Improving Genetic Programming Based Symbolic Regression Using Deterministic Machine Learning

Ilknur Icke
Morphology, Evolution and Cognition Lab.
Department of Computer Science
University of Vermont
Burlington, VT 05401
ilknur.icke@uvm.edu

Joshua C. Bongard
Morphology, Evolution and Cognition Lab.
Department of Computer Science
University of Vermont
Burlington, VT 05401
jbongard@uvm.edu

***Abstract*—Symbolic regression (SR) is a well studied method in genetic programming (GP) for discovering free-form mathematical models from observed data. However, it has not been widely accepted as a standard data science tool. The reluctance is in part due to the hard to analyze random nature of GP and scalability issues. On the other hand, most popular deterministic regression algorithms were designed to generate linear models and therefore lack the flexibility of GP based SR (GP-SR). Our hypothesis is that hybridizing these two techniques will create a synergy between the GP-SR and deterministic approaches to machine learning, which might help bring the GP based techniques closer to the realm of *big learning*. In this paper, we show that a hybrid deterministic/GP-SR algorithm outperforms GP-SR alone and the state-of-the-art deterministic regression technique alone on a set of multivariate polynomial symbolic regression tasks as the system to be modeled becomes more multivariate.**

***Index Terms*—symbolic regression, hybrid algorithms, elastic net, regularization**

I. Introduction

Symbolic regression is one the most popular applications of genetic programming and an attractive alternative to standard regression approaches due to its flexibility in generating free-form mathematical models from observed data without any domain knowledge. Indeed, user-friendly genetic programming based symbolic regression (GP-SR) tools such as Eureqa [1] have started to gain more attention from the scientific community over the last couple years. Despite various success stories ([2], [3], [4]) and claims that they will one day 'replace scientists', GP-SR applications (or any evolutionary computation based approach in general) have not yet been widely accepted as standard tools for the data scientists. Although many stochastic optimization algorithms such as stochastic gradient descent (SGD) [5] and metaheuristics such as simulated annealing [6] are well established in the mainstream ML, evolutionary computation methods are generally overlooked. GP suffers from various issues [7] that hinder its applicability to many real-world data science tasks. The theoretical foundations of GP are not as well understood as many of the standard machine learning (ML) algorithms due to the hard to analyze random nature of the technique. Scalability is also a very challenging problem. Efforts to increase scalability of GP via GPUs and cloud computing have been reported (such as in [8], [9]). It is our belief that, if GP-SR is to be a trustable *big learning* [10] tool, it needs to take advantage of the developments in the general ML as well as the parallel and distributed computing techniques.

The idea of studying evolutionary computation techniques from the standard ML perspective is not new. The behavior of GP has been studied in terms of the learning theory in [11] and [12] amongst others. The learnable evolution model (LEM) proposed in [13] is a technique to guide evolutionary processes with standard ML algorithms by creating hypotheses characterizing the differences between high performing and low performing individuals in the population.

Recently, it has been suggested that GP might not be the best option for SR and that stochasticity was not necessarily a virtue. In [14], a deterministic basis function expansion method used in conjunction with a state-of-the-art ML regression algorithm was proposed as an alternative to GP-SR. This algorithm that is known as the Fast Function Extraction (FFX) has been reported to outperform GP-SR on a number of real-world regression problems with dimensionality ranging from 13 to 1468. Our paper shares the same basic ideology, that is, SR should not stray away from the well-established techniques of ML. However, we argue that abandoning the GP approach might not be the best way for SR. Instead, we propose to hybridize the two approaches.

This paper explores one way to incorporate a deterministic ML regression technique into GP-SR in order to improve GP-SR. We report results on a suite of synthetic datasets that were generated to analyze performance as the problem difficulty increases. We believe that analyzing algorithm performance in this manner helps us understand the strengths/weaknesses of the approach before tackling more challenging real-world problems for which the ground truth is hardly ever available.

The organization of this paper is as follows: sections II and III discuss the background and related work. Our proposed algorithm to hybridize the GP-SR and deterministic ML approaches is detailed in section IV. Experimental results are presented and discussed in section V. Finally, section VI discusses conclusions and future work.

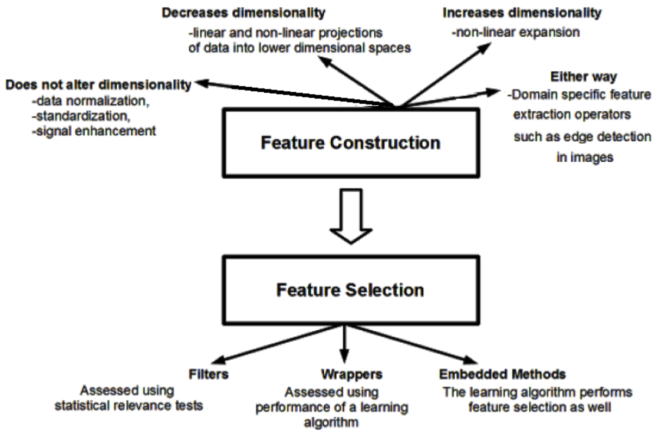
Decreases dimensionality
-linear and non-linear projections of data into lower dimensional spaces
Increases dimensionality
-non-linear expansion
Does not alter dimensionality
-data normalization,
-standardization,
-signal enhancement
Either way
-Domain specific feature extraction operators such as edge detection in images
Feature Construction
Feature Selection
Filters
Assessed using statistical relevance tests
Wrappers
Assessed using performance of a learning algorithm
Embedded Methods
The learning algorithm performs feature selection as well

Fig. 1: Feature extraction as a sequential process of creating features from the input variables and then selecting the most informative features.

