Bayesian Symbolic Regression

Ying Jin[1], Weilin Fu[2], Jian Kang[3], Jiadong Guo[4], and Jian Guo[5]

[1]Stanford University, Department of Statistics
[2,4,5]Peng Cheng Laboratory
[3]University of Michigan, Department of Biostatistics

October 28, 2021

Abstract

Interpretability is crucial for machine learning in many scenarios such as quantitative finance, banking, healthcare, etc. Symbolic regression (SR) is a classic interpretable machine learning method by bridging X and Y using mathematical expressions composed of some basic functions. However, the search space of all possible expressions grows exponentially with the length of the expression, making it infeasible for enumeration. Genetic programming (GP) has been traditionally and commonly used in SR to search for the optimal solution, but it suffers from several limitations, e.g. the difficulty in incorporating prior knowledge in GP; overly-complicated output expression and reduced interpretability etc.

To address these issues, we propose a new method to fit SR under a Bayesian framework. Firstly, Bayesian model can naturally incorporate prior knowledge (e.g., preference of basis functions, operators and raw features) to improve the efficiency of fitting SR. Secondly, to improve interpretability of expressions in SR, we aim to capture concise but informative signals. To this end, we assume the expected signal has an additive structure, i.e., a linear combination of several concise expressions, of which complexity is controlled by a well-designed prior distribution. In our setup, each expression is characterized by a symbolic tree, and therefore the proposed SR model could be solved by sampling symbolic trees from the posterior distribution using an efficient Markov chain Monte Carlo (MCMC) algorithm. Finally, compared with GP, the proposed BSR(**B**ayesian **S**ymbolic **R**egression) method doesn't need to keep an updated "genome pool" and so it saves computer memory dramatically.

Numerical experiments show that, compared with GP, the solutions of BSR are closer to the ground truth and the expressions are more concise. Meanwhile we find the solution of BSR is robust to hyper-parameter specifications such as the number of trees in the model.

Introduction

Symbolic regression is a special regression model which assembles different mathematical expressions to discover the association between the response variable and the

predictors, with applications studied in [30], [10], [11], etc. Without a pre-specified model structure, it is challenging to fit symbolic regression, which requires to search for the optimal solution in a large space of mathematical expressions and estimate the corresponding parameters simultaneously.

Traditionally, symbolic regression is solved by combinatorial optimization methods like Genetic Programming (GP) that evolves over generations, see [29], [9], [6], [28], etc. However, GP suffers from high computational complexity and overly complicated output expressions, and the solution is sensitive to the initial value, see [20]. Some modifications of the original GP algorithm have been proposed to address those problems including [1] which incorporates statistical information of generations, [23] which deterministically builds higher-level expressions from 'elite' building blocks, [16] which employs a hybrid of GP and deterministic methods, [26] which exploits the Pareto front of GP to balance accuracy and simplicity, [22] uses a divide and conquer strategy to decompose the search space and reduce the model complexity, and [19] which proposes a local optimization method to control the complexity of symbolic regression.

Although some efforts have been made to improve GP, its intrinsic disadvantages still remain unsolved. Some research work explores SR estimation methods other than GP. For example, [12] which introduces a new data structure called Interaction-Transformation to constrain the search space and simplify the output symbolic expression, [23] which uses pathwise regularized learning to rapidly prune a huge set of candidate basis functions down to compact models, [3] assumes regression models are spanned by a number of elite bases selected and updated by their proposed algorithm, [2] introduces a neuro-encoded expression programming with recurrent neural networks to improve smoothness and stability of the search space, [21]which introduces an expression generating neural network and proposes an Monte Carlo tree search algorithm to produce expressions that match given leading powers.

In this work, we consider to fit symbolic regression under a Bayesian framework, which can naturally incorporate prior knowledge, can improve model interpretability and can potentially simplify the structure and find prominent components of complicated signals. The key idea is to represent each mathematical expression as a symbolic tree, where each child node denotes one input value and the parent node denotes the output value of applying the mathematical operator to all the input values from its child nodes. To control model complexity, the response variable y is assumed to be a linear combination of multiple parent nodes whose descendant nodes (or leaf nodes) are the predictor `x`. We develop a prior model for the tree structures and assign informative priors to the associated parameters. Markov chain Monte Carlo (MCMC) methods are employed to simulate the posterior distributions of the underlying tree structures which correspond to a combination of multiple mathematical expressions.

The paper is organized as follows. First, we present our Bayesian symbolic regression model by introducing the tree representation of mathematical expressions. Then we develop an MCMC-based posterior computation algorithm for the proposed model. Finally, we demonstrate the superiority of the proposed method compared to existing alternatives via numerical experiments.

In the following parts, we will refer to our symbolic regression method based on Bayesian framework as Bayesian Symbolic Regression or BSR in exchange.

Bayesian Symbolic Regression with Linearly-Mixed Tree Representations

Denote by $\mathbf{x} = (x_1, \ldots, x_d) \in \mathbb{R}^d$ the predictor variables and by $y \in \mathbb{R}$ the response variable. We consider a symbolic regression model:

$$y = g(\mathbf{x}) + \epsilon,$$

where $g(\cdot)$ is a function represented by a combination of mathematical expressions taking predictors $\mathbf{x}$ as the input variable. Specifically, the mathematical operators such as $+$, $\times$, ..., and arithmetic functions like $\exp(\cdot), \cos(\cdot), \ldots$, can be in the search space of mathematical expressions. For example, $g(\mathbf{x}) = x_1 + 2\cos(x_2) + \exp(x_3) + 0.1$.

Choice of Basic Operators

All possible mathematical expressions are combinations of elements in a set of basic functions. The choice of basic operators is a building block of our tree representation, see [25]. In this paper, we adopt the commonly-used operators $+$, $\times$, `exp()`, $\texttt{inv}(x) = 1/x$, $\texttt{neg}(x) = -x$ and linear transformation $\texttt{lt}(x) = ax + b$ with parameters $(a, b) \in \mathbb{R}^2$. They are able to express $-$ and $\div$ with symmetric binary operators. In practice, the basic operators can be specified by users.

