INTERPRETING BLACK-BOXES USING PRIMITIVE PARAMETERIZED FUNCTIONS

Anonymous authors
Paper under double-blind review

ABSTRACT

One approach for interpreting black-box machine learning models is to find a global approximation of the model using simple interpretable functions, which is called a metamodel (a model of the model). Approximating the black-box with a metamodel can be used to 1) estimate instance-wise feature importance; 2) understand the functional form of the model; 3) analyze feature interactions. In this work, we propose a new method for finding interpretable metamodels. Our approach utilizes Kolmogorov superposition theorem, which expresses multivariate functions as a composition of univariate functions (our primitive parameterized functions). This composition can be represented in the form of a tree. Inspired by symbolic regression, we use a modified form of genetic programming to search over different tree configurations. Gradient descent is used to optimize the parameters of a given configuration. Using several experiments, we show that our method outperforms recent metamodeling approaches suggested for interpreting black-boxes.

1 INTRODUCTION

In the recent years machine learning (ML) algorithms made several breakthroughs in issuing accurate predictions. There is however a growing need to improve trustworthiness of these models. Providing accurate predictions is not enough in high-stake applications like healthcare where an agent (e.g. clinician) needs to interact with the model. In these applications the agent usually needs to understand how a particular prediction is issued. Especially, if the model prediction (say treatment plan) is different from what the clinician has in mind, explaining the model is vital. Complicated ML models like neural networks are essentially black-boxes to humans, and that is why interpretability methods are important and have gained significant attention in recent years (Ribeiro et al., 2016b; Lundberg & Lee, 2017; Guidotti et al., 2018; Arnaldo et al., 2014; Zhang et al., 2018; Alvarez-Melis & Jaakkola, 2018; Arrieta et al., 2020; Lou et al., 2013; Doshi-Velez & Kim, 2017).

There exist two key approaches to bring interpretability to machine learning models: (1) by designing inherently interpretable models Rudin (2019); Chen et al. (2019); Alvarez-Melis & Jaakkola (2018); or (2) by designing post-hoc methods to understand a pre-trained model Ribeiro et al. (2016a); Lipton (2016). In this work, we focus on the second approach that includes methods to analyse a trained model locally and globally (Montavon et al., 2018). The local interpretability methods focus on instance-wise explanations, which although useful, provide little understanding of a model's global behaviour (Ribeiro et al., 2016b; Lundberg & Lee, 2017). Hence, researchers have proposed multiple techniques to interpret how a ML model behaves for a group of the instances. Some examples of global analysis methods include permutation feature importance Molnar (2019), activation-maximisation Erhan et al. (2009), and learning globally surrogate models (Thrun, 1995; Craven & Shavlik, 1996).

Our work relates to the last approach that aims to learn interpretable proxies by approximating the behaviour of black-box ML models for multiple instances. Some efforts in this category include methods to approximate neural networks with if-then rules Thrun (1995) or decision trees Craven & Shavlik (1996) and the method to approximate matrix factorisation models using Bayesian networks and simple logic rules (Carmona et al., 2015). Our work is mostly relevant to a different category of approaches for learning interpretable proxies that focuses on approximating black-box functions with symbolic metamodels. A proper interpretable metamodel can enjoy benefits of different categories of interpretability methods. For example, a metamodel may provide insight into the interactions of different features and how they contribute in producing results. The metamodel can be locally approximated (e.g. using Taylor series) to generate instance-wise explanations. Moreover, it may be used for scientific discovery by revealing underlying laws governing the observed data (Schmidt & Lipson, 2009; Wang et al., 2019; Udrescu & Tegmark, 2020b).

Symbolic regression (SR) Koza (1994), has been the primary approach for finding approximate metamodels. In SR, there exist some fixed mathematical building blocks (e.g. summation operation), and the Genetic Programming (GP) algorithm searches over possible expressions that can be composed by combining the building blocks. We will explain SR in more details in Section 2 and compare it with our proposed method in Section 4. The major limitation of SR is that it uses a set of limited predefined building blocks and the search spaces grows when the number of building blocks increase. Two recent papers, which are the most relevant to our work Alaa & van der Schaar (2019); Crabbe et al. (2020), address this issue by suggesting the use of a parametric trainable class of functions instead of fixed building blocks. In particular, they suggest using Meijer G-functions (we briefly introduce this class in Section 2). Note that these are univariate functions, in order to use them in multivariate settings, Alaa & van der Schaar (2019) considers a heuristic approximation of Kolmogorov superposition theorem (KST) and Crabbe et al. (2020) considers the projection pursuit method (in Section 4, we show that their method can be also considered as an approximation of KST). Both these works start from a general framework, however, they make some restricting assumptions that limit the usability and coverage of their methods. For example, the simple function x_1x_2 (here x_i's are features) cannot be represented with the method given in (Crabbe et al., 2020). Similarly, the method in Alaa & van der Schaar (2019) fails to represent the product of three features $x_1x_2x_3$. Another limitation of the proposed approaches is that although most of familiar functions are indeed special cases of Meijer G-functions, for almost all parameters, Meijer G-functions do not have familiar closed form representation. Therefore, in practice, in the training of parameters it is very unlikely to obtain a set of parameters that are "interpretable".

In this work, we address the above challenges by proposing a new methodology to learn symbolic metamodels. Our approach considers a more general approximation of KST and involves representing the KST expression using trees with edges representing simple parameterized functions (e.g., exponential). Then we use gradient descent to train multiple trees and employ GP to search for the tree that most accurately approximates the black-box function. We demonstrate the efficacy of our proposed method through three experiments. The results suggest that our approach for estimating symbolic metamodels is comparatively more generic, accurate, and efficient than other symbolic metamodeling methods. In this work we are using our proposed method to provide interpretations, however, this method can be considered in general as a new GP method. Our method should be classified as a memetic algorithm where a population based method is paired with a refinement method (in our case gradient descent) (Chen et al., 2011). To the best of our knowledge, this is the first method that uses gradient descent not only for training numerical constants but also for the training of building blocks (i.e., primitive functions).

2 Preliminaries

In this section, we present a brief overview of building blocks of our proposed method: genetic programming; and classes of trainable functions.

Genetic Programming and symbolic regression: Genetic programming (GP) is an optimization method inspired by law of natural selection proposed by Koza in 1994 (Koza, 1994). It starts with a population of random programs for a particular task and then evolves the population in each iteration with operations inspired by natural genetic processes. The idea is that after enough iterations the population evolves and a fit program can be found in later generations. The two typical operations for evolving are crossover and mutation. In crossover, we choose the fittest programs (the fitness criterion is predefined for the task in hand) for reproduction of next generation (parents) and swap random parts of the selected pairs. In mutation operation, a random part of a program is substituted by some other randomly generated part of a program. One instance of using GP is for optimization in Symbolic Regression (SR), where the goal is to find a suitable mathematical expression to describe some observed data. In this setting, each program consists of primitive building blocks such as analytic functions, constants, and mathematical operations. The program is usually represented with a tree, where each node is representing one of the building blocks. We refer to Orzechowski et al. (2018); Wang et al. (2019) for more details on SR. GP as a population base optimization method can be paired with other refinement methods. For example, here we are using both GP and GD in our method. These methods are called memetic algorithms. in particular, our method should be classified as a *Lamarckian* memetic algorithm, where Lamarckian refers to the method of inheritance in GP search. we refer to (Emigdio et al., 2014; Chen et al., 2011) for more details on taxonomy of GP methods.