II. Background

Data dimensionality poses a great challenge for the numerical and symbolic regression algorithms alike. As the number of predictors increases, it becomes more difficult to identify the informative predictors and to build accurate models. This problem has been well-studied in ML. The task of seeking the best representation for a given dataset in order to optimize the performance of a ML algorithm is known as *feature extraction* [15]. Feature extraction can be seen as a sequential process where new features are first constructed from the input variables and then the most informative ones are selected amongst the constructed features (Fig. 1). Feature construction may or may not increase data dimensionality. If the input variables are suspected to have interactions, it is generally the practice to create additional features via non-linear expansion (such as $x_1 * x_2$).

As for feature selection, the simplest approach is to rank the features with respect to how well they correlate with the predicted variable. This method has the risk of eliminating such features that might not be informative by themselves but might as well be very informative together. Subset selection methods aim to address this issue by considering a subset of features together. These techniques are divided into three main groups: filters, wrappers and embedded methods. *Filters* are pre-processing techniques that, independent from the learning algorithm, select a subset of variables with respect to some criteria such as mutual information. The *wrapper* techniques consider the learning algorithm as a black-box and select the set of features that optimize the performance of the learner. The feature subsets are generated by either forward selection, that gradually adds features or backward elimination, that starts with the whole set of features and eliminates least informative ones. The wrapper approach is computationally expensive as the learning algorithm needs to be executed many times. The *embedded* methods incorporate feature selection within the learning algorithm itself. The decision tree algorithms are the earliest examples of embedded methods. More recent embedded methods utilize the *regularization* technique.

Within the context of the linear regression problem, regularization refers to imposing additional constraints on the coefficients in order to reduce overfitting. In linear regression, given a multivariate dataset $\mathbf{X}_{[M\times N]} = \{\vec{x_1}, \vec{x_2}, ..., \vec{x_N}\}$, a matrix of observations, the response variable $\mathbf{Y}$ is defined as:

$$\mathbf{Y} = f(\mathbf{X}) = \beta_0 + \sum_{j=1}^{N} \beta_j * \vec{x_j}$$

The coefficients are computed via the least squares estimation by minimizing the residual sum of squares over the dataset X:

$$RSS = min_\beta(\sum_{i=1}^{M} y_i - \beta_0 - \sum_{j=1}^{N} \beta_j * x_{ij})^2$$

Since the parameters are computed on the training data, overfitting occurs manifesting itself as large coefficient values. Therefore, an additional constraint on the coefficients is imposed in order to tame the coefficients ($\sum_{j=1}^{N} ||\beta_j||_1 \leq t$). This algorithm that is known as lasso (least absolute shrinkage and selection operator) shrinks the coefficients and also performs feature elimination since the l_1-norm promotes sparsity. Therefore, the coefficients of uninformative features will be close to 0. An l_2-norm constraint is also possible and it is called ridge regression. Ridge regression has the effect of grouping the correlated variables so that they are included in the model together [16], [17]. The elastic net approach [18], [19] is a hybrid of lasso and ridge regression and formulated as:

$$Y = f(X) = \beta_0 + \sum_{j=1}^{N} \beta_j * \vec{x_j} + \lambda_2||\beta||_2^2 + \lambda_1||\beta||_1$$

Generally, λ_1, λ_2 are balanced by defining one single parameter ($0 \leq \alpha \leq 1$) that is called the mixing parameter. At the extreme values of α, elastic net behaves like purely lasso or purely ridge regression. A very large value of λ forces all βs to be 0. As λ is relaxed, the coefficients start to take nonzero values. This sweep of λ values can be visualized as a regularization path (Fig. 2). The algorithm is named elastic net since the "regularization path is like a stretchable net that retains all the big fish"[18].

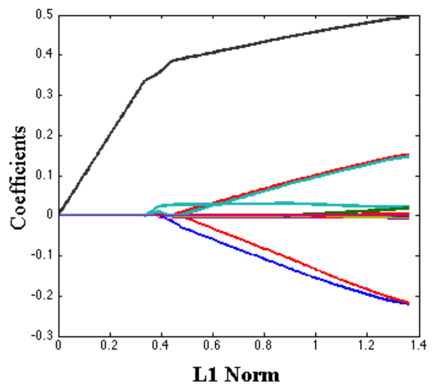

Fig. 2: Regularization path for elastic net on a 10-dimensional dataset. For each λ, the l_1-norm of the coefficients vector versus individual coefficient values are shown. Each line traces the change of coefficient values for one variable. At the beginning, no features are selected. Gradually, more features are added into the models as the coefficients become non-zero.

This basic linear regression algorithm applies to the generalized linear models (GLM) of the form:

$$Y = f(X) = \beta_0 + \sum_{j=1}^{N} \beta_j * b_j(X)$$

where $b_j(X)$ are nonlinear basis functions applied to the input variables in order to construct new features.

III. Related Work

GP-SR inherently performs feature selection when it finds sufficiently accurate data models; any feature that does not appear on the evolved expression can be considered redundant. However, when data dimensionality is high, the search space grows exponentially, making it difficult for GP-SR to find good solutions. The issue of feature selection in GP-SR algorithms have been studied by various researchers. Koza's automatically defined functions (ADF) [20] can be seen as a feature extraction method within the context of SR. A Pareto-GP based variable selection scheme was proposed in [21]. In [22], permutation tests were introduced in GP-SR to discriminate between informative and uninformative variables. In [23], feature selection capabilities of GP-SR and random forests were compared. The authors report that when it finds an accurate model, GP-SR captures the important features that are missed by the random forests algorithm.