From Expressions to Trees

The mathematical expression can be equivalently represented by a tree denoted by T, with non-terminal nodes indicating operations and terminal nodes indicating the selected features. T is a binary tree but not necessarily a complete tree.

Specifically, a non-terminal node has one child node if it is assigned a unary operator, and two if assigned a binary operator. For example, a non-terminal node with operator $+$ represents the operation that the values of its two child nodes are added up. For a non-terminal unary operator, for example `exp()`, it means taking exponential of the value of its child node. Note that some operators may also be associated with parameters, like linear transformation $\texttt{lt}(x) = ax + b$ with parameters $(a, b) \in \mathbb{R}^2$. We collect these parameters in a vector Θ.

On the other hand, each terminal node η specified by $i_k \in M$ represents a particular feature x_{i_k} of the data vector. Here M is the vector including features of all terminal nodes. For a tree of depth d, we start from the terminal nodes by performing the operations indicated by their parents, then go to their parents and perform upper-level operations accordingly. We obtain the output at the root node. For example, the tree in Figure 1 represents $g(\mathbf{x}) = \cos(x_1 + x_2)$, which consists of two terminal nodes $1, 2$ and two non-terminal nodes $\cos, +$.

In short, the tree structure T is the set of nodes $T = (\eta_1, \ldots, \eta_t)$, corresponding to operators with zero to two child nodes. Some operators involve parameters aggregated in Θ. From predictor $\mathbf{x}$, terminal nodes select features specified by $M = (i_1, \ldots, i_p)$, where i_k indicates adopting x_{i_k} of vector $\mathbf{x}$ as the input of the corresponding node η_k. The specification of T, Θ and M represents an equivalent tree for a mathematical expression $g(\cdot; T, M, \Theta)$.

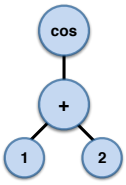

Figure 1: Tree representation of $\cos(x_1 + x_2)$

Priors on Tree Representations

Under a Bayesian modeling framework, it is critical to specify appropriate priors for parameters, as it has the flexibility to incorporate prior knowledge to facilitate more accurate posterior inferences. In our model, we are interested in making inferences on the tree structure T, the parameter Θ and the selected feature indices M.

To ensure the model interpretability, we aim to control the size of tree representations, or equivalently, the complexity of mathematical expressions. The default prior of operators and features are uniform distributions, indicating no preference for any particular operator or feature. They can be user-specified weight vectors to pose preferences.

For a single tree, we adopt prior distributions on T, M and Θ in a similar fashion as those for Bayesian regression tree models in [7] as follows. Of note, although the prior models are similar, our model and tree interpretations are completely different from the Bayesian regression tree model.

Prior of Tree Structure T

We specify the prior $p(T)$ by assigning the probabilities to each event in the process of constructing a specific tree. The prior construction starts from the root node.

A node is randomly assigned a particular operator according to the prior. The operator indicates whether it extends to one child node, or split into two child nodes, or function as a terminal node. Starting from the root, such growth performs recursively on newly-generated nodes until all nodes are assigned operators or terminated.

Specifically, for a node with depth d_η, i.e. the number of nodes passed from it to the node, with probability $p_1(\eta, T) = \alpha(1 + d_\eta)^{-\beta}$. It is a non-terminal node, which means it has descendants. Here α, β are prefixed parameters that guides the general sizes of trees in practice. The prior also includes a user-specified basic operator set and a corresponding weight vector indicating the probabilities of adopting each operator for a newly-grown node. For example, we specify the operator set (operator) as Ops= (`exp()`, `lt()`, `inv()`, `neg()`, +, $\times$) where $\texttt{lt}(x) = ax + b$, $\texttt{inv}(x) = 1/x$, $\texttt{neg}(x) = -x$, and the uniform weight vector $w_{op} = (1/6, 1/6, 1/6, 1/6, 1/6, 1/6)$. Such default choice shows no preference for any particular operator.

With probability $p_1(\eta, T)$, the node η is assigned an operator according to w_{op} if it is non-terminal and grows its one or two child nodes. Then its child nodes grow recursively. Otherwise it is a terminal node and assigned some feature in a way specified later. The construction of a tree is completed if all nodes are assigned or terminated.

Prior of Terminal Nodes M

When a node is terminated, it is assigned a feature of $\mathbf{x}$ according to the prior of features as input of the expression. The number and locations of terminal nodes are decided by structure of T. Conditioned on T, the specific feature that one terminal node takes is randomly generated with probabilities indicated by weight vector w_{ft}. The default choice is uniform among all features, i.e., $w_{ft} = (1/d, \ldots, 1/d)$. It can also be user-specified to highlight some preferred features.

Prior of `lt()` Parameters

An important operator we adopt here is linear transformation $\texttt{lt}(x) = ax + b$ associated with linear parameters $(a, b) \in \mathbb{R}^2$. `lt()` includes scalings and well enriches the set of potential expressions. Such operation is discussed in [17] and proved to improve the fitting. Pairs of linear parameters (a, b) are assembled in Θ and are considered independent.

Let $L(T)$ be the set of `lt()` nodes in T, and each node η is associated with parameters (a_η, b_η), then the prior of Θ is

$$p(\Theta \mid T) = \prod_{\eta \in L(T)} p(a_\eta, b_\eta),$$

where a_η's, b_η's are independent and

$$a_\eta \sim N(1, \sigma_a^2), \quad b_\eta \sim N(0, \sigma_b^2).$$

This indicates that the prior of the linear transformation is a Gaussian and centered around identity function. The prior of $\sigma_\Theta = (\sigma_a, \sigma_b)$ is conjugate prior of normal distribution, which is

$$\sigma_a^2 \sim IG(\nu_a/2, \nu_a\lambda_a/2), \quad \sigma_b^2 \sim IG(\nu_b/2, \nu_b\lambda_b/2),$$

where ν_a, λ_a, ν_b, λ_b are pre-specified hyper-parameters.