Class of trainable functions: In contrast with SR which uses fixed building blocks, our proposed approach (similar to (Alaa & van der Schaar, 2019) and (Crabbe et al., 2020)) uses a class of trainable parameterized functions as building blocks. One such class of functions is called Meijer G-functions and have been used in two recent approaches to learn symbolic metamodels (Meijer, 1946; 1936). A Meijer-G function $G^{m,n}_{p,q}$ is defined as an integral along the path $\mathcal{L}$ in the complex plane.

$$G^{m,n}_{p,q}\left(\begin{smallmatrix}a_1,\ldots,a_p\\ b_1,\ldots,b_q\end{smallmatrix}\,\middle|\,x\right) = \frac{1}{2\pi i}\int_{\mathcal{L}} \frac{\prod_{j=1}^{m}\Gamma(b_j-s)\prod_{j=1}^{n}\Gamma(1-a_j+s)}{\prod_{j=m+1}^{q}\Gamma(1-b_j+s)\prod_{j=n+1}^{p}\Gamma(a_j+s)}\, x^s\, ds, \tag{1}$$

where $0 \le m \le q$ and $0 \le n \le p$ are all integers, and $a_i, b_j \in \mathbb{R}$ for $1 \le i \le p$ and $1 \le j \le q$. $\mathcal{L}$ is a path which separates poles of $\Gamma(1-b_j+s)$ from poles of $\Gamma(a_j+s)$. By fixing m, n, p, q we have a class of parameterized functions (a_i's and b_i's are parameters), which can be trained using gradient descent. We refer to Beals & Szmigielski (2013) for a more detailed definition of these functions. Meijer G-functions are rich set of functions that have most of the familiar functions which we think of as interpretable as special cases. For example,

$$G^{0,1}_{3,1}\left(\begin{smallmatrix}2,2,2\\ 1\end{smallmatrix}\,\middle|\,x\right) = x, \quad G^{1,0}_{0,1}\left(\begin{smallmatrix}-\\ 0\end{smallmatrix}\,\middle|\,x\right) = e^{-x}, \quad \text{and} \quad G^{1,2}_{2,2}\left(\begin{smallmatrix}1,1\\ 1,0\end{smallmatrix}\,\middle|\,x\right) = \log(1+x).$$

However, when trained using gradient descent (GD), the final parameters for Meijer G-functions almost always will not have an interpretable closed form. This limits insight into the functional form of the black-box model. Hence, in this work, we propose using classes of simple, interpretable, parameterized functions that can be efficiently optimized using GD. The class of functions can be chosen by a domain expert for each particular task. We will discuss the selection of primitive functions further in Appendix C. Specifically, here we demonstrate our approach using the following five parameterized functions:

$$f_1(a,b|x) = ae^{-bx}, \quad f_2(a,b,c|x) = a\sin(bx+c), \quad f_3(a,b,c,d|x) = ax^3+bx^2+cx+d$$
$$f_4(a,b,c|x) = a\log(bx+c), \quad f_5(a,b,c,d|x) = ax/(bx^2+cx+d).$$

In Appendix C, we show that our presented results will not significantly change with using other set of primitive functions.

Remark. It is important to revisit that our proposed framework is generic and can accommodate any trainable class of functions, including Meijer G.

3 Method

Assume that a black box function $f : \mathcal{X} \to \mathbb{R}$ is trained on a dataset. Our goal is to find an interpretable function g which approximates f. To this end, we restrict g to belong to the class of functions $\mathcal{G}$ which are deemed to be interpretable. Therefore, we want to find the solution to the following optimization problem:

$$\arg\min_{g\in\mathcal{G}} \ell(f,g), \tag{2}$$

where ℓ is our loss function of choice. In this work, we assume ℓ to be mean square loss

$$\ell(f,g) = \int_{\mathcal{X}} (g(x)-f(x))^2 dx. \tag{3}$$

In order to approximate multivariate function f, we deploy Kolmogorov superposition theorem (Kolmogorov, 1957) which states that any multivariate continuous function (with d variables) has a representation in terms of univariate functions as follows:

$$g(\boldsymbol{x}) = g(x_1,\cdots,x_d) = \sum_{i=1}^{2d+1} g_i^{out}\left(\sum_{j=1}^{d} g_{ij}^{in}(x_j)\right). \tag{4}$$

In our setting, each of g_{ij}^{in} and g_i^{out} can be a function from $\mathcal{G}$. However, fully implementing this equation (especially, using computationally expensive Meijer G-functions) is impractical even for moderate values of d. Therefore, an approximation is proposed in Alaa & van der Schaar (2019) by considering a single outer function which is set to be identity and adding multiplication of all pairs of attributes to capture their correlation (we discuss this method in more details in Section 4). In this work we propose another method for approximating Equation (4).

3.1 APPROXIMATING KST

In our method, we approximate KST using trees with $L < 2d+1$ middle nodes, where each of them is connected to only a subset of inputs. We denote the middle nodes with h_i, for $1 \leq i \leq L$. Our approximation can be represented via a three layered tree (see Figure 1). There is a single root node at the top of the tree which is connected to L middle nodes. Each middle node is connected to a subset of bottom layer nodes. The bottom layer of the tree has d nodes corresponding to d features. For simplicity, when it is not confusing, we call the node corresponds to ith feature by x_i.

Note that each edge in the graph represents a univariate function. We denote the function corresponding to the edge between h_i and the root with g_{h_i} (these are the outer functions), and the function corresponding to an edge between h_i and x_j is denoted by g_{ij} (inner functions). The argument of g_{ij} is naturally the feature it is connected to, namely x_j, and the argument of g_{h_i} is the summation of all incoming functions to h_i. That is, $\sum_{j\in\mathcal{N}(h_i)} g_{ij}(x_j)$, where $\mathcal{N}(h_i)$ denotes the neighbours of node h_i in the graph. Finally, for the root node we sum all the outputs of all L middle layer functions. Therefore, each tree is representing a function from $\mathcal{X}$ to $\mathbb{R}$, which can be expressed as follows:

$$g(\boldsymbol{x}) = \sum_{i=1}^{L} g_{h_i}\left(\sum_{j\in\mathcal{N}(h_i)} g_{ij}(x_j)\right). \tag{5}$$

3.2 USING GP FOR TRAINING OF METAMODELS

Now we want to solve the optimization problem in (2), where $\mathcal{G}$ is the set of all functions that can be represented in form of Equation (5), where all g_{ij} and g_{h_i} are drawn from the class of primitive parameterized functions. We propose solving this optimization problem by running a version of genetic programming algorithm. The tree representation of Equation (5), resembles the trees in symbolic regression that represents each program. Note that, unlike normal GP, here our constructed trees has a fixed structure of three layers, and also edges are representing functions. Hence we need to modify GP accordingly. In this section, we explain the details of the GP algorithm, a sketch of the algorithm is presented in Figure 3 in Appendix.

3.2.1 PRODUCING RANDOM TREES

In the first step, we produce M random trees $T_1, \cdots, T_M$. Each tree T_i has L_i middle nodes, where L_i is an integer in $[l_1, l_2]$. l_1 and l_2 are important hyperparameters, determining number of middle nodes. For each of L_i middle nodes, a random subset of bottom layers will be chosen to be connected to this node. At first instance, for all $1 \leq u \leq L_i$ and $1 \leq v \leq d$, we connect h_u and x_v with probability $0 < p_0$. Then if there exist an x_v which is not connected to any of the middle nodes. We choose $1 \leq u \leq L_i$ uniformly at random and then connect x_v and h_u to ensure every x_i is connected to at least one of the middle nodes. p_0 is the parameter that controls sparsity of the produced graphs, which is one of the main factors that determine the complexity of the training procedure. Each edge is representing a function from our class of primitive functions, thus we uniformly at random choose one of the function classes for each edge and also initialize its parameters with samples from normal distribution.

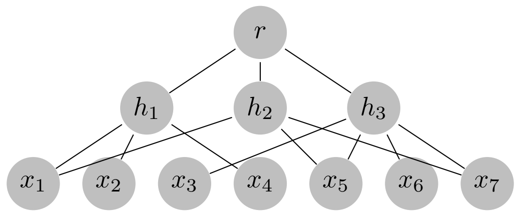

Figure 1: A sample tree structure, each edge is representing a univariate function

3.2.2 Training phases

In the training phase, for each tree, we update the parameters of each edge using gradient descent. We choose a constant k and apply k gradient descent updates on the parameters of functions g_{h_i} and g_{ij}. Let $g'_{h_i}(x) = \frac{dg_{h_i}(x)}{dx}$. For a one of the parameters of g_{ij} and b a parameter of g_{h_i}, the gradient of g with respect to a and b can be computed as follows (recall that g is representing the metamodel):

$$\frac{\partial g(\boldsymbol{x})}{\partial a} = \frac{\partial g_{ij}(x_j)}{\partial a} \cdot g'_{h_i}\left(\sum_{k\in\mathcal{N}(h_i)} g_{ik}(x_k)\right), \quad \frac{\partial g(\boldsymbol{x})}{\partial b} = \frac{\partial g_{h_i}}{\partial b}\left(\sum_{j\in\mathcal{N}(h_i)} g_{ij}(x_j)\right). \tag{6}$$

In this work, we choose a fixed learning rate and leave the exploration of using more advanced optimization techniques for future work (this is compatible with Alaa & van der Schaar (2019) and Crabbe et al. (2020), and allows us to have a fair comparison with these works).