The regularization approach has been applied to GP-SR in [24] for polynomial functional form discovery. The authors incorporate a function smoothness term into the fitness function as a way to decrease overfitting. The Fast Function Extraction (FFX) algorithm reported in [14] employs a nonlinear basis function expansion method that creates new features via unary and binary interactions of the input variables. The algorithm does not employ GP-SR to construct the features or the models. The new features are created in a deterministic manner and passed to the elastic net algorithm for model building. The algorithm generates multiple models for the λs on the regularization path. The non-dominated set of these models with respect to accuracy versus complexity are identified as the final models.

The difference of our proposed technique is that we perform feature extraction using an efficient deterministic ML algorithm and pass the features to GP-SR for model building. By taking advantage of the state-of-the-art ML, our algorithm aims to ease the burden of GP-SR in feature extraction and help it excel in model building.

IV. Improving GP-SR using Deterministic ML

The technique we propose in this paper has been largely inspired by the FFX algorithm [14]. However, the author had proposed to eliminate the GP for the symbolic regression problem in favor of a deterministic way to augment the dataset with polynomial features and then use a state-of-the-art machine learning algorithm (elastic net) for model building. In this paper, we propose to hybridize GP with the deterministic ML techniques so as to take advantage of the strengths of both approaches to solve symbolic regression problems more accurately and efficiently in comparison to either technique alone. The outline of the general idea behind FFX is presented in algorithm 1 (for a detailed description of the FFX algorithm, see[14]). The algorithm consists of three stages: feature construction, model building and model selection. The feature construction stage creates new features by applying binary nonlinear interactions (basis functions) and augmenting the original dataset (algorithm 2). It is possible to go beyond the binary interactions; however, this would increase the number of constructed features exponentially. In this paper, we considered only unary and binary features as in [14]).

Algorithm 1: The basic FFX algorithm

```
Input: V={v_1,v_2, ..., v_N}
Output: The set of non-dominated evolved models based on
        validation data error-model complexity (number of
        bases) trade-off
1  [ bases, expandedTrainingDataset] =
   basisFunctionExtraction(trainingDataset)
2  models={}
3  foreach $\alpha \in (0, 0.05, 0.1, ..., 1)$ do
4      models = models ⋃ glmnetfit(variables,trainingDataset)
5      nonDominatedModels =
6      extractParetoFrontier(models,expandedValidationDataset)
7      models=nonDominatedModels
8      models = models ⋃
       glmnetfit(bases,expandedTrainingDataset)
9      nonDominatedModels =
10     extractParetoFrontier(models,expandedValidationDataset)
       models=nonDominatedModels
11 end
```

The model building stage utilizes the coordinate descent elastic net algorithm (glmnet, line 4 of algorithm 1) that was proposed in [19]. As it is shown in Fig. 2, for each value of λ, one can build an expression using the corresponding coefficients. Therefore, the model building stage returns multiple expressions containing different numbers of basis functions.

Algorithm 2: *basisFunctionExtraction* : Polynomial basis function generation as new features form the observed data

```
Input: V={v_1,v_2, ..., v_N}
Output: Expanded Dataset: $V_e$={v_{e1},v_{e2}, ..., v_{eM} }
1  //Generate unary bases
2  foreach $v_1, v_2, ..., v_N$ do
3      unaryBases = unaryBases ⋃ $v_i$
4      foreach $exp_j$ do
5          unaryBases = unaryBases ⋃ $v_i^{exp_j}$
6      end
7      foreach $unaryOperator_k$ do
8          unaryBases = unaryBases ⋃ $unaryOperator_k(v_i)$
9      end
10 end
11 //Generate binary bases
12 foreach $u_i \in unaryBases$ do
13     foreach $u_j \in unaryBases$ do
14         binaryBases= binaryBases ⋃ $u_i * u_j$
15     end
16 end
```

Note that the models are built on the training data. At the model selection stage, the non-dominated set of models with respect to error on validation data versus expression complexity (the number of basis functions or bases for short) are identified.

Our proposed method to hybridize FFX/GP-SR is presented in algorithm 3. The process starts with a *variant* of FFX that was outlined in algorithm 1. From the set of all non-dominated models generated by FFX, all unique features (unary and binary) are extracted (line 2). These are the features that were found by FFX to be the most informative features for the given regression problem. Fig. 3 summarizes the process of identifying these features from the FFX output. Across all the models on the non-dominated set, each base is extracted and the coefficients are eliminated. The identified list of unique unary and binary basis functions are then utilized to create the new dataset with corresponding feature labels. The new dataset is then passed onto the GP-SR for model building.

Algorithm 3: The hybrid FFX/GP-SR algorithm

```
Input: V={$v_1$,$v_2$, ..., $v_N$}
Output: One best model with respect to the validation data
        error and complexity
1 nonDominatedModels = ffx(trainingDataset)
2 bases = extractBasisFunctions (nonDominatedModels,
3                                validationDataset)
4 newDataset=createNewDataset(bases)
5 bestModel = GP-SR(newDataset)
```

We hypothesized that for higher dimensional problems, pre-processing the dataset using a fast algorithm such as FFX would increase the chances of the GP-SR to succeed as opposed to expecting the GP-SR to perform feature extraction and model building simultaneously. The algorithm shown above may extract many basis functions for high dimensional datasets. In that case, further filtering of the uninformative features created by those basis functions can be done before passing the features to the GP-SR.