Find the Signal: Linear Mixture of Simpler Trees

Many popular machine learning techniques, such as neural networks, can approximate functions very well, but they are difficult to interpret. A widely celebrated advantage of symbolic regression is its interpretability and good performance of approximating functions. The model fitting of symbolic regression usually results in relatively simple mathematical expressions, it is straightforward to understand the relationship between the predictors $\mathbf{x}$ and the response variable y.

However, if symbolic regression produces too complicated expressions, the interpretation of the model fitting becomes challenging: there exists a tradeoff between simplicity and accuracy. To highlight the superiority of symbolic regression in interpretability over other methods, we aim at finding the most prominent and concise signals. If the features are strong and expressive, we assume that the expression should not involve too many features, and the transformation should not be too complicated.

Moreover, the real-world signal may be a combination of simple signals, where only a small amount of simpler ones play a significant role. A simpler idea has its roots in [18], where the output is appropriately scaled. SR has also been addressed with methods related to generalized linear models, summarized in [31].

In this sense, we model the final output y to be centered at some linear combination of relatively simple expressions

$$y = \beta_0 + \sum_{i=1}^{k} \beta_i \cdot g(\mathbf{x}; T_i, M_i, \Theta_i) + \epsilon, \quad \epsilon \sim N(0, \sigma^2)$$

where k is a pre-specified number of simple components, $g(\mathbf{x}; T_i, M_i, \Theta_i)$ is a relatively simple expression represented by a symbolic tree, and β_i is the linear coefficient for the i-th expression. The coefficients β_i, $i = 0, \ldots, k$ is obtained by OLS linear regression using intercept and $g(\cdot; T_i, M_i, \Theta_i), i = 1, \ldots, k$. Let $\{(T_i, M_i, \Theta_i)\}_{i=1}^k$ denote the series of tuples (T_i, M_i, Θ_i), $i = 1, \ldots, k$. Let $OLS()$ denote the OLS fitting result, then a simpler form is

$$y = OLS\big(x, \{(T_i, M_i, \Theta_i)\}_{i=1}^k\big) + \epsilon, \quad \epsilon \sim N(0, \sigma^2)$$

where the prior of the noise scale is the conjugate inverse gamma distribution

$$\sigma^2 \sim IG(\nu/2, \nu\lambda/2)$$

where ν and λ are pre-specified parameters. Additionally let $(T, M, \Theta) = \{(T_i, M_i, \Theta_i)\}_{i=1}^k$, the joint likelihood is

$$\begin{aligned}
&p(y, (T, M, \Theta), \sigma, \sigma_\Theta \mid x) \\
=&p(y \mid OLS\big(x, T, M, \Theta\big), \sigma^2) p(M, T) p(\Theta \mid T, \sigma_\Theta^2) p(\sigma_\Theta^2) p(\sigma^2) \\
=&p(y \mid OLS\big(x, T, M, \Theta\big), \sigma^2) p(\sigma^2) \\
&\times \prod_{i=1}^{k} p(M_i \mid T_i) p(T_i) p(\Theta_i \mid T_i, \sigma_\Theta^2)
\end{aligned}$$

Posterior Computation

We employ the Metropolis-Hastings (MH) algorithm proposed in [24] and [15] to make posterior inferences on the proposed model. Note that (T, M, Θ) represents the set of k trees $\{T_i, M_i, \Theta_i\}_{i=1}^k$, and (T^s, M^s, Θ^s) denotes the set of k trees that the MH algorithm accepts at the s-th iteration.

With a pre-specified number of trees k, our method modifies the structure of the i-th tree by sampling from proposal $q(\cdot \mid \cdot)$, and accepts the new structure with probability α, which can be calculated according to MH algorithm. Otherwise the i-th tree stays at its original form. The k trees are updated sequentially, so to illustrate, we first show how a single tree is modified at each time.

The sampling of a new tree consists of three parts. The first is the structure specified by T and M, which is discrete. In this subsection, T and M stand for a single tree. The second part is Θ aggregating parameters of all `lt()` nodes. The dimensionality

of Θ may change with (T, M) since the number of linear transformation nodes vary among different tree structures. Therefore we use the reversible jump MCMC (RJMCMC) algorithm proposed by [13] to solve the trans-dimensional sampling problem. For simplicity, denote by $S = (T, M)$ the structure parameters. The last part is the sampling of noise variance σ^2 from an inverse gamma prior.

Structure Transition Kernel

We first specify how the sampling algorithm jumps from a tree structure to a new one. Inspired by [7] and considering the nature of calculation trees, we design the following seven reversible actions. The probabilities from $S = (T, M)$ to new structure $S^* = (T^*, M^*)$ is denoted as the proposal $q(S^* \mid S)$.

- **Stay:** If the expression involves $n_l \geq 0$ `lt()` operators, with probability $p_0 = n_l/4(n_l + 3)$, the structure $S = (T, M)$ stays unchanged, and ordinary MH step follows to sample new linear parameters.

- **Grow:** Uniformly pick a terminal node and activate it. A sub-tree is then generated iteratively, where each time a node is randomly terminated or assigned an operator according to the prior until all nodes are terminated or assigned.

  To regularize the complexity of the expression, the proposal grows with lower probability when the tree depth and amount of nodes are large. The probability of Grow is $p_g = \frac{1-p_0}{3} \cdot \min\left\{1, \frac{8}{N_{nt}+2}\right\}$,where N_{nt} is the number of non-terminal nodes.

- **Prune:** Uniformly pick a non-terminal node and turn it into a terminal node by discarding its descendants. Then randomly choose a feature of $\mathbf{x}$ to the newly pruned node.

  We set the probability of Prune as $p_p = \frac{1-p_0}{3} - p_g$ such that Grow and Prune share one-third of the probability that the structure does not Stay.

- **Delete:** Uniformly pick a candidate node and delete it.

  Specifically, the candidate should be non-terminal. Also, if it is a root node, it needs to have at least one non-terminal child node to avoid leaving a terminal node as the root node. If the picked candidate is unary, then we just let its child replace it. If it is binary but not root, we uniformly select one of its children to replace it. If the picked candidate is binary and the root, we uniformly select one of its non-terminal children to replace it.

  We set the probability of Delete as $p_d = \frac{1-p_0}{3} \cdot \frac{N_c}{N_c+3}$, where N_c is the number of aforementioned candidates.