3.2.3 Evaluation fitness of metamodels

For evaluating fitness of the trained metamodels, we uniformly at random sample m points from $\mathcal{X}$ and query the output of black-box f and metamodels $g_1, \cdots, g_M$ on these m points and compute the mean square loss for the metamodels to approximate (3) (the output of f is considered as the ground truth). If any of the M models has a loss less than a predefined threshold we terminate the algorithm. Otherwise, we choose the s fittest metamodels and discard the rest. These s survived metamodels are the parents that will populate the next generation of trees in the evolution process for the next round of the algorithm.

Regularization: We can modify the fitness criterion to favor simpler models. For encouraging sparsity of the tree, we can add a term to the MSE error for penalizing trees that have more edges. Denoting total number of edges with E, we use this criterion for evaluation fitness of the trees (λ is a hyperparameter):

$$\text{Fitness of a given tree} = \text{MSE} + \lambda E. \tag{7}$$

It should be highlighted that the regularization term will be only used for selecting the surviving trees and not for the gradient descend part. Hence, we can flexibly modify this further if needed. For example, if there is a preference towards a particular primitive function (e.g. because they are simpler) we can reflect that in the above criterion.

3.2.4 Evolution phase

In the evolution phase, we create the next generation of metamodels using survived trees. Similar to conventional GP algorithm, here we also define two operations to perform on each tree: Crossover and Mutation. For each of the s chosen trees like T, we first pass on T to the next generation, then we randomly choose $\frac{M}{s} - 1$ times one of the two operations, perform it on T, and add the resulting tree to the cohort of the next generation trees. Thus, the total number of trees in the next cohort is also M. Here we define the two operations which preserve the three layer structure of the trees:

- In the crossover operation, for T, we first randomly choose one of the nodes at the second layer of T. Then we uniformly at random choose one of the other $s-1$ trees, and then again uniformly at random choose one of its second layer nodes and replace that node alongside with all edges connected to that node with the chosen node in T. Notice that the edge connected to the root node will be also replaced. Moreover, note that the new tree will inherit the functions corresponding to replaced edges and their parameters.
- In the mutation operation, one of these two actions will be applied on the tree: 1) changing the function class of an edge, 2) removing an edge between the middle and input layers. In each round of mutation, we apply n_m times one of these two actions on the tree. When we change the class of function for an edge, we also randomly reinitialize the parameters of the corresponding function.

The above two operations allow us to explore different configurations of trees and classes. A pseudo code of algorithm is provided in Appendix A. We call our proposed method symbolic metamodeling using primitive functions (SMPF).

3.3 Different types of interpretation using SMPF

Instance-wise feature importance: Similar to Alaa & van der Schaar (2019) and Crabbe et al. (2020) we can use the learned metamodel for estimating instance-wise feature importance. We can find the Taylor expansion of the metamodel around the data point of interest $\boldsymbol{x}_0$ and analyse its coefficients: $g(\boldsymbol{x}) = g(\boldsymbol{x}_0) + \nabla g(\boldsymbol{x}_0).(\boldsymbol{x} - \boldsymbol{x}_0) + (\boldsymbol{x} - \boldsymbol{x}_0).H_x(\boldsymbol{x}).(\boldsymbol{x} - \boldsymbol{x}_0) + \cdots$. First order partial derivative with respect to jth feature can be computed using chain rule:

$$\frac{\partial g(\boldsymbol{x})}{\partial x_j} = \sum_{h_i \in \mathcal{N}(x_j)} g'_{h_i} \left(\sum_{j \in \mathcal{N}(h_i)} g_{ij}(x_j) \right) g'_{ij}(x_j). \tag{8}$$

We will use this method in our second experiment. Importantly, we can also compute higher order coefficients for analyzing feature interactions.

Mathematical expressions: The final expression of the metamodel can provide insights into the functional form of the black-box function. For example, in the first experiment, we show that the metamodel correctly identifies that the black-box is an exponential function. Moreover, the inspection of mathematical expressions provides information about the interactions between the input features, and can potentially lead to understanding of previously unknown facts about the underlying mechanisms to domain experts.

An idea for exploring in future work is inspecting the final cohort of graphs. For example, if in the last iteration, the average degree of a node is large across different graphs, this can show the importance of the corresponding feature. Similarly, when a subset of features are connected to a middle node it can show the interaction of those features.

4 Related works

In the experiments section, we compare our approach with three symbolic metamodeling methods. This section briefly introduces these approaches, highlighting their strengths and weaknesses.

Symbolic Metamodeling (SM) Alaa & van der Schaar (2019): SM proposes using Meijer G-functions for interpreting black box models. In the derivation of their method, they also start with KST (4), however, with a different approximation: they consider only one outer function (g^{out}) and set that function to be identity (the inner functions are all Meijer G). This does not allow the features to interact, in order to fix this problem, they added multiplication of all pairs of actions $x_i x_j$ to the features. This setting has two main issues, firstly this method cannot capture interaction of more than two features and does not show other forms of interactions apart from multiplication. Secondly, this approach introduces many new features which makes it impractical when d increases. There are $\binom{d}{2} + d$ features in total and there is a Meijer G-function corresponding to each of them which makes using SM computationally costly.

Symbolic Pursuit (SP) Crabbe et al. (2020): SP is a subsequent work to SM and is designed to overcome some of its flaws. In particular, SP is designed to use fewer Meijer G-functions. The method is based on the Projection Pursuit algorithm in statistics (Friedman & Stuetzle, 1981). In each step of the algorithm, a Meijer G-function will be fitted which minimizes the residual error between the metamodel and the black-box. The final metamodel will be the summation of all these Meijer G-functions. The input of each function is a linear combination of features. Thus, the final function will have the following formulation:

$$g(\boldsymbol{x}) = \sum_{i=1}^{L} g_i \left(\sum_{j=1}^{d} c_{ij} x_j \right), \tag{9}$$

where g_i's are Meijer G-functions. Importantly, the authors use a modified version of (9) where the arguments of Meijer G-functions are normalized such that they lie in the open interval of 0 to 1. Moreover, SP involves adding weights to the outer summation to allow mitigating the contributions of previously found functions, if needed. Note that SP can be considered as one instance of our framework. The equation (9) is compatible with KST (4) and can be represented similar to Figure 1. In essence, all inner function (edges between bottom and middle layers) are restricted to be linear,

Table 1: Approximating two-variable functions using SM, SP, SR and SMPF.

		$f(\boldsymbol{x}) = e^{-3x_0+x_1}$	$f(\boldsymbol{x}) = \sin(x_0 x_1)$	$f(\boldsymbol{x}) = \frac{x_0 x_1}{(x_0^2+x_1)}$	$f(\boldsymbol{x}) = \text{sinc}(x_0^2 + x_1)$
SMPF	MSE	**0.001 ± 0.0002**	0.012 ± 0.002	**0.002 ± 0.0004**	**0.004 ± 0.0004**
	R^2	**0.996 ± 0.002**	0.962 ± 0.004	**0.895 ± 0.013**	**0.952 ± 0.003**
SM	MSE	0.174 ± 0.031	0.126 ± 0.009	0.108 ± 0.0104	0.193 ± 0.006
	R^2	0.273 ± 0.019	-2.039 ± 0.442	-5.461 ± 0.746	-0.263 ± 0.094
SP	MSE	0.009 ± 0.004	0.0008 ± 0.0001	0.002 ± 0.0003	0.009 ± 0.002
	R^2	0.958 ± 0.014	0.978 ± 0.003	0.878 ± 0.021	0.937 ± 0.015
SP^p	MSE	0.009 ± 0.001	0.024 ± 0.001	0.011 ± 0.001	0.010 ± 0.001
	R^2	0.953 ± 0.014	0.348 ± 0.082	0.345 ± 0.807	0.932 ± 0.013
SR	MSE	0.078 ± 0.018	**0.0004 ± 0.0002**	0.012 ± 0.002	0.016 ± 0.003
	R^2	0.658 ± 0.032	**0.988 ± 0.003**	0.256 ± 0.144	0.886 ± 0.034

basically, they are coming from class of $f(x) = cx$. There are L middle nodes, and outer functions are drawn from the class of Meijer G-functions. Also, in their setup $p_0 = 1$ (p_0 was the probability of connecting two nodes). A major problem with SP is its capability in representing non-linear correlations between the features. For example, a simple function like $x_1 x_2$ cannot be represented in SP formulation. Therefore, when using SP for explaining this function, in the best case scenario, by inspecting c_{i1} and c_{i2} we can understand that these two features are important but we cannot see how they interact. This can be potentially resolved in our framework by using a more general class of functions as inner functions.

