for these representations in our language looks essentially the same, as shown in Figure 2.

From these examples, it is clear that not only are programs able to represent the different interpretable representations succinctly, but the programming language can be much more expressive than any single representation. The key challenges is to actually synthesize the appropriate program, i.e. to make sure it is both a good approximation of the black-box model, and is as readable as the examples shown here. In the next section we will formalize this problem, and describe a prototype solution.

2 Inducing Program Explanations

In this section, we briefly outline our ideas on how to generate programs as explanations for complex systems, along with the description of a prototype implementation using simulated annealing.

Local, Model-Agnostic Explanations: Our goal here is to explain individual predictions of a complex machine learning system, by treating them in a black-box manner. The advantages of generating such model-agnostic explanations was described in Ribeiro et al. [18].

Our proposed work builds upon the ideas in Ribeiro et al. [17]. Let the black-box system be $f : \mathcal{X} \to \{0, 1\}$, and we are interested in explaining a specific prediction, i.e. $f(x) = y$. In order to generate an explanation that describes the behavior of f around x, we generate a number of random perturbations of x, denoted by Z. We then induce the program that both (1) accurately models the behavior of f on the samples Z (weighed by their similarity to x), and (2) is interpretable to the user. Specifically, we solve the following optimization:

$$\hat{p} = \arg\min_{p \in \mathcal{P}} \mathcal{L}(f, p, Z, \Pi_x) + \Omega(p) \tag{1}$$

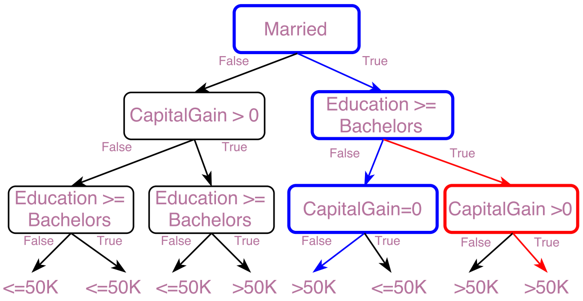
where P is the set of compatible programs (valid expressions that $\mathcal{X} \to \{0, 1\}$), $\mathcal{L}(f, p, Z, \Pi_x)$ is the loss between the outputs of f and p on the samples Z weighted by Π_x, and $\Omega(\cdot)$ denotes the complexity of the program (number of lines or the depth of the expression tree, for example).

Program Induction: Eq (1) is a challenging combinatorial optimization on a potentially complex surface (depending on the loss used). A related thread of research is *program induction*, where programs are synthesized automatically to match some desired goal [14]. Number of different variations of this problem have been introduced, depending on the syntax of the program and the formalism of the desired goals, with solvers ranging from genetic programming [2] to MCMC [12]. There has also been recent work in using probabilistic programs to identify such programs, mentioned as a possibility in Mansinghka [15] using Church [5], but with a recent implementation by Gaunt et al. [4]. However, we were unable to identify an off-the-shelf program inducer that can support an arbitrary loss (that depends on the domain) in order to identify the appropriate explanation.

Prototype Implementation: We implemented a prototype program inducer that approximately solves Eq (1) in order to generate program explanations. We use the same syntax as the one used in Section 1, i.e. boolean constants and operators, input features, real-valued constants and algebraic operators, and if-then-else conditions. In order to encapsulate the complexity of each program, we set $\Omega(.)$ to be 0 if the number of nodes in the expression tree is <8 and is ∞ otherwise, i.e. we are implicitly considering a family of short programs as $\mathcal{P}$. We use the negative of the weighted F_1 score as the loss $\mathcal{L}$, but the implementation supports any arbitrary function that is evaluated on the outputs of f and p on Z. The combinatorial optimization is solved using simulated annealing [9] with a logarithmically decreasing temperature schedule, and a proposal function that randomly grows, shrinks, or replaces nodes in the express tree to create valid perturbed expression trees.

3 Example Generated Programs

Using the program induction technique described in the previous section, here we present a few example program explanations for a number of classifiers (treated as black-boxes) on two datasets from the UCI repository [13]: *adult* and *hospital readmission*. In order to evaluate whether the programs are accurate as explanations, we also provide a visualization of the decision tree models.



(a) Decision tree (with the path highlighted)

Random Forests (true implies > 50K):

```
(if HoursPerWeek<=40:
   CapitalGain>0
else: True) and Married
```

Decision Tree (true implies > 50K):

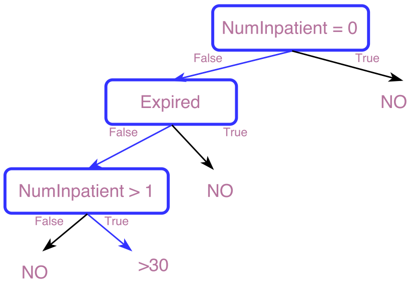
```
CapitalGain>0 and Married
```

Linear model (true implies > 50K):

```
if CapitalGain>0: Married
else : False
```

(b) Generated program explanations

Figure 3: **Adult dataset:** In (a), we show the learned tree, with the path for the instance in blue, and in red, we show that `Education` doesn't really matter for this instance. (b) shows the explanations for three classifiers (they got the prediction right), in particular showing that the explanation for the decision tree gets the more compact form.



(a) Decision tree (with the path highlighted)

Random Forests (true implies < 30):

```
if Diag:Other and not Tolbutamide:
    Discharged:Home
else: Diag:Other
```

Decision Tree (true implies > 30):

```
NumInpatient > 1.00
```

Linear model (true implies NO):

```
not Tolazamide
```

(b) Generated program explanations

Figure 4: **Hospital Readmission data:** (a) shows the learned tree, with the path for the instance in blue. Again, (b) shows the explanations for three classifiers (only the tree had the correct prediction), with the compact explanation for tree almost correct, except that it assumes the patient is alive.