- **Insert:** Uniformly pick a node and insert a node between it and its parent. The weight of nodes assigned is w_{op}. If the inserted node is binary, the picked node is set as left child of the new node, and the new right child is generated according to the prior.

  The probability of Insert is set as $p_i = \frac{1-p_0}{3} - p_d$ such that Delete and Insert share one-third of the probability that the structure does not Stay.

- **ReassignOperator:** Uniformly pick a non-terminal node, and assign a new operator according to w_{op}.

  If the node changes from unary to binary, its original child is taken as the left child, and we grow a new sub-tree as right child. If the node changes from binary to unary, we preserve the left sub-tree (this is to make the transition reversible).

- **ReassignFeature:** Uniformly pick a terminal node and assign another feature with weight w_{ft}.

  The probability of ReassignOperator and ReassignFeature is set as $p_{ro} = p_{rf} = \frac{1-p_0}{6}$

Note that the generation of the 'tree' is top-down, creating sub-trees from nodes. However, the calculation is bottom-up, corresponding to transforming the original features and combine different sources of information.

The above discrepancy can be alleviated by our design of proposal. **Grow** and **Prune** creates and deletes sub-trees in a top-down way, which corresponds to changing a "block", or a higher level feature represented by the sub-tree in the expression. On the other hand, **Delete** and **Insert** modify the higher-level structure by changing the way such "blocks" combine and interact in a bottom-up way.

Jump between Spaces of Parameters

Another issue of proposing new structure S^* is that the number of linear transformation nodes may change. Therefore the dimensionality of Θ may be different and RJMCMC (reversible jump Markov Chain Monte Carlo) proposed in [13] settles the problem well.

After we generate S^* from S, there are three situations.

- **No Change.** When the new structure does not change the number of `lt()` nodes, the dimensionality of parameters does not change. In this case, it is sufficient to use ordinary MH step. Here the set of `lt()` nodes may change, but the sampling of new parameters is i.i.d., so we are satisfied with the MH step.

- **Expansion.** When the number of `lt()` nodes increases, the dimensionality of Θ, denoted by p_Θ, increases. We may simultaneously lose some original `lt()` nodes and have more new ones. But due to the i.i.d. nature of parameters we only consider the number of all `lt()` nodes.

  Denote the new parameter as Θ^*. According to RJMCMC, we sample auxiliary variables $U = (u_\Theta, u_n)$ where $dim(u_\Theta) = dim(\Theta)$, $dim(u_n) + dim(\Theta) = dim(\Theta^*)$.

  The hyper-parameters $U_\sigma = (\sigma_a^2, \sigma_b^2)$ are independently sampled from the inverse gamma prior, then each element of u_Θ and u_n is independently sampled from $N(1, \sigma_a^2)$ or $N(0, \sigma_b^2)$ accordingly. The new parameter Θ^* along with new auxiliary variable U^* is obtained by

$$\begin{aligned}(U^*, \Theta^*, \sigma_\Theta^*) &= j_e(\Theta, U, U_\sigma) = j_e(\Theta, u_\Theta, u_n, U_\sigma) \\ &= \Big(\frac{\Theta - u_\Theta}{2}, \frac{\Theta + u_\Theta}{2}, u_n, U_\sigma\Big),\end{aligned}$$

where

$$U^* = \frac{\Theta - u_\Theta}{2}, \quad \Theta^* = (\frac{\Theta + u_\Theta}{2}, u_n), \quad \sigma^*_\Theta = U_\sigma.$$

Then we discard U^* and get $\Theta^*, \sigma^*_\Theta$.

- **Shrinkage.** Θ shrinks when the number of `lt()` nodes decreases. Similar to the Expansion case, we may lose some `lt()` nodes and also have new ones (especially in the ReassignOperator transition), but only the dimensionality is of interest. Assume that the original parameter is $\Theta = (\Theta_0, \Theta_d)$ where Θ_d corresponds to the parameters of nodes to be dropped. Denote the new parameter as Θ^*.

  Firstly, $U_\sigma = (\sigma_a^2, \sigma_b^2)$ are sampled independently from the inverse gamma prior. The new parameter candidate is then obtained by first sampling U, whose elements are independently sampled from $N(0, \sigma_a^2)$ and $N(0, \sigma_b^2)$, respectively, with $dim(U) = dim(\Theta_0)$. Then the new candidate Θ^* as well as the corresponding auxiliary variable U^* is obtained by

$$\begin{aligned}(\sigma^*_\Theta, \Theta^*, U^*) &= j_s(U_\sigma, U, \Theta) = j_s(U_\sigma, U, \Theta_0, \Theta_d) \\ &= (U_\sigma, \Theta_0 + U, \Theta_0 - U, \Theta_d),\end{aligned}$$

  where

$$\sigma^*_\Theta = U_\sigma, \quad \Theta^* = \Theta_0 + U, \quad U^* = (\Theta_0 - U, \Theta_d).$$

  Then we just discard U^* and get U^*, σ^*_Θ.

For simplicity, we denote the two transformation j_e, j_s as j_{S,S^*}, indicating that this is a transformation from parameters of S to those of S^*. The auxiliary variables are denoted as U and U^* respectively. Note that $dim(\Theta) + dim(U) = dim(\Theta^*) + dim(U^*)$ in both cases.

Accepting New Candidates

Return to the K-tree case. We sequentially update the K trees in a way similar to [8] and [14]. Suppose we start from tree $(T_j^{(t)}, M_j^{(t)}, \Theta_j^{(t)})$, that is, the j-th tree of the t-th accepted model, and that the newly proposed structure is $(T_j^*, M_j^*, \Theta_j^*)$. Denote

$$\begin{aligned}(T^{(t)}, M^{(t)}, \Theta^{(t)}) &= \{(T_i^{(t)}, M_i^{(t)}, \Theta_i^{(t)})\}_{i=1}^k, \\ (T^*, M^*, \Theta^*) &= \{(T_i^*, M_i^*, \Theta_i^*)\}_{i=1}^k,\end{aligned}$$

where $(T_i^*, M_i^*, \Theta_i^*) = (T_i^{(t)}, M_i^{(t)}, \Theta_i^{(t)})$ for $i \neq j$. Also let $S^* = (T^*, M^*)$, $S^{(t)} = (T^{(t)}, M^{(t)})$. And $(\sigma^*)^2$ is the newly-sampled version of $(\sigma^{(t)})^2$. For simplicity, let $\Sigma^{(t)} = \big((\sigma^{(t)})^2, \sigma^{(t)}_\Theta\big)$ and $\Sigma^* = \big((\sigma^*)^2, \sigma^*_\Theta\big)$.