Symbolic Regression: We briefly introduced SR in Section 2. SR searches over mathematical expressions that can be produced by combining a set of predetermined functions. In each program, the leaf nodes are either features or numerical values, and other nodes are mathematical operations. One main difference between SR and our method (also SM and SP) is that unlike SR our methods are based on a representation derived from KST. Furthermore, we use parametric functions (and GD) which cannot be accommodated in SR setting (note that GD has been suggested in SR but only for training of leafs, e.g. see (Topchy & Punch, 2001; Kommenda, 2018)). Importantly, SR has an advantage over SP and SM that the final result expression is guaranteed to be explainable, as it will be a combination of functions that we chose to include as the building blocks. However, when Meijer G-functions are used (in SM and SP), the resulting metamodel may not have a simple and explainable representation. This issue is resolved in our framework. There are several extensions on the original SR method, techniques introduced in some of them can be considered for future work to improve GP in our method as well (Arnaldo et al., 2014; Rad et al., 2018; Moraglio et al., 2012; Wang et al., 2019; Orzechowski et al., 2018; Chen et al., 2015; Udrescu & Tegmark, 2020a).

5 Experiments

We evaluate and compare our proposed method using three experiments. In the first experiment, we use our method to approximate four functions with simple expressions (similar to first experiment of (Alaa & van der Schaar, 2019)). In the second experiment, we use our method for estimating instance-wise feature importance for three synthetic datasets (similar to Alaa & van der Schaar (2019) and Chen et al. (2018)). Finally, in the third experiment, we consider black-boxes trained on real data and approximate it using our method (similar to (Crabbe et al., 2020)). The hyperparameters used for training the metamodels and additional results are reported in Appendix E.

5.1 Metamodels for fixed functions

In this experiment, we find metamodels for four synthetic functions with two variables. We compare the performance of our method (SMPF) with symbolic metamodeling (SM), symbolic pursuit (SP), polynomial approximation of SP (SP^p), and symbolic regression Orzechowski et al. (2018) (similar to Alaa & van der Schaar (2019) we use gplearn library Stephens (2015) for implementation of SR). We compare methods in terms of mean squared error (MSE) and R^2 score. Generally, our algorithm achieves a better accuracy as compared to other methods (we have the best score for three of the functions). The results are reported in Table 1. Furthermore, SMPF was able to correctly identify the functional form. For the first experiment, the

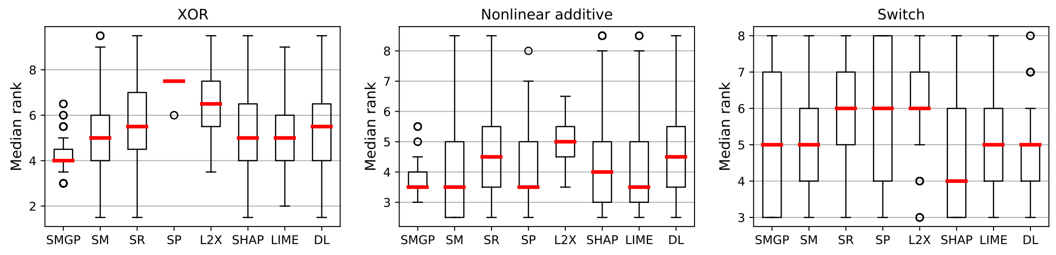

Figure 2: Box-plot of feature importance for three datasets. The red lines show the median ranks under each algorithm. Lower median ranks imply better performance. DL refers to DeepLIFT.

final expression of the metamodel is as follows (we rounded up coefficients here): $g(\boldsymbol{x}) = 0.854\exp\left(-2.438\sin(1.371x_0 - 0.0318) + \frac{0.684x_1}{0.016x_1^2+0.204x_1+0.426}\right)$. This shows an important advantage of our method in comparison to other methods. The expression found by SP algorithm has the following form ($P1$ here is a linear combination of the two inputs): $g(x) = 0.98\,G^{2,1}_{2,3}\left(\begin{smallmatrix}0.24,-0.06\\0.16,-0.47,0.43\end{smallmatrix}\,\middle|\,1.0[ReLU(P1)]\right)$. Note that it was not possible to find a closed form expression for this function. Also, for the second function, sin is correctly chosen as the outer function in SMPF. See Appendix E, where we provide results for synthetic functions with more variables.

5.2 Instance-wise feature selection

In this experiment, we evaluate the ability of our algorithm in finding the feature importance. In particular, we run the second experiment of (Alaa & van der Schaar, 2019) with addition of SMPF. Three synthetic datasets are used: XOR, Nonlinear additive features, and Feature switching. All the datasets have 10 features. In XOR, only the first two features contribute in producing the output. In Nonlinear additive features and switch datasets, the first four features and first five features are important, respectively.[1] First, a 2-layer neural network $f(\boldsymbol{x})$ is trained with 200 hidden neurons for estimating the label of each data point. Then, we run our algorithm to find metamodel $g(\boldsymbol{x})$ to approximate $f(\boldsymbol{x})$. The coefficient of each feature in the first-order Taylor approximation of $g(\boldsymbol{x})$ is a metric for its importance. The larger the coefficient, the more important it will be. We consider 1000 data points, and rank the features based on their importance. Then we find the median feature importance ranking of the relevant features. The median value determines the accuracy of our algorithm; a smaller median rank implies a better accuracy. Figure 2 compares our algorithm with Symbolic Metamodeling (SM) Alaa & van der Schaar (2019), Symbolic Pursuit (SP) Crabbe et al. (2020), Symbolic Regression Orzechowski et al. (2018), DeepLIFT Shrikumar et al. (2017), SHAP Lundberg & Lee (2017), LIME Ribeiro et al. (2016b), and L2X (Chen et al., 2018). As we can see, our algorithm performs competitively comparing with other algorithms. For XOR dataset we have the best median rank, and we are among the best for nonlinear additive dataset. On Switch dataset, SMPF performs similar to other global methods, i.e., SM, SP, and SR which are our direct competitors. SHAP is the only algorithm that has a better performance on this dataset.

5.3 Black-box approximation

In this experiment, we evaluate performance of our model on interpreting a black-box trained on real data, replicating the second experiment of (Crabbe et al., 2020). A Multilayer Perceptron (MLP), and Support Vector Machine (SVM) are trained as two black boxes using UCI dataset Yacht (Dua & Graff, 2017) (additional results are reported in Appendix E). In order to have the same setting as SP, we train the MLP and SVM models using the scikit-learn library Buitinck et al. (2013) with the default parameters. We randomly use 80% of the data points for the training of the black box model as well as SMPF model, and the remaining 20% is used to evaluate the performance of the model. This procedure is repeated five times to report the averages and standard deviations. We report the

[1]See Appendix B of Alaa & van der Schaar (2019) for more details.

Table 2: Interpreting black-boxes trained on real data using SMPF compared with SP

	Method	MLP		SVM	
		MSE	R^2	MSE	R^2
Black Box		0.689 ± 0.224	0.703 ± 0.019	0.448 ± 0.241	0.781 ± 0.061
Method v.s. Black Box	SMPF	0.007 ± 0.003	0.993 ± 0.003	0.029 ± 0.013	0.967 ± 0.120
	SP	0.008 ± 0.011	0.978 ± 0.016	0.014 ± 0.015	0.974 ± 0.078
Method	SMPF	0.674 ± 0.211	0.709 ± 0.015	0.344 ± 0.163	0.829 ± 0.037
	SP	0.682 ± 0.225	0.697 ± 0.027	0.471 ± 0.253	0.780 ± 0.048

MSE and R^2 score of the MLP and SVM against the true labels, MSE and R^2 of the metamodel against the black-box models, and the MSE and R^2 of the metamodel against the true labels (see Table 2). We observe that both SP and SMPF have very good performance in approximating the black-box. Interestingly, SMPF outperforms the black-box on the test set for both models which may indicate that the black-box overfits the dataset, but SMPF does not, as it uses simple functions.