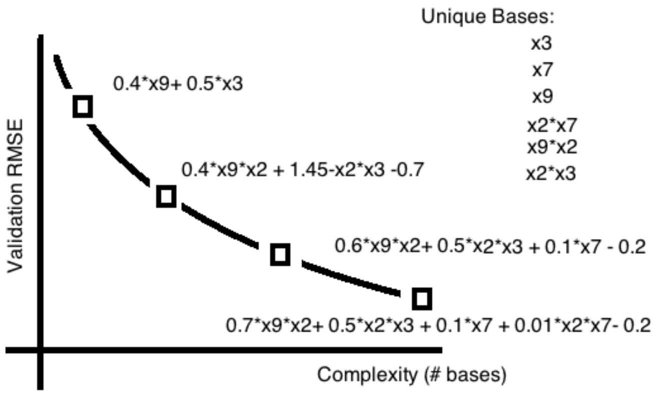
Unique Bases:
x3
x7
x9
x2*x7
x9*x2
x2*x3
0.4*x9+ 0.5*x3
0.4*x9*x2 + 1.45-x2*x3 -0.7
0.6*x9*x2+ 0.5*x2*x3 + 0.1*x7 - 0.2
0.7*x9*x2+ 0.5*x2*x3 + 0.1*x7 + 0.01*x2*x7- 0.2
Validation RMSE
Complexity (# bases)

Fig. 3: Generation of the new dataset based on FFX-generated expressions (extractBasisFunctions, line 2 of algorithm 3). Most frequent bases are extracted from the Pareto frontier and used as features for the new dataset.

V. Experimental Results

We implemented our GP-based Symbolic Regression application using the GPTIPS Matlab package downloaded from [25]. Our version of the FFX algorithm and FFX/GP-SR algorithms were also implemented in Matlab using the glmnet package downloaded from [26] and the GPTIPS package. All experiments were run on a cluster computing environment.

A. Synthetic Benchmark Data Suite and Evaluation Procedure

We tested our algorithms on a systemically generated suite of multivariate polynomial functions in order to analyze the performance as the difficulty of the problem is increased in terms of the number of variables(1-3, 10), order of the polynomial (1-4) and the number of basis functions each polynomial contains (1-4). Examples of such functions are presented in the following sections. For each polynomial, 2500 data points were generated as training points and separate sets of 1250 data points were held aside as validation and test data. All input variables were randomly sampled within the range [0,1].

The evaluation procedure is as follows: for each type of polynomial (such as order 2 with 2 bases), there are 30 different datasets generated by the 30 different polynomials of that type. For each such polynomial, we perform 30 independent GP-SR and FFX/GP-SR runs with 1 minute runtime budget. Since FFX is deterministic it runs only once. For FFX, the final set of non-dominated models are recorded. For GP-SR and FFX/GP-SR the best model with respect to the validation dataset is recorded for each run. In summary, for each type of polynomial, 900 runs of GP and 900 runs of FFX/GP-SR runs are performed.

Unlike the general approach where a close approximation with respect to the prediction error is satisfactory for evaluation of the success, in this paper, we also assess the outcomes in terms of how close the functional form of the hidden target expression is matched. For instance, if the hidden target expression is $\alpha_1 * x_1 + \alpha_2 * x_2 + \beta$, where α_i, β are real valued coefficients, we consider each evolved model with low prediction error that matches this functional form as a successful outcome regardless of the actual values of the coefficients. Namely, the degree of similarity between the hidden ground truth and the evolved polynomials is defined in terms of syntactic similarity.

For FFX, the evaluation is performed based on the whole set of non-dominated models (Fig. 4). If a model with the correct syntactic form exists in this set, then the FFX run is considered a success. For GP and FFX/GP, all 30 runs per unique polynomial are examined. If a model with the correct syntactic form exists in this set, the algorithm is considered successful on discovering that polynomial. We also record the syntactic similarity to the correct polynomial form. The similarity values range between 0 and 1; 1 meaning a perfect match to the true syntactic form and 0 meaning no match at all. For each unique polynomial, the model with best validation error is identified and its syntactic similarity and test error values are recorded as the outcomes for each algorithm.

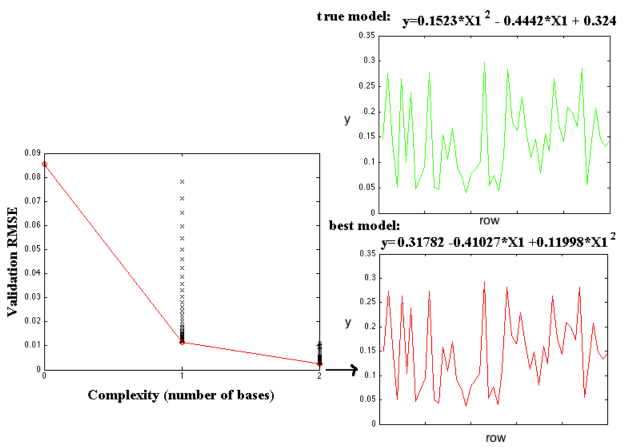

Fig. 4: Result of an FFX run on a second order 1D problem with two basis functions. The final model shown here is selected from the Pareto frontier as the one with lowest validation error.

In the following sections, we present the results separately for five sets of data organized with respect to dimensionality. We start from the simplest case: one variable and various polynomials ranging from linear with a single term to fourth order with four terms. We increase the number of variables to 2,3,10 then to 25 and repeat the same set of experiments. Finally, statistical significance tests are utilized in order to check if hybridizing the GP-SR with FFX helps improve the performance of GP-SR given the data dimensionality.

B. Results on the benchmark problems

Unless otherwise specified, all GP-SR and FFX runs were performed using the following default parameters:

TABLE I: Default GP-SR parameters

Parameter	Value
Representation	GPTIPS [25] Multigene syntax tree Number of genes: 1 Maximum tree depth: 7
Population Size Runtime Budget	500 1 minute
Selection Tournament Size	Lexicographic tournament selection 7
Crossover Operator Crossover Probability	Sub-tree crossover 0.85
Mutation Operator Mutation Probability	Sub-tree mutation 0.1
Reproduction Probability	0.05
Building Blocks	Operators: $\{+, -, *, protected/\}$ Terminal Symbols: $\{x_1, ..., x_N\}$
Fitness	$\frac{1}{N}\sqrt{\sum(y-\hat{y})^2}$
Elitism	Keep 1 best individual

1-dimensional polynomials: The following polynomials are examples of the 1-dimensional hidden target expressions (ground truth) used in our experiments. The polynomials are grouped with respect to the highest order variable interaction. Within each group, the syntactic complexity of the expressions increase as more basis functions are included gradually. For each expression, the number in the paranthesis on the left-hand side indicates how many types of nonlinear interactions (i.e unary, binary,...) are included in that expression.