If $dim(\Theta_i^{(t)}) = dim(\Theta^*)$, the ordinary MH step gives the acceptance rate

$$R = \frac{f(y \mid OLS(x, S^*, \Theta^*), \Sigma^*) f(S^*) q(S^{(t)} \mid S^*)}{f(y \mid OLS(x, S^{(t)}, \Theta^{(t)}), \Sigma^{(t)}) f(S^{(t)}) q(S^* \mid S^{(t)})}. \tag{1}$$

If $dim(\Theta_i^{(t)}) \neq dim(\Theta^*)$, the RJMCMC method gives the acceptance rate

$$
\begin{aligned}
R = & \frac{f(y \mid OLS(x, S^*, \Theta^*), \Sigma^*) f(\Theta^* \mid S^*) q(S^{(t)} \mid S^*)}{f(y \mid OLS(x, S^{(t)}, \Theta^{(t)}), \Sigma^{(t)}) f(\Theta^{(t)} \mid S^{(t)}) q(S^* \mid S^{(t)})} \\
& \cdot \frac{f(S^*) p(\Sigma^*) h(U^* \mid \Theta^*, S^*, S^{(t)})}{f(S^{(t)}) p(\Sigma^{(t)}) h(U^{(t)} \mid \Theta^{(t)}, S^{(t)}, S^*)} \\
& \cdot \left| \frac{\partial j_{S^{(t)}, S^*}(\Theta^{(t)}, U^{(t)})}{\partial(\Theta^{(t)}, U^{(t)})} \right|
\end{aligned} \tag{2}
$$

In each case, we accept the new candidate with probability $\alpha = \min\{1, R\}$ with R in Equation (1) or (2). If the new candidate is accepted, we next update the $(j+1)$-th tree starting from $(T^{(t+1)}, M^{(t+1)}, \Theta^{(t+1)}) = (T^*, M^*, \Theta^*)$ and $\Sigma^{(t+1)} = \Sigma^*$. Otherwise we update the $(j+1)$-th tree starting at $(T^{(t)}, M^{(t)}, \Theta^{(t)})$ with $\Sigma^{(t)}$.

Pseudo-codes of the Algorithm

We sum up the algorithm as follows.

Algorithm 1 pseudo-codes of MCMC-based Symbolic Regression for linearly-mixed signals

Input: Datapoints $x_1, \ldots, x_n$, labels $y = (y_1, \ldots, y_n)$; number of components K, number of acceptance N; transition kernel (proposal) $q(\cdot \mid \cdot)$, prior distributions $p(T, M, \Theta)$, likelihood function $f(y \mid OLS(S, \Theta, x))$;

Output: A chain of accepted models $(T^{(t)}, M^{(t)}, \Theta^{(t)})$;

From prior $p(T, M, \Theta)$, generate independently K tree models (structures and parameters) $(T_i^{(1)}, M_i^{(1)}, \Theta_i^{(1)}), i = 1, \ldots, K$;
Calculate linear regression coefficients $\beta^{(1)}$ from datapoints x_i, labels y_i and models $(T_i^{(1)}, M_i^{(1)}, \Theta_i^{(1)})$, $i = 1, \ldots, n$ using OLS;
Number of accepted models $m = 1$;
while $m < N$ **do**
    for $i = 1 \to K$ **do**
        Propose $S_i^* = (T_i^*, M_i^*)$ by sampling $S_i^* \mid S_i^{(m)} \sim q(\cdot; S_i^{(m)})$;
        if $dim(\Theta_i^*) \neq dim(\Theta_i^{(m)})$ **then**
            Sample $U_i^{(m)} \sim h(U_i^{(m)} \mid \Theta_i^{(m)}, S_i^{(m)}, S_i^*)$;
            Obtain $(U_i^*, \Theta_i^*) = j_{S_i^{(m)}, S_i^*}(\Theta_i^{(m)}, U_i^{(m)})$;
            Calculate linear regression coefficients β^* from datapoints x_i, labels y_i and models (T^*, M^*, Θ^*) using OLS;
            Calculate the ratio R in Equation (2);
        else
            Directly sample $\Theta_i^* \sim p(\cdot \mid S_i^*)$;
            Calculate coefficients β^* from $x_i, y_i, i = 1, \ldots, n$ and models $(T^{(m)}, M^{(m)}, \Theta^{(m)})$ using OLS;
            Calculate the ratio R in Equation (1);
        end if
        $\alpha \leftarrow \min(1, R)$;
        Sample $u \sim U(0, 1)$;
        if $u < \alpha$ **then**
            for $j = 1 \to K$ **do**
                if $j = i$ **then**
                    $S_j^{(m+1)} \leftarrow S_j^*, \Theta_j^{(m+1)} \leftarrow \Theta_j^*$;

else
$S_j^{(m+1)} \leftarrow S_j^{(m)}, \Theta_j^{(m+1)} \leftarrow \Theta_j^{(m)}$;
end if
end for
$\beta^{(m+1)} \leftarrow \beta^*$;
$m \leftarrow m + 1$;
end if
end for
end while

Experiments

The experimental results of GP and BSR methods are presented here. Firstly, we compare their generalization and domain adaptation ability by comparing RMSEs on both training and testing data. Secondly, we compare the complexity of the mathematical expressions generated by these two methods. Finally, we examine the robustness of the proposed BSR method by testing whether the estimated model is sensitive to the parameter K, which is the number of trees used in the linear regression.