6 Discussion

Complexity: In terms of run-time, for the last experiment, the training process of the SP algorithm for the MLP black-box takes 215 minutes, while the training process of our algorithm takes 45 minutes (both done on a personal computer). The reason that SP is more computationally expensive as compared to SMPF is that SP has to evaluate Meijer G-functions in each iteration of their optimization process. Evaluating a Meijer G-function is very expensive and takes about 1 to 4 seconds depending on the hyperparameters (i.e., m, n, p, q). This observation implies that SMPF has lower computational complexity which allows us to handle more variables and also enables the possibility of using more complex trees, as we suggest later in the future work. However, this should be highlighted that our method (similar to other symbolic methods) is not appropriate for high dimensional data like images.

Limitations: Even though we showed the performance of our model through extensive numerical experiments, our method lacks theoretical guarantees (theoretical analysis is particularly challenging because of the use of GP). Another limitation (also inherited from GP) is that there are several hyperparameters in our model to specify structure of the tree. As discussed, symbolic metamodels cannot handle high dimension inputs. Finally, the richness of functions we can create is limited, this can be compensated using more complex classes of functions or more complex tree structures.

Direct training vs using black-box: A natural question is why not directly use the training data to train the metamodel? There are two reasons for why we used the black-box for training. One is from the user point of view, we may have been given a task of interpreting a black-box, i.e., the user's question may be why this particular method is working, and not necessarily looking for another interpretable method. Secondly, and more importantly, we may not have access to the dataset for various reasons including privacy concerns. In this method we only need querying the black-box method and we can use random inputs (as many of them as we want). Directly using the dataset in all symbolic metamodeling methods (e.g. SR, SM, and SP) is certainly possible and can be relevant in many scenarios (e.g., discovering the underlying governing rules of a dataset).

Conclusion and future work: We proposed a new generic framework for symbolic metamodeling based on the Kolmogorov superposition theorem. We suggested using simple parameterized functions to get a closed-form and interpretable expression for the metamodel. The use of simple functions may seem restrictive when compared with SM and SP which use Meijer G-functions (a richer class of functions). However, this is compensated in our framework with a better approximation of KST. We used genetic programming to search over different possible trees and also possible classes of functions. There are several directions for the expansion of this work: 1) we can consider a more complex tree structure. For example, we can have trees with four layers instead of three, which allows us to construct more complex expressions (see Appendix D). 2) Other primitive functions can be used in our setup, e.g., Meijer G-functions. 3) The optimization in the training phase can be improved. The problem is non-convex, and gradient descent may not be able to find the global optimal point. This issue can be addressed by imposing convex relaxation or using more sophisticated non-convex optimization methods.

7 Ethics and Reproducibility Statements

Ethics Statement: This should be highlighted that we do not claim that the functional form found via our method is necessarily the correct function. This is the case even if the metamodel has a very good performance in approximating the black box.

Reproducibility Statement: We have included the code for all three experiments in the supplementary material. We also, included a detailed description in a read me file, explaining how to run the code. Details of the experiments and hyperparameters are reported in Appendix E.

References

Ahmed M Alaa and Mihaela van der Schaar. Demystifying black-box models with symbolic metamodels. In *Advances in Neural Information Processing Systems*, pp. 11304–11314, 2019.

David Alvarez-Melis and Tommi S. Jaakkola. Towards Robust Interpretability with Self-Explaining Neural Networks. In *Proceedings of the 32nd Conference on Neural Information Processing Systems (NeurIPS)*, pp. 7786–7795. Montréal, Canada, December 3–8 2018.

David Alvarez-Melis and Tommi S Jaakkola. Towards robust interpretability with self-explaining neural networks. *arXiv preprint arXiv:1806.07538*, 2018.

Ignacio Arnaldo, Krzysztof Krawiec, and Una-May O'Reilly. Multiple regression genetic programming. In *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation*, pp. 879–886, 2014.

Alejandro Barredo Arrieta, Natalia Díaz-Rodríguez, Javier Del Ser, Adrien Bennetot, Siham Tabik, Alberto Barbado, Salvador García, Sergio Gil-López, Daniel Molina, Richard Benjamins, et al. Explainable artificial intelligence (xai): Concepts, taxonomies, opportunities and challenges toward responsible ai. *Information Fusion*, 58:82–115, 2020.

Sebastian Bach, Alexander Binder, Gregoire Montavon, Frederick Klauschen, and Klaus-Robert Muller. On Pixel-Wise Explanations for Non-Linear Classifier Decisions by Layer-Wise Relevance Propagation. *PloS one*, 10(7), 2015.

Richard Beals and Jacek Szmigielski. Meijer g-functions: a gentle introduction. *Notices of the AMS*, 60(7):866–872, 2013.

Lars Buitinck, Gilles Louppe, Mathieu Blondel, Fabian Pedregosa, Andreas Mueller, Olivier Grisel, Vlad Niculae, Peter Prettenhofer, Alexandre Gramfort, Jaques Grobler, et al. Api design for machine learning software: experiences from the scikit-learn project. *arXiv preprint arXiv:1309.0238*, 2013.

Vicente Iván Sánchez Carmona, Tim Rocktäschel, Sebastian Riedel, and Sameer Singh. Towards Extracting Faithful and Descriptive Representations of Latent Variable Models. In *AAAI Spring Symposium on Knowledge Representation and Reasoning: Integrating Symbolic and Neural Approaches*. Palo Alto, California, USA, March 23–25 2015.

Chaofan Chen, Oscar Li, Alina Barnett, Jonathan Su, and Cynthia Rudin. This Looks Like That: Deep Learning for Interpretable Image Recognition. In *Proceedings of the 33rd Conference on Neural Information Processing Systems (NeurIPS)*. Vancouver, Canada, December 8–14 2019.

Jianbo Chen, Le Song, Martin Wainwright, and Michael Jordan. Learning to explain: An information-theoretic perspective on model interpretation. In *International Conference on Machine Learning*, pp. 883–892. PMLR, 2018.

Qi Chen, Bing Xue, and Mengjie Zhang. Generalisation and domain adaptation in gp with gradient descent for symbolic regression. In *2015 IEEE congress on evolutionary computation (CEC)*, pp. 1137–1144. IEEE, 2015.

Xianshun Chen, Yew-Soon Ong, Meng-Hiot Lim, and Kay Chen Tan. A multi-facet survey on memetic computation. *IEEE Transactions on Evolutionary Computation*, 15(5):591–607, 2011.

Jonathan Crabbe, Yao Zhang, William Zame, and Mihaela van der Schaar. Learning outside the black-box: The pursuit of interpretable models. *Advances in Neural Information Processing Systems*, 33, 2020.

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

Finale Doshi-Velez and Been Kim. Towards a rigorous science of interpretable machine learning. *arXiv preprint arXiv:1702.08608*, 2017.

Dheeru Dua and Casey Graff. UCI machine learning repository, 2017. URL `http://archive.ics.uci.edu/ml`.

Z Emigdio, Leonardo Trujillo, Oliver Schütze, Pierrick Legrand, et al. Evaluating the effects of local search in genetic programming. In *EVOLVE-A Bridge between Probability, Set Oriented Numerics, and Evolutionary Computation V*, pp. 213–228. Springer, 2014.

Dumitru Erhan, Yoshua Bengio, Aaron C. Courville, and Pascal Vincent. Visualising Higher-Layer Features of a Deep Network. Technical Report 1341, University of Montreal, June 2009.

Jerome H Friedman and Werner Stuetzle. Projection pursuit regression. *Journal of the American statistical Association*, 76(376):817–823, 1981.

Riccardo Guidotti, Anna Monreale, Salvatore Ruggieri, Dino Pedreschi, Franco Turini, and Fosca Giannotti. Local rule-based explanations of black box decision systems. *arXiv preprint arXiv:1805.10820*, 2018.

Andrei Nikolaevich Kolmogorov. On the representation of continuous functions of many variables by superposition of continuous functions of one variable and addition. In *Doklady Akademii Nauk*, volume 114, pp. 953–956. Russian Academy of Sciences, 1957.

Michael Kommenda. *Local Optimization and Complexity Control for Symbolic Regression/eingereicht von Michael Kommenda*. PhD thesis, Universität Linz, 2018.

John R Koza. Genetic programming as a means for programming computers by natural selection. *Statistics and computing*, 4(2):87–112, 1994.

Zachary C. Lipton. The Mythos of Model Interpretability. In *Proceedings of the International Conference on Machine Learning (ICML) Workshop on Human Interpretability in Machine Learning*. New York, USA, June 2016.

Yin Lou, Rich Caruana, Johannes Gehrke, and Giles Hooker. Accurate intelligible models with pairwise interactions. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 623–631, 2013.

Scott Lundberg and Su-In Lee. A unified approach to interpreting model predictions. *arXiv preprint arXiv:1705.07874*, 2017.