TABLE II: Default FFX parameters

Parameter	Value
Basis Function Expansion	Exponents : 1 Interactions : Unary, Binary Operators : { }
Elastic Net	α : $\{0, 0.05, 0.1, ..., 1\}$ λ : 100 λ values calculated by glmfit based on α Maximum basis functions allowed : 250
Model Selection	Non-dominated models with respect to validation data error versus number of bases

- order 1 polynomial:

$$(1) \quad y = 0.288 * x_1 + 0.8446$$

- order 2 polynomials:

$$(1) \quad y = 0.14 * x_1^2 + 0.629$$
$$(2) \quad y = 0.12 * x_1 + 0.03 * x_1^2 + 0.29$$

- order 3 polynomials:

$$(1) \quad y = -0.31 * x_1^3 - 0.11$$
$$(2) \quad y = 1.35 * x_1^2 - 0.83 * x_1^3 + 0.139$$
$$(3) \quad y = 0.13 * x_1 + 0.44 * x_1^2 + 0.34 * x_1^3 + 0.39$$

- order 4 polynomials:

$$(1) \quad y = 0.20 * x_1^4 + 0.13$$
$$(2) \quad y = 0.24 * x_1^3 + 0.23 * x_1^4 + 0.39$$
$$(3) \quad y = 0.75 * x_1^2 + 0.30 * x_1^3 + 0.35 * x_1^4 + 0.334$$
$$(4) \quad y = 0.02 * x_1 + 0.13 * x_1^2 + 0.301 * x_1^3 + 0.32 * x_1^4 + 0.91$$

Tables III, IV and V show the number of successful runs for each algorithm for each type of polynomial. As the results of the GP-SR runs indicate, even the 1-dimensional hidden target expressions become more challenging as the number of basis functions increases. Out of the 30 runs, the proportion of successful discovery of the correct functional form declines as the syntactic complexity of the target expressions increase.

TABLE III: Standalone GP-SR runs on 1D datasets (1 minute).

		Bases			
		1	2	3	4
Order of the Polynomial	1	30	-	-	-
	2	30	29	-	-
	3	30	27	19	-
	4	30	27	11	16

TABLE IV: FFX runs on 1D datasets with unary (x_i) and binary interactions $(x_i * x_j)$ (average run time: 7 seconds)

		Bases			
		1	2	3	4
Order of the Polynomial	1	30	-	-	-
	2	30	30	-	-
	3	0	0	0	-
	4	0	0	0	0

TABLE V: FFX/GP-SR runs on 1D datasets (1 minute GP run on FFX-generated dataset)

		Bases			
		1	2	3	4
Order of the Polynomial	1	30	-	-	-
	2	30	27	-	-
	3	30	26	19	-
	4	30	28	16	17

It is not surprising that FFX did not succeed at all when the target polynomials were cubic and fourth order, as we have only allowed for unary and binary basis functions. Our goal was to test how much FFX might help GP-SR discover 3rd and 4th order polynomials utilizing binary bases only. Fig. 5 shows that the hybrid did not outperform the plain GP-SR even on the quadratic polynomials as the problem was easy for GP-SR to handle within the given runtime budget.

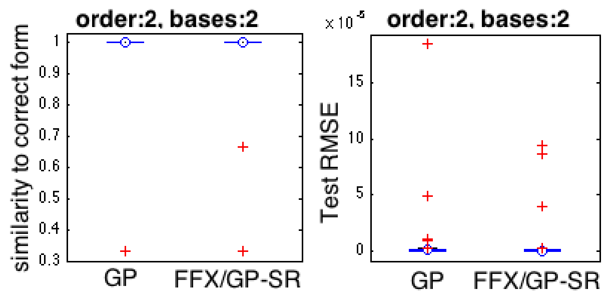

Fig. 5: Summary of runs on second order 1D polynomials with 2 basis functions. According to Wilcoxon rank sum tests, FFX/GP-SR does not outperform GP-SR in 1 minute runtime

2-dimensional polynomials: We repeated the experiments using a set of 30 polynomials for each listed form below:

- order 1 polynomial:

  (1) $y = 0.62 * x_2 - 0.854$

- order 2 polynomials:

  (1) $y = 0.22 * x_1^2 + 0.05$

  (2) $y = 0.12 * x_1 - 0.25 * x_1 * x_2 + 0.4$

- order 3 polynomials:

  (1) $y = 1.67 * x_1^2 * x_2 + 0.46$

  (2) $y = 0.17 * x_1 * x_2 + 0.369 * x_2^3 - 0.3$

  (3) $y = 0.03 * x_2 - 0.36 * x_1^2 + 0.22 * x_2^3 + 0.42$

- order 4 polynomials:

  (1) $y = 2.88 * x_1^2 * x_2^2 + 0.15$

  (3) $y = 0.4978 * x_1 * x_2^3 - 0.08 * x_1^4 + 0.36$

  (3) $y = 2.19 * x_1 * x_2^2 - 0.87 * x_2^3 + 0.87 * x_1^2 * x_2^2 + 0.39$

  (4) $y = 0.13 * x_2 - 1.313 * x_1 * x_2 - 0.1 * x_1^3$
  $0.4926 * x_1^2 * x_2^2 + 0.19$

TABLE VI: Standalone GP-SR runs on 2D datasets (1 minute).