Experiment Designs

Benchmark Problems

We set up a benchmark mathematical expression sets with six tasks presented from Equations (3) to (8). These task formulas borrow the ideas from [5],[27] and [4], and are used to evaluate the proposed model. The following task is to estimate a specific BSR model on each of the formulas. As we have mentioned, these problems have been widely used to test other symbolic regression methods, including those based on GP, and so it is convenient for us to compare BSR with GP directly.

$$f_1(x_0, x_1) = 2.5x_0^4 - 1.3x_0^3 + 0.5x_1^2 - 1.7x_1 \quad (3)$$

$$f_2(x_0, x_1) = 8x_0^2 + 8x_1^3 - 15 \quad (4)$$

$$f_3(x_0, x_1) = 0.2x_0^3 + 0.5x_1^3 - 1.2x_1 - 0.5x_0 \quad (5)$$

$$f_4(x_0, x_1) = 1.5\exp(x_0) + 5\cos(x_1) \quad (6)$$

$$f_5(x_0, x_1) = 6.0\sin(x_0)\cos(x_1) \quad (7)$$

$$f_6(x_0, x_1) = 1.35x_0x_1 + 5.5\sin((x_0 - 1)(x_1 - 1)) \quad (8)$$

Datasets

We conduct experiments on simulated data. The experiment designed and used in [5] is adopted here. For each task (corresponding to an expression), we have one training data set and three testing data sets. The training set consists of 100 samples. For each sample, its predictor variables are generated independently from uniform distributions in the interval $[-3, 3]$, and the response variable is generated by the corresponding

expression above. We consider three different testing sets, all with size of 30. Following the same way to generate training set, the three testing sets are generated within intervals $[-3, 3]$, $[-6, 6]$ and $[3, 6]$.

Parameter Settings

The detailed settings of the two methods are introduced as follows. Note that the GP algorithm consists of two nested iterations, the inner loop for population and the outer loop for generation. Therefore, the number of trees generated by GP is counted as $N_g \times N_p$, where N_g is the number of generations and N_p is the population size. We set $N_g = 200$ and $N_p = 100$ in this study, and therefore a total of 200,000 trees are generated. Meanwhile, the number of trees generated by BSR is set as 100,000. In addition, we specify two additive components of BSR are used for all tasks in our study. The basis function pool is set as $\{+, -, \times, \div, \sin, \cos, \exp, x^2, x^3\}$ for both BSR and GP method. In order to see the stability of their performances, we run the two methods in each task for 50 times independently.

Experiment Results

Accuracy and Generalization Abilities

We use root mean square error (RMSE) on training set to measure how good the model fits the data, and use RMSE on test set to examine the generalization ability. The performance of two methods in all task are summarized in Table 1. This table records the mean and standard deviation of RMSEs over 50 simulation replications for each task. It turns out that BSR outperforms GP in most tasks, except the task defined in equation (8). A plausible reason is that the structure of function (8) is far from linear structure, which is one of the key assumptions of BSR.

Complexity of Expressions

One of the most important aim we propose BSR is to improve interpretability of symbolic regression model by restricting the symbolic expression in a concise and readable form. Specifically, we introduce an additive symbolic tree structure for BSR model.

To check if BSR has advantage in interpretability, we summarize the complexity of the output from BSR and GP in Table 2, where the numbers are the means and standard deviations of the output tree sizes (number of nodes in each tree) out of 50 replications.

According to Table 2, the number of nodes on trees generated by BSR is significantly less that those generated by GP, leading to more concise and readable expressions. Table 3 lists some typical expressions output from BSR and GP. It turns out that expressions estimated by BSR are generally closer to the ground truth and they are shorter and more comprehensible. The simulation study here verifies that, in favourable scenarios, BSR reaches its aim and shows its advantage in both prediction accuracy and interpretability.

Table 1: RMSEs of Both Methods

Task	Dataset	**RMSEs**($mean \pm std$)	
		BSR	GP
f_1	train[-3,3]	2.00 ± 3.87	2.71 ± 2.43
	test[-3,3]	2.04 ± 3.27	4.25 ± 4.59
	test[-6,6]	92.09 ± 258.54	116.29 ± 97.59
	test[3,6]	118.53 ± 311.57	203.31 ± 168.34
f_2	train[-3,3]	7.30 ± 10.19	3.56 ± 5.79
	test[-3,3]	6.84 ± 10.10	2.92 ± 4.41
	test[-6,6]	95.33 ± 145.31	121.41 ± 126.19
	test[3,6]	128.27 ± 221.73	174.01 ± 173.71
f_3	train[-3,3]	0.19 ± 0.16	0.63 ± 0.33
	test[-3,3]	0.21 ± 0.20	0.60 ± 0.35
	test[-6,6]	9.38 ± 9.08	28.97 ± 20.68
	test[3,6]	15.19 ± 32.24	34.08 ± 25.41
f_4	train[-3,3]	0.14 ± 0.56	0.72 ± 1.01
	test[-3,3]	0.16 ± 0.62	0.84 ± 1.12
	test[-6,6]	6.96 ± 19.44	24.62 ± 29.66
	test[3,6]	12.06 ± 38.27	31.74 ± 36.77
f_5	train[-3,3]	0.68 ± 1.14	0.78 ± 0.96
	test[-3,3]	0.66 ± 1.13	0.72 ± 0.83
	test[-6,6]	1.09 ± 2.39	1.58 ± 1.55
	test[3,6]	1.41 ± 3.57	4.49 ± 5.07
f_6	train[-3,3]	3.99 ± 0.71	3.17 ± 0.79
	test[-3,3]	4.63 ± 0.62	3.70 ± 0.93
	test[-6,6]	12.22 ± 8.46	5.13 ± 1.91
	test[3,6]	14.44 ± 10.39	11.09 ± 12.58

Table 2: Complexity of Expressions

Task	**Number of Nodes**($mean \pm std$)	
	BSR	GP
f_1	$\mathbf{22.16 \pm 7.44}$	40.85 ± 21.34
f_2	$\mathbf{12.25 \pm 11.41}$	54.51 ± 38.89
f_3	27.23 ± 10.61	22.88 ± 8.62
f_4	$\mathbf{13.64 \pm 12.50}$	22.80 ± 8.82
f_5	31.28 ± 9.13	19.80 ± 10.28
f_6	$\mathbf{20.08 \pm 4.78}$	21.18 ± 25.73

Sensitivity to the Number of Components K

The number of additive components K is an important hyper-parameter in BSR model and it is interesting to study if the optimal expression selected by BSR is sensitive to the choice of K. To check this, we summarize the average RMSEs on testing set $[-3, 3]$ out of 50 replications in Table 4.