CS Meijer. Uber whittakersche bezw. besselsche funktionen und deren produkte (english translation: About whittaker and bessel functions and their products). *Nieuw Archief voor Wiskunde*, 18(2): 10–29, 1936.

CS Meijer. On the g-function. *North-Holland*, 1946.

Christoph Molnar. Interpretable Machine Learning: A Guide for Making Black Box Models Explainable, 2019. URL `https://christophm.github.io/interpretable-ml-book/`. Accessed December 17, 2019.

Grégoire Montavon, Wojciech Samek, and Klaus-Robert Müller. Methods for Interpreting and Understanding Deep Neural Networks. *Digital Signal Processing*, 73:1–15, 2018.

Alberto Moraglio, Krzysztof Krawiec, and Colin G Johnson. Geometric semantic genetic programming. In *International Conference on Parallel Problem Solving from Nature*, pp. 21–31. Springer, 2012.

Patryk Orzechowski, William La Cava, and Jason H Moore. Where are we now? a large benchmark study of recent symbolic regression methods. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 1183–1190, 2018.

Hossein Izadi Rad, Ji Feng, and Hitoshi Iba. Gp-rvm: Genetic programing-based symbolic regression using relevance vector machine. *arXiv preprint arXiv:1806.02502*, 2018.

Marco Túlio Ribeiro, Sameer Singh, and Carlos Guestrin. Model Agnostic Interpretability of Machine Learning. In *Proceedings of the International Conference on Machine Learning (ICML) Workshop on Human Interpretability in Machine Learning*. New York, USA, June 2016a.

Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. Why should i trust you?" explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pp. 1135–1144, 2016b.

Cynthia Rudin. Stop Explaining Black Box Machine Learning Models for High Stakes Decisions and Use Interpretable Models Instead. *Nature Machine Intelligence*, 1(5):206–215, 2019.

Michael Schmidt and Hod Lipson. Distilling free-form natural laws from experimental data. *science*, 324(5923):81–85, 2009.

Avanti Shrikumar, Peyton Greenside, and Anshul Kundaje. Learning important features through propagating activation differences. In *International Conference on Machine Learning*, pp. 3145–3153. PMLR, 2017.

T Stephens. Gplearn model, genetic programming. 2015.

Mukund Sundararajan, Ankur Taly, and Qiqi Yan. Axiomatic Attribution for Deep Networks. In *Proceedings of the 34th International Conference on Machine Learning (ICML)*, pp. 3319–3328. Sydney, Australia, August 6–11 2017.

Sebastian Thrun. Extracting rules from artificial neural networks with distributed representations. *Advances in neural information processing systems*, pp. 505–512, 1995.

Alexander Topchy and William F Punch. Faster genetic programming based on local gradient search of numeric leaf values. In *Proceedings of the genetic and evolutionary computation conference (GECCO-2001)*, volume 155162. Morgan Kaufmann, 2001.

Silviu-Marian Udrescu and Max Tegmark. Ai feynman: A physics-inspired method for symbolic regression. *Science Advances*, 6(16):eaay2631, 2020a.

Silviu-Marian Udrescu and Max Tegmark. Symbolic pregression: Discovering physical laws from distorted video. *arXiv e-prints*, pp. arXiv–2005, 2020b.

Yiqun Wang, Nicholas Wagner, and James M Rondinelli. Symbolic regression in materials science. *MRS Communications*, 9(3):793–805, 2019.

Xin Zhang, Armando Solar-Lezama, and Rishabh Singh. Interpreting neural network judgments via minimal, stable, and symbolic corrections. *arXiv preprint arXiv:1802.07384*, 2018.

A Pseudo code and flowchart of our algorithm

In this section, we summarize our algorithm in the following pseudo code. For the notational convenience, we denote g_{T_i} as the metamodel corresponding to tree T_i, E_i as the number of edges in tree T_i, Thr as a fixed threshold, lr as the learning rate, Max_Itr as the maximum number of iterations, and $\text{MSE}_i = \frac{1}{n}\sum_{j=1}^{n}(g_{T_i}(x_i) - f(x_i))^2$, where $\{x_1, \dots, x_n\}$ is the training dataset.

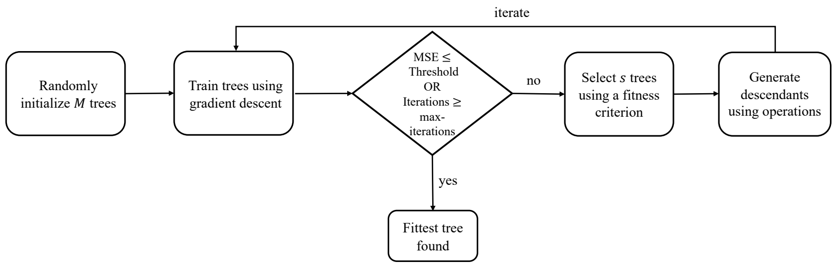
iterate
Randomly initialize M trees
Train trees using gradient descent
MSE ≤ Threshold OR Iterations ≥ max-iterations
no
Select s trees using a fitness criterion
Generate descendants using operations
yes
Fittest tree found

Figure 3: Flowchart of our Genetic programming algorithm

Algorithm 1: Pseudo code for SMPF

```
Input: Black-box: $f(x)$; Training set: $\{x_1, \dots, x_n\}$;
          Hyperparamters: $M, l_1, l_2, s, p_0, k$, Thr, lr, $\lambda$, $p_{\text{cross}}$, $p_{\text{del}}$, Max_itr.
Output: Metamodel $g(x)$ approximating function $f(x)$
Generates $L_i, i = 1, \dots, M$, where $L_i$ is a random integer between $l_1$ and $l_2$.;
Generates $M$ random trees $T_1, \dots, T_M$ using parameters $L_i$ and $p_0$ (details provided in Section
  3.2.1);
Itr $\leftarrow$ 0;
while $Itr \leq Max\_itr$   and   $\min_i MSE_i + \lambda E_i \geq Thr$ do
    for i = 1, 2,..., M do
        Update the parameters of the functions corresponds to the edges of tree $T_i$ using $k$
          gradient descent updates and learning rate lr;
        Calculate the mean squared error (MSE$_i$) corresponds to tree $T_i$;
        Fitness$_i$ $\leftarrow$ MSE$_i$ $+ \lambda \cdot E_i$;
    end
    Select $s$ trees with the smallest Fitness$_i$ and re-index them by $T_1, \dots, T_s$ ;
    for $i = 0, 1, \dots, s-1$ do
        for $j = 1, 2, \dots, M/s - 1$ do
            Generate random number $u$ uniformly in $[0, 1]$;
            if $u < p_{cross}$ then
                Generate tree $T_{i\cdot(M/s-1)+j+s}$ by crossover operation on tree $T_{i+1}$;
            else
                Generate tree $T_{i\cdot(M/s-1)+j+s}$ by mutation operation on tree $T_{i+1}$. In this step,
                  an edge is removed with probability $p_{\text{del}}$;
            end
        end
    end
    Itr $\leftarrow$ Itr + 1;
end
$i^* \leftarrow \arg\min_i$ MSE$_i$ $+ \lambda E_i$;
$g(x) \leftarrow g_{T_{i^*}}$;
```

B Related work table

In this section we provide a comparison table for several important interpretation methods. We adopted columns of this table from (Crabbe et al., 2020)[Table 3]. As it can be seen all of the

Table 3: Comparison of interpretability methods

Algorithm	Feature importance	Feature Interaction	Model independent	Global	Interpretable expression
LIME (Ribeiro et al., 2016b)	✓		✓		
SHAP (Lundberg & Lee, 2017)	✓	✓	✓		
DeepLIFT (Shrikumar et al., 2017)	✓				
GA^{2}M (Lou et al., 2013)	✓	✓	✓		
IG (Sundararajan et al., 2017)	✓				
LRP (Bach et al., 2015)	✓		✓		
L2X (Chen et al., 2018)	✓		✓		
SM (Alaa & van der Schaar, 2019)	✓	✓	✓	✓	
SP (Crabbe et al., 2020)	✓	✓	✓	✓	
SP^p (Crabbe et al., 2020)	✓	✓	✓	✓	✓
SR (Koza, 1994)	✓	✓	✓	✓	✓
SMPF	✓	✓	✓	✓	✓

approaches can provide feature importance, some of them are also capable of providing feature interactions. Most of the methods are indeed model independent. The subset of the methods that are global is determined. Finally, the interpretable expression here means whether the method is able to produce a closed-form expressions (this is only applicable for symbolic approaches).