		Bases			
		1	2	3	4
Order of the Polynomial	1	30	-	-	-
	2	30	29	-	-
	3	30	22	15	-
	4	30	20	10	2

TABLE VII: FFX runs on 2D datasets with unary (x_i) and binary interactions ($x_i * x_j$) (average run time: 9 seconds)

		Bases			
		1	2	3	4
Order of the Polynomial	1	30	-	-	-
	2	30	16	-	-
	3	0	0	0	-
	4	0	0	0	0

Similar to the 1-dimensional case, we found that FFX/GP-SR did not outperform GP-SR on 2-dimensional polynomial dataset (Fig. 6).

TABLE VIII: FFX/GP-SR runs on 2D datasets (1 minute GP run on FFX-generated dataset)

		Bases			
		1	2	3	4
Order of the Polynomial	1	30	-	-	-
	2	30	30	-	-
	3	30	19	14	-
	4	30	20	11	3

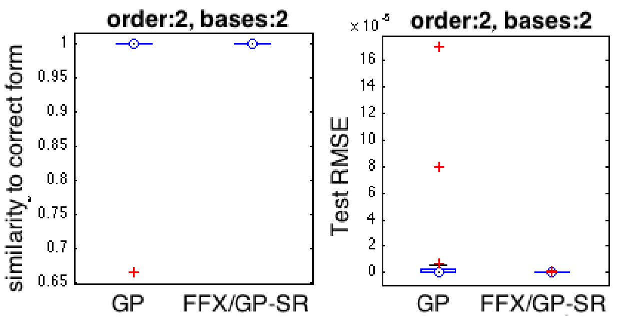

Fig. 6: Summary of runs on second order 2D polynomials with 2 basis functions. According to Wilcoxon rank sum tests, FFX/GP-SR does not outperform GP-SR in 1 minute runtime

3-dimensional polynomials: We repeated the experiments using a set of 30 polynomials for each listed form below::

- order 1 polynomial:

  (1) $y = 0.746 * x_3 + 0.8268$

- order 2 polynomials:

  (1) $y = 0.54 * x_3^2 + 0.4$

  (2) $y = 0.8651 * x_1 - 0.61 * x_2^2 - 0.30$

- order 3 polynomials:

  (1) $y = 0.84 * x_1 * x_2 * x_3 - 0.86$

  (2) $y = 0.93 * x_1 * x_2 - 0.46 * x_3^3 + 0.88$

  (3) $y = 0.04 * x_2 - 0.18 * x_2 * x_3 - 0.01 * x_1 * x_2^2 + 0.3$

- order 4 polynomials:

  (1) $y = 0.20 * x_1 * x_2^3 + 0.91$

  (2) $y = 0.73 * x_1^2 * x_2 - 0.07 * x_1^2 * x_2 * x_3 + 0.39$

  (3) $y = 1.2 * x_1 * x_2 + 0.68 * x_1^2 * x_2 +$
  $0.48 * x_1^2 * x_2 * x_3 + 0.41$

  (4) $y = 0.35 * x_3 - 0.32 * x_2 * x_3 - 0.35 * x_1 * x_2^2 -$
  $0.39 * x_3^4 + 0.24$

TABLE IX: Standalone GP runs on 3D datasets (1 minute)

		Bases			
		1	2	3	4
Order of the Polynomial	1	30	-	-	-
	2	30	25	-	-
	3	30	25	9	-
	4	30	13	12	3

TABLE X: FFX runs on 3D datasets with unary (x_i) and binary interactions ($x_i * x_j$) (average run time: 12 seconds)

		Bases			
		1	2	3	4
Order of the Polynomial	1	30	-	-	-
	2	29	16	-	-
	3	0	0	0	-
	4	0	0	0	0

TABLE XI: FFX/GP-SR runs on 3D datasets (1 minute GP run on FFX-generated dataset)

		Bases			
		1	2	3	4
Order of the Polynomial	1	30	-	-	-
	2	30	26	-	-
	3	30	28	14	-
	4	30	17	12	6

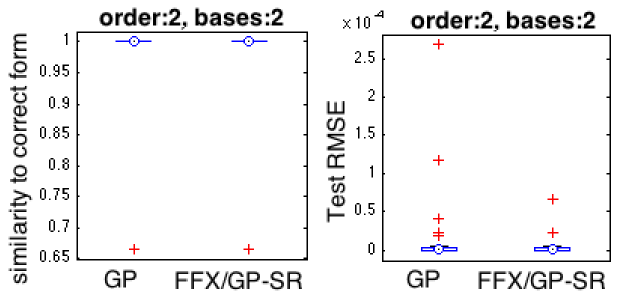

Fig. 7: Summary of runs on second order 3D polynomials with 2 basis functions. According to Wilcoxon rank sum tests, FFX/GP-SR does not outperform GP-SR in 1 minute runtime

Second Order Polynomials from 10 & 25-dimensional Datasets: In order to test our intuition that the FFX/GP-SR would perform better for higher dimensional data, we raised the dimensionality of synthetic data to 10 and 25. In this section, we present results of GP-SR and FFX/GP-SR runs on 30 second order polynomial functions with 2 basis functions such as the following: $y = 0.7 * x_3 - 0.23 * x_9 * x_7 + 0.2$ where $x_i \in x_1, ..., x_{10}$ and $x_i \in x_1, ..., x_{25}$ respectively.