From the results we see that the RMSEs of these tasks tend to be smaller as K grows, but the improvement of performance is not significant when K is too large. To our surprise, experiments show that even if K is set to be smaller than what it should be (ground truth), BSR can automatically find an approximately equivalent additive component structure in some single trees. On the other hand, when K is significantly larger than what it should be, BSR automatically "discards" the redundant trees by estimating the non-informative trees as insignificant components, making them similar

Table 3: Typical Expressions

Task		Expressions
f_1	Truth	$f_1 = 2.5x_0^4 - 1.3x_0^3 + 0.5x_1^2 - 1.7x_1$
	GP	$y = ((exp((\frac{-x_0}{0.80} + 0.81)) - (((\sin((0.80x_0)^2)$ $- \cos(x_1)^6) + \sin((0.80x_0)^2)) + \cos(x_1)))$ $-(((\frac{x_0}{-0.80}) + ((((\frac{x_0}{-0.80})$ $+ \cos(x_1)) + ((\sin((0.71x_0)^2)$ $-((\sin(((0.71x_0))^2) - 0.77))^2) + 1.0)) + x_1))$ $+(0.76 + x_1))) + (((\frac{x_0^2}{0.78}))^2 + \frac{x_0^2}{0.80})$
	BSR	$y = (-0.02) + (-1.30)[x_0^3 + 1.30x_1 + 0.09]$ $+(0.49)[5.05x_0^4 + x_1^2 + 0.31]$
f_2	Truth	$f_2 = 8x_0^2 + 8x_1^3 - 15$
	GP	$y = (exp(1.82)x_1^3) + 5.26(x_0^2 - (\cos((0.90x_0))$ $*(exp(0.187) + \cos((x_0^2 \cos(0.75))))))$ $+(x_1 - 0.77)^3 + exp(x_1 - 0.38)(x_1 - 0.38)$
	BSR	$y = (-0.02) + (-1.38)[-7.56x_0^2 + 2.85]$ $+(8.00)[-0.30x_0^2 + x_1^3 - 1.38]$
f_3	Truth	$f_3 = 0.2x_0^3 + 0.5x_1^3 - 1.2x_1 - 0.5x_0$
	GP	$y = (4x_1 - \sin(1.32x_1) - 0.69$ $-(\sin(\sin(1.32x_1)/0.50)/0.76))$ $- \sin(x_0) - \sin(\sin(\sin((\cos(x_1) + x_1))))$
	BSR	$y = (0.04) + (-0.30)[-0.67x_0^3 + 4.27]$ $+(-0.21)[-2.45x_1^3 + 2.45x_0 + x_1 - 0.93]$
f_4	Truth	$f_4 = 1.5\exp(x_0) + 5\cos(x_1)$
	GP	$y = (((((exp(\cos(x_0)) + 0.59 + x_0)$ $+exp(x_0)) - \cos(exp(\cos(x_1))))$ $- \cos(exp(\cos(\sin(x_1)x_0))))$ $-x_1^2 + x_0^2)$
	BSR	$y = (-0.01) + (0.28)[17.74\cos(x_1) + 0.45]$ $+(0.24)[6.26exp(x_0) - 0.47]$
f_5	Truth	$f_5 = 6.0\sin(x_0)\cos(x_1)$
	GP	$y = 0.77exp(exp(\sin(\sin(\cos(0.73x_0)))))$ $*x_0\cos(x_1)$
	BSR	$y = (-7.06 * 10^{-9}) + (6.00)[\sin(x_0)\cos(x_1)]$ $+(2.66 * 10^{-9})[\sin(\frac{0.34}{\sin^2(x_1)}$ $-0.93exp(x_0 + x_1) - 0.95)]$
f_6	Truth	$f_6 = 1.35x_0x_1 + 5.5\sin((x_0 - 1)(x_1 - 1))$
	GP	$y = ((((((x_1\sin(x_0) + x_1x_0$ $- \sin(\frac{-x_0}{0.36}) - \sin((x_0 + x_1)))$ $- \sin((x_0x_1)^2)) + \sin(\frac{x_1}{0.36}))$ $- \sin((x_1\sin(x_0))^2)) - \sin(\frac{-x_0}{0.36}))$ $- \sin(x_1\sin(x_0) + x_1x_0))$ $- \sin(x_1\sin(x_0) + x_1x_0)$
	BSR	$y = (-0.19) + (-0.85)[1.69x_0x_1 + 1.19]$ $+(7.00 * 10^{-3})[exp(sin(exp(exp(exp(x_1)$ $+(1.37x_1 - 1.01)^3)^3)^3)))]$

Table 4: RMSEs for different K

	RMSE($mean \pm std$)		
Task	K=2	K=4	K=8
f_1	2.04 ± 3.27	2.86 ± 5.04	0.64 ± 2.46
f_2	6.84 ± 10.10	0.02 ± 0.03	0.03 ± 0.1
f_3	0.21 ± 0.20	0.06 ± 0.03	0.03 ± 0.02
f_4	0.16 ± 0.62	0.03 ± 0.06	0.01 ± 0.01
f_5	0.66 ± 1.13	0.29 ± 0.80	0.42 ± 0.94
f_6	4.63 ± 0.62	4.00 ± 0.34	5.28 ± 4.38

with white noise.

Conclusions and Future Research

This paper proposes a new symbolic regression method based on Bayesian statistics framework. Compared with traditional SR methods such as GP, the proposed method exhibits its advantage in several aspects, including better model interpretability, more executable prior incorporate way and more cost-effective memory use etc.