C Primitive functions

In this section we discuss the selection of primitive functions. An advantage of our framework is its generality and the fact that it can work with any set of parameterized functions as long as its parameters can be trained using gradient descent. The primitive functions, in principle, can be chosen by the domain expert for the particular task in hand. For example, in the main text we considered five familiar functions as our primitive functions. The polynomial function there, can be thought of as Taylor approximation (hence it is reasonable to include it). The other four functions can be thought of as domain expert suggestion.

To show that our results is not particularly dependent on the choice of primitive functions. Here, we repeat our experiments for two more set of primitive functions. In the first set we are using only three of the five functions: polynomial, exponential, and sinusoidal. In the second set, we keep the polynomial function and replace the other two functions with new parameterized functions.

$$\text{Set 1:} \quad f_1(a,b|x) = ae^{-bx}, \quad f_2(a,b,c|x) = a\sin(bx+c),$$
$$f_3(a,b,c,d|x) = ax^3 + bx^2 + cx + d.$$

$$\text{Set 2:} \quad f_1(a,b,c,d|x) = ax^3 + bx^2 + cx + d, \quad f_2(a,b,c|x) = a\arctan(bx+c),$$
$$f_3(a,b,c,d|x) = \frac{ax+b}{cx+d}.$$

In Table 4, we report result of the first experiment for these two sets of primitive functions. It is interesting to note that removing sin function from Set 2 resulted in an inferior performance for sin and sinc functions. Also, inclusion of the function with fraction form ($f_3(a,b,c,d|x) = \frac{ax+b}{cx+d}$) in Set 2, improved the performance of this set on the third function.

Table 4: Approximating two-variable synthetic functions

		$f(\boldsymbol{x}) = e^{-3x_0+x_1}$	$f(\boldsymbol{x}) = \sin(x_0x_1)$	$f(\boldsymbol{x}) = \frac{x_0x_1}{(x_0^2+x_1)}$	$f(\boldsymbol{x}) = \text{sinc}(x_0^2+x_1)$
Set 1	MSE	0.0004 ± 0.00008	0.0023 ± 0.0004	0.002 ± 0.0006	0.001 ± 0.0002
	R^2	0.998 ± 0.0002	0.947 ± 0.0123	0.762 ± 0.042	0.991 ± 0.002
Set 2	MSE	0.004 ± 0.001	0.007 ± 0.0007	0.004 ± 0.0004	0.018 ± 0.005
	R^2	0.980 ± 0.003	0.840 ± 0.024	0.790 ± 0.028	0.876 ± 0.043

We have also repeated the experiment 3. The result for Yacht dataset is presented in Table 5. Both sets of functions perform reasonably. Set 1 functions approximates the black-box better, however, Set 2 functions have a better performance on the test dataset. The performance of original set of 5 functions and SP method is reported in Table 2.

We also report here the results for Energy dataset (another UCI dataset used in (Crabbe et al., 2020)). It can be seen that there was not a significant difference in most of the results. The performance of original set of 5 functions and SP method is reported below in Table 8.

Table 5: The result of two sets of primitive functions on Yacht dataset

	Method	MLP		SVM	
		MSE	R^2	MSE	R^2
Black Box		0.625 ± 0.153	0.725 ± 0.018	0.674 ± 0.356	0.754 ± 0.059
Method v.s. Black Box	Set 1	0.009 ± 0.005	0.992 ± 0.005	0.033 ± 0.017	0.967 ± 0.012
	Set 2	0.014 ± 0.008	0.989 ± 0.005	0.045 ± 0.018	0.951 ± 0.010
Method	Set 1	0.648 ± 0.170	0.720 ± 0.030	0.557 ± 0.257	0.794 ± 0.037
	Set 2	0.627 ± 0.073	0.728 ± 0.015	0.483 ± 0.253	0.798 ± 0.054

Table 6: The result of two sets of primitive functions on Energy dataset

	Method	MLP		SVM	
		MSE	R^2	MSE	R^2
Black Box		0.016 ± 0.002	0.919 ± 0.013	0.011 ± 0.001	0.945 ± 0.003
Method v.s. Black Box	Set 1	0.003 ± 0.001	0.982 ± 0.009	0.007 ± 0.001	0.961 ± 0.005
	Set 2	0.004 ± 0.002	0.973 ± 0.013	0.007 ± 0.001	0.963 ± 0.009
Method	Set 1	0.019 ± 0.003	0.903 ± 0.020	0.017 ± 0.002	0.914 ± 0.009
	Set 2	0.020 ± 0.002	0.898 ± 0.008	0.016 ± 0.003	0.917 ± 0.012

C.1 Future work: ideas for primitive functions selection:

Here we introduce two ideas for the selection of primitive functions. We leave exploration of usability of these methods for future work.

One possibility is generalizing the SP formulation (Crabbe et al., 2020). As we explained in Section 4, SP method can be regarded as an special case of our framework where the outer functions are chosen from Meijer G-functions and the inner functions are coming from the class of $f(x) = cx$. An immediate generalization is to consider more general inner functions. For instance, we can replace linear functions with polynomial functions of higher degree, or consider a set of classes to choose from.

Another possibility is including polynomials of variable degree among primitive functions. For example, for each function, we can randomly determine the degree by sampling from a decaying distribution. This allows us to occasionally include a polynomial of higher degree (i.e., a Taylor approximation with more accuracy). In order to avoid overfitting we can penalize the fitness criterion of each tree with the degree of the polynomial that it uses.

D Extension to more complex trees

Given a set of primitive functions, we can extend the set of possible metamodels that can be produced by these primitive functions by adding new layers to the tree structure. This extension enables us to create more complex metamodels which may be required for some black-boxes. For instance, in Figure 4, we show a sample tree with four layers. Similarly, if needed, we can add new additional layers (resembling hidden layers) to increase the capacity of the model.

Each hidden layer will have L_i nodes. We produce edges randomly, similar to three-layered trees, we can independently connect two nodes with probability p_0 (or we can have different connection probabilities for different layers). Also, we can similarly define two evolution operations. In the mutation operation, we randomly select some of the edges and either change the corresponding function or delete that edge. For crossover operation, we randomly choose one of the middle nodes, it can be from any of the hidden layers, and change all of its edges with another middle node (in the same layer) of another surviving tree.

The resulting equation of the metamodel is given in (10):

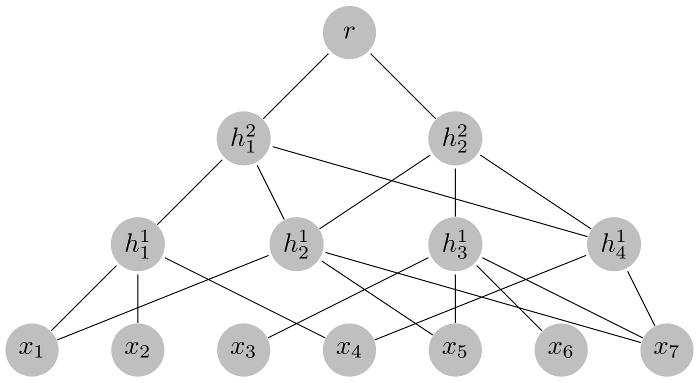

Figure 4: The structure of a sample four layer tree. Adding additional layers can allow us to create more complex metamodels.

$$g(\boldsymbol{x}) = \sum_{i=k}^{L_2} g_{h_k^2}\left(\sum_{i \in \mathcal{N}(h_k^2)} g_{h_i^1}\left(\sum_{j \in \mathcal{N}(h_i^1)} g_{ij}(x_j)\right)\right). \tag{10}$$

E Experiments

In this section we provide the details of our three experiments as well as some additional results.

E.1 Experiment 1

We used the following hyperparameters: $M = k = 20$, recall that M is the population of each generation, and k is the number of gradient updates in each iteration. We choose learning rate to be 0.1 and the maximum number of iterations to be 30. Also, $s = 4$, so in each iteration we choose 4 best trees to populate next generation. Here we have only one input, i.e. $l_1 = l_2 = 1$. Hence, there are exactly three edges in the graph (also we do not remove any edges in the mutation operation), and the GP algorithm in this experiment only searches over various possible ways for assigning classes to edges.