Since we only allowed unary and binary interactions in our FFX implementation, FFX/GP-SR did not significantly do better than GP-SR alone in terms of finding the correct functional form of the hidden polynomials with orders greater than 2. This was evident in 1:3-dimensional polynomial experiments reported in the previous section. Therefore, we only performed runs on the second order polynomials for the 10 and 25-dimensional data. As in the previous sections, the results reported here are aggregated over the runs on 30 different polynomials for a runtime budget of 1 minute..

On the 10-dimensional dataset, the FFX algorithm found the correct syntactic form (identified the correct variables and linear form) for 10 out of the 30 polynomials. The GP-SR algorithm by itself found 14 out of the 30 and the FFX/GP-SR hybrid found 22 out of the 30 target polynomial forms correctly. On the 25-dimensional dataset, the number of times each algorithm found the correct functional form was 18,1 and 26 out of the 30 target polynomials for the FFX, GP-SR and FF/GP-SR respectively.

Fig. 8 and Fig. 9 summarize the comparisons of GP and FFX/GP-SR algorithms based on the similarity to the correct polynomial form and prediction errors. As the dimensionality increases from 10 to 25, the performance of the GP-SR declines sharply in terms of recovering the correct functional form within the given runtime budget of 1 minute. The FFX/GP-SR hybrid, on the other hand, continues to succeed as the dimensionality increases. In summary, the utility of the hybrid algorithm becomes more significant as the number of variables increases. The hybrid algorithm discovers expressions that are significantly more similar to the ground truth and significantly more predictive.

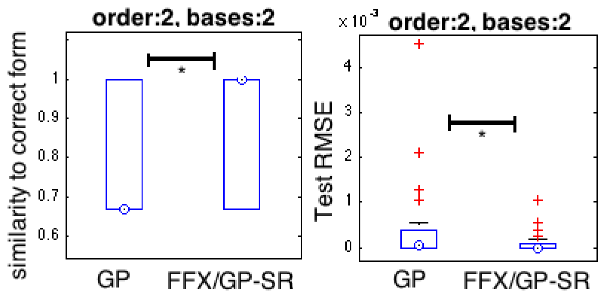

Fig. 8: Summary of runs on second order 10D polynomials with 2 basis functions. The final expressions found by FFX/GP-SR are significantly more similar to the ground truth as opposed to GP-SR alone (Wilcoxon rank sum right-tailed test, $\alpha = 0.05$, p-value:0.0198) and more predictive (Wilcoxon rank sum left-tailed test, $\alpha = 0.05$, p-value:0.005)

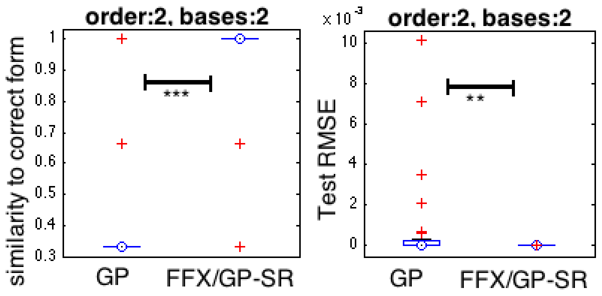

Fig. 9: Summary of runs on second order 25D polynomials with 2 basis functions. The final expressions found by FFX/GP-SR are significantly more similar to the ground truth as opposed to GP-SR alone (Wilcoxon rank sum right-tailed test, $\alpha = 0.001$, p-value $<< 0.001$) and more predictive (Wilcoxon rank sum left-tailed test, $\alpha = 0.01$, p-value$<<$0.01)

C. Discussion

Even though the hybrid algorithm did not provide additional advantage over plain GP-SR on low dimensional datasets, our results indicated that, as the data dimensionality increased (10 and then to 25, in this case), the FFX/GP-SR hybrid performed significantly better in finding more predictive expressions that are more similar to the hidden ground truth in comparison to GP-SR alone. By similar, we mean the success at which the algorithm captures the informative variables and their nonlinear interactions.

Based on our experiment results, we note that even though the FFX algorithm might not always find the correct functional form for the target expressions itself, the rich set of building blocks it provides to the GP-SR has the potential to boost the performance of the GP-SR. Since the GP-SR search space grows exponentially as the number of variables increases, eliminating the uninformative variables beforehand using a deterministic ML algorithm helps ease the burden of discovering the informative variables and constructing the useful nonlinear interactions for model building.

VI. Conclusion

Although GP-SR has been known for a couple of decades, only recently that tools such as Eureqa have started to attract a larger number of scientists due to almost zero-maintenance and user-friendly application interfaces along with various improvements on the metaheuristics and options for parallel and distributed computing. However, GP-SR is yet to be accepted as a standard data analysis tool. In this paper, we argued that the resistance from the ML community is not totally unfounded. First of all, theoretical underpinnings of the GP-SR such as converge proofs are not as well established as standard deterministic algorithms. GP-SR is computationally more expensive compared to most standard ML algorithms and even though many intuitive strategies might be built into the algorithm, there is no guarantee that optimal data models will emerge at the end of the run. More importantly, despite all the success stories, GP-SR techniques do not necessarily outperform the state of the art in ML, especially on high dimensional problems. On the other hand, the strength of GP-SR is in its model-free nature which makes it possible that the algorithm might discover optimal and more intelligible, novel models for the observed data. In summary, it is our belief that stochasticity can be a virtue for SR if it is directed intelligently.

In this paper, we showed that it would be possible to create synergy between the deterministic ML and GP-SR approaches by hybridizing them. The technique presented in this paper is just one way out of many possible options to combine the GP-SR and standard ML for regression problems. Here, we incorporated building blocks extracted by the deterministic regression algorithm into the GP-SR algorithm by means of re-creating the input dataset. Another option is to seed the GP-SR runs with the candidate solutions found by the deterministic approach. Genetic programming can also be used to evolve features (via the generation of the basis functions) that can be fed to the deterministic algorithm for model generation. Our current work focuses on investigating other possible ways to hybridize GP-SR and deterministic ML based approaches in order to address high dimensional real-world regression problems.