In Figures 3a and 4a we show the learned decision tree on these datasets. We also trained a random forest classifier and a logistic regression model. Figures 3b and 4b show the generated program explanations for both the datasets, demonstrating that the programs are compact and readable, and ones for the decision trees are accurate to the model as well. Further, it is clear that random forests, which is much more complex in structure than trees or linear models, requires more complicated programs as explanations, however these programs still make sense (Figure 3b, in particular).

4 Conclusions and Future Work

In this paper we motivated the need to use programs as model-agnostic explanations: programs are designed to be intuitive to humans and are incredibly expressive. We presented a prototype implementation that induces programs as local explanations of a classifier by fitting to the classifier's predictions on a set of perturbations of the instance being explained. We demonstrated example explanations generated for multiple datasets and classifiers.

There are a number of exciting avenues for future work on these ideas. We will investigate methods for inducing programs with a much more expressive syntax, including, for example, loops and variables. Instead of relying on combinatorial optimization techniques that may not scale to applications on more complex domains, syntax, and systems, we will explore the use of recently introduced *differentiable* program induction techniques such as in Neelakantan et al. [16] and Riedel et al. [19]. Finally, on real-world applications and using user studies, we will thoroughly evaluate the interpretability and utility of using programs as local explanations of complex machine learning systems.

References

[1] David Baehrens, Timon Schroeter, Stefan Harmeling, Motoaki Kawanabe, Katja Hansen, and Klaus-Robert Müller. How to explain individual classification decisions. *Journal of Machine Learning Research*, 11, 2010.

[2] Forrest Briggs and Melissa O'neill. Functional genetic programming with combinators. In *Asian-Pacific workshop on Genetic Programming (ASPGP)*, pages 110–127, 2006.

[3] Mark W Craven and Jude W Shavlik. Extracting tree-structured representations of trained networks. *Advances in neural information processing systems*, pages 24–30, 1996.

[4] Alexander L Gaunt, Marc Brockschmidt, Rishabh Singh, Nate Kushman, Pushmeet Kohli, Jonathan Taylor, and Daniel Tarlow. Terpret: A probabilistic programming language for program induction. *arXiv preprint arXiv:1608.04428*, 2016.

[5] Noah D. Goodman, Vikash K. Mansinghka, Daniel Roy, Keith Bonawitz, and Joshua B. Tenenbaum. Church: a language for generative models. In *Uncertainty in Artificial Intelligence (UAI)*, 2008.

[6] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *arXiv preprint arXiv:1512.03385*, 2015.

[7] Johan Huysmans, Karel Dejaeger, Christophe Mues, Jan Vanthienen, and Bart Baesens. An empirical evaluation of the comprehensibility of decision table, tree and rule based predictive models. *Decis. Support Syst.*, 51(1):141–154, April 2011. ISSN 0167-9236. doi: 10.1016/j.dss.2010.12.003. URL `http://dx.doi.org/10.1016/j.dss.2010.12.003`.

[8] Been Kim, Cynthia Rudin, and Julie A Shah. The bayesian case model: A generative approach for case-based reasoning and prototype classification. In Z. Ghahramani, M. Welling, C. Cortes, N.D. Lawrence, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 1952–1960. Curran Associates, Inc., 2014.

[9] S. Kirkpatrick, C.D. Gelatt, and M.P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598): 671, 1983.

[10] Himabindu Lakkaraju, Stephen H. Bach, and Jure Leskovec. Interpretable decision sets: A joint framework for description and prediction. In *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pages 1675–1684, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-4232-2. doi: 10.1145/2939672.2939874. URL `http://doi.acm.org/10.1145/2939672.2939874`.

[11] Benjamin Letham, Cynthia Rudin, Tyler H. McCormick, and David Madigan. Interpretable classifiers using rules and bayesian analysis: Building a better stroke prediction model. *Annals of Applied Statistics*, 2015.

[12] Percy Liang, Michael I Jordan, and Dan Klein. Learning programs: A hierarchical bayesian approach. In *International Conference on Machine Learning (ICML)*, pages 639–646, 2010.

[13] M. Lichman. UCI machine learning repository, 2013. URL `http://archive.ics.uci.edu/ml`.

[14] Zohar Manna and Richard Waldinger. A deductive approach to program synthesis. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 2(1):90–121, 1980.

[15] Vikash Kumar Mansinghka. *Natively probabilistic computation*. PhD thesis, Massachusetts Institute of Technology, 2009.

[16] Arvind Neelakantan, Quoc V Le, and Ilya Sutskever. Neural programmer: Inducing latent programs with gradient descent. In *International Conference on Learning Representations (ICLR)*, 2015.

[17] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. "why should i trust you?": Explaining the predictions of any classifier. In *Knowledge Discovery and Data Mining (KDD)*, 2016.

[18] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. Model-agnostic interpretability of machine learning. In *ICML Workshop on Human Interpretability in Machine Learning (WHI)*, 2016.

[19] Sebastian Riedel, Matko Bošnjak, and Tim Rocktäschel. Programming with a differentiable forth interpreter. *arXiv preprint arXiv:1605.06640*, 2016.

[20] Berk Ustun and Cynthia Rudin. Supersparse linear integer models for optimized medical scoring systems. *Machine Learning*, 2015.

[21] Fulton Wang and Cynthia Rudin. Falling rule lists. In *Artificial Intelligence and Statistics (AISTATS)*, 2015.