Here we also provide results of an additional experiments similar to first experiment with functions with three variables. We modified the first function in experiment 1 as follows:

$$f(\boldsymbol{x}) = e^{-3x_0 + x_1 + x_2^2}.$$

For this function (with same hyperparameters as before), our model was able to find a metamodel with $R^2 = 0.958 \pm 0.010$. Also, the functional form is correctly chosen:

$$g(\boldsymbol{x}) = 2.82 \exp(-1.10x_0^3 - 0.48x_0^2 - 1.96x_0 - 1.46 \exp(-1.04x_1) + 1.27 \sin(0.97x_2 + 0.05)).$$

Similarly, we added a variable to the second function, and used our model for approximating it.

$$f(\boldsymbol{x}) = \sin(x_0 x_1 + x_2).$$

Again our model correctly identified the functional form with $R^2 = 0.902 \pm 0.018$:

$$g(\boldsymbol{x}) = 1.00 \sin(0.18x_0^3 + 0.31x_0^2 + 0.015x_0 + 0.28x_2^3 + 0.41x_2^2 + 0.47x_2 + 0.50 \sin(0.97x_1 + 0.01) - 0.07).$$

Extension of Experiment 1: Here we consider synthetic functions with more than two variables. We consider functions with 4, 6, and 8 variables, and study how the performance of each model changes when we increase the dimension. Here in Table 7, for brevity we only report results of SP and SR, and exclude SM as it did not perform well even for two variables (as shown in Table 1). We have considered three following sets of functions, similar to Section 5.1, we consider an exponential function, a sin function and a ratio of two functions:

Table 7: Approximating multivariable functions using SM, SP, SR and SMPF.

		SMPF	SP	SP^p	SR
$e_4(\boldsymbol{x})$	MSE	0.119 ± 0.082	1.992 ± 0.430	2.112 ± 0.653	2.632 ± 0.492
	R^2	0.979 ± 0.007	0.649 ± 0.041	0.623 ± 0.039	0.412 ± 0.054
$e_6(\boldsymbol{x})$	MSE	0.174 ± 0.031	0.198 ± 0.044	0.201 ± 0.041	0.209 ± 0.088
	R^2	0.953 ± 0.029	0.432 ± 0.052	0.427 ± 0.050	0.401 ± 0.094
$e_8(\boldsymbol{x})$	MSE	0.009 ± 0.004	0.028 ± 0.011	0.029 ± 0.011	0.033 ± 0.032
	R^2	0.958 ± 0.014	0.498 ± 0.033	0.496 ± 0.033	0.248 ± 0.065
$s_4(\boldsymbol{x})$	MSE	0.003 ± 0.001	0.068 ± 0.020	0.071 ± 0.019	0.010 ± 0.008
	R^2	0.950 ± 0.008	-0.097 ± 0.047	0.101 ± 0.045	0.726 ± 0.010
$s_6(\boldsymbol{x})$	MSE	0.016 ± 0.001	0.022 ± 0.009	0.026 ± 0.009	0.036 ± 0.009
	R^2	0.964 ± 0.002	0.948 ± 0.022	0.923 ± 0.021	0.728 ± 0.022
$s_8(\boldsymbol{x})$	MSE	0.015 ± 0.002	0.018 ± 0.002	0.021 ± 0.002	0.033 ± 0.007
	R^2	0.976 ± 0.006	0.955 ± 0.020	0.901 ± 0.024	0.744 ± 0.033
$r_4(\boldsymbol{x})$	MSE	0.002 ± 0.001	0.001 ± 0.0001	0.001 ± 0.0001	0.010 ± 0.002
	R^2	0.793 ± 0.024	0.883 ± 0.025	0.861 ± 0.026	0.118 ± 0.049
$r_6(\boldsymbol{x})$	MSE	0.001 ± 0.001	0.001 ± 0.0001	0.001 ± 0.0002	0.017 ± 0.002
	R^2	0.723 ± 0.038	0.811 ± 0.022	0.808 ± 0.025	-0.013 ± 0.051
$r_8(\boldsymbol{x})$	MSE	0.004 ± 0.003	0.004 ± 0.002	0.005 ± 0.002	0.014 ± 0.001
	R^2	0.729 ± 0.079	0.779 ± 0.052	0.771 ± 0.055	0.002 ± 0.037

$$e_4(\boldsymbol{x}) = e^{-2x_1+x_2+3x_3-3x_4} \quad e_6(\boldsymbol{x}) = e_4(\boldsymbol{x})e^{-2x_5+x_6} \quad e_8(\boldsymbol{x}) = e_6(\boldsymbol{x})e^{-x_7x_8}$$
$$s_4(\boldsymbol{x}) = \sin(x_1x_2 + x_3 + 2x_4) \quad s_6(\boldsymbol{x}) = \sin(x_1x_2 + x_3 + 2x_4 + 3x_5 - x_6)$$
$$s_8(\boldsymbol{x}) = \sin(x_1x_2 + x_3 + 2x_4 + 3x_5 - x_6 + \sqrt{x_7}x_8) \quad r_4(\boldsymbol{x}) = \frac{x_1x_2}{(x_1^2 + x_2 + x_3^2 + 2x_4)}$$
$$r_6(\boldsymbol{x}) = \frac{x_1x_2}{(x_1^2 + x_2 + x_3^2 + 2x_4 + x_5^3 + x_6)} \quad r_8(\boldsymbol{x}) = \frac{x_1x_2 + \sqrt{x_7}x_8}{(x_1^2 + x_2 + x_3^2 + 2x_4 + x_5^3 + x_6)}$$

It can be seen that SMPF has the most robust results, and adding variables does not significantly deteriorate the performance. We have noted that SP's performance is particularly dependent on the random seed (RS). For example, with the default RS=50, the R^2 score for $e_4(\boldsymbol{x})$ was -0.008, in Table 7 we reported the performance for RS=10 (R^2 $= 0.649$). It can also be seen that while the performance of the model is fine for $s_6(\boldsymbol{x})$ and $s_8(\boldsymbol{x})$, it is unexpectedly low for $s_4(\boldsymbol{x})$ (we believe this is a result of using Meijer G-functions which might be very dependent on the initialization). Also, we have observed that the difference between the training error and test error for SR is significantly large which suggest overfitting.

E.2 Experiment 2

For this experiment we choose the same set of hyperparameters for all three datasets. We had $M = 20$, $s = 2$, and $k = 10$. Number of iterations were 10, and $lr = 0.01$. We had $p_0 = 0.7$ (probability of connecting a middle node to a feature node) and $p_{\text{cross}} = 0.7$ (the probability of choosing crossover operation). Thus, for each survived tree, we apply crossover with probability 0.7 and mutation with probability 0.3. For each mutation operation, with probability 0.5 we deleted a random edge from middle to bottom layer, and with probability of 0.5 we changed hyperparameters of one of the E edges. We chose, $l_1 = l_2 = 2$, therefore, all trees had exactly 2 middle nodes.

E.3 Experiment 3

The hyperparameters for this experiment are as follows. Similar to other experiments we initialized with $M = 20$ trees. We had $s = 4$ survived trees and number of evolution iterations was 20. With $k = 20$ gradient updates, and $lr = 0.05$. In this experiment we had $p_0 = 0.6$. Parameters regarding operations were similar to experiment 2.

Additionally, here we are providing results for Energy dataset (another UCI dataset). Here we also included the result for SR method (SM is not included as its performance is not meaningfully close to these methods). It can be seen that SMPF is performing competitively with SP. In particular, when tested on the real dataset. It should be mentioned that SMPF and SR has the advantage that unlike SP they can produce closed form expression.

Table 8: Interpreting black-boxes trained on Energy dataset

	Method	MLP		SVM	
		MSE	R^2	MSE	R^2
Black Box		0.015 ± 0.002	0.924 ± 0.008	0.014 ± 0.001	0.927 ± 0.003
Method v.s. Black Box	SMPF	0.005 ± 0.002	0.970 ± 0.011	0.007 ± 0.004	0.962 ± 0.023
	SP	0.001 ± 0.001	0.996 ± 0.001	0.004 ± 0.001	0.978 ± 0.001
	SR	0.012 ± 0.012	0.933 ± 0.067	0.011 ± 0.001	0.935 ± 0.003
Method	SMPF	0.019 ± 0.003	0.903 ± 0.010	0.018 ± 0.002	0.908 ± 0.014
	SP	0.020 ± 0.001	0.901 ± 0.006	0.017 ± 0.001	0.914 ± 0.001
	SR	0.026 ± 0.016	0.865 ± 0.067	0.019 ± 0.002	0.905 ± 0.005