Acknowledgment

This research was supported by a DARPA grant #FA8650-11-1-7155. The authors also acknowledge the Vermont Advanced Computing Core which is supported by NASA (NNX 06AC88G), at the University of Vermont for providing High Performance Computing resources that have contributed to the research results reported within this paper.

References

[1] M. Schmidt and H. Lipson, “Distilling free-form natural laws from experimental data,” *Science*, vol. 324, no. 5923, pp. 81–85, 2009.

[2] J. Koza, “Human-competitive results produced by genetic programming,” *Genetic Programming and Evolvable Machines*, vol. 11, pp. 251–284, 2010.

[3] “Robot biologist solves complex problem from scratch,” http://cacm.acm.org/careers/136345-robot-biologist-solves-complex-problem-from-scratch/fulltext, October 2011.

[4] R. Dubčáková, “Eureqa: software review,” *Genetic Programming and Evolvable Machines*, vol. 12, no. 2, pp. 173–178, Jun. 2011.

[5] “Stochastic optimization: Icml 2010 tutorial,” http://www.ttic.edu/icml2010stochopt/.

[6] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, “Optimization by simulated annealing,” *Science*, vol. 220, no. 4598, pp. 671–680, 1983.

[7] M. O’Neill, L. Vanneschi, S. Gustafson, and W. Banzhaf, “Open issues in genetic programming,” *Genetic Programming and Evolvable Machines*, vol. 11, no. 3-4, pp. 339–363, Sep. 2010.

[8] “Genetic programming on general purpose graphics processing units,” http://www.gpgpgpu.com/.

[9] D. Sherry, K. Veeramachaneni, J. McDermott, and U.-M. O’Reilly, “Flex-gp: genetic programming on the cloud,” in *Proceedings of the 2012 European conference on Applications of Evolutionary Computation, EvoApplications’12*, 2012, pp. 477–486.

[10] “Big learning, algorithms, systems and tools,” http://biglearn.org/.

[11] N. Amil, N. Bredeche, C. Gagn, S. Gelly, M. Schoenauer, and O. Teytaud, “A statistical learning perspective of genetic programming,” in *Genetic Programming*, ser. Lecture Notes in Computer Science, L. Vanneschi, S. Gustafson, A. Moraglio, I. Falco, and M. Ebner, Eds. Springer Berlin Heidelberg, 2009, vol. 5481, pp. 327–338.

[12] M. Castelli, L. Manzoni, S. Silva, and L. Vanneschi, “A quantitative study of learning and generalization in genetic programming,” in *Genetic Programming*, ser. Lecture Notes in Computer Science, S. Silva, J. Foster, M. Nicolau, P. Machado, and M. Giacobini, Eds. Springer Berlin Heidelberg, 2011, vol. 6621, pp. 25–36.

[13] R. S. Michalski, “Learnable evolution model: Evolutionary processes guided by machine learning,” *Machine Learning*, vol. 38, pp. 9–40, 2000.

[14] T. McConaghy, “Ffx: fast, scalable, deterministic symbolic regression technology,” *Genetic Programming Theory and Practice IX, Edited by R. Riolo, E. Vladislavleva, and J. Moore,Springer*, 2011.

[15] I. Guyon and A. Elisseeff, “An introduction to variable and feature selection,” *Journal of Machine Learning Research*, vol. 3, pp. 1157–1182, 2003.

[16] R. Tibshirani, “Regression shrinkage and selection via the lasso,” *Journal of the Royal Statistical Society (Series B)*, vol. 58, pp. 267–288, 1996.

[17] K. P. Murphy, *Machine Learning A Probabilistic Perspective*. The MIT Press, 2012.

[18] H. Zou and T. Hastie, “Regularization and variable selection via the elastic net,” *Journal of the Royal Statistical Society Series B*, vol. 67, no. 2, pp. 301–320, 2005.

[19] T. H. Jerome Friedman and R. Tibshirani, “Regularization paths for generalized linear models via coordinate descent,” *Journal of Statistical Software*, vol. 33, no. 1, January 2010.

[20] J. R. Koza, *Genetic programming: on the programming of computers by means of natural selection*. Cambridge, MA, USA: MIT Press, 1992.

[21] G. Smits, A. Kordon, K. Vladislavleva, E. Jordaan, and M. Kotanchek, “Variable selection in industrial datasets using pareto genetic programming,” in *Genetic Programming Theory and Practice III*, ser. Genetic Programming, T. Yu, R. Riolo, and B. Worzel, Eds. Springer US, 2006, vol. 9, pp. 79–92.

[22] R. K. McRee, “Symbolic regression using nearest neighbor indexing,” in *Proceedings of the 12th annual conference companion on Genetic and evolutionary computation,GECCO ’10*, 2010, pp. 1983–1990.

[23] S. Stijven, W. Minnebo, and K. Vladislavleva, “Separating the wheat from the chaff: on feature selection and feature importance in regression random forests and symbolic regression,” in *Proceedings of the 13th annual conference companion on Genetic and evolutionary computation,GECCO ’11*, 2011, pp. 623–630.

[24] N. Nikolaev and H. Iba, “Regularization approach to inductive genetic programming,” *Evolutionary Computation, IEEE Transactions on*, vol. 5, no. 4, pp. 359 –375, 2001.

[25] D. Searson, “Gptips package for matlab,” https://sites.google.com/site/gptips4matlab/.

[26] H. Jiang, “Glmnet for matlab,” http://www-stat.stanford.edu/ tibs/glmnet-matlab/.