In the future, we continue this research and further improve and develop better method for symbolic regression. For example, we will study new MCMC algorithms to improve the search and sampling efficiency; we will study a dynamic empirical bayes method to optimize hyper-parameters in BSR; we will also research how to extend the proposed algorithm in a paralleled form in order to improve the computational efficiency.

References

[1] Maryam Amir Haeri, Mohammad Mehdi Ebadzadeh, and Gianluigi Folino. Statistical genetic programming for symbolic regression. *Appl. Soft Comput.*, 60(C):447–469, November 2017.

[2] Aftab Anjum, Fengyang Sun, Lin Wang, and Jeff Orchard. A novel continuous representation of genetic programmings using recurrent neural networks for symbolic regression. *CoRR*, abs/1904.03368, 2019.

[3] Chen Chen, Changtong Luo, and Zonglin Jiang. Elite bases regression: A real-time algorithm for symbolic regression. *CoRR*, abs/1704.07313, 2017.

[4] Qi Chen, Bing Xue, Lin Shang, and Mengjie Zhang. Improving generalisation of genetic programming for symbolic regression with structural risk minimisation. In *Proceedings of the Genetic and Evolutionary Computation Conference 2016*, pages 709–716. ACM, 2016.

[5] 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)*, pages 1137–1144. IEEE, 2015.

[6] Qi Chen, Mengjie Zhang, and Bing Xue. Feature selection to improve generalization of genetic programming for high-dimensional symbolic regression. *IEEE Transactions on Evolutionary Computation*, 21(5):792–806, 2017.

[7] Hugh A. Chipman, Edward I. George, and Robert E. McCulloch. Bayesian cart model search. *Journal of the American Statistical Association*, 93(443):935–948, 1998.

[8] Hugh A. Chipman, Edward I. George, and Robert E. McCulloch. Bart: Bayesian additive regression trees. *Ann. Appl. Stat.*, 4(1):266–298, 03 2010.

[9] Vipul K Dabhi and Sanjay K Vij. Empirical modeling using symbolic regression via postfix genetic programming. In *2011 International Conference on Image Information Processing*, pages 1–6. IEEE, 2011.

[10] J. W. Davidson, D. Savic, and G. A. Walters. Method for the identification of explicit polynomial formulae for the friction in turbulent pipe flow. *Journal of Hydroinformatics*, 1(2):115–126, 10 1999.

[11] J.W. Davidson, D.A. Savic, and G.A. Walters. Symbolic and numerical regression: experiments and applications. *Information Sciences*, 150(1):95 – 117, 2003. Recent Advances in Soft Computing.

[12] Fabrcio Olivetti de Frana. A greedy search tree heuristic for symbolic regression. *Information Sciences*, 442-443:18 – 32, 2018.

[13] Peter J Green. Reversible jump markov chain monte carlo computation and bayesian model determination. *Biometrika*, 82(4):711–732, 1995.

[14] Trevor Hastie and Robert Tibshirani. Bayesian backfitting (with comments and a rejoinder by the authors. *Statist. Sci.*, 15(3):196–223, 08 2000.

[15] W. K. Hastings. Monte carlo sampling methods using markov chains and their applications. *Biometrika*, 57(1):97–109, 1970.

[16] I. Icke and J. C. Bongard. Improving genetic programming based symbolic regression using deterministic machine learning. In *2013 IEEE Congress on Evolutionary Computation*, pages 1763–1770, June 2013.

[17] Maarten Keijzer. Improving symbolic regression with interval arithmetic and linear scaling. In Conor Ryan, Terence Soule, Maarten Keijzer, Edward Tsang, Riccardo Poli, and Ernesto Costa, editors, *Genetic Programming*, pages 70–82, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[18] Maarten Keijzer. Scaled symbolic regression. *Genetic Programming and Evolvable Machines*, 5(3):259–269, Sep 2004.

[19] Michael [VerfasserIn] Kommenda. Local optimization and complexity control for symbolic regression, 2018.

[20] Michael F. Korns. *Accuracy in Symbolic Regression*, pages 129–151. Springer New York, New York, NY, 2011.

[21] Li Li, Minjie Fan, Rishabh Singh, and Patrick Riley. Neural-guided symbolic regression with semantic prior. *CoRR*, abs/1901.07714, 2019.

[22] Changtong Luo, Chen Chen, and Zonglin Jiang. A divide and conquer method for symbolic regression. *IEEE Transactions on Evolutionary Computation*, 05 2017.

[23] Trent McConaghy. *FFX: Fast, Scalable, Deterministic Symbolic Regression Technology*, pages 235–260. Springer New York, New York, NY, 2011.

[24] Nicholas Metropolis, Arianna W Rosenbluth, Marshall N Rosenbluth, Augusta H Teller, and Edward Teller. Equation of state calculations by fast computing machines. *The journal of chemical physics*, 21(6):1087–1092, 1953.

[25] Miguel Nicolau and Alexandros Agapitos. On the effect of function set to the generalisation of symbolic regression models. In *GECCO*, 2018.

[26] Guido F. Smits and Mark Kotanchek. *Pareto-Front Exploitation in Symbolic Regression*, pages 283–299. Springer US, Boston, MA, 2005.

[27] Alexander Topchy and William F Punch. Faster genetic programming based on local gradient search of numeric leaf values. In *Proceedings of the 3rd Annual Conference on Genetic and Evolutionary Computation*, pages 155–162. Morgan Kaufmann Publishers Inc., 2001.

[28] E. Vladislavleva. Model-based problem solving through symbolic regression via pareto genetic programming. Technical report, 2008.

[29] E. J. Vladislavleva, G. F. Smits, and D. den Hertog. Order of nonlinearity as a complexity measure for models generated by symbolic regression via pareto genetic programming. *IEEE Transactions on Evolutionary Computation*, 13(2):333–349, April 2009.

[30] M. . Willis, H. G. Hiden, P. Marenbach, B. McKay, and G. A. Montague. Genetic programming: an introduction and survey of applications. In *Second International Conference On Genetic Algorithms In Engineering Systems: Innovations And Applications*, pages 314–319, Sep. 1997.

[31] Jan Zegklitz and Petr Posík. Symbolic regression algorithms with built-in linear regression. *CoRR*, abs/1701.03641, 2017.